ÉCOLE CENTRALE DE NANTES UNIVERSITY OF GENOA

MASTER ERASMUS MUNDUS
EMARO+ "European Master in Advanced Robotics"

2016 / 2017

Master Thesis Report

Presented by

Ernest Skrzypczyk, BSc

On 30.07.17

UGV and UAV collaboration in an autonomous infrastructure scenario

Jury

President:	Phillipe Martinet	Professor, LS2N, ECN
Evaluators:	Olivier Kermorgant	Maître de conférences, LS2N, ECN
	Phillipe Martinet	Professor, LS2N, ECN
	Marco Baglietto	Associate Professor, DIBRIS, UNIGE
	David Marquez-Gamez	Researcher, IRT Jules Verne
Supervisors:	Olivier Kermorgant	Maître de conférences, LS2N, ECN
	David Marquez-Gamez	Researcher, IRT Jules Verne

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

I would like to thank my supervisor Professor Kermorgant for guidance and support, as well as my friend for being there for me.

Contents

List of Figures

List of Tables

Abstract

The conventional SLAM problem is well addressed in robotics, however current open problems in simultaneous localization and mapping require research attention. In particular there has been work done on the aspect of multi robot SLAM using homogeneous units, the heterogeneous approach, especially between two diametrically different types of robots like an UGV and UAV, is yet to be well established. Furthermore there has been work done on decentralized and distributed approaches, even their combination, but not within heterogeneous units. The master thesis aims at developing a collaborative, decentralized and heterogeneous SLAM algorithm that will advance current state of the art in the aforementioned fields of interest. Additionally the autonomous infrastructure is introduced and formalized.

Abbreviations

AI Artificial Intelligence

API Application Programming Interface

BA Bundle Adjustment

BF Bayesian Filter

BFS Breadth-first Search

BoW Bag of Words

BRIEF Binary Robust Independent Elementary Features

CFG Constraint Factor Graph

CML Concurrent Mapping and Localization

DDF Decentralized Data Fusion

DSO Direct Sparse Odometry

EKF Extended Kalman Filter

EM Expectation Maximization

FAST Features from Accelerated Segment Test

FG Factor Graph

FOV Field Of View

GPS Global Positioning System

GUI Graphical User Interface

GVG Generalized Voronoi Graph

HBVS Hybrid 2D/3D Based Visual Servoing

IBVS Image Based Visual Servoing

ICP Iterative Closest Point

IMU Inertial Measurement Unit

IR Infra-Red

KF Kalman Filter

LC Loop Closure

LIDAR Light Detection And Ranging

LSD Large Scale Direct

LSD-SLAM Large-Scale Direct SLAM

MAP Maximum-A-Posteriori [estimation]

MLE Maximum Likelihood Estimation

MR-SLAM Multi-Robot SLAM

ORB Oriented FAST and Rotated BRIEF

PBVS Position Based Visual Servoing

PDF Probability Distribution Function

PF Particle Filters

PG Pose Graph

PnP Perspective-n-Problem

RGBD Red, Green, Blue, Depth

RGB Red, Green, Blue

ROI Regions Of Interest

RRR Realizing, Reversing and Recovering

SAM Smoothing and Mapping

SfM Structure from Motion

SLAM Simultaneous Localization And Mapping

SODAR SOnic Detection And Ranging

Sonar SOund Navigation And Ranging (original acronym)

SVD Singular Value Decomposition

UAV Unmanned Air Vehicle

UGV Unmanned Ground Vehicle

UI User Interface

UKF Unscented Kalman Filter

UUID Universally Unique IDentifier

VO Visual Odometry

VS Visual Servoing

Symbols

x_k – Pose in the state vector at instance k;

l_a – Set containing the locations of all landmarks, $l = l_0, l_1, dots, l_n$ with n being the number of landmarks;

m_a – Set containing the locations of all landmarks, $m_a = m_0, m_1, dots, m_n$ with n being the number of landmarks;

$y_{0:i}$ – All measurements or observations of landmark locations l up to time instance i;

$Z_{0:k}$ – All measurements or observations of landmark locations m_a up to time instance k;

$u_{0:k}$ – All the control inputs up to time instance k, ergo the history of control input u;

m_i – Vector holding the static location of landmark i;

$z_{i,k}$ – Measurement of location of landmark i at time instance k, can hold multiple observations

$X_{0:k}$ – State vector history,

$U_{0:k}$ – Control inputs history,

$\mathbf{L_s}$ – Interaction matrix

s – Visual features vector

e – Error for visual features

R_i – Robot unit i

a_i – Sequential rendezvous attempts

l_i^R – Selected location of robot R_i

$t(a_i)$ – Time instance a_i

Notations

(Ai) – Algorithm i

(Ei) – Equation i

(Fi) – Figure i

"*name*" – Section *name*

(Ri) – Script i

(Si) – Section i

(Ti) – Table i

Chapter 1

Introduction

Autonomous infrastructure, even though not yet formally defined, has seen a high interest and development in recent years in certain fields related to security, data transmission [**?**] and transport. In the scope of global production, personal and products transport as well as services, it is highly dependent on robot collaboration. Autonomy of the individual units is desired, since the possible working scenarios are usually non trivial. For very complex problems the collaboration of robots, that have complementary capabilities, could achieve group autonomy. The master thesis deals with an Unmanned Ground Vehicle (UGV) and Unmanned Air Vehicle (UAV) collaboration for a decentralized multi SLAM (Simultaneous Localization and Mapping) process within the autonomous infrastructure aspect.

The proposed framework is a decentralized approach with units of different type. The robots conduct SLAM in an unknown environment, starting from an unknown location and store the map locally in memory. The relevant data is shared between units once in communication range and both local maps are optimized based upon virtual loop closures using Bundle Adjustment (BA) or similar techniques. In order to provide additional information and utilize rendezvous for data exchange, visual servoing and visual communication can be used. While the UAV follows the UGV marked with a QR code or a similar marker, it can provide additional information from another perspective about the environment. It is of particular interest to involve these two types of robots, because of their different capabilities and very different points of view into the environment.

By using LEDs or other light source devices around the marker, an unidirectional communication between the mobile platform and drone could be established. This can help with establishing rendezvous points between the robots, once a unit is ready to optimize it's local map, it's memory is full or it requires assistance in establishing next move. In the last case the UAV could be instructed to perform an area sweep in order to supply additional map information. The LEDs also provide a possibility of pose estimation using DeMenthon's POSIT algorithm [83].

One possible problem in this approach is the communication of limited range between units. Therefore data transmission optimizing measures should be performed. The basic data will consist of key frames coming from the output of an monocular SLAM approach like Large-Scale Direct monocular SLAM (LSD-SLAM) [57] or Direct Sparse Odometry (DSO) [59].

Context of the work

An autonomous infrastructure is one with the capability of independent self-adjustment and self-maintenance. Today this term is mostly associated in robotics and in a very limited way, in general with aspects of security, data networking and transport [**?**]. The master thesis formalizes the term therefore in more detail *"Definitions"*.

Motivation

The main reasons behind the master thesis are current developments in the technological sector as well as on the level of society. According to the United Nations Department of Economic and Social Affairs report, human population will reach approximately 9 billion in the year 2050 and 11 billion by the end of the century with a probability of 80% [98][62]. It can be observed that the current premise of the socio-economic structure is that of an ever growing economy with little regard to related physical processes. Yet the demand put onto the system grows exponentially, accelerated by the rapid population growth in developing countries, the market demand and diminishing finite resources availability.

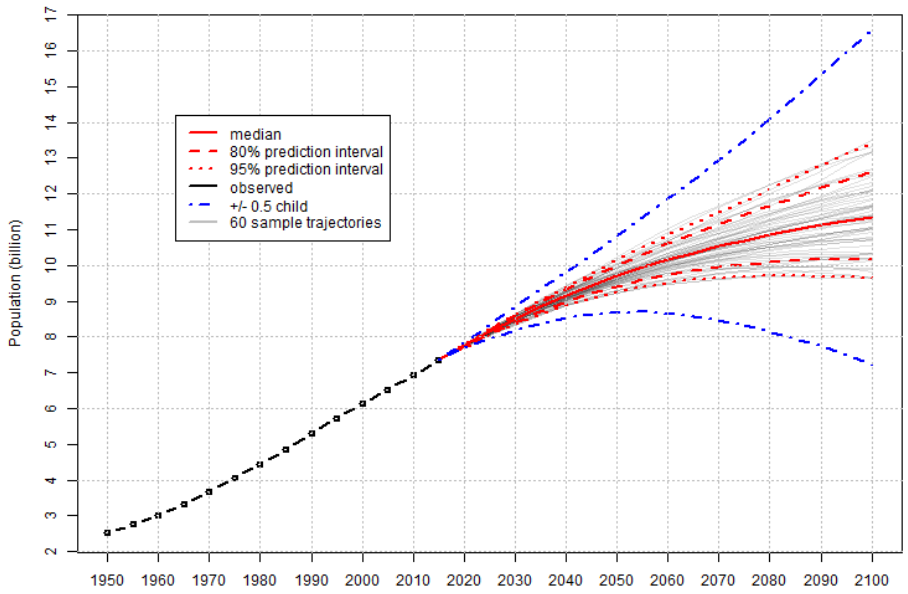


Figure 1.1: Total world population projection using Bayesian filtering [98].

Recent technological developments in the infrastructure led to highly complex systems, that are difficult to control. The power outage in Canada, that occurred in 2003, caused by a chain of events, mostly related to human error [60], clearly indicated that current state of control in the infrastructure is not satisfactory and mostly still a human responsibility.

On the other hand, the rapid development can be observed in various fields of technology and research. Most promising technologies, that would allow for increase of the system's overall efficiency, like nanotechnology and technological singularity, can also have detrimental effects by simply increasing the complexity of the system for example. As per projections of current developments [71][68], the planetary power output generated could elevate the Kardashev scale rating, shown in (E1.1), of humanity to a civilization type I [69], ergo approximately $10^{16} \div 10^{17} W$ power output.

$$K = \frac{log_{10} P - 6}{10} \tag{1.1}$$

K – Kardashev rating, P – power output

From these observations it can be stated that the current socio-economic system, in particular the infrastructure, is not prepared for future demand mentioned earlier. Therefore the master thesis introduces the idea of autonomous infrastructure and deals with a specific type of robot collaboration that is currently an open issue in the research aspect of the SLAM problem [44].

Robot collaboration offers a variety applications, from simple delivery tasks to complex rescue scenarios in dangerous or damaged environments [77] shown in (F1.2). Other applications for robot collaboration in search and rescue like forest fires, in other fields like cleaning operations, space and underwater exploration, military applications, security and surveillance, as well as maintenance investigations, are also of interest [91]. Current state of robot collaboration has a lot of potential for improvement, especially in the context of an autonomous infrastructure level 4, as defined in *"Definitions"*. Therefore is it the motivation of the master thesis to contribute to a problem scenario, that can benefit further from the collaboration aspect. It is also of interest to explore methods that can improve collaboration for solving the SLAM problem like visual servoing, visual communication and others.

(a) Damaged building serving as the experimental environment

(b) 2D occupancy grid maps

– UGV1, – UGV2, – UAV

Figure 1.2: Collaborative mapping of a damaged building performed by two UGV and one UAV [77].

Problem formulation

Based on previous work [49][46] development, implementation and evaluation of a decentralized and heterogeneous approach for collaborative SLAM with the possibility of further expansion, is the desired goal of the master thesis.

Contribution and innovation

The master thesis project aims at developing and implementing a working collaborative SLAM algo-

rithm between different type of robots, in particular between UAV and UGV. Furthermore the proposed solution uses a decentralized approach, where data communication is under a constraint related to the limited range of transmission. This addresses some of the current open problems in the SLAM research area [44]. Additionally aspects like visual servoing for the purpose of optimization of the SLAM process and visual communication are to be analysed and evaluated. The whole work occurs in context of autonomous infrastructure.

Chapter 2

Background theory

Definitions

Robot collaboration Process of coordinated actions performed by robots towards a specified goal.

Autonomy The ability of making independent decisions.

Unmanned vehicle Vehicle designed without support systems for humans, capable of similar performance as a manned vehicle.

Autonomous infrastructure An infrastructure with the capability of independent self-adjustment and self-maintenance.

To further formalize the term *autonomous infrastructure*, it can be classified by 5 levels:

0 The autonomy is local in very limited aspects, there is no autonomous chain. The majority of processes are human controlled and monitored.

1 Branches of the economy hold at least one full chain of product or services. Specific needs of the community are satisfied by the infrastructure. Autonomy is local and regional in limited aspects. The majority of processes are human controlled and monitored.

2 Major branches of the economy, like nutrition production and personal transport, hold full autonomous chains of product or services. Majority of the needs of the community are satisfied by the infrastructure with little human interaction, mostly limited to monitoring. Autonomy is national and pseudo-continental in many aspects, self-maintenance achieved.

3 Full economy autonomy, virtually no human interaction required. Complex dynamic demand is satisfied after infrastructure adjustment.

4 Planetary autonomy. Infrastructure extrapolates current and adjusts itself for future developments.

Simultaneous localization and mapping

The process of mapping of the environment and using it for localization, while at the same time localizing with the help of that map has been first defined as Simultaneous Localization And Mapping (SLAM) problem in the 1986 [42]. It still is one of the fundamental challenges in robotics and spreads into many other fields of research. It is being further developed in the robotics field with respect to current open problems [44]. SLAM is also known as the Concurrent Mapping and Localization (CML) [41]. The

complexity in the SLAM problem arises from the interdependence of both mapping and localization [91]. Artificial and natural features of the environment can be utilized for this purpose using a variety of sensors, whose measurements can be then fused and used for the localization and map building process.

The SLAM problem can be then expressed as probability distribution function (PDF) (E2.1)

$$P(x_k, m_a | Z_{0:k}, U_{0:k}) \tag{2.1}$$

where the following variables are defined at time instance k:

- x_k is the pose in the state vector at instance k;
- m_a is a set containing the locations of all landmarks, $m_a = m_0, m_1, \ldots, m_n$ with n being the number of landmarks;
- $Z_{0:k}$ represents all measurements or observations of landmark locations m_a up to time instance k;
- $u_{0:k}$ holds all the control inputs up to time instance k, ergo the history of control input u;
- m_i is a vector holding the static location of landmark i;
- $z_{i,k}$ represents measurement of location of landmark i at time instance k, can hold multiple observations
- $X_{0:k}$ set holding state vector history, $X_{0:k} = x_0, x_1, \ldots, x_k = X_{0:k-1}, x_k$
- $U_{0:k}$ set of control inputs history, $U_{0:k} = u_0, u_1, \ldots, u_k = U_{0:k-1}, u_k$

Equation (E2.1) describes joint posterior density for the SLAM problem scenario illustrated in (F2.1). Starting from an unknown location in an unknown environment the unit moves and conducts measurements. Detected landmarks are used to increase certainty of localization, which is described by a probability distribution. The certainty level is also known as belief or belief state [96], which is defined as an assumption of an information without empirical evidence. During exploration more landmarks are recognized and stored, a posteriori belief and PDF can be updated. For the iterative approach, which is most often desired in probabilistic SLAM, the expression (E2.2) can be calculated using Bayesian theorem when observation (E2.3) and transition (E2.4) models are present. It is assumed that locations of landmarks and current state x_k are known for the observation model and results in probability of making an observation z_k. The transition model relies only on current control input u_k and previous state x_{k-1} and results in estimation of current state x_k.

$$P(x_{k-1}, m_a | Z_{0:k-1}, U_{0:k-1}) \tag{2.2}$$

$$P(z_k | x_k, m_a) \tag{2.3}$$

$$P(x_k | x_{k-1}, u_k) \tag{2.4}$$

For most cases a Gaussian or normal distribution can be used and solved using mean or mathematical expectation and covariance. For non-linear probability distributions however, a Markov model is advised [46]. It makes the computed state x_k independent of measurements z and map m_a and only based upon previous state x_{k-1} and current control input u_k. This is the case for the transition model (E2.4).

To finalize the probabilistic SLAM model iterative computations of the prediction or time update and innovation or measurement update steps are defined in (E2.5) and (E2.7).

$$P(x_k, m_a | Z_{0:k-1}, U_{0:k}, x_0) = \int \underbrace{P(x_k | x_{k-1}, u_k)}_{\text{transition model}} P(x_{k-1}, m_a | Z_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1} \tag{2.5}$$

From Bayesian rule (E2.6) the measurement update relation is developed.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.6}$$

$$P(x_k, m_a|Z_{0:k}, U_{0:k}, x_0) = \underbrace{P(z_k|x_k, m_a)}_{\text{observation model}} \frac{P(x_k, m_a|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})} \tag{2.7}$$

The recursion is a function of both, the observation model (E2.3) and transition model (E2.4). Additionally it should be noted that, correlations between landmark estimates increase monotonically with an increase of measurements, ergo relative landmark location certainty improves with more observations independently of the actual robot motion.

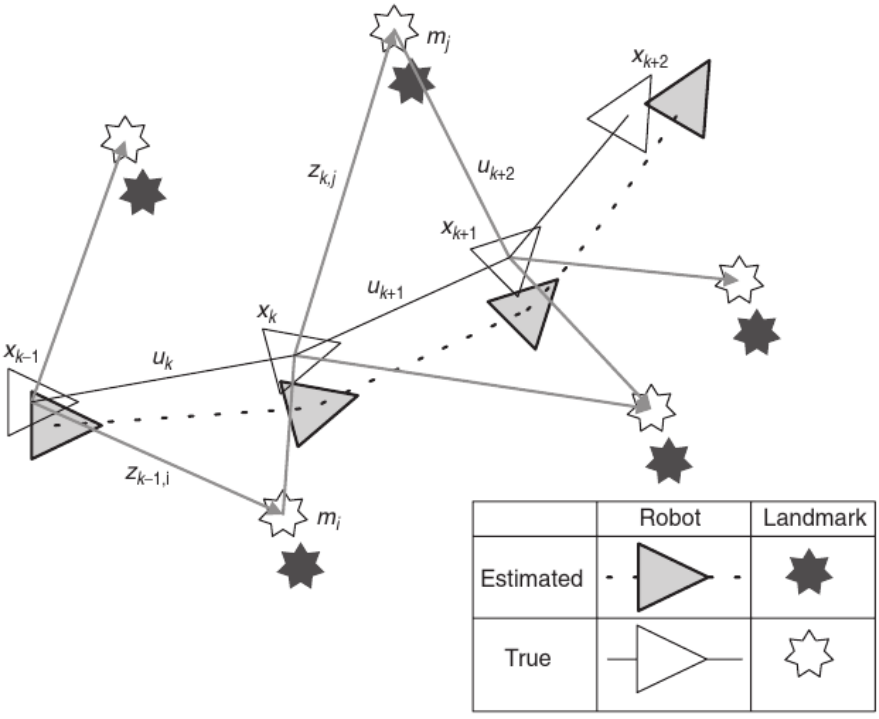


Figure 2.1: Simultaneous mapping and localization problem [54].

SLAM is most often defined as a maximum-a-posteriori (MAP) estimation problem [87][44], that can be expressed as (E2.8) and further factorized into (E2.9), if measurements are independent, ergo measurement noise is uncorrelated. Should no prior knowledge about x be available, the problem is reduced to Maximum Likelihood Estimation (MLE), since $P(x)$ becomes a constant, indicating a uniform distribution [44].

$$x_k^* = arg\,max_x P(x|Z_{0:m}) = arg\,max_{x_k} P(Z_{0:m}|x)P(x) \tag{2.8}$$

$$x_k^* = arg\,max_x P(x) \prod_{k=1}^{m} P(z_k|x) = P(x) \prod_{k=1}^{m} P(z_k|x_k) \tag{2.9}$$

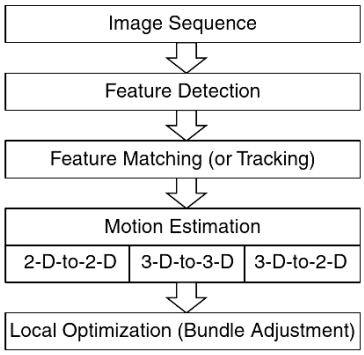
Often the formalism of factor graphs, described in *"Factor graphs"*, is used to express the relations among relevant variables [70][44]. Equation (E2.9) can be interpreted as the factorized MAP estimate and expressed in the factor graph formalism.

The main definitions and derivations for the basic SLAM problem can be found in the joint two part work of Durrant-Whyte and Bailey [54] and Bailey and Durrant-Whyte [42].

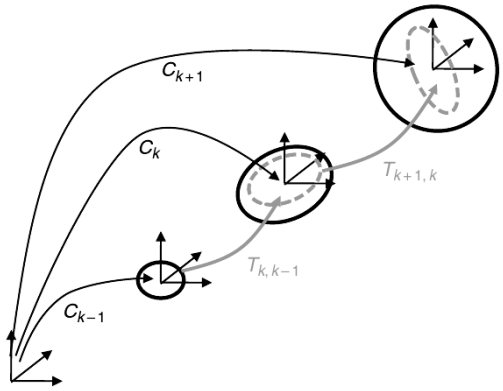
Visual SLAM

Visual SLAM (V-SLAM) corresponds to solutions of the SLAM problem using vision sensors as the main tool. Cameras are often combined with other types of sensors in order to increase the confidence of measurements. This variant of the SLAM problem often can be formulated as mapping environment and determining camera pose at given instance, ergo trajectory of poses across data stream. Depending on the architecture, representations of the map and overall approach, the solutions for V-SLAM can be very different.

Visual SLAM is directly related to Visual Odometry (VO) and Strufcture from Motion (SfM), which is described in more detail in *"Active perception"*. The main difference between V-SLAM and VO is that, the former aims to acquire a global, consistent estimate of the robot path, while the latter limits itself to a local trajectory and map consistency. VO works on a pose to pose base, determining the camera pose from frame to frame sequentially, ergo from an ordered set of images, according to the flowchart in (F2.2a) and optimizing a limited number of previous poses, also known as windowed bundle adjustment. Thus VO can be a part of V-SLAM. Visual SLAM usually has more constraints and is more precise, but not necessarily more robust. On the other hand VO is usually faster with lower resource requirements also for holding previous information [93]. Another similar approach is the Visual Intertial Navigation (VIN), which can be considered a reduced SLAM version, where also no LC takes place [44].



(a) Visual odometry flowchart [93]



(b) The uncertainty of camera pose C_{k-1} and the uncertainty of the transformation $T_{k;k-1}$ combined result in uncertainty of the camera pose at C_k [61]

Figure 2.2: Visual odometry flowchart and uncertainty of camera poses .

As with conventional SLAM approach, one of the more important aspects is the information certainty, as shown in (F2.2b), which can be increased by revisiting already explored areas. This can be interpreted by the system as a Loop Closure (LC), once detected. Visual SLAM systems have a LC detection block in their algorithms, in contrast to VO where this is no interest. Once triggered, the optimization algorithm reduces the drift that was accumulated through the LC path as shown in (F2.4). The blue path marks the LC position, the trajectory of the camera in green is updated based on the LC information, while the LC region itself is properly merged in the local map in red.

