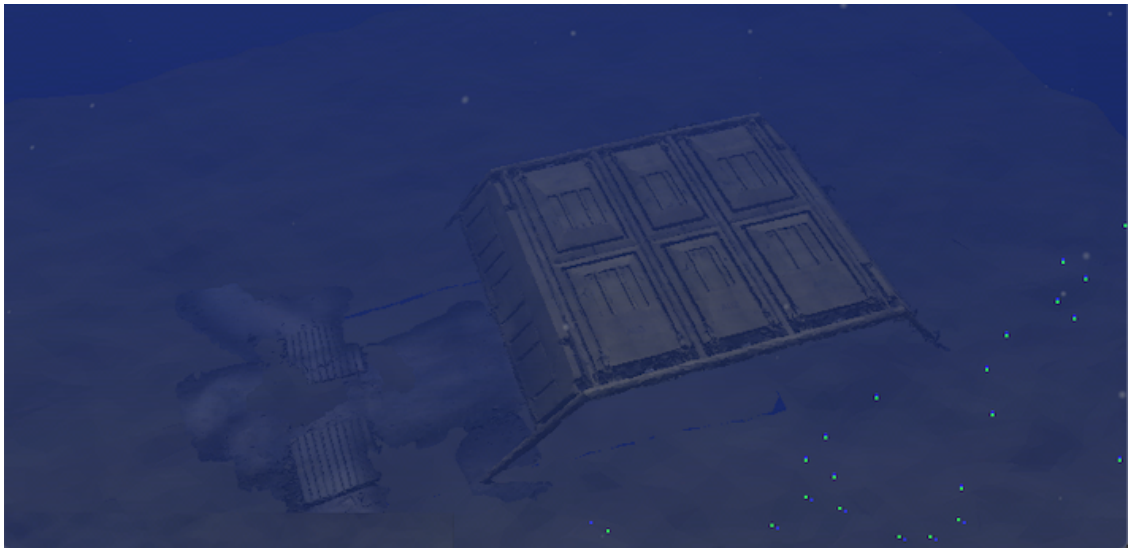
CHALMERS
UNIVERSITY OF TECHNOLOGY



Convolutional Linear Genetic Programming for Underwater Image Classification

Convolutional Linear Genetic Programming for Underwater Man-Made Structure Image Classification

Master's thesis in Applied Data Science

GABRIELĖ KASPARAVIČIŪTĖ

Department of Space, Earth and Environment
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

Convolutional Linear Genetic Programming for Underwater Image Classification

Convolutional Linear Genetic Programming for Underwater Man-Made Structure Image Classification

GABRIELĖ KASPARAVIČIŪTĖ

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Space, Earth and Environment
Chalmers University of Technology
Gothenburg, Sweden 2020

Convolutional Linear Genetic Programming for Underwater Image Classification
Convolutional Linear Genetic Programming for Underwater Man-Made Structure Image Classification
GABRIELĖ KASPARAVIČIŪTĖ

Supervisor: Peter Nordin, Department of Physical Resource Theory
Examiner: Claes Andersson, Department of Space, Earth and Environment, Physical Resource Theory

Master's Thesis 2020:NN
Department of Space, Earth and Environment
Physical Resource Theory
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Subsea floor and template taken from the real world 3D cloud point and imported into an underwater simulator.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Convolutional Linear Genetic Programming for Underwater Image Classification
Convolutional Linear Genetic Programming for Underwater Man-Made Structure Image Classification
GABRIELĖ KASPARAVIČIŪTĖ
Department of Space, Earth and Environment Chalmers University of Technology

Abstract

Real time 3D point cloud annotation of the subsea environment is an expensive and challenging task. Therefore, industry is looking at robotics and machine learning recent advancements for developing resident autonomous underwater vehicle (AUV). In order to be able to achieve this task simultaneous localization and mapping (SLAM) technique is utilized. However, a resident AUV is required to travel long distances between man-made objects that do not require such a high resolution annotation. This paper presents an innovative binary image classification method employing linear genetic programming which aids in differentiating between the images of interest and not. This allows to save battery and computational power to avoid high resolution 3D cloud points during these trips between man-made objects. The classifier's results show over 95% accuracy. This paper also provides an overview of SLAM techniques according to the industrial stakeholder's requirements followed by a review on the background of underwater simulations used in both industry and academia.

Keywords: convolutional, linear, genetic, programming, underwater, image, classification, binary, subsea, lgp.

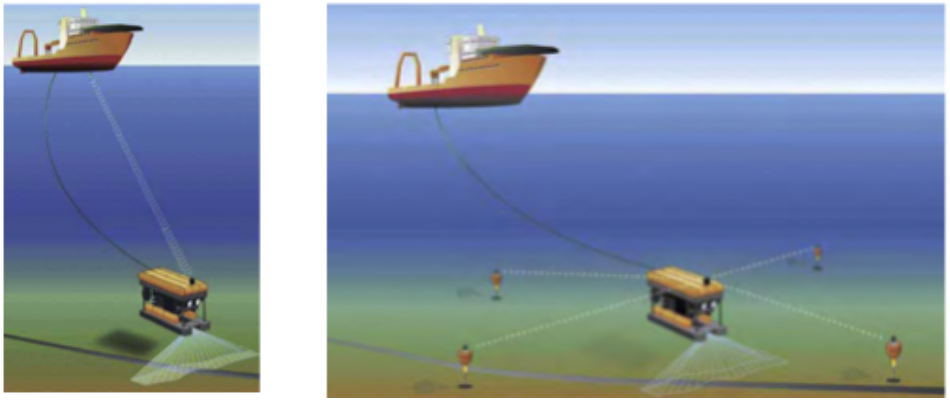
Contents

List of Figures

List of Tables

1
Introduction

The underwater world provides us with many resources: from fisheries, archaeological discoveries, and deep sea minerals to oil and gas extraction. Currently, these tasks are mostly accomplished manually by the use of remotely operated vehicles (ROV). Methods like Ultra Short Baseline (USBL) and Long Baseline (LBL) allow for more accurate ROV localization (Kim, 2012) (see fig. 1.1). USBL requires a vessel in the sea, with a transceiver mounted on the bottom of its hull, while the transponder is usually mounted on a diving ROV. By means of acoustic signals sent between the transceiver and the transponder, it is possible to obtain an accurate distance and bearing of the ROV. LBL works in the same way, except that it uses a network of transponders, also known as beacons, which are planted on the sea floor. Regardless, the ROV still requires an operator. As seen from the provided industry standard, current manual assignments are costly and time consuming, thus a different method of localization is needed.
Automation is crucial in order to be able to increase the efficiency of these tasks—ROVs need to be replaced by Autonomous Underwater Vehicles (AUVs). However, a difficult assignment falls on AUV software: it needs to navigate in a murky underwater world.
Simultaneous Localization and Mapping (SLAM) is one of the best techniques available for navigating with AUVs. SLAM has been used in many different scenarios, from augmented reality to assisted surgery. One of the most challenging environments for SLAM is underwater (Guth et al., 2014). The literature describes a variety of scenarios, from coral reefs (Williams et al., 2000) and ship wreck explorations (Williams et al., 2016), to surveys for the oil and gas industry (Shukla and Karki, 2016) and inspecting the hulls of ships (Kim and Eustice, 2013).



(a) Example of ultra short baseline.

(b) Example of long baseline.

Figure 1.1: Differences between underwater acoustic positioning methods. Taken from Kongsberg (2016).

1.1 Background

Research has suggested that SLAM can be divided on the basis of sensor combinations (Kim, 2012). For example, according to Yuan et al. (2017), as of 2017 the most popular solutions to underwater SLAM include filtering approaches which predominantly use imaging and side-scan sonars. Filtering techniques have a large number of disadvantages; from growth in complexity in particle filters, to Gaussian assumption and computational slowness in high dimensional maps. Yuan et al. (2017) suggested an improved Extended Kalman filter (EKF), with an augmentation state. The algorithm is based on features derived from stationary landmarks. Similar to the use of QR codes for the localization in indoors environments (Zhang et al., 2015).
Underwater SLAM can use another sensor combination consisting of cameras and/or laser scanners. Due to the rising popularity of these sensors, SLAM has faced two main issues: scalability and reliable data association (Kim, 2012). Furthermore, it is complicated by many camera options available, including stereo (Mahon, 2008), RGBD (Anwer et al., 2017), and monocular (Jung et al., 2016).
Manzanilla et al. (2019) showcased promising results using PTAM (Parallel Tracking and Mapping), techniques using monocular camera and Extended Kalman filter, to fuse data from an inertial measurement unit (IMU), a pressure sensor, and a magnetometer.
Aulinas Masó et al. (2011) investigated the combination of SLAM techniques in an offline context, where data was acquired using an AUV equipped with a downwards-facing camera, Doppler Velocity Log (DVL) [1], and IMU. A similar sensor approach has been shown by Jung et al. (2016), where they used keyframes and then, by applying least square optimization, obtained the final pose graph. The experiments were run in a basin and the final algorithm was compared to Monte-Carlo localization

[1] DVL acquires velocity measurements with respect to the sea floor.

and other inertial navigation systems.
Eustice et al. (2005) applied Information Filter SLAM on data acquired from the monocular camera of a ROV that visited the RMS Titanic. For the underwater vehicle to be capable of moving around in unknown environment without causing collisions, it requires the use of some SLAM methodology.

1.2 Problem Domain & Motivation

This paper focuses on AUVs and their inspection of underwater oil and gas structures. This issue has been emphasized by the stakeholder DeepOcean AS. Their goal is to have a resident subsea AUV in the vicinity of man-made structures that pump gas and oil at a depth of around 500 m. under the Norwegian sea. The company wants an underwater garage for the AUV, so it can continuously inspect the area and create a map of it for further visual inspections. The area requiring inspection may expand up to $15\,000 \times 50$ meters.
This is a large area, which means that while the AUV travels from one structure to another for inspection, it may create a map of the seafloor which is not needed. As a consequence, the AUV is consuming the scarce battery power and computer power on a task that does not produce value. The AUV should thus be able to detect the desired structure, to increase the quality of the map it creates. Furthermore, underwater SLAM has only been solved in theory, as the subsea environment brings many challenges with it (Yuan et al., 2017), such as:

- Sensors. Generally, this is a problem for all robotics solutions. Sensors have limited accuracy. Accumulated noise causes substantial errors when estimating localization and mapping, which usually means that the functions do not converge. Another problem is a sensor's unexpected collision or misalignment, for example, when a camera might abruptly change its angle, resulting in a motion blur (Younes et al., 2017).
- Feature extraction. SLAM requires features that can be easily and repeatedly observed. This aids in reducing uncertainty in its estimations. Furthermore, recognizable landmarks make data association processes much faster and less prone to error. In underwater scenarios, feature extraction may face such problems as occlusion, seafloor reflection, poor resolutions, and water turbidity (see fig. 1.2).
- Absolute location. Due to a lack of global positioning system, underwater vehicles require alternatives to be able to calculate their absolute locations. This is commonly done using triangulation systems, e.g., Long Base Line (LBL), Short Base Line (SBL), Ultra Short Base Line (USBL). These solutions have a few significant drawbacks: implementation is extremely expensive; some solutions (like LBL) require installation in the sub sea environment; and lastly, they restrict the working area.
- Computational complexity. SLAM computational complexity largely depends on the size of the exploration mission and its uncertainties, as the area may grow exponentially.
  In order to address the AUV's first challenge, i.e. the inspection of the correct man-made object, the robot needs to be able to classify the images/objects

it senses. The goal of classification is to take an input vector and assign it a discrete class. But in order to obtain the classifier, the model requires training. The usual approach to construct such a model is by using supervised learning, which requires correctly labeled data. In the case of a neural network technique, the optimal configuration of the neural network is not a known *a priori*. Furthermore, training times may take a long time, and the reasoning behind the final model is not clear (Oltean and Dioşan, 2009). Genetic Programming (GP) based techniques have an edge over some other known statistical approaches due to having no requirements for prior knowledge about the distribution of data. Additionally, interpretability is a preferred characteristic of GP. GP automatically selects features and different mechanisms to control the size of the classifier (Kishore et al., 2000). Finally, GP is efficient for parallel and distributed implementations, which reduces training time. Due to these advantages, this paper explores GP-based image classification.

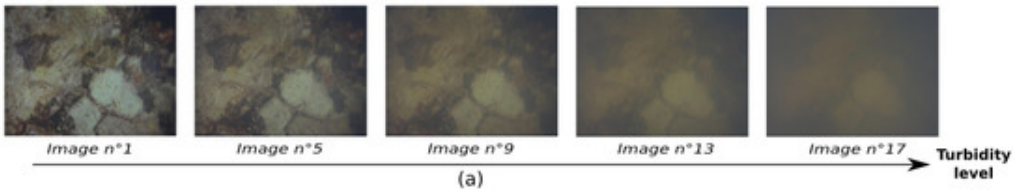


Figure 1.2: One challenge affecting detection of underwater objects is water turbidity. Example from Ferrera et al. (2019).

1.3 Research Goal

This paper has two goals. The first is an extensive literature review of SLAM algorithms based on the stakeholder's requirements—establishing an AUV in a subsea garage for monitoring oil and gas pumping structures. This is also called resident AUV, as the vehicle stays underwater even when not in use. The second goal is to examine a new method for image classification in the underwater world, using convolutional genetic programming and incorporating it into the chosen SLAM algorithm. It was not possible to use the actual location data, as it is very client sensitive. Image classification is therefore achieved by recreating one part of the real world subsea environment in an underwater simulator. This new approach to image classification is then evaluated by running the trained model on unseen labeled data and calculating the confusion matrix followed by other metrics from it.

