DEEP LEARNING USING VISION AND LIDAR FOR GLOBAL ROBOT LOCALIZATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Brett Gowling

May 2024
COMMITTEE MEMBERSHIP

TITLE: Deep Learning Using Vision and LiDAR for Global Robot Localization

AUTHOR: Brett Gowling

DATE SUBMITTED: May 2024

COMMITTEE CHAIR: John Seng, Ph. D.
Professor of Computer Science

COMMITTEE MEMBER: Jonathan Ventura, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Lubomir Stanchev, Ph. D.
Professor of Computer Science
ABSTRACT
Deep Learning Using Vision and LiDAR for Global Robot Localization
Brett Gowling

As the field of mobile robotics rapidly expands, precise understanding of a robot’s position and orientation becomes critical for autonomous navigation and efficient task performance. In this thesis, we present a snapshot-based global localization machine learning model for a mobile robot, the e-puck, in a simulated environment. Our model uses multimodal data to predict both position and orientation using the robot’s on-board cameras and LiDAR sensor. In an effort to minimize localization error, we explore different sensor configurations by varying the number of cameras and LiDAR layers used. Additionally, we investigate the performance benefits of different multimodal fusion strategies while leveraging the EfficientNet CNN architecture as our model’s foundation. Data collection and testing is conducted using Webots simulation software, and our results show that, when tested in a 12m x 12m simulated apartment environment, our model is able to achieve positional accuracy within 0.2m for each of the x and y coordinates and orientation accuracy within 2°, all without the need for sequential data history. Our results demonstrate the potential for accurate global localization of mobile robots in simulated environments without the need for existing maps or temporal data.
ACKNOWLEDGMENTS

Thanks to:

- My fiancée, Reese, for the love and encouragement you displayed through your unwavering confidence in my abilities, your seemingly endless patience and selfless acts toward me, and your passion for seeing God’s goodness in the simplest of things. Thank you for loving me best by loving Him most.

- My parents, Jeff and Tracy, for instilling in me the value of integrity, showing interest in my passions, backing my every pursuit wholeheartedly, and teaching me to love the Lord above all else.

- All my family and friends who, through steadfast and unshakeable prayer, supported me near and far.

- My roommate, David, for journeying through most of my college career with me, indulging my creative impulses, and doing practically everything together. Let’s grab Jamba sometime.

- Dr. John Seng, for inspiring this work and supporting my research, and Dr. Jonathan Ventura and Dr. Lubomir Stanchev, for fueling my desire to never stop learning.

To God be the glory.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
</tr>
</tbody>
</table>

## CHAPTER

1. Introduction                                                               1

2. Background                                                                 4

   2.1 Deep Learning                                                           4

   2.1.1 Convolutional Neural Networks                                         5

   2.1.2 EfficientNet                                                        5

   2.2 Transfer Learning                                                      6

   2.3 Sensors                                                                 7

   2.3.1 LiDAR                                                                8

   2.3.2 IR and Ultrasonic Distance Sensors                                   9

   2.3.3 Vision                                                              9

   2.4 Sensor Fusion                                                          11

   2.4.1 Fusion Strategies                                                    12

   2.5 Localization                                                          13

3. Related Work                                                               15

   3.1 Image-Based Localization                                               15

   3.2 Early vs. Late Fusion                                                  16

   3.3 Localization Techniques                                                17

   3.3.1 Landmark-Based Localization                                          17

   3.3.2 SLAM                                                                  18
3.3.3 OneShot Global Localization ........................................... 19
3.3.4 Pose Estimation from a Single Image ................................. 19

4 Implementation ........................................................................ 21
4.1 Webots ................................................................................. 21
4.2 Data Collection ...................................................................... 22
  4.2.1 Mobile Agent ................................................................... 23
  4.2.2 Robot Peripherals and Modalities ..................................... 24
  4.2.3 Robot Controller ............................................................. 25
  4.2.4 Data Samples and Interface ............................................. 29
  4.2.5 Datasets .......................................................................... 30
  4.2.6 Processing ....................................................................... 36
4.3 Network Design and Architecture ........................................... 37
  4.3.1 Pre-trained Network: EfficientNet ................................. 37
  4.3.2 Network Output ............................................................. 38
  4.3.3 Late Fusion Architecture ................................................ 39
  4.3.4 Early Fusion Architecture .............................................. 40

5 Experimental Design ................................................................. 41
  5.1 Performance Metrics .......................................................... 41
  5.2 Independent Variables and Parameters ............................... 43
  5.2.1 Experiments for Identifying Batch Size Influence .......... 44
  5.2.2 Experiments for Individual Sensor Contribution ............ 44
  5.2.3 Experiments for Comparing Early and Late Fusion ......... 46
  5.3 Inference .............................................................................. 46
  5.4 Testing and Development Environments .............................. 48

6 Results and Analysis ................................................................. 50
6.1 Batch Size Results .................................................. 50
6.2 Camera Quantity Impact ........................................... 52
6.3 LiDAR Layer Impact .................................................. 54
6.4 Data Fusion Performance .......................................... 55
6.5 Limitations ............................................................. 59
  6.5.1 Inference Outliers .............................................. 59
  6.5.2 Memory Demand ................................................ 62
7  Future Work ........................................................... 65
  7.1 3D Mapping ........................................................... 65
  7.2 Snapshot-Based to Sequential .................................. 65
  7.3 Scaling .............................................................. 66
8  Conclusion ............................................................. 67
BIBLIOGRAPHY .......................................................... 69
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>45</td>
</tr>
<tr>
<td>6.1</td>
<td>57</td>
</tr>
</tbody>
</table>

Table 4.1: Webots Camera Settings
Table 4.2: Webots LiDAR Settings
Table 4.3: Dataset Summary
Table 5.1: Expected Distances by Environment
Table 5.2: Vision Data Views by Camera Count
Table 6.1: Early vs. Late Fusion Absolute, Unstandardized Errors
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 EfficientNet-B0 Baseline Network Architecture, Taken from Tan and Le [43]</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Scaling Up EfficientNet-B0 with Different Methods, Taken from Tan and Le [43]</td>
<td>7</td>
</tr>
<tr>
<td>3.1 3D Object Detection Early Fusion Network, Taken from Bhanushali et al. [7]</td>
<td>16</td>
</tr>
<tr>
<td>3.2 3D Object Detection Late Fusion Network, Taken from Bhanushali et al. [7]</td>
<td>17</td>
</tr>
<tr>
<td>4.1 Modified Webots Square Environment</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Modified Webots Partial Apartment Environment</td>
<td>24</td>
</tr>
<tr>
<td>4.3 Modified Webots Complete Apartment Environment</td>
<td>25</td>
</tr>
<tr>
<td>4.4 GCtronic e-puck Robot</td>
<td>26</td>
</tr>
<tr>
<td>4.5 e-puck Peripherals (Sensors, LEDs, and Camera) from GCtronic Documentation [16]</td>
<td>27</td>
</tr>
<tr>
<td>4.6 Automated Data Collection Robot Controller Flowchart</td>
<td>28</td>
</tr>
<tr>
<td>4.7 Rendering of 128-layer LiDAR Point Cloud</td>
<td>30</td>
</tr>
<tr>
<td>4.8 Data Sample: e-puck Front (Top-Left), Rear (Top-Right), Left (Bottom-Left), and Right (Bottom-Right) Views</td>
<td>31</td>
</tr>
<tr>
<td>4.9 Webots Camera Imaging Overlay with Robot Location and Orientation</td>
<td>31</td>
</tr>
<tr>
<td>4.10 Complete Apartment Top View</td>
<td>33</td>
</tr>
<tr>
<td>4.11 Complete Apartment Data Collection Path, Testing Set</td>
<td>34</td>
</tr>
<tr>
<td>4.12 Complete Apartment Data Distributions, Testing Set</td>
<td>35</td>
</tr>
<tr>
<td>4.13 Complete Apartment $\theta$ Distributions, Testing Set</td>
<td>36</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The integration of mobile robotic agents into various sectors such as manufacturing, agriculture, transportation, and healthcare is no longer a futuristic concept. These agents have been adopted to optimize monotonous operations, enhance precision and safety in hazardous circumstances, and facilitate remote productivity. In nearly all applications involving mobile robotics, particularly when robots share a workspace with people, there is a critical need for a precise, accurate, and continuous understanding of the robot’s position and orientation, also known as its pose. The process of establishing and maintaining this awareness is referred to as localization, a historically challenging problem. The challenges encompass navigating a space without a map or model, relocalizing when moved to an arbitrary location, and extracting features from limited or noisy sensor data.

