1 2 9 0

UNIVERSIDADE D
COIMBRA

Eduardo Vilas Boas Varanda Guerra

JetNavigator
Computer Vision and Deep Learning Techniques to Improve Indoor Navigation

Internship Report in the context of the Master in Data Science and Engineering supervised by Prof. Carlos Lisboa Bento from the University of Coimbra and Eng. Rui Lopes from Critical Software SA. and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022

Eduardo Vilas Boas Varanda Guerra

JetNavigator
Computer Vision and Deep Learning Techniques to
Improve Indoor Navigation

1 2 9 0
UNIVERSIDADE D
COIMBRA

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

Department of Informatics Engineering

Eduardo Vilas Boas Varanda Guerra

JetNavigator

Computer Vision and Deep Learning Techniques to Improve Indoor Navigation

Internship Report in the context of the Master in Data Science and Engineering supervised by Prof. Carlos Lisboa Bento from the University of Coimbra and Eng. Rui Lopes from Critical Software SA. and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022

This page is intentionally left blank.

Abstract

Indoor localization systems are helpful in many practical applications where mobile and robotic operators require precise direction and location. These systems are mainly used to compensate for weak GPS signals in indoor environments.

However, existing indoor localization systems require additional equipment, such as Wi-Fi routers and distributed beacons, to accurately capture the necessary information to calculate a pose (consisting of location coordinates and orientation angles). These systems work perfectly in scenarios with pre-available suitable hardware, such as Wi-Fi routers, like in universities or schools. However, the same situation might not apply to large complex industrial installations, where extra hardware must be installed and maintained.

The main goal of this work is to present a new indoor localization method that significantly reduces the dependency on external hardware resources by leveraging state-of-the-art computer vision and deep learning techniques. To use this system, the user needs only a smartphone with a camera and our mobile application installed.

To achieve this objective, we divided our project into three steps:

- Dataset collection - We programmed a robotic agent to collect an image dataset.
- Image-based pose prediction model - We implemented and trained an image-based pose prediction model utilizing the previously built dataset.
- Mobile application - We created a prototype mobile application and a server. This mobile application is responsible for user interaction, sends the camera images to the server, and displays the path to the destination on the building map. The server utilizes the smartphone's pictures and the prediction model to calculate the user's pose. Then, using the user destination, it returns the calculated path to the mobile application to be displayed to the user.

The created mobile application did not perform well on actual test conditions. However, we have identified the issues that lead to poor results and challenges that future works need to address. This problem identification constitutes the main contribution of this work.

Keywords

Convolutional Neural Network, Deep Neural Network, Transfer Learning, ROS, Indoor Navigation

This page is intentionally left blank.

Acknowledgements

I want to thank my advisors, Professor Carlos Lisboa Bento and Engineer Rui Lopes, for all the support and availability. I learned a lot during this internship.

I also thank the Critical Software team for the integration opportunity and the positive team spirit. This project was a real challenge for me, and CSW people's expertise allowed me to achieve my goals.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

AI	Artificial Intelligence. 5
CNN	Convolutional Neural Network. 3, 5, 10, 41, 44
FPN	Feature Pyramid Network. 44
IPS	Indoor Positioning System. 1, 97
LSTM	Long short-term memory. 3, 5, 16, 17
ML	Machine Learning. 5, 6
RNN	Recurrent Neural Network. xv, 12, 16
ROS	Robot Operating System. 20, 22, 69, 97
RPN	Region Proposal Network. 14
SLAM	Simultaneous Location and Mapping. 27–29

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

This document reports the work developed during the curricular internship at Critical Software (CSW) within the scope of the Master's Degree in Data Science and Engineering at the Faculty of Sciences and Technologies of the University of Coimbra.

The main goal of this work is to present a new indoor localization method with reduced dependency on external hardware resources by leveraging state-of-the-art computer vision and deep learning techniques.

1.1 Context

Usually, a GPS is used to navigate from spot A to spot B. But GPS works with satellite signals, which can be disturbed or blocked when operating indoors. Therefore, Indoor Positioning Systems (IPS) are utilized to solve this issue, which can navigate users in indoor environments with weak or nonexistent GPS signals.

Several technologies, such as Wi-Fi devices, ultrasound, and magnetic fields, have been explored to implement these Indoor Positioning Systems. However, these technologies require using specialized hardware distributed inside the building to navigate. This solution is perfect in environments with pre-installed beacon devices, such as universities or schools with Wi-Fi routers. However, in locations with no pre-available hardware, such as large industrial/agricultural environments, those devices need to be installed and maintained.

With this project, Critical Software aimed to develop a cheaper alternative using state-of-the-art Computer Vision and Deep Learning techniques, with no need for additional specialized hardware.

Our main objective was to create a mobile application capable of navigating a user inside a building. Its best feature is that the user only needs a smartphone with a camera and the mobile application installed.

This project differentiates itself from other related projects because we implement and test it in a natural environment. While other projects aim to develop better pose (consisting of location coordinates and orientation angles) prediction models, our project goes further in deploying and testing it on a natural system, using a smartphone camera as a test device. While this project did not obtain good results, we have identified several challenges that future related projects need to address to be more successful.

1.2 Objectives and success criteria

During this internship at Critical Software, the main goal was to develop a new indoor localization method as an augmented mobile application using Computer Vision and Deep Learning techniques.

Due to the project's complexity and its broad range of employed technologies, it can be confusing what the project's real goals are. Therefore, here are explicitly described our objectives are/are not.

Our objectives include:

1. Development of an augmented reality mobile application to help a human user navigate (The augmented reality mobile application will be minimalist and will only serve as a proof of concept)

2. Implement and train an ML model that will use the images as input and output the user's pose.

    - Ideally, we will want to predict x, y coordinates and orientation (0 to 359).
    - However, due to the project's complexity, we might be forced to reduce its complexity.

3. Another less critical objective is the creation of an image-based location dataset. This dataset will be used in this project and also be helpful in future projects.

Our objectives do NOT include:

1. Robot self-navigation - our final users will be humans, not robots.

2. Building mapping algorithm development - we will use a pre-developed solution in our project.

The main objective of this internship is to create a mobile application. However, the prediction model of the pose (which consists of location coordinates and orientation angles) is the most exploratory component. Therefore, a solution for the pose prediction model will be researched and developed. As for the other parts, we will use pre-developed software.

1.3 Our solution

Our solution consists of a mobile application capable of solely navigating users indoors utilizing the smartphone camera. The mobile application would work as follows:

1. First, the user specifies in the mobile application what building they are.
2. The app calculates the user's pose and displays it in the mobile application, using the following steps:
   (a) The user records the environment with a camera.
   (b) The mobile application sends the images to a server.
   (c) The server uses the images as input for the prediction model of the pose (which consists of location coordinates and orientation angles).
   (d) The server sends the pose to the user. In addition, a plant of the building floor is also downloaded from the server and displayed on the screen.
   (e) The mobile application displays where the user is in the building floor plant and what direction they are facing.
3. The user specifies the destination.
4. The mobile application calculates the shortest path to the destination.
5. Both position, direction, and course are regularly updated until the user reaches its destination.

1.4 Document structure

We divide this report into chapters in the following way:

Chapter 1 - Introduction

The current chapter introduces and contextualizes the work carried out during the internship and describes its goals.

Chapter 2 - Prior Knowledge

This chapter exposes the differences between Machine Learning and Deep Learning and introduces the theoretical concepts of Convolutional Neural Networks and Long short-term memory neural networks. These architectures enable us to predict the location associated with an image.

Chapter 3 - Prototype

This chapter analyzes the devices CSW has made available for the data collection phase. We describe their technologies and their use for the different dataset collection steps.

Chapter 4 - State-of-the-art

This chapter analyzes the best methods available. Then, we will explore some solutions regarding environment mapping and image-based pose prediction algorithms.

Chapter 5 - Approach

This chapter describes the work methodology followed during the internship and its planning. First, we present the functional and non-functional requirements below, design prototypes, and perform risk analysis and the corresponding mitigation plans. Finally, we give the technical specifications.

Chapter 6 - Implementation

Here are all the tasks taken to reach the internship objectives and the reasoning behind the decisions made.

Chapter 7 - Tests

This chapter presents the adopted methodology and test results for the various prediction models, the mobile application prototype, and the final results. These final results are the quality of the collected dataset and the prediction model's performance in the test environment.

Chapter 8 - Conclusion

This chapter summarizes the work carried out during the internship. Then, we assess the experience and discuss the plans for future work.

Chapter 2

Prior knowledge

In this chapter, we expose important notions that allow the implementation of the most exploratory component of this project: a machine learning model that predicts user coordinates and orientation using images.

For this purpose, we will describe some Deep Learning methods and some most commonly used Machine Learning regularization techniques.

Finally, we will describe two types of neural networks - Convolutional Neural Networks (CNN) and Long short-term memory (LSTM) networks. CNNs are state-of-the-art image classification and object detection methods. LSTMs (Long Short-term Memory) are neural networks capable of establishing short and long-term dependencies.

2.1 Deep Learning

Before describing what defines Deep Learning, we need to refer to Machine Learning (ML) and Artificial Intelligence (AI).

AI is the art/science of enabling machines to mimic human intelligence rather than explicitly programming them to do as we tell them.

A subset of AI is Machine Learning (ML). ML uses techniques (such as deep learning) that enable machines to learn from experience and do tasks better. In ML, we usually follow these steps:

1. Provide data to an algorithm (before this step, we might use feature engineering techniques to improve the data quality).
2. Train, test, and tune our model.
3. Deploy our final model.
4. Use our model to automate tasks like stock market price prediction.

A subset of ML is Deep Learning. Deep Learning is characterized by *deep* neural networks with broad and numerous hidden layers. Posterior layers are responsible for performing feature engineering. In Deep Learning, the first layers capture low-level features (vertical and horizontal lines, for example). In contrast, the latter capture more complex features (depending on the problem, it might be faces, eyes, or car parts, for example). Deep learning is applied in many situations to process vast amounts of data, such as image processing or stock market price prediction.

There are some differences between Machine Learning and Deep Learning:

- **Data volume** - While other ML techniques, such as Random Forests or Support Vector Machines, work with smaller amounts of data, Deep Learning typically requires vast amounts of data.
- **Automatic feature extraction** - Many ML techniques require manual feature engineering. On the other hand, Deep Neural Networks automatically generate high-level features from raw data.
- **Number of hyper-parameters** - By definition, Deep Learning architectures have a much larger number of hyper-parameters compared to the traditional machine learning models.

2.2 Transfer Learning

Training deep neural networks often requires enormous amounts of training data, high-end computers, and time. Using Transfer Learning allows researchers to share and reuse previously trained models. This process enables the training of models for other purposes using substantially less data than the amount used for the original model.

The first set of layers usually contains simple features. In contrast, the final layers produce higher-level features closer to the domain in question. Using transfer learning, we can take a neural network trained for a specific problem and adapt its final layers for a different but related problem. For example, we can take a model trained for identifying flowers and apply it to identify fruits.

During training, the initial layers might have their weights frozen to maintain the original features or fine-tuned for the new problem.

2.3 Regularization

A significant problem in machine learning is creating an algorithm that performs well on the training data and new inputs - the algorithm must be able to generalize well. Many strategies used in machine learning reduce the test error at the expense of increased training error. These strategies are known collectively as regularization.

Regularization techniques are performed during training, allowing the neural network to generalize better.

We describe some of the more common techniques for regularization - Dropout and Parameter norm regularization.

2.3.1 Dropout

As described in [1], we can think of dropout as a method of making bagging practical for ensembles of very large neural networks. Bagging involves training and evaluating multiple models on each test example. Another way to look at dropout is to ensure each hidden unit performs well regardless of other hidden units in the model.

In the case of dropout, the parent neural network's parameters are shared across the models, with each model inheriting a different parameter subset.

Dropout trains the sub-networks formed by removing non-output units from an underlying base network, as illustrated in figure 2.1. The probability of a unit to be removed is a hyper-parameter fixed before training begins, generally defined as the dropout rate.

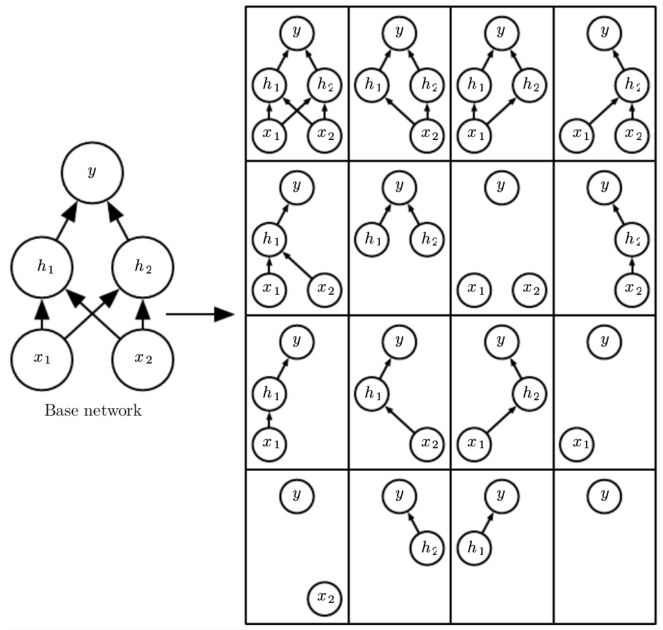


Figure 2.1: **Left**: Base network. **Right**: Ensemble of sub-networks; some networks have no path between inputs and outputs. This problem becomes insignificant in large networks. The image is depicted as in [1].

Typically, a deep neural network is complex enough that it would be impossible to generate all possible sub-networks. Instead, only a tiny fraction is generated and trained in each step. Due to parameter sharing, the remaining sub-networks arrive at good parameter settings.

2.3.2 Parameter norm regularization

Neural networks are trained using an algorithm that optimizes the value for an objective function (also known as loss function). This objective function varies according to the nature of the problem. For example, we might use the Mean Absolute Error (MAE) for a regression problem.

Therefore, as described in [1], another type of regularization approach limits the model's capacity. We do this by adding a parameter norm penalty $\Omega(\theta)$ to the model's objective function.

We denote the regularized objective function by $\tilde{J}$:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \tag{2.1}$$

where $J(\theta; X, y)$ is the objective function, $\tilde{J}(\theta; X, y)$ is the regularized objective function, and $\alpha \in [0, \infty)$ is a hyperparameter. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization. This approach can prevent the model from putting weights on fewer features, allowing it to generalize better.

There are two commonly used forms of parameter regularization - L^1 and L^2 (this last one is known as weight decay).

L^2 parameter norm regularization

L^2 parameter regularization, also known as weight decay, is the most common form of parameter norm regularization. A model regularized by L^2 parameter regularization has the following total objective function:

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^\intercal + J(w; X, y) \tag{2.2}$$

where $J(\theta; X, y)$ is the objective function, $\tilde{J}(\theta; X, y)$ is the regularized objective function, and $\frac{\alpha}{2} w^\intercal$ is the penalty. It has the corresponding parameter gradient

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla J_w(w; X, y) \tag{2.3}$$

L^1 parameter norm regularization

L^1 parameter regularization - adds a term $\sum_{i=0}^{n} ||w_i||$ (where w are the weights of the neural network model) to the objective function.

Thus, the regularized objective function is given by

$$\tilde{J}(w; X, y) = \alpha||w|| + J(w; X, y) \tag{2.4}$$

where $J(\theta; X, y)$ is the objective function, $\tilde{J}(\theta; X, y)$ is the regularized objective function, and $\alpha||w||$ is the penalty. It has the corresponding gradient

$$\nabla_w \tilde{J}(w; X, y) = \alpha sign(w) + \nabla J_w(w; X, y) \tag{2.5}$$

where sign(w) is the sign of w.

We can see in the equation above that the regularization factor does not scale linearly; instead, it is a constant factor. Therefore, this attribute means the optimal weight of w can be 0.

Compared to L^2 regularization, L^1 regularization results in a more sparse solution. In L^1 regularization, some parameters have an optimal value of zero.

2.4 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (also known as CNN) is a class of neural networks that process data in a grid-like topology, such as images.

CNNs are very good at picking up patterns in the input image, such as lines, gradients, circles, or more complex ones, such as eyes and faces. Additionally, a great advantage of using CNNs is that it is not necessary to do much data pre-processing.

A convolutional layer typically consists of three layers - convolutional layers, pooling layers, and fully connected layers.

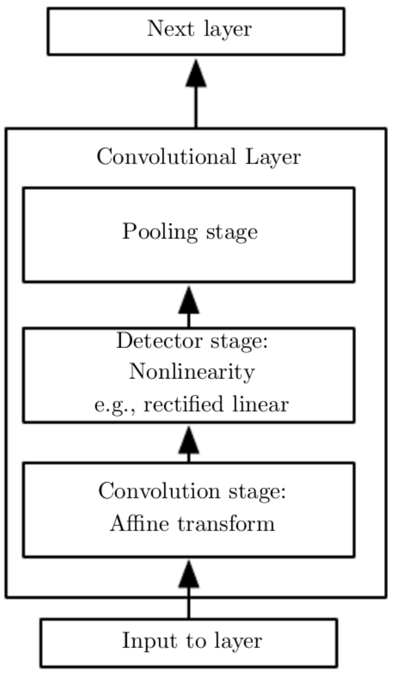


Figure 2.2: The components of a typical convolutional layer as depicted in [1].

2.4.1 Convolutional Layer

The convolution layer is the core building block of the CNN, enabling the CNN to capture the most relevant information on an image. While the first layers capture simple features, such as lines and curves, the latter can capture more complex ones, such as eyes or faces.

This layer performs a dot product between two matrices - the kernel and the input image.

As depicted in figure 2.3, the kernel is slid over the image and performs the dot product with each image section, looking for patterns. The operation returns a large positive value when we have a strong match. Otherwise, we get a smaller value.

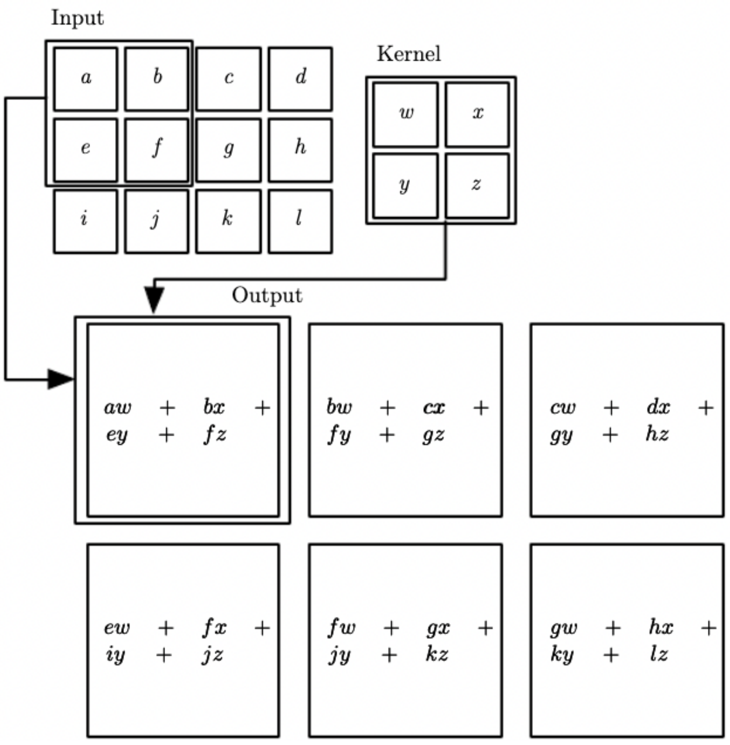


Figure 2.3: An example of 2-D convolution as depicted in [1].

In figure 2.4 we can see an example of a kernel operation.

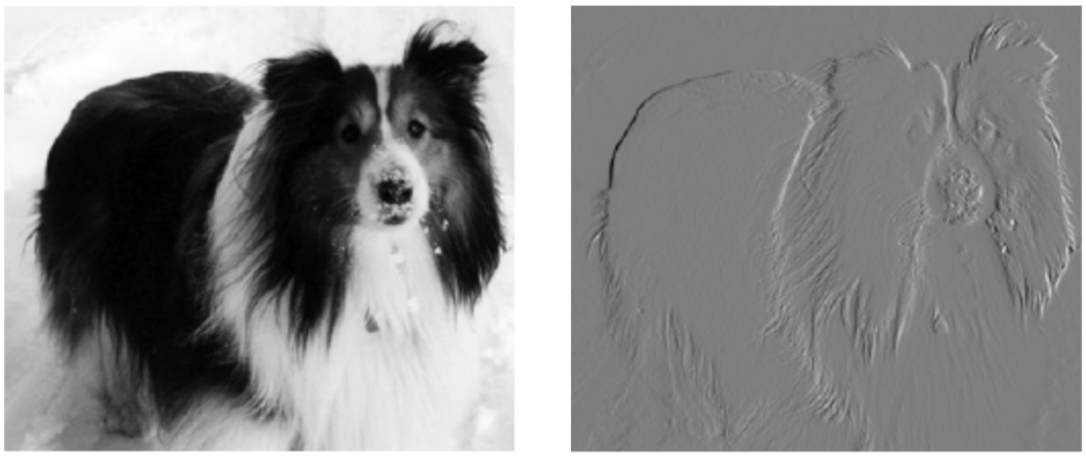


Figure 2.4: Example of image processing performed by a convolutional layer. According to [1], the "image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all the vertically oriented edges in the input image, which can be a useful operation for object detection." ([1], page 334).

Due to the kernel being applied to the same image and parameter sharing, there is no need to learn a specific pattern for each image section, substantially reducing the number of parameters. At the same time, it is exponentially more efficient than matrix multiplication.

2.4.2 Activation function

After the convolution layer, we apply a non-linear activation function to the output. This function's main objective is to add non-linearity to the convolutional layer output.

There are different activation functions, such as:

1. Sigmoid - $1/(1 + exp(-x))$ -
2. Hyperbolic Tangent - $(e^{2x} - 1)/(e^{2x} + 1)$ -
3. ReLU - max(x, 0)

2.4.3 Pooling layer

The pooling layer is used to sub-sample the input image to reduce the computational load, memory usage, and the number of parameters, which results in a more negligible risk of over-fitting. The pooling layer does this by preserving the strongest matches of the activation function output and disregarding the other values.

Another contribution of the pooling layer is the invariance to small changes in the input image. The pooling layer causes only the strongest matches to be preserved from layer to layer, disregarding the exact position. However, while this process can be helpful in some contexts, it can be detrimental in others.

Many algorithms are used for pooling, such as max pooling, average or the L^2 norm.

Due to the information destruction effect, the pooling layers must be used with caution.

2.4.4 Top layers

We place a neural network on top at the end of our convolutional layers sequence. This NN will use the outputted features to make a prediction. Depending on the task we have in hand, we can have a Fully Connected Neural Network followed by a soft-max function to output probabilities for object classification or an RNN if we are working on a video feed.

2.5 One-stage vs Two-stage Object Detectors

We have studied state-of-the-art object detection and classification. Object detectors need to find objects in the image and delimit their bounding boxes - the model accuracy is affected by the correct class prediction and the bounding box coordinates (this problem is called region proposal).