1.4 Contributions

The aim of this paper is to review current SLAM techniques based on stakeholder requirements and contribute with an implementation of a new approach for image classification based on convolutional linear genetic programming. The data for algorithm's test is acquired in Robot Operating System (ROS) software.

1.5 Scope

The industrial stakeholder's goal is to be able to have a resident AUV available to perform inspections at any given time. This also means that no USBL or LBL should be used during the inspections.
Due to privacy of the data, the stakeholder provided a 3D cloud point of the underwater structure with identifying numbers removed. The subsea template with its surrounding ground was extracted and placed into a simulation. UWSim, a package in Robot Operating System, has been chosen as the software simulation environment. The software is open source and integrates underwater dynamics.

1.6 Section levels

This paper is structured as follows. The second section introduces the reader to basics of simultaneous localization and mapping (SLAM) and establishes the common paradigms in solving SLAM. Additionally, the second section provides an overview of SLAM implementations and summarizes them in an appendix. The third section talks about the simulations used for subsea robotics. The fourth section presents the main algorithm used for image classification which is followed by the methodology. The last two sections develop results and conclusions from the experiments.

2

Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM, also known as concurrent localization and mapping) is one of the biggest challenges in robotics (Imani et al., 2018). It describes a problem where a robot moves within an unknown environment while at the same time making observations of the environment (Durrant-Whyte and Bailey, 2006). SLAM processes can be divided into the two main steps of localization and mapping, but at first both of these steps had to be solved separately. Fortunately, due to the proposal of Durrant-Whyte and Bailey (2006), SLAM problems evolved into the first comprehensive theoretical SLAM solution.
In this paper, the focus is on a single camera SLAM that can be fused with data from other possible sensors, e.g. inertial measurement units (IMU), doppler velocity logs (DVL), altimeters, and sonars.

2.1 Basics

The focal interest of this paper is to examine Visual and Visual-Inertial data (camera data fused with inertial sensor data) and SLAM solutions applicable to the problem posed by the stakeholder. Before digging into the details of SLAM, it is important to understand the basics.
In order for the robot to be able to apply SLAM, it requires motion and at least one sensor, in this case a monocular camera. SLAM is a probabilistic solution, therefore the probability distribution requires it to be calculated for all times k:

$$P(x_k, m | Z_{0:k}, U_{0:k}, x_0) \tag{2.1}$$

where:

- x_k is the state vector describing the location and orientation of the robot (in this case it is *x, y, z,* θ). *x, y z* are the robot's coordinates and θ is the angle the robot is facing.
- u_k is the control vector which is applied on the robot to drive it from state x_{k-1} to x_k.
- m_i is a vector describing the locations of the ith landmark.
- z_{ik} is an observation of the ith landmark taken at time k at some location.

SLAM utilizes Markov process. Thus due to it, the motion model's next state transition depends only on the previous state x_{k-1}. Therefore, the motion model

for the robot consists of finding the current state, based on the previous state and the motion at the current time k:

$$P(x_k|x_{k-1}, u_k) \tag{2.2}$$

The last step is the observation model (z_k) for the vehicle, which given a map calculates the probability of sensing landmarks at the current state x_k. The observation model is described in the following form:

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

This means that the probabilities for the observations are calculated by the given state vector and described landmarks locations.
The next subsections go into more detail about the fundamentals of SLAM.

2.1.1 Mapping

Mapping is a part of the challenge of SLAM, as the vehicle has no prior knowledge of the environment in which it travels, and its sensors provide noisy data. The former results in the imprecise pose estimation of landmarks. Landmarks are features in the environment that are easily recognized in the distance and help to establish location. Figure 2.1 shows that the estimation of each landmark is not certain, and is thus surrounded in an ellipse of possible poses.

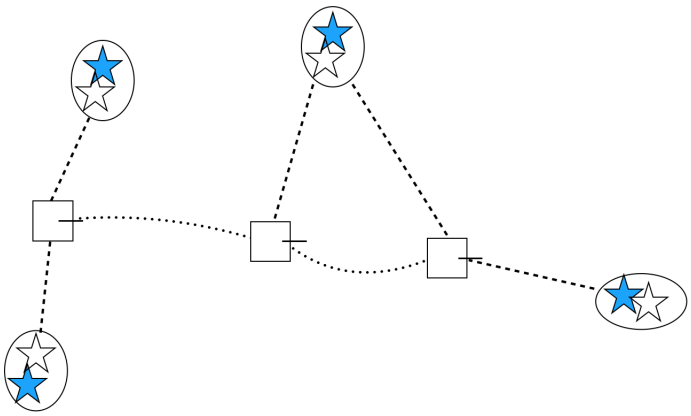


Figure 2.1: Mapping example from Imani et al. (2018).

2.1.2 Localization

Localization is another feature of the SLAM algorithm, which is employed to estimate vehicle's pose. When the vehicle moves through unknown environments at discrete time intervals, it utilizes data from the mapping process. As can be seen from the figure 2.2 below, the vehicle moves along a trajectory and utilizes its sensors to recognize the environment and its landmarks. Since the vehicle knows the approximate pose of landmarks, it can estimate its own pose in relation to them. The example in figure 2.2 portrays a situation where a vehicle (square) started off by calculating its pose correctly according to the most left stars/landmarks. However,

later on, the actual motion model provided poor advice for the vehicle to turn left, when in reality it should turn right. But, by observing the landmarks, the vehicle was capable of correcting itself.

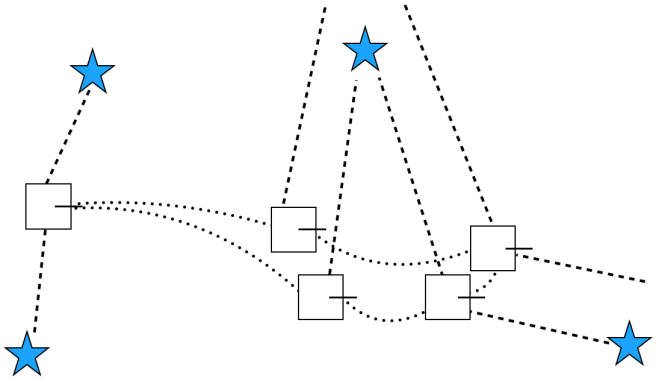


Figure 2.2: Localization example from Imani et al. (2018).

2.1.3 SLAM

The main idea behind SLAM is to allow the vehicle to build a map while at the same time localizes itself within the map (see fig. 2.3). This is usually referred to as the chicken and egg problem, since the vehicle needs to know the map in order to be able to keep track of its location.

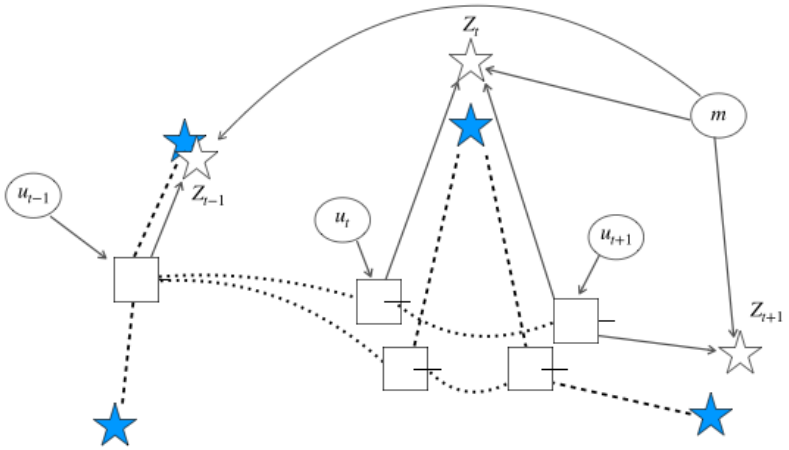


Figure 2.3: SLAM example from Imani et al. (2018).

Due to the nature of stakeholder's proposed problem, this paper focuses on Visual and Visual-Inertial SLAM. This is because that after leaving the stationary underwater garage, the AUV may be swept away by underwater currents, a possible landslide, or other underwater disturbances. Visual and Visual-Inertial SLAM is a different category of general SLAM. As the name suggests, its main sensor is a camera or multiple cameras. Different researchers have been using discrete camera setups to solve Visual SLAM problems. This includes, but is not limited to, the monocular camera presented in Mur-Artal and Tardós (2017a), RGBD in Sturm

et al. (2012), and a stereo camera in Engel et al. (2015). Visual SLAM can be supplemented by other sensors, for example, inertial navigation systems, which aids in increasing the accuracy of the algorithm. The sensors complementary to Visual SLAM are called Visual-Inertial SLAM (Fuentes-Pacheco et al., 2015).

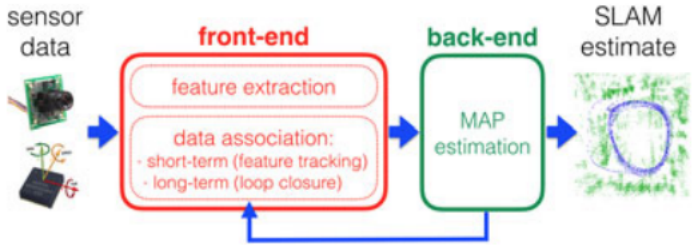


Figure 2.4: Front-end and back-end of SLAM from Cadena et al. (2016).

Visual-Inertial SLAM (later on referred as just SLAM) has been researched a lot in the last decades. There are many ways to divide it into different systems, strategies or parts. Firstly, the most common separation of SLAM is into front-end and back-end, following Cadena et al. (2016) (see fig. 2.4). As mentioned above, the first step in SLAM is extracting sensor data and matching it to the vehicle's appropriate state. Since in Visual and Visual-Inertial SLAM the main sensor is a camera, extracting the most important data and representing it in the vehicle's state is challenging. Therefore, front-end plays the role of extracting the most crucial features from the environment and presenting it in a way understandable to the back-end. For example, by taking pixel locations of the most important features in the viewpoint, it allows the vehicle to further send this data to back-end. Furthermore, front-end aids in associating sensor data to specific landmarks in the present environment. This step is also called data association. Back-end acts as a process where the vehicle estimates its pose and landmark locations (i.e., creates the map). The loop between back-end and front-end is there because the back-end provides feedback on the features to the front-end for loop closure Yuan et al. (2017).

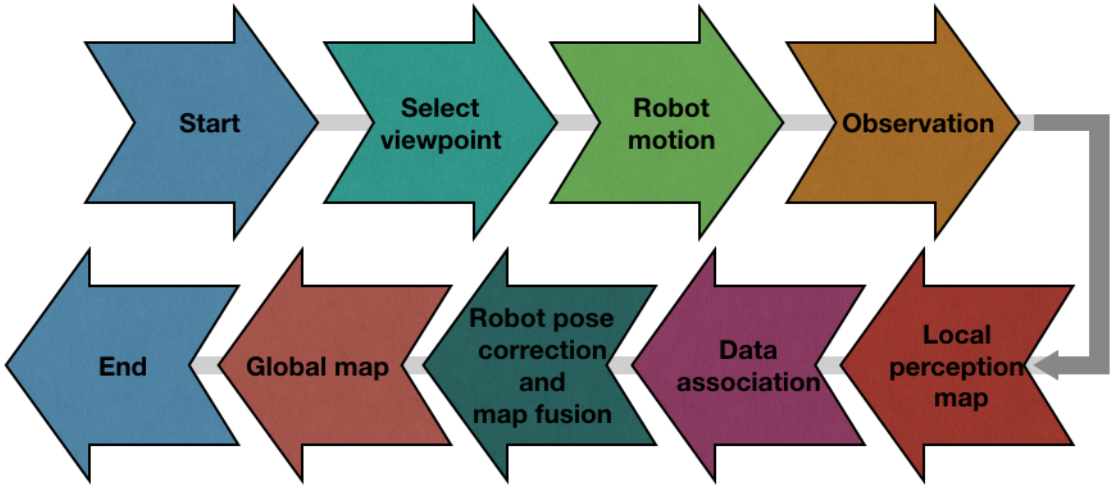


Figure 2.5: The flowchart for SLAM, based on Yuan et al. (2017).

The common flowchart of the SLAM process is shown in figure 2.5 based on Yuan et al. (2017). SLAM consists of seven main steps. Works by others have reduced the number to five fundamental steps (Naminski, 2013). These are as follows, excluding the start and finish:

1. *Select viewpoint.* This step allows the vehicle to focus its camera on some object.
2. *Robot motion.* This step is also called location prediction, as it stores the information of the vehicle's control into a vector. The control vector collects the information of the different controls affecting vehicles. This vector usually also provides knowledge of the vehicle's orientation and changes over travelled distance.
3. *Observation.* The observation phase includes gathering data of the observed environment. It also contains a matrix of the calculated landmark's pose.
4. *Local perception map.* After the vehicle has collected data of the environment, it is substantial enough to estimate the vehicle's local pose using triangulation. In this step the vehicle can also adjust its location.
5. *Data association.* The vehicle attempts to associate previously observed landmarks with those it currently observes.
6. *Robot pose correction and Map fusion.* Through triangulation of the observed maps and the vehicle's motion model, the vehicle is capable of computing the probability of its current pose. Without this step, the vehicle can neither map nor localize itself since it cannot use any previous information about its pose.
7. *Global map.* This final step estimates the landmarks' poses. The global map step also contains the process of calculating correlations between landmarks. After this, the map of environment is updated.

