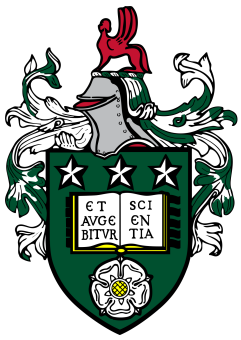
Robotic navigation and inspection of bridge bearings



Harriet Anne Peel

School of Civil Engineering

School of Computing

University of Leeds

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

March 2019

Declaration

The candidate confirms that the work submitted is their own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

The work in Chapter 4 of the thesis has appeared in publication as follows:

Localisation of a mobile robot for bridge bearing inspection; H. Peel, S. Luo, A.G. Cohn, R. Fuentes Automation in Construction 94 (2018) 244–256.

I was responsible for the majority of the work, including data collection, data processing and writing the paper. The contribution of the other authors was in help with data collection (by Dr Shan Luo, mentioned in the associated text) and academic contribution through ideas, editing and reviewing the paper. A copy of this paper has been included as an Appendix to this thesis.

Harriet Anne Peel

March 2019

Acknowledgements

I would like to thank the many people who have contributed to the work in this thesis in various ways.

First, to my supervisors Professor Raul Fuentes and Professor Anthony Cohn. I would like to thank them for the continuous support and commitment to this project and to my personal development; I am very grateful for the many valuable opportunities that they have extended to me. I would also like to thank the ESPRC and University of Leeds for providing the funding that has enabled me to do this project.

I would also like to thank those who have also contributed academically to this project including: Shan Luo, Mohammed Abdellatif, Jason Liu and Leo Pauly and to Mohannad Al Omari for his advice in the early stages of my project.

Thank you my fellow doctoral students in the School of Computing, the School of Civil Engineering and the Robotics at Leeds PGR Network for their friendship, encouragement and conversation (and their company at the pub) during the last four years.

Finally, I would like to thank my family and Hugh, my best possible friend, for their love and support.

Abstract

This thesis focuses on the development of a robotic platform for bridge bearing inspection. The existing literature on this topic highlights an aspiration for increased automation of bridge inspection, due to an increasing amount of ageing infrastructure and costly inspection. Furthermore, bridge bearings are highlighted as being one of the most costly components of the bridge to maintain.

However, although autonomous robotic inspection is often stated as an aspiration, the existing literature for robotic bridge inspection often neglects to include the requirement of autonomous navigation. To achieve autonomous inspection, some methods for mapping and localising in the bridge structure are required. This thesis compares existing methods for simultaneous localisation and mapping (SLAM) with localisation-only methods. In addition, a method for using pre-existing data to create maps for localisation is proposed.

A robotic platform was developed and these methods for localisation and mapping were then compared in a laboratory environment and then in a real bridge environment. The errors in the bridge environment are greater than in the laboratory environment, but remained within a defined error bound. A combined approach is suggested as an appropriate method for combining the lower errors of a SLAM approach with the advantages of a localisation approach for defining existing goals. Longer-term testing in a real bridge environment is still required.

The use of existing inspection data is then extended to the creation of a simulation environment, with the goal of creating a methodology for testing different configurations of bridges or robots in a more realistic environment than laboratory testing, or other existing simulation environments.

Finally, the inspection of the structure surrounding the bridge bearing is considered, with a particular focus on the detection and segmentation of cracks in concrete. A deep learning approach is used to segment cracks from an existing dataset and compared to an existing machine learning approach, with the deep-learning approach achieving a higher performance using a pixel-based evaluation. Other evaluation methods were also compared that take the structure of the crack, and other related datasets, into account.

The generalisation of the approach for crack segmentation is evaluated by comparing the results of the trained on different datasets. Finally, recommendations for improving the datasets to allow better comparisons in future work is given.

Table of contents

List of figures

List of tables

Abbreviations

Datasets

BSDS Berkeley Segmentation DataSet

CF Crack Forest

CSSC Concrete Structure Spalling and Crack

DRIVE Digital Retinal Images for Vessel Extraction

HRF High Resolution Fundus

STARE STructured Analysis of the Retina

Hardware, Software and Sensors

BIM Building Information Modelling

CAD Computer Aided Design

CCD Charged Coupled Device

CMOS Complementary Metal-Oxide-Semiconductor

DSLR Digital Single-Lens Reflex

EKF Extended Kalman Filter

ER electrical resistivity sensor

XML Ground-Penetrating RADAR

GPS Global Positioning System

IMU Inertial Measurement Unit

IR Infra-Red

IRT Infra-Red Thermography

LiDAR Light Detection and Ranging

LWIR Long-Wave Infra-Red

RGB-D Red Green Blue-Depth

ROS The Robot Operating System

UAV Unmanned aerial vehicle

URDF Unified Robot Description Format

XML eXtensible Markup Language

Other

2D Two dimensional

3D Three dimensional

AUC Area Under Curve

DoF Degree of Freedom

FN False Negative

FOV Field of View

FP False Positive

fps frames per second

HD High Definition

RGB Red-Green-Blue

TN True Negative

TP True Positive

Methods and Algorthims

AMCL Adaptive Monte Carlo Localisation

CNN Convolutional Neural Network

DEMs Digital Elevation Models

DSN Deeply Supervised Network (layers of HED)

FCN Fully Connected Network

HED Holistic Edge Detection

ICP Iterative Closest Point

kNN k-Nearest Neighbour

MVS Multi-View Stereo

ORB Orientated Rotated Brief

RANSAC Random sample consensus

ROC Recall Operator Characteristic

SFA Structured Forest Approach

SfM Structure from Motion

SGD Stochastic Gradient Descent

SLAM Simulateous Localisation and Mapping

SSIM Structural Similarity Measure

SGD Support Vector Machine

Chapter 1

Introduction

1.1 Automation of bridge bearing inspection

Bridge bearings are mechanical components in a bridge that constrain unwanted movement and allow the transfer the loads from the superstructure of bridges (e.g., the deck) to the abutments or intermediate supports of the bridge, which then transfer these loads to the bridge foundations. An example diagram of load transfer in a bridge bearing is shown in Figure 1.1.

Bearings are therefore an integral part of bridge structures, providing the critical link between the superstructure and substructure of the bridge (Ryan et al., 2012), and their failure can have considerable impact on the life of the bridge (J. S. Cho et al., 2014; Niemierko, 2016), leading to the overall failure of the entire structure (Aria and Akbari, 2013). It is not uncommon for bridge bearings to be replaced at high costs and disruption (e.g., Yanagihara et al. (2000)).

One of the main problems affecting bridge bearings are reflected by changes to geometry, regardless of the source of the problem or the type of bearing (Freire and de Brito, 2006; Freire et al., 2014). These problems include: out-of-position translation, rotation or deformation of the bearing. Current methods to measure changes in the bearing geometry are somewhat rudimentary and involve inaccurate and non-repeatable measurements such as (Freire et al., 2014): metric tapes, gap gauges, air bubble levels, quadrant

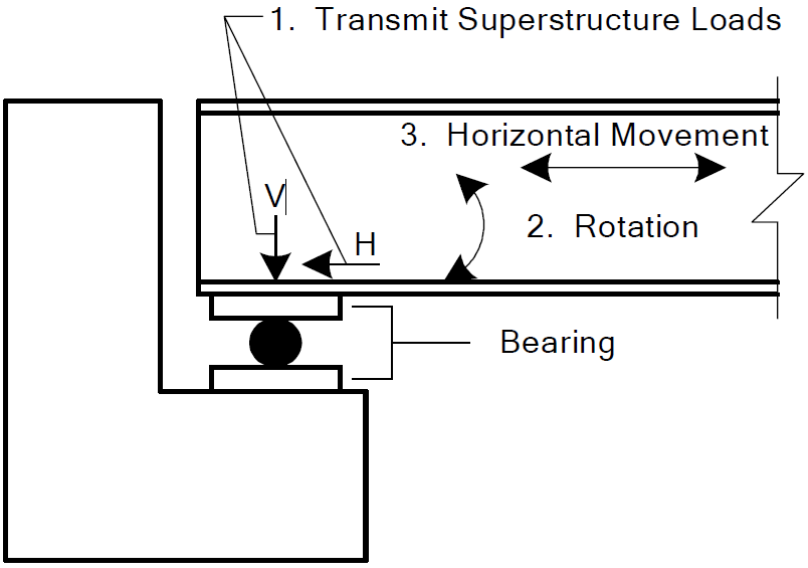


Figure 1.1: A diagram of the function of bridge bearing for load transfer from the superstructure to the substructure of the bridge. This figure is taken from Ryan et al. (2012).

rulers, compasses and verniers, levelling and topographic surveys or direct visual observations. Other, more sophisticated, systems include displacement transducers (Shibasaki et al., 2016), tell-tales (Shiau et al., 2008) and other instruments that do not measure geometry but measure the actual effect of changes on the bearing or structure directly (e.g., cells and strain gauges (Aria and Akbari, 2013; Freire et al., 2014), fibre optics (J. Liu et al., 2012), radar interferometries (Maizuar et al., 2017), magnetorheological elastomers (Behrooz et al., 2016)), but these are typically outside the norm, with the primary method of inspection being a visual inspection performed by humans (Rossow, 2006; Ryan et al., 2012).

Other anomalies in bridge bearings are related to deterioration and degradation of the structural material in and surrounding the bearing. Similar to other civil engineering structures, these anomalies typically manifest as corrosion (Hoeke et al., 2009), cracks or crushing (Freire and de Brito, 2006) that are visible during visual inspections. Such information also has the potential to be extracted from vision sensors (Noh et al., 2017; Prasanna et al., 2016). In addition, a visual inspection will also record additional anomalies, such as build up of debris and vegetation growth (Freire et al., 2014).

As described by Freire et al. (2014), the service life of a bridge is typically over 50 years, but the designed service life of a bridge bearing is shorter. In addition, the service and maintenance period of a bridge's life-cycle is estimated to be 88–$92\,\%$ of the life of a bridge (T. Chen, 2017), with some bridge bearings, such as support bearings in road bridges, frequently accounting for the major part of the maintenance budget of a bridge (Freire et al., 2014). Furthermore, the manufacture of a bearing may be discontinued or a bearing may become obsolete but remain in service. Hence, there is a need to inspect, maintain or replace bearings throughout the life of a bridge and detecting small anomalies early can prevent serious damage and cost (Freire et al., 2014). Some authors (e.g., Baimas and McClean (1998) and Spuler et al. (2015)) have shown, through a life-cycle cost analysis, that replacement of bearings due to poor maintenance can be partially prevented through appropriate inspection methods. A description of the required stages of bridge bearing inspection are outlined in Table 1.1.

Table 1.1: A description of different types and intervals of inspection methods for bridge bearing inspection, with information taken from Freire et al. (2014) and BS-EN-1337 (2003).

Type of inspection	Details
Initial inspection	Carried out at the beginning of a bridges life to give a reference for initial state of support bearing.
Regular inspection	"close visual inspection without measurements, spaced at equal, reasonably frequent, intervals" (BS-EN-1337, 2003) Carried out regularly when no anomalies are expected. Around 15 months for roadways with significant traffic or older bridges (12 months with 3 month lag to allow inspection in different seasons).
Principal inspection	"similar to regular inspection but in more detail and including precise measurement" (BS-EN-1337, 2003) Carried out every 60 months. Mostly visual with some simple measurement equipment. Measure translations and rotations of support bearings. Detect and evaluate existing anomalies. Requires qualified personnel.
Special inspection	Required if something is found in a routine or principal inspection that requires further attention. Also carried out after events caused by severe weather or due to collision e.g., with structural members.

Sutter et. al describes how the number of bridges is increasing faster than the capacity to perform inspections (Sutter et al., 2018). Moreover, the time between subsequent inspections of a bridge vary from country to country with examples in the literature varying from two years (Javadnejad et al., 2017) to five years (Sutter et al., 2018), and for specific components, such as bridge bearings, visual inspection is required every 15 months (Freire et al., 2014). Methods for increasing the efficiency of inspecting bridges and bridge bearings has been researched for a number of years, with solutions ranging from database systems to robotic systems. For example, Sommer et al. (1993) describe methods for improving efficiency in bridge inspections using probabilistic methods. Sommer et al. (1993) also highlights the variation between countries in the number and period for the required inspections, with visual inspection requirements varying from four times a year to once every five years for visual inspection. In more recent literature, data management systems with damage correlation studies (Freire et al., 2014) are being developed for bridge inspection/evaluation to incorporate building information modelling (BIM), remote sensing data, 3D geometry detection into a damage detection tool (Sacks et al., 2018; Javadnejad et al., 2017). Agnisarman et al. (2019) explores the state-of-the-art for visual inspection systems that include some automated and some human-controlled elements, with applications in infrastructure inspection. Bridge inspection was found to be a key area of interest for automation of visual inspection, with 20 of the 53 papers reviewed by Agnisarman et al. (2019) focusing on this topic.

In recent years, there has been a growing interest in using robotics for the task of bridge inspection. The literature presents clear motivations for researching and developing automated and robotic systems for bridge and bridge bearing inspection which include (Lattanzi and Miller, 2017; Jahanshahi et al., 2017; Agnisarman et al., 2019): increased safety for inspection of infrastructure that is difficult to access; improved reliability, reduced subjectivity and increased repeatability of inspection results; decreased cost and the increased frequency of inspection or reduced cost of inspection of infrastructure.

1.2 The state of the art in robotic bridge inspection

Lattanzi and Miller (2017) summarise the progress, limitations and challenges in the literature for robotic infrastructure inspection systems. The literature reviewed by Lattanzi and Miller (2017) is multidisciplinary and multifaceted in nature and shows that there is not a single correct solution to the problem of infrastructure inspection. One inspection example, such as bridge inspection, not only has different focuses for the target of the inspection (e.g., bridge deck (H. La et al., 2013) or the underside of the bridge (C. Yang et al., 2015)), but can be solved using different platforms (e.g., an unmanned aerial vehicle (UAV) (S. Chen et al., 2011) or ground-based vehicles (Van Nguyen et al., 2018)) and sensors (e.g., cameras (Kim et al., 2014) or ground penetrating radar (Le et al., 2017)). In general, the literature for robotic inspection of infrastructure has three key focuses: the specific application or target of the inspection (e.g., bridges (K. Cho et al., 2013; Pham and H. La, 2016), roads (Varadharajan et al., 2014), tunnels (Victores et al., 2011; Protopapadakis et al., 2016)), the robotic platform development (e.g., Takada et al. (2017)) and the development of sensing systems to go on-board the robotic platforms (e.g. Kim et al. (2014)). Table 1.2 summarises the different approaches in the literature that focus on bridge inspection, with a summary of the type of robot, sensors and the focus of the robotic inspection.

Much of the recent research for bridge inspection incorporates the use of UAV technology (see Table 1.2,), partly because off-the-shelf UAVs are becoming cheaper and more readily available (Lattanzi and Miller, 2017). The use of UAVs has many advantages for infrastructure inspection since UAVs with cameras are useful as an extension of a human visual inspection system, particularly for high structures that are difficult to access, which gives the benefit of being most accessible to engineers who most understand the requirements of the inspection. Furthermore, many different sensors (e.g., thermal cameras, LiDARs) can be attached as a payload to the UAV depending on the inspection tasks required. However, although UAVs are becoming a mature technology area, development is still required to increase flight time and payload capacity, and there are current legislation/certification issues, which need considering (Hoffer et al., 2017). Besides, some UAVs are unable to get close enough to the underside of the bridge due to problems with global positioning system

(GPS) sensors (Hoffer et al., 2017) and turbulence and wind gusts that occur around built-up structures, where it is possible that the human operator may not be able to take control quickly enough to maintain stability in these situations (Darby and Gopu, 2018), especially in areas where the operator has to move around to maintain eye-of-sight contact with the UAV (Hallermann and Morgenthal, 2014). These factors may make the UAV unsuitable for specific tasks, such as the inspection of the bearings of a bridge.

As well as UAV-based platforms, there are alternative approaches using legged walking robots (Mazumdar and Asada, 2009), a flying-walking combination robot (Ratsamee et al., 2016) and swarm robots (Jahanshahi et al., 2017). In one example, Akutsu et al. (2017) present a suction-based solution that was small enough to enable passage through narrow spaces in-between bridge trusses. The platform can move on both concrete and steel surface types (including surfaces that had peeled due to corrosion) using six air pads, with air provided by an air supply connected to an air pump and compressor on the ground. The authors also perform testing in a real bridge environment using a charged coupled device (CCD) camera to inspect the surface of truss members on the bridge as the robot moves along. However, there is a trade-off for these climbing and crawling systems: although they can reach more inaccessible locations in a bridge structure, the locomotion systems are more complex than the wheeled alternatives, which tend to have more cost and power requirements associated with them and also require further research to understand their long-term feasibility and reliability (Lattanzi and Miller, 2017).

Currently for robotic bridge inspection, the majority of research using ground-based robots focus on the application of bridge deck inspection. All but one of the ground-based examples in Table 1.2 are related to the same project (see examples marked with an asterisk in Table 1.2), where bridge deck inspection was implemented using off-the-shelf platform and sensors on-board a robotic platform (Lim et al., 2011; H. La et al., 2013; Gibb et al., 2017; Van Nguyen et al., 2018). In this research, sensors such as ground penetrating radar (GPR), electrical resistivity (ER) sensor and thermal and visual cameras are used to automate non-destructive testing with applications in crack detection and classification in images of concrete (H. La et al., 2014; Van Nguyen et al., 2018). The other wheeled platform in

Table 1.2 is a ground-based platform with magnetic wheels which allow the platform to attach to bridge girders. The main focus of this work was testing the magnetic system, although other sensors are integrated for localisation in future research (Pham and H. La, 2016).

Other reported research using ground-based platforms fall into the category of 'snooper truck' (Lattanzi and Miller, 2017). A snooper truck is a road or rail driven vehicle with an articulating arm and platform that allows visual inspection of the underside of the bridge. Robotic alternatives to this platform use the snooper truck as the basis for the platform, but a hydraulic boom was attached to the articulating arm that allows scanning or photography of the underside of the bridge. Oh et al. (2007) design a robotic system, mounted with cameras, to attach to this boom. Similarly, B. Lee et al. (2012) uses several remote controlled devices attached to the boom (also mounted with cameras) to perform crack detection. Sutter et al. (2018) conducted similar research, but developed a semi-autonomous robotic trolley to which the boom was mounted. In autonomous mode, the robotic trolley collects sensor data whilst moving along the bridge, using wheel encoders to monitor distance moved and a laser range finder to monitor distance to the edge of the bridge deck.

Table 1.2: A review of recent literature that focuses on bridge inspection and automation using robotics or computer vision based approaches. Related projects are marked with *.

Literature Summary			
Year	**Reference**	**On-board sensors**	**Description of approach**
Aerial vehicle			
2011	S. Chen et al. (2011)	Camera	Photo-stitching with the application of detecting cracks on a bridge deck
2014	Hallermann and Morgenthal (2014)	Camera	Evaluating flight paths, used camera images for 3D reconstruction of structure
2015	C. Yang et al. (2015)	Camera	Tested flight near a bridge, collected photographs
2016	Ellenberg et al. (2016b)	Camera	Bridge bearing deformation and crack and corrosion assessment of surrounding structure
2016	Ellenberg et al. (2016a)	Infrared thermography (IRT)	Detection of potential areas of delamination using IRT
2017	Hoffer et al. (2017)	Camera	Detection of cracks in concrete structure.
2017	Sanchez-Cuevas et al. (2017)	None	Testing ground effect on UAV control.
2017	Eschmann and Wundsam (2017)	Camera, two-dimensional (2D) LiDAR, long-wave infrared (LWIR)	3D, geo-referenced reconstruction of a bridge, crack detection, humidity detection using LWIR.
2018	Hiasa et al. (2018)	HD camera, IRT	Detection of simulated cracks (printed on paper) on bridge deck, IRT not used for detection application.
2018	Khaloo et al. (2018)	Two cameras	3D model produced from images taken on a drone and compared to 3D terrestrial LiDAR data.

Continuation of Table 1.2

Year	Reference	On-board sensors	Description of approach
Wheeled			
2011	Lim et al. (2011)*	LiDAR, high resolution camera	Crack detection in bridge deck
2014	H. M. La et al. (2013)*	Ultrasound, Ground penetrating radar (GPR), electrical resistivity sensor (ER)	Developing localisation for bridge deck inspection
2014	H. La et al. (2013)*	Ultrasound, GPR, electrical resistivity sensor (ER)	Crack detection in bridge deck
2013	H. La et al. (2014)*	LiDAR, GPR, panoramic camera, GPS, surface camera, acoustic array.	Crack detection in bridge deck
2017	Le et al. (2017)*	GPR, camera, ER	GPR for detection of corrosion of rebar in bridge deck, ER for corrosion and concrete deterioration
2018	Van Nguyen et al. (2018)*	GPR, camera, IRT	Crack detection on bridge deck using deep learning
2016	Pham and H. La (2016)	Depth camera, camera, LiDAR	Tested effectiveness of magnetic wheeled platform. Initial steps towards mapping using LiDAR
Gibb et al. (2018)	CNN with genetic algorithm to optimise CNN structure	Patch-based classification Asphalt	Developed own dataset, but perhaps made comparison to Y.-J. Cha et al. (2017) $3,000:1,500$ for testing, $1,500$ for training 256×256

Continuation of Table 1.2

Year	Reference	On-board sensors	Description of approach
'Snooper truck'			
2012	B. Lee et al. (2012)	GPS, cameras, ultrasound	Road vehicle with actuated boom. Remote control devices fitted with cameras are attached to the boom and are used to inspect the underside of the bridge.
2018	Sutter et al. (2018)	Motorised cameras	Mobile vehicle for road or train bridge with actuated boom, fitted with cameras, to inspect underside of bridge. Extensive testing and certification performed.
Other			
2011	R. R. Murphy et al. (2011)	Cameras, sonar	Unmanned marine vehicle, used to create map of corrosion
2013	K. Cho et al. (2013)	None	Build and testing control system of robot to move along suspension cables
2014	Ward et al. (2014)	Depth camera	Testing control system of magnetic robot and obtaining 3D reconstructions of the environment.
2016	Ratsamee et al. (2016)	None	Testing control system of hybrid flying crawling robot.
2017	Jahanshahi et al. (2017)		
2017	Takada et al. (2017)	Camera	Development of a magnetic tracked robot for steel bridge inspection. Camera used for autonomous navigation

Continuation of Table 1.2

Year	Reference	On-board sensors	Description of approach
Computer vision			
2008	J. H. Lee et al. (2008)	Camera	Machine vision system for crack detection in concrete bridge structure
2014	Kim et al. (2014)	Camera	Machine learning for rust detection in steel bridge structures
2014	L. Li et al. (2014)	Camera	Image processing for crack detection in structural concrete
2014	Lattanzi and Miller (2015)	Camera	3D reconstruction of bridge pillar
2015	Sakagami (2015)	IRT	Crack detection in metal bridge structure using thermal images
2017	L. Yang et al. (2017)	Camera	Deep learning with fine-tuning for crack detection in concrete bridge structure
2017	Y. J. Cha et al. (2018)	Camera	Deep learning for corrosion detection in metal bridge structures
2017	Hoskere et al. (2018)	Camera	Deep learning for automated bridge component recognition
2018	Sato (2018)	DSLR Camera	Image processing using V-shaped detector for segmentation of cracks in concrete bridge pier
2019	S. Chen et al. (2018)	Camera	Outlier detection in point clouds of bridges

1.3 Sensors for bridge inspection

As outlined in Table 1.1, the European Standard (BS-EN-1337, 2003) describe regular inspection as: "close visual inspection without measurements, spaced at equal, reasonably frequent, intervals", with inspections occurring at least as often as the bridge structure is assessed. Specifically, the standard requires that the bearings are assessed for visible defects

including: cracks, incorrect position of the bearing, unforeseen movements and deformations of the bearing and visible defects on the bearing or surrounding structure (BS-EN-1337, 2003).

Of the approaches and platforms described in the literature summarised in Table 1.2, the sensor most commonly used is a monocular RGB camera, which was used in every case except where the focus of the paper was the development of the control system of a novel platform (e.g., Ratsamee et al. (2016) and K. Cho et al. (2013)). The use of a camera sensor matches the requirement for visual inspection of bridges, as camera data can be reviewed later by a human operator for visible defects. In addition, three-dimensional (3D) reconstructions of the environment were a desirable outcome of inspection data from camera sensors. For example, Lattanzi and Miller (2015) and Khaloo et al. (2018) used Structure from Motion (SfM) to reconstruct structural elements of bridge, Ward et al. (2014) used a depth camera to obtain a 3D point cloud of the environment and Eschmann and Wundsam (2017) used SfM to create a geo-referenced reconstruction of the bridge for digital inspection, which was then added to a database with other defect information relating to bridges.

Of the papers reviewed in Table 1.2, the main focus of inspection was the structural condition or material degradation (Noh et al., 2017; Prasanna et al., 2016; Yeum and Dyke, 2015; Akutsu et al., 2017; Pham and H. La, 2016; C. Yang et al., 2015) of the bridge, typically through detection of cracks or corrosion in photographs. One specific application was crack detection in the structure of the bridge and the bridge deck. In the most basic approaches to crack detection, the robotic platforms were used to collect photographic data that was later reviewed by a human operator (S. Chen et al., 2011; Hoffer et al., 2017; Hiasa et al., 2018). More sophisticated approaches implemented image processing and computer vision methods, such as median filtering of grey-scale images (Ellenberg et al., 2016b) or edge detection using a Laplacian of Gaussian filter or the Canny algorithm (L. Li et al., 2014). Variability of lighting conditions caused shadows and poorly illuminated images and vibrations of the UAV robotic platform caused image blurring (Hallermann and Morgenthal, 2014; Hoffer et al., 2017; S. Chen et al., 2011). Van Nguyen et al. (2018) implemented a

method for classification of image patches that contain cracks using a convolutional neural network (CNN). In research for crack detection in concrete and asphalt pavements, machine learning approaches (including CNNs) has become a common approach; the literature for crack detection in concrete structures will be reviewed further in Chapter 5.

Only one of the reviewed papers developed an approach specifically for bridge bearing inspection (Ellenberg et al., 2016b). Ellenberg et al. (2016b) used a small UAV equipped with two different cameras and a laboratory-based set-up of a bridge bearing to monitor the deflection of a rubber bearing and a steel bearing by manually counting pixels in frames before and after a deflection to the bearing had been applied. The laboratory set-up was useful for replicating the predicted defects in the bearing, although very few experimental results were reported in this instance. In the same paper, the authors also conducted experiments for monitoring corrosion and cracking in the structure around the bearing (Ellenberg et al., 2016b), although most of the results were gathered on surfaces, such as brick and breeze block walls, which are not representative of the structure surrounding a bridge bearing.

1.4 Robotics for other inspection applications

The use of robots for inspection is also developing in several other research fields. For example, robotic platforms have been used in the nuclear industry for half a century, with much of the research focused on investigating the geometry and condition of unknown environments. Many of the robotic applications in nuclear inspection are also tele-operated, although the literature presents some work for introducing autonomous navigation using LiDAR (M. Lee et al., 2018), although successful use is still limited. In addition, use of tools such as the Robot Operating System (ROS) (see Section 3 for more details) were trialled (Jalón-Monzón et al., 2016) and showed potential benefits for integrating the localisation and mapping algorithms into the robotic system.

Autonomous robotic navigation has also been evaluated in the agriculture research field. Agricultural environments also tend to be unstructured and changeable and there are some requirements for human interaction (Bechar and Vigneault, 2016). However, since the

environments in which these robots are required to operate tend to be open agricultural environments, there are many opportunities for deploying off-the-shelf ground vehicles. Post et al. (2017) develops a robotic system for autonomous navigation of agricultural fields using open-source modules developed using ROS for their robotic system. GPS, a 2D laser scanner, a stereo camera and IMU are mounted on a robotic ground platform. Sensor fusion was used to combine visual odometry, laser-based odometry and wheel odometry using an Extended Kalman Filter, which was used for localisation of the robotic platform. 3D mapping was implemented using a simultaneous localisation and mapping (SLAM) approach called RTAB-MAP, but was found to produce poor reconstructions in outdoor environments (Post et al., 2017).

Ground-based platforms are often used in tunnel inspection due to the planar motion that is required when moving through the tunnel. An autonomous system has been developed as part of the ROBO-SPECT project for tunnel inspection (Protopapadakis et al., 2016; Menendez et al., 2018), where a mobile platform with automated crane is mounted with a robotic arm, 3D vision system, laser profiler, ultrasonic sensors and tele-operated cameras. Non-destructive evaluation, including crack, spalling and efflorescence detection, of the tunnel was performed using contact sensors mounted on the robotic arm and cameras. Automated navigation of the mobile platform was implemented by combining a wall-following approach with SLAM of the tunnel using laser data and beacons that are placed along the tunnel. Again ROS was used for integration of sensor communications and Gazebo, a simulation environment (Koenig and Howard, 2004), was used to perform initial experiments. The system has been trialled alongside moving traffic without human intervention (Menendez et al., 2018).

Due to a lack in maturity of the research in automation of robotic systems for civil infrastructure inspection, the main use for robotic systems presented in the literature are aids to current human inspectors. Of the papers reviewed by Agnisarman et al. (2019), most of the robotic systems were tele-operated although UAV platforms were able to use GPS way-points for autonomous operation. Where localisation was addressed, the Extended Kalman Filter (EKF) algorithm was used to improve dead-reckoning approaches

using ultrasound or GPS as the main navigation sensor (Gibb et al. (2017) and Ridao et al. (2010), respectively), although EKF has also been performed with various sensors, including surveying-grade optical sensors in McLoughlin et al. (2018). Lattanzi and Miller (2017) states the reasons for lack of autonomy in civil inspection environments is because the field of navigation in general robotics requires further development to become sufficiently mature, but also due to difficulties of the inspection environments, such as hazards, obstacles and variable environmental conditions. However, autonomous navigation has not been addressed comprehensively in the current literature and further research is required to better understand the current limitations of navigating autonomously in these environments.

D. Liu et al. (2014) also summarises the research and engineering challenges that exist for the use of robotics in infrastructure inspection. D. Liu et al. (2014) states that a key research barrier for robots to operate autonomously for civil infrastructure is: 'robot environmental awareness, localisation and mapping of true 3D complex environments such as trusses'. Other research and engineering challenges highlighted by D. Liu et al. (2014) include: navigation of infrastructure such as bridges, which are complex, but also compact, 3D environments; the robotic system needs to be able to deal with uncertainty, such as collision avoidance in real time and have a fail-safe design; the robots may need to interact with humans, either operators or the public; the robots to be able to perform tasks with high payloads but to maintain dexterity in complex environments.

1.5 Summary

Existing literature presents a need for the development of more efficient approaches to bridge inspection to meet the increasing demands of inspection due to ageing infrastructure (Freire et al., 2014) and an increasing number of bridges (Sutter et al., 2018), with many researchers proposing opportunities for robotic bridge inspection. Maintenance and inspection of bridge bearings is the major part of the budget for bridge maintenance (Freire et al., 2014) but, although there is interest in increasing the efficiency of this process, there is

very little focus on automating this inspection in the literature, though it is often highlighted as a long-term goal.

Technology is developing to allow inspection of civil structures to be performed remotely, mainly using visual sensors. The main tool used in the literature for robotic inspection of the bridge is the monocular RGB camera, which is then commonly used for 3D reconstruction using methods including SfM or for detection of material degradation, such as cracks. In addition, the use of robotic platforms and UAVs is allowing the development of inspection methods of structures that are otherwise difficult or dangerous to reach for human inspectors.

Although the literature suggests an interest in the development of robotic or automated inspection of bridges, there is limited research in this area, particularly for navigation of robots in inspection environments, although autonomous navigation was stated as a desirable goal for future research. Of the research into aided bridge inspection, only a couple of examples consider autonomous navigation as an area of research, most research focused primarily on the platform development or computer vision methods for inspection. Advances in automated infrastructure inspection often rely on laboratory-based environments and there is limited comparison to real-world environments. Therefore, research is needed to develop the methods that are used for navigation in these structures.

1.6 Problem statement, aims and scope

Reflecting on the literature reviewed in this chapter, this thesis will focus on the development of a robotic platform for bridge bearing inspection. Specifically, this thesis will focus on evaluating the effectiveness and suitability of existing methods for localisation and mapping for navigating in a real bridge inspection environment, using tools that are implemented more generally in robotics research, but not tested for bridge inspection applications. Computer vision approaches for inspection of the bridge bearing environment will also be considered, with a focus on crack detection in concrete and 3D reconstructions of the environment, both using 2D camera images.

In order to meet this aim, the following will be undertaken:

- Investigate and identify existing sensors and methods for localisation and mapping that could be used for navigating in a real inspection environment.
- Develop a robotic platform for the purpose of testing localisation and mapping algorithms in a real bridge environment.
- Use existing tools that are commonly used in robotics research for inspection applications (e.g., the Robot Operating System).
- Develop a mapping approach that can make use of existing surveying data.
- Test existing state-of-the-art localisation and mapping techniques in a real environment and compare to laboratory experiments.
- Use existing data from bridge surveys to develop a simulation environments in order to test and prototype in different bridge configurations.
- Test machine learning and deep learning approaches for segmentation of cracks in photographs of concrete.
- Identify evaluation metrics from the literature and use these metrics for the evaluation of the machine learning and deep learning methods for crack segmentation.
- Identify and compare different datasets and the effects of training and testing using a mixture of these datasets on the chosen methods.

1.7 Thesis structure

This thesis is organised into six main chapters, split into two primary research topics which cover the localisation and mapping of a robotic platform in a bridge bearing enclosure followed by the inspection of the structure surrounding a bridge bearing. A summary of the chapters in this thesis is as follows:

Chapter 1 : Introduction.
A review of the current literature for robotic inspection, with a focus on bridge inspection.

Chapter 2 : Sensors, SLAM and localisation.
Sensors that are used in robotic navigation are reviewed in this chapter, followed by an overview of the background and methodology of SLAM and a description of the SLAM and localisation approaches that are tested in this thesis.

Chapter 3 : Robotic system development.
A description of the robotic platform that was used in this work, including the choice of sensors and hardware and a description of relevant tools.

Chapter 4 : Localisation for a bridge bearing inspection robot.
Existing localisation and SLAM approaches are evaluated and compared using the robotic platform described in Chapter 3 in both a laboratory environment and a real bridge environment. In addition, existing inspection data was used to generate simulation environments for testing navigation algorithms to provide a proof-of-concept approach for testing robots in different configurations of bridge structures.

Chapter 5 : Application of computer vision techniques to visual inspection tasks.
In this chapter, existing literature for crack detection in concrete was reviewed. Crack detection was then performed through the application of deep-learning algorithms to segmentation of cracks in photographs of cracks, including investigation of mixed datasets using a retinal image database.

Chapter 6 : Summary and Conclusions.
This chapter summarises the results and findings of Chapters 4 for mapping and localisation and the findings of Chapter 5 for crack detection.

Chapter 2

Sensors, SLAM and localisation

2.1 A review of sensors for robotic navigation

Sensors on-board robots typically have one of two roles (Agnisarman et al., 2019): to aid data collection for some task (e.g., inspection of infrastructure) or for robotic navigation. In both cases, external information about surroundings is collected. For navigation, the robot uses the data collected from these sensors to understand the features in its current environment to find its position and, depending on the application, to create a map of the environment for future use. Many different sensors can be used to provide information about the robot's environment. However, the choice of sensor is highly dependent on the operating environment of the robot with other relevant factors including sensor range and resolution, power consumption, weight or size of the sensor, navigation algorithms and sensor price (Zaffar et al., 2018).

In general, sensors used for autonomous navigation can be defined as exteroceptive or proprioceptive, where exteroceptive sensors collect information about the surroundings external to the robot and proprioceptive sensors collect information internal to the robot (e.g., speed, rotation, acceleration or displacement, but also battery levels or robot arm positions). Sensors are also classified as passive or active, where active sensors emit energy into the environment and measure the reaction from the environment (Everett, 1995). Generally, proprioceptive sensors cannot be used for reliable long-term navigation due to

accumulation of errors (K. Murphy, 2000). External information from the environment can be used to correct these errors, but they are often used to assist or improve the position estimates made using exteroceptive sensor information (Everett, 1995). In general, proprioceptive are used for dead reckoning approaches to navigation, whereas exteroceptive sensors are used with map-based methods, although many approaches combine the two. In this chapter, the main types of sensors used for navigation are described and the main factors that should be considered when choosing a sensor for robotic navigation are outlined; methods for navigation are then described.

2.1.1 Exteroceptive sensors

Ultrasonic sensors

Ultrasonic sensors are time of flight sensors that transmit ultrasonic pressure waves and measure the time between the emitted pulse and the returning signal (that has been reflected off some surface in the environment), to calculate the distance to landmarks in the environment (Everett, 1995). Generally, these sensors are used to give point distance measurements (e.g., Peel et al. (2016)), but can be arranged as an array of multiple sensors rotated to detect objects in a $360°$ range (e.g., Großmann and Poli (1999)). Sound waves from the ultrasonic sensor propagate in a cone like manner, which can affect the detection of obstacles in the environment because the cone increases with distance from the sensor. Measurements from ultrasonic sensors are also affected by specular reflection and crosstalk with other ultrasonic sensors and may also be affected by environmental conditions such as changes in temperature and strong winds (Vivet et al., 2013). A similar technology that is less affected by environmental conditions is RADAR (RAdio Detection And Ranging), which propagates electromagnetic waves rather than sound waves. RADAR is less common in robotic applications but can also be used for robotic navigation (e.g., Vivet et al. (2013)).

LiDAR

One of the main types of Light detection and Ranging (LiDAR) sensors used in robotics is triangulation LiDAR (as opposed to phase-shift LiDARs or time-of-flight measurements). A point of light (e.g., laser or infrared) is reflected off objects in the environment. These reflections are measured at the detector, which comprises of a charged coupled device (CCD) sensor; the angle of the incoming beam is determined from its position on the sensor and the angle of the out-going light. This laser beam can then be rotated to obtain the position of objects in the environment in a $360°$ sweep. Since the beam has small divergences over large distances, LiDAR sensors are more precise and tend to have a higher range than ultrasonic sensors (Siegwart and Nourbaksh, 2011). Typically, LiDAR sensors contain a mirror that is rotated mechanically and are available for mobile robots in 2D, 2.5D or 3D configurations (Markom et al., 2016; Arth et al., 2015; Nüchter et al., 2018) (where 2.5D typically incorporates rotation of a 2D LiDAR to obtain 3D information), with fields of view up to $360°$, 2D LiDAR can also be used for 3D sensing by tilting the LiDAR (e.g., Chong et al. (2013)). There are some limitations for using LiDAR when surfaces in the environment are highly reflective because it causes coherent reflection of the light energy (Siegwart and Nourbaksh, 2011) or with materials such as glass where the emitted light defracts through the surface. The required spatial resolution and dimensionality (i.e., 2D or 3D) leads to an increase in power consumption and cost of the sensors (Zaffar et al., 2018). The mechanical components in LiDARs may also wear out over time, which should be taken into consideration when designing a system for long-term use. (Zaffar et al., 2018)

Monocular camera

A monocular camera refers to a single camera sensor (e.g., an Red Green Blue (RGB) camera sensor) and is a common choice of sensor on-board a robotic platform because visual camera data has many uses, including inspection applications. Monocular sensors also tend to be compact, cheap and have low power requirements (Zaffar et al., 2018). As described by Siegwart and Nourbaksh (2011), visual sensors may be affected by changes in

lighting and glare and reflection off surfaces in the environment, a monocular camera also cannot recover depth in a scene without additionally provided scale information.

RGB-D camera

Red Green Blue-Depth (RGB-D) cameras combine a typical monocular camera with an infrared (IR) transmitters and receivers in order to provide estimated depth information at each pixel of the RGB camera image. The RBG-D sensor was not developed with robotics in mind, but has been used in many applications in recent years, particularly for robot navigation in indoor environments (e.g. Henry et al. (2012)). There are two main types of RGB-D camera, one which uses the time-of-flight of infrared and one which uses a structured light approach. The time-of-flight RGB-D cameras work in a similar way to the LiDAR by calculating the time taken for the infrared light to bounce off objects in the environment. The structured light sensors work by interpreting the distortion of a pattern of infrared light when projected onto objects in the environment when compared to a reference pattern at a known depth. Although RGB-D sensors have a range of up to $4.5\,\mathrm{m}$, their performance in outdoor environments can be poor since bright sunlight can obscure the infrared light (Kulich et al., 2017).

2.1.2 Thermal camera

Thermal cameras use infrared radiation emitted from objects to create images, rather than detecting light that bounces off an object (as in a monocular camera), can be used in low-light or no-light environments and have been used for inspection applications such as power line monitoring Luque-Vega et al. (2014). Although RGB-D sensors also use infrared, thermal cameras are different to RGB-D sensors: RGB-D sensors emit a structured infrared pattern in the near-infrared range ($700\,\mathrm{nm}$ to $1400\,\mathrm{nm}$), whereas thermal imaging detects radiation with wavelengths between $5500\,\mathrm{nm}$ to $14\,000\,\mathrm{nm}$ (visual light is in the range of $400\,\mathrm{nm}$ to $700\,\mathrm{nm}$). In general, thermal cameras work in a similar way to a monocular camera, with an array of infrared sensitive sensors that can then be interpreted as pixels in an image. However, sensor resolution tends to be much lower (for the same cost) compared to

typical monocular camera sensors. In addition, thermal sensors are sensitive to heat in the environment – more accurate thermal use active cooling (Nilsson, 2008).

Stereo camera

In contrast to a monocular camera, a stereo camera has two sensors mounted at a separation (known as the stereo baseline) in order to use the disparity between the two captured images to calculate the depth information of a scene; this configuration is inspired by the human vision system (Everett, 1995). Unlike monocular cameras, stereo cameras can use the disparity of the two images to calculate the scale of the scene. To calculate the depth, features are detected in the camera scene, these features are then matched between the left and right stereo cameras and epipolar geometry is used to triangulate the depth of these features in the scene (Everett, 1995). The length of the stereo baseline is important since the disparity of the scene can only be calculated where the field of views of the two cameras overlap (Everett, 1995). Problems may also arise due to occlusion of objects, where objects are only visible in one of the two cameras, similarly stereo cameras are also affected by changes in lighting conditions and surface glare (Siegwart and Nourbaksh, 2011).

GNSS and GPS

Global Navigation Satellite System (GNSS) is the over-arching term for systems that use data from satellites to determine their current position. GNSS calculate the current position of the receiver through trilateration using the synchronised position and time information broadcast from satellites orbiting the earth. Therefore, GNSS is a passive exteroceptive sensor because it uses external landmarks to determine the position of the robot (Siegwart and Nourbaksh, 2011). A minimum of four satellites are required to find the current position. Global Positioning System (GPS) is one implementation of GNSS that uses the NAVSTAR constellation of satellites (Parkinson and Gilbert, 1983). The accuracy of typical GPS sensors is of the order of metres (around $5\,\mathrm{m}$ for a smart phone GPS in 2015 (van Diggelen and Enge, 2015)), but accuracy of the order of millimetres to centimetres (relative to a known base station) can be obtained using real-time kinematic methods (e.g., Heinrich

et al. (2018)). However, GNSS and GPS technology is limited when navigating in indoor or obstructed environments (Siegwart and Nourbaksh, 2011).

2.1.3 Proprioceptive sensors

Inertial sensors

An inertial measurement unit (IMU) typically combines a gyroscope and an accelerometer to give measurements of rotational velocity and linear acceleration in six degrees of freedom; these sensors can also be combined with a magnetometer to give heading measurements (Ahmad et al., 2013). The combination of these sensors is used to reduce drift in the individual sensor errors. Accelerations are measured in each of the three directional axes and integrated over time to calculate position and velocity. The accelerometer is used to calculate linear acceleration, a Gyroscope preserve their orientation relative to a fixed reference frame and are used in the IMU to calculate angular velocity and rotational angle (pitch, roll and yaw) (Ahmad et al., 2013). The magnetometer is also used to calculate rotational angle in yaw and angular velocity, but can be affected by nearby ferromagnetic material (Ahmad et al., 2013). More recent applications in robotics have implemented micro-electrical IMUs, which tend to be low-cost, compact and have low processing power (Ahmad et al., 2013).

Odometry

In order to create a motion model of the robot (see Section 2.2), some means of describing the motion of the robot is required. Wheel encoders are placed on the robot and as the wheels rotate the sensors monitors the change in light shone through the encoder disk and translates this rotational movements into the equivalent forward motion of the robot. Although the optical encoders tend to have accuracies close to $100\,\%$ (Siegwart and Nourbaksh, 2011), errors can be introduced due to the robot wheels slipping on the surface or moving over rough terrain (Everett, 1995); hence odometry alone is not reliable for robust navigation of the robot.

2.2 Methods for autonomous navigation – SLAM

Simultaneous Localisation and Mapping (SLAM) is a fundamental problem in robot navigation in which a robot builds a model of an unknown environment whilst concurrently determining the state of the robot within that environment. State typically refers to the pose and orientation of the robot (for a robot operating on a planar surface, the current configuration of the robot can be described using the 2D cartesian coordinates x and y and the orientation in yaw, θ), but may include variables such as the velocity of the robot and the location of landmarks in the surrounding environment (Siciliano and Khatib, 2016). A depiction of the SLAM problem is shown in Figure 2.1. The robot begins at the position labelled 1 (see Figure 2.1) and make measurements of landmarks m_i and m_j using onboard sensors (corresponding to $z_{i,t-1}$ and $z_{j,t-1}$) and, using this information, estimates its position in the environment (shown in green in Figure 2.1). The robot then moves to position '2' using control input u_{t-1}. As the robot moves through the environment (from '1' to '4' in Figure 2.1) noise in the measurements made by the robot causes a divergence in the estimated and true positions of the robot and landmark positions. The true position of the landmarks and the robot's position relative to these landmarks is never known, only estimated from sensor measurements and the control inputs given to the robot (Durrant-Whyte and Bailey, 2006) and SLAM algorithms are required to minimise the error between the true and estimated position of the robot.