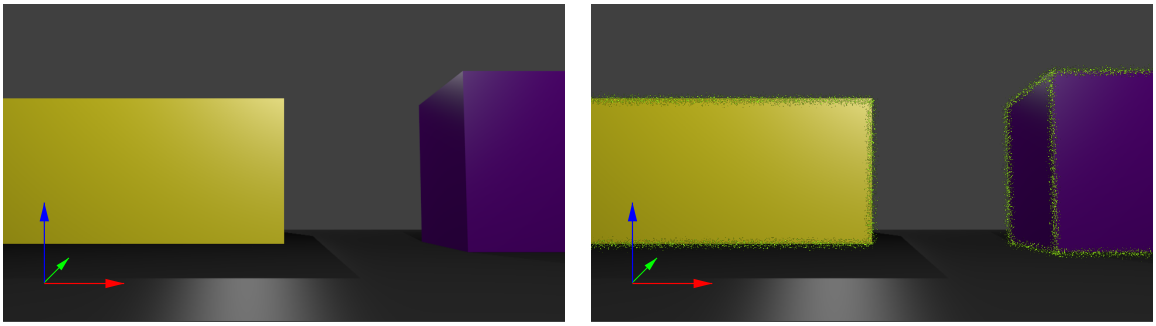
The scale ambiguity is a problem predominantly in monocular SLAM systems. However, the formulation used by approaches like Large-Scale Direct SLAM (LSD-SLAM) that operate on *Sim(3)* allows to detect the scale drift [57]. It can hold 3 dimensional similarity transforms $\mathbf{S} \in Sim(3)$ consisting of rotation matrix $\mathbf{R} \in SO(3)$, translation vector $t \in \mathbb{R}^3$ and the scaling factor $s \in \mathbb{R}^+$ defined as:

$$\mathbf{S} = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \tag{2.10}$$

This allows the SLAM system to operate within 7 DOFs, where 6 DOFs are for the position and orientation screw, and the last DOF is for the scale of environment. For a 6 DOF system in total, the scaling factor $s = 1$. For a sequence of input images $I_{0:n} = I_0, \ldots, I_n$, the transformation matrix between two frames $k-1$ and k, or registered at time instances $k-1$ and k depending on the notation convention, can be expressed as in (E2.11). This then leads to the camera transformation ${}^{k}C_{k-1}$ in (E2.12).

$$\mathbf{{}^{k}T_{k-1}} = \begin{bmatrix} {}^{k}R_{k-1} & {}^{k}t_{k-1} \\ 0 & 1 \end{bmatrix} \tag{2.11}$$

$${}^{k}C_{k-1} = C_{k-1} \cdot {}^{k}T_{k-1} \tag{2.12}$$

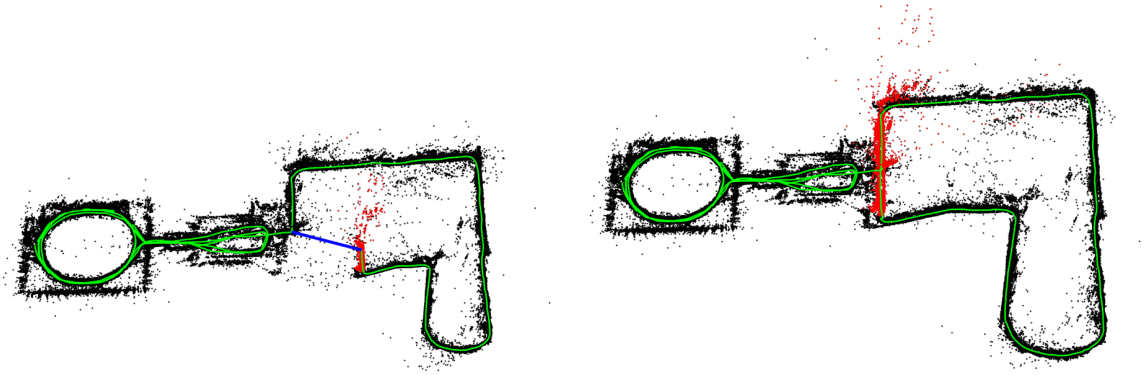


(a) Raw frame with and indicator of the current unite pose.

(b) Processed frame with an sparse approach detecting edges of objects.

Figure 2.3: Visual SLAM using camera as sensor and a sparse or feature based approach for processing.

The approaches of V-SLAM typically are either dense, semi-dense or semi-sparse, or sparse [44]. Figure (F2.3) shows rendered raw and processed frames for a specific view of the environment in a semi-sparse approach, where lines and edges are Regions Of Interest (ROI) and therefore serve as information holders of the scene.



(a) Loop closure detection

(b) Optimization after loop closure

Figure 2.4: Loop closure procedure for the ORB-SLAM approach [79].

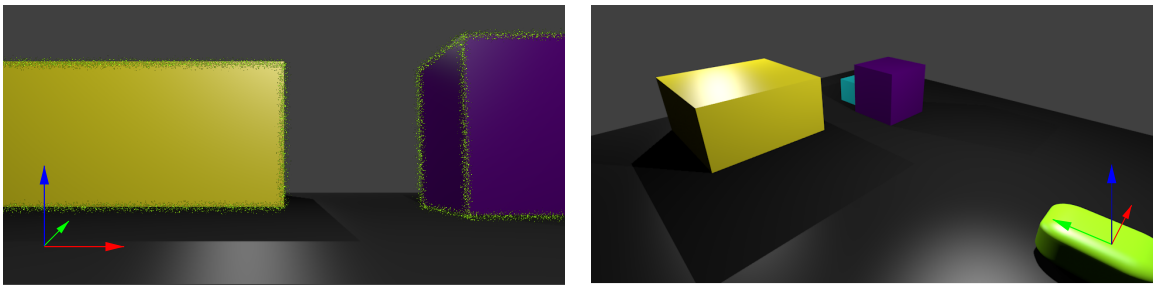
– Local map at the LC time instance – Trajectory of the camera – LC match

The main work for the V-SLAM and the related VO can be found in the joint two part work of Scaramuzza and Fraundorfer [93] and Fraundorfer and Scaramuzza [61], as well as in presentation form in Scaramuzza [92]. Furthermore a survey dedicated to visual SLAM has been conducted in Lowry et al. [74].

Collaborative SLAM

In contrast to multi robot SLAM (MR-SLAM), also known as multiple robot SLAM [91], that can be defined as SLAM performed by multiple units, a collaborative SLAM has an active effort to use capabilities of all units *actively* in order to solve the SLAM problem.

A scenario of mapping the environment shown in (F2.6) portrays how a multi or collaborative SLAM can be performed. Frames from both units, an UGV (F2.6a) and an UAV (F2.6b) are shown in processed and raw state respectively. The camera of the UGV does not have the small cube, right behind the one, in sight, while UAV can perceive it. Through communicating the map to the UGV, it can recognize that what UAV captured with the camera relates to the same perceived scene, which corresponds to a loop closure. The UGV can merge the maps from both units and in that manner perceive the otherwise blocked from view object.



(a) UGV camera view – processed (b) UAV camera view – raw

Figure 2.6: Collaborative visual SLAM scenario with UGV and UAV units.

– UGV, – Obstacles

Factor graphs

A Factor Graph (FG) is defined as bipartite graph expressing factorization structure [70]. In other words factor graphs are an abstract holding factorization relations between a global function and local functions, that depend on subsets of variables [70]. One advantage of using factor nodes is the visualization of the problem [44].

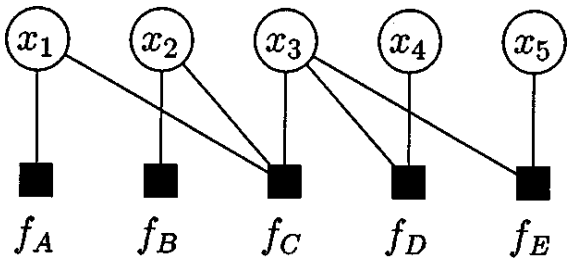
A factor graphs consists of:

- variables x_i,
- local functions f_j,
- variable nodes $\forall x_i$,
- factor nodes $\forall f_j$,
- edges $e_{i,j}$ connecting $\forall x_i \wedge f_j \; iff \; x_i = arg(f_j)$.

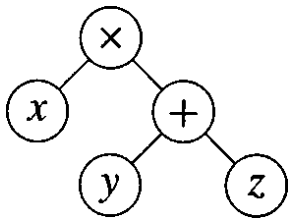
Figures (F2.8) and (F2.9) show examples of factor graphs. One of the more important aspects is the *sum-product update rule* stating:

> The message from node x_i over the edge $e_{i,j}$ is equivalent to the product of local function f_j or unit function x_j if it is a variable node, including all information received at x_i from other edges $e_{n,m} \neq e_{i,j}$ and summarized for the variable associated with $e_{i,j}$.

For the specific application of simultaneous mapping and localization, the factor graphs can be used as a map representation. Kschischang et al. [70] goes over more complex aspects in detail, like probabilistic systems



(a) Factor graph for the function (E2.13)

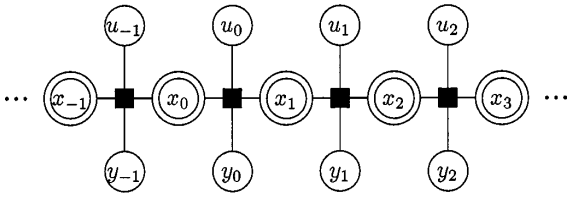


(b) Factor graph for the function $x(y+z)$

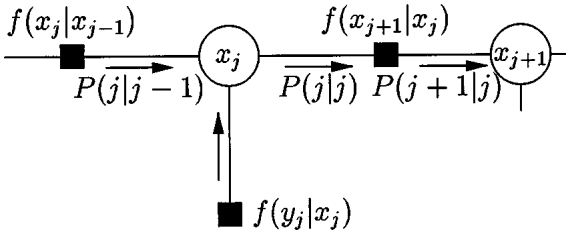
Figure 2.8: Factor graphs examples [70].

modelling, ergo collections of interacting variables. Since the factor graph can be used for representation of a posteriori joint probability mass function of variables of the system [70], it can be used directly to represent the SLAM MAP estimation.

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) \tag{2.13}$$



(a) State space model represented as a generic factor graph



(b) Kalman filter excerpt representing (E2.14) in form of a factor graph

Figure 2.9: Systems modelling using factor graphs [70].

$$f(x_1, \ldots, x_k|y_1, \ldots, y_k) = \prod j = 1kf(x_j|x_{j-1}) f(y_j|x_j) \tag{2.14}$$

Similarly equation (E2.9) can be expressed in the factor graph formalism, where nodes of the graph correspond to the variables. In particular $P(z_k|x_k)$ and the prior $P(x)$ are factors containing probabilistic constrains over a subset of nodes [44].

The main work on factor graphs formulation was done by Kschischang et al. [70].

Data processing

Data processing can be divided in a simplified way into data fusion and distribution [91].

Data fusion and processing

SLAM algorithms can fail if the data association and fusion do not work precisely. The former can provide false positives and lead the latter to inconsistent information. Data association is one of the main sources of errors in a SLAM algorithm [44]. This is of particular importance, since one of the most common solutions based on filter approach, like Extended Kalman Filter SLAM (EKF-SLAM), are usually highly sensitive to incorrect associations [80].

The correlation of measurements or visual features to landmarks, as it usually is the case in V-SLAM, can be therefore misleading in form of perceiving another landmark in the place of the correct one. This

can have drastic consequences in case of a loop closure detection, especially when the loop is long. In real world scenarios this is not uncommon, since a landmark can look differently from different angles [54], even just because of lighting conditions. This makes certain SLAM solutions particularly unfit for large scale map building [41], since the errors along the trajectory, ergo the drift, are not corrected properly after LC. Another aspect related to this problem is the data association between different frames. Correct data association is required to perform SLAM correctly.

Data distribution

The computational task mostly refers to generation and optimization of the map and can be classified as shown in (T2.1) [91][73].

Table 2.1: Data distribution description.

Data distribution	Characteristics
Centralized	Computational task is assigned to one predetermined unit with appropriate capabilities, that can be internal (part of the team) or external (server), processed and output is provided to relevant units.
Decentralized	Computational task is performed by a group of units that are required to provide sufficient capabilities to respond to the demand.
Distributed	Computational task is divided among units.
Undistributed	Computational task is performed by each unit on its own.

Active perception

Active perception or sensing can be defined as an application of control theory, such that reasoning, decision making and control are used for optimization of acquisition of desired data [43].

Structure from motion

Structure from Motion (SfM) is a well studied topic in robotics [94]. It deals with the reconstruction of 3D objects using a camera, whose motion or relative poses are also estimated. The active SfM variant covers the aspect of the optimal control for the purpose of recovering missing structure from the scene. V-SLAM, VO and SfM are all correlated, visual odometry is in fact a specific case of SfM. Final optimization is usually performed using bundle adjustment and the computational time is proportional to the provided image sets. The BA can be applied on parts of the data structure or globally. The frames do not need to be in a sequence, should that be the case however, then the relations between non-consecutive frames can be used as additional constraints [93]. Furthermore an on-line approach is then possible, since SfM could be treated as a recursive or filtering task.

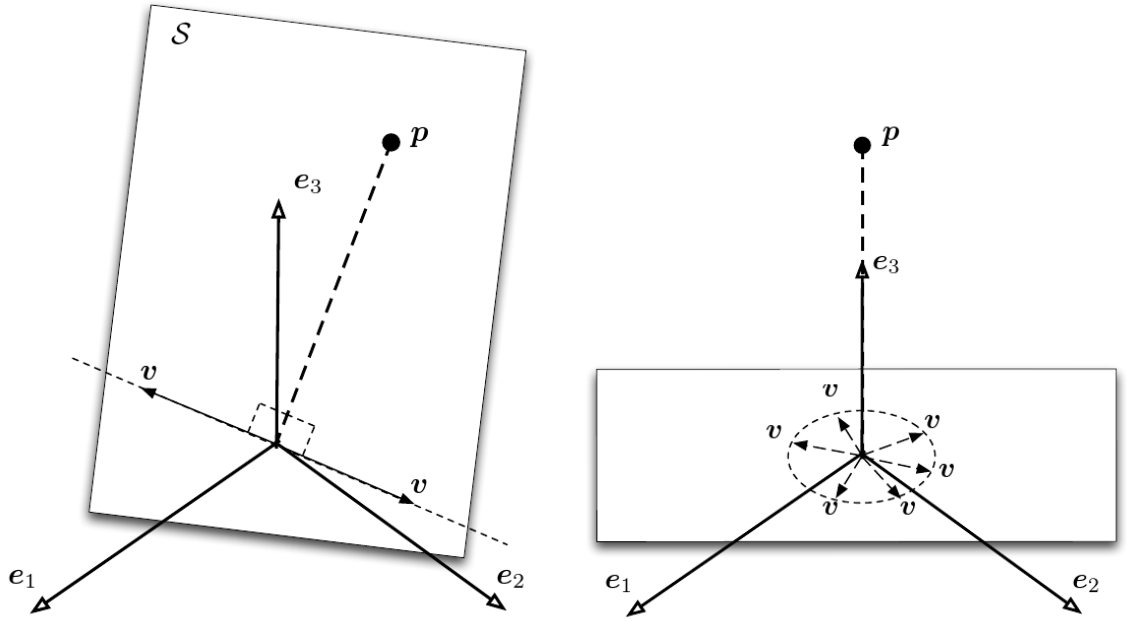
For the purpose of the master thesis, structure from motion is of less importance, however the active SfM provides insights into optimization of reconstruction time.

A non-linear active structure estimation for a 3D point, a sphere and a cylinder can be formulated as an estimation problem of a single quantity if the parametrization is chosen properly [94]. The described approach is an on-line and active algorithm for structure estimation, assuming the camera motion is known and controlled. Furthermore the solution is capable of imposing a desired and optimize-able error transient response, that can be represented as a linear 2nd order system with specified poles. This allows for taking into account real world constraints like limitations of camera velocity.

The approach goes from a system description with a measurable s and unmeasurable X component to an observer with an total error $e = (\xi, z)$, consisting of the measurable $\xi = s - \hat{s}$ and unmeasurable $z = X - \hat{X}$ component. This error can be shaped, as its transient response can. The two main components

capable of speeding up the convergence are the gain and camera velocity V_c. Since often there are limits on camera velocity, the adjustment of the gain can decrease the time for converging.

The results show that the best camera movement is across a line that is a normal to the plane spanned by the two vectors of the optical axis and the projection vector of point P as shown in (F2.10a). The error e is then minimized in an optimal manner. If the two vector coincide as shown in (F2.10b), then any camera motion, ergo in any direction is valid. This state however does not provide optimal information. Therefore in order to optimally reconstruct a point object the camera should not point at it, but still have it in the FOV. Furthermore it has been concluded that optimal positions for the point feature are in the 4 corners of the image plane of the pinhole camera.



(a) Projection of point P offset from optical axis (b) Projection of point P coincides with optical axis

Figure 2.10: Optimality conditions for camera linear velocity [94].

Work on the aspect of active perception has been done by Bajcsy [43]. Recently Spica et al. [94] and Spica et al. [95] have made interesting contributions to the active SfM problem.

Visual servoing

Visual Servoing (VS) or visual servo control is defined as a robot motion control using computer vision information [47] and can be defined in general as (E2.15).

$$e(t) = s(m(t), a) - s^* \tag{2.15}$$

The components are:

- vector $m(t)$ holding image measurements,
- vector $s(m(t), a)$ of k visual features,
- vector s^* storing desired visual features,
- set a consisting of parameters representing potential additional knowledge about the system,
- $e_{i,j}$ connecting $\forall x_i \wedge f_j \; iff \; x_i = arg(f_j)$.

The aim of visual servo control is to minimize the error $e(t)$. The way visual features s are defined is the main difference between schemes, from which of the most popular are:

- Image Based Visual Servoing (IBVS),
- Position Based Visual Servoing (PBVS),
- Hybrid 2D/3D Based Visual Servoing (HBVS).

For a velocity screw of the camera $V_c = \begin{pmatrix} v_c \\ \omega_c \end{pmatrix}$, with v_c and ω_c being the instantaneous linear and angular velocities of the camera frame F_c, the interaction matrix $\mathbf{L_s} \in \mathbb{R}^{k \times 6}$ representing the relationship between the change in visual features $\dot{s}$ and camera velocity V_c can be established. For $\mathbf{L_s} = \mathbf{L_e}$ equation (E2.17) can be obtained. Assuming V_c as control input for the robot to manipulate the camera and an exponential decoupled decrease of error $\dot{e} = -\lambda e$, equation (E2.18) can be derived [47].

$$\dot{s} = \mathbf{L_s} V_c \tag{2.16}$$

$$\dot{e} = \mathbf{L_e} V_c \tag{2.17}$$

$$V_c = -\lambda \widehat{\mathbf{L_e^+}} e \tag{2.18}$$

The Moore-Penrose pseudoinverse [**?**] of $\widehat{\mathbf{L_e^+}}$, ergo $\mathbf{L_e^+} = (\mathbf{L_e^T L_e})^{-1} \mathbf{L_e^T}$, is an approximation of the $\mathbf{L_e^+}$ that in practice cannot be known perfectly.

Since the master thesis has its initial main focus on collaborative and decentralized SLAM aspects, the only basics of VS are introduced, where further details can be found in the extensive VS tutorial consisting of two parts Chaumette and Hutchinson [47] and Chaumette and Hutchinson [48] for the more advanced approaches.

Stereo vision

Epipolar geometry, stereo vision and depth estimation are correlated aspects [40].

Stereopsis

Stereo vision or stereopsis refers to the aspect of extracting 3D information, like perception of depth and 3D reconstruction, from two or more images acquired from different viewpoints. This is similar to the functioning of two eyes in human vision.

The difference in retinal position of the eyes helps build the 3D perception of the world in human brain. Human eyes are positioned side-by-side. Each eye takes view of the same area from a different angle. The two images arrive simultaneously in human brain and are then combined. The small differences in the two images result in building 3D stereo picture. The same principle is applied to camera images taken from two

different viewpoints. Here feature points are detected, matched and their positions compared. In a calibrated stereo vision system, which implies knowing internal and external camera parameters.

Epipolar geometry

Epipolar geometry illustrated in (F2.11), is a very significant part of stereo vision. It refers to the geometric relations between 3D points in the world frame and their projections onto 2D image planes of the cameras, acting as a constraint between the image points.

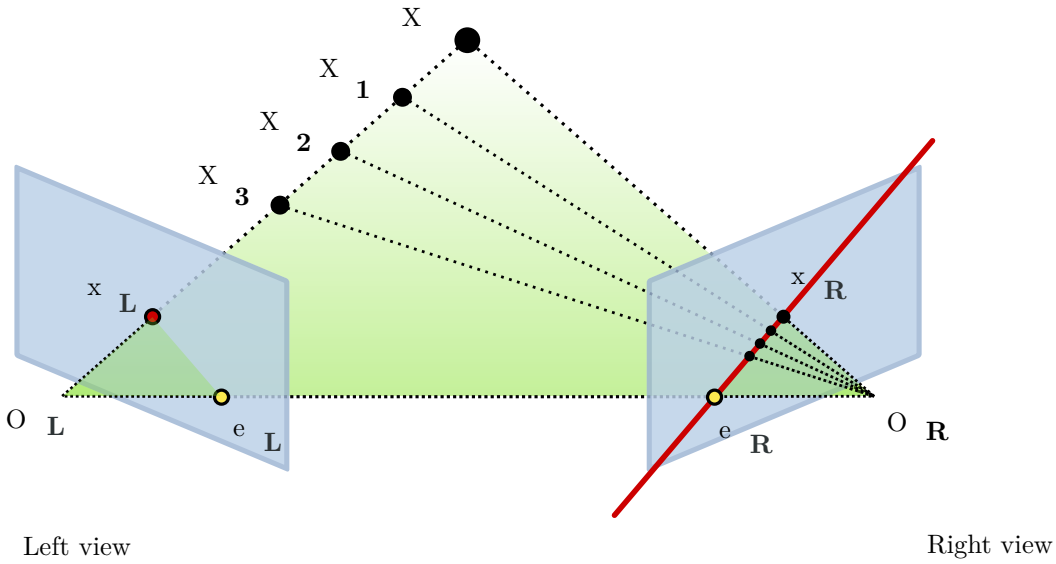


Figure 2.11: Epipolar geometry [82].

Two image planes of a stereo vision system are illustrated in blue for the two cameras, respectively left/main and right/secondary. The virtual image planes are placed in front of the focal plane of each camera. Points O_L and O_R represent the origins of symmetry of the two cameras lenses. X represents the point of interest in the 3D world, while x_L and x_R are the projected points on left and right image plane respectively.

1. Epipole: Projection of optical centers of the cameras lenses O, into the other camera image plane:
   (a) Left epipole: Projection of O_R on the left image plane e_L.
   (b) Right epipole: Projection of O_L on the right image plane e_R.
2. Baseline: Line connecting O_L and O_R. The baseline intersects each image plane at the epipoles e_L and e_R.
3. Epipolar plane: Plane containing 3 points in space X, O_L and O_R.
4. Epipolar line: Intersection of the epipolar plane with the image plane.

The line $O_L - X$ is seen by left camera as a point, because it is directly in line with that camera's centre of projection. This means the points X, X_1, X_2, X_3 on this line will be projected onto x_L. However, the right camera sees this line as an actual line in its image plane. The projection of this line is in fact an epipolar line. In the same manner, line $O_R - X$ is projected on the epipolar line $x_L - e_L$ on the left image plane.

The epipolar plane, shown in green in the figure, contains the baseline O_L–O_R and intersects the image planes at point e_R and e_L. For each point in an image plane, its corresponding point in the other image can be found by looking only along its epipolar line. This is called an **epipolar constraint**. The epipolar lines are an important constraint for relation between corresponding points of two stereo images.

Fundamental matrix F

In stereo vision geometry, points in one image plane can be mapped on to the epipolar lines of the others. The basic relation of fundamental matrix F with the corresponding points is given by equation (E2.19). It can be considered a generalization of the essential matrix [64].

$$x_R^T F x_L = 0 \tag{2.19}$$

Following are a few properties of fundamental matrix playing important role in its estimation and applications:

1. The matrix encodes information on both the intrinsic and extrinsic parameters.
2. It is a 3×3 homogeneous matrix of rank 2, with 7 degrees of freedom.
3. If x_L and x_R are corresponding points, $x_R^T F x_L = 0$
4. For the same condition, if x_R and x_L are corresponding points, $x_L^T F^T x_R = 0$
5. Epipolar line x_R–e_R corresponding to x_L is equal to $F x_L$.
6. Similarly, epipolar line x_L–e_L corresponding to x_R is equal to $F^T x_R$.