2.2 Principal paradigms to solve SLAM

The main challenge of SLAM is uncertainty, which makes SLAM a probabilistic problem. The uncertainty comes from noisy sensor measurements and difficult environments with many factors affecting a vehicle's localization and mapping making the problem even more demanding.
One approach is to pick one frame at a time from a video stream and estimate the camera poses with respect to the 3D structure obtained so far.
SLAM can be divided into the following strategies: incremental (picks one frame and estimates the camera pose by triangulation with respect to observed landmarks), hierarchical (uses balanced trees obtained from trifocal lenses gathered over a video frame) and global (also referred to as batches) approaches (Patra et al., 2017). According to Younes et al. (2017), monocular SLAM solutions are grouped into and based on either filter or key frames. The former uses filters such as Kalman, including its variations, e.g., Extended Kalman Filter and Unscented Kalman Filter (Yousif et al., 2015). As stated by Younes et al. (2017), filter-based SLAM was most popular before 2010. Later on the non-filter, also called keyframe-based SLAM, had greater popularity. The filter-based systems update the camera pose and the locations of all landmarks in the map for every handled frame. While key frame based systems update camera pose over a small subset of the map, this is where

the optimization takes place. According to Younes et al. (2017), key frame based SLAM solutions outperform filter based ones. Thus, it is not surprising to see most up to date research is based on key frame based SLAM. This section of the paper digs deeper into the discussions of different SLAM techniques and explores them.

2.2.1 Filter-based

One of the solutions for SLAM uses filters. The most popular filters are Extended Kalman Filter and Particle filter.

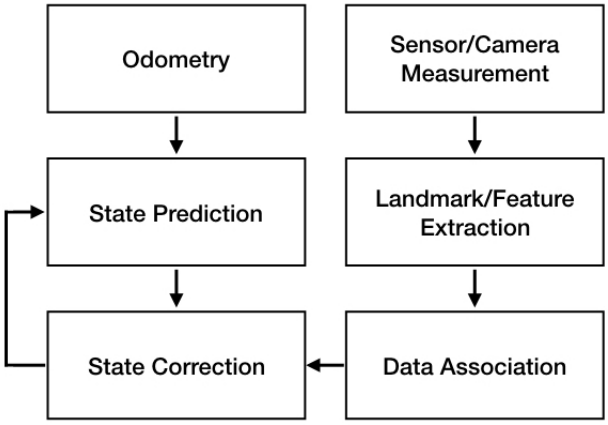


Figure 2.6: Flow chart of filter based SLAM based on Yousif et al. (2015).

(a) The Extended Kalman Filter's (EKF) foundations are based on the Bayes technique. It computes the probability distribution of the state estimates by using a recursive equation, which permits combining the control input and observation model. EKF has two steps; the first one is called prediction. During this step the vehicle takes measurements from a motion model. This process allows it to propagate the robot state estimate. In other words, the predicted belief keeps information about the mean and the covariance matrix of the state. The mean estimate is calculated by using a nonlinear function g. The covariance matrix computes uncertainty by using a Jacobian of the motion model and adding the noise that has occurred during motion.
The second step, also called correction, takes into consideration the observation measurements and includes observed landmark pose predictions. This step also uses the so called Kalman gain which is the correction of the mean estimate and the covariance matrix. The correction step for the mean multiplies the Kalman gain by the observation model and the predicted observation. The last equation calculates the correction of the covariance matrix.

(b) The Particle Filter algorithm has been derived to also be used for nonlinear motion models. The algorithm is also part of the recursive Bayes filter, along with the Extended Kalman Filter. The idea behind Particle Filter is to use random weighted samples that show the possible pose estimates of the robot. Particle Filter consists of three main steps. The first step is called sampling

step. During this step the algorithm produces random samples according to the arbitrary distribution. The second step computes the importance weights for the proposed particles. This includes taking the target distribution and dividing it by the proposal distribution. In other words, it considers differences between the proposed distribution and the target. The final step swaps the most likely particles with less probable ones.

(c) Graph-SLAM. The idea behind Graph-SLAM is to use graphs to represent the problem, where each node corresponds to a pose and edge shows a spatial constraint between them. While the vehicle moves, it generates constraints between successive poses. This means that Graph-SLAM is an optimization problem where it needs to find the proper node configurations that minimizes the introduced constraints.

2.2.2 Keyframe-based

Keyframe-based SLAM techniques focus on optimization in order to be able to estimate the motion and map. In contrast to filter-based solutions where they perform both localization and mapping at the same time, keyframe-based techniques separate them into steps. These steps include camera pose estimation and key frame optimization.

Keyframe-based SLAM consists of 7 components: visual initialization, data association, pose estimation, topological/metric map generation, map maintenance, failure recovery, and loop closure (see fig. 2.7).

According to Younes et al. (2017), in keyframe-based SLAM the camera pose and the structure of environment are unknown during initialization. Therefore, the first step for initialization is to initiate these two estimates. With each new key frame, data association uses the previous camera pose to estimate the new pose. Then the new estimate is used for establishing a 3D map and its associations. The error vector calculated by the difference between the true measurement and the matched pose is iteratively minimized. If the data association step fails, failure recovery boots up. If the frame was chosen as the key frame, then it is used to triangulate landmarks between two key frames and create the map. Map maintenance optimizes the map by removing outliers and detecting loop closures. The keyframe-based components in detail are as follows.

(a) Visual initialization is a challenging start of the keyframe-based SLAM techniques. The main problem arises from the fact that neither pose nor structure is known at initialization. Therefore, the usual tactics to solve this problem are as follows; the first frame is always set as the key frame. The subsequent frames monitor if the data associations between the frames have reached some distance. Usually Homography [1] or Fundamental matrix [2] are used to estimate initial camera pose and scene structure. Sometimes random depth initialization is used to address the degenerate cases. This is done by assigning random depth values and updating them with each frame until depth variance converges.

[1] Used for relating translations between two planes.

[2] Used for relating corresponding points in stereo images.

(b) Data association can be direct, feature-based, and hybrid. Further down, direct data association is separated into dense and semi dense. The formal direct data association method looks at every pixel while the semi dense method looks at the most significant gradients of the image.
The direct data association looks at a window of a location of interest in the source image, and tries to find the transformation that minimizes the same window of a location of interest in the target image. The fundamental idea behind the direct method is called brightness consistency. It is based on the idea that the location of interest in two consecutive images will have similar brightness intensity.
The feature-based methods are advantageous compared to the direct data association methods, as their aim is to reduce computational costs. Instead of looking at windows of location of interest, the feature-based methods introduce the idea of matching salient image locations. Sometimes these locations are called features or keypoints. The features must be distinctive and not be affected by illumination, blur or image noise problems. Since computational speed and a high resolution of features are difficult to solve, researchers have shown that a trade-off is a usual practice in SLAM. Feature detectors are algorithms that allow to extract the most interesting parts of the image. The feature detectors examples include but are not limited to Hessian corner detector, Harris detector, Laplacian of Gaussian detector, and FAST. Feature descriptors, as the name suggests, describes the extracted features from the previous step. Examples of feature descriptors can be as simple as raw pixel values or can be represented as a bag of words, local orientation, etc. Researchers have worked on this problem continuously over the past decades, with many proposed representations for feature descriptors such as SURF, SIFT, HoG, and ORB.
The last data association method is a hybrid of both feature-based and direct-based methods. They use a combination of both to update camera estimates and create maps.

(c) Pose estimation allows to make assumptions and make the data association step easier. Most systems use the constant velocity motion model, thus the prior for the two consecutive frames stays the same. Having a good estimation of the pose allows the system to establish better estimates for feature locations and plays a role as a basis for the minimization procedure.
Camera pose optimization is done by minimizing the measure error between frames. These frames are achieved by projecting a 3D landmark into a 2D frame.

(d) A map generation/expansion process is responsible for generating a map for an unseen environment with unknown landmarks. Usually they are represented as dense or sparse cloud points. Every time a new key frame is added to the map, the data association is performed between the new key frame and the neighbouring key frames. Then triangulation is executed with new landmarks. In case the map has pose graphs, it updates the pose graph's links. Otherwise, the information is forwarded to the map maintenance module.
The options for maps in SLAM techniques are twofold; metric and topological.

Metric maps are created by projecting the 3D landmarks between two or more key frames using epipolar geometry. It is done by associating 3D landmarks projected into 2D by minimizing the sum of all projections. Topological maps use nodes and arcs. These maps allow distorting point poses without changing spatial relationships between the nodes.

(e) Map maintenance is achieved by map optimization through bundle adjustment or pose graph optimization. This step is important, as it aids in minimizing the accumulated camera pose errors. Map maintenance begins by establishing data association between either all landmarks and key frames or just a batch. This step determines if a global or local bundle adjustment is going to be used.

(f) Global localization is a required step which is needed when the camera loses its pose track and needs to localize itself in the global map. Two forms are considered to be part of global localization; failure recovery and loop closure. Failure recovery is usually a problem when the camera loses sight of the feature. Loop closure allows reducing the accumulated drift error when and if the loop closure process comprehends that the same feature has been observed again. This is usually done using pose graph optimization or bundle adjustment.

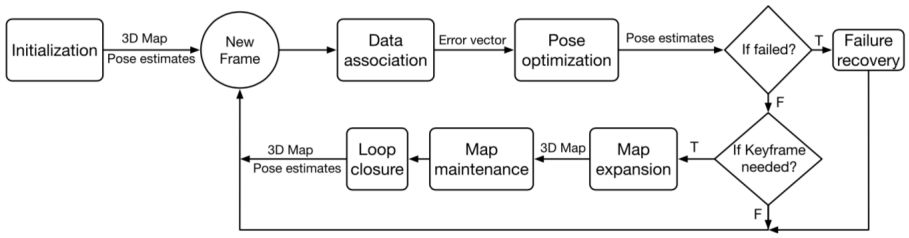


Figure 2.7: Flow chart of key frame based SLAM.

2.2.3 Overview of SLAM implementations

In order to be able to solve the problem of allowing the AUV to leave the underwater garage, identify a man made sub-sea structure, e.g., a template (see fig. 2.8), navigate towards it, do a survey around it, and come back again to the garage with a 3D map of the said structure, it is crucial to choose the proper SLAM method.

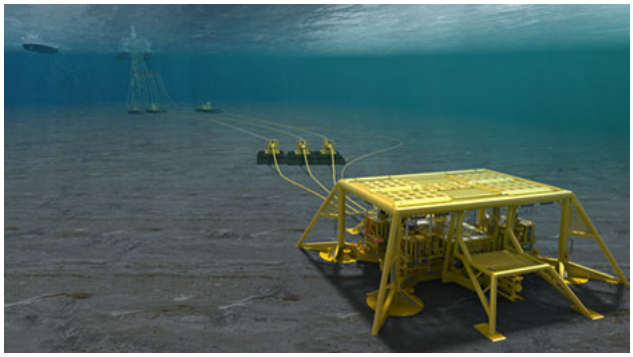


Figure 2.8: Example of an underwater template taken from Engineering (2016).

The main requirement for the SLAM technique is to ensure that the sensor used is a monocular camera that also has the possibility to incorporate IMU and possibly other sensors in the future. The second reason is to ensure that the SLAM is a complete SLAM technique, as in this case we are not interested in separate front-end (e.g. Direct Sparse Odometry) or back-end (e.g. RatSLAM) systems of SLAM. Furthermore, in order to solve the problem proposed by the stakeholders, SLAM requires to have a loop closure. Finally, it needs to be open source. Following these requirements, the following table has been created (see table A.1) based on work by Kahlefendt (2017).

2.2.3.1 DolphinSLAM

Researchers have long questioned just how living beings navigate through the world. By taking their studies into account, a new development of SLAM techniques, called DolphinSLAM, has come to the surface (Silveira et al., 2015). DolphinSLAM has been inspired by another animal related SLAM technique called RatSLAM, developed by Milford et al. (2004). DolphinSLAM is purposely built to solve SLAM problems in underwater environments. Input is passed through a Continuous Attractor Neural Network which computes the robot's pose estimation in unseen environments.

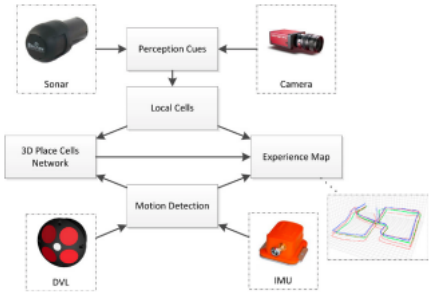


Figure 2.9: DoplphinSLAM architecture by Silveira et al. (2015).

DoplphinSLAM has the advantage of linear computations and its capability to involve optical and acoustic image sensors. These sensors are: sonar, camera, Doppler Velocity Log [3], and IMU (see fig. 2.9).
A Perception Cue module is responsible for extracting important information from the sensors. A Hessian feature detector is applied on optical sensors measurements, followed by SURF descriptors. Sonar image registration uses the HU image moment calculation, as it allows to determine clear geometric shapes. All vehicle perceptions are mapped into neurons called local view cells. The FabMAP algorithm applied over of these neurons allows it to determine if the environment is being seen for the first time or if it has been visited before. At each moment only one local view cell is activated. A motion detector module is responsible for extracting information from IMU and DVL sensors. The 3D place cell network is a Continuous Attractor Neural Network. Each neuron is accountable for only one specific area in the environment. Each neuron is then passed to a competition of the most reliable vehicle's pose estimation. The winner neuron will become an attractor, where the main focus of

[3]DVL acquires velocity measurements with respect to the sea floor.

the Continuous Attractor Neural Network is condensed. An experience map module is in charge of creating the environment map. Every time a new place is visited, the experience map creates a new node.