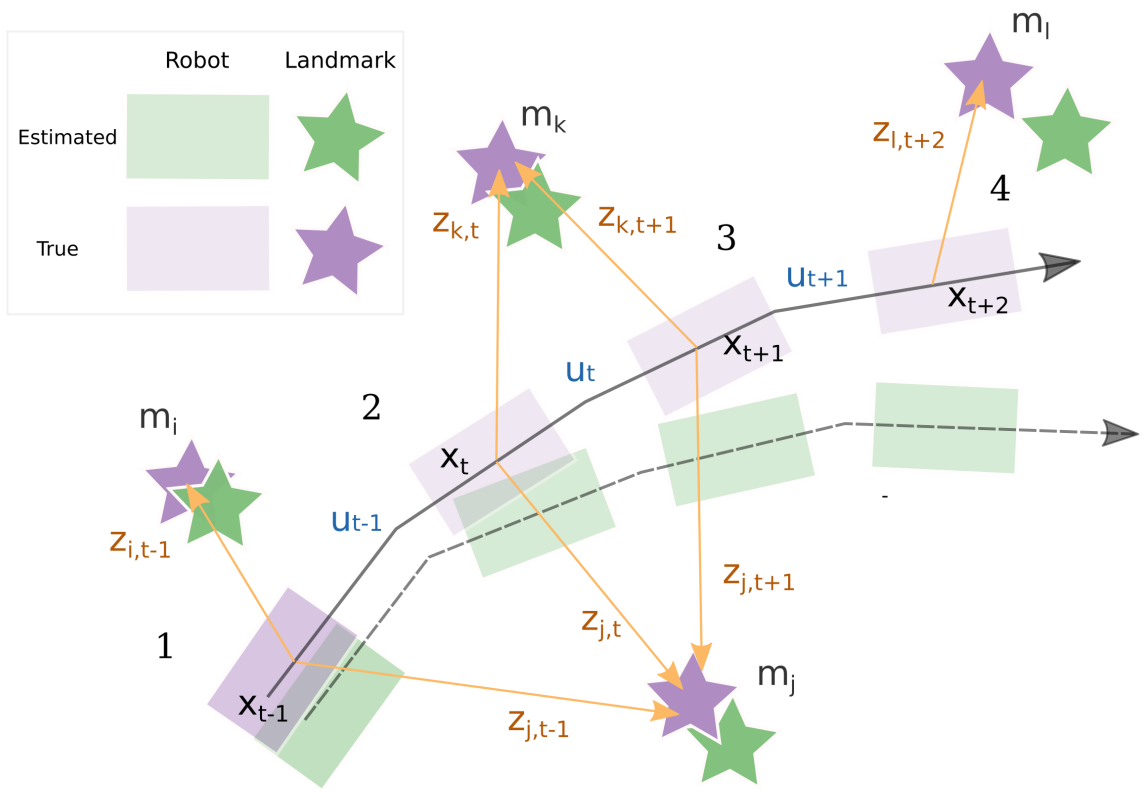


Figure 2.1: This image is based on the figure from Durrant-Whyte and Bailey (2006) and shows a robot moving through a scene over four subsequent stages (from '1' at $t-1$ to '4' at $t+2$) using a set of control inputs (u_{t-1}–u_{t+2}) and making estimates about its location based on observations of features in the environment (m_i) from on-board sensors (z_t) at each time step. The estimated position of the robot and landmarks is shown in green and the true position is shown in purple. The location of landmarks and the control inputs are estimated and there is error between the true robot position and the estimated robot position due to inherent noise in the control inputs and sensor measurements, which diverges over time. SLAM algorithms can be used to improve the error between the true and estimated positions of the robot and landmarks in the environment.

In Figure 2.1, the control input to the robot does not match the true motion of the robot due to inherent uncertainties caused by environmental factors, such as higher than expected floor friction or un-detected obstacles. Similarly, there are errors in sensor measurements; for example, errors occur for range-based sensors (i.e., a LiDAR or ultrasonic sensor) from cross-talk or reflection of the emitted energy off of an object, which introduces uncertainty with respect to the position of objects and number of objects in the environment. To

account for these uncertainties, the SLAM problem is defined in a probabilistic manner as follows (Grisetti et al., 2010; Durrant-Whyte and Bailey, 2006):

$$p(x_t, M \mid z_{0:t}, u_{0:t}, x_0) \tag{2.1}$$

where x_t is the state of the robot at time t, M is the map as a vector of landmarks $(m_i, m_j \dots m_l)$, $u_{0:t}$ is the set of control inputs from the first to current time step, $z_{0:t}$ is the set of observations of the environment from the sensor data over all time steps, x_0 is the initial robot state and $p(x_t, M \mid z_{1:t}, u_{1:t})$ is the conditional probability of a particular robot state and the position of set of landmarks in the environment at time t given the control inputs and sensor readings from the initial time step to the current time step t. This probability posterior is also referred to as the belief of the robot, which reflects the internal estimate of the current pose of the robot and the surrounding landmarks (Siciliano and Khatib, 2016).

The Bayes filter algorithm can be used to calculate belief distributions from control data and measurement inputs (Siciliano and Khatib, 2016). This algorithm has two main stages: a prediction stage and a correction stage. During the prediction stage, the control input is applied to the robot state to give the new estimated state; then in the correction stage, sensor measurements are used to correct the error in the estimated state. In order to compute the posterior in Equation 2.1, two models that describe the motion of the robot and the expected sensor measurements are required. The motion model (also called the state transition probability (Siciliano and Khatib, 2016)), is required to represent the control input to the robot between two states, and a measurement model, specifies how the measurements at a given point in time are generated from a given environment. The general formulation for the motion model is:

$$p(x_t | x_{t-1}, u_t) \tag{2.2}$$

In practice, there are two commonly used motion models: one generated using odometry and one generated using velocity information. An odometry motion model typically

requires a wheel encoder sensor (although other sensors such as LiDAR can be used) to estimate the distance travelled for a control input, and a velocity-based model uses the input velocity commands to the robot and the time taken to complete the motion to calculate the distance travelled based on dead-reckoning, i.e., accumulating the velocity commands from a start location.

Next, the measurement model is considered. The general formulation of the measurement model for filter based SLAM is defined as:

$$p(z_t|x_t, M) \tag{2.3}$$

Sensor models can be developed from the known characteristics of particular sensors. For range-based sensors, a sensor model can be defined that includes expected noise, such as measurement noise, unexpected obstacles, random measurement and maximum range errors as a mixture of probability densities (Siciliano and Khatib, 2016).

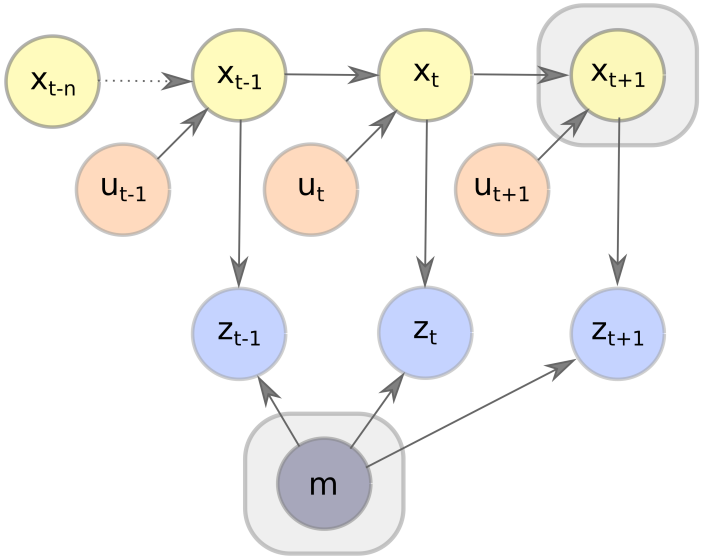
There are two alternative formulations for SLAM (Siciliano and Khatib, 2016). The posterior in Equation 2.1 involves estimation of variables at the current time, t, and is referred to as online SLAM; the alternative is full SLAM where the posterior is calculated over the full robot path, together with the map, rather than at a single point in time, i.e.,:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{2.4}$$

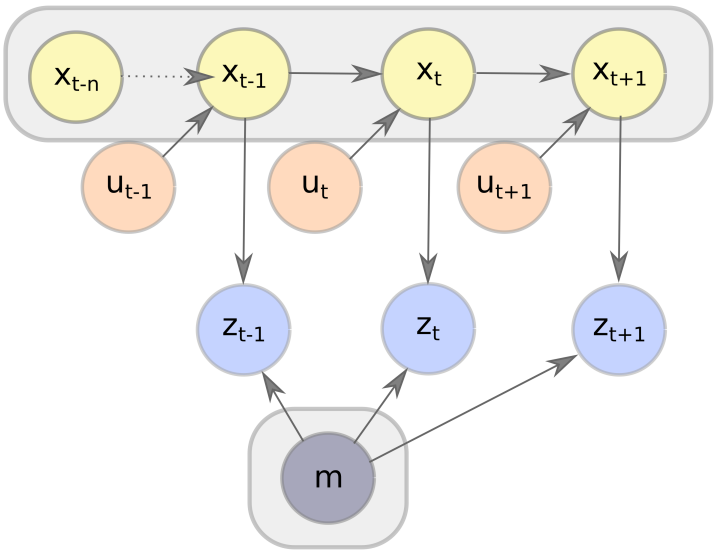
The difference between online SLAM and the full SLAM problem is represented in Figure 2.2 by the highlighted nodes in the Bayesian network representation of the SLAM problem. In addition, the online SLAM solution can be obtained incrementally integrating out poses from the full SLAM problem as follows:

$$p(x_t, M \mid z_{1:t}, u_{1:t}) = \int \int \dots \int p(x_{1:t}, M \mid z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1} \tag{2.5}$$

The difference between the full and online representations of SLAM is important when implementing algorithmic solutions. For example, EKF SLAM, which will be addressed in



(i): Bayesian network of the online representation of SLAM



(ii): Bayesian network of the full representation of SLAM

Figure 2.2: This image is adapted from Siciliano and Khatib (2016) and shows two Bayesian network representations of SLAM. u_t are the control inputs, z_t are the sensor readings, x_t is the robot position in time and depends on the current robot pose and control input and m is the map which depends on the observed readings over time. In Figure 2.2i, for online SLAM the posterior is estimated over the current pose, whereas for full SLAM, in Figure 2.2ii, all of the poses over time are used to calculate the posterior.

Section 2.2.1, is an online SLAM implementation and is characterised by an incrementally updating solution, whereas FastSLAM, addressed in Section 2.2.3, is a type of full SLAM and uses particles to test many different possible solutions at once. In addition, different SLAM algorithms make different assumptions about the nature of the environment in which the algorithm is used. For example, one common assumption is that the world is static, so that none of the landmarks in the environment move and that the past and future states are independent if the current state, x_t, is known. Another assumption may be that the correspondence between sensing data and landmarks (i.e., the data association problem) in the environment is known. Other differences between SLAM implementations include: the type of map that is generated (feature-based or metric-based), the number of robots that can be used (i.e., single-robot and multi-robot SLAM) and active and passive approaches, where path-planning is used to increase the accuracy of the map.

It is infeasible to directly calculate the full posterior in Equation 2.1, as the parameter space of poses of the robot and map landmarks is highly dimensional; there are also a large number of discrete correspondences between landmarks that have to be calculated (Siciliano and Khatib, 2016). Therefore, solutions to the SLAM problem find appropriate representations of the motion and measurement models in order to allow efficient and consistent computation of the posterior distribution. The solutions to the SLAM problem, are often split into three categories, which then form the building blocks of other methods. Namely, these three categories are: filtering methods, graph-based methods and particle-based methods, which will be described in turn in the following sections.

2.2.1 Filter-based approaches: the Extended Kalman Filter

In the review of robotic inspection of infrastructure in Chapter 1, a common approach for both localisation and SLAM were based on the Extended Kalman Filter (Cox and Wilfong, 1990). EKF SLAM (Smith et al., 1987; Moutarlier, 1989; Philippe and Chatila, 1990) and other filter-based methods, were some of the first methods for SLAM, but have become less popular recently due to limitations for computational properties (Cadena et al., 2016;

Siciliano and Khatib, 2016) related to the number of landmarks that can be tracked in the environment at one time.

EKF requires a landmark based representation of the environment, where objects can be represented as points in space. A single state vector is used to estimate the locations of the robot and the set of features in the environment. A covariance matrix is used to represent the uncertainty in the estimates of the robot and feature locations (Siciliano and Khatib, 2016; Durrant-Whyte and Bailey, 2006). The state estimates are represented by a multivariate Gaussian:

$$p(x_t, M, c_t \mid z_{1:t}, u_{1:t}) = \mathcal{N}(\mu_t, \Sigma_t) \tag{2.6}$$

where μ_t is the vector containing the estimate of the position of the robot and landmarks in the environment and Σ_t is the covariance matrix of the expected error in μ_t. In addition, non-linear functions with Gaussian noise are used to define the measurement and motion models. The motion and measurement models are then linearised about the current state estimate using Taylor series expansions. However, linearising the non-linear motion models can lead to divergence in the errors in estimated position, so that convergence is only guaranteed in the linear case (Durrant-Whyte and Bailey, 2006).

A schematic example of EKF SLAM for online SLAM is given in Figure 2.3. The robot moves through the environment starting from a known pose (Figure 2.3i). Initially, the uncertainty (shown by coloured ellipses: yellow for the robot uncertainty, purple for the landmark uncertainty) of the robot position and the landmark positions are small. As the robot progresses through the environment, these uncertainties grow (Figure 2.3ii–iii). In Figure 2.3iv, the first landmark is detected again in the environment and the uncertainties of the robot position and landmarks decrease. This update in uncertainty is then propagated to all landmarks in the environment.

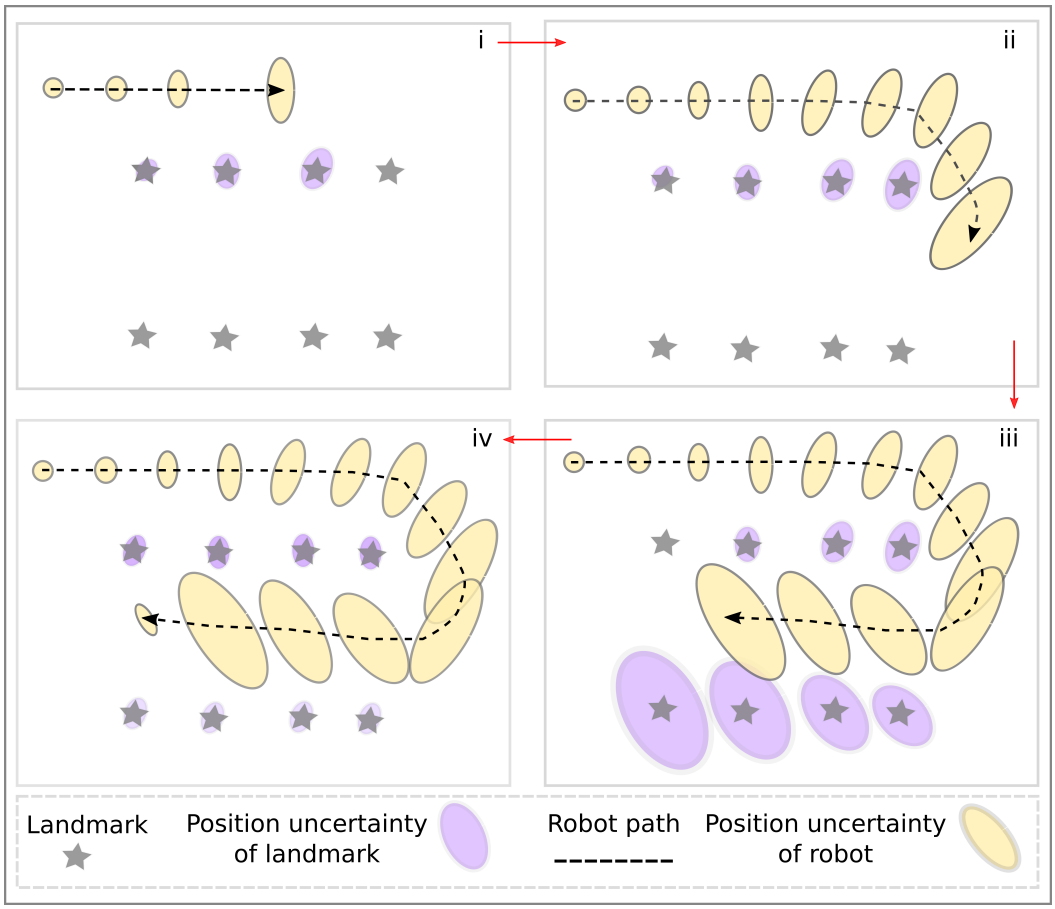


Figure 2.3: EKF applied to an online SLAM problem. A robot moves through the environment from Figure i to iv sequentially, where the path of the robot is shown by a dotted line and the uncertainties of observed landmark and sequential robot positions are shown by ellipses. Initially, the position uncertainties are small in i, but grow from ii–iii until the first landmark is sensed again iv. At this point, the uncertainty in all landmark positions decreases, as does the current robot pose.

Since landmarks in the environment are assumed to be stationary, false associations from sensor observations can lead to the divergence of the solution (Fuentes-Pacheco et al., 2012). Similarly, data association, especially loop closure, is difficult (Durrant-Whyte and Bailey, 2006) and can also be due to accumulation of errors in position estimates (Hidalgo and Braunl, 2015). Computational efficiency can also be an issue for EKF SLAM because all landmarks and the relationship between landmarks are updated for each new observation; hence the computation cost grows quadratically with the number of landmarks (Fuentes-Pacheco et al., 2012).

2.2.2 Graph-based SLAM

An alternative way of representing the SLAM problem is through a graphical representation and is shown in Figure 2.4. Graph-based SLAM represents the SLAM problem as a sparse graph, where the nodes of the graph correspond to robot pose and landmark positions, and the graph edges represent spatial constraints obtained from measurements from on-board sensors (Grisetti et al., 2010), with a probability distribution defined by the sensor measurement model. In graph-based SLAM, a graph is built and an optimisation approach is used to minimise the error introduced by the spatial constraints.

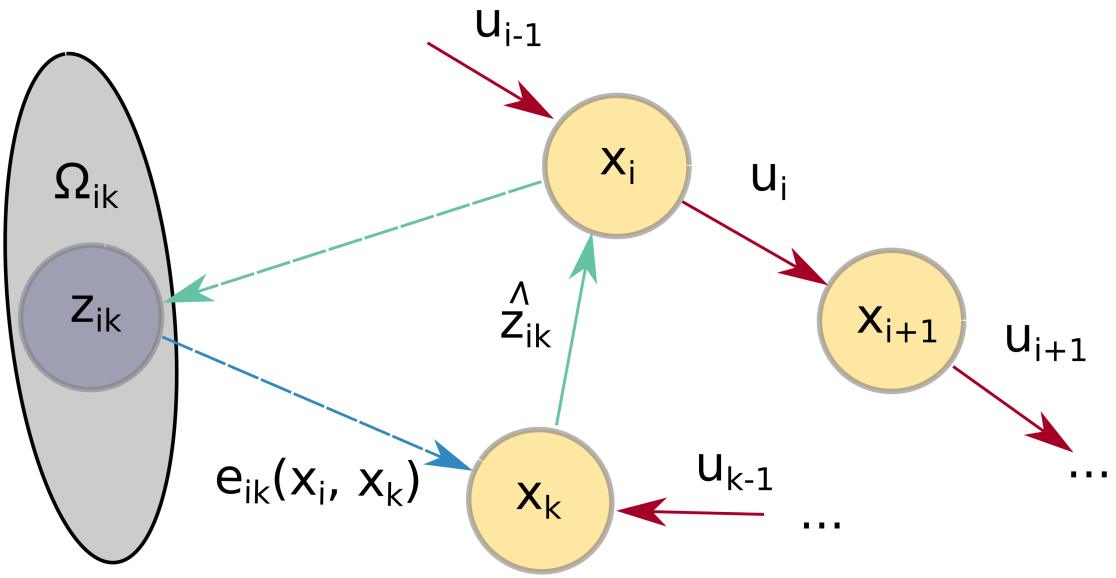


Figure 2.4: A schematic showing the Graph SLAM algorithm. The nodes, x_i, and x_k represent two (non consecutive) robot poses. Edges between the nodes marked u represent odometry between two robot poses. The node z_{ik} represents a landmark with position calculated from sensor measurements taken from the robot pose at x_i, and x_k. $\hat{z}_{ik}$, is a virtual measurement which represents the estimated position of x_k as seen from x_i. e_ij is the error between the expected measurement and real measurement of the environment between nodes x_i and x_j. Ω_{ij} is the information matrix of the measurement z_{ij} which accounts for the uncertainty in the sensor and odometry measurements.

In Figure 2.4, a robot is moved from position x_i to x_{i+1}. Each of these states is represented by a node in the graph and the edge between these nodes represents odometry information from sensors. At some other robot pose x_j, the same part of the environment is observed that was also observed at the robot pose x_i. Another edge called a 'virtual measurement' ($\hat{z}_{ik}$ in Figure 2.4) can be made between nodes x_i and x_k that represents the position of x_k as seen from x_i. An error function, e_{ik}, is defined that describes the difference between the expected and real measurement of the landmark z_{ik}, where the expected measurement is determined from the transformation between the two nodes x_i and x_k. This expected measurement has a Gaussian distribution (Grisetti et al., 2010). A cost function can then be defined that is dependent on the error function and the uncertainty of the virtual measurement (Grisetti et al., 2010). The edge constraints and node locations in the graph are considered as a spring-mass model (Siegwart and Nourbaksh, 2011; Siciliano and Khatib, 2016), where finding the solution to the SLAM posterior is equivalent to computing the minimal energy of the spring-mass model, i.e.,:

$$x^* = \arg\min_x \sum_{ik} e_{ik}^T \Omega_{ij} e_{ik} \tag{2.7}$$

where x^* is the set of poses that minimises the log-likelihood of the observations ($\sum_{ik} e_{ik}^T \Omega_{ij} e_{ik}$). e_{ik} is the error between the expected displacement and real displacement and Ω_{ik} is the information matrix that accounts for the uncertainties in the position estimates.

Optimisation techniques (such as Levenberg–Marquardt, Least Squares or Gauss-Newton (Grisetti et al., 2010)) can then be used to minimise the error function by adjusting the overall position of the nodes in the graph to find which arrangement gives the best representation of robot position.

2.2.3 Particle filters for SLAM

Particle filters, also known as recursive Monte Carlo sampling (Durrant-Whyte and Bailey, 2006), have also been used to solve the SLAM problem, particularly through the application of Rao-Blackwellised particles (K. Murphy, 2000). The individual map errors are conditionally independent for each of the particles, hence the posterior can be factorised as follows (Siciliano and Khatib, 2016) :

$$p(x_{0:t}, M|z_{1:t}, u_{0:t}) = p(x_{0:t}|z_{1:t}, u_{1:t}) \prod_{n=1}^{M} p(m_n|x_{0:t}, z_{1:t}) \tag{2.8}$$

The first part of the factorisation in 2.8 ($p(x_{0:t}|z_{1:t}, u_{1:t})$) allows the SLAM problem to be split into a localisation problem, which assumes a known map and can be solved using a method such as Monte-Carlo Localisation. Using the pose estimates from the localisation calculation, the mapping step of SLAM is then performed with respect to each pose. This factorisation also allows each of the landmarks in the environment to be represented by separate, low-dimensional, EKFs rather than a single Gaussian that represents all of the features jointly, as for EKF SLAM, and allows efficient computation by particle-based methods Siciliano and Khatib (2016). These landmarks can then be represented and summed using a set of M particles (i.e., for $\prod_{n=1}^{M}$).

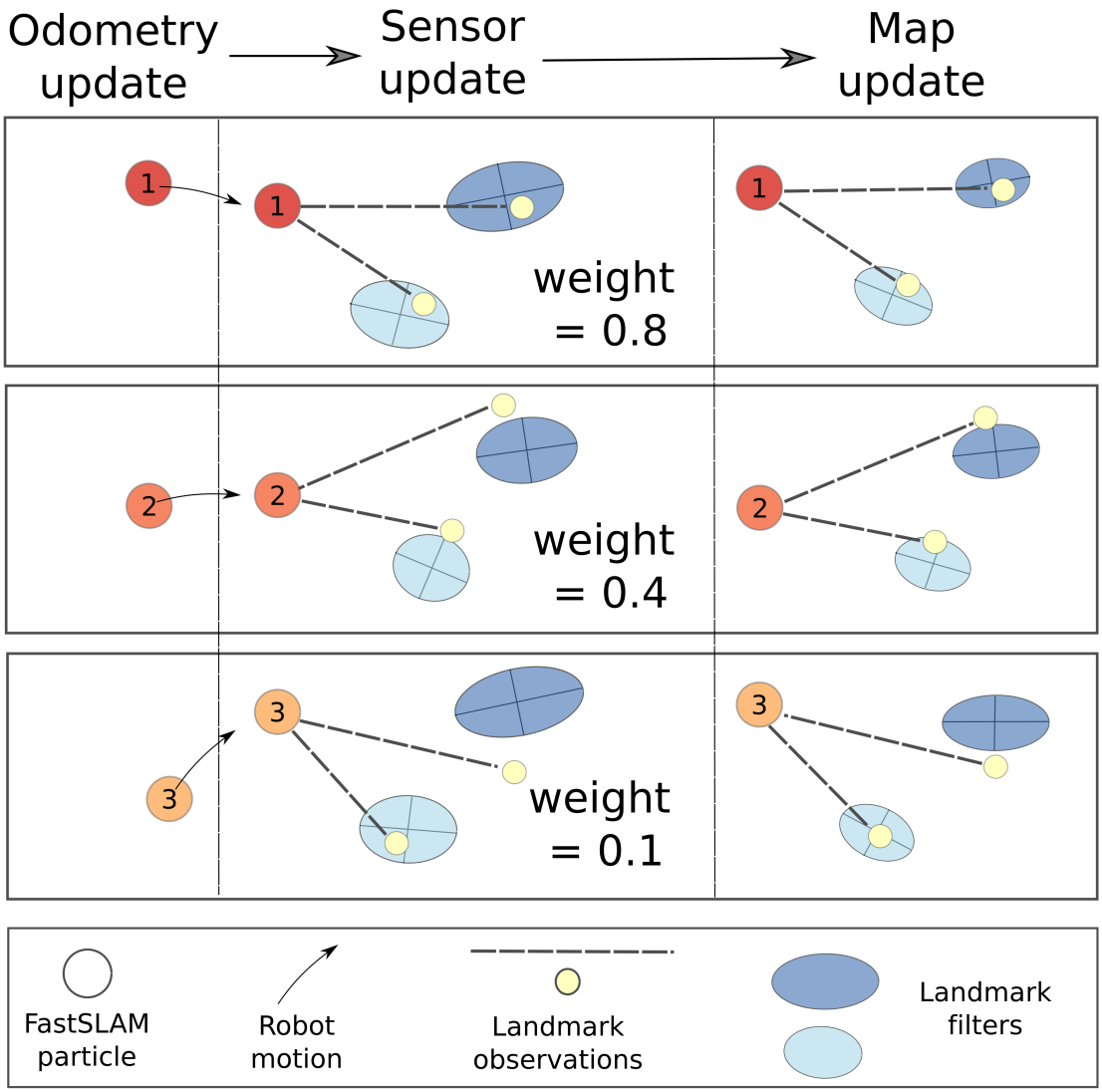


Figure 2.5: A schematic showing the FastSLAM particle-based SLAM algorithm for three exemplar particles. Initially (left), the position of the particles is updated using odometry information. Next, information of the location of landmarks from previous steps is used to estimate the position of landmarks in the environment at the current step. Using current sensor information, weights are calculated based on the probability that the particle is at the true robot location and the mean and covariance of observed landmarks is updated. Figure adapted from Figures in Siciliano and Khatib (2016).

A specific application of Rao-Blackwellised particle SLAM, known as FastSLAM (Montemerlo et al., 2002), represents the belief of the robot by randomly sampling the probability distribution over the full path of the robot. Each of the particles stores an estimate for the pose of the robot, a known map with respect to the particle, a set of Kalman filters with mean and covariance for each of the features in the map. A schematic of the FastSLAM algorithm is depicted in Figure 2.5. The first step in the FastSLAM algorithm is retrieval, where the pose of the particle at the previous time step is obtained. Next a prediction step occurs, based on the current odometry information, to obtain a new pose for each of the particles. Then, a measurement update is performed, where the correspondence of the current sensor information for each of the observed features is identified and incorporated into the EKFs of the landmarks by updating the mean and covariance of the position uncertainty. The importance weight is then calculated for each particle, which is based on the probability that the particle position is the true robot position. Finally, the re-sampling stage occurs, where the number of particles is maintained by replacing, removing and updating particles according to their value of importance factor (Siegwart and Nourbaksh, 2011), normalising the particle weighting to add up to one (Sim et al., 2005).

A set number of particles are maintained at each time step (Siegwart and Nourbaksh, 2011). Each particle contains an estimate of the robot path (Siegwart and Nourbaksh, 2011) with estimates for the landmarks represented by a probability distribution function. Due to the sampling process, non-linear models can be used. The particles will populate areas of the system space where the real state is more likely (Dueñas, 2015). The use of random sampling means that no linearisation of the motion model is required and so non-Gaussian distributions can be used (Siegwart and Nourbaksh, 2011).

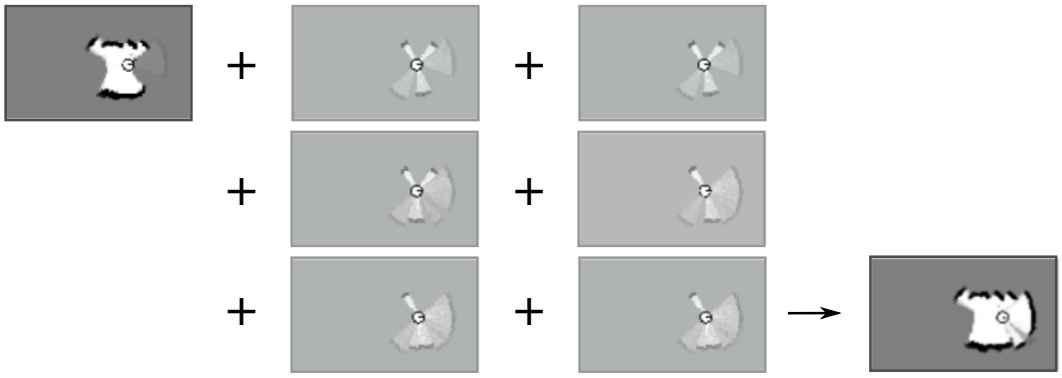
Complexity grows logarithmically, rather than quadratically as for EKF (Siegwart and Nourbaksh, 2011). The key issue is that it is not possible to determine how many particles are required to accurately estimate the robot position, as a result more particles than required may be used which can impact performance due to memory requirements, but using too few particles also impacts accuracy (Fuentes-Pacheco et al., 2012) A key property

of FastSLAM is that the observed landmarks are independent because separate EKFs are used for each landmark (Fuentes-Pacheco et al., 2012).

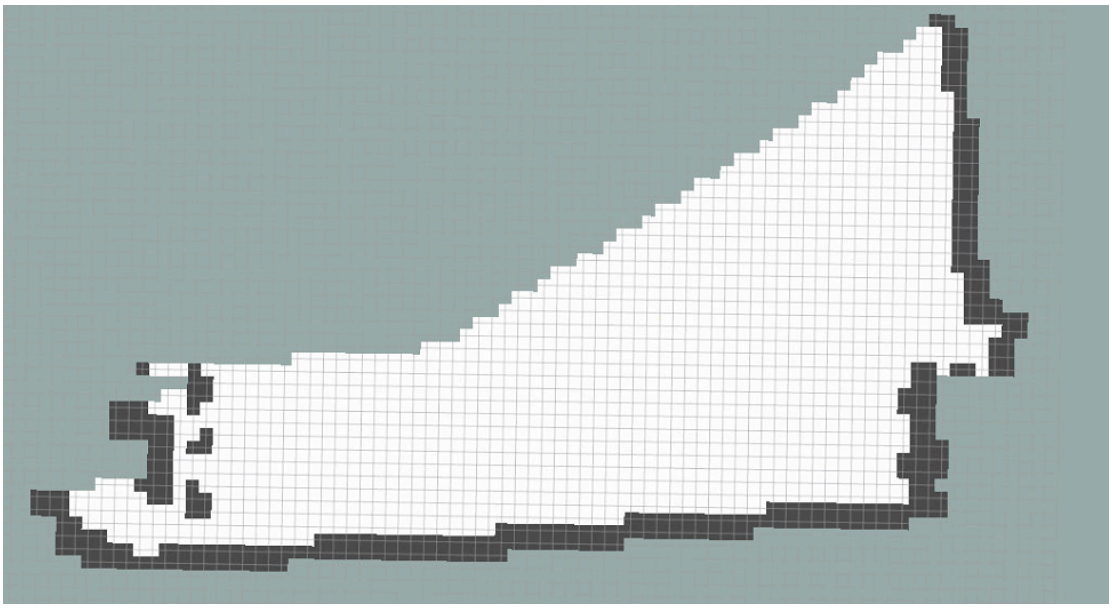
2.3 Map representation

As described above, a representation of the landmarks in the environment for SLAM to inform further tasks, such as path-planning is needed, and this representation is typically described as a map. Two common methods of map representations are grid-based and feature-based. Feature-based methods were discussed previously, where a landmark is represented as a point in the parameter space.

In grid-based mapping methods, the world is split into regular, rigid cells and each cell is assumed to either represent occupied or unoccupied space. One such grid-based representation is an occupancy mapping; an example is shown in Figure 2.6. Each cell in the occupancy map is updated by a binary Bayes filter, using a sensor model to update the probability that the cell is either occupied or unoccupied. It is assumed that the occupancy of each cell in the map is independent of the other map cells, that the robot poses are known and that the map is static. Occupancy maps are useful in representing environments such as floor plans as they describe a 2D slice of a 3D world. If the probability that a cell is occupied by an object is high, the cell is filled in black, whilst white shows free space and grey is unknown as it is outside of the mapped area, i.e., areas out of the sensor range. This approach reduces the effect of transient objects, such as people, in the map. The uses of occupancy maps will be considered further in Chapter 4.



(i): Incremental update of an occupancy map.



(ii): An example occupancy map with grid cells.

Figure 2.6: This image is adapted from Siciliano and Khatib (2016) and shows an example of the incremental update of a 2D occupancy map. The upper left image shows the initial map and the lower right image shows the finished map. An example occupancy map with grid cells is shown in 2.6ii. The black map cells corresponds to occupied regions, the white regions show unoccupied space and the darker grey region shows regions outside the map.

2.4 SLAM implementations

The architecture of current SLAM systems is often composed of two main parts: the front-end and the back-end. The front-end is responsible for abstracting the sensor data into models that are suitable for state estimation and the back-end performs inference on the abstracted data from the front end (Cadena et al., 2016). For example, in a graph-based SLAM approach, range or visual data can be used to construct the graph of the environment in the front-end and optimisation of the graph to minimise the error in position estimate would occur in the back-end (see Figure 2.7).

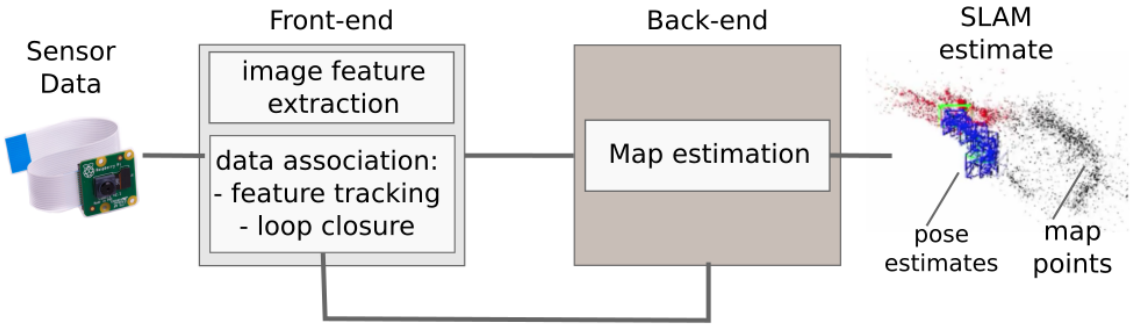
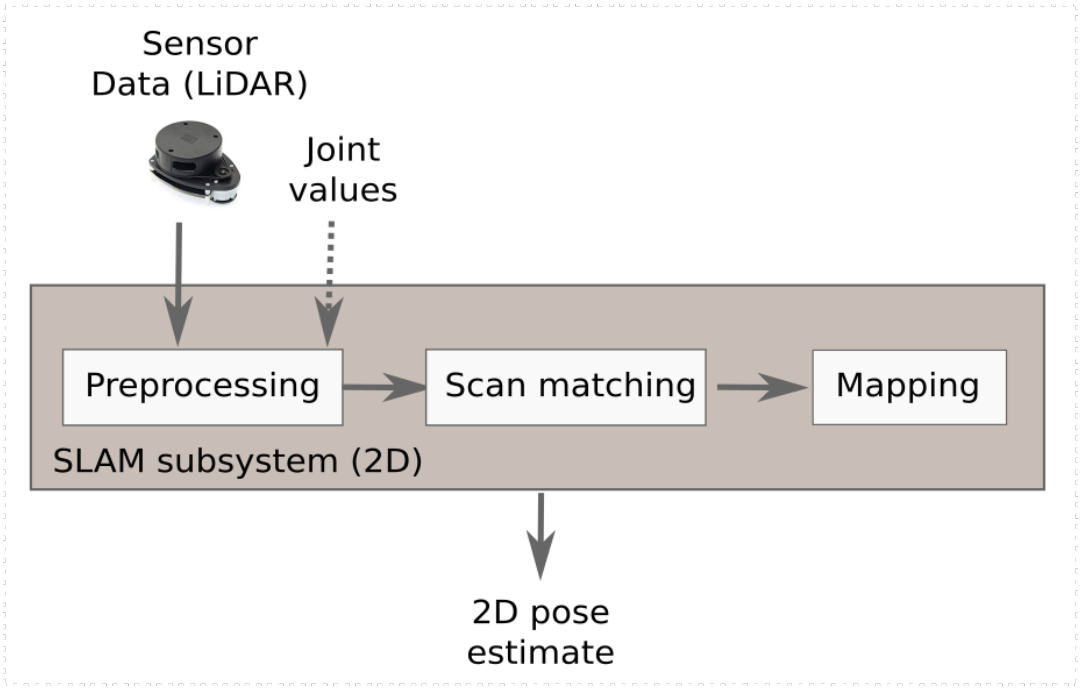


Figure 2.7: This image is adapted from Cadena et al. (2016) and shows the front-end and back-end in a typical SLAM system. The front end takes in and abstracts sensor data, the back end performs inference about the environment using the abstracted features and feedback is provided between the back and front ends for loop closure and verification.

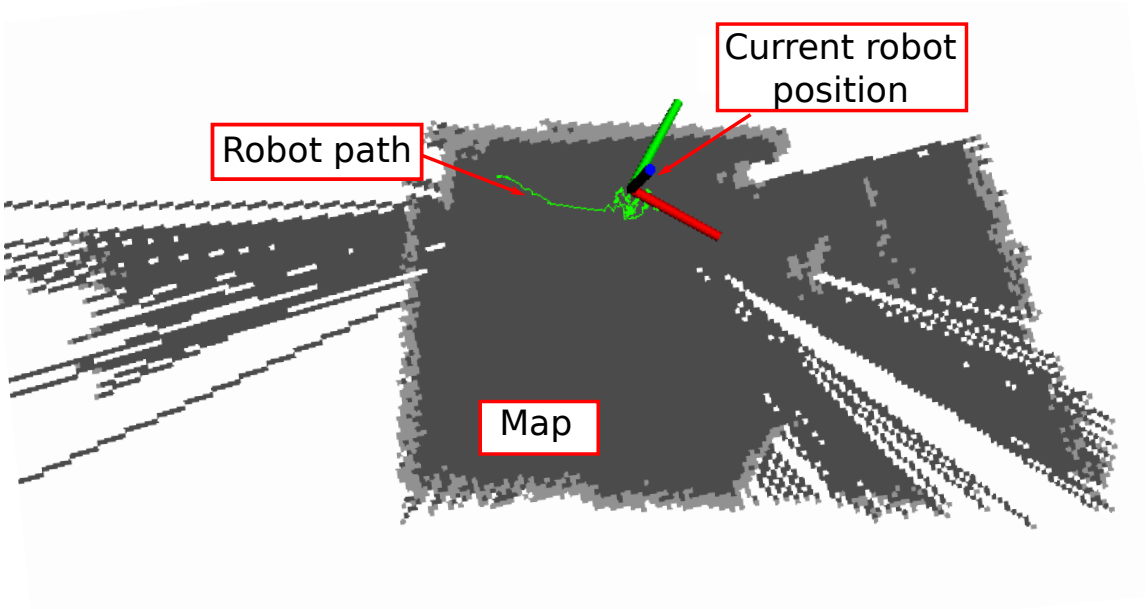
2.4.1 SLAM implementation using a 2D LiDAR

Hector SLAM

Hector SLAM (Kohlbrecher et al., 2011; Kohlbrecher et al., 2014) is primarily a 2D SLAM approach that incorporates 2D LiDAR scans into a planar occupancy map. The contribution of Hector SLAM is as a frontend-only SLAM, with no backend optimisation of the pose-graph being required (see Figure 2.8). Hector SLAM does not require any external method of odometry (e.g., wheel encoders), but uses fast scan matching approaches to provide this information. Traditionally, scan matching is done using Iterative Closest Point (ICP) which is computationally expensive. In Hector SLAM however, a fast scan-matching approach is used, which takes advantage of the low distance measurement noise and high scan rates of modern LiDAR (Kohlbrecher et al., 2011). The endpoints of the LiDAR beams are aligned with the occupancy map generated by Hector SLAM and a Gauss-Newton approach is used to find the transformation of the current scan that best minimises a cost function to find the correct alignment of the scan to the map. Using this approach, the data association problem is avoided and an exhaustive search of the pose of the robot is not required since matching takes into account the previous poses used to generate the current map. To allow pose estimation in six degrees of freedom, an EKF is used to combine velocities and position estimates from gyroscopes and accelerometers with the 2D LiDAR and range sensor to estimate height from the ground. Exemplar results generated using Hector SLAM are shown in Figure 2.8b.



(i): Hector SLAM method overview.



(ii): Hector SLAM output

Figure 2.8: An overview of the SLAM and navigation subsystems in the Hector SLAM package (2.8i), where the dashed lines show optional information, this figure is adapted from Kohlbrecher et al. (2011). And an example of the output from Hector SLAM (2.8ii). The robot's path over time, the robot position and the map generated using 2D LiDAR data are annotated.

2.4.2 SLAM implementations for camera data

ORB-SLAM

Orientated Rotated Brief SLAM (ORB-SLAM) was identified for use in this work because it has been implemented in real-time with the Raspberry Pi (Ponnu et al., 2016) and has both monocular and stereo implementations[1]. In addition, ORB-SLAM has a ROS package called ORB-SLAM2 that can be configured to run on many platforms and has also been tested in a wide range of environments including datasets from indoors, micro-UAVs and a car in an urban setting (Mur-Artal and Tardos, 2017). Both the stereo and monocular implementations of ORB-SLAM will be tested in this thesis, hence the methods for the monocular and stereo implementations are considered here.

ORB-SLAM is an open-source, feature-based technique that uses a monocular camera implementation (Mur-Artal et al., 2015) and also stereo and RGB-D camera implementations (Mur-Artal and Tardos, 2017). A brief overview of the method is given here, but full details of the implementation can be found in Mur-Artal et al. (2015) and Mur-Artal and Tardos (2017).

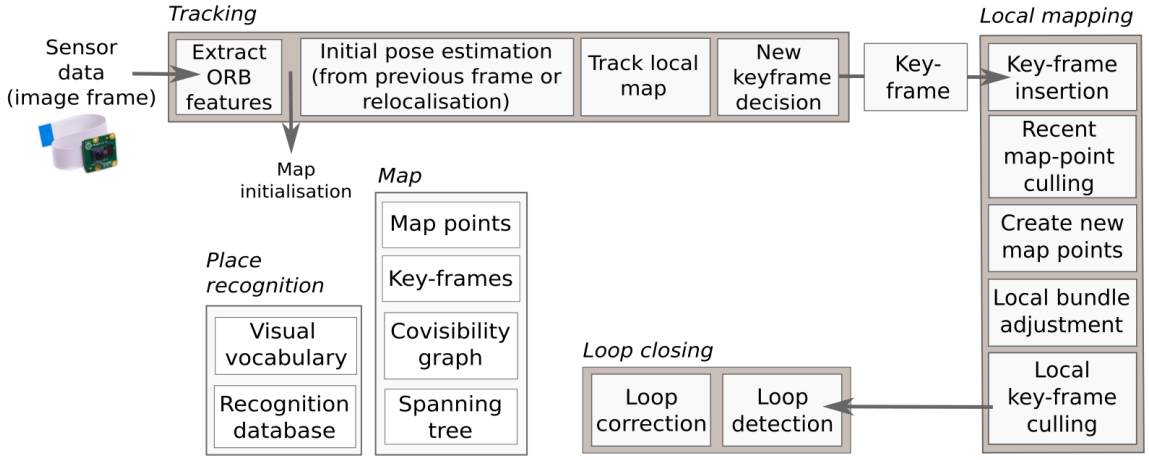
In the monocular implementation of ORB-SLAM, the process falls into three main processes: tracking, local mapping and loop closing. For an initial input image frame from a video feed, ORB feature detection is performed, with ORB features being chosen for their invariance in rotation and scale, which allows them to be recognised both quickly and with invariance to view-point (Mur-Artal et al., 2015).