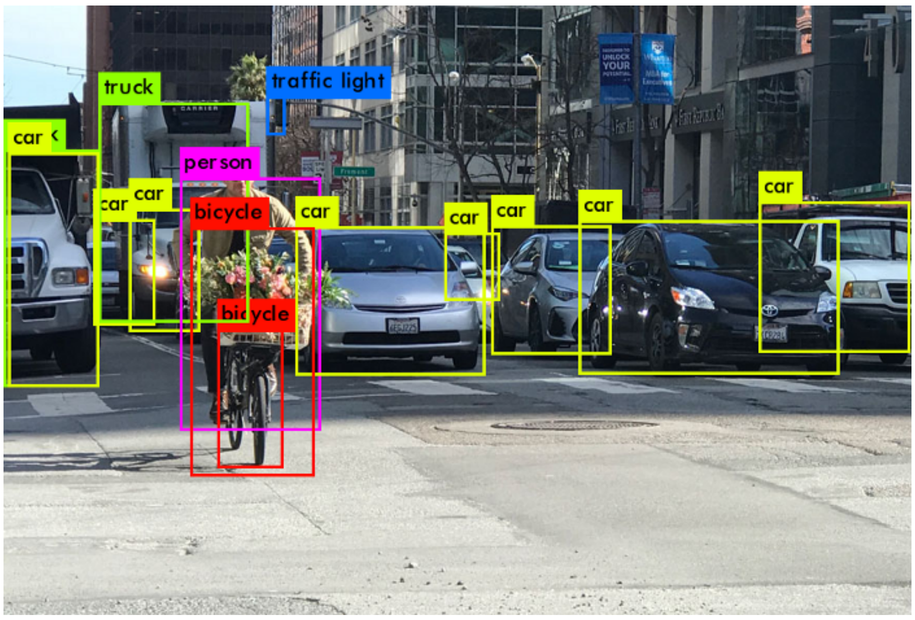


Figure 2.5: Example of an object detector used in a traffic picture, as depicted in [2].

There are two approaches for region proposal and classification - one-stage and two-stage object detectors. In this project, all proposed networks use anchors (also known as "priors" or "default boxes"). Anchors are a set of bounding boxes overlaid on the image at different spatial locations and with different sizes and aspect ratios. The size of these anchors can be parameterized and inferred from the training dataset. These anchors act as a default object size and help networks find objects. However, they are used differently in one-stage and two-stage detectors.

One-stage object detectors treat the region proposal and classification as a simple regression issue and make both predictions in a simple step. Then, the model is trained to make two predictions for each anchor:

1. a discrete class prediction for each anchor.

2. the positioning vector between the anchor and the ground-truth bounding box.

A commonly used one-stage object detector is YOLO [2]. YOLO (depicted in figure 2.6) detects objects and classifies them in a single step.

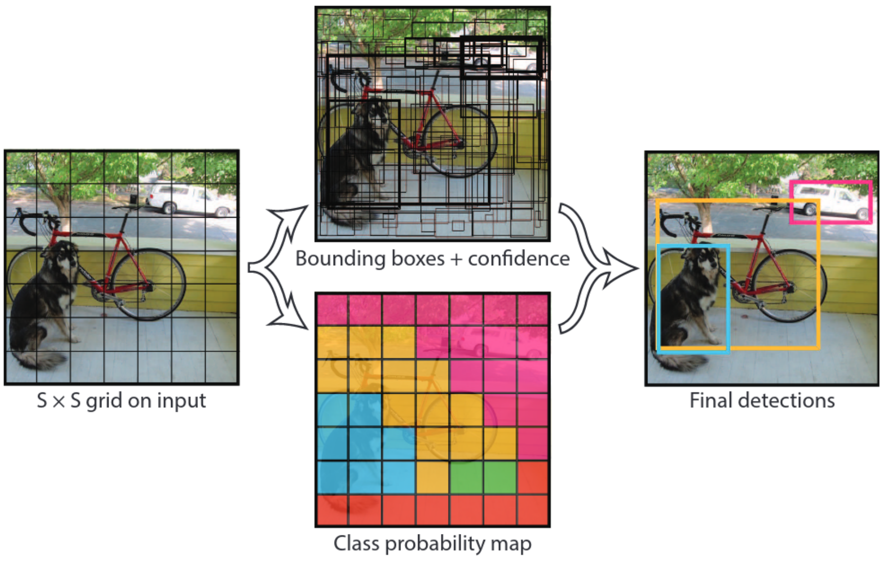


Figure 2.6: YOLO is a one-stage object detector and models detection as a regression problem, as depicted in [3].

Accordingly to the article [27] (page 2), the "advantage of having a regular grid of anchors is that predictions for these boxes can be written as tiled predictors on the image with shared parameters (i.e., convolutions) and are reminiscent of traditional sliding window methods."

Two-stage object detectors handle these issues separately - one NN is responsible for processing the image and proposing bounding boxes that contain objects. Then, another NN classifies the objects in those bounding boxes. Sparse R-CNN belongs to this class.

A Region Proposal Network (RPN) "takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score" ([3], page 2). The objectness score indicates the presence or not of an object in the proposed region.

An example of an RPN utilization is displayed in figure 2.7.

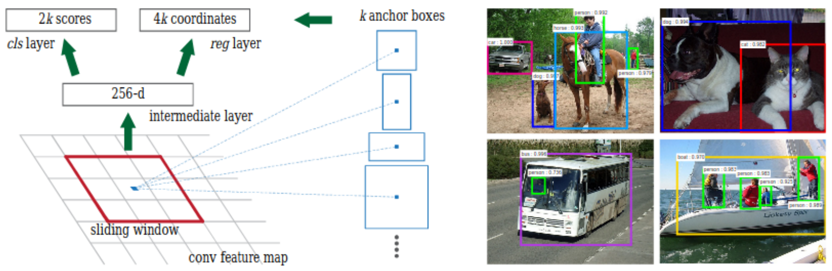


Figure 2.7: **Left:** Region Proposal Network (RPN) as depicted in [3]. **Right:** Example detections using RPN proposals on PASCAL VOC 2007 test as depicted in [3].

One-stage object detectors are typically faster, while two-stage object detectors generally are more accurate.

2.6 Long Short-Term Neural Networks

LSTMs (Long short-term memory) are a subclass of Recurrent Neural Networks capable of dealing with long-range dependencies. Furthermore, unlike standard feed-forward neural networks, LSTMs have feedback connections and can process single data points and sequences of data.

2.6.1 Recurrent Neural Networks

A Recurrent Neural Network(RNN) is a type of Neural Network where we use the output from a previous step as an input to the current one (as we can observe in figure 2.8). In other neural networks, all the inputs and outputs are independent. On the other hand, in areas such as Natural Language Processing or stock market price prediction, the past values are essential to predicting the next.

The fundamental feature of RNN is the Hidden state, which remembers some information about a sequence.

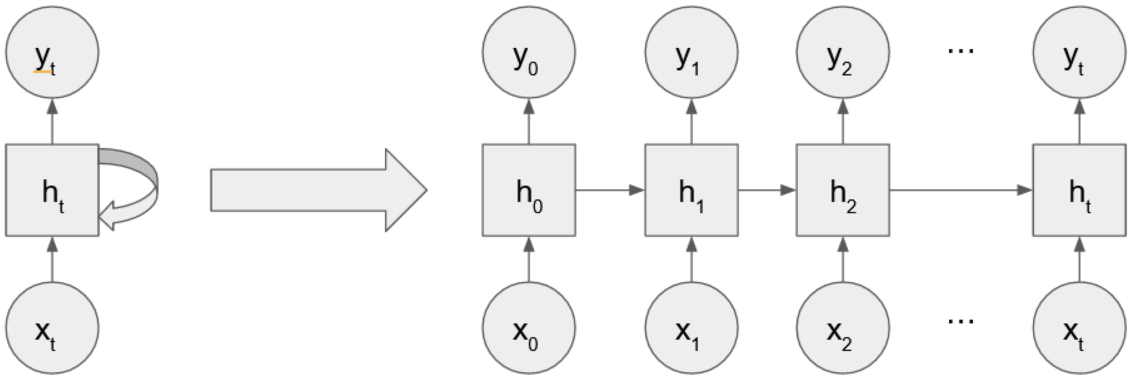


Figure 2.8: Basic architecture of an RNN.

RNNs retain information about what was previously calculated. Each layer uses the same parameters for each input, reducing the number of parameters. By stacking layers on top of each other, RNN can capture very complex patterns, such as predicting the next word of a sentence.

2.6.2 Long Short-Term Neural Networks - a subclass of RNN

Despite being very useful, traditional RNNs are not very good at capturing long-range dependencies. When we have a huge dataset and a deep neural network, we risk the vanishing or exploding gradient problem preventing our neural network from being trained.

It is because of this issue that LSTMs[28] were introduced. LSTMs are capable of remembering RNNs weights for an extended time. In addition to "hidden state", "cell state" is passed down to the next block.

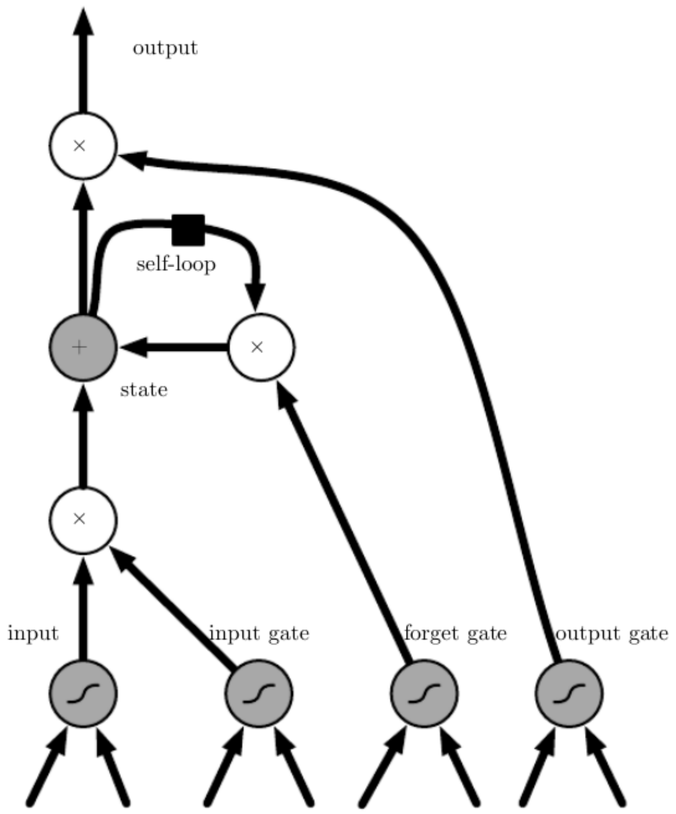


Figure 2.9: Block diagram of the LSTM recurrent network cell, as depicted in [1].

LSTMs do this by employing three main gates (gate scheme is depicted in figure 2.9):

1. Forget gate - Removes information that is no longer useful in the current cell state.
2. Input Gate - Useful information is added to the cell state.
3. Output Gate - Additional helpful information is added to the cell state.

This gate mechanism determines which information must be forgotten, ignored, or added to the memory state.

Chapter 3

Prototype

To create our image dataset, we need a robotic agent capable of moving around our testing environment, collecting our images, and associating each image with a pose (which consists of location coordinates and orientation angles). Furthermore, both our localization and direction values must have a reference point for values to be consistent.

In this chapter, we describe the work performed on the robotic agent throughout the first and second semesters.

3.1 First semester

While several different devices are available in the market, we have decided to utilize pre-available devices to collect our image dataset to save time and money - the robotic agent (JetRacer AI Kit [4]) and the laser scanner device (RPLidar A2 [5]). A past project used this equipment to map the environment using a laser scanner using LIDAR technology.

LIDAR is a method for determining ranges by emitting laser and measuring the time for the reflected light to return to the receiver. It has been used for decades for a variety of applications [29].

We used LIDAR to create a map of the environment and used that map to associate images with coordinates and orientation (relative to a reference point). For this purpose, Critical Software has made available the following devices:

- JetRacer AI Kit [4] - This is an AI Racing Robot kit based on Jetson Nano Developer Kit, and it supports deep learning, auto line following, autonomous driving, and so on. We can see an image of this device in figure 3.1.

- RPLIDAR A2 Laser Range Scanner [5] - Performs a 360-degree omnidirectional laser range scanning for its surrounding environment and then generates an outline map for the environment. We can see an image of this device in figure 3.2.

3.1.1 Robotic Operating System (ROS)

The Robot Operating System [30] (ROS) is an open-source set of software libraries and tools for robot applications and it is widely used in robotics companies, universities, and research institutes. ROS has packages for many problems, such as navigation and obstacle avoidance.

For more details, please check ROS homepage at `http://wiki.ros.org/`.

3.1.2 JetRacer AI Kit

JetRacer AI Kit [4] (we can see an image of this device in figure 3.1) is an AI Racing Robot powered by a Jetson Nano that was designed to be a cheap prototype for self-driving and deep learning experiments. NVIDIA Jetson Nano is a small computer supporting many AI frameworks, such as TensorFlow and Keras.

JetRacer AI Kit comes installed with a compatible camera.

Jetson Nano's operating system is Ubuntu 18.04 LTS, and it can be connected to peripherals, which allows scripts to be developed and executed directly on it. We can also manage Jupyter Notebooks scripts via a web browser.

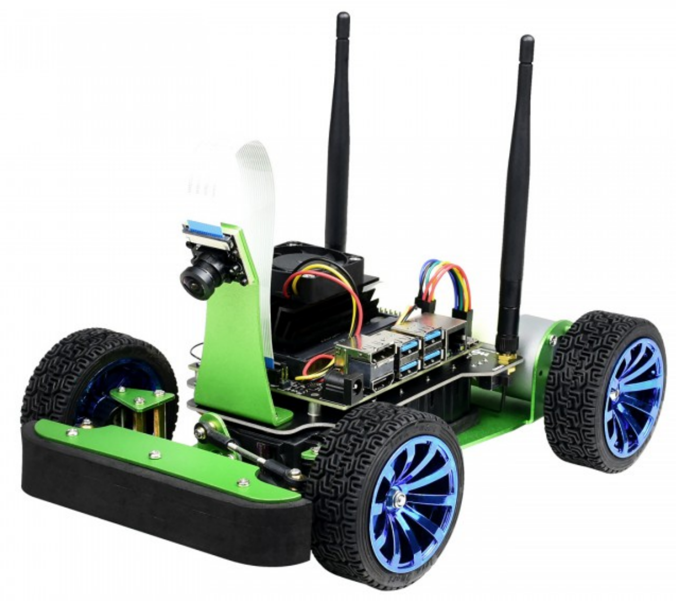


Figure 3.1: JetRacer AI Kit as depicted in [4].

NOTE: Due to the robotic agent's limitations and time constraints, we will limit the environment to a single Critical Software office space floor.

3.1.3 RPLIDAR A2

RPLIDAR A2 [5] (we can see an image of this device in figure 3.2) is a laser range scanner. The core of RPLIDAR A2 runs clockwise to perform a 360-degree omnidirectional laser range scanning for its surrounding environment and generates an outline map. It has a maximum range of 16 meters and a sample of up to 8000 points.

Figure 3.2: RPLIDAR A2 as depicted in [5].

3.1.4 Work done in the first semester

We utilized ROS to integrate the RPLIDAR A2 and our JetRacer robot. To test our prototype, we used the package hector_slam ([31]), a ROS package that implements Hector SLAM [8]. Hector SLAM only requires laser scan data (unlike Gmapping, which requires odometry data), and it does not require any parameter setup/tuning (unlike Cartographer).

To physically integrate the RPLIDAR into JetRacer, we needed a way to connect them and keep RPLIDAR stable while scanning. We created a simple cardboard structure for the prototype, but it was fragile and started deteriorating rapidly (we can see this prototype in figure 3.3). So the second prototype was made of 3D printed pieces designed to integrate our devices (we can see this second in figure 3.4).

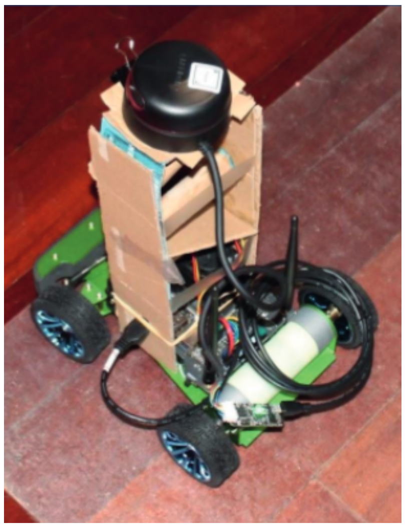


Figure 3.3: Cardboard prototype.

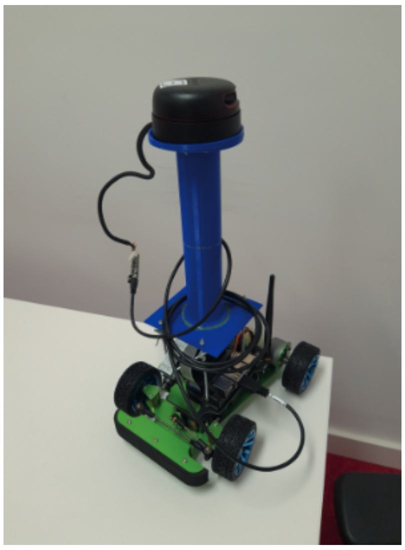


Figure 3.4: 3D printed the second prototype.

In figure 3.5 we can observe a map of Critical Software Coimbra Office B, first floor.

Hector SLAM generated this map in our robotic agent (second prototype) after completing a full circle inside the office, with the lower-left room being the start and endpoint. The map generated is not very accurate. However, the purpose of this test was not to test the algorithm's accuracy but to check if the prototype was robust enough for future experiments.

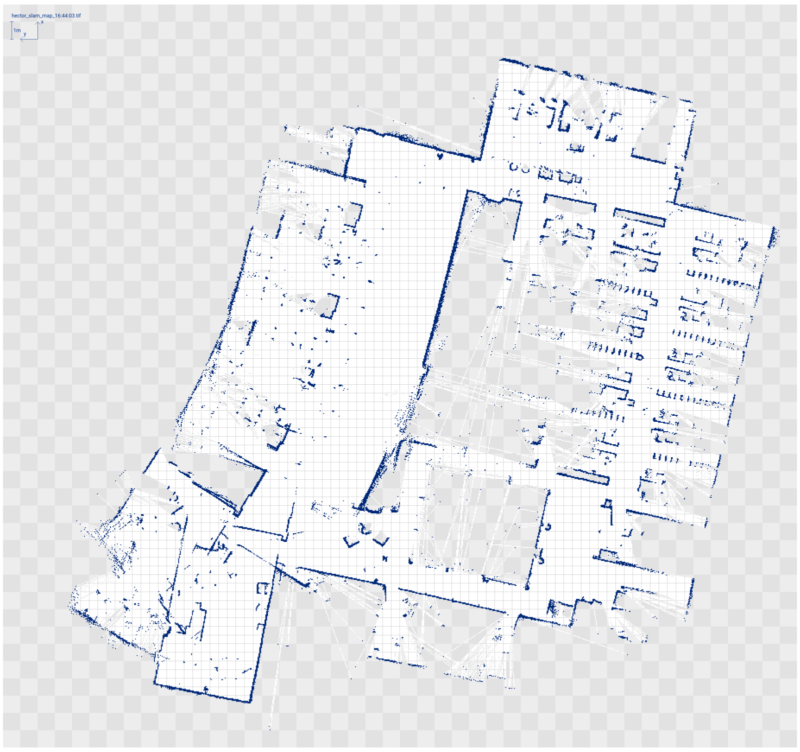


Figure 3.5: Critical Software Coimbra Office B, first floor. Image generated by Hector SLAM in our robotic agent.

3.2 Second semester

We integrated the robotic agent and the laser scanner device in the first semester using Robot Operating System (ROS). We tested the device's integration by mapping Critical Software's building. We expect these devices to be able to obtain all necessary data. However, after analyzing that our laser scanner could not provide us with the robotic agent's orientation angle, we have decided to add a device - Environment Sensors Module [6], designed for Jetson Nano (the processing unit of the robotic agent). It can be directly connected and used out-of-the-box, with no need for wire connections or software setup.

3.2.1 Environment Sensors Module

After integrating the robotic agent with the laser scanner, we observed that obtaining an orientation angle from the data was impossible. Therefore, an alternative was necessary.

After some search, we found a suitable device called Environment Sensors Module([6]). This module has an accelerometer, a gyroscope, and a magnetometer. It would be possible to extract an orientation angle using this sensor data. Another reason for the choice of this device was that it was purposely designed for the Jetson Nano (JetRacer AI Kit's computational unit), which saves time in hardware and software integration.

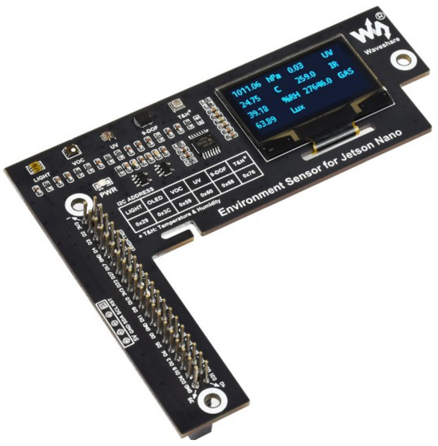


Figure 3.6: Environment sensor module designed for Jetson Nano as depicted in [6].

3.2.2 Work done in the second semester

To connect our sensor module to our prototype, we joined the male and female pins of Jetson Nano and the Environment sensor module. After that, we can extract the sensor data by simply utilizing Python scripts available on the WaveShare wiki page.

Chapter 4

State-of-the-art

In this chapter, we will present the state-of-the-art approaches for:

- **Mapping** - To associate images with coordinates and build our dataset, we must be able to map the environment. Here we discuss different algorithms for mapping which use LIDAR.
- **Robotic auto-exploration** - We research some algorithms that would allow our robotic agent to move autonomously, simplifying the setup necessary to implement our system in a new environment. Here, we discuss some alternatives. However, due to time constraints, we had to drop this feature to take care of more critical matters.
- **Image-based pose prediction** - In this section, we will expose different algorithms developed for image-based pose (localization + direction/orientation) prediction and compare them to find the best solution.
- **ContextualNet** - After analyzing the different image-based pose prediction algorithms available, we discuss the best solution, ContextualNet.
- **State-of-the-art object detectors** - In this section, we explore and compare different models which improve ContextualNet. Unfortunately, due to time constraints, we abandoned this research to focus on more pressing matters.

4.1 Robotic Auto-exploration

We wanted to implement auto-exploration in our robotic agent. This feature would allow our robotic agent to autonomously explore the environment and simplify the setup of our system in a new environment. This auto-exploration algorithm must also be ROS-compatible. Unfortunately, we had to drop this feature due to other more concerning matters more critical for this project's success.

Exploration consists of goal allocation (1) and path planning and navigating (2). Frontier-based allocation is the most referenced method for goal allocation. First, a set of frontiers is extracted from the current map; the allocator selects a frontier as the next goal based on different strategies (e.g., random, nearest, etc.).

Due to the devices utilized, we must use ROS-compatible solutions which can utilize a map to find unknown areas and explore them.

Following these requirements, we found **frontier_exploration** (`http://wiki.ros.org/frontier_exploration`) and **move_base** (`http://wiki.ros.org/movebase`). However, these packages have some limitations. For example, we need to specify a boundary area for **frontier_exploration** to specify which area to explore, and **move_base** has some difficulty passing through narrow spaces, such as doorways.

In the project, [7] researchers created two packages to improve the original ones -

- **frontier_allocation** (`http://wiki.ros.org/frontier_allocation`) - works efficiently with the entire actual actual map to enable out-of-the-box autonomous exploration.
- **adaptive_local_planner** (`http://wiki.ros.org/adaptive_local_planner`) - move_base plugin that acts as a local planner and allows the robot to navigate flexibly in a narrow space.

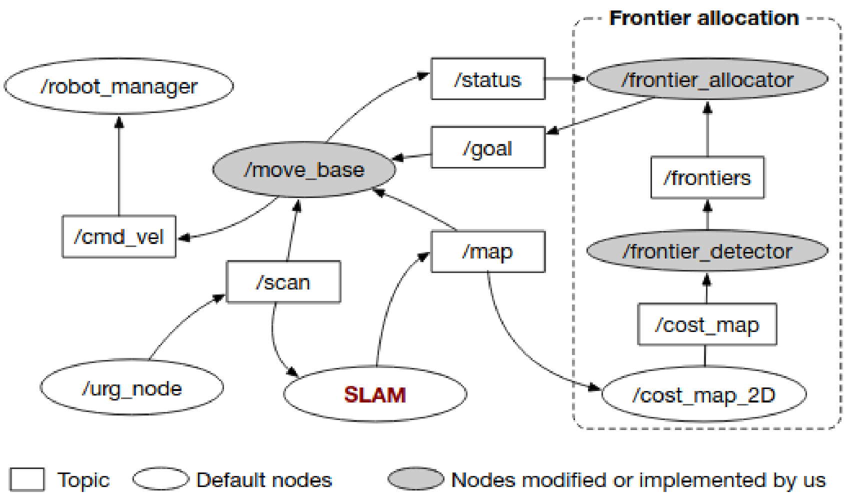


Figure 4.1: Proposed ROS framework as depicted in [7].