2.2.3.2 DT SLAM

DT SLAM was created to unify the ability to triangulate points (Herrera et al., 2014). It has been tested on videos taken in cities by a moving robot arm. It consists of three main parallel threads: tracking, mapping, and bundle adjustment (see fig. 2.10).
The tracking module is responsible for feature matching. It uses the previous pose estimation to be able to track the feature matching. The originality of the tracking module is the use of 3D and 2D features as both of them provide information to a better pose estimate.
The mapping thread checks if a new keyframe is contributing to the map creation. Firstly, it checks the new keyframe against all possible matches. If a feature has been found, it needs to ensure that it is over some baseline threshold. Lastly, the mapper thread updates the pose of the new keyframe and the locations of the features using 1-frame bundle adjustment.
The bundle adjustment thread takes care of keyframe optimization as it runs uniformly in the background. Keyframe optimization is achieved by taking both 2D and 3D features into consideration using the same framework as in the tracking module.

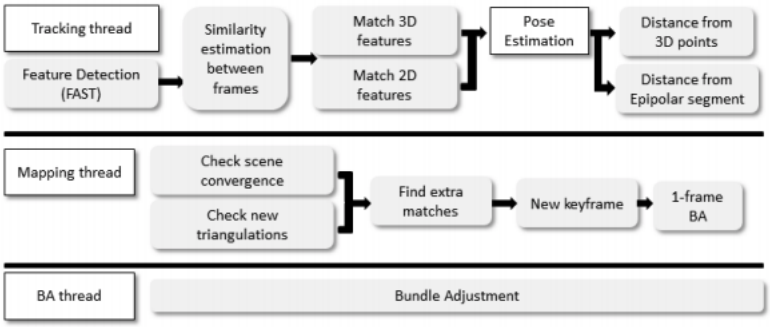


Figure 2.10: DT SLAM architecture.

2.2.3.3 FAB MAP

Fast Appearance-based Mapping algorithm was created by Cummins and Newman (2008) and further extended by Cummins and Newman (2011) to enable this probabilistic appearance-based place recognition SLAM technique to work on maps of thousands of kilometers. The idea behind FAB MAP is that it compares images and computes the probability for re-visiting a location seen in one of the images. It also provides the probability for seeing the new location for the first time. FAB MAP has a feature-based data association using a visual bag of words database. It thus requires to train its model on images similar to those it may encounter while traversing the unseen map. The system uses the SURF descriptor/detector. FAB MAP was tested on imagery collected by a mobile robot in a neighborhood.

2.2.3.4 EKF MonoSLAM

The Extended Kalman Filter Monocular SLAM technique has been extensively researched by Grasa et al. (2011). In order to achieve SLAM using EKF operations, it is important to map events to SLAM and EKF actions. For example, the robot motion maps to the EKF prediction step, while the camera's detection of a new landmark portrays landmark initialization and state augmentation in EKF SLAM. If a camera detects a known landmark, that means in EKF SLAM it goes into the next, Kalman filter correction, step where it adjusts the position of the observed landmark. Finally, if the mapped landmark is deprecated, the EKF has a module called state reduction to take care of it.
EKF Monocular SLAM has been implemented both indoors (Hwang and Song, 2011) and outdoors (Herrera et al., 2014), and even to aid surgeons (Grasa et al., 2011).

2.2.3.5 LSD SLAM

The Large Scale Direct SLAM technique is based on using direct data associations, as opposed to the previous feature-based techniques (Engel et al., 2014). This means that the full input image is kept by using photometric error minimization, also known as intensity difference, which allows calculating per-pixel depth to create a semi-dense depth map. The idea behind LSD SLAM is to reduce computational costs when it comes to extracting and matching features (see fig. 2.11).
By using a semi-dense depth map on keyframes it allows the reconstruction of whole images. LSD SLAM process goes as follows: a structure in the input video is tracked by using SE(3) alignment to current keyframe. A frame is only considered to be the new keyframe if the tracked structure is no longer visible in the input image. In that case, the depth map is propagated on the new keyframe. If the previous keyframe is not replaced by a new one, the former is refined by adjusting its depth map. Each new keyframe is incorporated into the map and is checked against previous keyframes of a loop closure. Map optimization is achieved by using g2o pose graph-based framework.

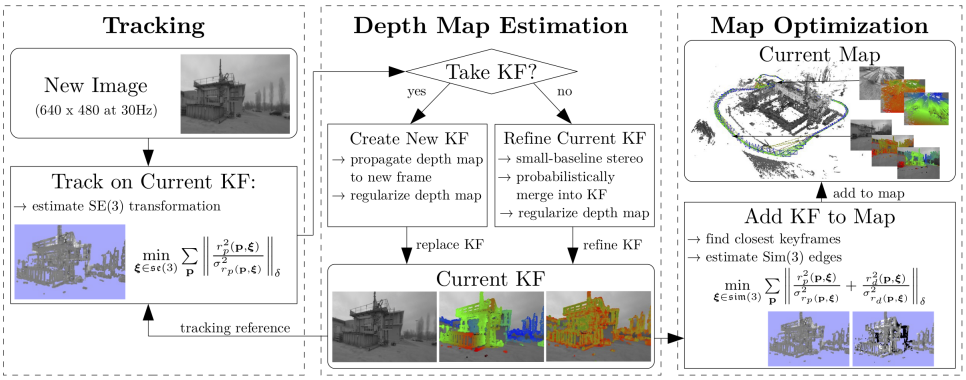


Figure 2.11: Flow chart of LSD SLAM.

LSD SLAM's achievements have been displayed in a few scenarios. One example is by von Stumberg et al. (2017) where the authors have demonstrated LSD SLAM working on an autonomous drone by using a monocular camera.

2.2.3.6 ORB SLAM

Oriented FAST and Rotated BRIEF SLAM is an efficient and successful alternative to the closed source SIFT and SURF data association types (Mur-Artal et al., 2015). ORB SLAM adopts feature based data association, where the feature type is FAST and feature descriptor is ORB (see fig. 2.12).
ORB SLAM starts with map initialization where it keeps information on the map points and keyframes. Map points contain details on keyframes that were observed from a 3D point, while keyframes store data on camera pose, its parameters and ORB features. ORB SLAM extracts corners using FAST through eight pyramid levels. The descriptors are stored into a bag of words. Criterion to add a new keyframe depends on the significance of the scene appearance change. One of ORB SLAM's advantages in visual initialization is the fact that it calculates both Fundamental matrix and Homography in parallel. In case the tracking shows poor results and not enough feature correspondences, the visual initialization is disposed of.
Pose estimation is established through a constant velocity motion model. ORB SLAM detects matched features among keyframes. ORB SLAM has shown successful results in large environments. This is due to utilization of subsets of the global map. These subsets are called local maps and they consist of keyframes that share edges.
Map maintenance is attained by using local bundle adjustment. It requires using two topological and one metric map. One of the topological maps is called a co-visibility graph that contains information in the arcs between edges. This information includes the visibility of keyframes among each other. The other topological map only allows two arcs per edge which display the strongest visibility of keyframes.
ORB SLAM keeps a database of the keyframes, which allows to compare a keyframe against other ones to find loop closure. Instead of searching all databases, the loop closure module only looks at the edges of the co-visibility graph.
This SLAM technique has shown promising results in both indoor and outdoor situations (Strasdat et al., 2010). For example, Mur-Artal and Tardós (2017b) implemented monocular ORB SLAM with an incorporated IMU in an aerial drone.

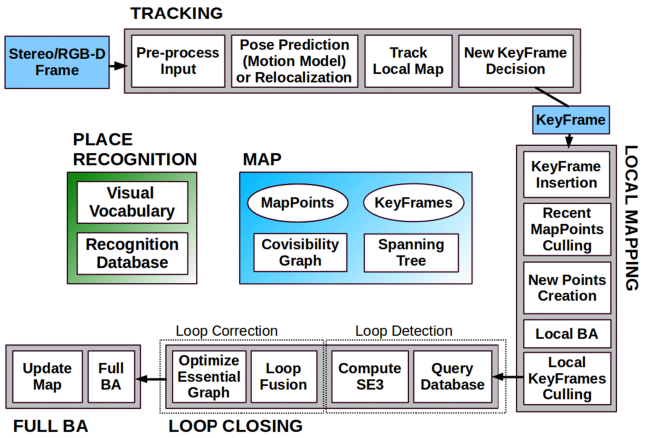


Figure 2.12: Flow chart of ORB SLAM. Works with monocular camera as well.

2.2.3.7 Pop-up SLAM

Pop-up SLAM developed by Yang et al. (2016) is used for low-texture scenes or also defined as scenes that do not have many features. The presented SLAM technique has been tested in indoors corridors. The main idea of this SLAM technique is to utilize monocular sequences and create a single pop-up (a rough 3D model of the scene). This SLAM technique achieves it by detecting the edges between ground and walls using a Convolutional Neural Network and estimates the camera rotation. Due to SLAM's general under-constraint nature, Pop-up SLAM employs LSD SLAM. This is due to dual reasons; firstly, LSD allows to estimate depth which is much needed for the 3D model. Secondly, the poses provided by the LSD aid Pop-up SLAM to constraint odometry.

Figure 2.13: Example of Pop-up SLAM.

2.2.3.8 RKSLAM

Robust Keyframe-based SLAM was initiated with the idea to develop a robust, monocular SLAM able to withstand strong rotation and fast motion (Liu et al., 2016). This is achieved by two main changes compared to LSD SLAM and DT SLAM; firstly, RKSLAM uses multihomography based feature tracking model for 3D points; and secondly, unlike other keyframe based systems, RKSLAM executes the local map expansion and optimization in the main thread which aids in handling motion and rotation. RKSLAM extracts multihomography; global, local, and plane homographies (see fig. 2.14). The latter two functions help align the local regions of the image while the global one aligns the whole image.

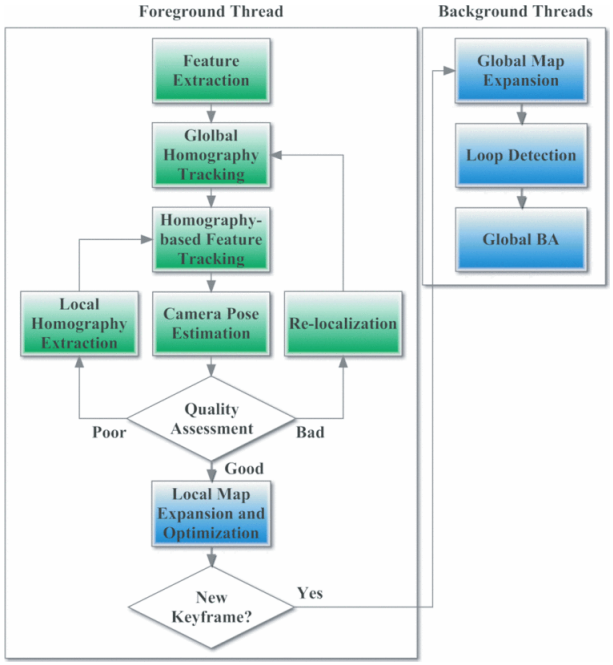


Figure 2.14: Flow chart of RKSLAM.

The experiments were conducted on both a PC and a mobile device in indoors and outdoors environments. Neither of the environments included large scale maps.

2.2.3.9 Selecting an Algorithm

As seen from the discussed techniques above, there are many algorithms to choose from, but we must select the one most appropriate to the use case (see table A.1). Strasdat et al. (2010) have compared filter-based SLAM systems against non-filtering systems. They found that filter-based SLAM techniques are better when using hardware within a constrained budget. In the case of this paper, the AUV will have a high-end computer, so we can focus only on algorithms with non-filtering approaches.
The number of applicable algorithms is further reduced by the need for the SLAM to operate in a large, featureless underwater environment. As fig. 2.15 shows (green: successful results, yellow: some failed experiments, red: completely failed trials), in this case only ORB SLAM is capable of delivering positive results. Furthermore, the SLAM framework is required to create and record a map. Additionally, we also require that the SLAM framework be open source, use at minimum a monocular camera with a possibility to include IMU, and incorporate loop closure—The only algorithm that fits all of the criteria is ORB SLAM.

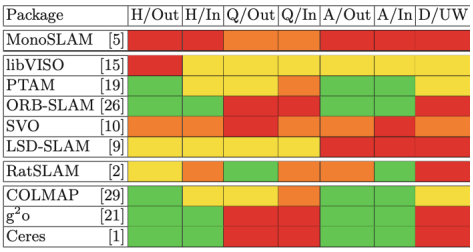


Figure 2.15: Qualitative analysis of open source packages using the new datasets. Data sets: Husky (unmanned ground vehicle) outdoor (H/Out); Husky indoor(H/In); Quadrotor outdoor (Q/Out); Quadrotor indoor (Q/In); Aqua on coral reef (A/Out); Aqua inside wreck (A/In); Drifters on coral reef (D/UW). Taken from Liu et al. (2016).