In response to the escalating demand for advanced localization solutions, many have turned to the success of increasingly complex artificial neural networks due to their straightforward implementations and wide-ranging applications. One method of leveraging neural networks for mobile robot localization involves feeding onboard sensor data into a trained model to output position and orientation data such as $x$ and $y$ coordinates. This approach circumvents the need for a 3D model or map of the surrounding environment, addresses the issue of arbitrary relocation, and does not require sequential data to make an accurate inference. Additionally, deep learning facilitates combining sensor data across different modalities, a process known as sensor or data fusion, in a manner that yields new and unique understanding of an environment not previously available from independent data.
In this thesis, we seek to construct a snapshot-based global localization neural network for a mobile robot operating within a simulated static environment. This is achieved by leveraging multimodal sensor fusion, specifically utilizing camera imaging and LiDAR data. Our model is designed to predict the agent’s current global $x$ and $y$ coordinates, along with its orientation, $\theta$, when supplied with input data in the form of images, LiDAR data, or both. For our network, we adapt a pre-trained version of the EfficientNet architecture, which forms the foundation of all the models we train. Additionally, we conduct experiments with various camera and LiDAR configurations, and we train models under two distinct sensor fusion strategies with the aim of achieving the most accurate localization.

In summary, the primary contributions of this thesis are as follows:

- We adapt a state-of-the-art machine learning network, thus enabling precise and accurate pose estimation for a static simulated environment when provided with images, LiDAR data, or both.

- We analyze the extent to which varying sensor configurations influence performance.

- We compare sensor-fusion strategies for snapshot-based localization models and optimize our neural network architecture to minimize inference error.

The remainder of this thesis provides essential background information, contextualization in the form of related works, implementation specifics for the development and evaluation of our models. Subsequently, it supplies outlines of our experimental design and analysis of our experimental results, concluding by proposing avenues for future research to build upon the work we have accomplished. Chapter 2 covers key background information regarding deep learning, transfer learning, sensors,
fusion, and localization. Chapter 3 provides an overview of contemporary research and studies investigating different fusion strategies and localization techniques. In Chapter 4, we explain steps we took to implement our design by describing our chosen simulation software, Webots, the data collection process, and the design and architecture of our network. After the implementation, Chapter 5 covers the experimental design by defining metrics to measure performance, independent variables we manipulate, and the testing and development environments employed during experimentation. In Chapter 6, presents an analysis of the results obtained from conducting the aforementioned experimentation and identifies patterns from recurrent behaviors observed. Chapter 7 suggests future opportunities and applications that can build upon our work, and Chapter 8 concludes with key takeaways.
Chapter 2

BACKGROUND

In this chapter, we cover key background information required to reap the most from our experimentation. Topics include high-level overviews of deep learning, transfer learning, sensors, sensor and data fusion, and the problem of localization. These background sections are in no way comprehensive but serve as a basis for defining technical terms and establishing components vital to our experimental design.

2.1 Deep Learning

In the broad, ever-expanding field of machine learning (ML), the subset of deep learning is concerned with mimicking the work of human brains through the implementation of complex neural networks. These artificial neural networks are generally composed of many layers, and layers are in turn composed of digital neurons (simple nodes of computation, consisting of parameters like input weights and biases). Though linear networks consisting of a single layer have proven useful, learning becomes “deep” upon introducing multiple layers to the network, enabling the solving of more complex problems.

Just in the last decade, deep neural network research has been leveraged to take leaps and bounds in the areas of computer vision, prediction, semantic analysis, natural language processing, information retrieval, and much more [40]. Deep learning’s rapid adoption has often been attributed to its adaptability, as its implementations are often not task-specific [1]. In other words, deep learning is seen as somewhat of a universal learning approach that can be applied to solve problems in a plethora of domains.
Previous attempts at learning had to be carefully hand-crafted, but modern solutions often result from allotting additional computation time and model complexity until the desired product is achieved.

2.1.1 Convolutional Neural Networks

Modern deep learning models are extremely versatile tools, but certain neural network architectures seem to excel over others in specific applications. One particular type of neural network that demonstrates exceptional performance in vision-processing tasks is the Convolutional Neural Network (CNN). CNNs adapt in learning spatial hierarchies of features from grid-like input data, like images or video. CNNs are widely used in the areas of computer vision, object detection, image classification, and image generation for their ability to learn patterns or features within an image.

CNNs are comprised of convolutional layers, activation layers, pooling layers, and then end with a fully-connected layer, but the convolutional layer is really what sets its architecture apart. The convolutional layer uses a smaller matrix called a filter that acts like a sliding window, getting multiplied with each element of the image matrix. These filters excel at highlighting edges or common geometric patterns in data where neighbor matrix elements share relations.

2.1.2 EfficientNet

In 2019, AI researchers at Google published their work on a powerful and efficient new CNN architecture, EfficientNet [43]. Previously, traditional CNNs were developed for a fixed resource cost, but when additional resources were made available, the architectures were scaled up arbitrarily. Scaling techniques often vary by scaling
depth, width, resolution of input images, or a relatively groundless combination of the three at the architect’s discretion, but EfficientNet takes a different approach. The base EfficientNet architecture, EfficientNet-B0, establishes a baseline CNN crafted to optimize both accuracy and floating-point operations per second (FLOPS). This baseline model can then be scaled up using EfficientNet’s novel compound scaling approach. Compound scaling maintains the optimal balance of width, depth, and resolution by uniformly scaling using a fixed ratio of scaling coefficients, resulting in a more computationally efficient network with minimized FLOPS. The compound scaling method can be visualized in Figure 2.2 [43].

### 2.2 Transfer Learning

The EfficientNet models are products of transfer learning, a starting point for machine learning where pre-trained models can easily be adapted and fine-tuned to cater to a distinct use case. This significantly saves training time while enabling the attainment of high-performing results not previously possible with limited training data and computational resources, as fundamental patterns have already been learned.
Transfer learning appears in various forms, including in fine-tuning the entire model or just specific layers, but for the purposes of this paper, the pre-trained weights of EfficientNet will be used as a starting point for identifying key features of a robot’s surrounding environment, ultimately resulting in superior performance and expedited convergence compared to methods that learn from scratch.

2.3 Sensors

Sensors in a robotic system are pivotal to understanding environments, and their multifarious nature complicates the process of understanding. Many sensors are only able to capture information in a single modality (see Section 2.4), so selecting a diverse collection of sensors for your system is essential for peak performance. Only sensors selected for use in this study are detailed below, as a full list is much more extensive.
2.3.1 LiDAR

Light detection and ranging (LiDAR) is both a process and sensor that quantifies the duration required for light, in the form of a laser, to reflect off the surrounding environment and return to a receiver. With the understanding of how fast light travels, this temporal measurement can easily be converted into a spatial distance. A single distance measurement offers limited utility, so the LiDAR laser typically emits rapid pulses at many different angles in a sweeping motion in order to collect a discrete collection of distances with increased density. This collection of distances can be visualized in three-dimensional space in the form of a LiDAR point cloud, a compilation of points plotted at corresponding collected distances that, when sufficiently dense, reveal contours of the surrounding environment.

The density of a LiDAR point cloud is influenced and defined by multiple factors. A LiDAR sensor’s horizontal field of view (FOV) is an angular parameter that determines the breadth of distances received around the singular point of data collection. A horizontal FOV of 360 degrees creates a point cloud encompassing the entire LiDAR sensor, offering a comprehensive view of the environment, whereas a 120-degree FOV results in a more constrained and directional point cloud, capturing merely a third of the surrounding environment but with potentially a more representative and accurate point cloud. Additionally, point cloud density is determined by the horizontal resolution, or the quantity of data points collected within a given horizontal FOV. These horizontal parameters are also defined vertically through the vertical FOV and vertical resolution, more commonly referred to as the number of LiDAR layers or the quantity of simultaneous laser beams emitted by the LiDAR sensor at various vertical angles. Increasing horizontal and vertical resolutions directly increases the density, but data storage and processing concerns may surface at significantly high resolutions.
One of the biggest drawbacks to using LiDAR sensors is simply monetary. Advanced LiDAR systems are significantly more expensive than traditional image alternatives like photogrammetry, but the benefit comes in the form of highly accurate data. Depth data can be useful for altimetry, autonomous vehicles, high-resolution mapping, and much more where precise depth understanding is imperative.

2.3.2 IR and Ultrasonic Distance Sensors

Infrared (IR) and Ultrasonic sensors are both similar to LiDAR in that they both detect distances. Ultrasonic sensors are operationally closest by calculating distances via reflections, however, they use high-frequency sound instead of light. Distance is computed using the speed of sound through air (343 meters per second) and the amount of time it takes the sensor to receive an echo from the sound pulse it creates. Conversely, Infrared sensors do not rely on timing, but rather the intensity of reflections. IR sensors use an IR LED to emit light, and the intensity of that radiation is then detected by the sensor’s IR photodiode. Both Ultrasonic and IR sensors are useful in their respective applications, but for the purposes of this paper, they will only have limited uses in the data collection process. The distance data is relatively limited compared to LiDAR, typically capturing only two dimensional information instead of three, but this amount of information is enough to aid a robot in navigating its environment by indicating obstacles as it approaches them.

2.3.3 Vision

When capturing images from cameras, vision can be described based on the image quality, the number of sources used to produce an image, the methods behind how any synthesized or fused images were produced, and much more. Two common types
of vision include binocular and monocular vision, both differentiated by the number of image sources and how each source’s view relates to other views.

In binocular vision, multiple sources of image data (cameras, human eyes, etc.) are able to focus on the same subject, or the same subject appears in the field of view for more than one source. We experience this personally when we decide to look directly at something in front of us, as the subject can be seen with both eyes. Our brains have the ability to stitch the two perspectives returned by each individual eye into one unified image, appropriately filtering out duplicate data as if there was just a single occurrence.