The tracking step allows localisation of each new frame by matching features to a local map. Certain input image frames from a video feed are selected as keyframes (which store the camera pose camera parameters and ORB features) and bundle adjustment is performed on these key-frames rather than all the dataset, which allows large environments to be mapped (Mur-Artal and Tardos, 2017). A process for culling redundant keyframes and map points can also be implemented to retain only high-quality keyframes and points. Next, loop closing is performed to reduce any accumulated drift errors in the map. A fourth

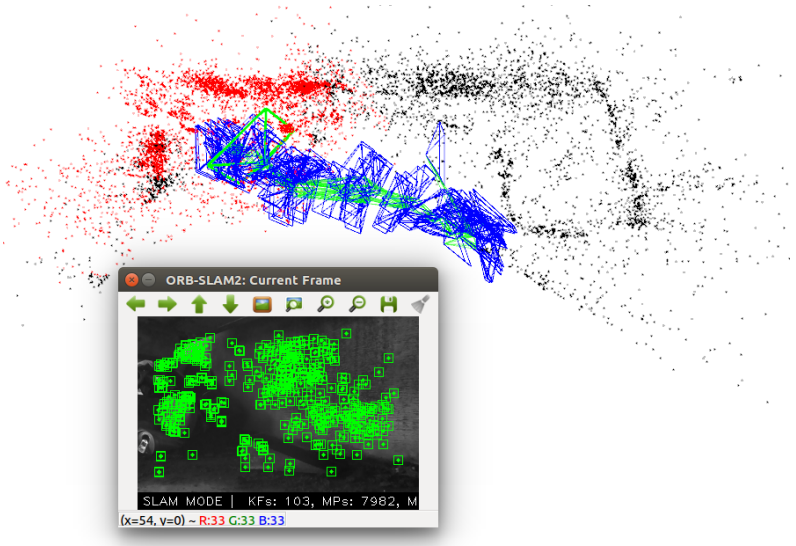
[1] https://github.com/raulmur/ORB_SLAM2 (Date last accessed:01/02/19)

process can be launched after loop-closing to perform a bundle adjustment on the full map in parallel to the standard mapping and localisation process before the resulting updates are merged. A map is created of 3D world coordinates and a place recognition module named Bag of Words (DBoW2) (Galvez-López and Tardos, 2012) is implemented to allow re-localisation in previously mapped scenes. For re-localisation, a visual vocabulary is initially created offline by extracting ORB-descriptors from a large set of images to create a database of general scenes that can be used for multiple environments (Mur-Artal et al., 2015). In addition, keyframes are added to the database with the most recent frames are stored most recently and in the event of loss of position, the current frame is turned into a bag of words representation and the Bag of Words database is checked against to find the last known keyframe.

In the stereo implementation, the ORB features are extracted in both images and points from the left image are matched with the horizontal coordinate of points in the right image. The features that are extracted from the images are labelled as far keypoints (greater than 40 times the stereo baseline) and near keypoints. Close keypoints can be triangulated from one frame and far keypoints are triangulated if they appear in many views. Using the far keypoints gives a good estimation of rotation. One important difference between the stereo and monocular implementations is that the real scale of a scene can be determined from triangulation of the stereo points; as a result, there is reduced drift in scale in the stereo implementation (Mur-Artal and Tardos, 2017).



(i): ORB SLAM method overview.



(ii): Monocular ORB SLAM implementation

Figure 2.9: An overview of the ORB SLAM algorithm (2.9i) adapted from Mur-Artal et al. (2015) and an example implementation of monocular ORB SLAM (2.9ii). Figure 2.9ii shows the detection of the ORB features (green squares) that are tracked in the current frame. These features are mapped into 3D space as cartesian points, where the black points in Figure 2.9ii were detected in previous frames and the red points are visible in the current image frame. Blue rectangles in Figure 2.9ii shows the position of the key frames.

2.5 Localisation-only approaches to navigation

The key difference to the SLAM methods described in Section 2.2 is that the map of the environment already exists, which reduces the complexity of the SLAM problem. In the simplest case, localisation can be performed using only odometry calculations, with the current position of the robot being determined with respect to an initial position. However, over long periods of time, the position estimate can drift. Data from sensors (such as LiDAR, ultrasound, camera, etc.) can be used to recognise landmarks (such as walls, corners, etc.) to reduce the drift of the robot's position within the map. The position estimate can be improved using filtering techniques such as Kalman Filters. For example, Moore and Stouch (2016) implement an EKF that combines sensor information from multiple GPS and IMU devices to vastly improve on the dead-reckoning position estimation of a robot. Without GPS, EKF localisation allows the combination of sensors to improve position tracking, which is important for accurate navigation, but they do not necessarily allow re-localisation if the position of the robot is lost. Global localisation (when the robot is placed in the map with no guess for the current location) is possible if GPS is available or when using grid-based methods, such as Markov Localisation (Burgard et al., 1998) and Monte-Carlo localisation (Fox et al., 1999).

Adaptive Monte-Carlo localisation

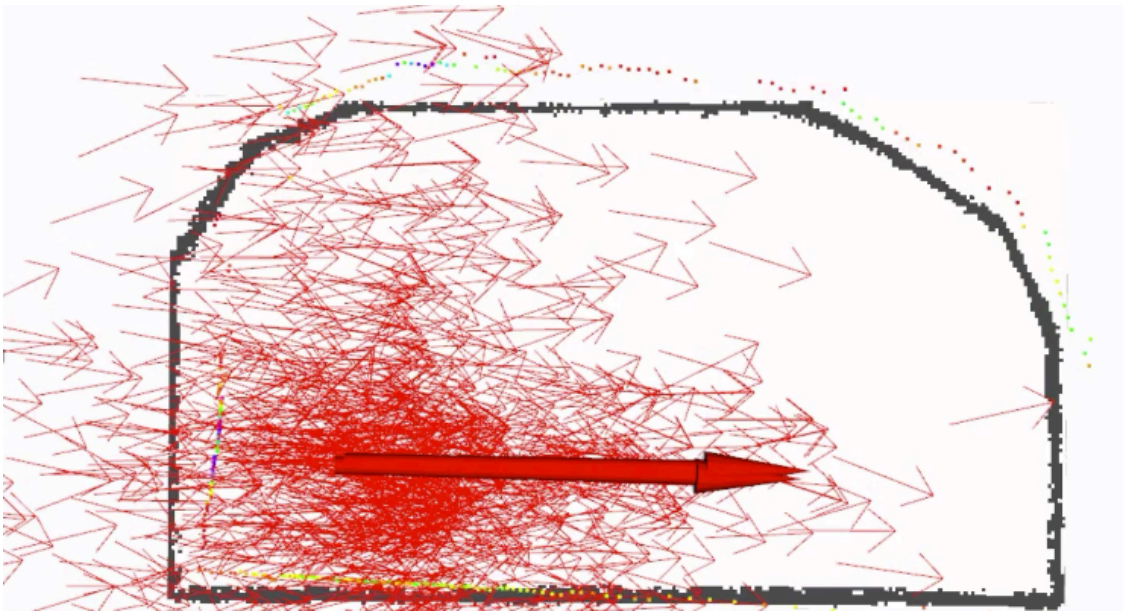
Monte-Carlo localisation (Fox et al., 1999; Thrun et al., 2005) is similar to the particle filter methods for SLAM seen in Section 2.2.3, but a map of the environment already exists. The knowledge that the robot holds about its environment is represented by a set of particles, where a particle represents a 'guess' for the pose of the robot. In the prediction step, the motion commands given to the robot are applied to the particles using some form of odometry, updating their position in the map. At the new particle locations, measurements to map landmarks are obtained and are compared to the expected values from the current particle positions. The set of particles begin with a uniform belief of the true location, but are then re-weighted depending on the probability that a given particle could obtain the

current sensor readings in its current location. The particles are then re-distributed for the next time-step based on this probability. The estimated position of the robot at the current time step is centred on the accumulative probability mass of the particles (Thrun et al., 2005).

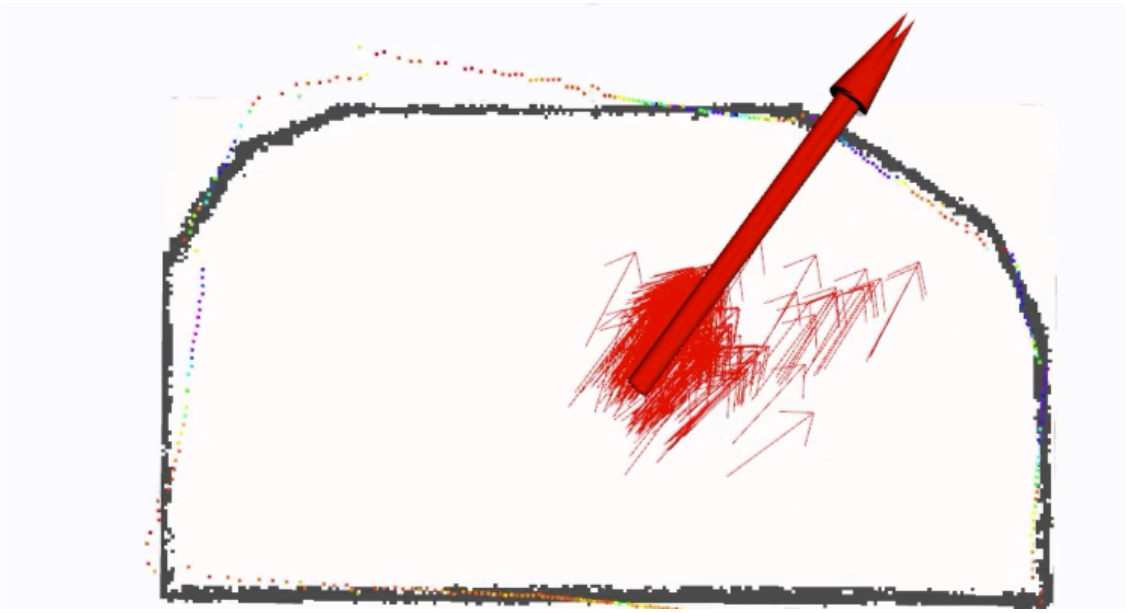
If the starting position of the robot in the map is unknown, the particles are spread across the map used for localisation, with each particle representing a potential position and orientation for the robot. Alternatively, a known starting location can be given as an input, around which the particles spread. The first of these cases is referred to as global localisation and the second as local localisation.

Adaptive Monte-Carlo Localisation (AMCL) is a version of Monte-Carlo Localisation where the number of particles are adapted over time using Kullback-Leibler Divergence Sampling to determine the number of particles that are required at each time step (a detailed description of the approach is given in Thrun et al. (2005)). Particles are added to the map until a statistical bound is satisfied, where the error between the true pose and the sample-based approximation is less than a fixed value. Changing the number of particles over time allows better computational efficiency, since fewer particles are required if many particles have a similar belief about the robot's position. The number of particles required tends to decrease over time as the robot moves around the map and the belief regarding the current position improves Thrun et al. (2005). An example of the initial and final steps of AMCL, after the number of particles has reduced due to increased certainty, is shown in Figure 2.10. The implementation of AMCL used in this thesis is an existing ROS package[2] based on the work in Thrun et al. (2005).

[2]http://wiki.ros.org/amcl (Date last accessed:01/02/19)



(i): AMCL at initial time step



(ii): AMCL after several time steps

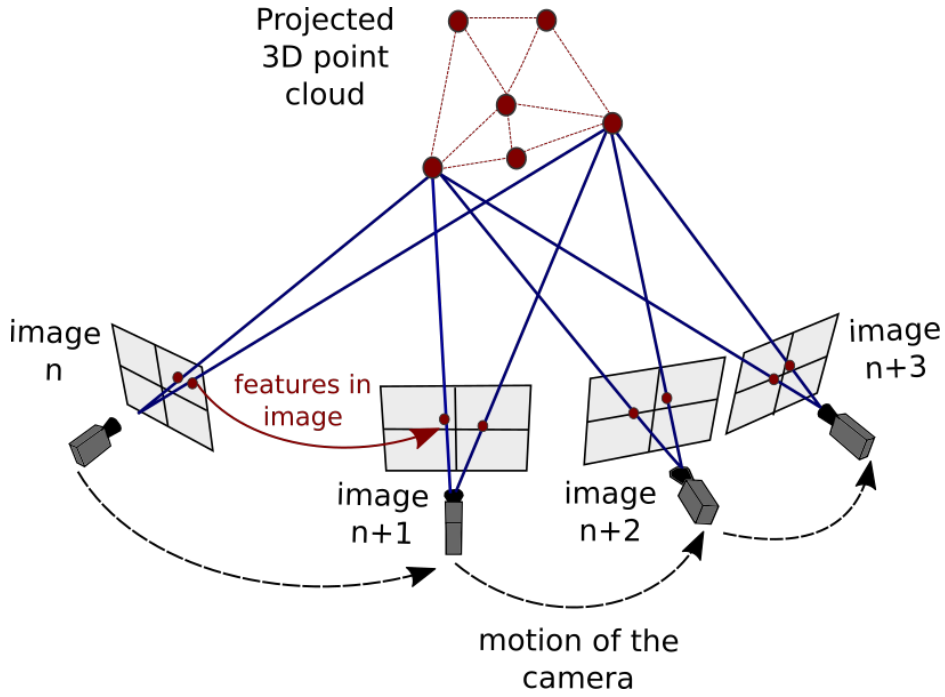
Figure 2.10: Example of the AMCL algorithm at an initial time step (2.10i), where there the uncertainty in the robot's current state is high, and at a later time step (2.10ii) where the uncertainty has decreased along with the number of particles.

2.6 Structure-from-Motion

Structure-from-Motion (SfM) uses two-dimensional image data from photographs, or video frames, to find the three-dimensional geometry (the structure) of a scene, or an object, by taking the photographs from several different physical locations (the camera has motion), see Figure 2.11i. The aim of SfM is to find the relative positions of the cameras and features seen in the camera views. In Chapter 1, SfM was a common approach used in inspection of infrastructure using RGB camera information. In this thesis, SfM will also be used as part of robotic data-collection and the point clouds generated using SfM and Multi-View Stereo (MVS) will be used for map creation, inspection applications and improving robotic simulation environments.

Before the SfM calculations can begin, suitable data capture is required. The data should be in the form of many overlapping images or video frames taken from several locations, with suggestions for number of photos in the order of tens to hundreds for higher accuracy (Micheletti et al., 2015). The data collection requires some careful consideration since it is recommended by several sources (e.g., Micheletti et al. (2015) and Leberl et al. (2010)) that at least $60\,\%$ overlap is required between images.

The SfM pipeline consists of the following steps (Toldo et al., 2015): data capture, keypoint detection, keypoint matching, computing 3D points from corresponding points in multiple images (also known as intersection), calculation of the camera matrices from known 2D-3D correspondences (also known as resection), relative orientation (finding the relative camera poses and orientations), combining these triangulated and resected views and then optimising the results using bundle adjustment. The SfM process is often followed by MVS, in which the sparse information from SfM is embellished by extracting extra image features to create a dense point cloud.



(i): The reconstruction of an SfM point cloud from the image features

(ii): An example point cloud data generated using SfM

Figure 2.11: Figure to show the basic principal of Structure-from-Motion (2.11i, adapted from http://theia-sfm.org/sfm.html, date accessed: 01/02/19), where motion of a camera taking 2D images can be used to reconstruct the 3D environment. An exemplar 3D Cartesian point cloud data generated using SfM is also shown (2.11ii).

The point clouds produced using SfM and MVS are initially unscaled. For applications such as construction and inspection, some form of scaling that is representative and accurate of the object or area is required. There are several ways to obtain these coordinates depending on the level of accuracy required for the application.

The methods discussed in this chapter will be implemented on a robotic platform and evaluated in a laboratory environment, a real bridge environment and a simulated environment in Chapter 4. Chapter 3 will describe the development of the robotic platform.

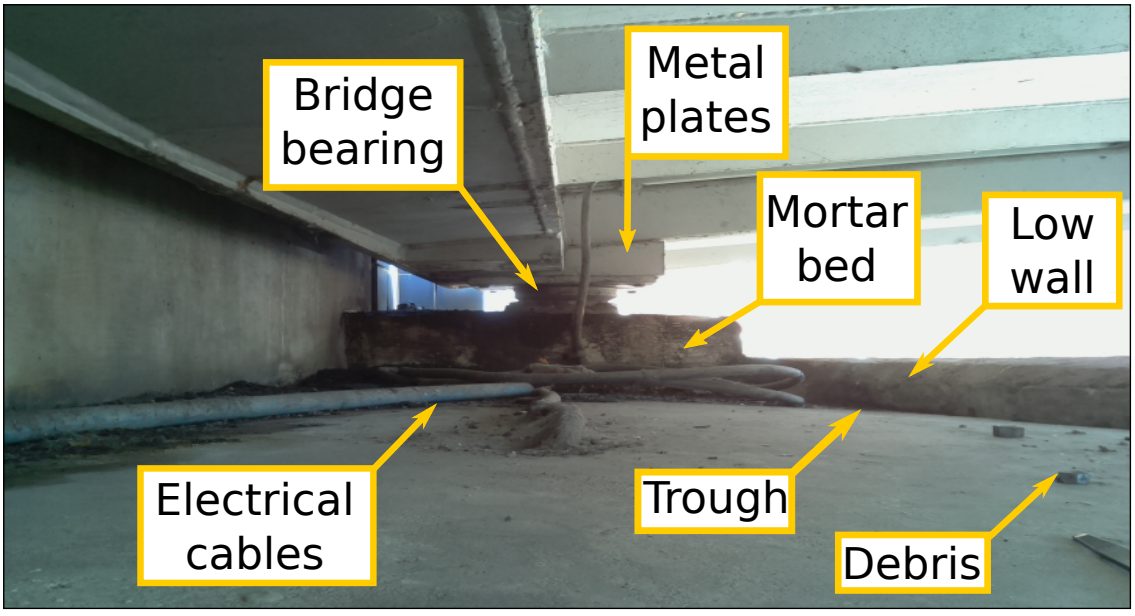
Chapter 3

Robotic platform and system development

3.1 Description of the inspection environment

The bridge and associated bearings that form the experimental environment in this thesis are located at The Millennium Bridge Leeds, UK. Millennium Bridge (Figure 3.1i) is a cable-stayed footbridge crossing the River Aire, spanning approximately $57\,\mathrm{m}$. Only the bearings on the north side of the river are studied in this work; these bearings are located in a space approximately $2.8\,\mathrm{m}$ by $1.2\,\mathrm{m}$, which has a total height of $0.5\,\mathrm{m}$. The layout of this site gives an enclosed region to be referred to as bearing enclosure, see Figure 3.1ii. Bridge bearings tend to be located on the underside of a bridge, where the weather conditions and lighting conditions may be variable or poor. The bearing enclosure used in this thesis is compact and narrow, there are also steep drops and uneven terrain and obstacles to overcome. A representative example of a bridge bearing enclosure is shown in Figure 3.1ii and Figure 3.2.

(i): A photograph of Millennium Bridge



(ii): Exemplar bridge bearing enclosure

Figure 3.1: Photographs of the bridge (Figure 3.1i) and bridge bearing enclosure (Figure 3.1ii) used for testing purposes in this thesis. The bridge bearing, surrounding structure and potential obstacles in the environment are annotated. More photographs of the bearing enclosure are shown in Figure 3.2.

Figure 3.2: Photographs of the bridge bearing enclosure taken from on-board the robotic platform described in this Chapter. The arrows in the image show the sequence the photographs were taken in.

3.2 Qualitative review of sensors and SLAM

As described Chapter 2, on-board sensors are required to measure the control input to the robot and the landmark information in the surrounding environment. The qualitative advantages and disadvantages of the sensors described in Chapter 2 are summarised in Table 3.1.

Odometry information is required for SLAM algorithms to provide updates for the motion model. Odometry can be provided by wheel encoders, GPS, monocular camera, stereo camera and LiDAR. Alternatively, velocity information can be provided to the motion model from the inertial measurement unit or from dead reckoning of known velocity commands. The odometry information available from LiDAR sensors tends to have higher accuracy and precision than for wheel encoders (Kohlbrecher et al., 2011). GPS was used in many of the literature examples reviewed in Chapter 1, but is not sufficient for navigation in the bridge bearing enclosure, mainly due to the obstruction from the bridge structure to the GPS signal. Of the reviewed sensors, 3D LiDAR is the most robust for navigating in variable conditions. However, the bearing inspection area is compact, which limits the size of the robotic platform that can be used. 2D LiDAR is more compact and has lower power requirements than 3D LiDAR and can be used for both mapping and odometry purposes.

Since the primary purpose for developing a robotic platform is to perform inspection of the bridge bearings, some sensors for inspection are required. The literature review in Chapter 1 highlighted that a camera is the most popular sensor for bridge inspection since it provides intuitive information that can be reviewed by human inspectors and can aid the requirements for visual inspection. Since a camera is required for inspection, there may be utility in using the camera information to aid localisation and mapping of the bridge bearing enclosure. It is possible to calculate odometry information using a stereo camera, although it requires more computation than 2D LiDAR (Kohlbrecher et al., 2011). However, 3D data can also be obtained from the camera sensors, which may be beneficial for object and debris detection in the inspection environment. Hence, a 2D LiDAR, stereo camera and monocular camera are considered for navigation purposes in this thesis, with the camera sensors serving a dual purpose by providing inspection information.

Table 3.1: A summary of the sensor attributes for common sensors.

Summary of sensors		
Sensor	**Advantages**	**Disadvantages**
Ultrasonic Sensor	Low cost, lightweight, compact, low power consumption.	Data is affected by cone angle of the sensors and specular reflection
2D LiDAR	Higher accuracy and range than ultrasonic sensors, with up to $360°$. More compact than 3D LiDAR. Typically used for occupancy grid mapping, with outdoor SLAM implementations.	Data is affected by reflective and transparent surfaces.
3D LiDAR	Higher accuracy and range than ultrasonic sensors. Greatest depth range of SLAM sensors. Sensor data can be used for inspection and navigation purposes.	High power consumption and bulky. Data is affected by reflective and transparent surfaces.
Monocular camera	Low cost, lightweight, compact. Existing odometry and SLAM implementations in outdoor environments. Data is also useful for inspection.	Data is affected by vibrations and changes in lighting conditions. Some method of scaling is required to provide 3D data.
RGB-D camera	Many SLAM implementations for indoor applications, with occupancy mapping output. Gives 2D and 3D data.	Not reliable in outdoor conditions.
Stereo camera	Sensor data can be used for inspection and navigation purposes. Gives 2D and 3D data. Data can be used to provide odometry.	Required image processing has higher computational cost than other sensors.
GPS	Has been used in outdoor robotic research for localisation and SLAM.	Doesn't work in GPS-denied environments

Continuation of Table 3.1

Sensor	Advantages	Disadvantages
Wheel encoders	Low cost, lightweight, compact, low power consumption.	Data is affected by wheel slip and rough terrain. Cannot be used for SLAM without other sensors.
IMU	Low cost, lightweight, compact, low power consumption	Drifts over time. IMU with magnetometer affected by ferromagnetic material. Cannot be used for SLAM without other sensors.

3.3 Summary of SLAM methods

Cadena et al. (2016) states that for an indoor environment, the SLAM problem is considered largely solved for a robot with wheel encoders and laser moving in a 2D map, with examples now appearing in industrial settings. Both Fabresse (2018) and Filatov et al. (2018) compare 2D LiDAR packages in ROS for unknown indoor environments for metrics such as: map quality (structural similarity and nearest neighbour measure to ground truth for Fabresse (2018) and corner count, proportion of occupied cells compared to ground truth and number of enclosed map regions for Filatov et al. (2018)), CPU load and memory load (Fabresse, 2018). In Filatov et al. (2018), Cartographer, Gmapping, Hector SLAM and tinySLAM were reviewed, and Fabresse (2018) reviewed Cartographer, Gmapping and Karto SLAM were reviewed.

Fabresse (2018) found that Cartographer, Gmapping and Karto SLAM provided consistent maps, but that Cartographer needed tuning to specific parameters for different environments. With tuning, Cartographer provided the best quality maps, but without tuning, Cartographer failed in all test scenarios, whereas Gmapping and Karto SLAM succeeded. Filatov et al. (2018) found that Gmapping gave the best map quality for indoor environ-

ments. However, Kohlbrecher et al. (2011) reported problems for Gmapping for when roll and pitch motions were required. Hector SLAM has been used for outdoor applications(Kohlbrecher et al. (2011) and Droeschel et al. (2017)), and was originally designed to compete in the RoboCup competitions for robotic post-disaster search and rescue applications in challenging urban environments (Kohlbrecher et al., 2014). The Hector SLAM ROS implementation[1] also has support for full six degrees of freedom (DOF) with the addition of an IMU.

Visual SLAM uses a camera (e.g., monocular, stereo or RGB-D) as the primary sensor. Example implementations of visual SLAM include PTAM (Klein and Murray, 2007), RTAB-mapping (Labbe et al., 2014) and ORB-SLAM (Mur-Artal et al., 2015). Giubilato et al. (2018) compared common visual SLAM implementations and found that ORB SLAM was more robust than RTAB-mapping and PTAM and drifted less than LibViso2. ORB SLAM has also been implemented in outdoor environments (Mur-Artal et al., 2015) and in real-time on the Raspberry Pi (Ponnu et al., 2016) and NVidia TX1 (Giubilato et al., 2018).

For localisation in a known map the most common approaches use Monte-Carlo localisation. Adaptive Monte Carlo is more efficient than the original implementation due to the method for reducing the number of particles when not required (Thrun et al., 2005). AMCL has also been used for long term navigation of indoor environments (Nitsche et al., 2014) and in urban environments (Rohde et al., 2016).

[1] https://github.com/tu-darmstadt-ros-pkg/hector_slam

3.4 Choice of sensors for navigation and inspection

After reviewing the available sensors and SLAM methods, the following sensors were chosen for use with SLAM, localisation, map creation and inspection applications in this thesis:

- A 2D LiDAR, called RPLiDAR, of size of 7×10× $5\,\mathrm{cm}$ with a range of $6\,\mathrm{m}$. See Table 3.2 for sensor attributes. The LiDAR is used in this thesis to collect data for testing mapping algorithms and to provide odometry for the SLAM and localisation algorithms.
- Raspberry Pi Camera V2, an 8 MegaPixel RGB complementary metal-oxide-semiconductor (CMOS) camera. See Table 3.4 for sensor attributes. This camera is used in this work to collect data for testing mapping algorithms and for SfM for mapping and inspection purposes.
- The stereo camera, called ZED, which can used from $640\,\mathrm{pixel} \times 480\,\mathrm{pixel}$ at 100 frames-per-second (fps) to $1920\,\mathrm{pixel} \times 1080\,\mathrm{pixel}$ at $15\,\mathrm{fps}$. See Table 3.3 for sensor attributes. This camera is used to collect data for testing mapping algorithms in this thesis.

Table 3.2: A summary of the attributes for the RPLiDAR 2D LiDAR [2]

Sensor attribute	Value
Distance Range	0.15 m to 6 m
Angular Range	0° to 360°
Distance Resolution	$\leqslant$ 1 % of the distance
Angular Resolution	$\leqslant 1°$
Scan Rate	Min 1 Hz, Typical 5.5 Hz, Max. 10 Hz
Sample Frequency	$\geqslant$2000 Hz
Weight	190 g
Power	100 mA / 5 V
Dimensions	98.5 mm × 70 mm × 55.1 mm

Table 3.3: A summary of the sensor attributes of the ZED stereo camera [3].

Sensor attribute	Value
Video resolution (side by side)	2×1920 pixel × 1080 pixel, 30 fps 2×1280 pixel × 720 pixel, 60 fps 2×640 pixel × 480 pixel, 100 fps
Field of View	Max. 110°
Depth Range	1 m to 15 m
Baseline	120 mm
Focal Length	2.8 mm
Weight	159 g
Power	380 mA / 5 V
Dimensions	175 mm × 30 mm × 33 mm

Table 3.4: A summary of the sensor attributes for the Raspberry Pi monocular camera [4].

Sensor attribute	Value
Resolution	8 Mpixel
Still image resolution	3280 pixel × 2464 pixel
Video resolution	1920 pixel × 1080 pixel, 30 fps 1280 pixel × 720 pixel, 60 fps 640 pixel × 480 pixel, 90 fps
Focal Length	3.04 mm
Weight	159 g
Power	200 mA to 250 mA
Dimensions	23.86 mm × 25 mm × 9 mm

3.5 Camera calibration

For the visual SLAM methods tested in this thesis, camera calibration is required. The monocular and stereo cameras were calibrated using the chess-board method. The camera calibration script from ROS [5] was used to calibrate both the Raspberry Pi monocular camera and the ZED stereo camera. An 8x6 chessboard printed on A3 paper was placed on a planar surface and the camera was moved to obtain different viewpoints. The calibration software indicates when sufficient samples have been collected to perform a calibration. However, further samples were taken until the software indicated that the samples taken were above a 'good' threshold for translation in x, y and skew (indicated in a traffic light system from red to green). This number of samples translates to around 120 readings, which is comparable to the calibration samples collected for ORB SLAM in (Ponnu et al., 2016).

[5] https://github.com/ros-perception/image_pipeline/ (Date last accessed:01/02/19)

3.6 Robotic platform description

A robotic platform was developed using off-the-shelf technology and sensors. This platform is described in this chapter and are used in subsequent chapters for testing localisation and mapping algorithms in a real bridge environment. Of the platforms reviewed in Chapter 1, the most commonly used platform is the unmanned aerial vehicle (UAV). UAVs are commercially available platforms, with little hardware development required, the payload can also be changed depending on the inspection application. However, the aerodynamics underneath a bridge can lead to a tunnelling effect that can make the UAV difficult to fly and get close to the bridge bearings to perform an inspection. Many alternative approaches were summarised in Chapter 1, including magnetic wheeled platforms and platforms with suction, however more research is required before these are viable platforms for inspection. (Lattanzi and Miller, 2017) state that for highly unstructured or unstable environments, or in situations that only require planar motion, wheeled platforms may be a preferable option because of improved robot balance and stability. In addition, bearing enclosures typically allow planar motion (see Figure 3.2), and wheeled platforms are readily available and the on-board sensors for data collection can be adapted. Since the focus of this work is testing localisation and mapping for bridge inspection, a stable, wheeled ground platform was identified as a suitable option.

The robotic platform used in this work is a commercial product, the PiBorg DiddyBorg[6]. The platform is a six wheeled robot with a Perspex chassis, that is built around the Raspberry Pi single-board computer, which allows the adoption of tools such as the Robot Operating System (see Section 3.7). The DiddyBorg platform is compact, approximately $25 \times 18 \times 23\,\mathrm{cm}$ in size, see Figure 3.3.

[6]https://www.piborg.org/blog/diddyborg-overview (Date accessed: 01/02/19)

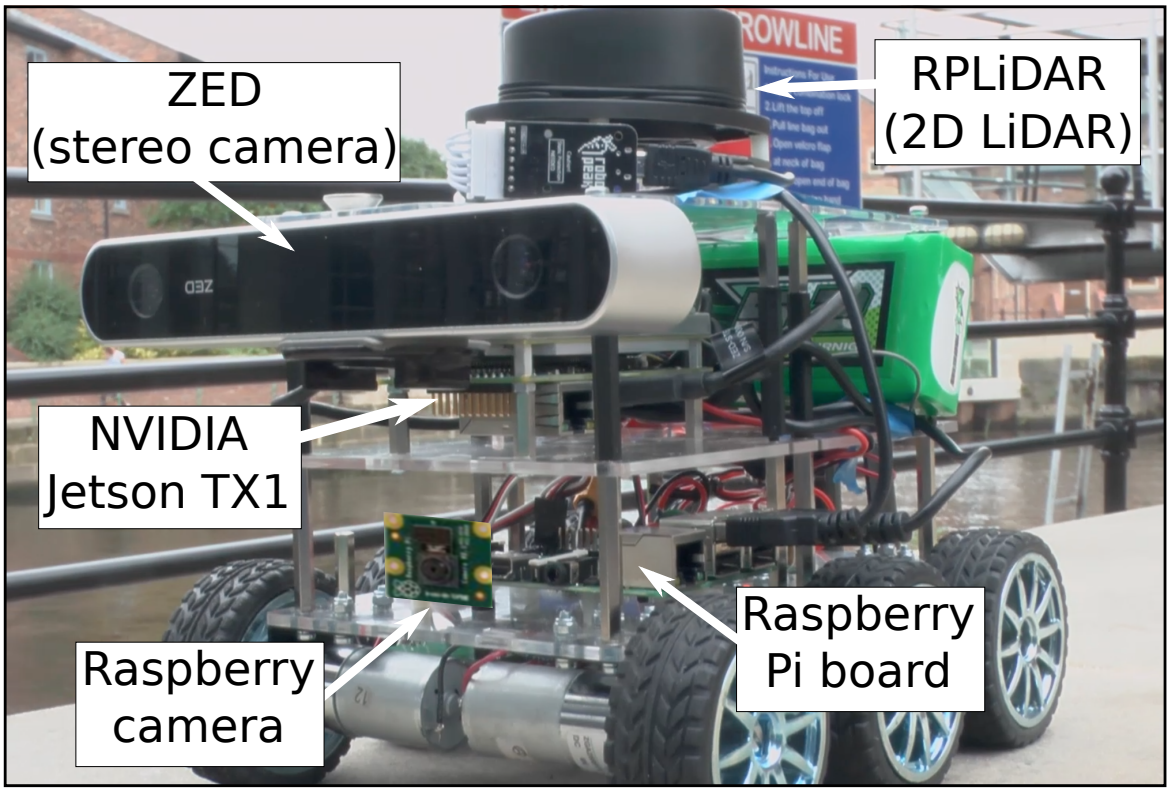


Figure 3.3: A photograph of the modified DiddyBorg robotic platform used in this work, with the relevant on-board sensors labelled in the image.

The DiddyBorg platform was modified to accommodate additional sensors and hardware. The RPLiDAR and ZED camera are mounted on top of the robot, and the Raspberry Pi camera was mounted on the front (see Figure 3.3). Both cameras are used to evaluate visual SLAM methods and the Raspberry Pi camera was also used for inspection applications (see Section 5). The on-board processing, required for sensor operation and data collection, was provided by the Raspberry Pi and NVIDIA Jetson TX1 which was added to the platform to allow real-time visual processing. Both of these boards were running an Ubuntu-based Linux operating system, in order to interface with the Robot Operating System (see Section 3.7).

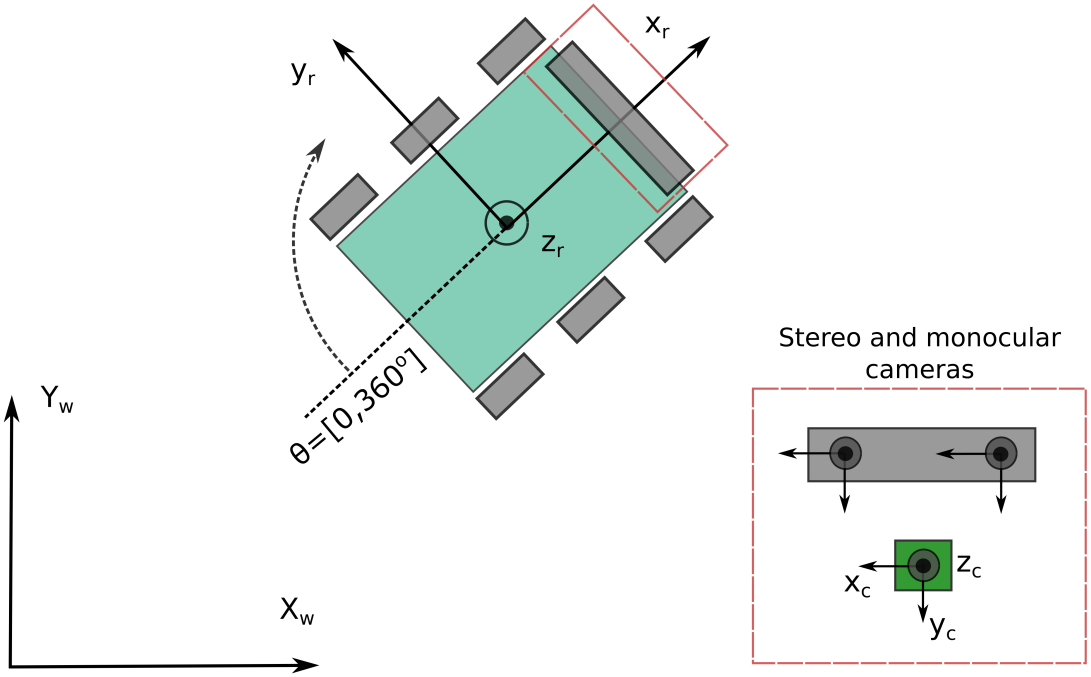


Figure 3.4: A schematic of the coordinate system for the robotic platform used in this thesis. x_r, y_r and z_r are defined relative to the robotic platform, x_c, y_c and z_c are defined relative to the camera frame and X_w and Y_w are defined relative to the world frame.

Figure 3.4 shows the reference system defined for use with the robotic platform. The defined convention follows the right-hand rule with x_r forward relative to the robotic platform, y_r left relative to the robotic platform and Z_r is upwards relative to the robot, this convention was the same for the 2D LiDAR sensor. A second reference frame was defined for the camera sensors where z_c is forward, x_c is right and y_c is down relative to the camera.

3.7 The Robot Operating System

The Robot Operating System (ROS) [7] is an open-source meta-operating system for robots, with a large on-line, open-source community. ROS was originally designed and implemented for projects developing large-scale service robots (Quigley et al., 2009; Quigley et al., 2007; Kramer and Scheutz, 2007). Software is available as packages or stacks that can be distributed, shared and developed in multiple languages with the aim of allowing code reuse in robotics research and development (Quigley et al., 2009). The software is usually created as groups of independent processes called nodes; nodes communicate by connecting to a master service and by sending messages that are organised into named topics. Nodes can send information by publishing messages on a topic and other nodes can listen for and subscribe to messages coming from topics. There are defined message types that can be used for specific purposes such as lasers scan messages, camera messages and geometry messages for navigation. ROS also provides tools for visualising and monitoring data and tools for logging and recording data that can be re-played as though running in real-time on the robot system.

For the platform used in this thesis, the 'ROS Kinetic' distribution of ROS was used to facilitate data collection. Sensor data was transferred as messages to different machines (the Raspberry-Pi, TX1 and a laptop), for processing using a wide variety of open-source algorithms or for recording data for later use. An overview of the software architecture, including an example of the software modules used with the 2D LiDAR scan data in ROS, is given in Figure 3.5.

[7] http://www.ros.org/ (Date last accessed:01/02/19)

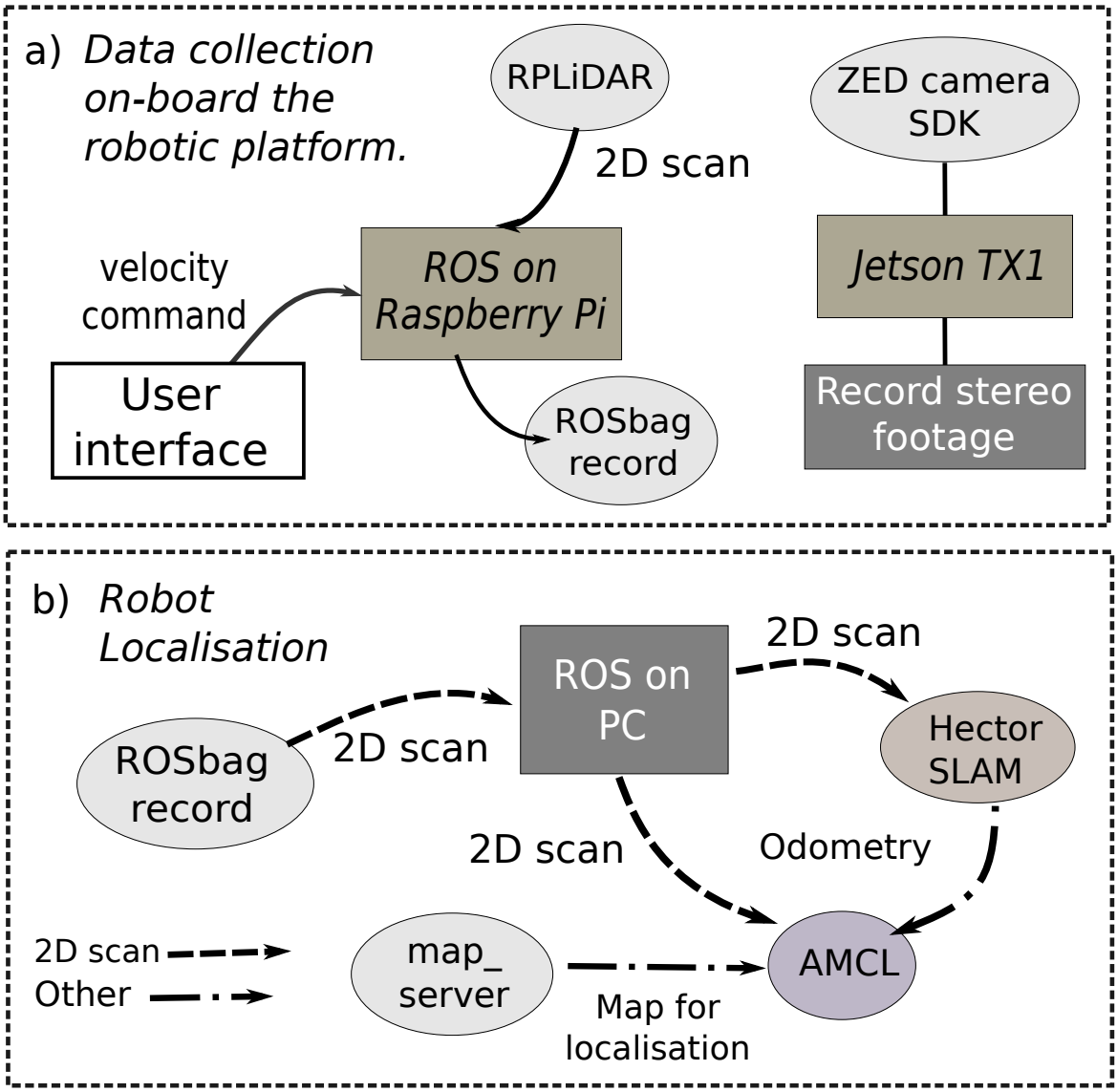


Figure 3.5: a) An overview of the tasks completed by hardware and software on-board the robot in data-collection. b) An overview of the different types of data collection and processing used with the localisation methods in this work.

As well as the reference coordinate system defined in Section 3.6, there are reference frames that are defined for use with mapping, navigation and other tasks in ROS. The relevant frames used in this thesis are as follows (Meeussen, 2010):

- base frame: a reference frame that is attached rigidly to the robot and is defined in Section 3.6.
- odometry frame: is a reference frame fixed in the world-fixed frame and provided by odometry sensors can be used as an accurate short term reference, although it drifts over time. However, the position of the robot is guaranteed to be continuous without jumps.
- map frame: is a reference frame fixed in the world-fixed frame. The robot position should not drift in the map frame, but may be subject to jumps and discontinuities

3.8 Robot motion and data collection

The robotic platform has six wheels, with three on each side of the robot. Most of the experiments in this work were carried out using tele-operation, using a motion script written in Python[8] to interface with the motors of the DiddyBorg [9]. During testing in the lab and bridge testing environments, motion commands were provided from a laptop or phone device via a local WiFi network. Figure 3.6 shows where data was transferred on-board the robot: sensor data from the 2D LiDAR and Monocular camera was transferred to the Raspberry Pi board; sensor data from the stereo camera was transferred to the Jetson TX1 board; both boards are in contact with the base-station to start and stop data recording and motion control was provided from the base-station and the Raspberry Pi to the motors. Although tele-operation was used for most experiments in this thesis, a differential drive controller was implemented in ROS using the default navigation packages. This controller

[8]https://www.python.org (Date accessed: 01/02/19)
[9]http://www.piborg.org/downloads/picoborgrev/examples.zip (Date last accessed:01/02/19)

was required for autonomous navigation, but was used primarily for simulation purposes in Chapter 4.

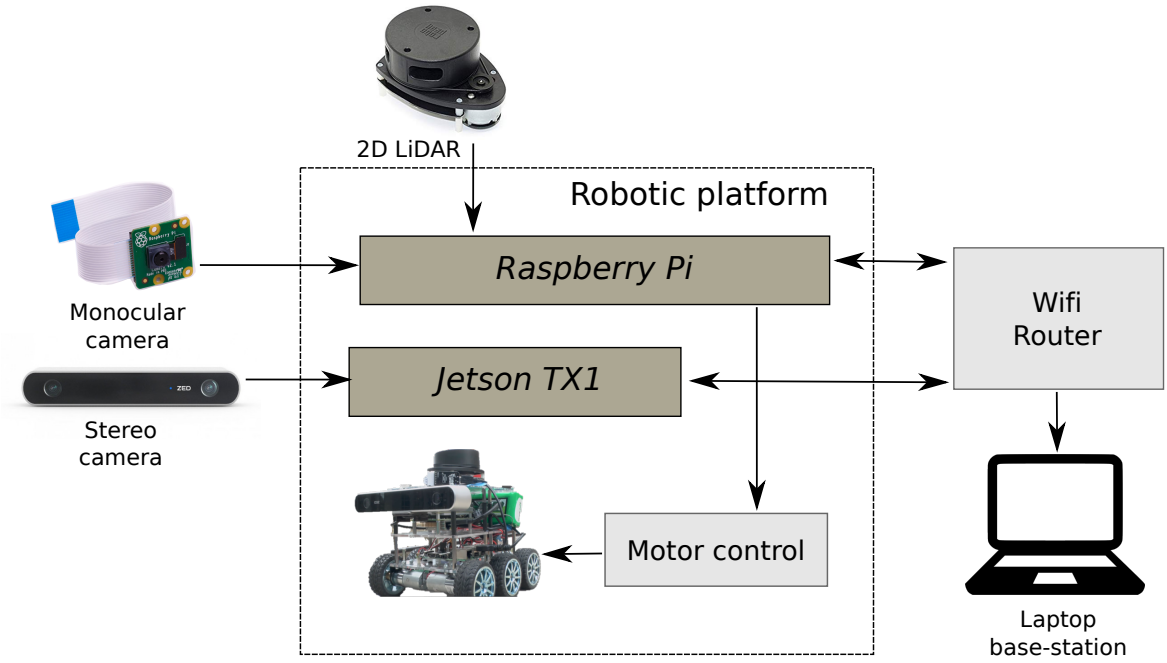


Figure 3.6: A schematic giving an overview of the architecture of the system hardware for the robotic platform. Arrows show the transfer of data to and from sensors and hardware, such as the computational boards on the robot and the laptop base-station. Pictures of 2D LiDAR (sensor images from https://www.stereolabs.com/zed/, https://coolcomponents.co.uk/ and https://www.raspberrypi.org/; date accessed: 01/02/19)

In order to test the SLAM and localisation algorithms in this work, sensor data was collected in both the bridge and lab environments. This sensor data was collected whilst tele-operating the robotic platform. Sensor data consisted primarily of RGB images from the monocular camera and stereo cameras and data from the 2D LiDAR and this data was stored in rosbags. Rosbags are a data collection format provided by the Robot Operating System (ROS) that allows real-time play-back of the sensor data. RViz (ROS visualisation software) will also be used to visualise data collected and processed using ROS.