2.3 Evolutionary Algorithms in SLAM and Image Classification

In order to summarize achievements within SLAM and image classification when applying genetic algorithms/evolutionary algorithms, it is crucial to understand the basics of the evolutionary paradigm. Genetic algorithms (GA) are an artificial intelligence technique based on biological principles of evolution, fitness, crossover, and mutation. GAs allow us to find an approximate solution to a problem, are thus often compared to neural networks. Applied Darwinian principles allow significant improvements to the space of solutions by avoiding the local optima (Banzhaf et al., 1998).
GA solutions depend on the representation of the solution, which may vary from one problem to another. The gene in LGP is an instruction, e.g., $a = b + c$, where a is a register and b and c are operands. The operands can also be constant values, e.g. $a = 2+c$. The constant values are called terminal sets, according to the Koza tableau (Koza and Rice, 1991). The collection of operators that can be selected is known as the function. Examples of an operator can be $+$, -, $/$, $*$, $\sqrt{x}$. Another way is a tree representation, where in the previous LGP example, nodes depict the variables and edges represent the basic operation (addition, subtraction, multiplication, division). A number of instructions compile an individual, while many individuals amass to a population. The chromosome or individual in that case is a number of instructions that make up one solution, e.g., in this case the individual (chromosome) consists of 3 instructions (genes) $(a = b + c, b = a - c, a = 2 * a)$.
Chromosomes or individuals that consist of ineffective instructions, i.e. that have no impact to the final fitness value, are called introns. An example of an intron is $b = 0 + b$. Introns have both an advantage and disadvantage. The disadvantage is that the individual/chromosome may fill up itself with useless instructions just to reach the maximum size. This may lead to the population being stuck in local minimum, longer training times and of course computational costs. The benefit of having junk code is that it protects the important instructions from the destructiveness of crossovers (Espejo et al., 2010).
The next step is to evaluate all these individuals, acquired by calculating their fitness

value. Fitness function is authentic to problem at hand. However, one of the key factors of this process is that it must be fast, since it will be run thousands of times per population.

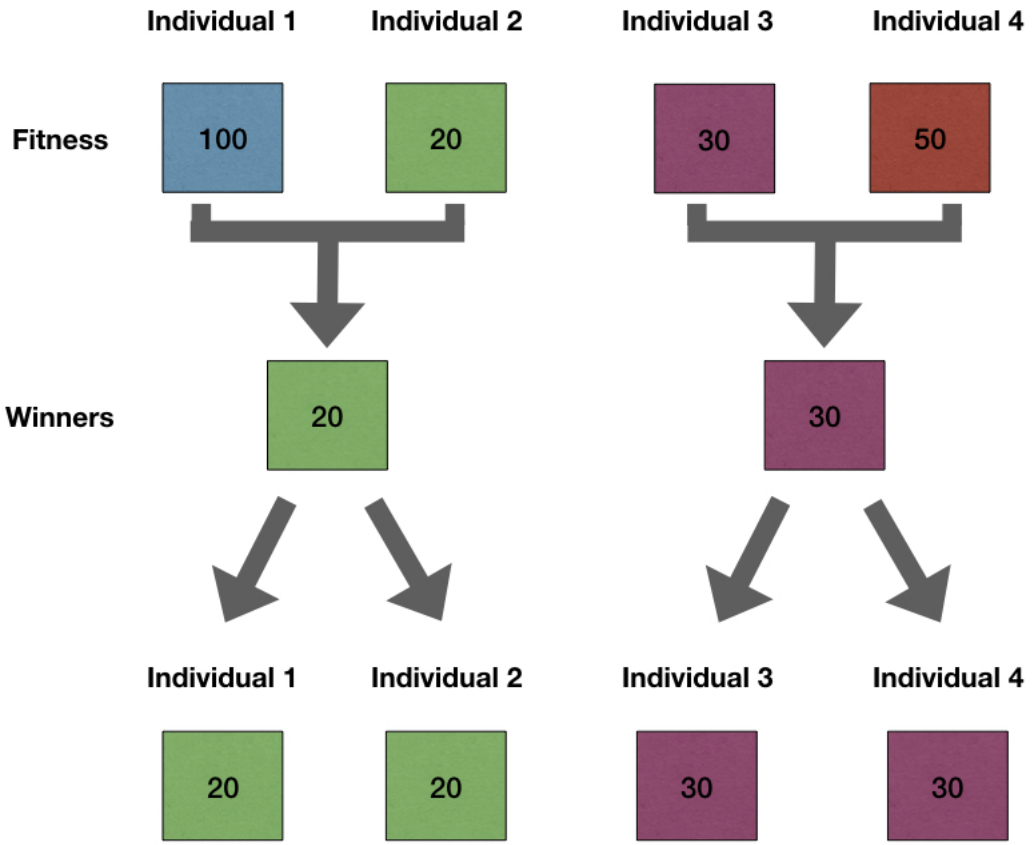


Figure 2.16: Example of a tournament selection. This example uses standard fitness value, which means the best fitness value is zero.

The usual choice for the GAs is a steady-state algorithm, where 4 individuals are chosen from the population. They are evaluated among each other by the fitness function. Then, the 2 highest scoring individuals are copied for further crossover and mutation and the lowest scoring individuals are overwritten with the highest scoring ones (see fig. 2.16). This allows keeping the most fit individuals in the population (Nordin and Banzhaf, 1997).

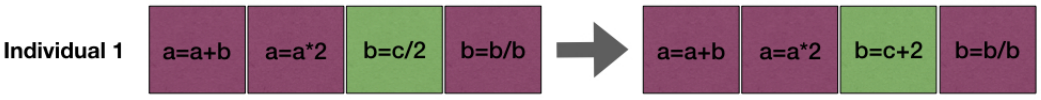


Figure 2.17: Example of a mutation in LGP where an operator changed in one instruction.

Variation operations provide changes in the population. Otherwise the population would not be able to adapt. One of the examples of variation operations is a crossover

where two individuals (sometimes called parents) are chosen and portions of each individual are exchanged (see fig. 2.18.) The idea behind mutation is that any component of the instruction (destination register, operands, and operator) can be swapped for another alternative (see fig.2.17).

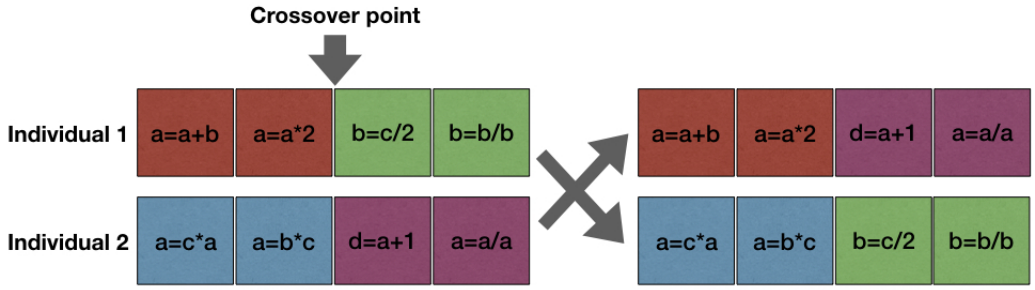


Figure 2.18: Example of a crossover in LGP.

2.3.1 Existing approaches

GAs in SLAM have already been employed. One of the most famous techniques in SLAM, Monte Carlo localization, is a light form evolutionary algorithm (Dellaert et al., 1999). The Monte Carlo localization (MCL) algorithm uses particles (possible locations of the robot/vehicle) randomly drawn from various distributions to show a robot's position. The highest weighted particles have a higher chance of survival for the next check. This looks a lot like evolutionary algorithms, but MCL exploits neither mutation nor crossover. It is therefore not surprising that more particle filter-like techniques, combined with evolutionary algorithms, exist in the research field (Dong et al., 2007).

Researchers in Duckett et al. (2003) proposed a SLAM technique based completely on GAs. Authors suggested to find the best robot trajectory using GA-SLAM which is done by initialising the first population by looking at the odometry data. GA-SLAM then creates an occupancy grid for each individual, and checks its fitness by evaluating its correctness against observation (in this case a laser scanner). If a travelled trajectory is represented by a vector $[\delta_1, \alpha_1, .., \delta_N, \alpha_N]$ where δ_j and α_j depict relative distance and rotations travelled by the robot in a step j, then the candidate individual is portrayed as $[\Delta\delta_1, \Delta\alpha_1, .., \Delta\delta_M, \Delta\alpha_M]$. $\Delta\delta_M$ and $\Delta\alpha_M$ are correction factors for a segment M. All trajectory is divided into segments M to allow better map optimization. Fitness function has two major factors; map consistency and map compactness. The former measures the conflict between sensor readings by looking at the occupancy grid map's cells that are taken and that are empty by taking the minimum of that. Map compactness includes rewarding the GA for producing smaller and compact maps by fitting a bounding box to the map. The results were promising.

Lindblad et al. (2002) proposed an interesting way to find features in the images. The authors proposed to create graph trees that show different object locations in the image followed with various size scalars and colors. The individual's genotype is composed of nodes, which are positions and objects. The position node carries

information such as translation and rotation of every object in the sub tree, while the edge depicts where the object is—for example, near a parent, flat on the ground, flat on top of the parent, and such. Object represents size scalar, color, and type of object (in this case a block). All positions have a child node which is an object. An object may have any number of children that define the relative position of each sub tree. Just like in Duckett et al. (2003), Lindblad et al. (2002) also highlight the importance of the complexity of the model in the fitness function.
Kasparavičiūtė et al. (2018) and Alvarez et al. (2004) describe the use of GAs for autonomous underwater vehicles. The former presented an idea for an ocean cleaning AUV. The fitness function is a multi-objective function that has two major parts; navigation and object detection. The latter describe application of GAs for path planning, focusing especially on achieving energy efficient solutions in a strong current characterized environment.
Image classification utilizing genetic programming (GP) has been approached by a few authors. Due to GP's wide variety of application, each author applies GP in a different way. Johnson et al. (2000) produced a chromosome for each class. In their example, the system is required to run twice each time for one class. The fitness function is the root mean square error of predictions of each class. Fitness function also included penalty to reinforce chromosomes that are shorter. Huang et al. (2006) have showcased a similar idea, but they allowed the GP to create several individuals per class. Eggermont et al. (1999) chose to create one chromosome/individual for all labeled data. However, the algorithm defines its own fitness function while training itself by modifying weights at the run time. Hope et al. (2007) used GP to classify mammograms for breast cancer. The researchers have taken only the windows of images that have microcalcification (a feature that shows if the mammogram displays breast cancer or not) and selected such features like mean, second and third moments.
As mentioned before, image classification task is usually solved by employing deep neural networks. However, for a neural network to achieve over 90% accuracy, it generally requires at least 100 images in a training set (Chan et al., 2015). Another example of use of deep neural networks also shows that it requires some pre-processing on images (Cireşan et al., 2012). Cireşan et al. (2012) also show that in order to evaluate one image may take up to a thousand or more connections (calculations) in the neural network.

3

Background on Underwater Simulation

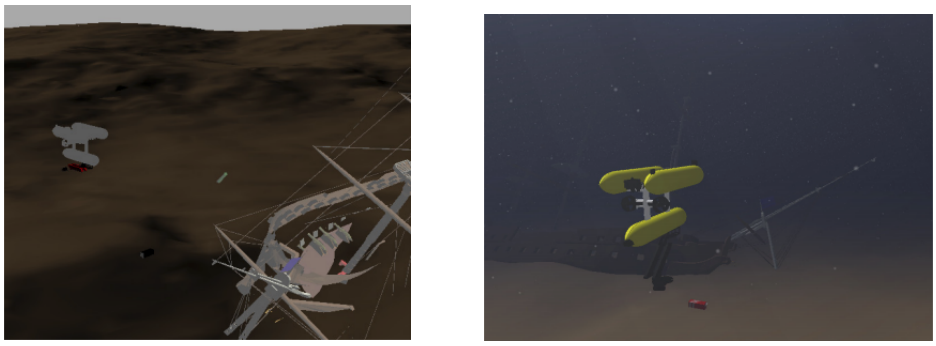
Autonomous underwater vehicles have become increasingly popular in industry and research, as using remotely operated vehicles costs in not only man hours but also logistics—it is expensive to deploy a ship full of engineers to an offshore location to investigate man-made structures, such as the wells and risers used for oil and gas pumping. Testing and experimentation with AUVs is challenging as it also requires vessels to move underwater vehicles and engineers to set them in place, and there exist many uncertainties in the environment, which may result in the destruction or loss of the AUV. Therefore, there is a high demand for AUV simulators. Usually, small missions can be carried out in swimming pools, but these lack the intensity of the real world environment.

3.1 Game engines

Game engines offer the capabilities in graphics and physics needed when creating simulations. Most popular game engines are used for AUVs, such as Unity3D, Unreal 4 Engine, and CryEngine (Chin et al., 2018). An example of such an AUV simulator is developed by Abyssal Technology (Parente, 2018). However, this simulator is not open source. Chin et al. (2018) have described the development steps needed to make an underwater simulator using the Unity3D game engine. Some of the inspiration for future simulators has been taken from older generation simulators, such as MVS, CADCON, NEPTUNE, SUBSIM, etc. Matsebe et al. (2008) provided an extensive review of all of them.

3.2 Robot Operating System

Nowadays, most open source simulators are based on Robot Operating System (ROS). It is a robot software framework consisting of the following components; nodes, messages, and topics. Nodes are software modules that communicate with each other in a peer-to-peer manner. Messages are published by the nodes that other nodes can subscribe to. This means that many nodes can be connected to one topic, and vice versa. The message holds the data and its structure.
UWSim is an example of an underwater simulator based on ROS (Prats et al., 2012). It uses an open source 3D graphics application called OpenSceneGraph and renders realistic underwater images through osgOcean. Figure 3.1 showcases the traditional



(a) View in Gazebo **(b)** View in UWSim

Figure 3.1: Difference in rendered views taken from Kermorgant (2014)

and underwater-oriented rendered views in simulations. UWSim allows the addition of different sensors including DVL, GPS (which works only up to a certain depth), robotic claw, etc. UWSim contains drawbacks, as it is mostly a kinematic simulator. Gazebo, on the other hand, manages dynamics and contact physics, but is mostly designed for terrestrial robots. Kermorgant (2014) presented a combination of both simulators by incorporating Gazebo into UWSim, called freefloating Gazebo.
MORSE is another simulator for academic robotics (Echeverria et al., 2011). Its rendering is based on Blender Game Engine and it supports 6 different open-source middlewares, including ROS, Mavlink, etc. MORSE comes with a range of different sensors that are able to simulate human robots, terrestrial, aerial and underwater vehicles. Furthermore, it is able to handle dozen of robots. On the other hand, MORSE does not include advanced algorithms e.g., for path planning nor does it have a GUI.
Carvalho et al. (2014) also presented an underwater simulator using ROS as a middleware, but they utilized Virtual Reality Engine. Its name, SimUEP-Robotics, is based on the name in Portuguese. The aim of their simulator is to enable testing of ROVs and AUVs for the oil and gas industry. The system supports multiple robots simultaneously, equipped with the usual expected sensors and actuators. SimUEP-Robotics has also integrated trajectory visualizations, ghostview animations, scenario editors, etc.

