FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Západočeská Univerzita v Plzni
Fakulta Aplikovaných Věd

Katedra Kybernetiky

Přístup ke snížení výpočetní náročnosti systémů rozpoznávajících prostředí mobilního robota

DISERTAČNÍ PRÁCE

k získání akademického titulu doktor
v oboru **Kybernetika**

Ing. Petr Neduchal

Školitel: Doc. Ing. Miloš Železný, Ph.D.

Plzeň, 2021

FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Faculty of Applied Sciences

Department of Cybernetics

Approach for reducing the computational cost of environment classification systems for mobile robots

DOCTORAL THESIS

submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy in the field of
Cybernetics

Ing. Petr Neduchal

Advisor: Doc. Ing. Miloš Železný, Ph.D.

Plzeň, 2021

Acknowledgments

I would like to thank to all people who supported me in my studies. First of all, I want to thank my adviso Doc. Ing. Miloš Železný Ph.D. Second special thanks go to my colleague Miroslav Flídr Ph.D. for his valuable advice on my study topic. The third thank goes to all the colleagues in our department for the support and encouragement they provided.

Finally, I would thank my wife, my daughter, my family, and my friends for their psychological support and for all that moments when I could forget about the worries associated with studying. These moments helped me relax and increased my morale.

Abstrakt

Disertační práce se věnuje problému změny prostředí v úlohách mobilní robotiky. Zaměřuje se na využití jednodimenzionálních nevizuálních senzorů za účelem redukce výpočetních nároků. V práci je představen nový systém pro detekci a klasifikaci prostředí robota založený na datech z kamery a z nevizuálních senzorů. Nevizuální senzory zde slouží jako prostředek detekce probíhající změny, která iniciuje klasifikaci prostředí pomocí kamerových dat. To může významně snížit výpočetní nároky v porovnání se situací, kdy je zpracováván každý a nebo každý n-tý snímek obrazu. Systém je otestován na případu změny prostředí mezi vnitřním a venkovním prostředím.

Přínosy této práce jsou následující: (1) Představení systému pro detekci a klasifikaci prostředí mobilního robota; (2) Analýzu state-of-the-art v oblasti Simultánní Lokalizace a Mapování za účelem zjištění otevřených problémů, které je potřeba řešit; (3) Analýza nevizuálních senzorů vzhledem k jejich vhodnosti pro danou úlohu. (4) Analýza existujících metod pro detekci změny ve 2D signálu a představení dvou jednoduchých přístupů k tomuto problému; (5) Analýza state-of-the art v oblasti klasifikace prostředí se zaměřením na klasifikaci vnitřního a venkovního prostředí; (6) Experiment porovnávající metody studované v předchozím bodu. Jedná se dle mých znalostí o nejrozsáhlejší porovnání těchto metod na jednom jediném datasetu. Navíc jsou do experimentu zahrnuty také klasifikátory založené na neuronových sítích, které dosahují lepších výsledků než klasické přístupy; (7) Vytvoření datasetu pro testování navrženého systému na sestaveném 6-ti kolovém mobilním robotu. Podle mých znalostí do této doby neexistoval dataset, který by kromě dat potřebných k řešení úlohy SLAM, naíc přidával data umožňující detekci a klasifikaci prostředí i pomocí nevizuálních dat; (8) Implementace představného systému jako open-source balík pro Robot Operating System na platformě GitHub; (9) Implementace knihovny pro výpočet globálního popisovače Centrist v C++, taktéž dostupná jako open-source na platformě GitHub.

Klíčová slova

Detekce prostředí klasifikace prostředí, mobilní robotika, simultánní lokalizace a mapování, senzory, strojové učení

Abstract

This dissertation thesis deals with the problem of environment changes in the tasks of mobile robotics. In particular, it focuses on using of one-dimensional non-visual sensors in order to reduce computation cost. The work presents a new system for detection and classification of the robot environment based on data from the camera and non-visual sensors. Non-visual sensors serve as detectors of ongoing change of the environment that initiates the classification of the environment using camera data. This can significantly reduce computational demands compared to a situation where every or every n-th frame of an image is processed. The system is evaluated on the case of a change of environment between indoor and outdoor environment.

The contributions of this work are the following: (1) Proposed system for detection and classification of the environment of mobile robot; (2) State-of-the-art analysis in the field of Simultaneous Localization and Mapping in order to identify existing open issues that need to be addressed; (3) Analysis of non-visual sensors with respect to their suitability for solving change detection problem. (4) Analysis of existing methods for detecting changes in 2D signal and introduction of two simple approaches to this problem; (5) State-of-the-art analysis in the field of environment classification with a focus on the classification of indoor vs. outdoor environments; (6) Experiment comparing the methods studied in the previous point. To my best knowledge, this is the most extensive comparison of these methods on a single dataset. In addition, classifiers based on neural networks, which achieve better results than classical approaches, are also included in the experiment. (7) Creation of a dataset for testing the designed system on an assembled 6-wheel mobile robot. To the best of my knowledge, there has been no dataset that, in addition to the data needed to solve the SLAM task, adds data that allows the environment to be detected and classified using non-visual data. (8) Implementation of the proposed system as an open-source package for the Robot Operating System on the GitHub platform. (9) Implementation of a library for calculating the Centrist global descriptor in C++ and Python. Library is also available as open-source on the GitHub platform.

Keywords

Environment detection, Environment Classification, Mobile robotics, Simultaneous Localization and Mapping Sensors, Machine learning

Declaration

Hereby I declare that I compiled this Ph.D. thesis independently, using only the listed literature and resources.

In Pilsen,

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artifitial Neural Networks
AR	Augmented reality
BCPD	Bayesian Changepoint Detection
CNN	Convolutional Neural Networks
CPD	Changepoint Detection
CUSUM	Cumulative Sum
DNN	Deep Neural Networks
DOF	Degrees of Freedom
ECD	Environment Change Detection (module)
ECS	Environment Classification System
EDS	Environment Detection System
EKF	Extended Kalman Filter
GBP	Global Binary Patterns
GMM	Gaussian Mixture Model
GNSS	Global Navigation Satelite System
GPS	Global Positioning System
HMI	Human-machine interface
HSV	Hue Saturation Value
IBEC	Image Based Classification (module)
ICP	Iterative Closest Point
JCBB	Joint Compatibility Branch and Bound
LBP	Local Binary Patterns
LiDAR	Light Detection And Ranging
LSM	Least Square Method

MAP	Maximal a Posteriori
MRPT	Mobile Robot Programming Toolkit
MSAR	Multi-resolution, Simultaneous Autoregressive Model
NN	Neural Networks
PF	Particle Filter
RANSAC	Random Sample Consensus
RBA	Robot Behaviour Adaptation (module)
RBPF	Rao-Blackwellized Particle Filter
RGB	Red Green Blue Color Model
RGB	Red Green Blue
RGBD	Red Green Blue Depth
ROS	Robot Operating System
SAR	Search and Rescue
SEIF	Sparse Extended Information Filter
SLAM	Simultaneous Localization And Mapping
SVM	Support Vector Machine
UKF	Unscented Kalman Filter
USB	Universal Serial Bus

Chapter 1

Introduction

We are living in the computer age. Charles Babbage (1791 – 1871) dreamed of this age when he and Countess Ada Lovelace discussed the possibility of creating the first computing machine. He took the first step towards it by inventing what he called a Difference Engine [8] in 1822. Since then, computers have evolved and become computationally capable of solving many problems of the modern world.

Moreover, robotics applications play an essential role in our society's life, from robotics manipulators in the industry up to autonomous robotic vacuum cleaners, vehicles, or drones. The part of robotics and especially mobile robotics become significant nowadays.

Mobile robotics applications often depend on the ability of the autonomous movement of the robot in its environment. This ability is enabled by navigation in the map built by Simultaneous Localization And Mapping (SLAM) algorithms. These algorithms are actively addressed by scientists in the field for the last two decades. Consequently, many approaches based on various sensors attached to a mobile robot were proposed [9], [10].

As will be noted in Chapters 2.9 and 3, many published papers and implementations assume that the environment is static [5], [11] [12]. They are somehow robust against the occurrence of dynamic objects, but they not handle them on purpose. It is an appropriate restriction of the problem for single environment applications that – almost – does not contain dynamic objects. Unfortunately, it is also usually a too restrictive assumption. Our world is not static, and it includes various environment types. Indoor and Outdoor environments can be mentioned as significantly different ones.

Basic problem deals with the static environment. It is usually solved by semantic segmentation with deep neural networks[13], i.e., processing an image to compute its segmentation into clusters representing image objects. Objects are then tracked in a sequence of frames. Moving objects are finally ignored by the SLAM algorithm. Another problem is dealing with the transition between environments to adapt the robot behavior to the most suitable for the current environment. The analysis and a proposed solution to this particular problem is the main topic of this thesis.

The robot behaviour can be defined similarly to behaviour of the person. German-American psychologist Kurt Levin proposed [14] that behavior B is the function of the person P in its environment E

$$B = f(P, E). \tag{1.1}$$

Similarly, the behavior B of the robot can be defined as a function of robot R on a specific mission M in its environment E

$$B = f(R, M, E). \tag{1.2}$$

In other words, the robot should adapt its behavior based on its properties (shape, sensors, actuators), its mission (can be understood as a robot task) and its environment. The properties of the robot are usually unchanging during a mission. Similarly, the mission is defined. Thus, both variables, R, and M are known in advance and can be assumed as static. They do not have to be static in practice, but their configuration, at least for particular states, should be known in advance.

The last variable – the environment – is more complicated than the previous ones. It is not possible to know all information about the robot environment in advance. Thus, it is appropriate to analyze the environment – by detection and classification – in order to adapt robot behavior. The choice of the right robot behavior can improve the robustness and other properties of the algorithms used in mobile robotics. The rest of this chapter is devoted to a description of the cornerstones of this thesis. The first one is localization and mapping, the second one is robot perception, and the third one is environment detection and classification. Finally, the last section of this chapter describes the outline of this thesis.

1.1 Simultaneous Localization and mapping

SLAM [2] is a fundamental problem of current robotics. In SLAM, a mobile robot equipped with one or multiple sensors is used to observe its environment. The robot's goal is to create a map of the environment and localize itself in the created map simultaneously. Both actions are performed in real-time, which means that the estimate is recalculated when the new observation is made. The perfect solution to the SLAM problem would be a holy grail of the field of robotics [15]. Unfortunately, there are still many crucial issues to solve [9] as discussed in Chapter 2.9.

A localization task can be performed using the Global Navigation Satelite System (GNSS) such as Global Positioning System (GPS)[1], GLONASS[2], or Galileo[3]. But it can be used only in outdoor environments with a good signal from GNSS satellites. Moreover, there are many GNSS-denied environments. It is impossible to use GNSS indoor, in mines [16], underwater [17], or on Mars. It is possible to create a positioning system for a particular environment based on Wi-Fi hotspots [18] or beacons placed in known locations. A disadvantage of this solution is usually a lack of accuracy, which can be crucial for mobile robotics applications.

There are many scenarios where the SLAM is the best or the only suitable solution. Exploration of a tunnel [19] or mine is one of them. In the mine, a mobile robot can map an environment based on the data from Light Detection And Ranging (LiDAR) sensor. However, it is usually impossible to establish a connection with the GNSS satellites. The solution

[1]GNSS operated by USA
[2]GNSS operated by Roscosmos (Russian Federation)
[3]GNSS operated by the European GNSS Agency

of SLAM can be also used as a part of the augmented reality (AR) [20] task. The AR system [5] [21] has to know the robot's position and heading within the environment. The best way here is to combine visual information with sensors like a compass, accelerometers, or gyroscopes. In these scenarios, GNSS can register the map created using the SLAM system with the global map – e.g., robot heading on a bookstore in a particular part of the city. The place's observation showing the bookstore logo can be used as a modern type of QR code. Other examples of scenarios for SLAM algorithms are autonomous agriculture operations [22], warehouse management systems [23] Search and Rescue (SAR) operations [24] in hardly accessible environments, people monitoring systems during festivals or demonstrations. The last example of SLAM suitable task is mapping a human body in medicine [25]. It can be used for visualization, which should help the surgeon during an operation.

The base of all mentioned applications is robot perception or perception of the environment in general. It is easy for the human brain to perceive the environment and localize itself within it. Unfortunately, it is more tricky to obtain similar results in the case of a mobile robot.

1.2 Robot Perception

The process of sensing the environment by a mobile robot is called robot perception [2]. Information about the environment is obtained by different kinds of sensors attached to the robot. Human senses can be understood as a group of sensors too. But, there are significant differences between the human and the robot. The robot makes a measurement and processes it as the batch of local points with a defined neighborhood instead of all data's global processing. The second difference is a knowledge base about the environment. The brain of an adult human is full of knowledge based on his experiences. He can recognize the state of his environment. For example, deciding whether a door is open or closed is a simple task for the human. The robot can be successful, too, but there will always be some amount of uncertainty [26]. Therefore, it is necessary to define the problem in a probabilistic way. The model should take into account all types of uncertainty.

Uncertainty in robotics arises from multiple sources:

- **Sensors:** The accuracy of sensors is limited by their resolution, range, and noise in the data. The second limitation is based on physical laws. For example, a thermal imaging camera can't see through walls as presented in Hollywood movies.
- **Environment:** The stochastic character of the world makes perception unpredictable. Moreover, the uncertainty in structure, size, and dynamic of the environment are the main issues of mobile robotics. For example, the task of moving a robot among empty rooms in the building is much easier than moving among the streets of a big city full of people..
- **Actuators:** Actuators of the mobile robots aren't deterministic too. Every actuator has slightly different characteristics, and the same input can or does not have to create the same output. It is related to the price of the actuators. Usually, a cheaper actuator is more inaccurate than the expensive one. The combination with the environment can cause even more uncertainty. For example, the robot wheel can slip on the terrain,

and the resulting motion is different from the one computed by a robot's mathematical model.

- **Models:** Mathematical models of the robot, its sensors, or the environment are always an abstraction of the real world. As the abstraction, the model will never be accurate. Higher-orders models are not usually used in real-time applications, which causes inaccuracy and increases uncertainty.

- **Real-time computation:** All mobile robots have to work with real-time data. It means that the computation has to be done in the short time between two consecutive measurements. It is impossible to handle the data the same way as in off-line computing. Only one or a few last measurements are used instead of a big batch of data. Often, there has to be some sort of approximation to speed up the whole computation process.

- **Size of the data and refresh rate:** Real-time computation capabilities are depended on the size of the data and the refresh rate of sensors. It is relatively easy to do real-time processing of a variable from some sensor that measures one value and has a refresh rate of 10 Hz. The opposite extreme is data from a high definition camera. Processing data from a full HD or 4K camera can be tricky even for a high-performance computer.

As mentioned at the beginning of this section, measurements from sensors contain data from certain places of the environment. The consequence is that the robot has only local information instead of a global view of its environment. The robot has a batch of the data, and it can decide whether it sees is a wall or free space. On the other hand, it is tough to make robots recognize objects in the data and connections between them. Nowadays, the research in deep neural networks [27] made progress in understanding the observed scene.

On the other hand, the SLAM problem's solution is still an essential step to understanding the global scene. Besides scene understanding, it is crucial to detect and classify the environment of the robot. The next section is focused on describing the basic information about this particular problem.

1.3 Environment Detection And Classification

The environment is a complex system containing much information that can be perceived by the robot. This information can be used to change robots' behavior or even increase the robustness of robots' algorithm. The solution to the problem consists of two parts. The first part is obtaining information from the environment and detect changes in (of) the environment. The second part is the classification of the environment into defined classes.

Information about the state of the environment can be obtained using an arbitrary suitable sensor. Usually, the sensor with the most considerable information value for the change to be detected should be used. For example, a light intensity sensor can be used to detect light changes in the environment. Another example can be the temperature and humidity sensor to detect transitions between indoor and outdoor environments. These examples are based on the physical properties of the environment.

Another possibility of choosing a sensor is to use some knowledge about the environment. For example, the ultrasonic distance sensor pointing to the room ceiling can be used to measure its ceiling height. Thus, it can detect the transition between indoor and outdoor environments, the transition between room and hallway, or even the transition between two rooms.

Another difference between the mentioned sensors is that some sensors – e.g., temperature sensors – have a delay of their measured value concerning the real value of this particular variable in the environment.

There is also a third possibility. A camera or a depth camera can be used to detect some properties of the environment. As shown in Chapter 4, the use of the camera as the only sensor for environment detection and classification can be computationally expensive for the systems that should run real-time.

In this thesis, I focus on the detection of changes from sensors with 1D signal output. The detection is based on searching for abruptions in a 1D signal using suitable methods. These methods are described in Chapter 4.1. Moreover, results from the experiments are discussed in Chapter 6.1. Unfortunately, these kinds of sensors are usually not sufficient to decide on the real change of the environment. In other words, it can detect that there is a change, but it can not determine whether this change is significant for the robot – i.e., whether the transition happened or not. Fortunately, there is a possibility of finding out how significant the change is by classifying the environment using camera information.

It can be a little confusing now because a few paragraphs above, I said that camera data for detecting changes in the environment is computationally expensive. The use of a camera for the classification of the environment can also be computationally expensive. On the other hand, as will be discussed in Chapters 4.2, 6.3, and 6.4, it is unnecessary to perform classification at each step. Thus, it is suitable to use it in the mobile robot system in real-time.

1.4 Thesis Outline

The goal of this thesis is to analyze the problem of transition between different environments during robot missions. The main contribution is the design and implementation of the Environment Classification System based on visual and non-visual sensors.

The thesis is structured as follows. In Chapter 2, the SLAM is analyzed to discover open problems that should be addressed. One of the issues is the transition between environments during robot missions. Based on this finding, the dissertation problem definition and goals are described in Chapter 3. In Chapter 4, the theory necessary for solving a defined problem is summarized. The main contribution of this thesis is contained in Chapters 5 and 6. The first one is focused on the design and description of the proposed Multi-Environment Robot System (MERoS). The second one is focused on performed experiments. In the last chapter, thesis contributions are discussed, and where the future work is defined.

Chapter 2

Simultaneous Localization And Mapping

The history of the SLAM problem started in 1986 at IEEE Robotics And Automation conference in San Francisco. Scientists such as Peter Cheeseman [26] or Hugh Durrant-Whyte [15] [28] wanted to apply probabilistic methods and estimation theory to localization and mapping tasks. Many discussions were made about the possibilities of consistent mapping and accurate localization of robot pose. They found out that SLAM is a fundamental problem in mobile robotics. As a completely new research area, SLAM had many open issues. Some of them are solved, but as described in Chapter 2.9, there are still many topics to address in modern SLAM solutions. Before defining the SLAM problem, let's look closer at its principle and particular algorithmic tasks that must be considered during localization and mapping.

2.1 Introduction to SLAM

The diagram in Figure 2.1 shows the three time-steps of the SLAM algorithm. The mobile robot observes landmarks in the environment and estimates their positions and the position of itself. There is shown a difference between dark grey and light grey positions. It is called drift, and its minimization is the goal of the SLAM problem. Without SLAM algorithm, the drift accumulates, and the resulting map becomes inconsistent with error diverging to infinity. Drift minimization by SLAM solving algorithm reduces mapping error and guarantees consistency of the map.

Every SLAM problem solution comprises a core algorithm and a set of essential tasks such as information extraction [29][30], data association, or loop closing. These are tasks that are necessary for obtaining an accurate and consistent result.

The core algorithm is a crucial part of the SLAM solving system. There are two possible groups of core algorithms. The first one is a group of filter-based solutions – e.g., based on Extended Kalman Filter [31][2]. The second group is based on a non-linear least-square optimization algorithm, a core of the graph-based SLAM [32] approach.

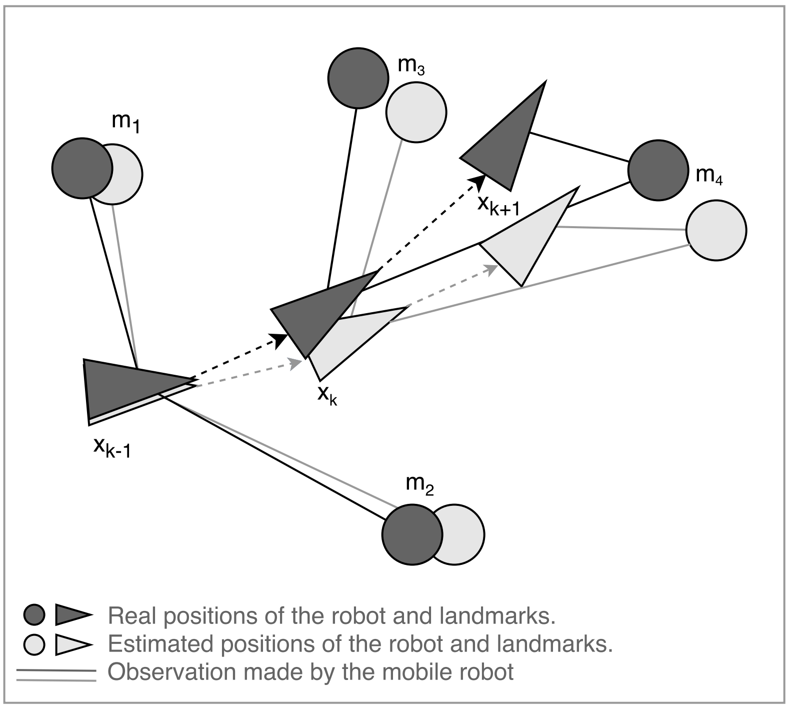
m1
m3
m4
m2
xk+1
xk
xk-1
Real positions of the robot and landmarks.
Estimated positions of the robot and landmarks.
Observation made by the mobile robot

Figure 2.1: SLAM principle

2.1.1 Information Extraction

The world is full of objects with different shapes, colors, and textures [33]. These properties are useful for information extraction algorithms that detect a specific property type – such as shape – and measure it. The result is a piece of information that describes the object. In the ideal world, every object is unique and distinguishable. It is not true in the real world because:

- There exist different objects that looks similar.
- Particular object can look different from various angles.

Thus the SLAM solving system has to deal with the problem of extracting information about objects. It can be different for various types of sensors. There exist two approaches based on the type of sensors. The first one is a raw data approach [11], which uses all data from the measurement. The second one is a feature-based approach [34]. Features are assumed as unique or sufficiently distinguishable parts of the data. This approach is based on feature extraction algorithms [35][36]. These algorithms extract information about the feature from raw data and then computes the so-called feature vector. The vector describes the feature, and it is supposed to be as stable as possible. It means that the description of a particular feature should be similar regardless of observation conditions. These vectors can be compared to each other by some comparing technique. One of the well-known basic ways for comparing two feature vectors is the Euclidean distance.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}, \tag{2.1}$$

where **x** and **y** are vectors of the length N. The distance between two equal feature vectors is 0. It is critical to decide whether detected features in two consecutive measurements represent one feature or two different features.

A feature successfully observed in the consecutive measurements is added to the map as a unique landmark. The algorithm searches for these landmarks when the particular place is revisited.

2.1.2 Data Association

A system for solving the SLAM problem has information about all landmarks discovered in the environment. A data association task aims to determine whether a landmark observed in current data is a new one or if it is already known.

A new landmark is saved in the first case, and spatial constraints – to the robot and other landmarks – are computed. In revisiting known locations with known landmarks, the data association algorithm has to pick the right landmark to pair with the observed one. Observation is then used to refine information about landmarks, which leads to a more accurate map of the environment.

There are two types of revisiting the place with known landmarks. In the first case, the robot observes the same landmarks in two consecutive time steps. It is a more manageable situation because the whole scene is almost the same. The Nearest neighbor algorithm can be used in the case of slow movement w.r.t the sensor measurement rate. Unfortunately, it is a too strict condition. The Joint Compatibility Branch and Bound (JCBB) [37] or some algorithm from the Random Sample Consensus (RANSAC) family [38], [39] can obtain a better solution. The same method can be used in a more complicated case of revisiting called loop closing.

A slightly different approach is performed in the least-squares based SLAM, where the scan-matching is used. The scan-matching algorithm tries to align two scans or a scan to the map. Mathematically, it maximizes the likelihood of the robot's current state and the map relative to the previous state. There are more approaches to scan matching. For example, the Iterative Closest Point [40] or the RANSAC algorithm can be mentioned. The scan-matching algorithm is not sufficient to obtain a correct map in the least-squares based SLAM, and the optimization has to be performed.

2.1.3 Loop Closure

Loop closing is a situation when the robot revisits a particular location of the environment. For example, a robot in a large building goes one direction from the main hall, and several minutes later, it comes to the same hall from the other side. Data association algorithm should recognize this situation and compare new landmarks with all other saved landmarks. The general algorithm for loop closing is still an open problem of SLAM[9].

2.1.4 Motion Model of the Vehicle

The world is non-linear. The consequence is that the system for solving the SLAM problem has to handle the real world using probabilistic models [2]. The mobile robot is defined by its motion model **F**. The model is an approximation of the robot's motion. It is used to predict the robot's location to the next time step $k+1$ before the measurement and map update is computed. The model usually describes the linear and angular motion of the robot. Motion models are based on the type of mobile robot – i.e., One motion model is used for a ground vehicle and another for aerial vehicles.

2.1.5 Map generation

All tasks mentioned in previous paragraphs lead to the map generation. Different core algorithms and used sensors are suitable for different types of maps. In Figure 2.2, three types of maps are shown. The first type, (Figure 2.2 a)) is an occupancy grid map typically used in LiDAR-based 2D systems for solving the SLAM problem. It is a map composed of cells containing real numbers between 0 (free space) and 1 (obstacle). The value represents the probability of the obstacle in the cell. In the example, white color represents free space, and black represents obstacles. The grey color is used for unknown cell values. The blue curve is the trajectory of the robot.

The second type of map is shown in Figure 2.2 b). It is a sparse map based on individual landmarks. It is usually a result of visual SLAM systems based on the visual feature extraction methods [35][36][30]. Last type Figure (2.2 c)) is a graph-like map composed of robot path with nodes and edges. Nodes represent robot poses, and edges represent non-linear spatial constraints between two poses.

Every type of map is suitable for combining the core algorithm, used sensors, and degrees of freedom. Sometimes the combination of mentioned maps is created. The most used type of map in state-of-the-art 2D SLAM systems is the grid map – e.g., Hector SLAM [41]. For 3D sparse visual SLAM, it is usually a combination of the graph and landmark-based map – e.g., ORB-SLAM 1 [12] and ORB-SLAM 2 [34]. Finally, the point cloud can be used for dense visual SLAM such as RTAB-Map[42].

In the following sections, two definitions of the SLAM problem will be described. The first one is a feature-based definition of the SLAM problem based on the papers of Hugh Durrant-Whyte and Tim Bailey [28], [15]. Nowadays, it is usually called the classic definition. The second definition of the SLAM problem based on the book of Sebastian Thrun et al. [2] and the paper of Cadena et al. [9] is called graph-based SLAM or optimization-based SLAM. The solution of optimization-based SLAM leads to solving the non-linear least-squares problem.

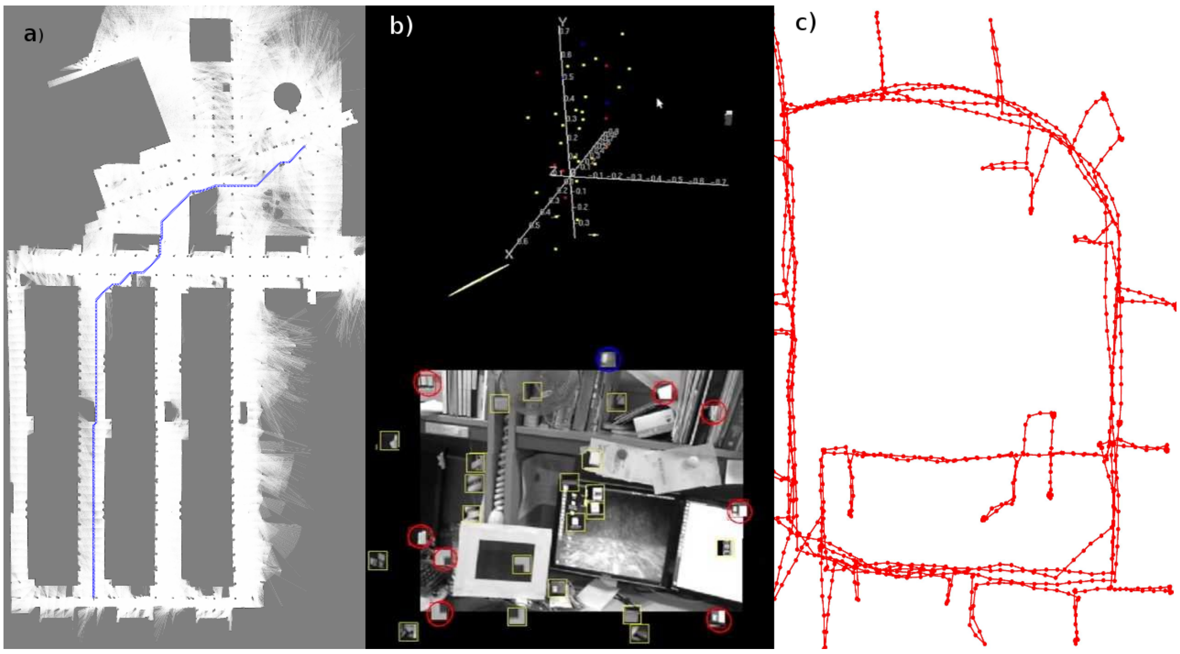

Figure 2.2: Maps generated by SLAM solving algorithms.

2.2 The classic definition of SLAM

The SLAM problem at discrete time step k is defined as searching for the probability density function

$$p\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right). \tag{2.2}$$

It is the joint posterior density function of a robot location $\mathbf{x}_k$ and the map $\mathbf{m}$. The initial pose of the robot $\mathbf{x}_0$, a set of all observation $\mathbf{Z}_{0:k}$, and all control inputs $\mathbf{U}_{0:k}$ – from time step 0 to time step k – are given at the calculation time. Let's assume that the density function $p\left(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0\right)$ at a time $k-1$ is known. The density function (2.2) is computed using a two-step recursive algorithm based on Bayes Theorem.

2.2.1 Two-step recursive algorithm

The solution for SLAM is implemented in two steps. The first step is called a time-update or a prediction step

$$\begin{aligned} p\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) = & \int p\left(\mathbf{x}_k, \mid \mathbf{x}_{k-1}, \mathbf{u}_k\right) \\ & \times p\left(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0\right) dx_{k-1} \end{aligned}. \tag{2.3}$$

The second step is a measurement update or a correction step (2.4).

$$p\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) = \frac{p\left(\mathbf{z}_k, \mid \mathbf{x}_k, \mathbf{m}\right) p\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0\right)}{p\left(\mathbf{z}_k \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}\right)}. \tag{2.4}$$

A model of the robot motion

$$p\left(\mathbf{x}_k, \mid \mathbf{x}_{k-1}, \mathbf{u}_k\right) \tag{2.5}$$

is a part of the equation (2.3). It models properties and the uncertainty in the robot's movement. Another name for the motion model is the state transition model because it models the transition of robot state between time step $k-1$ and k based on a control input $\mathbf{u}_k$. State $\mathbf{x}_k$ depends only on the previous state and the control input.

Similarly, an observation model

$$p\left(\mathbf{z}_k, \mid \mathbf{x}_k, \mathbf{m}\right) \tag{2.6}$$

is a part of the measurement update equation (2.4). The model describes observation $\mathbf{z}_k$ in time step k based on the robot's state and the map of the environment.

Dependence of the observation model on the robot pose and location of all landmarks is the reason why the joint posterior (2.2) can't be partitioned in Equation (2.7). An inconsistent estimation would be the result of such a partitioning step.

$$p\left(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) \neq p\left(\mathbf{x}_k \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) p\left(\mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) \tag{2.7}$$

Observations are the source of the error between estimated and true landmark locations. Fortunately, the relative location between two landmarks, i and j, may be estimated with high accuracy. It is possible because the relative position of the landmarks is changing slowly due to the robot movement. The error between the landmarks is correlated. Even the robot pose itself is correlated to the map.

The correlation is increased with every observation made. In practice, this implies that error in relative location decreases every time a new observation is made. As an example of the correlations between landmarks, the visualization is presented in Figure 2.3. It is an analogy to a network of connected springs. All landmarks are connected, and the strength of the connection is determined by its width – spring of specific stiffness. Consequently, an update of the landmark j observed at time step $k+1$ is propagated back to update non-observed landmark i.

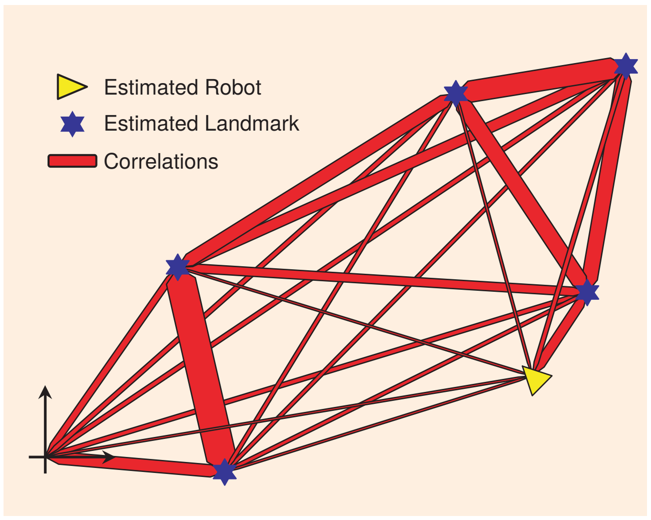

Figure 2.3: Spring analogy of SLAM problem. Taken from [1]

2.3 Approaches based on the classic definition

Solution based on the classic definition of the SLAM problem is usually based on an estimation filter. The first popular solution was the Extended Kalman Filter SLAM (EKF-SLAM)[2]. It is a solution that is easy to implement but with several disadvantages. Newer solutions are Unscented Kalman filter SLAM (UKF-SLAM), Sparse Extended Information Filter SLAM (SEIF-SLAM), and Particle filter based SLAM[3]. All the mentioned approaches are described in the next sections.

2.3.1 EKF-SLAM

EKF-SLAM is the basic filter-based approach to solve the SLAM problem. The core is the Extended Kalman Filter (EKF) algorithm [2]. There is an assumption that all probability density functions have Gaussian distributions. The consequence is that instead of estimating the whole distribution, it is possible to estimate the mean and covariance of that distribution only. The motion model (2.5) and the observation model (2.6) can be rewritten in a non-linear function with additive Gaussian noise. The robot motion model is described by the equation

$$\mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k, \tag{2.8}$$

where $f()$ represents the robot kinematics model, and $\mathbf{w}_k$ is the additive, zero mean, and uncorrelated Gaussian noise with the covariance matrix $\mathbf{Q}_k$. The observation model has the form of the equation

$$\mathbf{z}_k = h\left(\mathbf{x}_k, \mathbf{m}\right) + \mathbf{v}_k, \tag{2.9}$$

where $h()$ represents the robot observation model, and $\mathbf{v}_k$ is the additive, zero mean, and uncorrelated Gaussian noise with covariance matrix $\mathbf{R}_k$. Non-linear functions f and h are linearized using Taylor expansion [2]. The mean and the covariance of joint posterior probability density (2.2) can be computed using the two-step EKF algorithm.

I) Time-update step

Calculation of the mean vector

$$\hat{\mathbf{x}}_{k|k-1} = f\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right), \tag{2.10}$$

and the covariance matrix

$$\mathbf{P}_{k|k-1} = \begin{bmatrix} \mathbf{P}_{rr,k|k-1} & \mathbf{P}_{rm,k|k-1} \\ \mathbf{P}_{rm,k|k-1} & \mathbf{P}_{m,k|k-1} \end{bmatrix} = \begin{bmatrix} \nabla f \mathbf{P}_{rr,k-1|k-1} \nabla f^\top + \mathbf{Q}_k & \mathbf{P}_{rm,k-1|k-1} \\ \mathbf{P}_{rm,k-1|k-1} & \mathbf{P}_{m,k-1|k-1} \end{bmatrix} \tag{2.11}$$

where subscript rr denotes a covariance of robot state, subscript mm is a covariance of map landmarks, and subscript rm denotes a mixed covariance between robot and landmarks. Only the P_{rr} changes in the time update step. The term $\nabla f()$ is the Jacobian of the nonlinear function f calculated at the estimated state $\hat{\mathbf{x}}_{k-1|k-1}$.

II) Observation-update step

Computation of the mean

$$\begin{bmatrix}\hat{\mathbf{x}}_{k|k} \\ \hat{\mathbf{m}}_k\end{bmatrix} = \begin{bmatrix}\hat{\mathbf{x}}_{k|k-1} & \hat{\mathbf{m}}_{k-1}\end{bmatrix} + \mathbf{K}_k \left[\mathbf{z}_k - h\left(\hat{\mathbf{x}}_{k|k-1}, \hat{\mathbf{m}}_{k-1}\right)\right], \tag{2.12}$$

and the covariance

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k \left[\nabla h \mathbf{P}_{k|k-1} \nabla h^T + \mathbf{R}_k\right] \mathbf{K}_k^T \tag{2.13}$$

where $\mathbf{K}_k$ is the Kalman gain, ∇h is a Jacobian of a non-linear function $h()$ calculated at the robot state $\hat{\mathbf{x}}_{k|k-1}$, and the map state $\hat{\mathbf{m}}_{k-1}$.

Pros and cons

- ⊕ **Basic approach**: EKF is a well-known estimation algorithm that is easy to understand. EKF-SLAM is easy to understand too.
- ⊕ **Implementation**: EKF-SLAM is easy to implement, and it is suitable for a wide range of SLAM applications.
- ⊖ **Convergence**: The covariance matrix converges in limit monotonically to the value of the initial covariance of the robot location estimate [1]. In Figure 2.4, an example of landmark estimation convergence is shown. Every landmark has a high value of uncertainty when it is observed for the first time. Values converge monotonically in time, but they cannot be uncertainty free, which is also visible in Figure 2.4 as a positive non-zero lower bound of all values. The initial uncertainty of the sensor determines the lower bound.

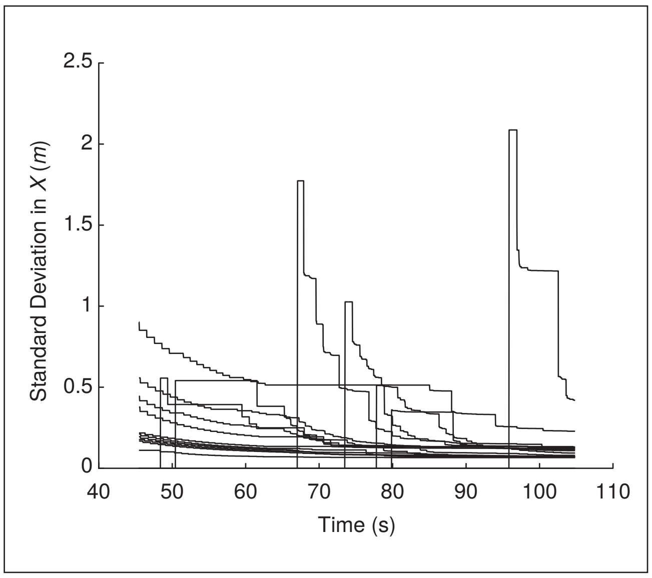

Figure 2.4: Convergence of estimated poses of landmarks in EKF-SLAM. Taken from [1]

- ⊖ **Computational complexity**: EKF has quadratic computational complexity ($O(n^2)$) for updating the covariance matrix P at every observation-update step. The covariance matrix size for a 2D SLAM problem is $3 + 2n \times 3 + 2n$, where n is the number of observed landmarks. It limits real-time calculation only on several dozen or hundreds of landmarks. There exist solutions that can update certain parts of the covariance matrix P individually and thus increase the algorithm's efficiency.
- ⊖ **Data Association**: It is one of the most fragile parts of SLAM and especially the EKF-SLAM solution. One incorrect association of observed features with map landmarks can cause a fatal error, which leads to failure of the algorithm and unrepairable degeneration of the map. The association during loop-closure can be much more challenging because the algorithm tries to associate new observations with older landmarks and figure out if there are any correspondences.
- ⊖ **Linearization**: Both the motion model and the observation model are linearized in EKF-SLAM. However, the real behavior of motion and geometry is non-linear. The consequence is that the algorithm may diverge because there is a significant difference between the model and the real world.

2.3.2 UKF-SLAM

The second filter based algorithm is called UKF-SLAM [2], and it is based on the Unscented Kalman filter (UKF). The principle is similar to the EKF-SLAM except for the way how the linearization of $f()$ is calculated. Instead of Taylor expansion, the unscented transform is used.

Unscented transform

Unscented transform linearizes function $f()$ using so-called sigma-points. Sigma-points are points that are deterministically chosen from the non-linear probability distribution using equations (2.14)-(2.18). Unscented transform avoids linearization around the mean of the distribution as Taylor expansion does. The computation of integrals is replaced with the calculation of a weighted sum of the sigma-points. The result of linearization is usually more accurate than in the case of EKF-SLAM. An algorithm of the unscented transform is written in the three steps as follows.

1. The computing of a set of sigma-points $\chi^{[i]}$ and weights $\omega^{[i]}$ are constrained as follows

$$\begin{aligned} \sum_i \omega^{[i]} &= 1, \\ \bar{\mathbf{x}} &= \sum_i \omega^{[i]} \chi^{[i]}, \\ \mathbf{P} &= \sum_i \omega^{[i]} (\chi^{[i]} - \bar{\mathbf{x}})(\chi^{[i]} - \bar{\mathbf{x}})^T, \end{aligned} \tag{2.14}$$

where $\bar{\mathbf{x}}$ and $\mathbf{P}$ are the mean vector and the covariance matrix of the probability density function reconstructed from sigma-points. The equations (2.14) are constraints ensuring that the mean vector $\bar{\mathbf{x}}$ and the covariance matrix $\mathbf{P}$ can be reconstructed from the sigma-points $\chi^{[\mathbf{i}]}$ and the weights $\omega^{[i]}$. There is no unique solution for $\omega^{[i]}$ and $\chi^{[\mathbf{i}]}$. Following equations are used to calculate sigma-points $\chi^{[\mathbf{i}]}$

$$\begin{aligned} \chi^{[\mathbf{0}]} &= \bar{\mathbf{x}}, \\ \chi^{[\mathbf{i}]} &= \bar{\mathbf{x}} + \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i, \qquad for \quad i = 1,\dots,n, \\ \chi^{[\mathbf{i}]} &= \bar{\mathbf{x}} - \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_{i-n}, \qquad for \quad i = n+1,\dots,2n, \end{aligned} \tag{2.15}$$

where n is a dimensionality of the distribution, λ is a scaling parameter. I.e., greater lambda increases the distance between computed sigma-points and the mean of the distribution. Subscripts i or $i-n$ are indexes of the particular column used after the matrix in brackets is calculated.

Calculation of weights $\omega^{[i]}$ are described by equations (2.16), (2.17) and (2.18)

$$\omega_{\mathbf{m}}^{[\mathbf{0}]} = \frac{\lambda}{n+\lambda} \tag{2.16}$$

where subscript m is an index of weights which are used to compute the mean of the distribution.

$$\omega_{\mathbf{c}}^{[\mathbf{0}]} = \omega_{\mathbf{m}}^{[\mathbf{0}]} + \left(1 + \alpha^2 + \beta\right) \tag{2.17}$$

where subscript c is an index of weights used to compute the covariance of the distribution and α, β are parameters of the unscented transformation. The value of parameter α is usually in the interval $(0, 1\rangle$. The optimal value of parameter β is usually 2 for Gaussian systems. A superscript [0] is an index of the sigma-point matched to the weight – in this case, equations (2.16) and (2.17) compute the weight for zero sigma-point. Weights for other sigma-points are calculated by equation (2.18).

$$\omega_{\mathbf{m}}^{[\mathbf{i}]} = \omega_{\mathbf{c}}^{[\mathbf{i}]} = \frac{1}{2\,(n+\lambda)} \qquad for \quad i = 1,\dots,2n \tag{2.18}$$

2. Process each sigma-point through nonlinear function $g(\chi^{[i]})$ to obtain a new set of transformed sigma-points. Considering SLAM, the sigma-points are transformed by the functions of the robot motion and observation model.

3. The last step is to recover Gaussian from the transformed weighted sigma-points using equations (2.19), (2.20).

$$\bar{\mathbf{x}}' = \sum_{i=0}^{2n} \omega_{\mathbf{m}}^{[\mathbf{i}]} g(\chi^{[i]}) \tag{2.19}$$

$$\mathbf{P}' = \sum_{i=0}^{2n} \omega_{\mathbf{c}}^{[\mathbf{i}]} \left(g(\chi^{[i]}) - \bar{\mathbf{x}}'\right)\left(g(\chi^{[i]}) - \bar{\mathbf{x}}'\right)^T \tag{2.20}$$

Moreover, there are two constraints for the selection of the λ parameter. It is defined by formulas

$$\kappa \geq 0,$$
$$\lambda = \alpha^2(n+\kappa)/n.$$

The selection of κ and α influences the distance of sigma-points from the mean of the probability distribution.

Pros and cons

- $\oplus$ **Better approximation**: A better approximation of non-linear models than EKF-SLAM.
- $\oplus$ **Jacobians**: No Jacobians are needed during the calculation.
- $\ominus$ **Complexity**: UKF-SLAM belongs to the same complexity class as the EKF-SLAM. Also, it is usually slower than the EKF-SLAM. Moreover, the model has to be Gaussian as well.

2.3.3 SEIF-SLAM

The SEIF-SLAM [43] approach is based on the Sparse Extended Information Filter (SEIF). It is an extension of the information filter, which is a dual filter to the Kalman Filter. The representation of the information filter is usually called the canonical representation. In the paragraphs below, the information filter is introduced, and its extended version (EIF), and then the sparsification step providing better performance with a large number of landmarks are described.

Information Filter

Similar to previous approaches, the information filter represents belief by a Gaussian An information vector ξ and an information matrix $\mathbf{\Omega}$ are used instead of the first two moments $\bar{\mathbf{x}}$ and $\mathbf{P}$ of the probability distribution. Information form – canonical representation – can be obtained from the moments using equations (2.21) and (2.22).

$$\mathbf{\Omega} = \mathbf{P}^{-1}, \tag{2.21}$$

$$\xi = \mathbf{P}^{-1}\bar{\mathbf{x}}. \tag{2.22}$$

Let's assume a linear motion

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k, \tag{2.23}$$

and observation model

$$\mathbf{z}_k = \mathbf{C}_k \mathbf{x}_k + \mathbf{v}_k, \tag{2.24}$$

where $\mathbf{A}_k$ is the matrix that describes the robot's motion, $\mathbf{B}_k$ is the matrix that describes the influence of the control $\mathbf{u}_k$. The mathematical relationship between the state $\mathbf{x}_k$ and the observation $\mathbf{z}_k$ is described by the matrix $\mathbf{C}_k$. Finally, $\mathbf{w}_k$ and $\mathbf{v}_k$ are random variables representing the additive zero-mean process and measurement noise with covariance $\mathbf{R}_k$ and $\mathbf{Q}_k$.

The complexity of particular steps in the algorithm is dual to the Kalman filter. For example, the time-update step is trivial in the moment representation, but it is expensive in the canonical representation. It is visible in equations (2.25) and (2.26) where the inverse of the matrix has to be computed. Similarly, the observation update step is expensive in the moment representation, but it is trivial in the canonical representation – see Equations (2.27) and (2.28).

$$\mathbf{\Omega_{k|k-1}} = (\mathbf{A}_k \mathbf{\Omega}^{-1}_{k-1|k-1} \mathbf{A}_k^\top + \mathbf{R}_k)^{-1}, \tag{2.25}$$

$$\boldsymbol{\xi}_{k|k-1} = \boldsymbol{\Omega}_{k|k-1}(\mathbf{A}_k\boldsymbol{\Omega}^{-1}_{k-1|k-1}\boldsymbol{\xi}_{k-1} + \mathbf{B}_k\mathbf{u}_k)^{-1}, \tag{2.26}$$

$$\boldsymbol{\Omega}_{k|k} = \mathbf{C}_k^\top\mathbf{Q}_k^{-1}\mathbf{C}_k + \boldsymbol{\Omega}_{k|k-1}, \tag{2.27}$$

$$\boldsymbol{\xi}_{k|k} = \mathbf{C}_k^\top\mathbf{Q}_k^{-1}\mathbf{z}_k + \boldsymbol{\xi}_{k|k-1}, \tag{2.28}$$

The information filter described above is suitable only for linear systems. The solution to non-linear systems is EIF. It is an analogy to the EKF.