The contrast of this binocular effect is seen when we close one of our eyes, leading us to experience monocular vision. Monocular vision is most often the use of one eye or camera as the only source of image data. Though simple to use, monocular vision (in general) yields a more restricted understanding of the environment and surroundings in both field of view and depth understanding when compared to binocular vision [8]. A unified image produced from multiple perspectives naturally has the ability to cover more of its surroundings than a single image, and having two images taken from two physical poses allows for depth estimation via triangulation. This ability to estimate depth by relying on binocular disparity, or the difference in vision source perspectives, is called stereoscopic vision [4]. Sometimes referred to as binocular stereopsis, stereoscopic vision leverages the fact that disparity between images from different poses or perspectives is inversely proportional to depth, but seeing that monocular vision is constrained to a single perspective, depth is not as easily estimated.

For the purposes of this paper, an expanded form of monocular vision will be explored while stopping short of what we have defined as binocular vision. This paper will explore training multimodal ML models using multiple cameras, but each camera operates independently. This lack of fusion between cameras in a two-camera system
is called bimonocular vision or two-eyed monocular vision, and this paper extends this idea up to a four-camera system for a form of four-eyed monocular system. Though the independent fields of view struggle to estimate depth, this paper hopes to highlight the extent to which additional access to LiDAR data aids in the global localization problem [39].

2.4 Sensor Fusion

The sensors described in Section 2.3 each have unimodal access to unique information about the environment, but in order to develop a more complete picture of the surrounding space, more information is required. Without access to a multimodal sensor, data from different unimodal sensors can be compounded to achieve comparable results. Similar to increasing dimensionality in a plot or drawing, this process, called sensor fusion, reveals a more complete picture of the desired subject through synthesizing individual pieces of the puzzle into a complete, unified, multimodal view of the surrounding environment [29, 36]. Such unification addresses the problem of sensors only obtaining insight for specific attributes or dimensions of the environment while also addressing the problem of receiving noisy data from one or more individual sensors. Data or sensor fusion has been accepted as a common practice used to achieve four types of gain, defined by researchers as the following [6]:

- *Gain in representation:* The data resulting from the fusion process has increased granularity or dimensionality that provides a richer meaning or semantic on the data than each of the initial data sources.

- *Gain in certainty:* The data resulting from fusion yields increased certainty when compared to the joint probabilities of the initial datasets.
• **Gain in accuracy:** The fusion data has reduced variance when compared to the initial data. Data fusion reduces noise by suppressing erroneous data, and accuracy is gained with redundant and concordant data.

• **Gain in completeness:** New information resulting from the fusion process helps develop a more complete view of the environment.

An example of a gain in representation and completeness would be the fusion of LiDAR and vision data, or simply LiDAR-vision fusion. On its own, LiDAR acts as a superb depth sensor but lacks the ability to detect appearance beyond shape, while a monocular camera can clearly recognize appearance but is inadequate at determining distance or depth. Together, through various compounding and synthesis techniques, the data from these sensors paints a more complete picture of the surrounding environment, consisting of both detailed depth information and appearance.

There are many ways to go about fusing sensor data. Many visual approaches simply overlay information on top of each other. Others compute similar findings or key differences from varying modes to leverage. For the purposes of this paper, I will focus on sensor fusion as it pertains to deep learning. In a deep neural network, data can be fused in different layers, and by concatenating modes with enough processing to follow, the model is likely learn correlations if they exist. Deep learning allows for a less hand-crafted approach to learning, but is effective nonetheless.

### 2.4.1 Fusion Strategies

Two commonly implemented fusion techniques are late and early fusion. Late fusion, the more common and simple fusion approach, fully processes each unimodal stream of data separately before merging modes [23]. Late fusion excels at identifying key features embedded in unimodal data and then synthesizing collections
of disconnected features to yield a multimodal semantic representation [42]. Lever-
age unimodal processing streams allows for more targeted architectures, tailored
to modality-specific properties, but a significant drawback of the technique is the
learning expense resulting from having supervised learning stages for each individual
modality. In general, outputs from each unimodal processing stream get combined or
fused into a multimodal semantic representation through concatenation or averaging,
and this representation requires its own final supervised learning stage.

Early fusion on the other hand strives to merge modalities near the input to the
network. This is also referred to as feature-level fusion, as it fuses different inputs
before any feature extraction and learning. Early fusion helps expose relations and
patterns between modalities at an early stage that might otherwise get suppressed
or go unnoticed when processed separately with late fusion. The main drawback to
combining modalities early is physical: Sampling rates often differ among sensors,
and all data must be available at the time of processing, so the bottleneck is the
least-available modality.

In the middle of early and late fusion lies an emerging type of fusion. This fusion,
referred to as halfway fusion [31], middle fusion [17], intermediate fusion, and cross
fusion, combines modalities after some features have been extracted but before the
classification stage. The effectiveness and use cases for this type of fusion are still
largely being explored. This paper will focus more on the contrasted effectiveness of
early and late fusion techniques.

2.5 Localization

Robot localization is defined as the process of determining where a mobile robot is
located with respect to its environment, and in most scenarios, localization includes
estimating both robot position and orientation [26]. Robot localization is often encompassed with the phrase pose estimation, a broader term referring to estimating the position and orientation of the robot, part thereof, or object relative to the robot.

There are three commonly accepted classes of robot localization: Global localization, position tracking, and the kidnapped robot problem [37]. For the purposes of this paper, we will be focusing on the class of global robotic localization, implying that the robot does not possess any knowledge on its initial position, and the robot strives pinpoint its pose within its environment. In the global localization problem, the robot typically is equipped with a map or similar understanding of the surrounding environment as well as sensors to observe its surroundings, but our implementation leverages a neural network instead of a map. More about the implementation specific to this paper is detailed in Chapter 4.

In addition to being labeled one of three major classes, localization can be either active or passive [11]. Active localization classifies a localization technique where the robot’s motion or sensor orientation is influenced by what is likely to provide the most insight and aid in the localization process, while passive localization a less involved process, and the potential for insight has no effect on robot behavior. This paper will only be exploring instances of passive localization.
This chapter discusses recent studies that contend data fusion strategies and localization techniques that our experimental design and implementation are founded on and adapted from.

3.1 Image-Based Localization

By inspecting existing localization methods that do not leverage multimodal sensor fusion, we are able to develop a relative baseline for our model’s performance. Looking at strictly image-based localization techniques, we observe impressive performance from researchers at the Visual Learning Lab at Heidelberg University [10]. Using a fully convolutional neural network for dense scene-coordinate regression, their models were able to achieve accuracy within 5cm and 5° at over 60% of the time, consistently, when trained without access to a 3D model. Their methods rely on discovering 3D scene geometry automatically, and all from a single RGB image.

PoseNet, an approach from three years prior, outlines similar performance for 6-DOF camera pose regression [27]. PoseNet is an example of transfer learning, transferring understanding from advanced classification models, that achieves 0.5m and 5° accuracy indoors while remaining robust to suboptimal lighting and motion.

Additionally, using a bag-of-words approach to learn place appearances, the FAB-MAP approach developed in 2008 offers a probabilistic solution for place recognition [15]. In addition to localization, this method is able to determine when it is
experiencing a previously unseen environment, proving useful in dynamic or expansive environments.

3.2 Early vs. Late Fusion

The benefits that come from leveraging early and late data fusion are empirically evident [12, 44, 46], but generalized rules for selecting one method over the other are still being explored and depend on the specific application.

In a 2020 study comparing fusion strategies for 3D object detection, researchers found that early fusion outperformed late fusion when correctly classifying and annotating 3D bounding boxes [7]. Using the three most prevalent classes in the KITTI dataset [25] ("Car", "Pedestrian", and "Cyclist"), early fusion yielded only a 2%-4% improvement over a baseline LiDAR-only control according to mean average precision (mAP) results. As seen in Figure 3.1 and Figure 3.2, the fusion strategies consisted of concatenation either before or after feature extraction. Additionally, the early and late fusion architectures had 32ms and 34ms inference times per point cloud respectively. The difference between these times is fairly negligible, but both at least doubled the inference time when compared to the baseline LiDAR implementation.
3D object detection is just one of many areas of study seeking to find which fusion strategy performs best. In general, studies comparing fusion strategies show that early fusion thrives when modalities are highly correlated or temporally aligned, whereas late fusion excels when features can be extracted with a degree of modality separation and modalities are more temporally discrepant [21,23].

3.3 Localization Techniques

As previously stated in Section 2.5, robot localization is the process of estimating a mobile robot’s location and orientation by observing its surroundings with sensors [26], but current solutions continue to be optimized through copious localization techniques. Some modern techniques include landmark-based localization, SLAM, and OneShot global localization, all detailed below.