4

Convolutional Genetic Programming Algorithm

This paper proposes a new way for solving binary image classification problems, based on Linear Genetic Programming (LGP). The basics of LGP are explored in section 2.3. The population is created and all individuals are evaluated according to a fitness function. A few individuals are chosen among the population pool, and they compete against each other. The winners replace the losers to maintain highly evaluated individuals. Then, a few of the selected winners go through mutation and crossover to create new individuals.
The process of image classification can be seen in figure 4.1, showing the learning/-training process and testing procedure.

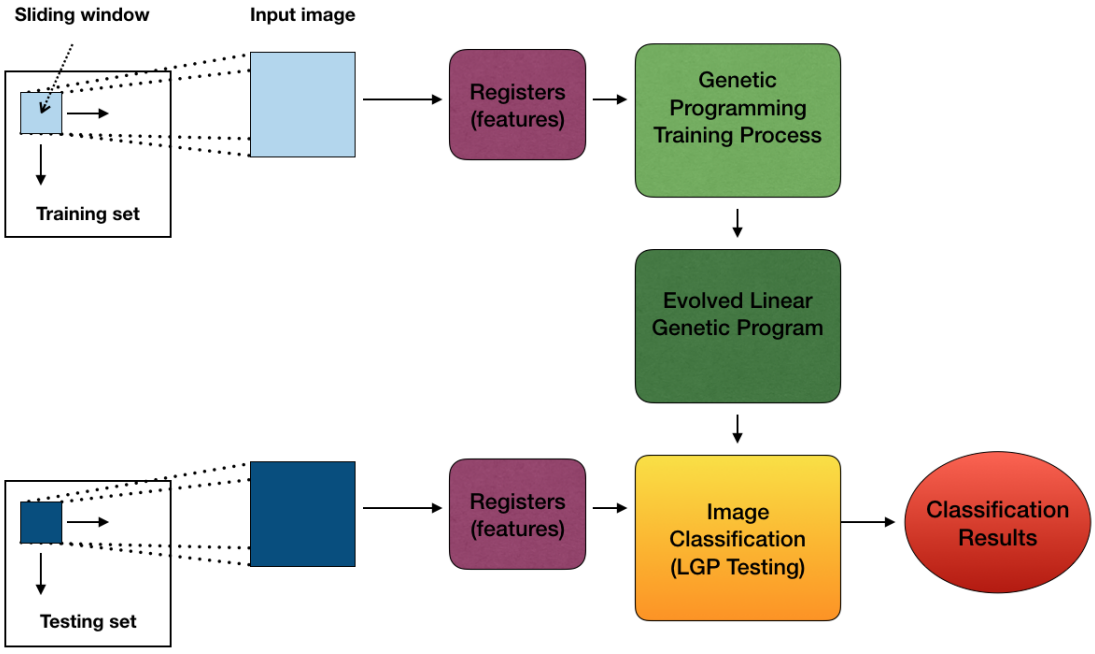


Figure 4.1: LGP approach to object classification based on Zhang and Lett (2006).

The training starts with taking the first window of the training set image of size n by n. This window becomes the input image and is then passed as an array, instead of a matrix, to the LGP training process. The individual or chromosome is then

executed using the passed registers (image's raw pixels in the chosen window). More information on fitness function is in section 4.1. Each window moves s number of steps, also referred to as a stride, and is then passed as a register. When all windows have been passed through an individual, the accumulated result from the fitness function is compared to the label of the image and then added to the total of the error of individuals. The error of an individual is the accumulated squared difference between the label and retrieved result. All images are evaluated in the same way.

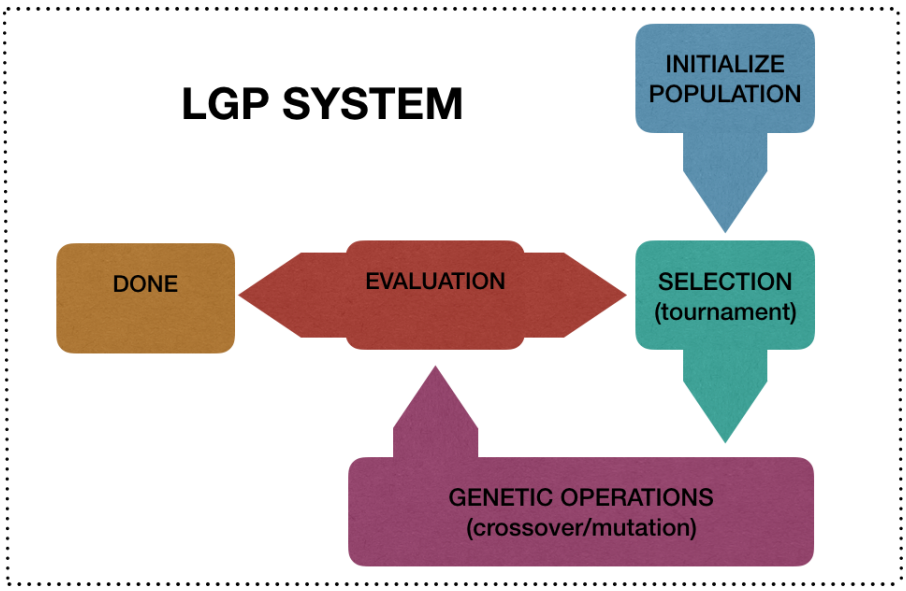


Figure 4.2: LGP system flowchart example taking place in the Genetic Programming Training Process in fig. 4.1 based on Kasparavičiūtė et al. (2018)

.

The general pseudo-code example of a genetic algorithm is shown below. The initialization requires establishing variables like populationSize, individualLength (which could be divided to minimum and maximum possible individual length), probability of crossover, probability of mutation, and max number of generations (or other stopping criteria). When the population is initialized (all population individuals are set to the maximum possible fitness value), the main process of LGP starts which can also be seen in the figure 4.2). Usually 4 individuals are randomly chosen for a tournament from the total population, and then compete against each other. The two winners then replace the losers of the tournament, and the original winners proceed to the genetic operations step.
In crossover, the winners become the parents and some random crossover point is chosen so that the parents are able to create two new individuals. These two individuals are then exposed to mutation, after which they are placed back into the population. The original parents are left untouched and added to the population.
The total population is then evaluated and the best individual is kept. When the maximum number of generations have reached, the bestIndividual, or solution, is then retrieved. The arrow to the left represents assigning.

```
Input: populationSize, individualLength, pCrossover, pMutation, maxGenerations,
    tournamentMembers
Output: bestIndividual
Population <− InitializePopulation(populationSize, individualLength)
bestIndividual <− First individual of Population

While(generation < maxGenerations)
    Children <− TournamentSelection(Population, tournamentMembers)
    Child1, Child2 <− Crossover(Children, pCrossover)
    Children <− Mutate(Child1, pMutation)
    Children <− Mutate(Child2, pMutation)
    EvaluateChildren(Children)
    Population <− PutChildrenBackToPopulation(Population, Children)
    bestIndividual <− RetrieveBestIndividual(Population)
End
Return (bestIndividual)
```

Listing 4.1: General evolutionary algorithm pseudo-code.

4.1 Fitness function

Fitness function allows evaluation of the individual. The fitness function for convolutional genetic programming starts with the *scratchVariable* and registers (windows of the images). The *scratchVariable* is saved in the last position of a register array. This value gives awareness to the function about the image (if an object exists or not). It can be set in the function in any way but is not cleared between subimage loop-steps, it is only cleared at the start of the new image.
When a window is passed to the decoder, all the individual's instructions are executed and only the first and last register values are returned. The first register value is added to the total of *y_result* which keeps track of the image (see eq. 4.1). *scratchVariable* is passed on to the next window and is reused by the individual. When all of an image's windows have been evaluated, the result is calculated by multiplying *scratchVariable* by a high value alpha and adds the accumulated *y_result* (see eq. 4.2).
The final step for the error calculation is to look at the label of the image and subtract the result from it in the form of absolute value (see eq. 4.3). The difference is then squared and added to the final error variable that keeps track of each individual's fitness (see eq. 4.4).

$$y\_result = \sum_{n=1}^{n} registers[0] \tag{4.1}$$

$$RET = \alpha * scratchVariable + y\_result \tag{4.2}$$

$$DIFF = abs(LABEL - RET)^2 \tag{4.3}$$

$$Fitness = \sum_{i=1}^{i} DIFF_i \tag{4.4}$$

where:

- y_result is the sum of first registers in each window of an image.
- n is the number of windows per image.
- RET is the intermediary result from one image.
- $scratchVariable$ is the value that allows the algorithm to keep notes about the subpictures/windows.
- α is the number that shows the weight of $scratchVariable$.
- $DIFF$ is the calculated squared absolute difference between the label value and the RET.
- i is the number of images in the set.
- $Fitness$ is the final fitness value.

One individual's evaluation example code can be seen below in listing 4.2.

```
1  float ConvLinearGeneticProgramming::evaluateIndividual(Individual individualToEvaluate)
2  {
3      float error = 0.0f;
4      for (int m = 0; m < imagesNumber; m++)
5      {
6          float scratchVariable = SCRATCH_VAR;
7          float y_result = 0.0f;
8
9          for (int row = 0; row <= images[m].rows - WINDOW_SIZE; row += STEP)
10         {
11             for (int col = 0; col <= images[m].cols - WINDOW_SIZE; col+= STEP)
12             {
13                 cv::Rect windows(col, row, WINDOW_SIZE, WINDOW_SIZE);
14                 cv::Mat roi = images[m](windows); //Region Of Interest
15
16                 std::pair<float, float> answer = decodeIndividual(individualToEvaluate, roi,
       scratchVariable);
17                 scratchVariable = answer.second;
18                 y_result += answer.first;
19             }
20         }
21         float y_result_final = scratchVariable * 1000 + y_result;
22         float y_groundtruth  = labels[m];
23         float diff = abs(y_groundtruth - y_result_final);
24         error += diff * diff;
25     }
26     return error;
27 }
```

Listing 4.2: Fitness function pseudo-code.

5
Methodology

The following chapter describes how data was acquired in order to be able to test the convolutional linear genetic programming.

5.1 Collection of Data

Data acquisition was performed in two steps. First, the stakeholders could not share their sensitive data from subsea ROV dives, so an alternative was proposed. The stakeholders used a Structure from Motion (SfM) [1] where they gathered a 3D cloud point of a subsea template and the subseafloor. The stakeholders were able to remove the identification numbers from the template to provide us with a 3D model of the template and subseafloor, with a texture that looks exactly like how it does at a depth of 500 m. under the Norwegian sea. Secondly, both the mesh of the template and subseafloor and textures were imported into an underwater simulator called UWSim that utilizes Robot Operating System (ROS) (see fig. 5.1).

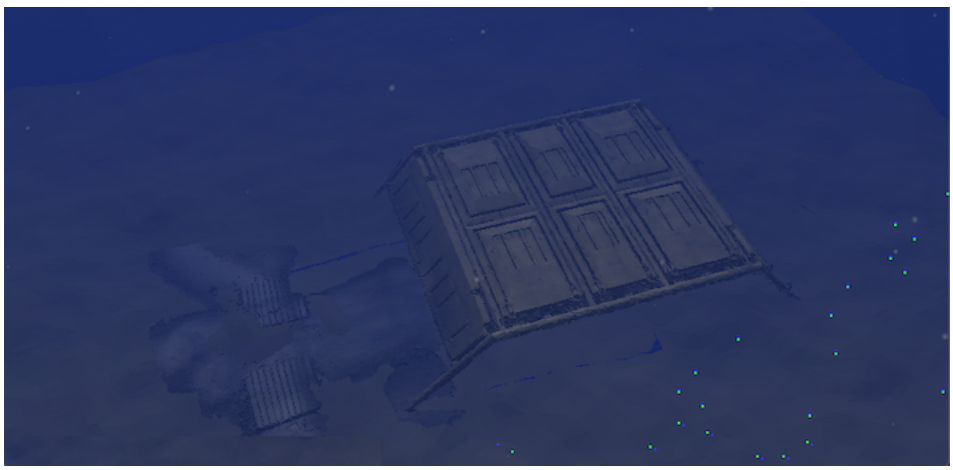


Figure 5.1: Subseafloor and template taken from the real world 3D cloud point and imported into an underwater simulator.