Extended Information Filter

In this algorithm, the same principle as in the case of the EKF is used. The non-linear system is linearized using Taylor expansion. Equations (2.29) - (2.34) are the equations of EIF. They are similar to Equations (2.25) - (2.28), but non-linear functions $f()$ and $h()$ and their Jacobians are used instead of system matrices $\mathbf{A_k}$, $\mathbf{C_k}$, and $\mathbf{B_k}$. Moreover, the prediction and correction state vectors $\mathbf{x}$ has to be calculated to compute $\boldsymbol{\xi}$.

$$\mathbf{x}_{k-1|k-1} = \boldsymbol{\Omega}^{-1}_{k-1|k-1}\boldsymbol{\xi}_{k-1|k-1} \tag{2.29}$$

$$\boldsymbol{\Omega}_{k|k-1} = (\nabla\mathbf{f}_k\boldsymbol{\Omega}^{-1}_{k-1|k-1}\nabla\mathbf{f}_k^\top + \mathbf{R}_k)^{-1} \tag{2.30}$$

$$\mathbf{x}_{k|k-1} = f(\mathbf{u}_k, \mathbf{x}_{k-1|k-1}) \tag{2.31}$$

$$\boldsymbol{\xi}_{k|k-1} = \boldsymbol{\Omega}_{k|k-1}\mathbf{x}_{k|k-1} \tag{2.32}$$

$$\boldsymbol{\Omega}_{k|k} = \nabla\mathbf{h}_k^\top\mathbf{Q}_k^{-1}\nabla\mathbf{h}_k + \boldsymbol{\Omega}'_{k|k-1} \tag{2.33}$$

$$\boldsymbol{\xi}_{k|k} = \nabla\mathbf{h}_k^\top\mathbf{Q}_k^{-1}(\mathbf{z}_k - h(\bar{\mathbf{x}}_{k|k-1}) + \nabla\mathbf{h}\bar{\mathbf{x}}_{k|k-1}) + \boldsymbol{\xi}_{k|k-1} \tag{2.34}$$

The performance of the extended information filter is not suitable for a dense information matrix. Significantly better performance can be obtained by sparsification of the normalized information matrix.

Sparsification

The normalized information matrix can be interpreted as a graph of links between landmarks. Most of the landmarks have only a small number of links to other landmarks. Larger values of O are usually between nearby landmarks. Again, it is an analogy with the net of springs – It is shown in Figure 2.3. Most of the off-diagonal elements are close to 0. The solution to the sparsification step is to set that element's values to 0. Figure 2.5 is shown a comparison of SEIF-SLAM without and with sparsification. It is an example of SEIF-SLAM with 50 landmarks in the environment taken from the publication of Thrun et al. [2]. The effect of sparsification is seen on the graph on the left. The version b) contains significantly fewer links between landmarks. The information matrices on the right look almost the same because high values are the dark ones. Thus, the white cells are close to zero – case a – and zero case b.

In the SEIF-SLAM algorithm, only a subset of all landmarks is used for computation of the next step. They are referred to as active landmarks. The active landmark is a landmark observed by the robot in the current time step. The robot is connected only to the active landmarks. All landmarks are connected only to nearby landmarks – landmarks that are active at the same time. Other connections are weak, and they are set to 0.

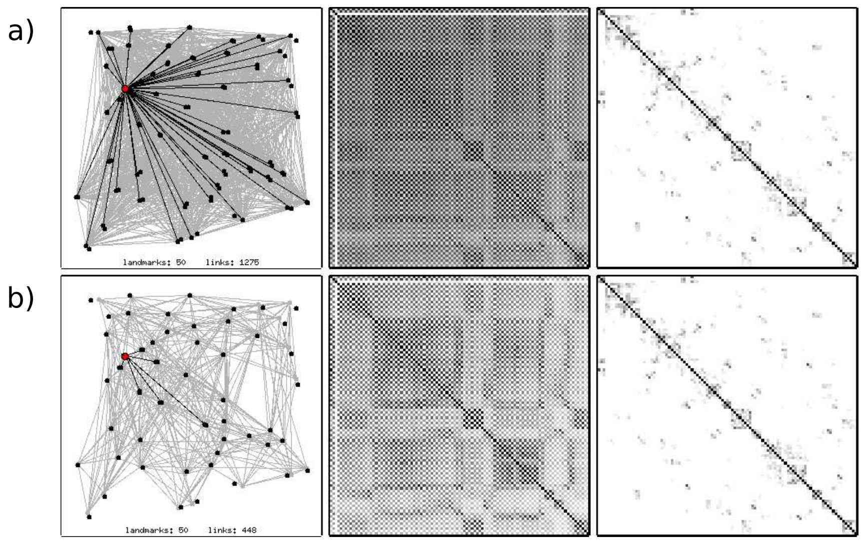

Figure 2.5: The effect of the sparsification step in SEIF-SLAM. Taken from [2]

Pros and cons

- ⊕ **Memory and CPU**: Significantly better performance than EKF-SLAM. SEIF-SLAM needs much less memory, thanks to the sparsification of the information matrix. The average CPU time for one iteration increases slowly with increasing the number of landmarks compared to EKF – according to [44]. The SEIF-SLAM requires constant time in the map size. Therefore the SEIF-SLAM is more efficient in large scale scenarios.
- ⊖ **Accuracy**: The SEIF-SLAM is less accurate than EKF because of sparsification, which causes the loss of information about the correlation between farther landmarks. Thus, the EKF-SLAM is more suitable for small-scale scenarios.

2.3.4 Particle Filter SLAM

The approaches described above have a significant disadvantage. They assume that the distribution is Gaussian. Better results could be achieved when the arbitrary distribution can be assumed. This goal can be met using a particle filter algorithm.

Particle filter

A particle filter is a recursive Bayes filter [2]. It is representative of a non-parametric filter. The particle filter's key idea is to use so-called samples to represent an arbitrary probability distribution. The sample is a state hypothesis drawn from a probability distribution of the system state. It is the first part of the particle. The second part is the weight of a sample. More than one particle is used in the particle filter algorithm. A group of particles is called a particle set.

The particle set χ is a set of pairs composed of sample $x^{[i]}$ and its weight $\omega^{[i]}$

$$\chi = \left\{ \left\langle x^{[i]}, \omega^{[i]} \right\rangle_{i=1\ldots n} \right\}, \tag{2.35}$$

where $[i]$ denotes i-th particle in the particle set. The posterior distribution can be obtained using equation

$$p(x) = \sum_{i=1}^{N} \omega^{[i]} \delta_{x^{[i]}}(x), \tag{2.36}$$

where $\delta_{x^{[i]}}(x)$ is the Dirac delta function in the location of the state hypothesis $x^[i]$.

The particle filter algorithm has three steps. The first step is a sampling. The second one is computing the importance of weight, and the third is the resampling step. The particle set is created by sampling from the proposal distribution π.

The importance sampling principle says that it is possible to use probability distribution π to generate samples from an arbitrary distribution of f. Distribution π is called the proposal distribution, and the distribution f is the target distribution. It is a fundamental idea because the sampled values from the proposal distribution can be transformed into the target distribution. In the first step, samples are drawn from the proposal distribution, and then all samples are weighted in the correction step by the equation (2.37) to obtain samples of f.

$$\omega = \frac{f(x^{[i]})}{\pi(x^{[i]})}. \tag{2.37}$$

The accuracy of the target distribution approximation increases with the number of samples. In the case of an infinite number of samples, the target distribution can be reconstructed exactly

In the resampling step, the most unlikely particles are drawn from the particle set, and they are replaced with the more likely ones. It is a trick to avoid samples that cover unlikely states. It is necessary because the number of samples is always finite. There are many resampling approaches like the roulette wheel approach, stochastic universal sampling [45], residual or systematic resampling [46]. However, the solution to the resampling that could be denoted as the best step doesn't exist.

FastSLAM

The SLAM problem defined by Equation (2.2) can not be efficiently solved by the particle filter. The reason is that the distribution's state space is high-dimensional, but the particle filter is effective only for low dimensional spaces. The idea of FastSLAM [3], [47] is to exploit dependencies between the different dimensions of state space by using the particle filter to represent only the robot's state. Each particle is assumed to be a hypothesis of a robot path and an individual map of landmarks. Landmarks are computed for each sample. These properties are obtained using the Rao-Blackwellization (R-B) process based on the chain rule described by the equation

$$p(a, b) = p(b \mid a)p(a). \tag{2.38}$$

It is using conditional probabilities to calculate any member of the joint probability distribution. The application of R-B on the probability density function (2.2) is shown in the equation

$$p\left(\mathbf{x}_{0:k}, \mathbf{m}_{1:M} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}\right) = p\left(\mathbf{m}_{1:M} \mid \mathbf{x}_{0:k}, \mathbf{U}_{0:k}\right)_{(1)} p\left(\mathbf{x}_{0:k} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}\right)_{(2)}. \qquad (2.39)$$

In term denoted by subscript $_{(1)}$, the path of the robot is known. The particle filter can effectively solve the second term denoted by subscript $_{(2)}$. The consequence is that all landmarks are independent, and the second term can be rewritten as follows

$$p\left(\mathbf{m}_{1:M} \mid \mathbf{x}_{0:k}, \mathbf{U}_{0:k}\right) \approx \prod_{i=1}^{M} p\left(\mathbf{m}_{i} \mid \mathbf{x}_{0:k}, \mathbf{U}_{0:k}\right). \qquad (2.40)$$

In the case of 2D SLAM, landmarks in the equation (2.40) can be processed independently by the update step of the EKF algorithm.

One cycle of FastSLAM algorithm

1. Compute new pose from the proposal distribution.

   $$x_k^{[i]} \sim \pi\left(x_k \mid x_{k-1}^{[i]}, u_k\right)$$

   The control vector u_k, and a pose of i-th particle $x_{k-1}^{[i]}$ is given.

2. Compute of particle weight

   $$\omega^{[i]} = \mid 2\pi Q \mid^{-\frac{1}{2}} \exp -\frac{1}{2}\left(z_k - \bar{z}^{[i]}\right)^T Q^{-1}\left(z_k - \bar{z}^{[i]}\right),$$

   where $\bar{z}^{[i]}$ is expected observation and Q is a measured covariance.

3. Update belief of observer state. Each particle contains the sampled value of the robot pose, its weight, and map landmarks. Landmarks are represented by pairs of a mean vector μ and a covariance matrix Σ. Updating of belief of observed landmarks is the third step of the algorithm. There is only a single landmark observed during one cycle of the algorithm. There are two possible cases.

   - If the particular landmark is not observed, its values are only copied to the current time step

     $$\left\langle \mu_k^{[m]}, \Sigma_k^{[m]} \right\rangle_{[i]} = \left\langle \mu_{k-1}^{[m]}, \Sigma_{k-1}^{[m]} \right\rangle_{[i]}$$

     where superscript m denotes the m-th landmark in the map, and subscript $[i]$ denotes i-th particle from the particle set.

   - When the landmark is observed, it may be a landmark already on the map or a new previously unobserved landmark. The EKF is initialized for the new landmark. In the case of existed landmark, the EKF update step is performed.

4. Resample. The resampling step draws a subset of particles with replacement.

The FastSLAM algorithm described above has linear computational complexity in the number of landmarks $N - O(MN)$, where M is the number of particles and N is the number of landmarks. The authors of the algorithm came with an efficient implementation using balanced binary trees to represent a particle. The computational complexity can be improved using binary trees to logarithmic in the number of landmarks – $O(MlogN)$. In the bottom part of Figure 2.6, an example of the binary tree for one particle is shown. The landmarks mean vectors and covariance matrices are saved in the leaves of the tree. Therefore, the picking of the particular leaf has $O(1)$. When the new particle is obtained during the updating step, only one path from the root to the leaf is created. The other values are only referenced from the old particle. The situation is also shown in Figure 2.6.

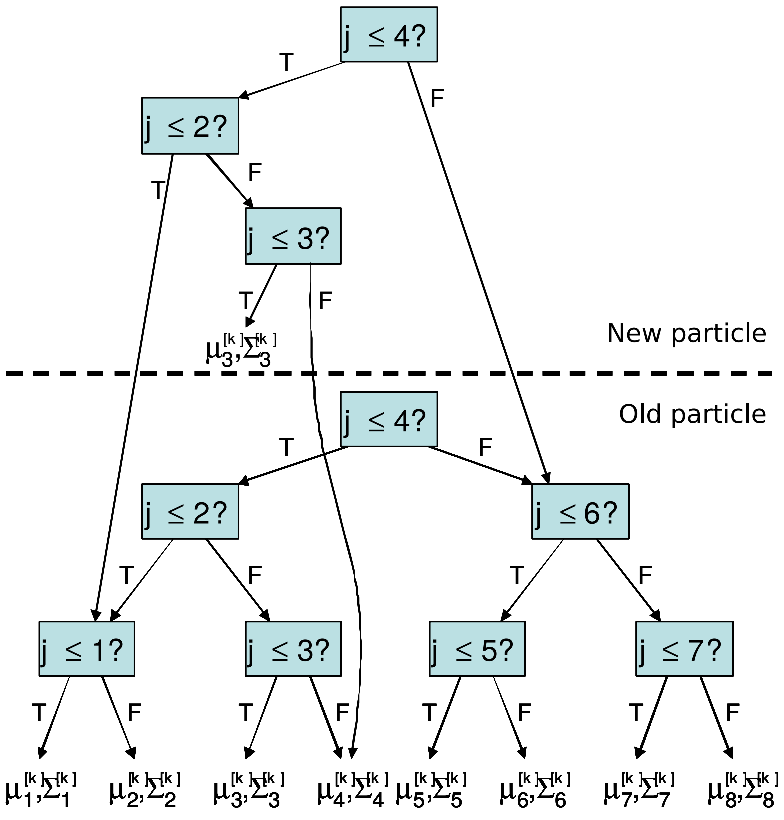

Figure 2.6: An example of balanced binary tree with 8 landmarks. Taken from [3]

FastSLAM 2.0

Improved version of this algorithm called FastSLAM 2.0 was proposed in paper [48]. Different version of the proposal distribution is used by this algorithm. In particular, samples are obtained from the distribution in Equation 2.41. Current measurement is considered during the sampling step.

$$x_k^{[i]} \sim \left(x_k \mid x_{k-1}^{[i]}, u_k, z_k \right). \tag{2.41}$$

FastSLAM 2.0 algorithm can process 10^6 and more landmarks, and it is robust in the data association.

Similar to FastSLAM 2.0, the papers of Grisetti et al. [49], [50] proposed a suitable solution for grid maps. It is achieved similarly to the FastSLAM 2.0. – current measurement is considered in the proposal distribution.

2.4 Optimization based SLAM

The second group of approaches is based on optimization instead of filters. The goal is to minimize the error between a real and an expected measurement. The approach is based on the graph-based structure, which represents the map. The graph contains nodes and edges. The minimization process is supposed to find the correct spatial configuration of the graph nodes. It leads to using the Nonlinear Least Squares (NLS) algorithm.

2.4.1 Nonlinear Least Squares

The NLS is a well-known algorithm which is a standard approach for computing a wide range of problems. For example, the first problem solving by NLS was the computing of the future position of the asteroid Ceres in 1801.

The difference between the real and an expected measurement is called the error function $\mathbf{e}_i$, and it is defined as follows

$$\mathbf{e}_i(x) = z_i - f_i(x), \tag{2.42}$$

where z_i is the real measurement and $f_i(x)$ is the expected measurement based on the state of the system x. In the SLAM problem, x is a position of the robot. The error is usually assumed to be normally distributed with a zero mean and the information matrix Ω_i . The squared error e_i is computed from Equation (2.42) as follows

$$e_i(x) = \mathbf{e}_i^T \Omega_i \mathbf{e}_i. \tag{2.43}$$

The result value of the squared error is a scalar. Two assumptions have to be fulfilled

1. All error functions of measurements are smooth in the neighborhood of a global minimum.
2. The initial guess of x is available.

The goal is to find state x, which minimizes the error function of all measurements – i.e., the global error.

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} F(\mathbf{x}) = \arg\min_{\mathbf{x}} \sum_i e_i = \arg\min_{\mathbf{x}} \sum_i \mathbf{e}_i^T \Omega_i \mathbf{e}_i, \tag{2.44}$$

The error function is non-linear in general. It has to be linearized around initial guess of $\mathbf{x}$. The Taylor expansion is used for linearization

$$\mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{e}_i + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x} \tag{2.45}$$

where $\mathbf{J}_i(x)$ is a Jacobian of the error function concerning $\mathbf{x}$. The next step is to replace the terms in the squared error equation

$$\begin{aligned} e_i(\mathbf{x}) &= \mathbf{e}_i^T(\mathbf{x} + \Delta\mathbf{x})\Omega_i \mathbf{e}_i(\mathbf{x} + \Delta\mathbf{x}) \\ &\simeq (\mathbf{e}_i + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x})^T \Omega_i (\mathbf{e}_i + \mathbf{J}_i(\mathbf{x})\Delta\mathbf{x}) \\ &= \mathbf{e}_i^T \Omega_i \mathbf{e}_i + \mathbf{e}_i \Omega_i \mathbf{J}_i \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega_i \mathbf{e}_i + \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega \mathbf{J}_i \Delta\mathbf{x} \\ &= \mathbf{e}_i^T \Omega_i \mathbf{e}_i + 2\mathbf{e}_i \Omega_i \mathbf{J}_i \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega \mathbf{J}_i \Delta\mathbf{x} \\ &= \mathbf{c}_i + 2\mathbf{b}_i^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x} \end{aligned} \tag{2.46}$$

where $\mathbf{c}_i = \mathbf{e}_i^T \Omega_i \mathbf{e}_i$, $\mathbf{b}_i^T = \mathbf{e}_i \Omega_i \mathbf{J}_i$ and $\mathbf{H}_i = \mathbf{J}_i^T \Omega \mathbf{J}_i$. The result above is a linearized error function of a single measurement. The next step is to calculate a linearized version of the global error

$$\begin{aligned} F(\mathbf{x} + \Delta\mathbf{x}) &= \sum_i e_i = \sum_i (\mathbf{c}_i + 2\mathbf{b}_i^T \Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x}) \\ &= \sum_i \mathbf{c}_i + 2(\sum_i \mathbf{b}_i^T)\Delta\mathbf{x} + \Delta\mathbf{x}^T(\sum_i \mathbf{H}_i)\Delta\mathbf{x} \\ &= \mathbf{c} + 2\mathbf{b}^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}. \end{aligned} \tag{2.47}$$

The global error F is derived as a quadratic form concerning $\Delta\mathbf{x}$

$$\frac{\partial F}{\partial \Delta\mathbf{x}} \simeq 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x}. \tag{2.48}$$

The derivative is set to zero

$$0 = 2\mathbf{b} + 2\mathbf{H}\Delta\mathbf{x}, \tag{2.49}$$

and the resulting linear system

$$\mathbf{H}\Delta\mathbf{x}^* = -\mathbf{b} \rightarrow \Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b} \tag{2.50}$$

can be solved using Cholesky decomposition, QR factorization, or some iterative methods.

The process described above is the principle of the Gauss-Newton algorithm, which can be summarized as follows

1. Linearize the global error function around the initial guess of the state $\mathbf{x}$.
2. Compute $\mathbf{b}^T$ and $\mathbf{H}$.
3. Solve the linear system $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$.
4. Update the state $\mathbf{x} := \mathbf{x} + \Delta\mathbf{x}^*$.
5. Iterate until convergence.

2.4.2 Graph SLAM

The NLS based SLAM approach is usually called the graph SLAM [32] [2] [51]. It is usually decomposed into two components called the frontend and the backend. The frontend is supposed to create a graph that represents the map of the environment. The goal of the backend is to optimize the graph.

The graph is an abstraction of the environment. It is composed of nodes and edges. There are two types of nodes. The first one is the poses of the robot. The second type is the landmark locations. When is the right time to add a new node that represents the pose of the robot? The answer is not simple. Usually, it is solved by some heuristically determined threshold. The nodes are connected by edges, which represent spatial constraints. Therefore there are three types of edges. The edges between two consecutive robot pose, the edges between robot and landmarks, and the edges created during loop closing. An example of the graph is shown in Figure 2.7

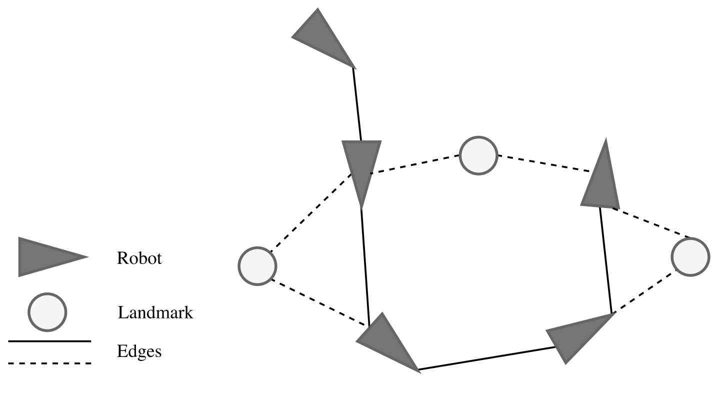

Figure 2.7: Graph slam

The graph is created by the frontend component, which usually uses some matching algorithm. The goal of the matching algorithm is to estimate the transformation between two nodes. The popular approaches are feature-based matching, descriptor-based matching, and dense scan-matching. The first one is based on the positions of the features in the data. The second one usually compares the feature vectors calculated by some feature descriptor method. The third type uses all data during matching.

The graph made by the frontend isn't usually correct. It is because the frontend is only adding nodes to the graph without resolving this information. The edge between two consecutive nodes is an analogy to an odometry measurement. The robot observes the same part of the environment. Therefore it is possible to calculate the virtual measurement - using rigid body transformations - of current data seen from the previous robot pose.

This process is called lazy data association. It is the opposite of the active data association in filter-based approaches where added data are resolved immediately. The graph is corrected by the backend. It is achieved by applying the NLS error minimization defined in Equation (2.44).

The optimization process is based on the error function $e_{i,j}$ of the edge between two poses $\mathbf{x}_i$ and $\mathbf{x}_j$. The Jacobian of the error will be non-zero only in the rows corresponding to these poses

$$\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}} = \left(0 \cdots \frac{\partial \mathbf{e}_{ij}(\mathbf{x_i})}{\partial \mathbf{x}_i} \cdots \frac{\partial \mathbf{e}_{ij}(\mathbf{x}_j)}{\partial \mathbf{x}_j} \cdots 0\right) \tag{2.51}$$

and the Jacobian can be written as follows

$$\mathbf{J}_{ij} = (0 \cdots \mathbf{A}_{ij} \cdots \mathbf{B}_{ij} \cdots 0) \tag{2.52}$$

Moreover, the sparse structure of $\mathbf{J}_{ij}$ results in the sparse structure of matrix $\mathbf{H}$. An adjacency of the nodes in the graph is reflected in that structure. The coefficient vector $\mathbf{b}_{ij}^T$ is computed as follows

$$\begin{aligned} \mathbf{b}_{ij}^T &= \mathbf{e}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{J}_{ij} \\ &= \mathbf{e}_{ij}^\top \mathbf{\Omega}_{ij} (0 \cdots \mathbf{A}_{ij} \cdots \mathbf{B}_{ij} \cdots 0) \\ &= \left(0 \cdots \mathbf{e}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} \cdots \mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij} \cdots 0\right). \end{aligned} \tag{2.53}$$

Similarly, the calculation of the coefficient matrix $\mathbf{H}_{ij}$

$$\begin{aligned}
\mathbf{H}_{ij} &= \mathbf{J}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{J}_{ij} \\
&= \begin{pmatrix} 0 \\ \vdots \\ \mathbf{A}_{ij}^\top \\ \vdots \\ \mathbf{B}_{ij}^\top \\ \vdots \\ 0 \end{pmatrix} \Omega_{ij} \left(0 \cdots \mathbf{A}_{ij} \cdots \mathbf{B}_{ij} \cdots 0\right). \\
&= \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & \mathbf{A}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{B}_{ij} & \cdots \\ & \vdots & & \vdots & \\ \cdots & \mathbf{B}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{B}_{ij} & \cdots \\ & \vdots & & \vdots & \end{pmatrix}.
\end{aligned} \tag{2.54}$$

The sparseness allows us to calculate the vector $\mathbf{b}^\top$ and matrix $\mathbf{H}^\top$ individually as a sum of all edges connected to the particular node. Every edge contributes into two values in vector $\mathbf{b}^\top$ and into four values in matrix $\mathbf{H}$. Let's assume that there is an edge between the node $\mathbf{x}_i$ and $\mathbf{x}_j$. The contribution of this edge can be calculated as follows

$$\begin{aligned}
\mathbf{b}_i^\top &= \mathbf{b}_i^\top + \mathbf{e}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} \\
\mathbf{b}_j^\top &= \mathbf{b}_j^\top + \mathbf{e}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{B}_{ij} \\
\mathbf{H}_{ii} &= \mathbf{H}_{ii} + \mathbf{A}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} \\
\mathbf{H}_{ij} &= \mathbf{H}_{ij} + \mathbf{A}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{B}_{ij} \\
\mathbf{H}_{ji} &= \mathbf{H}_{ji} + \mathbf{B}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{A}_{ij} \\
\mathbf{H}_{jj} &= \mathbf{H}_{jj} + \mathbf{B}_{ij}^\top \mathbf{\Omega}_{ij} \mathbf{B}_{ij}
\end{aligned} \tag{2.55}$$

After summing up all contributions, the vector $\mathbf{b}^\top$ and matrix $\mathbf{H}$ of the linear system are calculated, and finally, the system can be solved. Some sparse versions of the decomposition algorithm can be used. As an example, sparse Cholesky decomposition or conjugate gradients method.

The optimization task can be computationally expensive to perform frequently. Therefore, the hierarchical pose graph is used. It groups topologically nearby nodes at the bottom level of the graph. Every group represents one node in the higher level of the graph. This process is repeated until a particular number of levels is created. The key idea is to correct only the structure of the graph instead of every single node. The optimization process is started on the highest level, and it is propagated using rigid body transformation to lower levels only in the local neighborhood of the current position. Therefore, only the upper level is completely optimized. It improves the efficiency of the graph SLAM algorithm.

An essential part of the theory necessary to understand the SLAM problem was introduced in preceding sections. Common approaches were described as a core for state-of-the-art SLAM systems described in Chapter 2.8.

2.5 Robot platform

This section aims to describe the standard types of robot platforms. As mentioned in Chapter 1, the SLAM problem is usually solved by a mobile robot equipped with several sensors [41] [52]. The selection of a type of mobile robot depends on the environment and application performed. Concerning the environment, ground, aerial and underwater robots can be considered. The more general camera handheld device can be used in all environments, but it has to deal with the missing of other sensors.

2.5.1 Ground vehicle

The first type of robot platform is a ground vehicle. The ground vehicle is a platform with a differential or the Ackerman driving geometry. It operates on the surface. Therefore only 2D SLAM is usually performed. It is capable of operating in both indoor and outdoor scenarios. Ground vehicles are useful for inspecting old abandoned mines, search and rescue missions in a forest or other poorly accessible places. Two examples of the ground vehicles are shown in Figure 2.8 – both are developed and assembled in the robotic laboratory at NTIS research centre. The left one is a smaller platform based on the Wild Thumper chassis with differential driving geometry. The one on the right is a larger platform based on racing RC chassis with Ackerman driving geometry. The chassis with Ackerman is usually not suitable for indoor scenarios. In this work, I used a robot based on the Wild Thumper chassis with differential driving geometry.

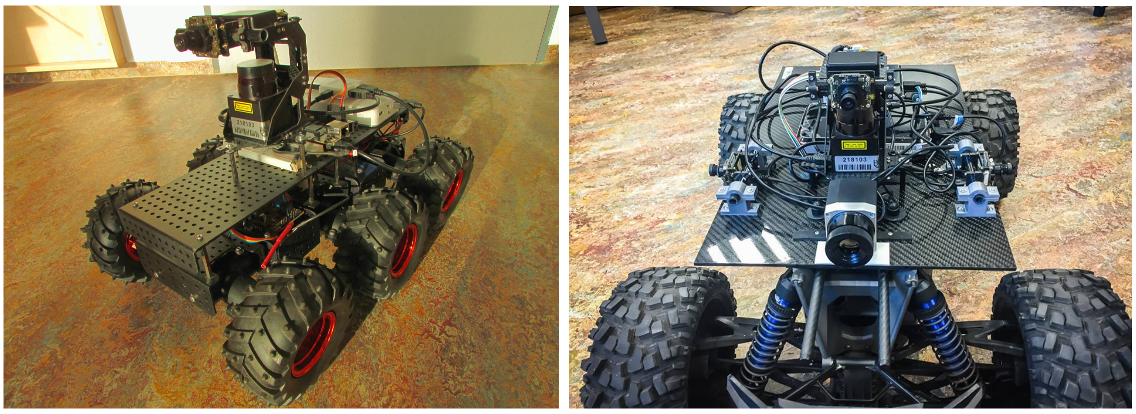

Figure 2.8: Ground vehicles

Another application of the SLAM problem suitable for using ground vehicles is self-driving cars. Several car companies around the world invest a lot of money in the research of autonomous vehicles. The self-driving car in the traffic has to process a big amount of data and reacts in milliseconds. Unfortunately, this technology isn't perfect, and the driver has to be ready to take over the control to avoid dangerous situations.

2.5.2 Aerial vehicles

The second type is aerial vehicles. A quadrotor drone usually represents this category. The advantage is that aerial vehicles can operate in three-dimensional space instead of 2 dimensions, as in ground vehicles. On the other side, the motion of the aerial vehicle is more complex. Therefore, it is more expensive in the sense of memory and computational requirements. Another problem arises when the robot loses a signal or the battery goes low. In the case of a ground vehicle, it only stops. The aerial vehicle has to land safely because, in the opposite case, it can fall and shatter itself on the ground – or worse, hurt someone. The example of an aerial vehicle is shown in Figure 2.9. It is Asctec Pelican from Ascending technologies.

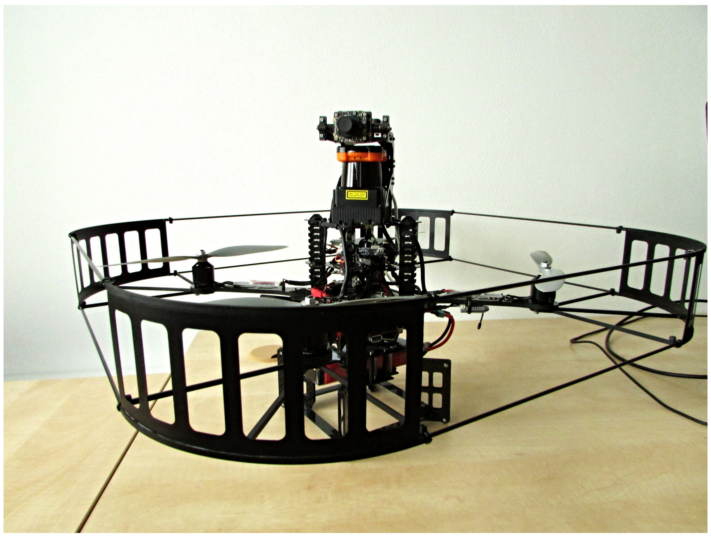

Figure 2.9: AscTec Pelican

2.5.3 Underwater vehicles

Many scenarios can be performed underwater – e.g., the condition monitoring of corals in the oceans. This kind of application can be solved using underwater vehicles – i.e., submarine robots. Standard sensors for ground and aerial vehicles can be useless in underwater scenarios. It is the reason why the underwater scenarios are often more complicated than previous types.

2.5.4 Handheld devices

Another type of platform is a handheld device – usually some smartphone device. It isn't the mobile robot platform, but many scenarios can be performed on handheld devices. The smartphone device usually has a camera sensor, an accelerometer, and a gyroscope. It is a sufficient combination to provide a map using visual SLAM algorithms. On the other hand, there is a problem with the performance of device hardware. The solution to this problem can be found in a cloud technology that can compute SLAM on a high-performance server. The crucial question is how to minimize data transfer between device and server without loss in the map's accuracy.

2.6 Sensors

In the following paragraphs, sensors suitable to obtain data for the SLAM task will be described. There are three types of sensors called non-vision sensors, vision sensors, and support information sensors. Based on this classification, existing SLAM algorithms can be divided into non-vision, vision, and combined – the combined category of SLAM algorithms using data from both types of sensors. Similarly, the non-vision or the vision category are based on one or more sensors from the same category. Support sensors are supposed to provide support information like the direction of movement or speed of robot wheels.

The selection of suitable sensors for a particular application is crucial for obtaining good and accurate results. For example, the camera is the right choice for environments with good light conditions and distinguishable objects, but it is useless in dark or low texture environments. Another example of the problematic object for the camera-based SLAM can be a mirror because the movement of the object in the mirror is different, and it can cause the failure of the algorithm.

2.6.1 Non-vision

The category of non-vision sensors is represented by distance sensors based on different technologies. Nowadays, the most common distance sensor is LiDAR. An example of a LiDAR sensor is shown in Figure 2.10.

Figure 2.10: Hokuyo LiDAR

The LiDAR provides 2D or 3D scans of the environment. The data is composed of distances and relative angles to points around the sensor. Because of that, LiDAR can be classified as a range-bearing sensor. Measuring is based on a laser beam, which is used to illuminate a target object. The receiver then detects the reflected light. The distance is calculated from the time of the flight of the laser beam. In Figure 2.11, LiDAR data visualization is shown.

Endpoints of the laser beam are visualized as cubes. The red line is the trajectory of the robot.

Another types distance sensors can be based on infrared technology represented by infrared distance sensors, a sound-based sensor such as sonar, RADAR sensor, or sensors based on magnetism, inductance, or capacity.

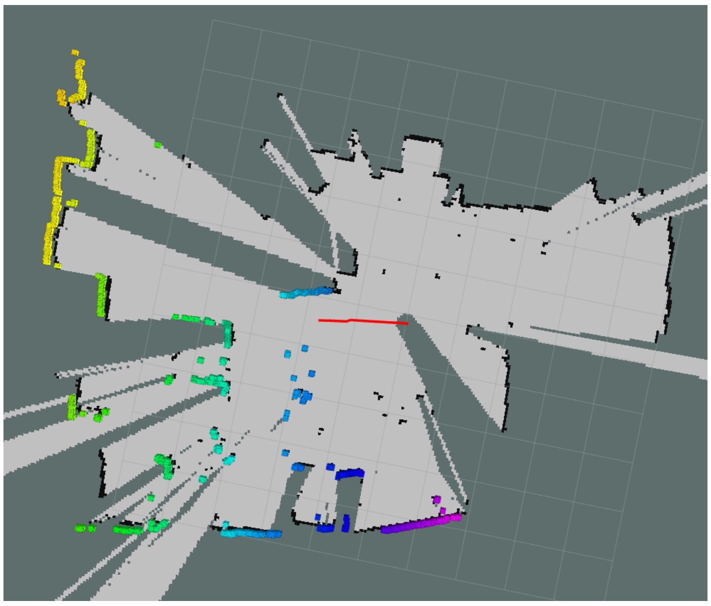

Figure 2.11: LiDAR data visualization

2.6.2 Vision sensors

A wide range of cameras represents the second category. In this section, basic types of vision sensors will be described. Only bearing information is provided by most of the vision sensors. Fortunately, missing information about the distance can be obtained from the motion in the sequence of images. The advantage of the vision sensor is the information contained in the image data. A lot of information – more than LiDAR – about the environment is included in the camera image.

Consequently, it is possible to use image data to solve the SLAM problem and the number of detection or tracking tasks. A large amount of data in the image causes the computational complexity to rise with its resolution. The most common vision sensor is a monochrome or an RGB camera sensor. This type of sensor will be denoted as the camera in the next sections.

Camera

The main argument for using the camera is its price. A USB webcam is easy to use a low price solution. A better way can be using an industrial camera sensor that is more expensive but has better performance. The theoretical frame rate of the cheap webcam is 30Hz, but the real number can be only 10Hz. On the other hand, the frame rate written in the documentation of an industrial camera is guaranteed.

Data obtained from the camera is a matrix of pixels. The size of the matrix is called the resolution of an image. Every pixel has a defined number of bits. In the case of an RGB camera, it is 8 bits per color – 24 bits in total. The monochrome pixels have only one channel, but it can have more bits – e.g., 12 bits. The final amount of data in bits can be computed as follows

$$d = w \times h \times d, \tag{2.56}$$

where w is the width of the image, h is the height of the image, and d is data type of the image. The data type is usually 32 bits integer for RGB or 8 bits integer for monochrome. An example of the camera sensor is shown in Figure 2.12.

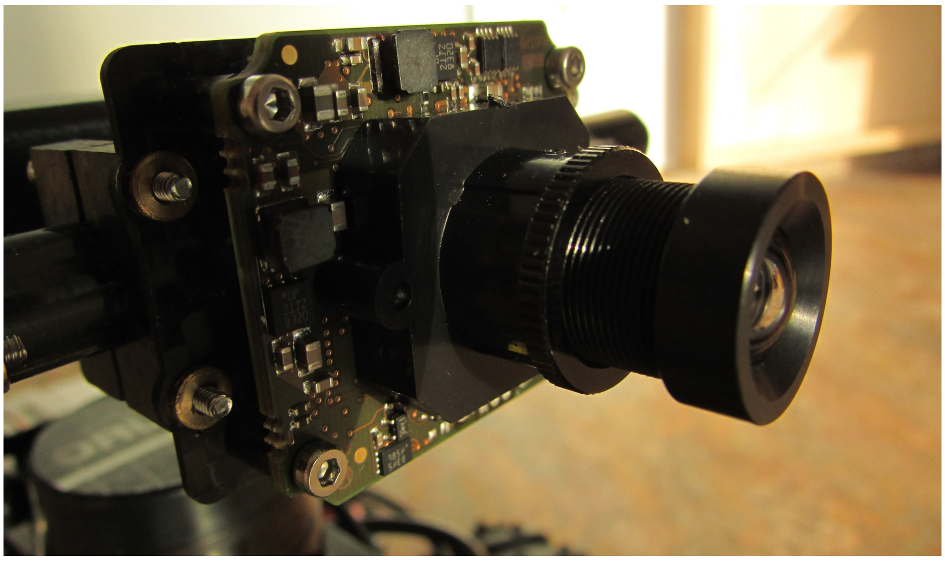

Figure 2.12: Camera sensor

The system can be based on data from one camera – monocular system – or from more cameras. The most common multicamera system is two cameras system – i.e., a stereovision. The stereovision is accomplished by a pair of cameras which are set in parallel in the known distance. The information about the distance between cameras is used to compute the object's distance in the image. The object has to be observed in the images from both cameras. Of course, a higher number of cameras can be used.

A computing of the third coordinate is possible only if the camera or the stereovision is calibrated. Calibration is a process by which the parameters of a camera are estimated. The information about image calibration can be found in a wide range of image processing books – e.g., chapter 11 in [33]. It is also briefly described in Appendix A.2 of this thesis.

Similar to the classic camera sensor is a thermal imaging camera. Instead of the visible part of a light spectrum, the thermal imaging camera is sensitive to infrared part of the spectrum. The disadvantage of a thermal imaging camera is its price. It is expensive, but in some environments, it can be more useful than the classic camera. Especially environments with several warm or heating objects will be suitable for using thermal imaging cameras.

RGBD camera

Another vision sensor is the RGBD camera, where D is the first letter from the word depth. This kind of camera provides distance data together with the RGB image. Depth data is similar to the matrix in the case of the monochrome camera. The depth and RGB matrices are registered with each other. The consequence is that it is possible to find the RGB and D values of particular points of the scene in the same coordinates in both matrices.

The RGBD camera is widely used in the last years because there is no need to calculate the distance from the RGB image data. The best-known RGBD sensor is Microsoft Kinect, but a lot of companies came with their solution. In Figure 2.13, an example of the RGBD camera is shown.

Figure 2.13: RGBD camera

Event camera

Event camera is a relatively new sensor for SLAM – since 2008. Other names for this type of sensor are Dynamic Vision Sensor (DVS)[1] or Asynchronous Timebased Image Sensor (ATIS)[53]. Instead of the acquisition of whole frames in a fixed frame rate, local changes at the pixel level are obtained. Reaction to the change in a particular pixel is quick and independent of the other pixels. Moreover, there are two types of changes based on the positive or negative change of brightness value. The result is a continuous stream of events.

The event camera's average latency is 1 microsecond, and the measurement rate goes up to 1 Mhz. It is much faster than the high-speed monochrome or RGB camera, with a frame rate of up to thousands of frames per second. The advantage of the event camera is the capability to provide data without blur, even in high-speed movement. The Requirements for computation performance and data storage is reduced. An example of DVS is shown in Figure 2.14. On the other hand, there is a need to create a new version of standard image processing and computer vision algorithms like feature detection or tracking algorithms.

[1]https://inilabs.com/

Figure 2.14: Event camera

2.6.3 Support sensors

The third category is represented mainly by inertial sensors. It is not sensors that can be used standalone while solving the SLAM problem. Instead, they are the source of some support information about the environment or the vehicle's ego-motion. The typical example is an incremental rotary encoder. This type of sensor is attached to the motor shaft of the vehicle. The data about the angular position of the wheel is then provided. The information about speed and acceleration can be calculated from it.

Another example is an accelerometer, which measures proper acceleration. Linear velocities in all three dimensions could be computed. A gyro sensor measures angular velocity instead of the linear one. Both sensors are usually attached in the sensor called an Inertial Measurement Unit (IMU). The IMU is a useful sensor for obtaining vehicle odometry.

There are plenty of other useful sensors that can be used in algorithms solving the SLAM problem. – i.e., Magnetometers, bumpers, light detectors, acoustic sensors, etc. Using these sensors depends on the conditions in a particular application.

2.7 Available open source tools, implementations and datasets

There are plenty of implementations, tools, and datasets, which can be use within mobile robotics research instead of starting from scratch. This chapter is structured as follows. The first section contains useful tools for robotics. In the second section, the list of the state of the art implementations will be presented. The last part of this chapter presents the list of datasets that are available online.

2.7.1 Tools

In this section, several tools and frameworks will be described. An essential tool for experiments is the ROS framework [54],[55] because it supports a wide range of hardware – robots and sensors – and several systems for solving the SLAM problem.

ROS

ROS is a software framework composed of various ready to use tools and a robust mechanism for the communication between applications – and even between computers – and many drivers for communication with connected hardware. The mechanism is based on so-called ROS messages: data structures defined by a simple text file and then built by a building system. Built messages are then used as data types for real messages, used for communication between ROS applications, usually called ROS Nodes. The amazing part is that messages don't go directly from one application to another. They are published by application to the space called ROS topic. ROS topic has its unique name. For example, image data from RGB camera is usually in the topic with the name composed of camera serial number and the text image or image_raw – e.g., /realsenseimage_raw. Slashes are used for namespace creation so that more topics can be created in namespace /realsense/. Every other application that needs information from the topic subscribe to a particular topic and gets the data right after publishing the message by another application. The principle of ROS message system with one publisher and one subscriber is shown in Figure 2.15.

The only important rule is that the ROS message type has to be the same for the publisher and the subscriber. ROS is robust against mistakes in the name of the topic. The Subscriber application doesn't crash. It just doesn't subscribe to a message.

ROS message mechanism runs in the ROS environment called the ROScore. It can run on one computer or on a couple of computers where one computer is ROS Master, and the others are slaves. ROS Master is the only computer where ROS core is running. The other computers have set their ROS variables ROS_MASTER_URI to the ROS Master computer, and it is then automatically connected after an arbitrary ROS node has started on the slave computer. The messages are automatically sent between computers based on publishers and subscribers attached to particular topics. It is a benefit for the developer to use more computers without creating the network layer for communication between them.

Another useful concept contained in ROS is a data file called ROS bag. It works as a record of the ROS environment. The file includes all topics and messages which were published at the time of recording of the file. It is a powerful mechanism to rerun applications with

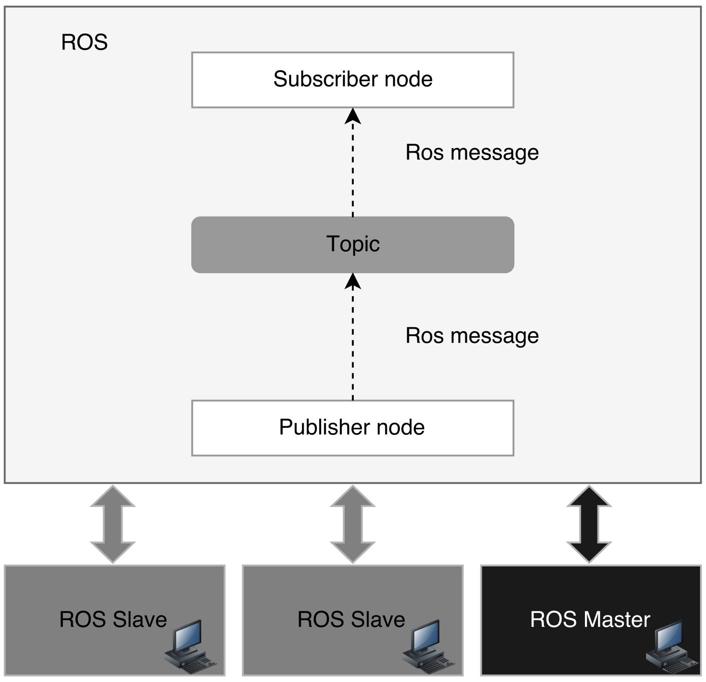

Figure 2.15: Principle of a communication inside ROS environmet.

different parameters on the same data multiple times.

Mobile Robot Programming Toolkit

Mobile Robot Programming Toolkit (MRPT) is a set of open-source C++ libraries and applications covering a wide range of algorithms and data structures in mobile robotics. The advantage is the existence of ROS packages that allow running MRPT applications inside the ROS environment. MRPT also contains libraries designed to create GUI of an application, OpenGL module for graphics creation, or vision module, which extends the functionality of OpenCV [56].

NVIDIA Isaac

Isaac is a relatively new platform determined for the development and deployment of AI-powered robots. It provides a collection of algorithms. Many of them are GPU-accelerated. It can be developed directly using C or Python API. Moreover, there exists a ROS-bridge to provide a connection with a ROS framework. Isaac is supposed to run on NVIDIA hardware – such as the NVIDIA Jetson computers family. Thus it can utilize hardware potential to the maximum.

Simulation

During the research in robotics, there are three possible ways to test an arbitrary algorithm. The first option is to run an algorithm on a real device or record data from an actual device. A better idea is to use an online dataset. Another option is to use a simulation environment. There are two well-known solutions with the active community.

The first solution is called the Gazebo [57][58]. It is a simulation platform that was a part of the ROS. The current version is the standalone application with libraries allowing use in the ROS. In older versions, the crucial disadvantage was the absence of visual-based model creation. The model was created only by the XML-based file called URDF. The current version has a GUI model editor and a lot of new features.

he second option is the Virtual robot experimentation platform (V-REP) [59]. It is a multiplatform simulation environment. The advantage is that V-REP supports a wide range of programming languages and control approaches. By control approaches, the following options are meant: the embedded script, plugin, ROS node, remote API client, or completely custom solution. The environment supports four physics engines. The whole list of features can be found on the project webpage[2] Nowadays, both solutions are an excellent choice to work in a simulated environment.

Visualization

In most applications, it is necessary to visualize the map and data from sensors. There are several options. The first one is to write a visualization application from scratch. The second option is to use tools for built-in ROS. It is a particularly rviz application that can visualize the map and a wide range of sensors. Extensions with new functionality can be used in this application. The third option is to use Robot Web Tools[3] [60]. It is a web-based framework capable of communication between ROS and web applications. The third option has several advantages. It is multiplatform, it is easy to learn, and it can be used as a web-based control panel for the robot.

2.7.2 SLAM Implementations

In Table 2.1, the list of available open-source implementations is shown. Implementations are sorted by used sensors. The second criterion is whether the particular implementation uses ROS or not. The implementation using ROS is often easy to use on arbitrary hardware.