3.3.1 Landmark-Based Localization

A common localization technique is landmark based, where the robot is able to leverage a distinguishable object or landscape feature in its surroundings to determine its
pose. Avgeris et al. implement a single-camera system to test this method, achieving satisfactory results. In their implementation, the mobile robot calculates its position and orientation based on the robot’s perceived position relative to two arbitrarily positioned landmarks with positions known *a priori* [3]. Using a bilateration method for the robot’s position and projective geometry for the robot’s orientation, the autonomous agent is able to self-localize. This landmark-based localization was able to achieve an estimated orientation within 6 degrees of the ground truth on average, and this result serves as a goal we seek to improve upon with our neural network approach.

### 3.3.2 SLAM

Simultaneous Localization and Mapping (SLAM), a term first coined in the mid-1990s paper "Localization of Autonomous Guided Vehicles" [20], is a technique used to solve the localization problem in unknown environments. In general, a mobile robot using SLAM strives to incrementally build a consistent map of an unknown environment while simultaneously localizing itself within the map it has built [5,19]. SLAM continues to evolve today through continued innovation: Google has released Google Cartographer as its own open-source LiDAR-based SLAM technology [28], vision-based SLAM employs advanced computer-vision techniques [22], and fusion SLAM has been thoroughly explored for extracting the most useful information from various sensor inputs [18].

Part of SLAM’s success is due to its temporal dependence. SLAM is built to be a sequential process where a mobile robot’s next pose estimation is influenced by its previously estimated position and orientation. Our paper explores snapshot-based models as an alternative to the empirically proven SLAM method for certain use cases. Though not attempting to map and localize simultaneously, neural networks
can be more computationally efficient than SLAM. Additionally, the neural networks used in this paper do not possess the same temporal dependence required by SLAM enabling localization accuracy to be independent of previous estimations.

### 3.3.3 OneShot Global Localization

As seen in SLAM, many effective pose estimation solutions include a temporal element, leveraging input over time or previous estimations to influence the next, but the global localization problem is further complicated when only a single scan is used. Ignoring temporal context and previous estimations greatly inhibits a neural network’s ability to accurately and consistently localize, but accurate results can still be achieved through state-of-the-art techniques.

The OneShot algorithm, proposed in 2020, estimated a robot’s 6-DOF pose using only a single 3D LiDAR scan [38]. The proposed technique relied on sparse 3D point cloud segmentation. Segmentation is a common approach for extracting information from sparse point clouds, but OneShot was able to leverage fusing visual appearance via camera imaging with their single-scan segmentation to boost the performance of their segment descriptor. Through LiDAR-vision fusing, OneShot was able to improve segment descriptor retrieval rates by over 17% when compared to a purely LiDAR baseline. OneShot’s image fusion is dependent on the Vector of Locally Aggregated Descriptors (VLAD) layer found in the NetVLAD neural network they leveraged for producing image global descriptors [2].

### 3.3.4 Pose Estimation from a Single Image

A similar pose estimation technique proposed by Lee et al. used a single RGB image to estimate an external camera’s pose with respect to the robot [30]. Though the
robot in this study was not mobile, the single image was processed by a deep neural network to identify keypoints relevant to the robot’s orientation. As an extension, Lee et al. explored the effect of using additional frames from the same static pose, demonstrating a significant error reduction that this paper explores as well but with reported performance metrics.
In this chapter, we strive to outline the physical processes through which our experimentation was conducted. This includes detailing the chosen simulation environment, Webots, the data collection process, and key network design choices for our localization model’s architecture.

4.1 Webots

For all simulations, we used Webots, a popular open-source 3D robot simulation software maintained by Cyberbotics Ltd. used for education, research, and industrial development [32, 33]. Users are able to model, program, and simulate robots using the software’s realistic physics engine, a fork of the widely used open-source Open Dynamics Engine (ODE) [41].

One of the main benefits of Webots is it allows users to directly synthesize realistic sensor values from robotic simulations. This includes the camera and LiDAR sensor values we desired, meaning that all LiDAR-vision fusion experimentation could be conducted via simulation. The realistic Webots sensor values can make the process of exporting code and trained models to a physical system relatively seamless. Real-world physical environments can be created within Webots using the world editor, modifying existing object models or starting from scratch, or the user can leverage existing world files (or environments) if a generalized model is desired.
For the purposes of this paper, three different Webots environments were used and slightly modified. We have given them titles listed below along with their original Webots world file names.

- Square: empty.wbt
- Partial Apartment: apartment.wbt
- Complete Apartment: complete_apartment.wbt

For preliminary research and testing, the default Webots world, which we refer to as Square, was used. This world file was initially an empty 4x4 grid with walls, but as seen in Figure 4.1, additional object models were imported and used as landmarks in initial localization tests. The need for these objects came about after initial tests showed poor localization attempts resulting from training with seemingly indistinguishable input data. Figure 4.2 depicts the intermediate world file used for localizing in a larger, slightly more complex environment representing a small apartment possessing a living room and kitchen. The primary and most complex Webots world used was the Complete Apartment as rendered in Figure 4.3. This two-bedroom, two-bath apartment with a full kitchen, dining and living area, hallway, and staircase to a nonexistent second story provides both a larger environment to localize within while increasing the density of unique landmarks that aid the localization process.

4.2 Data Collection

In order to train an accurate and precise global localization model, we collected an abundance of training, validation, and testing data within the Webots environments using a mobile robot agent. The process of data collection has profound implications
for the overall performance of the model, so collecting useful and representative training data is just as important as the network architecture and training processes. For the purposes of this paper, one of the mobile agent’s tasks was to gather datasets of camera images and LiDAR data collected at random coordinates and orientations throughout the its environment. The goal was for the collected dataset to be uniformly representative of the environment, maintaining roughly equal data density throughout the environment when possible.

4.2.1 Mobile Agent

To gather consistent data, a mobile agent was placed and driven around the environment. For this task, we chose the GCtronic e-puck, a small mobile robot developed at the Swiss Federal Institute of Technology Lausanne for educational purposes [35]. The original e-puck used for data collection (which has since been improved upon [24]) is a differential wheeled robot driven by step motors on each of its left and right sides, allowing the robot to turn in-place. This e-puck’s tight-turning mobility, compact
design, and broad academic adoption [14] were the only major criteria for selecting a mobile agent base, and it should be noted that there were alternatives to the e-puck that would have satisfied these principles as well. No pre-built Webots model possessed all necessary peripherals, so modifications to existing designs were necessary.

4.2.2 Robot Peripherals and Modalities

The baseline e-puck model is equipped with eight infra-red proximity sensors, a single front camera, an accelerometer, and other useful mobile robot peripherals. The layout of many of these peripherals is visualized in Figure 4.5. Though the proximity sensor data was not collected, its use is detailed in Section 4.2.3. Required modifications to the robot included adding a central GPS, compass, LiDAR sensor, and three additional cameras facing the robot’s left, right, and rear. Due to the near-exhaustive detail of Webots simulations, most sensor parameters equipped with their default
values, but the main peripherals of interest, the cameras and LiDAR sensor, have parameters detailed in Table 4.1 and Table 4.2. Additionally, some scaling of the peripherals was required in response to scaling up the e-puck for faster data collection, but all resulting sensor data remained consistent between scaled and default datasets.

4.2.3 Robot Controller

The Webots Robot object possesses a String field for the name of the robot’s current controller. The default controller for the e-puck is the "e-puck_avoid_obstacles" controller, which simply uses the proximity sensors to stop the robot and pivot within a
certain distance from an object. Otherwise, the e-puck drives straight. This behavior was insufficient for data collection due to the relatively stiff orientation when the robot drove straight, and the lack of random influence led to predictable pathing or cyclical behavior.

I developed two robot controllers for the purpose of collecting data. The first and more straightforward robot controller was for manual data collection and consisted of controlling the e-puck’s movement with the arrow keys. This controller proved more useful for inference however, testing specific coordinates and orientations within the environment that were thought to be underrepresented. The second controller was used for automated data collection. This controller had no human input, dependent
only on onboard sensors to explore the surrounding environment. The logic is outlined in Figure 4.6, and key terms describing the controller’s behavior are defined below.

- **Sensor Reading:** At this state, the robot reads its proximity sensors. If any sensor reads beyond a certain threshold (indicating a close object), a randomized Obstacle Avoidance duration is set, entering the Obstacle Avoidance state for 25 to 100 cycles. Otherwise, the robot enters the Free Roam state.

- **Free Roam:** Both motor speeds are set to full in the forward direction, and the next state is set to be Sensor Reading.

- **Obstacle Avoidance:** Based on a left or right obstacle flag (indicating the side where the obstacle was detected) set in the Sensor Reading state, the robot performs a tank turn away from the obstacle.
Figure 4.6: Automated Data Collection Robot Controller Flowchart
• **Random Course Adjustment:** Each cycle, the robot has a 2.5% chance of taking a slight turn in the direction of the robot’s current tendency (either left or right). This tendency introduces variation into the robot’s orientation, as the robot seems to drive in a slight arc.

• **Random Reverse:** Each cycle, the robot has a 0.01% chance to reverse for a random number of cycles between 150 and 200. The intention here is to account for scenarios where the robot may have gotten itself stuck or navigated into a place it is not able to turn in.

• **Random Tendency Change:** Each cycle, the robot has a 0.01% chance to toggle the current tendency direction. If the robot currently tends left, now tend right, and vice versa.