Structure-from-Motion (SfM) (see Chapter 2, page 51) was required for map creation for localisation (Chapter 4) and inspection applications (Chapter 5). SfM uses multiple 2D image views to find the 3D geometry (i.e., the structure) of a scene or an object by taking images from different viewpoints (i.e., the camera has motion). The images that are

collected do not need to be organised/ordered, nor do the camera locations need to be planned.

When collecting data for SfM reconstructions, a specific motion of the robotic platform was required to provide sufficient overlap between images (see Section 2.6). To ensure sufficient image overlap for the reconstructions, the robotic platform was rotated on the spot by a small increment, then stopped before capturing the next photo (see Figure 3.7). This method may not be optimal for SfM reconstructions in general, which typically requires motion around the object being constructed, rather than on the spot rotation [10]. However, the motion used in Figure 3.7 was required when space is limited such as in bearing enclosures. An alternative method of collecting data for SfM might be using a camera mounted to the side of the robotic platform. However, in this work the data from the camera was also considered for inspection applications, which is better suited to a front-mounted camera. The photographic data was then processed using SfM software to create a dense 3D point cloud of the enclosure and scaled using known measurements. The resulting point cloud was then made into the map (see Chapter 4, page 83).

[10] http://3dflow.net/zephyr-doc/3DF%20Zephyr%20Manual%202500%20English.pdf (Date last accessed: 01/02/19)

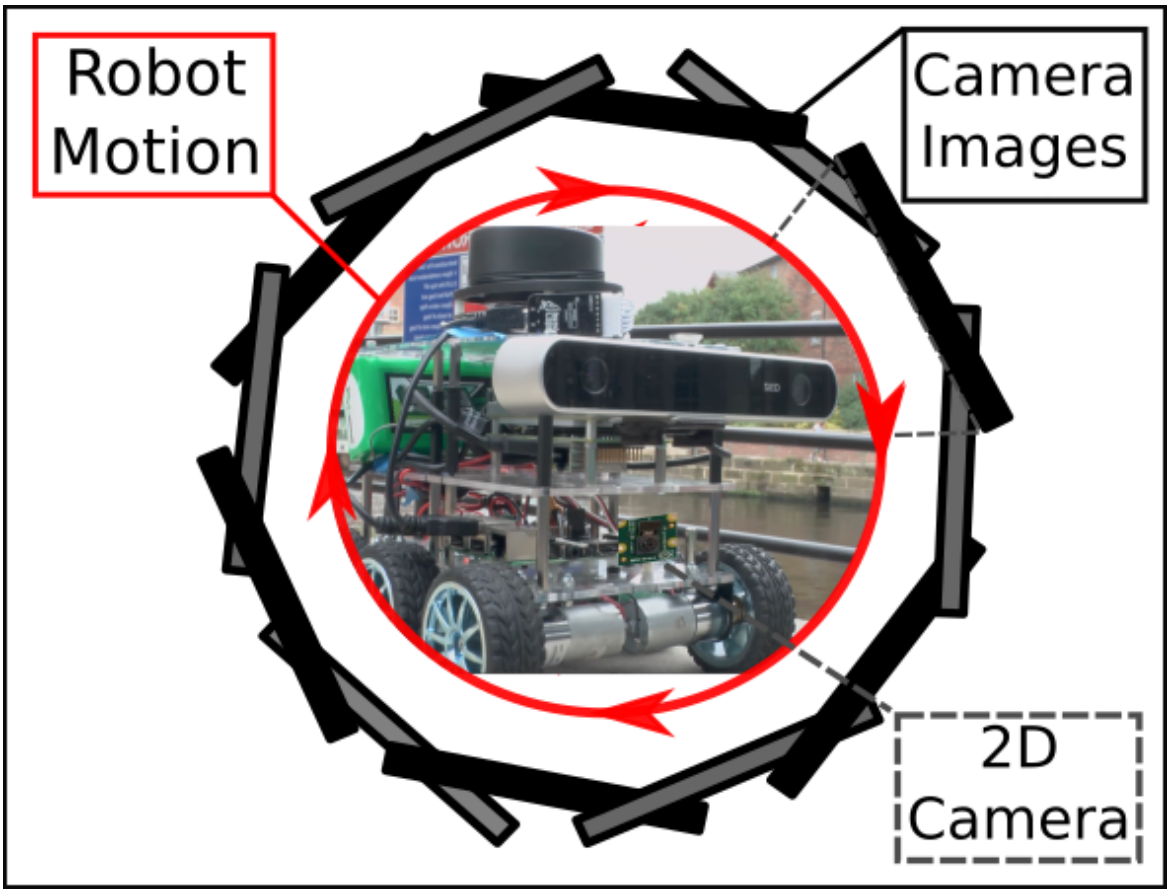


Figure 3.7: The method used by the robot to collect photographs for SfM reconstructions. The figure does not represent how many photos were taken through this process, only the manner in which the photos were collected.

3.9 Adapting the robotic platform for simulation

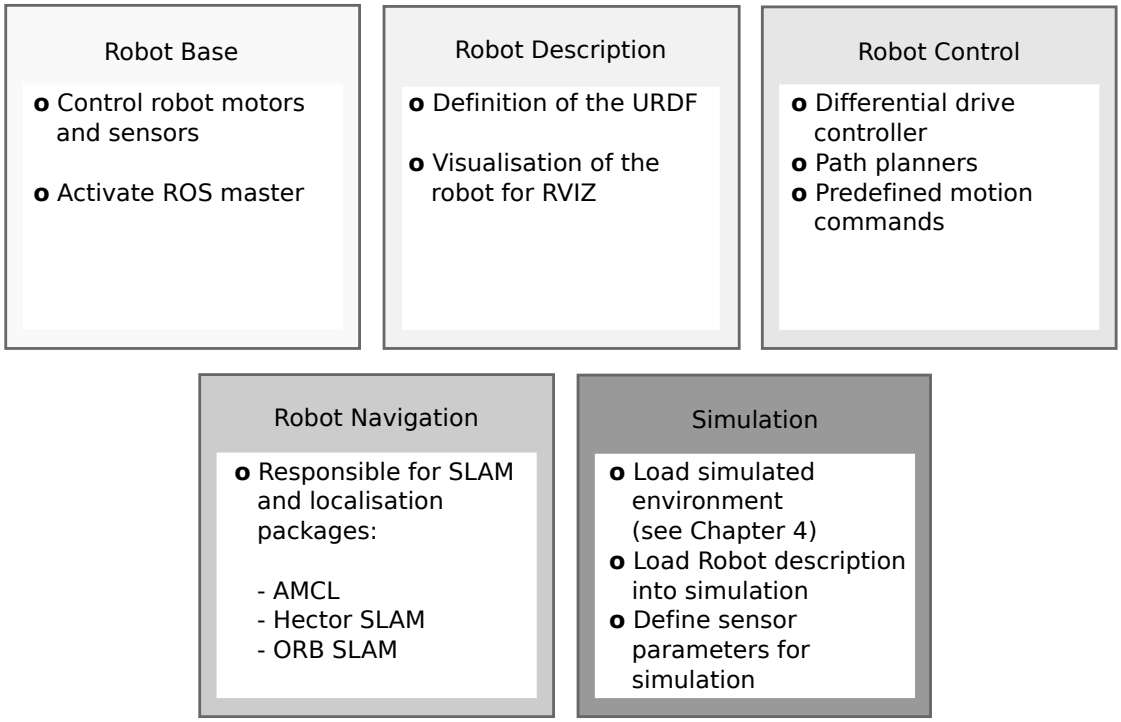


Figure 3.8: An overview of the breakdown of the tasks performed by different software modules in ROS. The simulation module is added to define the robot and environment in simulation. The remaining modules are used on the robot in the real environment.

In order to develop the robotic platform in simulation (see Chapter 4, page 119), some means of simulating the sensors onboard the robot was required. The general architecture of the different software elements that are used in this work is shown in Figure 3.8. Some understanding of creating the robot in simulation was developed using the work by Husarion's ROSbot[11].

[11] https://github.com/husarion/rosbot_description (Date accessed: 01/02/19)

As can be seen in Figure 3.8, different tasks such as control of the motors and sensors is separated from the ROS packages that are responsible from localisation and mapping. Using this approach means that the software for different tasks can be implemented on different configurations of robots. In addition, the robot description can be replaced by the simulation module to create the definition of the robot in simulation, whilst using the same ROS packages that can be used with the real robotic platform. Further details of the parameters used to define the robot geometry and the sensors in simulation is given in Appendix A.

In Figure 3.8, the robot definition is created using the unified robot description format (URDF). The URDF of the robot can be used to define the kinematics, collision elements and visual representations of the robot. URDF is an eXtensible Markup Language (XML) format that allows the definition of mechanical elements of a robot using joints and links. Links represent the rigid body elements of the robot and are used to describe solid properties such as mass and inertia of parts of the robot and also constrain where parts of the robot will collide. The connection between these links is represented by joints. Joints can be moveable as in the case as a robot with a manipulator or can be fixed and can be used to represent the location of sensors on-board the robot. Figure 3.9 shows the robotic platform defined using URDF. The links are represented as markers showing the reference system of the robot at each joint and sensor and the joints are shown as yellow lines connecting the links. The geometry of the robot has also been defined for visualisation.

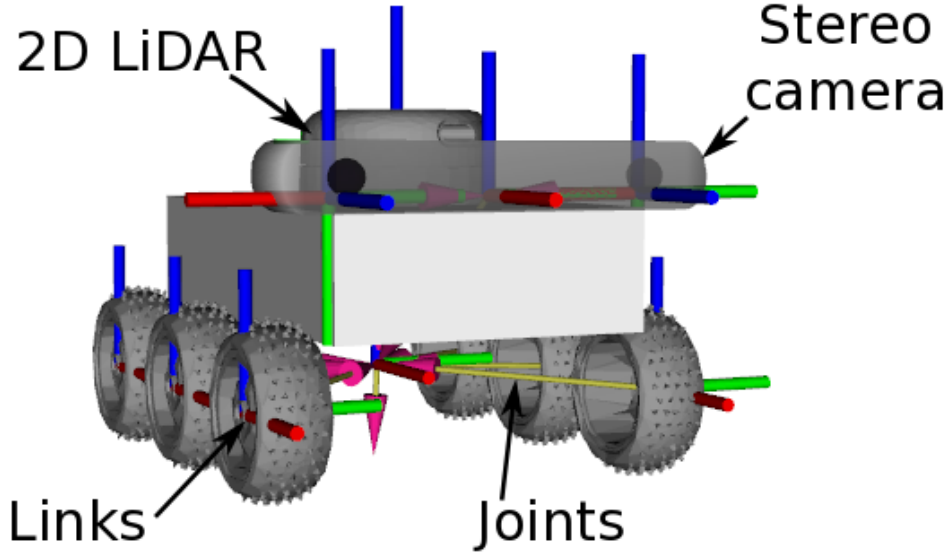


Figure 3.9: The URDF for the robotic platform used in this work. A CAD model has been defined for visualisation of the robotic platform, wheels and sensors on-board the robot during operation using ROS and Gazebo. The URDF allows the definition of collision objects and the location of the sensors on-board the robot. Elements such as wheels and sensors have cartesian coordinates defined, the position of which are defined shown (annotated).

Chapter 4

Localisation for a bridge bearing inspection robot

4.1 Introduction

The review of current literature in Chapter 1 presented a need for the development of more efficient approaches for bridge inspection, to meet the increasing demand in the amount of inspection due to ageing infrastructure and increasing number of bridges (Sutter et al., 2018). Many researchers highlighted opportunities for robotic bridge inspection (e.g. Pham and H. La (2016), Van Nguyen et al. (2018), and M. Lee et al. (2018)). For automated bridge bearing inspection, Ellenberg et al. (2016b) developed a small UAV equipped with two cameras and used a laboratory-based set-up of a bridge bearing to monitor the deflection of a rubber bearing and a steel bearing. A laboratory set-up is useful for replicating the common defects in and around the bearing, although very few experimental results were reported by Ellenberg et al. (2016b). In addition, many examples in the literature show that UAVs may not be suitable for navigating close to the bridge structure due to difficult aerodynamic conditions (Hallermann and Morgenthal, 2014), failure of navigation sensors, such as GPS and magnetometers (C. Yang et al., 2015; Hiasa et al., 2018) and certification issues; hence ground-based platforms may be more suitable for bridge bearing inspection (Lattanzi and Miller, 2017; Pham and H. La, 2016).

SLAM is a fundamental problem in robot navigation in which a robot builds a model of an unknown environment whilst concurrently determining the state of the robot within that environment (see Chapter 2, page 27). Robotic localisation is a reduced version of this problem where a pre-existing map is utilised, and only the position of the robot requires calculation. For inspection applications it is potentially beneficial to have a pre-existing map. On one hand, the robotic platform can be directed to a specific location in the environment and, for infrastructure applications, there is often existing inspection or surveying data that can be utilised to create maps for robot localisation. On the other hand, localisation in a known map has its own disadvantages, since the environment is changeable and the map may become inconsistent with the real environment. In this chapter, a method for map creation using existing point cloud data is established and tested using localisation algorithms and compared to SLAM methods, where the map was created using sensors on-board the robotic platform.

Of the papers reviewed in Chapter 1, localisation was implemented on the ground-based platform using an EKF by combining sensor information from a GPS, an IMU and wheel odometry (H. M. La et al., 2013; H. La et al., 2013), which showed better localisation results than using GPS only when navigating around a university campus (Gibb et al., 2017). In addition, Pham and H. La (2016) integrated sensors for localisation and mapping, and highlighted that the development of visual-inertial odometry to support autonomous navigation was an important step in required future work for their magnetic wheeled platform, but no localisation or mapping was investigated in their current research. The majority of UAV platforms reviewed in Chapter 1 used GPS for localisation, typically by setting way-points for the UAV to follow (C. Yang et al., 2015), and had one or two human operators, whom were required when turbulence around the bridge caused difficult flying conditions (Hallermann and Morgenthal, 2014) and, in the case of Hiasa et al. (2018), when the metal railings on the bridge caused errors in the magnetometer readings. Using GPS for localisation was also found to be insufficient when inspecting underneath the bridge, where the GPS signal was weak (C. Yang et al., 2015).

In this chapter, it is assumed that there is some mechanism in place for transporting the robotic platform to the bearing enclosure. This transportation could take many forms, such as using a combined robotic system using a UAV to drop the robotic off or deployment by the bridge inspectors using a system similar to a snooper truck, or perhaps installing the robot permanently at the bridge. None of these scenarios have been researched in this chapter, and they would require addressing as part of the future development of the robotic platform. It is also assumed that the majority of the inspection environments where the robotic platform could be deployed would be of a known type, and that the type of bearings present in these environments are also known. Table 4.1 outlines a set of scenarios, with the suggestion that the robotic platform could be deployed in scenarios where the bearing enclosure and environment are known and include certain features, such as adequate lighting and a level floor plane on which the robot would traverse. It is also assumed that some method of initially mapping the environment (e.g., through terrestrial LiDAR, SfM or using existing CAD data) is in place, which could also be used as an opportunity to check the suitability of the environment and type of bearing in the environment.

Table 4.1: Assumed scenarios for robotic bearing inspection.

		Enclosure Type	
		Unknown	Known
Bearing Type	Unknown	0%	5%
	Known	20%	75%

In Chapters 2 and 3, the different types of sensors that are commonly used for autonomous robotic navigation were described and the most suitable sensors for the application of bridge bearing inspection were chosen, i.e., the 2D LiDAR and monocular and stereo cameras. The SLAM and localisation approaches that can be used with each sensor were reviewed and Hector SLAM, ORB SLAM and AMCL were selected. The methodology for these approaches was described in Chapter 2, on page 42. To summarise, these choices were made because:

- Hector SLAM, ORB SLAM and AMCL have been demonstrated in urban environments (e.g., Kohlbrecher et al. (2011), Kohlbrecher et al. (2014), Rohde et al. (2016), and Mur-Artal et al. (2015)).
- Hector SLAM and ORB SLAM can be used to create maps of the environment.
- AMCL can be used to localise a robot in a pre-existing map.
- The data from the 2D LiDAR can be compared to existing point cloud data and used for both Hector SLAM and AMCL and to provide odometry.
- Hector SLAM, ORB SLAM and AMCL methods have been demonstrated on the Raspberry-Pi or the Nvidia Jetson TX1 (e.g., Kurtzer et al. (2017), Ponnu et al. (2016)).

This chapter describes a process for creating a map using existing inspection data for the purpose of robotic localisation in inspection environments. Existing SLAM and localisation algorithms for different sensors in a real inspection environment will be evaluated and these approaches will be compared in a real bridge environment. In addition, the inspection data will be used to create an environment for simulating the real bridge environment. Finally, further recommendations are made for how this approach can be extended to autonomous robotic navigation for bridge bearing inspection.

Table 4.2: A summary of the methods compared in this Chapter.

Method (map)	Sensor	Environment / Localisation	Section
Hector SLAM	2D LiDAR	Laboratory environment	Section 4.5
AMCL-SfM	2D LiDAR	Laboratory environment	Section 4.5
AMCL-Hector	2D LiDAR	Laboratory environment	Section 4.5
ORB SLAM	Stereo and Monocular camera	Laboratory environment	Section 4.6
Hector SLAM	2D LiDAR	Bridge environment	Section 4.8
AMCL-SfM (map from SfM)	2D LiDAR	Bridge environment	Section 4.8
AMCL-LiDAR (map from terrestrial LiDAR)	2D LiDAR	Bridge environment	Section 4.8
AMCL-Hector (map from Hector SLAM)	2D LiDAR	Bridge environment	Section 4.8

4.2 Maps for localisation

In Chapter 2, the requirement for the creation of a map for robotic navigation was described. Map creation can either be performed at the same time as localisation (i.e., SLAM) or can be be performed in advance (i.e., localisation only). These maps are often represented in either a topological format, where the map is decomposed into significant places in the environment, or a metric representation, where the map is decomposed into fine-grained cells of uniform size (Thrun et al., 2005).

Of the SLAM methods described in Chapter 2, ORB-SLAM, uses a topological map to represent the environment, using a sparse land-mark based map of key features in the environment. In contrast, Hector SLAM creates a metric map, where the environment is split into grid cells that contain the probability that each cell is occupied by an object in the real world. This approach is known as occupancy mapping and is described in Section 2.3. Example maps were shown in Chapter 2 on pages 47 and 44 for ORB SLAM and Hector SLAM, respectively.

The localisation method AMCL also uses an occupancy map for localisation. Both topological and metric map representations can be used for localisation by the same sensors that created them, but they can also be created using other approaches. For example, Behzadian et al. (2015) successfully performed localisation in simulated environments using hand-drawn maps and Monte-Carlo Localisation for areas where no accurate maps were available. In the inspection environment it may not be possible to create maps using sensors on-board a robot since tele-operation of the platform may be required, which may be difficult in some environments. An alternative approach to obtaining maps for localisation in the inspection environment is described in 4.3.

In addition to finding the current position of the robot, a map of the environment may be useful for providing navigation goals to a robotic platform. Furthermore, maps have other uses for tasks where a human operator may be involved, such as visualisation by a human operator (Cadena et al., 2016). However, in general it is expected that the bridge inspection environment is a dynamic environment, which may cause localisation problems if changes

in the environment occur and the pre-defined map no longer represents the environment. The effectiveness of localisation using pre-defined maps will be investigated in this chapter.

4.3 Generating maps from point cloud data

As discussed in Chapter 1, camera data is commonly collected as part of robotic bridge inspection tasks (e.g., Ellenberg et al. (2016b), Le et al. (2017), and Sutter et al. (2018)). This data is often reviewed by human inspectors for aiding visual inspection and may be processed to monitor the degradation of a structure (e.g., Kim et al. (2014)) or to recover the geometry of the structure, especially using 3D reconstruction methods such as SfM (see Chapter 2, page 51) and creating photo-mosaics (Lattanzi and Miller, 2015; S. Chen et al., 2011). Other examples in the literature show the use of laser scanning (Sacks et al., 2018; Javadnejad et al., 2017) and CAD drawings (Sutter et al., 2018) to supplement bridge inspection tasks. Since the data collected from terrestrial LiDAR scanning and SfM is typically represented as a 3D point cloud, it lends itself to map creation using occupancy mapping since points representing objects in the environment can be discretised into a grid-based representation. The conversion of 3D point clouds into 2D occupancy maps will form the basis of the map creation method in this chapter.

In this chapter, occupancy maps created using Hector SLAM are compared to occupancy maps created from terrestrial LiDAR or SfM data, both in a laboratory environment and a real bridge environment. First, in order to collect data for SfM, photographs were collected by tele-operating the robot around the test enclosure while taking photographs at multiple locations using the approach shown in Figure 3.7, page 73. Next, the photographic data was processed using SfM software to create a dense 3D point cloud of the enclosure and scaled using known measurements (Section 4.4). The method for obtaining data for SfM was described further in Chapter 3 on page 70 and the external and internal camera parameters were calculated in the SfM software. The approach for collecting 3D terrestrial LiDAR data is described in Section 4.9. A summary of the methods used in this Chapter, the sensor data they use and the relevant sections is given in Table 4.2.

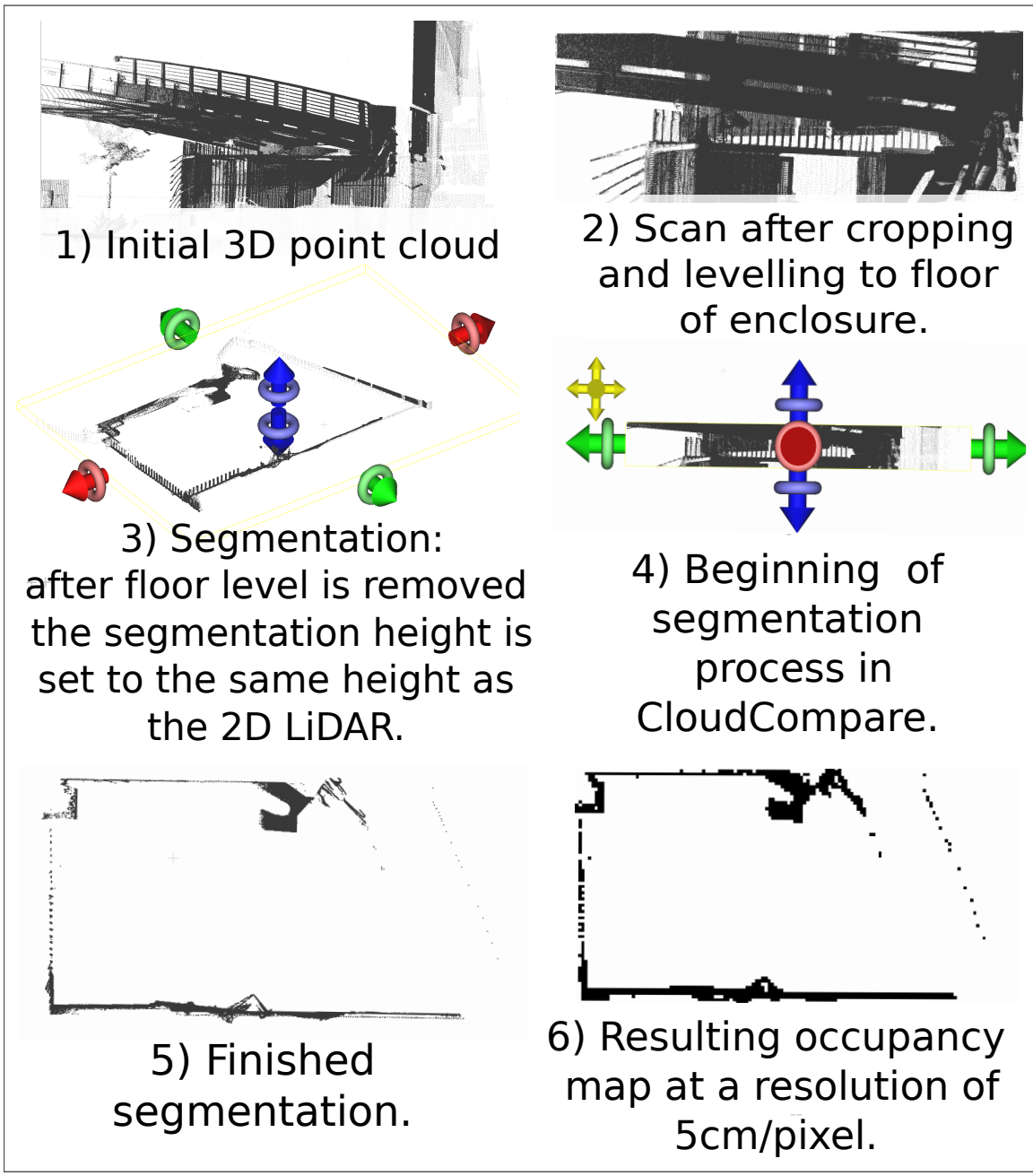


Figure 4.1: An overview of the steps for converting a 3D point cloud (either from SfM or a 3D terrestrial LiDAR) into a 2D occupancy map for use in 2D localisation using Adaptive Monte-Carlo Localisation.

Next, the process for converting the 3D SfM and terrestrial LiDAR point clouds into a 2D occupancy map is described. The steps performed to create the occupancy maps from 3D point cloud data are as follows (also depicted in Figure 4.1): the 3D point cloud was cropped to the area of interest around the bearing enclosure using the point cloud manipulation software CloudCompare[1]; the floor level of the point cloud was determined using the levelling tool in CloudCompare by manually selecting points that are at floor level (see step 2 in Figure 4.1); from this level a constant height of the LiDAR in the environment is assumed and a slice through the point cloud was extracted (using the segmenting tool) corresponding to the location of the 2D LiDAR on the robot (see steps 3 and 4 in Figure 4.1). This slice was three-dimensional, with two dimensions corresponding to the plane of the sensor data collected by the 2D LiDAR and a third dimension, of approximately the same height from the floor of the bearing enclosure as the 2D LiDAR sensor is on the robot, in order to ensure that the majority of points that could be detected by the 2D LiDAR are included in the map. Finally, a binary occupancy grid was extracted from this point cloud slice using the BinaryOccupancyGrid function in the MATLAB Robotics Systems Toolbox[2] (see step 6 in Figure 4.1). Example occupancy maps created using this process are shown in Figure 4.2.

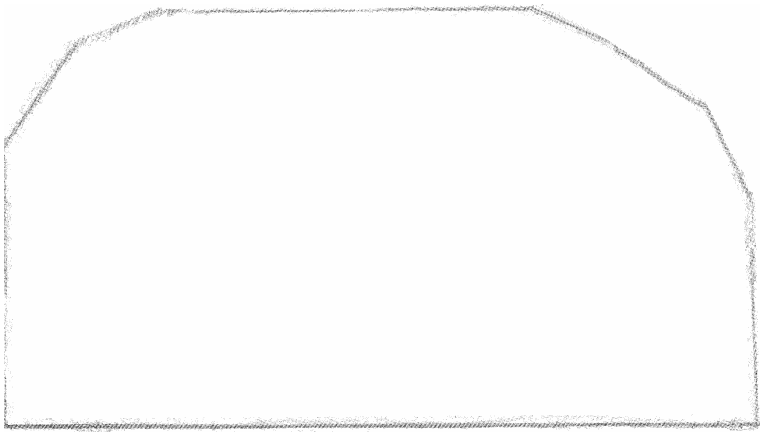
An occupancy map is split into uniform grid cells representing locations in the real-world. It was possible to vary the resolution of the occupancy maps created in this chapter by varying the grid size, e.g., to $1\,\mathrm{cm}$/pixel or $1\,\mathrm{mm}$/pixel (highest available resolution for the BinaryOccupancyGrid function). For the SfM point clouds, scaling must be applied before creating the occupancy map, see Section 4.4. For Hector SLAM, the occupancy maps had a resolution of $5\,\mathrm{cm}$/pixel, so that each grid cell represents $5\,\mathrm{cm}$ in the real-world (higher resolutions were tested, but gave noisy results that were unsuitable).

[1] http://www.cloudcompare.org/ (Date last accessed:01/02/19)
[2] https://uk.mathworks.com/products/robotics.html (Date last accessed:01/02/19)

Since the slice taken from the point cloud data was 3D, and some method was required to represent this slice in the 2D occupancy map. A grid cell in the map was considered occupied if a point from the point cloud was present in the grid cell. Since the grid cell can only be set as occupied or unoccupied, multiple points in the same grid cell were discarded to give a 2D map. The effect of varying the resolution of the occupancy maps created from point clouds will be discussed further in Section 4.10. 2D occupancy maps are required to work with AMCL, future work could investigate the expansion of this method into 3D.

(i): Example sparse SfM point cloud using data collected in the laboratory environment.

(ii): Example occupancy map, with a resolution of $1\,\mathrm{cm}$ using the SfM point cloud shown in Figure 4.2i.



(iii): Example occupancy map, with a resolution of $1\,\mathrm{mm}$ using the SfM point cloud shown in Figure 4.2i.

Figure 4.2: Example output of SfM data-collection from the laboratory environment (4.2i) and two example occupancy maps created from SfM data collected in the laboratory environment with a grid resolution of $1\,\mathrm{cm}$ per grid square (4.2ii) and $1\,\mathrm{mm}$ (4.2iii).

4.4 Scaling the SfM point cloud

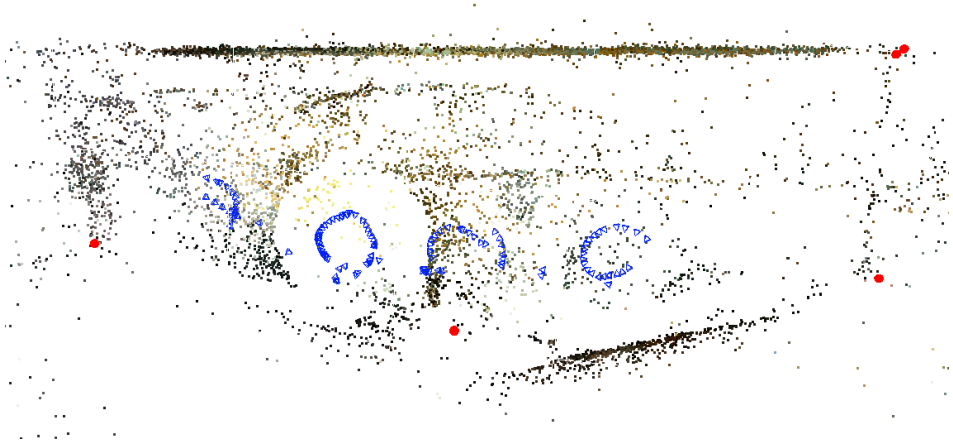
The 3D reconstruction software Zephyr-Aerial (Zephyr)[3], produced by the company 3DFlow, was used to generate the 3D point clouds from RGB camera images collected by the robotic platform. The first output from the software was the SfM point cloud – a sparse point cloud. More details can be added to the sparse, scaled, point clouds using Multi-View Stereo (MVS), to create a dense point cloud. SfM is described further in Chapter 3 on page 70. Exemplar sparse and dense point clouds, which were generated using Zephyr, are shown in Figure 4.3i and 4.3ii, respectively.

It should be noted that the sparse point clouds generated by SfM are at an arbitrary scale; therefore some method of scaling was required to recover the global scale of the point cloud. To scale the point cloud, control points were selected from the photographs used for the SfM reconstruction using the control point selection software in Zephyr (see Figure 4.3iii, where control points are shown as red dots). Using this software, multiple instances of the control point in different images were selected. Each control point must be selected in a minimum of two image views [4], although accuracy of the control point location improves if more images are selected.

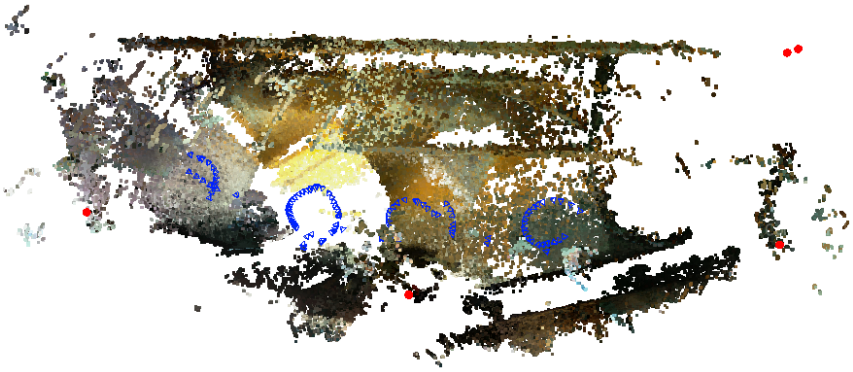
Visually distinguishable features, such as the corners of the bearing pads in the bridge environment, were selected as control points (see Figure 4.3iii). The world-scale was recovered using measurements taken manually with a tape measure for each control point from a datum. Approximately eight control points were selected from across the laboratory and bridge environments. It is also possible to provide scale to the point clouds using dimensions from construction drawings. However, it may be challenging to match these dimensions with visible features from the photographic dataset. Scaling can also be performed directly by matching points in the SfM cloud with points in a terrestrial laser scan, but this is only possible if laser scan data is available. The effectiveness of this method for scaling the SfM point clouds is validated against 3D terrestrial LiDAR data in Section 4.9.

[3] https://www.3dflow.net/ (Date last accessed: 01/02/19)

[4] http://3dflow.net/zephyr-doc/3DF%20Zephyr%20Manual%202500%20English.pdf (Date last accessed: 01/02/19)



(i): Example sparse SfM point cloud using photographs collected in the bridge environment. The blue triangles show the location of the robot when it collected the photographs.



(ii): Example dense MVS point cloud from sparse SfM data in Figure 4.3i.

(iii): Selection of control points for scaling

Figure 4.3: Example SfM data from the bridge bearing enclosure (4.3i), with dense data added using MVS (4.3ii) and scaling with control point selection (4.3iii). Control points are shown as red dots in all three images.

4.5 Results from the laboratory environment

The laboratory environment used for testing the localisation approaches was an unfinished, domestic structure. This environment had similar textures to the bridge environment and similar lighting conditions, with bare concrete walls, naturally changing lighting conditions and dark or unlit areas. The testing area was approximately $2.5\,\mathrm{m}$ by $1.35\,\mathrm{m}$, which is a similar size to the bearing enclosure, described in Chapter 3 on page 55. The SLAM and localisation methods for the 2D LiDAR, described in Chapter 2, will be evaluated in this chapter using the data collected in this environment.

To compare the different mapping approaches for localisation, the robot was moved around the test enclosure to collect 2D LiDAR data using pre-programmed velocity commands. A reference trajectory was determined from the robot odometry (which was calculated using the 2D LiDAR processed by the Hector-SLAM ROS package) and velocity commands. Repetitions of the trajectory were performed to test its accuracy and repeatability. The 2D LiDAR data was then post-processed for use in mapping and AMCL. The maps tested for localisation with AMCL in the laboratory environment were:

1. A map created from Hector SLAM.
2. A map created from SfM point cloud data.

Predefined motion commands were given to the robot and the associated odometry information from the 2D LiDAR was used as a reference trajectory to compare the performance of the localisation and SLAM methods and was recorded for each run of the experiment. The experiments for each type of map were repeated three times, and the 2D LiDAR data was recorded as rosbags, the ROS data collection tool. The SLAM and localisation algorithms were then applied to the data collected in the rosbags at a later point.

In the laboratory environment, the trajectories resulting from AMCL when localising using the map created from Hector SLAM (referred to henceforth as AMCL-Hector) are plotted against the trajectories from AMCL when localising in the map created from a SfM point cloud (referred to henceforth as AMCL-SfM), and both trajectories are compared

to the reference odometry trajectory for the robot and to the trajectory calculated by Hector SLAM. As described in Chapter 2, in AMCL, the knowledge the robot holds about its current location is represented by a set of particles. The current sensor information is compared to the map of the environment to calculate a guess for the current position of the robot. The uncertainty that the current pose is correct is represented by a 3-by-3 covariance matrix and can be visualised as an ellipse by using the eigenvalues of the covariance matrix to calculate the minor and major axes of the ellipse and the eigenvectors of the ellipse to calculate the rotation of the ellipse. An example of the covariance ellipses for the AMCL particles can be seen in 4.4. The dimensions of the ellipses in this work were calculated using a python script [5].

[5]https://github.com/joferkington/oost_paper_code (Date accessed: 01/02/19)

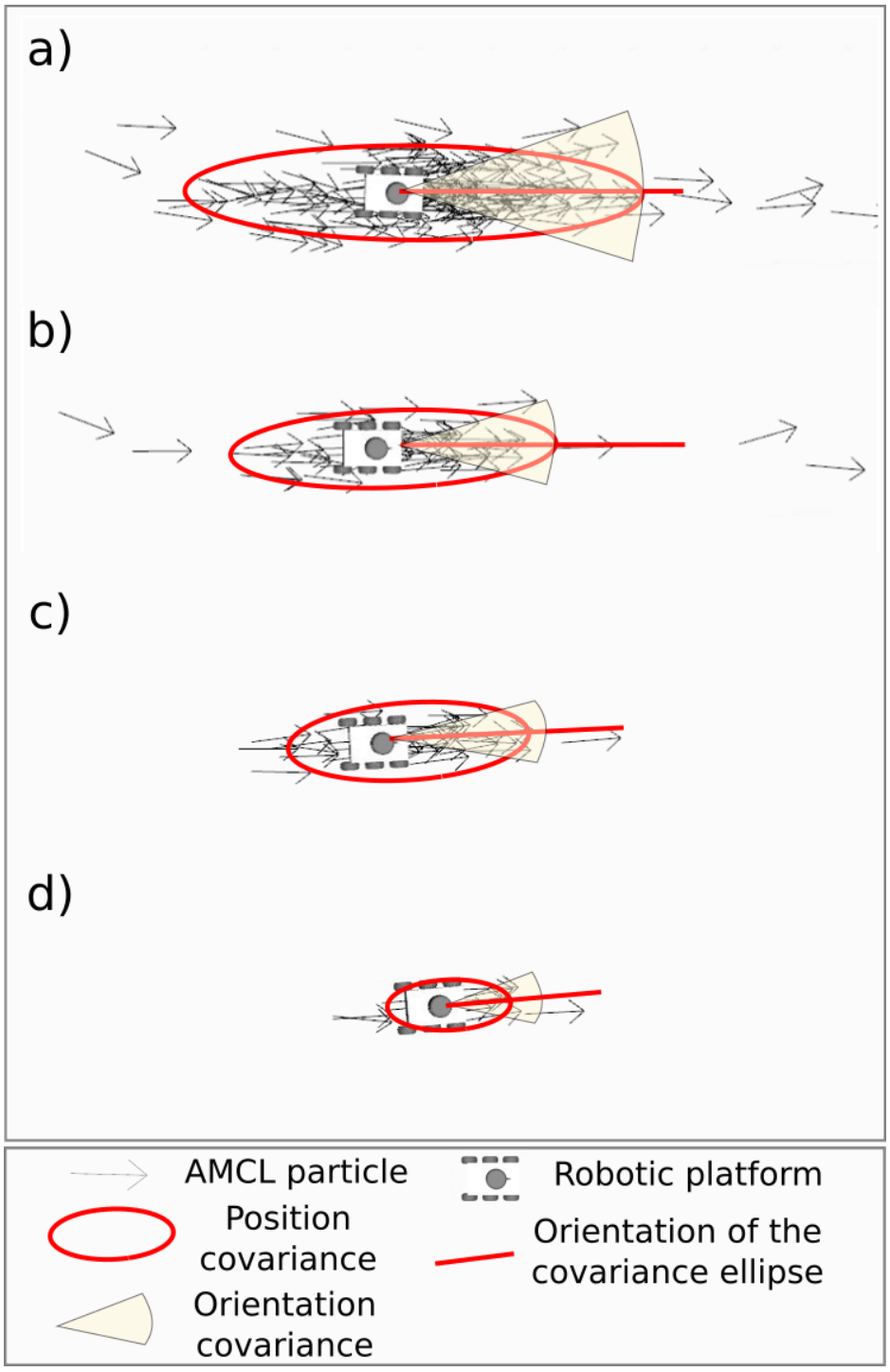


Figure 4.4: Steps a to d show sequential steps in the trajectory of the robot for local AMCL in a known map (not shown). As the robot moves, the spread of the particles (black arrows) decreases showing an increasing certainty in the position estimate. The spread of the particles can also be represented by an ellipse, the dimensions of which are calculated from the eigenvalues of the covariance matrix for AMCL.

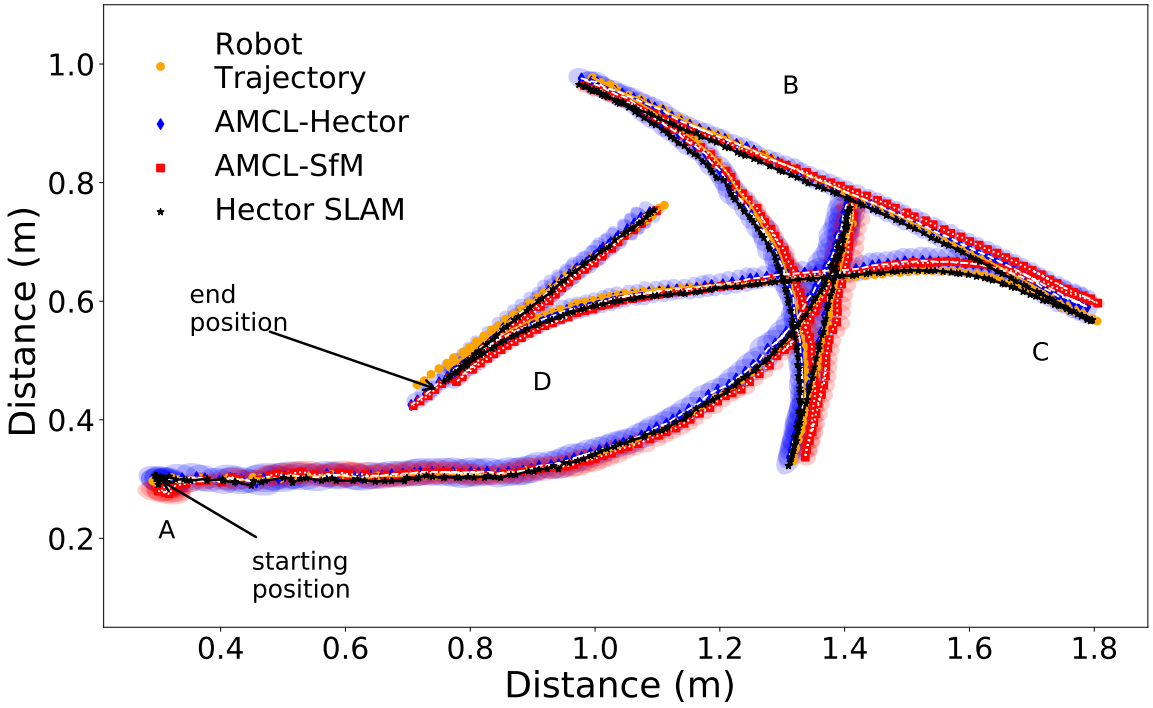
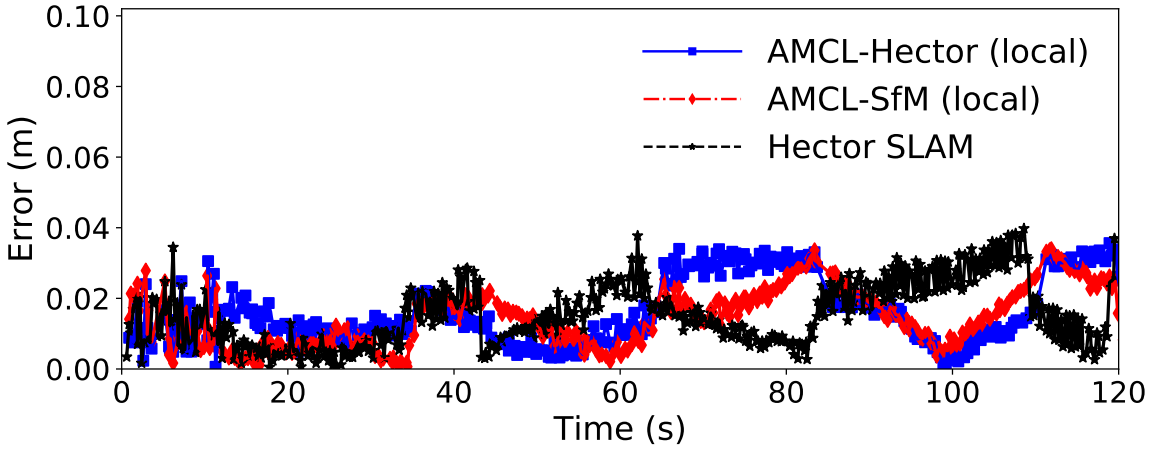


Figure 4.5: A comparison of the trajectories calculated using local AMCL-SfM and AMCL-Hector, the trajectory calculated by Hector SLAM and the reference trajectory calculated from odometry information. The ellipses that represent the covariance of the pose calculated by AMCL-Hector and AMCL-SfM are also plotted at each step in the trajectory. The start and end points are marked, alongside locations of interest (A-D). The associated errors and covariance for each point in the robot trajectory can be seen in Figure 4.6.