An interesting question may be which SLAM solution from the particular group is the best one. There are few papers comparing implementations [69] or SLAM algorithms [31] directly. For example, a series of SLAMBench ([70], [71] and [72]) papers were published. Another example is paper [19] in which authors compare gMapping and Hector SLAM in the tunnel scenario. The gMapping is an older solution, but thanks to integrating information about the movement by particle filter, it creates a more accurate map of the long, almost straight

[2]http://www.coppeliarobotics.com/
[3]http://robotwebtools.org/

Approach	**Reference**	**Sensors**				**ROS**
		LiDAR	**Camera**	**RGB-D**	**IMU**	
tinySLAM	[61]	✓	✗	✗	✗	✓
GMapping	[50]	✓	✗	✗	✓	✓
Hector SLAM	[41]	✓	✗	✗	✓	✓
MonoSLam	[62]	✗	✓	✗	✗	✗
OrbSLAM 1, 2	[12], [34]	✗	✓	✗	✗	✓
LSD-SLAM	[11]	✗	✓	✗	✗	✓
FAB-MAP	[63]	✗	✓	✗	✗	✗
PTAM	[5]	✗	✓	✗	✗	✗
DTAM	[64]	✗	✓	✗	✗	✗
DPPTAM	[65]	✗	✓	✗	✗	✓
KinectFusion	[66]	✗	✗	✓	✗	✗
ElasticFusion	[67]	✗	✗	✓	✗	✗
DynamicFusion	[68]	✗	✗	✓	✗	✗
RTABMAP	[42]	✓	Stereo	✓	✓	✓

Table 2.1: Available SLAM implementations.

tunnel. As we proposed in the paper [73], the ROS is an appropriate tool for creating a benchmark tool for the wide range of SLAM implementations.

2.7.3 Using of SLAM solving systems

Several SLAM solving systems were tested to learn how they work and how to use them to get accurate results. Significant advantages have the systems which run in the ROS environment. Its use is much easier because it is connected to the ROS message system. Thus, it is possible to work uniformly with these systems.

This uniformity is evident in the parameters of the SLAM solving systems. For example, lidar-based systems have the same parameters for configuration of transformations (tf) between coordinate frames. The robot usually has several coordinate frames:

- **base_frame** Coordinate frame of the robot base.
- **map_frame** Coordinate frame of the map.
- **odom_frame** Frame attached to the odometry system.
- **laser_frame** Frame attached to the laser sensor.
- **camera_frame** Frame attached to the camera.

As mentioned, transformations between coordinate frames have to be defined and published into the ROS environment. Particularly it is necessary to define tf between laser sensor frame and robot base. Equally important is the tf between the robot base and the map.

All mentioned parameters are usually set in the so-called launch file, the XML file capable of running multiple ROS applications. Moreover, it is possible to create several launch files for testing different configurations of the system.

In Figures 2.16 and 2.17, an example of maps and trajectories created by gMapping, Hector SLAM, and Google Cartographer are shown. MIT Stata center indoor dataset [74] was used. It contains the ground truth of the robot positions during the mapping task. Results can be used to compare the accuracy of used SLAM systems. It is visible in Figure 2.17 that Hector SLAM is not accurate when it is mapping long corridors[4]. The estimated corridor in the center of the map (red rectangle) is shorter than the ground truth version. Other systems are more accurate at that place. On the other hand, all systems were accurate enough in this scenario.

2.7.4 Datasets

The list of available public datasets suitable for testing SLAM approaches is shown in Table 2.2. There is information about the type of date which is contained in the particular dataset.

[4]Maps and trajectories were created in cooperation with undergraduate student Petr Štrunc during work on his bachelor thesis.

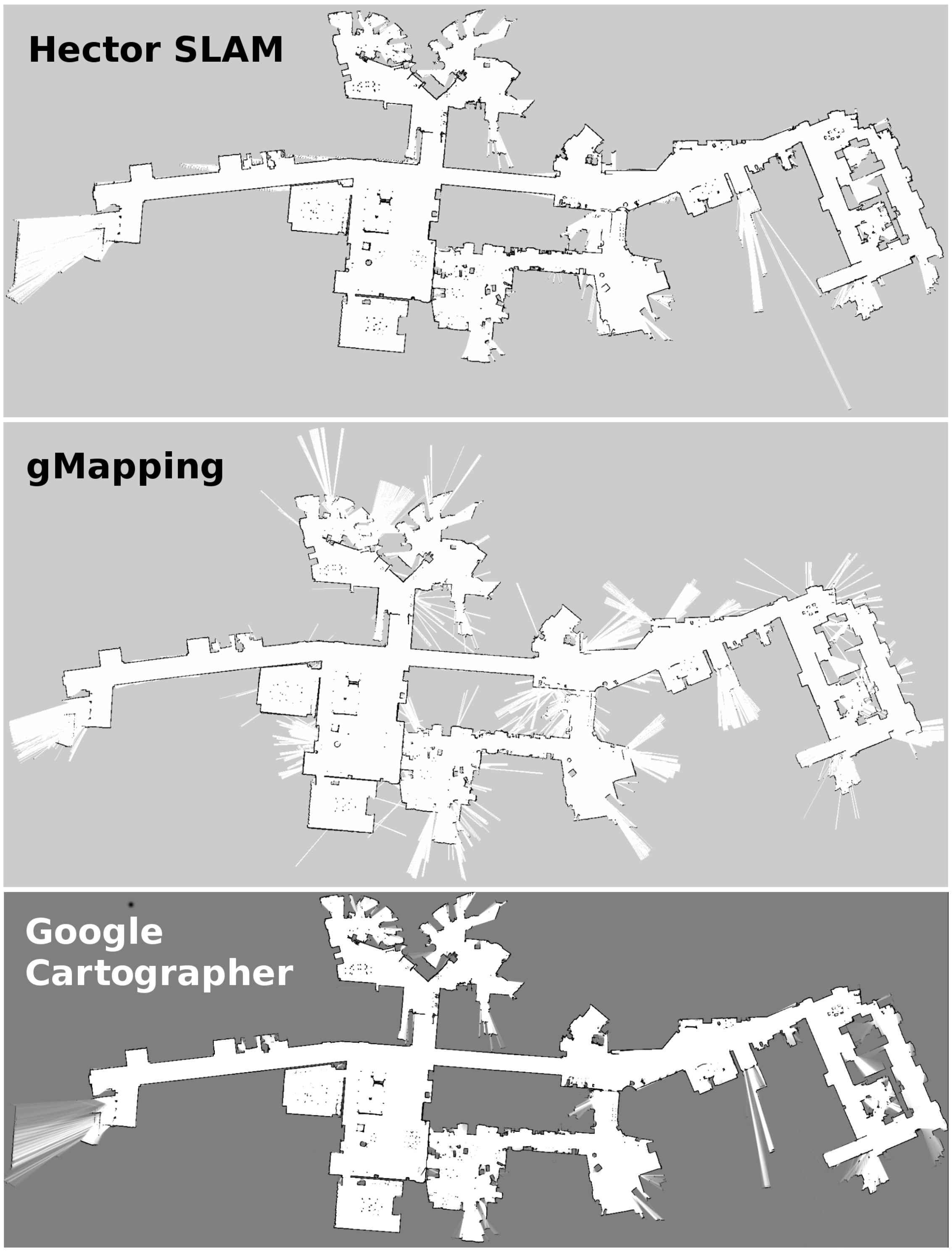

Figure 2.16: Map of MIT Stata Center created by three SLAM solving systems.

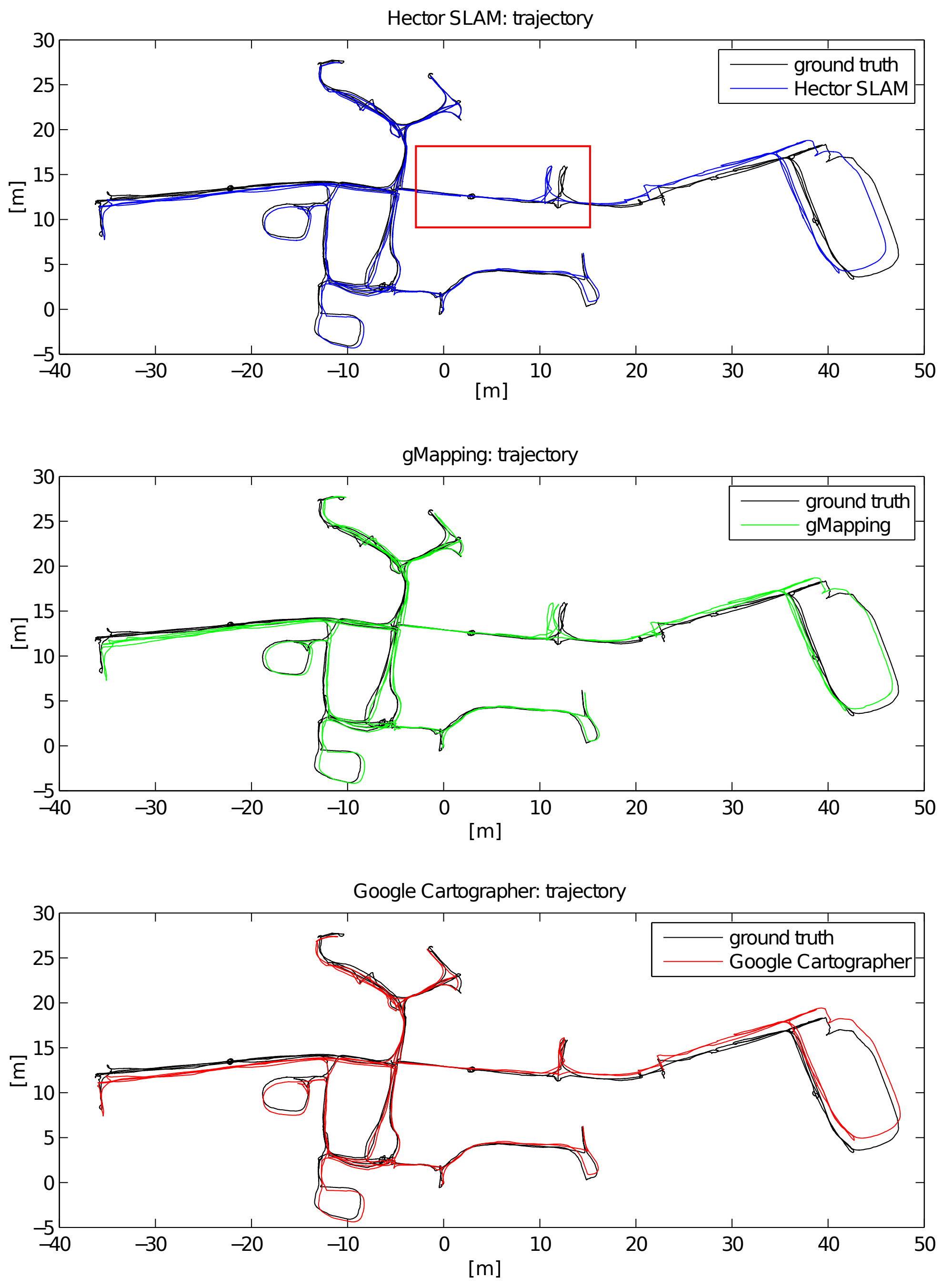

Figure 2.17: Trajectories of robot in the map estimated by three SLAM solving systems.

Dataset	**Reference**	**Sensors**			
		Lidar	**Camera**	**RGB-D**	**IMU**
Navlab SLAMMOT	[75]	✓	✓	✗	✗
KTH SLAM	✗[a]	✓	✗	✗	✗
Radish	[76]	✓	✓	✗	✓
TUM[b]	[77] [78]	✗	✓	✓	✗
Cheddar Gorge	[79]	✓	✓	✗	✓
MIT Stata center	[74]	✓	✓	✓	✓
Kitti	[80]	✓	✓	✗	✗
Velodyne SLAM	[81]	✓	✓	✗	✗
MRPT repository	[82]	✓	✓	✓	✓
ASL Repository	✗[c]	✓	✗	✗	✗
NYU DEPTH	✗[d]	✗	✓	✓	✗
Event Camera	[83]	✗	✓[e]	✗	✗

[a] http://www.nada.kth.se/ johnf/kthdata/dataset.html
[b] Technische Universität München
[c] http://projects.asl.ethz.ch/datasets/doku.php
[d] http://cs.nyu.edu/ silberman/datasets/
[e] RGB + Event camera data

Table 2.2: Available SLAM datasets

In the tables in this section, a couple of the SLAM implementations and datasets is shown. GitHub[5] is the source of many implementations, but the list above contains only the well-known SLAM approaches with research papers in their backgrounds. In the next section, the open problems of the current research of the SLAM problem will be described.

2.8 State of the Art

In this section, the state-of-the-art research of the SLAM problem based on the core approaches described in Section and sensors mentioned in Section 2.5 will be described. The section is divided into two parts. The first part is focused on the non-vision algorithms. Papers focused on the vision-based SLAM solving algorithms are mentioned in the second part of the chapter. Usually, non-vision solutions use a laser range finder and odometry. The camera is used in the vision approaches as the primary source of the data.

2.8.1 Non-vision

Non-vision methods were the first solution because computers weren't capable of processing images in real-time. The SLAM package of Tim Bailey[6] became a valuable material for beginners in the field of the SLAM problem. The package contains simple 2D simulators for the EKF-SLAM, UKF-SLAM, and both the fastSLAM algorithm versions. The code is written in Mathworks MATLAB. In the next paragraphs, the most important algorithms will be briefly described.

The solution which is worth to be mentioned is TinySLAM. It is the SLAM solution, which is unique because the implementation has less than 200 lines in the C programming language. It is composed of two operations. The first one performs the distance calculation between a laser scan and the map. The second operation is an update of the map. The stand-alone version of the algorithm is proposed in the mentioned paper. It uses the Monte-Carlo algorithm [84] for matching a new scan with the map. Implementation is described in detail in [61]. Furthermore, it is possible to use TinySLAM with a particle filter algorithm to increase the method's accuracy.

The well-known method called gMapping was presented in [49], [50]. It is a particle filter-based approach with an open-source implementation available. The authors came with two improvements to the particle filter algorithm. The first improvement is using the accurate proposal distribution, which considers both the robot's movement and the most recent observation. The second improvement is an adaptive resampling technique. It is based on the decision of whether or not it is necessary to apply the resampling phase. The decision value is computed using an equation

$$N_{eff} = \frac{1}{\sum_{i=1}^{B} (\omega^{(i)})^2} \tag{2.57}$$

where ω is a weight of the normalized particle i and N is the number of particles. The value is the so-called Effective Sample Size, and it estimates how well the particle set represents the target posterior. In the proposed method, the resampling step is performed when the value of

[5]http://github.com

[6]https://openslam.org/bailey-slam.html

N_{eff} drops below the threshold defined by number $N/2$. An implementation of this method is available online, and it is frequently used in current robotics applications. Moreover, in some scenarios, it has better results than newer solutions.

A more current SLAM solution developed at the Technische Universität Darmstadt in Germany is called Hector SLAM [4] [41]. This method combines a 3D attitude estimation system based on inertial sensing like IMU or wheel odometry with a 2D SLAM based on the graph-SLAM algorithm. The overview of Hector SLAM is shown in the Figure 2.18. The state of a 3D position is defined as follows

$$\mathbf{x} = (\mathbf{\Omega}^\top, p^\top, v^\top)^\top \tag{2.58}$$

where Ω is a vector composed of Euler Angles – roll ϕ, pitch $\times$, and yaw ψ. Variables p and v are 3×1 vectors of a robot pose and velocity. The state is estimated using the EKF filter algorithm. Inertial sensors contain noise, which causes an increase of the error between a true and an estimated position. The SLAM component is then used to reduce the error.

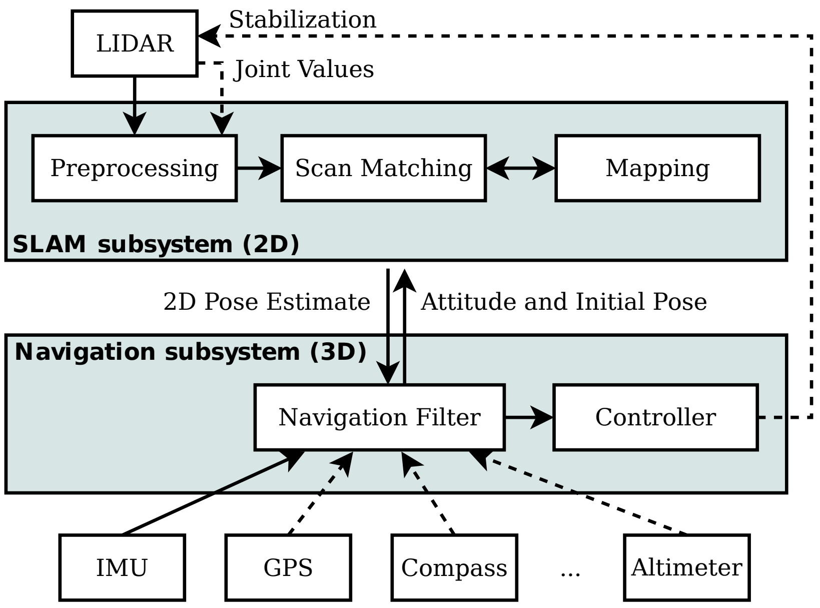

Figure 2.18: Hector SLAM overview.Taken from [4]

The second component of the approach is to create a map using the LiDAR sensor data. Scan matching algorithm based on the Gauss-Newton method is used. The advantage of the Gauss-Newton method is that there is no need to perform data association. The current scan is aligned with the map by the scan-matcher. The cooperation between SLAM and EKF is mutual. The position estimated by the EKF is the initial point of the scan-matcher algorithm.

Another well-known LiDAR-based SLAM system Google Cartographer (GC), was proposed in the paper of Hess et al. [52]. In Figure 2.19, the GC system overview is shown. The research's motivation was to create a system that makes a floor plan for a new building. GC creates a 2D grid map with an accuracy of approximately 5 cm. The approach is composed of two optimization components. The first one is the local component, and it performs optimization on the submap. Submap is a small set of aligned scans that contains information about a small portion of the environment. Every submap is finished when the robot moves a certain distance. There is an assumption that the submap is sufficiently accurate for a short time before it is finished. Scan matcher based on non-linear least squares is used. The optimization problem is defined as follows

$$\arg\min_{\xi} \sum_{k=1}^{K} (1 - M_{smooth}(T_{\xi} h_k))^2 \tag{2.59}$$

where M_{smooth} is the smoothed version of probability values in the local submap. The values represent the probability of obstacles in cells of the grid map. Variable h_k is a k-th point of the scan. Each point is transformed to the submap frame using rigid body transformation T_{ξ}, where ξ is the pose of the scan frame. The finished submap is put to the system's loop closure component, and no new scans will be inserted into it.

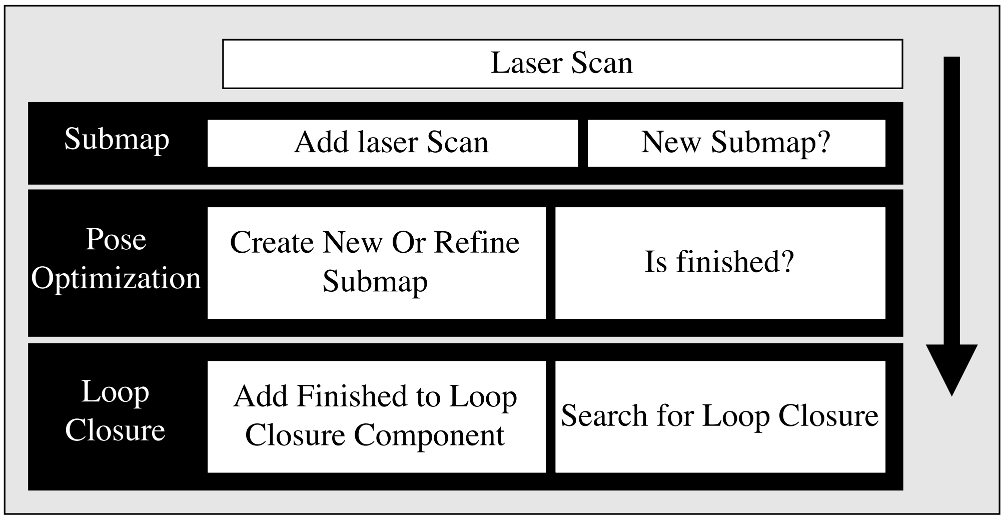

Figure 2.19: Google Cartographer overview.

The loop closure is a global optimization component. In this case, it processes all data. The goal is to reduce the error accumulated during movement in the environment. The local optimization does not reduce the error. In the global optimization, all pairs of scans and submaps are considered for loop closing. If the scan matcher finds a good match, the corresponding relative pose of the scan is added to the optimization problem. The global optimization component runs in the background. Every few seconds, it performs an optimization step – using Ceres solver [85] – defined as follows

$$\arg\min_{\Xi^m \Xi^s} \frac{1}{2} \sum_{ij} \rho(E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij})) \tag{2.60}$$

where Ξ^m are the submap poses ξ_i^m, and Ξ^s are the scan poses ξ_j^s. The variable ξ_{ij} represents relative poses between submaps and scans. Therefore, it is a set of constraints with the associated covariance matrices Σ_{ij}. Function ρ is a Hubber loss [86] used to reduce the influence of outliers in the data. The term $E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij})$ represents the residual which can be computed by

$$E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij}) = \left[\xi - \begin{pmatrix} R_{\xi_i^m}^{-}1(t_{\xi_i^m} - t_{\xi_j^s}) \\ \xi_{i;\Theta}^m - \xi_{j;\Theta}^s \end{pmatrix}\right]^T \Sigma_{ij}^{-}1 \left[\xi - \begin{pmatrix} R_{\xi_i^m}^{-}1(t_{\xi_i^m} - t_{\xi_j^s}) \\ \xi_{i;\Theta}^m - \xi_{j;\Theta}^s \end{pmatrix}\right] \tag{2.61}$$

where R and t are rotation matrix and translation vector that transforms local scan or submap frame to world frame and subscript Θ denotes the orientation of the scan or the submap. The problem defined in Equation 2.60 is called Sparse Pose Adjustment described in [87]. The optimization's quality is then improved using the Branch-and-bound scan matching algorithm described in detail in the mentioned paper. The paper results show that the method can successfully map indoor environments with high accuracy. An implementation of this approach is open-sourced, easy to use, and it is available online[7].

Sileshi et al. [88] recently present the implementation of an adaptive particle filter-based approach implemented on the FPGA. The approach is successfully tested on the simulated data and the dataset to speed up the particle filter approach. As mentioned above, the gMapping can be better than a newer approach such as Hector SLAM. Speeded up gMapping would be a good solution for a wide range of applications, even on low-performance hardware. Flat2D [89] is another system that using 3D LiDAR over the 2D map system. It can be useful in the scenarios of mapping of long corridors, the real-time created map is 2D, but in the postprocessing step, the map can be recomputed in a full 3D version. There exists research that uses an unusual type of sensors. An example of such a system is EchoSLAM, presented in the paper of Kreković et al. [90]. It uses a microphone to create the map based on the sound reflected from the walls.

2.8.2 Vision based approaches

Current camera-based SLAM algorithms are usually built on the top of the graph SLAM approach. It is generally called the KeyFrame approach in the visual SLAM terminology, and it is processed by a method called Bundle Adjustment[86]. On the other hand, the first approaches used filters such as EKF. Therefore, it is similar to non-vision approaches. A typical property of most of the solutions mentioned in the next paragraphs is the limitation on the smaller static scenarios. The limitation is not strict in the most recent systems, but there are still some problems in larger scenarios. On the other hand, it is a significantly better situation in contrast to the first approaches.

The first vision algorithms were based on the EKF. The first system capable of performing indoor SLAM in real-time was the MonoSLAM system developed by Davison et al. [91] [62] in 2007. In 2008 the UKF-SLAM [92] approach was successfully used inside the MonoSLAM. The MonoSLAM algorithm uses a constant velocity motion model to estimate the movement of the camera

$$\begin{pmatrix} \mathbf{r}^W_{new} \\ \mathbf{q}^{WR}_{new} \\ \mathbf{v}^W_{new} \\ \boldsymbol{\omega}^R_{new} \end{pmatrix} = \begin{pmatrix} \mathbf{r}^W + (\mathbf{w}^W + \mathbf{V}^W)\Delta t \\ \mathbf{q}^{WR} \times q((\boldsymbol{\omega}^R + \boldsymbol{\Omega}^R)\Delta t) \\ \mathbf{v}^W + \mathbf{V}^W \\ \boldsymbol{\omega}^R + \boldsymbol{\Omega}^R \end{pmatrix}, \tag{2.62}$$

where $\mathbf{r}$ is a 3 dimensional vector of linear position of the camera, $\mathbf{q}$ is a quaternion – 4 dimensional vector – of the angular position, $\mathbf{v}$ and $\boldsymbol{\omega}$ are linear and angular velocity vectors,$\mathbf{V}$ and $\boldsymbol{\Omega}$ are impulses of linear and angular velocities, and $\mathbf{q}$ is a quaternion defined by the angle-axis rotation vector $(\boldsymbol{\omega}^R + \boldsymbol{\Omega}^R)\Delta t$.

The map created by the MonoSLAM was featured-based and sparse. There were many limitations to this system. It was usable only for room size indoor scenarios with a limited

[7]https://github.com/googlecartographer

number of landmarks caused by the computational complexity of the EKF algorithm.

The important question is how to estimate the object's distance in the image and whether it is possible to estimate it during feature initialization. Usually, the object's distance is called the depth, and it is denoted by d. The first solutions were based on the delayed initialization of the feature. The depth was estimated from the robot's movement before adding a feature vector to the covariance matrix. The undelayed solution based on adding multiple features with different depths was proposed by Sola et al. [93]. The feature with the best estimation of the depth is the only one that survives. The other ones diverge, and then they are deleted. The problem is in the efficiency of this approach. With every new feature, multiple features are added to the covariance matrix, which causes an increase of the computational complexity – i.e., the size of the covariance matrix increase quickly. A better solution was presented in papers [94] and [95]. Instead of depth d, an inverse depth parametrization $\rho = \frac{1}{d}$ is used in the algorithm. The initialized feature vector

$$\mathbf{y} = (x_c \;\; y_c \;\; z_c \;\; \theta \;\; \phi \;\; \rho)^\top, \tag{2.63}$$

encodes the ray from the first camera position from which the feature was observed. Variables x_c, y_c, z_c are components of the camera location vector, θ and Φ are azimuth and elevation of the ray. Arbitrary feature coded in inverse depth parametrization can be transformed to the euclidean space when it satisfies linearity index constraint. Transform from inverse depth parametrization to 3-D point is defined as follows

$$\mathbf{x} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{\rho} m(\theta, \phi), \tag{2.64}$$

$$m = (\cos\phi \sin\theta - \sin\phi \cos\phi \cos\theta)^\top, \tag{2.65}$$

where X, Y, Z are components of the 3D-point vector, x, y, z are components of the inverse depth vector, and m is the unit vector pointing from point x_c, y_c, z_c to point X, Y, Z. Details of this approach can be found in [95]. More recently the inverse depth parametrization is used in RGB-D based system in the paper of Gutierrez et al. [96].

An adapted version of MonoSLAM was used in Castle et al. [97] to the task of augmented reality and object recognition. The Shi-Tomasi corner detector [98] was used in the original MonoSLAM. In this version of MonoSLAM, the Scale-invariant feature transform (SIFT)[35] detector and descriptor are used. Another adaptation was proposed in the paper [39]. The special version of the RANSAC algorithm called 1-point RANSAC was proposed in the paper to perform the sub-task of data association. The original MonoSLAM uses Joint Compatibility Branch and Bound (JCBB) instead of RANSAC. A more recent system based on the adapted version of MonoSLAM was presented in 2014 in the paper of Atashgah et al. [99]. The authors used MonoSLAM inside a virtual environment for testing aerial SLAM applications. Another example of a modern EKF based SLAM system is proposed in paper [100]. Authors combine the EKF approach with the graph SLAM in the task of visual-inertial odometry. Medical use of the MonoSLAM was presented in [25]. The algorithm of MonoSLAM is adapted to the problem of tissue mapping in surgery. The Maximally Stable Extremal Regions (MSER) [101] are used as features in this research.

In 2007, Parallel Tracking and Mapping (PTAM) was proposed by Klein et al. [5]. The PTAM system is based on the KeyFrame based SLAM approach. The overview of the PTAM approach is shown in Figure 2.20. Authors split tracking and mapping into two parallel tasks. There are restrictions on the size of the environment or workspace. Exploration of the environment wasn't supported. Therefore only a static scene is assumed. The map and the whole algorithm are initialized from a stereo pair of images using the 5-point algorithm [102]. In practice, a user has to move in the direction of a camera plane. The algorithm can to handle thousands of features, which is much better than dozens or hundreds in the MonoSLAM algorithm. The pose update is based on the minimization of the objective function of the reprojection error

$$\boldsymbol{\mu}' = \arg\min_{\boldsymbol{\mu}} \sum_{j \in S} Obj\left(\frac{|\, \mathbf{e}_j \,|}{\sigma}, \sigma_T\right) \tag{2.66}$$

where $Obj(, \sigma_T)$ is the Tuckey biweight objective function and σ_T a robust estimate of the standard deviation of the distribution computed from all the residuals. The reprojection error vector $\mathbf{e}_j$ is defined as follows

$$\mathbf{e}_j = \begin{pmatrix} \bar{u}_j \\ \bar{v}_j \end{pmatrix} - CamProj(exp(\boldsymbol{\mu}) E_{cw} \mathbf{p}_j), \tag{2.67}$$

where $\hat{u}_j$, $\hat{v}_j$ is the pixel location in the image, $exp(\boldsymbol{\mu})E_{cw}$ represented changes in camera pose, and p_j is the coordinates of the $j-th$ point in the map. The $CamProj$ is a pinhole camera projection function with a radial distortion model. Details can be found in [103]. The adapted version of the algorithm suitable to run on mobile phones was later presented in paper [104].

FAB-MAP [63], [105] is an appearance-based method for SLAM and loop closing. It is based on the Bag of Visual Words (BoVW) approach when revisiting the particular place in the map. The BoVW approach is based on the pre-trained vocabulary of visual words. The observed scene is then defined by the combination of visual words in the vocabulary. The vocabulary is trained from the features extracted from a set of images. The first step is to use some feature detector and descriptor such as SIFT [35] or Speeded-up Robust Features (SURF) [29]. The result is a big set of vectors of a particular length based on the used feature description method. Extracted features are clustered to N classes by a clustering algorithm such as k-means [106]. The variable N defines a number of visual words in the vocabulary. Observation of the scene is defined as the binary vector Z of length N. The vector is defined as follows

$$Z_i = \begin{cases} 1 & \text{if the ith word is observed in the scene} \\ & 0 \quad \text{otherwise} \end{cases}. \tag{2.68}$$

The authors of this particular approach study the influence of the co-occurrence of some words in the scene. The influence proved to be a significant property of the word in the vocabulary. They proposed an approach based on the trained tree-structured Bayesian network [107] to capture the posterior density function of the co-occurrences of the words in the vocabulary. Chou Liu algorithm [108] is used in the FAB-MAP approach. In paper [109], the authors proposed an improved FAB-MAP 2, capable of working in environments with large maps. The improvement is based on the inverted index data structure, which provides the mapping from words to scenes in which the words were observed. The FAB-MAP 2 approach was successfully tested on the large scale dataset contained approximately 1000 km.

A solution for monocular SLAM called ORB-SLAM was proposed in the paper of Murray et al. [12]. The overview of ORB-SLAM is shown in Figure 2.21. It is one of the best solutions for small and medium-sized environment. The approach is based on the Bundle Adjustment algorithm. It is capable of recognizing loop closing and provides relocalization

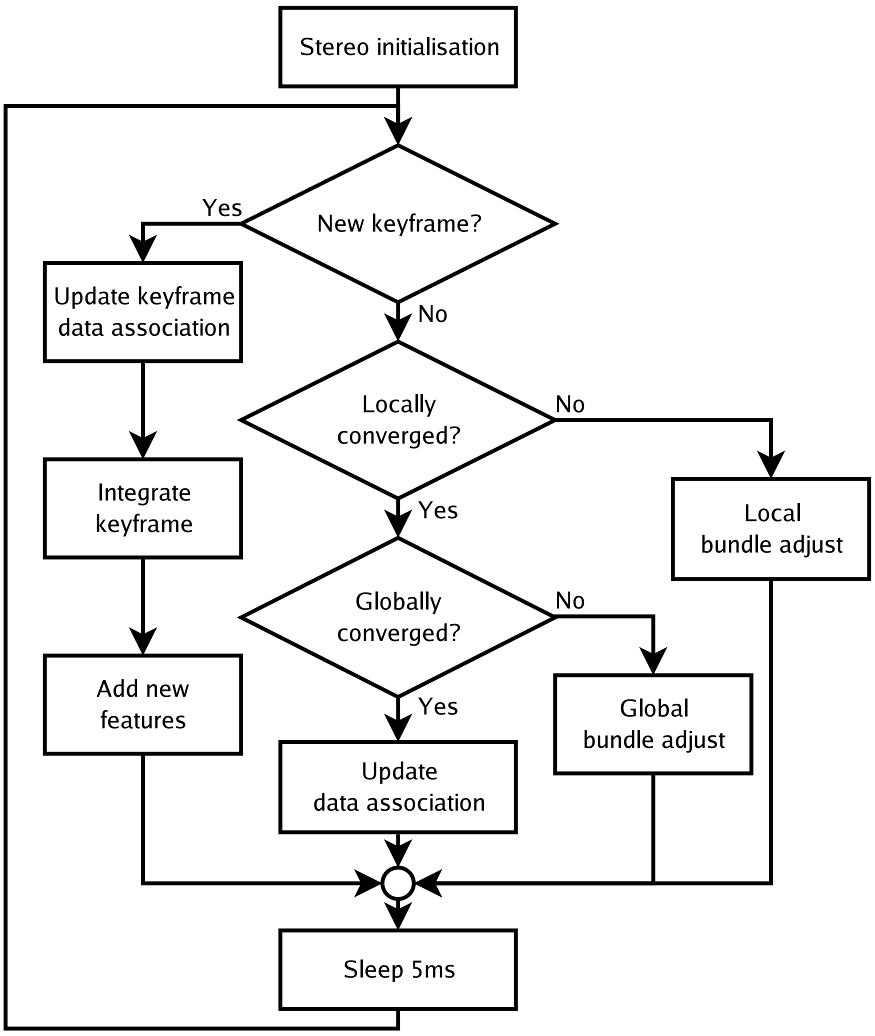

Figure 2.20: PTAM system overview. Taken from [5]

when the algorithm started over the existed map. An interesting property is using the same features to all important steps of the algorithm. The Oriented Fast and Rotated Brief (ORB) [30] features are used for tracking, map, relocalize, and loop closing. The name of the ORB features is based on the used combination of a feature detector and descriptor. The FAST [110] method is used as a detector. The descriptor is an acronym for the Binary Robust Independent Elementary Features [111]. For this purpose, the BoVW vocabulary based on ORB features is used inside the algorithm. The third-party library called DBOW2 [112] is used there.

The approach is unique in the way of initialization of the map. During the initialization process, the right method is chose based on the planarity of the scene. If the scene is planar, the homography between consecutive frames is computed. The fundamental matrix is calculated in the opposite case.

The original algorithm was improved in 2016 to work with stereo vision and RGB-D cameras. The approach is called ORB-SLAM2 [34]. The important change was the dependency on using the Robot Operating System (ROS) – see Chapter. Using ORB-SLAM2 in ROS is optional,

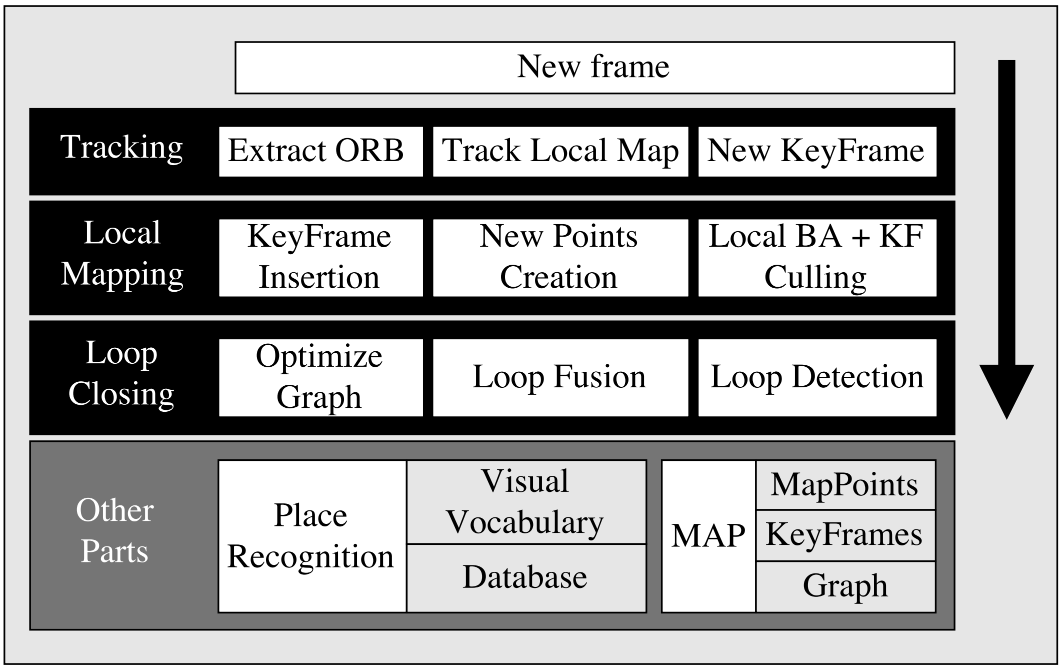

Figure 2.21: ORB-SLAM system overview.

but the original ORB-SLAM can run only within the ROS. The most current research in ORB-SLAM based on the information from the project website[8] and GitHub repository[9] is the application of ORB-SLAM2 in the task of augmented reality. The authors published a simple demo in which the cube is rendered into the scene. The second innovation is the support of current versions of computer vision (OpenCV 3.x) and linear algebra (Eigen 3.3) libraries. The consequence is that the installation and use of the ORB-SLAM2 become much more straightforward on current Linux systems. The method has an active community of users and researchers that works on the improvement of the method.

Another recently presented or improved methods solving Visual SLAM or visual odometry task are ORB-SLAM Atlas [113] with support for multi-map SLAM, ORB-SLAM 3 [114] with support of Visual-Inertial SLAM, Semidirect Visual Odometry (SVO) [115], COP-SLAM [116], [117], Direct visual odometry [118], image mosaicing system for UAV [119] or RatSLAM [120], [121]. The RatSLAM is a unique solution because it is a biologically inspired solution using a particular type of neural network called a competitive attractor network.

Another monocular SLAM system is the Large-Scale Direct SLAM (LSD-SLAM) [122], [11]. In contrast with feature-based ORB-SLAM, the LSD-SLAM represents direct methods. It doesn't work with features. It works with image intensities directly. The approach consists of three main components. The first one continuously estimates the rigid body transform between new camera images and the current keyframe concerning the last estimated pose. The second component is the depth map estimator responsible for the refinement of depth estimation of the subset of points in the frame. Therefore LSD-SLAM is a semi-dense approach. The last component is a map optimization subsystem. It is based on the graph

[8]http://webdiis.unizar.es/~raulmur/orbslam/
[9]https://github.com/raulmur/ORB_SLAM2

optimization framework g2o [123] which is available online on the web[10]

LSD-SLAM is computed only on the CPU. The approach was extended in the last years to handle stereo-vision [124], omnidirectional cameras [125], multicamera system [126] and smartphones [21]. The Smartphone version is only the odometry version of LSD-SLAM. The trajectory of the system is estimated but without the semi-dense map of the environment. Another approach for mobile devices was presented in [127]. It creates a 3D model of the small-scale environment in real-time using both CPU and GPU performance of the device. This approach is useful for 3D scanning rather than for the exploration of the environment. In Figure 2.22, the overview of LSD-SLAM is shown.

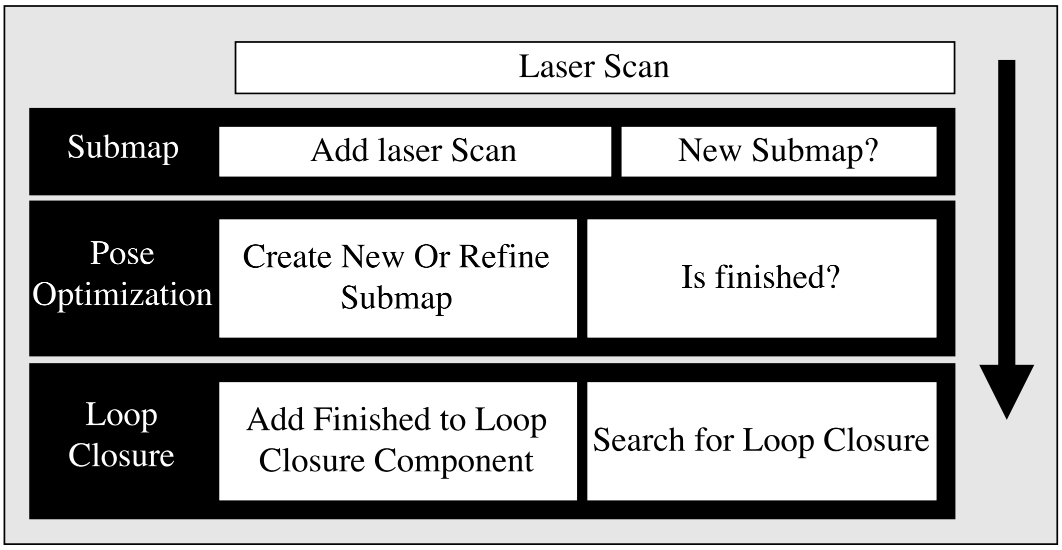

Figure 2.22: LSD-SLAM system overview.

There are more interesting approaches that create semi-dense and dense maps. Dense Piecewise Planar Tracking and Mapping [128], [65] creates both the semi-dense and dense map. Firstly the semi-dense map is created. Then the low-gradient regions are added to the map. They are assumed to be planar. The final map is dense [17]. In contrast, the work of Mur-Artal and Tardós [129] is a semi-dense approach build over the feature-based SLAM.

In the papers of Newcombe et al. [130], [64], the system named Dense Tracking and Mapping was proposed (DTAM). It is a system for dense reconstruction of a static scene from a single RGB camera. It is a direct method based on the graphSLAM approach. The implementation of DTAM is parallelized, and it's running on GPU. It is necessary because the dense representation of the scene may contain millions of vertices. The significant limitation of this approach is the assumption of the static scene with static light conditions.

Visual-Inertial Direct SLAM [131] is a current approach using an IMU sensor to improve the speed and accuracy of visual SLAM algorithms. Especially the scale of the environment is corrected by the IMU data. The same type of research was proposed in the paper of Leutenegger et al. [132]. The implementation of this research is available online[11] under the BSD license. An approach called MOARSLAM [133] is using IMU and camera too.

[10]https://github.com/RainerKuemmerle/g2o.

[11]http://ethz-asl.github.io/okvis/

MOARSLAM can handle multi-robot scenarios. This kind of problem is called multi-robot SLAM or co-op SLAM.

Since the Microsoft Kinect has been released, there is active research in the field of RGB-D based SLAM. The well-known solution named KinectFusion [134] [66] was proposed in 2011. The approach is processing all data to obtain the scene model of the room-sized environment.

The pipeline of the system starts with surface measurement. It is a pre-processing of raw depth measurements. The result of this component is a dense vertex map and normal map pyramid. The second component is a pose estimation part. The estimation is based on the multi-scale ICP algorithm [40]. It is an alignment between the predicted surface model and the new sensor data. The next component is responsible for reconstruction update. Integration of the surface measurement is integrated into the scene model using Truncated Signed Distance Function (TSDF) [135]. The last part of the system is called a surface prediction. The purpose of the component is to search for the loop closure.

Dense planar SLAM [136] is surfel[12] based system. There are identified two types of surfels regions. The first type is planar region surfels, which are characterized by low curvature. If planar regions overlapping and have similar properties, they can be merged into one region. The second type of surfels is non-planar regions. Data association management of map entities is easier in the surfel based system in contrast to the voxel-based system used in the KinectFusion.

The approach proposed in [67] named ElasticFusion is slightly different because it is based on non-rigid surface deformations used to refine the map rather than pose graph optimization method. The surfel-model is decomposed into two parts. The first one is an active part, a segment of the model that is observed, managed, and refined. The second inactive part of the map is the part in which the loop is searched. When the loop is closed, the particular inactive area is reactivated, and then it is refined to obtain a more accurate result. The recent research based on the ElasticFusion is called SemanticFusion. [27]. The authors used ElasticFusion and extended it to create a system that can recognize an object in the scene using deep neural networks. Another method that can handle non-rigid deformations is Dynamic Fusion of NewCombe et al. [68]. CPA-SLAM [137] is a new method that combines direct alignment of consecutive images with a global plane model expectation-maximization (EM) algorithm [138]. An interesting approach is PinPoint SLAM [139]. The method extracts features from RGB images using SURF features, and then it tries to find point correspondences of all types – 2D-to-2D, 3D-to-2D, and 3D-to-3D – using the RANSAC algorithm. It is a reason why the method is called a hybrid approach. Moreover, there is an off-line postprocessing phase of the method, which refines obtained results using all data and predicted poses from the on-line phase.

A special type of vision approach is solutions that using an object instead of low entities like points or edges. The well-known method is called SLAM++ [140]. It takes advantage of prior knowledge of repeated structures and objects in the scene. Objects are recognized by the algorithm and directly tracked to build a map. The more current solution was presented in the paper [141]. The method uses BoVW to classify objects based on a prepared dictionary that includes 500 recognizable 3D objects. Both approaches are suitable for small scale static scenarios. Another recent research in the object SLAM is Multi-object SLAM (MO-SLAM) [142] or SLAM with the object using a non-parametric pose graph proposed in the paper

[12]Surface element

[143].

Since 2016, many researchers have started to focus on deep learning in Simultaneous Localization and Mapping. Several recent papers describe methods addressing semantic segmentation SLAM such as closing-loop semantic segmentation research paper [144], deep-learning-based semantic segmentation [145] for masking out background, semantic-based motion removal for mapping in dynamic environments [146], SceneCode approach in paper [147], SemanticFusion approach [148], semantics and structure of the environment from single depth image in paper [149], or paper on semantic visual slam in the populated environment [150]. Some of the other papers are summarized in the survey paper [10].

Another direction that is built on deep learning and semantics is object level SLAM. There were some papers on this topic without using deep learning, such as previously mentioned SLAM++. Deep learning-based approaches, on the other side, were proposed in the last few years. Modern techniques were proposed in papers [151] describing Fusion++ approach, Quadric SLAM [152], where objects are represented as dual quadrics – i.e., 3D surfaces such as ellipsoids, or in [153], where the whole representation is obtained using a deep learning approach.

Moreover, professor Andrew Davison from Imperial College London – a recognized expert in visual SLAM who propose the first real-time monocular SLAM in 2007 – published two interesting papers focusing on the future of simultaneous localization and mapping. In the first paper called Future Mapping: The Computational Structure of Spatial AI systems [154]. Prof. Davison describes the evolution of SLAM into a more generic problem called geometric and semantic Spatial AI. He also explores the requirements and constraints of real application and finally explores the computational structure of this future SLAM. In the second paper [155], Prof. Davison argues for using Gaussian Belief Propagation (GBP) for probabilistic estimation in Spatial AI.

All of the mentioned papers are promising approaches in the field, but they usually focused on the semantics of the scene or handling with dynamic objects. On the other hand, there is no significant research in the field of multi-environment mobile robot missions – the system capable of recognizing transition between environments and adapting system behavior based on this information.

In summary, there are plenty of solutions in the field of SLAM. A lot of them were mentioned in the paragraphs of this section. Some others can be found in papers [156], [9], or [157], [10]. The mostly used non-vision solutions are gMapping, Hector SLAM, and Google Cartographer. Similarly, the best camera-based solutions are LSD-SLAM, ORB-SLAM2, and RTAB-MAP. In the next section, open problems of SLAM will be listed and described. Based on this list, dissertation goals will be defined in Chapter 3.

2.9 Open Problems

The SLAM problem is a highly researched area, but it still contains many open problems that need to be addressed. In this section, several open problems will be mentioned. The chapter is partitioned into two parts. The first one is focused on the problems of the SLAM itself. The second one is related to the applications of the SLAM problem.