4.2 Mapping Algorithm

As stated in section 3, due to time constraints and pre-available hardware, we will use LIDAR technology to map the environment by mounting a laser scanner device on top of a robotic agent. We can use this map to build an image dataset with all image coordinates about the same reference point.

To get the most precise pose values, we researched the best mapping algorithms that used LIDAR. However, we will solely use the laser scanner data to map the environment without additional data due to time constraints. Additionally, as described in section 3, the algorithms must have a ROS-compatible implementation to integrate with our laser scanner. Finally, we must clarify that the robot will navigate exclusively using the laser scan data and never the images captured by the camera.

For this section, three different alternatives are available:

1. Hector SLAM [31] - The main idea of this algorithm is to use the high frequency of laser scan devices to match laser prints to build a map representation of the environment.

2. Gmapping [32] - Uses a Particle Filter SLAM approach for integrating laser scan data and odometry (data from motion sensors).

3. Cartographer [10] - Developed by Google, the main idea is to create and combine sub-maps (maps of the neighborhood) to create a more precise global map.

4.2.1 Frontier-based Exploration

Both Hector SLAM, Gmapping, and Cartographer are frontier-based approaches [33]. In Frontier-Based Exploration, the main idea is to gain the newest information about the world by moving to the boundary between open space and uncharted territory. Frontiers are regions on the border between open space and unexplored space. When a robot moves to a frontier, it can see into an unknown room and add new information to its map.

The environment is represented by an occupancy grid, where each cell is classified as either open (clear space), occupied, or unknown (need further exploration). As more and more area is explored, the occupancy grid is constantly updated until we have explored all unknown spaces.

4.2.2 Hector SLAM

The central concept behind Hector SLAM [8] functioning is scan matching, which aligns laser scans with or with an existing map. Modern laser scanners have low distance measurement noise and high scan rates.

The objective is to find the transformation which allows the last scan to align better with our map at the current coordinates.

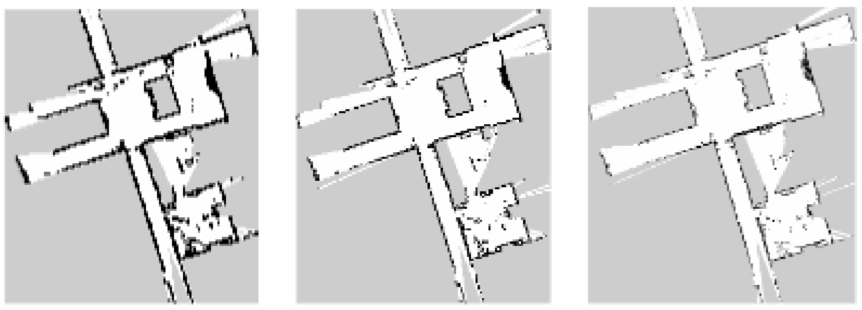


Figure 4.2: Multi-resolution representation of the map as depicted in [8]: **Left:** 20cm grid cell length **Center:** 10 cm grid cell length, **Right:** 5cm grid cell length.

As scans align with the existing map, the matching is implicitly performed with all initial scans.

Hector SLAM outperforms algorithms that use other data types (such as motion sensors) on uneven terrains because it relies solely on laser scan data [34].

4.2.3 Gmapping

Gmapping is a laser-based SLAM algorithm [9] using a Particle Filter SLAM approach. First, we compute the robot localization by integrating the most recent sensor observations with the odometry robot motion model (odometry is data from motion sensors). This step decreases the uncertainty about the robot's position in the prediction step of the particle filter. Therefore, the quality of the laser scan matching process reduces the number of particles, and the robot's pose is more accurate.

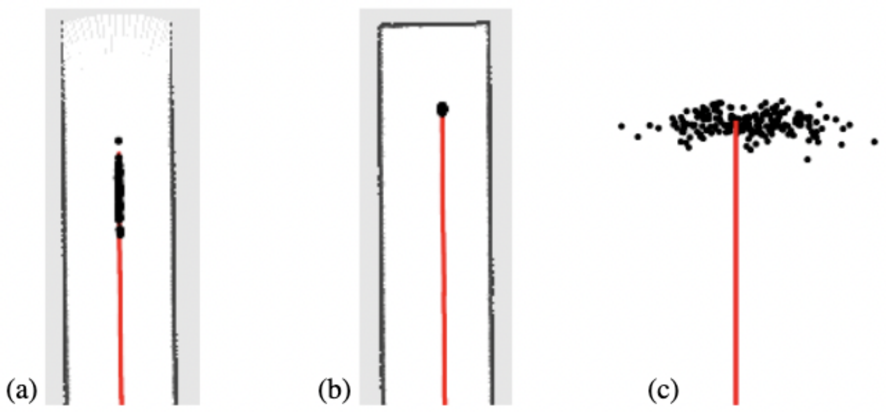


Figure 4.3: These images illustrate the particle distributions in different mapping scenarios. According to [9], "in an open corridor, the particles distribute along the corridor (a). In a dead-end corridor, the uncertainty is small in all dimensions (b). Such posteriors are obtained because we explicitly take into account the most recent observation when sampling the next generation of particles (...) (c)" ([9], page 4).

4.2.4 Cartographer

The Google Cartographer algorithm[35] comprises two subsystems - local SLAM and global SLAM. The local SLAM's job is to build sub-maps (each sub-map representing a region of the environment), while the global SLAM combines the different sub-maps to generate a complete map of the whole environment.

To reduce computation, a scan is only inserted into the current sub-map if its motion is above a certain distance, angle, or time threshold. Otherwise, the scan is dropped. A sub-map is complete when the local SLAM has received a given amount of range data.

The other subsystem is global SLAM. It runs in background threads, and its main job is to find loop closure constraints. Loop closure is when an algorithm connects ending sections of the global map, such as a corridor around a building. It does that by scan-matching scans against sub-maps. The final objective is to identify the most consistent global solution.

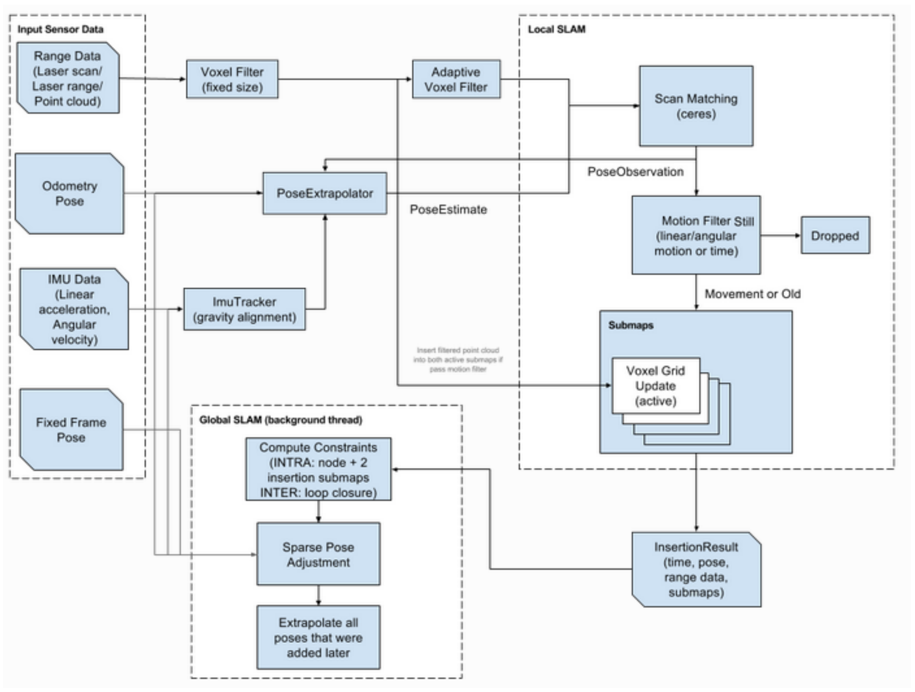

Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Insert filtered point cloud into both active submaps if pass motion filter
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)

Figure 4.4: Cartographer overview as depicted in [10].

Cartographer outperforms other algorithms that use different types of data (laser scan, odometry, linear and angular acceleration) [36], but it needs to be tuned to function correctly.

4.2.5 Mapping algorithm comparison

Since Hector SLAM relies solely on laser scan data, it tends to perform better when other types of data (such as angular or linear speed) do not work so well, such as on uneven terrain.

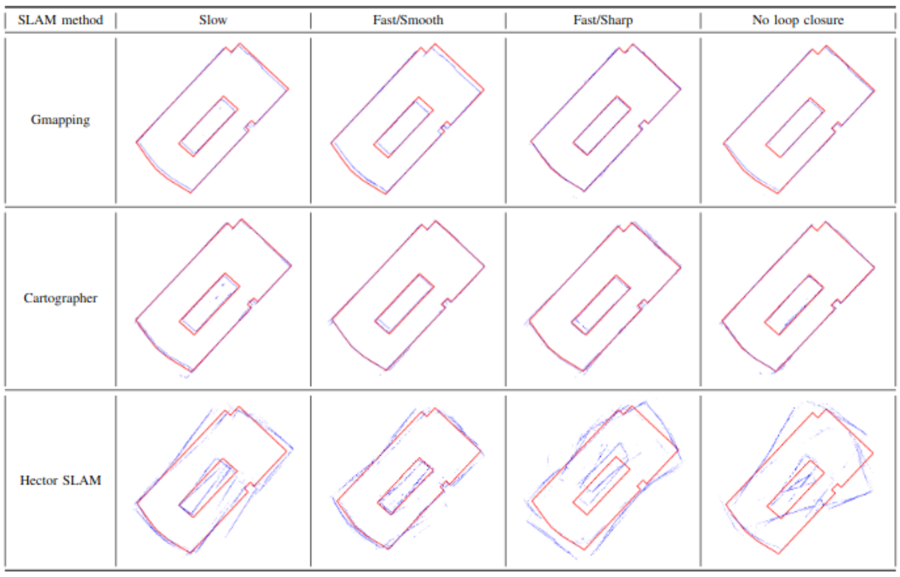
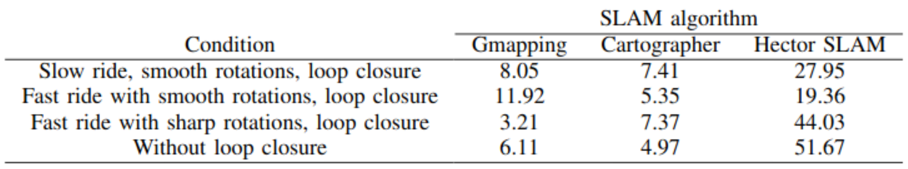


Figure 4.5: Mapping algorithms in different situations as depicted in [11].



Condition	SLAM algorithm		
	Gmapping	Cartographer	Hector SLAM
Slow ride, smooth rotations, loop closure	8.05	7.41	27.95
Fast ride with smooth rotations, loop closure	11.92	5.35	19.36
Fast ride with sharp rotations, loop closure	3.21	7.37	44.03
Without loop closure	6.11	4.97	51.67

Figure 4.6: Error calculated with ADNN (average distance to the nearest neighbor) metrics for SLAM methods relative to the ground truth, in cm, as depicted in [11].

Since Hector SLAM does not provide any solution for loop-closure, it is outperformed by the other approaches.

On the other hand, while Gmapping does not require tuning like Google Cartographer, Cartographer outperforms it in most situations. Its ROS package [32] does not provide a straightforward solution to work without odometry (data from other sensors), unlike Google Cartographer [10], which can rely solely on laser scan data.

4.3 Image-based Pose Prediction

Even a small RGB image of 224x224 pixels consists of 224x224x3=150528 different values, which is too big to be analyzed without feature extraction to reduce the total number of feature values.

Therefore, all image-based pose prediction models have two parts - a feature extraction section and a prediction top section. The feature extraction section will process the image and output a feature vector. The prediction top section will utilize the feature vector to compute a prediction.

There are two main methods of image preprocessing:

- **Local feature detectors methods**, such as speeded up robust features [12] (SURF). These algorithms follow a previously implemented algorithm - they are not trained like CNNs. These methods detect and identify key points and features of the images according to the previously implemented algorithm. These sorts of algorithms are beneficial for comparing images or finding duplicates. Another advantage these methods have over Convolutional Neural Networks is that they work out of the box, requiring no pre-training.

- **Convolutional Neural Networks (CNN)** can also be used to preprocess images and reduce the total number of features. They work by identifying patterns but must be trained in a data set to extract meaningful features.

4.3.1 Local feature detectors

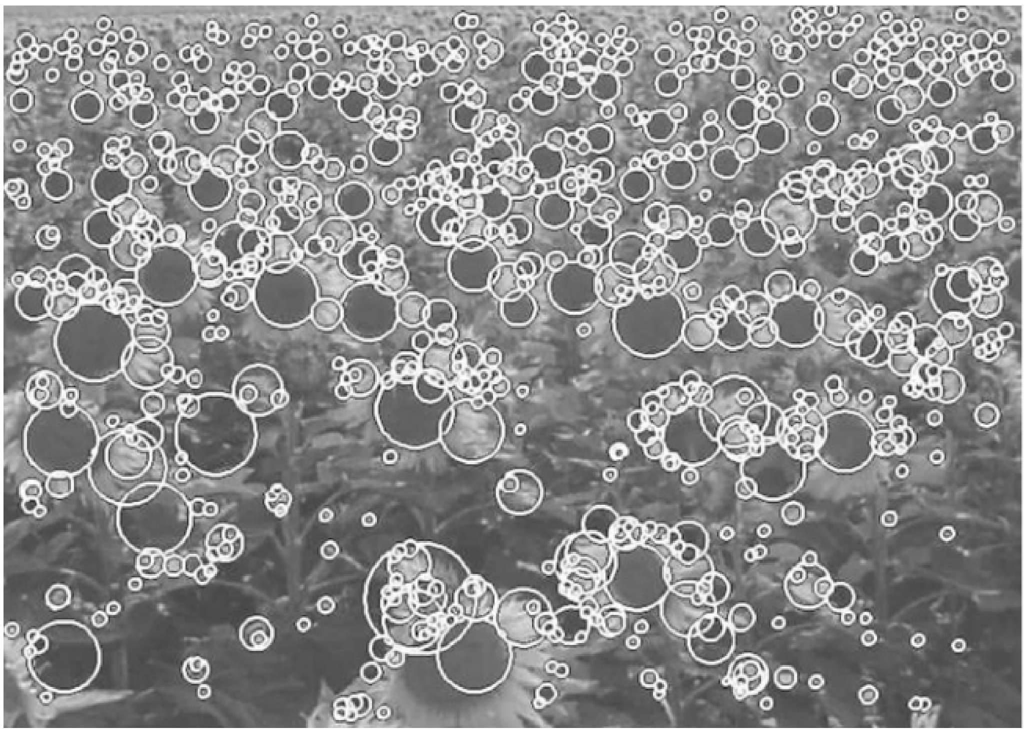


Figure 4.7: Example of detected interest points for a Sunflower field by SURF algorithm as depicted in [12].

«

Unlike Convolutional Neural Networks, which can be trained to identify different patterns in different contexts, Speeded up robust features (SURF) [12] is a pre-implemented algorithm that detects key points and features. This can be an advantage because they do not have to be trained to be able to see patterns.

In some works, researchers applied these algorithms for feature extraction. In [24], SURF is combined with LSTM layers to predict x,y coordinates and θ orientation of an image on a building floor. SURF is used for feature extraction before inputting the resulting key points and feature vectors into a bi-directional LSTM to predict its pose. In another work [23], SURF is utilized for feature extraction before inputting the resulting key points and feature vectors into a CNN. This approach dramatically reduces training time and increases accuracy, removing the need for a pre-trained neural network.

4.3.2 Convolution Neural Networks (CNN)

Several works have been done regarding image-based pose (which consists of location coordinates and orientation angles) prediction algorithms that utilize CNNs.

In some approaches, we directly compare the image extracted feature vectors to get a label. For example, in [37], the CNN ResNet50 is used to extract features from images from indoor scenes before a query is made using the k-nearest neighbors (kNN) to find the closest classification label (corridor, elevator, etc.). A similar approach is used in [38], where CNNs are also used to extract features from the image, but instead of kNN, the Euclidean distance between them is used to predict the correct label.

In order approaches, we insert a top prediction module on top of a pre-trained CNN backbone to predict coordinates and orientation. In PoseNet [25], they use a modified CNN, based on the GoogLeNet model [14], to indicate coordinates and orientation angle of camera images. Other projects improved on this model by adding the additional feature. On one hand, ContextualNet [13] implements LSTM layers on top of GoogleNet [14] and uses past frames to improve the current forecast. On the other hand, Directional-PoseNet [26] uses LSTMs for structured dimensionality reduction on the feature vector, leading to improved localization performance.

4.3.3 Comparison

All algorithms are trained and tested in the same dataset - 7-scenes [22].

The 7-scenes dataset is composed of tracked RGB-D camera frames in different scenarios. All scenes were recorded from a handheld Kinect RGB-D camera at 640×480 resolution, and the predicted poses from the Kinect are considered the "ground truth". In addition, there are predefined training and test sequences to train and test different algorithms and compare them.

Dataset	SurfCNN	SURF-LSTM	PoseNet	Directional PoseNet	ContextualNet
Chess	0.19/8.10	0.21/7.98	0.32/8.12	0.24/5.77	0.15/6.12
Fire	0.24/8.20	0.22/11.87	0.47/14.40	0.34/11.90	0.16/10.93
Heads	0.17/12	0.16/12.1	0.29/12.0	0.21/13.70	0.25/13.2
Office	0.35/7.05	0.32/7.05	0.48/7.68	0.30/8.08	0.25/7.45
Pumpkin	0.36/10.80	0.40/9.94	0.47/8.42	0.33/7.00	0.26/6.62
Red Kitchen	0.37/10.25	0.36/7.60	0.59/8.64	0.37/8.83	0.20/6.97
Stairs	0.28/10.14	0.35/10.10	0.47/13.80	0.40/13.70	0.17/10.83
Average	0.28/9.17	0.29/9.39	0.44/10.43	0.31/9.85	**0.20/8.87**

Table 4.1: The median error in position (m)/ orientation (degrees) for the 7-scenes dataset [22], of SURF-based methods, such as SurfCNN [23], SURF-LSTM [24], and purely CNN methods, such as PosetNet [25], Directional PoseNet [26] and ContexualNet [13]. The results were transcribed from the work reported in [24] and [13].

In table 4.1, we can observe that the best-fitted model for most scenarios is ContextualNet.

4.4 ContextualNet

Our proposed ContextualNet model consists of a GoogLeNet backbone [14], 2 LSTM layers, and two separate fully connected layers, connected as depicted in figure 4.9. In figure 4.8 we can visualize how ContextualNet compares precisely with the original PoseNet, which consists solely of the CNN backbone and fully connected layers.

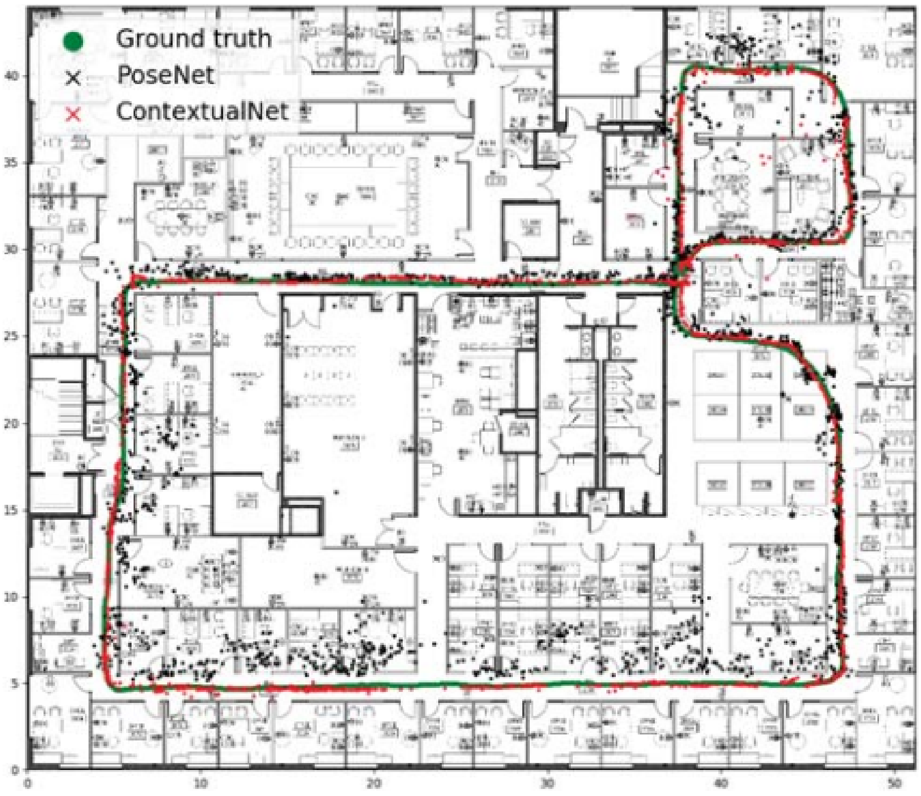


Figure 4.8: Map with predicted poses as depicted in [13]. Comparison of estimated position and orientation by our Contextual model and PoseNet with ground truth.

The original ContextualNet architecture is depicted in figure 4.9. The feature vector generated by the GoogLeNet backbone is input to the first LSTM layer. The first LSTM layer contains 512 hidden cells, while the second layer contains 50. The second LSTM layer is connected to 2 separate fully connected layers used for estimating the robot pose, P.

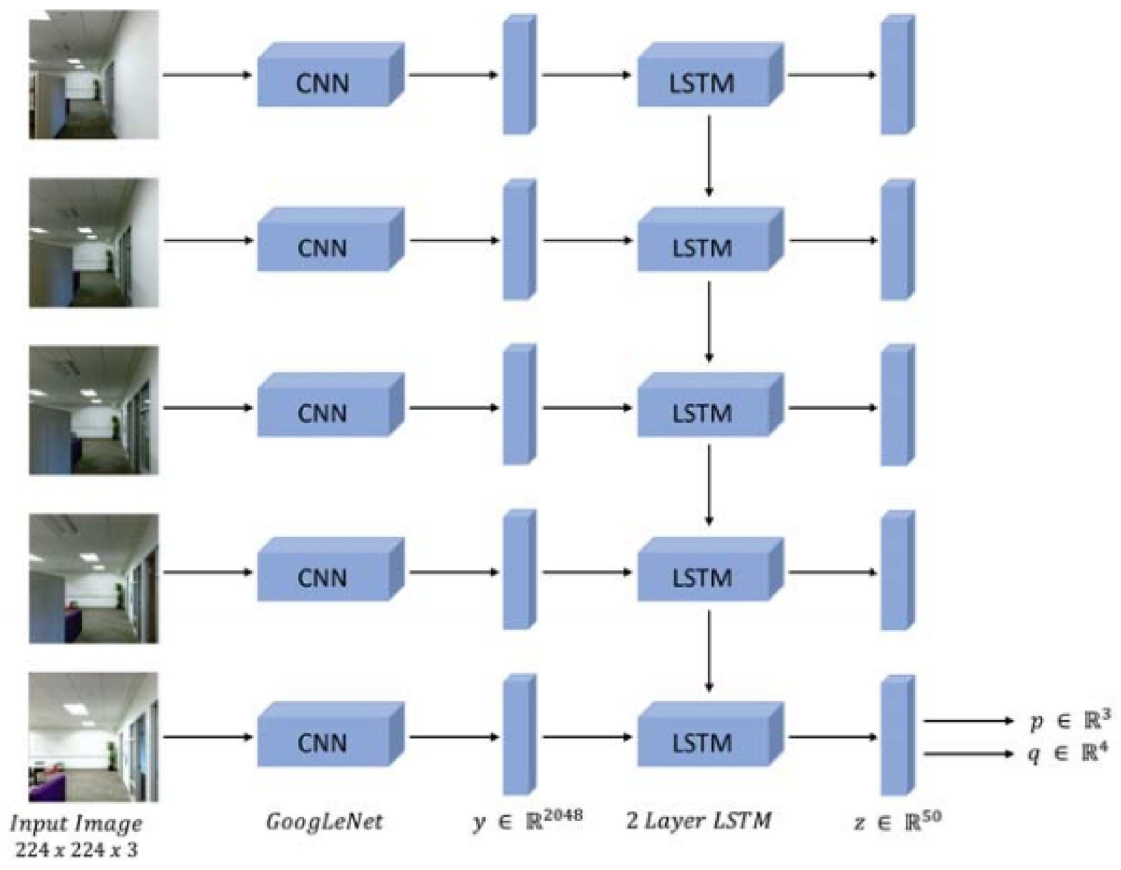


Figure 4.9: Model's architecture as depicted in [13].