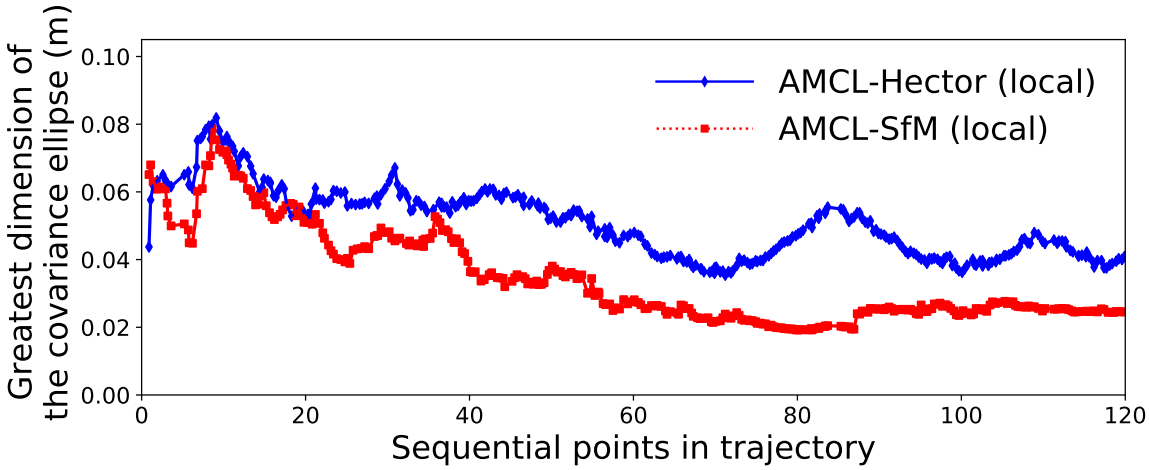
For local AMCL, an initial guess for the position and orientation of the robot is required and was provided by a user-input as a Cartesian map coordinate, and orientation was provided as an angle with respect to the origin of the map. For global AMCL, particles are sampled across the full map and no additional input was required to calculate the initial position of the robot. Furthermore, the spread (i.e., confidence) of the particles in the AMCL calculations are visualised in Figure 4.5 by plotting ellipses representing the covariance of the robot pose at each point in the robot's trajectory and also graphically in Figure 4.6ii, where a small ellipse represents a small spread of particles in the AMCL calculations.

In Figure 4.5, the initial errors for the local AMCL trajectories and the reference odometry trajectory (errors plotted in Figure 4.6i) are less than $3\,\mathrm{cm}$ for all methods. Over several repetitions (see Figure 4.7) this initial error was as large as $10\,\mathrm{cm}$ for AMCL-SfM and $8\,\mathrm{cm}$ and AMCL-Hector (both in the third repetition – see Figure 4.7), where there was an error in the initial position guess of approximately $4\,\mathrm{cm}$, which causes recalculation of the robot's position over the first 10 time-steps before converging to the reference trajectory. Once the robot began to move, the error remained below $4\,\mathrm{cm}$ for all AMCL methods for the remainder of the trajectory. This error was comparable to Hector SLAM for all repetitions, which also had a maximum error of $4\,\mathrm{cm}$. Slight peaks in the error value occurred at points where the robot changes direction, such as the regions labelled B and C in Figure 4.5.

In Figure 4.6ii, the greatest dimension of the covariance ellipses for AMCL are plotted for each point in the robot trajectory to show the change in the spread of the particles over time. The uncertainty of the AMCL particles for local AMCL, are double the initial trajectory error with values around $6\,\mathrm{cm}$ in Figure 4.6ii ($3\,\mathrm{cm}$ either side of the robot). , the The spread of particles was initially large to ensure calculation of the correct position and the uncertainty increased. Once this position was calculated, the uncertainty and the number of particles reduced and the covariance value fluctuates around $5\,\mathrm{cm}$ for AMCL-Hector and $3\,\mathrm{cm}$ for ACML-SfM for the rest of the robot motion.



(i): Comparing the errors between the reference trajectory of the robot and the trajectories calculated by both local AMCL-Hector and AMCL-SfM for each point in the reference robot trajectory.



(ii): Comparing the covariance of the AMCL particles over the duration of the robot trajectory, as calculated from the greatest dimension of the covariance ellipses seen in Figure 4.5

.

Figure 4.6: A comparison of the error between the trajectories calculated using different localisation and SLAM approaches and the reference robot trajectory (4.6i) and covariance plots for AMCL with different initial maps in the laboratory environment (4.6ii).

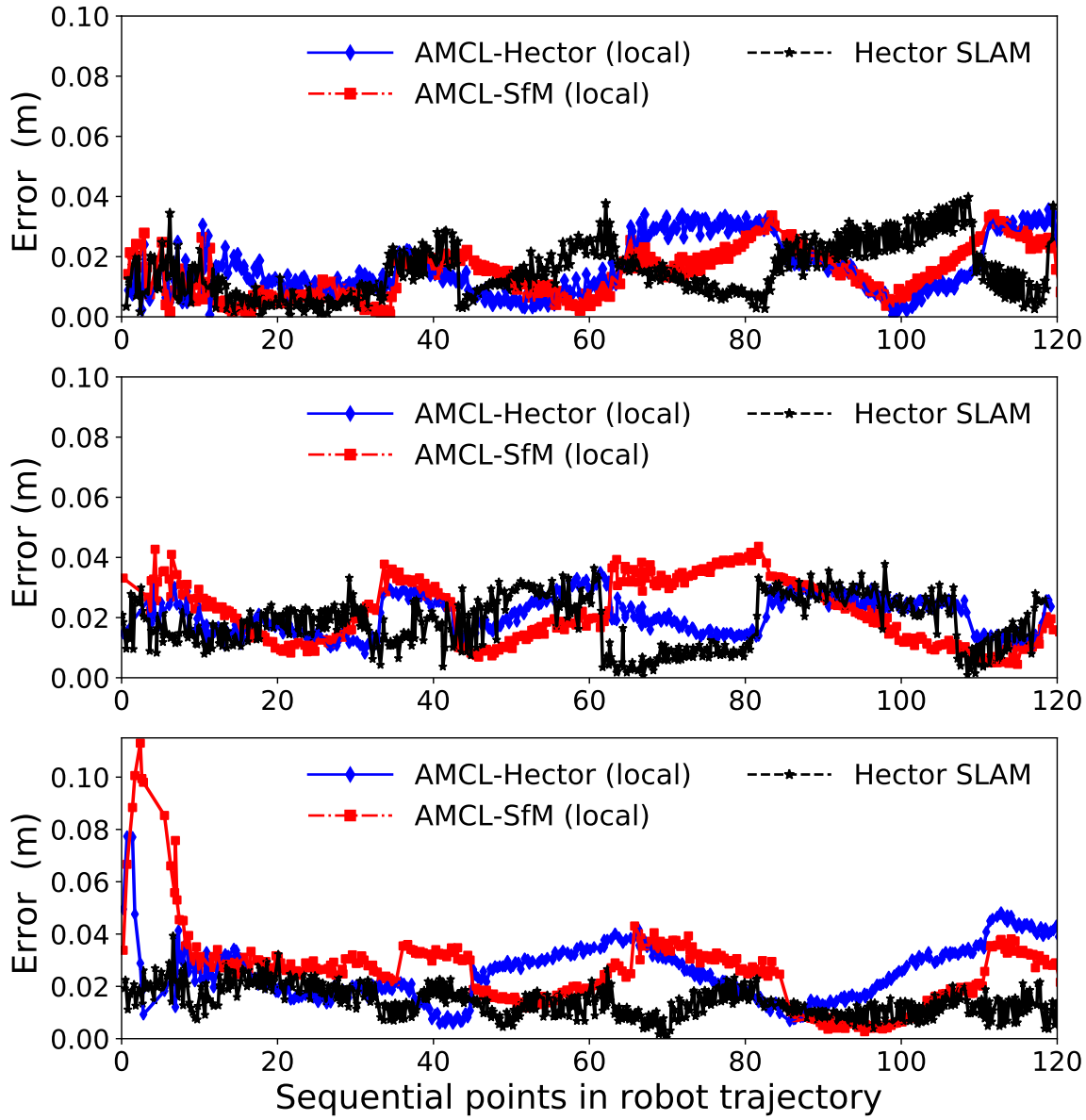


Figure 4.7: A comparison of the error between local AMCL-Hector and AMCL-SfM when compared to the reference robot trajectory for three repetitions of a set of input commands to the robot.

Furthermore, AMCL is robust to small changes in the map. Figure 4.8 shows that the LiDAR sensor readings (red dots in Figure 4.8) do not entirely overlap with the map points (black pixels in Figure 4.8). The difference in the map in Figure 4.8 was due to a wall that can be observed from the 2D LiDAR. However, this wall does not feature in the map because the camera height was positioned lower than the LiDAR on the robot and the wall was not visible in the reconstruction. The second difference (Figure 4.8) was due to a change in the environment layout. Since the environment was made of temporary materials, there was a slight change in the shape of the enclosure between the collections of photographs for the SfM dataset and the testing of AMCL using the resulting map. These changes were caused by slight movements of the temporary structure between experiments (since the experiments were carried out on several different days). Some robustness is essential for localisation in an environment which is prone to changes, as is expected for the bridge bearing environment.

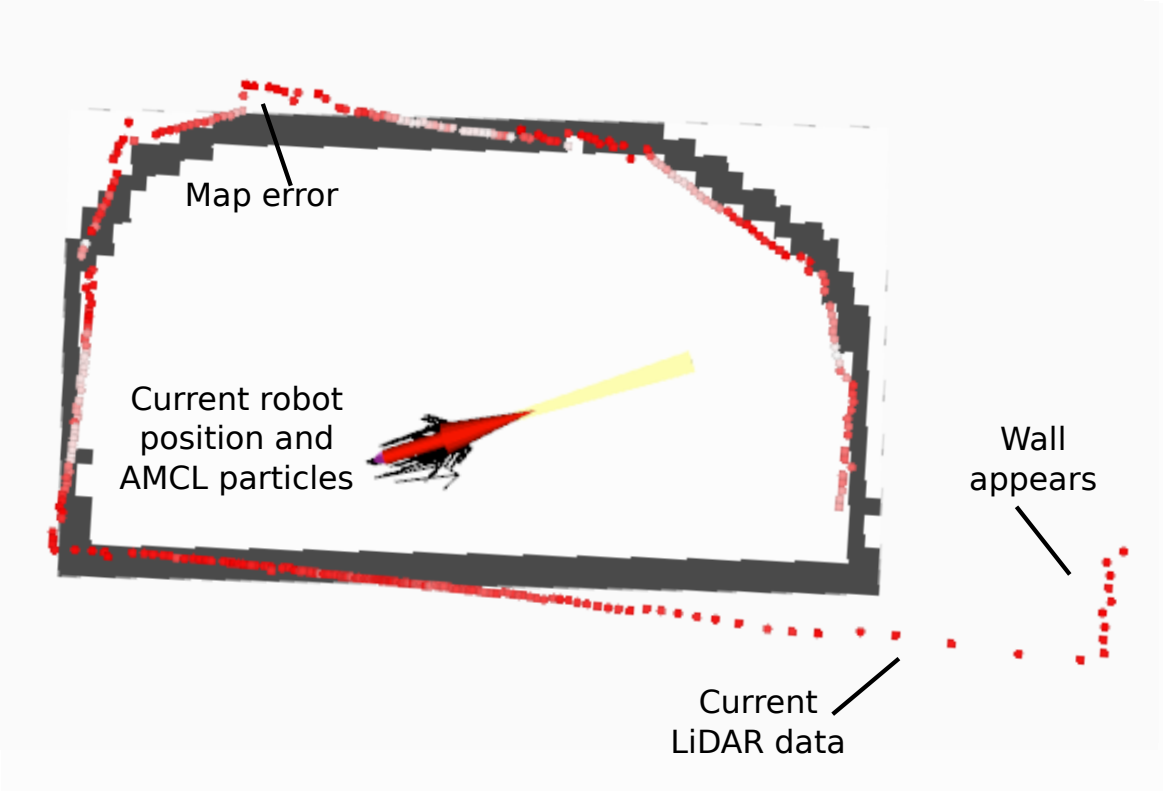
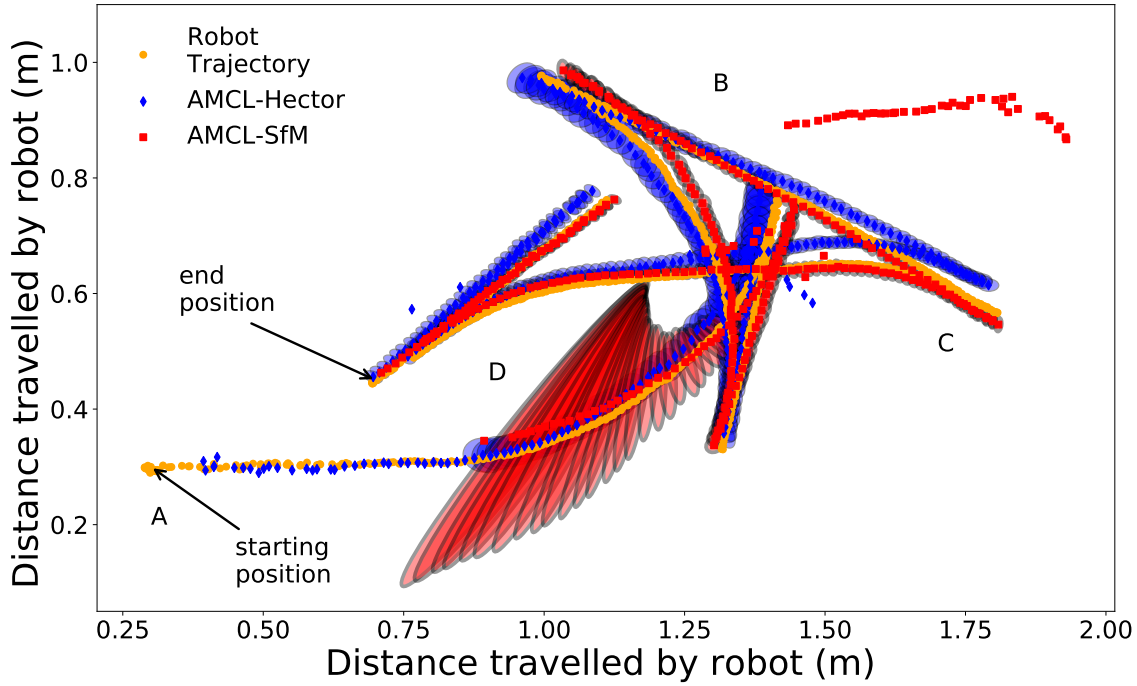
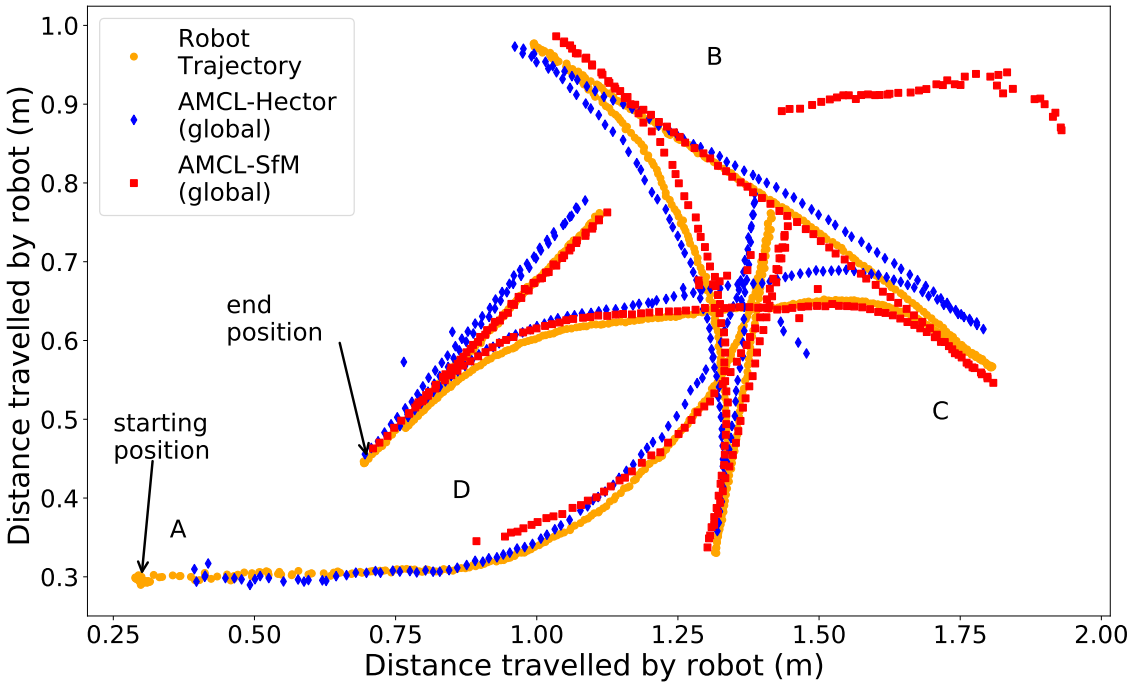


Figure 4.8: The pictorial result of AMCL-SfM plotted in RVIZ showing the 2D map created from the SfM point cloud, the position of the robot and the current sensor readings from the 2D LiDAR labelled in the figure.

Next, the difference between results for global and local AMCL was considered. As expected, the covariance of the particles was higher for global AMCL than for local AMCL since the particles begin spread across the map. For Figure 4.9i the error between the global AMCL-Hector and the reference trajectory started at $1.25\,\mathrm{m}$, with an initial guess for the robot location at the centre of the map. The current position was recalculated and the trajectory began to converge to the reference trajectory. After seven time-steps the error approached a value similar to the error seen in the local localisation approaches. Similarly, AMCL-SfM also began with a large error of $1.1\,\mathrm{m}$ (see Figure 4.10i), which increased as the robot began to move, before converging to the reference trajectory after 20 time steps. The initial location calculations can be seen to the right of label B in Figure 4.9i and Figure 4.9ii as the robot trajectory points and associated ellipses move from the top right of the figure to the bottom left where the initial starting pose was. Although the error in the robot position for AMCL-SfM converged after 20 time-steps, it took 80 time-steps for the covariance to converge (compared to 30 time-steps for AMCL-Hector). Due to the large errors in the initial time-steps, global localisation will be discounted, since it is not suitable for the bridge bearing environment.

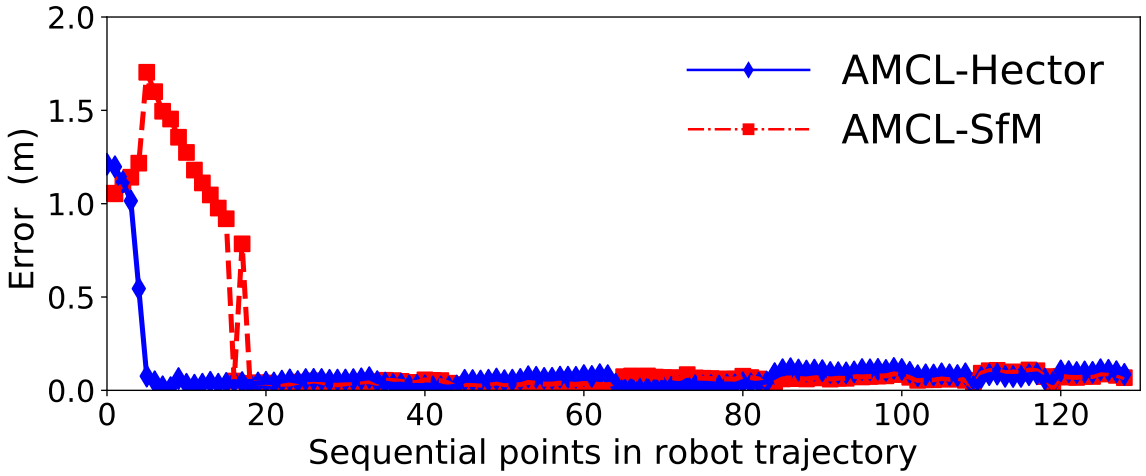


(i): Trajectory of the robot calculated using global AMCL and ellipses to show the uncertainty at the current pose. Only the ellipses after the pose begins to converge are shown for readability.

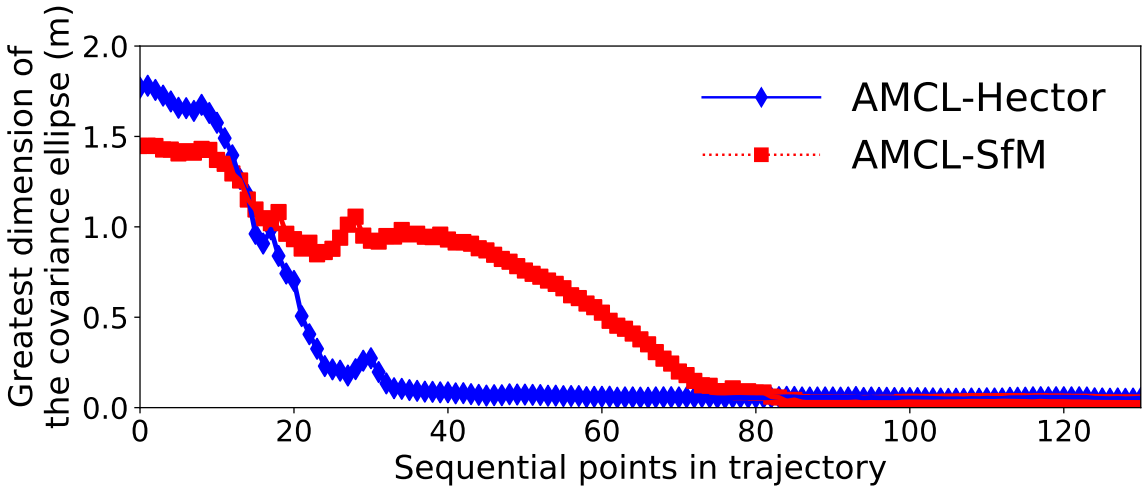


(ii): Trajectory of the robot calculated using global AMCL

Figure 4.9: A comparison of trajectories from AMCL-Hector and AMCL-SfM to the reference robot trajectory for the global initialisation of AMCL. Coloured ellipses in Figure 4.9i show the level of the uncertainty of the particles at the current position. Only the ellipses from The trajectory without the ellipses is shown in 4.9ii.



(i): Discrepancy between trajectories calculated using AMCL-SfM and AMCL-Hector and the reference robot trajectory.



(ii): Greatest dimension of the covariance ellipse for the AMCL particles over time.

Figure 4.10: The error calculated between the reference trajectory of the robot and the trajectories calculated by both AMCL-Hector and AMCL-SfM when global localisation is implemented (4.10i) and the covariance of the AMCL particles over time, as calculated from the greatest dimension of the covariance ellipses seen in Figure 4.9 (4.10ii)

4.6 Evaluation of ORB SLAM

The monocular and stereo ORB SLAM implementations were also tested in the laboratory environment described in Section 4.5. Data was collected in the same manner as described in Section 4.5, with the robot using a pre-determined set of velocity commands. Alongside the 2D LiDAR data, monocular RGB images were also collected from the Raspberry Pi Camera at a resolution of $1280\,\text{pixel} \times 960\,\text{pixel}$ at $30\,\text{fps}$ and stereo footage was collected using the ZED stereo camera at a resolution $1280\,\text{pixel} \times 720\,\text{pixel}$ at $30\,\text{fps}$. See Chapter 3 for further description of the sensors. The motion commands were chosen to complement ORB SLAM, which requires rotational motion to be combined with translational (Mur-Artal et al. (2015) and Mur-Artal and Tardos (2017)). As a result, no in-place rotation of the robot was performed and all rotations were performed in arcs and changes in direction were performed by reversing and turning at the same time before moving in the new direction. The results for the ORB SLAM in the laboratory environment are shown in Figure 4.11 and Figure 4.12.

Both implementations of ORB SLAM had a greater discrepancy to the reference trajectory of the robot than Hector SLAM. In Figure 4.11, the error in the scale of the path of the robot for the stereo implementation is comparable to the monocular implementation, even though no scale was calculated for the monocular implementation. There are also greater errors in the shape of the trajectory for the stereo implementation for the same number of frames per second as the monocular approach, with gaps occurring in the motion path due to loss of tracking of key points by ORB SLAM. This discrepancy was likely due to the greater camera resolution used in the monocular implementation for the same number of frames per second so that more ORB features can be reliably tracked over time.

To further evaluate the effect of camera resolution on the results of ORB SLAM, the stereo implementation was tested for three different camera resolutions on a new set of input commands. This set of motion commands was repeated three times, once for each of the three available resolutions of the ZED stereo camera, and the resulting robot trajectories are shown in Figure 4.12.

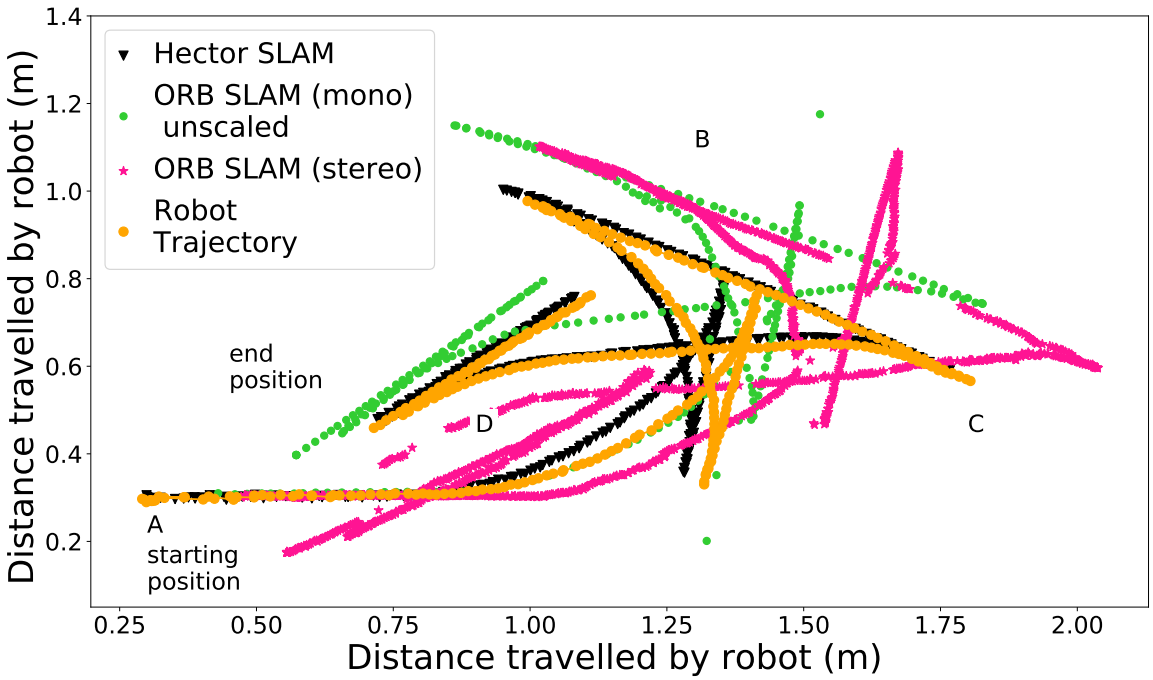


Figure 4.11: A comparison of the trajectories calculated from the monocular and stereo implementations of ORB SLAM, Hector SLAM and the robot odometry. The stereo implementation used data recorded at a resolution of $1280\,\text{pixel} \times 720\,\text{pixel}$, $30\,\text{fps}$.

Figure 4.12 shows that the resulting trajectories for all camera frame rates and resolutions were incorrect when compared to the reference robot trajectory. The lowest resolution of the ZED stereo camera was $640\,\text{pixel} \times 480\,\text{pixel}$ at $100\,\text{fps}$. For this camera resolution, the initial part of the robot motion was recovered, but was lost at the region labelled C in Figure 4.12. It is possible that there was a change in the lighting conditions that caused the loss in tracking of the ORB features at C in Figure 4.12, since before the robot position was lost the higher frame rate of $60\,\text{fps}$, meant there were more points in the motion path from the implementation of ORB SLAM at a resolution of $640\,\text{pixel} \times 480\,\text{pixel}$. Another reason for the loss in recorded trajectory may be that not enough ORB features could not be extracted in this part of the environment for the given camera resolution. Future work may consider visual SLAM methods that use different image features, such as straight edges (e.g., Maity et al. (2018)), which are more common in an urban environment.

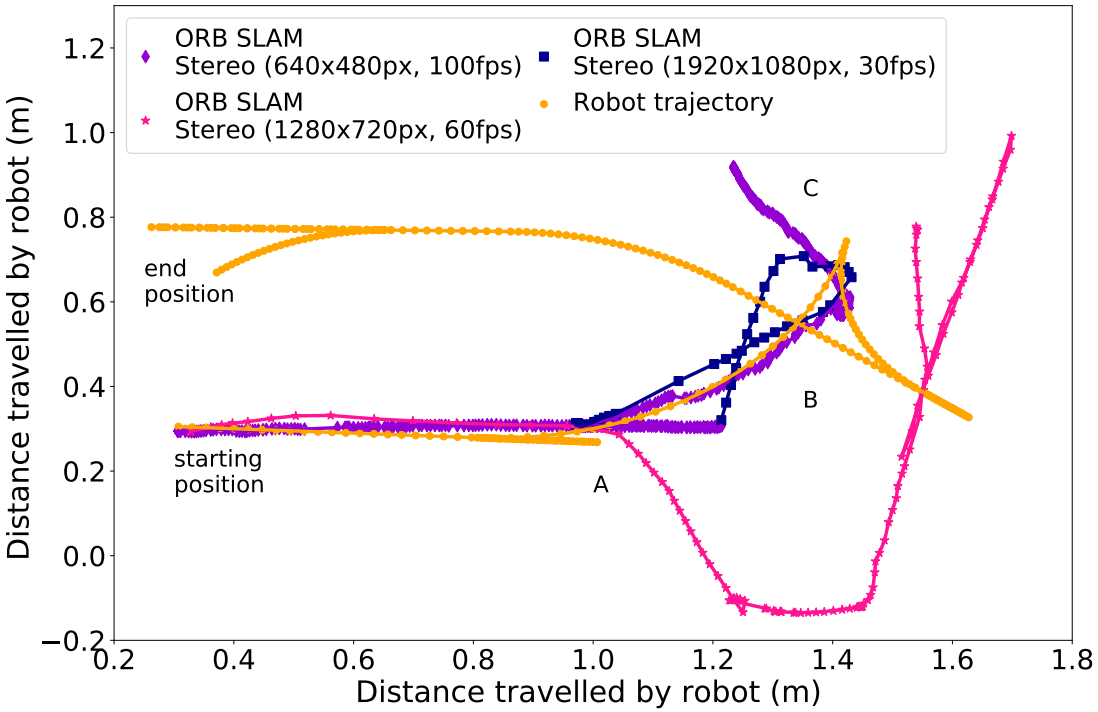


Figure 4.12: A comparison of robot trajectories calculated using ORB SLAM for three different resolutions and frame-rates of the ZED stereo camera. These robot trajectories are compared to the reference robot trajectory, plotted as orange points.

Using the full stereo camera resolution of $1920\,\text{pixel} \times 1080\,\text{pixel}$ at $30\,\text{fps}$ gave the motion with the greatest error (compared to the reference trajectory) with the position of the robot being lost between B and C in Figure 4.12. These errors were caused at the lower frame rate of $30\,\text{fps}$, where ORB features were lost between frames and tracking of these features failed.

The robot motion was also repeated for a camera resolution of $1280\,\text{pixel} \times 720\,\text{pixel}$ at $60\,\text{fps}$. The increase of frames per second from $30\,\text{fps}$ to $60\,\text{fps}$ meant that the full trajectory of the robot was recorded. Dropped frames occasionally caused problem for the datasets collected at $1280\,\text{pixel} \times 720\,\text{pixel}$ and $60\,\text{fps}$, although the main problem was due to an error in the orientation calculated by ORB-SLAM. However, there was a problem with the scale of the robot trajectory calculated by ORB SLAM, and the motion of the robot was falsely recorded (see the incorrect change in direction at location A for the pink points

in Figure 4.12). The main cause of this error was that the ORB SLAM algorithm requires a bundle adjustment stage to correct errors in the map.

In ORB SLAM, bundle adjustment is calculated on the full robot trajectory periodically during mapping and localisation and is triggered by loop closure (Mur-Artal et al., 2015; Mur-Artal and Tardos, 2017). However, due to the confined space in the test enclosure it was difficult to perform a robot motion that allowed loop closure. In addition, the test environment was traversed only once and this appears not to have been enough time to perform multiple loop closures to correctly adjust scale and orientation of the ORB-SLAM trajectories. Overall, ORB SLAM was found to be sensitive to lighting changes and to be unsuitable for environments where small small trajectories are required, and hence was not tested the real bridge environment.

4.7 Summary of findings in the laboratory environment

Overall, the experimental results from the laboratory environment showed that the trajectories calculated by local AMCL (for all map types) and Hector SLAM gave an error within $4\,\mathrm{cm}$ of the reference trajectory.

In the real bridge environment there is danger that the robot could fall from height. Hence, a maximum error limit was required to evaluate the different methods with the aim of reducing the risk of failure, especially since the maps in the bridge environment were more noisy than in the laboratory environment. For the purposes of this work, this maximum error limit was defined as $10\,\mathrm{cm}$ with respect to the reference trajectory, which was chosen as it is approximately the diameter of one of the wheels on the robot, because if a wheel was to move over the edge of bearing enclosure, it was anticipated that the robot would not be able to recover from an error greater than this value. This error bound is defined to allow comparison of the performance of the algorithms considered in this chapter in the real and laboratory environments. Next, to understand whether such an error bound is

achievable in practice, AMCL was tested in the bridge environment using the same robotic platform.

4.8 Data collection in the bridge environment

Next, AMCL and Hector SLAM were tested in the bridge environment using the same robotic platform as tested in the laboratory environment in Section 4.5. In this section, an additional map, created from 3D terrestrial LiDAR scans, was used as an alternative method for occupancy map creation. Again, all results from AMCL were compared to Hector SLAM and the reference robot trajectory, which was calculated using odometry information from the 2D LiDAR. In all cases, only the local implementation of AMCL was considered. The maps used for AMCL calculations are as follows:

1. A map created using Hector SLAM.
2. A map created using SfM data from photographs collected in the bridge bearing enclosure.
3. A map created using 3D terrestrial LiDAR (map creation process was the same as for SfM in Section 4.3, see Figure 4.1). AMCL results using this method will henceforth be referred to as AMCL-LiDAR.

The results of AMCL-SfM and AMCL-LiDAR were also considered for two different map resolutions: $5\,\mathrm{cm}$/grid-cell and $1\,\mathrm{cm}$/grid-cell. Examples of the different maps tested in the bridge environment are given in Figure 4.13.

Sensor data was collected in the same manner as described in Section 4.5, with the robot traversing a pre-set trajectory from a given set of velocity commands. The data in this section was collected in the real bridge environment with the assistance of Dr. Shan Luo.

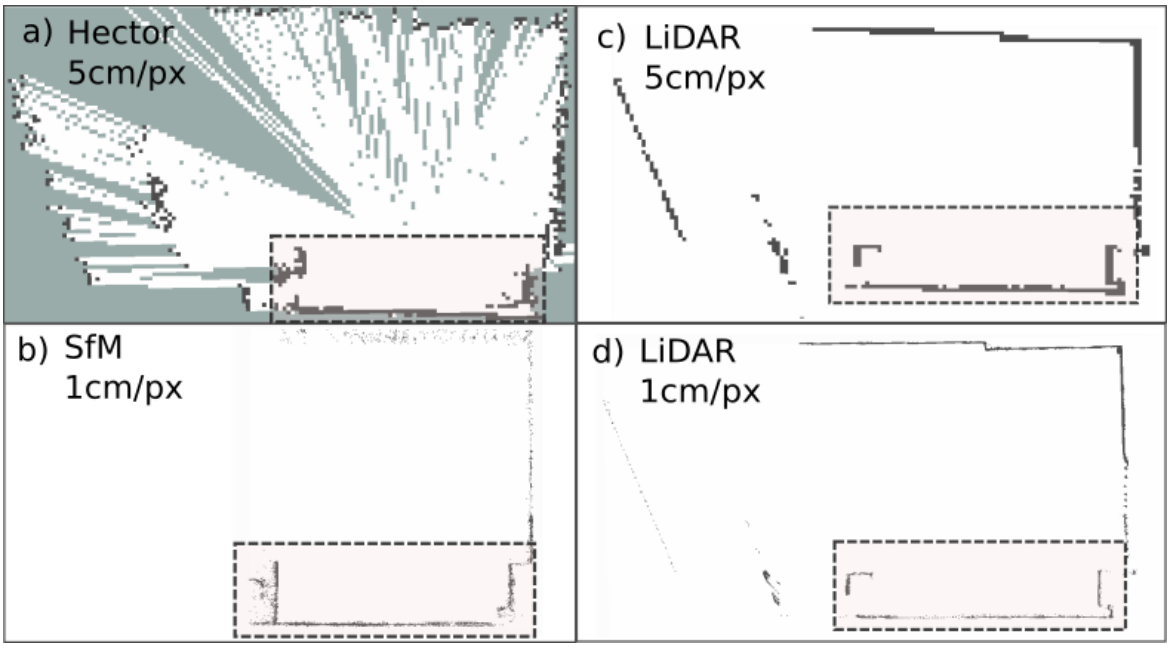
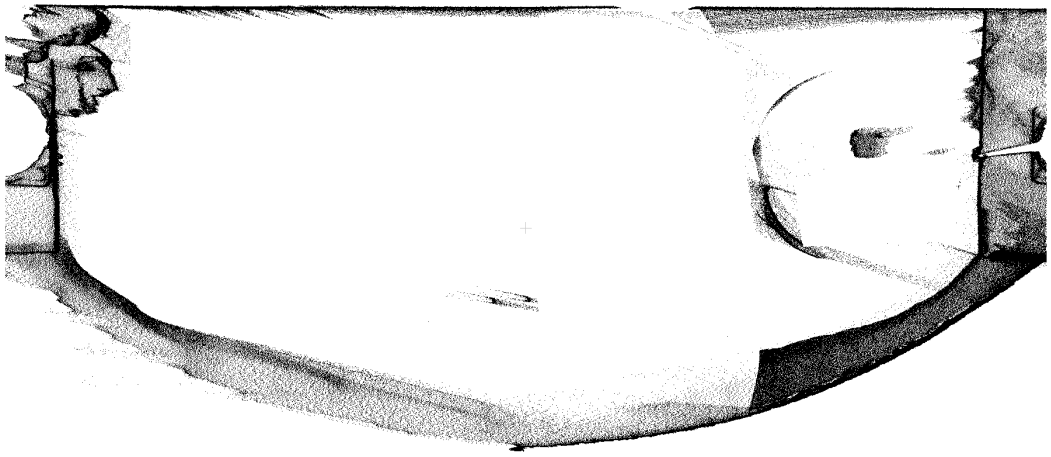


Figure 4.13: a) 2D occupancy map created using Hector SLAM at a resolution of $5\,\mathrm{cm}$/grid-cell. b) 2D occupancy map created using SfM $1\,\mathrm{cm}$/grid-cell. c) 2D occupancy map at $5\,\mathrm{cm}$/grid-cell created from 3D terrestrial LiDAR point cloud. d) 2D occupancy map at $1\,\mathrm{cm}$/grid-cell created from 3D terrestrial LiDAR point cloud. The black pixels show areas of occupied space, the white pixels show areas of unoccupied space and the dark-grey pixels (in a) show regions where it is unknown if the space is occupied or unoccupied.

4.9 Validation of SfM data against 3D terrestrial LiDAR data

In the bearing enclosure, point cloud data is also available from a 3D terrestrial laser scanner, the RIEGL VZ-1000. Detailed scans of the bearing enclosure were taken from three different perspectives and the clouds were registered using targets with a registration error of less than $3\,\mathrm{mm}$. CloudCompare was then used to compare the SfM and 3D terrestrial LiDAR point clouds. The SfM and terrestrial laser scan point clouds were aligned by selecting common features that are visible and easy to select by eye. Iterative Closest Point (ICP) alignment was used to further align the point clouds. A cloud-cloud distance calculation allows the comparison of a point cloud to a reference point cloud, where the distance of points can be calculated relative to associated points in the reference cloud. A scalar field was generated showing the number of points and the distance of those points compared to the points in the reference cloud. $90\,\%$ of points from the SfM point cloud are within a distance of $4\,\mathrm{cm}$ from the terrestrial 3D LiDAR point cloud and $60\,\%$ of points are within $1\,\mathrm{cm}$ of the terrestrial 3D LiDAR point cloud. This result gave confidence that the SfM map was accurate and suitable for map creation. Furthermore, using two sets of maps gives an indication of whether localisation will be affected by scale. A visual comparison of the difference between SfM and terrestrial LiDAR data for the bridge bearing enclosure is shown in Figure 4.14.



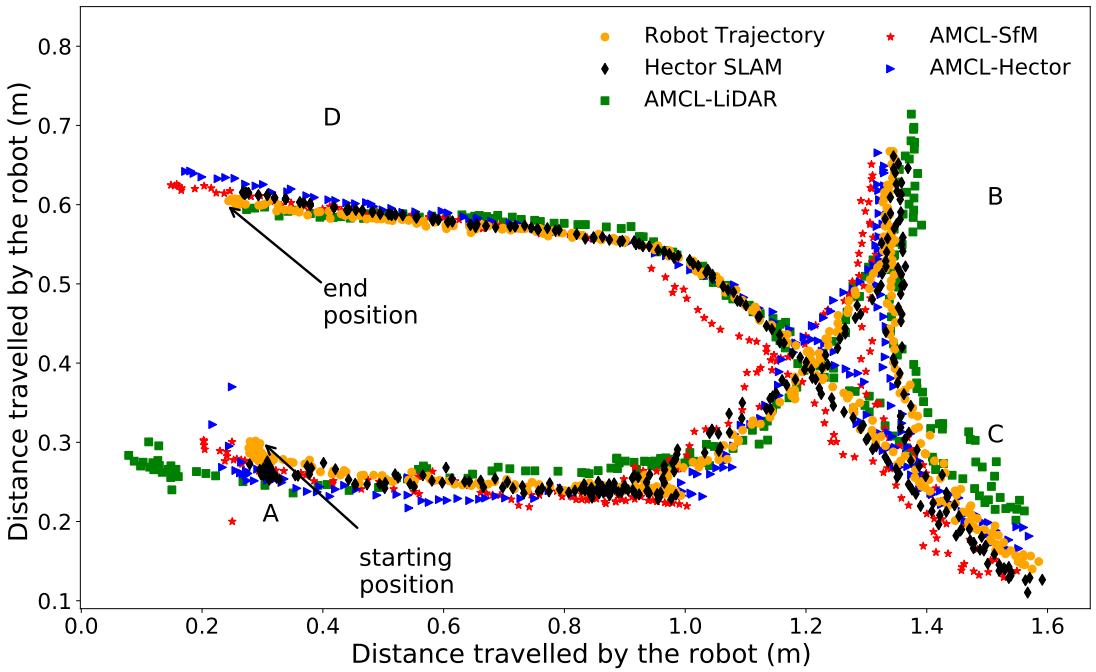
(i): 3D terrestrial LiDAR point cloud of the bearing enclosure.

(ii): 3D SfM point cloud of the bearing enclosure.

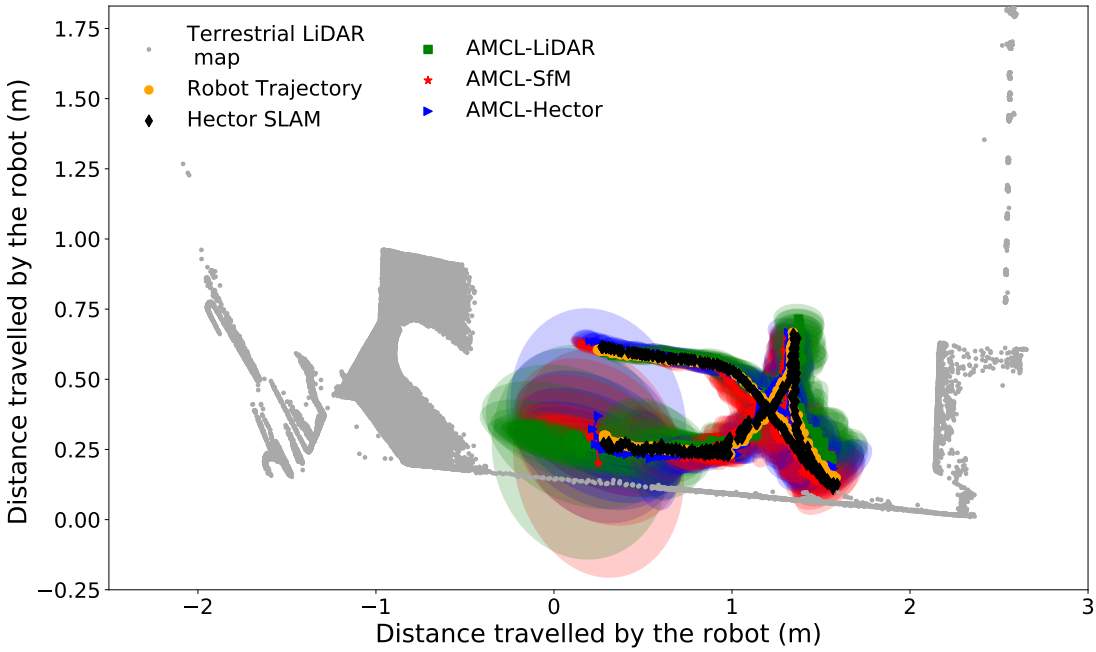
Figure 4.14: Exemplar terrestrial LiDAR data (4.14i) and sparse SfM data (4.14ii) for the bearing enclosure. The dots that make up each of the image represent a point in 3D Cartesian space. The terrestrial LiDAR point cloud in Figure 4.14i is denser than the SfM point cloud in Figure 4.14ii and has less noise (see the distribution of points around the outside of the point cloud in Figure 4.14ii). Both images are cropped to a region of the bridge that includes only the bearing enclosure.