Essential matrix E

The essential matrix is a specialization of the fundamental matrix [64]. In particular normalized image coordinates are used. The essential matrix E with the corresponding points is given by equation (E2.20).

$$x_{Rn}^T E x_{Ln} = 0 \tag{2.20}$$

Alternatively the essential matrix E can be defined exactly by the equation (E2.21), including the scale, where the vector cross product matrix $[t]_\times$ for the translation t between image planes is defined by (E2.22) and R is the rotation matrix between image planes.

$$E = [t]_\times R \tag{2.21}$$

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{2.22}$$

Following are properties of essential matrix with respect to fundamental matrix:

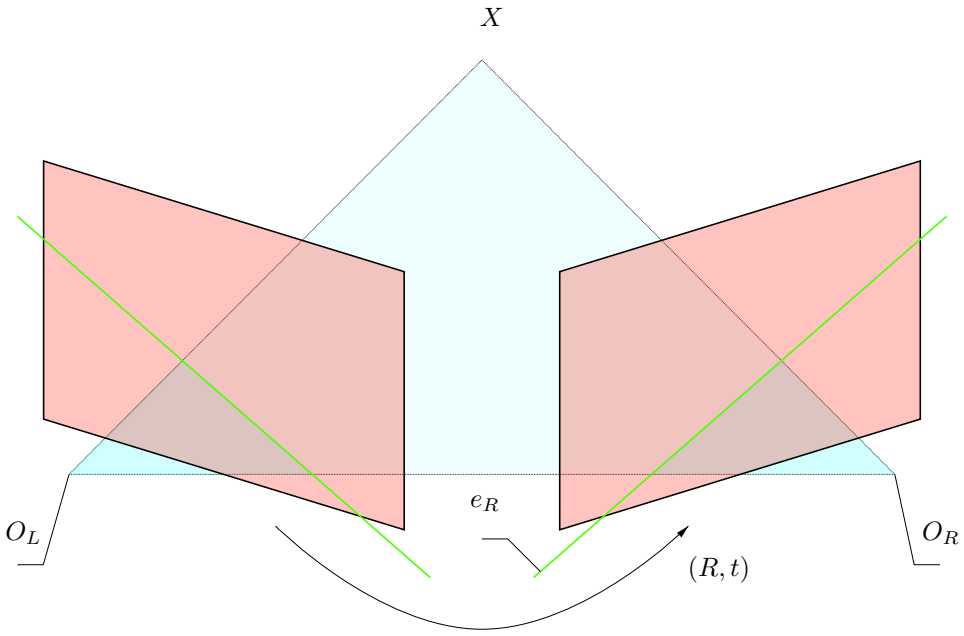
1. The matrix encodes information the extrinsic parameters, intrinsic parameters have to be known.
2. It is a 3×3 homogeneous matrix with only 5 degrees of freedom.
3. If x_{Ln} and x_{Rn} are corresponding points in normalized image coordinates, then $x_{Rn}^T E x_{Ln} = 0$

Image rectification

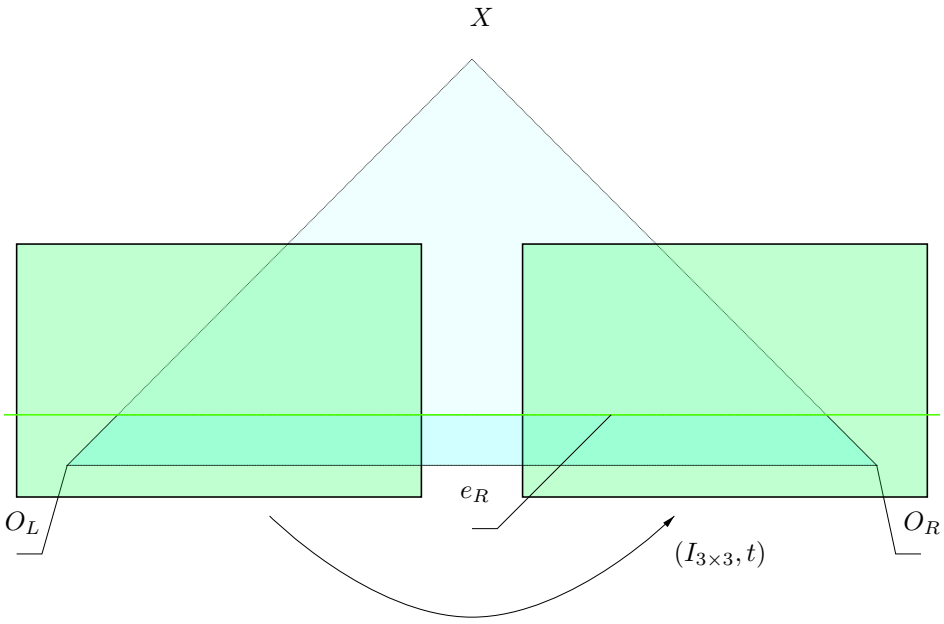
In stereo vision geometry an important issue is image rectification. It is a process of transformation of a stereo vision system into a one with simplified epipolar constraint, where all the epipolar lines are parallel and coinciding with an image axis. This results in corresponding points being located on the same horizontal or vertical line, depending on the geometric arrangement of the stereo vision system, ergo having the same corresponding image coordinate for one axis [86].

An image at a given resolution is the multiplication of width by height, can be reduced to just one line of one of the dimensions, usually the horizontal resolution, since most stereo vision systems are resembling human sight. The reduction in computation load is proportional to the skipped dimension to resolution as well as to the number of corresponding points.

The result of image rectification with respect to (F2.11) has been illustrated in (F2.12). From each image planes, the main and secondary one, the transformation can be achieved towards the other by horizontal translation only.



(a) Not rectified stereo vision system with a rotational and transitional components.



(b) Rectified stereo vision system with horizontal translation only.

Figure 2.12: Epipolar lines before and after image rectification of a stereo vision system with a projected 3D point X onto main/left and secondary/right image plains.

Chapter 3

State of the art

Overview of current state of the art has been presented most recently in Cadena et al. [44], where an current developments and open problems were addressed in detail. Additional information are provided in the single SLAM – Aulinas et al. [41] and multi SLAM – Saeedi et al. [91] surveys.

Simultaneous localization and mapping

Simultaneous mapping and localization is still being actively developed in several fields of research [44], especially in aspects of multi robot approaches [91].

Architecture

The architecture of a typical SLAM system shown in figure (F3.1). It consists of a back-end and a front-end, where the former performs inference on the data produced by the latter [44]. The input is the data from sensors and the output is usually a Maximum-a-posteriori (MAP) estimation [87], the current standard SLAM formulation [44]. There can be feedback coming from the back-end to the front-end.

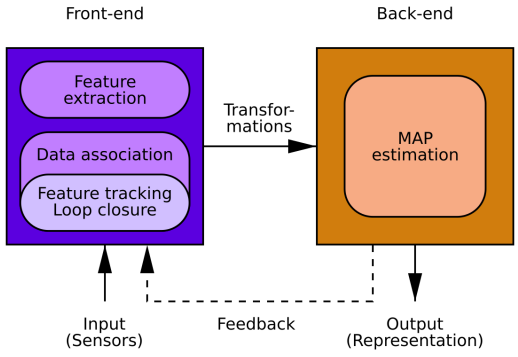
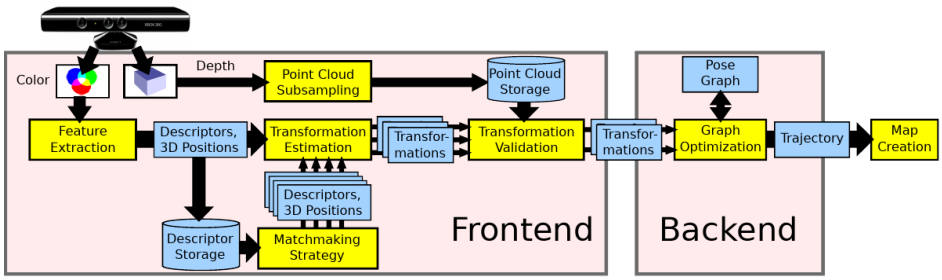


Figure 3.1: SLAM architecture consisting of a back-end and a front-end with sensors data as input and a map as output[44].

The architecture is specific for each SLAM approach. For an RGBD based system the architecture in (F3.2a) was used and the visualization of the generated information was performed using voxel occupancy map [56][55].



(a) SLAM architecture for the RGBD approach [56]

(b) Occupancy voxel representation for the same RBGD method [55]

Figure 3.2: RGB SLAM approach architecture and map visualization.

Sensors

A SLAM algorithm can utilize a wide variety of sensors and tools [44][91], including these shown in (T3.1). Some of these fields intersect. For example RGBD sensors use both, vision and distance from IR (infra-red) registered laser reflections. Some are restricted to limited spaces, for example a motion capture system. The main aspect influencing a choice of sensors is the specific user case.

Representations

Representation deal with modelling geometry in SLAM [44]. The two main approaches are metric and semantic reasoning.

Metric reasoning

Metric reasoning representation encodes the environments structure. For 2D scenarios predominant are occupancy grid, ergo discretization of environment into cells, and landmark based maps, which models environment as a set of sparse landmarks. For 3D scenarios there are several representations that are still being developed and improved, including landmark based sparse, low-level raw dense, boundary and spatial-partitioning dense and high-level object-based representations.

Table 3.1: List of sensors compatible with current SLAM approaches

Range	Odometry and direct localization	Vision	Sound
• Infra-red • Laser • LIDAR (Light Detection And Ranging) • Sonar	• Encoder (ground vehicles) • Differential pressure (air vehicles) • GPS – Global positioning system • IMU – Inertial measurement unit • Magnetometer • Motion capture system	• Camera – Monochrome – Broad FOV cameras (360 deg) – RGB (color) – Camera model • RGBD (color and depth) • Multi camera system • Range camera • Light field camera visual • Event based camera	• Acoustic (echolocation) • Sonar • Sodar

Semantic reasoning

Semantic reasoning deals with associating geometry in the environment though semantics, allowing this way for a higher level of abstraction. It is built upon level and detail as well as organization of semantic concepts.

Map representations

Map representations for SLAM algorithms can be divided into feature, view or location, appearance, occupancy grid, semantic, polygon and hybrid based [91].

For particularly V-SLAM the level of map abstraction can be divided into [74]:

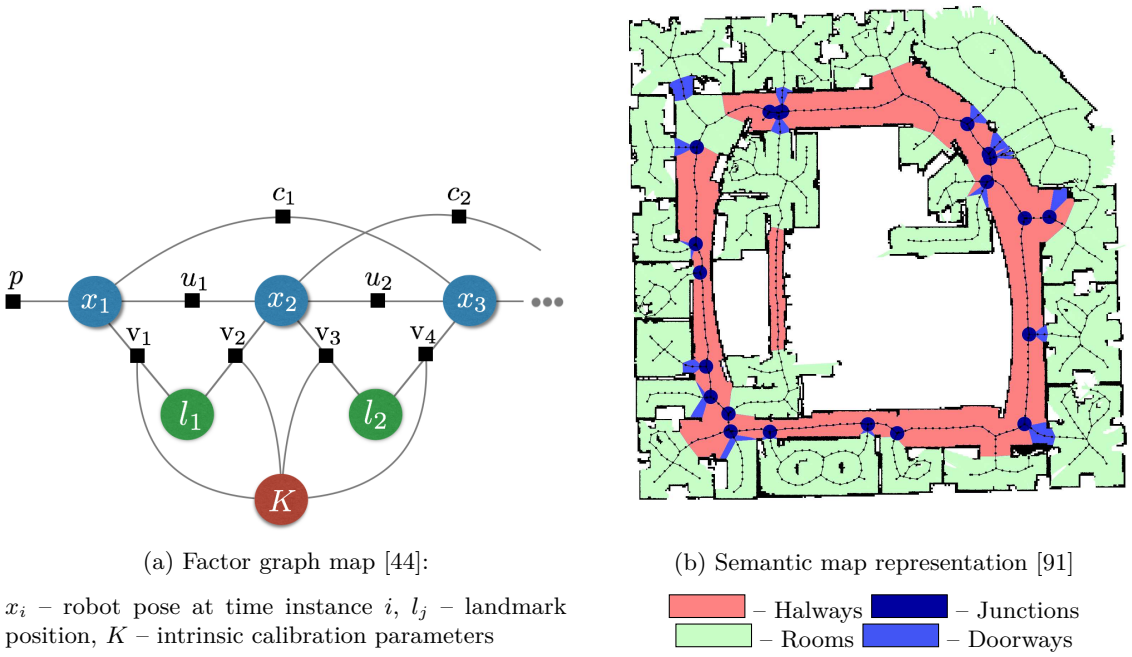
- Pure image retrieval – No further position information, use of visual image for information only
- Topological – Relative positions only, no metric data about the environment
- Topological-metric – A combination of image and relative position information

Solutions

SLAM problem solutions are spread over a variety of fields. Vision is one of the most dominant tools that is used to solve the problem, but so are laser and other general range based approaches. For one mobile unit the most common solutions include Kalman Filters (KF) (EKF, UKF, etc.), Particle Filters (PF), Expectation Maximization (EM) and Pose Graph (PG) methods, whenever the multi unit case still is developing [91]. The filter methods are for the most part derivatives of the Bayesian Filter (BF) [41], ergo recursive Bayesian theorem [54] and still dominate in SLAM [41]. Factor graph based PG, is a combination of camera poses and the conventional factor graph applied to the SLAM problem. Often keyframes are being also used in pose graphs holding the relevant visual information. The main advantage of FG based approaches is the potential for complexity. Very abstract systems can be easily implemented. This of course increases computational requirements too.

Different SLAM techniques with respect to map representations can be classified based upon [91]:

- feature – sparse approach, where features are extracted from the environment and build the map



(a) Factor graph map [44]:

x_i – robot pose at time instance i, l_j – landmark position, K – intrinsic calibration parameters

(b) Semantic map representation [91]

Figure 3.3: Factor graph and semantic map representations.

- view or location – uses entire patches of environment instead of just features, usually used with range based sensors, can handle dynamic environments better than feature based
- appearance – efficient with LC, new measurements are matched against previous, computationally demanding
- polygon – highly detailed maps, computationally demanding

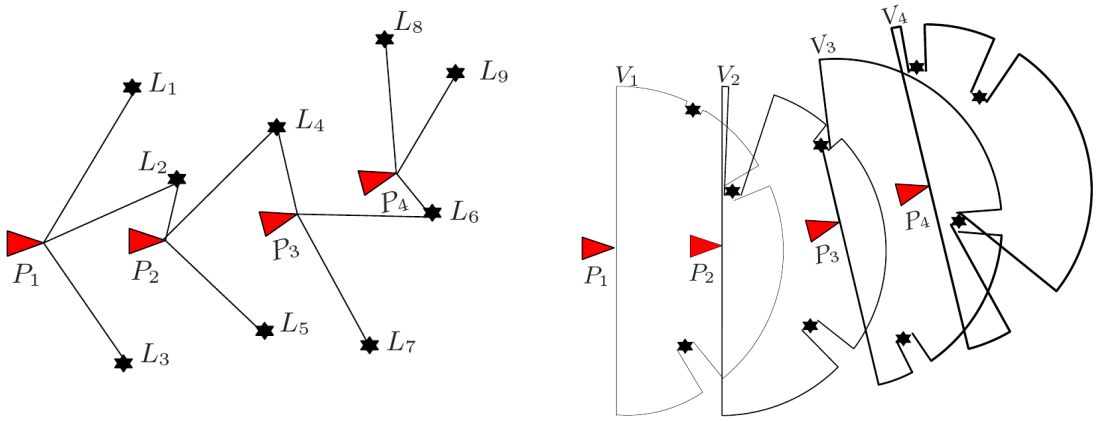
and with respect to different data processing approaches can be classified into following SLAM techniques [91]:

- filtering – derivatives of BF, used for estimation of current pose or optimization of the global trajectory
- smoothing – optimize the whole trajectory
- Artificial Intelligence (AI)
  - topological
  - semantic

Feature and view based maps and algorithms are illustrated in (F3.4).

Of particular interest are two V-SLAM solutions: Large Scale Direct SLAM (LSD-SLAM) [57][58] and Direct Sparse Odometry (DSO) [59], where the latter requires additionally a LC detector and merger for SLAM.

LSD-SLAM is a direct, ergo featureless, method operating on intensity values, capable of real time operation. It uses a PG to contain keyframes with associated probabilistic semi-dense maps [58]. Similarity transformation on $Sim(3)$ is used to align two consecutive frames and also solves the scale ambiguity problem. Furthermore estimated depth maps have noise modelled into them based on a probabilistic approach. The representation is appearance based or semi-dense, ergo a point clouds, capable of precise 3D environment reconstruction. Test results prove robustness and versatility, even hand held devices can execute the algorithm.



(a) Feature based representation (b) View or location based representation

Figure 3.4: Feature based and view based representations [91].

Similarly to LSD-SLAM, DSO is a direct, and therefore a featureless, and sparse approach capable of real time operation [59]. It optimizes the photometric error based on images over a window of recent frames and the full likelihood for all involved model parameters, including camera poses and intrinsics, inverse depth values as well as geometry parameters. Since the algorithm is implemented as monocular VO, it does not deal with loop closures. Additionally, a full photometric calibration is incorporated. The combination of direct and sparse formulation for SfM provides benefits from both. For the direct part it is the ability to reconstruct all points and not being limited to certain features. The sparse approach enables an efficient operation and joint optimization of all model parameters. Prior geometric information are skipped in favour of computations on the photometric error.

Since the master thesis deals with an collaborative decentralized heterogeneous SLAM approach, it is not of high priority to detail all aspects of certain methods. Additionally, since the aim is to conduct experiments, the most probable approach will be to influence selection a or several SLAM techniques depending on hardware capabilities of the equipment.

Collaborative SLAM

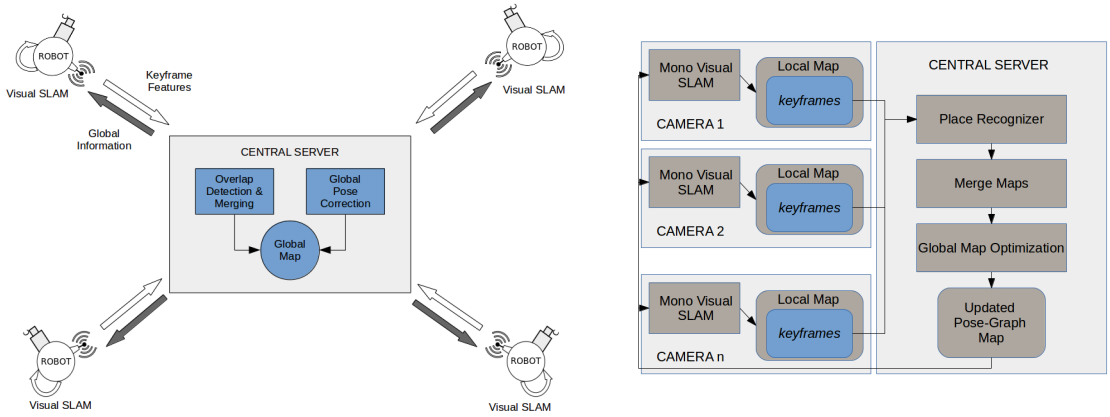
Multi robot approach is a general term for multiple units performing a task, while collaborative SLAM has an active component in the control in order to achieve the goal. In order to map environments efficiently, teams of robots can be deployed, each conducting SLAM in a smaller part of the whole area. There are two main types of this multi SLAM approach: centralized and decentralized. The former has a central unit receiving local maps from each unit, processing it into a global map and sharing it with robots. The latter has no central data fusion, exchange of information occurs locally between units within communication range [44].

Collaborative Visual SLAM Framework for a Multi-Robot System

The collaborative visual SLAM is a centralized approach, where each unit equipped with a monocular vision system performs V-SLAM individually and is structured according to (F3.5) [49].

The described algorithm used for monocular SLAM estimates 7 degrees of freedom, including the scale of the scene using the group $sim(3)$. The environment is mapped to a pose graph of key frames that hold a depth map obtained by a semi-dense approach based on LSD-SLAM [57]. A central server runs a place

recognition software that constantly monitors transferred data from mobile units (F3.5a). In case of an overlap detection, that is performed in appearance space by comparison of extracted visual features from the key frames with the help of Bag of Words (BoW) technique, the map merging algorithm is executed. RANSAC version of the traditional Horn's algorithm is utilized for the initial transformation estimate between matched key frames from the overlap detection block (F3.5b). Optimization is performed using the calculated estimate as a starting point for similarity transformation estimation, that is additionally processed using and iterative closest point (ICP) algorithm. As next step map merging is conducted into a global map with an additional constraint between the two matched key frames. Finally optimization using bundle adjustment is performed on the global graph and information in form of updated poses is communicated to mobile units as feedback. The robots then perform optimization of the localization of the unit, based on the newly received and already stored information.

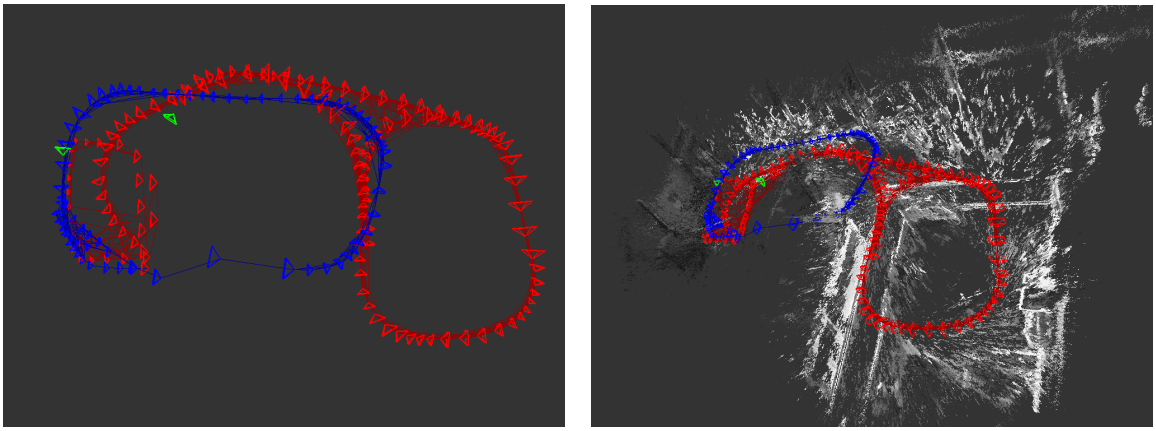


(a) Centralized approach architecture (b) Procedural scheme

Figure 3.5: Structure of the collaborative visual SLAM approach [49].