GoogLeNet

GoogLeNet [14] is a 22 layers-deep network, winner of the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14), which contained several innovations which helped achieve high accuracy. The architecture of GoogleNet is shown in figures 4.11 (first layers, including input) and 4.10 (last layers, including output) - we split the image due to its size.

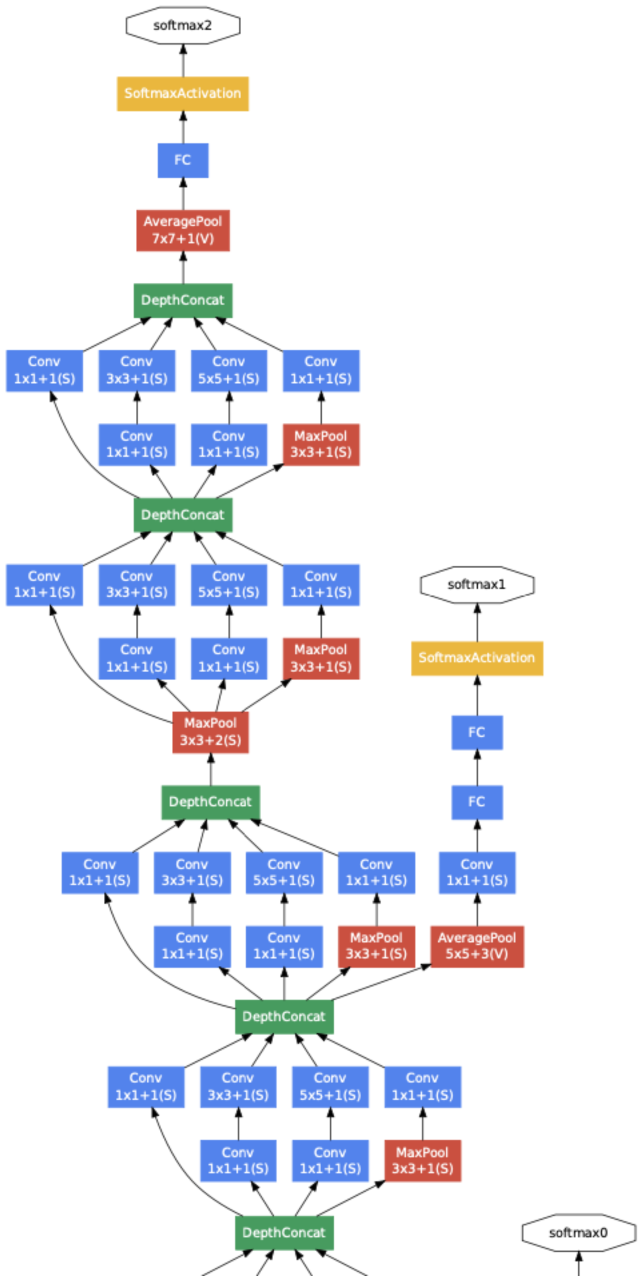


Figure 4.10: Last layers of GoogLeNet architecture, as depicted in [14]. The bottom "DepthConcat" block is the top "DepthConcat" block of the figure 4.11.

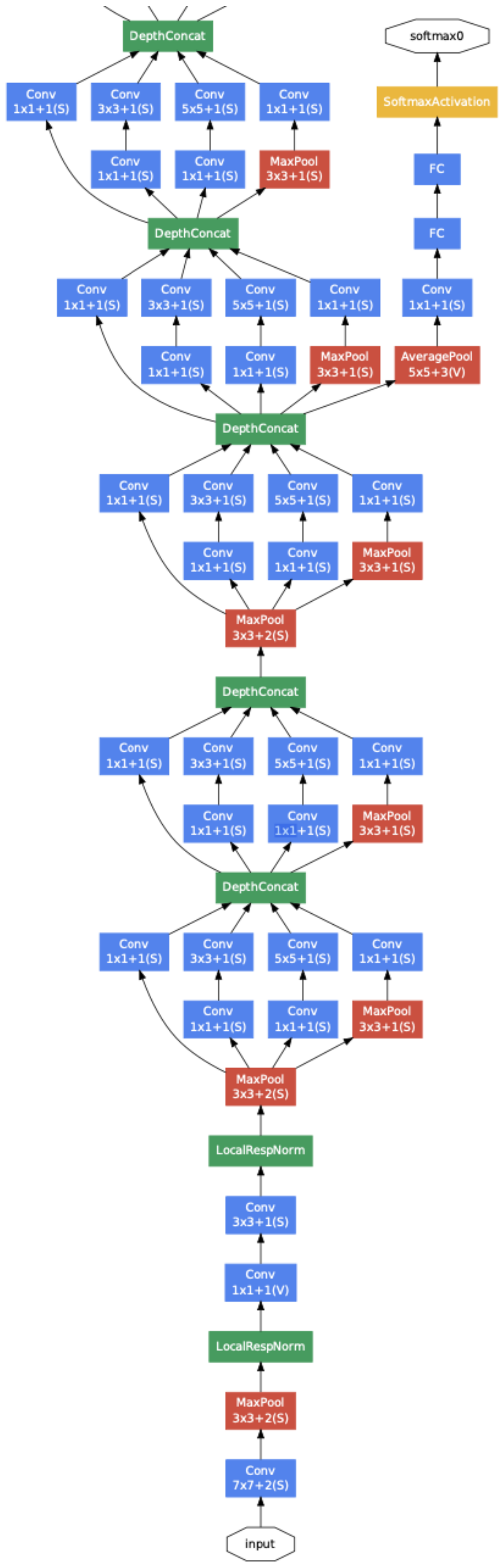
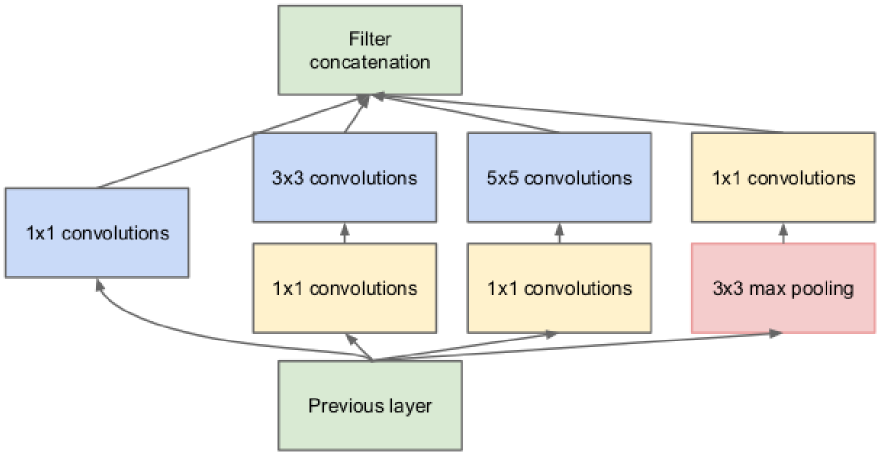


Figure 4.11: First layers of GoogLeNet architecture, as depicted in [14]. The top "DepthConcat" block is the bottom "DepthConcat" block of the figure 4.10.

GoogLeNet is a network formed by several combined "Inception Modules" which is a "network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid." ([14], page 4).



(b) Inception module with dimensionality reduction

Figure 4.12: Inception blocks as depicted in [14].

In the Inception module, "1x1 convolutions are used to compute reductions before the expensive 3x3 and 5x5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation making them dual-purpose" ([14], page 4).

The design follows the practical intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from the different scales simultaneously. In addition, the improved use of computational resources allows for increasing the width of each stage and the number of stages without getting into computational difficulties.

4.4.1 Model output - pose representation and loss Function

Given a sequence of images, ContextualNet outputs a pose P, which consists of location coordinates ($\mathbf{p} \in \mathbb{R}^3$) and orientation ($\mathbf{q} \in \mathbb{R}^4$) angles. The angles are represented as quaternions.

Quaternions

In the original article, orientation is represented by quaternions. The general form to express quaternions is:

$$q = s + xi + yj + zk \quad s, x, y, z \in \mathbb{R} \tag{4.1}$$

with i, j and k being imaginary numbers such as:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{4.2}$$

i, j, and k imaginary numbers could be used to represent three cartesian unit vectors **i**, **j**, and i, j, and k imaginary numbers could be used to represent three cartesian unit vectors **i**, **j**, and **k** with the same properties of imaginary numbers, such that $\mathbf{i}^2$=$\mathbf{j}^2$=$\mathbf{k}^2$=-1.

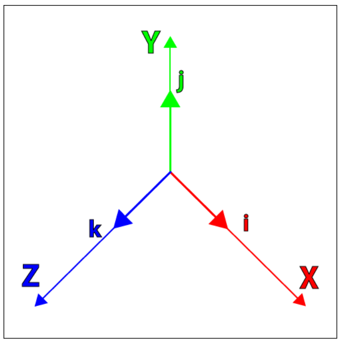


Figure 4.13: Representation of cartesian unit vectors **i**, **j**, and **k**, as depicted in `https://www.3dgep.com/understanding-quaternions/`.

A quaternion **q** is said to be a normalized quaternion when $||\mathbf{q}|| = 1$, which corresponds to $\sqrt{s^2 + x^2 + y^2 + z^2}$=1.

Loss function

In the original ContextualNet article, the following loss function is utilized:

$$loss_i = ||\hat{p}_i - p_i|| + \beta * ||\hat{q}_i - \frac{q_i}{||q_i||}|| \tag{4.3}$$

$(\hat{p}, \hat{q})$ are the predicted robot poses and (p, q) are the true value. β is a parameter that keeps the same scale's penalty values for location and rotation errors. In the original article, $\beta = 250$.

4.5 State-of-the-art Object Detectors

As discussed in the section 4.3, CNNs are state-of-the-art in image feature extraction. They can process millions of images at speeds and accuracy no other algorithms could achieve. Furthermore, CNNs compose the most complex part of the image-based pose prediction models by processing and extracting features from the images. Therefore, they have a significant impact on the model performance.

As observed in the section 4.3, ContextualNet is the most accurate model with a GoogLeNet backbone. GoogLeNet is an object detector utilized in ContextualNet for image feature extraction.

In the last few years, several object detectors algorithms have achieved better results than GoogLeNet. It would have been interesting to study the effect these improved CNNs would have on ContextualNet's performance. Unfortunately, due to several issues throughout this project, it was not possible to perform research regarding this topic. Therefore, this research will have to be conducted in future works.

To select the best CNN backbone, we looked at the current state-of-the-art for Object Detection and Classification. CNN backbones that have been trained in this domain will be suited for indoor localization, and we can take their features and use them to predict the image x,y coordinates, and orientation.

After reading about the state-of-the-art and learning more about the subject, the following state-of-the-art methods were selected as candidates to be used in future work: YOLO[39], Sparse R-CNN[17], EfficientDet, [19] and YOLOR[20].

Some criteria that will define the final choice for a CNN backbone alternative to GoogLeNet are:

- **Speed** - Considering that our mobile app needs to update our location every 1-2 seconds (for user-friendliness purposes) and that the image processing time needs to be relatively small, the CNN needs to be able to operate in real-time.

- **Precision** - We need a CNN that achieves the best possible results. We need our features to be accurate.

- **Computational cost** - Available computing resources are not vast, so this is something to be considered.

YOLO (You Only Look Once)

YOLO is a one-stage object detector and classification network which looks at the entire image at once and only once — hence the name You Only Look Once — which allows it to capture the context of detected objects.

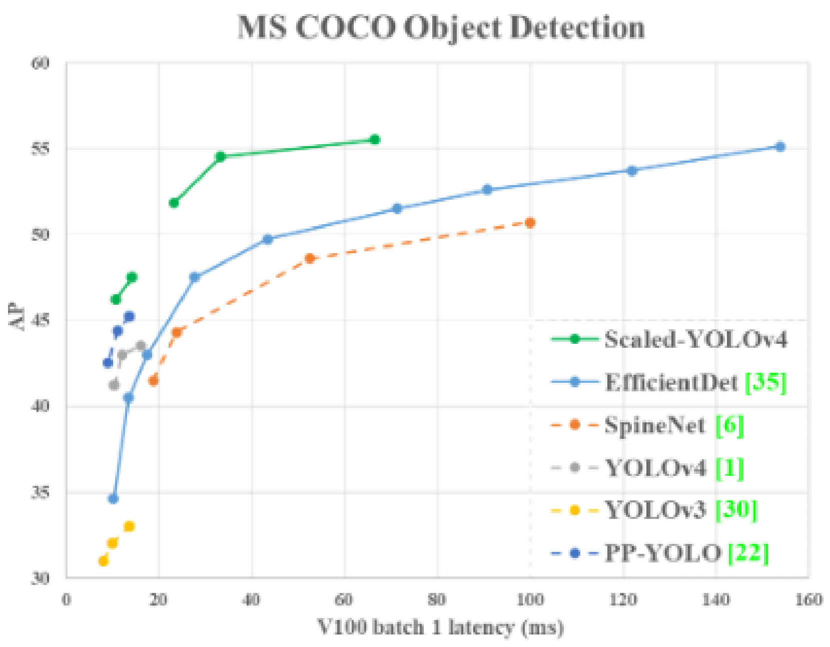


Figure 4.14: Scaled-Yolov4 achieves new state-of-the-art capable of achieving 55.5% AP on COCO dataset as depicted in [15].

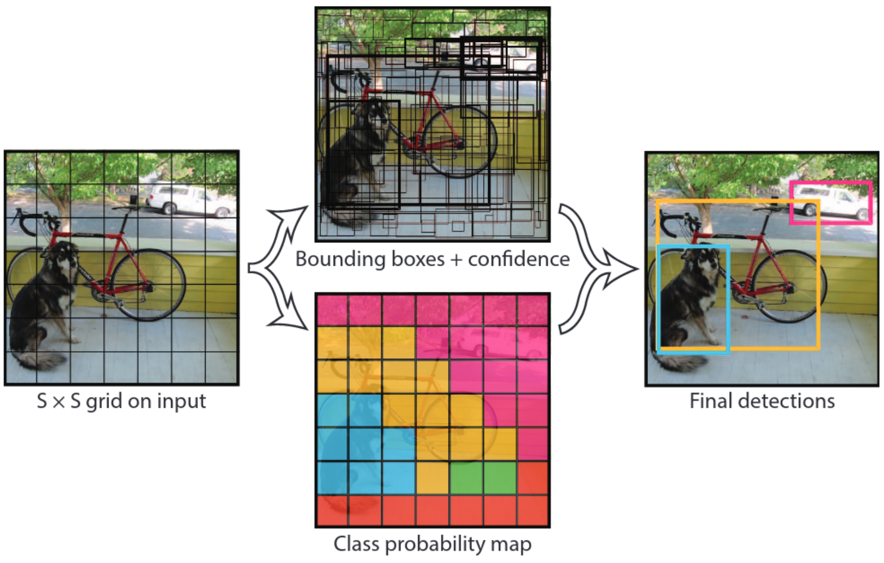


Figure 4.15: "Our system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B * 5 + C) tensor." ([2], page 2).

The cells are then agglomerated according to their classes and confidence scores to generate the whole output bounding boxes.

In Scaled-Yolo [15], YOLOv4 is upgraded by improving the Darknet CNN backbone using the Cross Stage Partial approach (CSP) [16]. According to the original article

[16], the "main purpose of designing CSPNet is to enable this architecture to achieve a richer gradient combination while reducing the amount of computation. This aim is achieved by partitioning feature map of the base layer into two parts and then merging them through a proposed cross-stage hierarchy.".

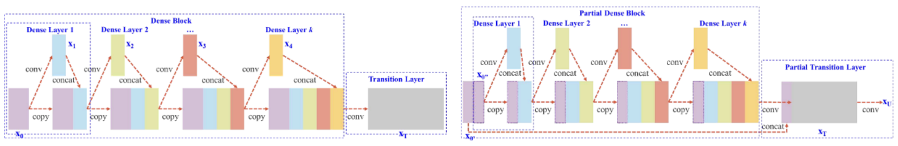


Figure 4.16: **Left:** DenseNet and **Right:** Cross Stage Partial DenseNet (CSP-DenseNet). "CSPNet separates feature map of the base layer into two part, one part will go through a dense block and a transition layer; the other one part is then combined with transmitted feature map to the next stage.". ([16], page 3).

This reduces CSP-Darknet53 computational cost by 50% compared to Darknet53. There are also scaled-up and scaled-down architectures (with different convolutional layer sizes) developed to give users other options according to their accuracy/speed requirements.

Sparse R-CNN

Sparse R-CNN speeds up exponentially by eliminating the need to calculate objectness scores for all possible bounding boxes and their locations by limiting the number of total possible bounding boxes to N.

In figure 4.17 we can see a comparison between different object detection pipelines and we can observe that Sparse R-CNN has much fewer parameters.

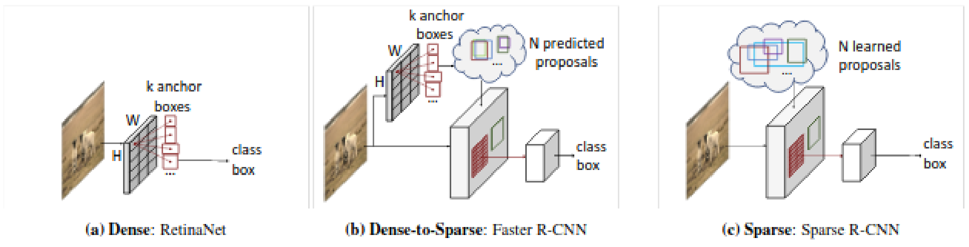


Figure 4.17: "Comparisons of different object detection pipelines. (a) In dense detectors, HWk object candidates enumerate on all image grids, e.g. RetinaNet. (b) In dense-to-sparse detectors, they select a small set of N candidates from dense HWk object candidates, and then extract image features within corresponding regions by pooling operation, e.g. Faster R-CNN [37]. (c) Our proposed Sparse R-CNN, directly provides a small set of N learned object proposals. Here N « HWk." ([17], page 1).

After these proposal boxes are obtained from the image, they are mapped into a proposed feature vector, which is expected to encode the rich instance characteristics.

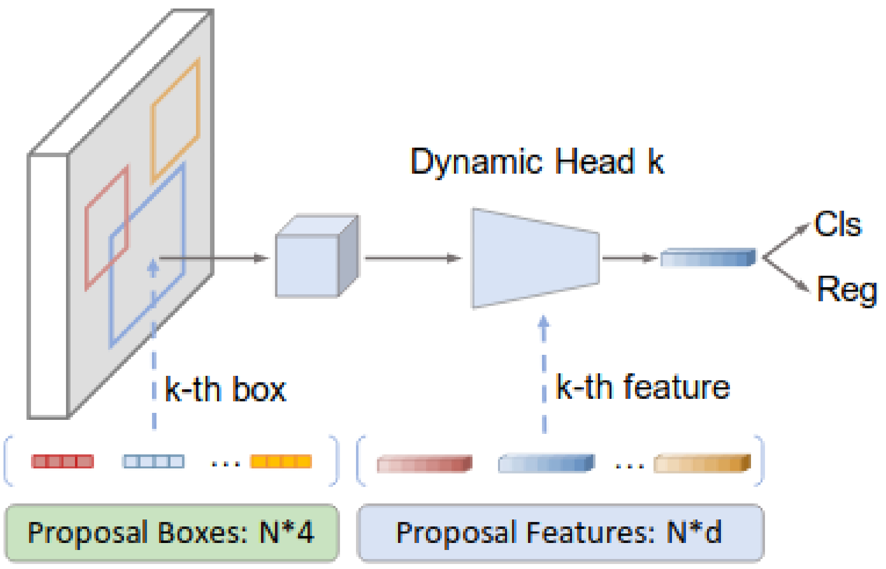


Figure 4.18: "An overview of Sparse R-CNN pipeline. The input includes an image, a set of proposal boxes and proposal features, where the latter two are learnable parameters. The backbone extracts feature map, each proposal box and proposal feature are fed into its exclusive dynamic head to generate object feature, and finally outputs classification and location" ([17], page 3).

Both proposal boxes and features are randomly initialized and optimized with other parameters in the whole network.

EfficientDet

To understand EfficientDet we need to first know about Feature Pyramid Network.

Features at different scales contribute differently to the final output prediction. Feature Pyramid Networks were created to address this issue. A Feature Pyramid Network (FPN) [18] takes a single-scale image as input and outputs proportionally sized feature maps at multiple levels in a fully convolutional fashion, independently of the backbone used by CNN. A scheme of FPN is depicted in figure 4.19.

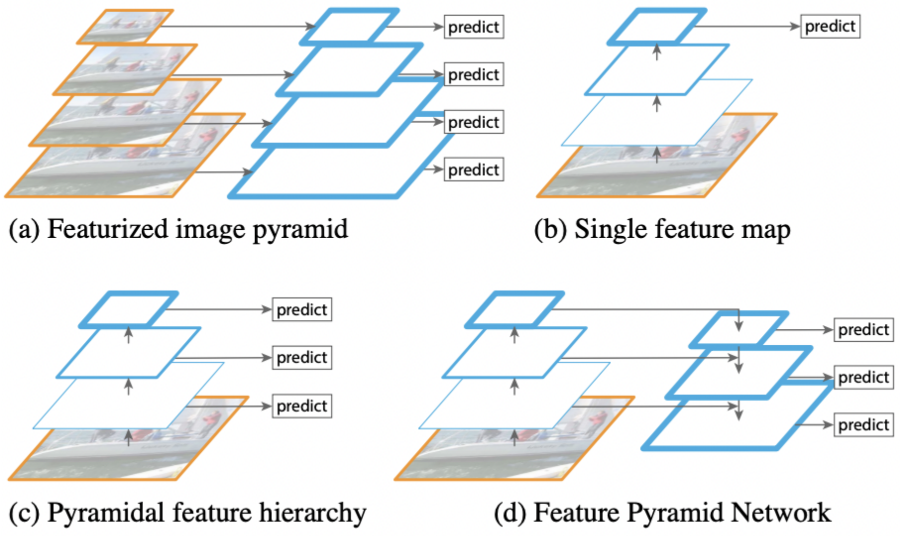

predict
(a) Featurized image pyramid
(b) Single feature map
(c) Pyramidal feature hierarchy
(d) Feature Pyramid Network

Figure 4.19: "(a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.". ([18], page 1).

EfficientDet employs a weighted bi-directional feature pyramid network (BiFPN) with adjustable weights to learn the importance of different input features while repeatedly applying top-down and bottom-up multi-scale feature fusion which allows it to understand which scales work better in each situation and have higher performance. A representation of EfficientDet's architecture is depicted in 4.20.