### 4.2.4 Data Samples and Interface

The Webots live rendering makes data inspection and quality assurance quite simple. Using LiDAR point cloud optional rendering, we can visualize the distance values that the LiDAR sensor is collecting. Figure 4.7 depicts the e-puck robot in one of the bedrooms in the Complete Apartment environment, and the robot is seen surrounded by points colored by the distances they represent. Closer distances are bright red, and further distances are darker blue. Figure 4.7 specifically depicts many of the surrounding structures and objects, like the bookshelf, bed frame, and large chair, in relatively high resolution, using 128 LiDAR layers.

In addition to visualizing LiDAR, the Webots simulation interface allows the user to overlay any camera imaging onto its main display. Seeing as the e-puck used for our experimentation was equipped with 4 cameras, we were able to overlay all 4 views. A
sample of these views are seen in Figure 4.8, where you can see a preview of images the robot is capturing.

These different views can all be seen live through the Webots live render. The overlay of the 4 perspectives is pictured in Figure 4.9 along with a post-capture annotation of a circle and arrow to better visualize the robot’s position and orientation at the time of capture.

4.2.5 Datasets

The automated data collection robot controller was used to generate datasets for the Empty, Partial Apartment, and Complete Apartment environments using various configurations of LiDAR layers, cameras, and data collection frequencies. The datasets collected and used are summarized in Table 4.3, and each dataset is composed of training, validation, and testing subsets, each represented as 70%, 10%, and 20% of the total dataset size respectively. For each dataset, the total number of images is equal to the product of the number of cameras and the total collection points,
Figure 4.8: Data Sample: e-puck Front (Top-Left), Rear (Top-Right), Left (Bottom-Left), and Right (Bottom-Right) Views

Figure 4.9: Webots Camera Imaging Overlay with Robot Location and Orientation
Table 4.3: Dataset Summary

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Environment</th>
<th>Collection Type</th>
<th>Cameras</th>
<th>LiDAR Layers</th>
<th>Collection Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>Square</td>
<td>Manual</td>
<td>1</td>
<td>None</td>
<td>3,800</td>
</tr>
<tr>
<td>A</td>
<td>Partial Apartment</td>
<td>Manual</td>
<td>4</td>
<td>4</td>
<td>3,800</td>
</tr>
<tr>
<td>B</td>
<td>Partial Apartment</td>
<td>Auto</td>
<td>4</td>
<td>4</td>
<td>6,250</td>
</tr>
<tr>
<td>C</td>
<td>Partial Apartment</td>
<td>Auto</td>
<td>4</td>
<td>Raised</td>
<td>6,250</td>
</tr>
<tr>
<td>D</td>
<td>Partial Apartment</td>
<td>Auto</td>
<td>4</td>
<td>32</td>
<td>6,250</td>
</tr>
<tr>
<td>E</td>
<td>Complete Apartment</td>
<td>Auto</td>
<td>4</td>
<td>32</td>
<td>25,000</td>
</tr>
<tr>
<td>F</td>
<td>Complete Apartment</td>
<td>Failed</td>
<td>4</td>
<td>128</td>
<td>25,000</td>
</tr>
<tr>
<td>G</td>
<td>Complete Apartment</td>
<td>Auto</td>
<td>4</td>
<td>128</td>
<td>25,000</td>
</tr>
</tbody>
</table>

whereas the LiDAR datapoint count is equal to the product of the LiDAR layers, the LiDAR horizontal resolution (512), and the total collection points.

The goal for each dataset was to uniformly collect images and LiDAR data that was fully representative of the surrounding environment. Without balanced representation from many collection points within each world, any localization models trained from the data would develop strong biases towards where their data collection was more dense. By visualizing different distributions of the data, we can predict the potential for our data to develop a bias.

Take the Complete Apartment environment for example. Figure 4.10 shows an aerial view of the apartment, and Figure 4.11 depicts the data collection path that the mobile robot took for the Dataset E testing set. In the data collection figure, points were plotted at each data collection point, and curved lines were used to connect the points as they were collected sequentially. The color spectrum represents the collection over time, and from this, we can observe that the mobile robot was able to fully navigate the Complete Apartment, as the resulting plot clearly outlines the Complete Apartment floor plan. Note that in areas where the data collection points are less dense, there are generally significant obstacles present, like in the top left of Figure 4.11, the empty white spots represent the dining table and chairs. It appears
the robot was able to capture many different orientations at similar coordinates as well, represented by all the curved lines that intersect.

But the distributions of data can be further unpacked. Figure 4.12 contains three different plots with various distribution analysis data. The first is a heatmap displaying the densities of coordinates where data was collected. Each tile on the heatmap is a square half-meter, and data that is more dense in a given tile is depicted as brighter. The ideal dataset heatmap would have equal frequencies across each tile, but randomized data collection introduces a slight variance. Relatively, with the exception of a few tiles placed in corners and of the apartment, the density of data in Figure 4.12 appears distributed evenly among drivable areas. The last two plots in the figure depict the x and y distributions among the data collected. These histograms repre-
Figure 4.11: Complete Apartment Data Collection Path, Testing Set

sent the frequency of data collection points that fall within different $x$-coordinate and $y$-coordinate bins, and histograms for an unbiased dataset are ideally uniform. As seen in the plots, with the exception of specific dips and spikes, the data is relatively uniformly distributed, noting that dips and spikes in certain bins can be accounted for by comparing coordinate values with Figure 4.10 and Figure 4.11. These views reveal that spikes and bins result from significant increases or decreases in drivable areas, and this happens when the coordinates fall on walls or significantly open spaces.
Figure 4.12: Complete Apartment Data Distributions, Testing Set
Every $C$ cycles, the robot took in sensor input and wrote data to a local directory. The number $C$ was proportional to how long the data collection process ran, and in theory, the larger $C$ was, the more random the dataset, as the difference between sequential data collection points would be greater. All images were collected in standard definition, 720x480, and stored with the following naming scheme: 
"[x-coordinate]_[y-coordinate]_[orientation in degrees]_view.jpeg". An example might be "-0.356.-4.0921_260.3_front.jpeg". For the LiDAR sensor, distances were written to a CSV in the form of a one-dimensional list of floats along with all ground truths for location and orientation in adjacent columns. Pre-processing of the LiDAR data included replacing all "Inf" values recorded (where the recorded distance exceeded
the "maxRange" parameter, commonly set to 10 meters) with the maxRange value itself.

4.3 Network Design and Architecture

Once all of the input data has been acquired, it is fed into the global localization neural network. As seen in Figure 4.14, the first step reduces the dimensionality of both the LiDAR and vision data. The LiDAR data, represented as a 1D tensor of floats, is passed through a linear layer, reducing the dimensionality from 2048, 16384, or 65536 down to 8, where the input dimensionality depends on how many LiDAR layers were recorded during data collection. This dimensionality reduction can be used to extract features from the LiDAR data before being passed to either an early or late fusion stage. Concerning the vision data, up to 4 concatenated images are passed into a 1x1 convolutional layer for channel reduction. The input channel count is equal to three times the number of images concatenated, and the convolutional layer reduces the data to a three-channel tensor which the main pre-trained network expects.

4.3.1 Pre-trained Network: EfficientNet

The choice of which pre-trained network to adapt for this localization task was made with efficiency and state of the art performance in mind. As detailed in Section 2.1.2, EfficientNet exhibits strong accuracy and efficiency scores due to its reduction in parameter count and reduced FLOPS, making it a prime candidate for a high-speed inference model.
As seen in Figure 4.15, EfficientNet is extremely effective in minimizing the number of required parameters while maintaining state-of-the-art accuracy. Though the most accurate EfficientNet-B7 architecture was compatible with the workstations used for training (see Section 5.4), EfficientNet-B5 was selected for requiring less than half of the parameters that EfficientNet-B7 required (30M vs 66M) while sacrificing very little accuracy. In addition, early training and testing was conducted on EfficientNet-B0 for rapid results, as B0 has only 5.3 million parameters, less than 20% of the B5 architecture.

The EfficientNet-B0 and EfficientNet-B5 architectures produce 1024 and 2048 outputs respectively, which are then fed into a final fully connected layer.

4.3.2 Network Output

The final stage of the model is a fully connected layer, leading into four outputs. The number of inputs to the final layer depends on both the fusion strategy and
Figure 4.15: Model Size vs. ImageNet Accuracy, Taken from Tan and Le [43]

the base EfficientNet model used (which was most often EfficientNet-B5), but the output quantity is consistent. The four values that the model predicts are the robot’s \( x \)-coordinate, \( y \)-coordinate, the \( x \)-component of the orientation unit-vector, and the \( y \)-component of the orientation unit-vector. By separating the orientation into its unit-vector components, we avoid the wraparound or circular problem, where a model develops a bias at the threshold where 360 degrees turns back into 0 degrees as you traverse around a circle. This stark contrast, going from 360 degrees back to 0, creates a false sense of substantial error across the threshold, which a machine learning model has difficulty understanding.