2.9.1 Open Problems of the SLAM

Open problems mentioned in this section are problems of individual parts of algorithms for solving SLAM. A lot of mentioned problems are highly discussed nowadays. One of the topics that will be highly researched is semantic reasoning in the SLAM problem. It is related to the field of neural networks that recently became popular in computer vision.

The map managment

The result of many SLAM solving systems is an occupancy grid map or a landmark-based sparse representation of the environment. They are both well-known and suitable types of maps for the SLAM problem. There are few open issues that can be focused on in the next years. A high-level representation of the environments is one of them. Almost all known SLAM solving systems can handle simple point features, point clouds, or polygonal soups – a group of non-overlapping triangles. Only a few research papers came with the solution for describing objects in the environment, saving them to the map, and recognize them in the future. The system proposed in paper [153] is one of the more recent research on this topic. The question is how to describe a solid representation of the environment. The example of representation can be Parametrized Primitive Instancing, which decides about the shape family of the object – i.e., cylinder, sphere, rectangle – and then defines a set of the parameters for them.

The open problem from the same category is focused on the choosing of the optimal representation for the particular task. How to choose criteria? And even more important question: How to select the optimal representation automatically? The SLAM solution with the adaptive representation of the environment can be useful during the long-term mapping scenario. Is there any possibility to switch on-the-fly between a less or a more complex representation based on the complexity of the environment? Current systems are all dependent on the decision of the expert.

A similar situation is in the question of automatic parameter tuning. There are a lot of thresholds and other parameters that should be tuned to the particular scenario. Unfortunately, parameters must be tuned by the expert in the recent SLAM solving systems. Furthermore, correct tuned parameters may not be sufficient in the case of hardware or software failure. Modern systems don't contain any fail-recovery subsystem for re-establishing proper operation of the SLAM solving algorithm.

The static environment is often assumed in the state of the art SLAM solving systems. Future research should be able to handle the dynamic environment, interaction with human and deformable objects. The result of such research will be the system with the capability of creating of non-rigid and dynamic map with all information about moving objects inside. There is some interesting research in this area. For example, Pentland et al. [158], or Torresani et al. [159], but both require some prior knowledge or restrictions of the geometry.

The problem of map creation also concerns the size of the map. There is a lack of innovation in map maintenance in the meaning of memory load and performance requirements on the hardware of the computer. This problem is more frequent in visual SLAM, where the map grows quickly, and it becomes problematic. The goal of map management is to create an efficient map in an arbitrary scenario.

With the memory and performance requirements arise another interesting research area of the current SLAM problem. How to perform SLAM problem in real-time on low-performance platforms like mobile phones? Constrains created by HW can affect the communication between the robot and the environment or between robots in multi-robot SLAM scenarios. Optimization of communication is one of the crucial open issues which will be important in almost all robotics problems and not only for SLAM.

Semantic reasoning in the SLAM problem

A different group of open problems is focused on semantic reasoning. The goal is to classify places and objects according to a set of labels. This kind of problem is usually task-driven. The reasoning itself is task-dependent. It means that a different classification has to be used in various applications. Similarly, a different level of detail in the knowledge is used based on the task goal.

An interesting direction can be an organization of experience and knowledge of the robot. The robot should recognize pieces of furniture or different rooms in the building, and the robot should recognize the non-visible properties of objects. It is not only the decision about these properties but also the ability to do some action with the object based on its properties. For example, a humanoid robot should be able to sit on a recognized chair. And even more complex knowledge of connections between objects and their properties can be very useful in the SLAM solving system. For example, if the robot moves through a room full of people. It should assume that the walls are behind people and other objects even when the robot doesn't "see" them by its sensors.

There is some research in the domain of SLAM that uses Semantics as support information to improve the estimation of the map. The SLAM++ system mentioned above is an example of such a SLAM solving system. And in contrast, there exists the research of semantic reasoning, which is supported by SLAM. The research of monocular SLAM system which improves the performance of object recognition task of Pillai and Leonard [160] is a good example of this research direction. Another example is Pop-up SLAM [161] or SemanticFusion [27] SLAM system

The theory of the SLAM solving algorithm

It is necessary to mention the research focused on the SLAM problem theory. For example, the initialization for iterative nonlinear optimization in graph-based SLAM is a crucial task because it can improve the accuracy and efficiency of the algorithm. But even the best initialization process is useless in the case of convergence failure. The research on the convergence of optimization methods is still full of important questions. How to avoid the convergence to the local minima? There is a possibility to use convex relaxation in the research of Liu et al. [162] How to solve the problem globally? There is some research on this topic. For example in the paper of Carlone and Dellaert [163] authors use convex semidefinite programming (SDP) [164] to solve problem globally.

But there is a slightly similar problem as in the metric SLAM. The proposed approach can be optimal for that particular case but is it still optimal in the general case? Unfortunately, it isn't, and the generality of the solution is still one of the open problems.

Active SLAM

The robot is usually a platform equipped with a couple of sensors. In a standard SLAM algorithm, the robot process data but doesn't try to help the algorithm on purpose. There are few ways to improve the accuracy of the map by controlling the robot. One approach is to search places where the loop can be closed because it can significantly reduce the spatial error during mapping. This principle of minimizing the uncertainty of the map is called Active SLAM or Simultaneous Planning Localization and Mapping (SPLAM). It was addressed for the first time in the paper of Leung et al. [165].

In the active SLAM, the robot can improve the result by selecting suitable future action to ensure the high accuracy result. Recently, a popular approach is based on the selection from the finite set of alternatives. It has three steps. In the first step, the robot identifies locations suitable for visiting. In the second step, the best action is chosen from the set. And the last step is to take the selected action. Then the robot continues in the same action, or the action is terminated, and the process is repeated from the first step. Used theory for this task is Model predictive control [166] or Partially Observed Markov Decision Process (POMDP) [167]. The question is whether it is necessary to use an active SLAM approach in each time step or whether there is a possibility of switching between Active and standard SLAM. The advantages of standard SLAM are lesser memory requirements, but the accuracy will probably be lesser too. The active SLAM solution could help in the long-term operations where the error of mapping can increase a lot with the processed time steps.

Sensors

The research of new hardware is an essential part of the SLAM problem too. Particularly, the research of new sensors can be the source for new approaches to solving the SLAM problem. One of the discoveries of the last decade is called an event camera, and it is described in Chapter 2.5. This kind of sensor is probably a similar breakthrough as a 2D laser range finder, which allowed to create of robust SLAM approaches as [41]. As it first appeared in 2008, it is still a relatively new technology. The consequence is that there is a good possibility to improve algorithms working with such different data in contrast to the standard vision sensor. Few event-based SLAM systems were proposed in the last years. for example, event-based visual odometry (EVO) [168]

The goal here is to reduce the uncertainty of the sensors and to maximize the motion speed of the robot. The speed was always a problem during the SLAM. In classic cameras, the images start to be blurred, and then it would be impossible to detect structures in the image and recognize known objects. As the answer, an event camera, which can handle fast-movement without influence on the obtained data, can be mentioned.

Multi-agent SLAM

All single-agent SLAM open problems mentioned above are actual for multi-agent SLAM too. Moreover, there are a lot of problems which are related to communication and data sharing. Based on the paper [157] two problems can arise during a scenario in the communication between robots. The first one is called cyclic update. It is a situation in which the same measurement is used repeatedly. The second one is called out-of-sequence measurements. A situation in which some set of measurements from one robot is received in the wrong order by the second robot.

Data sharing is the second important topic in multi-agent SLAM. It isn't only about sending and receiving data from one robot to another. It is also the problem of the amount of data and deciding what is essential to sharing and how to minimize communication between robots. The related problem is how to perform merging of multiple local maps into the global map. Will the computation be done on a server or each robot individually? There are a lot of questions and open problems in the multi-agent SLAM problem. The open problems mentioned in the second part of this chapter are related to the practical applications of SLAM, and all of them are extendable to the multi-agent SLAM.

2.9.2 Practical applications

In the second part of this chapter, the group of open problems that arise in particular SLAM applications will be described. Some of them may overlap with the problems mentioned in the first part of this chapter. All mentioned problems are focused on the scenarios based on the UGV or UAV equipped with sensors.

Multi environment SLAM

The general solution to this problem can help in many scenarios. The problem lies in the situation where the robot has to operate in two different environments – typically indoor and outdoor – with a different set of suitable sensors and available technologies. Let's assume that the scenario contains a large outdoor environment of a parking place and a smaller indoor part of a parking house. The robot is supposed to control cars parked in both places whether they paid for the parking. While the GPS signal is available over the parking place, it is denied in the parking house. Some sensors may have a similar problem during the transition between the outdoor parking and the parking house. The crucial part is then to handle the transition between environments and, at the right time, step switch between algorithms designed for the outdoor and the indoor scenario.

Augmented reality for the unmanned vehicle or for a handheld device

The Augmented reality is another interesting topic that connected the SLAM problem with computer vision and computer graphics. There is much useful application such as inspection of utilities in the buildings to search for defects and abnormalities or indoor navigation for handheld devices. The smartphone is capable of obtaining short videos of the environment and a record of the data from accelerometers and gyroscope. This data can be analyzed, compared with the map of the building. The system should then be able to navigate the user to a particular place in the building by a visual aid on the display of the smartphone or by a set of voice instructions.

SLAM in cloud

This idea was mentioned in Chapter 2.5. It is a useful solution for a situation when a low-performance computer or handheld device is available. Still, we can send all data or some subset to the cloud and obtain the map of the environment. The computer or handheld device is only supposed to collect data and do some preprocessing to minimize data transfer with the cloud server.

Intelligent unmanned UGV for Search And Resque operations

A wide range of robotics tasks is contained in the SAR scenarios. The operation can be set in both indoor and outdoor environments. It can be performed by an arbitrary type of mobile robot. Furthermore, the SAR operation isn't only about localization and mapping in an unknown environment. It is also about an interaction with a human operator and about precise motion planning. Moreover, first aid and communication with the hurt person can be performed by the robot before arriving at the paramedic.

But the SAR operation isn't only about wounded people. The same kind of robot can be used by the police, the fire brigade, or the army to save human lives and detect potential danger in the environment.

UGV or UAV for an intelligent agriculture

Agriculture is an essential source of livelihood for many people around the world. Farmers are usually operated on several hectares of fertile ground. It is the big area and tasks like the security of filed against robbers or observing or harvesting the crop. These tasks can be expensive and difficult to handle by employees.

Better results can be obtained using robot platforms capable of moving in the field area and observing the crop periodically. Furthermore, sensors on the robot can provide some useful information about the environment. The temperature of the air or the ground and the average number of plants on the square meter can be computed by the robot and plants' mean height in a particular field area. All these tasks are grouped in the task of intelligent agriculture. Of course, it is possible to determine similar tasks in the industry and warehouse management.

Unmanned robot for transporting things and living organisms

The warehouse management mentioned in the previous paragraph is used for transporting objects from one place to another. The more difficult task can be transporting of a living organism or especially humans. From the SLAM view, it is the same task as the management of the warehouse. The task is unique in the problems which have to be solved during transport. For example, what is a good reaction of the robot – e.g., unmanned hospital bed – when the patient wants to stand up? The robot has to recognize this behavior and then trigger an alarm and stop. This kind of robot has to move carefully through the environment to avoid all static and dynamic obstacles and ensure the transported object's safety.

The problem of a closed door

There can arise many problems during transport of an object from place A to place B. For example, the robot's planned path in the building goes through several doors, and one door unexpectedly closed. In the best case, the robot would be able to open the door. Nowadays, the robot is not capable of opening the door. In that case, the robot should call for help from the operator or find another way to the goal place of the planned path.

The sensory network as a support of the indoor SLAM

The indoor SLAM can be deployed as a solution to many tasks. But there is always a problem of the uncertainty in sensors. The outdoor scenarios can operate with the support of GPS localization. There is no comparable alternative in the indoor SLAM. The only possibility of helping SLAM from the outside of the robot is creating a sensory network and then fusing obtained data. The sensory network can be composed of Wi-Fi hotspots, cameras capable of detecting the robot, or another sensor that can add information about the robot's location. As a bonus, the positions of devices of the sensory network can be estimated during the SLAM. I.e., the complete map of the environment with the positions of all devices and the robot can be created.

Multi agent SLAM with various agents taking different goals

Usually, the multi-agent SLAM is solved for the couple or the swarm of robots with the same goal. On the other hand, various kinds of robots supposed to meet different goals can be research areas filled with open problems. The utilization of this task can be found in arbitrary agriculture operations. There can be few security drones, several unmanned harvesters, and a couple of robots dedicated to clean leftovers. The common goal is then the flawless process in which the drones ignore harvesters in the security task. The cleaning machines wait until plants are harvested, and harvesters don't "harvest" the cleaning machine. And of-course, all unmanned robots will do their jobs as well as a human would do it.

Chapter 3

Disseration goals

In this chapter, the goals of this thesis are defined. They are chosen based on research made in previous chapters. The chapter is decomposed into two parts. In the first part, the motivation for the research is described. Consequently, the particular goals of this thesis are summarized. In the second part, individual tasks arising from defined goals will be described.

3.1 Motivation

As shown in Chapter 2.9, there are many open problems within the SLAM that should be solved. Moreover, some of the mentioned open problems are rather problems of mobile robotics instead of only SLAM. The problem I want to address in this dissertation thesis is using one-dimensional non-visual sensors to detect and classify the robot's environment. In other words, the use of sensors other than a camera to detect the transition between environments when a robot moves from one environment to another. – such as a move from the inside of the building to a park in front of the building.

Current mobile robot systems are usually developed for use in a specific environment – including SLAM systems that are typically tuned for robust and accurate results in a single static or dynamic environment. When the transition occurred, it can be useful to detect this transition to prevent system failure or adapt the robot's behavior. The first reason arises from the fact that some sensor or even SLAM system is accurate in the particular environment, but it can have worse results in a different one. For example, the LiDAR sensor can be mentioned. Two particular problems can occur when the robot moves from an indoor environment to an outdoor one. The first one is the problem of distance to nearest objects. Some LiDARs maximum distance can be only 4 meters. For example, it can be too short for outdoor environments. The second problem can be based on the technology of LiDAR (or depth sensor) because that can occur glare from the sun's rays, which can affect data.

The later mentioned reason can be shown in the example of a medical robot that moves between rooms in a hospital ward. Its behavior should be set differently when it is in the hallway, where it can act as a guide for people (both patients and visitors) and another when it is in the patient room. In this case, it can work as a nurse who reminds the patient of medication or measures his body temperature by a contactless thermometer. This example shows that the detection of the transition between environments can be helpful not only for the SLAM system's accuracy but also for the general mobile robot system. Naturally, it depends on the mission or application of the mobile robot. Similarly, the term environment should be understood in the resolution scale defined by the mobile robot's current mission.

The original motivation for this work was the paper [23]. The authors of the paper solve the transition between an indoor environment of a warehouse and an outdoor environment of the space in front of the warehouse. A different set of sensors is used in each environment. However, the solution is dependent on the tracking of the position of warehouse gate doors. The transition could be detected based on the strength of the GNSS signal as well. The problem is that the solution is dependent on the particular gate doors and the presence of GNSS signal – it is not general enough. The more general solution will be able to handle the transition between these two environments without detection of particular gate doors and without using any external sensors that are not joined to the robot.

In literature, there exist few papers such as [169] or [170] that focus on environment classification by camera data only. For the system that is supposed to work in real-time, it can be computationally expensive to process every frame from the attached camera. Thus, it should be useful to research the use of one-dimensional non-visual sensors to significantly reduce the computational cost of the environment classification system. Based on this hypothesis, the following goals can be defined to confirm it

- Investigation of the problem of using one-dimensional non-visual sensors for detecting the transition between two environments to allow a mobile robot to operate within multiple environments.
- Research one-dimensional non-visual sensors for their suitability to detect the transition between two environments.
- An analysis of data from these sensors concerning the environment change detection.
- An analysis of the camera-based approaches for classification of the environment of the robot.
- Design and implementation of the system for detecting and classifying the robot's environment.
- Prepare UGV equipped with all necessary hardware to record dataset for experimental validation of the proposed system.

3.2 Formulation of tasks behind defined goals

In this part, tasks behind defined goals will be introduced and described. Before describing individual tasks, it should be mentioned that they will be investigated on the special case of transition between indoor and outdoor environments.

The first task that arises from research one-dimensional non-visual sensors is an analysis of these sensors and analysis of methods for change detection in the signal data. Thus, it is necessary to search for sensors that change values when the transition between two environments occurs. As an example, the temperature sensor can be mentioned. The measured temperature value changes when the robot moves between two environments with significantly different air temperatures. Similarly, humidity or air pressure sensors can be mentioned.

Based on the results of change detection algorithms, it is necessary to analyze whether this information is sufficient to change the robot's behavior. In other words, it is necessary to investigate whether the decision based on the one-dimensional non-visual sensor has to be validated by information from the camera. The hypothesis is that camera-based validation (environment classification) is necessary because the change in the one-dimensional data can be caused by other reasons than the transition between environments. Thus, it is necessary to address the problem of environment classification from camera data.

It is a known problem called scene classification. In this thesis, the special case of indoor vs. outdoor classification will be addressed. It worth mentioning that it is usually solved as a problem of classification between many disjunct classes that are usually special its visual appearance. For example, airport images usually contain airplanes, medical rooms usually contain white beds and medical equipment. In the case of classification of the general environment into indoor or outdoor class, it should be mentioned that it can be harder because of visual appearance variability in both classes. To prevent misunderstood previous sentences, it is not a simple task to do scene classification into multiple classes. It is just an example of a situation when the classification into two classes can be more problematic than classification into more classes thanks to the overlap of a subset of indoor and outdoor scenes. The resulting task is to compare the state of the art approaches to indoor vs. outdoor scene classification on a sufficiently large dataset.

Previous tasks are the parts of the system that is mentioned in the next goal. The task is composed of a proposal of system design for environment detection and classification and its implementation. The design of the system will be inspired by systems proposed in papers [169] or [170]. Its implementation will be written in Python programming language for the ROS framework.

The next important step is to use the implementation of the system on real data. It should be mentioned that there is no available mobile robot dataset that also includes data from onedimensional non-visual sensors such as temperature sensor or humidity sensor. Thus, it is necessary to assemble a mobile robot that will record data from multiple sensors. This task consists of building the robot, developing its control software, and recording software. Finally, the system must be validated on recorded data, especially concerning computational cost, to confirm the hypothesis mentioned in the first part of this chapter.

Chapter 4

Environment Change Detection and Classification

The goal of this chapter is to introduce additional background for the practical part of the thesis. The chapter is organized as follows. In Section 4.1, Environment Change Detection (ECD), based on analysis of time-series data from non-visual sensors, is described. Section 4.2 is focused on a description of environment classification based on image data. Finally, in the last section, existing related research on multi-environment systems is described.

4.1 Environment Change Detection

The solution of ECD is based on the analysis of time-series data from non-visual sensors. Its goal is to detect that some change has occurred in the environment. It is worth mentioning that there is no active direct research of online change detection of the robot environment. Research is usually directed to other applications such as medical condition monitoring, speech detection, or human activity analysis. Thus, all methods mentioned in this section represent approaches to solving the general problem of the change or abrupt detection – both online and offline.

Approaches to ECD can be divided into two groups based on the time delaying of the sensor's value. In the case of non-significant time delay, the approach is straightforward because the sensor's value changes immediately when the change occurs. An example of this type of sensor is an ultrasonic distance sensor pointing to the room's ceiling. When the robot moves outdoor, the distance value is changed to the max range (e.g., 3000 cm) of the sensor – the sensor and a graph containing recorded data by the real robot are shown in Figure 4.1. It shows an example of the signal of an ultrasonic distance sensor measuring the room's ceiling height. Measured value changes immediately from approximately 250 cm to 3000 cm when the robot moves from the building to the outdoor environment.

Similarly, it changes back when the robot moves to the building. Moreover, the transition between room and hallway can also be detected because the door's height is lower than the ceiling height. Another example of the sensor without delay can be a binary detector of a magnetic field. Thus, the solution to ECD based on the sensors with non-significant time delay is the step-change detection.

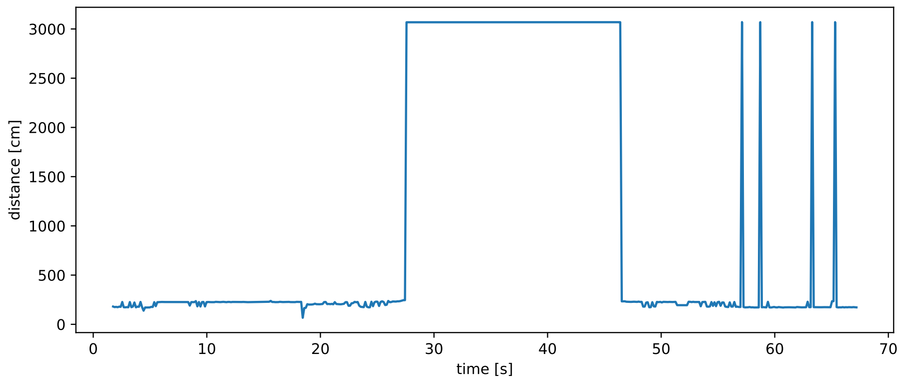

Figure 4.1: An example of ceiling height values during robots mission captured by ultrasonic distance sensor.

The more difficult case is the analysis of the time-series data with significant time delay. The value from these sensors changes smoothly without step changes. Examples of these sensors are temperature sensor or humidity sensor. An example of the temperature sensor data is shown in Figure 4.2.

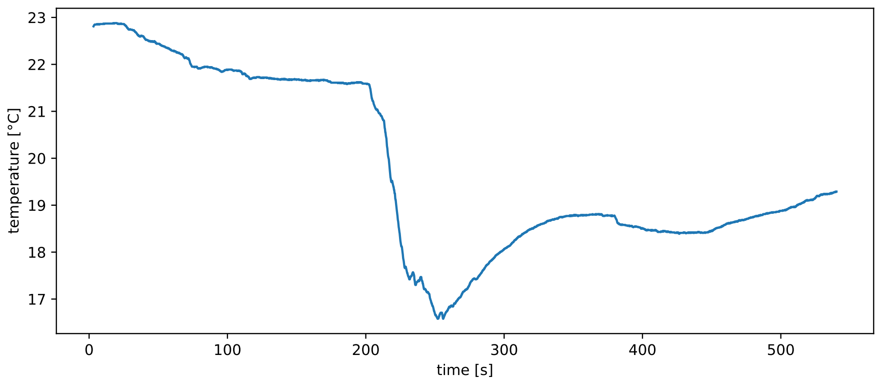

Figure 4.2: An example of temperature sensor signal during robots mission.

It is an example of a temperature (time-delayed) sensor signal. The robot moves from the building around timestep 200s. Then the temperature changes – continuously decreasing. When the robot moves back to the building around timestep 250s, the temperature value starts increasing. There are two things visible. The temperature value is not stabilized – still decreasing – when the robot enters back to the building. The second notable property is the speed of the change. The temperature outside was around zero at the time of this record. The sensor value falls quickly from 22 degrees of Celsius to 17 degrees of Celsius. But temperature increasing time is much longer when the temperature value increases back to 22 degrees of Celsius. There is a significantly smaller difference between the current sensor state and actual temperature in the environment. This property makes detection more difficult because the sensitivity of the detection algorithm can be out of the range of ongoing change.

The approach to change detection of this type of sensor is called Changepoint Detection (CPD) [171]. Methods for CPD searches for change points – i.e., a timestep of the beginning of the change – in a time-series data stream $S = \{x_1, x_2, \dots x_i, \dots\}$. CPD for time series in the time step interval $\langle m, n \rangle$ can be defined as the problem of testing hypothesis H_A – change occurs – and null hypothesis H_0 – no change occurs. It can be written as follows

$$\begin{array}{c} H_0 : P_{X_m} = \dots = P_{X_k} = \dots P_{X_n} \\ H_A : P_{X_m} = \dots = P_{X_k^*} \neq P_{X_{k^*+1}} = \dots P_{X_n} \quad where \quad m < k^* < n \end{array}, \tag{4.1}$$

where P_x is a probability density function of the sliding windows starting at x_m, and k^* is a change point.

CPD is usually assumed to be applied offline on the whole stationary time-series. In the robotic application, CPD has to be run online. Moreover, time-series is not typically stationary. It comes with several additional difficulties. The first one is that the online algorithm does not see the whole time-series – it has to detect change points base on the current signal state. The challenge is to detect it as soon as possible. Thus, in online CPD, there always be a delay between timestep when change occurs and its detection.

A lot of approaches mentioned below are supposed to work offline on the whole time-series. Fortunately, some of them can be used online on the most current data without future values knowledge. The principle of finding change points is usually based on event/anomaly detection or edge detection. In the next sections, there are described both supervised and unsupervised methods for CPD.

4.1.1 Supervised methods

Supervised approaches for CPD are based on learning a mapping from input data using target class information – machine learning classification approach. In other words, they are trained to find boundaries between individual states. States are binary (0 – no change occurs and 1 – change occurs)for stable signals. Moreover, it can be divided into several non-stable signal classes (e.g., 0 – signal is stable, 1 – signal increasing, 2 – signal decreasing).

As it was mentioned, standard supervised machine learning methods can be used in this case. For example, they are Decision Trees, Support Vector Machines, Bayesian Nets, Hidden Markov Models, Conditional Random Fields, or Gaussian Mixture Models. The problem with these methods is that they need a large amount of training data. Therefore, they are only suitable for CPD of two environments with stable properties – i.e., stable temperature, humidity, or air pressure. Unfortunately, it is not a common situation in real-world conditions – especially in indoor vs. outdoor classification where the outdoor environment's properties can change significantly during the time – e.g., hourly, daily, or seasonally. These properties make them not suitable for CPD in mobile robotics applications. More information about machine learning focusing on classification can be found in Chapter 4.2.

4.1.2 Unsupervised methods

In contrast to supervised methods, they do not need information from the teacher during a training phase. The goal of unsupervised methods is to search for change point based on the statistical analysis of the time-series data. They usually do not need a large amount of training data, which makes them more suitable for environment change detection. It is caused by the fact that these methods can handle different situations during robot missions without prior training for each situation. Unsupervised methods can be divided into several categories that are described in the following list.

Likelihood ratio methods are based on the assumption that the probability of two consecutive time intervals of time-series are the same when they belong to the same state. It usually consists of two phases. In the first one, the probability density of two consecutive intervals is calculated. Then, a ratio of these densities is computed. The most common method from this category is the Cumulative Sum (CUSUM)[172] approach. It accumulates deviations relative to the specified target of incoming measurements. Then, it controls the value against a threshold and indicates when the cumulative sum is over that threshold.

Another method is called Change Finder [173] which is based on outlier detection using a fitting with an autoregression model

$$x_t = \omega x_{t-k}^{t-1} + \varepsilon, \tag{4.2}$$

where x_{t-k}^{t-1} are previous observations, ω is a vector of constants, and ε is generated noise variable – usually white noise. At each timestep, the probability density function is calculated. Then, auxiliary time-series y is generated by giving a score to each data point from x. The new time-series represents differences in consecutive time series intervals. Finally, the score for each interval is computed by one of the following equations

$$Score(y) = -log\ p_{t-1}(y), \tag{4.3}$$

and

$$Score(y) = d(p_{t-1}, p_t), \tag{4.4}$$

where $d(.,.)$ is a distance function. The first equation is the average of the log-likelihood function and the second one is the statistical deviation function. A higher score of the interval indicates a higher probability of change point within this interval.

Previous methods are based on probability density estimation. A more straightforward approach is to estimate probability density-ratio between two consecutive intervals. An example of this method is the Kullback-Leibler importance estimation procedure (KLIEP)[174]. The method is based on the computation of Kullback-Leibler (KL) divergence:

$$KL\left[p(x) \mid\mid p'(x)\right] = -\int p'(x) log \frac{p(x)}{p'(x)} dx \tag{4.5}$$

The goal of KLIEP is solving importance estimation as a convex optimization problem – e.g., using the gradient projection method. Besides KLIEP, there are other methods based on the estimation of density ratio. Semi-Parametric Log-Likelihood (SPLL) change detector [175] is also based on KL divergence. For example, there is a family of approaches based on the Unconstrained Least-Squares Importance Fitting (uLSIF) method[176]. The uLSIF method is based on Pearson (PE) divergence instead of Kullback-Leibler divergence. There also exists relative uLSIF (RuLSIF) [177] with relative density parameter alpha used in PE divergence.

Subspace modeling methods are based on the representation of the time series using state spaces. Change points are then detected by predicting parameters of state space. There are two methods that worth mentioning. The first one is Subspace Identification (SI)[178], and the second one is Singular Spectrum Transformation (SST)[179].

SI works with a linear state space model

$$\begin{aligned} x(t+1) &= Ax(t) + Ke(t) \\ y(t) &= Cx(t) + e(t) \end{aligned}, \tag{4.6}$$

where A and C are system matrices, $e(t)$ is a system noise, and K is a Kalman gain. For each time interval, the SI method estimates the observability matrix. It is done by using LQ factorization and Singular Value Decomposition of the normalized conditional covariance. Diagonal matrix D from the decomposition is then compared to a predefined threshold.

SST method is based on a state-space model without system noise. It is based on singular value decomposition of trajectory – Hankel – matrix. Changepoint is searched by a comparison of the singular spectrum of consecutive trajectory matrices. This method is more sensitive to parameter choices than SI because of the absence of the model's noise.

Probabilistic methods compute the probability distribution of the new interval based on the observed data since the previous change point candidate. There are two main approaches that worth mentioning. The first one is Bayesian change point detection (BCPD)[180], and the second one is Gaussian Process (GP)[181] based.

BCPD is the first online method from this family. It is based on calculating so-called run-length distribution based on the Bayes theorem

$$P(r_t \mid x_{1:t}) = \frac{\sum_{r_{t-1}} P(r_t \mid r_{t-1}) P\left(x_t \mid r_{t-1}, x_t^{(r)}\right) P(r_{t-1}, x_{1:t-1})}{\sum_{r_t} P(r_t, x_{1:t})} \tag{4.7}$$

where r_t is a run-length variable representing time length from the last change point and $x_t^{(r)}$ is a time-series which started at last change point.

GP models time-series observations x_t using following equation

$$x_t = f(t) + \varepsilon_t, \tag{4.8}$$

in other words $x(t)$ is a value from Gaussian distribution function f with addition noise $\varepsilon = \mathcal{N}(0, \sigma_n^2)$. Function $f(t)$ is defined as a Gaussian Process distribution function with zero mean and covariance function K defined as follows

$$K(t_1, t_2) = \sigma^2 \exp\left(-\frac{(t_1 - t_2)^2}{2l^2}\right) \tag{4.9}$$

where t_1, t_2 are time steps and l is a scaling parameter. Given the time-series, GP estimates prediction of the distribution in time t using previous observations. The probability value is then computed from this distribution using current observation. The result is compared with a threshold value. In general, GP is a more complicated method but usually more accurate than BCPD.

Kernel-based method map series observations onto a higher-dimensional feature space and detect change points in that space. They are usually used in supervised learning, but several papers (such as paper [182] use kernels on unsupervised audio segmentation. It is based on testing the homogeneity of data in time-series using the past and present sliding window. CPD is based on comparing the kernel ratio – such as Kernel Fisher Discriminant Ratio (KFDR) – with a threshold value. This family of methods is usually sensitive to the choice of the kernel function.

Graph-based methods represent observations as a graph. Search for a change point in the graph using statistical tests. Graph-based methods [183] for CPD are non-parametric. A graph is constructed for each sliding window sequence. Nodes are observations, and edges are differences between observations computed using function Z_G:

$$Z_G(t) = -\frac{R_G(t) - E[R_G(t)]}{\sqrt{VAR[R_G(t)]}}, \tag{4.10}$$

where E and VAR are expectation and variance of R_G. Function $R_G(t)$ represents the number of connected points between different point groups. In particular between group from past interval $\tau \in \langle 1, t-1 \rangle$ and from the future interval $\tau > t$. Change point is detected when Z_G value is greater than defined threshold.

Clustering methods aggregate time-series into clusters based on defined states. Search for change point by searching for differences between clusters. There exists various approaches such as Sliding window and Bottom-up (SWAB) [184], Minimum Description Length (MDL)[185], Shapelet Method[186] or Model fitting[187]. They are all based on the fact that the change point is detected when the time series value in time t is assigned to different clusters than the previous value in time $t-1$.

As it was mentioned, unsupervised methods are more suitable for online CPD. In Chapter 6.1, the change detection experiment will be described and results will be discussed. The next section is focused on the related work in image-based classification of the environment.

4.2 Image-Based Environment Classification

As mentioned in the previous section, the change detection is not usually sufficient to classify the environment. Thus, a more sophisticated approach has to be done based on the image data from the camera. In other words, the approach can be named as the scene classification. In the first part of this section, supervised algorithms for classification will be summarized. The next parts of this section are mainly focused on the problem of indoor vs. outdoor classification. Approaches will be divided based on the classic machine learning classification schemes on Basic, Multi-scale, and Two-stage. Then, a neural nets based approach will be described. The end of the section will be focused on suitable existing datasets for indoor vs. outdoor classification.

4.2.1 Classification

The goal of the classification task is to assign correct category $\mathbf{y}$ to an input vector $\mathbf{x}$. The input vector $\mathbf{x}$ is composed of l features calculated from the original object – that is supposed to be classified – using particular feature extraction methods. It is crucial to choose a relevant feature extraction of the input data within the solved task. In image data, vector $\mathbf{x}$ can be, brightness histogram, texture description vector, or color description vector. The feature vector $\mathbf{x}$ can be denoted as follows

$$\mathbf{x} = [x_1, x_2, \dots x_l]^T . \tag{4.11}$$

The classification task is solved by the algorithm called classifier trained to map input $\mathbf{x}$ to the output $\mathbf{y}$ based on training set of $\mathbf{x}$ and $\mathbf{y}$ pairs. We can define training set as a matrix $\mathbf{X}$ of N rows, where each row of the matrix contains one feature vector $\mathbf{x}$. The second part of the training set is the vector $\mathbf{Y}$ containing correct class information – i.e., information from classifier "teacher" – for each feature vector on its rows.

The classification has two primary stages. In the first stage, the classifier is trained using pairs $\{x_i, y_i\}$ where $i \in \langle 1, N \rangle$ denotes row in $\mathbf{X}$ and $\mathbf{Y}$. In the second stage, the classifier is ready to classify new – unseen – samples. Then the classification process can be visualized, as is shown in Figure 4.3.

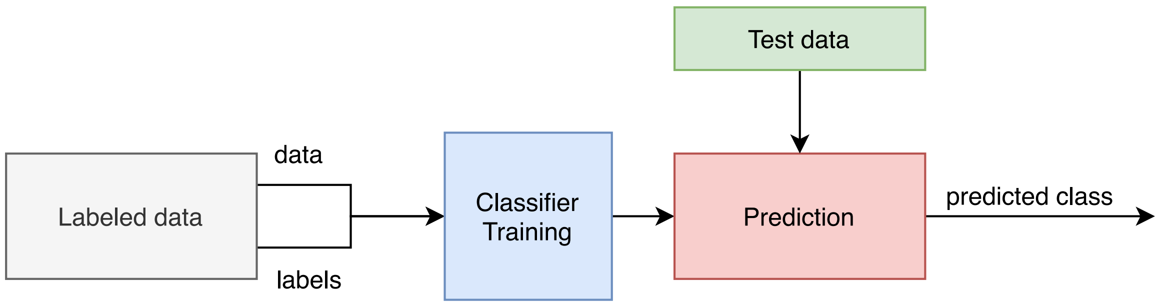

Figure 4.3: Supervised machine learning (classification).

Naive Bayes

The first mentioned classifier is Naive Bayes – more precisely it is a family of classifiers – based on the Bayes theorem with an assumption on independence between features in the feature vector. Thus, it is denoted as naive because this assumption is not usually fulfilled in the real world. Bayes theorem is defined as follows

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)} \tag{4.12}$$

where A and B are probability events. $P(A \mid B)$ and $P(B \mid A)$ are conditional probabilities – the likelihood of event A occurring given that B is true and vice versa. Probabilities $P(A)$ and $P(B)$ are marginal probabilities of event A and B respectively.

Naive Bayes classifies vectors of features, $x \in \{1, \dots K\}^D$. Variable K is the number of values for each feature, and D is the number of features. Thus, the feature vector has a total length $K \cdot D$.

In the context of classification, equation 4.12 will have the following form:

$$p(c_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid c_i)p(c_i)}{p(\mathbf{x})}., \tag{4.13}$$

where c_i is a particular class i. The denominator $p(\mathbf{x})$ does not depend on c – it is constant representing scaling factor. Thus, in practice, we can use only the numerator of Equation 4.13.

By applying the conditional independence assumption of the features and class labels on Equation 4.13, we get the following formula

$$p(c_i \mid \mathbf{x}) = \frac{1}{S} p(c_i) \prod_{k=1}^{n} p(x_k \mid c_i) \tag{4.14}$$

where S is a scaling factor. Equation 4.14 represents the Naive Bayes probability model. It is usually combined with a decision rule to get a Naive Bayes classifier. The most common is the maximum a posteriori (MAP) decision rule. It is defined as follows

$$\hat{y} = \arg\max_i p(c_i) \prod_{k=1}^{n} p(x_k \mid c_i) \tag{4.15}$$

Prior class probability $p(c_i)$ can be calculated by various techniques. For example, using fraction $\frac{1}{l}$ where l is the number of classes or using a ratio of class samples in training data concerning the total number of samples.

The second term $p(x_k \mid c_i)$ can be usually computed in three ways. One of the common ways is called Gaussian Naive Bayes, and it is based on the Gaussian probability density function.

$$p(x \mid c_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(v-\mu_i)^2}{2\sigma_i^2}} \tag{4.16}$$

Other possibilities are Bernoulli Naive Bayes, Multinomial Naive Bayes. For details see Chapter 3 in [188].

Decision Trees

A decision tree is a method based on a tree-like graph. During the decision process, it goes from the root of the tree through branches, which represents observation about objects. Finally, each branch ends by leaves representing class decision value (in the case of classification). Branches can also be understood as conjunctions of object features that lead to a particular class label.

The decision tree is built by splitting the training set – root is an entire training set – into subsets based on classification rules applied on features. This splitting is repeated until all items in the same subset has the same class label.

During years many algorithms for constructing decision tree was developed. The most used algorithms are Iterative Dichotomiser 3 (ID3), C4.5, Classification And Regression Tree

(CART), Chi-square automatic interaction detection (CHAID), and Multivariate adaptive regression spline (MARS). These algorithms use various metrics for constructing a decision tree. For example, ID3 and C4.5 uses Information gain, which is defined as follows

$$Ig(S,F) = H(S) - H(S \mid F) = H(S) - \sum_{t \in T} p(t)H(t), \tag{4.17}$$

where $H(S)$ is an entropy of set S, $H(S \mid F)$ is the entropy of all subsets created from set S by splitting based on the feature atribute F, T denotes all subsets t created by spliting S, $H(t)$ is the entropy of subset t and finally $p(t)$ is the ratio of number of elements in t with respect to number of elements in S. The best split is the one with the most information gain.

Another example – used in CART – is GINI impurity based on the value of probability $\sum_{k \neq i} p_k = 1 - p_i$ which represents a mistake in the classification of the item that belongs to class i. The last example of measure is Variance reduction – also used in CART –, but it is usually used in regression problems that are not in this thesis's scope.

K-Nearest Neighbors

K-nearest neighbors (k-NN) algorithm is based on the fact that similar objects usually have similar properties. In machine learning, it is possible to say that similar objects are close – in terms of distance – to each other in the feature space.

Based on the previous assumption, the new object added to the feature space is assigned to the most common class – majority voting – among k nearest neighbors of the object's feature vector. Constant k can be any positive integer number. In the case f $k = 1$ algorithm searches for only one neighbor with the minimum distance between feature vectors.

The training phase is based on storing all pairs $\{\mathbf{x}, \mathbf{y}\}$ in the feature space. The classification depends on the metric used for calculating a distance between feature vectors. There are many metrics that can be used for this purpose. The most common are, for example, Euclidean distance, Manhattan distance, Mean-Squared Error (MSE), or Hamming distance. In general, any suitable metric that compares feature vectors can be used.

Sometimes there is a problem with selecting class using "majority voting" because the distribution of features in feature space is not ideal. The solution is to weigh nearest neighbors by the inverse of the computed distance. It can avoid the wrong assignment – e.g., the situation for $k = 5$ when two closest objects in feature space are from class c_i and the three others from class c_j are significantly far away.

Support Vector Machine

Support Vector Machine (SVM) classifier is a classification method based on hyperplane construction in N-dimensional feature space. Hyperplanes are then used for the classification of the new point. The good hyperplane is as far as possible from the nearest points of any class in the feature space – there is a margin between hyperplane and points. Thus, the objective of SVM is to find a hyperplane between classes with maximum margin to points from classes. An example of a good hyperplane for points from two classes is shown in Figure

4.4[1].

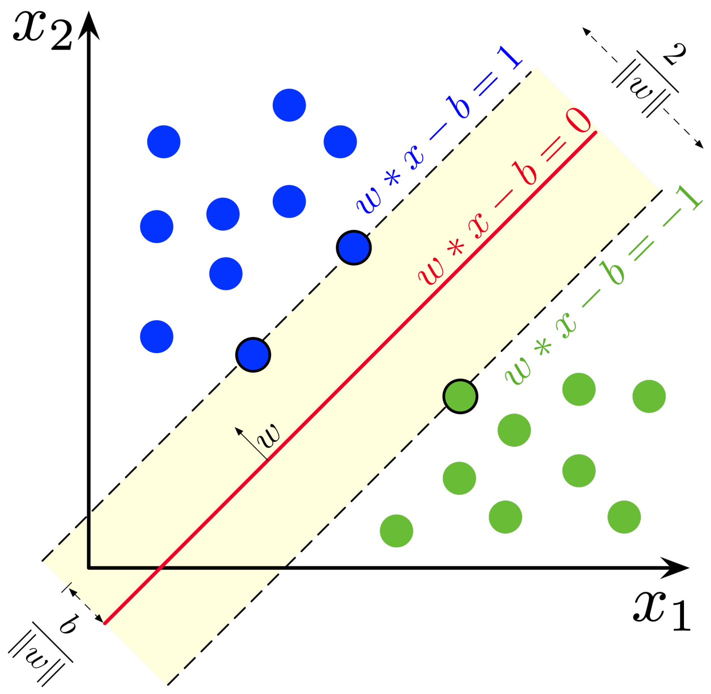

Figure 4.4: SVM hyperplane example for 2 classes

The hyperplane type depends on the number of features – i.e., length of the feature vector – in the feature space. For two features, a hyperplane is a straight line. Similarly, it becomes a plane for three features. Generally, for n features, $n - 1$ dimensional hyperplane is used.

SVM constructs hyperplane using so-called support vectors, which are a subset of points – from training set – in feature space that is hardest to classify – are nearest to the constructed hyperplane. The position of the dividing hyperplane will change if these points are removed. The finding of the optimal hyperplane can be solved using optimization techniques such as Lagrange multipliers. Support vectors are shown in Figure 4.4 as the points lying on the margin of the hyperplane.

Linear SVM

In this case, the hyperplane between two sets of points – points from two classes – is defined by equation

$$w \cdot x_i + b \geq 1 \quad when \quad y_i = 1, \tag{4.18}$$

and

$$w \cdot x_i + b \leq -1 \quad when \quad y_i = -1, \tag{4.19}$$

where w is a normal vector – usually normalized – to the hyperplane, $\mathbf{x}_i$ is a vector of i-th point in the feature space and b is bias vector and y_i is a label of class from the set $\{-1, 1\}$. The new point that satisfies one of these equations belongs to one class. In other words, each

[1]Taken from https://en.wikipedia.org/wiki/Support-vector_machine, the image is under the CC BY-SA 4.0 licence: https://creativecommons.org/licenses/by-sa/4.0/

point has to lie on the correct side of the margin. Equations 4.18 and 4.19 can be rewritten in the form

$$y_i \cdot (w \cdot x_i + b) \geq 1. \tag{4.20}$$

The goal is to minimize norm of the w to obtain solution closest to $y_i \cdot (w \cdot x_i + b) = 1$. Finally, the linear SVM classifier can be defined using the sign function as follows

$$\mathbf{x} \rightarrow sgn(w \cdot x_i + b). \tag{4.21}$$

Thus, new point $\mathbf{x}$ is assigned to the one class based on the sign of the formula $w \cdot x_i + b$.

Note that the definition above is suitable for a linearly separable set of points. In the opposite case, the hinge loss function can be used

$$max(0, 1 - y_i \cdot (w \cdot x_i + b)). \tag{4.22}$$

The function is 0 when Equation 4.20 is satisfied. Otherwise, its output is proportional to the distance from the margin. Finally, the goal is to minimize the following formula

$$\left[\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i \cdot (w \cdot x_i + b))\right] + \lambda \mid\mid w \mid\mid^2, \tag{4.23}$$

where λ is an influence parameter of the margin size. For sufficiently small λ problem change to the more simple solution in Equation 4.21.

Non-linear SVM
It is also possible to use SVM in the nonlinear case. An approach called the kernel trick has to be applied. The trick is based on applying a nonlinear kernel which map points in the training set to a higher dimension – e.g., from 2D to 3D space – in which the training set is separable – hyperplane can be easily found. The nonlinear kernel is used instead of dot products in Equation 4.23

A commonly used kernel is the Gaussian Radial basis function, which is defined as follows

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma||x_i - x_j||^2} \quad for \quad \gamma > 0, \tag{4.24}$$

element γ is usually set to $\frac{1}{2\sigma^2}$ where σ is a free parameter.

Artificial Neural Networks

Another approach to deal with the classification problem is to use artificial neural networks (ANNs). ANN is a system of connected components organized in layers called neurons. It is shown in Figure 4.5.

It is inspired by biological neural networks. Thus, the neuron is composed of inputs, body, and one output. Each part has a unique function in the neuron. Inputs supply signals and apply weights on each input. Body sum up all inputs. Finally output part performs the so-called activation function f. The scheme of a neuron is shown in Figure 4.6.

This scheme can be mathematically written as

$$f(\sum_i w_i^T x_i + b), \tag{4.25}$$

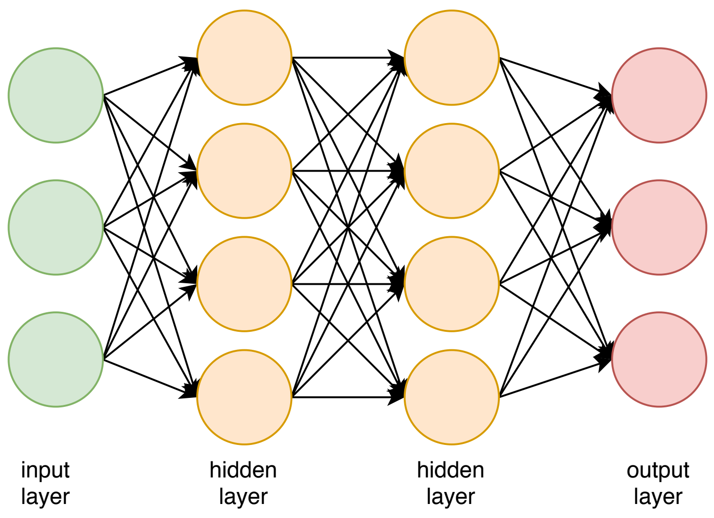

Figure 4.5: ANN scheme with two hidden layers

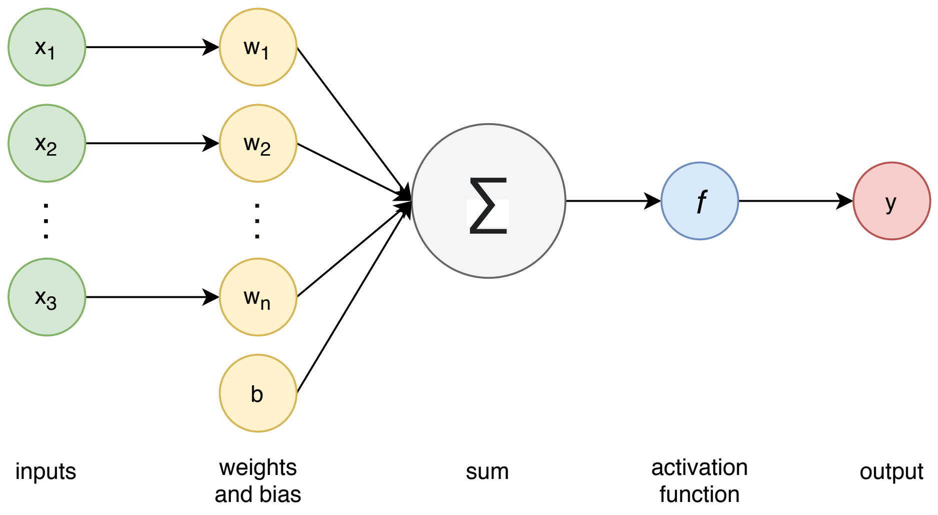

Figure 4.6: ANN neuron scheme

where x_i is i-th input of the neuron w_i is weight for i-th input of the neuron and b is a bias. Weights W defines how each input is important for the neuron – i.e., strength of the connection with previous neuron.