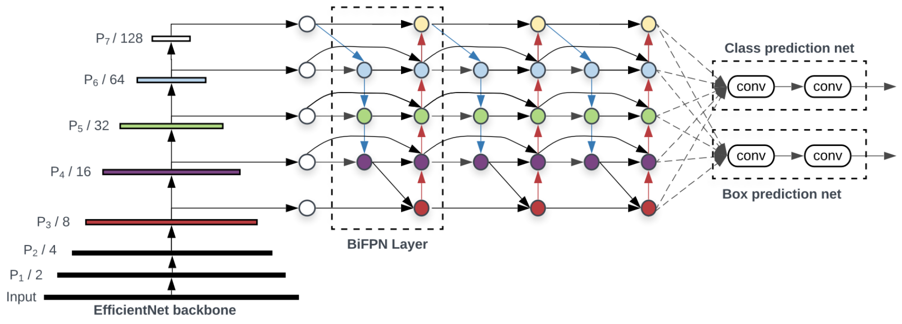


Figure 4.20: "EfficientDet architecture – It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints" ([19], page 5).

.

YOLOR (You Only Learn One Representation)

Humans can learn and understand the physical world based on vision, hearing, tactile (explicit knowledge), and experience (implicit knowledge). Therefore, humans can effectively process entirely new data using abundant experience from prior learning gained through regular learning and stored in the brain.

But what is explicit and implicit knowledge in neural neural networks? According to the original article [20], "the knowledge that directly correspond to observation as explicit knowledge. As for the knowledge that is implicit in the model and has nothing to do with observation, we call it as implicit knowledge." ([20]).

Not to be confused with YOLO (You Only Look Once), YOLOR [20] is proposed as "a unified network to encode implicit knowledge and explicit knowledge together, just like the human brain can learn knowledge from normal learning as well as subconsciousness learning. The unified network can generate a unified representation to simultaneously serve various tasks" ([20]).

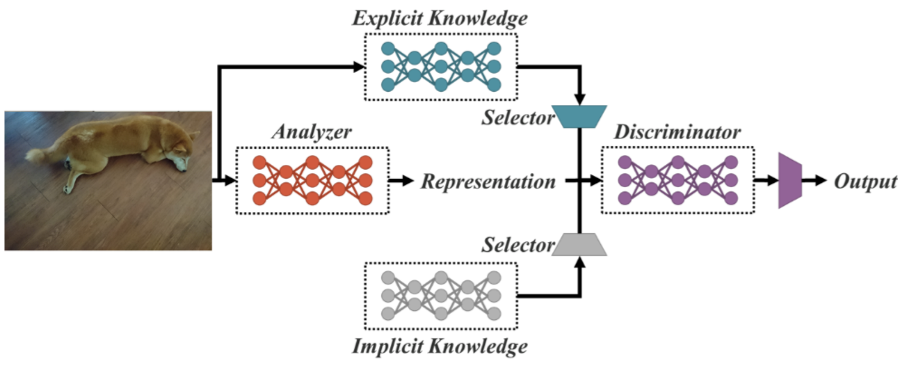


Figure 4.21: One representation with explicit knowledge and implicit knowledge for serving multiple tasks as depicted as in [20].

To train the proposed unified networks, explicit and implicit knowledge are used together to model the error term and guide the multi-purpose network training process.

Object detectors comparison

Best model configuration	Backbone	FPS	AP
YOLOv4-P7	CSP-P7	17/16	55.5%
EfficientDet-D7x	EfficientNet-B7	6.5	55.1%
Sparse R-CNN	ResNeXt-101-DCN	19	51.5%
YOLOR	YOLOR shared backbone	30	55.4%

Table 4.2: Comparison of the best configuration for each algorithm, according to [15] and [17]. Evaluated run time on NVIDIA Tesla V100 GPU.

Chapter 5

Approach

5.1 Requirements

Requirements are the features, properties, and restrictions expected for specific software. They are divided into functional and non-functional requirements, described in the following.

5.1.1 Functional requirements

As the name implies, functional requirements describe what our system should do. Therefore, in this section the functional requirements to be implemented during the internship are presented.

The main objective is to create an augmented reality mobile application that allows the user to know their location on a particular floor of a building and guide them to their destination. This mobile application will send the images collected by the smartphone camera to a server, which will input them into a machine learning model to obtain a pose prediction. But, first, we must clarify that we will limit the environment to a single floor of Critical Software office space due to the robotic agent's limitations and time constraints.

The mobile application must be able to guide the user to reach their destination **(FR-3)**. We do not plan to create a fully functional product; instead, we only want to create a minimalist prototype that serves as a proof of concept.

For our mobile application to predict the user's location, we must develop a model to use the images from the smartphone camera as input to predict the user's pose. **(FR-2)**.

Moreover, to train this model, we must collect a dataset from the Critical Software office space consisting of images associated with its pose **(FR-1)**. Additionally, we would like to program the robotic agent completely autonomously **(FR-4)**. However, we had to drop this feature due to time constraints and other more critical requirements. So instead, our robotic agent is remote-controlled.

The high-level requirements described above are represented in the table below. Each requirement is given a priority value of:

- High - mandatory requirement.
- Moderate - adds a lot of value to the final prototype, but it is not mandatory.
- Low - the requirement adds some weight to the final prototype, but it is not mandatory.

Table 5.1: High-level functional requirements.

ID	**NAME**	**PRIORITY**	**DESCRIPTION**	**Implemented**
FR-1	Dataset collection	High	Collection of images to build an image dataset, where each image is associated with a pose.	✓
FR-2	Image-based pose prediction model	High	The machine learning model must be able to predict the pose of images captured from the building's floor.	✓
FR-3	Mobile application	High	The mobile application must be able to guide the user to reach their destination.	✓
FR-4	Robotic auto-exploration	Low	Implementation of an algorithm to enable the robotic agent to explore the environment and collect our dataset autonomously.	✗

Dataset collection

Our main objective in this stage was to construct a dataset consisting of images **(FR-1.1)** associated with poses (location coordinates **(FR-1.2)** and orientation **(FR-1.3)**). Ideally, we would be able to associate images with this information. However, due to unforeseen predicaments, we were forced to abandon the orientation part and stick only to location coordinates. In alternative to our model being able to predict the orientation angle, which would allow the application to provide orientation in any smartphone, we manually calibrated the application using a compass.

Firstly, we implemented the mapping algorithm Google Cartographer (`https://google-cartographer-ros.readthedocs.io/en/latest/` - a better algorithm than the one used in the prototype testing in section 3) to associate images with location coordinates (**FR-1.4**).

Secondly, we wanted to utilize the Environment Sensor Module data to get an orientation relative to the magnetic North to associate images with orientation (**FR-1.5**).

Table 5.2: Functional requirements related to FR-1:Dataset collection.

ID	NAME	PRIORITY	DESCRIPTION	Implemented
FR-1.1	Dataset collection	High	Collection of an image dataset.	✓
FR-1.2	Associate location coordinates with images	Medium	Each image must be associated with location coordinates.	✓
FR-1.3	Associate orientation with images	Low	Each image must be associated with orientation.	✗
FR-1.4	Google Cartographer implementation	Medium	Implementation of the Google Cartographer algorithm.	✓
FR-1.5	Associate orientation with images	Low	Utilization of the Environment Sensor Module to extraction an orientation angle relative to the North Pole.	✗

Image-based pose prediction model

The image-based pose prediction model corresponds to the exploratory component of this internship. However, developing an indoor position and mobile navigation application is our primary objective and must be accomplished before proceeding with more experimental and less critical procedures.

Therefore, the most critical requirement related to **FR-2** is the creation of a baseline model (**FR-2.1**). To validate our baseline model, we will need to validate it in two datasets:

- **7-Scenes dataset (FR-2.3)** - The original paper of ContextualNet tested its performance in the 7-Scenes dataset [22]. An exemplary implementation of the ContextualNet is expected to have a similar result as the original.

- **Our datasets (FR-2.4)** - Our robot is going to collect images of building floors (CSW office spaces, which will act as our testing environments) and associate them with coordinates and orientation. We must test our model performance on these datasets before proceeding further(**FR-3.3**).

Another optional requirement is related to the improvement of the original ContextualNet architecture and consists of:

- substitution of original GoogleNet backbone by YOLOR (best candidate as described in section 4.5)

- LSTMs for structured feature correlation (as described in [26])

These methodologies were incrementally tried, and we used the best combination for our final model **(FR-2.5)**. However, this requirement was dropped due to more pressing matters.

Table 5.3: Functional requirements related to FR-2:Image-based location.

ID	NAME	PRIORITY	DESCRIPTION	Implemented
FR-2.1	Implementatior of a baseline model	High	Creation of a baseline model based on the ContextualNet architecture.	✓
FR-2.2	7-Scenes dataset model validation	High	Our model must be able to perform accordingly in the 7-Scenes dataset.	✓
FR-2.3	Our datasets model validation	High	Our model must be able to perform accordingly in the datasets collected by our robot.	✓
FR-2.4	Baseline model improvement	Low	Experimentation with different methodologies in order to improve the baseline model.	✗

Mobile application

The main objective of our internship is the development of a mobile app. We do not plan to create a fully functional product; instead, we only want to create a minimalist prototype that serves as a proof of concept. This prototype mobile application will have the following functionalities:

- **(FR-3.1)** - allow the user to specify the building they are in.
- **(FR-3.2)** - allow the user to choose the destination.
- **(FR-3.3)** - must integrate with the pose prediction model.
- **(FR-3.4)** - display the building floor plant.
- **(FR-3.5)** - calculate the shortest path from the user's current position to the destination and display it.
- **(FR-3.6)** - get user orientation and display it.

To use our location prediction model, we must develop a server that will be responsible for using our model and integrating it with our mobile application (**(FR-3.7)**). For example, given a model predicted user position and a destination, the server calculates the shortest path from the user's current position to the destination and returns it to the mobile application **(FR-3.8)**.

Table 5.4: Functional requirements related to FR-3:Mobile application.

ID	NAME	PRIORITY	DESCRIPTION	Implemented
FR-3.1	Building selection	Low	Our mobile app must allow users to specify the building they are in.	✓
FR-3.2	Destination selection	Medium	Our mobile app must to allow the user to select its dest2ination.	✓
FR-3.3	Prediction model integration	High	Our mobile app must to be able to use the location prediction model to predict the user location and orientation.	✓
FR-3.4	Building floor plant	Medium	Our mobile app must show the building floor plant with the user location.	✓
FR-3.5	Path display	High	The mobile application must display the path calculated by the server.	✓
FR-3.6	User orientation display	Medium	The mobile application must display the path calculated by the server.	✓
FR-3.7	Server implementation	High	Responsible to receive the user's camera captures, forward the model's predictions, and return the pose prediction to the mobile application.	✓
FR-3.8	Calculate shortest path	High	The server must calculate the shortest path to the destination based on the map of the building's floor, the location predicted by the model, and the destination predicted by the user. It then must return to the mobile application.	✓

Robotic auto-exploration

Ideally, we want a robotic agent to autonomously explore the environment and collect our dataset, which would incredibly simply be our system set up in a new environment. However, we could not implement this step due to time constraints and delays in more critical issues. This requires the implementation of a self-exploration algorithm **(FR-4.1)** to enable autonomous exploration and image collection.

Table 5.5: Functional requirements related to FR-4:Robotic auto-exploration.

ID	NAME	PRIORITY	DESCRIPTION	Implemented
FR-4.1	Robotic auto-exploration	Low	Implementation of a self-exploration algorithm to enable autonomous exploration and image collection.	✗

5.1.2 Non-functional requirements

Non-functional requirements define constraints (or goals) on how the system will do so and include everything that is not related to the functional aspects of the software system. In this section, the non-functional requirements that must be present in the system are presented.

The first non-functional requirement **(NFR-1)** refers to the need for compatibility of the navigation and exploration modules implemented with ROS since these modules must work on the Jetson Nano, which serves as our agent's computational unit.

The second non-functional requirement **(NFR-2)** is related to our internship's primary objective - to propose a more accessible and cheaper alternative to beacon-emitter signals. Therefore, our robot robotic should need the minimum setup for it to run in a new environment - it would only be necessary to put our robot inside the building floor to be explored and turn it on, with the minimum setup required for it to run. Unfortunately, in this project, this goal was not achieved because we had to drop the robotic auto-exploration requirement **(FR-4)** and add the manual calibration of the application using a compass **(FR-3.3)**.

The third non-functional requirement **(NFR-3)** is related to the fact that our robot will use exclusively the laser scan data (and never the images captured by the camera) for navigation and exploration purposes. This is because laser scan data is more reliable and less susceptible to environmental changes (for example, images are affected by illumination factors, such as time of day). However, we also did not want to increase the system complexity, which was a compromise given time and resource constraints.

The fourth non-functional requirement **(NFR-4)** is that the application must give the user clear directions to their destination. It should be easy for users to follow the app instructions to reach their goals.

The fifth non-functional requirement **(NFR-5)** is that the software must be capable of rendering our application. Since heavy computation is performed on the smartphone, the only condition is the smartphone's operating system being compatible with the application software.

The sixth non-functional requirement **(NFR-6)** is that the application must periodically consult the server to update the user location and correct any navigation errors (once every 3 seconds). This is the reason why, in section 4.5 we included Speed as one of the requirements for a future CNN backbone.

Table 5.6: Non-functional requirements.

ID	**NAME**	**DESCRIPTION**	**Implemented**
NFR-1	ROS compatibility	The navigation and exploration modules must be implemented or compatible with ROS.	✓
NFR-2	Robot minimum setup	The robot robotic should need the minimum setup for it to run in a new environment.	✗
NFR-3	Navigation just with laser scan data	Our agent will navigate solely using the laser scan data, as a compromise between performance/complexity.	✓
NFR-4	Clear directions	The application must give the user clear directions to their destination.	✓
NFR-5	Smartphone	The user smartphone must be capable of render our application.	✓
NFR-6	Server real-time operation	Server must be able to output pose predictions and calculate the path every 3 seconds.	✓

5.2 Work methodologies

Two types of meetings are held to keep track of work and ensure all objectives are achieved: weekly meetings with the Eng. Rui Lopes and monthly meetings with both advisors. An assessment of the work is carried out, and adjustments are made if necessary. In the monthly meetings, the participants are Professor Carlos Lisboa Bento, Eng. Rui Lopes and the author. In these sessions, the work developed and the difficulties encountered are discussed, and the next steps are planned, taking into account the supervisor's feedback and making any adjustments to the work planning.

5.3 Risk analysis

Any software project is likely to encounter difficulties that, when they occur, can harm the final product or even prevent its completion and force the modification of development plans. These potential difficulties are risks and must be identified early so that ways to mitigate them and reduce their effects on the project's performance can be planned. To facilitate its analysis, risks have associated attributes such as the expected impact, the probability of happening, and the time window in which they can occur. The impact of a risk is the effect that risk has on the project's success criteria. It is divided into:

1. High when it prevents reaching the success criteria
2. Medium when it is possible to get the success criteria but with great difficulties
3. Low when the success criteria are attainable without great difficulties

The probability of a risk happening is divided into:

1. High when the probability of happening is over 70% probability of occurring
2. Medium when the probability is between 70% and 40%
3. Low when the probability is less than 40%

The time window of risk corresponds to the period from identifying the risk to when it is necessary to deal with it. It is divided into:

1. Large when it is estimated that the risk will occur in an interval greater than three months
2. Medium when the window is between 3 months and one month
3. Short if the time window is less than one month

The identified risks, as well as their attributes and mitigation plan, are presented in the following tables.

Table 5.7: Risk 1 - Destruction/Damaging of JetRacer robot or peripheral devices.

RISK	JetRacer is a very fragile device, manually assembled with several different components. Therefore, there is a risk of destruction/damage to the JetRacer robot or peripheral devices due to an accident, resulting in the device's inability to continue functioning correctly.
IMPACT	Medium
PROBABILITY	Medium
Temporal Window	Short
Mitigation plan	Testing of the robot only on ground-level floors, and only at non business hours.

Table 5.8: Risk 2 - Indoor positioning/orientation model not working.

RISK	While the algorithm has worked on other projects, there is no guarantee it will perform well on the collected dataset.
IMPACT	High
PROBABILITY	Low
Temporal Window	Short
Mitigation plan	Investigation and implementations of alternative models more suitable for the tasks at hand.

Table 5.9: Risk 3 - Augmented reality application not being a helpful user to navigate indoors.

RISK	While other works focus more on developing image-based pose prediction algorithms, this project's objective is to implement an indoor navigation system in an actual test environment. Therefore, there might be unforeseen issues that condition our system performance.
IMPACT	High
PROBABILITY	Medium
Temporal Window	Medium
Mitigation plan	Building several different prototypes with different approaches and choosing the one with the best results.

Table 5.10: Risk 4 - Tasks not completed within deadlines.

RISK	Development of high-complexity features can cause tasks to not be completed within deadlines.
IMPACT	High
PROBABILITY	Medium
Temporal Window	Long
Mitigation plan	Monthly review of the task planning and estimation of necessary time. Adjust approach to achieve goals within deadlines.

Table 5.11: Risk 5 - Lack of mobile app development experience leads to performance issues.

RISK	Lack of mobile app development experience leads to performance issues
IMPACT	High
PROBABILITY	Medium
Temporal Window	Long
Mitigation plan	Enrich theoretical knowledge on the subject.

Table 5.12: Risk 6 - Lack of familiarity with ROS leads to delays.

RISK	Lack of familiarity with ROS leads to delays and the risk of missing deadlines.
IMPACT	Medium
PROBABILITY	High
Temporal Window	Long
Mitigation plan	Enrich theoretical knowledge on the subject. Prioritize the most critical requirements

5.4 Technical specifications

This section presents components diagrams to help plan the software architecture.

This system consists of three distinct sections:

1. Data collection - Robotic agent responsible for constructing a dataset with images associated with poses.
2. Image-based pose prediction model - Machine learning model trained on that dataset to predict poses.
3. Mobile application - Mobile application responsible for handling user interaction; Server responsible for path calculation and use of pose prediction model.

5.4.1 Data collection

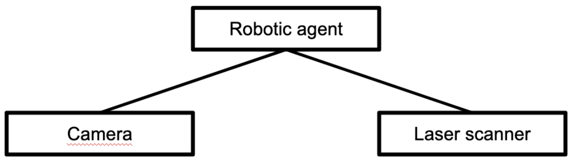


Figure 5.1: Component diagram of the Data Collection component. In the center, we have the robotic agent (JetRacer AI Kit), which communicates with three devices - a laser scanner (RPLIDAR A2) and a camera (pre-installed on JetRacer).

JetRacer AI Kit is the basis of the robotic agent, and it communicates with the other devices:

- Camera - Pre-installed with JetRacer. Communication is done via a Jupiter Notebook.
- Laser Scanner - RPLIDAR A2. Communication is done via ROS (Robotic Operating System).

This robotic agent is responsible for constructing a dataset of images associated with location coordinates.

In a first step, the remote-controlled robotic agent will explore and map the environment. Unfortunately, due to how Google Cartographer works, using a pre-available map in a traditional format (such as a building plant) is impossible.

In the second step, we utilize the produced map to associate images with location coordinates. The map allows all coordinates to have the same reference point - photos from the same place on different time instances will have similar location coordinates (differences derived from scanning error).

5.4.2 Image-based pose prediction model

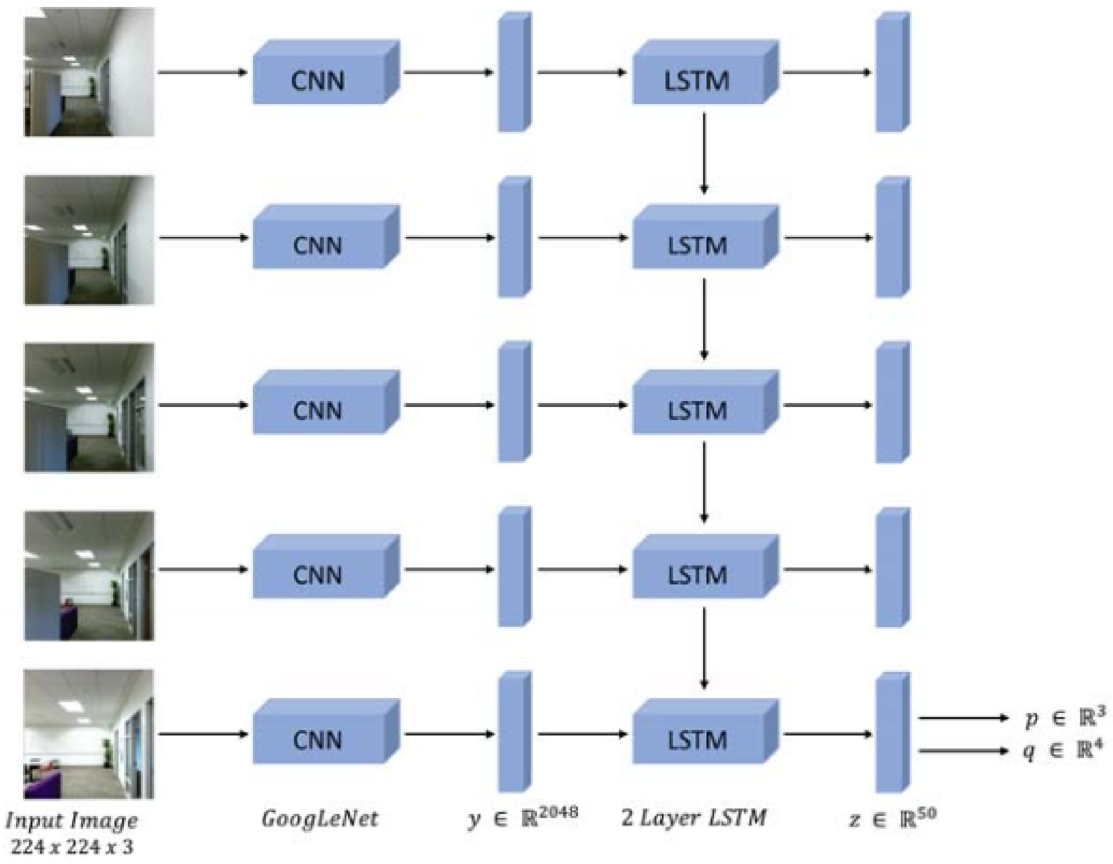


Figure 5.2: Original ContextualNet Model as depicted in [13].

Our image-based pose prediction model is based on ContextualNet [13]. It receives a sequence of images and predicts a pose.

We will use the dataset built by the dataset collection robotic agent to train our model. Therefore, while the original article outputs a full pose (location coordinates and orientation), our model only outputs location coordinates.

5.4.3 Mobile application

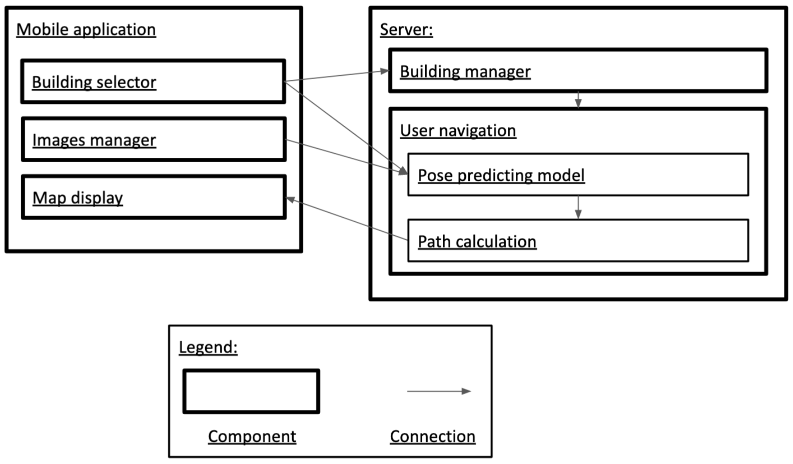


Figure 5.3: Component diagram of the mobile application system's architecture.

In figure 5.3 we find the component diagram of the user mobile application system, composed of two main modules - the mobile application and the server.

The mobile application handles all communication between the user and the system. At the same time, the server manages the building plants and models, uses the pose prediction model to predict user positions, and calculates the user path.

This is how the different components interact with each other:

- The **building selector** sub-module allows the user the select the building they are in. This information is transmitted to the server's component Building manager.
- The **building manager** receives the information from the building selector and returns to the mobile application the building's floor plant.

and the map display sub-module is responsible for displaying the building floor plant and calculating the shortest path from the current user position to its final destination. Finally, the camera capture module will display for the user the recent images being captured for the camera and send these images regularly to the User Navigation module.