4.3.3 Late Fusion Architecture

When training a late-fusion model, the LiDAR and image data are processed separately. The LiDAR data is processed by the LiDAR layer and reshaped to have 2 dimensions, while the image data is passed through the EfficientNet base then concatenated. The reason for this separation is due to the nature of how EfficientNet
pre-trained models were trained. EfficientNet was originally designed for processing images, and the pre-trained weights were not trained for LiDAR data, so high performance would not be expected. This decision also saves space, as passing LiDAR through its own EfficientNet network in addition to the image data EfficientNet network would almost double the trainable parameters and storage size.

4.3.4 Early Fusion Architecture

Under an early fusion architecture, the LiDAR and image data are concatenated and then processed together. Before concatenation, the LiDAR data dimensionality is first reduced with the LiDAR layer for consistency. Then the LiDAR data is expanded to have the same spatial dimensions as the image data, which can then be concatenated along the channel dimension. Once concatenated, a 1x1 convolution is applied to reduce the overall channel count to 3, treating the fused data as an image to be passed through the main EfficientNet model. The output is then flattened and passed into the fully connected layer.
Chapter 5

EXPERIMENTAL DESIGN

5.1 Performance Metrics

Due to the continuous nature of the trained model output, inference could not be judged by simple binary correctness. The primary metric is instead measured as the average difference between the predicted pose and the ground truth pose. This error metric is composed of the individual average differences for the $x$ coordinate, $y$ coordinate, and $\theta$-orientation in degrees, or $\Delta x$, $\Delta y$, and $\Delta \theta$ respectively, and the goal in training our prediction model is to minimize this error.

One limitation of comparing the raw average differences is attempting to compare results for metrics with different distributions. Though the $x$, $y$, and $\theta$ data roughly resemble uniform distributions, the domains are very different (Ex: 12 meters for the $x$ coordinate and 360 degrees for $\theta$). To address this, the errors can be standardized. Because the training and testing datasets attempt to fully represent the virtual environment for all $x$, $y$, and $\theta$ dimensions, we assume roughly uniform distributions across all three variables and can standardize results by dividing the average observed error by the expected error for a randomized $x$, $y$, or $\theta$. This standardization technique also gives us a reasonable comparison to a realistic randomized position and orientation baseline.

This standardization process differs depending on the error being standardized. For example, given the $x$-axis and a domain length $L$, the expected distance between two random points on that axis based off a uniform distribution is a solved problem [13].
Table 5.1: Expected Distances by Environment

<table>
<thead>
<tr>
<th>Environment</th>
<th>Env. Size</th>
<th>$E[\Delta x_{rand}]$</th>
<th>$E[\Delta y_{rand}]$</th>
<th>$E[\Delta \theta_{rand}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>1.0m x 1.0m</td>
<td>0.3m</td>
<td>0.3m</td>
<td>90°</td>
</tr>
<tr>
<td>Partial Apartment</td>
<td>9.9m x 6.6m</td>
<td>3.3m</td>
<td>2.2m</td>
<td>90°</td>
</tr>
<tr>
<td>Complete Apartment</td>
<td>12.0m x 12.0m</td>
<td>4.0m</td>
<td>4.0m</td>
<td>90°</td>
</tr>
</tbody>
</table>

The expected distance is one-third the domain or length of the distribution and can be represented as the following:

$$E[\Delta x_{rand}] = \frac{L}{3} \quad (5.1)$$

This standardization principle is the same for the $y$ dimension but differs slightly for $\theta$. To standardize the distance between two angles, we set our baseline to be the expected minimum angle between two unit vectors. This is also a solved problem. The expected minimum difference between the two vectors is 90° [34]. These expected distances are organized by environment and pose parameter in Table 5.1, where the $x$ and $y$ expected distances are calculated by dividing the corresponding environment dimension by 3, and the $\theta$ expected distance is consistently 90° due to the unchanging 360° domain.

Additionally, we define weights to scale the standardized average differences appropriately. The combined error was designed to have position and orientation errors have equal influence, each scaled at 50% of the overall combined error value. Both the average $x$ coordinate difference and the average $y$ coordinate difference were weighted equally within the position error component, as marked by the coefficients in Equation 5.2. This equation can be further reduced to show that the combined error is composed of 25% standardized $\Delta x$, 25% standardized $\Delta y$, and 50% standardized $\Delta \theta$. The combined error metric will be used for both mean and median error values, where
the median error will be considered when faced with appreciable outlier quantities and values. Additionally, we equate a reduction in error to an increase in accuracy, and throughout our analysis, we will equate standardized error comparisons with accuracy comparisons, only conclusions will be opposite in nature due to the opposed definitions of error and accuracy.

\[
Combined\ Error = \frac{1}{2} \left( \frac{\Delta x}{2E[\Delta x_{rand}]} + \frac{\Delta y}{2E[\Delta y_{rand}]} \right) + \frac{\Delta \theta}{2E[\Delta \theta_{rand}]} \tag{5.2}
\]

A secondary metric worth considering for real-time systems is the inference time. This is simply the average time it takes data to completely pass through the network and produce a prediction. For the purposes of this paper, we do not alter the network to optimize the inference time beyond selecting EfficientNet as the network’s base, as the network’s primary goal is accurate localization, but inference time may be reported for purposes of replicability or future work.

5.2 Independent Variables and Parameters

After establishing the network to process data, we can now contrast fusion strategies. Through various network ablations and sensor configurations, we attempt to address the following research questions:

1. To what extent does isolation of information available to a localization system affect performance?

2. For a localization network leveraging both LiDAR and vision input, does the selected fusion strategy significantly affect performance?
5.2.1 Experiments for Identifying Batch Size Influence

Before answering any of the previously stated research questions, a preliminary test was conducted to establish the most effective training parameters and conditions for producing the most accurate localization models. The test was conducted under controlled and identical hyperparameters with the exception of batch size during training. The purpose of this test was to observe the influence that batch size had on the trained model’s performance, and thus determine the optimal batch size for training our localization model.

5.2.2 Experiments for Individual Sensor Contribution

In order to get proper baselines for data-fused localization performance, we conducted tests for vision-only and LiDAR-only systems. These baselines serve as the standards that the fused models attempt to improve upon.

For the vision-only system and late fusion architecture, we conducted experiments to determine the optimal number of cameras to be used in the data-collection and training process. Four cameras were mounted on the e-puck robot, each facing either the robot’s front, rear, right, or left. Data collection consisted of saving images from all four viewpoints simultaneously, but the data loading process filtered out images from unused cameras. The configuration for a system intended to use less than all four cameras can be found in Table 5.2.

The vision experiments with varying camera quantities aimed to visualize the expected diminishing return on accuracy by increasing the number of available views. It was thought that having additional information available during training would
Table 5.2: Vision Data Views by Camera Count

<table>
<thead>
<tr>
<th>Cameras</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Front</td>
</tr>
<tr>
<td>2</td>
<td>Front, Rear</td>
</tr>
<tr>
<td>3</td>
<td>Front, Rear, Right</td>
</tr>
<tr>
<td>4</td>
<td>Front, Rear, Right, Left</td>
</tr>
</tbody>
</table>

provide auxiliary information to the robot, but the degree to which this added knowledge was unknown.

For the LiDAR-only system, we conducted experiments to determine the optimal quantity of LiDAR layers to be used for future data-fusion models. In these experiments, the 512 horizontal resolution was kept constant, but we varied the total number of LiDAR layers used. The goal of this experiment was to visualize the diminishing return on accuracy by increasing the amount of LiDAR data available to the model and establish the optimal number of LiDAR layers to test for future sensor-fused configurations. The data collection process collected up to 128 LiDAR layers, and the data loader limited the layer count to 4 and 32 for tests requiring less than 128 layers. In the case of limiting LiDAR layers, the data loader simply read in every \(i\)th layer, where \(i\) was defined as the total number of collected layers divided by the desired number of layers. Most datasets collected LiDAR scans with at most 32 layers due to the storage cost of storing 65,536 floating-point numbers (128 layers \(\times\) 512 horizontal resolution) at each collection point, which resulted in reading data in from a 25GB CSV file.

After performing experiments for both vision and LiDAR baselines at varying camera and LiDAR layer configurations, we then conducted experiments comparing a simple fusion of the two modalities to their individual performances. This experiment set out to prove that under the selected LiDAR and vision configurations, fusion of the
two datasets would produce greater understanding of the environment than that of either sensor individually, and thus improve localization performance.

5.2.3 Experiments for Comparing Early and Late Fusion

In an effort to identify the LiDAR-vision configuration that minimizes localization error, both early and late data-fusion strategies were explored and tested. To keep the results comparable, the storage impact of each network architecture was constrained to be the same. For this experiment, both an early fusion model and a late fusion model were trained and tested using the same training and testing datasets. Performance was then measured by recording average absolute errors for the $x$, $y$, and $\theta$ values, then computing the standardized error score for each model. The model yielding the lowest standardized error would be identified as the model with the most effective fusion strategy for this scenario.

The early and late fusion strategy architectures are outlined in Figure 4.14. For the early fusion architecture, both the LiDAR and image data are preprocessed and reshaped for concatenation before being passed through EfficientNet. For the late fusion architecture, the LiDAR data is processed independently and concatenated later in the model with the output of EfficientNet, the product of purely image data.