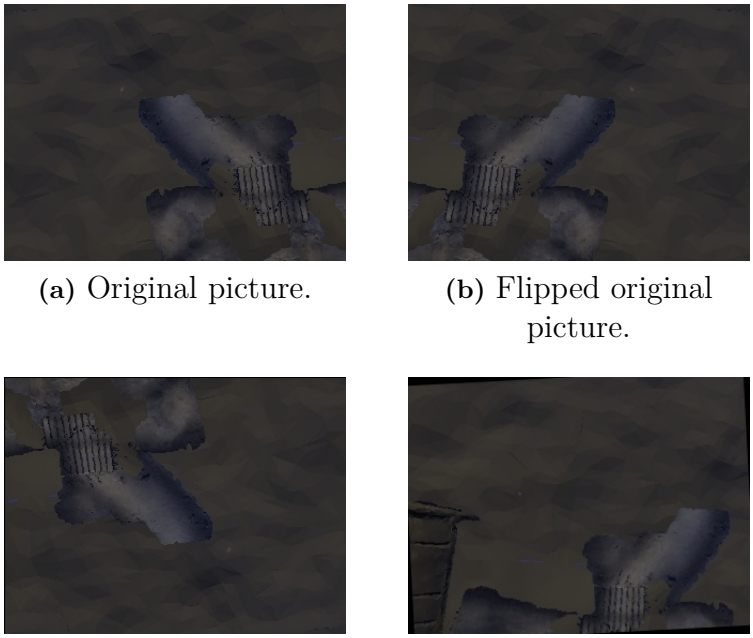
The template is 25 m. × 25 m. × 20 m. in size. The subseafloor is approximately 100 m x 100 m in size. The depth chosen for the simulator was 500 m. The autonomous underwater vehicle (AUV) used for exploring this area and gathering the images is called Girona500 (Ribas et al., 2012), and is the default AUV in UWSim. The AUV

[1] A technique used for creating 3D objects from 2D image sequences.

required some alterations to follow the stakeholders' requirements. Firstly, a change was made to the camera angle from 180°(downward looking) to 135°. Secondly, the camera's intrinsic and extrinsic parameters were changed to mimic imperfect conditions (i.e., to mimic water turbidity).
The AUV was controlled manually using the keyboard. It began at the edge of the sea floor and moved towards the template, where it moved in a zigzag pattern in order to be able to investigate it from all angles.

5.2 Experiments

Two runs were made to gather the images. Firstly, the AUV was controlled to simply move towards the template, with just a few changes in depth, where it was able to see part of the template at all times. The second run was accomplished by controlling the AUV from the side opposite the first run. The AUV changed its depth often, which meant it would lose sight of the template more often. This was done in order to simulate waves or other disturbances in the water. Both runs were approximately 90 seconds long and acquired around 200 frames each.
Each image is 320 × 240 pixels in size. The original pictures were rotated, flipped, and scaled with several different possible configurations (e.g., rotated and flipped, only rotated, rotated and scaled, and so on). The parameters for picture transformations were completely random. The pictures were labeled into *IS TEMPLATE* (see fig. 5.3) and *NOT TEMPLATE* (see fig. 5.2). The algorithm was trained on a total number of 30 images, where half were labeled *IS TEMPLATE* and the other half *NOT TEMPLATE*.
The validation or test set includes 576 images, which, like the training set, are augmented in various ways. Each class has half images.
Before running the experiments, some parameters relevant to the problem such as window size and step/stride were needed, so an extra experiment was performed to find them out. Each of these short experiments was conducted 3 times, with the average shown in table 5.1. The winner hyperparameters are in the experiment that used window and stride of 5 pixels each.



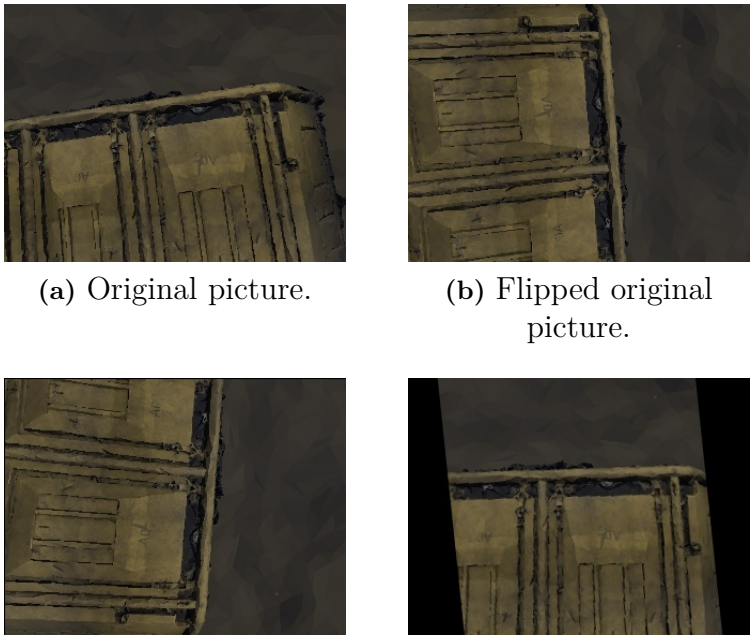
(a) Original picture.

(b) Flipped original picture.

(c) Flipped and rotated original picture.

(d) Rotated and scaled original picture.

Figure 5.2: *NOT TEMPLATE* images taken from the simulator.



(a) Original picture.

(b) Flipped original picture.

(c) Flipped and rotated original picture.

(d) Rotated original picture.

Figure 5.3: *IS TEMPLATE* images taken from the simulator.

Table 5.1: Selection of window and step size/stride, average taken after 3 repeats of each experiment.

Experiment	Training data set size	Window	Step/stride	Fitness	Best individual's size	Termination criteria
1	30	5	1	26 190	2898	200 gens.
2	30	5	5	22 042	485	200 gens.
3	30	8	4	121 294	964	200 gens.
4	30	8	8	35 581	874	200 gens.
5	30	10	5	68 182	2206	200 gens.
6	30	10	10	75 370	830	200 gens.
7	30	20	10	52 912	468	200 gens.
8	30	20	20	86 590	2309	200 gens.

6
Results

This section describes the results of the experiment. In order to be able to discuss the results, we first need to discuss the parameters used to achieve them. The table for these parameters is sometimes referred to as a Koza tableau (Koza and Rice, 1991). Most of these parameters have been discussed in the previous section (see sec. 2.3). Probabilities of crossover, tournament, and mutation have been chosen according to the standard (Hu, 1998). Window and step size/stride have been chosen after running short experiments (max 200 generations) and registering the achieved fitness value (see section 5). The number of images in the training set is 30, while the validation set consists of 576 images. The validation set includes unseen data.

Table 6.1: Koza tableau for convolutional LGP experiment.

Parameter	Setting
Fitness function	Standardized, root mean squared error
Terminal set	None
Function set	[+, -, *, /, $\sqrt{x}$]
Safe division enabled?	Yes
Window size	5
Step size/stride	5
Number of registers	26
Population size	10000
Crossover probability	0.7
Tournament probability	0.95
Mutation probability	0.05
Selection	Steady-state tournament
Number of tournament members	4
Termination criteria	None
Minimum individual length	10
Maximum individual length	3000
Label for *IS TEMPLATE*	100
Label for *NOT TEMPLATE*	0
Alpha	1000
Scratch variable	0

The best individual reached a length of 205, including junk instructions. After introns were removed, the length of the individual was reduced by 85%, to 29 in-

structions. Classification results have an accuracy of 99.8%. The best individual was then used on a video gathered from the simulator, achieving a classification rate of 55 frames per second. This means that the proposed algorithm has a potential to evaluate 55 frames per second in a provided video. As a comparison, the usual movies are shown at 24 frames per second. The achieved result shows a high potential.
The confusion matrix reveals classification performance on the validation data set (see table 6.2). As seen in table 6.1, the value for *IS TEMPLATE* is shown as 100, while the value for *NOT TEMPLATE* is 0 (zero). Therefore, the threshold for differentiating between these two classes has been chosen as 50. The confusion matrix also provides rate information in table 6.3, where it can be seen that the overall accuracy of the proposed classifier is high (0.996)—the classifier made only two mistakes (see pictures in fig. 6.1). The calculated predicted values for both of these mistakes was very close to the threshold (55 and 52 respectively, compared to the threshold set at 50). When the classifier predicts *IS TEMPLATE*, it is correct 99.3% of the time. It was also important to show that half the images belong to one class and the other half belongs to the other, which is shown in the prevalence rate of 0.5. The weighted average of the recall and precision, also called F1 Score, was also high.

(a) First classification mistake.

(b) Second classification mistake.

Figure 6.1: The only two mistakes the algorithm made.

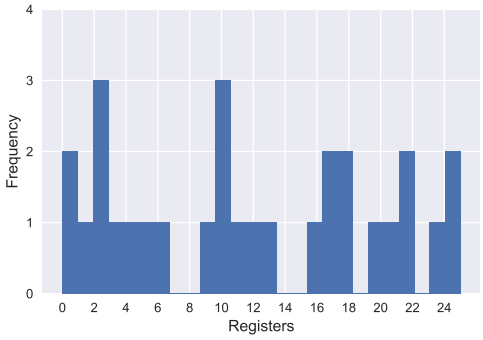
Table 6.2: Confusion matrix for the experiment run on validation data set.

		Actual values		
		IS TEMPLATE	NOT TEMPLATE	Total
Predicted values	IS TEMPLATE	288	2	290
	NOT TEMPLATE	0	286	286
	Total	288	288	576

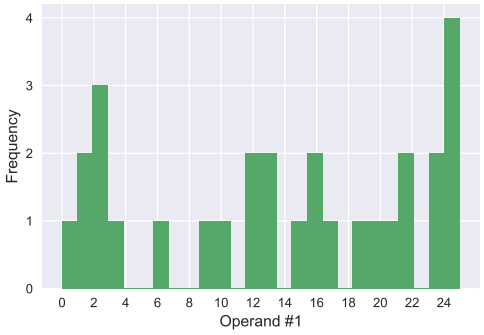
The best results and the shortest individual were achieved when the window and step sizes were 5×5, therefore these parameters have been also chosen for the main experiment. The number of registers is calculated by the following equation:

$$registers\_number = WINDOW\_SIZE * WINDOW\_SIZE + 1 \qquad (6.1)$$

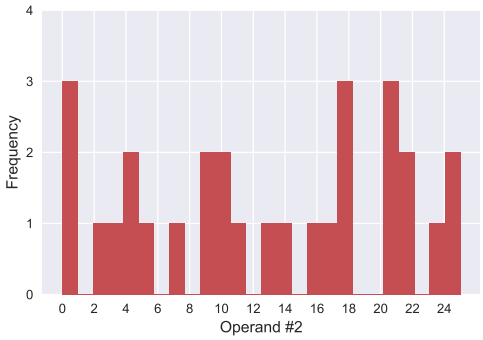
The extra one register is the added scratch variable for the algorithm.
Due to linear genetic programming's transparency advantage, it is possible to give the trained classifier a closer look. This can be achieved by looking at the histograms of each instruction, e.g., $(a = c+2), member, i.e., register(a), operand\#1(c), operand\#2(2), andopera$
As previously discussed, the proposed image classification method has many advantages compared to the regular neural network. Firstly, it does not require any pre-processing done on images. Secondly, due to the removal of introns, the final amount of instructions is reduced by at least an order of magnitude. Furthermore, Nordin and Banzhaf (1996) have provided evidence that by using programmatic compression it is possible to run LGP retrieved result on embedded devices and achieve even higher rate per frames, e.g. 60 frames per second. Lastly, the provided convolutional linear genetic programming image classification technique was able to distinguish between images by training it only on 30 images.



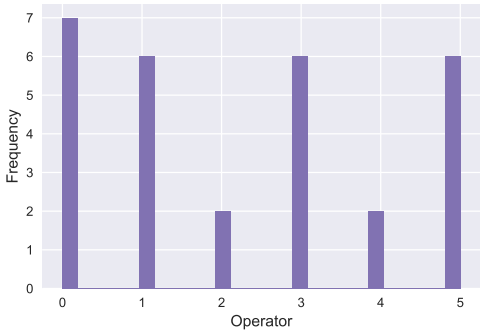
(a) Registers histogram.



(b) Operand #1 histogram.



(c) Operand #2 histogram.



(d) Operators which are addition, subtraction, multiplication, division, and square roots of both operands, respectively.

Figure 6.2: Histograms of each of the best individual's 29 instruction members.

Table 6.3: Rates and calculations based on the confusion matrix.

Rate	Formula	Calculation
Accuracy	(TP+TN)/total	0.996
Misclassification Rate	(FP+FN)/total	0.003
True Positive Rate/Recall/Sensitivity	TP/actual *IS TEMPLATE*	1.0
False Positive Rate	FP/actual *NOT TEMPLATE*	0.007
True Negative Rate/Specificity	TN/actual *NOT TEMPLATE*	0.993
Precision	TP/predicted *IS TEMPLATE*	0.993
Prevalence	actual *IS TEMPLATE*/total	0.5
F1 Score	$2 * \frac{Precision * Recall}{Precision + Recall}$	0.996

7
Conclusion

This paper focuses primarily on implementing a classification algorithm for an autonomous underwater vehicle (AUV) based in a subsea garage, where it will perform inspections and other related work. In order to allow the AUV to roam the sea floor without any manual intervention, the AUV requires some sort of SLAM technique to navigate its surroundings. In order for the AUV to accomplish it, Visual and Visual-Inertial SLAM has been separated into front- and back-end, where front-end takes care of the feature extraction and data association and back-end aids in map estimation. The principal paradigms to solve SLAM are filter and keyframe-based. The latter one has showcased much more promising results than its predecessor, filter-based SLAM.

This paper reviewed the basics of SLAM techniques such as localization and mapping, followed by highlighting major SLAM methodologies, i.e., DolphinSLAM, DT SLAM, FAB MAP, EKF MonoSLAM, LSD SLAM, ORB SLAM, Pop-up SLAM, RKSLAM. These have been chosen from many other possible SLAM techniques due to how their design meets the following requirements: they utilize monocular cameras with the possibility to fuse their output with other sensors such as inertial measurement units, they are open source, they use a loop closure, and finally, they offer a complete SLAM technique. The selected algorithm for the stakeholder's use case is ORB SLAM as it has been proven to show positive results in large, featureless underwater environments.