The services module comprises two subparts - the building floor plants storage sub-module and the predictor sub-module.

The first one is responsible for storing machine learning models and plants of a different building. According to the facility selected by the user, this sub-module will select other machine learning models (trained explicitly for that building) and floor plants. We will then send the floor plants to the mobile app (to display in the map display sub-module) and tell the predictor sub-module which machine learning model will be used.

This second part - the predictor module - consists of two subparts - an image storage sub-module which stores the most recent images captured by the user (and deletes old photos), and the machine learning sub-module, which uses these images (and the building selected by the user) to output a coordinates and orientation prediction.

5.4.4 Components integration

5.4.5 System workflow

This system workflow is divided into three phases:

1. **Image dataset collection** - In this phase, we used a robotic agent to collect images and associate them with a pose (location coordinates). The location of the automated agent is calculated using the LIDAR device.
2. **Pose prediction model implementation** - In this phase, we implement and train a machine learning model using the previously built dataset of images associated with a pose.
3. **Mobile application development** - After obtaining our prediction model, we deploy it to a server to use in our mobile application to predict the user's pose.

The first step is the creation of the dataset of the environment's images associated with position coordinates (an x,y position). This phase consists of two steps:

- Environment mapping - In the first step, we use the laser scanner device to create a map using the Google Cartographer mapping algorithm.
- Dataset collection - Using the previously built map, we can collect images and associate them with coordinates. These coordinates all have the same reference point, defined by the mapping algorithm. The map also allows collecting images at different instances while keeping the same reference point.

In the second step, we train the model of our algorithm with the dataset of our environment's images. Finally, we split the photos into distinct train and test sequences and used them to select the best model.

Finally, we deploy this machine learning model to predict the user's coordinates in our mobile application system. We also used the map generated by the Google Cartographer mapping algorithm as a plant to help the user navigate.

5.4.6 Components introduced error

These different phases introduce several sources of error in the final model. Therefore, to reduce the system's total error, it is necessary to measure each phase's error and reduce it. The following sections discuss how we measure each phase error and the steps taken to reduce it. Finally, we perform tests to determine the system's total error.

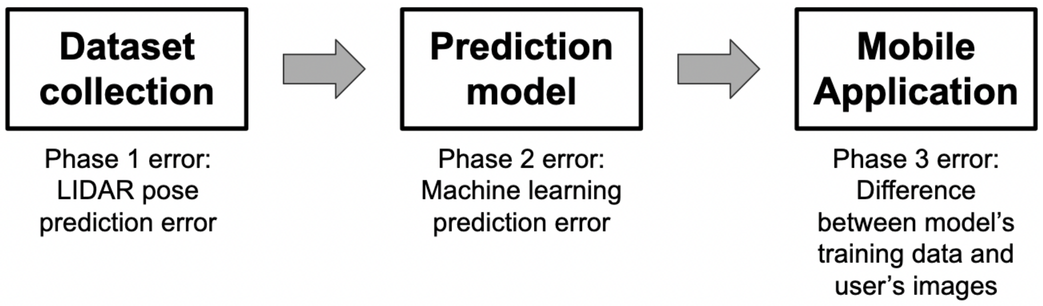

Dataset collection
Prediction model
Mobile Application
Phase 1 error: LIDAR pose prediction error
Phase 2 error: Machine learning prediction error
Phase 3 error: Difference between model's training data and user's images

Figure 5.4: **Phase 1** - Our Cartographer ROS uses sensor data to map the environment and predict the robotic agent's pose. The closer these predicted values are to their real values, the less error is generated from this step. **Phase 2** - Using the previously constructed image dataset to train a pose prediction model. The more precise this model, the less error it has. **Phase 3** - After training our prediction model, we deploy it to a server to be used in our mobile application. Due to the difference between our dataset images and the user smartphone images, there is always some error generated from this step.

We will analyze the generated error in each phase and work on ways to reduce it.

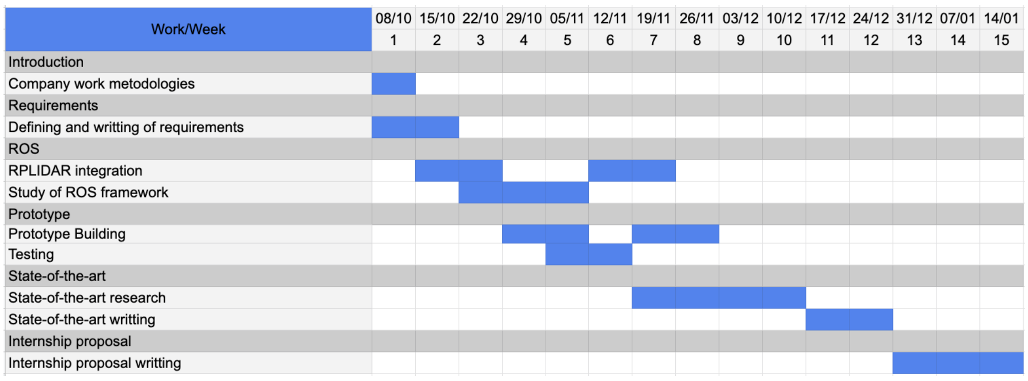
5.4.7 Project limitations and constraints

Due to the software and hardware utilized, this project presents several limitations that will have to be addressed in future work, which include:

- **Only works on one floor** - Due to our prototype limitations and time constraint, we limited the scope of our system to a single floor of the Critical Software building.
- **Very susceptible to dynamic scenarios** - This solution works better in a static environment. The pose prediction model assumes that the environment does not change, but it does. Drastic environmental changes can affect the model's performance because it was trained in a different environment from what it currently looks like. A solution to handle dynamic environments will have to be found in future projects.

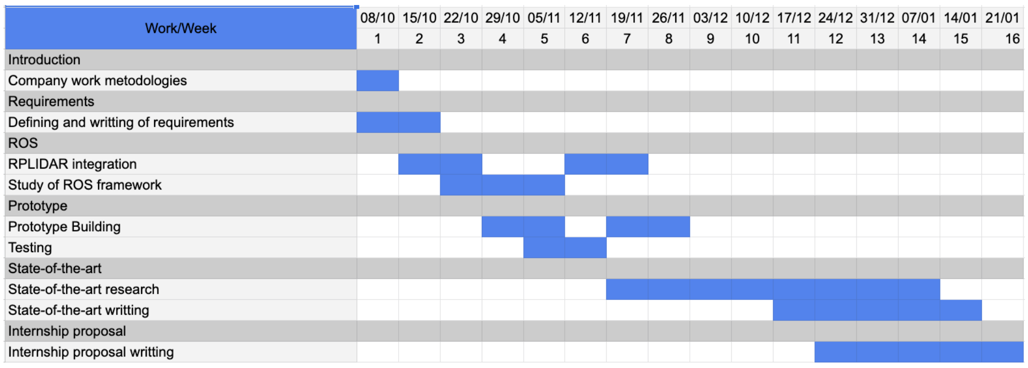
5.4.8 Planning

In this section, the main tasks performed during the first semester are detailed here. We created this Gantt diagram so that a plan can be easily formulated and to make analyzing and adapting tasks (depending on the work done in each timeline) easier. The initial diagram and the final diagram for the first semester are presented below.



Work/Week	08/10	15/10	22/10	29/10	05/11	12/11	19/11	26/11	03/12	10/12	17/12	24/12	31/12	07/01	14/01
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Introduction															
Company work metodologies	🟦														
Requirements															
Defining and writting of requirements	🟦	🟦													
ROS															
RPLIDAR integration		🟦	🟦			🟦	🟦								
Study of ROS framework			🟦	🟦	🟦										
Prototype															
Prototype Building				🟦	🟦		🟦	🟦							
Testing					🟦	🟦									
State-of-the-art															
State-of-the-art research							🟦	🟦	🟦	🟦					
State-of-the-art writting											🟦	🟦			
Internship proposal															
Internship proposal writting													🟦	🟦	🟦

Figure 5.5: The first semester planned work schedule.



Work/Week	08/10	15/10	22/10	29/10	05/11	12/11	19/11	26/11	03/12	10/12	17/12	24/12	31/12	07/01	14/01	21/01
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Introduction																
Company work metodologies	🟦															
Requirements																
Defining and writting of requirements	🟦	🟦														
ROS																
RPLIDAR integration		🟦	🟦			🟦	🟦									
Study of ROS framework			🟦	🟦	🟦											
Prototype																
Prototype Building				🟦	🟦		🟦	🟦								
Testing					🟦	🟦										
State-of-the-art																
State-of-the-art research							🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
State-of-the-art writting											🟦	🟦	🟦	🟦	🟦	
Internship proposal																
Internship proposal writting												🟦	🟦	🟦	🟦	🟦

Figure 5.6: First semester actual work schedule.

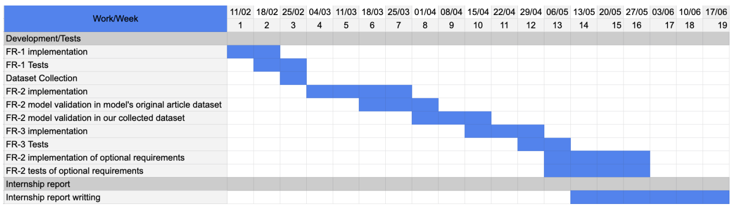
The initial tasks remained unaltered. Initially, I started by learning about the company policies and more about the project, and the requirements were defined.

Using ROS to integrate RPLIDAR and the JetRacer AI Kit was necessary. Due to the unfamiliarity with this framework, it is essential to allocate some time to study it properly.

Two prototypes were built - one cardboard and another with 3D printed pieces. Unfortunately, the first one was too fragile and unstable, which forced us to design a better-suited one.

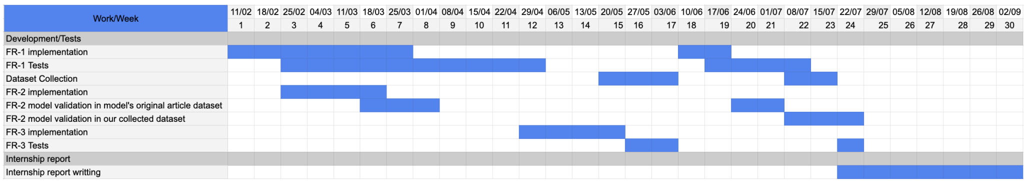
Both of these parts were smoother than expected, and also, because some exam dates were pushed forwards due to COVID, more time was spent on research plus the state-of-the-art, which took more time than anticipated.

The initial diagram and the final diagram for the second semester are presented below.



Work/Week	11/02	18/02	25/02	04/03	11/03	18/03	25/03	01/04	08/04	15/04	22/04	29/04	06/05	13/05	20/05	27/05	03/06	10/06	17/06
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Development/Tests																			
FR-1 implementation	🟦	🟦																	
FR-1 Tests		🟦	🟦																
Dataset Collection			🟦																
FR-2 implementation				🟦	🟦	🟦	🟦												
FR-2 model validation in model's original article dataset						🟦	🟦	🟦											
FR-2 model validation in our collected dataset								🟦	🟦	🟦									
FR-3 implementation										🟦	🟦	🟦							
FR-3 Tests												🟦	🟦						
FR-2 implementation of optional requirements													🟦	🟦	🟦	🟦			
FR-2 tests of optional requirements													🟦	🟦	🟦	🟦			
Internship report																			
Internship report writting														🟦	🟦	🟦	🟦	🟦	🟦

Figure 5.7: The second semester planned work schedule.



Work/Week	11/02	18/02	25/02	04/03	11/03	18/03	25/03	01/04	08/04	15/04	22/04	29/04	06/05	13/05	20/05	27/05	03/06	10/06	17/06	24/06	01/07	08/07	15/07	22/07	29/07	05/08	12/08	19/08	26/08	02/09
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Development/Tests																														
FR-1 implementation	🟦	🟦	🟦	🟦	🟦	🟦	🟦											🟦	🟦											
FR-1 Tests			🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦							🟦	🟦	🟦	🟦								
Dataset Collection															🟦	🟦	🟦					🟦	🟦							
FR-2 implementation			🟦	🟦	🟦	🟦																								
FR-2 model validation in model's original article dataset						🟦	🟦	🟦												🟦	🟦									
FR-2 model validation in our collected dataset																						🟦	🟦	🟦						
FR-3 implementation												🟦	🟦	🟦	🟦															
FR-3 Tests																🟦	🟦							🟦						
Internship report																														
Internship report writting																								🟦	🟦	🟦	🟦	🟦	🟦	🟦

Figure 5.8: The second semester actual work schedule.

This significant delay in FR-1 is due to the materialization of the risks 1 (Destruction/Damaging of JetRacer robot or peripheral devices) and 6 (Lack of familiarity with ROS leads to delays):

- Destruction/Damaging of JetRacer robot or peripheral devices - We ended up replacing the RPLIDAR A2, our laser scanner device, because it was damaged. While it significantly reduced our mapping error, it consumed some time. Additionally, in the final stages of the project, the robotic agent stopped working. It took some time to figure out what was wrong, and in the end, we discovered that one of the servo motors had broken down. After replacing it, we resumed our dataset collection phase.

- Lack of familiarity with ROS leads to delays - In the first semester, we believed that Cartographer ROS and RPLIDAR A2 would be able to provide both location and orientation. However, a more in-depth analysis concluded that only location data was available, which forced us to come up with an alternative solution using the Environment Sensor module, which introduced more complexity to our system.

This significant delay in FR-1 consumed too much time and eliminated any time we had for FR-4 and the optional requirements of FR-2. Unfortunately, therefore, we were not able to proceed with their requirements.

The other tasks were not so affected and were implemented and tested while components ordering and other functions for FR-1 were on standby.

Chapter 6

Implementation

6.1 Image Dataset Collection

We need a dataset of the environment's images associated with the location coordinates to train our pose prediction model. A remote-controlled robotic agent performs this process.

Two different processes were utilized to obtain location coordinates (x, y) and orientation (θ), which will be described in the sections below. Unfortunately, we were not successful in obtaining the images' orientation angles, forcing us to devise an alternative solution: both the orientation retrieving process and the alternative solution, which will be depicted below.

6.1.1 Localization

To obtain the localization of our robotic , we utilized a mapping algorithm called Google Cartographer [10], which uses laser scans from our RPLIDAR A2 to map and predict the position of our robotic agent. We integrate the software and the LIDAR communicated using ROS (Robot Operating System). LIDAR data can be affected by reflective3 and transparent surfaces.

The reason we utilized this technology was that Critical Software already had these components in its possession, which greatly reduced research and acquisition time. CSW had in its possession a RPLIDAR A2 [5] and a JetRacer AI Kit [5].

Localization stages

When collecting our images and associating them with x,y location coordinates, we want to make sure all these coordinates have the same reference point so that pictures in the exact location inside the building have similar coordinates values. To achieve this goal, the image dataset collection process has two distinction steps:

1. **Map construction** - In this first stage, we will transverse our environment

(in our case, the ground floor of Critical Software building A in Coimbra) to map it. This map has an origin point that will serve as a reference point for the image position values. We can use the origin point with other metadata to get any point's x,y position, with the origin point defined as (0, 0).

2. **Image dataset collection** - In this next step, we use our previously built map to position our robotic agent automatically - the same place in the building at different time instances should generate approximated coordinates. Our map's reference point always ensures that images from other time instances on the exact location have similar coordinates values.

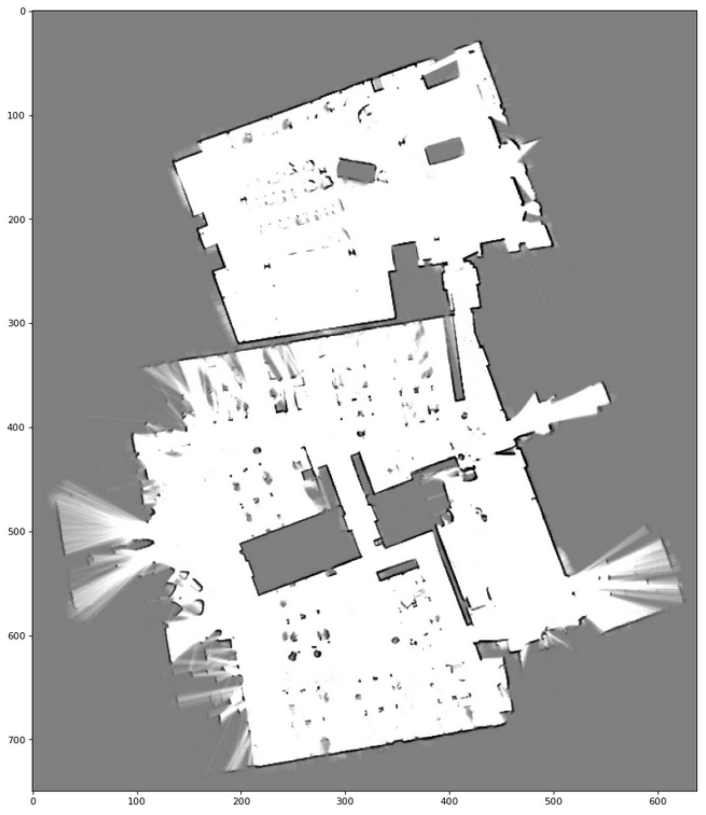


Figure 6.1: Example of a map computed by Google Cartographer using laser scan data. The white cells represent open space (rooms, corridors, and others), black cells represent occupied space (walls, chairs, structural beams, for example), and grey cells represent unknown or uncertain space (unexplored regions).

6.1.2 Orientation

Orientation is the angle the robotic agent currently faces relative to the North Pole. In this project, we planned to use our Environment sensor module's accelerometer and magnetometer to create a compass heading, which would be the cardinal directions used for navigation and geographic orientation.

Orientation original solution

To obtain a compass heading - the orientation of the robotic agent in relation to the north magnetic pole - we followed the formulas described in [40], which describes how we can convert accelerometer and magnetometer values into a tilt2-compensated compass heading.

A magnetometer is a device that measures a magnetic field or magnetic dipole moment. A compass is one such device that measures the direction of an ambient magnetic field, in this case, the Earth's magnetic field. If a compass were sitting in the local horizontal plane, then the roll and pitch angles would be zero, and the heading would be calculated as:

$$Heading = arcTan(Yh/Xh) \tag{6.1}$$

where Xh and Yh represent the Earth's horizontal magnetic field components; as the compass is rotated, the compass heading would sweep 0° to 360°, referenced to the magnetic north.

However, if the compass is tilted, the tilt angles (roll and pitch) and all three magnetic field components (X, Y, Z) must be used to calculate the heading. The formula required to calculate the compass heading is defined as:

$$\begin{aligned} Xh &= X_{normalized} * cos(pitch) + Y * sin(roll) * sin(pitch) - Z * cos(roll) * sin(pitch) \\ Yh &= Y_{normalized} * cos(roll) + Z * sin(roll) \end{aligned} \tag{6.2}$$

where

$$\begin{aligned} X_{normalized} &= X - (X_{max} - X_{min})/2 \\ Y_{normalized} &= Y - (Y_{max} - Y_{min})/2 \end{aligned} \tag{6.3}$$

where X_{max}, Y_{max}, X_{min} and Y_{min} are the biggest and smallest values of the magnetometer X and Y reading values (This last equation is done to offset the magnetometer's inclination effect on the reading values).

Lastly, to calculate the compass heading angle, we apply the following formula:

$$|\theta| = \begin{cases} 180 - arcTan(Yh/Xh) & \text{if } Xh < 0 \\ -arcTan(Yh/Xh) & \text{if } Xh > 0, Yh < 0 \\ 360 - arcTan(Yh/Xh) & \text{if } Xh > 0, Yh > 0 \\ 90 & \text{if } Xh = 0, Yh < 0 \\ 270 & \text{if } Xh = 0, Yh > 0 \end{cases}$$

Any acceleration will affect the tilt or accelerometer outputs and result in heading errors. For example, the robotic agent makes a turn or accelerates. In that case, that will cause the tilt sensors to experience additional forces in addition to gravity, and the compass heading will be in error.

We programmed the car to stop every 3 seconds to combat this acceleration effect and measure the accelerometer and magnetometer values. Then, at the end of the image collection process, we calculate the angle using the sensor's data and associate them with the images.

Orientation of original solution results

We were not able to collect quality data from the Environment Sensor Module. Even when the sensor was still and not moving, it gave erratic values. The most likely cause was the magnetic fields generated by the robotic agent's electric engines.

6.1.3 Orientation proposed alternatives

Due to the issues found in the initially proposed solution for retrieving the image orientation, we were forced to come up with alternative solutions:

- **Trajectory inferred orientation** - Using the location coordinates (x,y) provided by Google Cartographer, we could derive an orientation angle throughout the time. This solution could provide orientation for the application on all smartphones.
- **Map orientation calibration** - Using a smartphone compass and 2 points manually chosen in the map, we can calibrate the map and then use the smartphone compass to get an orientation. This solution would not allow us to associate images with direction, so our model could not provide orientation. This means orientation information would be limited to smartphones equipped with a compass.

6.2 Location and orientation prediction Model

As stated in section 4.3, the best model to accomplish our objectives given our requirements is the ContextualNet ([13]). Unfortunately, we were unable to find a pre-implemented model. Therefore we implemented this model using PyTorch. This process consisted of a few steps:

1. Model implementation (according to specifications in the original article).
2. Model validation in the 7-Scenes Dataset [22]. This dataset consists of RGB-D camera images and contains sequences in different scenarios. It is used in [25], [13] and [26] to validate and compare their results and is composed of several image sequences in different scenarios. However, due to time constraints, we are validating our model in solely one scene - "chess". Therefore, we tested other parameters and configurations to obtain the best possible configuration.
3. Model training and evaluation in the dataset of our test environment - ground floor of CSW office building in Coimbra. (collected as described in 6.1). We will use the best parameter configuration obtained in the 7-Scenes dataset model validation.
4. Model evaluation in images captured from a smartphone camera. This step will give us a sense of the performance of our final mobile application.

Model implementation

As mentioned before, we could not find an implemented version of ContextualNet. Therefore, one was implemented from scratch. Fortunately, we found a GoogleNet implementation and Places weights on `https://github.com/nemoxb/PoseNet-PyTorch`, reducing our implementation effort. All code was implemented on the PyTorch framework.

Some notes about the model extracted from the original article:

- GoogLeNet's input image size is $224 \times 224 \times 3$.
- GoogleNet's last fully connected layer is modified to generate a feature vector of size 2048 instead of 1024. The first LSTM layer uses this feature vector as input.
- The weights of the CNN layers were initialized with the weights trained on the Places dataset (as indicated in [41]).
- The first LSTM layer contains 512 hidden cells while the second layer contains 50. The second LSTM layer is connected to 2 separate fully connected layers with a many-to-one configuration.
- The sequence size of the LSTM layers was set to 3 - 3 images inputted into our model to obtain a pose.

- The training process consists of two steps - In the first step, the weights of the GoogleNet are fine-tuned, and the weights of the LSTM layers are frozen. In the second step, we only optimize the weights of the LSTM layers and freeze the weights of GoogleNet.
- The weights of the LSTM layers were initialized using a glorot uniform distribution.
- In this article, it was utilized the Adam [42] optimizer with the following parameters - $\epsilon = 10\hat{-}8$, $\beta1 = 0.9$, and $\beta2 = 0.999$.

Some implementation details are missing from the article that can impact the model performance, specifically the applied regularization techniques. For example, while we can infer from the article that both dropout and weight decay were applied to this model, we are only given the weight decay value - 10^{-8}. It is also not explicitly said what other forms of regularization are used in addition to dropout and weight.

6.2.1 Model validation in the 7-Scenes Dataset

The 7-scenes dataset [22] is composed of tracked RGB-D camera frames in different scenarios. All scenes were recorded from a handheld Kinect RGB-D camera at 640×480 resolution, and the predicted poses from the Kinect are considered the "ground truth". In addition, there are predefined training and test sequences to train and test different algorithms and compare them.