5.3 Inference

As soon as each of the localization models were trained, they were tested by repeated inference on the corresponding testing set. The testing set comprised 20% of the overall dataset and was used exclusively for testing fully trained models, but the models were also tested manually through live inference in Webots.
All three Webots environments were modified to include one green sphere object and a set of three blue sphere objects for the sake of visualizing live inference. The green sphere was intended to indicate the agent’s predicted location by hovering a few meters above the $x$ and $y$ coordinate pair, whereas the blue spheres trailed off in a straight line from the green sphere in the direction of the agent’s predicted orientation. This inference representation gave us visual feedback for the model’s performance based on how closely the visual tracked with the agent’s movement in real time.

Testing was primarily conducted using the automated controller, where we were able to obtain quantitative insights into the model’s overall performance, but by using the manual controller, we were able to observe where the model performed best and learn where it needed additional data in order to improve.
All network development and model training took place on a remote workstation equipped with an AMD Ryzen Threadripper 3990X 64-Core Processor for the CPU, which runs between 2.2GHz and 2.9GHz, and an NVIDIA RTX A6000 with 48GB of RAM. Robot controller development and Webots data collection was conducted on a local machine using a 2.3GHz 8-Core Intel Core i9 CPU and two graphics processors, the Radeon Pro 560X 4GB and an integrated Intel UHD Graphics 630 with 1536 MB. The robot controller, running in the PyCharm IDE, leveraged the Webots external controller API, allowing for more efficient controller development and testing.

Inference testing was conducted on both workstations. Quantitative results, like average differences between predictions and ground truth, were gathered on the remote workstation while qualitative observations from real-time inference testing were made.
using the local, more visual workstation. Throughout development, a Python virtual environment was used. The environment consisted of the following installations:

- Python 3.8.10
- torch 1.10.1+cu113
- cuda 11.3

This specific versioning was largely dictated by the GPU-accelerated remote workstations which had CUDA 11.3 installed on their systems. Recent PyTorch versions (like torch 2.2.2 at the time of writing) required a more up-to-date CUDA version, and due to lack of permissions, updating CUDA was not an option, so using an older torch version was necessary. Additionally, EfficientNet required at minimum torchvision 0.11 which in-turn required torch 1.10+, so torch 1.10.1+cu113 was selected.
Chapter 6

RESULTS AND ANALYSIS

This chapter covers the results of the previously outlined experimentation along with analysis of the results produced. All tests were conducted with the datasets defined in Table 4.3.

6.1 Batch Size Results

The results from the batch size experiment were clear, demonstrating that error values appear to be an increasing logarithmic function of the increasing batch size. The relationship is depicted as near linear in Figure 6.1 and Figure 6.2 due to logarithmic scaling on the batch size axis, scaled with the logarithmic base of four.

It is important to note that despite the results of this batch size experiment, many of the experiments to follow were conducted using suboptimal batch sizes for training time purposes. Larger batch sizes significantly reduced the amount of training time from over 120 hours to just over 2, and for experiments comparing relative performance between models, the best possible performance was not required, only comparative performance of a given model relative to another. In other words, many additional experiments used a batch size of 64 to measure performance relative to other models instead of absolute performance, purely for training-time purposes, but the batch size for our best performing models was optimal.
Figure 6.1: Influence of Batch Size on Raw Performance
The experiment to observe the impact of camera count on standardized error was conducted using multiple datasets. Results in Figure 6.3 reflect performance of models trained using Dataset ‘E’ under the late fusion configuration with a batch size of 64 for training speed purposes and comparative analysis, not peak performance.

The general trend we observed was a decline in standardized error as the quantity of cameras increased. As the number of cameras increased, the neural network’s architecture did not change outside of modifying the input layers to handle additional images. Error reduction from an increased camera count was consistent across all datasets, and in more simple environments like the Square and Partial Apartment where fewer unique landmarks exist to aid in localization, the benefit was more substantial. For example, models trained using Dataset ‘A’ saw a 19.7% reduction in combined error when going from just 1 camera to 2. This direct return generally diminished as the number of cameras increased.
The direct benefit of using additional cameras was not always apparent, however. Though the Combined Error for all Dataset ‘E’ models was strictly decreasing as the number of cameras increased, individual standardized errors for both the $\Delta y$ and $\Delta \theta$ pose parameters experienced small spikes that interrupted the strict decrease. This noise could exist for multiple reasons. For any multi-camera configuration, additional views can be a hindrance if they take away the model’s ability to learn more useful features from the other images, like if a camera was obstructed by a wall or large obstacle. This scenario and principle is further explained in Section 6.5.1. An additional cause for noise, specifically for the 3-camera configuration, could be the inherent asymmetry of the data, as the third camera was only facing right of the robot by design. No other configuration exhibited asymmetry along the robot’s longitudinal axis.

Figure 6.3: Error Across Different Camera Counts and Metrics
6.3 LiDAR Layer Impact

In an attempt to better understand the influence of LiDAR data density on the success of a localization model, we set out to compare the performance of models trained using different LiDAR layer configurations. For this experiment, LiDAR data was isolated, meaning no image data was used in training and testing regardless of availability in order detach the LiDAR-layer benefit from other confounding variables. Using two datasets from the Partial Apartment environment, Dataset ‘B’ and Dataset ‘D’, there was a 24.7% improvement in combined accuracy when using 32 LiDAR layers as opposed to 4. In both models, the architecture was not altered beyond the input layers to adjust for the appropriate number of incoming LiDAR datapoints.

This benefit, visualized in Figure 6.4 does not translate to experimentation beyond 32 layers, as Figure 6.5 depicts an increase in standardized error, and thus a decrease in performance, when the layer quantity was increased from 32 to 128. The 32 and 128-layer experiment was conducted using Dataset ‘E’ and Dataset ‘G’, and though the
environment was more detailed and complex, the increase in LiDAR layers brought about a 9.2% increase in combined standardized error from 32 layers.

Though the quantity of LiDAR layers had not been fully optimized, we selected the 32-layer model structure to move forward with for fusion strategy experimentation.

### 6.4 Data Fusion Performance

A final experiment was conducted to determine the fusion strategy that performs best in this simulated scenario. As previously mentioned, the two fusion strategies we tested were the early and late fusion architectures.

First, we review the early fusion performance. After testing the early-fusion model on all 5,000 data collection points in the Dataset ‘E’ testing set (4 images and 16,384 LiDAR points per data collection point), we found an average raw error of about 0.243m for $\Delta x$, 0.255m for $\Delta y$, and 5.75° for $\Delta \theta$. After standardization, which we
calculate by dividing the raw error by the expected error for randomized values within
the Complete Apartment dimensions (12m x 12m), we achieve about 0.0607, 0.0637,
and 0.0639 for $x$, $y$, and $\theta$ respectively. Reflecting back on our definition of standard-
ized error, the early fusion model is over 93% more accurate than the randomized
expected error across all three pose dimensions, coming out to a combined error to-
total of 0.0631. One can observe that standardized performance was roughly equal
across all pose dimensions, and this uniform performance suggests that the model
was able to learn all dimensions relatively equally, which might be explained by the
eyear combination of both camera and LiDAR modalities.

Testing our late fusion model, however, revealed even greater performance than the
early fusion architecture. For the same Dataset ‘E’ testing set mentioned above, the
model exhibited an average raw error of about 0.184m for $\Delta x$, 0.198m for $\Delta y$, and
1.98° for $\Delta \theta$. Post-standardization values came out to be about 0.0460, 0.0494, and
0.0220 for $x$, $y$, and $\theta$ respectively, which means the combined error total for the
late-fusion model was about 0.0349, almost a 45% boost in error reduction compared
to the early-fusion model, and 96.5% more accurate than our baseline.

The comparison between the two fusion strategies is visualized in Figure 6.6. The
early fusion model (blue) consistently infers with higher standardized errors than the
late fusion model (orange). Additionally, the early fusion variation between the $x$, $y$,
and $\theta$ values is insignificant, while late fusion model displays similar $x$ and $y$ behavior
with a drastic decrease in $\theta$ error.

In Section 6.5, we analyze the inference error distributions for both models and
conclude that the slightly skewed distributions render the mean for both raw and
standardized errors less representative of each model’s performance than the median
errors. As a result, the median raw errors are reported in Table 6.1, and the stan-
dardized results are then visualized in Figure 6.7, reflecting an increase in accuracy
Table 6.1: Early vs. Late Fusion Absolute, Unstandardized Errors

<table>
<thead>
<tr>
<th>Fusion Strategy</th>
<th>$\Delta x$</th>
<th>$\Delta y$</th>
<th>$\Delta \theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early</td>
<td>0.24297m</td>
<td>0.25470m</td>
<td>5.7544°</td>
</tr>
<tr>
<td>Late</td>
<td>0.18395m</td>
<td>0.19765m</td>
<td>1.9808°</td>
</tr>
</tbody>
</table>

Median Results

<table>
<thead>
<tr>
<th>Fusion Strategy</th>
<th>$\Delta x$</th>
<th>$\Delta y$</th>
<th>$\Delta \theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early</td>
<td>0.14499m</td>
<td>0.16108m</td>
<td>1.1274°</td>
</tr>
<tr>
<td>Late</td>
<td>0.13790m</td>
<td>0.14995m</td>
<td>1.0780°</td>
</tr>
</tbody>
</table>

Figure 6.6: Mean Standardized Error for Early and Late Fusion Models

over the mean error by 59.7% and 31.2% for the early-fusion and late-fusion models respectively.