4.10 Results and discussion for the bridge environment

Figure 4.15 shows a comparison between the calculated robot trajectories and the reference trajectory. The associated error and covariance values are also plotted in Figure 4.16. The areas labelled B and C in Figure 4.15i show the greatest discrepancies between the AMCL trajectories and the reference trajectory. Overall, the associated trajectory errors in the real bridge environment are greater than the laboratory results, with values varying between $2\,\mathrm{cm}$ and $10\,\mathrm{cm}$ (see Figure 4.16i). The maximum value for covariance is also greater than in the laboratory environment, with values between $20\,\mathrm{cm}$ and $1\,\mathrm{m}$ (see Figure 4.16ii), which covers a large portion of the bearing enclosure in Figure 4.15ii. Again, the largest error and covariance occurs after initialisation of the particles, with the largest error value of $9\,\mathrm{cm}$ produced by AMCL-SfM, and the largest covariance value of $1\,\mathrm{m}$, produced by all methods (see Figure 4.16ii). This error is slightly greater than in the laboratory environment due to greater uncertainty in the initial position given to the robot, but this error converges to within $3\,\mathrm{cm}$ of the reference trajectory after three time-steps. Errors in the initial position guess occur due to the difficulty of measuring the location by hand in the bridge environment. In future work, some method of accurately determining the starting position of the robotic platform in the bearing enclosure could be developed. One potential solution may be to use a charging dock in a fixed location or to combine information from both Hector SLAM and AMCL to improve the initial position estimate.

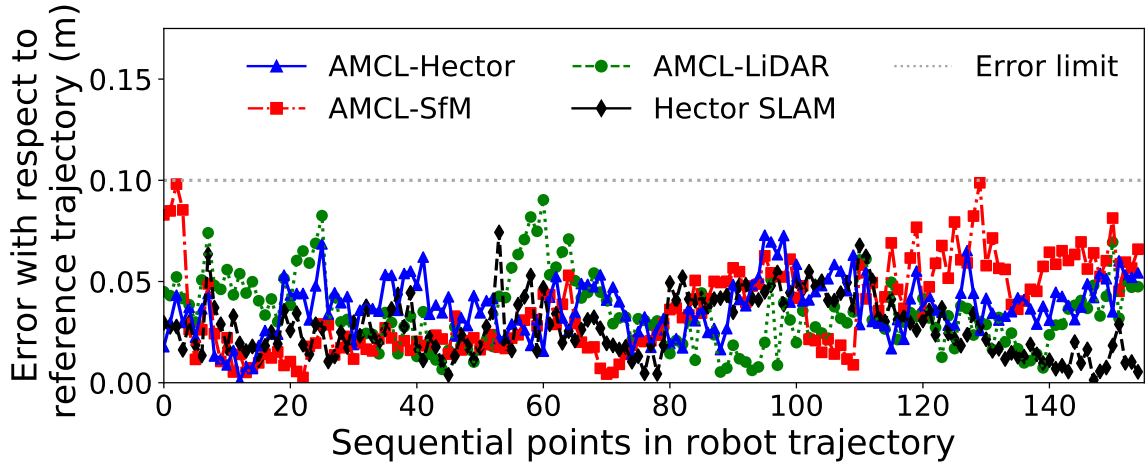


(i): A comparison of the trajectories calculated for the robot in the bearing enclosure.

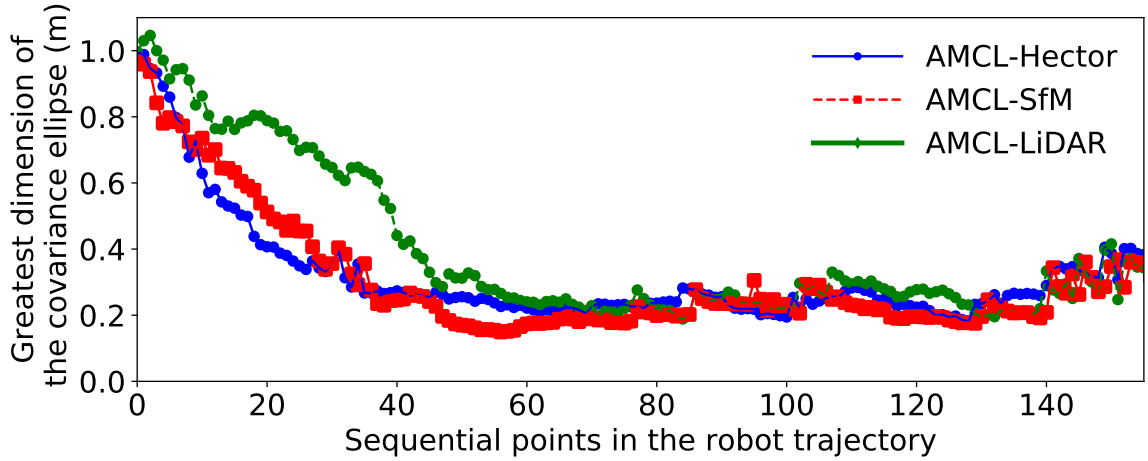


(ii): The trajectories of the robot with covariance ellipses from AMCL and terrestrial LiDAR map for scale

Figure 4.15: The trajectories calculated using AMCL (local) in the bridge environment for three different maps created using either SfM, 3D terrestrial LiDAR data or Hector SLAM (4.15i). In 4.15ii, the same trajectories as in 4.15i are shown, but are overlaid with ellipses representing the covariance of the pose at each step is shown in 4.15ii. Map points from the terrestrial 3D LiDAR scan are shown in 4.15ii to give a sense of scale of the trajectories relative to the bearing enclosure.



(i): Comparing the error between the reference trajectory of the robot and the trajectories calculated by both local AMCL-Hector, AMCL-SfM and AMCL-LiDAR for each point in the reference robot trajectory. The error limit of $10\,\mathrm{cm}$ between the calculated trajectory and the reference trajectory is also marked.



(ii): Comparing the covariance of the AMCL particles over the duration of the robot trajectory, as calculated from the greatest dimension of the covariance ellipses seen in Figure 4.15ii

Figure 4.16: The error calculated between the reference trajectory of the robot and the trajectories calculated by AMCL-Hector, AMCL-LiDAR and AMCL-SfM in the bridge environment, where the error was calculated for each point in the reference robot trajectory (Figure 4.16i). The covariance of the AMCL particles over time for the same approaches as in 4.16i, with the covariance value being calculated from the greatest dimension of the covariance ellipses seen in Figure 4.15ii.

The initial error for AMCL-LiDAR is lower than AMCL-SfM, but slightly greater than AMCL-Hector and Hector SLAM ($5\,\mathrm{cm}$, $2\,\mathrm{cm}$ and $4\,\mathrm{cm}$ respectively in Figure 4.16i). In addition, the covariance for AMCL-LiDAR takes the longest to converge (see Figure 4.16ii). This initial uncertainty is visible in Figure 4.17a, where the particles are spread over the majority of the bearing enclosure and in Figure 4.17b appear to split into two groups of particles representing the robot position. However, by Figure 4.17c the covariance has decreased to $20\,\mathrm{cm}$ (i.e., between the areas labelled A and B in Figures 4.15 and 4.16). One potential reason for the greater uncertainty for AMCL-LiDAR is that the available detail in the original point cloud is greater than for the 2D LiDAR. Therefore, some features have been included in the map that are not representative of what was visible for the 2D LiDAR, and as a result, the AMCL particles group in the true position and another position, but as the robot begins to move, it becomes apparent which is the correct position and the uncertainty and error decreases.

An increase in error is observed at the points labelled B and C in Figure 4.15i (time-steps 60 and 100 in Figure 4.16) as the robot turns towards the less featured regions of the map (i.e., regions where there is not much bridge structure: top left region of figures a-d in Figure 4.13). At these points, potential features in the map are at a distance close to the maximum sensor range of the 2D LiDAR ($6\,\mathrm{m}$). When comparing the maps in Figure 4.13a-c, it is apparent that the sensor data from the 2D LiDAR can reach some of regions of the map but not others, which contributes to a greater error in the robot position. Although the increase in errors are similar for all methods, the error for AMCL-SfM does not recover as well as the other methods towards the end of the trajectory at point D. In addition, the error at D for AMCL-SfM is higher when the map resolution was increased to $1\,\mathrm{cm}$/grid-cell (see Figure 4.18ii). The map for AMCL-SfM in these regions is noisier, but also the sensor data does not overlap well, suggesting that the point cloud scaling is worse in these regions. AMCL-SfM has the least detail in this region since the image features used to create the SfM point cloud are far away are not as clear as nearby features.

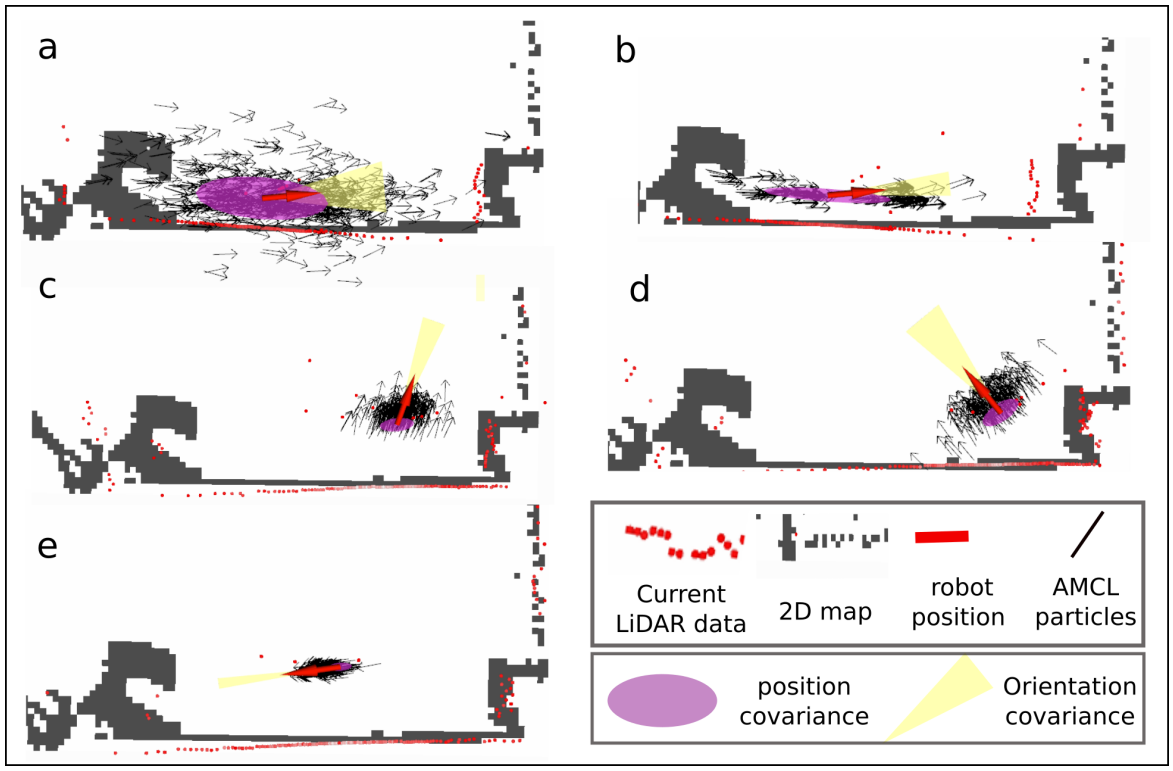
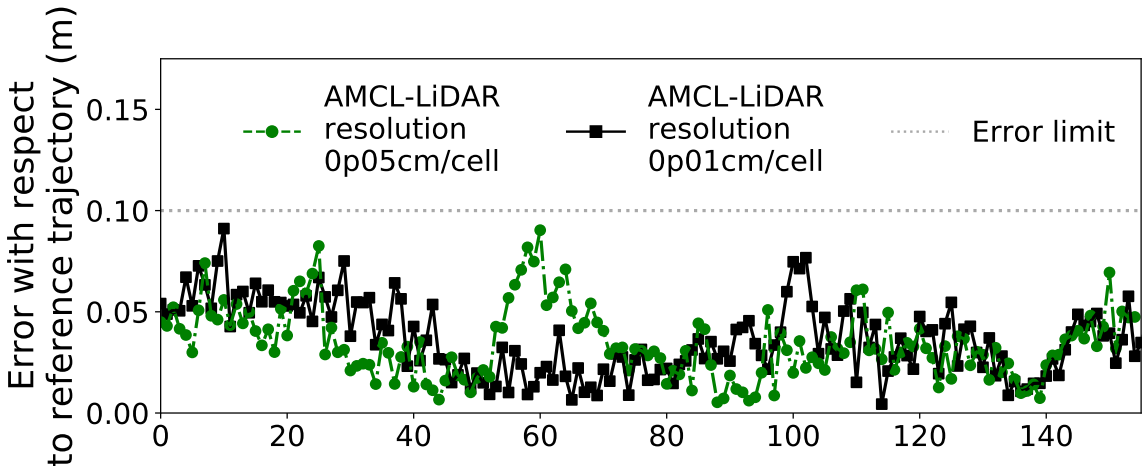
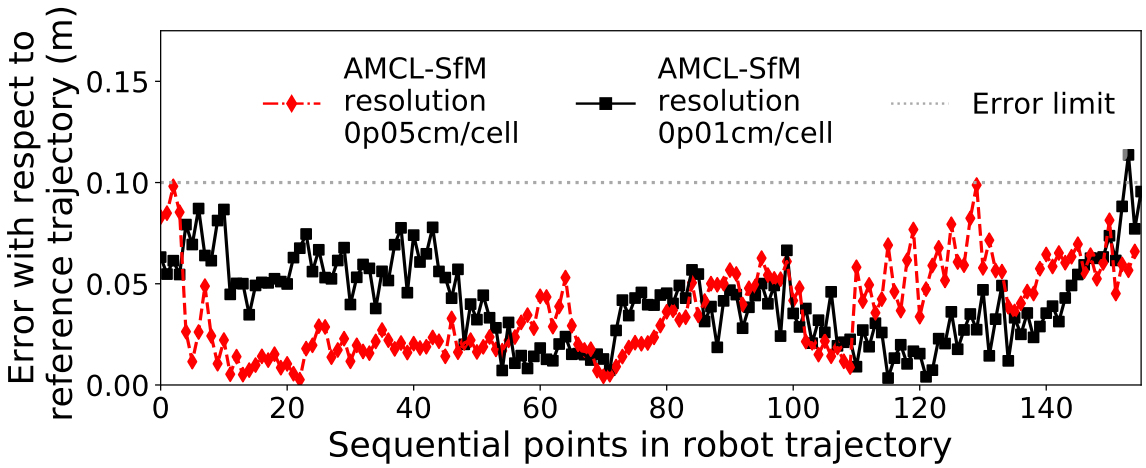


Figure 4.17: The pictorial representation of AMCL-LiDAR plotted in RVIZ (ROS visualisation software) showing the robot position within map created using 3D terrestrial LiDAR data (black pixels). The covariance of the robot pose and robot orientation varies as the robot moves around the map. a) shows the robot position shortly after initialisation of AMCL-LiDAR with particles widespread around the current guess for robot position; b) shows the AMCL particles in two small groups, but with misalignment in the current LiDAR data and the map there is uncertainty in the current robot position; c) shows the particles converge to one position and an improvement in the alignment of the current LiDAR data and the map; d) shows an increase in the covariance for robot position and orientation as the robot turns; e) is towards the end of the robot motion and shows that AMCL-LiDAR converges to the current robot position.



(i): A comparison of errors with respect to the reference trajectory for AMCL-LiDAR for two map resolutions



(ii): A comparison of errors with respect to the reference trajectory for AMCL-SfM for two map resolutions

Figure 4.18: The error calculated between the reference trajectory of the robot and the trajectories calculated by AMCL-LiDAR and AMCL-SfM in the bridge environment, where the error was calculated for each point in the reference robot trajectory. Map resolutions of 1 cm/grid-cell and 5 cm/grid-cell are compared for both methods.

Otherwise, there is little difference in the errors produced from AMCL-LiDAR and AMCL-SfM when increasing the map resolution to $1\,\mathrm{cm}$/pixel (see Figure 4.18). For AMCL-SfM, the error converges quicker for the lower map resolution of $5\,\mathrm{cm}$/pixel, but the error for both resolutions of the AMCL-LiDAR maps match well. The map resolution of $5\,\mathrm{cm}$/pixel is used for creating the maps for localisation in the bridge environment as it appears to allow some smoothing of noisy regions, but provides adequate detail for localisation.

4.11 A combined approach for localisation

For the majority of the trajectories recorded in Figure 4.15 and Figure 4.16 the error with respect to the reference robot trajectory is equal to or below the desired bound of $10\,\mathrm{cm}$ (marked in Figure 4.16). However, for AMCL-SfM the error becomes greater than this bound. The method with the lowest trajectory error is Hector SLAM, and the associated error is below this threshold at all times. The output from Hector SLAM was combined with a AMCL-SfM by taking the average of the trajectory coordinates from Hector SLAM and AMCL-SfM and plotting the error of this new trajectory with respect to the reference trajectory (Figure 4.19). By combining the two approaches, it is observed that the averaged trajectory maintains a value below $6\,\mathrm{cm}$ for all time. In addition, for the greatest error produced by AMCL-SfM ($10\,\mathrm{cm}$), the equivalent error for the averaged trajectory is $4\,\mathrm{cm}$ and when Hector SLAM spikes to $6\,\mathrm{cm}$ at time-step 50 in Figure 4.19 the error for the averaged trajectory remains below $4\,\mathrm{cm}$. This solution is considered as a basic approach to allow some combination of the benefits of the individual methods (namely that AMCL allows the use of pre-existing map, but Hector SLAM is more accurate) and if one of the individual methods fail, this solution provides redundancy to reduce the risk of failure of the entire system. However, since the completion of this work, algorithms that allow SLAM in previously mapped environments have been developed (e.g., Google Cartographer), although these approaches do not take advantage of existing inspection data. It is recommended for future work that the AMCL approach used in this work be compared to the newer

algorithms, and that methods for combining the new approaches with AMCL using inspection data be considered as a more robust method for combining SLAM and localisation in a known map.

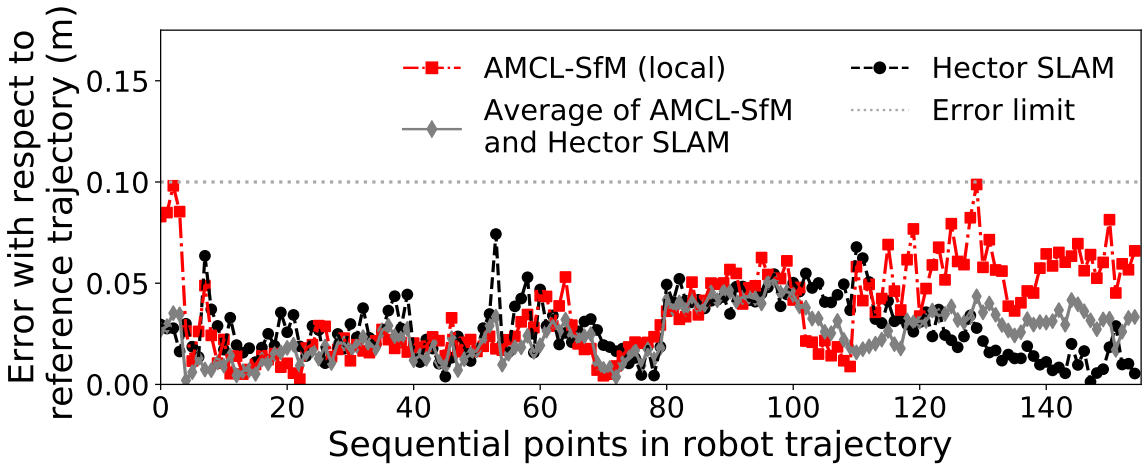


Figure 4.19: The error calculated in the bridge environment between the reference trajectory of the robot and the trajectories calculated by AMCL-SfM, Hector SLAM and a trajectory determined from the average of the two trajectories. The error limit of $10\,\mathrm{cm}$ between the calculated trajectory and the reference trajectory is also marked.

4.12 Using inspection data in a simulated environment

In addition to creating maps for localisation, there are other uses for existing inspection data for assisting the development of robotic inspection platforms. In particular, there may be advantages to using the existing data to aid the development of a robotic platform in a simulation environment before performing testing in the real environment. However, validation is required to understand how realistic the simulation environment is compared to the real environment. This section explores whether the 3D point clouds collected for inspection applications (but also used for localisation maps previously in this chapter) can be used to create a simulation environment that complements the testing that can be performed in a laboratory environment.

In this section, the simulation software known as Gazebo (Koenig and Howard, 2004)[6] was used to simulate the robotic platform that was used for testing localisation and mapping algorithms in this chapter. Gazebo is a popular simulation tool and has been used broadly in robotics research from robot manipulation tasks (e.g., Qian et al. (2014)) to simulation of quad-rotor UAVs (e.g., Meyer et al. (2012)). In particular, Gazebo is often chosen due to its compatibility with ROS that allows the development of systems that can then be implemented on a real robotic platform (e.g., Agüero et al. (2015)). There are some notable applications of using Gazebo with Digital Elevation Maps to simulate outdoor environments with difficult terrain (e.g., Neves et al. (2015) and Portugal et al. (2015)). Digital Elevation Models (DEMs) are 3D models of a surface terrain and are typically collected through remote-sensing approaches, such as radar satellites (Z. Li et al., 2004), but can also be created from point cloud data such as those created by SfM or terrestrial LiDAR (Westoby et al., 2012; Z. Li et al., 2004).

At the time of writing, there are no known examples in the literature of directly using SfM or 3D terrestrial LiDAR data to mimic an existing environment in simulation. Using SfM or terrestrial LiDAR to create a simulation environment may be particularly useful for robotic inspection of infrastructure since this data is often collected, or could be collected, using cameras on-board robotic platforms such as drones (e.g, Hallermann and Morgenthal

[6] http://gazebosim.org/ (Date accessed: 01/02/2019)

(2014) and Lattanzi and Miller (2015)). In this work, the 3D point cloud data that was used to create maps for localisation in Section 4.3 was used to create an environment for simulation. In addition, the robotic platform used in this chapter in the laboratory and real bridge environment was adapted for simulation. More details on creating the robotic platform for simulation can be found in Chapter 3 (page 74).

4.13 Creating the simulation environments

In this section, the point cloud data from the 3D terrestrial LiDAR scan of the bridge bearing enclosure is used to create a simulation environment. It should be noted that the same process could also be applied to the SfM data to create a simulation environment, but only terrestrial LiDAR data is used in this chapter. However, the maps that were created for AMCL using both terrestrial LiDAR data and SfM data in Section 4.10 are used to compare AMCL for the real bridge environment with the simulation environment.

In Gazebo, there are two main considerations for the development of the simulation environment. The first consideration is that some geometry needs to be defined for the purposes of collision and physics calculation (e.g., the friction between the surface and the wheels on the robotic platform). The second consideration is for the need of a visual representation of the environment which may be useful for monitoring the simulation, but also to provide features for the navigation algorithms.

In order to create the collision and visual geometries, the 3D point cloud data (from LiDAR or SfM) needs converting into a format that is compatible with Gazebo; in general, a mesh format is required (Open Source Robotics Foundation, 2014). In addition, stereolithography format (STL) has been recommended as a mesh format for collision calculations and digital asset exchange format (DAE) for mesh visualisation, since the collision mesh does not need to be as detailed and the STL format can be used to simplify mesh geometries (Kohlbrecher, 2016; Morena et al., 2016).

The approach for converting the point clouds into a mesh has three main steps, which are also outlined in Figure 4.20. First, the input 3D point cloud can be cropped or cleaned

using point cloud manipulation software such as CloudCompare. This step is similar to the process used in Section 4.3 and can be used to reduce the size of the environment by removing points outside of the maximum range of the navigation sensors. In this section, a reduced area of the bridge bearing enclosure is used for demonstration and evaluation of the simulation environment. Next, the normals of the points in the point cloud (the vector that is perpendicular to a point in the point cloud) are computed in CloudCompare, by approximating local planes across the point cloud using a number of nearest neighbours for each point. This step is required prior to to generating a mesh.

Next, the point cloud needs converting into a mesh. This is possible using several different software, but in this work was completed using MeshLab [7]. The points of the point cloud become the vertices of the mesh, which can be joined by edges and filled in by faces to create a solid surface. Mesh reconstruction was performed using Poisson surface reconstruction (Kazhdan et al., 2006), as recommended by Kooi (2013). Since the Poisson surface reconstruction creates a 'water-tight' mesh, large faces were then removed from the mesh to reveal the bridge bearing enclosure. This process was performed automatically in MeshLab.

[7] http://www.meshlab.net/ (Date accessed: 01/02/2019)

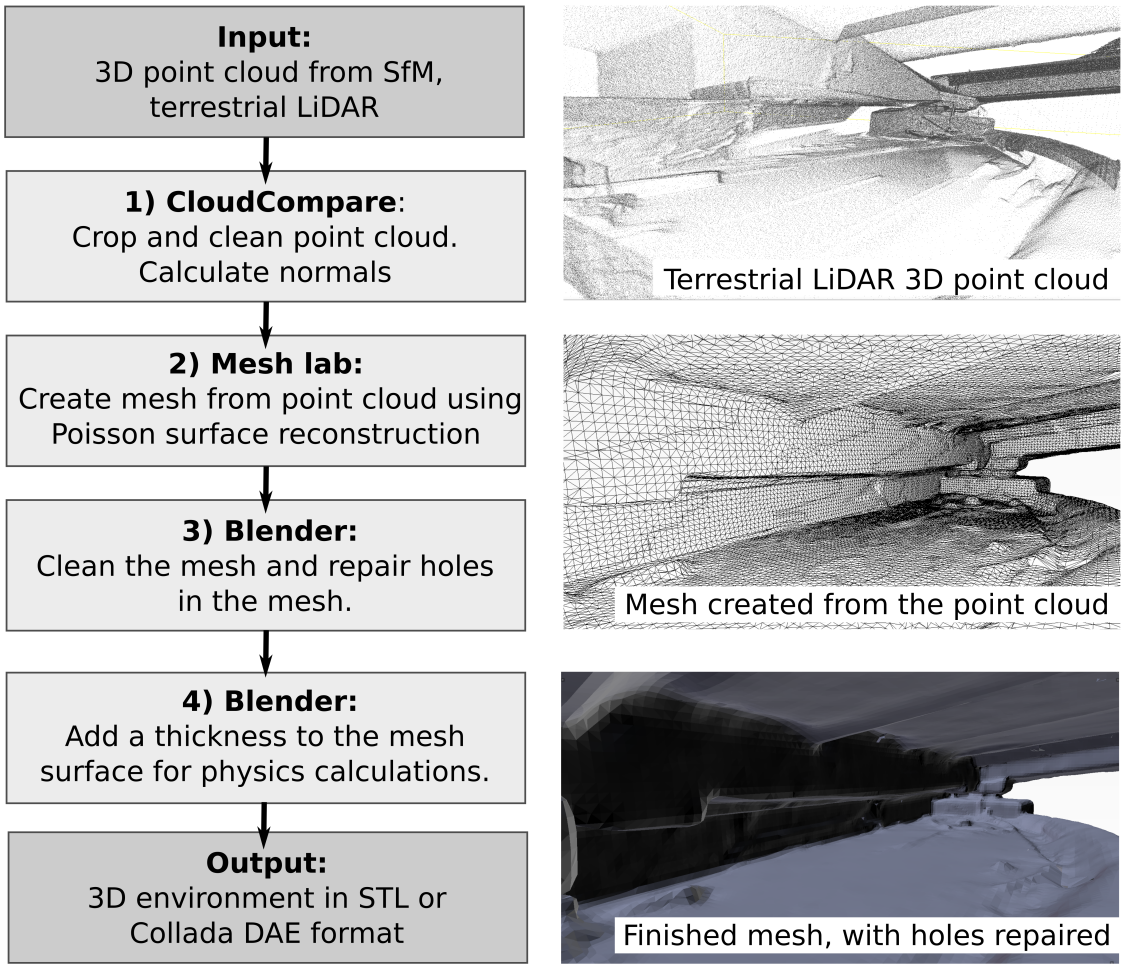


Figure 4.20: An overview of the main steps required to create the simulation environment from 3D point cloud data (left). A visual example of each of these steps (right). The input was a 3D point cloud from the 3D terrestrial LiDAR, which is converted into a mesh, cleaned and exported in DAE or STL format, for the Gazebo simulation.

The final step is to mend holes (caused by removing large faces from the mesh as Poisson surface reconstruction creates a water-tight mesh) and clean the mesh, which is required to ensure the robot has a solid surface to move on in simulation. In this chapter, the mesh cleaning step was performed in Blender[8], a mesh manipulation software. In addition to cleaning the mesh, a thickness of around two millimetres was added to the mesh in Blender, to allow better physics calculations when the mesh is imported into Gazebo (Morena et al., 2016). The mesh can then be exported from Blender as DAE and STL format as a visual mesh or a collision mesh for use in Gazebo. Figure 4.21 shows the final environment, with the simulated robot and simulated 2D LiDAR. Further description of the setup of the simulated robot is given in Chapter 3.

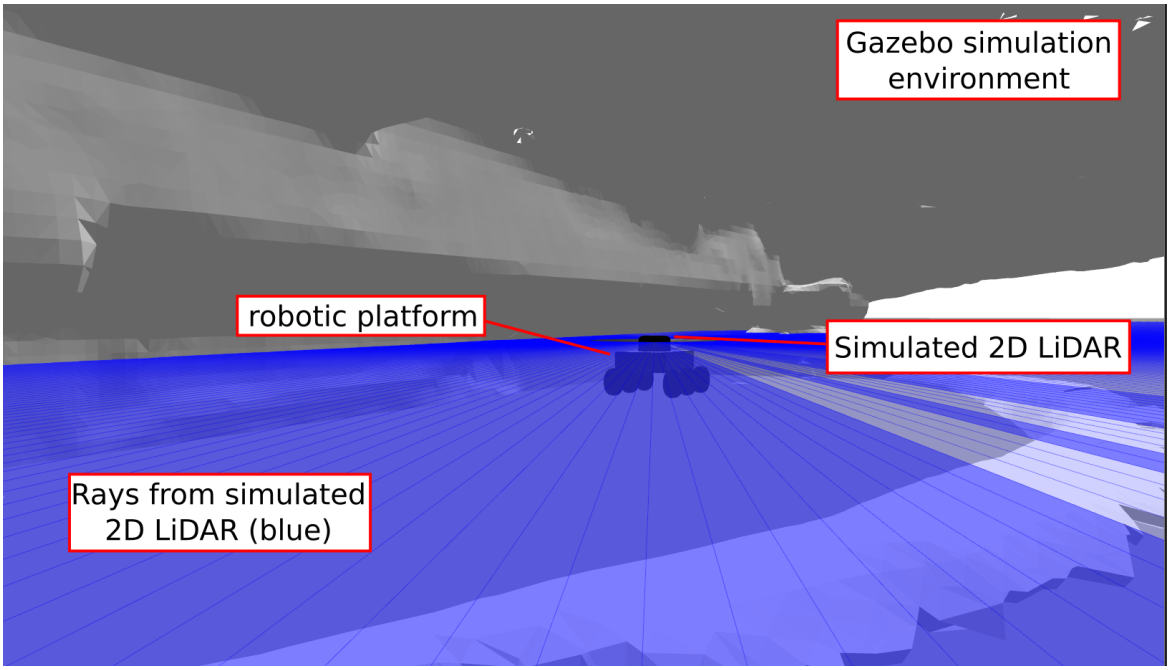
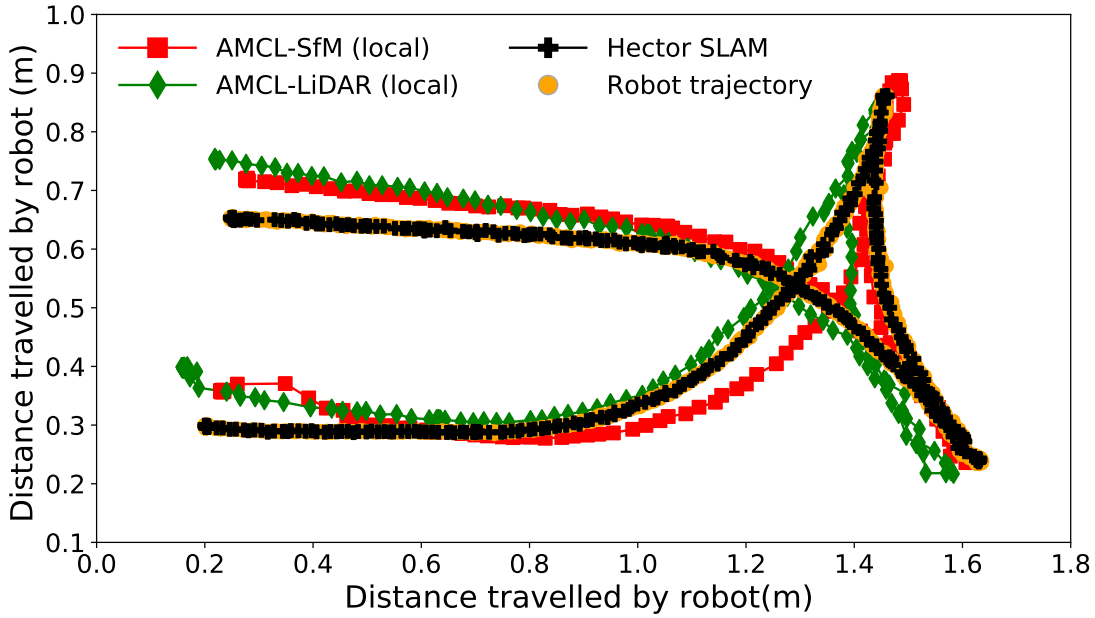


Figure 4.21: The final simulation environment with the robotic platform (labelled) and the simulated 2D LiDAR shown as the blue rays (labelled).

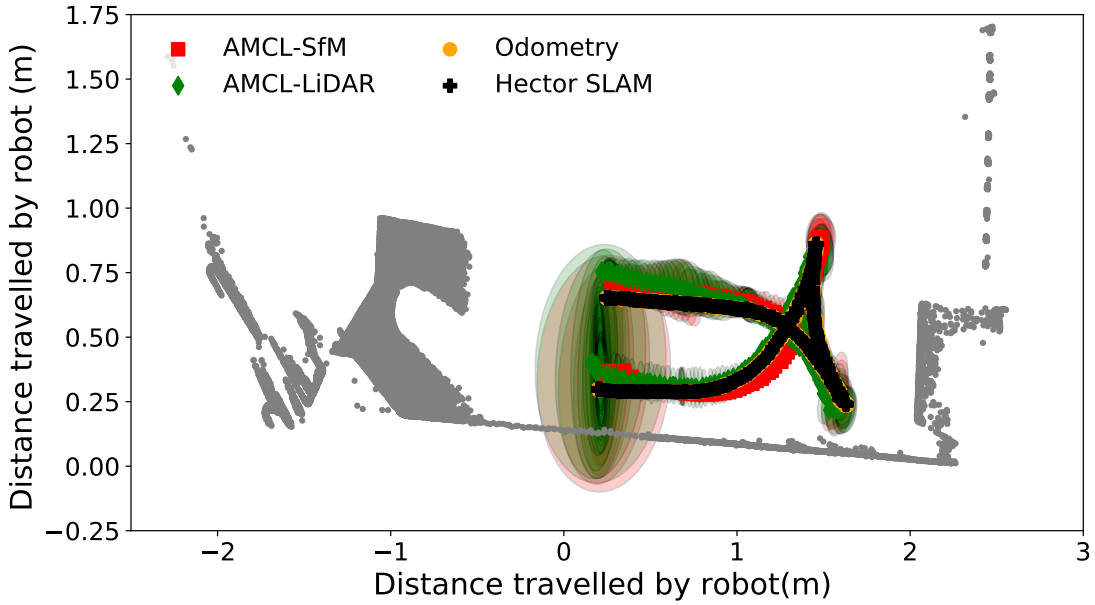
[8]https://www.blender.org/ (Date accessed: 01/02/19)

4.14 Results from the simulated environment

Next, the experiments from the real bridge environment were repeated in simulation. Motion commands were given to the simulated robotic platform, and the data from the simulated 2D LiDAR was recorded. This data was used for Hector SLAM and for AMCL using the same maps from SfM and the terrestrial LiDAR as used in the real bridge experiments in Section 4.8. The trajectory of each of these approaches was recorded and compared to the reference trajectory determined from odometry data generated using the 2D LiDAR and the Hector SLAM package. These trajectories are plotted in Figure 4.22i (with the corresponding uncertainties in Figure 4.22ii) and the errors with respect to the reference trajectory are plotted in Figure 4.24i. In order to compare to the real bridge environment, the results from Section 4.8 are replotted in this section, see Figure 4.23 and Figure 4.25.

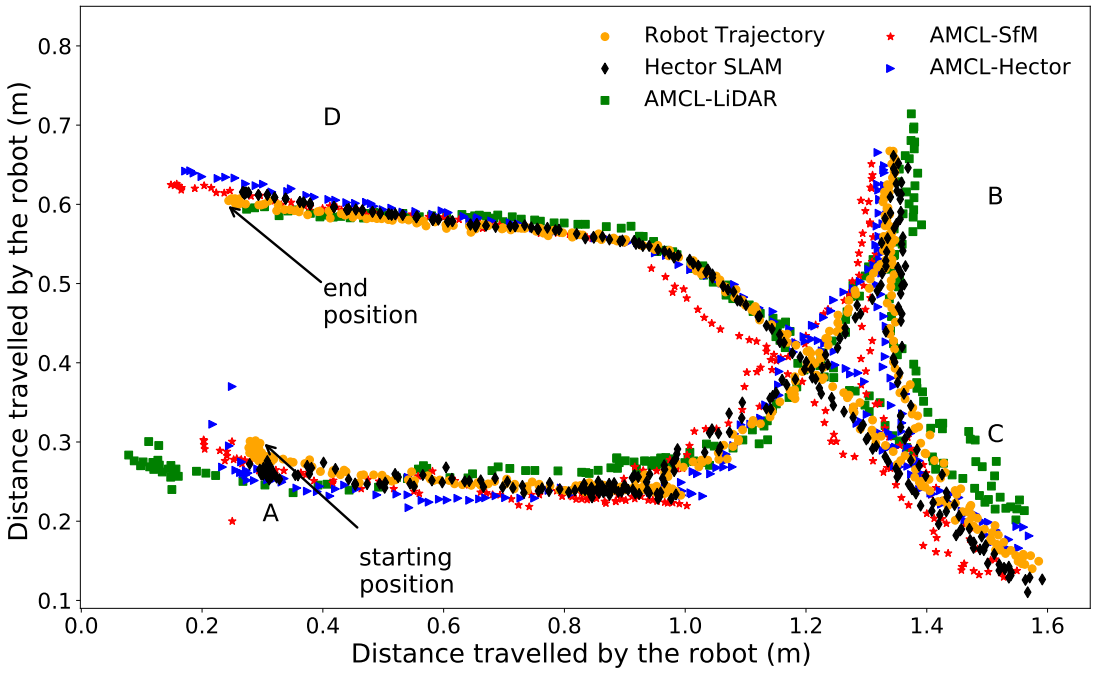


(i): The trajectories of the robot calculated from the simulation environment for different methods

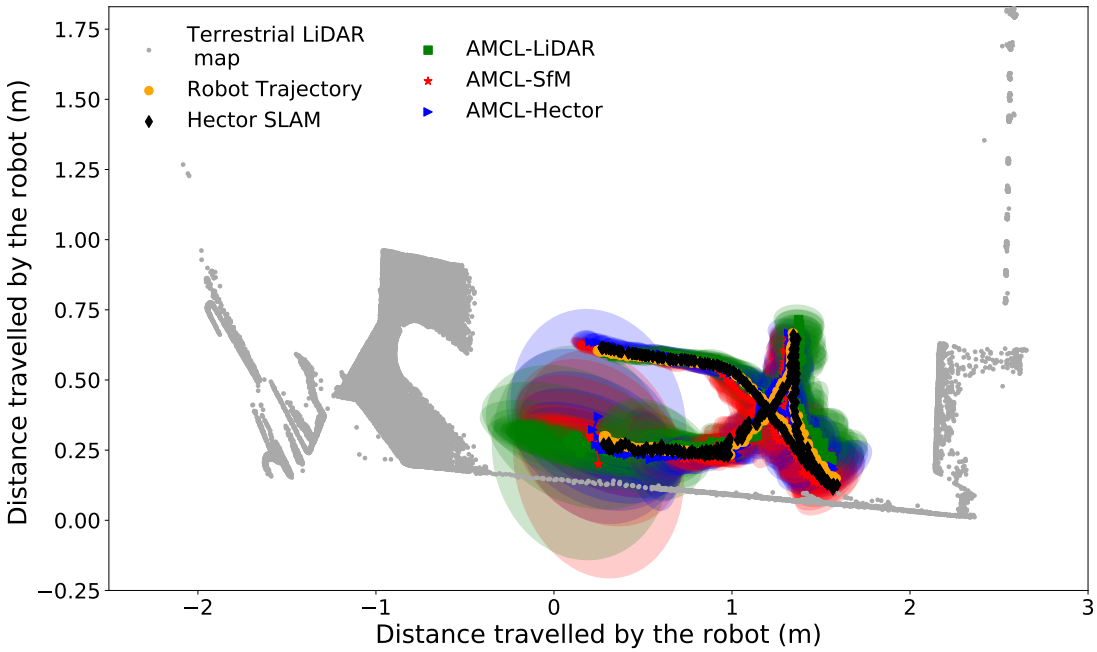


(ii): The same trajectories as in 4.22i, but with the uncertainties represented by ellipses of the AMCL particles for each point in the robot trajectory.

Figure 4.22: A comparison of the trajectories calculated using local AMCL-SfM and AMCL-LiDAR, the trajectory calculated by Hector SLAM and the reference trajectory calculated from odometry information. The ellipses that represent the covariance of the pose calculated by AMCL-Hector and AMCL-SfM are also plotted at each step in the trajectory in Figure 4.22ii.

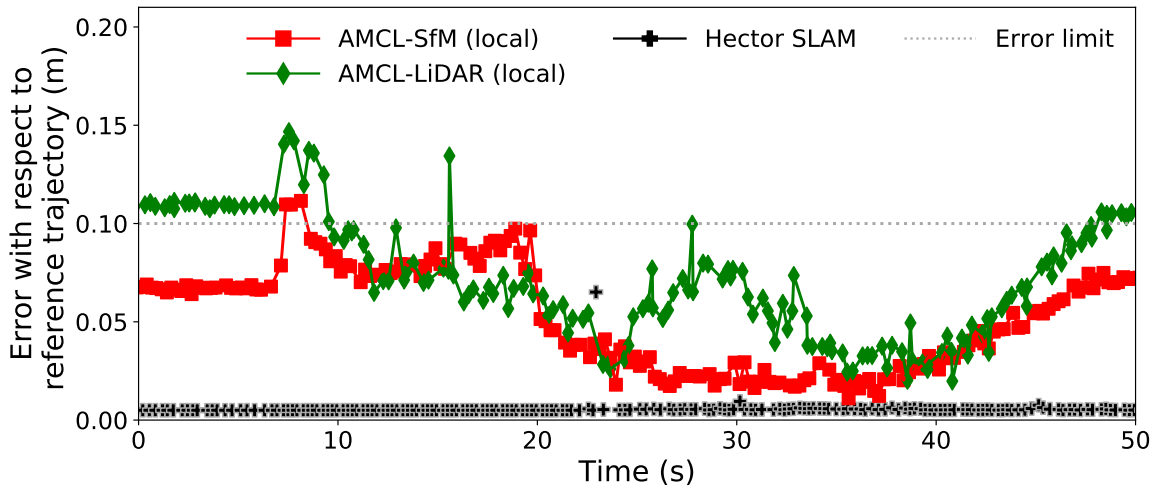


(i): A comparison of the trajectories calculated for the robot in the real bearing enclosure.

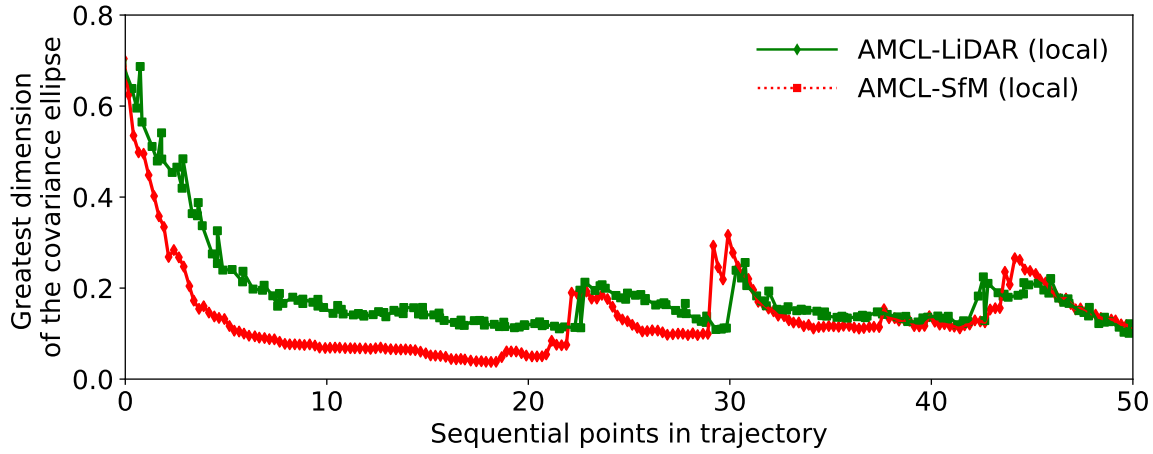


(ii): The trajectories of the robot with covariance ellipses from AMCL and terrestrial LiDAR map for scale

Figure 4.23: The trajectories calculated using AMCL (local) in the bridge environment for three different maps created using either SfM, 3D terrestrial LiDAR data or Hector SLAM (4.23i). This Figure is a duplicate of Figure 4.15, and is provided for comparison to the simulation results.