Each sequence consists of 500-1000 frames, with each frame associated with three files:

- **Color** - frame-XXXXXX.color.png . It consists of RGB images.
- **Depth** - frame-XXXXXX.depth.png (depth in millimeters). It consists of the distance of the points in the image to the camera (NOT used in this project).
- **Pose** - frame-XXXXXX.pose.txt (camera-to-world, 4×4 matrix in homogeneous coordinates). It consists of a matrix encoding the translation and rotation of the camera in relation to the origin point.

In this project, we only utilized the RGB images and the homogeneous coordinates matrices (to extract the poses).

As explained in this lecture of the course of Computer Science at Colombia University (`https://www.cs.columbia.edu/~allen/F19/NOTES/homogeneous_matrices.pdf`), homogenous coordinates are a mathematical expression of a set of rotations and translations encoded into a 4x4 matrix. Homogeneous coordinates matrices are useful in computer graphics to transform a point from a certain point of view into a topic from a different point of view.

A homogenous transformation matrix **H** is defined as

$$\mathbf{H} = Translation(x_1, y_1, z_1) * Rotation_x(x_2) * Rotation_y(y_2) * Rotation_z(z_2) \quad (6.4)$$

where x_1, y_1, z_1 are the translation components along the x,y,z axis and x_2, y_2, z_2 are the rotation components along the x,y,z axis.

More specifically, each component formula is defined as

$$Translation(x_1, y_1, z_1) = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotation_x(x_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(x_2) & -sin(x_2) & 0 \\ 0 & sin(x_2) & cos(x_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotation_y(y_2) = \begin{bmatrix} cos(y_2) & 0 & sin(y_2) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(y_2) & 0 & cos(y_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rotation_z(z_2) = \begin{bmatrix} cos(z_2) & -sin(z_2) & 0 & 0 \\ sin(z_2) & cos(z_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is necessary to retrieve the position and orientation from these matrices:

- The position is obtained directly from the matrices - the right-hand 3×1 column describes its position.
- The upper left 3×3 matrix represents the object's orientation. However, the conversion is a bit more tricky. Therefore, we utilized the Python library Pyquaternion (https://pypi.org/project/pyquaternion/), which converts the matrices into quaternions.

6.2.2 Model validation in the collected dataset

Utilizing the dataset collected by our robotic agent and the parameters used for the 7-scene dataset validation, we intend to train our model to be used in the Critical Software office space.

To have more accurate results, we collected image sequences using a smartphone. More details and the results are presented in section 7.3.

6.3 Mobile Application System

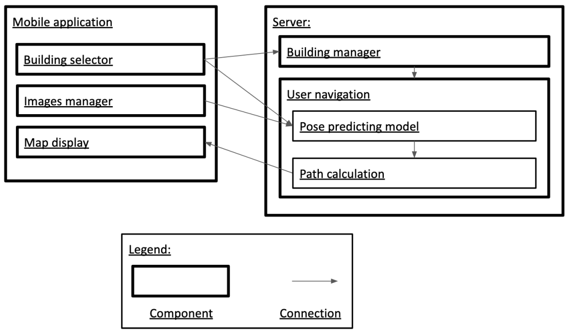


Figure 6.2: Component diagram of the mobile application system's architecture.

In this section, we will discuss the implementation of the mobile application system.

Our mobile application was developed using Flutter (`https://flutter.dev/`), an open-source framework developed by Google that allows building applications for Android and IOS from a single codebase.

Our server, a lightweight web application framework, was implemented in Python using the Flask library (`https://pypi.org/project/Flask/`).

6.3.1 Mobile Application

In figure 6.3 we can see a screenshot of a working prototype of our mobile app. This is how the different parts interact:

- **Building selector** - On the top, we have a scroll-down button where the user selects the building they are in (at this moment, it only has one working option). After choosing an alternative, the mobile app requests the building plant's server.

- **Camera preview** - This component captures images and sends them to the image manager. It also displays the video feed to the user.

- **Map display** - Displays the building plant and shows some interesting points on top (points manually set on the configuration file). After the user has touched a point in the map (setting it as the destination), the mobile app will send images (stored in the image manager) every 3 seconds to the server to return the user position and path to the destination. In the case of a mobile phone with a compass, it displays the orientation with a blue arrow. Otherwise, the arrow is removed.

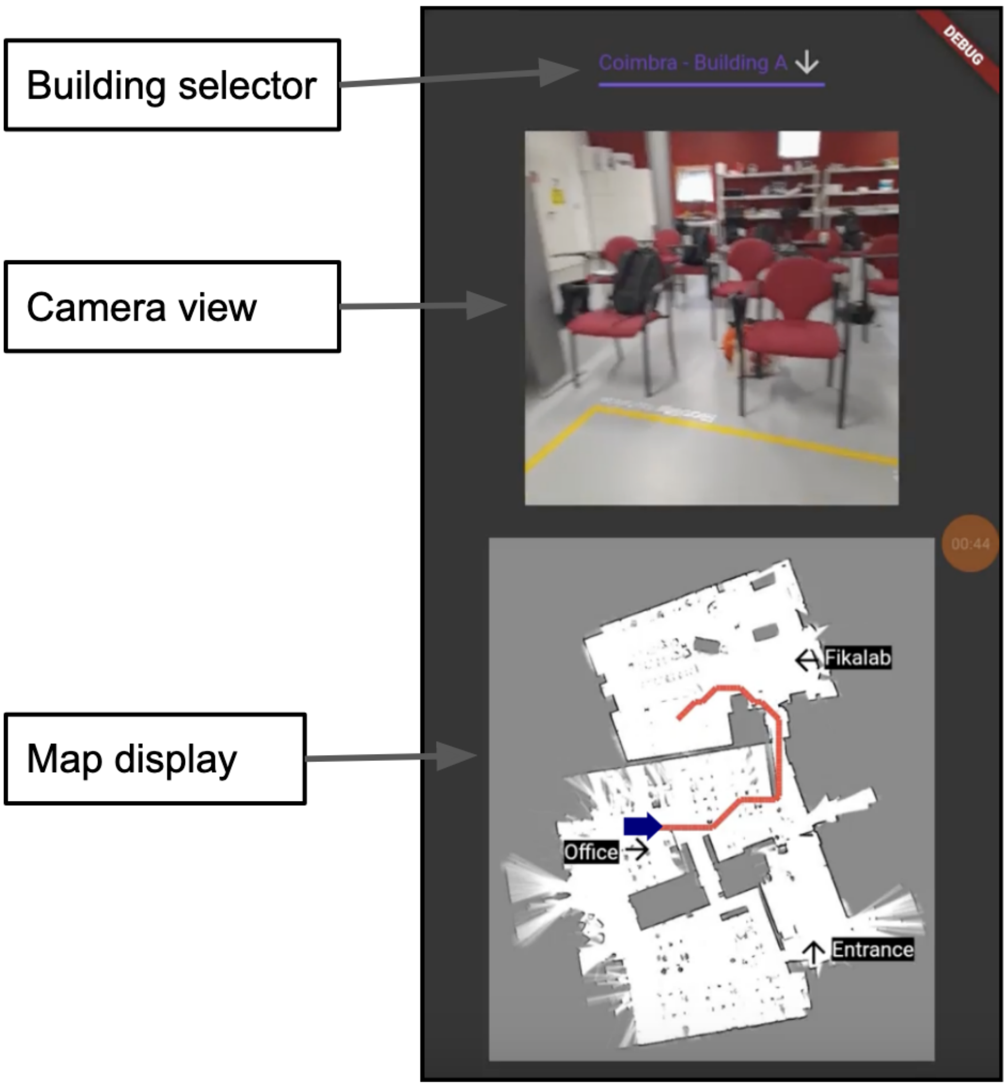


Figure 6.3: Example of use of the mobile application.

6.3.2 Server

The server comprises two components - the Building manager and the User navigation.

The Building manager is responsible for managing the building plants and the models. According to the option selected in the Mobile App's Building Selector, this component provides the correct building plant for the Mobile App's Map display and the right building plant and prediction model for the User Navigation Model.

The User Navigation module is responsible for two things:

1. Calculating the user position - By using the image sequences provided by the Image Manager and the prediction model supplied by the Building Manager, the server calculates the user position.

2. Calculating the path - Given the previously calculated position, the destination

selected by the user, and the map provided by the Building Manager, we calculate the path. We then return it to our mobile app.

To calculate our path, the server implements the A* algorithm (`https://pypi.org/project/pathfinding/`). We chose this algorithm because it was the fastest readily available option and satisfied our requirements.

A*

A* Star [43] was created as part of the Shakey project (`https://www.sri.com/hoi/shakey-the-robot/`). A* Star works to find a path from the start to the destination.

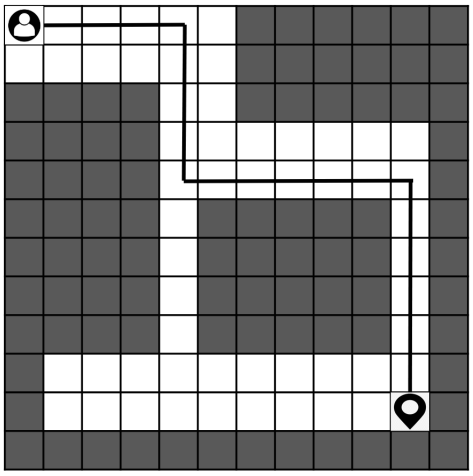


Figure 6.4: In a 2D Grid, obstacles are represented by dark cells, and the free path is represented by white cells. For example, A* Star is an algorithm to find a path from the user to its destination. Our mobile application uses the Dataset Collection's generated map as a 2D grid for the A* algorithm.

At each step, A* picks the node according to a value f, with $f = g + h$. We define g and h as:

- g - the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
- h - also known as a heuristic, this is the estimated movement cost to move from that given square on the grid to the final destination. There can be many ways to calculate this heuristic.

At each step, it picks the node/cell having the lowest f and processes that node/cell.

The algorithm works as follows:

- We initialize the Open list - List of possible next steps. The first element is our starting points, with $f = 0$.

- We initialize the Closed list - Already analyzed points.
- While the open list is not empty:
  - Find the node with the smallest f on the open list, call it q.
  - Remove q from the open list.
  - Analyse q's eight children (neighboring points) and set their parents to q.
  - For each child:
    - If the child is the goal, stop searching.
    - Else, compute g and h to obtain *f*.
    - If a node with the same position as the child is in the OPEN list and has a lower *f*, skip this child.
    - If a node with the same position as the child is in the CLOSED list, which has a lower *f*, skip this successor. Otherwise, add the node to the open list.
  - push q on the closed list.

To calculate the heuristic, the implementation we utilize uses the Manhattan distance.

Chapter 7

Results

In this chapter, we describe and present the results on the tests performed for all phases - pose quality of the dataset, pose prediction model accuracy, and mobile application performance.

7.1 Dataset

7.1.1 Localization

Evaluation methods

After obtaining our map and collecting our image dataset, we evaluate the predicted pose values with the actual coordinates. Ideally, we would mark a path the robotic agent would follow and compare the predicted trajectory pose values with the actual trajectory pose values. Unfortunately, that is impossible because the test environment is a Critical Software office space used by other employees and may not be damaged. Instead, we will follow the same procedure as described in [21] - we define a set of evaluation points spread across the map, from which we know the actual world coordinates, and compare the predicted points by our robotic agent.

The original RPLIDAR A2 device presented a crack and had some bad connections issues, but it mostly performed well and was utilized in the earlier experiments. Later, we ordered a brand new RPLIDAR A2 laser scanner and repeated the experiments.

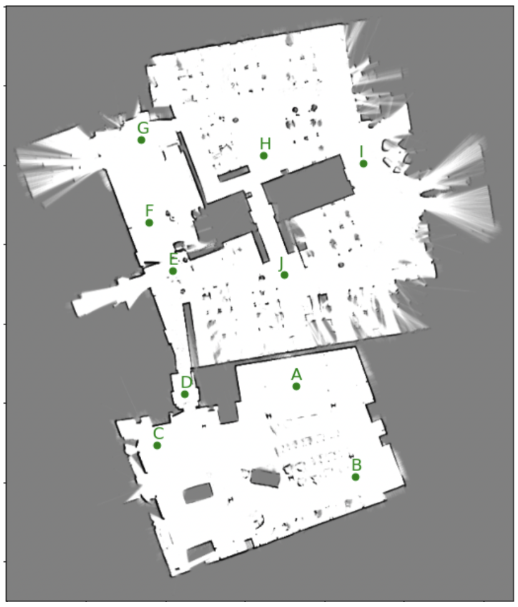


Figure 7.1: **Left** - Set of points defined in the test environment (Critical Software building's ground floor) as evaluation points. **Right** - Point D marked in building floor.

Old RPLIDAR results

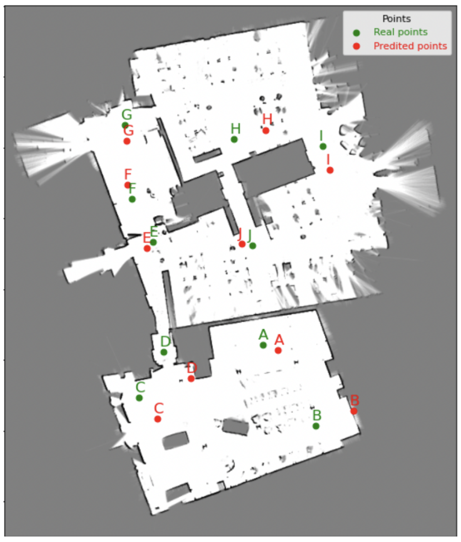


Figure 7.2: Best results obtained with old LIDAR device - **average error of 1.622 m and a variance of 0.590 m**.

After testing several different configurations for Cartographer on our original RPLIDAR A2, our best result was an **average error of 1.622 m and a variance of 0.590m**. However, some glass walls and doors were not detected in this environment, affecting the final location prediction.

New RPLIDAR results

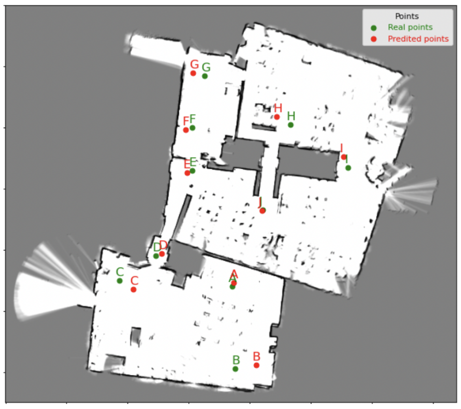


Figure 7.3: Results obtained with the new LIDAR device - **average error of 0.830 m and a variance of 0.238 m** - using the same configuration as in the old LIDAR device.

After acquiring a new RPLIDAR A2 device, we tested the same Cartographer parameters at the same points. Again, we achieved much better results - **an average error of 0.830 m and a variance of 0.238 m**.

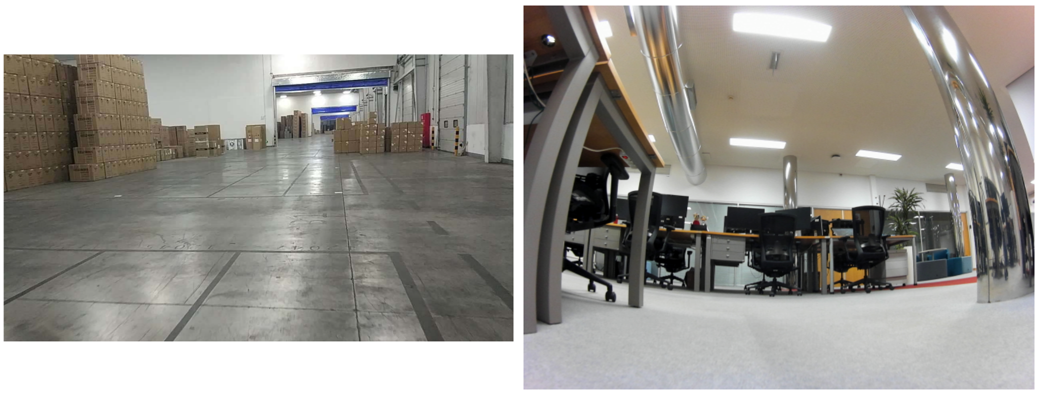


Figure 7.4: Left: In the test environment of the article, [21], Google Cartographer algorithm was able to achieve an average error of less than 10 cm. However, we must point out that the environment does not contain reflective or transparent surfaces and is more simple. **Right:** As we can observe, the CSW text environment is much more complex, with lots of chairs, tables, and reflective and transparent surfaces.

In the article [21] the Google Cartographer algorithm achieved an average error of less than 10 cm, a result better than we reached. However, it must be noted that our environment is much more complex - the testing environment of [21] is a warehouse filled with boxes. In contrast, our testing environment is an office space with many objects that can either move positions or reflect or not the laser, such as chairs, glass walls, and doors. All these factors affecting the algorithm precision (environments as depicted in figure 7.4).

7.1.2 Orientation

As previously discussed in section 6.1.2, an alternative to the original solution based on the Environment Sensors module data is:

- **Trajectory inferred orientation** - Using the location coordinates (x,y) provided by Google Cartographer, we could derive an orientation angle throughout the time. This solution could provide orientation for the application on all smartphones.

- **Map orientation calibration** - Using a smartphone compass and 2 points manually chosen in the map, we can calibrate the map and then use the smartphone compass to get an orientation. This solution would not allow us to associate images with direction, so our model could not provide orientation. This means orientation information would be limited to smartphones equipped with a compass.

Trajectory inferred orientation

We performed some tests to determine if we could utilize the trajectory to infer the angles.

Using a similar approach to the mapping approach, we defined some pair points (whose coordinates were known), from which we can derivate two orientation angles (all angles are anti-clockwise):

- Real angle - Angle derived from the points actual coordinates.

- Calculated angle - Angle derived from the point coordinates calculated by the Google Cartographer mapping algorithm.

We tested this orientation method in the best possible scenario - points separated by a long distance, where the coordinates values error would be reduced.

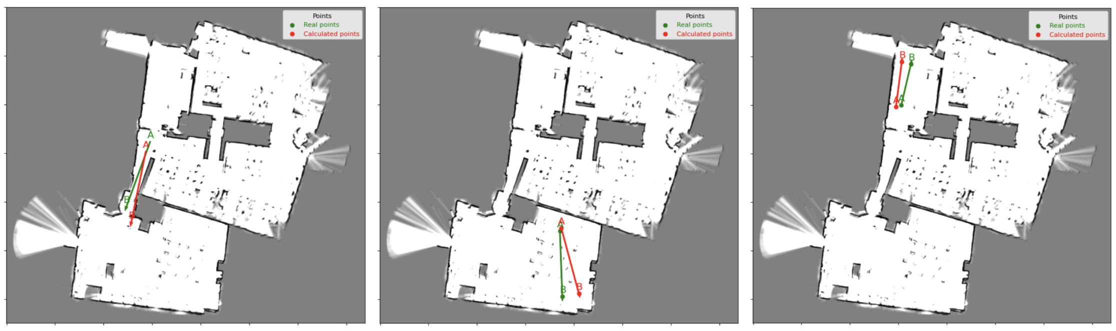


Figure 7.5: Green lines are the real angles, and red lines are the predicted angles. The line's direction is from their respective point A to point B. Here are the errors for the different pair points - **Left:** 8.49º; **Center:** 12.46º; **Right:** 6.19º.

Considering the magnitude of the errors, we cannot use this solution. These values happen in the best scenario - pairs of distant points - and we can expect that closer pair points would have more significant error values. If we trained our model using these values, it would not provide good results.

Map orientation calibration

Using the coordinates of the real point, we can draw a line on the map and get an orientation value of the line.

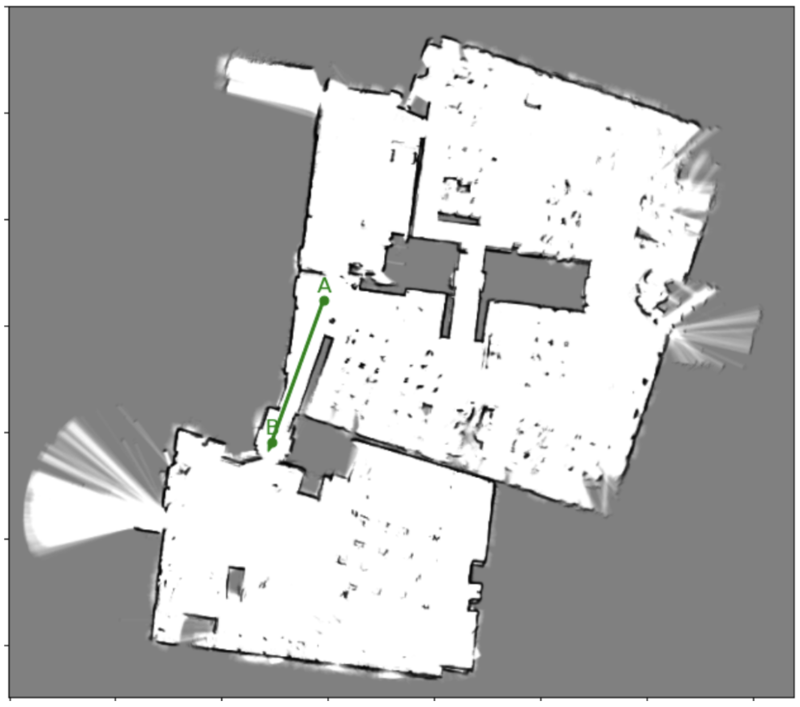


Figure 7.6: We can calibrate our map using a calculated offset using two marked points and a compass. For example, the angle of line A->B is 160º, and the compass heading (angle to the magnetic north) is 33º (anti-clockwise). With these two values, we get an offset of 127º.

Afterward, we used a compass to measure the line orientation according to the magnetic north. Using these two angles, we calculate an offset, passed on to the mobile app using a configuration file, which we provide to the mobile application through a configuration file.

Even though this solution limits the orientation information to mobile phones with a compass, it is the only option that can provide good results. Therefore, we proceeded with this solution.

7.1.3 Generated dataset

We generated four sequences of images (collected at 2Hz) associated with location coordinates (x,y):

- Train sequences (4978 images):
  - Sequence 0 - 1570 images
  - Sequence 1 - 1617 images
  - Sequence 2 - 1791 images
- Sequence test - 1611 images

7.2 Location and orientation prediction Model

To overcome these knowledge gaps and improve our model results, we performed several different experiences to obtain the best parameters:

- **Dropout rate and Weight decay** - The article does not specify all regularization methods and parameters. Therefore, we experimented with several dropout and weight decay values to select the best ones.
- β - By changing the parameter β, we expect to obtain a better compromise between position/orientation accuracy.

7.2.1 7-scenes validation - Experiments results

For training our model, we followed the instructions as depicted in the original article:

- Due to time constraints, we only tested on scenario "chess" of the 7-scenes dataset.
- We used the same training/testing sequences, indicated as in the 7-Scenes dataset documentation.
- We used a batch size of 64.
- In the first stage, We start by freezing the GoogLeNet backbone and training the LSTM layers. Then, we freeze the LSTM layers in the second stage and train the backbone. We repeat this process three times.
- We utilized the Adam optimizer with the same parameters as the original article [13].
- Another change we implemented to the training procedure was using the Plateau detection [44]. This method reduces the learning rate by a factor $/alpha$ when detecting that a specific metric has stopped improving and can improve the model performance. In our case, we have defined the learning rate to decrease by ten after the model failed to reduce the training error by 10% after ten iterations. After detecting the second plateau, we switch the layers being frozen. This process effectively means that we start with an initial learning rate of 10^{-4} and then decrease it to 10^{-5} when detecting a plateau.

In this experiment, we will determine the best values for the following parameters:

- **Dropout rate**
- **L1 penalty**
- **Weight decay**

- $\boldsymbol{\beta}$

ContextualNet accuracy uses the median position/orientation error as the performance measure to compare our algorithm implementation with the original article.

In each table, we underlined the parameters that provided the best results.

7-scenes validation - Experiments results - L1 penalty

We started by determining the best l1 penalty value while keeping the following parameters static:

- **Dropout rate** = 0.5
- **Weight decay** = 10^{-8}
- $\boldsymbol{\beta}$ = 250

Table 7.1: Prediction model experiment table 1 - L1 penalty.