Because the inference distributions are skewed towards large errors, the models’ median errors are even more promising than the mean metrics. Despite the early fusion model displaying a greater boost in performance than the late fusion model when inspecting median error, late fusion still consistently outperforms early fusion but by a smaller margin of 5.6%, a significant reduction from the 45% discrepancy when comparing mean performance. Overall, based on the combined median standardized error metric, both the early and late fusion models were able to improve upon our ran-
Figure 6.7: Median Standardized Error for Early and Late Fusion Models

domized baseline performance by at least 97.4%, with the late fusion model achieving a raw median orientation error of just over 1° and both x and y raw median errors below 15cm in the 12m x 12m Complete Apartment environment.

It should be noted that these late fusion results display a 46% improvement from the LiDAR-only models. This performance boost appears significantly larger than the fusion performance improvement observed with OneShot, which demonstrated a LiDAR-to-fusion benefit of only 26% [38]. Additionally, the 1.98° average raw θ error is a significant improvement upon the landmark-based localization solution produced by Avgeris et al, demonstrating a 6° error with their vision-based system [3].

Additionally, both fusion strategy implementations exhibited essentially equivalent inference times. Average inference times were recorded at 21.4ms and 21.5ms for the early and late architectures respectively, thus the speed and efficiency of a particular strategy did not influence the overall view of either model’s performance over the other.
6.5 Limitations

There were several factors that limited both model performance and the scope of our analysis. The two most prominent included both the model’s tendency to learn behaviors that consistently produced inference outliers when it sensors were obstructed and the training process’s substantial memory demand when reading in appreciable data quantities. These limitations are explained in detail below.

6.5.1 Inference Outliers

Throughout testing different sensor configurations and data-fusion strategies, similar behaviors suppressing performance were observed. The e-puck developed common areas of struggle in its attempt to localize as a result of obstructed sensors. Take for example the Complete Apartment environment. We visualized these struggle areas in a heatmap by coloring cells according to the frequencies of where the robot experienced the least accurate inferences. As seen in Figure 6.8 and Figure 6.9, depicting outlier data for models trained under early and late fusion architectures respectively, the robot had a select few cells where it struggled considerably. These figures in particular only highlight areas where the model of interest made an inference whose error was greater than 3 standard deviations away from the mean error for any of the 3 values it inferred, and we will refer to these as inference outliers. Each cell in the heatmaps is representative of a 0.5m x 0.5m area, and the brightness is determined by the frequency of inference outliers that resulted based on ground truth data in that area. Figure 6.9 had fewer inference outliers overall due to its model’s superior performance.
To address the cells of greatest outlier frequency in both heatmaps, like those around (-7, 0), (-7, -11.5), and (-4, -4), we observe that the model’s areas of struggle usually find themselves adjacent to obstructions on at least two sides. The (-7, -11.5) and (-4, -4) cells specifically are surrounded by walls on two sides where the mobile agent finds itself with very few distinguishable landmarks in its remaining views. Additionally, areas like (-7, 0) are adjacent to a single wall, but additional views are obstructed by nearby object models that the agent was able to position itself tightly against, thus creating a similar limited-vision scenario. When cameras and the LiDAR sensor are largely obstructed, the model has less landmark information available to it to distinguish its position and orientation. When all the agent sees is a blank wall, it requires additional data to pinpoint which wall it’s facing and where it exists on the wall. This data deprivation led to many of the inference outliers we observe in our results.

Additionally, among the inference outliers, an interesting behavior was observed regarding the model’s orientation predictions. First, we inspect the distribution of the
Figure 6.9: Late Fusion Error Outlier Heatmap

$y$ coordinate inference outliers. Figure 6.10 and Figure 6.11 show a decaying and skewed-right distribution peaking at the outlier threshold which we have placed at three standard deviations away from the mean absolute error for each graph. This is expected from a normal distribution of errors, as the frequency decays as it strays further from the mean.

This same decaying behavior is not observed, however, in the $\theta$ inference outlier distributions. Illustrated in Figures 6.12 and 6.13, the distributions exhibit sudden peaks or clusters centered around 90 and 180 degrees of absolute error. This represents the model inferring an orientation vector that is either near perpendicular or parallel but opposite to the ground truth orientation vector, and such behavior indicates that the localization model has the ability to roughly learn the agent’s orientation despite inferring an incorrect cardinal direction relative to the agent’s pose. These round 90 and 180 degree predictions, being so distant from the means of 5.75 and 1.98 degrees for the early and late models respectively, are significant influences that skew the
6.5.2 Memory Demand

Collecting, storing, and reading 65,536 LiDAR data points for each of the 25,000 data collection points in addition to the 100,000 images in Dataset ‘G’ was no small task, totaling around 50GB in storage. Though the images could all be read individually, the LiDAR data was stored in the form of a large CSV file, mapping the data to its ground truth collection points. This made accessing the LiDAR data quite challenging. The training process often crashed due to lack of available memory, so without modifying the date collection process or storage format, scaling up was not a feasible option. This limited us from further testing data availability extremes, though our previous conclusions would deem such experimentation unnecessary.
Figure 6.11: Late Fusion $y$ Coordinate Inference Outlier Distribution

Figure 6.12: Early Fusion $\theta$ Inference Outlier Distribution
Figure 6.13: Late Fusion $\theta$ Inference Outlier Distribution
Chapter 7

FUTURE WORK

Though we found our experiments and results considerably perceptive and successful, we believe that certain aspects of our experimentation could be expanded upon. Here we discuss potential paths for future efforts to expand upon and further our work.

7.1 3D Mapping

One of the main advantages of using a neural network for localization is avoiding the need for a map of the agent’s environment. Many alternatives to neural networks require a 3D map or model of the surrounding terrain, whereas the neural network creates its own abstract map embedded in the neuron weights of its network. Despite this advantage, the ability to create or reference a complete model would likely improve performance considerably.

Juxtaposing our model’s performance with a map-dependent solution would provide significant insight into its viability. Furthermore, integrating our model’s architecture into existing mapping strategies is a logical next step. SLAM, as referenced in Section 3.3.2, is an effective solution to the localization problem in unknown environments, as it creates and updates a map through continued localization attempts.

7.2 Snapshot-Based to Sequential

A unique property of our model’s design is its snapshot-based architecture. The input data being sourced from strictly one point in time from multiple perspectives
demonstrates a lack of sequential data dependence. Each prediction is independent from any previous and future inference which has its advantages and disadvantages. Because the model’s accuracy is not influenced by previous predictions or locations, it performs well under arbitrary relocation scenarios. However, this snapshot behavior also leaves the agent susceptible to seemingly infrequent but erroneous predictions, as seen in our model’s inference outlier learned tendencies.

Future work may contain significant comparison between snapshot-based and sequential-based solutions, analyzing the trade-offs between data demand, efficiency, and localization accuracy.

7.3 Scaling

Though much of our experimentation surrounded scaling agent peripherals and available data, we left much to be desired in the areas of environmental scaling. In future work, this may look like assessing our model’s abilities in a larger-scale environment, like a simulated outdoor environment. Additionally, testing our data collection, training, and testing methodologies on a physical mobile robotic system would be a logical next step. Furthermore, in many practical applications, mobile agents are required to coordinate their movement with other agents in a highly dynamic, multi-agent system, so testing how our model reacts to dynamic environments would be worth investigating.
Chapter 8

CONCLUSION

In this paper, we have presented a snapshot-based global localization machine learning model for a mobile robot in a simulated static environment. Our model and methodology leverage multimodal data from the agent’s on-board cameras and LiDAR sensor to localize, predicting both position and orientation. We prioritize future physical implementations by using a computationally efficient base using the EfficientNet architecture, and our modified network design was able to achieve impressive results. Using late-fused Webots data collected in the Complete Apartment environment, our model was able to achieve positional accuracy within 0.2m for each of the $x$ and $y$ pose parameters and orientation accuracy within $2^\circ$ of the ground truth value, all without need for sequential data history or a preexisting map.

Throughout experimentation, we found our late-fusion architecture to be more successful at accurately localizing the e-puck than our early-fusion architecture. According to our combined error metric, the late-fusion architecture was almost 45% more accurate on average, but the performance was more comparable when median accuracy was considered, at a difference of only 5.6%. Both models were using the camera and LiDAR layer quantities that we independently optimized to be 4 cameras and 32 LiDAR layers.

Our research demonstrates the potential for accurate and robust mobile robot localization in simulated environments without the need for existing maps or temporal data. With the understanding that the ability to accurately determine a robot’s position and orientation in real-time is a fundamental aspect of autonomous naviga-
tion systems, we hope our research exemplifies and strengthens the potential of using multimodal data for such purposes.
BIBLIOGRAPHY


[34] Mohammad. How to find expected angle between two randomly generated vectors?, Jan 2016.