The overall system can be listed as following function blocks:

- Monocular V-SLAM
  1. Visual features tracking
  2. Semi-dense depth map estimation
  3. Optimization and management of the map
- Place recognizer
- Map merge
  1. Initial transformation estimate using Horn's method
  2. Estimate improvement using $sim(3)$ tracker
  3. Map correction using ICP
  4. Global map update
- Feedback system



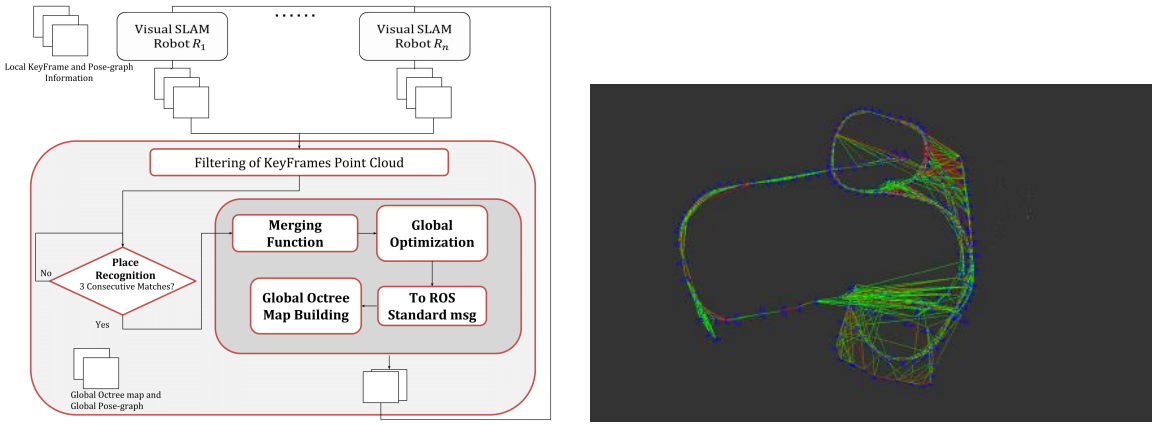
(a) Merged map holding the trajectory of both Turtlebots

(b) Generated depth map

Figure 3.6: Results of the collaborative visual SLAM approach on the global map[49].

Collaborative Visual SLAM

Work for the collaborative visual SLAM is based and a continuation of [49][46][45]. Instead of continuing to use LSD-SLAM, a direct approach was utilized in accordance to the system structure shown in (F3.7a). The proposed solution was experimentally validated, the computation were done off-line, however the system was able to combine over 200 keyframes, each with 300 000 points on average. The resulting optimized pose graph is shown in (F3.7b).



(a) Overall system structure

(b) Optimized global pose graph

Figure 3.7: System structure and global pose graph [46].

Map API - Scalable Decentralized Map Building for Robots

The described approach is a distributed and decentralized SLAM back-end with the characteristics of a reduced bandwidth, asynchronous updates to registered units and collaboratively build a global map without a central supervisor unit Cieslewski et al. [51]. Data are hold in the swarm and can be searched through using distributed hash tables. Modifications to the data need to be synchronized however, which raises complexity of this approach, therefore a map version control system was proposed, which allows simultaneous read and write access to the map data to multiple units. This is done with the help of the concept of views, which

is basically a local copy of specific map segment made at the request time. Updates can be committed to the swarm database similarly to conventional version control systems like git. In order to avoid conflicts a registration and trigger system has been proposed, where a unit can register to perform operations on the map database. This helps with reduction of communication bandwidth, which is crucial for distributed systems. One of the main open problems is the connection loss aspect.

MAP API architecture consists of 3 layers:

- top – executables layer,
- middle – application layer,
- bottom – generic map data management.

DDF-SAM: Fully Distributed SLAM using Constrained Factor Graphs

The Smoothing And Mapping (SAM) and Decentralized Data Fusion (DDF) was combined as a solution to the SLAM problem in Cunningham et al. [52]. The described approach consists of three modules:

- local module: Running SAM on a single unit, generating condensed local graph, performing local optimization,
- communication module: Transmit local graphs between units,
- optimizer module: neighbourhood graph optimizer combines local graphs into maps of the vicinity of a unit.

Main requirements for the described DDF system consist of scalability in computational cost, communication bandwidth, especially when the number of robots increases, and robustness to node failure as well as to changes in network topology [52]. The described DDF-SAM introduces the Constrained Factor Graph (CFG) approach, that fulfils set requirements. Assumptions of the method include unique identifiers for landmarks and known data association. Additionally, each unit is assumed to be capable of performing SLAM independently and the observed landmarks can be at least correlated between robots. Furthermore the positions of units are not known to each other. Finally, the neighbourhood map is assumed to be landmark based only, which is significant for the limited communication aspect.

The SAM problem can be expressed as (E3.1) and further as linearised least square sub-problem (E3.2),

$$X^* = argmin_x \frac{1}{2} || \underbrace{h(X)}_{\text{generative observation model}} - Z||^2_\Sigma \tag{3.1}$$

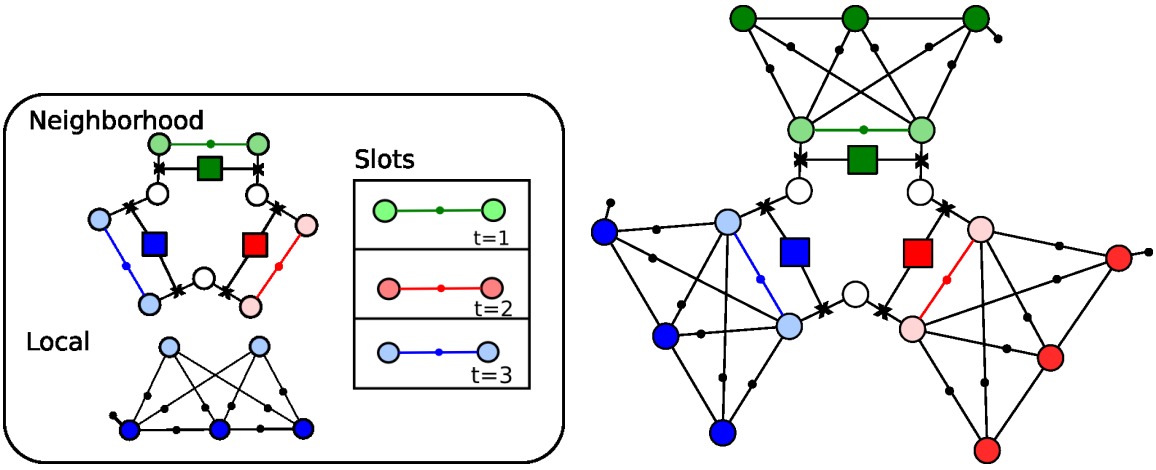
where Σ is the measurement covariance matrix.

$$\delta^* = argmin_\delta \frac{1}{2} ||A\delta - b||^2_\Sigma \tag{3.2}$$

The local optimizer module holds the main SAM algorithm. Variables elimination is the first step, which is possible due to the sparse representation of the SAM problem, and leads to a condensed graph and higher computational efficiency, given a sparse matrix solver. For the compact representation, the poses are eliminated through re-linearisation around best current estimate of the state X^*, after which the poses are discarded. The information about landmarks is untouched.

The local neighbourhood graph is build from condensed graphs, which are stored as a local cache for each robot in slots S with relevant robot identifiers and timestamps. Data transfer between robots performed by the communication module, usually is limited to two robots, where the number of total available robots can vary depending on network reachability. The communication procedure consists of two steps: the announcement of available condensed graphs and the transfer itself of updated graphs.

The neighbourhood optimizer module merges cached in slots condensed graphs into a single neighbourhood map with the help of Constrained Factor Graphs (CFG), which represent transforms between local and neighbourhood reference frames. A base frame of reference is introduced and shown as squares in (F3.8), which express neighbourhood origin in the robot frame. Direct assignment of the neighbourhood to local landmark copies is marked with a cross. Nonlinear optimization takes place on the minimized graph data in order to create a neighbourhood map based upon the base frame of references.



(a) Structures managed by each unit (b) Partially eliminated local graphs

Figure 3.8: Structures and partial elimination in DDF [52].

Structures stored by each unit are shown in (F3.8a).

Distributed and Decentralized Cooperative Simultaneous Localization and Mapping for Dynamic and Sparse Robot Networks

One solution of a collaborative SLAM is the distributed and decentralized approach for dynamic and sparse robot networks described in Leung et al. [73]. It defines the distributed aspect as sharing of computation load among available units and decentralized as that each unit estimates just its own state comprised of all poses and landmark positions. This approach provides a guarantee that all robots can recover the centralized equivalent estimate. The described approach is heavily based upon concept of checkpoints, in particular distributed checkpoints, that is events at specific time and place holding previous information in form of state estimates and measurements. This concept allows to reach consensus estimate among the swarm. Furthermore there is need for a unit to keep track of actions of other units and it can apply the Markov property after generating consensus estimate.

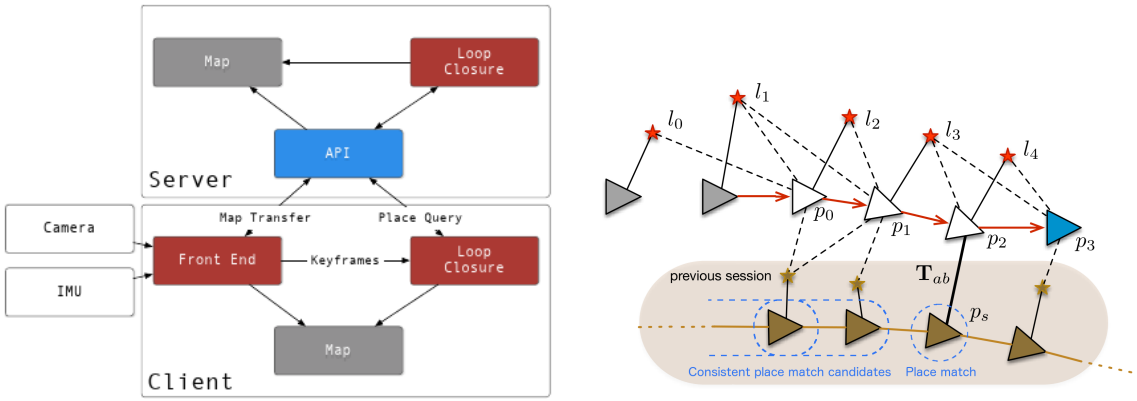
The last can be also seen as a disadvantage, since there is little to no coordination between robots, ergo there is no control over environment mapping time or ensuring a certain level of mapping quality.

Scalable Multi-Device SLAM

Crowdsourced mapping from multiple devices has been proposed by Morrison et al. [78]. The algorithm can be run on cellphones, which allows an ever growing user base to cooperatively create large scale maps using the internet infrastructure. This is usually a problematic aspect with SLAM [44], beside the scale ambiguity problem, scaling does not always deliver satisfactory results, since errors are also scaled. Additionally each time a new information occurs the global map usually needs to be optimized in its entirety. That is of concern for maps of large scales.

The client-server architecture of the approach is shown in (F3.9) is capable of simultaneously build and share large scale 3D maps from multiple monocular devices. The maps are stored on the servers and can be generated by clients in multiple sessions. The algorithm running on the cellphones has capability to fully perform SLAM, which allows the clients to work off-line and submit generated maps whenever on-line. Maps are received by the clients from the servers after submitting an query, that uses images and an non complex Application Programming Interface (API), which consists of 3 requests types: queries of places, downloads and uploads. Because of purely image driven queries, accurate global location is not needed. Detected loop closures trigger map merging by sending a query to the servers, however features computed by the front-end are reused instead of recalculating them. Loop consistency is checked by using the map graph.

The map representation introduced by Mei et al. [76] consists of relative pose graph with metric landmark information. Nodes store key frames, which hold landmarks and their information, edges hold the relative transformations. It used by the server and client side alike and because of its relative framework, it's possible to generate large scale maps. This factor graph approach allows for limited resources usage in form of storage and operational memory consumption, which makes it possible to expect constant level long-term performance. Each session is associated by an identified by an Universally Unique IDentifier (UUID), that is then used to label all objects of the system, providing uniqueness among the whole hierarchy. Additionally information about camera calibration is stored in each session. This ensures certainty of relating to the same objects between servers and clients. One of the main characteristics of this representation is the ability to interact with parts of the map directly, without the need to transfer more than necessary information. This is due to the use of multiple frames, which make the information about a global coordinate frame obsolete. It allows for simultaneous update of the map from multiple sources.



(a) System architecture outline.

(b) Constraints after loop closure.

Figure 3.9

The API allows for the following calls:

1. Place Query
   (a) Client provides a query in form of an image
   (b) Server searches for a match in the database
   (c) Key frame is loaded, relative position based on it is estimated
   (d) New edge is connecting the found frame and the query frame and is transmitted to the client
   (e) Access to landmarks based on the provided connection to the global map allows for localization and Breadth-First Search (BFS)
2. Map Download

(a) Request a map section using parameters frame ID and BFS depth

(b) Establish nearby connections to the rest of the map

(c) Send found information, including all related objects, and the potential connections to the client

3. Map Upload

(a) Client order all objects

(b) Server receives information from the client

(c) Map merging occurs if loop closure is detected among stored and received data

The loop closure system works with Oriented FAST and Rotated BRIEF (ORB) descriptor and binary Bags of Words (BoW). The LC detector is used by both, the servers and clients. Places in the environment are described with the help of BoW. The client computes features using the ORB descriptor, which ensures, that there is a small number of points, that show high repeatability of detection and are distributed across the image. Associated landmarks have their 3D poses estimated. Reuse of features already computed improves calculation time and reduces energy consumption. A search is started using descriptors converted into a BoW vector against the global map. The matches are filtered by using the top 100 candidates, that are grouped if their reference frames are below a threshold in the map. The group with the highest score is used. Additional information from IMU or stereo systems can be used. Loop closure procedure is initiated when minimum 3 points are close to each other, as shown in (F3.9b). The descriptors are compared, their relative correspondences to the 3D landmarks are computed and used in within the Perspective-n-Problem (PnP) in order to acquire relative transformation, that is optimized using other measurements. If the operation was successful, a new constraint in form of a new edge is added between the query and matched frames. No information is lost. All landmarks are stored and the newly found correlation is represented as a new edge. Bundle adjustment or similar techniques do not take place. The performance data from experiments showed that the tracking of points of interest was far more efficient than calculating features in the image again.

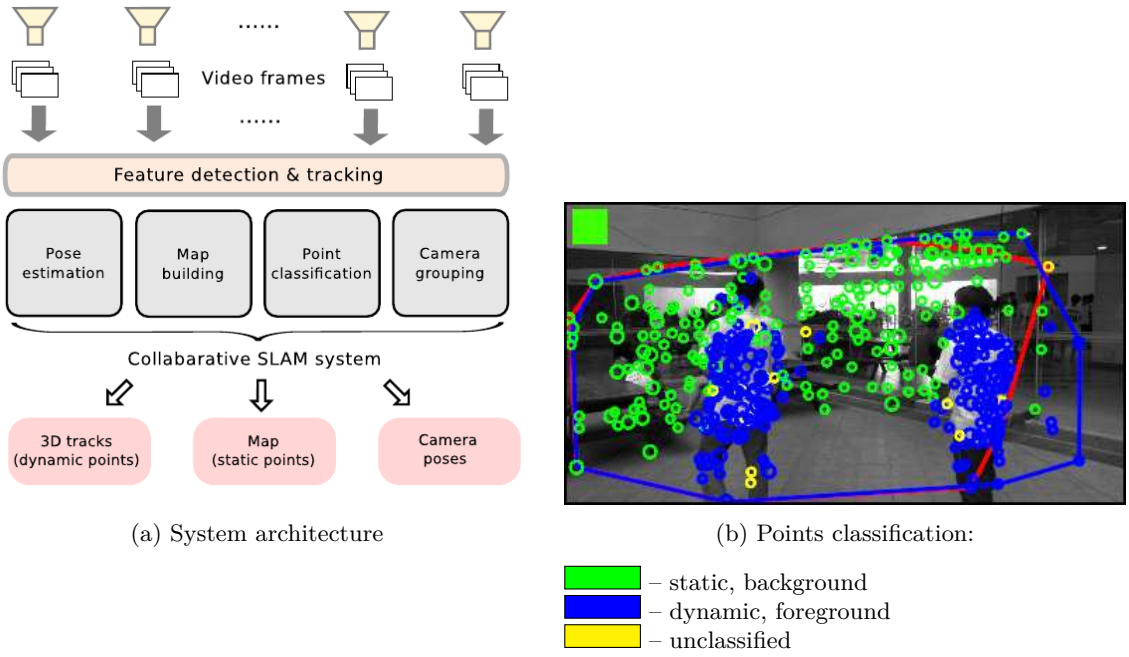
The described approach establishes a working architecture for SLAM dealing with scalability. Suggested improvements include applying Realizing, Reversing and Recovering (RRR) technique to filter out incorrect associations.

CoSLAM: Collaborative Visual SLAM in Dynamic Environments

A centralized vision based multi SLAM in dynamic environments running in real time is described in Zou and Tan [99]. A global map is generated, which includes information about the static background and dynamic foreground as shown in (F3.10b), stored as points. Inter camera pose estimation and mapping are introduced, allowing operations within the proposed abstracts. Cameras are grouped according to view overlap, each group can merge or split with other cameras, when met or separated. A group works collaboratively to map the same view providing robustness. Each map point is assigned a position uncertainty as a covariance matrix that can be iteratively refined. The overall system architecture is shown in (F3.10a).

Robust and Efficient Multi robot 3-D Mapping Merging With Octree-Based Occupancy Grids

A occupancy grid approach based on octree maps applied to multi robot mapping of 3D environments is presented in Jessup et al. [67].



(a) System architecture

(b) Points classification:

Figure 3.10: Overall system architecture and point classification [99].

Open problems

Main open problems and aspects of simultaneous localization and mapping [44] can be summarized as:

- Robust performance
  - Failsafe process and recovery from failure
  - Robustness to hardware failure
  - Metric relocalization
  - Time variant and deformable map
  - Parameter autotuning
- Scalability
  - Map maintenance
  - Robust distributed mapping
  - Learning, forgetting, remembering
  - Resource-constrained units
- Metric representations
  - High-level and expressive representations
  - Optimal representations
  - Adaptive representations
- Semantic representations
  - High complexity and size of concept
  - Ignorance, awareness, and adaptation
  - Reasoning with semantics
- High level understanding
- Resource awareness
- Task-driven inference
- Heterogeneous multi SLAM approaches

For multi robot SLAM systems the open problems evolve around [91]:

- Double counting – Acquiring redundant information
- Line of sight observations – At least on unit observes another, this information could be used to improve the observed units localization estimate
- Heterogeneous vehicles and sensors – Processing and fusing of different type of information
- Communications – Several constraints are usually available, such as limited bandwidth, interference from the environment
- Synchronization – Times of acquired information need to be related to each other, among sensors of each unit and among a group of units

These comprehensive open problems lists makes clear, that the motivations for the collaborative decentralized heterogeneous SLAM approach are currently in demand.

Exploration and rendezvous tasks

The process of exploration, ergo visiting unmapped or unknown places in the environment, has been in focus of robotics for a long time [44]. It can be combined with the process of exploitation, ergo revisiting places in environment. Active SLAM is a variant, where the decision making process has been integrated in such manner, that the robot and data acquisition are controlled with the goal of minimizing the uncertainty of localization and created map. A suitable approach prioritizes both aspects in a balanced manner [97].

The rendezvous aspect applied to multi SLAM deals with multiple units meeting at a specified location. This can be used in decentralized SLAM approaches. At the rendezvous instance the acquired data are shared between the units.

The approach proposed by [75] is combination of exploration and rendezvous task using cost of reaching a location and reward for its characteristics. The cost of reaching the target location is proportional to the distance between the unit and the location, while the reward for reaching the target location is proportional to the estimated gain of relevant information, ergo map at the location.

Assumptions of the method include local sensing information only with a communication range limit. To overcome the latter limitation, a group of hierarchical or role-based robots starting from unknown location, can use it's explorer units to gather data, while constantly sharing information to the main data centre through relay units [66]. The approach from [75] however, deals with the rendezvous problem [89], that derived from the field of game theory and is specified as a search problem. It is defined as meeting at pre-specified time at a rendezvous location in an unknown environment without any communication. The described approach is grid-based, overcomes the range limit of communication and performs minimization of combined exploration and rendezvous time.

Mapping and exploration

For each unit the mapping process is performed in the grid-based approach. From the acquired data Hilditch's algorithm [65] is used to generate a skeleton structure similar to generalized Voronoi graph (GVG) [50]. The main difference between the two methods is the construction, where GVG uses the edges to close the graph. The reason for using Hilditch's algorithm is that it can create a skeleton structure with a limited sensor range that each unit has. The structure of the skeleton shown in (F3.11a) is stored as a graph, where the nodes, representing potential rendezvous points, are then used for exploration. In particular the described approach uses depth-first traversal. Using flood-fill algorithm [72] the adjacent nodes are determined as well as the filled area is calculated. Path planning is performed by an A* search algorithm [63].

Rendezvous

The set of meeting locations is specific for the environment, where each rendezvous place is extended from the conventional point to a finite area. For the problem statement three rendezvous strategies are proposed:

- asymmetric sequential,
- symmetric sequential,
- and exponential.

Each of the rendezvous strategy characterizes a landmark by determining it's distinctiveness measure and ranks it among other landmarks. Three ranking criteria summarized in (T3.2) were proposed.

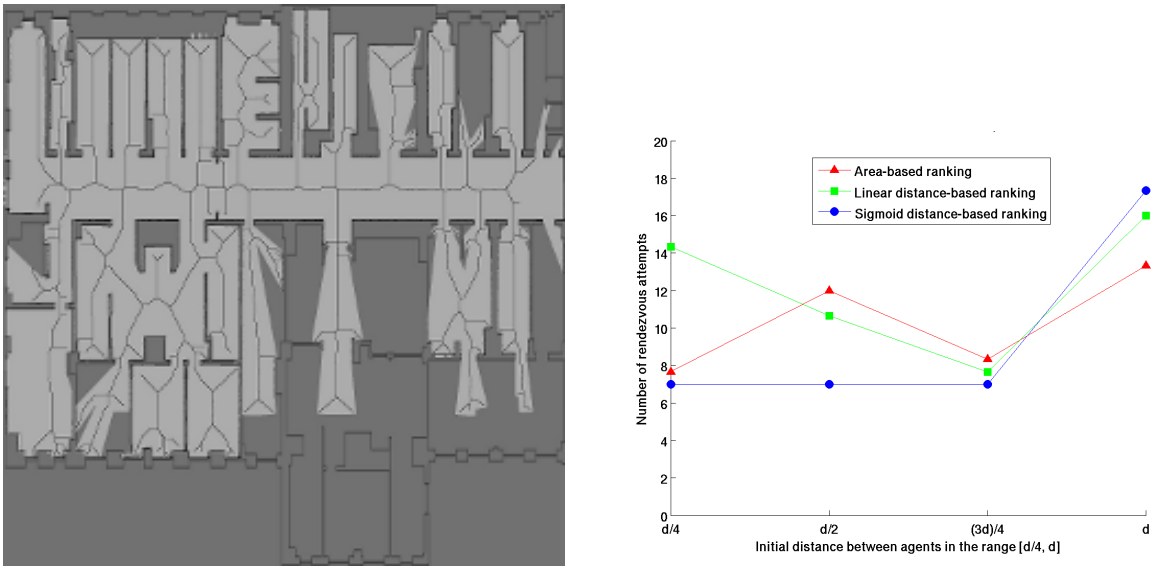
The rendezvous process itself consists of sequential attempts a_i performed at time instances $t(a_i)$ by robots $R \in 1, 2$ at a selected location l_i^R that is determined by each unit based on acquired data. If $l_i^1 = l_i^2$ at time instance $t(a_i)$, then robots 1 and 2 met at the same location, which concludes the rendezvous process, otherwise the robots continue performing previous actions until the next rendezvous time instance $t(a_{i+1})$.

Table 3.2: Ranking criteria and formulations.

Ranking criteria	Formulation
Area	$rank(l_i^R) = area(l_i^R)$
Distance	$rank(l_i^R) = \frac{area(l_i^R)}{distance(l_i^R)}$
Sigmoid distance	$rank(l_i^R) = \frac{area(l_i^R)}{sigmoid(l_i^R)}$

Test results

The provided test results shown in (F3.11b) clearly state that the sigmonoid distance based rendezvous strategy obtains best results, ergo minimal number of rendezvous attempts in minimal time for all studied scenarios.