The activation function defines the final signal output of the neuron. The common activation functions are, for example, the Sigmoid function defined as

$$f(\xi) = \frac{1}{1 + e^{\xi}} \tag{4.26}$$

where ξ is an inner part of the function in Equation 4.25. It is visible that the output of this activation function is in the range between 0 and 1. n the other hand, the saturation can cause problems during the network's training phase for values of ξ that are close to limit values because the gradient is almost zero.

Another example of the activation function is Rectified Linear Unit (ReLU). It is defined as follows:

$$f(\xi) = max(0, \xi) \tag{4.27}$$

This activation function is probably the most popular one for its simplicity. Moreover, it has advantages connected to the convergence of the training process – stochastic gradient descent – which converges faster than other activation functions.

The third basic activation function that worth mentioning is Softmax. It is based on equation

$$f(\xi)_j = \frac{e^{\xi_j}}{\sum_N e^{\xi}_N}, \tag{4.28}$$

where j represents the fact that the equation is defined for j-th neuron in the neural network layer. Parameter N is the number of possible outcomes – usually the number of neurons in the layer. This activation function maps raw values from the neuron into a posterior probability value. Usually, it is used in the final layer of the neural network. Of course, there are many other activation functions such as Step function, Maxout, Leaky Relu, Tanh or Swish, and many more.

As it was mentioned at the beginning of this section, neurons are organized into layers. The first layer of the network is called the input layer. The size of this layer – the number of neurons in the layer – is based on the input data's size – e.g., $M \times N$ neurons for images of size $M \times N$ pixels. Similarly, the last layer is called the output layer. The size of the output layer is based on the number of possible classes in the classification problem. Between these two layers, there are N hidden layers of various types. The most common types are the Fully-Connected layer, where each neuron in the layer is connected to all neurons in the previous layer.

Another example is the convolutional layer, which has an essential role in Convolutional neural networks (CNNs). Each neuron in this layer is connected to a local region of neurons in the previous layer. This region's size is optional, and it defines a so-called receptive field of the actual neuron. The result is that it applies convolution on this receptive field, and the output is the response of the convolutional filter. Moreover, the depth of this layer can be greater than one. Thus, more neurons can be connected to the same region in the previous layer.

The next important layer is the pooling layer – usually Max pooling or average pooling – where neurons are also connected on a small region in the previous layer. The difference is that instead of convolution, it applies some function that performs a reduction in the size of data – reduces the number of neurons in the input dimension. For example, max-pooling connected to a region of 2x2 neurons choose the max value from this region and set it as the layer's output.

The last layer type that I mention is the loss layer. This layer is at the end of the neural network during the training phase. It is supposed to compute loss function L. Gradient of function L is used to optimize of network parameters during training. The most common loss function is cross-entropy loss computed as

$$L = -\sum_{i}^{N} p_i log \hat{p}_i \tag{4.29}$$

where N is a number of classes in the classification problem, and $p_i, \hat{p}_i$ are target and predicted probability distributions.

Another example of the common loss function hinge loss, which is defined similarly as in the case of the SVM classifier

$$L = \sum_{p \neq y} max(0, f_p - f_y + 1), \tag{4.30}$$

where f_p is predicted class value and f_y is a target class value.

The scheme of connected layers to the neural network is called neural network architecture. When it is constructed, the next step is to perform learning the network to become a good classifier for a particular problem. Learning is based on adjusting weights in neurons to reduce classification error – increase accuracy of classification. The goal is to minimize error as much as possible.

The learning is based on the back-propagation algorithm. It consists of three phases. The first one is the forward pass. In this phase, the sample from the training set is sent to the network, which computes the output – class prediction – with current weights and biases. Then the error – using selected loss function L – is calculated in the second phase. Finally, the backward pass of the network is performed using the gradient of the loss function L for individual neurons – its weights and bias – to optimize their values. Values are updated in the direction of the most significant gradient descent. To perform this update, Stochastic Gradient Descent, Momentum, Nesterov Momentum, RMSprop, or Adam are used.

The classification theory was summarized in this section. The next section addresses the problem of indoor vs. outdoor environment classification. In the first part of the section, classic approaches will be described. The second part is focused on a modern approach based on neural networks. Related work is mentioned in both parts.

4.2.2 Classic classification approaches

In this section, classification using so-called classic – except neural networks – will be described. It contains various methods for data description, various classifiers, and even various classification schemes. Firstly, three classification schemes are used in the related work of this particular problem will be introduced.

The basic classification scheme (Basic) is shown in Figure 4.7. IIt consists of three simple steps. Loading of the image – example image was taken from Miniplaces dataset –, applying descriptor to calculate feature vector of length N and performing classification using particular classifier such as SVM in the example.

This classification scheme can be modified by dividing the input image into $n \times n$ tiles and creating a feature vector by concatenating feature vectors of individual tiles.

The second type of classification scheme is called Multiscale. It is based on creating a pyramid by changing the resolution of the image and dividing each resolution on a defined number of tiles. An example is shown in Figure 4.8. F Feature vectors from the whole pyramid are concatenated into one feature vector. In the example, data are divided by changing the resolution and tiling of the image into a feature vector of length $21 \times N$, which is significantly longer than in the Basic scheme.

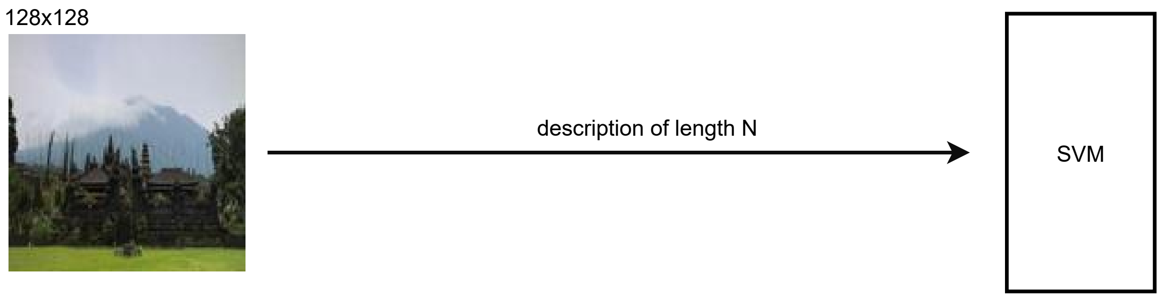

Figure 4.7: An example of Basic classification scheme used for indoor vs. outdoor classification.

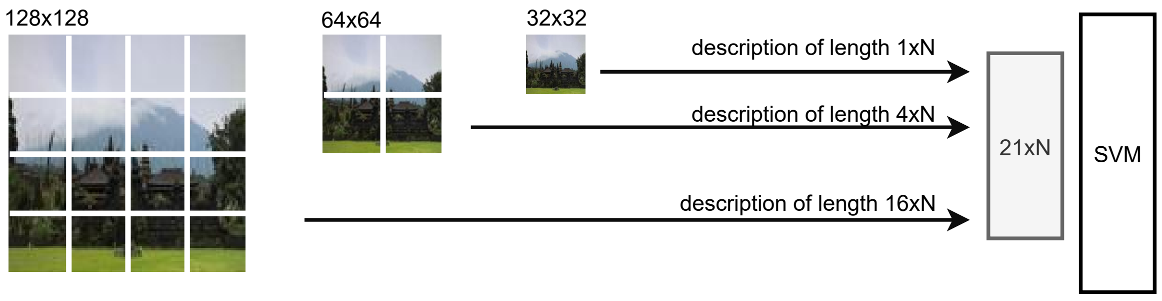
128x128
64x64
32x32
description of length 1xN
description of length 4xN
description of length 16xN
21xN
SVM

Figure 4.8: An example of Multiscale classification scheme used for indoor vs. outdoor classification.

The last classification scheme is the most complex. A scheme is called Two-Stage, and it is shown in Figure 4.9. In the first step, it creates tiles from the input image in a similar way as in previous schemes. Then it calculates descriptions of individual tiles. The important change is that there is also an individual classifier trained for each tile. In the example, there are in total 32 classifiers for 16 tiles created from the input data – 16 color classifiers and 16 texture classifiers. Finally, another classifier is trained using predictions from tiles classifiers. Thus, the size of the input vector for the last classifier is 32 bins.

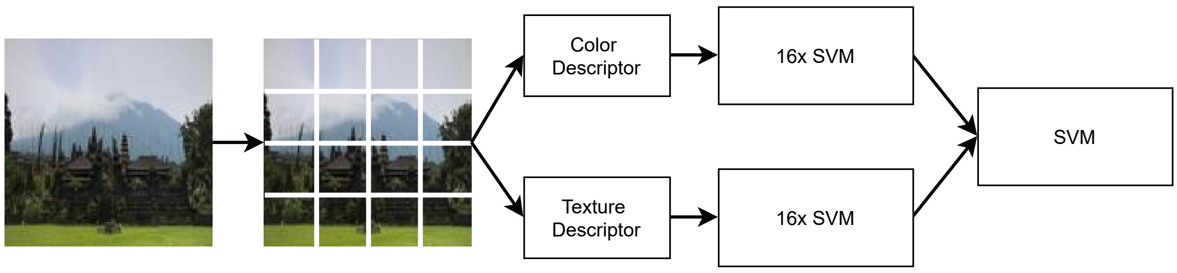

Figure 4.9: An example of Two-Stage classification scheme used for indoor vs. outdoor classification.

This scheme can be modified in three ways. The first one is to sum all indoor predictions and all outdoor predictions and use the vector of only two bins as input for the last classifier.

The second one changes the output of tiles classifiers. Instead of predictions, they took real numbers – e.g., distance from hyperplane – which should correspond to each classifier's level of certainty. Again it can be sent to the last classifier as a vector or summed up into two bins.

The last modification is based on the exchange of the last classifier for the majority voting approach. All tiles predictions are counted, and the final decision is based on the most occurrence value.

The next paragraphs, related work of scene classification – it includes some papers that address the problem of indoor vs. outdoor classification – will be mentioned. The first significant paper that addresses the problem of indoor vs. outdoor classification was the work of Szummer and Picard [6]. All experiments they performed were achieved on the Kodak dataset – see Section 4.2.4 They use a color and texture approach based on a Two-Stage scheme. They use kNN as a classifier. Instead of Euclidean norm, they use histogram intersection norm defined by the following formula:

$$dist(h^1, h^2) = \sum_{i-1}^{N} (h_i^1 - min(h_i^1, h_i^2)). \tag{4.31}$$

They tested RGB histogram and OHTA histogram as color description and a multi-resolution, simultaneous autoregressive model (MSAR) [189] as texture description. They discovered that OHTA color space (accuracy 73.2%), which is defined as follows

$$I_1 = R + G + B$$

,

$$\begin{aligned} I_2 &= R - B, \\ I_3 &= R - 2G + B, \end{aligned} \tag{4.32}$$

where R, G and B are RGB color space channels, has better performance than RGB color space (accuracy 69.5%). MSAR accuracy was 86% and Two-Stage combined accuracy was 90.3% using OHTA and MSAR. Their Two Stage scheme is shown in Figure 4.10. It is based on 32 kNNs and majority voting from kNNs predictions.

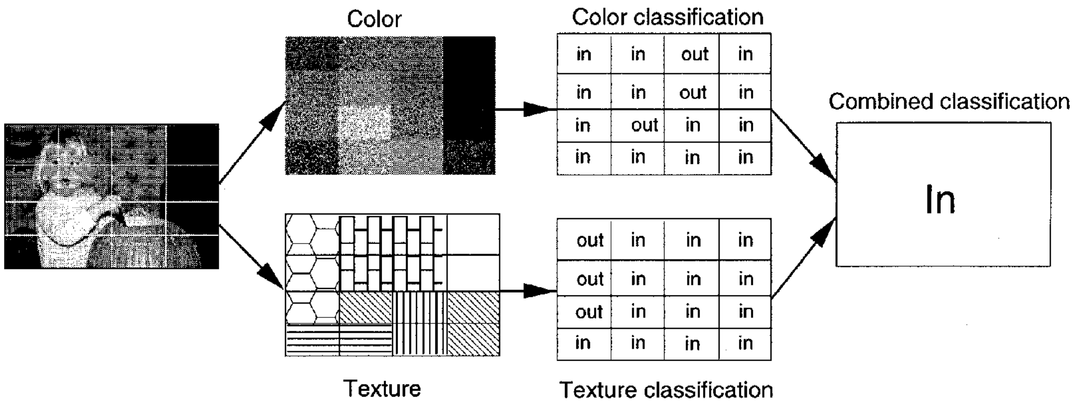

Figure 4.10: Two-stage classification combining color and texture. Taken from [6]

Kodak dataset is also used in the paper [190]. The authors used SVM classifiers on LST color space (6bins per channel) and two-level wavelet decomposition [191] as a texture descriptor.

Using a Two-stage classification scheme, they achieved 90.2% on the test set. Moreover, the same authors incorporate a semantic-based approach in their paper [192]. Their approach is based on detecting sky and grass. By using a two-stage approach together with semantic information, they get an overall accuracy of 92.8%.

Payne et al. use edge-based features in their papers [193], [194]. Authors discover that images that contain organic objects – usually occurred in an outdoor environment – have a larger number of small erratic edges than synthetic objects – usually occurred indoor – that have straighter and less erratic edges. Thus, they measured the straightness of edges in the image by the following equation

$$S_i = \frac{v_i}{d(e_i)}, \tag{4.33}$$

where e_i is the current edge, v_i is the Euclidean straight line distance between the start and end points of e_i , and $d(e_i)$ is defined as the pixel distance – number of pixels – of e_i. They achieved 90.71% on the database of 872 photographs.

Another approach based on using Bag of features is used in paper [7]. Lazebnik et al. use three levels spatial pyramid with a matching technique called histogram intersection. They used multiple features in one pyramid – in particular, SIFT and GIST[195]. The GIST is a global feature descriptor proposed in [196]. An example of the pyramid is shown in Figure 4.11. Features are extracted from each level of the pyramid. All results are concatenated and

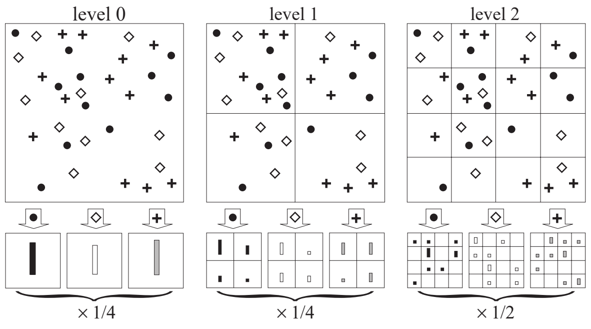

Figure 4.11: An example of constricting a three-level pyramid with tree feature types. Taken from [7]

weighted, as is also shown in the figure.

In 2010 Kim et al. [197] proposed an approach that combines edge and color features in Edge and Color Orientation Histogram (ECOH). They compute edges and color orientation histograms individually. Histograms are then concatenated. Orientations are quantized into K angles. Moreover, the input image is unequally decomposed into subblocks. They tested the approach on a small dataset of 626 images.

Another bag of features approach was used by Battiato et al. in their paper [198]. They propose to use texture description called textons, calculated in a four-level image pyramid. Calculated feature vectors are used to create a weighted Bag of Visual Words (BOVW) dictionary. The authors used a 15 scene dataset. A similar approach was also used in paper [199].

An interesting texture-based approach was the Centrist texture descriptor – based on census transform – introduced in paper [200]. It is similar approach to the Local Binary Patterns (LBP) [201] with one exception. It has different numbering of pixels in the local neighborhood during the calculation of individual responses. Moreover, the Centrist feature descriptor is a global descriptor. Thus, they propose the process of calculating one feature vector for the input image. It contains 31 feature vectors computed from subblocks of the pyramid. I implemented both LBP and Centrist libraries (for C++ and Python) during my studies. It is available as open-source on my Github[2]. Implementations are based on paper [202], where authors proposed real time implementation of LBP algorithm.

Paper [203] introduces a combined visual descriptor called GBPWHGO. It combines Gradient Binary pattern (GBP) with Weighted Histogram of Gradient Orientation (WHGO). GBP is inspired by LBP. The calculation is based on applying of following masks $G_1, \dots, G_4$:

$$G_1 = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, G_2 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, G_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, G_4 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.34}$$

GBP value in the pixel location i, j is then computed as:

$$\mathbf{GBP} = s(\|\mathbf{G_1}\| - \|\mathbf{G_4}\|) + s(\|\mathbf{G_3}\| - \|\mathbf{G_4}\|) \cdot 2^1 + s(\|\mathbf{G_1}\| - \|\mathbf{G_2}\|) \cdot 2^2 + \sum_{k=1}^{4} s((\|\mathbf{G_k}\|) \cdot 2^{7-k}. \tag{4.35}$$

where

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & otherwise \end{cases} \tag{4.36}$$

Pixel intensities in the image are normalized to the interval $\langle 0, 1 \rangle$ before calculation.

The second part of the GBPWHGO description – WHGO – describes the gradient's distribution within the image. It is similar to gradient information extracted by SIFT or HOG methods. Gradient orientation is almost equal in natural scenes but not equal in man-made scenes, usually more horizontal and vertical orientations.

WHGO is calculated as follows. Image is decomposed into 2×2 subblocks. WHGO is computed for each subblock – concatenated at the end. Orientations $\theta_{ij} \in \langle -\pi, \pi \rangle$ are discretized into B bins using following formula

$$\mathbf{O}_{ij} = ceil\left(B \times \frac{\pi + \theta_{ij}}{2\pi}\right) \tag{4.37}$$

where function ceil is used to round computed value to the nearest greater integer value. Orientations are computed for each pixel in the input image. Then weights has to be calculated as follows

$$\mathbf{W}_{\delta,k} = \frac{\sum_{(i,j)\in\delta} \mathbf{M}_{ij} \mathbf{Q}_{ij}^k}{\sum_{(i,j)\in\delta \mathbf{M}_{ij}}} \tag{4.38}$$

where δ is a subregion 1-4 and k is a k-th dimension of a histogram, $\mathbf{M}_{ij}$ is the gradient magnitude of the pixel i, j and where $Q_i j$ is computed as follows:

$$\mathbf{Q}_{ij}^k = \begin{cases} 1, & \mathbf{O}_{ij} = k \\ 0, & otherwise \end{cases} \tag{4.39}$$

[2]https://gihtub.com/neduchal

Finally the WHGO descriptor is computed using formula

$$\mathbf{H}_{\delta,k} = \mathbf{W}_{\delta,k} \cdot \frac{\sum_{(i,j)\in\delta} \mathbf{Q}_{ij}^{k}}{N_\delta}. \tag{4.40}$$

The final GBPWHGO descriptor is then created by concatenation of computed GBP and WHGO vectors.

In their experiments, they train SVMs. Based on their conclusions, the method outperforms SIFT, LBP, and GIST descriptors in scene classification. Unfortunately, I was not able to confirm this in my experiments. For example, method GBP itself had poor results in my experiments, especially for cases with fewer bins in the feature vector.

In the same year as the previous paper, the same authors published another paper where they propose an approach based on BOVW created. The codebook for BOVW is created using the k-means algorithm. It is computed on the image pyramid – it is similar to the approach in paper [198].

The authors of paper [204] propose an approach based on various so-called experts. Experts are image feature descriptors. They tried six experts based on color or texture from previous papers and three new methods. The final decision is then made by majority voting. Moreover, voting is performed in disjoint subspaces created by data grouping in data space. In each subspace, a result is combined based on a trained classifier. A similar approach was introduced in paper [205].

Raja et al. propose Normalized Bins of Hue and Saturation (NBHS) in their paper [206]. They also use Principal Component Analysis (PCA) for dimensionality reduction. Finally, they use the BOVW approach and Sparse representation classifier (SRC).

Paper [207] proposes a texture based BOVW approach. Interesting is using of the cascade of indoor vs. outdoor classifiers. In the first step, indoor vs. outdoor classification is performed. Then more indoor and outdoor labels can be set to input images based on various features.

Review paper [208] on indoor vs. outdoor classification mentioned several works published in previous years. Moreover, they mentioned several open challenges such as dynamic scenes, high-level features, or deep learning.

Finally, the last representative of the classic classification scheme that will be mentioned is paper [209]. The authors of this particular paper proposed a new scene descriptor called Spatial Color GIST Wavelet Descriptor (SCGWD) – combining OHTA GIST wavelet descriptor with census transform histogram (Centrist) spatial pyramid representation.

4.2.3 Neural nets based approaches

Few recent works are based on neural networks. In the past, even these approaches count on the use of some feature descriptor. For example, paper [210] presents a neural network based approach where description using color, entropy, discrete cosine transform (DCT) coefficients, and edge orientations are used as an input of several disjunctive neural networks. Their outputs are concatenated and used as the input to the second stage neural network classifier that makes the final decision.

Another example is the paper of Tahir et al. [211]. The approach is based on GIST features – of length 512 bins – as input for feedforward neural network. Their neural networks have three hidden layers with 13, 13, and 2 neurons. The output is a feature vector of length 2. Their experiments achieved accuracy 90,8% on their dataset containing 2420 images – created by combining one outdoor dataset and one indoor dataset.

The more general approach can be found in papers [212] and [213], where the scene is classified into multiple classes – they do not focus on indoor vs. outdoor classification. Moreover, it is an example of deep learning, where the whole image is used as an input of the neural network. Both papers use the Places [214] dataset. Its subset, called Miniplaces, is used in my experiments too. A list of datasets suitable for this problem is presented in the next section.

4.2.4 Datasets

During research on indoor vs. outdoor environment classification, I discover only three datasets directly designed for this problem. And only two of them are available online. They are named IITM-SCID in two versions. Both versions consist of the small number of images. Version 2 has 907 images, and version 1 only 180 images.

A slightly larger dataset was Kodak Stock Image Database dataset with 1343 images. Unfortunately, this dataset is no longer available online.

On the other hand, several datasets exist for the general classification of the environment – scene classification. Moreover, they usually contain both indoor and outdoor environments classes. It is possible to use them for indoor vs. outdoor classification by creating a file to map each class to either indoor or outdoor label.

All datasets that were explored are listed in Table 4.1. It is particularly dataset SUN397, dataset Places in three versions – Places, Places365 and MiniPlaces. Finally, there is mentioned the dataset MIT Indoor 67, but it worth noting that it is the indoor-only dataset. In my experiments, I used Miniplaces because it offered sufficient size and data diversity.

Data in Miniplaces consists of 100000 training images, 10000 validation images, and 10000 testing images. All images have a resolution of 128×128 pixels. Unfortunately, testing images was determined for use in the online challenge. Thus, in my experiments, I used validation data for testing, and training data was divided into 90000 training sets and 10000 validation set.

Dataset	Reference	# of Categories	# of Images
Kodak	–	2	1343
MIT Indoor 67	[215]	67	15620
15-Scene Dataset	[216],[217], [7]	15	4485
IITM-SCID	–	2	180
IITM-SCID2	–	2	907
SUN397	[218], [219]	397	108754
Places	[214]	205	2500000
Places365	[214]	365	1803460
Miniplaces	[214]	100	120000

Table 4.1: Available Scene classification datasets

4.3 Environment classification system – related work

In the field of environment classification systems, several papers propose systems based on GPS signal presence such as the paper of Urcola [23]. In my work, I focus on environments without the presence of GPS. I discovered only three papers that addressed the design of such a system. The first one is called SmartSLAM. [169]. The second one is based on discrete events [170]. And the third one is the paper of Collier [220]. But they are similar. All systems use some hierarchical system that sets up the best SLAM system for a particular environment. The design of systems presented in the mentioned papers is shown in Figure 4.12. The disadvantage of both systems is that they are described and experimentally tested, but there is no available implementation.

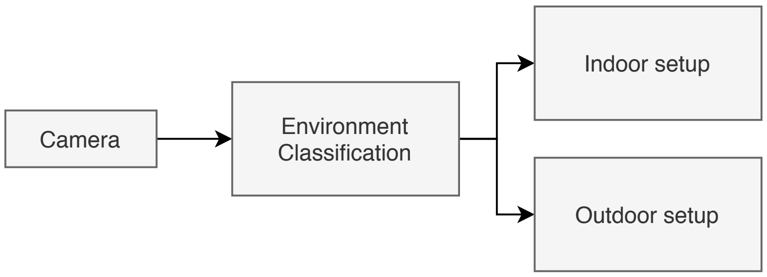

Figure 4.12: Design of multi environment robot system based on related papers

Unfortunately, to my best knowledge, there is no current research in the field of a multi-environment robot system. The only exception is my papers [221] that introduce the improved concept of such a system, and [222] that addresses the problem of indoor vs. outdoor classification. The system will be described in the next chapter.

Chapter 5

Environment classification system

In this chapter, the solution to the environment classification system will be proposed. The problem statement based on the previous parts of the thesis will be introduced in the first part. Then, the proposed solution to the problem will be described. The third part of this chapter is focused on the mobile robot that was assembled and operated to record data for experiments. The fourth section aims to describe the recorded dataset as well as the whole process of data recording. The end of this chapter is devoted to describing the implementation of the proposed Environment Classification System (ECS).

5.1 Problem statement

The problem of the environment classification system system is based on the fact that most mobile robotic applications are designed for a single environment. Thus, there is a significant probability of system failure when it moves to another – in a particular manner – environment. Based on this general description, it is possible to define a goal that should be achieved.

The goal of a environment classification system is to detect the change of the environment before or during the transition stage. The transition stage is a moment in time of a certain length when the robot moves through the border between two environments. Based on the detected change, the system is also supposed to classify the type of the current environment and adjust the behavior of the robot properly.

In the practical part of my thesis, I will deal mainly with change detection and classification of the environment during the transition. In the next section, the proposed system for solving the defined problem will be described in detail.

5.2 Proposed system concept

The aim of this section is to describe ECS for multi-environment robot system. It consists of three parts summarized in the following list

Environment Change Detection (ECD) This module is responsible for detecting particular changes in the environment and generating a trigger when the change is large enough.

Image Based Environment classification (IBEC) This module classifies the current robot environment based on the image based data. The environment classification is triggered by the ECD module.

Robot Behaviour Adaptation (RBA) This module is supposed to adjust robot behavior. It can be done by changing the parameters of a particular part of the system or switching between onboard software.

For greater clarity are the described parts shown in Figure 5.1. This picture also shows the links between individual modules. In particular, the ECD module sends triggers to the IBEC module. Then IBEC module sends its decision about the environment to the RBA module. Finally, the RBA module adjusts the behavior of the robot by changing active software parts and their parameters. Besides these links, there is also a link from non-visual sensors to the ECD module. Moreover, data from the Camera sensor is linked to ECD and IBEC module simultaneously. It is caused by the fact that some changes can be detected in the image data – e.g., lightning condition of the environment – and can also be used to classify environment type.

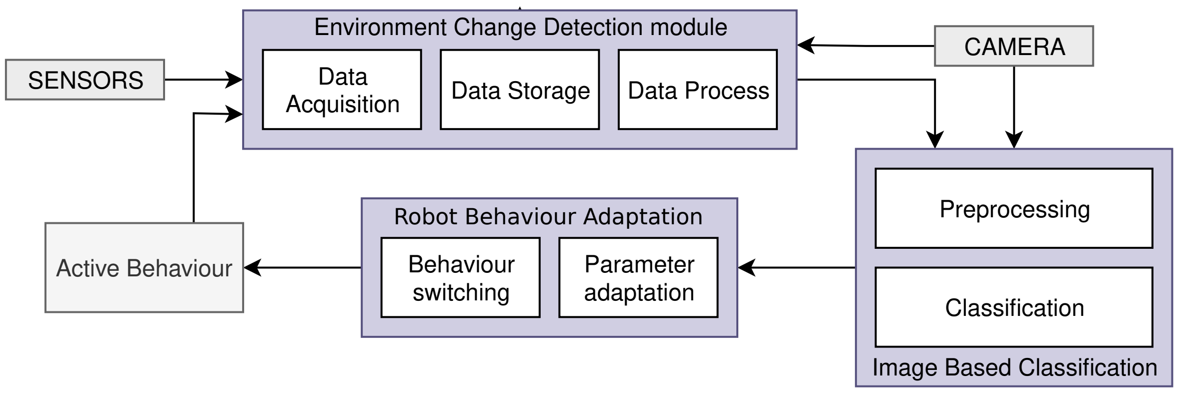

Figure 5.1: The concept of the environment classification system.

Modules in Figure 5.1 comprise inner rectangles that represent essential parts of each module. In the following sections, all three modules will be described in detail. The description is ordered by way of the data from sensors up to the RBA module.

5.2.1 Environment Change Detection Module

The first module of the multi-environment robot system is the ECD module. It collects data from all available sensors on its input, processes them, and generates triggers when the change of particular value in the environment is detected. When the trigger is generated, the probability of the transition between environments increases.

Figure 5.1 shows the inner structure of the ECD module. It consists of three parts called Data Acquisition (DA), Data Storage (DS), and Data Process (DP). The concept of this module was introduced in my paper [221]. In the next paragraphs, each sub-module will be described in more detail. In the first part, the inner communication between sub-modules will be presented. Then, the trigger generation will be described independently.

Inner communication structure

The attached sensors are connected directly with the DA sub-module. It is supposed to receive the sensor data, do necessary preprocessing, and then send them to the DS sub-module using a defined message type suitable for a wide range of the sensor data. In particular, the message is defined as the pair of map layer information – each sensor has its layer in the map handled by the DS sub-module – and floating-point number for the data. The structure of the DA sub-module is visualized in Figure 5.2.

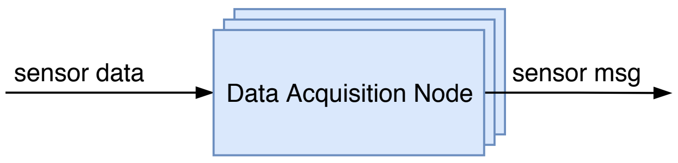

Figure 5.2: The structure of the Data Acquisition sub-module

The essential property which is visible in Figure 5.2 is that it is possible to run more DA sub-modules within one system. Each running DA sub-module sends preprocessed data in the same way. Thus, all data is sent to only one DS module.

Data Storage sub-module is the most complex part of the ECD system. It is responsible for storing all data from sensors in the form of multi-layered 2D map. Therefore, DS is build on top of an open-source GridMap[223] library[1], which allows working with a multi-layered map. The structure of the DS sub-module is shown in Figure 5.3.

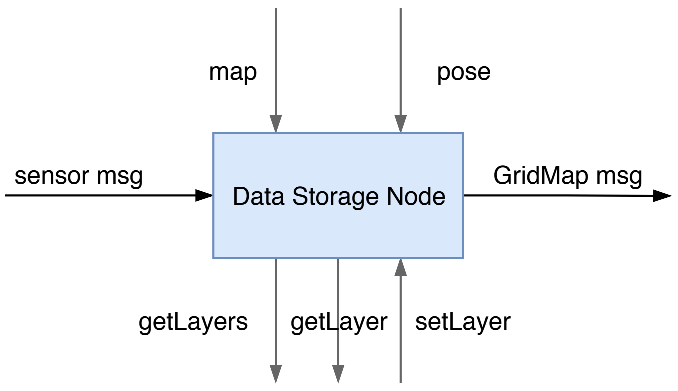

Figure 5.3: The structure of the Data Storage sub-module

[1]https://github.com/ANYbotics/grid_map

The sub-module has three main inputs – map, sensor msg, and pose. The first main input is the `map` , which is a map from the mapping software. In the implementation of the system, it is possible to use a 2D occupancy grid map – e.g., the map generated by gMapping or the map transformed from the 3D map created by RTABMap. The second main input is the `sensor msg` described above. The map layer information is used to store value in the particular layer in the map.

Moreover, the third input, named `pose`, is supposed to add the information about the robot's actual pose in the environment. Thus, this information is used to save data from the sensor message at the robot's position. Examples of the map generated by the ECD module will be shown in section 6.2.

The main output of the DS module is only one. It is called GridMap msg, and it allows the DS module to publish information of the generated map or its subset – base map + particular layers – to other parts of the robot system.

Besides main inputs and outputs, there are also optional inputs and outputs. In particular they are called `getLayers`, `getLayer` and `setLayer`. They have defined communication messages between the DS sub-module and the DP sub-module. The structure of the DP sub-module is shown in Figure. 5.4.

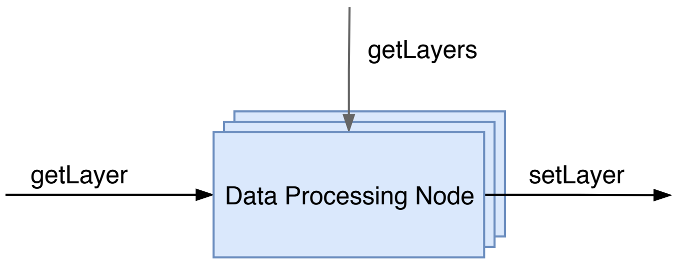

Figure 5.4: the structure of the Data Processing module

Is it visible that it has inputs and outputs with the same name as the one mentioned above. The input called `getLayers` is supposed to load a list of all layers stored in the map. Then it is possible to load particular layer or layers by `getLayer` input. DP processed loaded data, and then it can return processed data into the DS sub-module by the `setLayer` message. Moreover, it is possible to run multiple DP sub-modules within one robot system.

In the previous paragraphs, the inner communication within the ECD module is described. It worth mentioning that the ECD without trigger functionality was originally named Environment Detection System, and it is available as an Open-source ROS package on GitHub[2].

Trigger generation

The main purpose of the ECD is to generate a trigger for the IBEC module. It can be done in either DA or DP modules. In general, it is based on approaches described in Section 4.1. In particular cases, the trigger can be sent directly to the RBA module – e.g., in the case

[2]https://github.com/neduchal/env_detection

of sufficiently different known temperatures in both environments. In such cases, the RBA module can react to trigger generated by some change detection functions such as CUSUM, DifRatio, or VarRatio.

The policy of trigger generation is fully within the competence of the particular DA or DP sub-module implementation. The policy consists of change detection sensitivity and frequency of trigger generation in the case of consecutive detection.

Moreover, another trigger policy can be applied. All triggers can be sent to the trigger fusion module – which is not a native part of the proposed system, but it is identified as one of the tasks for future work. The final trigger is sent when multiple sensors generate their trigger. Or the number of triggers can be reduced by not sending all triggers to other modules – e.g. when the decision function generates triggers with high frequency. By default, it is the IBEC module, which is described in the next paragraphs.

5.2.2 Image Based Environment Classification

This module is responsible for deciding on the type of environment. The structure of the IBEC module is shown in Figure 5.5. As input, it takes image data from the camera and the trigger from the ECD module. The trigger can be understood as an enable signal which activates the whole module to process a single image frame. Image data first go to the preprocessing unit, where feature extraction and description is applied. Feature vector or preprocessed image then go to classification unit of the module. Classification unit contains trained classifier – e.g., one of the classic machine learning or neural nets based classifiers described in Section 4.2 – which takes feature vector and return integer value corresponds to the class of the environment. In our case, it can be either 0 – indoor – or 1 – outdoor.

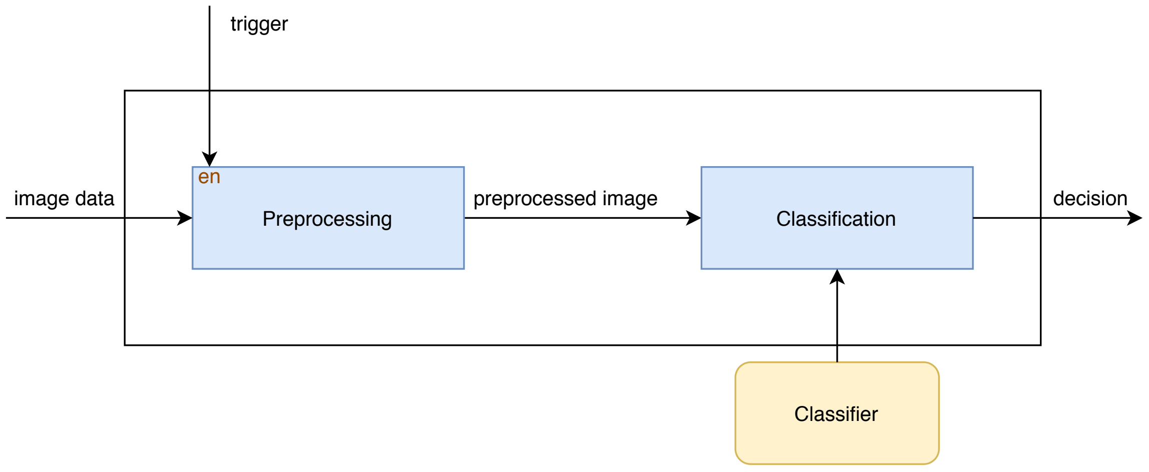

Figure 5.5: Scheme of IBEC module.

Moreover, the classification unit can be composed of multiple classifiers in the case of the Two-stage scheme approach as described in Section 4.2. Of course, the preprocessing unit has to be implemented based on the used classifier because the classifier's input data must have a particular size, type and has to be prepared by particular feature extraction and description method/s.

5.2.3 Robot Behaviour Adaptation Module

The last part of the proposed system is the RBA module. Its purpose is to take information about the classification of the environment and then to set robot behavior – parameters, software – based on the predefined configuration for the particular type of environment. The structure of the module is shown in Figure 5.6.

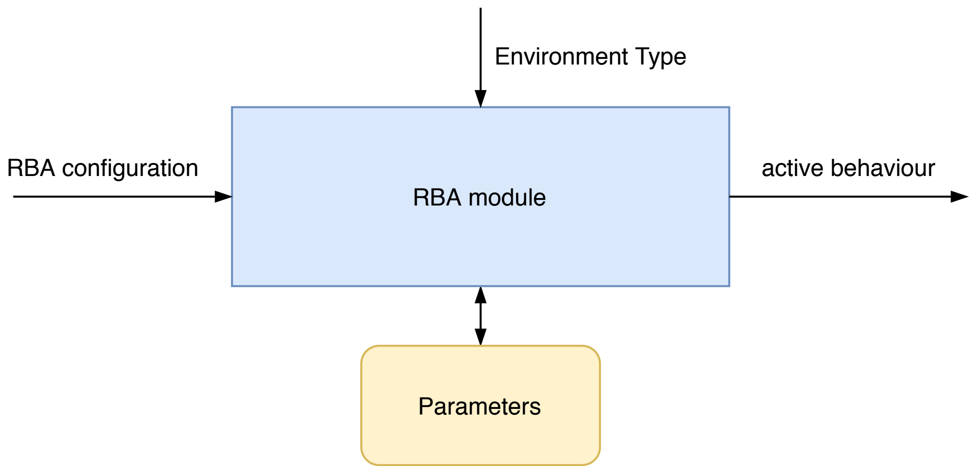

Figure 5.6: Example short run visualization

The RBA configuration is predefined, and it is stored on the hard disk of the on-board computer. In my implementation, the configuration file is saved in a YAML format because of the native support of this format in ROS. In Figure 5.6 there is also block named Parameters. It is a representation of loaded parameters in the system. For a change of the environment, it could be necessary to get some loaded parameters first. In particular, the ROS contains a parameter server that can be used for this purpose.

The RBA module's output is the set of changes in the system that adapt robots' behavior. For example, in the ROS framework, the RBA module can be built to kill active nodes or whole launch files and run new ones.

5.3 Hardware

For my research, I assemble and operate a ground robot platform equipped with multiple sensors. The platform was then used to record data and test algorithms both in terms of results and performance. In the following paragraphs, the ground robot platform is described.

5.3.1 Robot Chassis

The robot is based on the Wild Thumper 6WD chassis. The chassis base comprises metal parts with holes for easy assembly of computers, sensors, and other equipment. In particular, the chassis has three high levels that are intended to assembly equipment. The chassis is equipped with six drive DC motors with 75:1 steel gearboxes. Moreover, the motors attached to middle wheels are equipped with rotary encoders. The maximum speed of this chassis is 3 km/h. On the other hand, during recording, I realized that the motors' performance at low speed is poor, and robot movement is not smooth. The robot is shown in Figure 5.7. The

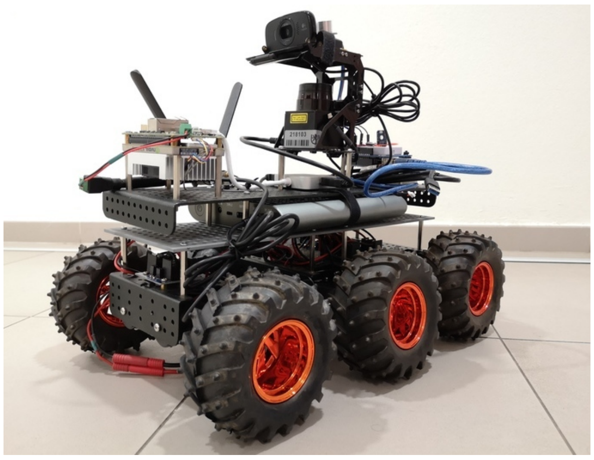

Figure 5.7: A photo of the Wild Thumper robot platform.

robot is equipped as follows. At the lowest level, there is a motor controller called T-Rex Board, a battery intended to power the motors and motor, controller board. The robots's computer is powered from a notebook powerbank which is placed in the middle level of the robot. The computer itself is placed on the top level together with all the sensors.

5.3.2 T-rex controller board

This controller board – shown in Figure 5.8 – is responsible for controlling the motors and receiving information from encoders and robots' batteries. It is equipped with microchip ATMega328P. For programming the board, Arduino IDE can be used. It also contains a 3-axis accelerometer. For communication, three different approaches can be used. In particular, it is I2C, RC, and Bluetooth. In my case, I2C is used.

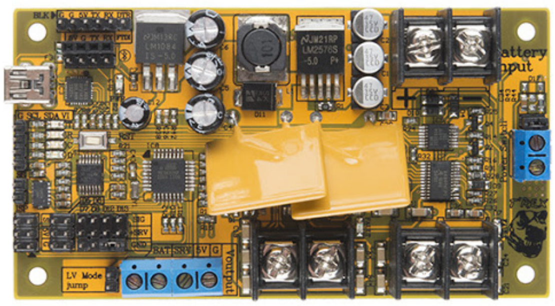

Figure 5.8: T-Rex controller board

5.3.3 On-board computer and software

The main computer of the created mobile robot is NVIDIA Jetson Xavier – see Figure 5.9 – which is an arm based computer with 8-core CPU, 32 GB of RAM, and 32 GB of space in eMMC storage. Moreover, I have added a 512 GB SSD disk to be able to save big data onboard – e.g., Rosbags of robot runs.

Figure 5.9: Onboard computer Nvidia Jetson Xavier

Previously, we used another computer from the Jetson family. In particular, it was Jetson TX2. However, the Xavier was chosen for the final configuration due to the 20x performance increase compared to the TX2 computer.

Ubuntu Linux 18.04. is used as the operating system of the mobile robot. It is a modified version for the Jetson computers family. I used this operating system because of the native support of the ROS framework. In particular, I used ROS Melodic.

5.3.4 Equipped sensors

The robot is equipped with a set of sensors. It is particularly an RGBD camera, Row LiDAR, Thermometer, Air pressure sensor, Humidity sensor, UV light sensor, Ultrasound distance sensor, and an inertial measurement unit. All sensors will be briefly described in the next paragraphs.

RGBD Camera Intel Realsense D435

The main sensor for the image-based classification of the environment is the RGBD camera Intel Realsense D435 – shown in Figure 5.10. RGBD is an abbreviation of the words Red, Green, Blue, and Depth. Thus, it is a color camera with depth information. Depth is obtained by the combination of stereovision and laser projection of known pattern in infrared spectrum. RGB and depth data are represented by images. 24-bit in the case of RGB image (8-bits for every channel) and 16-bits for depth image.

Figure 5.10: Intel Realsense D435

Row LiDAR Hokuyo

The second important sensor is the row LiDAR. The row means that the LiDAR moves in one axis and its data represents one line in the space. It can be imagined as the floor plan of the space in front of a robot. The data is represented as an array of float values indicating the distance in the particular angle. The angle is computed as follows

$$\alpha = \alpha^{-} + i \cdot r, \tag{5.1}$$

where α^{-} is a minimum of the angle range of the LiDAR – e.g., -120° –, i is an index of value in the array and r is an angular resolution of the LiDAR. In particular, I used row LiDAR Hokuyo URG-04LX-UG01 – see Figure 5.11. It is wide-range LiDAR with field of view 240°. The distance range of the LiDAR is 0 - 5600 mm and its accuracy in distance is +- 30 mm for short distances and ±3% for distance above 1000 mm.

Figure 5.11: LiDAR Hokuyo URG-04LX-UG01

Non-visual Arduino based sensor set

To record non-visual environment data such as temperature, humidity, UV light value, air pressure, and ceiling height, I assembled an Arduino based sensor set. It is shown in Figure 5.12. It consists of Arduino UNO3 as a controller board, BME 280 sensor, Arduino UV Light sensor, and Ultrasound distance sensor. The communication with the main computer is based on a serial connection with the use of ROS messages. All equipped sensors are introduced in the following list.

BME280 is a sensor composed of multiple parts measuring environment temperature, humidity and air (barometric) pressure. A temperature sensor is calibrated on temperatures in the range from -40 to 85°C. Its resolution is 0.01°C and accuracy ±1°. The

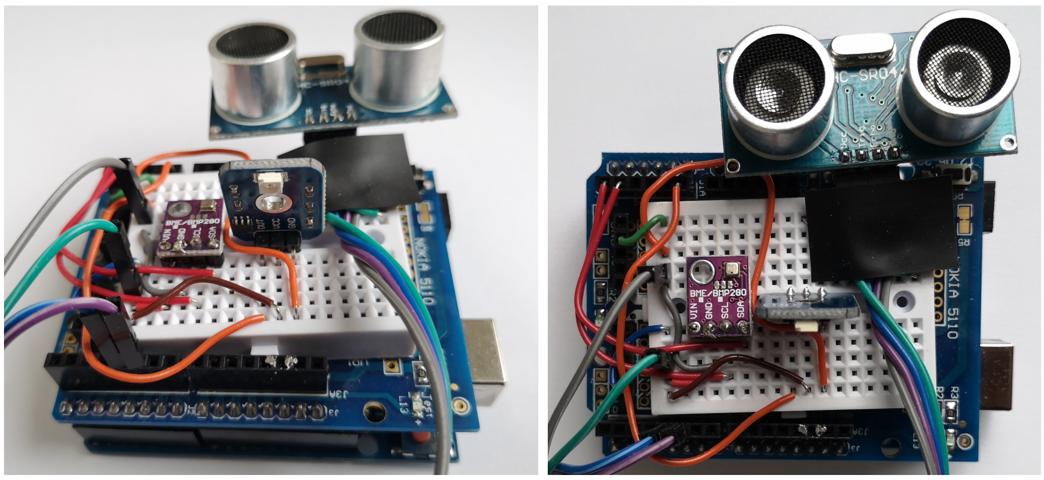
Figure 5.12: Arduino based sensor set

Humidity sensor measures relative humidity in the range from 0% to 100%. with resolution 0.008% and accuracy ±3%. Finally, the air pressure sensor measures in range from 30 to 110 kPa (0.3 – 1.1 bar) with resolution 0.18 Pa and accuracy ±1 Pa. Sensor communicate with Arduino over I2C protocol.

Arduino UV Light sensor measures UV Light in the environment. UV index can be computed from obtained values. Unfortunately, in my case, the UV light was usually low or zero during the data recording.

Ceiling height sensor is an ultrasonic distance sensor that measures the distance between the robot and the room ceiling. It is an accurate sensor with accuracy ±3 mm. The sensor's official distance range is from 2 to 450 cm with a field of view lesser than 15°.