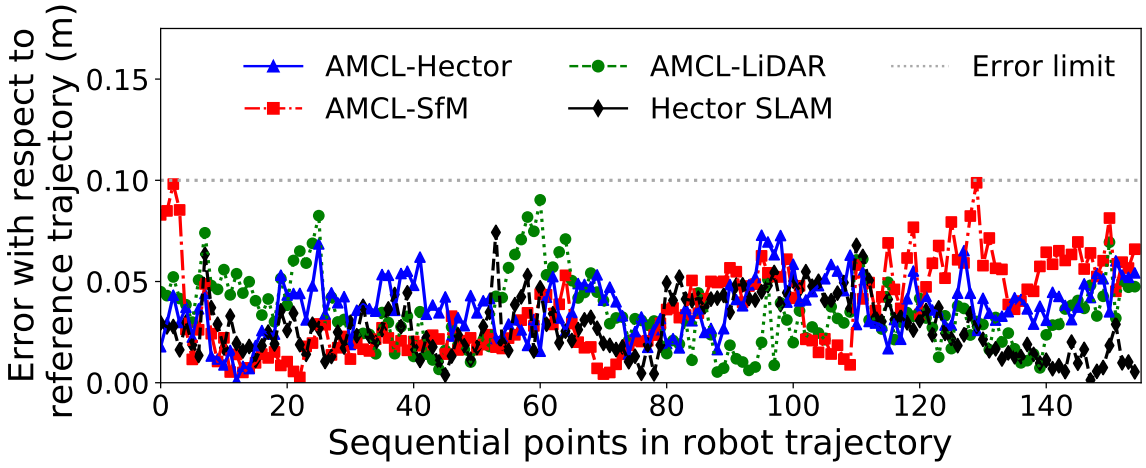


(i): Comparing the errors between the reference trajectory of the simulated robot and the trajectories calculated by both local AMCL-LiDAR and AMCL-SfM for each point in the simulated reference robot trajectory.

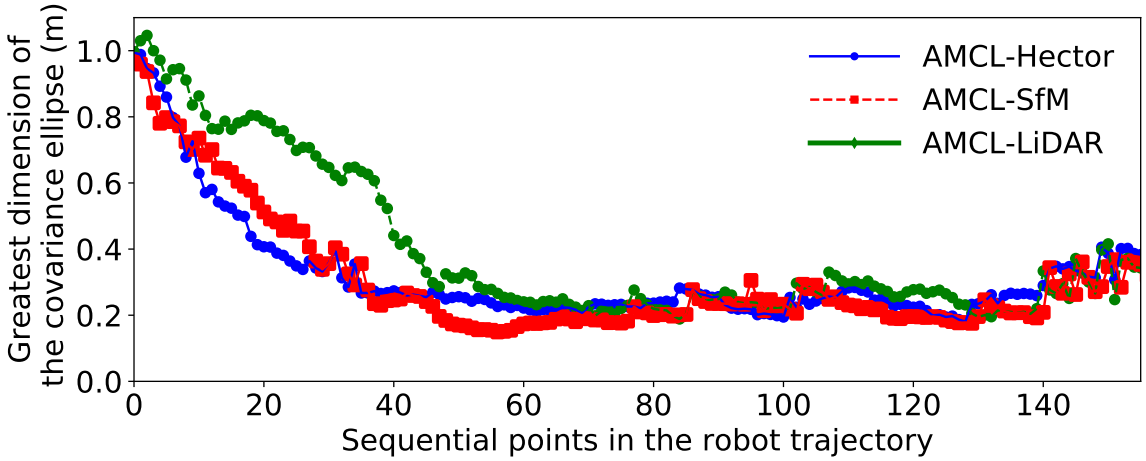


(ii): Comparing the covariance of the AMCL particles over the duration of the simulated robot trajectory, as calculated from the greatest dimension of the covariance ellipses seen in Figure 4.22ii .

Figure 4.24: The error between the reference trajectory of the robot and the trajectories calculated by AMCL-SfM, AMCL-LiDAR and Hector SLAM calculated in the simulation environment for subsequent points in the robot trajectory shown in Figure 4.22i (4.24i). The greatest value of the covariance, seen as the greatest dimension of the ellipses in Figure 4.22ii for each point in the robots trajectory (4.24ii).



(i): Comparing the error between the reference trajectory of the robot and the trajectories calculated by both local AMCL-Hector, AMCL-SfM and AMCL-LiDAR for each point in the reference robot trajectory. The error limit of $10\,\mathrm{cm}$ between the calculated trajectory and the reference trajectory is also marked.

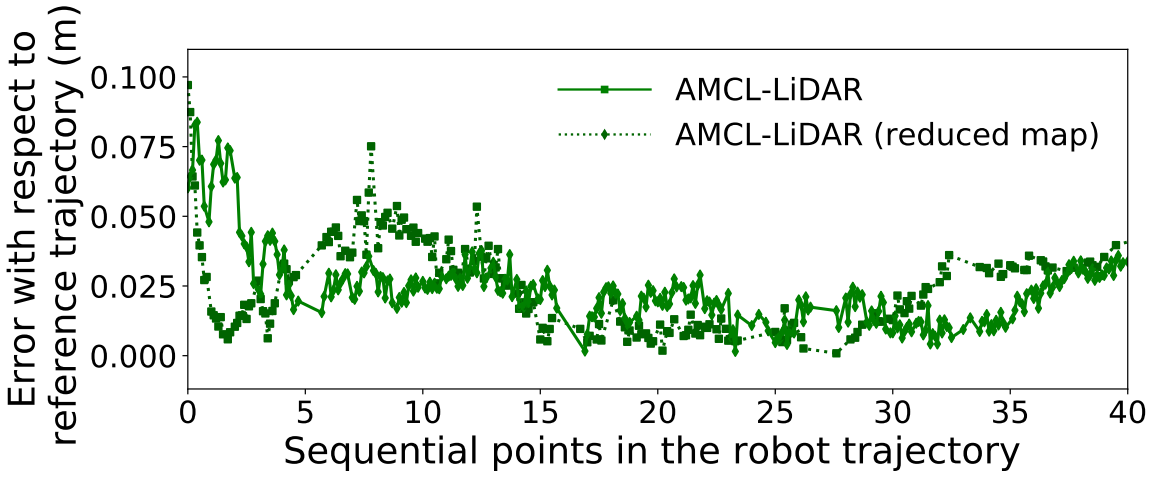


(ii): Comparing the covariance of the AMCL particles over the duration of the robot trajectory, as calculated from the greatest dimension of the covariance ellipses seen in Figure 4.15ii

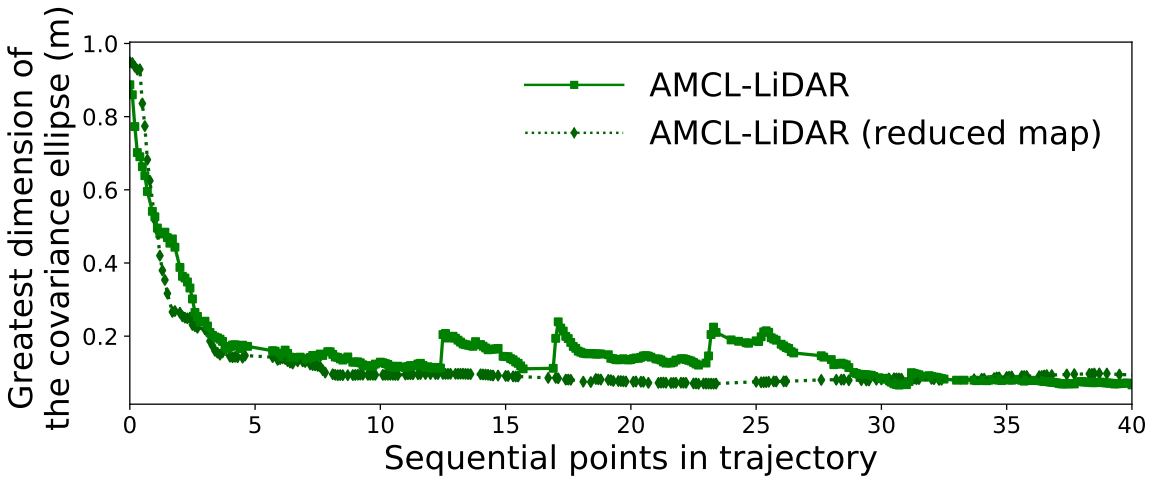
Figure 4.25: Error and covariance plots for AMCL local in the bridge environment. This Figure is a duplicate of Figure 4.16, and is provided for comparison to the simulation results.

As in the real environments, the uncertainty (seen in the covariance ellipses in Figure 4.22ii and the greatest dimension of these ellipses, as plotted in Figure 4.24ii) for AMCL-LiDAR and AMCL-SfM is greatest at the beginning of the robot motion and generally decreases over time. The values for the uncertainty, as represented by the greatest dimension of the covariance ellipse (Figure 4.24ii) are of a slightly lower magnitude compared the results from the real bridge environment (see Figure 4.25).

The error values for AMCL-LiDAR and AMCL-SfM are also similar to the real bridge environment, for most time steps. However, the error is greater for AMCL-SfM and AMCL-LiDAR in simulation than for the bridge environment (Figure 4.23) towards the end of the robot trajectory. To check whether the higher error is due to a difference in the size of the simulated environment compared to the real environment, the simulation for AMCL-LiDAR was repeated with a cropped map that better represents the simulated environment and plotted in Figure 4.26. Overall, Figure 4.26 shows that there is an increase in uncertainty as the platform moves towards the regions of the simulated environment which are not representative of the real environment (see Figure 4.27). However, there is little difference between the error values for the map from the real environment and the cropped map.



(i): Comparison of the error with respect to the reference trajectory for AMCL-LiDAR in simulation for the original and cropped AMCL-LiDAR maps.



(ii): Comparison of the covariance for AMCL-LiDAR in simulation for the original and cropped AMCL-LiDAR maps.

Figure 4.26: Comparison of the error and covariance for AMCL-LiDAR in simulation using the map for AMCL-LiDAR in the bridge environment and a cropped map that is more representative of the simulated environment.

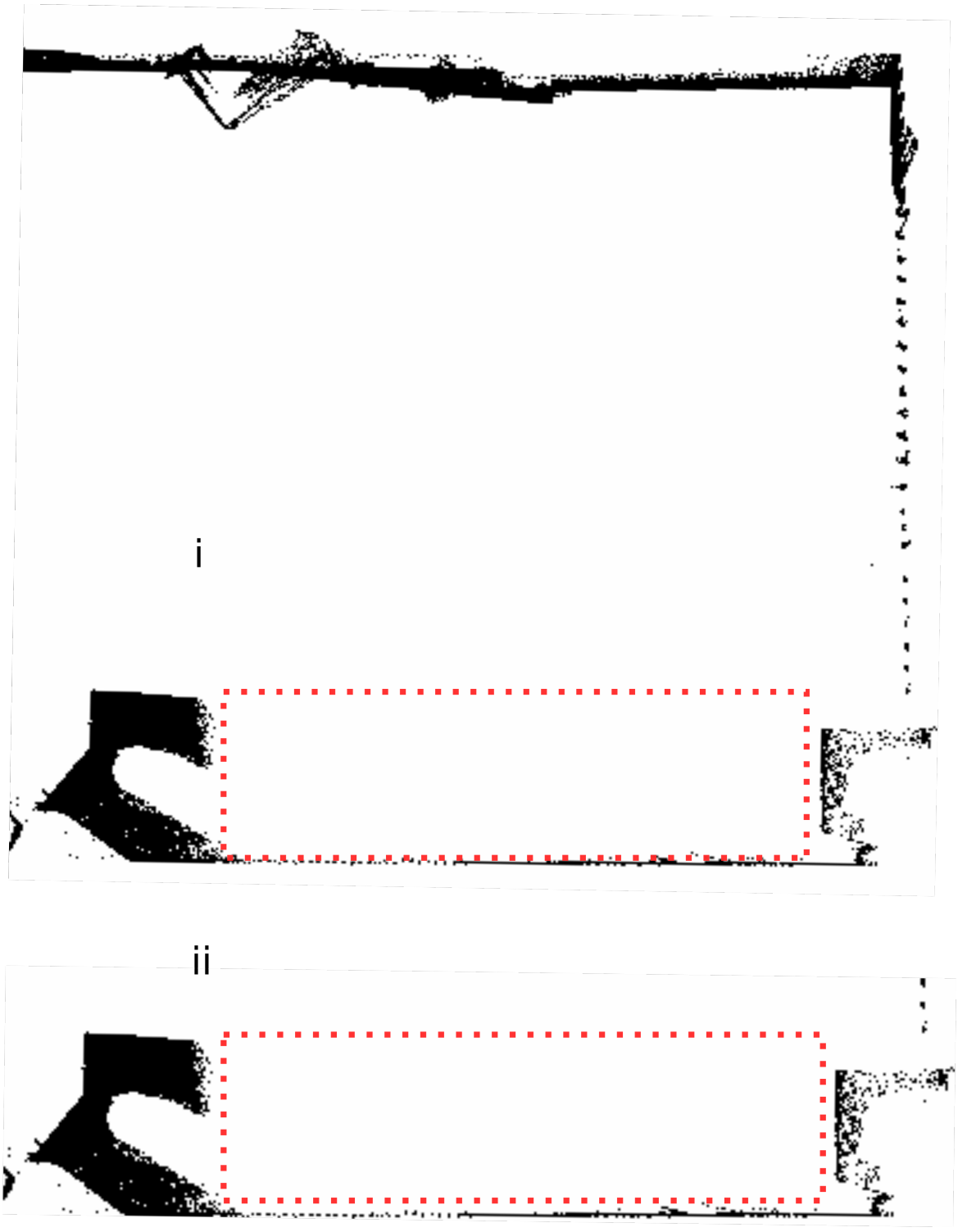


Figure 4.27: A comparison of the maps used for AMCL-LiDAR in both the real bridge environment (i) and simulation environment (i and ii). The approximate region of motion available to the robotic platform has been highlighted.

One of the potential benefits of using simulated environments is that different sensors can be tested and compared. For example, wheel encoder sensors were added to the robotic platform in simulation as an alternative method for providing odometry. The difference between the wheel odometry and the odometry from the 2D LiDAR is shown in Figure 4.28.

In Figure 4.28, the trajectories generated from the simulated wheel odometry sensor (purple) and the simulated 2D LiDAR odometry data (orange) are compared. The trajectory from AMCL-LiDAR from Figure 4.22 is also plotted for reference. In addition, the difference between the trajectories calculated using wheel and LiDAR odometry is also plotted in Figure 4.29 alongside the difference between the trajectories calculated using AMCL-LiDAR and the wheel odometry. Initially, the difference between the two odometry methods is low, but grows over time; this difference may be because the wheel encoders are prone to error caused by the wheels slipping. The error between AMCL-LiDAR and the wheel odometry trajectories is similar to the difference between AMCL-LiDAR and the reference trajectory (from 2D LiDAR odometry) in Figure 4.24i, but lower towards the end of the trajectory (comparing time-step 40 onwards for Figure 4.24i and Figure 4.29). It is possible that the lack of features in the simulated environment (see Figure 4.27) in this region means that wheel odometry gives a more accurate estimate for the robot position at this location. However it may also be coincidental, since the location with the fewer features is also at the end of robot trajectory, where the errors in wheel odometry due to slippage are also likely to be highest. This relationship should be verified further in future experiments.

Overall, there is scope to expand the work in this chapter by experimenting and prototyping different configurations of robotic platform with different sensors for localisation and mapping, but to also test different inspection environments. Future work may expand the initial investigations in this Section by comparing simulation environments generated using terrestrial LiDAR to simulation environments generated using CAD or BIM models for multiple bridge architectures and for scenarios where 3D point cloud data is not available. In addition, additional methods for SLAM could be tested in simulation, including visual SLAM

approaches. Further work might also investigate the need for the application of textures and colours to the meshes used in these simulation environments.

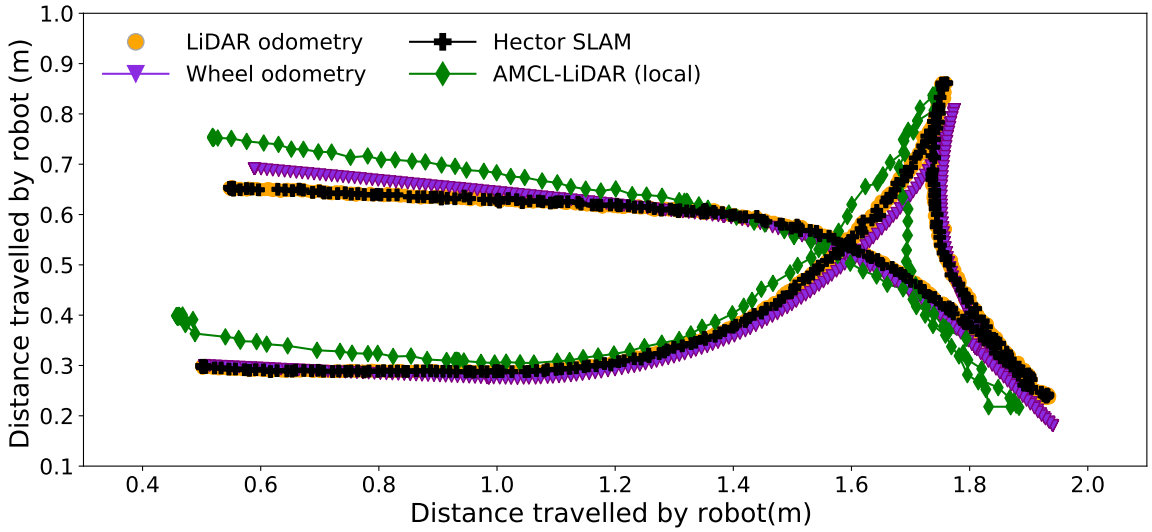


Figure 4.28: A comparison of the robot trajectories from Figure 4.22 (Hector SLAM, AMCL-LiDAR and odometry from the 2D LiDAR) and the trajectory calculated using a simulated wheel odometry sensor.

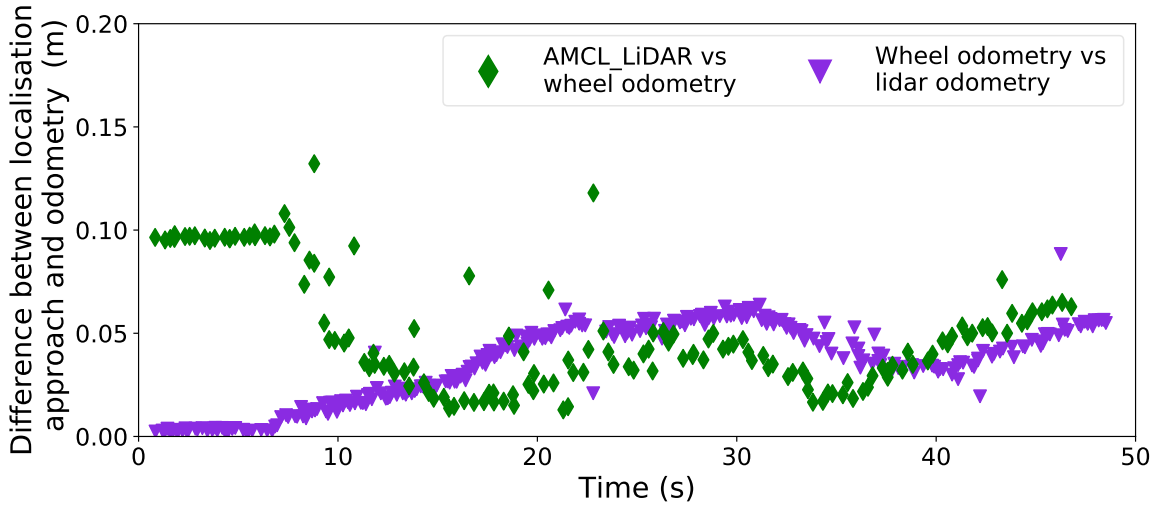


Figure 4.29: The difference between the odometry calculated using simulated wheel encoders and the simulated 2D LiDAR.

4.15 Summary and scope for future work

Autonomous inspection offers opportunities for reducing the risk, whilst also increasing the repeatability of bridge bearing inspection. Using a pre-defined map is important for autonomous navigation to allow targeted inspection. The map can be created from existing data, such as 3D terrestrial scans data, which is particularly useful in environments where the robot cannot be tele-operated in order to create a map. In addition, existing information (which may be difficult or expensive to obtain) can be supplemented and expanded over successive robotic inspections and this could be explored as part of future work. In this chapter, 2D occupancy maps were created from 3D SfM terrestrial LiDAR point cloud data.

A particle filter localisation approach called Adaptive Monte-Carlo Localisation was used to localise the robot in each of the maps created from point cloud data and also a map created from Hector SLAM. A 2D LiDAR SLAM approach called Hector SLAM, and a stereo and monocular visual SLAM approach called ORB SLAM, were also considered as an alternative solution to AMCL. Three environments were considered for testing, a laboratory environment, a real bridge environment and a simulated environment that was created using terrestrial LiDAR data from the real bridge environment.

In the laboratory environment, a typical trajectory error of $3\,\mathrm{cm}$ with respect to the reference was obtained for all methods, with the largest error of $8\,\mathrm{cm}$ occurring on initialisation of the particle filter for AMCL-SfM. AMCL with global localisation also converged, but with a greater error of $75\,\mathrm{cm}$ which was sustained for several time-steps and made it unsuitable for use in a real bridge environment. Future work could investigate suitable methods (e.g., rotation of the robotic platform on the spot) to ensure adequate convergence of global AMCL before the robotic platform begins any inspection tasks. An error bound of $10\,\mathrm{cm}$ was defined in the laboratory environment, with considerations being made about operating in the real bridge environment.

Two implementations (using a stereo and monocular camera) of a visual SLAM method called ORB SLAM were tested. When comparing error to the reference trajectory, neither implementations performed as well as Hector SLAM, with problems caused by the scale recovered by ORB SLAM and the loss of features in the environment at low frame rates

and camera resolutions. In addition, trajectories that require turn-in-place motions cannot be used with ORB SLAM, and bundle-adjustment is mostly performed on loop-closures. Since the space in the bridge bearing limits the trajectories that are possible, ORB SLAM was not tested further in the real bridge environment.

In the bridge environment, the trajectory error was generally greater than in the laboratory due to increased noise in the maps and in the environment, with a typical error ranging between $1\,\mathrm{cm}$ and $8\,\mathrm{cm}$. However, all methods remained below the defined error bound, with the exception of AMCL-SfM. By combining the results from AMCL-SfM with Hector SLAM, a more robust method for determining the trajectory is possible that keeps the trajectory error below $6\,\mathrm{cm}$ throughout. For AMCL, the largest errors occurred at the beginning of the robot trajectories, which was due to uncertainty in the initial position estimate of the robot.

Terrestrial LiDAR data was used successfully to create a simulation environment. Again, AMCL and Hector SLAM were compared to the reference trajectory created using the simulated 2D LiDAR data. These methods were implemented successfully, although the results were different to the results in the real environment. The main reason for the difference between the real and simulated environment is likely due to the limited region of the bearing enclosure that was used in simulation, but may also be related to the physics parameters used in the simulations; these areas should be the subject of future research. However, for AMCL, the errors seen in the simulation were arguably closer to the real bridge environment than those from the laboratory environment, which shows good potential for using the simulation environment for testing different robot configurations prior to testing in the real environment.

To conclude, the following points are suggested as areas for further investigation:

- Further evaluation of the combined approach between AMCL and Hector SLAM proposed in this chapter.
- Exploration of SLAM methods that combine data, such as LiDAR and camera data.
- Expand the 2D implementation in this work to 3D using sensors such as the stereo camera to detect objects on the floor of the enclosure and the edges of the enclosure.

The known bearing enclosure could also be identified using cost-maps (which can be used by ROS) generated from the existing inspection data.

- Adding sensors, such as an IMU to give improved performance for measuring uncertainty in rotation
- Exploration of the use of a docking station to allow repeatable initialisation of AMCL or the combined approach to improve the initial position estimate of the robotic platform.
- Further testing and integration of SLAM methods, such as line-based visual SLAM and 3D LiDAR-based approaches.
- Further testing of the methods from this chapter in different real bridge configurations.
- Long-term testing of the methods and sensors proposed in this chapter to better understand the implications for the robustness of the chosen SLAM approach.
- Testing different configurations of bridge structure in simulation to evaluate the generalisation of the methods presented in this chapter.
- Testing different configurations of the robotic platform in simulation, including different sized platforms and the addition of sensors, such as IMU.
- Testing of SLAM methods such as Google Cartographer, which allows localisation in a previously mapped area, and visual SLAM with different features, such as ORB SLAM where the ORB features are replaced with line features (which are common features in urban environments).
- Further testing of the effect of the physics parameters (such as friction and sensor noise) on the results in the simulated environment.

Chapter 5

Application of computer vision techniques to visual inspection tasks

5.1 Crack detection using computer vision methods

The literature review in Chapter 1 highlighted that cameras and computer vision are useful tools for aiding the inspection of civil infrastructure, with almost all of the current developments in robotic bridge inspection including at least one camera sensor. In addition, crack detection in the concrete structure of the bridge was the most commonly reported purpose for the camera on the robotic system. The aim of using robots mounted with cameras was to increase the automation of the inspection process and to reduce the subjectivity and increase the occurrence of such inspections (Lattanzi and Miller, 2017; Agnisarman et al., 2019). The use of a camera also aids the requirements of visual inspections, which is required for components such as the bridge bearings (BS-EN-1337, 2003; Sutter et al., 2018). A second focus of crack detection is to find defects in road surfaces. For example, Varadharajan et al. (2014) mount a camera in a vehicle in order to monitor the cracks in asphalt road surfaces around the city. Road surfaces commonly consist of concrete or asphalt, and there is overlap in the methods used for detection of cracks in these surfaces. Due to a lack of general datasets for crack detection in concrete infrastructure, the datasets and methods for crack detection in concrete and asphalt roads will be considered in this chapter. A

summary of recent literature for computer vision approaches for crack detection in images is given in Table 5.1.

Table 5.1: A summary of the review of recent literature that focuses on computer vision for crack detection in photographs. Rows highlighted in grey show research with published datasets. Related projects are marked with *.

Literature Summary			
Reference	**Description of approach**	**Task**	**Dataset**
Image processing methods			
Lim et al. (2011)	Image processing with Laplacian of Gaussian	Segment Asphalt	N/A
Su (2013)	Image processing with weighted median filter, image opening Otsu's thresholding and measurement of morphological features	Segment Concrete pavement	Private dataset collected by hand. 50 images with cracks, 50 non-crack. 1536×2048 px resized to 335×413 px
H. Li et al. (2017)	Multiple image scales using gaussian blurring. Group crack seeds into clusters and use window minimal intensity path algorithm to grow cracks. Cracks matched across image scales.	Segment. Asphalt	Compared results to dataset from Shi et al. (2016) 480×320 px
Nayyeri and Hou (2016)	Local structure extraction to preserve strong edges. Followed by binarisation. K-means used to group background texture	Segment from Concrete and asphalt road surface	Private dataset of 704 images. 352 for training, 352 for testing, repeated 10 times. 400×500 px

Continuation of Table 5.1

Reference	Description of approach	Task	Dataset
Sato (2018)	Image processing using V-shaped detector for edge detection	Segment Bridge pier - concrete	Private dataset collected in lab, 3–7,m from concrete specimen using DSLR camera. 6016×4000 px
Machine learning approaches			
Varadharajan et al. (2014)	Feature extraction and description with SVM classifier	Segment super-pixel regions Asphalt	Private dataset. 220 images for training, 140 for testing
L. Li et al. (2014)	Image processing for crack segmentation, with a back-propagation neural network for crack recognition. Work also focuses on separately classifying linear and crocodile cracks.	Segment and classify Asphalt	Private dataset collected using an Automated Road Analyser (ARAN) 400 images, 60 % used for training, 20 % validation, 20 % testing
Prasanna et al. (2016)	SVM, Adaboost and random forest trained on different feature vectors generated using RANSAC. Method tested on two distinct datasets.	Segment. Bridge surface - concrete	Private dataset collected using robotic system. 100 images split into corresponding to a bridge segment for 2 different bridges 1000 samples taken from two datasets for training and validation. Equal positive and negative instances of cracks 920×1280 px

Continuation of Table 5.1

Reference	Description of approach	Task	Dataset
Shi et al. (2016)	Structured forests with SVM classifier	Segment Asphalt	Developed dataset called Crack Forest. 118 images taken on smart phone camera. $60\,\%$ for training $40\,\%$ for testing. Images reduced to 480×320 px.
G. Li et al. (2017)	SVM classifier with features generated using the ratio, regularity, aspect ratio and eccentricity ratio of connected pixels and the Hu invariant moment.	Segment Concrete	Used own dataset generated using DSLR camera, flash lamp, autofocusing apparatus and laser distimeter and angular transducer for distance and shooting angle of the camera from the crack.
Cubero-Fernandez et al. (2017)	Image filtering with logarithmic transformations, bilateral filter, Canny edge detection and morphological filtering. Decision tree for classification	Segment Asphalt	Private dataset, labelled by expert. Unspecified collection conditions. 600 for training, 400 for testing
Deep-learning approaches			
Makantasis et al. (2015)	Convolutional Neural Network (CNN) compared to SVM, kNN and classification tree	Classification Tunnel walls-concrete	Data collected by hand-held DSLR camera $100,000$ samples split into $8:1:1$ ratio for training validation and testing
2016 F. Yang et al. (2016)	CNN	Classification then segmentation Asphalt	Developed dataset called CrackNet $640,000$ patches of 99×99 px for training

Continuation of Table 5.1

Reference	Description of approach	Task	Dataset
2017 L. Yang et al. (2017)	CNN. Fine-tuned VGG16 network	Patch-based classification Bridge structure - concrete	Developed shared dataset. 954 images split into regions of interest of 100×100 and 130×130 px. Fine-tuned with images collected at bridge
Y.-J. Cha et al. (2017)	CNN	Concrete surfaces	Private dataset collected by hand. 277 images for training cropped to $40,000$ patches. $4,928 \times 3,264$ px split into 256×256 px.
Pauly et al. (2017)	CNN	Patch-based classification Asphalt	Dataset from F. Yang et al. (2016) 100000 image patches for training, $40,000$ for testing. 99×99 px.
Yokoyamma and Matsumoto (2017)	CNN	Patch-based classification Concrete surfaces	Private dataset. 2000 images for training, 1000 for testing. $64x64$ px patches.
Maeda et al. (2018)	Object detection CNN	Crack region detection Asphalt	Private dataset. Bounding box of defect is labelled. $7,240$ images for training. $1,813$ for testing. 600×600 px reduced to 300×300 px.
Nhat-Duc et al. (2018)	CNN combined with Canny and Sobel edge detection	Patch-based classification. Custom dataset collected on a mobile phone 150×150 px	

Continuation of Table 5.1

Reference	Description of approach	Task	Dataset
Zhang et al. (2018a)*	CNN with cross-entropy loss	Segment Asphalt	Private 3D dataset collected using PaveVision3D system with cameras and laser. $2,500$ images for training, 300 images for validation, 200 images testing. $1,025 \times 512$ px
Zhang et al. (2018b)*	Comparing Recurrent Neural Network architectures	Segment Asphalt	Private 3D dataset collected using PaveVision3D system with of cameras and laser $3,000$ images for training, 500 images for testing, 500 for validation. $1,025 \times 512$.
Gibb et al. (2018)	CNN with genetic algorithm to optimise CNN structure	Patch-based classification Asphalt	Developed own dataset, but perhaps made comparison to Y.-J. Cha et al. (2017) $3,000 : 1,500$ for testing, $1,500$ for training 256×256
2018 Özgenel and Sorguç (2018)	Compared the performance of fine-tuning many existing, pre-trained, CNN architectures	Patch-based classification	Largest training data size was $28,000$ patches of 227×277 pixels. Published dataset of Middle East Technical University.
X. Yang et al. (2018)	CNN, using the fully connected network with pre-trained network (VGG-19)	Segmentation dataset available online. 267×241 px.	

Continuation of Table 5.1

Reference	Description of approach	Task	Dataset
Vu Dung and Duc Anh (2019)	CNN, using the fully connected network with pre-trained network (VGG-16)	Segment Asphalt	Used dataset from Özgenel and Sorguç (2018), which uses the method from F. Yang et al. (2016). Selected 600 images and manually segmented using MATLAB tool. 227×227 px

In most of the examples in Table 5.1, the aim of the research was to segment the crack location from the image (i.e., to find the pixel coordinates of the crack within the image). Segmentation can also be understood as a pixel-wise classification task, where the pixel belongs to either a category of crack or non-crack. The two most common applications for crack detection in photographs are for cracks in asphalt (i.e., road surfaces (H. Li et al., 2017; Varadharajan et al., 2014; Cubero-Fernandez et al., 2017)) or cracks in concrete (i.e., structures such as bridges (Sato, 2018; Prasanna et al., 2016; L. Yang et al., 2017)).

The initial approaches to segmentation of cracks from photographs were performed using computer vision methods that focused on edge detection algorithms (e.g., Sobel, Canny, fast Haar transform (Abdel-Qader et al., 2003)) or colour discrepancy (since a crack typically has darker pixels than the normal surface (Chambon and Moliard, 2011)) to allow thresholding of the image to find the regions of the image that contain a crack. However, these methods tend not to be robust to noise (e.g., caused by background texture in the image) leading to results with many false positives (Chambon and Moliard, 2011) and require tuning of parameters for best results in different scenarios. Edge detection methods can also fail to join the crack segments, and produce poor results in cluttered images (H. Li et al., 2017). To reduce noise in images, preprocessing steps are applied by filtering the images, such as median filters or opening and closing morphological filters to join crack segments (Noh et al., 2017; Su, 2013).

Recent advances in the field of machine-learning, has initiated a growth in the research for automated crack detection in concrete and asphalt. Commonly, features are extracted from the images using image processing methods and then classifiers, such as support vector machines (SVM), random forests or k-nearest neighbour (KNN) algorithms are used to classify whether a test image contains a crack. In the literature shown in table 5.1, features were created using filtering methods such as Gaussian blurring , and SVM was the most commonly used classifier.

However, most of the recent applications for crack detection in images use deep learning, specifically an approach based on convolutional neural networks (CNNs), also referred to as deep learning. One reason for the increase in use of CNNs for crack detection is because they do not require hand-crafted features as developed in other machine learning approaches, and appear to be more invariant to noise (Nayyeri and Hou, 2016). However, rather than segmenting the crack from the image, the majority of these approaches focus on classification of image patches as either a crack or not a crack. It is likely that the reason for using CNNs for classifying image patches, rather than segmenting the pixels containing cracks, is due to the use of CNNs more broadly in the literature for tasks such as object classification (e.g., (Simonyan and Zisserman, 2014)). In addition, the burden for creating ground truth datasets for segmentation of cracks from images is much greater than for patch-based classification, since labelling is required at a pixel level, especially since a large amount of data tends to be required for deep learning approaches. Zhang et al. (2018a) highlight how using small patches for crack detection introduces false positives into the classification results since cracks and background image texture are similar at small scales.

The architecture of CNNs is adapting in the broader literature and being used for a wide range of tasks. The development of public datasets and architectures for edge detection has been used as inspiration for medical segmentation of retinal vessels from images of eyes for diagnosing patients with diseases such as Coat's disease and diabetes (Staal et al., 2004; Hoover, 2000). Similarly, deep learning networks that were developed for medical segmentation have also been used to extract road networks from satellite images (Panboonyuen et al., 2017). Research is moving towards the use of CNNs for segmentation of cracks from

images, but this has not yet been fully implemented, with only one recent example by Vu Dung and Duc Anh (2019). Vu Dung and Duc Anh (2019) use the dataset from Özgenel and Sorguç (2018) and manually segment the image patches. Again, it is difficult to compare the results of this paper with others from the literature as no comparison of the approach by Vu Dung and Duc Anh (2019) is made to existing datasets or methods, nor is there implementation (with networks parameters etc.) publicly available. In addition, the adapted dataset has not been published (at the time of writing).

One challenge of using deep learning for crack detection is the quantity and quality of the available datasets on which learning of tasks can be performed, since ground-truth labels are required in order to perform training and typically a greater amount of data is required than for other machine learning approaches. This restriction creates an overhead for preparing the datasets, particularly if an expert labeller is required to annotate the ground truth images. In recent years, this challenge has been reduced for some applications (e.g., house-hold object classification) due to the development of open-source, publicly available datasets (e.g., the ImageNet dataset (Fei-Fei et al., 2010)).

The literature reviewed in Table 5.1 shows a lack of agreement around how datasets for the segmentation of cracks should be used, particularly for deep learning applications where there is little agreement or comment about the amount of training data required, the size of images that should be used and the conditions under which the images should be collected. There are also very few public datasets that are agreed upon as being a gold-standard for testing new algorithms and approaches, with many of the examples in Table 5.1 being collected from scratch. In contrast, in other fields of research, algorithms are often bench-marked against one dataset, such as the BSDS dataset for image boundary segmentation (Martin et al., 2001). The use of a common dataset allows the evaluation of the effectiveness of the algorithm being presented, and in the current literature it is unclear how much of an effect is due to the quality of the private datasets (i.e., datasets that are not being made publicly available). The need for more public datasets has also been highlighted by Zhang et al. (2018a) and Cubero-Fernandez et al. (2017). Some datasets have developed recently (e.g. Özgenel and Sorguç (2018)), but due to the popularity of CNNs in

the literature, many of the datasets are in the form of small patches, rather than full-image pixel-wise segmentations.

A potential solution to the lack of datasets for training CNNs is the use of transfer learning to a deep learning network. Transfer learning is applicable in different types of machine learning algorithms and refers to 'the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned' (Torrey and Shavlik, 2009). A related approach is the so-called fine-tuning of a network, where a previously trained architecture (or a set of weights generated from a previously trained network e.g., VGG16 (Simonyan and Zisserman, 2014)) can be introduced to a new dataset. Fine-tuning a network is therefore useful if a large dataset does not exist for the current application, since it can be used to extract general features from the previous larger dataset. The result of this process is a network specialised on the current dataset, but without the need for large training time and datasets. Recently, (Özgenel and Sorguç, 2018) reviewed the performance of several existing and pre-trained network architectures (AlexNet, variations of VGG, GoogleNet, and variations of ResNet), which were then fine-tuned using a dataset of cracked and non-cracked concrete surfaces. The authors concluded that the features from these pre-trained networks were applicable for crack detection, with all of the tested networks giving an accuracy of over $90\,\%$. The authors also published their classification dataset of patches of cracked and non-cracked concrete surfaces.

There is also disagreement in the literature regarding the methods that should be used for evaluating the effectiveness of the crack detection algorithms. Most of the approaches in Table 5.1 adopted some measure of precision and recall, which are common metrics in machine learning and will be described further in Section 5.8. However, there were inconsistencies with whether the mean or maximum values should be used. The methods for evaluating crack segmentation algorithms will be discussed further in Section 5.8.

Overall, the use of deep learning for crack detection in images appears to be a promising approach for addressing the need for tuning in traditional computer vision algorithms. However, the use of classification CNNs is limited due to the addition of false positives caused by similarities at a patch level and the computational overhead.

In this chapter, segmentation of cracks from images of concrete will be performed using a deep learning network which has been used for boundary segmentation (Xie, 2015; Xie and Tu, 2017), edge detection and blood vessel segmentation in the retina (Maninis et al., 2016). In addition, a dataset that was developed for a machine learning approach for crack segmentation in asphalt will be used to test the deep learning approach and to compare this method with the state of the art. Furthermore, to address the shortage of good-quality datasets of cracks in concrete infrastructure for training deep learning networks, data from different domains, including asphalt, concrete and the retina, were used in this work to fine-tune the deep learning network for a crack segmentation application; these fine-tuned networks are then compared on two datasets of cracks to evaluate their performance.

5.2 Method overview: deep learning

The area of research referred to as 'deep learning' is a subsection of machine learning and artificial intelligence, that has its origin in research dating back to the 1940s, with artificial neural networks being researched as computational models for biological learning (Goodfellow et al., 2015). In 1990, (Le Cun et al., 1990) developed a multi-layer artificial neural network for classification of handwritten digits, which has been recognised as a key influence in the development of modern CNNs (Gu et al., 2015). In general, these feed-forward and back-propagation networks define a mapping $y = f(x; \Theta)$ from an input x to output y by learning parameters Θ (typically a set of weights and biases) that best approximate the mapping function (Goodfellow et al., 2015). This mapping function tends to made up of a network of many layers, where the depth of such a network refers to the number of chained function layers and includes common layers, such as convolution layers, rectified linear unit layers (ReLu), pooling layers and fully-connected layers.

More recently, popularity of deep learning methods has been influenced by improvement in graphics processing units and open source datasets that has allowed large-scale image classification objects, with well-known CNNs including: AlexNet, VGG-Net, GoogleNet and ResNet (Gu et al., 2015). The trend during the development of these CNNs was

in increasing the number of layers (i.e, the depth of the network), which allows better approximation of target function and hence classification of objects in images, but at increased risk of over-fitting (Gu et al., 2015). An example of the VGG-16 architecture is shown in Figure 5.1.

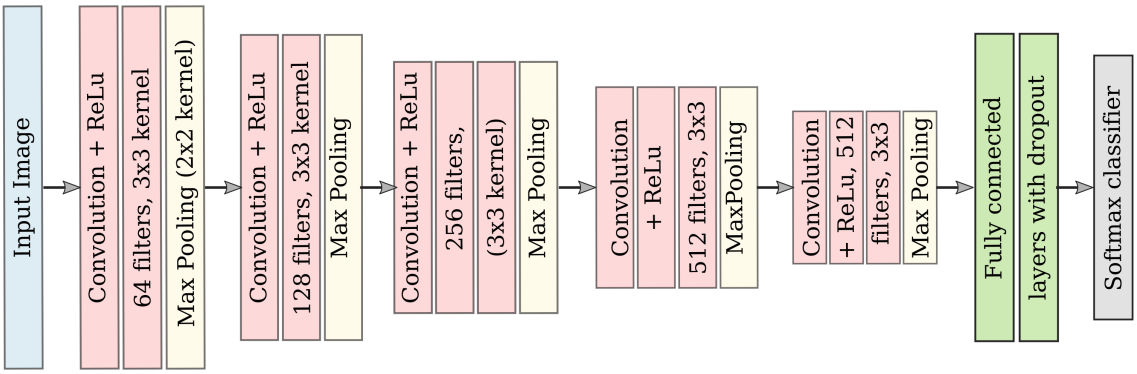


Figure 5.1: Example structure of the VGG-16 network architecture (Simonyan and Zisserman, 2014). Convolution, ReLu and Max-Pooling are applied in successive layers. Finally fully connected and softmax layers are applied to calculate the loss for an input image and to classify the output.

CNNs are most applicable to grid-like topologies, such as time-series data, image data and speech recognition. The architecture of CNNs varies, but there are several key components: convolution, pooling and fully-connected layers. Convolution layers are composed of several convolution kernels that are used to compute feature maps (Gu et al., 2015). Convolutional networks apply convolution in the place of matrix multiplication in at least one layer (Goodfellow et al., 2015). In CNNs, convolution is the mathematical operation of applying a kernel (or filter) across the input data, e.g., an image. The kernel usually has a width and height smaller than the input data, but has the same depth. The output of this process is an activation map of the response to that filter across the input data. Convolution is commonly followed by a ReLu layer which performs element wise comparison on the convolution output and uses an activation function to convert negative values to zero, which is required to introduce non-linearity into the model (Gu et al., 2015).

Pooling layers tend to occur between convolutional layers and are used to add invariance in translation of an input by reducing the resolution of the feature maps (Gu et al., 2015),

typically by taking the maximum or mean in regions across the feature map. The stacking of convolutional and pooling layers allows the extraction of features from the input data at different scales. The benefits of this process will be discussed further in Section 5.10.2. Finally, fully connected layers are typically where the training loss (difference between training data and ground truth labels) is computed. A common CNN architecture stacks convolution, ReLu and pooling layer in a linear manner and use the fully-connected layer to calculate the loss. However, different types of layers and architectures are developing over time. The specific architecture used in this work is discussed in Section 5.3.

5.3 The HED network architecture

Two approaches are compared in this chapter for the segmentation of cracks from images of concrete and asphalt. The main approach is a deep learning network for Holistic Edge Detection (HED) (Xie, 2015; Xie and Tu, 2017). The HED network is compared to a machine learning approach for crack segmentation that uses a structured forests and a support vector machine (SVM), see Section 5.4.

The architecture of the HED network is built upon the VGG-16 convolutional neural network (Simonyan and Zisserman, 2014), the fully connected neural networks (Long et al., 2015) and the deeply-supervised network (DSN) (C. Lee et al., 2015) architectures. Similarly to the VGGNet architecture, the first five stages of the network comprise of between two and four convolution layers followed by a max-pooling step. However, the HED network (Xie and Tu, 2017) takes inspiration from DSN by using the whole image, rather than image patches, as an input and output and using the outputs from the final convolution layer of each step (referred to henceforth as side outputs) to learn the features at multiple image scales. Each of the side-outputs is associated with a classifier and an objective function, such that:

$$\mathscr{L}_{side}(\mathbf{W}, \mathbf{w}) = \sum_{m=1}^{M} \alpha_{(m)} l^{m}_{side}(\mathbf{W}, \mathbf{w}^{m}) \tag{5.1}$$

where $\mathbf{W}$ is the collection of standard network parameters, $\mathbf{w}$ is the set of weights optimised for each of the side-outputs ($\mathbf{w} = (\mathbf{w}^1, \ldots, \mathbf{w}^m)$); l_{side} is the image-level loss function; M is the number of side outputs in the network — five for HED — and $\alpha_{(m)}$ is the loss weight for the m^{th} layer.

The image-level loss function, l_{side}, has been defined to include a term, β, that is used to balance the loss between positive and negative classes (e.g., pixels that contain edges and not edges). Here, $\beta = |Y_-|/|Y|$ and $(1-\beta) = |Y_+|/|Y|$; $|Y_-|$ denotes the set of edge ground truth pixels and $|Y_+|$ denotes the set of non-edge ground truth pixels. This term is required because there is a typical imbalance in images of natural scenes where around

$90\,\%$ of the image is a negative class, i.e., not an edge (Xie and Tu, 2017). The loss function at each side output is defined as:

$$l^m_{side}(\mathbf{W}, \mathbf{w}^m) = -\beta \sum_{j \in Y_+} logp(y_j = 1|X; \mathbf{W}, \mathbf{w}^{(m)}) - (1-\beta) \sum_{j \in Y_-} logp(y_j = 0|X; \mathbf{W}, \mathbf{w}^{(m)}) \tag{5.2}$$

The probability, p, that a pixel in a given image is an edge is calculated using a sigmoid function, σ, on the activation of the model (for the m^{th} side output) output value at the pixel, i.e., $p(y_j = 1|X; \mathbf{W}, \mathbf{w}^{(m)}) = \sigma(a^{(m)}_j) \in [0, 1]$. The output of each of side outputs is a predicted edge (or crack or retina) map, such that $\hat{Y}^{(m)}_{side} = \sigma(\hat{A}^{(m)}_{side})$, where $\hat{A}^{(m)}_{side} \equiv a^{(m)}_j, j = 1, \ldots, |Y|$ for side output m. Each of these side outputs can be considered as a mini-network that classifies edges at different scales, where the output becomes more coarse with progression through the network (see Figure 5.2ii).