(a) Generated skeleton structure with nodes representing potential rendezvous points.

(b) Test results for the examined rendezvous strategies.

Figure 3.11: Rendezvous and exploration approach [75].

Visual communication

The visual communication aspect has been addressed mostly in context of robot control [53]. This is similar only in limited way to the anticipated approach of the master thesis, where visual communication would be used in order to help with the rendezvous aspect and visual servoing.

For example, using light source devices like LEDs around the QR marker that will be mounted onto the UGV, one-way communication between the mobile platform and drone could be established. Different operations with light could indicate an action like ”rendezvous mode, waiting for another unit to get into communication range”. Other information could include a ”unit is ready to optimize local map”, ”memory full” or it ”require assistance for next move”. In the last case, when the message comes from the UGV, the UAV could interpret the message and perform an area sweep in order to supply additional map information

to the UGV. The LEDs also provide a possibility of pose estimation using DeMenthon's POSIT algorithm [83], which could be useful in case of using visual servoing for the collaborative SLAM.

The aspect of visual communication in context of the master thesis is an additional idea that can potentially be implemented.

Chapter 4

Proposed work

Experimental setup

Since the main focus of the master thesis is on UAV and UGV collaboration, these heterogeneous robots will be used for the experimental stage of the work. The initial setup will most probably consist of a Turtlebot [85] for the UGV and Parrot drone [90] for the UAV. Additionally the initial setup will most likely hold cameras as main sensors. After evaluation of previous code base and adjustment to new hardware, further sensory can be included.

Before any experiments will be conducted however, simulations providing ideal conditions for testing of the developed algorithm will be conducted on the simulation software framework V-REP [88], otherwise Gazebo [84].

Collaborative, decentralized and heterogeneous SLAM

The proposed within the developed SLAM framework will be defined by the following aspects:

- Previous work of Chebrolu et al. [49] and Camous [46] as starting point for code base
- Use of different types of sensors
- Analysis of possible augmentation of the system
  - Visual servoing as an additional component for active SLAM
  - Acoustic SLAM in form of human-like echolocation
  - Visual communication to support overall operation and especially the rendezvous aspect and VS

Assumptions

The practical implementation part is based on several initial assumptions:

- Collaborative SLAM, ergo active components to perform the SLAM task;
- Heterogeneous robot team, consisting of UGVs and UAVs;
- Decentralized approach assumes exchange of map information between 2 units at a time;
- Experimental hardware might be different from these initial assumptions;
- Start work with a pair of UGV and UAV, then expand to 3 units, and if possible then swarms of both;

- Use of different types of sensors;
- UGV unit is equipped with high computational power hardware and the energy consumptions aspects are of no concern, ergo negligible;
- UAV unit is equipped with low computational power hardware, low memory capacity and programmed to perform with relatively small amount of computations;
- Limits on accumulated data.

These can be modified during the actual work, because of further hardware or decision based constraints.

Scenarios

Several scenarios are to be defined and evaluated in simulation and also preferably during the experimental stage and could be related to conventional environment mapping, search and rescue scenarios as well as dynamic environments. VS could be a special mode for the collaborative SLAM approach.

Map characteristics

Because of availability of diametrically different map representations, it should be considered developing a new map format, possibly using already well established ones, like factor graphs, but also incorporating work described in Morrison et al. [78].

Map quality and characteristics:

- Able to work with an active SLAM
  - Additional abstraction of information for the collaborative aspect
- Performance and quality index
  - Proportional to number of frames in a given area
  - Inversely proportional speed of the unit or optical flow magnitude
  - Proportional to number of features
  - Proportional to angle rotational sweep
- Potential field approach
  - 4 levels with ability to add more
- Areas of interest
  - Determined in a similar way to rendezvous uniqueness in [75]
- Motion planning based on current state of local map

Chapter 5

Implementation

V-REP

V-REP is a robot simulator developed by Coppelia Robotics [88], it includes an integrated development environment with support for several languages, including C/C++, Python, Lua and Octave. It allows to design custom robots from within the application, as well as import of standardized collada [27] models. The functionality can be further extended using one of the interface APIs or internal Lua scripts handling almost all aspects of the simulation and it's entities. It's vast customizability allows for a replacement of Gazebo [84].

Based on gathered experience, V-REP can be characterized by the following advantages and disadvantages:

+ Custom approach to a problem possible;
+ Easy manipulation of simulation scenes;
+ Interactive approach of simulations, also during run-time;
– Not always full, consistent or up to date documentation of function calls;
– User interface is only slowly responsive, often ambiguous;
– Necessity of rebuilding of the ROS external interface/plugins in order to implement custom messages or apply changes;
– Script editing and save routine;
– Debugging;

The current version is 3.4.0 [16] released in April 2017. However due to compatibility issues, the version 3.3.2 [17] is being used. There is a Lua script handling the Velodyne Puck sensor in the newer version however, with the goal to provide a guide how the code has to be upgraded.

Intensive development of installation scripts has been conducted for both, Ubuntu 14.04 [22] and Arch Linux [2]. The latter requires specific packaging using the Arch User Repository (AUR) [3]. Installation scripts for both versions 3.3.2 and 3.4.0 have been developed for both platforms based on the initial installation script for the software. V-REP provides a variety of interface options. The main component relevant to ROS framework is the ROS plugin and external interface. It is used to communicate with the ROS framework through the message mechanism, service calls and through ROS parameter server, which has been included just recently in version 3.4.0. The installation routine largely stayed the same, which allowed for a development of homogeneous shell code structure, thus providing more efficient maintenance. Additional

repositories that are necessary for version 3.4.0 with respect to version 3.3.2, have been added and implemented properly. The scripts allow the user to choose from a local or remote GitHub repository copy based on the value of LOCAL. Another change in the 3.4.0 version is that "vrep_common" and "vrep_plugin" have been dropped. The main interest of developing the installation scripts was the expansion of ROS external interface plugin to allow using custom messages in the simulation, which are used in the previous code base and also the new development.

Detailed description of V-REP

Scripting inside V-REP is performed using the Lua launguage [14]. The language is characterized to be relatively easy to read, understand and use, with several semantic and programmatic aspects similar to Python. Scripts are associated with V-REP objects listed in the hierarchy view. There are different types of scripts in current version of V-REP:

- add-on scripts,
- child scripts,
  - non-threaded child scripts,
  - threaded child scripts,
- callback scripts,
- customization scripts,
- main scripts.

The dominant focus of implemented work was in non-threaded child scripts, since neither real-time performance, as far as it could be achieved by V-REP, nor threaded child scripts specific functions have been required. There is however, a threaded child script used for way points navigation routine, as an example how this type of scripts has to be developed. It's relevant, since there are significant differences between threaded and non-threaded scripts, mostly related to code structure and execution aspects.

The scripts have been developed from a general approach point of view, where the user has control of how the routines are executed. The customization aspects allows for a better implementation on an application basis and more control over performance.

V-REP also allows for an integration with outside scripts, even those not directly supported. Several shell scripts have been developed in order to improve efficiency working with the simulator. The start and stop of a simulation is relatively slow and tedious to repeat, especially in case of debugging. An external unsupported script like "vrep-restart.sh" , can be used to restart the simulation remotely through a sequence of V-REP service calls. For custom time other than the default $3s$, a shell call with an argument can be used $ bash scripts/shell/vrep-restart.sh 1, resulting in waiting for $1s$ between service calls. One should be aware that this is not necessarily always feasible, since the simulation has a system resource dependent latency. A possible improvement would be to include a wait routine, that would start after sleeping time, determining the state of the simulation from the topic */vrep/info*, as it is done in "record-vrep-interrupt.sh" .

Script 5.1: Bash shell alias definition identical to vrep-restart-simulation.sh

```
#!/bin/bash
if [ $# -eq 0 ]; then SLEEP=3; else SLEEP=$1; fi

if (pidof vrep &>/dev/null); then
  rosservice call /vrep/simRosStopSimulation;
  sleep "$SLEEP";
  rosservice call /vrep/simRosStartSimulation;
```

```
else
  echo "V-REP not running";
fi
```

Custom messages are an integral part of the ROS external interface. In case of CoSLAM there is the message format "keyframeMatchInfo.msg" from the "coslam_msgs" package and several message formats: "keyframeGraphMsg.msg" , "keyframeImg.msg" and "keyframeMsg.msg" from the modified "lsd_slam" package, in particular "lsd_slam_viewer" .

Additionally the Velodyne messages from package "velodyne_msgs" are used in the VLP16 script for publishing of raw messages in "velodyne_msgs/VelodynePacket" format. Even though, currently only a dummy message is being published with an empty point cloud in raw data, the main routine creating the message data is properly implemented. The package "velodyne_pointcloud" provides an algorithm for conversion from raw data, that the Velodyne driver extracts from the sensor, to a PointCloud2 [19] format from "sensor_msgs" package. The reverse of that algorithm is not as trivial, since the forward conversion includes several corrections [36].

The developed installation scripts allow for implementation of custom messages statically, through a patch file or archive. The former can be created by modifying the three relevant files "CMakeLists.txt" , "package.xml" and "messages.txt" in the folder "$VREP_ROOT/programming/ros_packages/catkin/src/" followed by using the $ **diff** -BNu tool. The latter can be created by modifying the mentioned files and creating a *7z* archive with relative path. Another option is to use a dynamic patching method using $ sed, as it currently the case.

As of version 3.4.0 there is still no option to start a simulation without the scene rendering active, which requires the user to manually disable it. There is the option to start V-REP in headless mode using $ vrep -h switch however, that is without any GUI. The control over the simulation due to lack of service calls or other interface options is very limited compared to conventional simulation method.

Script 5.2: Standard output excerpt in terminal, where V-REP was started, indicating that ROS plugins have been loaded successfully and that roscore is running.

```
  Plugin 'Ros': loading...
  Plugin 'Ros': load succeeded.
  Plugin 'RosInterface': loading...
  Plugin 'RosInterface': load succeeded.
```

ROS

Several nodes have been developed within the ROS framework. Whenever possible, a more universal approach has been taken in order to minimize code reproduction. Previous work on collaborative SLAM (CoSLAM) has been used as reference. Furthermore already developed code from previous projects was reused wherever applicable. The universal approach enables the user to have more control over the code and to adjust provided framework functionality to each application case.

The main application of the UGV and UAV collaboration is simultaneous localization and mapping using heterogeneous sensors. It is of particular interest, because The initial approach focuses on using a laser range finder, a Velodyne Puck (VLP16), and a stereo vision system. Further development foresees monocular vision instead of stereo vision.

CoSLAM V-REP package structure

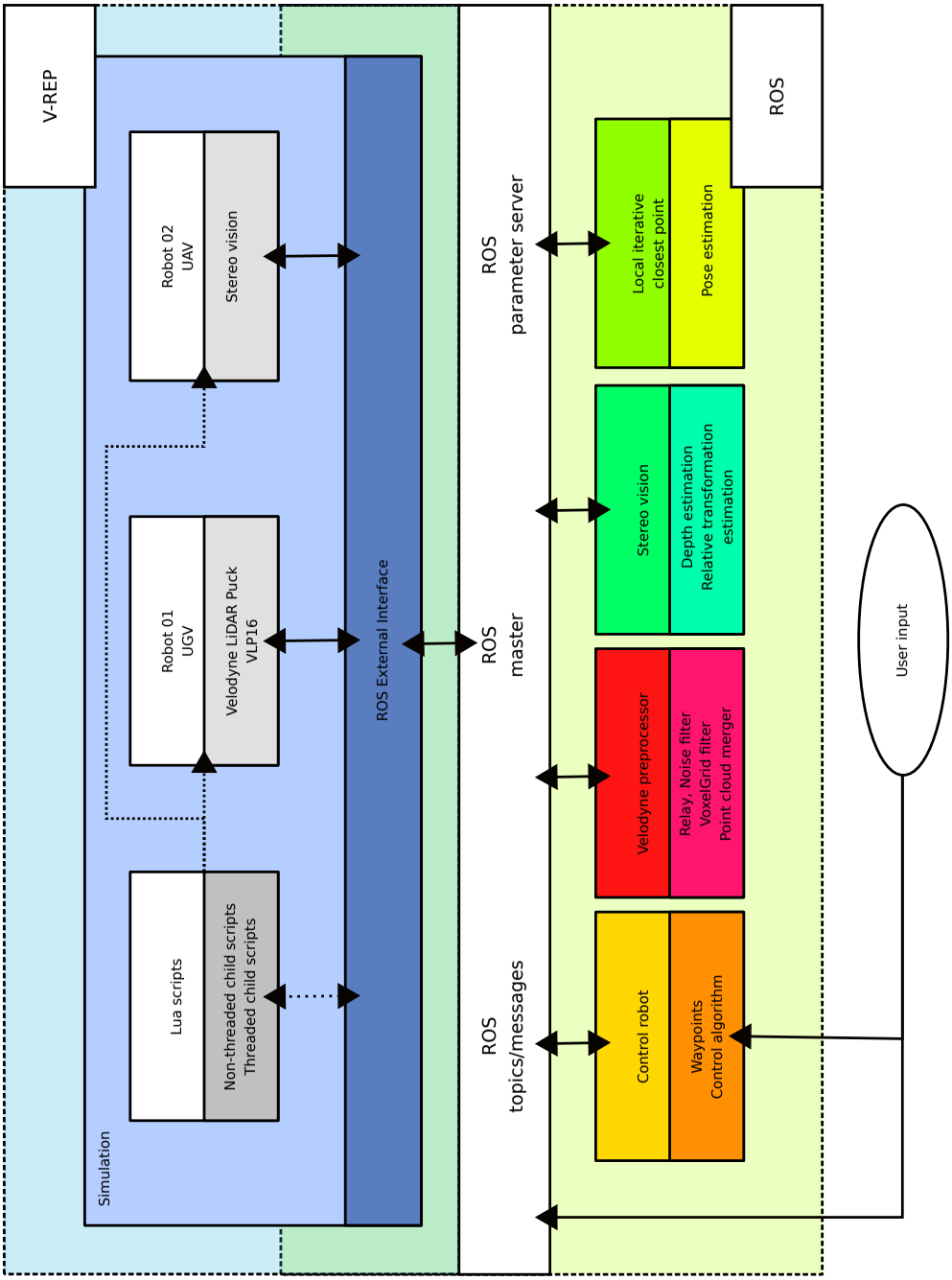


Figure 5.1: CoSLAM – V-REP package structure

The developed framework, illustrated in (F5.1), consists of following ROS nodes:

- Velodyne Puck (VLP16) preprocessor (vlp16),
- stereo vision depth estimator (svdpe),
- local ICP point cloud registration (licpr),
- stereo image preprocessor (sippr),
- robot control (rbctl).

Velodyne preprocessor

V-REP provides models of the Velodyne sensors HDL64ES2 and VLP16, only the latter was used in the performed simulations [**?**][26]. The Velodyne Puck, model name VLP16, is currently the smallest Velodyne sensor available. It utilizes LiDAR technology [28] to map environment with real time performance. Summary of specifications is illustrated in (T5.1).

Table 5.1: Specifications of Velodyne LiDAR PUCK / VLP16

Parameter	Value
Range	$100m$
Accuracy – typical	$\pm 3cm$
Field of view – horizontal/azimuth	360°
Field of view – vertical	$30° \pm 15°$
Angular resolution – vertical	2°
Channels	16
Frequency – data points	$300\ 000 point/s$
Frequency – rotation rate	$5 \div 20Hz$
Laser class	1
Wavelength	$905nm$
Operating voltage range	$9 \div 32V_{DC}$
Power consumption	$8W$
Footprint	$\varnothing 103 \times 72mm$
Weight	$830g$
Network throughput – Ethernet	$100Mbps$
Network protocol	UDP
Environmental protection – IP specification	$IP67$

The V-REP model of VLP16 emulates the sensors behaviour, however there are significant differences. The modelled sensor consists of 4 cameras, each with a field of view of 90°, accounting to a full rotation of 360°. The data is represented in floats, which is one of the reasons "sensor_msgs/PointCloud2" message format is being used for transmission. The raw data provided by the actual hardware controlled by the velodyne_driver [23] is represented as unsigned integers in 8-bit resolution. According to specifications [**?**][26] it holds information about distance, calibrated reflectiveness, rotation angles and synchronized time stamps. The driver uses two type of message formats: velodyne_msgs/VelodynePacket [24] and velodyne_msgs/VelodyneScan [25]. The difference between them is the standard message header with a frame identificator and time stamp. However, only VelodynePacket publishing was successfully implemented in the Lua script, with VelodyneScan a problem most probably related to Lua syntax and table inclusion prevented implementation.

Algorithm 2 test.

```
1: procedure PREPROCESSOR(pointcloud)
2:     Subscribe to topic
3:     Convert ROS message to PCL PointXYZ format
4:     Update buffers
5:     if filterNoise then
6:         Filter noise
7:     Detect partial cloud quadrant
8:     Merge partial point clouds
9:     if filterVoxelGrid then
10:        Filter using VoxelGrid
11:    Convert PCL PointXYZ format to ROS message
12:    Publish in sensor_msgs/PointCloud2 format
```

It is important to note that the developed preprocessor ROS node can be tuned on-line, meaning that settings can be changed on the fly and accustom to specific scenarios without restarting the node or recompiling. The general algorithm of the VLP16 node is represented in (A1).

The conversions to PCL native PointXYZ format make handling of the point cloud much easier. Buffers are updated if the same quadrant message is detected. Based on *filterNoise* and *filterVoxelGrid* switch states, the appropriate filtering process takes place. In case all partial clouds are present, the point cloud is assembled and published.

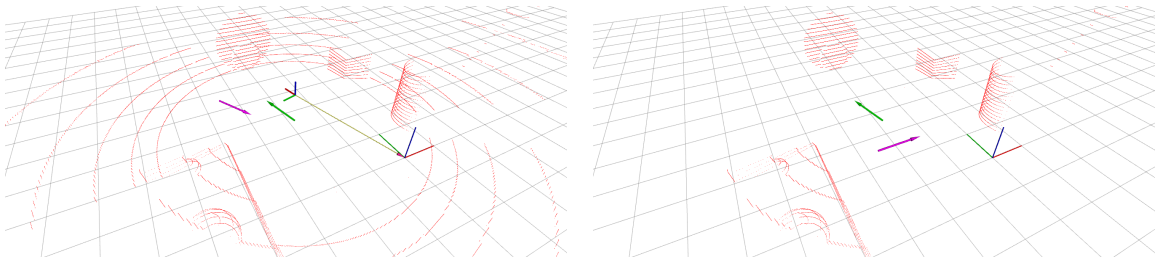
The developed C++ code provides fast computations, there is virtually no significant latency between unfiltered and filtered published messages. During the development process however, because of the initial approach of filtering data out, ergo deleting points from the point cloud that did not meet criteria described by (E5.1), the filtered point cloud was published with approximately a $2s$ delay. This was due too many CPU extensive pointcloud.points.erase(index) calls, since the reduction of number of points was around 80% from 60 000 to 12 000 points. Current implementation uses inverted criteria (E5.2) and (E5.3). An optimal approach might use a detection routine, where based on the number of points meeting both criteria, the decision would be made if a new point cloud should be assembled point by point when lot of noise is present, or in case of only a relative small number of points would have to be deleted, to erase points from the original point cloud. This is a very significant aspect, since the amount of points that VLP16 delivers can reach up to 300 000$points/s$. The current implementation assumes that major part of the point cloud is noise, therefore the point cloud gets assembled. It is assumed however, that the current implementation is also suitable for working with real hardware even during maximum performance.

$$i_n < filterNoiseThreshold : i = x, y, z\ coordinate; n = 1, \ldots, N \tag{5.1}$$

$$i_n > filterNoiseThreshold : i = x, y, z\ coordinate; n = 1, \ldots, N \tag{5.2}$$

$$i_n < filterNoiseThresholdMax : i = x, y, z\ coordinate; n = 1, \ldots, N \tag{5.3}$$

The maximum distance threshold is related to the fact, that based on height of the mounted Velodyne sensor, only a limited area can be covered, which can be reduced to a simple distance quantifier *filterNoiseThresholdMax*. For testing purposes the height of $1m$ was used for the VLP16 installed on the UGV, which resulted in approximately $16m$ for the *filterNoiseThresholdMax* parameter. This parameter is fixed, however it is influenced by the noise threshold parameter, since it can completely eliminate the detected floor if set to a value $threshold = 0.02m$ as illustrated in (F5.2).



(a) Noise threshold set at $threshold = 0.001m$. (b) Noise threshold set at $threshold = 0.02m$.

Figure 5.2: Effects of */vlp16/noise/threshold* variable on the maximum distance.

Stereo vision depth estimator

For the heterogeneous sensor aspect of the work, a stereo vision node was developed in Python. The main functions are depth estimation from a calibrated and known stereo vision system, as well as relative pose estimation of the main camera based on consecutive frames.

During development, simulation scenes had only an UGV unit with both sensors, namely the Velodyne Puck and the stereo vision system. The Lua script for controlling the robot is well developed and less demanding on the CPU time, while the UAV is far more complex, would require more development time and maintenance, which was not part of the main objective.

Standard V-REP vision sensor have been used with parameters described in (T5.2). For square pixels with no skew $k_u = k_v = 1$, $s = 0$, the focal length can be computed from the field of view and resolution with accordance to (E5.4). The horizontal field of view is the dominant one in V-REP, meaning it determines vertical field of view (E5.5) based on resolution and in accordance with (E5.4) [37][**?**]. The focal length is expressed in pixels [64, p. 175][7]. The resulting intrinsic camera parameters are identical for both cameras and are expressed in (E5.6).

$$focalLength = \frac{width}{2} tan\left(\frac{fovh}{2}\right)^{-1} \tag{5.4}$$

$$fovv = 2arctan\left(\frac{2focalLength}{width}\right) \tag{5.5}$$

Table 5.2: Parameters of vision sensors used as camera in V-REP

Parameter	Value
Width vw	$640px$
Height vh	$480px$
Field of view $fovh$ – horizontal	60°
Field of view $fovv$ – vertical	120°
Focal length $focalLength$	$554.26px$
Pixel ratio $ku = kv$	1

$$K_1 = K_2 = \begin{bmatrix} f & s & cu \\ s & f & cv \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 554.26 & 0 & 240 \\ 0 & 554.26 & 320 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

Intrinsic camera parameters describe the mapping of a 3D point projected onto an image plane of a camera expressed in pixels. Depending the camera projection model used, these parameters usually include radial and tangential distortions [4][5]. Extrinsic camera parameters describe the position and orientation

in the absolute frame. In the stereo vision case however, usually one camera is used as reference for the transformation to the other camera. Usually the main camera or reference frame is the left camera (E5.7), while the secondary or referred frame is the right camera (E5.8).

$$C_1 = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

$$C_2 = \begin{bmatrix} 1 & 0 & 0 & 0.2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

Table 5.3: Intrinsic camera parameters

Parameter	Variable
Skew	s
Focal length	f
Optical point x	cu
Optical point y	cv

Algorithm 3 Stereo vision

procedure STEREO VISION($inputImage01_k, inputImage01_{k+frameSkip}, inputImage02_{k+frameSkip}$)
  Subscribe to topics from both cameras
  if Images from both cameras in buffer **then**
    procedure PERFORM DEPTH ESTIMATION($inputImage01_{k+frameSkip}, inputImage02_{k+frameSkip}$)
      Find feature points
      Match and filter key points
      Convert matched point to normalized coordinates
      Estimate depth
      Remove invalid values
      Assemble and publish point cloud
  else
    Update the appropriate buffer
  if Two consecutive frames from main camera in buffer **then**
    procedure PERFORM RELATIVE POSE TRANSFORMATION($inputImage01_k, inputImage01_{k+frameSkip}$)
      Find or copy already retrieved feature points
      Match and filter key points
      Find essential matrix
      Recover relative pose between two consecutive frames
      Update absolute pose estimate and publish
  else
    Update the appropriate buffer
  Update buffers

Similarly to the VLP16 preprocessor node, the stereo vision ROS node can be tuned on-line through ROS parameters server. The general algorithm of the stereo vision node is represented in (A3).