Secondly, this paper proposes a new method for image classification. The main advantages of linear genetic programming are; the transparency of the results, the prior knowledge about the distribution is unnecessary thus pre-processing is not required, it automatically selects features and different mechanisms to control the size/length of the classifier; easy parallelization which reduces training time. Due to these reasons the proposed binary image classifier utilizes linear genetic programming. The data was acquired from the actual subsea environment that extracted 3D cloud points, reaching depths of 500 m. in the Norwegian sea. The AUV was manually controlled in Robot Operating System, where it gathered different realistic images of the undersea environment. The images were sorted (labeled as *IS TEMPLATE* and *NOT TEMPLATE*), augmented, and then transferred into the convolutional linear genetic programming system, which created a classifier based on only 30 images from a training set. The validation set included nearly 600 images and the classifier achieved 99.8% accuracy. Due to the best individual's short length of instructions, it was possible to achieve a classification rate of 55 frames per second.

Due to the use of Robot Operating System (ROS), the paper also provided an

assessment of current underwater simulators. The most popular simulators are based on game engines, such as Unity3D, Unreal 4 Engine, and CryEngine. There are a few simulators that are mostly used in academia. These include but are not limited to UWSim and Gazebo which are both based on ROS and MORSE.
Future work includes performing classification on real world images. Furthermore, next steps include enhancing the new methodology's capabilities, and to also use it for object detection.

Bibliography

Alvarez, A., Caiti, A., and Onken, R. (2004). Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2):418–429.

Anwer, A., Ali, S. S. A., Khan, A., and Mériaudeau, F. (2017). Underwater 3-d scene reconstruction using kinect v2 based on physical models for refraction and time of flight correction. *IEEE Access*, 5:15960–15970.

Aulinas Masó, J. M., Carreras Pérez, M., Lladó Bardera, X., Salvi, J., García Campos, R., and Prados Gutiérrez, R. (2011). Feature extraction for underwater visual slam. *© Oceans (2011: Santander: Espanya). OCEANS, 2011 IEEE-Spain*.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332.

Carvalho, F., Raposo, A., Santos, I., and Galassi, M. (2014). Virtual reality techniques for planning the offshore robotizing. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 353–358. IEEE.

Chan, T.-H., Jia, K., Gao, S., Lu, J., Zeng, Z., and Ma, Y. (2015). Pcanet: A simple deep learning baseline for image classification? *IEEE transactions on image processing*, 24(12):5017–5032.

Chin, C., Zhong, X., Cui, R., Yang, C., and Mohan, V. (2018). Virtual simulation platform for training semi-autonomous robotic vehicles' operators. In *Autonomous Vehicles*. IntechOpen.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.

Cummins, M. and Newman, P. (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665.

Cummins, M. and Newman, P. (2011). Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. In *ICRA*, volume 2, pages 1322–1328.

Dong, J. F., Wijesoma, W. S., and Shacklock, A. P. (2007). An efficient rao-blackwellized genetic algorithmic filter for slam. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2427–2432. IEEE.

Duckett, T. et al. (2003). A genetic algorithm for simultaneous localization and mapping. *IEEE International Conference on Robotics and Automation*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110.

Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular openrobots simulation engine: Morse. In *Proceedings of the IEEE ICRA*.

Eggermont, J., Eiben, A. E., and van Hemert, J. I. (1999). A comparison of genetic programming variants for data classification. In *International Symposium on Intelligent Data Analysis*, pages 281–290. Springer.

Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer.

Engel, J., Stückler, J., and Cremers, D. (2015). Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942. IEEE.

Engineering, C. S. (2016). Measurement of system reliability part 1.

Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):121–144.

Eustice, R., Singh, H., Leonard, J. J., Walter, M. R., and Ballard, R. (2005). Visually navigating the rms titanic with slam information filters. In *Robotics: Science and Systems*, volume 2005, pages 57–64.

Ferrera, M., Moras, J., Trouvé-Peloux, P., and Creuze, V. (2019). Real-time monocular visual odometry for turbid and dynamic underwater environments. *Sensors*, 19(3):687.

Fuentes-Pacheco, J., Ruiz-Ascencio, J., and Rendón-Mancha, J. M. (2015). Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81.

Grasa, O. G., Civera, J., and Montiel, J. (2011). Ekf monocular slam with relocalization for laparoscopic sequences. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4816–4821. IEEE.

Guth, F., Silveira, L., Botelho, S., Drews, P., and Ballester, P. (2014). Underwater slam: Challenges, state of the art, algorithms and a new biologically-inspired approach. In *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 981–986. IEEE.

Herrera, C. D., Kim, K., Kannala, J., Pulli, K., and Heikkilä, J. (2014). Dt-slam: deferred triangulation for robust slam. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 1, pages 609–616. IEEE.

Hope, D., Munday, E., and Smith, S. (2007). Evolutionary algorithms in the classification of mammograms. In *2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing*, pages 258–265. IEEE.

Hu, Y.-J. (1998). A genetic programming approach to constructive induction. In *Genetic Programming 1998: Proc. 3rd Annual Conf*, pages 146–151.

Huang, J.-J., Tzeng, G.-H., and Ong, C.-S. (2006). Two-stage genetic programming (2sgp) for the credit scoring model. *Applied Mathematics and Computation*, 174(2):1039–1053.

Hwang, S.-Y. and Song, J.-B. (2011). Monocular vision-based slam in indoor environment using corner, lamp, and door features from upward-looking camera. *IEEE Transactions on Industrial Electronics*, 58(10):4804–4812.

Imani, V., Haataja, K., and Toivanen, P. (2018). Three main paradigms of simultaneous localization and mapping (slam) problem. In *Tenth International Conference on Machine Vision (ICMV 2017)*, volume 10696, page 106961P. International Society for Optics and Photonics.

Johnson, H. E., Gilbert, R. J., Winson, M. K., Goodacre, R., Smith, A. R., Rowland, J. J., Hall, M. A., and Kell, D. B. (2000). Explanatory analysis of the metabolome using genetic programming of simple, interpretable rules. *Genetic Programming and Evolvable Machines*, 1(3):243–258.

Jung, J., Choi, S., Choi, H.-T., and Myung, H. (2016). Localization of auvs using depth information of underwater structures from a monocular camera. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2016 13th International Conference on*, pages 444–446. IEEE.

Kahlefendt, C. (2017). List of slam and visual odometry algorithms. `https://github.com/kafendt/List-of-SLAM-VO-algorithms/`.

Kasparavičiūtė, G., Nielsen, S. A., Boruah, D., Nordin, P., and Dancu, A. (2018). Plastic grabber: Underwater autonomous vehicle simulation for plastic objects retrieval using genetic programming. In *International Conference on Business Information Systems*, pages 527–533. Springer.

Kermorgant, O. (2014). A dynamic simulator for underwater vehicle-manipulators. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 25–36. Springer.

Kim, A. (2012). Active visual slam with exploration for autonomous underwater navigation. Technical report, MICHIGAN UNIV ANN ARBOR DEPT OF MECHANICAL ENGINEERING.

Kim, A. and Eustice, R. M. (2013). Real-time visual slam for autonomous underwater hull inspection using visual saliency. *IEEE Transactions on Robotics*, 29(3):719–733.

Kishore, J. K., Patnaik, L. M., Mani, V., and Agrawal, V. (2000). Application of genetic programming for multicategory pattern classification. *IEEE transactions on evolutionary computation*, 4(3):242–258.

Kongsberg, M. (2016). Hipap high precision acoustic positioning.

Koza, J. R. and Rice, J. P. (1991). Genetic generation of both the weights and architecture for a neural network. In *IJCNN-91-seattle international joint conference on neural networks*, volume 2, pages 397–404. IEEE.

Lindblad, R., Nordin, P., and Wolff, K. (2002). Evolving 3d model interpretation of images using graphics hardware. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 225–230. IEEE.

Liu, H., Zhang, G., and Bao, H. (2016). Robust keyframe-based monocular slam for augmented reality. In *Mixed and Augmented Reality (ISMAR), 2016 IEEE International Symposium on*, pages 1–10. IEEE.

Mahon, I. (2008). Vision-based navigation for autonomous underwater vehicles. Technical report, University of Sydney.

Manzanilla, A., Sanchez, S. R., Rangel, M. A. G., Ravell, D. A. M., and Lozano, R. (2019). Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision. *IEEE Robotics and Automation Letters*.

Matsebe, O., Kumile, C., and Tlale, N. (2008). A review of virtual simulators for autonomous underwater vehicles (auvs). *IFAC Proceedings Volumes*, 41(1):31–37.

Milford, M. J., Wyeth, G. F., and Prasser, D. (2004). Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163.

Mur-Artal, R. and Tardós, J. D. (2017a). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

Mur-Artal, R. and Tardós, J. D. (2017b). Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803.

Naminski, M. R. (2013). An analysis of simultaneous localization and mapping (slam) algorithms. Technical report, Macalester College.

Nordin, P. and Banzhaf, W. (1996). Programmatic compression of images and sound. In *Proceedings of the 1st annual conference on genetic programming*, pages 345–350. MIT Press.

Nordin, P. and Banzhaf, W. (1997). Real time control of a khepe. ra robot using genetic programmmg. *Control Cybern*, 26(3).

Oltean, M. and Dioşan, L. (2009). An autonomous gp-based system for regression and classification problems. *Applied Soft Computing*, 9(1):49–60.

Parente, M. (2018). Leveraging solutions from other industries.

Patra, S., Gupta, K., Ahmad, F., Arora, C., and Banerjee, S. (2017). Batch based monocular SLAM for egocentric videos. *CoRR*, abs/1707.05564.

Prats, M., Perez, J., Fernández, J. J., and Sanz, P. J. (2012). An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 2577–2582. IEEE.

Ribas, D., Palomeras, N., Ridao, P., Carreras, M., and Mallios, A. (2012). Girona 500 auv: From survey to intervention. *IEEE/ASME Transactions on mechatronics*, 17(1):46–53.

Shukla, A. and Karki, H. (2016). Application of robotics in offshore oil and gas industry—a review part ii. *Robotics and Autonomous Systems*, 75:508–524.

Silveira, L., Guth, F., Drews-Jr, P., Ballester, P., Machado, M., Codevilla, F., Duarte-Filho, N., and Botelho, S. (2015). An open-source bio-inspired solution to underwater slam. *IFAC-PapersOnLine*, 48(2):212–217.

Strasdat, H., Montiel, J., and Davison, A. J. (2010). Real-time monocular slam: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664. IEEE.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE.

von Stumberg, L., Usenko, V., Engel, J., Stückler, J., and Cremers, D. (2017). From monocular slam to autonomous drone exploration. In *Mobile Robots (ECMR), 2017 European Conference on*, pages 1–8. IEEE.

Williams, S. B., Newman, P., Dissanayake, G., and Durrant-Whyte, H. (2000). Autonomous underwater simultaneous localisation and map building. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 1793–1798. IEEE.

Williams, S. B., Pizarro, O., and Foley, B. (2016). Return to antikythera: Multi-session slam based auv mapping of a first century bc wreck site. In *Field and Service Robotics*, pages 45–59. Springer.

Yang, S., Song, Y., Kaess, M., and Scherer, S. (2016). Pop-up slam: Semantic monocular plane slam for low-texture environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1222–1229. IEEE.

Younes, G., Asmar, D., Shammas, E., and Zelek, J. (2017). Keyframe-based monocular slam: design, survey, and future directions. *Robotics and Autonomous Systems*, 98:67–88.

Yousif, K., Bab-Hadiashar, A., and Hoseinnezhad, R. (2015). An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311.

Yuan, X., Martínez-Ortega, J.-F., Fernández, J. A. S., and Eckert, M. (2017). Aekf-slam: a new algorithm for robotic underwater navigation. *Sensors*, 17(5):1174.

Zhang, H., Zhang, C., Yang, W., and Chen, C.-Y. (2015). Localization and navigation using qr code for mobile robot in indoor environment. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2501–2506. IEEE.

Zhang, M. and Lett, M. (2006). Genetic programming for object detection: Improving fitness functions and optimising training data. *IEEE Intelligent Informatics Bulletin*, 7(1):12–21.

A

Appendix 1

Name	Sensors	Front-end	Back-end	Refs	Notes
DolphinSLAM	Monocular, IMU, Sonar, DVL	Sensor dependant, Sonar: HU moments, Image: SURF	RatSLAM back-end	Silveira et al. (2015)	Tested underwater
DT SLAM	Monocular	FAST	Bundle adjustment	Herrera et al. (2014)	Tested with mobile robot
FAB MAP	Monocular	Visual bag of words with SURF	Place recognition in appearance-space	Cummins and Newman (2008)	Tested with mobile robot
EKF MonoSLAM	Monocular	Active search, FAST	1-Point RANSAC, Randomised List Relocalization	Grasa et al. (2011)	Tested both indoors and outdoors
LSD SLAM	Monocular, possible to use IMU	Semi-dense	g2o	Engel et al. (2014)	Tested in ground and aerial environment
ORB SLAM	Monocular, possible IMU	ORB feature descriptor	Bundle adjustment	Mur-Artal et al. (2015), Mur-Artal and Tardós (2017a)	Tested in subsea, ground and aerial environments
Pop-up SLAM	Monocular	CNN	LSD SLAM	Liu et al. (2016)	Tested indoors
RKSLAM	Monocular, IMU	FAST	Bundle adjustment	Liu et al. (2016)	Tested both indoors and outdoors

Table A.1: List of most suitable SLAM techniques for solving the stakeholder's problem.