Inertial Measurement unit

It is a sensor composed of an accelerometer and gyro. In the system, it can be used as odometry information both individually or together with wheel encoders. The sensor is shown in Figure 5.13

Figure 5.13: IMU Lord 3DM-CV5

5.3.5 Software equipment of the robot

As it was mentioned before, the operation system of the robot is Ubuntu 18.04. with installed ROS Melodic. The robot uses software that is included in ROS and the 3rd party software. In particular, open-source SLAM systems such as gMapping or RTAB-MAP can be used with the system.

5.4 Dataset

The first task for the described mobile robot platform was to record the dataset for testing the proposed multi-environment system. The dataset was planned to be divided into several sequences. Each sequence is supposed to be unique. It was achieved by three approaches:

- Recording in the different time of the day (excluding night) / different part of the year.
- Recording each environment separately / together – i.e. records with transition.
- Recording with all sensors / with subset of sensors.

5.4.1 Recording

Data was recorded primarily at two different locations – university campus and prefabricated housing estate. Examples of images are visible in Figure 5.14. The robot was driven manually using a wireless joystick.

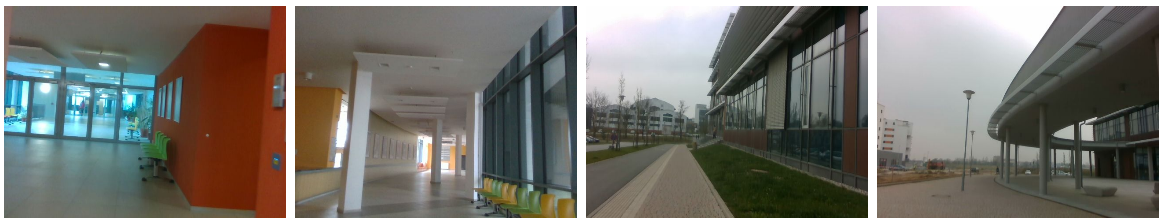

Figure 5.14: Examples of an environments in dataset.

5.4.2 Description of recorded data

The recorded dataset contains 17 sequences of various lengths in the form of Rosbags. The longest sequences recorded on the university campus have more than 10 minutes in length. Thus, they can be divided into several shorter sequences. As it was mentioned, the data in Rosbags vary a little. In other words, not all Rosbags sequences contain all types of data. In the following list, there are described data types recorded by the robot.

Image data is compressed and it uses sensor_msgs/CompressedImage message. The frequency of image data is 30 Hz and the resolution is 320 px $\times$ 240 px for RGB images. Similarly 30Hz and 320 px $\times$ 240 px for depth images.

LiDAR data is based on the standard sensor_msg/LaserScan ROS message with frequency 10 Hz.

Rotary Encoders data are collected on T-REX control board. It is sent to the main computer in two messages of type Int64. Frequency is based on the moving of the robot, but it is usually up to 100 Hz.

Non-visual sensor data is collected on connected arduino. Its frequency is 24 Hz and it is sent in the created ROS message which contained temperature sensor data, humidity sensor data, air pressure sensor data, UV Light sensor fata and Ceiling height sensor data.

IMU data send number of information such as raw IMU data, pose data and magnetic field data. The frequency is 100 Hz.

5.5 Implementation notes

In this section, an implementation of the proposed system will be briefly described. Implementation is programmed in C++ and Python programming languages for ROS. It can be found as a repository on my Github[3]. The package contains all parts described in this chapter. Moreover, it is possible to use it together with GridMap Library. I plan to extend the capabilities of the system continually. In the next paragraphs, I provide a summary of the current state.

The whole system can be configured in two ways. The first one is to change the parameters in provided launch files. In particular, the names of the system topics for messages between parts of the system and other nodes can be adapted here. Moreover, there is a YAML file in the config subfolder, which contains a detailed system configuration. Individual parts of the system can be affected by this file.

For example, it is possible to change the behavior of the ECD module. Currently, the ECD module implements three methods for CPD. In particular it is CUSUM method and two proposed method called DifRatio and VarRatio (see Chapter 6.1). In the configuration file, the method and its parameters can be set.

IBEC module is currently prepared for arbitrary trained classifier from Scikit Learn python package. Moreover, it is possible to create a Python class that satisfies function names for arbitrary classifiers, including neural networks. In the future, I want to provide the same configuration as in the case of ECD. Thus, it is supposed to be able to configure the classifier in the YAML file.

Switcher implementation is based on the roslaunch python package. It can run and kill both individual nodes and whole launch files. It can be configured in the YAML file as well. All states with their activation names and names of associated nodes or launch files can be set here. Moreover, the default state after running the switcher can be defined as well.

[3]https://github.com/neduchal/ecs

```
# ECS CHANGE DETECTOR PARAMETERS
ecs_change_detector:
  input_topic: /ecs/sensor_values
  output_topic: /ecs/trigger
  sensors:
    - name: sensor1
      method: difratio
      republish: 1
ecs_map:
  input_topic: /ecs/map/value
ecs_switcher:
  input_topic: /ecs/switcher/switch
  processes:
    - name: test1
      pkg: ecs_switcher
      process: node1.launch
      default: yes
    - name: test2
      pkg: ecs_switcher
      process: node2.launch
```

Chapter 6

Experiments

In this chapter, the performed experiments will be described. They cover individual parts of the researched problem. Specifically, it is a description of an experiment for processing data from non-visual sensors, creating a system for adding data into the created map, an experiment to classify the environment into indoor and outdoor, and testing the proposed system on real sequences.

6.1 Environment change detection

This experiment is based on data processing from non-visual sensors. The following description will be divided into a description of individual sensors, including the suitability for solving the task, a demonstration of the results using several methods, and a subsequent discussion of the obtained results.

6.1.1 Description of used sensors

Temperature sensor measures the air temperature of the environment around the robot. Figure 6.1 shows data without changing the environment, and Figure 6.2 shows data with changing the environment. Specifically, the robot starts in the indoor environment, then moves to the outdoor environment, and finally returns to the indoor environment.

The first important property visible in Figures 6.1 and 6.2 is a delay of the temperature value. Even if the difference between indoor and outdoor temperature is significant, the change of the measured value is not immediate. Moreover, the delay is also contained in the data without transition because it took tenths of seconds to get a stable value. On the other hand, it is visible in Figure 6.1 that the changes in value are insignificant.

Humidity sensor measures the relative humidity of the environment. Again, Figure 6.3 shows a waveform without changing the environment, and Figure 6.4 shows a waveform without changing the environment. They are the same records as in the temperature sensor. The same recorded data will be used for other sensors too.

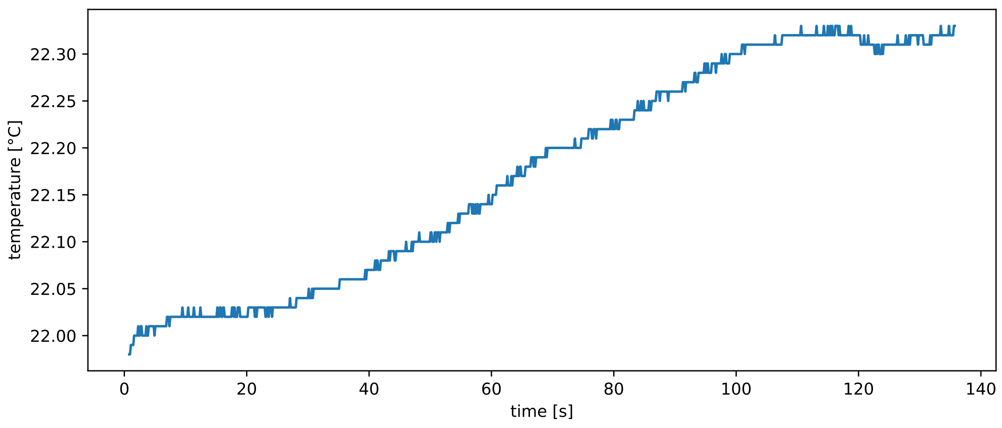

Figure 6.1: Example of temperature sensor data with non-significant change.

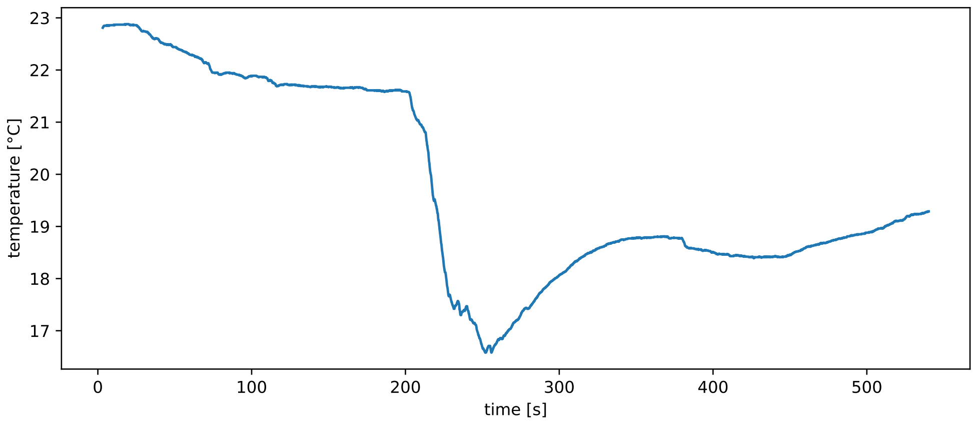

Figure 6.2: Example of temperature sensor data with significant change.

Humidity sensors have similar properties to the temperature sensor. It contains a significant delay. Moreover, it is more sensitive to minor changes in the environment. In Figure 6.1, there is visible noise in the data. It has to be filtered to remove its influence on the detection. Data in Figure 6.4 shows the behavior of the humidity sensor during the transition. In this particular case, the robot moves from an air-conditioned building to an outdoor (winter) environment. The humidity starts growing. The value then drops after the opposite transition.

Air pressure sensor sensor measures the barometric pressure of the environment. Examples are shown in Figures 6.5 and 6.6.

In the first case, it is visible that the air pressure value does not change significantly. On the other hand, there is a noise that can cause problems. The second case shows that the value of the sensor changes in the order of hundreds of pascal. But the reason for the change is not a transition between environments. It is caused by the change in altitude. In particular, the robot is in the elevator moving down around the timestep

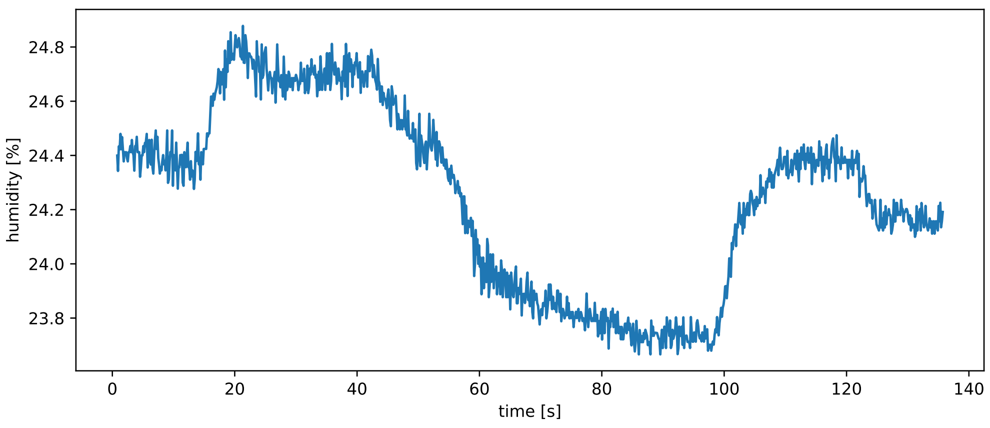

Figure 6.3: Example of humidity sensor data with non-significant change.

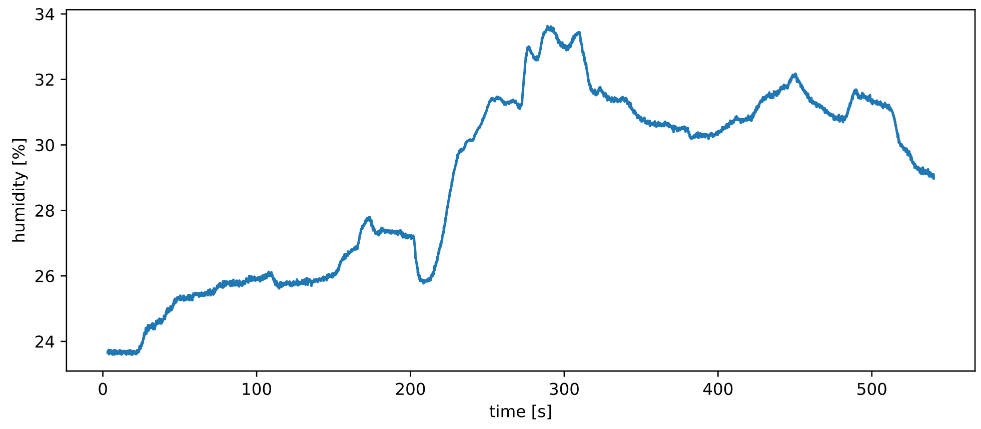

Figure 6.4: Example of humidity sensor data with significant change.

100s.

Similarly, the robot is in the elevator, moving up around the timestep 400s. The transition between environments is visible between timesteps 200s and 300s as a minor increase in the sensor value. Unfortunately, this increase is insignificant because the change is in order of units of pascal.

Distance sensor measures the height of the room ceiling, so it could be useful in detecting the transition between indoor and outdoor environments. The sensor value is immediate without any delay. Examples are shown in Figures 6.7 and 6.8.

Figure 6.7 shows the run without the transition. The robot moves from the office to a hall. Then it moves to the lecture room with several chairs and tables. Then it returns to the starting point. There are visible timesteps of transition between rooms (around timesteps 15, 40, 75, and 100). Low values between timesteps 50 and 70 are caused by chairs close to the robot, and the distance sensor registers them. Similarly, the robot stops next to the table after timestep 100. In conclusion, a transition between rooms

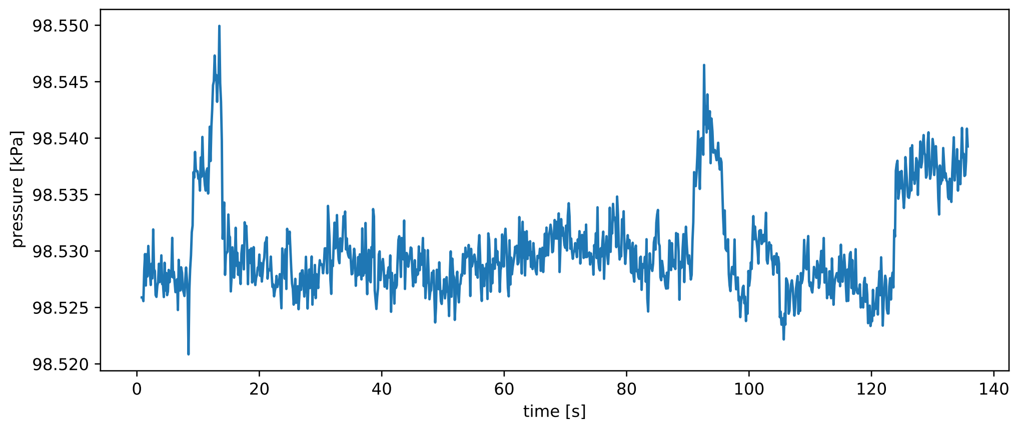

Figure 6.5: Example of air pressure sensor data with non-significant change.

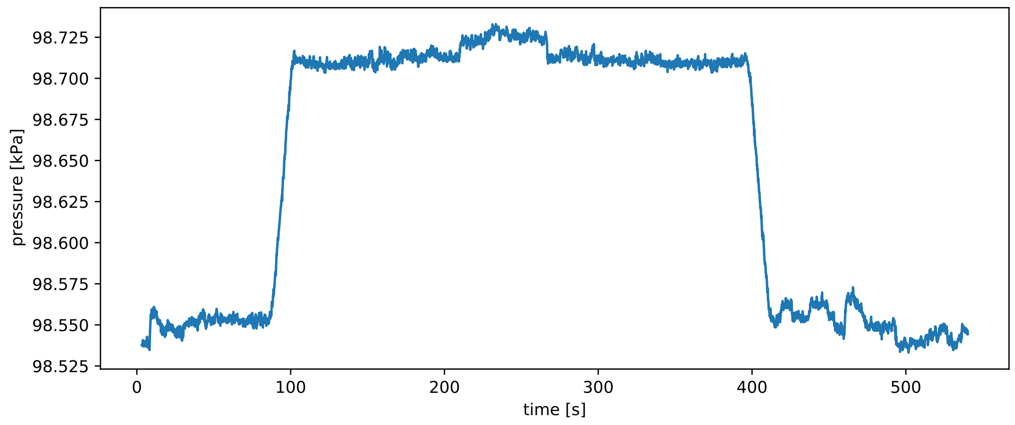

Figure 6.6: Example of air pressure sensor data with significant change.

can be detected by the distance sensor because door height is usually lower than the ceiling height.

In Figure 6.8, the robot moves outside of the building and then returns back. The value of the sensor increases significantly in this case. It worth mentioning that there is a canopy over the entrance of the building. Thus, the increased value is not continuous because the robot registered the height of the canopy.

Summary: All sensors mentioned in the previous paragraphs can be used on the particular type of transition. The temperature and the distance sensor can be used for detecting transitions between indoor and outdoor environments. Moreover, the distance sensor can be used to detect the transition between rooms. The humidity sensor can be used for transition between indoor and outdoor environments but only in the case of active air conditioning or heating systems in the building. Finally, the air pressure sensor is not suitable for indoor vs. outdoor transition detection. On the other hand, it can be used to detect altitude changes – e.g., ride in the elevator.

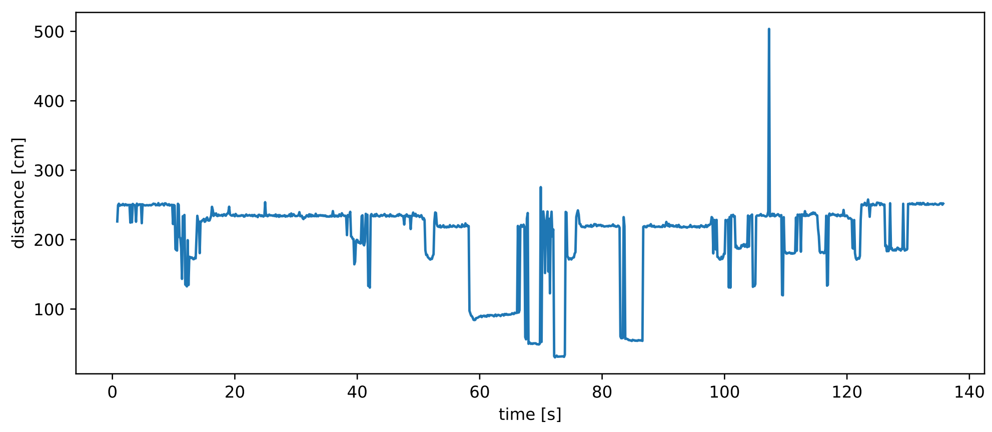

Figure 6.7: Example of distance sensor data with non-significant change.

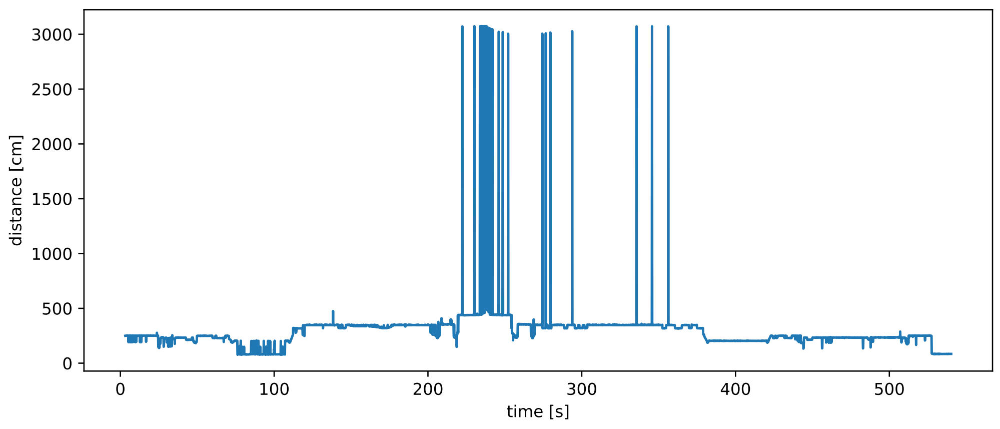

Figure 6.8: Example of distance sensor data with non-significant change.

Based on the findings, the experiment on detecting indoor and outdoor transition will be performed only on the temperature sensor and distance sensor data.

6.1.2 Trigger generation

This experiment aims to test the suitability of the above data to detect the transition between indoor and outdoor environments. It is important to note that the main result of this experiment is a reduction in the computational cost of the classification of the environment based on image data. Thus, it is not critical to generating triggers only at transition points, but it is important not to generate triggers when nothing happens. The experiment's output is the set of triggers that should correspond with timesteps of transition between environments. The discussion will describe the advantages and disadvantages of individual sensors and the suitability of using different processing methods.

First, two simple methods that can be used to detect changes in the one-dimensional signal will be proposed. The first method is based on the ratio of average short-term and average long-term value of the signal's first difference (DifRatio). The second method is based on the ratio between the current signal variance and the stable signal variance (VarRatio).

DifRatio This method is defined as the ratio between two values of the average first difference of the signal. Thus, if the method result is near 1, there is no change in the signal. On the other hand, when the value is near zero or significantly larger than 1, there is a possibility that change has happened. It can be written by the following equation

$$D\left(\Delta x, n_l, n_s\right) = \left(\frac{1}{n_l}\sum_{i=t-n_l}^{t} \Delta x_i + C\right) \cdot \left(\frac{1}{n_s}\sum_{j=t-n_s}^{t} \Delta x_j + C\right)^{-1} \tag{6.1}$$

where Δx is a first difference of an input signal, n_l, n_s are defined lengths of the long-term and short-term signal windows, t is a current timestep, and C is a small constant to avoid division by zero. Moreover, n_l can be set dynamically as a number of timesteps from the last generated trigger. Finally, a maximum of D and D^{-1} is taken as a result. Comparing with a defined threshold is then used for the trigger generating.

VarRatio This method is defined as ratio of variances of current and stable signal. Thus, the result value increases when the current variance increases considering stable signal variance. It can be mathematically written as follows

$$V(x, n, v) = \begin{cases} 1 & if\ var\left(x\left(t-n:t\right)\right) > \alpha \cdot V \\ 0 & otherwise \end{cases}, \tag{6.2}$$

where n is a window length, var is a signal variance calculation – standard deviation can be used instead of signal variance, t is a current timestep, v is a stable signal variance, and α is a defined sensitivity constant.

Of course, many other methods are suitable for detecting changes in the signal. To solve this thesis problem, it is necessary to choose the method that is not computationally expensive – i.e., it can run in real-time – and with as few parameters as possible.

6.1.3 Experiment discussion

In this section, an example of three methods for change detection will be presented on the temperature data signal. In particular, DifRatio, VarRatio, and CUSUM [1] (for details, see Section 4.1) method responses and generated triggers will be shown in the following figures.

In Figure 6.9, there is a record of temperature sensor data during the robot run. The robot starts inside on the fourth floor of the building. It goes to the first floor and then out of the building. After a short time, it goes back to the building.

There is a fast decrease of the temperature around the 4th minute of the record in the figure. The starting and ending point (red circles) are timesteps when the robot goes outside the

[1] All methods were implemented in Python programming language for this purpose

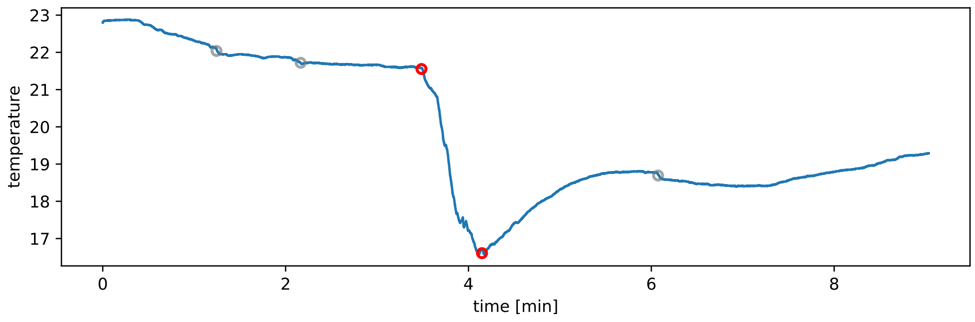

Figure 6.9: Temperature signal with labeled time steps of minor (grey) and major (red) changes.

building and returns inside. They are major changes – transition between environments. Moreover, three timesteps (grey circles) represent minor changes – transition between room and hallway or vice versa.

The first method that will be presented is DifRatio, which was proposed in the previous chapter. Its response is shown in Figure 6.10. The method is defined to detect changes in the first difference of the signal by calculating the absolute value of the long-term and short-term average differences. Thus, the response increasing significantly around the fourth minute, where the transition between environments occurred. A small increase is also visible in timesteps of minor changes. On the other hand, the increase is not significant. Thus it is hard to detect in the response.

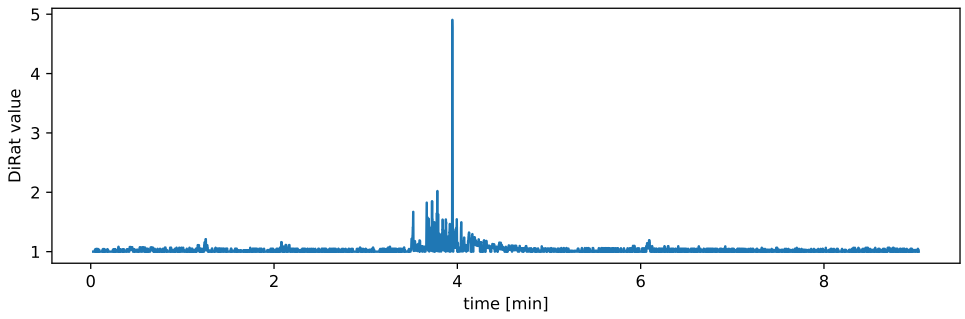

Figure 6.10: Response of DifRatio function on temperature signal

Triggers generated from the DifRatio response is shown in Figure 6.11. It is done by thresholding by a constant value. Thresholding is suitable in this case because the response for the stable signal without significant changes is approximately 1. The DifRatio method is simple, and it detects significant changes in the signal. In this case, it generates triggers for all responses greater than threshold constant 1.5. There are two approaches to change the sensitivity of the DifRatio method. The first one is to select a lower threshold constant. The second one is to select shorter time windows for calculation short and long-term averages.

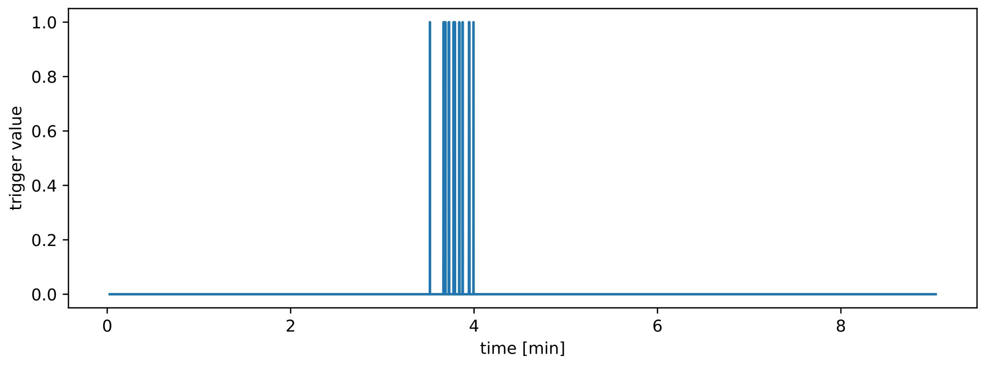

Figure 6.11: Generated triggers based on DifRatio response.

The second method is VarRatio. Its response is shown in 6.12. It is visible that the response is slightly similar to the DiRat method, except the range in the y axis is different. Minor changes are more significant considering the maximum response in this case. Thus it can be detected to generate triggers.

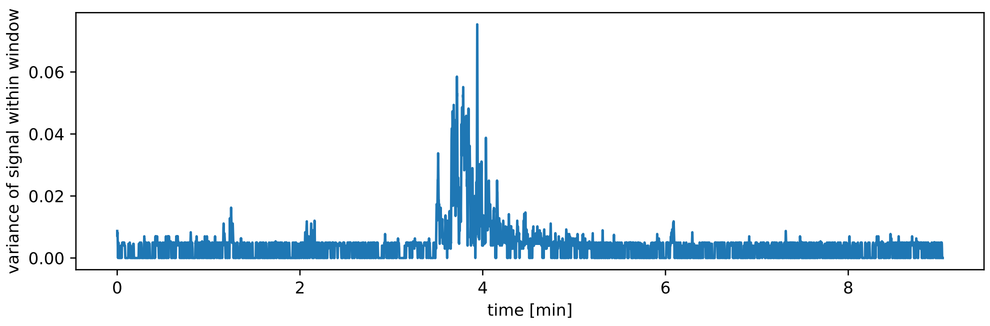

Figure 6.12: VarRatio method response on temperature signal.

Triggers for the VarRatio method are shown in Figure 6.13. The method is more sensitive than the DifRatio function in this case. Triggers are generated by thresholding against the threshold constant. In this case, threshold constant T_c is computed as follows

$$T_c = c \cdot avg(S_s), \tag{6.3}$$

where $avg(S_s)$ is a average value of the stable signal and $c >= 1$ is small constant that increase threshold value to be grater than maximum of stable signal. In the case of presented example, T_c was set to 1.8. Constant c is supposed to change sensitivity of the trigger generation. Another approach is to use the maximum value of stable signal instead of average. The method is again able to generate triggers in the timesteps where the signal change occurred.

The last method that will be presented in this example is the CUSUM method. It is shown in Figure 6.14 to compare it with the proposed methods. As the reader can see, the response

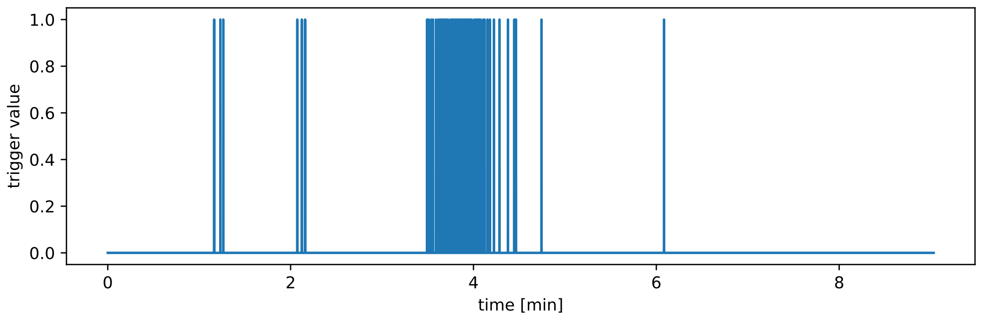

Figure 6.13: Generated triggers based on VarRatio response.

of the CUSUM method is similar to previous methods. On the other hand, the stable signal case response is noisier in several parts of the robot run – e.g., between the start and second minute of the robot run.

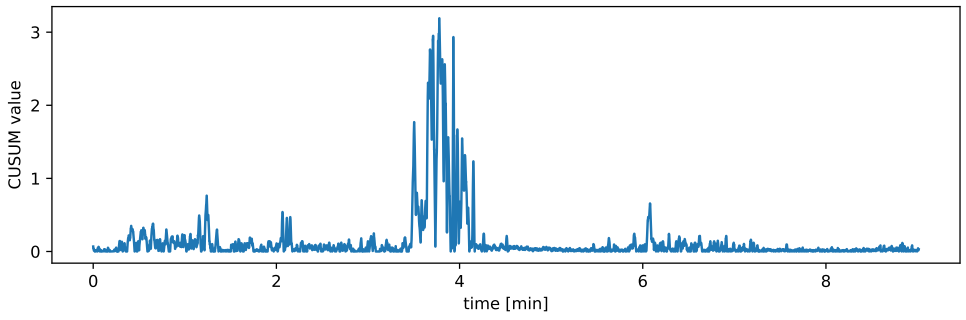

Figure 6.14: Response of CuSum method on temperature signal.

Despite the noisy response, it is still possible to generate triggers using thresholding. Generated triggers are shown in Figure 6.15. Thanks to the noise, it can be difficult to set the CUSUM method's sensitivity to detect minor changes. Thus, only major changes are detected in this particular case.

6.1.4 Computational cost

The computational cost of the previously mentioned methods and two others called BOCD[2] and RulSIF[3] will be discussed in this part. BOCD is an acronym for Bayesian Online Change-point Detection, which is an implementation of the BCPD method described in chapter 4.1. RulSIF method is an implementation of the RulSIF method, which belongs to the Likelihood ratio methods group, similarly to the CUSUM method.

[2]Used implementation: https://github.com/y-bar/bocd

[3]Used implementation: https://github.com/hoxo-m/densratio_py

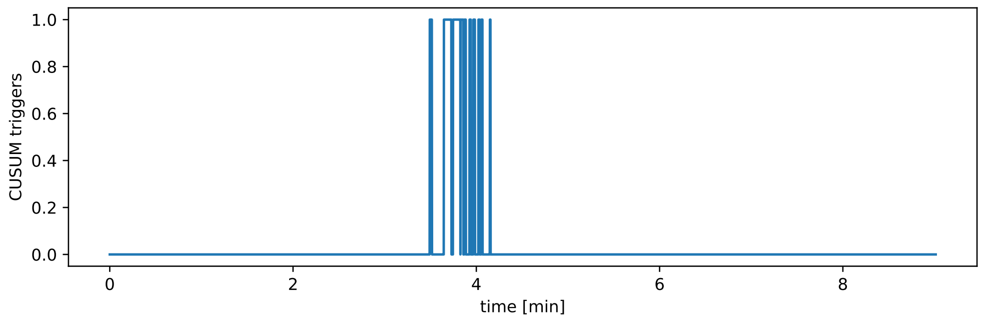

Figure 6.15: Generated triggers based on CUSUM function response.

Speed results of the tested methods are listed in Table 6.1. It is visible that all methods performance are capable of running hundreds or even thousands of iterations per second.

Method	**Processing time [ms]**
CUSUM	0.017
BOCD	0.981
RulSIF	1.386
DifRatio	0.374
VarRatio	0.022

Table 6.1: Results of speed of tested change point detection methods.

These results support (not confirm) the hypothesis proposed in Chapter 3 that the use of one-dimensional non-visual sensors for environment change detection and environment classification can significantly reduce the computational cost of this task.

6.2 Multilayered map generation

In this section, generated multilayered map will be shown on data from the temperature sensor and the ceiling height sensor. A basic map was created using the gMapping SLAM system. Layers were added in real-time by the proposed system described in the previous chapter. Results shown in Figures 6.17 and 6.16 are rendered using Rviz software based on the GridMap package Rviz plugin.

Data for testing described system was recorded on the described robot. In this case, it contains indoor-only runs, and it is composed of LiDAR data, wheel odometry, and non-visual sensors data. In the next paragraphs, temperature and ceiling height data from two particular runs will be discussed to describe the capabilities of sensors with and without delay.

Two maps composed of basic map and ceiling height data are shown in Figure 6.16. The first map is created during the short run when the robot went approximately 5 meters from the office to the hallway. The robot starts at the right part of the map. As the reader can see, the ceiling height is stable (yellow color) up to the moment when the robot reaches the door (purple). Then the ceiling height change again (green) in the hallway.

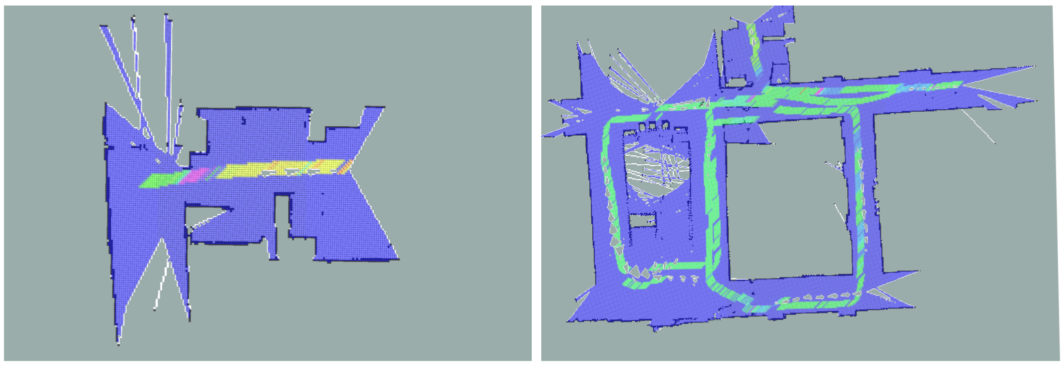

Figure 6.16: Examples of multi-layered maps composed of basic map and ceiling height data.

Similarly, the second map was created during the long-run of the robot. The ceiling height value is often without change (green color), but there are purple several times (doors), and in the office (upper part of the map), it goes to yellow color.

Based on the evidence in figure 6.16, the robot can detect changes in the ceiling height immediately. Moreover, it can detect doors, hallways, office based on the known distances. On the other hand, the used ultrasonic sensor has a bigger range of sense in the horizontal axis than desired. Thus, it sometimes measures the reflection of a different object than the ceiling of the room – blue parts of the robot trajectory in the long run map.

In Figure 6.17 there are maps created based on the temperature sensor BME280. Like ceiling height maps, the first one is created during the short-run of the robot and the second one during the long run of the robot. The first visible thing is a delay in the change of the temperature value. It is not immediate, as in the case of ceiling height. Thus, it is not possible to detect changes between rooms, hallways, etc.

Moreover, the delay can cause problems even in the environment with small changes in the temperature. It is visible in the long run map. The temperature in the same place has different colors when the robot comes from different parts of the building. For example, place in the upper part of the map when the robot comes out of the office to the hallway (green color). When the robot reaches this place again, the temperature is different (orange color).

Thus, it should be important to apply some post-processing on the temperature data in the map to estimate the temperature – e.g., using the average value. In conclusion, the data with delay are usually useful for big changes in the signal that took a longer period – such as the transition between indoor and outdoor environment. On the other hand, they are usually not suitable for quick changes that can be detected by non-delay sensors such as mentioned ceiling height.

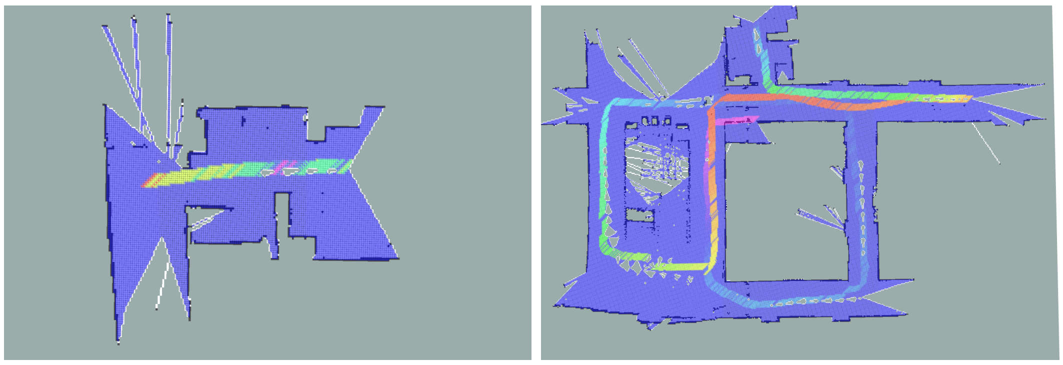

Figure 6.17: Examples of multi-layered maps composed of basic map and temperature data.

6.3 Environment Classification

This experiment is focused on the classification of the environment based on the image data. The section is composed as follows. In the first part, the dataset and the reasons for its usage are described. In the second part, results obtained based on the classic approaches – see Chapter 4.2.2. In the third part, results obtained based on the neural nets are described – see Chapter 4.2.3. The theory for both cases is described in section 4.2.1. The fourth section is focused on the speed of the best methods from both the classic and neural nets approaches. In the last part of this section, the results are discussed

6.3.1 Dataset

In the Section 4.2.4, it was mentioned that there does not exist any large dataset that is supposed to be used directly for indoor vs. outdoor classification. Thus, I decide to use one of the datasets for the general scene classification problem. The advantages of this approach are the size of these datasets and their diversity. On the other hand, its disadvantage is that some images or even whole classes can be ambiguous. It is hard to tell whether they belong to an indoor or outdoor environment – e.g., the image of pizza in the class pizzeria or both indoor and outdoor images in the class airport.

For my work, I chose between Places 365 and Miniplaces dataset. I chose Miniplaces because the size of the dataset is big enough for training neural nets, and it is not too much for classic approach algorithms at the same time. Examples of the Miniplaces dataset are shown in Figure 6.18.

All results mentioned in the next subsections are trained on this dataset. Dataset is divided into 90000 images for training, 10000 for testing, and 10000 for validation. Moreover, it was necessary to create a file containing a mapping from Miniplaces classes to either indoor or outdoor label.

Figure 6.18: Examples of Miniplaces database.

6.3.2 Classic approaches

To evaluate classic approaches based on Chapter 4.2.2, I implemented a pipeline for training and testing in Python programming language using the Scipy[4] package. In the pipeline, it is possible to choose between various classifiers such as kNN, SVM, etc. Moreover, it is possible to enable the PCA algorithm for dimension reduction

In the experiment, I tested more than 280 combinations of image descriptors, classifiers, and its settings. Of course, combinations include classification schemes described in Chapter 4.2.2 as well. For clarification, they are the Basic scheme, Multi-Scale scheme, and Two-stage scheme.

Image splitting into the grid is also used in the case of the Basic scheme. In particular, grids of 2×2, 3×3, and 4×4 tiles were used. Each splitting setting was tested in several variants of description vector length – e.g., 4, 8, 16. The feature vector of the image is concatenated from the feature vector calculated for each tile. In the case of one channel image data, the length of the calculated feature vector is computed as follows

$$l(\mathbf{x}) = M \times N \times D \tag{6.4}$$

where M and N are the numbers of tiles in horizontal and vertical direction. Variable D is the length of a single tile feature vector. For three channels, the number has to be multiplied by three. All mentioned variations of feature vector computation are supposed to examine their influence on classification accuracy. Multi-Scale and Two-Stage schemes are used in the form as they were described in Chapter 4.2.2. Again, various lengths of feature vectors are used to examine its influence.

To compute feature vectors following methods were chosen. To examine the influence of color space on the classification, histograms of RGB, HSV, LUV, and OHTA OHTA[6] color

[4]https://www.scipy.org/

spaces were used. Texture based approaches are represented by Gist[209], Census Transform (Centrist)[200], Global Binary Patterns (GBP)[203], Weighted Histogram of Gradient Orientation (WHGO) [206], Wavelets [190] and by the approach where texture descriptor was used on the image containing edges detected in the source image.

For training, the following classifiers (that are described in Chapter 4.2.1) were used. As the baseline, Naive Bayes was used. Other classifiers are K-Nearest Neighbors, Decision Trees, Linear SVM, and SVM with Radial Basis Function (RBF) based kernel.

In Tables 6.2, 6.3[5] and 6.4, there are Top 20 results of each classification scheme. The complete list of results is published online in repository[6] on my GitHub account.

Approach	**Des. Length**	**Classifier**	**Accuracy**
Gist	960	SVM	85.44%
Gist-PCA-512	512	SVM	85.43%
HSV-Centrist-256	1024	SVM	84.83%
RGB-Centrist-256	1024	SVM	84.66%
Gist-PCA-256	256	SVM	84.65%
HSV-Centrist-128	512	SVM	84.6%
RGB-Centrist-128	512	SVM	84.04%
HSV-Centrist-64	256	SVM	83.81%
RGB-Centrist-64	256	SVM	83.47%
HSV-Centrist-32	128	SVM	83.13%
RGB-Centrist-32	128	SVM	83.04%
Centrist-4x4x16	256	SVM	82.44%
Gist-PCA-128	128	SVM	82.29%
Centrist-3x3x16	144	SVM	82.06%
HSV-4x4x16	768	SVM	81.82%
Hog-32-64	32	SVM	81.70%
HSV-4x4x8	384	SVM	81.56%
RGB-4x4x16	768	SVM	81.40%
RGB-4x4x8	384	SVM	81.26%
HSV-3x3x16	432	SVM	81.08%

Table 6.2: Results of TOP 20 basic classification approaches

In the case of the Basic scheme, all the best results were achieved with the SVM classifier. In the Top 20, there are five color based descriptors, seven texture-based descriptors, and eight combined descriptors. Besides classification accuracy, there are values of feature vector length.

[5]Note: BAG prefix means Bootstrap Aggregation Machine Learning approach
[6]https://github.com/neduchal/io_classification_experiment

As the reader can see, the best descriptor for the Basic scheme is Gist. It achieved an accuracy of 85.44% and has a feature vector of lengths 960 bins. By reducing dimension to 512 bins using PCA, it was achieved the same accuracy with a shorter descriptor. They were the only methods that reach 85% in the Basic scheme.

Other methods do not achieve the same accuracy as Gist, even with a longer feature vector. Thus, it makes Gist the most suitable descriptor for the Basic scheme. On the other hand, the method called GBP had very poor results in my experiments, which is in contrast to the original paper presenting this method.

Approach	**Des. Length**	**Classifier**	**Accuracy**
Centrist-64	1344	SVM	85.48%
Centrist-128P	3968	SVM	85.34%
Centrist-256P	7936	SVM	85.05%
Centrist-32	672	SVM	84.42%
Centrist-256P-pca-512	512	SVM	83.20%
Hog-32	672	SVM	82.84%
Centrist-64	1344	BAG16-SVM	82.66%
Hog-64	1344	SVM	82.61%
Centrist-64P	1984	LSVM	82.27%
Centrist-256P-pca-256	256	SVM	82.22%
Centrist-16	336	SVM	82.02%
Centrist-256P-pca-128	128	SVM	81.94%
Hog-16	336	SVM	81.57%
BAG64-Centrist-64	1344	BAG64-SVM	81.14%
Centrist-256P-pca-64	64	KNN	81.09%
Centrist-32P	992	LSVM	80.75%
Centrist-16P	496	LSVM	78.75%
Hog-8	168	LSVM	77.89%
Centrist-4	84	SVM	77.20%
Centrist-8	168	SVM	76.72%

Table 6.3: Results of TOP 20 Multi-scale classification approaches

Result for Multi-Scale scheme shows that texture-based approaches are in the lead in this scheme. The best method, in this case, is Centrist in several variations. Particularly in various histogram lengths of the Centrist description and even in two versions of the Multi-Scale scheme. Approaches based directly on the Centrist paper[200] are denoted P at the end of the approach name.

The results of the best Multi-Scale methods are similar to the best Basic scheme methods. The disadvantage of Multi-Scale methods is the usually bigger length of feature vector – e.g., 7936 in Centrist-256P. In summary, based on the evidence of similar results, it seems to be better to use Basic scheme methods.

Approach	**Des. Length**	**Accuracy**
HSV-Centrist-256-D	32 (256)	91.87%
RGB-Centrist-256-D	32 (256)	91.61%
HSV-Centrist-256-DS	2 (256)	91.11%
HSV-Centrist-128-D	32 (256)	90.91%
RGB-Centrist-256-DS	2 (256)	90.85%
BAG16-HSV-Centrist-CT-256	32 (256)	90.66%
BAG1-HSV-Centrist-CT-256	32 (256)	90.61%
HSV-Centrist-CT-256	32 (256)	90.60%
HSV-Centrist-256	32 (256)	90.47%
RGB-Centrist-128-D	32 (256)	90.27%
BAG1-HSV-Centrist-256	32 (256)	90.13%
RGB-Centrist-256	32 (256)	90.13%
BAG16-HSV-Centrist-256	32 (256)	90.11%
HSV-Centrist-64-D	32 (256)	89.99%
HSV-Centrist-128-DS	2 (256)	89.96%
RGB-Centrist-CT-256	32 (256)	89.95%
HSV-Centrist-CT-128	32 (256)	89.35%
RF-HSV-Centrist-256	32 (256)	89.32%
RGB-Centrist-128-DS	2 (256)	89.28%
HSV-Centrist-128	32 (256)	89.10%

Table 6.4: Results of TOP 20 Two-stage classification approaches

The Two-Stage results show that classifiers trained in the Two-Stage scheme have better accuracy than the previous one. The best accuracy is achieved by combining the HSV color space histogram with 256 bins for every channel and the Centrist texture descriptor. The accuracy of the method is 91.87%.