The developed node has little to no overhead in Python, since all the OpenCV calls have bindings to C/C++. Python is a general language, more precisely a dynamic high-level general-purpose language [31] allowing rapid development through scripts [33]. The benefit of writing ROS nodes in Python include that

changes to the script do not require recompilation and debugging is usually easier. Also the developer is forced to write in a specific format using indentations, making the code more readable. And finally, ROS supports Python natively through *rospy*.

There are various settings for both main routines of depth and relative pose estimation. One aspect that influences both routines, is the feature detection method presented in node excerpt (R5.3).

Script 5.3: Stereo vision node excerpt illustrating various options for feature detection.

```
## Set feature detection method
#
# @warning Not compatible with OpenCV3
# Prefix possible values: "","Grid","Pyramid"
# Suffix possible values: "FAST","GFTT","HARRIS","MSER","ORB","SIFT","STAR","SURF"
global selectFeatureDetectionMethod
_selectFeatureDetectionMethod = rospy.get_param('/stereo_vision/ featuredetectionmethod', selectFeatureDetectionMethod)
# Check if feature detection method was changed
if selectFeatureDetectionMethod != _selectFeatureDetectionMethod:
  # Clear stereo vision buffer data
  self.clearStereoVisionBuffer()
  # Update feature detection method
  selectFeatureDetectionMethod = _selectFeatureDetectionMethod
  return -1

## Create feature detector
if selectFeatureDetectionMethod == 1:
  featureDetectionMethod = "AKAZE"
  featureDetector = cv2.AKAZE_create()
elif selectFeatureDetectionMethod == 2:
  featureDetectionMethod = "SIFT"
  featureDetector = cv2.xfeatures2d.SIFT_create()
elif selectFeatureDetectionMethod == 3:
  featureDetectionMethod = "SURF"
  featureDetector = cv2.xfeatures2d.SURF_create()
else: # Default == 0
  featureDetectionMethod = "ORB"
  featureDetector = cv2.ORB_create()
```

$$x_{1p} = \begin{bmatrix} x_{1px} \\ x_{1py} \\ 1 \end{bmatrix} = K_1 \begin{bmatrix} x_{1nx} \\ x_{1ny} \\ 1 \end{bmatrix} \tag{5.9}$$

The relation between pixel coordinates of point x_{1p} on main camera image plane and normalized coordinates x_{1n} is represented in (E5.9), where K_1 is defined as in (E5.6).

The epipolar constraint implies that a point x_{1n} in main image plane has a correspondence on the epipolar line in the secondary image plane, defined as Ex_{1n}, according to (E5.11), which can be also expressed using pixel coordinates (E5.10).

$$x_{2p}^T K^{-T} E K^{-1} x_{1p} = 0 \tag{5.10}$$

$$x_{2n}^T E x_{1n} = 0 \tag{5.11}$$

Alternatively, if the stereo vision system is not calibrated, the fundamental matrix F can be directly

used on pixel coordinates as per equation (E5.12).

$$x_{2p}^{T} F x_{1p} = 0 \tag{5.12}$$

In order to reduce the computational load during matching, stereo vision rectification can take place, as described in detail in *"Image rectification"*. If a physical stereo vision system cannot be described only by translation between the camera frames, ergo if there is a rotational component to transformation, one or both image planes can be rotated.

In the simulation scenarios, a calibrated, ergo known and with no distortions, and rectified stereo vision system has been used. The horizontal translation between the left and right camera planes is $t_x = 0.2m$. This is still a reasonable and feasible distance for a hardware system, even though it is not compact.

For depth estimation a triangulation method available in OpenCV could be used, if projection matrices are provided [11]. However instead the depth estimation is performed based upon (E5.13), where disparity $x_{1nx} - x_{2nx}$, which is inversely proportional to depth of point of interest, and translation between cameras t_x are being used. Estimated point are expressed in the left/main camera frame C_1, in the code referred as camera01.

$$z = \frac{t_x}{x_{1nx} - x_{2nx}} \tag{5.13}$$

The depth estimation is conducted from both cameras according to (A3), where features are detected, their correspondences in the other image matched and depth estimated according to relation between horizontal translation between camera image planes and disparity as stated in (E5.13).

Additionally to depth estimation, the stereo vision node also performs relative pose estimation from consecutive frames from the left/main camera as stated in (A3). Here the parameter frameSkip is being used to determine how many frames between transformation estimation have to be skipped. It is highly inefficient and in most cases inaccurate to perform the operation directly frame after frame, since usually the translations are minimal and might be more susceptible to noise.

The initial procedure is identical to depth estimation. In order to improve computational efficiency the same feature points are reused, ergo feature extraction does not take place twice for the same frame. To recover the pose an OpenCV call recoverPose() is used, which basically uses a 5 point algorithm for pose estimation [81]. Points with negative depth, ergo behind the image plane should be rejected, which should result in two possible solutions, out of which, the one resulting in a closer validation of the epipolar constraint (E5.11), should get chosen. To compare the OpenCV implementation the described procedure could be also implemented as an alternative. The decomposition of rotation and translation components can be accomplished by using singular value decomposition [34].

Current implementation of the relative pose estimation diverges relatively fast from the initial pose at world origin $O = [x, y, \theta]^T = [0, 0, 0]^T$. There is no mechanism that resets the initial pose estimate or updates pose estimation parameters. A variation of a Kalman filter might be a possible tool to update the estimation parameters, where the prediction model would relay on current relative pose estimation routine, however a measurement would be required as well. In the current implementation the local iterative closest point node aims at registering the stereo vision estimated point cloud in the VLP16 point cloud. This could serve as measurement for the Kalman filter.

Furthermore, for future work the essential matrix should be defined by manually, since the stereo vision system is known, however at the same time providing the current implementation to coexist, so that an unknown system could be also used with presented algorithm by user requests.

Local iterative closest point

The local ICP node performs a cloud registration of point cloud received from stereo vision depth estimation, and then publishes calculated transformation as main camera pose.

The node has been developed using Python bindings supported by Straw Lab [30][21], which has frequently used functions implemented.

Algorithm 4 Local iterative closest point

```
1: procedure LOCAL ICP(inputPointCloud01, inputPointCloud02)
2:     Subscribe to topics for source and target point clouds
3:     if Both point clouds in buffer then
4:         procedure PERFORM POSE ESTIMATION(inputPointCloud01, inputPointCloud02)
5:             Use one of the available ICP methods
6:     elseUpdate the appropriate buffer
7:     Publish estimated pose
```

The algorithm (A4) describes the routine of the developed node. Current implementation however, does not provide satisfactory results, most likely, because of implementation issues. Suggested course of action is to shift to a C++ implementation of the same algorithm.

Robot control

Robot control node is a very basic program to control the units inside the simulation. Currently direct user input sequences are being implemented, however the main goal is to move the robots through user specified way points. Current completed implementation uses predefined sequences of way points. Future improvement could include user input of an array of way points, each with three coordinates composing a 2D pose.

The Lua script for UGV-01 "ugv-05–robot01.lua" has an analogue control scheme with the specifications according to (T5.4).

Table 5.4: Lua script control parameters for UGV-01.

Wheels	
Wheel radius r	0.975 m [1]
Wheel base L l	0.1655 m $l = 2L$
Initial velocities	
Linear v	0 m/s
Angular w	0 m/s
Tolerance	
Position t_p	0.3 m
Orientation t_o	$^{15}/_{360}$ rad
Control parameters	
Position control gain k_p	8.0
Orientation control gain k_o	16.0

Parameters of the UGV model[1] are taken directly from the data sheet of the manufacturer [13].

The control laws are developed based on the parameters in (T5.4) and traditional linear and angular velocity control for a robot of type (2,0).

The 3 errors in position e_p, heading towards the way point e_h and final orientation e_o are defined in (E5.14), (E5.15) and (E5.16).

$$e_p = \sqrt{{waypoint_x}^2 + {waypoint_y}^2} \tag{5.14}$$

[1] Datasheet of the Pioneer 3DX specifies a diameter of $\varnothing 195mm$, ergo $r = 0.0975\ mm$

$$e_h = \mod(\operatorname{arc\,tg}(waypoint_y, waypoint_x) + \pi, 2\pi) - \pi \tag{5.15}$$

$$e_o = waypoint_\theta - robot_\theta \tag{5.16}$$

The control inputs are then defined in

$$v = k_p \cdot (waypoint_y - tolerance_p) \tag{5.17}$$

$$\omega = k_o \cdot e_h \tag{5.18}$$

In the current implementation the final orientation error e_o is ignored, ergo the robot does not go through the way point with the orientation of the way point, but instead with the resulting orientation θ that satisfies the position control.

There are 2 modes of control: normal and with increased gains. It has been observed that the UGV performs poorly near the target way point, therefore an increase for the control gains k_p and k_o has been established based on experimentation with, respectively, an increase for both gains described in (E5.19) and (E5.20).

$$v = 18k_p \cdot (waypoint_y - tolerance_p) \tag{5.19}$$

$$\omega = 40k_o \cdot e_h \tag{5.20}$$

Finally the wheel velocities are set with respect to provided geometric description (E5.21) and (E5.22).

$$v_l = \frac{v}{r} - \omega \cdot \frac{L}{r} \tag{5.21}$$

$$v_r = \frac{v}{r} + \omega \cdot \frac{L}{r} \tag{5.22}$$

Stereo image preprocessor

In order to validate achieved results from *"Stereo vision depth estimator"*, the package stereo_image_proc was used. It provides basic image rectification and disparity processing capability.

A very simple node was developed in order to provide appropriate data for the *stereo_image_proc* package. The data consists of image sequences from the stereo vision system and camera information using the CameraInfo message format [18]. However due to problems with implementation of raw transport protocol and remapping of compressed stream of the stereo vision images in the Python node, it was not used for comparison. Further detail on the camera information and the process of image and distortion rectification is available on the ROS wiki [8].

CoSLAM V-REP

Developed package

The structure of the CoSLAM V-REP package is as illustrated in (R5.4). For the most part, a strict, yet universal approach was undertaken in order to guarantee compatibility with anticipated future developments and modifications. The presented package is structured using the ROS package guidelines as reference and based on gathered experience.

Script 5.4: CoSLAM V-REP package directory structure.

```
coslam_vrep
 binary
 configuration
 documentation
 graphics
 headers
 include
      coslam_vrep
 launch
 logs
 models
 scenes
 scripts
      lua
      python
      shell
 sessions
 source
 textures
```

Several sessions have been recorded for the ability to work without the simulator. The ROS bag files holding the content are located in sessions directory.

Scenes

During the development of the CoSLAM V-REP package several scenes for V-REP have been designed. For the most part they represent the progress during the development and can act as milestones. A scene with both UGV and UAV has been created, however at current development status, only the UGV is fully controllable. The main reason is that the UGV can easily be controlled from both within V-REP using Lua scripts, and from outside using robot control node. The UAV requires a more sophisticated approach, which at current development stage is of lower priority. Finally, UGV can hold both, laser and vision sensors just as well, and thus emulated the behaviour of the drone.

Script 5.5: Scenes file and directory structure.

```
scenes
 latest.ttt -> ugv-05.ttt
 ugv-01.png
 ugv-01--robot01.lua
 ugv-01.ttt
 ugv-01.txt
 ugv-01.txt.170614.084347
 ugv-01--vlp16.lua
[...]
 ugv-05.png
 ugv-05--robot01.lua
 ugv-05--robot01.pose2d-control.lua
 ugv-05--scripts--threaded-move-waypoint.lua
 ugv-05.ttt
 ugv-05.txt
 ugv-05--vlp16.lua
 ugv-06.ttt
 ugv-uav-01.png
```

```
 ugv-uav-01--robot01.lua
 ugv-uav-01--robot02.lua
 ugv-uav-01--scripts.lua
 ugv-uav-01.ttt
 ugv-uav-01.txt
 ugv-uav-01--vlp16.lua

0 directories, 39 files
```

Each scene has been provided with a short description file, extracted Lua scripts from the scene for fast access and comparison, as well as a screenshot. The developer documentation in Doxygen can take full benefit of this structure, making further contributions easier and more efficient.

Developed scripts and tools

In order to overcome shortcomings of V-REP and the overall structure of the developed framework, several scripts and tools have been created. For all the files in launch directory holding different ROS launch configuration, some of them modular, a starter script has been created, that allows the user to select and start a specific session. In case of a crash or after completion of use, another session can be started. The contents of the script are in (R5.6).

Script 5.6: V-REP starter script in Bash.

```
#!/bin/bash
source ~/.profile;
export VREP_ROOT="/opt/vrep";
ros-jade || exit 1;

export CV="$HOME/.ros/workspace/src/coslam_vrep"
cd "$CV/launch/";
echo -e "[Select launch file for V-REP]\n[Working directory: $PWD]";

select LAUNCHFILE in *.launch; do
  roslaunch "$PWD/${LAUNCHFILE}";
  echo '[Select launch file for V-REP]';
done
```

A very helpful tool is the "record-vrep.sh", which allows for quasi repeatable recording sessions, since the point of starting the simulation in V-REP cannot be determined with exact precision. In collaboration with "record-vrep-interrupt.sh" it makes a reliable tool for session recordings, with extensive information available without delay in a text file and repetitive tasks automated.

Script 5.7: Bash script for recording a session in V-REP.

```
#!/bin/bash
# Settings
## Latest symlink filename
LATEST="latest-vrep.bag";
## Restart V-REP switch
RESTARTVREP=1;

vrep-restart-simulation() {
  if [[ "${ROS_MASTER_URI-stest}" == "stest" ]]; then echo "ROS environment not set ▼9
    9▲ "; exit 1; fi
  if [ $# -eq 0 ]; then SLEEP=3; else SLEEP=$1; fi
```

```
  rosservice call /vrep/simRosStopSimulation;
  sleep $SLEEP;
  rosservice call /vrep/simRosStartSimulation;
}

CDATE="$(date '+%d%m%y')";
ICDATE="$(date '+%y%m%d')";

if [ $# -lt 2 ]; then
echo -e "Usage:\t${0} FILENAME TOPIC1 [TOPIC2] [...]";
cat << EOF
# Syntax elements
<robot> - vehicle, robot, platform
<location> - place, area
<session> - session run ID
<run> - variations of session
<date> - preferably in "%Y%m%d" format for easier sorting, string date will be ▼27
    27▲ automatically overridden to preferred format

# Additional syntax
|delimiters| - whitespace discouraged, because of escape characters in shell, ▼30
    30▲ preferably single hyphen for linking elements and double hyphen for linking ▼30
    30▲  element sub-information

# Examples
Example command naming convention:
rosbag play <location>-<date>-<robot>-<session>-<run>.bag

Example 1:
rosbag play ecn-20170214-turtlebot-loop-forward.bag

Example 2:
rosbag play ecn--parkinglot-20170214-turtlebot--a-loop--01-forward--a.bag

Format of example 2 can be translated into format from example 1 using sed:
\$ echo rosbag play ecn--parkinglot-20170214-turtlebot--a-loop--01-forward--a.bag | ▼43
    43▲  sed 's/\(--[^-]*-\)/-/g;s/\(--[^-]*\.\)/./' rosbag play ecn-20170214- ▼43
    43▲ turtlebot-loop-forward.bag
EOF

else
  # Set first argument to filename
  FILE="${1/date/${ICDATE}}";
  shift;

  # Check filename extension
  if [[ "${FILE,,}" != *"bag" ]]; then FILE="${FILE%.}.bag"; fi

  # Set topics from remaining arguments
  TOPICS="${@}";

  if [ -f "$FILE" ]; then INPUTFLAG=1;
  while [ $INPUTFLAG -eq 1 ]; do read -n 1 -p "Overwrite? (yes/No/modify)" KEYINPUT ▼58
    58▲ ;
```

```
  case "$KEYINPUT" in
    y|Y)
      INPUTFLAG=0;
      continue;
    ;;
    n|N)
      echo "Exiting";
      exit 1;
    ;;
    m)
      KEYINPUT="$FILE";
      while [[ "$KEYINPUT" == "$FILE" ]] || [[ -z "$KEYINPUT" ]]; do
        echo; read -p "Provide new filename for $FILE:" KEYINPUT;
      done
      FILE="${KEYINPUT/date/${ICDATE}}";

      # Check filename extension
      if [[ "${FILE,,}" != *"bag" ]]; then FILE="${FILE%.}.bag"; fi
      INPUTFLAG=0;
    ;;
    *)
      echo "Input not recognized";
  esac
  unset KEYINPUT;
  done; fi

  echo rosbag record -O "$FILE" "${TOPICS[@]}";
  read -n 1 -p "Press key to restart V-REP" KEYINPUT;
  { if [ $RESTARTVREP -eq 1 ]; then vrep-restart-simulation $SLEEP; fi; } \
  && rosbag record -O "$FILE" ${TOPICS[@]};

  echo "# ${0} -- $CDATE -- $(hostname --long)@$(hostname -I) -- $FILE" > "${FILE/.bag/.txt}";
  rosbag info "$FILE" >> "${FILE/.bag/.txt}";

  ## Update symlink
  if [ -f "$FILE" ]; then ln -sfv "$FILE" "$LATEST"; fi
fi
```

Guidelines

During development several conventions and guidelines have been designed in order to improve overall development efficiency. One of the more important conventions is the SquashFS [20] dataset archive creating convention in (R5.8).

Script 5.8: Compressing ROS bag datasets into SquashFS archives.

```
# Compressing ROS bag datasets into SquashFS

# General information
SquashFS is a read-only filesystem that enables compression of not only files, but
    also permissions, inodes and attributes. It provides extremely efficient
    compression with large number of small size files relative to filesystem
```

```
    4▲ block. Furthermore the different compression algorithms allow for extended  ▼4
    4▲ accessibility.
5
# Important notes
*It is possible to append files to an already created archive and to overwrite  ▼7
    7▲ existing archive with the -noappend option.
*In order to be able to mount the archive, compression algorithms gzip, which is  ▼8
    8▲ the default one, lz4 or xz need to be used.

10 # Syntax elements
<archive> - archive file holding the SquashFS according to datasets-naming-  ▼11
    11▲ convention.txt
<comp> - compression algorithm
<files> - files to be included into the filesystem archive
<files-extract> - files to be extracted from the filesystem archive
15 <mnt> - mount destination
<username> - User name stored in ${USER}

# Examples
Example compression with default algorithm:
20 mksquashfs <files> <archive>

Example compression with specified algorithm
mksquashfs <files> <archive> -comp <comp>

25 Example 1:
mksquashfs ecn-icars-A-2017-02-09-*.bag ecn-icars-A-2017-02-09.sfs

Example mounting of created filesystem:
# mount -t squashfs <archive> <mnt>
30
Example 1:
# mount -t squashfs ecn-icars-A-2017-02-09.sfs /mnt

Example mounting with udevil to a /media/<username>/<archive>/ directory
35 $ udevil mount <archive>

Example 1:
$ udevil mount ecn-icars-A-2017-02-09.sfs

40 Example unmounting with from /media/<username>/<archive>/ directory
# umount /media/<username>/<archive>

Example 1:
$ udevil umount /media/ecn-icars-A-2017-02-09.sfs/
45
Example unmounting with udevil of <archive> from /media/<username>/<archive>/  ▼46
    46▲ directory
$ udevil umount <archive>

Example 1:
50 $ udevil umount /media/ecn-icars-A-2017-02-09.sfs/

Example 2:
```

```
$ udevil umount ecn-icars-A-2017-02-09.sfs

Example listing of archive contents:
$ unsquashfs -l <archive>

Example 1:
$ unsquashfs -l ecn-icars-A-2017-02-09.sfs

Example extraction of all archive contents:
$ unsquashfs <archive>

Example 1:
$ unsquashfs ecn-icars-A-2017-02-09.sfs

Example extraction of selected archive contents:
$ unsquashfs <archive> <files-extract>

Example 1:
$ unsquashfs ecn-icars-A-2017-02-09.sfs ecn-icars-A-2017-02-09-14-37-36.bag
```

Several other files describe the experience with previous framework, how to deal with encountered problems, what steps are necessary to patch the code, how to implement new messages in V-REP, etc.

A great effort has been undertaken to make the developed framework compatible with not just Ubuntu 14.04, but also Arch Linux. Packages for Arch Linux and installation scripts for Ubuntu have been developed.

Virtual machine drive

Due to massive problems with installation and setup of the previous code base, mostly related to an ambiguous configuration for non free libraries for OpenCV in "CMakeLists.txt" , it has been decided to use a virtual machine drive using QEMU [15]. QEMU is an open source emulator and virtualizer with performance enhancing features. It can be found in repositories of most GNU/Linux distributions.

This step was taken in order to secure a working environment, so that future development can be continued without any delays.

Currently a number of Bash scripts is being developed in order to make the use of the virtual environment easier. One example is a simple start of previously prepared virtual drive file

Script 5.9: Run virtual machine using QEMU for *qemu-ubuntu14x64.raw* virtual drive file.

```
#!/bin/bash
FILENAME="$(basename ${0%.sh})";
DIR="$(dirname ${0})";
VDISK="$FILENAME.raw";
cd "$DIR"
qemu-system-x86_64 -enable-kvm -m 4096 -machine type=pc,accel=kvm -smp 4 -cpu host -vga qxl -name "${FILENAME^}" -net nic -net user,smb=/tmp/smb -drive format=raw,file="$VDISK" "$@"
```

Comparison

The previously developed collaborative framework *CoSLAM* by Chebrolu et al. [49] and Camous [46], has been used in many ways as a reference with the final goal of merging it with presented development.

However due to initial difficulties an expansion of provided work has been developed instead, with the current status being almost ready for the merge.

There are significant differences between *CoSLAM* and *CoSLAM - V-REP*. Currently the latter can be seen as an expansion of the former package, however it can function as a standalone.

Since V-REP is the main tool in this package, all the advantages and disadvantages elaborated in detail in *"Detailed description of V-REP"*, are also applicable to the package. Additionally, the initial goals and assumptions of addressing the heterogeneous robot type and sensor aspect, the collaborative aspect and decentralized aspect have a great influence on the package characteristics.

Shortly, the advantages over previous framework version can be listed as:

- General
  - Far more options for the user to interact with the software on-line and off-line
  - Nodes can be tuned on-line to a large extent, filters for the preprocessor for example, which improves overall usability
  - Various tools make further expansions, modifications and creation of additional content, like ROS bag recorded sessions, easy and productive
  - An extensive effort to document the source code, the content and overall approach in the project using Doxygen documentation
  - Programming on a more modular and dynamic basis, without the use of absolute paths or otherwise limiting methods
  - Several nodes written in Python for a more cross platform approach, where the algorithms can be evaluated outside of the developed framework and then improvement can be fed back into the main project
  - No statically enforced use of non-free modules, like SIFT and SURF features, allowing for more compatibility even with systems that do not have the non-free components of OpenCV
- Heterogeneous aspect
  - Simulation of various sensors compatible with currently developed framework, ergo laser range based and vision based, that are available in V-REP
  - Easily expandable to different robot types
- Decentralized aspect
  - On-line approach in CoSLAM V-REP versus off-line in CoSLAM
  - Current framework aims at providing pseudo real time performance, even though Python is used extensively in several nodes
  - Compressed image transport is used to simulate artefacts and reduce bandwidth required for communication
- Collaborative aspect
  - Preparation for implementation of algorithm based on rendezvous [75]

There are also disadvantages:

- V-REP specific simulation issues, like slow response UI and relatively low scripting capabilities for simulation control

- Simulation Lua scripts require reprogramming at this stage, V-REP offers creation of trivial GUIs for further user feedback and control and also script parameters can be exported and changed from the general V-REP UI level.
- Additional development is necessary for implementation of new sensors and/or ROS messages
- Additional maintenance of the ROS external interface is required, especially after system updates

Chapter 6

Results

Preliminary results

Preliminary results from the developed framework are quite limited at the current stage.