ID	L1 penalty	Validation error	Median position/orientation errors (m/º)
Original article	-	-	0.15 / 6.12
0	0	4489	<u>**0.215 / 6.678**</u>
1	10^{-3}	6931	0.257 / 6.618
2	10^{-4}	3713	0.247 / 6.224
3	10^{-5}	4362	0.279 / 6.644
4	10^{-6}	1545	0.271 / 6.412

7-scenes validation - Experiments results - Weight decay

We started by determining the best weight decay value while keeping the following parameters static:

- **L1 penalty** = 0
- **Dropout rate** = 0.5
- $\boldsymbol{\beta}$ = 250

Table 7.2: Prediction model experiment table 2 - Weight decay.

ID	Weight decay	Validation error	Median position/orientation errors (m/º)
Original article	10^{-8}	-	0.15 / 6.12
5	0	5350	0.257 / 7.157
6	10^{-7}	6544	0.258 / 6.889
0	10^{-8}	4489	**0.215 / 6.678**
7	10^{-9}	4364	0.308 / 6.821
8	10^{-10}	4384	0.267 / 6.487
9	10^{-11}	4642	0.245 / 6.706

We found the best weight decay value from the experiments is 10^{-8}.

Experiments results - Dropout rate

We started by determining the best dropout rate value while keeping the following parameters static:

- **L1 penalty** = 0
- **Weight decay** = 10^{-8}
- $\boldsymbol{\beta}$ = 250

Table 7.3: Prediction model experiment table 3 - Dropout rate.

ID	Dropout rate	Validation error	Median position/orientation errors (m/º)
Original article	-	-	0.15 / 6.12
10	0.4	4209	0.245 / 7.149
0	0.5	4489	**0.215 / 6.678**
11	0.6	4384	0.275 / 7.346

From the experiments, we found that the best dropout rate is 0.5.

7-scenes validation - Experiments results - β

Finally, we experimented with decreasing β, which decreases the orientation loss error. We expected this to reduce the position error, approximating our algorithm results to the original ones. We kept the following static parameters:

- **L1 penalty** $= 0$
- **Weight decay** $= 10^{-8}$
- **Dropout rate** $= 0.5$

ID	Beta	Validation error	Median position/orientation errors (m/º)
Original article	250	-	0.15 / 6.12
12	50	1091	0.233 / 7.187
13	100	2548	0.235 / 7.292
14	150	3372	0.291 / 7.568
15	200	3712	0.267 / 7.046
0	250	4489	**0.215 / 6.678**

7.2.2 7-scenes validation - Experiments results - final parameters

After all these experiments, we obtained the following parameter values:

- **L1 penalty** $= 0$
- **Weight decay** $= 10^{-8}$
- **Dropout rate** $= 0.5$
- $\boldsymbol{\beta} = 250$

Our final model obtains a median position/orientation error of 0.215m / 6.678º. Compared to the original article's results of 0.15m / 6.12º, we can observe that our model is significantly worse.

As stated before, there was no pre-available implementation of ContextualNet, and it was necessary to create our own according to the information provided in the original article. Therefore, the most likely reason for our model's lack of performance is the omission of some implementation detail. We did experiment with some parameters to find the best combination to overcome this issue, but our model still is not as good as the original one.

However, our model's performance (0.215m / 6.678º) is comparable to the original one (0.15m / 6.12º). Therefore, due to lack of time and more critical issues, we consider our model validated in the 7-Scenes data.

7.3 Mobile Application System

Our mobile app operates at the required speed, updating the path and user position every 3 seconds.

In the case of a mobile phone with a compass, it displays the orientation with a blue arrow. Otherwise, we remove the arrow.

In this next part, we will analyze the prediction model's performance and how it affects the whole user experience.

7.3.1 Prediction model validation - Experiment process

We collected video sequences on the same evaluation points defined for the mapping evaluation process to get a sense of the model's performance in real test environments. We associate the difference between the actual evaluation points and the predicted coordinates as the model's error in the test environment.

These image sequences represent real sequences of images collected from users' smartphones. Therefore, we consider the results of these images a measure of our system in a real environment.

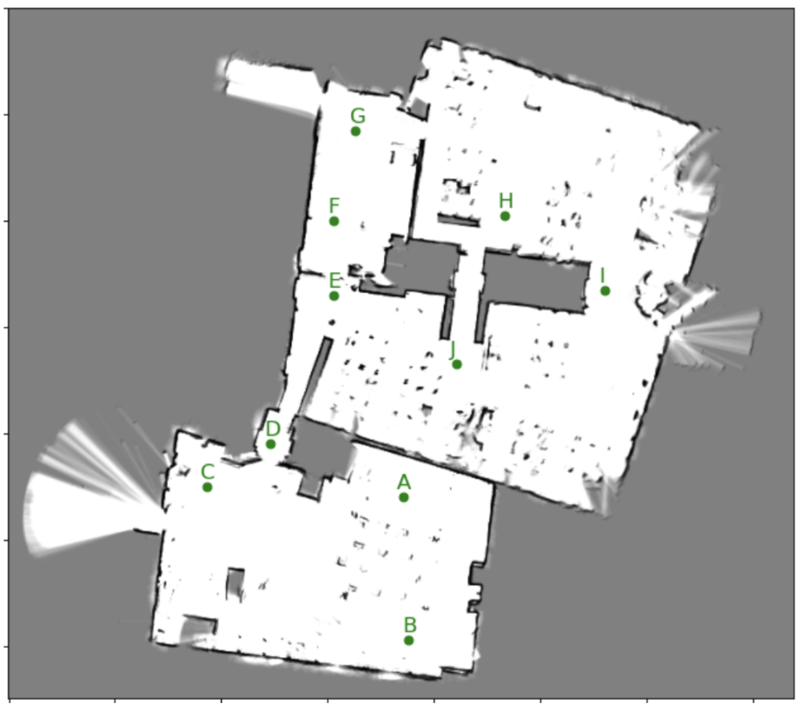


Figure 7.7: We used the same points defined in the mapping process because we know the actual coordinates. We recorded a video sequence that ended on the marked points and extracted images from these videos to evaluate the prediction.

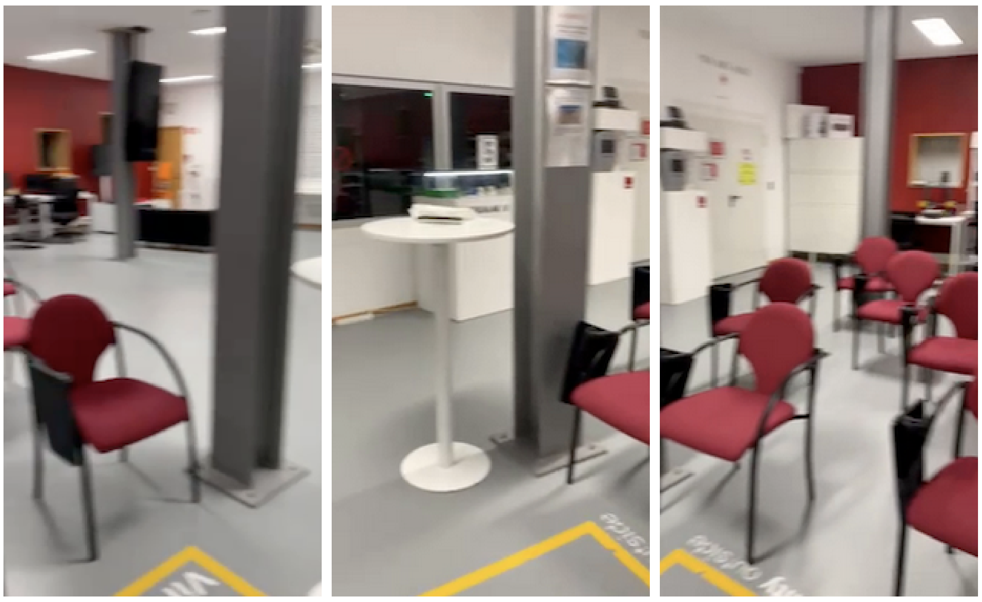


Figure 7.8: Captured images using a mobile phone camera for point A. Before inputting them into the model, they are resized and cropped into 320x240 dimensions (the exact dimensions utilized in the mobile application).

We must clarify that we did not use these evaluation points in the model training phase.

7.3.2 Prediction model validation - Model training process

As a result of our dataset collection phase, we generated four sequences of images (collected at 2Hz) associated with location coordinates (x,y):

- Train sequences (4978 images):
  - Sequence 0 - 1570 images
  - Sequence 1 - 1617 images
  - Sequence 2 - 1791 images
- Sequence test - 1611 images

Our robotic agent generated these pictures, and they are the only images utilized in our model training. The smartphone's photos are only used to validate our model.

We followed the same training procedure in the model validation on the 7-Scenes dataset in section 6.2. Additionally, due to lack of time and other more critical matters, we solely utilized the parameters obtained during the model's validation on the 7-Scenes dataset. While the images are different and additional experiments could improve the results, we observed throughout the experiments for the model's validation that the performance difference is not too big, and we disregard it to address other critical issues. However, future work might benefit from further experiments on the model tuning phase.

7.3.3 Prediction model validation - Experiment results

With the training/validation image sequences and the evaluation points image sequences, we obtain the following results:

Dataset	Average/variance of position errors (m)
Robotic agent's validation images	0.686 / 2.123
Evaluation points' image sequences	7.924 / 8.957

We can observe that we obtain very different results on the two image sets. More importantly, we get bad results on the evaluation points' image sequences - an **average positioning error of 7.924m and a variance of 8.957m**, which means we cannot be used our system in a real environment.

The considerable difference between the performance of the two image sets indicates that the likely cause for our system's poor performance is too much difference between our robotic agent's images and the evaluation points' image sequences. As a result, our model cannot learn from the robotic agent's images to predict image poses in a real environment.

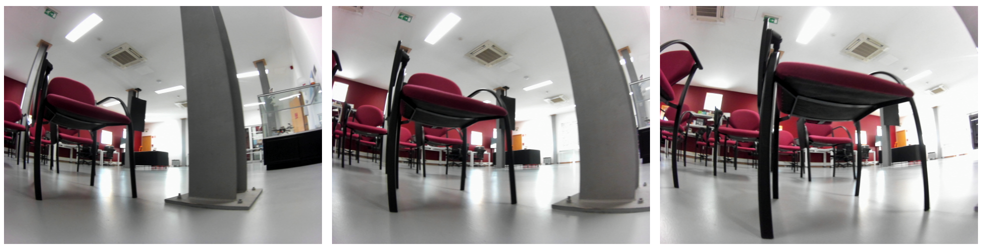


Figure 7.9: Captured images using the robotic agent.

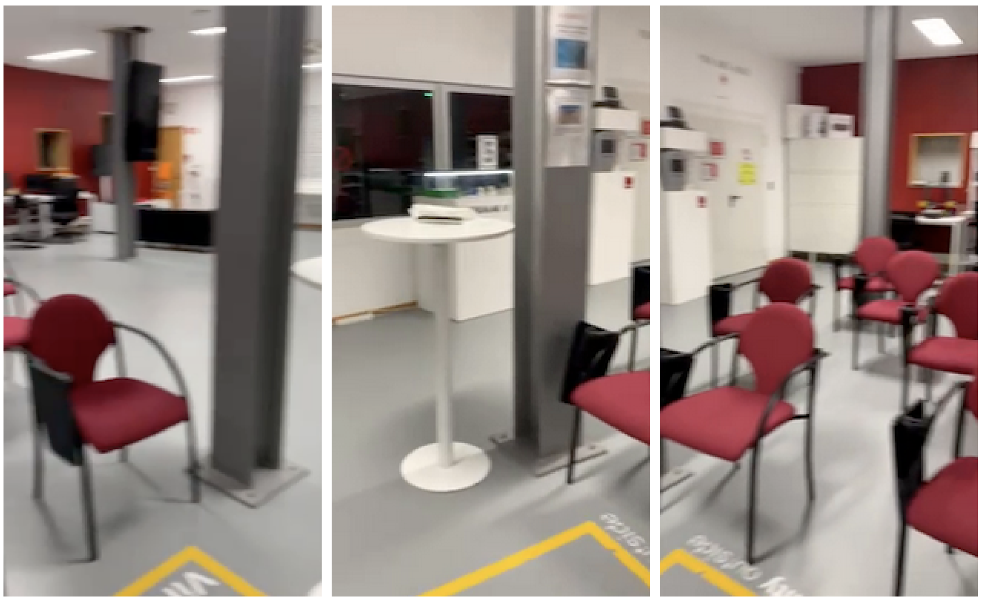


Figure 7.10: Captured images using a mobile phone camera (the images were cropped to be the same dimension as utilized on the mobile application).

As we can visualize in figures 7.9 and 7.10, the images present a number of differences:

- The robotic agent is very small; it has only around 20cm in height, much shorter than any human user would hold its smartphone.
- The smartphone's images are much blurry and present rotation.
- The robotic agent travels at a very stable speed and smooth rotation, while a user trajectory is much more erratic.
- The robotic agent's images present a distortion called Fish eye, which results in images becoming more curved around the edges. An example is shown in figure 7.11.

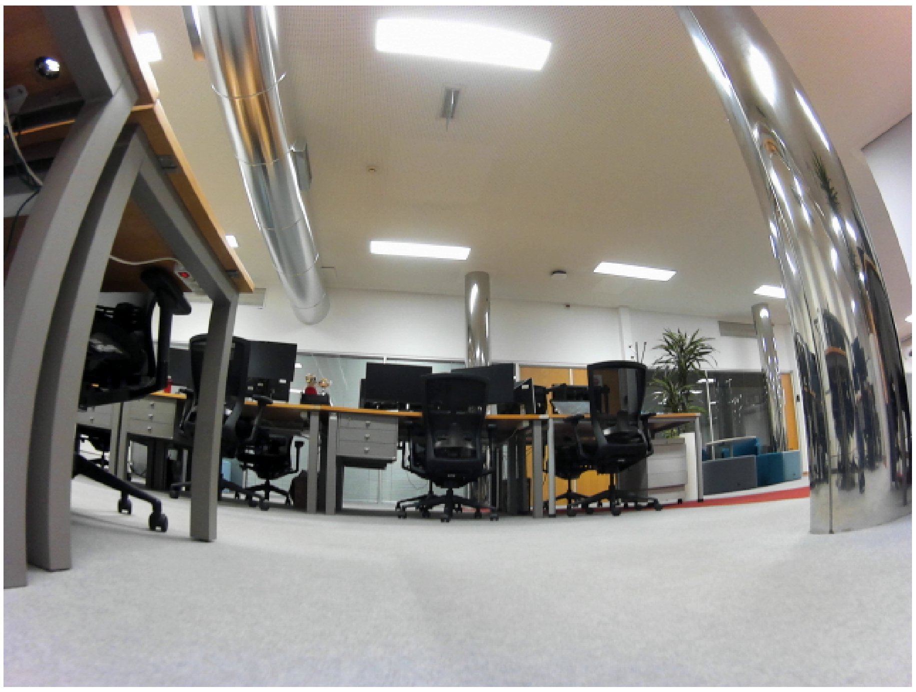


Figure 7.11: Example of Fish eye. The metal beam on the right is in reality straight.

These results indicate that the main focus of a future project is the improvement of the dataset collection phase, which includes the use of a prototype able to more accurately reproduce a user's smartphone point of view. Additional data augmentation techniques might also be applied, such as blurring and rotation, and correction of the Fish eye distortion.

7.3.4 Mobile application performance

Due to our prediction's model poor performance, the calculated user position has a significant margin of error. This ultimately leads to a path of inferior quality that does not help the user reach its destination.

Regarding using the smartphone's compass to obtain user orientation, practical tests have revealed that the compass fails to provide proper user orientation. The reason is that when we place the compass horizontally, it works well. However, the compass does not work well when we put it vertically, which is the normal position when utilizing the app.

Chapter 8

Conclusion

During this internship at Critical Software, the main goal was to design and implement an Indoor Positioning System using Computer Vision and Augmented Reality Techniques. Therefore, the objectives were:

1. Collect an image dataset using a robotic agent (based on JetRacer AI Kit [4]) and associate each image with x,y coordinates, and orientation angle.

2. Define a Deep Learning architecture and train an ML model that will use the images as input and output the user's pose.

3. Develop an augmented reality mobile application using this ML model to locate the user inside the building floor.

During the first semester, we studied the current state-of-the-art mapping algorithms and object detection and classification models. In addition, we studied past solutions for image-based indoor positioning systems. We also familiarized ourselves with the ROS framework and developed a prototype to map the environment using LIDAR data.

During the second semester, we collected a dataset of images associated with localization coordinates. Then, we used this dataset to train our prediction model. Furthermore, we also collected smartphone image sequences to validate our model and assess its performance for user navigation. Finally, we deployed our prediction model in our indoor positioning system.

This project had many challenges, and we had to work with different technologies for the project's steps. Some challenges include learning how to work with ROS (Robot Operating System) and the 3D printer for constructing the robotic agent, using the PyTorch framework to implement the prediction model, and using the Flutter programming to develop our mobile application. This diversity of challenges introduced me (the author, Eduardo Guerra) to many different technologies and made this project an exciting challenge.

This project's main goal was not achieved - the implemented mobile application does not perform well in a real test environment. Additionally, there was a failure to obtain the robotic agent's orientation angle, which led to adopting an alternative, less ideal solution.

Even though our project was not able to succeed in its final objective - the tests reveal the mobile application is unable to perform in a real scenario - at the end we were able to identify tasks to be developed in future work to improve on this work:

- Ground-truth trajectory - Due to the limitations imposed by Critical Software's office space, we could only mark a set of points as the ground truth. However, in future work and in a more appropriate environment, we would be able to define a trajectory (lines on the ground), which could more precisely define our dataset and prediction model's performance.
- More suitable robotic agent - Our robotic agent is very short (only around 20cm), compromising the machine learning model's training. In addition, we cannot ignore the difference in perspectives between the robotic agent's and a smartphone's point of view.
- A precise orientation retrieving sensor module - A failure of this project was not considering how complex it would be to extract the direction of a robotic agent on the move. A proper solution will have to be researched and implemented in future work.

We believe that identifying these issues provides helpful insight for future image-based indoor positioning systems and constitutes a main contribution of this work. This project was an enriching experience that increased my knowledge in various areas.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, Las Vegas, NV, USA, June 2016. IEEE.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, January 2016. Number: arXiv:1506.01497 arXiv:1506.01497 [cs].

[4] Waveshare. Bworld robot control software. `https://www.waveshare.com/wiki/JetRacer_AI_Kit`.

[5] Slamtec. Rplidar a2. `https://www.slamtec.com/en/lidar/a2`.

[6] Waveshare. Environment sensors module for jetson nano. `https://www.waveshare.com/environment-sensor-for-jetson-nano.htm`.

[7] Xuan Sang Le, Luc Fabresse, Noury Bouraqadi, and Guillaume Lozenguez. Evaluation of Out-of-the-Box ROS 2D SLAMs for Autonomous Exploration of Unknown Indoor Environments. In Zhiyong Chen, Alexandre Mendes, Yamin Yan, and Shifeng Chen, editors, Intelligent Robotics and Applications, volume 10985, pages 283–296. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[8] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. In 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, pages 155–160, Kyoto, Japan, November 2011. IEEE.

[9] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. IEEE Transactions on Robotics, 23(1):34–46, February 2007.

[10] google. Cartographer. `https://google-cartographer-ros.readthedocs.io/en/latest/`, 2022.

[11] Rauf Yagfarov, Mikhail Ivanou, and Ilya Afanasyev. Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth. In 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), pages 1979–1983, Singapore, November 2018. IEEE.

[12] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding, 110(3):346–359, June 2008.

[13] Mitesh Patel, Brendan Emery, and Yan-Ying Chen. ContextualNet: Exploiting Contextual Information Using LSTMs to Improve Image-Based Localization. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 1–7, Brisbane, QLD, May 2018. IEEE.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–9, Boston, MA, USA, June 2015. IEEE.

[15] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling Cross Stage Partial Network, February 2021. Number: arXiv:2011.08036 arXiv:2011.08036 [cs].

[16] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pages 1571–1580, Seattle, WA, USA, June 2020. IEEE.

[17] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, and Ping Luo. Sparse R-CNN: End-to-End Object Detection with Learnable Proposals. In 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 14449–14458, Nashville, TN, USA, June 2021. IEEE.

[18] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 936–944, Honolulu, HI, July 2017. IEEE.

[19] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10778–10787, Seattle, WA, USA, June 2020. IEEE.

[20] Chien-Yao Wang, I.-Hau Yeh, and Hong-Yuan Mark Liao. You Only Learn One Representation: Unified Network for Multiple Tasks, May 2021. Number: arXiv:2105.04206 arXiv:2105.04206 [cs].

[21] Qin Zou, Qin Sun, Long Chen, Bu Nie, and Qingquan Li. A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles. IEEE Transactions on Intelligent Transportation Systems, pages 1–15, 2021.

[22] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images. In 2013 IEEE Conference on Computer Vision and Pattern Recognition, pages 2930–2937, Portland, OR, USA, June 2013. IEEE.

[23] Ahmed M. Elmoogy, Xiaodai Dong, Tao Lu, Robert Westendorp, and Kishore Reddy Tarimala. SurfCNN: A Descriptor Accelerated Convolutional Neural Network for Image-Based Indoor Localization. IEEE Access, 8:59750–59759, 2020.

[24] Ahmed Elmoogy, Xiaodai Dong, Tao Lu, Robert Westendorp, and Kishore Reddy. SURF-LSTM: A Descriptor Enhanced Recurrent Neural Network For Indoor Localization. In 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), pages 1–5, Victoria, BC, Canada, November 2020. IEEE.

[25] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 2938–2946, Santiago, Chile, December 2015. IEEE.

[26] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers. Image-Based Localization Using LSTMs for Structured Feature Correlation. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 627–637, Venice, October 2017. IEEE.

[27] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3296–3297, Honolulu, HI, July 2017. IEEE.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, November 1997.

[29] R. T. H. Collis. Lidar. Appl. Opt., 9(8):1782–1788, August 1970. Publisher: Optica Publishing Group.

[30] ROS. Ros. `https://www.ros.org/`.

[31] Stefan Kohlbrecher. Hector slam. `http://wiki.ros.org/hector_slam`, 2022.

[32] Brian Gerkey. Gmapping. `http://wiki.ros.org/gmapping`, 2022.

[33] B. Yamauchi. A frontier-based approach for autonomous exploration. In Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation', pages 146–151, Monterey, CA, USA, 1997. IEEE Comput. Soc. Press.

[34] Xuexi Zhang, Jiajun Lai, Dongliang Xu, Huaijun Li, and Minyue Fu. 2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots. Journal of Advanced Transportation, 2020:1–14, November 2020.

[35] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1271–1278, 2016.

[36] X Fan, Y Wang, and Z Zhang. An evaluation of Lidar-based 2D SLAM techniques with an exploration mode. Journal of Physics: Conference Series, 1905(1):012021, May 2021.

[37] Boney Labinghisa and Dong Myung Lee. A Deep Learning based Scene Recognition Algorithm for Indoor Localization. In 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pages 167–170, Jeju Island, Korea (South), April 2021. IEEE.

[38] Shaopeng Liu and Guohui Tian. An Indoor Scene Classification Method for Service Robot Based on CNN Feature. Journal of Robotics, 2019:1–12, April 2019.

[39] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection, April 2020. Number: arXiv:2004.10934 arXiv:2004.10934 [cs, eess].

[40] M.J. Caruso. Applications of magnetic sensors for low cost compass systems. In IEEE 2000. Position Location and Navigation Symposium (Cat. No.00CH37062), pages 177–184, San Diego, CA, USA, 2000. IEEE.

[41] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. Advances in Neural Information Processing Systems (NIPS) 27, page 487–495, 2014.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. Number: arXiv:1412.6980 arXiv:1412.6980 [cs].

[43] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, July 1968.

[44] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection, September 2016. Number: arXiv:1608.08021 arXiv:1608.08021 [cs].