In all cases, only an SVM classifier was used because of experience from Basic and Multi-Scale schemes where the SVM outperforms other classifiers. The Two-Stage scheme results are the best from the classic classification approaches. Thus, it is the best choice in the case of offline classification. As will be discussed in the following subsection focusing on the time consumption of the approaches, Two-Stage methods are usually improper for real-time systems.

6.3.3 Neural nets

In this section, a deep learning approach to the indoor vs. outdoor classification will be described. In this particular experiment, transfer learning is used as a good approach – based on the benefits mentioned in paper [224] – to train a neural network for this particular task. Thus, all used network architectures were pretrained on the ImageNet database [225]. The last layer (fully-connected with 1000 outputs) was replaced by the layer with two outputs – indoor, outdoor – and randomly initiated weights.

In this experiments, eleven network architectures was trained. In particular they are VGG16 [226], ResNet-50 [227], ResNetWide-50 [228], ResNeXt-50 [229], InceptionResNet [230], Xception [231], InceptionNetv4 [230], DenseNet [232] and three Big Transfer (BiT) [233] based architectures.

VGG16 architecture is a baseline because it is a common architecture for a wide range of classification problems nowadays. The rest of the architectures can be divided into three groups. The first one is based on ResNet architecture – or more precisely, on utilizing residual skip connections. The second one is based on the inception module, which computes multiple convolution filters on the same network level instead of stacking them sequentially. The last group is based on BiT models, which utilizing upstream and downstream components proposed by Google.

On the contrary to the classic approach, Neural Networks expects the whole image on its input. In the case of the Miniplaces dataset, it is an RGB image of size 128 × 128 pixels. Although the size of the dataset is big enough for the classic approach, it is not sufficiently large for NNs. Therefore, augmentations had to be used during the training stage to enrich the dataset. In particular horizontal flip and random crop with resizing to 128 × 128 pixels were used.

Training parameters are set as follows. Training is stoped after 20 epochs. Mini-batch has size 64. SGD optimizer is used with a learning rate $l = 0.001$. Moreover, momentum $m = 0.9$ is used as well. Cross-entropy loss is used. During training, the optimizer can fine-tune all parameters because its freezing did not lead to satisfactory results.

The accuracy of all architectures is listed in Table 6.5. It worth mentioning that all neural networks overcome the accuracy of classifiers learned using the classic approach. In other words, the worst trained neural network accuracy starts at the point where is the accuracy of the best classic approach classifier.

The best results are achieved by the usage of the BiT training protocol on Resnet-50 and Resnet-101 architectures. It improves accuracy by more than 1.5% against the same architectures without BiT. The best architecture is the largest network based on Resnet-50 with BiT (denoted as BiT-M-R50x3). It achieved 96.17% on the test set.

Computational cost

The accuracy is not only one property that is necessary to evaluate in this case. The approach has to run in (almost) real-time[7]. Thus, the best three methods from the classic approach and five neural networks were tested on its inference time. In the case of the classic approach, it consists of data preprocessing and classification itself. Table 6.6shows the speed result of the top three classic approaches from all mentioned classification schemes.

The first notable information is that the Two-stage scheme is not suitable to run in real-time.

[7]All approaches were tested on computer with Intel Core i7-8750H CPU and GTX 1060 GPU.

Architecture	**Development set**	**Test set**
BiT-M-R50x3	96.70%	96.17%
BiT-M-R101x1	96.44%	95.49%
BiT-M-R50x1	96.39%	94.97%
DenseNet-161	94.65%	93.64%
ResNetWide	94.57%	93.55%
ResNeXt	94.56%	93.49%
ResNet-50	94.33%	93.34%
Xception	94.00%	93.17%
VGG16	93.98%	92.56%
InceptionResNet	93.38%	92.49%
InceptionNetv4	93.37%	91.87%

Table 6.5: Results of neural network based approaches. Sorted by test set accuracy.

Approach	**Des. Length**	**Scheme**	**CPU inf. time (ms)**
Gist-960	960	Basic	71.6
Gist-PCA-512	512	Basic	54.8
HSV-Centrist-256	1024	Basic	10.1
Centrist-64	1344	Multi-Scale	68.8
Centrist-128P	3968	Multi-Scale	189.7
Centrist-256P	7936	Multi-Scale	256.7
HSV-Centrist-256-D	32 (256)	Two-Stage	1125.4
RGB-Centrist-256-D	32 (256)	Two-Stage	1111.1
HSV-Centrist-256-DS	2 (256)	Two-Stage	1194.5

Table 6.6: Speed comparison of classic approach methods.

Its accuracy is better than the accuracy of other schemes, but its run time is above 1 second. Significantly lesser computational cost is measured in the case of both Basic and Multi-Scale schemes. Thus, the best trade-off between classification accuracy and computational cost is represented by the Gist descriptor for the Basic scheme and 64 bins length Centrist descriptor for the Multi-Scale scheme.

Similarly, inference times of Neural network based approaches are shown in Table 6.7. There are listed both CPU and GPU inference times. It's worth mentioning that the largest architectures – such as ones based on Inception – are omitted from this comparison.

In the case of GPU inference time, all mentioned are capable of running in real-time. On the other hand, in the case of CPU – still, the real-life situation when the robot does not have GPU – only network BiT-M-R50x1 can run at least 5 frames per second. Thus it offers the best trade-off between accuracy and computational cost for usage on CPU.

Method	**# of Param.**	**CPU inf. time (ms)**	**GPU inf. time (ms)**
VGG16	134277186	258	12.4
DenseNet-161	26476418	223	21.6
BiT-M-R50x1	23504450	152	9.8
BiT-M-R101x1	42496578	1238	26.1
BiT-M-R50x3	211186370	708	15.6

Table 6.7: Comparison of the # of parameters and single image average inference times.

6.3.4 Edge detection experiment

At the beginning of this part, it is worth mentioning that this experiment is older than the classification experiment above and uses different – and smaller – datasets. The dataset was created by combining images from the dataset mentioned in Subsection 5.4 and the Kitti dataset[8]. The dataset consists of 18980 images equally distributed to indoor and outdoor classes. Its purpose was to examine the influence of edge detection on the training process and the performance of a simple neural network (shown in Figure 6.19).

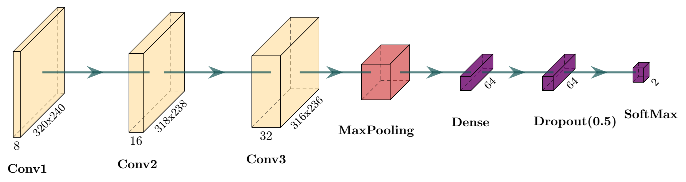

Figure 6.19: Simple Neural Network Architecture

The results are shown in Table 6.8. I tested several approaches. All methods have the same number of epochs, same learning rate, and same batch size. Only Canny and Roberts edge detector worth mentioning. They had significantly better accuracy on the test set than other methods or methods without edge detection.

Input preprocessing	**Training set accuracy**	**Test accuracy**
Without preprocessing	50.43%	51.34%
Fourier Transform	88.10%	50.45%
Sobel	94.25%	49.36%
Roberts	95.93%	86.39%
Canny	93.34%	92.45%

Table 6.8: Results of neural network trained on data from various edge detection methods.

It seems that edge detection improves learning speed in the case of small nets and small

[8]http://www.cvlibs.net/datasets/kitti/eval_odometry.php

datasets. On the other hand, larger networks trained on larger datasets can achieve the same or better accuracy without any preprocessing. Thus, they should be more robust than networks trained only on edges.

6.4 Proposed system design test

In this section, the system concept proposed in Chapter 5 will be tested. The goal of this experiment is to confirm the hypothesis proposed in Chapter 3. The experiment is based on an analysis of a recorded dataset from two points of view. The first one is a verification of the proposed system concerning its proper working. The second point of view is statistical. In other words, it is focused on the system accuracy and speed. In both cases, the results will be compared with the system design proposed in paper [169] based on image classification only – it will be referred to as the baseline system.

In the first part of this section, the system's results will be evaluated qualitatively on three records with various levels of difficulty. In three cases, the distance sensor will be used. In the case of the most difficult record, the temperature sensor will be used as well to show results on the sensor with a time delay (for details, see Chapter 4.1).

The second part is focused on quantitative evaluation. It consists of average classification accuracy on the recorded dataset and average runs per second of presented methods. Moreover, several runs will be analyzed to show speed performance differences between proposed and baseline systems.

6.4.1 Qualitative evaluation

The first evaluation criterion is based on human experts, which compares ground-truth data with the response of both the proposed and the baseline systems. It will be done on two shorter and one long records. All records start in an indoor environment. Then the robot moves through the building and goes outside of the building. Finally, it returns inside after some time.

Every single record is recorded with a mobile robot presented in Chapter 5.3 in a different place (different building, different door space) and with a different difficulty level.

Record 01

The first record is the simplest one. The camera is pointed to the front of the robot with no camera occlusion and balanced white color. In Figure 6.20, there are four example frames from the record.

Figure 6.21shows a graph with ground-truth (GT) information for every timestep of the record. The robot moves indoor for approximately the first 10 seconds of the record. For the next 30 seconds robot moves in an outdoor environment. Finally, it comes back to the indoor environment. The situation is more complicated. The exact timestamp of the change

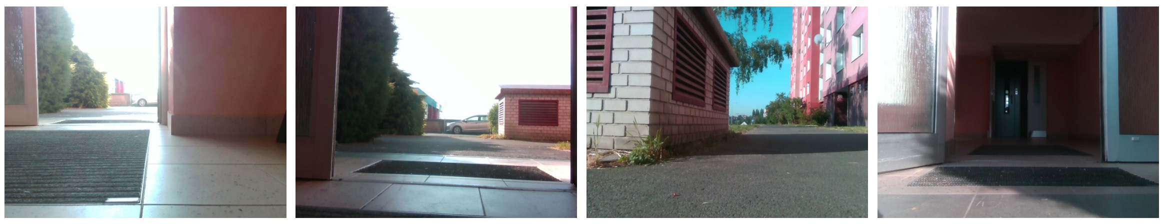

Figure 6.20: Examples frames from record 1

between environments is hard to determine, even for human experts. It will be shown in Figures 6.22 and 6.23 that distance sensor and image classifier usually predict a change in different timestep than GT information.

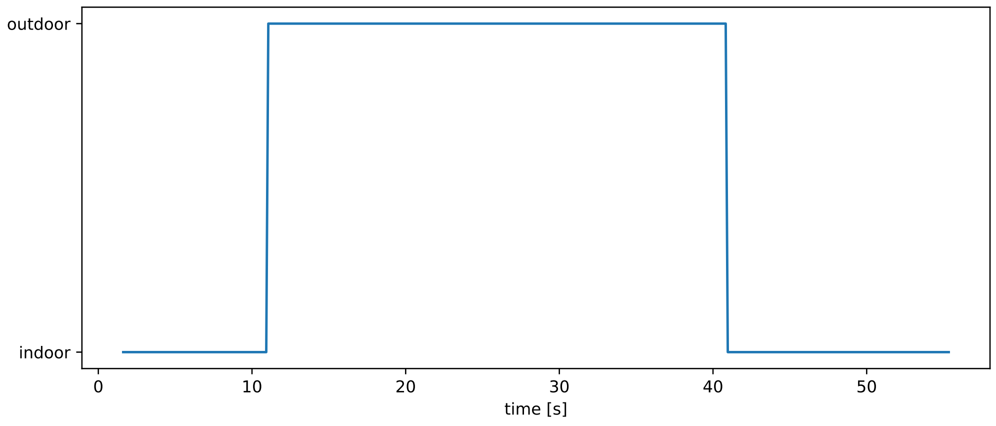

Figure 6.21: Groundtruth labels for Record 1

The distance sensor data shown in Figure 6.22 are similar to GT data, but the value changes several times on the edge between indoor and outdoor environments. Moreover, the change from indoor to outdoor occurred approximately three seconds later than GT data. The change in the opposite direction, on the other hand, occurred 1 second sooner than GT data.

The result of the baseline system to detect change between environments is shown in Figure 6.23. It is visible that the system detects transition from the indoor to the outdoor environment in the same second as GT data. In all Figures of this type, the blue data line is the system result, and the orange data line is GT data. On the other side, it changes back significantly sooner than GT data – As mentioned above, it is not usually an error because it is difficult to select a single frame where the transition occurred. It is influenced by the scene viewed by the camera, where the building entrance is visible.

Moreover, there are also several changes in timesteps where no change occurred. It is caused by the accuracy of the classification system, which can incorrectly classify blurred data. Overall, the system behaves as expected.

Figure 6.24 shows the triggers generated by the ECD part of the system – using the DifRatio function in this case. There are two groups of triggers in the graph corresponding with the

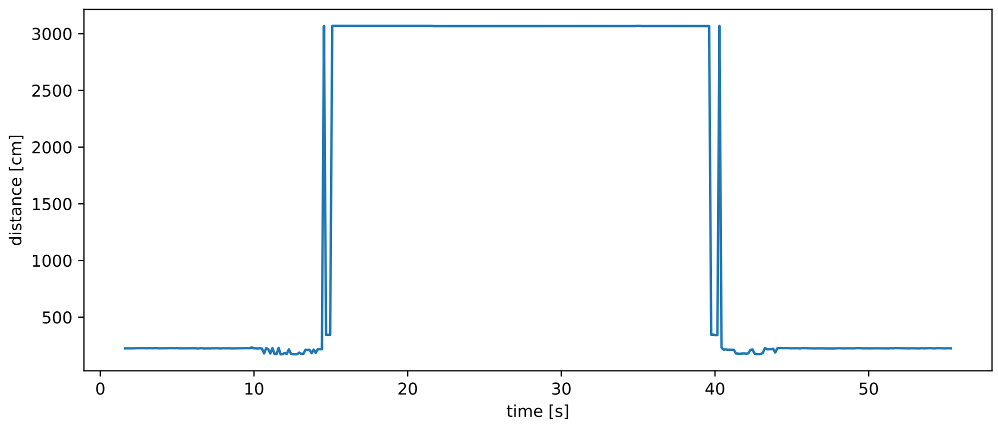

Figure 6.22: Distance data for Record 1

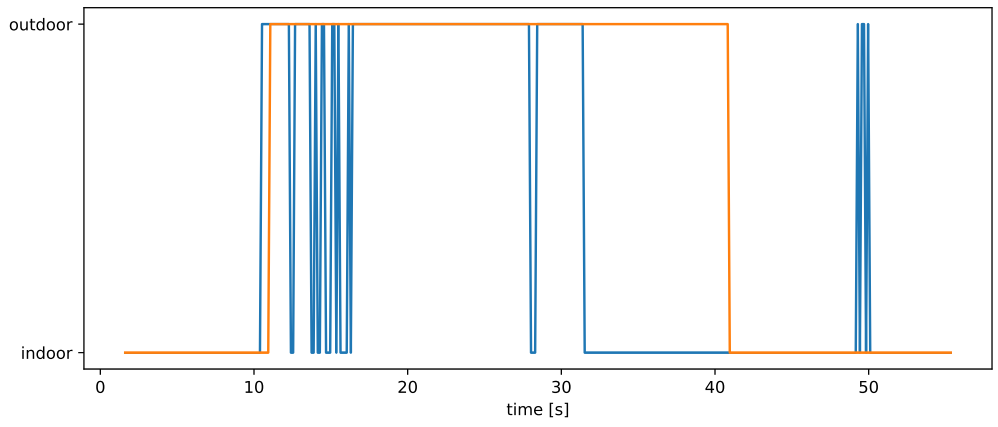

Figure 6.23: Baseline system results for Record 1

first and second transition between environments.

Finally, the result of the proposed system is shown in Figure 6.25. It is visible that the system reacts only in timesteps where triggers are generated. Thus, the first transition occurred later than in the case without ECD triggers. On the other hand, there is much fewer wrong classification in this case.

In more detail, the baseline system's accuracy – using ResNext Neural Network architecture – is 77% against GT data. In the case of the proposed system, it is 91%. It is worth mentioning that real accuracy is better because it is compared with strict GT data containing exact timesteps labeled by a human expert.

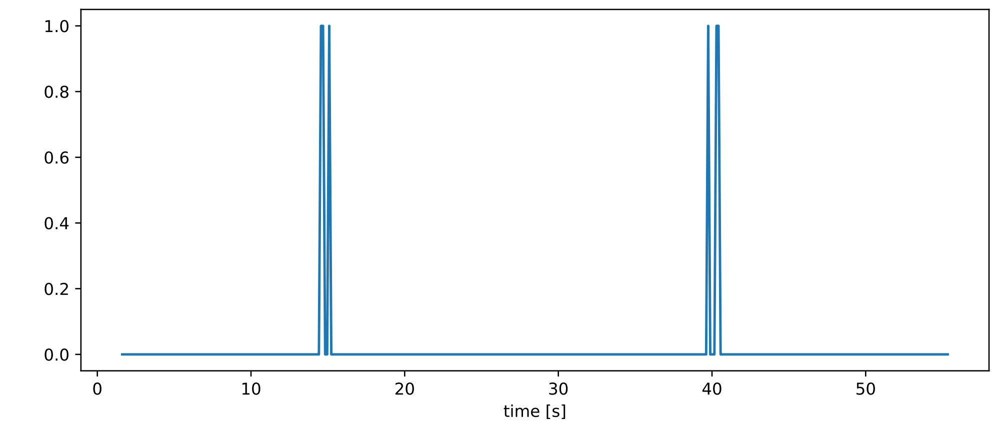

Figure 6.24: Generated triggers from distance data for Record 1

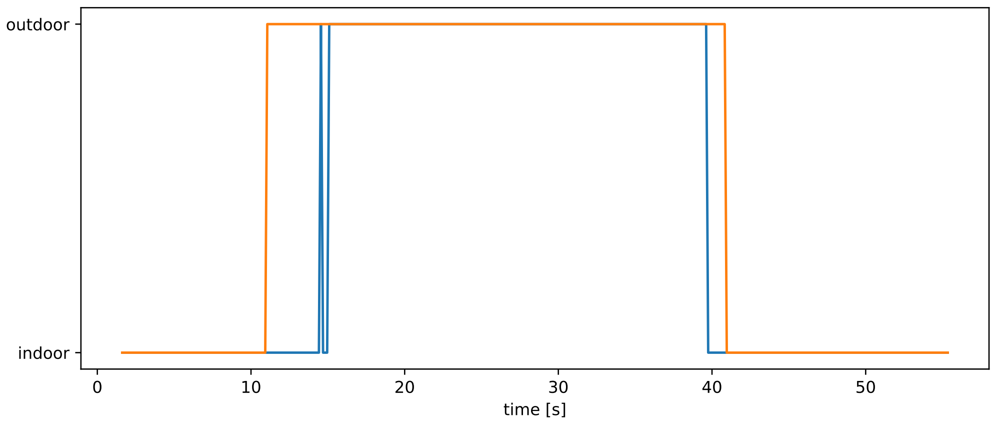

Figure 6.25: The proposed system result for Record 1

Record 02

The second record is similar to the first record, but the camera is occluded by black wire, as shown in 6.26.

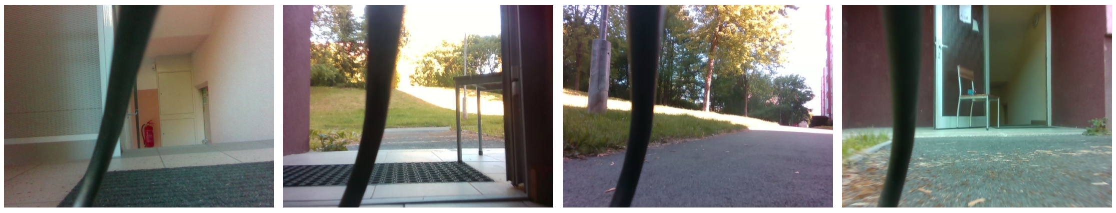

Figure 6.26: Example frames from Record 2

Figure 6.27 shows a graph with ground-truth (GT) information for every timestep of the record. The robot moves indoor for approximately the first 24 seconds of the record. The next 24 seconds robot moves in the outdoor environment. Finally, it returns to the indoor environment.

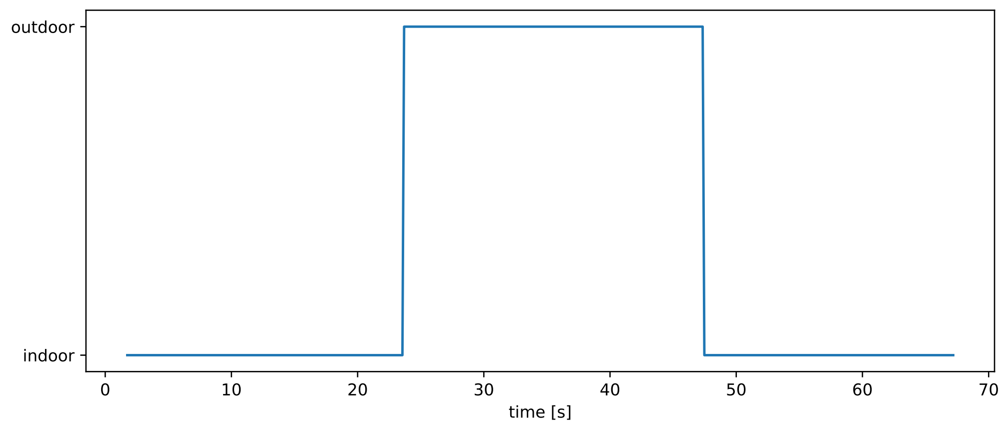

Figure 6.27: Groundtruth labels for Record 2

The distance sensor data shown in Figure 6.22 shows the transition between environments. Moreover, there are four peeks after 55 seconds of the record. It is an error in the data because no transition occurred in that timestamps. Thus, the proposed system will generate triggers for the classification module.

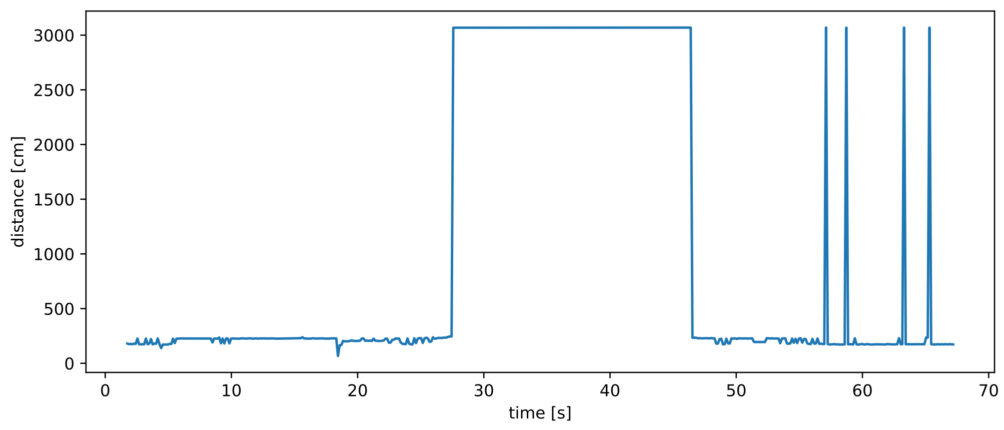

Figure 6.28: Distance data for Record 2

The baseline system result is shown in Figure 6.29. The system behaves similar as in the case of Record 1. The system behaves similarly as in the case of Record 1. There is a more wrong classification, which can be caused by two factors. The first one is the black wire in the camera view. The second one is sun glare visible in examples in Figure 6.26. On the other hand, the system behaves reasonably accurately.

The triggers generated from distance sensor data are shown in 6.30. Two triggers are corre-

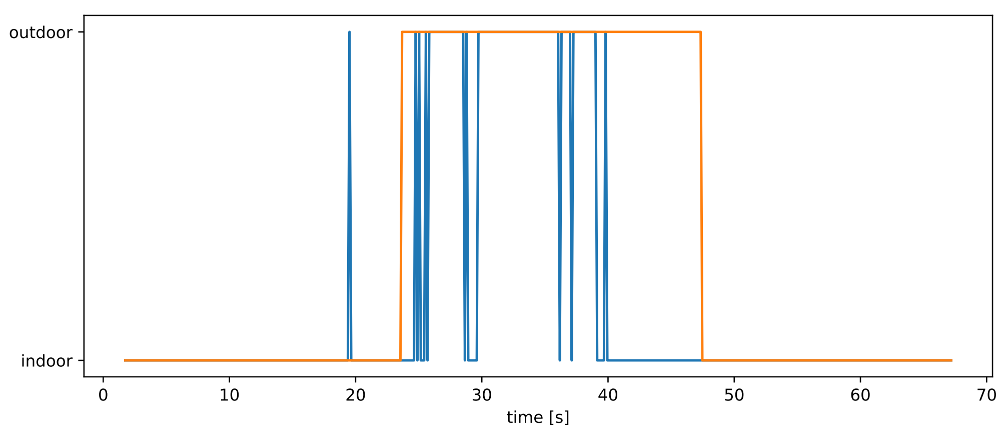

Figure 6.29: The baseline system result for Record 2

sponding to the transition between environments. Moreover, after 55 seconds of the record generated based on the error values, there are four false-positive triggers. Thus, the image classification is also performed in these timesteps.

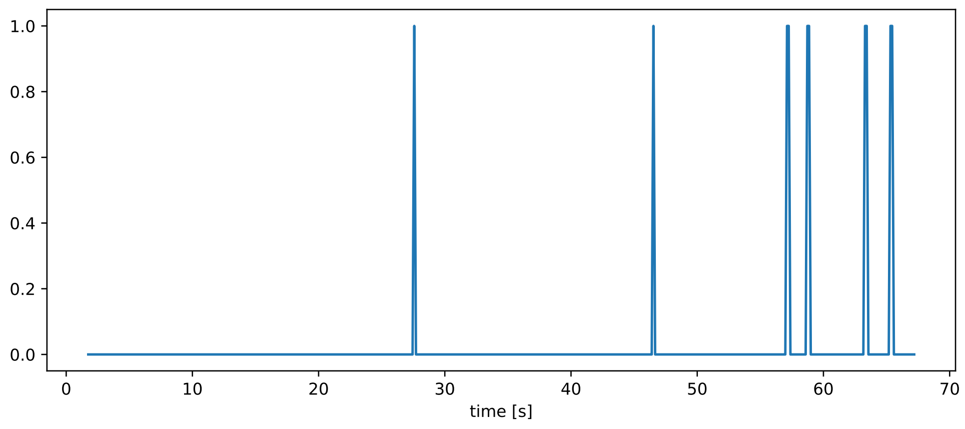

Figure 6.30: Generated triggers for Record 2

As shown in Figure 6.31, false-positive triggers do not influence the results because the image classification module classifies them correctly. Moreover, the results follow GT data. The proposed system again behaves reasonably.

The accuracy of the baseline system was 84% in this case. The proposed system achieved an accuracy of 93%. Similarly, as in record 1, it reduces errors when the robot moves through one environment without transition.

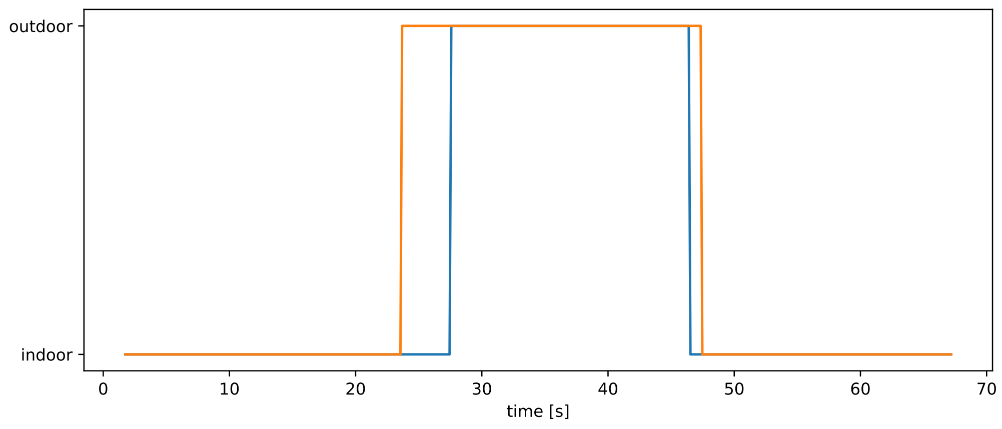

Figure 6.31: The proposed system result for Record 2

Record 03

The third record is the hardest one considering the training data of the image classifier. The mounted camera images are color unbalanced – more significant red channel – and the camera itself is pointed 20 degrees up in contrast to previous records. Moreover, the building contains many windows, which allows the robot to see the outdoor environment. It can confuse the image classification module. Examples of recorded frames are shown in Figure 6.32.

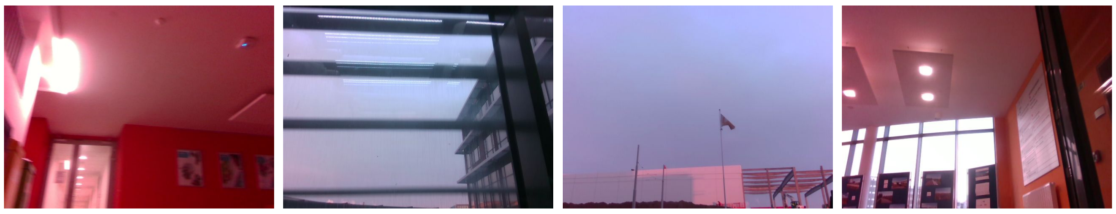

Figure 6.32: Examples frames from record 3

In this case, the robot moves through the indoor environment most of the time of the record. The robot starts on the 4th floor of the building. Then, it moves to the elevator and the ground floor. In the next part, the robot moves in front of the building, turn around, and goes back to the elevator and the 4th floor. The ground truth data is shown in Figure 6.33.

Figure 6.34 contains distance sensor data for Record 3. They are more interesting than in the previous cases. There are visible changes in the ceiling height in the data, which are different on the 4th floor – approximately 3 meters and the first floor – approximately 5 meters. Moreover, there is the visible lower ceiling height of the elevator around timesteps 100s and 400s. The robot is outside of the building between timestep 220s and 260s of the record. The rest of the peeks in data are measurement errors.

The difficulty of the record is visible in Figure 6.35. There is a lot of the wrong classification caused by different camera angles and by many situations when the robot looks outside

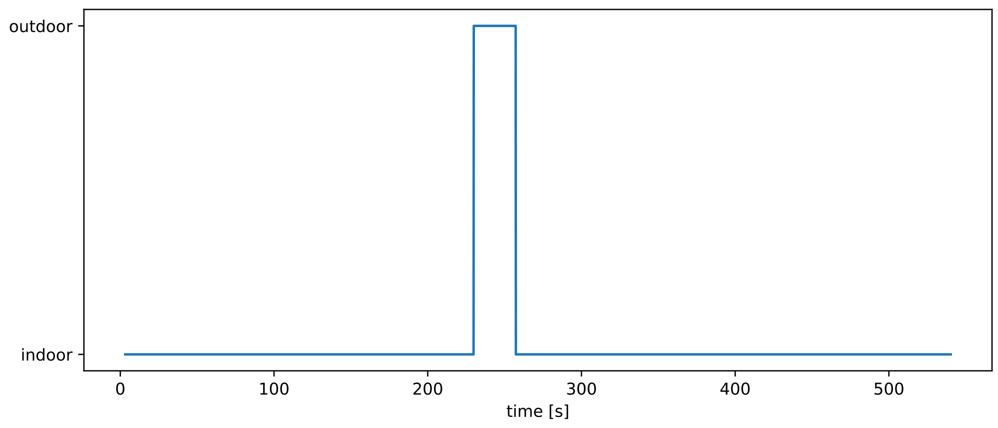

Figure 6.33: Groundtruth labels for Record 3

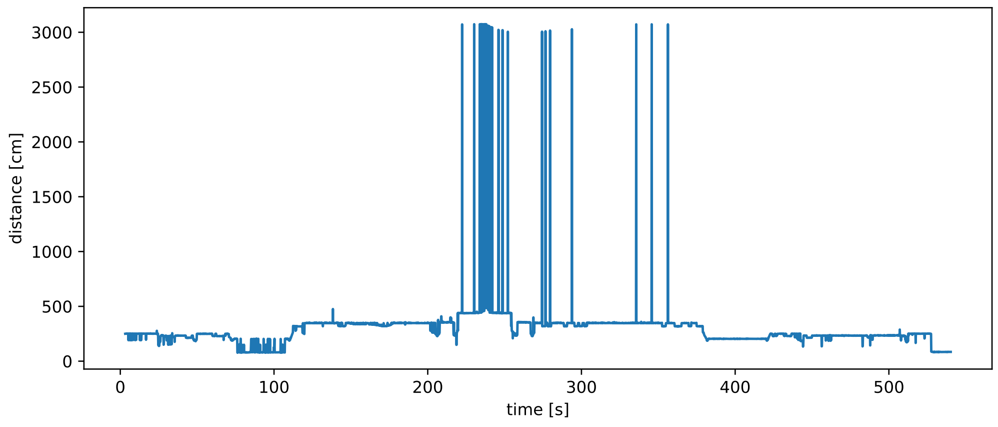

Figure 6.34: Distance data for Record 3

through the window. For example, it is a case of the wrong classification around timestep 100s of the record. Moreover, an industrial look of the outdoor environment in front of the building also causes wrong classification – indoor instead of outdoor.

In Figure 6.36, there are again triggers generated by the change detection module. It generates a lot of triggers when the robot is outside of the building. It is caused by the fact that there is an outdoor canopy at a certain distance from the building. Thus, the distance sensor measured no roof in one timestep and outdoor roof in another step. The result is that the image classification module is triggered more times than in previous cases. There are also several triggers after timestep 260s of the record. They corresponded with the error measurement mentioned above.

Finally, Figure 6.37 shows the result of the proposed system. It eliminates most of the wrong classifications. On the other hand, the classification module generates only two short sequences for the outdoor environment.

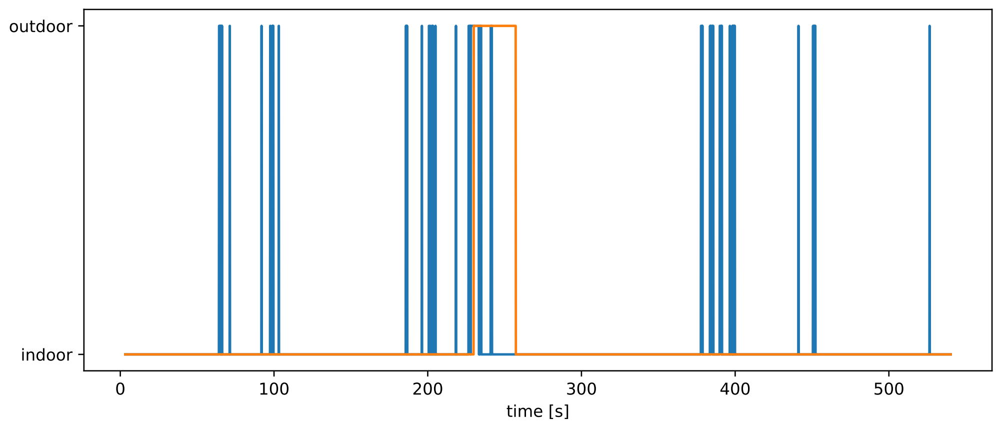

Figure 6.35: The baseline system result for Record 3

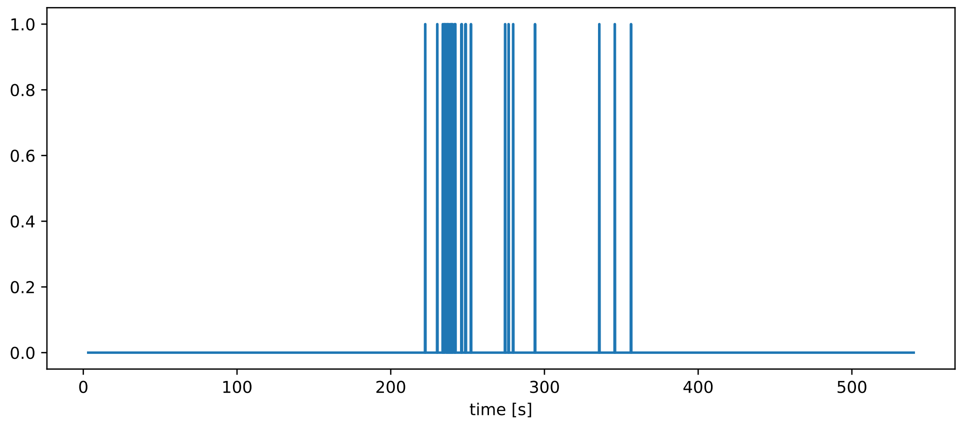

Figure 6.36: Generated triggers for Record 3

Overall, the accuracy of the baseline system for this record is 95%, and the accuracy of the proposed system is 97%. The high accuracy of both systems is caused by the record length. Thus, the length hides the number of wrong classifications because there are many more correct classifications.

In the case of this record that was recorded in winter when the difference between indoor and outdoor temperature was sufficient, the temperature sensor data can also be used for trigger generation. Temperature data are shown in Figure 6.38. The temperature decreases slowly in the first 200 seconds of the record. It is caused by the measurement delay of the temperature sensor when the sensor gradually adapts to the temperature of the environment. Suddenly, the temperature starts to decrease after the robot moves through a door to the outdoor environment.

Similarly, the temperature starts increasing around timestep 250s of the record. The important is that the temperature was not stable at the time of returning to the indoor environment.

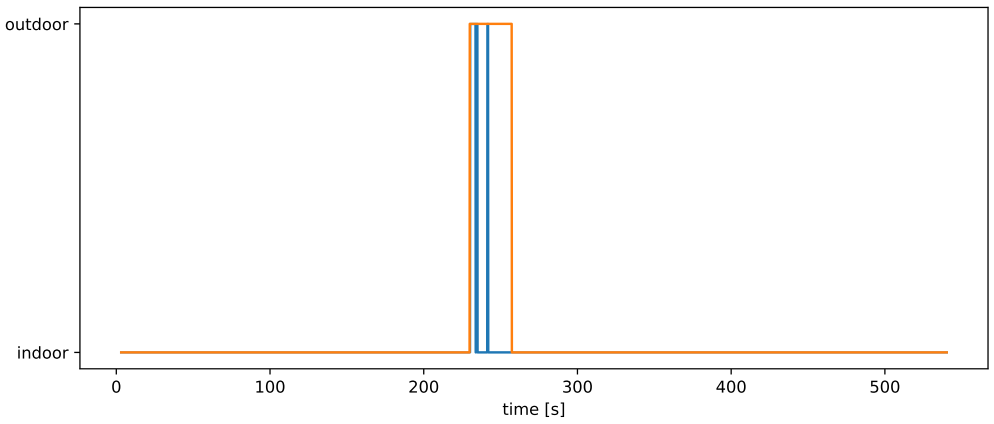

Figure 6.37: The proposed system result for Record 3

Then the temperature data gradually increasing with a slowing tendency. The temperature sensor's behavior can causes problems for the change detection when it is set to too little or too sensitive to the temperature changes.

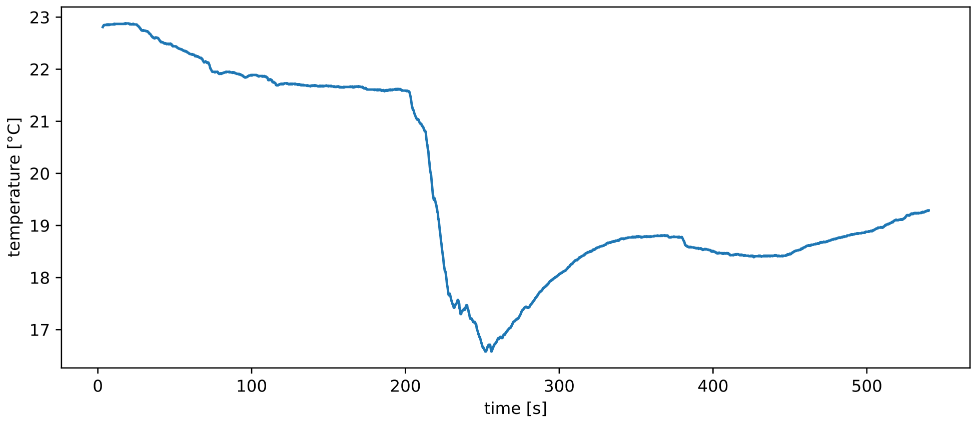

Figure 6.38: Temperature data for Record 3

Triggers generated by the DifRatio method based on temperature data are shown in Figure 6.39. It is visible that it generates several triggers in the correct part of the record. Moreover, one trigger is generated when the robot moves to the elevator, which temperature is lower than in the rest of the building.

The result of the proposed system based on temperature data is shown in Figure 6.40. It classifies the outdoor environment between timestep 200s and 250s. It is reasonable behavior because the premature switch to outdoor and back to the indoor environment is caused by the change in temperature, which occurred before the change in GT data created from visual information. The accuracy of the proposed system is, in this case, 95%, which is worse than in the case of a distance sensor, but it is identical to baseline system results.

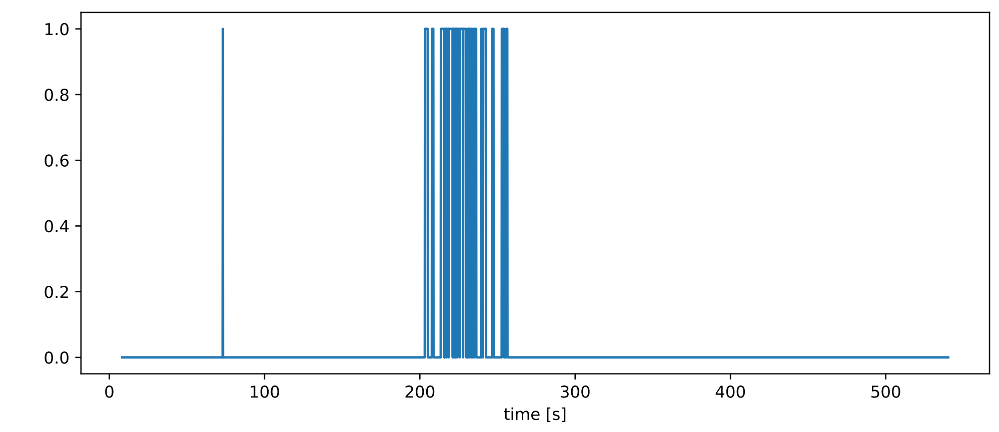

Figure 6.39: Generated triggers from temperature data for Record 3

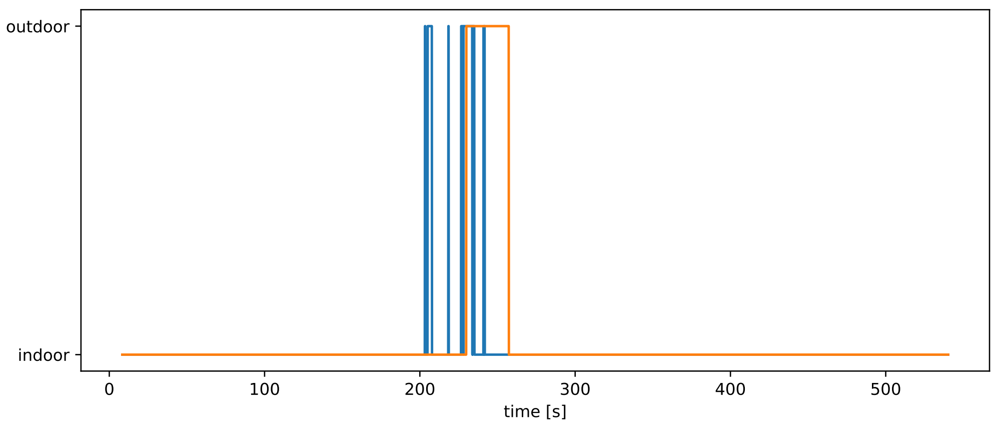

Figure 6.40: The proposed system result for Record 3 – temperature data.

It is visible from the presented data that the baseline system and the proposed system are usually similar in terms of accuracy. In the next section, the average accuracy and speed of each type of system will be discussed.

6.4.2 Quantitative evaluation

The results presented in this section are divided into two parts. In the first part, the baseline system's average accuracy based on the classic and NN classification approach will be compared with the proposed system. As will be discussed, the results suggest better accuracy of the proposed system are not as clear-cut as they seem. On the other side, the speed performance results shown in the second part of this section suggest that the proposed system is more suitable for real-time processing on the mobile robot. It worth mentioning that these tests were performed on a computer with Intel Core i5-6500 CPU and NVIDIA GeForce GTX

1050 Ti graphic card. The number of image frames was reduced to 10 per second to ensure better readability of the presented graphs.

Table 6.9 shows the average accuracy[9] of both the baseline system and the proposed system on the recorded dataset. As it is visible, the proposed system's accuracy is higher than the baseline system in both cases – Classic approach and NN approach.

Classification Approach	**Baseline system**	**Proposed system**
Classic approach	54.66%	88.36%
Neural Networks	84.72%	93.12%

Table 6.9: Average accuracy of classic approaches and NN approaches on recorded dataset for baseline and proposed systems.

Nevertheless, it worth mentioning that the proposed system was set to perform classification only when a trigger from the change detection module is generated. It means that the system is entirely dependent on the change detection module, and it does not check the environment between the triggers. It is a reasonable approach for many cases where the sensor can be trusted – such as the ceiling height sensor –, but it could be appropriate to check the environment once every few seconds, for example, in the case of a temperature sensor with a small difference between temperatures in both environments.

It is worth mentioning that classic approach classifiers are less robust against different input data kinds than the NN approach. It is visible on the difference between accuracy achieved by baseline system for these two approaches – i.e., the difference between 54.7% for classic and 84.7% for NN approach.

Speed results are shown in Table 6.10. It is represented by runs of each method per second (RPS). It is computed for three cases of image-based classification and two cases of change detection. As visible from the table, the fastest image-based classification approach is the NN approach using GPU. It achieves 57.2 RPS on desktop PC with mentioned CPU and GPU. Thus it can run realtime. The other image classification approaches are worse. Unfortunately, the NN approach runs on CPU, and the Classic approach is significantly slower.

Moreover, the performance of the methods running on embedded hardware such as Nvidia Xavier or Jetson TX2 is worse. For example, based on experiences while testing, the GPU computation is 3-5 times slower than mentioned desktop hardware on Nvidia Xavier. This problem can be solved by using non-visual sensors processing.

The results of change detection algorithms show that it can run in significantly more RPS than image-based classification. It is caused by processing much less data than image-based methods. The processing of distance sensor data achieved almost 64 000 RPS and temperature sensor data processing more than 24 000 RPS. Thus, it is reasonable to give priority to the processing of these methods before image-based methods.

The difference between cumulative time requirements results of baseline and the proposed system is also shown in Figures 6.41 and 6.42. The first one shows the Classic approach results, and the second one the NN approach, where the blue line is the time requirements

[9]Computed on results of multiple classic and NN based image classifiers trained on Miniplaces database

Process	**Runs per second**
Image Classification, Classic Approach	11.037
Image Classification, NN Approach, GPU	57.193
Image Classification, NN Approach, CPU	8.400
Distance sensor change detection	6.397e+05
Temperature sensor change detection	2.414e+05

Table 6.10: Average runs per second of image classification, distance data processing and temperature data processing – written for both the classic approaches and neural network approaches on recorded dataset.

result of the baseline system and the orange line result of the proposed system. It is visible that baseline system time requirements are significantly higher than in the proposed system.