The simulation framework has been intensively developed and is robust, allowing to perform various types of scenarios, currently with one robot type, that can however be used as a replacement for other unit types. The Lua scripts allow for extensive custom approach, depending on the scenario. An in-depth analysis through different settings is possible, like visualization of the emulated VLP16 sensor.

A very good performance has been achieved with the preprocessor node for Velodyne Puck VLP16, described in more detail in *"Velodyne preprocessor"*. There is virtually no delay in processing of approximately 60 000 points and reduction of the unfiltered point cloud by approximately 80%. These results can be verified by running a test session and starting the "vlp16" node with verbose flag set to an integer value above 0. However one should note, that the initial approach of removing points from the input point cloud resulted in significant delays of approximately $2s$.

The visual representation of achieved results is illustrated in (F6.2) and (F6.3) for the scene (F6.1). It is clearly visible that the noise introduced from the V-REP side, most probably by the ROS external interface plugin, is significant. In fact, the majority of points from a unfiltered full scan message is noise located along the origin axes. That is the reason point cloud reduction is at the level of approximately 80%.

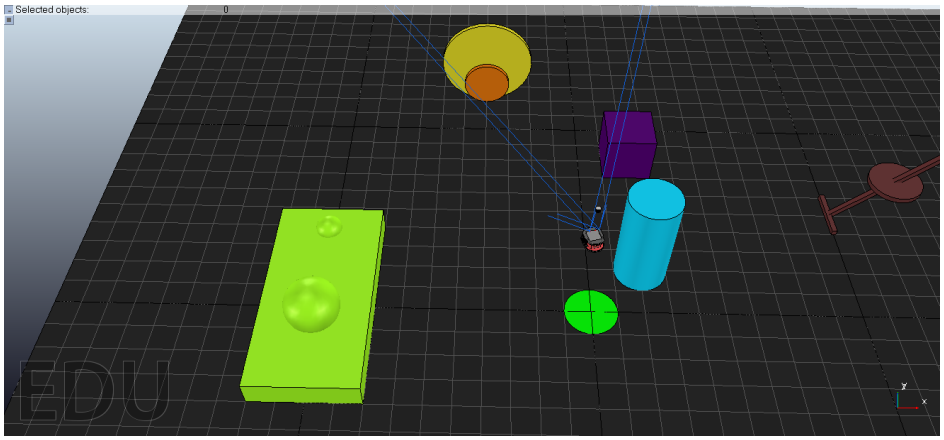
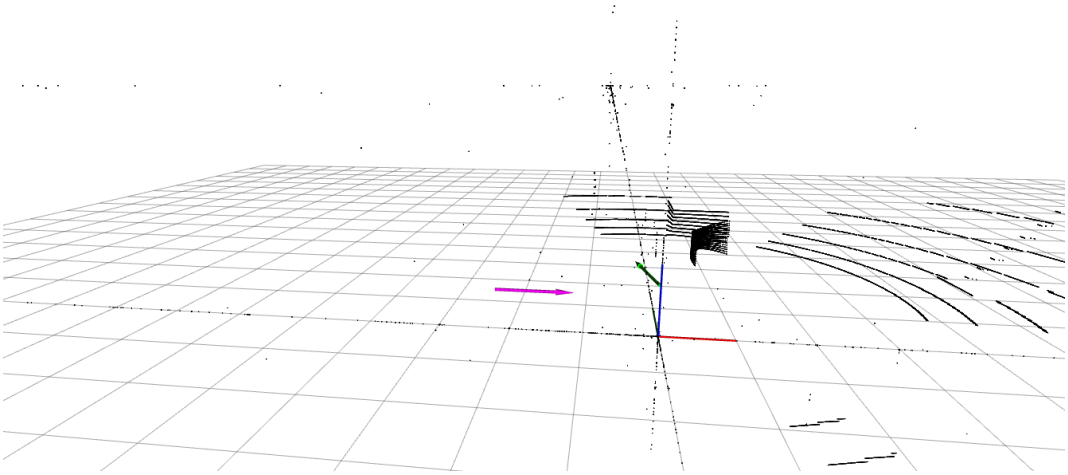
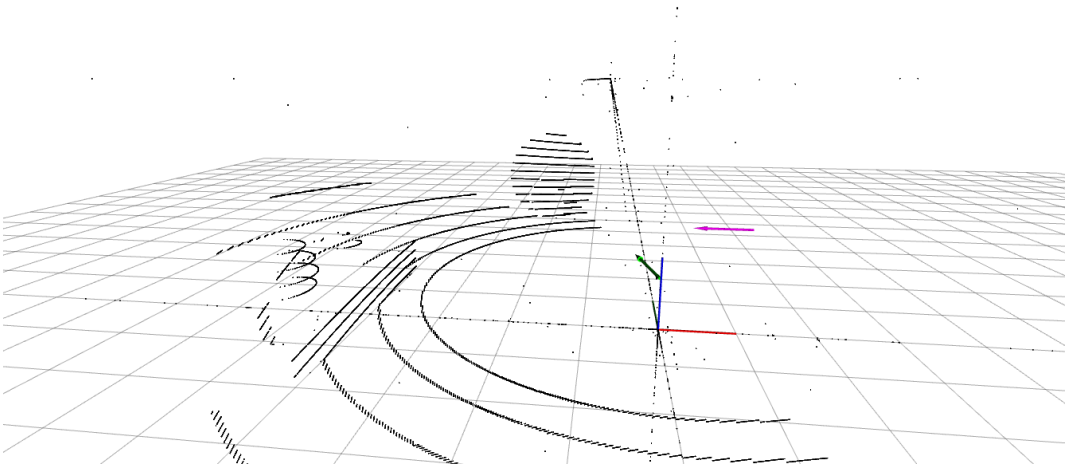


Figure 6.1: UGV 05 scene with various unique objects.

The stereo vision node was used to estimate depth and pose of the robot, in particular the pose of the main/left camera. One of the most influential factors on depth estimation of correspondences are the different



(a) First half of the full scan.



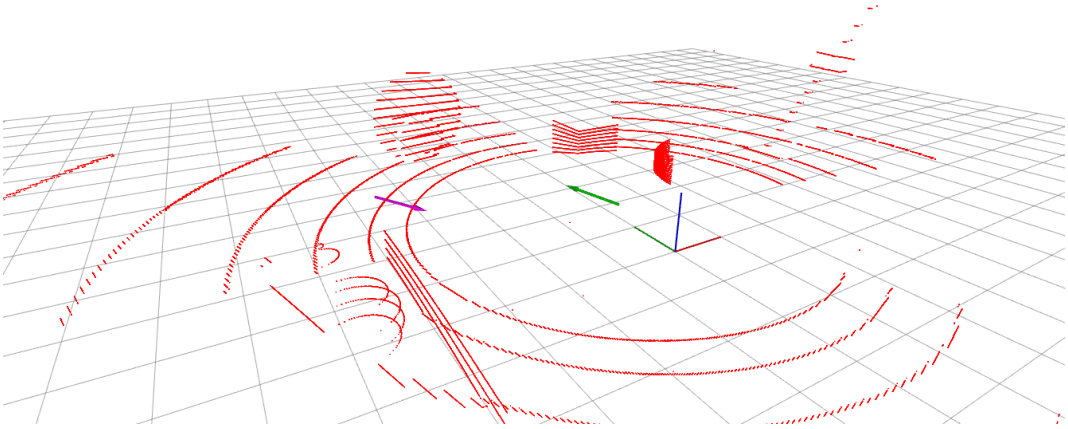
(b) Second half of the full scan.

Figure 6.2: Unfiltered data provided from V-REP via Lua script emulating the Velodyne Puck VLP16 laser sensor.

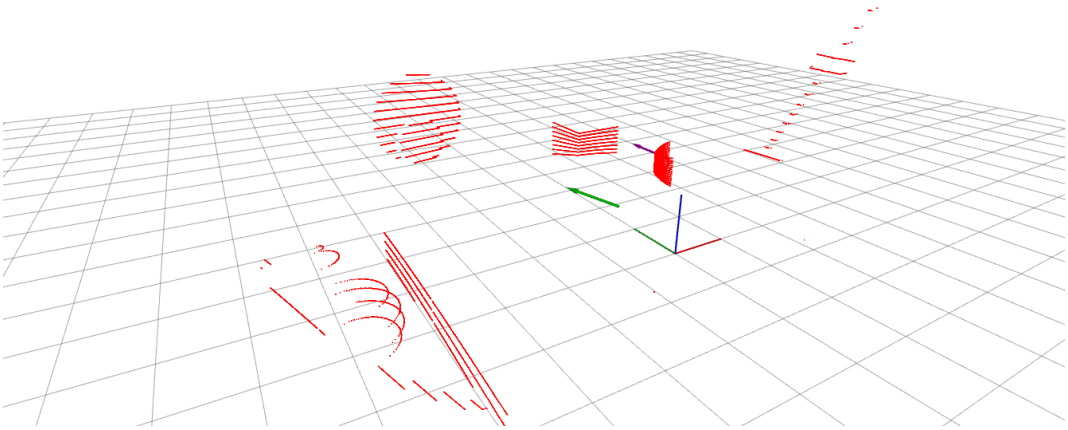
feature detection methods. The user has a choice between ORB [12][29], AKAZE [1][39], SIFT [9] [32] and SURF [9][35], where the last two are part of the non-free modules of OpenCV. This is an important fact, since not all distributions, like Ubuntu, have the non-free modules available in their official repositories. By default the ORB feature detector is used. The node allows for a comparison between computational performance between different detectors, if the *VERBOSE* flag or the ROS parameter /stereo_vision/verbose is set to a value higher than 0. This switch allows for the determination of the time durance for detecting features. The number of detected points is also printed in the standard output. It is not necessarily a very precise tool of estimation, since the time calls cannot be guaranteed to be accurate, however based on observations it is sufficient to get an general performance overview.

A comparison for the scene illustrated in (F6.4) is presented in (T6.1). It should be noted that the UGV was immobile. Even though the environment is a simulation, there are differences between frames of the same camera view, due to micro movement inflicted by the physical engine and inaccuracies of computational methods. Noticeable is also a difference in certain feature points, however this might be due to internal mechanisms of the detector. It should be taken into future consideration, that even if the unit is still the detected feature points might be very different between frames.

The performance comparison is purely based on detected features. The number of features that are



(a) Noise threshold set at $threshold = 0.001$.



(b) Noise threshold set at $threshold = 0.02$, eliminating the floor.

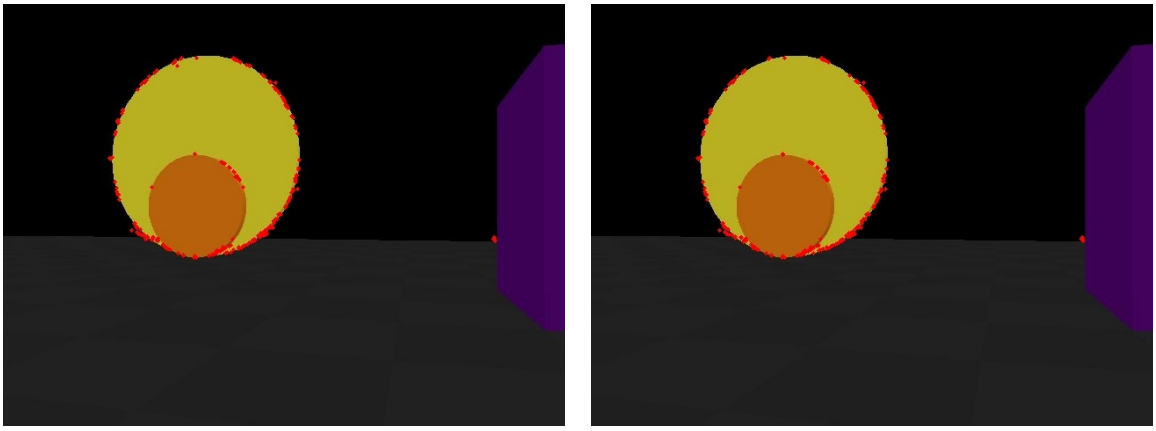
Figure 6.3: Filtered data using VLP16 node preprocessor.

used in the final routine of depth estimation is usually much lower, which has been also shown in (T6.1) by the reduction of and used correspondences entries. Current implementation uses findEssentialMat call to find inliers using a RANSAC method with $threshold = 5$, which then is used to reduce the number of correspondences. This is suboptimal and could be replaces by simply limiting the number of, optionally sorted, correspondences to a limit of 100 points. The experimental scenario is the "scene 05" with an immobile UGV unit, equipped with the stereo vision system having the disc shaped object in the field of view. According to acquired results, ORB is the fastest detector. It provides also the highest number of feature correspondences, making it, based only on these aspects, the most suited feature detection method. For further processing, especially the registration of point clouds acquired from the depth estimation, a higher number of points is preferred. One should take into consideration that the environment is a simulation, therefore the artificial aspects of objects probably significantly reduce the number of detected features. One possibility of expansion of this experiment, would be to add textures onto the objects. Abstract evaluation, as shown in , also plays a significant role, since having points resemble the shape of the object should make it more probable to find a correct solution for the ICP algorithm during registration of the point cloud. The times for feature detection in AKAZE, SIFT and SURF are higher by a factor of $6 \div 10$ with respect to time for ORB. SIFT provides the lowest number of points used in the final. SIFT has the lowest number of

Table 6.1: Comparison of different feature detection methods based on approximate data from scene 05.

Feature detector	ORB	AKAZE	SIFT	SURF
Time t $[s]$	0.04	0.30	0.40	0.25
Number of detected features n [1]	240	120	30	235
Time per feature $\frac{t}{n}$ $[ms]$	0.167	2.5	13.3	1.06
Reduction of correspondences p [%]	40	37	25	55
Number of used correspondences m [1]	144	76	23	106

detected and used features, which in itself is not necessarily of disadvantage. It means less point to match, therefore the overall computation load might not necessarily be higher considering that SIFT has the highest relative time per feature, higher than ORB by a factor of almost 100. In general all feature detection methods perform well for a human observer, since the shape of the disc object is recognizable. For the purpose of point cloud registration however, ORB is still the preferred choice, because of the high number of features detected in the shortest amount of time and for using features on edges and corners of objects. These create a point cloud resembling the actual object, and not what could be considered a scaled copy, a ring or a double edge as illustrated (F6.5d) with SURF or possibly even AKAZE (F6.5b). One practical observation is that the SURF feature detection method is most prone to cause an error and result in termination of the node.



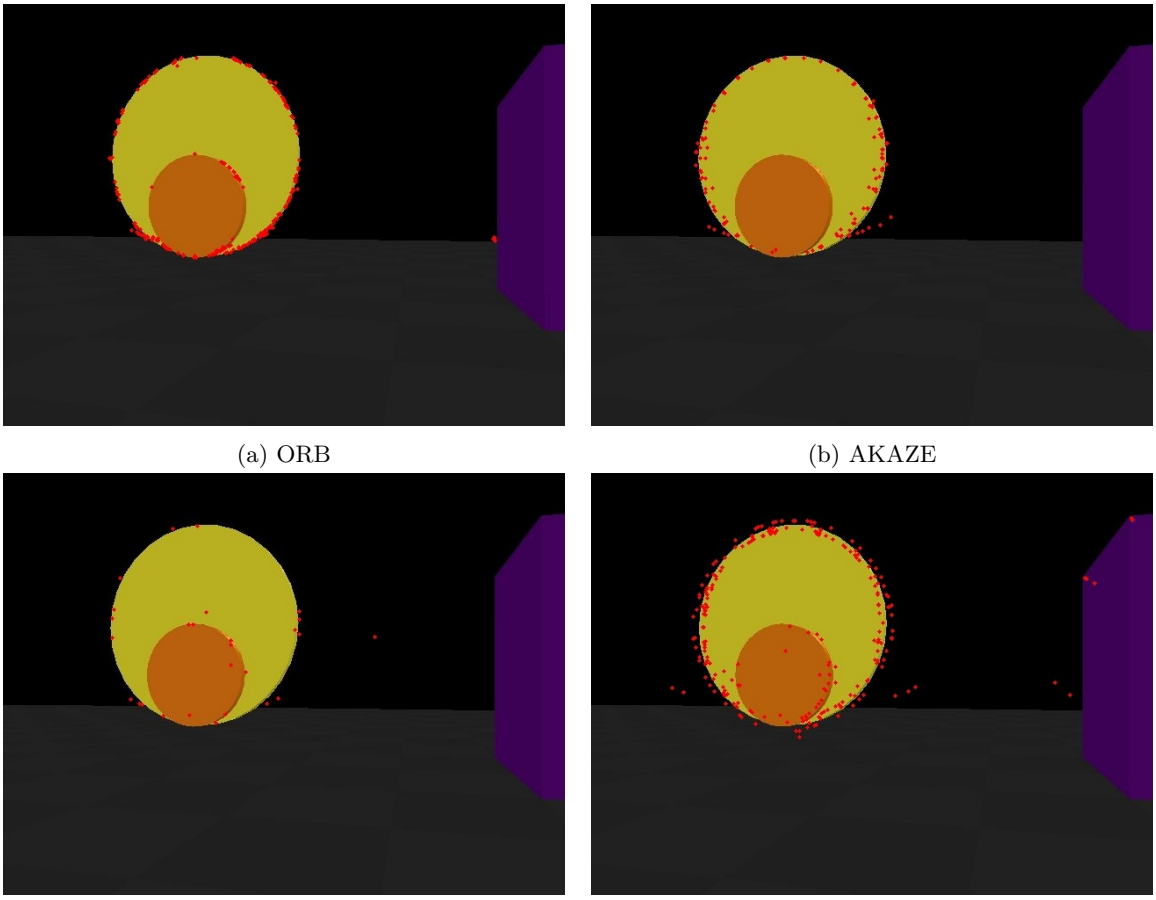
(a) Frame A. (b) Frame B.

Figure 6.4: Stereo vision feature detection using ORB method for the same camera view, without any movement.

The image (F6.5) illustrates the same camera view of the same scene with drawn detected features. This is easily achieved by changing the ROS parameter /stereo_vision/featuredetectionmethod to a value between 0 and 3 with accordance to (R5.3).

What should be noted is that compressed streams are being used for the images. The artefacts of compression are recognizable on careful inspection. This is a step towards the decentralized aspect with the constraint of limited communication: in bandwidth, range, volume and time duration.

Depth estimation creates a corona effect around objects from the stereo vision images as illustrated in (F6.6). This effect is magnified by the artefacts resulting from compression of input images. Since the feature detectors usually extract points of interest around edges of objects, these edges through estimation seem projected throughout space, while maintaining the shape from the point of view of the vision sensor.



(a) ORB (b) AKAZE

(c) SIFT (d) SURF

Figure 6.5: Comparison of different feature detection methods.

In case of the SURF descriptor, that extracts features at a certain distance from the edges, the estimated shape of the object is significantly different from the actual shape. There is also a significant offset visible.

In order to correct the depth estimation, a disparity based depth map was intended to be used. The process illustrated in (F6.7) was also performed in the stereo vision node. However the acquired results were different from anticipated as shown in (F6.8).

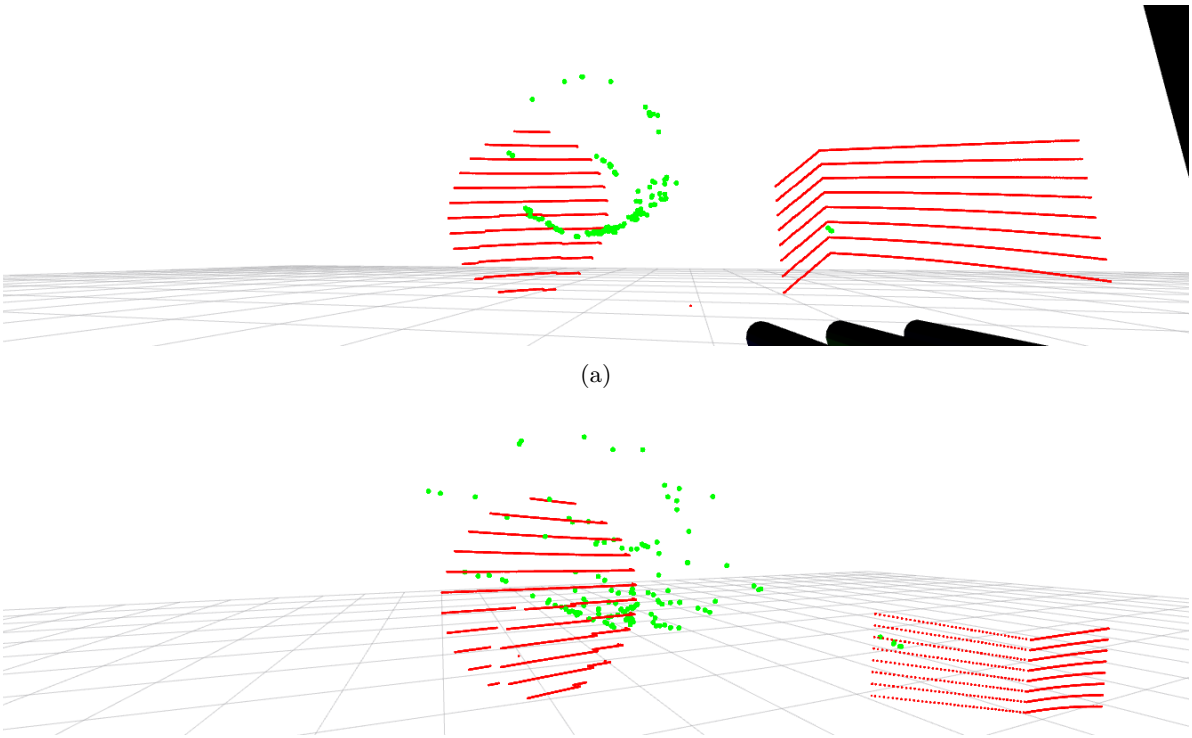
Current implementation does not detect camera movement properly.

It was anticipated to present more empirical results, especially with respect to the pose estimations. For that the estimation errors have to be defined and computed. The error for estimated pose and actual pose of the left/main camera in stereo vision system can be computed as defined in (E6.1). Similarly the pose estimation error can be calculated for the local ICP routine, based on ICP result from estimated cloud registration in the Velodyne cloud as portrayed in (E6.2). The absolute pose known from the simulation environment is noted as $M_{simulation}$. These errors can be further divided into rotational and translational components.

$$e_{stereo-vision} = M_{simulation} - M_{stereo-vision-estimated} \tag{6.1}$$

$$e_{local-icp} = M_{simulation} - M_{local-icp-estimated} \tag{6.2}$$

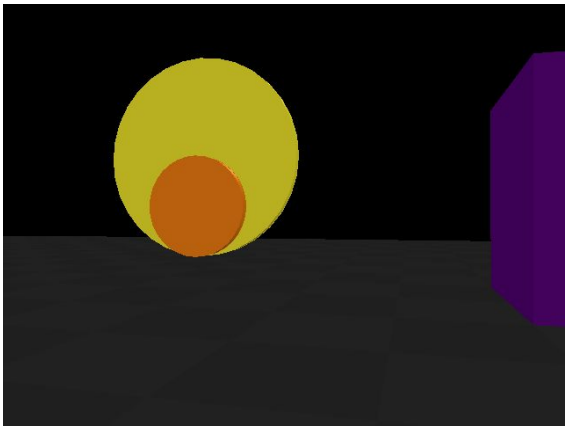
At the current development stage, further specifics cannot be reliably provided.