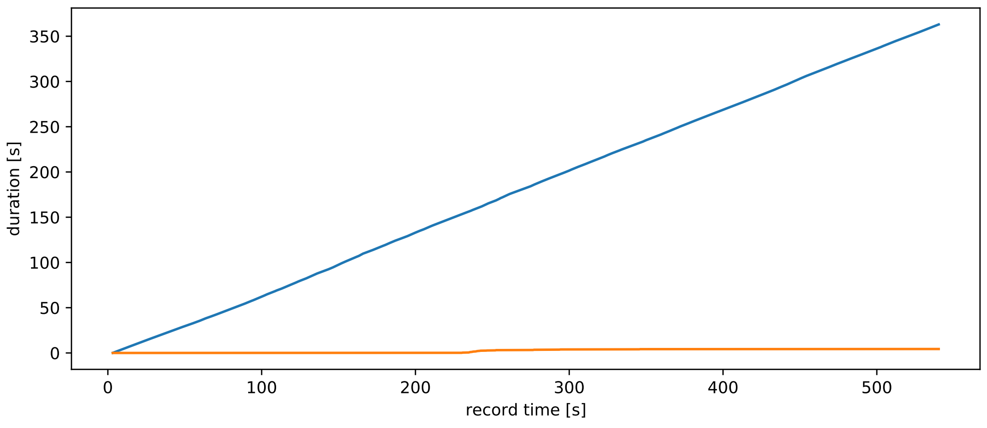

Figure 6.41: Example of time requirements results of baseline and proposed system using Centrist method.

The results support and, in fact, confirm the hypothesis proposed in Chapter 3 that the use of non-visual one-dimensional sensors can reduce the computational cost of the baseline system. As mentioned, the accuracy of the proposed system is usually better than the accuracy of the baseline system. On the other hand, complete trust in non-visual sensor information can causes problems as well. Thus, it is not the best solution to perform image-based classification only in the timesteps where triggers are generated. The better solution is to compromise between the system's speed and its robustness based on the image-based checking of the current environment. Moreover, the switching strategy analysis should be mentioned.

The simplest switching strategy was used in previous parts—the system switch immediately after receiving classification to the opposite class. During system tests, the experiments with using of switching coefficient were performed. It usually does not change the system's accuracy, but in some cases, it can increase robustness because the system does not switch quickly. Instead, smooth switching between two or multiple environments can be achieved

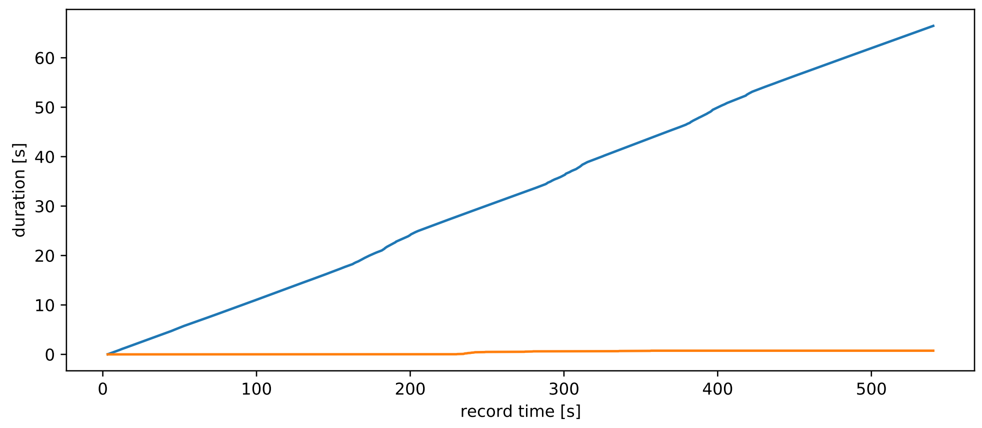

Figure 6.42: Example of time requirements results of baseline and proposed system using NN approach on GPU.

using the following approach.

The switching coefficient k can be defined as a number in the interval $(0, 1\rangle$. Thus it is supposed to slow down the switching process. For two classes, the switching strategy is defined by equation

$$s_j^t = s_j^{t-1} + k \cdot class, \qquad class \in \{-1, 1\} \tag{6.5}$$

where s_j^t is the value of the state j in step t, and $class$ is either value -1 or 1 based on the classification to the first or the second class – such as the indoor and theoutdoor class. The value of k increases the state value of the class to which the image is classified. Similarly, the other class is decreased by this number. Of course, the saturation to values 0 and 1 have to be used to prevent excessive distance increase between states. For more than two classes, following equation can be used

$$s_j^t = \begin{cases} s_j^{t-1} + k & c = j \\ s_j^{t-1} - k & c \neq j, \end{cases} \tag{6.6}$$

where c is the number of class to which data is classified, and $j \in \{1, N\}$ is the class number. It works same way as the two classes case in equation 6.5. Equation 6.6 can be rewritten into following formula

$$s_j^t = s_j^{t-1} + k \cdot c_j - k \cdot (c_j) = s_j^{t-1} + k \cdot (2c_j - 1), \tag{6.7}$$

where c_j is the member of one-hot vector C where $c_j = 1$ when the class j is selected as a class of classified image data. Values s_j^t are members of vector S^t of all states. Finally the following equation is used to select active class C^* – i.e., robots environment.

$$C^* = \arg\max_j s_j^t. \tag{6.8}$$

In other words the class with maximal state value is selected.

In the end, it worth mentioning that smaller values of k can cause problems with slow switching between environments. Thus, it is a good idea to generated multiple triggers with predefined pauses between them. The number of triggers can be defined as follows

$$tr = floor\left(\frac{1}{k}\right), \tag{6.9}$$

where $floor$ is a rounding to the nearest lower integer number.

A particular setting of the classifier, trigger generation, and switching strategy should be based on the robot's current mission, defined by the robot's environments, equipped sensors, and required behaviors. On the other hand, experiments in this chapter show that the proposed system significantly reduces computational costs and time requirements of the baseline system presented in paper [169].

Chapter 7

Conclusion

The conclusion of this thesis is divided into three main parts. The first one is focused on the summary of the thesis. In the second one, an evaluation of the defined goals is discussed. Finally, possible future work of this research is described in the last part of this chapter.

7.1 Thesis summary

The thesis is composed of 7 Chapters, but it can be merged into two main parts. The first part, including Chapters 1, 2, and 3, is focused on the theoretical background of mobile robotics focusing on the Simultaneous Localization and Mapping (SLAM) problem. This part of the thesis is closed with a Chapter defining dissertation goals based on previous chapters. This thesis's primary goal is to confirm the hypothesis that using one-dimensional non-visual sensors can reduce the computational cost of an image-based robot environment classification system. It is explained in Chapter 3 that in the rest of the thesis, the addressed problem will be investigated in the general mobile robot system instead of focusing on SLAM only systems. It worth mentioning that the thesis is also focused on the particular transition between indoor and outdoor environments because it is an important problem for the robot to operate both indoor and outdoor.

The second part focused on the addressing of defined goals is composed of the rest of the thesis. The first addressed problem is an analysis of change point detection, which can detect the transition between two environments. Then, the problem of analysis of image-based scene classification focusing on indoor vs. outdoor environment is described. The end of the theoretical Chapter 4 is focused on existing research of mobile robot systems capable of working in multiple environments.

The next chapter is focused on the description of the proposed system and assembled mobile robot designed for dataset recording. The proposed system is presented and described from two points of view. The first one has focused on the system's design, and the second one is focused on implementing such a system in the Robot Operating System. The proposed system and its parts are evaluated in the experiment in Chapter 6. Experiments on transition detection capability by non-visual sensors, image-based environment classification, and proposed system evaluation are described.

7.2 Discussion on defined goals and experiment results

In this part, all defined goals are discussed to confirm the hypothesis from Chapter 3. The hypothesis says that one-dimensional non-visual sensors can significantly reduce computational cost (especially time requirements) of the environment classification system.

This thesis's first defined goal was an investigation of using one-dimensional non-visual (environmental) sensors for detecting the transition between two environments to allow a mobile robot to operate within multiple environments. This particular goal is analyzed and evaluated in the second part of this thesis, and its solving is dependent on solving all other defined goals.

The second goal is research on the suitability of individual one-dimensional non-visual sensors to detect the transition between environments. This particular problem is addressed theoretically in 4.1 and evaluated in practice in Chapter 6.1. TThe first mentioned Chapter contains the review of change point detection algorithms. The latter one is focused on the use of these algorithms for a set of selected non-visual sensors. It is a temperature sensor, humidity sensor, air pressure sensor, and ceiling height (distance) sensor. The sensors are divided into two groups based on the delay in reaction to the change of the variable they measured.

Based on the findings in the first part of this experiment – which is also the solution of the third goal defined in Chapter 3 –, only the temperature sensor and ceiling height sensor are used in the second part. Two simple methods for transition detection, named DifRatio and VarRatio, are presented there. The result of the experiment is that one-dimensional non-visual methods are capable of detecting transition between defined environments. Online change detection methods usually have similar results based on their parameters. Of course, the parameters of detection methods should be adapted to particular cases. For example, in indoor vs. outdoor transition detection, the sensitivity of detection methods for temperature sensors should be higher in summer than in winter weather.

The base design for the proposed system in this thesis is an image-based environment detection system proposed in [169]. It is one of two papers that focuses on developing a multi-environment mobile robot system in the last decades. The fourth goal is focused on the analysis of robot environment classification methods. Thus, the theory containing both classic machine learning and neural network approaches is described in Chapter 4.2. Moreover, datasets suitable for training indoor vs. outdoor classifiers are listed. Unfortunately, it showed that there is no sufficiently large dataset focusing on this problem for training neural networks. Thus, general scene classification datasets that classify images into tens or hundreds of classes have to be used.

The second experiment in Chapter 6.3 is focused on a comparison of both classic and neural network methods for image-based scene classification – indoor vs. outdoor case. To my knowledge, it is the most extensive comparison of this problem that has ever been realized. It is mentioned many articles tested their method on their own dataset. The problem is that they usually compare their own results with results of other methods obtained on different datasets. Thus, some methods have worse results in this comparison than in their original papers. Finally, modern Neural network architectures are evaluated here, including the Big transfer[233] (BiT) training protocol proposed by Google. The best neural network architecture capable of running in realtime on a desktop PC is based on Resnet-50 architecture with BiT training protocol. It achieved 94.97 accuracy on the test set. The best neural network architecture, in general, is another version of BiT Resnet-50 architecture that achieved 94.17%.

The classic approach using Support vector machines as a classifier was worse than neural networks in all cases. The best method was the Two-stage scheme [1] method using HSV color description and Centrist texture description with length 256 bins and 33 classifiers trained on individual parts of the image divided into 16 regions. It achieved 91.87% on the test set of the used dataset. Unfortunately, it is not a sufficiently fast solution for real-time processing. Thus, the best trade-off among classic approaches between speed and accuracy has the Gist method in the Basic scheme and Centrist method in the Multi-scale scheme. Both achieved accuracy over 85% and inference time under 75 ms per one image classification. To use Centrist in Python, it was necessary to implement my own version. The implementation can be found on Github in the repository `centrist`[2]. It is a fast implementation of this method based on the similar implementation of Local Binary Patterns in repository `LbpLibrary`[3] that I made during work on my Bachelor thesis. Both implementations are focused on avoiding unnecessary if-else clauses and using pointers in the C++ programming language proposed in paper [202].

The important part of the thesis is the design and the implementation of the system to detect and classify the robot's environment. Its design is proposed in Chapter 5, together with a description of its implementation. The implementation itself can be found on Github in two versions. The first version in repository `ecs_tests`[4] contains scripts for evaluating system design. The second one in repository `ecs`[5] contains ROS implementation of the system. There are several change detection methods implemented in the proposed system, and it is prepared for use with the `sklearn` package in Python. Moreover, it worth mentioning the respository `env_detection`[6] which is a part of the proposed system capable of creating multilayer maps of the environment. The example of created can be seen in the experiment in Chapter 6.2.

The last goal of this thesis is the evaluation of the proposed system. The evaluation is performed and described in Chapter 6.4. In this experiment, the proposed system is used to classify records from the recorded dataset. The results show that using one-dimensional non-visual sensors can significantly reduce computational cost and especially time requirements of the proposed system compared to the baseline system. In the performed experiment, it was more than 2000 times less time-consuming. It is achieved by performing image-based classification only in the timesteps when change is detected in non-visual sensor data.

Moreover, it can improve the system's accuracy in some cases because it eliminated the wrong classification in situations when no transition occurred. On the other hand, it can be less robust to rely only on non-visual information because there can occur situations when image-

[1]For information about schemes see Chapter 4.2
[2]https://github.com/neduchal/centrist
[3]https://github.com/neduchal/lbpLibrary
[4]https://github.com/neduchal/ecs_tests
[5]https://github.com/neduchal/ecs
[6]https://github.com/neduchal/env_detection

based classifiers make wrong decisions and can be set in the system until another trigger is generated by change point detection. Thus, an alternative approach of using compromise between triggers and image-based classification at a lower rate is proposed. Moreover, switching strategies are discussed at the end of this experiment.

Based on the conclusions described in previous paragraphs, it can be pointed that the hypothesis defined in Chapter 3 is confirmed. On the other side, there are still several tasks that should be addressed in future work.

7.3 Future work

After implementing the system and performing experiments, few other tasks capable of improving the proposed solution were discovered. The first one is using multiple one-dimensional sensors for combined change point detection. It is based on the fact that some environmental properties usually change together. For example, temperature and humidity are both usually changes when the robot moves from inside to outside. Thus, it can be an interesting research direction to analyze these mutual relationships.

Another research direction can be focused either on image-based classification accuracy or its speed on embedded computers. The solution of the first-mentioned direction can be in using semantics segmentation to do preprocessing of the image that is classified. In particular, masking all objects that can be found in multiple classes can be performed. A similar solution for another problem of Simultaneous Localization and Mapping was used in paper [145].

The later mentioned direction can be based on the optimization of the neural networks for run faster on embedded computers. This approach can be based on Tensor RT optimizer from NVIDIA company as in the case of paper [234].

The last future work is the extension and publication of a recorded dataset containing both data needed for performing Simultaneous Localization and Mapping, image stream from the equipped camera, and non-visual sensors data. This dataset can also improve image-based classification of the environment because it allows training classifiers directly from the robot's point of view. In the time of writing this thesis, I met two researchers that show interest in this dataset.

In my future work, I would like to address all four mentioned tasks because they can improve the results presented in this thesis.

Bibliography

[1] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, Jun 2001.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* MIT Press, 2005. [Online]. Available: http://www.probabilistic-robotics.org/

[3] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Aaai/iaai*, 2002, pp. 593–598.

[4] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on.* IEEE, 2011, pp. 155–160.

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on.* IEEE, 2007, pp. 225–234.

[6] M. Szummer and R. W. Picard, "Indoor-outdoor image classification," in *Proceedings 1998 IEEE International Workshop on Content-Based Access of Image and Video Database.* IEEE, 1998, pp. 42–51.

[7] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 2169–2178.

[8] L. F. Menabrea and A. Lovelace, "Sketch of the analytical engine invented by charles babbage," 1842.

[9] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Simultaneous localization and mapping: Present, future, and the robust-perception age," *arXiv preprint arXiv:1606.05830*, 2016.

[10] L. Xia, J. Cui, R. Shen, X. Xu, Y. Gao, and X. Li, "A survey of image semantics-based visual simultaneous localization and mapping: Application-oriented solutions to autonomous navigation of mobile robots," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 1729881420919185, 2020.

[11] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *Computer Vision–ECCV 2014.* Springer, 2014, pp. 834–849.

[12] R. Mur-Artal, J. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *Robotics, IEEE Transactions on*, vol. 31, no. 5, pp. 1147–1163, 2015.

[13] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," 2020.

[14] K. Horstmann, M. Ziegler, and J. Rauthmann, "Putting lewin's equation to the test: Assessing the person-situation interaction with the b5ps," Ph.D. dissertation, Master thesis, Humboldt-Universität zu Berlin, Berlin, 2015.

[15] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (slam): Part 1," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–108, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1638022

[16] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro *et al.*, "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 79–91, 2004.

[17] A. Concha, P. Drews-Jr, M. Campos, and J. Civera, "Real-time localization and dense mapping in underwater environments from a monocular sequence," in *OCEANS 2015-Genova*. IEEE, 2015, pp. 1–5.

[18] C. Luo, L. Cheng, M. C. Chan, Y. Gu, J. Li, and M. Zhong, "Pallas: Self-bootstrapping fine-grained passive indoor localization using wifi monitors," *IEEE Transactions on Mobile Computing*, 2016.

[19] M. Leingartner, J. Maurer, A. Ferrein, and G. Steinbauer, "Evaluation of sensors and mapping approaches for disasters in tunnels," *Journal of Field Robotics*, 2015.

[20] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators and virtual environments*, vol. 6, no. 4, pp. 355–385, 1997.

[21] T. Schöps, J. Engel, and D. Cremers, "Semi-dense visual odometry for ar on a smartphone," in *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 145–150.

[22] P. Lottes, M. Hoeferlin, S. Sander, M. Müter, P. Schulze, and L. C. Stachniss, "An effective classification system for separating sugar beets and weeds for precision farming applications," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5157–5163.

[23] P. Urcola, M.-T. Lorente, J. L. Villarroel, and L. Montano, "Robust navigation and seamless localization for carlike robots in indoor-outdoor environments," *Journal of Field Robotics*, 2016.

[24] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, 2011.

[25] P. Mountney, D. Stoyanov, A. Davison, and G.-Z. Yang, "Simultaneous stereoscope localization and soft-tissue mapping for minimal invasive surgery," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2006, pp. 347–354.

[26] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[27] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *arXiv preprint arXiv:1609.05130*, 2016.

[28] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (slam): Part 2," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[29] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[30] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011, pp. 2564–2571.

[31] Z. Kurt-Yavuz and S. Yavuz, "A comparison of ekf, ukf, fastslam2. 0, and ukf-based fastslam algorithms," in *Intelligent Engineering Systems (INES), 2012 IEEE 16th International Conference on.* IEEE, 2012, pp. 37–43.

[32] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[33] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision.* Cengage Learning, 2014.

[34] R. Mur-Artal and J. D. Tardos, "Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras," *arXiv preprint arXiv:1610.06475*, 2016.

[35] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[36] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[37] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on robotics and automation*, vol. 17, no. 6, pp. 890–897, 2001.

[38] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[39] J. Civera, O. G. Grasa, A. J. Davison, and J. Montiel, "1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, vol. 27, no. 5, pp. 609–631, 2010.

[40] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on.* IEEE, 2001, pp. 145–152.

[41] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk, "Hector open source modules for autonomous mapping and navigation with rescue robots," in *Robot Soccer World Cup.* Springer, 2013, pp. 624–631.

[42] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[43] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng, "Simultaneous mapping and localization with sparse extended information filters: Theory and initial results," in *Algorithmic Foundations of Robotics V.* Springer, 2004, pp. 363–380.

[44] Y. Liu and S. Thrun, "Results for outdoor-slam using sparse extended information filters," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 1227–1233.

[45] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the second international conference on genetic algorithms*, 1987, pp. 14–21.

[46] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on.* IEEE, 2005, pp. 64–69.

[47] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using fastslam," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2. IEEE, 2003, pp. 1985–1991.

[48] ——, "Fastslam 2.0: An improved paparti filtering algorithm for simultaneous localization and mapping that provably converges," *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, pp. 63–90, 2007.

[49] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on.* IEEE, 2005, pp. 2432–2437.

[50] ——, "Improved techniques for grid mapping with rao-blackwellized particle filters," *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 34–46, 2007.

[51] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[52] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 1271–1278.

[53] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on.* IEEE, 2008, pp. 2130–2133.

[54] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[55] W. Garage, "Robot operating system (ros)," 2012.

[56] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[57] C. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 494–506, April 2015.

[58] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[59] M. F. E. Rohmer, S. P. N. Singh, "V-rep: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[60] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, "Robot web tools: Efficient messaging for cloud robotics," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 4530–4537.

[61] B. Steux and O. El Hamzaoui, "tinyslam: A slam algorithm in less than 200 lines c-language program," in *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on.* IEEE, 2010, pp. 1975–1979.

[62] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, 2007.

[63] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[64] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011, pp. 2320–2327.

[65] A. Concha and J. Civera, "Dpptam: Dense piecewise planar tracking and mapping from a monocular sequence," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 5686–5693.

[66] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on.* IEEE, 2011, pp. 127–136.

[67] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph," in *Robotics: science and systems. Vol. 11*, 2015.

[68] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 343–352.

[69] J. Machado Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–6.

[70] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. Kelly, A. J. Davison, M. Luján, M. F. O'Boyle, G. Riley *et al.*, "Introducing slambench, a performance and accuracy benchmarking methodology for slam," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5783–5790.

[71] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison *et al.*, "Slambench2: Multi-objective head-to-head benchmarking for visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[72] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. O'Boyle, A. J. Davison, P. H. Kelly, G. Riley, B. Lennox *et al.*, "Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6351–6358.

[73] P. Neduchal and M. Flídr, "Development of a laboratory framework for testing simultaneous localization and mapping approaches," *IFAC-PapersOnLine*, vol. 49, no. 25, pp. 493–498, 2016.

[74] M. Fallon, H. Johannsson, M. Kaess, and J. J. Leonard, "The mit stata center dataset," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1695–1699, 2013.

[75] C.-C. Wang, D. Duggins, J. Gowdy, J. Kozar, R. MacLachlan, C. Mertz, A. Suppe, and C. Thorpe, "Navlab slammot datasets," www.cs.cmu.edu/~bobwang/datasets.html, May 2004, carnegie Mellon University.

[76] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[77] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. C. 7, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[78] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," in *arXiv:1607.02555*, July 2016.

[79] R. Simpson, J. Cullip, and J. Revell, "The cheddar gorge data set," Technical report, Tech. Rep., 2011.

[80] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and PatternRecognition (CVPR)*, 2012.

[81] F. Moosmann and C. Stiller, "Velodyne slam," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 393–398.

[82] J.-L. Blanco-Claraco, F.-Á. Moreno-Dueñas, and J. González-Jiménez, "The málaga urban dataset: High-rate stereo and lidar in a realistic urban scenario," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 207–214, 2014.

[83] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *arXiv preprint arXiv:1610.08336*, 2016.

[84] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[85] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[86] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[87] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 22–29.

[88] B. Sileshi, J. Oliver, R. Toledo, J. Gonçalves, and P. Costa, "On the behaviour of low cost laser scanners in hw/sw particle filter slam applications," *Robotics and Autonomous Systems*, vol. 80, pp. 11–23, 2016.

[89] R. Goeddel, C. Kershaw, J. Serafin, and E. Olson, "Flat2d: Fast localization from approximate transformation into 2d," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on.* IEEE, 2016, pp. 1932–1939.

[90] M. Kreković, I. Dokmanić, and M. Vetterli, "Echoslam: Simultaneous localization and mapping with acoustic echoes," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2016, pp. 11–15.

[91] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *ICCV '03 Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, 2003.

[92] S. Holmes, G. Klein, and D. W. Murray, "A square root unscented kalman filter for visual monoslam," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on.* IEEE, 2008, pp. 3710–3716.

[93] J. Sola, A. Monin, M. Devy, and T. Lemaire, "Undelayed initialization in bearing only slam," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on.* IEEE, 2005, pp. 2499–2504.

[94] J. Civera, A. J. Davison, and J. M. Montiel, "Unified inverse depth parametrization for monocular slam," in *In Proceedings of Robotics: Science and Systems.* Citeseer, 2006.

[95] ——, "Inverse depth parametrization for monocular slam," *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 932–945, 2008.

[96] D. Gutiérrez-Gómez, W. Mayol-Cuevas, and J. Guerrero, "Inverse depth for accurate photometric and geometric error minimisation in rgb-d dense visual odometry," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on.* IEEE, 2015, pp. 83–89.

[97] R. O. Castle, G. Klein, and D. W. Murray, "Combining monoslam with object recognition for scene augmentation using a wearable camera," *Image and Vision Computing*, vol. 28, no. 11, pp. 1548–1556, 2010.

[98] J. e. a. Shi, “Good features to track,” in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on.* IEEE, 1994, pp. 593–600.

[99] M. A. Atashgah and S. Malaek, “An integrated virtual environment for feasibility studies and implementation of aerial monoslam,” *Virtual Reality*, vol. 16, no. 3, pp. 215–232, 2012.

[100] P. Tanskanen, T. Naegeli, M. Pollefeys, and O. Hilliges, “Semi-direct ekf-based monocular visual-inertial odometry,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 6073–6078.

[101] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and vision computing*, vol. 22, no. 10, pp. 761–767, 2004.

[102] D. Nistér, “An efficient solution to the five-point relative pose problem,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[103] F. Devernay and O. Faugeras, “Straight lines have to be straight,” *Machine vision and applications*, vol. 13, no. 1, pp. 14–24, 2001.

[104] G. Klein and D. Murray, “Parallel tracking and mapping on a camera phone,” in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on.* IEEE, 2009, pp. 83–86.

[105] M. J. Cummins and P. M. Newman, “Fab-map: Appearance-based place recognition and mapping using a learned visual vocabulary model,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 3–10.

[106] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[107] F. R. Bach, M. I. Jordan *et al.*, “Thin junction trees,” in *NIPS*, vol. 14, 2001, pp. 569–576.

[108] C. Chow and C. Liu, “Approximating discrete probability distributions with dependence trees,” *IEEE transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, 1968.

[109] M. Cummins and P. Newman, “Appearance-only slam at large scale with fab-map 2.0,” *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[110] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” *Computer vision–ECCV 2006*, pp. 430–443, 2006.

[111] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *European conference on computer vision.* Springer, 2010, pp. 778–792.

[112] D. Gálvez-López and J. D. Tardós, “Bags of binary words for fast place recognition in image sequences,” *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, October 2012.

[113] R. Elvira, J. D. Tardós, and J. Montiel, “Orbslam-atlas: a robust and accurate multi-map system,” *arXiv preprint arXiv:1908.11585*, 2019.

[114] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, “Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam,” *arXiv preprint arXiv:2007.11898*, 2020.

[115] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, “Svo: Semidirect visual odometry for monocular and multicamera systems,” *IEEE Transactions on Robotics*, 2016.

[116] G. Dubbelman and B. Browning, “Closed-form online pose-chain slam,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013, pp. 5190–5197.

[117] ——, “Cop-slam: closed-form online pose-chain optimization for visual slam,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1194–1213, 2015.

[118] H. Alismail, M. Kaess, B. Browning, and S. Lucey, “Direct visual odometry in low light using binary descriptors,” *IEEE Robotics and Automation Letters*, 2016.

[119] S. Bu, Y. Zhao, G. Wan, and Z. Liu, “Map2dfusion: Real-time incremental uav image mosaicing based on monocular slam,” in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on.* IEEE, 2016, pp. 4564–4571.

[120] M. J. Milford, G. F. Wyeth, and D. Prasser, “Ratslam: a hippocampal model for simultaneous localization and mapping,” in *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 403–408.

[121] M. Milford, A. Jacobson, Z. Chen, and G. Wyeth, “Ratslam: Using models of rodent hippocampus for robot navigation and beyond,” in *Robotics Research.* Springer, 2016, pp. 467–485.

[122] J. Engel, J. Sturm, and D. Cremers, “Semi-dense visual odometry for a monocular camera,” in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 1449–1456.

[123] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g 2 o: A general framework for graph optimization,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 3607–3613.

[124] J. Engel, J. Stuckler, and D. Cremers, “Large-scale direct slam with stereo cameras,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 1935–1942.

[125] D. Caruso, J. Engel, and D. Cremers, “Large-scale direct slam for omnidirectional cameras,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 141–148.

[126] L. Heng, G. H. Lee, and M. Pollefeys, “Self-calibration and visual slam with a multi-camera system on a micro aerial vehicle,” *Autonomous Robots*, vol. 39, no. 3, pp. 259–277, 2015.

[127] P. Ondruska, P. Kohli, and S. Izadi, “Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 21, no. 11, pp. 1251–1258, 2015.

[128] A. Concha and J. Civera, “Using superpixels in monocular slam,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 365–372.

[129] R. Mur-Artal and J. D. Tardós, “Probabilistic semi-dense mapping from highly accurate feature-based monocular slam,” *Proceedings of Robotics: Science and Systems, Rome, Italy*, vol. 1, 2015.

[130] R. A. Newcombe and A. J. Davison, “Live dense reconstruction with a single moving camera,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on.* IEEE, 2010, pp. 1498–1505.

[131] A. Concha, G. Loianno, V. Kumar, and J. Civera, “Visual-inertial direct slam,” in *2016 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2016, pp. 1331–1338.

[132] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual–inertial odometry using nonlinear optimization,” *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[133] J. G. Morrison, D. Gavez-Lopez, and G. Sibley, “Scalable multirobot localization and mapping with relative maps: Introducing moarslam,” *IEEE Control Systems*, vol. 36, no. 2, pp. 75–85, 2016.

[134] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, “Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology.* ACM, 2011, pp. 559–568.

[135] S. O. R. Fedkiw and S. Osher, “Level set methods and dynamic implicit surfaces,” *Surfaces*, vol. 44, p. 77, 2002.

[136] R. F. Salas-Moreno, B. Glocken, P. H. Kelly, and A. J. Davison, “Dense planar slam,” in *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on.* IEEE, 2014, pp. 157–164.

[137] L. Ma, C. Kerl, J. Stückler, and D. Cremers, “Cpa-slam: Consistent plane-model alignment for direct rgb-d slam,” in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 1285–1291.

[138] T. K. Moon, “The expectation-maximization algorithm,” *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[139] E. Ataer-Cansizoglu, Y. Taguchi, and S. Ramalingam, “Pinpoint slam: A hybrid of 2d and 3d simultaneous localization and mapping for rgb-d sensors,” in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 1300–1307.

[140] R. Salas-Moreno, R. Newcombe, H. Strasdat, P. Kelly, and A. Davison, “Slam++: Simultaneous localisation and mapping at the level of objects,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1352–1359.

[141] D. Gálvez-López, M. Salas, J. D. Tardós, and J. Montiel, "Real-time monocular object slam," *Robotics and Autonomous Systems*, vol. 75, pp. 435–449, 2016.

[142] T. Dharmasiri, V. Lui, and T. Drummond, "Mo-slam: Multi object slam with run-time object discovery through duplicates," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on.* IEEE, 2016, pp. 1214–1221.

[143] B. Mu, S.-Y. Liu, L. Paull, J. Leonard, and J. P. How, "Slam with objects using a nonparametric pose graph," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on.* IEEE, 2016, pp. 4602–4609.

[144] Z. Zhang, Z. Cui, C. Xu, Z. Jie, X. Li, and J. Yang, "Joint task-recursive learning for semantic segmentation and depth estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 235–251.

[145] M. Kaneko, K. Iwami, T. Ogawa, T. Yamasaki, and K. Aizawa, "Mask-slam: Robust feature-based monocular slam by masking using semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 258–266.

[146] S. Wang, X. Lv, J. Li, and D. Ye, "Coarse semantic-based motion removal for robust mapping in dynamic environments," *IEEE Access*, vol. 8, pp. 74 048–74 064, 2020.

[147] S. Zhi, M. Bloesch, S. Leutenegger, and A. J. Davison, "Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 776–11 785.

[148] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," in *2017 IEEE International Conference on Robotics and automation (ICRA).* IEEE, 2017, pp. 4628–4635.

[149] B. Yang, Z. Lai, X. Lu, S. Lin, H. Wen, A. Markham, and N. Trigoni, "Learning 3d scene semantics and structure from a single depth image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 309–312.

[150] L. Riazuelo, L. Montano, and J. Montiel, "Semantic visual slam in populated environments," in *2017 European conference on mobile robots (ECMR).* IEEE, 2017, pp. 1–7.

[151] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger, "Fusion++: Volumetric object-level slam," in *2018 international conference on 3D vision (3DV).* IEEE, 2018, pp. 32–41.

[152] L. Nicholson, M. Milford, and N. Sünderhauf, "Quadricslam: Constrained dual quadrics from object detections as landmarks in semantic slam," *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[153] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2560–2568.

[154] A. J. Davison, “Futuremapping: The computational structure of spatial ai systems,” *arXiv preprint arXiv:1803.11288*, 2018.

[155] A. J. Davison and J. Ortiz, “Futuremapping 2: Gaussian belief propagation for spatial ai,” *arXiv preprint arXiv:1910.14139*, 2019.

[156] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, “Visual simultaneous localization and mapping: a survey,” *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[157] S. Saeedi, M. Trentini, M. Seto, and H. Li, “Multiple-robot simultaneous localization and mapping: A review,” *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[158] A. Petland and B. Horowitz, “Recovery of nonrigid motion and structure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 7, pp. 730–742, 1991.

[159] L. Torresani, A. Hertzmann, and C. Bregler, “Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 5, pp. 878–892, 2008.

[160] S. Pillai and J. Leonard, “Monocular slam supported object recognition,” *arXiv preprint arXiv:1506.01732*, 2015.

[161] S. Yang, Y. Song, M. Kaess, and S. Scherer, “Pop-up slam: Semantic monocular plane slam for low-texture environments,” in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on.* IEEE, 2016, pp. 1222–1229.

[162] M. Liu, S. Huang, G. Dissanayake, and H. Wang, “A convex optimization based approach for pose slam problems,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* IEEE, 2012, pp. 1898–1903.

[163] L. Carlone, D. M. Rosen, G. Calafiore, J. J. Leonard, and F. Dellaert, “Lagrangian duality in 3d slam: Verification techniques and optimal solutions,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 125–132.

[164] L. Vandenberghe and S. Boyd, “Semidefinite programming,” *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[165] C. Leung, S. Huang, and G. Dissanayake, “Active slam using model predictive control and attractor based exploration,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 5026–5031.

[166] C. Leung, S. Huang, N. Kwok, and G. Dissanayake, “Planning under uncertainty using model predictive control for information gathering,” *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 898–910, 2006.

[167] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[168] H. Rebecq, T. Horstschafer, G. Gallego, and D. Scaramuzza, “Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real-time,” *IEEE Robotics and Automation Letters*, 2016.

[169] D. C. Asmar, J. S. Zelek, and S. M. Abdallah, "Smartslam: localization and mapping across multi-environments," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, vol. 6. IEEE, 2004, pp. 5240–5245.

[170] X. Dai and J. Du, "Hierarchical simultaneous localization and mapping based on discrete event systems," in *2009 International Conference on Measuring Technology and Mechatronics Automation*, vol. 2. IEEE, 2009, pp. 234–237.

[171] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[172] M. Basseville, I. V. Nikiforov *et al.*, *Detection of abrupt changes: theory and application.* prentice Hall Englewood Cliffs, 1993, vol. 104.

[173] S. Liu, M. Yamada, N. Collier, and M. Sugiyama, "Change-point detection in time-series data by relative density-ratio estimation," *Neural Networks*, vol. 43, pp. 72–83, 2013.

[174] Y. Kawahara and M. Sugiyama, "Sequential change-point detection based on direct density-ratio estimation," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 2, pp. 114–127, 2012.

[175] L. I. Kuncheva and W. J. Faithfull, "Pca feature extraction for change detection in multidimensional unlabeled data," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 1, pp. 69–80, 2013.

[176] T. Kanamori, S. Hido, and M. Sugiyama, "A least-squares approach to direct importance estimation," *The Journal of Machine Learning Research*, vol. 10, pp. 1391–1445, 2009.

[177] M. Yamada, T. Suzuki, T. Kanamori, H. Hachiya, and M. Sugiyama, "Relative density-ratio estimation for robust distribution comparison," *Neural computation*, vol. 25, no. 5, pp. 1324–1370, 2013.

[178] Y. Kawahara, T. Yairi, and K. Machida, "Change-point detection in time-series data based on subspace identification," in *Seventh IEEE International Conference on Data Mining (ICDM 2007).* IEEE, 2007, pp. 559–564.

[179] N. Itoh and J. Kurths, "Change-point detection of climate time series by nonparametric method," in *Proceedings of the world congress on engineering and computer science*, vol. 1. Citeseer, 2010, pp. 445–448.

[180] R. P. Adams and D. J. MacKay, "Bayesian online changepoint detection," *arXiv preprint arXiv:0710.3742*, 2007.

[181] Y. Saatçi, R. D. Turner, and C. E. Rasmussen, "Gaussian process change point models," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* Citeseer, 2010, pp. 927–934.

[182] Z. Harchaoui, F. Vallet, A. Lung-Yut-Fong, and O. Cappé, "A regularized kernel-based approach to unsupervised audio segmentation," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing.* IEEE, 2009, pp. 1665–1668.

[183] H. Chen, N. Zhang *et al.*, "Graph-based change-point detection," *The Annals of Statistics*, vol. 43, no. 1, pp. 139–176, 2015.

[184] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proceedings 2001 IEEE international conference on data mining.* IEEE, 2001, pp. 289–296.

[185] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans, "Time series epenthesis: Clustering time series streams requires ignoring some data," in *2011 IEEE 11th International Conference on Data Mining.* IEEE, 2011, pp. 547–556.

[186] J. Zakaria, A. Mueen, and E. Keogh, "Clustering time series using unsupervised-shapelets," in *2012 IEEE 12th International Conference on Data Mining.* IEEE, 2012, pp. 785–794.

[187] D.-H. Tran, "Automated change detection and reactive clustering in multivariate streaming data," in *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF).* IEEE, 2019, pp. 1–6.

[188] K. P. Murphy, *Machine learning: a probabilistic perspective.* MIT press, 2012.

[189] J. Mao and A. K. Jain, "Texture classification and segmentation using multiresolution simultaneous autoregressive models," *Pattern recognition*, vol. 25, no. 2, pp. 173–188, 1992.

[190] N. Serrano, A. Savakis, and A. Luo, "A computationally efficient approach to indoor/outdoor scene classification," in *Object recognition supported by user interaction for service robots*, vol. 4. IEEE, 2002, pp. 146–149.

[191] I. Daubechies, *Ten lectures on wavelets.* Siam, 1992, vol. 61.

[192] N. Serrano, A. E. Savakis, and J. Luo, "Improved scene classification using efficient low-level features and semantic cues," *Pattern Recognition*, vol. 37, no. 9, pp. 1773–1784, 2004.

[193] A. Payne and S. Singh, "Indoor vs. outdoor scene classification in digital photographs," *Pattern Recognition*, vol. 38, no. 10, pp. 1533–1545, 2005.

[194] ——, "A benchmark for indoor/outdoor scene classification," in *International Conference on Pattern Recognition and Image Analysis.* Springer, 2005, pp. 711–718.

[195] Z. Li and L. Itti, "Saliency and gist features for target detection in satellite images," *IEEE Transactions on Image Processing*, vol. 20, no. 7, pp. 2017–2029, 2010.

[196] A. Oliva and A. Torralba, "Building the gist of a scene: The role of global image features in recognition," *Progress in brain research*, vol. 155, pp. 23–36, 2006.

[197] W. Kim, J. Park, and C. Kim, "A novel method for efficient indoor–outdoor image classification," *Journal of Signal Processing Systems*, vol. 61, no. 3, pp. 251–258, 2010.

[198] S. Battiato, G. M. Farinella, G. Gallo, and D. Ravì, "Exploiting textons distributions on spatial hierarchy for scene classification," *EURASIP Journal on Image and Video Processing*, vol. 2010, no. 1, p. 919367, 2010.

[199] L. Zhou, Z. Zhou, and D. Hu, "Scene classification using a multi-resolution bag-of-features model," *Pattern Recognition*, vol. 46, no. 1, pp. 424–433, 2013.

[200] J. Wu and J. M. Rehg, "Centrist: A visual descriptor for scene categorization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 8, pp. 1489–1501, 2010.

[201] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[202] T. Mäenpää, M. Turtinen, and M. Pietikäinen, "Real-time surface inspection by texture," *Real-Time Imaging*, vol. 9, no. 5, pp. 289–296, 2003.

[203] L. Zhou, Z. Zhou, and D. Hu, "Scene classification using multi-resolution low-level feature combination," *Neurocomputing*, vol. 122, pp. 284–297, 2013.

[204] C. Chen, Y. Ren, and C.-C. J. Kuo, "Large-scale indoor/outdoor image classification via expert decision fusion (edf)," in *Asian Conference on Computer Vision.* Springer, 2014, pp. 426–442.

[205] S. S. Cvetkovic, S. V. Nikolić, and S. Ilic, "Effective combining of color and texture descriptors for indoor-outdoor image classification," *Facta Universitatis, Series: Electronics and Energetics*, vol. 27, no. 3, pp. 399–410, 2014.

[206] R. Raja, S. M. M. Roomi, and D. Dharmalakshmi, "Robust indoor/outdoor scene classification," in *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR).* IEEE, 2015, pp. 1–5.

[207] M. Shahriari and R. Bergevin, "A two-stage outdoor-indoor scene classification framework: experimental study for the outdoor stage," in *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA).* IEEE, 2016, pp. 1–8.

[208] Z. Tong, D. Shi, B. Yan, and J. Wei, "A review of indoor-outdoor scene classification," in *2017 2nd International Conference on Control, Automation and Artificial Intelligence (CAAI 2017).* Atlantis Press, 2017.

[209] A. Ganesan and A. Balasubramanian, "Indoor versus outdoor scene recognition for navigation of a micro aerial vehicle using spatial color gist wavelet descriptors," *Visual Computing for Industry, Biomedicine, and Art*, vol. 2, no. 1, p. 20, 2019.

[210] L. Tao, Y.-H. Kim, and Y.-T. Kim, "An efficient neural network based indoor-outdoor scene classification algorithm," in *2010 Digest of Technical Papers International Conference on Consumer Electronics (ICCE).* IEEE, 2010, pp. 317–318.

[211] W. Tahir, A. Majeed, and T. Rehman, "Indoor/outdoor image classification using gist image features and neural network classifiers," in *2015 12th International Conference on High-capacity Optical Networks and Enabling/Emerging Technologies (HONET).* IEEE, 2015, pp. 1–5.

[212] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, 2014, pp. 487–495.

[213] L. Wang, S. Guo, W. Huang, Y. Xiong, and Y. Qiao, "Knowledge guided disambiguation for large-scale scene classification with multi-resolution cnns," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 2055–2068, 2017.

[214] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[215] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2009, pp. 413–420.

[216] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International journal of computer vision*, vol. 42, no. 3, pp. 145–175, 2001.

[217] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 524–531.

[218] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE, 2010, pp. 3485–3492.

[219] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva, "Sun database: Exploring a large collection of scene categories," *International Journal of Computer Vision*, vol. 119, no. 1, pp. 3–22, 2016.

[220] J. Collier and A. Ramirez-Serrano, "Environment classification for indoor/outdoor robotic mapping," in *2009 Canadian Conference on Computer and Robot Vision.* IEEE, 2009, pp. 276–283.

[221] P. Neduchal, L. Bureš, and M. Železný, "Environment detection system for localization and mapping purposes," *IFAC-PapersOnLine*, vol. 52, no. 27, pp. 323–328, 2019.

[222] P. Neduchal, I. Gruber, and M. Železnỳ, "Indoor vs. outdoor scene classification for mobile robots," in *International Conference on Interactive Collaborative Robotics.* Springer, 2020, pp. 243–252.

[223] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, A. Koubaa, Ed. Springer, 2016, ch. 5.

[224] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[225] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[226] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[227] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[228] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[229] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” *CoRR*, vol. abs/1611.05431, 2016.

[230] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[231] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[232] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[233] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, “Big transfer (bit): General visual representation learning,” 2019.

[234] H. Liu, R. A. R. Soto, F. Xiao, and Y. J. Lee, “Yolactedge: Real-time instance segmentation on the edge (jetson agx xavier: 30 fps, rtx 2080 ti: 170 fps),” *arXiv preprint arXiv:2012.12259*, 2020.

Authored and Co-authored Works

- Neduchal, P., Gruber I., and Železný, M. (2020). Indoor vs. Outdoor Scene Classification for Mobile Robots. In Proceedings of International Conference on Interactive Collaborative Robotics 2020. – *Document Type: Conference Paper.*

- Neduchal, P., and Bureš, L., and Müller, L. (2020). Automatic Information Extraction from Scanned Documents. In Proceedings of International Conference on Speech and Computer 2020. – *Document Type: Conference Paper.*

- Gruber I., Ircing, P., Neduchal, P., Hrúz, M., Hlaváč, M., Zajíc, Z., Švec, J., and Bulín, M. (2020). An Automated Pipeline for Robust Image Processing and Optical Character Recognition of Historical Documents. In Proceedings of International Conference on Speech and Computer 2020. – *Document Type: Conference Paper.*

- Neduchal, P., and Železný, M. (2020). Environment Classification Approach For Mobile Robots. In Proceedings of 15th International Conference on Electromechanics and Robotics "Zavalishin's Readings". Springer, Singapore. – *Document Type: Conference Paper.*

- Neduchal, P., and Železný, M. (2020). Frontier Detection in Consecutive Grid Maps with Set Reduction. In Proceedings of 14th International Conference on Electromechanics and Robotics "Zavalishin's Readings" (pp. 441-453). Springer, Singapore. – *Document Type: Conference Paper*

- Neduchal, P., Bureš, L., and Železný, M. (2019). Environment detection system for localization and mapping purposes. IFAC-PapersOnLine, 52(27), 323-328. – *Document Type: Conference Paper*

- Bureš, L., Gruber, I., Neduchal, P., Hlaváč, M., and Hrúz, M. (2019). Semantic text segmentation from synthetic images of full-text documents. SPIIRAS Proceedings, 18(6), 1381-1406. – *Document Type: Journal Article (Jsc)*

- Neduchal, P. (2019). Koncept systému pro detekci změny prostředí a jeho klasifikaci, SVK FAV 2019 – *Document Type: Conference Paper*

- Neduchal, P., Flídr, M., and Železný, M. (2018, September). Fast Frontier detection approach in consecutive grid maps. In International Conference on Interactive Collaborative Robotics (pp. 192-201). Springer, Cham. – *Document Type: Conference Paper*

- Bureš, L., Neduchal, P., Hlaváč, M., and Hrúz, M. (2018, September). Generation of synthetic images of full-text documents. In International Conference on Speech and Computer (pp. 68-75). Springer, Cham. – *Document Type: Conference Paper*

- Zajíc, Z., Skorkovská, L., Neduchal, P., Ircing, P., Psutka, J., Hrúz, M., ... and Müller, L. (2018, May). Towards Processing of the Oral History Interviews and Related Printed Documents. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018). – *Document Type: Conference Paper*

- Neduchal, P. (2018). Detekce cílů z grid mapy v úloze explorace prostředí, SVK FAV 2018 – *Document Type: Conference Paper*

- Neduchal, P., Berka, F., and Železný, M. (2017, September). Stationary device for drone detection in urban areas. In International Conference on Interactive Collaborative Robotics (pp. 162-169). Springer, Cham. – *Document Type: Conference Paper*

- Skorkovská, L., Neduchal, P., Zajíc, Z., Ircing, P., Müller, L., and Bureš, L. (2017). First insight into the processing of the historical documents from the period of totalitarian regimes. – *Document Type: Conference Paper*

- Neduchal, P., and Flídr, M. (2017). Vývoj multifunkčního kolového robota, SVK FAV 2017 – *Document Type: Conference Paper*

- Jirik, M., and Neduchal, P. (2016). Experiments with automatic segmentation of liver parenchyma using texture description. Pattern Recognition and Image Analysis, 26(3), 572-575. – *Document Type: Journal Article (Jsc)*

- Neduchal, P., and Bureš, L. (2016). Využití dvourozměrné Fourierovy transformace v úloze zarovnání naskenovaného dokumentu, SVK FAV 2016 – *Document Type: Conference Paper*

- Neduchal, P., and Flídr, M. (2016). Development of a laboratory framework for testing simultaneous localization and mapping approaches. IFAC-PapersOnLine, 49(25), 493-498. – *Document Type: Journal Article (Jsc)*

- Neduchal, P., and Bureš, L. (2015). Systém automatické kontroly odevzdávaných semestrálních prací (SAKo), SVK FAV 2015 – *Document Type: Conference Paper*

- Bureš, L., and Neduchal, P. (2015). Identifikace příznaků diabetické retinopatie z obrázků očí, SVK FAV 2015 – *Document Type: Conference Paper*

- Neduchal, P. (2014). Texturní analýza 3D dat pomocí metody LBP, SVK FAV 2014 – *Document Type: Conference Paper*

- Neduchal, P. (2013). Texturní analýza pomocí knihovny LbpLibrary, SVK FAV 2013 – *Document Type: Conference Paper*

- Neduchal, P. (2013). Návrh a testování metod vizuální simultánní lokalizace a mapování. Diplomová práce

- Neduchal, P. (2012). Texturní analýza pomocí metody LBP počítaná v reálném čase, SVK FAV 2012 – *Document Type: Conference Paper*

- Neduchal, P. (2011). Texturní analýza pomocí metody LBP, SVKB FAV 2011 – *Document Type: Conference Paper*