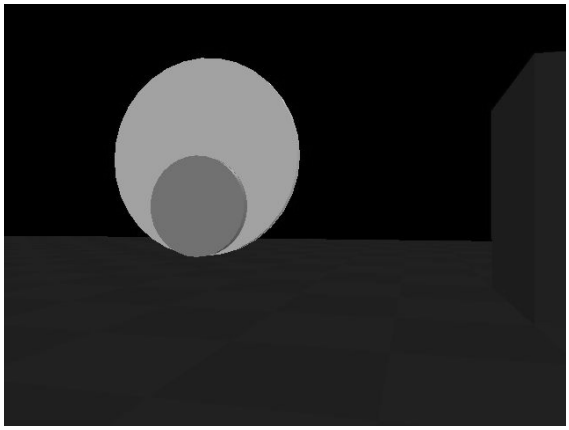
(a)

(b)

Figure 6.6: The corona effect observed in point cloud acquired from depth estimation routine from the stereo vision system.

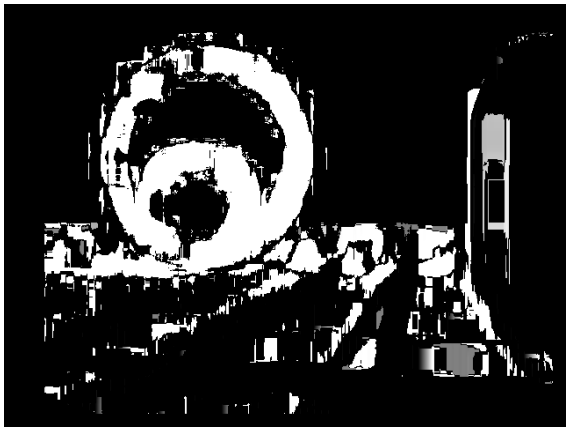


(a) Input image from main/left camera in RGB color space



(b) Input image from main/left camera in grayscale color space

(c) Depth map for $nd = 16$ and $bs = 5$



(d) Depth map for $nd = 32$ and $bs = 31$

Figure 6.7: Process of generating depth map using disparity from the stereo vision system.

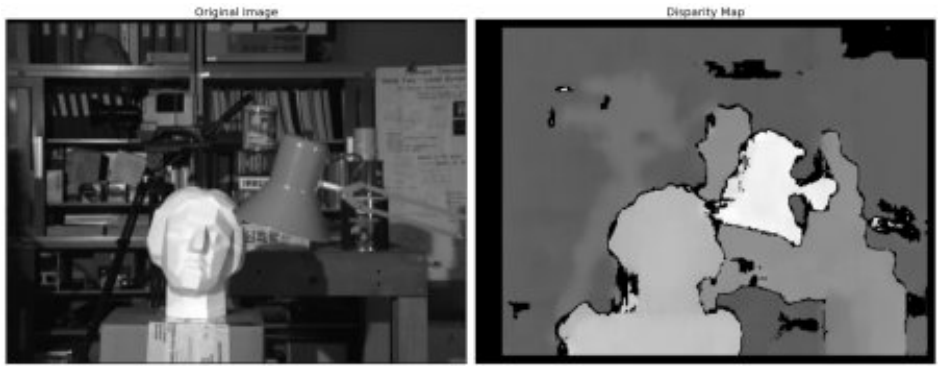


Figure 6.8: OpenCV disparity map example [**?**].

Chapter 7

Conclusions

Future work

The presented developed framework has a lot of potential for further improvement in general approach of the heterogeneous sensors and robot types, decentralized and collaborative aspects. Further integration with previous CoSLAM framework would be another milestone.

The Velodyne preprocessor is basically finished, unless the processing of raw data is required or desired. In that case the already provided skeleton for raw message processing can be expanded. Furthermore programmatic aspects like functions for filters and point cloud processing could be implemented. This would make the main function more readable, although care has been taken in order to provide a transparent coding style and sufficient commenting in order to guarantee understanding of each step. On the V-REP side the structure for emulating publishing of raw messages is also provided with successful sending of a dummy message. The processor of converting the available data in floats into the specifically formatted raw Velodyne messages however, is not trivial. Considering the forward conversion, several corrective steps would have to be undertaken, that are not documented.

For stereo vision and local ICP nodes there a lot of improvement potential. The main reason for using Python was the ability to quickly introduce changes to the code and that way troubleshoot as well as evaluate results. Most of the code is already inside classes that the nodes provide, however in case of the stereo vision node, the depth and relative pose estimation are still inside the main loop. Additionally most of used the variables are global. It might be desired to shift almost all of the estimation code into separate functions in order to make the main call easier to follow and allow for reuse of the programming routines elsewhere.

Currently the estimation results are not satisfactory, so a further in-depth analysis is necessary. There is a specific redundancy, because of the initial approach, that could be optimized as well. In general both nodes perform their operations fast enough, especially without verbose output. An analytical approach in solving the problem of depth and pose estimation might be better, even more so, since the essential matrix is basically known from the geometrical assembly of the stereo vision system. Currently, dedicated OpenCV calls are made to recover pose of the camera for example. It would be interesting to see if similar performance is achievable and to confirm the results of the current approach with one using the essential matrix. Another interesting point would be to assume that the stereo vision system is unknown, and try to recreate the current routine with fundamental matrix.

Furthermore implementation of octree objects instead of simple point clouds or possibly as additional information, as well as the use of a filter for data fusion, especially for the relative pose estimation problem, might be of benefit.

V-REP simulation scenes also provide potential for improvement. Currently simple control algorithm is implemented, mostly with the goal to accomplish the decentralized and collaborative aspects, by using the rendezvous technique to exchange data and perform multi-robot SLAM. The scripts for sensors are complete and only require configuration based on user specific application. The UGV script could benefit from the

new features introduced in V-REP version 3.4.0, where access to ROS parameter server is possible. This would globally allow for customization of the same scene file by setting initial parameters of each unit in the simulation. More specifically however, the configuration of each robot and it's script could be controlled through the ROS parameter server as well. This would make long simulation sessions possible, without the need of restarting after each parameter change. From experience it is advisable to use the ROS framework for configuration instead of the V-REP provided Lua script outside parameters or UI. This can be an optional addition, but the main process should take place outside of V-REP. Using YAML [10][38] files for initial configuration of robots would be one possibility to expand scenes. In that way a launch file referring to a different YAML file could loaded with the same scene providing other initial configurations. This would provide a very clean and straightforward way of configuring scenes, that would otherwise have to be saved as a separate file for each variation.

Contribution and innovation

The main contribution of the presented work is an initial framework CoSLAM V-REP for multi SLAM with V-REP as the simulation environment. The work allows for further development and integration with previous code base, especially with respect to custom messages, which are a significant part of CoSLAM. The heterogeneous aspect with respect to sensors for the SLAM application is the most developed out of the three, where initial work towards decentralized and collaborative aspects has been completed. However at current stage it is still not enough to fulfil the initial assumptions.

As already stated in the *"Comparison"*, the main improvement over previous framework is the capability of performing all routines on-line. The merge of different sensors allows for further development in the are of multi-unit SLAM.

Conclusions

The presented developed framework allows for an integration with the CoSLAM to a certain extent. The thesis was for most part concentrated on development of CoSLAM V-REP package, which can be considered an extension to previous code base. The undertaken development has potential to complete all three initial aspects: heterogeneous, in sensors and robot types, decentralized and collaborative. Initial results suggest that more work needs to be done in the implementation of relative pose estimation. The depth estimation needs further improvement as well as does the local iterative closest point routine.

Bibliography

[1] AKAZE local features matching OpenCV 3.0.0-dev documentation, . URL http://docs.opencv.org/3.0-beta/doc/tutorials/features2d/akaze_matching/akaze_matching.html.

[2] Arch Linux - ArchWiki, . URL https://wiki.archlinux.org/index.php/Arch_Linux.

[3] Arch User Repository - ArchWiki, . URL https://wiki.archlinux.org/index.php/Arch_User_Repository.

[4] Camera Calibration and 3d Reconstruction OpenCV 2.4.13.2 documentation, . URL http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[5] Camera calibration With OpenCV OpenCV 2.4.13.2 documentation, . URL http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[6] Depth Map from Stereo Images - OpenCV 3.0.0-dev documentation, . URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html.

[7] Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix, . URL http://ksimek.github.io/2013/08/13/intrinsic/.

[8] image_pipeline/CameraInfo - ROS Wiki, . URL http://wiki.ros.org/image_pipeline/CameraInfo.

[9] Introduction to SIFT (Scale-Invariant Feature Transform) OpenCV 3.0.0-dev documentation, . URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro.

[10] The Official YAML Web Site, . URL http://yaml.org/.

[11] OpenCV: Camera Calibration and 3d Reconstruction, . URL http://docs.opencv.org/3.2.0/d9/d0c/group___calib3d.html#gadb7d2dfcc184c1d2f496d8639f4371c0.

[12] ORB (Oriented FAST and Rotated BRIEF) OpenCV 3.0.0-dev documentation, . URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[13] Pioneer P3-DX | Mapping & Navigation Robot, . URL http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx.

[14] The Programming Language Lua, . URL http://www.lua.org/.

[15] QEMU, . URL http://www.qemu.org/.

[16] Release notes V-REP 3.4.0, . URL http://www.coppeliarobotics.com/helpFiles/en/versionInfo.htm/#3.4.0.

[17] Release notes V-REP 3.3.2, . URL http://www.coppeliarobotics.com/helpFiles/en/versionInfo.htm/#3.3.2.

[18] sensor_msgs/CameraInfo Documentation, . URL http://docs.ros.org/api/sensor_msgs/html/msg/CameraInfo.html.

[19] sensor_msgs/PointCloud2 Documentation, . URL http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html.

[20] SQUASHFS - A squashed read-only filesystem for Linux, . URL http://squashfs.sourceforge.net/.

[21] Straw Lab: Home, . URL https://strawlab.org/.

[22] Ubuntu 14.04.5 LTS (Trusty Tahr), . URL http://releases.ubuntu.com/14.04/.

[23] velodyne_driver - ROS Wiki, . URL http://wiki.ros.org/velodyne_driver.

[24] velodyne_msgs/VelodynePacket Documentation, . URL http://docs.ros.org/indigo/api/velodyne_msgs/html/msg/VelodynePacket.html.

[25] velodyne_msgs/VelodyneScan Documentation, . URL http://docs.ros.org/indigo/api/velodyne_msgs/html/msg/VelodyneScan.html.

[26] Velodyne PUCK VLP16 Datasheet, . URL http://velodynelidar.com/docs/datasheet/63-9229_Rev-F_Puck%20_Spec%20Sheet_Web.pdf.

[27] COLLADA, May 2017. URL https://en.wikipedia.org/w/index.php?title=COLLADA&oldid=781473515. Page Version ID: 781473515.

[28] Lidar, July 2017. URL https://en.wikipedia.org/w/index.php?title=Lidar&oldid=791103525. Page Version ID: 791103525.

[29] ORB (Oriented FAST and Rotated BRIEF) OpenCV 3.0.0-dev documentation, 2017. URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[30] python-pcl: Python bindings to the pointcloud library (pcl), July 2017. URL https://github.com/strawlab/python-pcl. original-date: 2012-05-16T10:26:35Z.

[31] Python (programming language), July 2017. URL https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=791042630. Page Version ID: 791042630.

[32] Scale-invariant feature transform, June 2017. URL https://en.wikipedia.org/w/index.php?title=Scale-invariant_feature_transform&oldid=788248019. Page Version ID: 788248019.

[33] Scripting language, June 2017. URL https://en.wikipedia.org/w/index.php?title=Scripting_language&oldid=787917984. Page Version ID: 787917984.

[34] Singular value decomposition, July 2017. URL https://en.wikipedia.org/w/index.php?title=Singular_value_decomposition&oldid=788566637. Page Version ID: 788566637.

[35] Speeded up robust features, March 2017. URL https://en.wikipedia.org/w/index.php?title=Speeded_up_robust_features&oldid=771988954. Page Version ID: 771988954.

[36] velodyne: ROS support for Velodyne 3d LIDARs, July 2017. URL https://github.com/ros-drivers/velodyne. original-date: 2013-04-03T22:37:10Z.

[37] Vision sensor properties, 2017. URL http://www.coppeliarobotics.com/helpFiles/en/visionSensorPropertiesDialog.htm.

[38] YAML, July 2017. URL https://en.wikipedia.org/w/index.php?title=YAML&oldid=790867350. Page Version ID: 790867350.

[39] Pablo F. Alcantarilla. akaze: Accelerated-KAZE Features, July 2017. URL https://github.com/pablofdezalc/akaze. original-date: 2013-11-11T14:44:29Z.

[40] Rabbia Asghar and Ernest Skrzypczyk. Estimating the fundamental matrix. Technical report, Universit degli studi di Genova, December 2015.

[41] Josep Aulinas, Yvan R. Petillot, Joaquim Salvi, and Xavier Llad. The SLAM problem: a survey. In *CCIA*, pages 363–371. Citeseer, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.6439&rep=rep1&type=pdf.

[42] T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): part. *IEEE Robotics & Automation Magazine*, 2(13):99–110, 2006.

[43] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5968.

[44] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jos Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. URL http://ieeexplore.ieee.org/abstract/document/7747236/.

[45] Nicole Camous. Collaborative Visual SLAM. Bibliography, cole centrale de Nantes, Nantes, France, 2016.

[46] Nicole Camous. *Collaborative Visual SLAM.* Master thesis, cole centrale de Nantes, Nantes, France, 2016.

[47] Franois Chaumette and Seth Hutchinson. Visual servo control. I. Basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4015997.

[48] Franois Chaumette and Seth Hutchinson. Visual servo control, Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007. URL https://hal.inria.fr/inria-00350638/.

[49] Nived Chebrolu, David Marquez-Gamez, and Philippe Martinet. Collaborative Visual SLAM Framework for a Multi-Robot System. 2015. URL http://ppniv15.irccyn.ec-nantes.fr/material/session1/Chebrolu/paper.pdf.

[50] Howie Choset and Joel Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In *IEEE international conference on robotics and automation*, pages 1643–1643. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1995. URL https://www-preview.ri.cmu.edu/pub_files/pub1/choset_howie_1995_2/choset_howie_1995_2.pdf.

[51] Titus Cieslewski, Simon Lynen, Marcin Dymczyk, Stphane Magnenat, and Roland Siegwart. Map API-scalable decentralized map building for robots. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 6241–6247. IEEE, 2015. URL http://ieeexplore.ieee.org/abstract/document/7140075/.

[52] Alexander Cunningham, Manohar Paluri, and Frank Dellaert. DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030. IEEE, 2010. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5652875.

[53] Gregory Dudek, Junaed Sattar, and Anqi Xu. A visual language for robot control and programming: A human-interface study. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2507–2513. IEEE, 2007. URL http://ieeexplore.ieee.org/abstract/document/4209460/.

[54] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*, 13(2):99–110, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1638022.

[55] Felix Endres, Jrgen Hess, Nikolas Engelhard, Jrgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696. IEEE, 2012. URL http://ieeexplore.ieee.org/abstract/document/6225199/.

[56] Felix Endres, Jrgen Hess, Jrgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6594910.

[57] Jakob Engel, Thomas Schps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014. URL http://link.springer.com/chapter/10.1007/978-3-319-10605-2_54.

[58] Jakob Engel, Jrg Stckler, and Daniel Cremers. Large-scale direct SLAM with stereo cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1935–1942. IEEE, 2015. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7353631.

[59] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *arXiv:1607.02565 [cs]*, July 2016. URL http://arxiv.org/abs/1607.02565. arXiv: 1607.02565.

[60] U.S.-Canada Power System Outage Task Force. Final Blackout Report - Chapters 1-3 - ch1-3.pdf, 2004. URL https://www.ferc.gov/industries/electric/indus-act/reliability/blackout/ch1-3.pdf.

[61] Friedrich Fraundorfer and Davide Scaramuzza. Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90, June 2012. ISSN 1070-9932. doi: 10.1109/MRA.2012.2182810. URL http://ieeexplore.ieee.org/document/6153423/.

[62] Patrick Gerland, Adrian E. Raftery, Hana evckov, Nan Li, Danan Gu, Thomas Spoorenberg, Leontine Alkema, Bailey K. Fosdick, Jennifer Chunn, Nevena Lalic, Guiomar Bay, Thomas Buettner, Gerhard K. Heilig, and John Wilmoth. World Population Stabilization Unlikely This Century. *Science*, 346(6206):234–237, October 2014. ISSN 0036-8075. doi: 10.1126/science.1257469. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4230924/.

[63] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4082128.

[64] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. 2004. ISBN 978-0-511-18711-7 978-0-511-18618-9 978-0-511-18895-4 978-0-511-18535-9 978-0-511-18451-2 978-0-511-81168-5. URL http://dx.doi.org/10.1017/CBO9780511811685. OCLC: 171123855.

[65] J. Hilditch. *Linear skeletons from square cupboards, Machine Intelligence 4 (B. Meltzer & D. Michie, eds.), 403/420*. Edinburgh Univ. Press, 1969.

[66] J. Hoog, Stephen Cameron, Arnoud Visser, and others. Selection of rendezvous points for multi-robot exploration in dynamic environments. 2010. URL http://dare.uva.nl/record/1/324475.

[67] James Jessup, Sidney N. Givigi, and Alain Beaulieu. Robust and Efficient Multirobot 3-D Mapping Merging With Octree-Based Occupancy Grids. *IEEE Systems Journal*, pages 1–10, 2015. ISSN 1932-8184, 1937-9234, 2373-7816. doi: 10.1109/JSYST.2015.2422615. URL http://ieeexplore.ieee.org/document/7098348/. 00002.

[68] Michio Kaku. *Physics of the future: how science will shape human destiny and our daily lives by the year 2100*. Doubleday, New York, 2011. ISBN 978-0-385-53081-1. URL http://ebookdownload.3m.com/sites/prototypes/web/media/themes/mmm_patron/img/landing_page/step1.png. OCLC: 708581888.

[69] Nikolai S. Kardashev. Transmission of Information by Extraterrestrial Civilizations. *Soviet Astronomy*, 8:217, 1964. URL http://articles.adsabs.harvard.edu/full/1964SvA.....8..217K/0000219.000.html.

[70] Frank R. Kschischang, Brendan J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=910572.

[71] Ray Kurzweil. *The singularity is near: when humans transcend biology*. Viking, New York, 2005. ISBN 978-0-670-03384-3.

[72] Jean-Claude Latombe. Robot Motion Planning (The Kluwer International Series in Engineering and Computer Science). 1990. URL http://www.citeulike.org/group/5970/article/2782334.

[73] Keith YK Leung, Timothy D. Barfoot, and Hugh HT Liu. Distributed and decentralized cooperative simultaneous localization and mapping for dynamic and sparse robot networks. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3841–3847. IEEE, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5979783.

[74] Stephanie Lowry, Niko Snderhauf, Paul Newman, John J. Leonard, David Cox, Peter Corke, and Michael J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016. URL http://ieeexplore.ieee.org/abstract/document/7339473/.

[75] Malika Meghjani and Gregory Dudek. Combining Multi-robot Exploration and Rendezvous. pages 80–85. IEEE, May 2011. ISBN 978-1-61284-430-5. doi: 10.1109/CRV.2011.18. URL http://ieeexplore.ieee.org/document/5957545/.

[76] Christopher Mei, Gabe Sibley, Mark Cummins, Paul Newman, and Ian Reid. RSLAM: A system for large-scale mapping in constant-time using stereo. *International journal of computer vision*, 94(2):198–214, 2011. URL http://link.springer.com/article/10.1007/s11263-010-0361-7.

[77] Nathan Michael, Shaojie Shen, Kartik Mohta, Yash Mulgaonkar, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, Kazunori Ohno, Eijiro Takeuchi, and Satoshi Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, September 2012. ISSN 15564959. doi: 10.1002/rob.21436. URL http://doi.wiley.com/10.1002/rob.21436.

[78] Jack Morrison, Dorian Glvez-Lpez, and Gabe Sibley. Scalable Multi-Device SLAM. 2014. URL http://doriangalvez.com/papers/MorrisonRSS14.pdf.

[79] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, October 2015. ISSN 1552-3098, 1941-0468. doi: 10.1109/TRO.2015.2463671. URL http://ieeexplore.ieee.org/document/7219438/.

[80] Jos Neira and Juan D. Tards. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on robotics and automation*, 17(6):890–897, 2001. URL http://ieeexplore.ieee.org/abstract/document/976019/.

[81] David Nistr. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004. URL http://ieeexplore.ieee.org/abstract/document/1288525/.

[82] Arne Nordmann. English: Epipolar geometry, 2007. URL https://commons.wikimedia.org/wiki/File:Epipolar_geometry.svg.

[83] Denis Oberkampf, Daniel F. DeMenthon, and Larry S. Davis. Iterative pose estimation using coplanar points. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*, pages 626–627. IEEE, 1993. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=341055.

[84] Inc Open Source Robotics Foundation. Gazebo, 2017. URL http://gazebosim.org/.

[85] Inc Open Source Robotics Foundation. TurtleBot, 2017. URL http://www.turtlebot.com/.

[86] Daniel Oram. Rectification for any epipolar geometry. In *BMVC*, volume 1, pages 653–662, 2001. URL https://pdfs.semanticscholar.org/277b/5cf85fa53adc0d2402e020681d31d904b1b9.pdf.

[87] Hossein Pishro-Nik. Introduction to probability, statistics and random processes. 2014. URL http://scholarworks.umass.edu/ece_ed_materials/1/.

[88] Coppelia Robotics. Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot., 2017. URL http://www.v-rep.eu/.

[89] Nicholas Roy and Gregory Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, 2001. URL http://link.springer.com/article/10.1023/A:1011219024159.

[90] Parrot SA. Drones | Parrot Store Official, 2017. URL https://www.parrot.com/us/drones#decouvrez-notre-gamme-.

[91] Sajad Saeedi, Michael Trentini, Mae Seto, and Howard Li. Multiple-Robot Simultaneous Localization and Mapping: A Review. *Journal of Field Robotics*, 33(1):3–46, 2016. URL http://onlinelibrary.wiley.com/doi/10.1002/rob.21620/full.

[92] Davide Scaramuzza. Tutorial on Visual Odometry, 2012. URL http://rpg.ifi.uzh.ch/visual_odometry_tutorial.html.

[93] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92, December 2011. ISSN 1070-9932. doi: 10.1109/MRA.2011.943233. URL http://ieeexplore.ieee.org/document/6096039/.

[94] Riccardo Spica, Paolo Robuffo Giordano, and Franois Chaumette. Active structure from motion: Application to point, sphere and cylinder. *IEEE Transactions on Robotics*, 30(6):1499–1513, 2014. URL https://hal.archives-ouvertes.fr/hal-01086866/.

[95] Riccardo Spica, Paolo Robuffo Giordano, and Franois Chaumette. Coupling visual servoing with active structure from motion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3090–3095. IEEE, 2014. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6907303.

[96] Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93, 2000. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/1534.

[97] Sebastian B. Thrun and Knut Mller. Active exploration in dynamic environments. In *Advances in neural information processing systems*, pages 531–538, 1992. URL http://papers.nips.cc/paper/573-active-exploration-in-dynamic-environments. 00150.

[98] Department of Economic and Social Affairs United United Nations, Population Division. *World population prospects: The 2015 revision*. ES Affairs United Nations, NY, USA, 2015. URL https://esa.un.org/unpd/wpp/.

[99] Danping Zou and Ping Tan. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):354–366, 2013. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6193110.