An edge map is also obtained by fusing the results of the side-outputs. This stage also has a loss function:

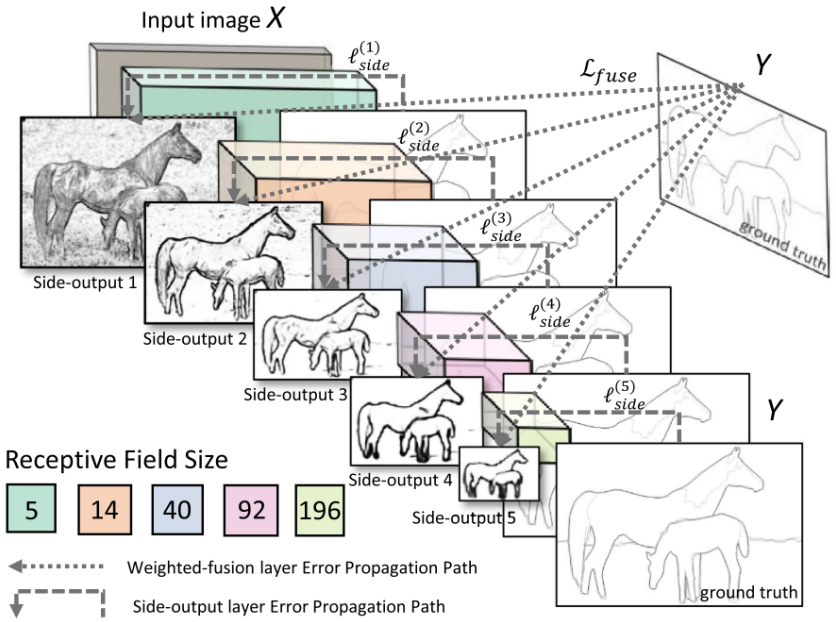
$$\mathscr{L}_{fuse}(\mathbf{W}, \mathbf{w}, \mathbf{h}) = Dist(Y, \hat{Y}_{fuse}) \tag{5.3}$$

where $\mathbf{h} = (h_1, \ldots, h_M)$ are the weights for fusing each of the side-outputs and $Dist(Y, \hat{Y}_{fuse})$ is the distance between the fused predictions and ground truth label map, which is a cross-entropy loss calculation.

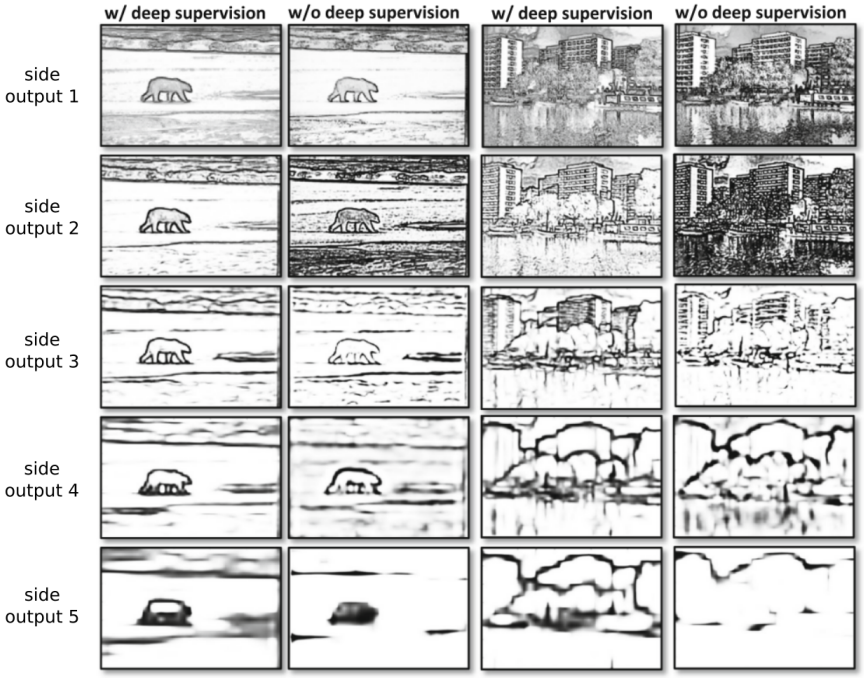
The optimal solution is:

$$(\mathbf{W}, \mathbf{w}, \mathbf{h})^* = argmin(\mathscr{L}_{side}(\mathbf{W}, \mathbf{w}) + \mathscr{L}_{fuse}(\mathbf{W}, \mathbf{w}, \mathbf{h})) \tag{5.4}$$

The architectures of HED is shown in Figure 5.2i.



(i): HED network architecture



(ii): Output from HED

Figure 5.2: Network architecture of the HED network (5.2i) and two examples of the HED network with the output of each of the side-outputs shown at each row for two initial images (left and right in Figure 5.2ii). For each example, the results become coarser with progression through the network. Using deep supervision gives better object boundary definition.

5.4 Summary of the Structured Forest Approach

A second method, henceforth referred to as 'the structured forest approach' (SFA) (Shi et al., 2016), will be compared to the deep learning approach described in Section 5.3. An overview of SFA is given in this section.

First, the authors created an annotated dataset. This dataset is the Crack Forest dataset and is described in more detail in Section 5.5.1. Next, 16×16 patches were extracted from the images along with their corresponding ground truth annotations. Patches that contain an instance of a crack at the centre pixel of the patch were considered a positive sample, other patches were discarded. Tokens (binary image patches with white pixels representing cracks and black pixels representing background) that describe the cracks are then extracted from the patches by computing features such as: the patch mean, standard deviation and integral channel features. Example tokens are shown in Figure 5.3.

These tokens were then grouped using a structured random forest, see Figure 5.3. The ground truth segmentation patch and associated features is recursively passed down the branches of a tree until they reach a leaf, where tokens with the same features are gathered. The number of leaves on the structured forest tree is equal to the number of independent tokens. The most representative tokens in each leaf are extracted from the structured forest tree and the statistical histogram (showing the occurrence of each token in the dataset) and neighbourhood histogram (which shows the occurrence of two tokens next to each other) can then be extracted for a given input image as statistical descriptors. These descriptors are used to train different classifiers (e.g., a KNN and SVM classifier) to allow cracks to be discriminated from noise in an image (Shi et al., 2016).

A test image is split into patches and the features are calculated using the same approach as for the training samples. These test tokens are then passed down the trained structured forest tree. A threshold is applied to the resulting images to remove the areas where there is low probability that a crack is present. Finally, classification using an SVM removes noise. The output of the testing process is a probability map, where each pixels are classified using a probability that a crack is present at that pixel.

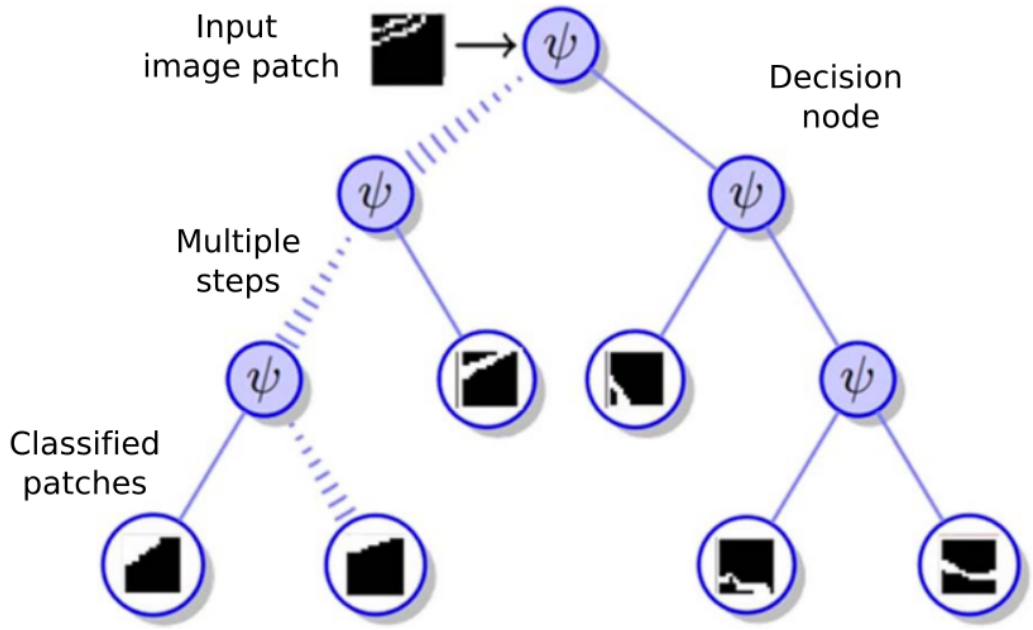


Figure 5.3: Visual example of SFA, with an input image token being passed down the decision tree, where ψ represents the decision at each branch of the decision tree. The image token passes down the decision tree to reach a 'leaf' (circled end nodes) that best matches its features. The dashed edges of the tree represent multiple branching stages of the tree. Image adapted from Shi et al. (2016).

5.5 Description of existing datasets

The literature review in Section 5.1 showed that the datasets that have been developed for CNNs are for classification of image patches. However, the most valuable information is arguably found in segmentation datasets, although the creating a ground truth for segmentation may be more time consuming than for crack classification datasets. A summary of the currently available datasets for crack segmentation in photographs of concrete and asphalt are described in the following sections and in Table 5.2.

5.5.1 Dataset 1: The Crack Forest segmentation dataset

The Crack Forest (CF) dataset is composed of 118 images, collected in Beijing, China, and consists of cracks on the surface of asphalt roads. Each image has a resolution of 480×320 and has a corresponding hand-labelled ground truth segmentation. All the images are taken by an iPhone 5 with focus length of $4\ \mathrm{mm}$, aperture of f/2.4 and exposure time of $1/134\ \mathrm{s}$ (Shi et al., 2016). An example of the CF dataset and corresponding ground truth label is shown in Figure 5.4i and 5.4iii, respectively. The CF dataset has been used as a benchmark for comparison of different algorithms and against an existing dataset by (Shi et al., 2016; H. Li et al., 2017).

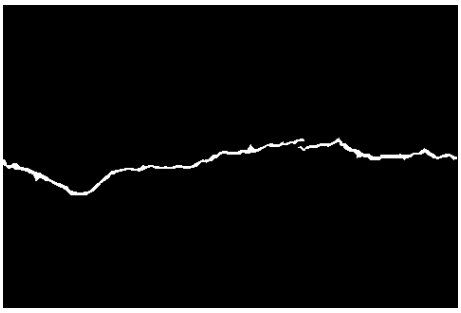
5.5.2 Dataset 2: Concrete Structure Spalling and Crack segmentation dataset

The Concrete Structure Spalling and Crack (CSSC) (L. Yang et al., 2017) dataset is unique in the current literature for crack detection as it uses search engine results (e.g., from Google, Yahoo, Bing and Flickr) to collect images of both cracking and spalling (where the concrete surface delaminates) in concrete, rather than hand-collected data. This dataset is different from the CF dataset, which focuses on cracks in asphalt materials, because factors, such as the background material of the images, in the dataset consist of a much wider variety of colours and textures and intensity. The images were manually labelled and although the total dataset consists of 954 crack images, 149 have been made available with manually segmented

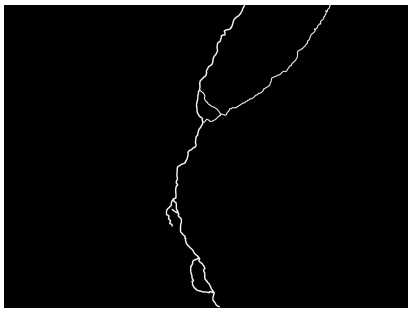
ground truth images. An example of the CSSC dataset and corresponding ground truth label is shown in Figure 5.4ii and 5.4iv, respectively. CSSC is a more recent dataset than CF and currently has not been compared with other datasets through evaluation on different network architectures.

(i): Crack Forest

(ii): CSSC



(iii): Crack Forest ground truth



(iv): CSSC ground truth

Figure 5.4: Example colour images (5.4i–5.4ii) and corresponding hand labelled ground truths (5.4iii–5.4iv) from datasets of cracks in concrete material. In Figures 5.4iii and 5.4iv, black pixels correspond to image regions from 5.4i and 5.4ii where there are no instances of a crack and white pixels correspond to image regions from 5.4i and 5.4ii where there are cracks.

Table 5.2: A summary of original datasets of cracks in photographs of concrete and asphalt. A brief description of the dataset is given along with the associated image resolutions and number of images for each dataset.

Dataset name	Dataset type	Image resolution (pixels)	Total number of images at original resolution
Crack Forest	Cracks in asphalt	480×320 px	118
CSSC	Crack in concrete	varying	149

5.6 Creating Training and Testing Datasets

The CF dataset will be used to test the performance of the HED network for crack segmentation in this chapter. As described in Section 5.5.1, the CF dataset comprises of cracks in asphalt (not concrete), but is considered as a dataset in this work due to a general lack of crack datasets in concrete that have a segmentation ground-truth, and because the CF dataset has also been used for some comparisons with other methods in the literature. Cross-dataset evaluation will also be performed using a mixture of CF and CSSC for training and testing. The resolution of the images is different for the CF and CSSC datasets, and is different across images in the CSSC dataset. However, many of the existing deep learning network architectures require the input data to have a consistent size and typically this is achieved by either creating image patches (e.g., Alom et al. (2018)) or by resizing the images (e.g., Simonyan and Zisserman (2014)). Creating image patches allows variability in the input data size, but creates a large computational overhead for preprocessing and for training of deep learning networks (Guo et al., 2018).

The implementation of the HED network architecture (described in Section 5.3) uses a full input image resolution of 500×500 pixels, but the resolution of the CF dataset is 480×320 pixels. In order to compare SFA with the HED network approach, the HED network was trained using the CF image resolution, i.e., an input image resolution of 480×320, see Section 5.10.1. To determine whether the image resolution has an impact on the crack segmentation results, the HED network is also trained using CF scaled to 500×500 pixels (see Section 5.10.4).

It is common to use a data split of around $60\,\%$ of the total dataset for training and $40\,\%$ of the dataset of testing (e.g., Xie and Tu (2017) and Zitnick and Dollár (2015)). If network validation is performed, around $40\,\%$ of the allocated training data (i.e., the training data is around $36\,\%$ of the total, the validation dataset is around $24\,\%$ of the total and the testing dataset is around $40\,\%$ of the total). Validation of the networks is discussed further in Section 5.7.1.

The CF datasets were collected using the same camera for all images and of cracks in the photographs all appear on one type of surface, i.e., asphalt. In contrast, the CSSC

dataset comprises of many different types of images collected from image search engines with a much greater variety of surface texture and lighting conditions, which has the potential for giving increased generalisation when training the deep learning networks. The generalisation of the datasets will be explored in Section 5.10.3 and Section 5.11.

5.6.1 Data preprocessing and augmentation

Data augmentation can also be used to increase the generalisation of the deep learning network. In HED, a pooling layer follows a block of convolution layers (see Section 5.3). These pooling layers allow the addition of some invariance to translation in the input image. However this invariance can be improved further by performing dataset augmentation. Dataset augmentation typically includes the application of some transform, such as rotation, translation or scaling, to the set of input images. Since a crack can appear at different image scales and at any orientation in an image, rotation and scaling (to 0.5 and 1.5 of the input images) transformations were applied. These augmentations effectively increase the input data size by a factor of 32 (Xie and Tu, 2017). Further augmentations may also be relevant (e.g., skew), but have not been applied in this chapter. The augmentations of rotation, translation and scaling were chosen to be consistent with the original HED implementation.

In the HED network, the base architecture is the VGG network (Simonyan and Zisserman, 2014). For VGG, no preprocessing of the images is implemented other than to calculate the mean of the image channels across the dataset, i.e., calculating the mean red channel value, the mean green channel value and the mean blue channel value for each image in the training set and then averaged across the dataset. Once the mean value of each channel is calculated it is subtracted from each of the images in the training set. Mean subtraction is required to centre the dataset in order to ensure the model converges to a solution (Simonyan and Zisserman, 2014).

The data augmentation and mean subtraction steps implemented in the original HED implementation were also followed in this work. The training data was rotated and rescaled to 0.5 and 1.5 times the original image size. Following the methodology of Simonyan and Zisserman (2014) and Xie and Tu (2017), the mean of each of the image channels was

calculated for the augmented dataset, the average of which was subtracted from the augmented dataset. Further research may consider the affect of using alternative methods for normalising the input data, such as using an image range of 0–1.

5.7 Training the Networks

The network architectures described in this chapter were implemented using a publicly available deep learning framework called Caffe (Chu et al., 2013). The networks were trained using the GPU nodes on the advanced research computing (ARC3)—part of the High Performance Computing facility at the University of Leeds[1]—and tested on a desktop PC with a NVIDIA Titan X GPU. In order to train the deep learning networks on ARC3, a Singularity (Kurtzer et al., 2017) image was created containing the same build of Caffe as used in the public implementation of HED [2].

5.7.1 Parameters for training the networks

An optimisation approach is required to find the set of weights in the deep learning networks that best reduce the loss-functions described in Section 5.3. One of the most common optimisation approaches in deep learning is stochastic gradient descent (SGD), which is a variation of the gradient descent algorithm.

Generally, the value of the cost function for the network is computed over the set of training samples. It is desirable to increase the amount of training data to increase the generalisation of the model, but as the amount of training data is increased, the cost of computing the gradient over the whole training set becomes prohibitively large (Goodfellow et al., 2015). Instead, SGD calculates the gradient over a reduced number of samples from the training set, and can vary from only a single training example to a few hundred.

In HED, and other deep learning approaches, the batch-size for SGD is one training sample (Xie and Tu, 2017). Using a larger batch-size may give a better estimate of the

[1] https://arc.leeds.ac.uk/systems/arc3/ (Date accessed 01/02/19)
[2] https://github.com/s9xie/hed (Date last accessed:01/02/19)

gradient for optimisation, but is inefficient for large dataset sizes. In addition, calculating the gradient using a batch-size of a single sample can increase the generalisation of the model, perhaps since bias between samples is removed (Goodfellow et al., 2015).

For stochastic gradient descent, there are several training parameters that can be modified in order to ensure the model converges to a solution; for example, SGD depends greatly on a parameter called learning rate. When using a batch-size smaller that the training dataset size (i.e., mini-batch stochastic gradient descent), it is recommended that the learning rate should be decreased over time (Goodfellow et al., 2015). The learning rate is typically chosen through trial and error or by monitoring the error curves (Goodfellow et al., 2015; Zitnick and Dollár, 2015). A term called 'momentum' is added to SGD to allow accelerated learning for noisy data, which weighs the gradients obtained by SGD with a preference towards the direction of the previously computed gradient.

The authors of HED (Xie and Tu, 2017) state that the parameter choice for low level edge detection requires more care than for image classification tasks, with the risk of the network not converging. In order to choose the parameters for training, the authors of HED follow the method outlined in Zitnick and Dollár (2015), which involves trial and error tuning of the initial parameters by observing the effect on the precision-recall curves on a validation dataset. In HED, the model weights were initialised using the weights from a previously trained model (VGG16). For the HED network, a publicly available implementation exists [2], which includes the initial VGG model for fine-tuning and all details about model structures and parameters.

To obtain the initial parameters for SGD, a validation dataset was selected which consisted around $24\,\%$ of the CF dataset. The model was tested on this validation dataset periodically during the training process. The model was scored for the chosen initial parameters throughout the training process. These parameters were then varied and training was repeated. The selected initial parameters were the ones that gave the best result in terms of F_1 score and training loss on the validation dataset.

The main parameters to vary are: the base learning rate, the maximum number of iterations, momentum, weight decay and step-size. The trial and error process using validation

curves to obtain the parameters used in this chapter is given Appendix B and the final values are shown in Table 5.3.

Table 5.3: The parameters for fine tuning HED with stochastic gradient descent.

Parameter	Value for training HED
learning rate	$1e-08$
momentum	0.9
weight decay	0.0002
training iterations	20000
learning rate step	10000

5.8 Methods for testing and evaluating the networks

The literature review in Section 5.1 showed that there is little agreement in the current methods for evaluating the performance of a crack detection algorithms. Various options will be explored in this chapter, with inspiration taken from the fields of medical segmentation and edge detection.

5.8.1 Pixel-based evaluation

In order to asses the performance of a particular approach, some metrics for evaluation are required. The output of the HED network is a grey-level image with pixels in the range of 0–1, where each pixel of the output image contains a probability that the corresponding pixel in the input test image contains the feature that was being learned (e.g., the probability that the pixel contains a crack). These outputs were evaluated by comparison to the corresponding pixels in the ground truth image. The performance of an image classification algorithm is usually determined by first calculating the following metrics:

- True Positives (TP): when a test correctly classifies, for a given probability threshold, a pixel as being a crack. For example, there is a crack in the pixel where the model says there is a crack.
- True Negatives (TN): when a test correctly classifies, for a given probability threshold, a pixel as not being a crack. For example, there is not a crack in the pixel where the model says there is not a crack.
- False Positives (FP): when a test incorrectly classifies, for a given probability threshold, a pixel as being a crack. For example, there is not a crack in the pixel where the model says there is a crack.
- False Negatives (FN): when a test incorrectly classifies, for a given probability threshold, a pixel as not being a crack. For example, there is a crack in the pixel where the model says there is not a crack.

In the current literature for crack segmentation and classification, these values are inconsistently reported, with some researchers reporting the average results across the test dataset (e.g., L. Yang et al. (2017)) and some reporting the maximum value for the test dataset (e.g., Vu Dung and Duc Anh (2019)). In addition, some authors used a boundary-based evaluation of the crack (e.g. Shi et al. (2016)) or by selecting an image threshold to convert the probability map into a binary image and removing pixels with a probability below the chosen threshold, but with no justification for the choice of image threshold (e.g., Shi et al. (2016)).

However, other fields of research, including object detection and medical segmentation research, precision and recall are typically calculated for multiple thresholds of the output probability map to obtain multiple binary images. Again the image pixels for each new image are compared to the corresponding pixels in the ground truth image. A recall operator characteristic curve (ROC) or precision-recall curve can then be generated by plotting the true positive rates against the false positive rates or the precision values against the recall values, respectively, for all images. The ROC curve has many reported benefits for algorithm evaluation (Flach, 2015; Davis and Goadrich, 2006). For example, the area under the curve (AUC) of the ROC can be used as a measure of accuracy of the test, and the major diagonal from 0–1 of the ROC curve shows the line of random performance, which can be used as a benchmark for algorithmic performance (Flach, 2015).

In order to choose the parameters for evaluating the model or algorithm, the classification task at hand requires some thought. For example, for the image segmentation task of classifying the number of tumours in an image, the penalty for false negatives should be high because to miss a tumour could have fatal consequences. In this scenario, it may be favourable to allow for an algorithm with an increased rate of false positives if the number of false negatives are minimised. To represent the above values graphically, the following metrics can be calculated:

- Sensitivity, True Positive Rate (TPR) or Recall $= \frac{TP}{TP+FN}$
- Specificity or False Positive Rate (FPR) $= \frac{TN}{TN+FP}$

- Precision $= \frac{TP}{TP+FP}$
- Accuracy $= \frac{TP+TN}{TP+FP+TN+FN}$

However, the data used for training and testing the segmentation of cracks in images is biased by the number of true negatives in the dataset; the background pixels were correctly classified as not containing cracks, but this is not necessarily useful information. Therefore, the results shown in the ROC curve are also biased since the ROC curve uses the number of true positives in calculating the false positive rate (FPR) and are not a suitable representation of the performance of the algorithm or method (Flach, 2015; Davis and Goadrich, 2006). This bias is also present in criteria such as overall accuracy, which is commonly reported in the literature for crack classification and segmentation (e.g., Prasanna et al. (2016), Y.-J. Cha et al. (2017), and L. Yang et al. (2017)).

One approach for addressing the imbalance caused by the large number of true negative pixels is to discount them completely, i.e., to use the metrics which do not rely on this value, such as precision and recall. The precision-recall curves do not have the same aforementioned benefits as the ROC curve, such as the AUC metric representing a measure of accuracy (Flach, 2015). However, the precision and recall can be combined into a metric named the F_β score, which is defined as Van Rijsbergen (1979):

$$F_\beta = \frac{(1+\beta^2)Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{5.5}$$

where β determines the preference of the precision and recall. Typically, equal weighting is given to the precision and the recall (Flach, 2015), i.e., $\beta = 1$ and F_1 is the harmonic mean of the precision and recall. A single F_1 value can then be obtained for the test dataset as follows:

$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{5.6}$$

5.8.2 Structure-based evaluation

Pixel level evaluations are not necessarily the most useful evaluation measure for segmentation of cracks from photographs. As described in Fan et al. (2017), since the common metrics used in segmentation evaluation (i.e., TP, FN, TN, FP, precision and recall) are evaluated at the pixel level they cannot fully capture the structure information from the image and can be affected by errors in the ground truth labels. For crack detection applications, it may be more useful to obtain the overall structure of the crack, rather than every pixel containing a crack, and some approaches (e.g., Aldea and Le Hégarat-Mascle (2015)) focus on removing false alarms, as only cracks over a certain size usually require investigation.

Object boundaries have also been used for algorithm evaluation in semantic segmentation task. Object boundaries can be generated where neighbouring pixels in the ground truth label have different values (e.g., Zitnick and Dollár (2015)). These boundaries often need to be annotated by hand because automated approaches can give poor results for intricate shapes. As a result, there are very few datasets with these ground truth labels that exist for crack segmentation. This data is available for the dataset that accompanies the SFA method (i.e., the Crack Forest dataset), but was not used as an evaluation method in this work as the results could not be compared with other existing crack segmentation datasets. This approach is especially useful for applications, such as segmenting objects an image scene, where there are multiple objects of interest since some of the errors caused by mis-labelling in a per-pixel evaluation are removed. In addition, boundary segmentation datasets may not be as useful for crack segmentation since a crack tends to be a thin line, sometimes with a width of one pixel, which is not well represented using a boundary-based approach.

There may be alternative metrics available in the wider literature that can be used for the evaluation of the deep-learning networks for crack segmentation. For example, Guo et al. (2018) note that pixel evaluation methods do not allow the recovery of the global structure of the retina for medical segmentation tasks and introduce a structure similarity measure (SSIM) based on work by Fan et al. (2017) and Zhou Wang and Alan C. Bovik (2004). Fan et al. (2017) introduce a structure measure (S-measure) as an alternative to precision and

the F_β measures. This measure uses the structural similarity measure, which is used in the image quality assessment field as a measure for success of image de-noising (Zhou Wang and Alan C. Bovik, 2004). SSIM is defined as follows (Zhou Wang and Alan C. Bovik, 2004):

$$ssim = \frac{2\bar{x}\bar{y}}{(\bar{x})^2 + (\bar{y})^2} \cdot \frac{2\sigma_x\sigma_y}{(\sigma_x)^2 + (\sigma_y)^2} \cdot \frac{\sigma_{xy}}{\sigma_x\sigma_y} \tag{5.7}$$

where $x = x_i|i = 1, 2, \ldots, N$ and $y = y_i|i = 1, 2, \ldots, N$ are the pixels belonging to the saliency map and corresponding ground truth; $\bar{x}$, $\bar{y}$ are the corresponding mean values of x and y and σ_x and σ_y are the standard deviations.

Fan et al. (2017) divide the image into sections and use SSIM across different regions of the image and weight the result depending on the amount of the object that are present in the different regions to create a region-aware structural similarity measure:

$$S_r = \sum_{k=1}^{K} w_k \times ssim(k) \tag{5.8}$$

Where K is the number of image blocks, SSIM is the structural similarity measure introduced in Equation 5.7 and w_k is the weighting proportional to the foreground region the block covers in the ground truth. An object-aware measure is also defined, as follows:

$$S_o = \mu x O_{FG} + (1 - \mu) x O_{BG} \tag{5.9}$$

where μ is the ratio between the foreground area (i.e., the crack) in the ground truth image and the total image area. O_{FG} is defined as:

$$O_{FG} = \frac{2\bar{x}_{FG}\bar{y}_{FG}}{(\bar{x}_{FG})^2 + 1 + 2\lambda \times \sigma_{x_{FG}})} \tag{5.10}$$

where $\bar{x}_{FG}$ is the mean of the foreground of the saliency map (i.e., the foreground of the output of the HED network), λ is a constant that balances the ratio of the standard deviation to the mean (assigned as 0.5 by Fan et al. (2017)) and $\sigma_{x_{BG}}$ is the standard deviation of the background of the saliency map. O_{BG} is defined as:

$$O_{BG} = \frac{2\bar{x}_{BG}\bar{y}_{FG}}{(\bar{x}_{BG})^2 + 1 + 2\lambda \times \sigma_{x_{BG}})} \tag{5.11}$$

where $\bar{x}_{BG}$ is the mean of the foreground of the saliency map (i.e., the foreground of the output of the HED network) and $\sigma_{x_{BG}}$ is the standard deviation of the background of the saliency map.

Finally, the region-aware similarity measure and the object-aware similarity measure are combined to give:

$$S = \gamma \times S_o + (1 - \gamma) \times S_r \tag{5.12}$$

where γ is used to weight the region and object structure measures and is set as $\gamma = 0.5$ in Fan et al. (2017). This measure is used in this chapter to assess how much of the structure of the crack has been recovered for the different datasets with respect to the ground truth for different crack segmentation approaches. The S-measure was found to be robust to errors in mis-labelling of the ground truth (Fan et al., 2017).

5.9 Overview of experiments

In this chapter, deep learning approaches for segmentation of cracks from photographs are evaluated. In order to evaluate these approaches, the CF dataset described in Section 5.5.1 is used as a benchmark to compare the HED network with the SFA approach and other approaches from the literature. The CF and CSSC datasets are then evaluated individually and in combination for training and testing the HED network. The datasets referred to as CF_CSSC indicate mixing the CF and CSSC dataset to create a combined dataset of different sizes (70 or 140 for training and 48 and 96 for testing), but with 50 % from each of the original datasets.

A summary of comparisons that are made using the datasets of cracks in this chapter is given in Table 5.4.

Table 5.4: A summary of the experiments performed by the two different crack datasets, with the method that is tested and the number of images that are used for training and testing.

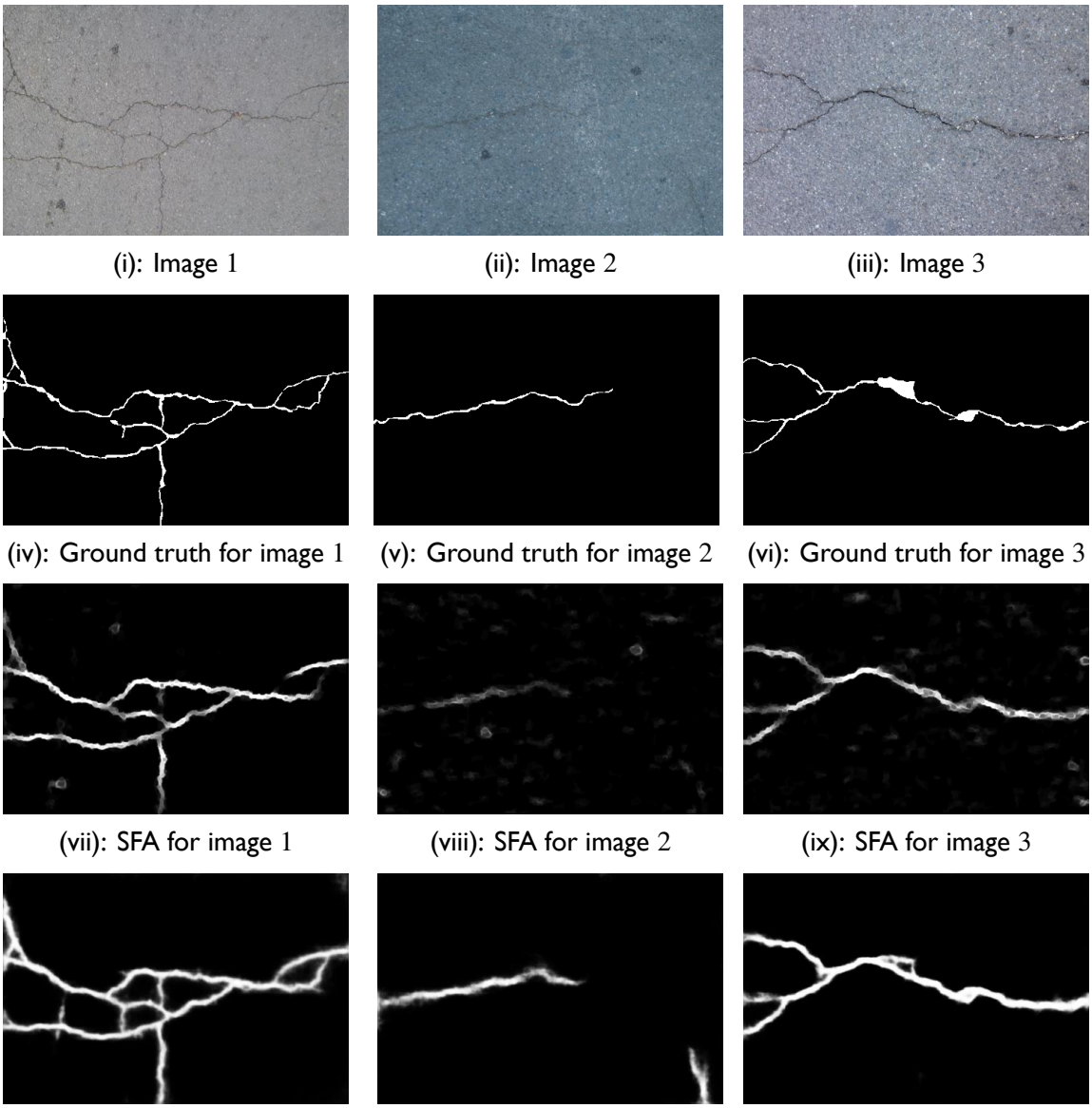
Dataset label	Dataset for training	Dataset for testing	Num. images training /testing	Method	Section training
CF_480_320_Aa	CF	CF	70 / 48	SFA	5.10.1
CF_480_320_Aa	CF	CF	70 / 48	HED	5.10.2
CSSC_70_Ba	CSSC	CF	70 / 48	HED	5.10.3
CSSC_70_Bb	CSSC	CSSC	70 / 48	HED	5.10.3
CF_CSSC_70_Ca	CSSC+CF	CF	70 / 48	HED	5.10.3
CF_CSSC_70_Cb	CSSC+CF	CSSC	70 / 48	HED	5.10.3
CF_CSSC_140_Da	CSSC+CF	CF	140 / 96	HED	5.10.3
CF_CSSC_140_Db	CSSC+CF	CSSC	140 / 96	HED	5.10.3
CF_500_500_Ee	CF (500×500 px)	CF (500×500 px)	70 / 48	HED	5.10.4

5.10 Results and discussion

5.10.1 Comparison of HED and SFA

As a benchmark, the results of SFA were compared to the results of HED. Both implementations were trained using the CF dataset, with 70 images for training and 48 images for testing. After passing the testing data through the trained HED model, the output is a probability map that describes the probability whether a pixel is a crack or not. The initial output of the SFA is also a probability map. However, Shi et al. (2016) also applied post-processing steps, including morphological opening to join crack segments and a threshold to remove pixels below a probability of 0.9, to create a binary image (Shi et al., 2016) (i.e., a pixel containing a crack is represented by a 1 and non-crack pixels are represented by a 0). Since HED does not have a post-processing step, the results of SFA were initially compared to HED without post-processing, see Figures 5.5–5.6. Then, post-processing in the form of the application of different image thresholds and morphological opening followed by thresholding was applied to both SFA and HED (see Figure 5.8).

First, the results for the final layer of HED (referred to henceforth as HED fused) and SFA without post-processing were compared. As described in Section 5.8.1, a single probability map was created by stacking all of the images that are tested on HED or SFA into a single vector and by applying a probability threshold. This process created a single binary map for the whole test dataset and repeated for many probability thresholds. The precision, recall and F_1 values were calculated for each of the binary maps and are plotted as a precision recall curve. The precision-recall curve for the HED network is shown in Figure 5.6 alongside the precision recall curve for SFA. The F_1 score where precision was equal to recall is marked for each method in Figure 5.6 and exemplar probability maps (and the corresponding ground truth images and original images) from each method are given in Figure 5.5. The F_1 score where precision was equal to recall is used as the comparision point in line with traditional computer vision analysis.



(i): Image 1 (ii): Image 2 (iii): Image 3

(iv): Ground truth for image 1 (v): Ground truth for image 2 (vi): Ground truth for image 3

(vii): SFA for image 1 (viii): SFA for image 2 (ix): SFA for image 3

(x): HED for image 1 (xi): HED for image 2 (xii): HED for image 3

Figure 5.5: A comparison of the example images from the CF dataset, with associated ground truth labels (5.5iv–5.5vi) followed by the results of the SFA algorithm and the HED network on the CF dataset. Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks. Pixels containing a crack or not-crack pixel are hand labelled by an annotator in 5.5iv–5.5vi and determined by SFA or HED in 5.5vii–5.5xii.

The example results in Figure 5.5 show that the majority of the cracks were segmented from the test images for both SFA and HED implementations, but more background noise was segmented using SFA than when using HED, although both HED and SFA mis-classify pixels, such as the marks made by oil in Figure 5.5i. A higher amount of background noise is expected for SFA since it is a patch-based approach and cracks appear similar at different image scales (Zhang et al., 2018a). The examples in Figure 5.5viii and Figure 5.5xi show that HED was more successful for faint cracks. Figure 5.5x also highlights a problem when evaluating against a mis-labelled ground truth dataset, since HED segments part of a crack that was not labelled in the ground truth in Figure 5.5iv, but is faintly visible in the original image in Figure 5.5i. Similarly, there is discrepancy for the per-pixel evaluation for the ground-truth in Figure 5.5vi and the result of HED in Figure 5.5xii, where the ground truth label has a block of pixels, but the actual crack in Figure 5.5iii is a more subtle line. Further cause for error between the segmented image and the ground truth label is the thickness of the cracks that are segmented by both SFA and HED, since these pixels will be regarded as false positives in the per-pixel evaluation.

For all probability thresholds applied to the SFA and HED results, the precision is higher for HED than SFA for the associated recall value in Figure 5.6. Hence, for all pixels where HED and SFA reported some probability that a crack was present, the pixels returned by HED were more likely to be a crack. The F_1 score for the probability threshold at which precision is equal to recall is 0.546 for SFA and 0.638 for HED (also marked on Figure 5.6). A different threshold can be chosen to meet some other criteria at the expense of reducing either recall or precision. The authors of SFA (Shi et al., 2016) chose a probability threshold of 0.9, the equivalent precision and recall values for this threshold are also marked in Figure 5.6 and shown qualitatively in Figure 5.7.

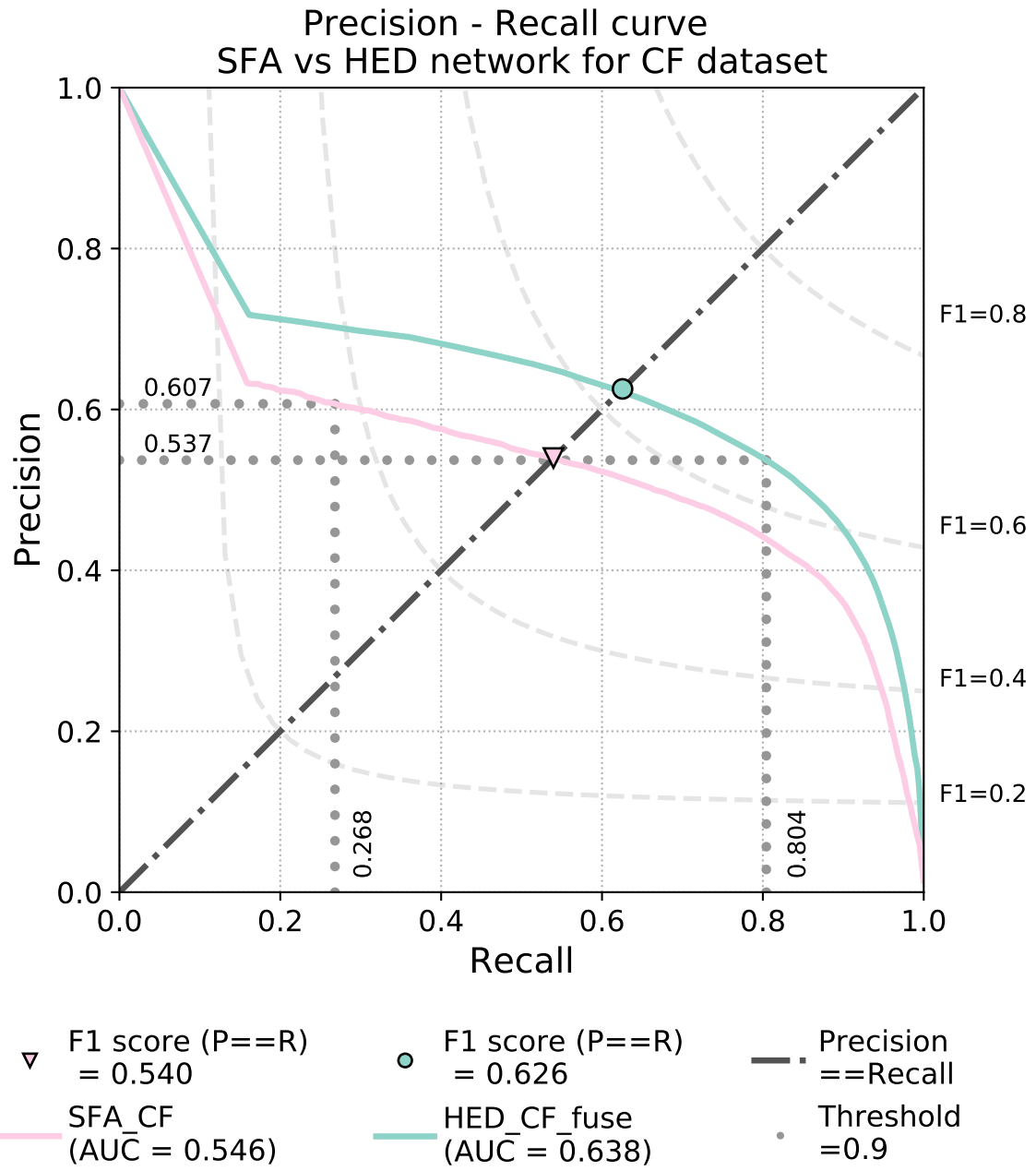


Figure 5.6: Comparison of the precision-recall values for the results of SFA and the fused output (final layer) of the HED network. The F_1 score where the precision is equal to the recall are marked for each method along with the corresponding probability threshold, and the values are given in the legend.

For SFA, using a probability threshold of 0.9 gave a much lower recall of 0.287, but a precision of 0.607, so that fewer pixels were being reported as cracks, but those that were reported were more likely to be cracks. In contrast, for HED, using a probability threshold of 0.9 gave a higher recall of 0.804, but a lower precision of 0.537, so that more pixels were classified as cracks, but the number of false positives increased. Aldea and Le Hégarat-Mascle (2015) reported that it may be desirable to have a higher precision than recall to reduce the number of false positives that are detected when performing crack segmentation from images, since only pixels containing a high probability of crack are returned. A precision-recall curve, such as the one in Figure 5.6, can be used to determine the trade-off between precision and recall and the corresponding probability threshold can be obtained from the raw data used to create the precision-recall curve. This threshold value can then be applied in order to achieve desirable results when implementing the classifier on a new test dataset.

To show the effect of using different probability thresholds on the qualitative results, a threshold of 0.604 was applied to the probability maps from SFA and a threshold of 0.957 was applied to the probability maps from HED, (i.e., the thresholds that were applied to the probability map to give the F_1 score where precision is equal to recall for SFA and HED, respectively in Figure 5.6) and the threshold of 0.9 was applied to the probability maps from both SFA and HED. Exemplar binary images resulting from different probability thresholds are shown in Figure 5.7 and correspond to the original probability maps shown in Figure 5.5.

By comparing the resulting binary maps for the different thresholds in Figure 5.7, the effect of increasing and decreasing recall in Figure 5.6 can be seen. As expected, the crack in Figure 5.7iv is much more pronounced than in Figure 5.7vii where only pixels above a probability of 0.9 remain. For the same threshold for HED, a more complete crack was returned than for SFA in all instances (comparing Figure5.7vii–5.7ix and Figures 5.7xiii–5.7xv), although both SFA and HED perform poorly for the cracks corresponding to the ground truth label in Figure 5.7ii.

In order to improve the results of SFA, (Shi et al., 2016) applied post-processing steps using morphological opening where a 4×4 filter is passed over the probability map to

remove small areas of noise and to join gaps between cracks before applying a threshold of 0.9 to the post-processed probability map. To compare the effect of this approach on the resulting binary maps, post-processing was applied to both SFA and HED and the results are shown in Figure 5.8. However, rather than applying a single threshold of 0.9, the thresholds of 0.604, 0.957 and 0.9 were applied to SFA, HED and both methods (see Figure 5.8).

As expected, comparing the results in Figure 5.7 and Figure 5.8, shows that by applying the morphological opening, less pixels containing cracks were removed when applying the thresholds for each approach when compared to just thresholding in Figure 5.7. Again, changing the threshold affects the number of pixels that were kept. For HED, applying the threshold of 0.959 (obtained from the F_1 score where precision was equal to recall in Figure 5.6) gave a much thicker crack than for the equivalent threshold for SFA (compare Figures 5.8iv – 5.8vi and 5.8x – 5.8xii). However, this increase in thickness comes at a loss of subtle details, such as the gaps in some of the cracks (compare Figure 5.8xii and 5.8xv).

(i): Ground truth for image 1 (ii): Ground truth for image 2 (iii): Ground truth for image 3

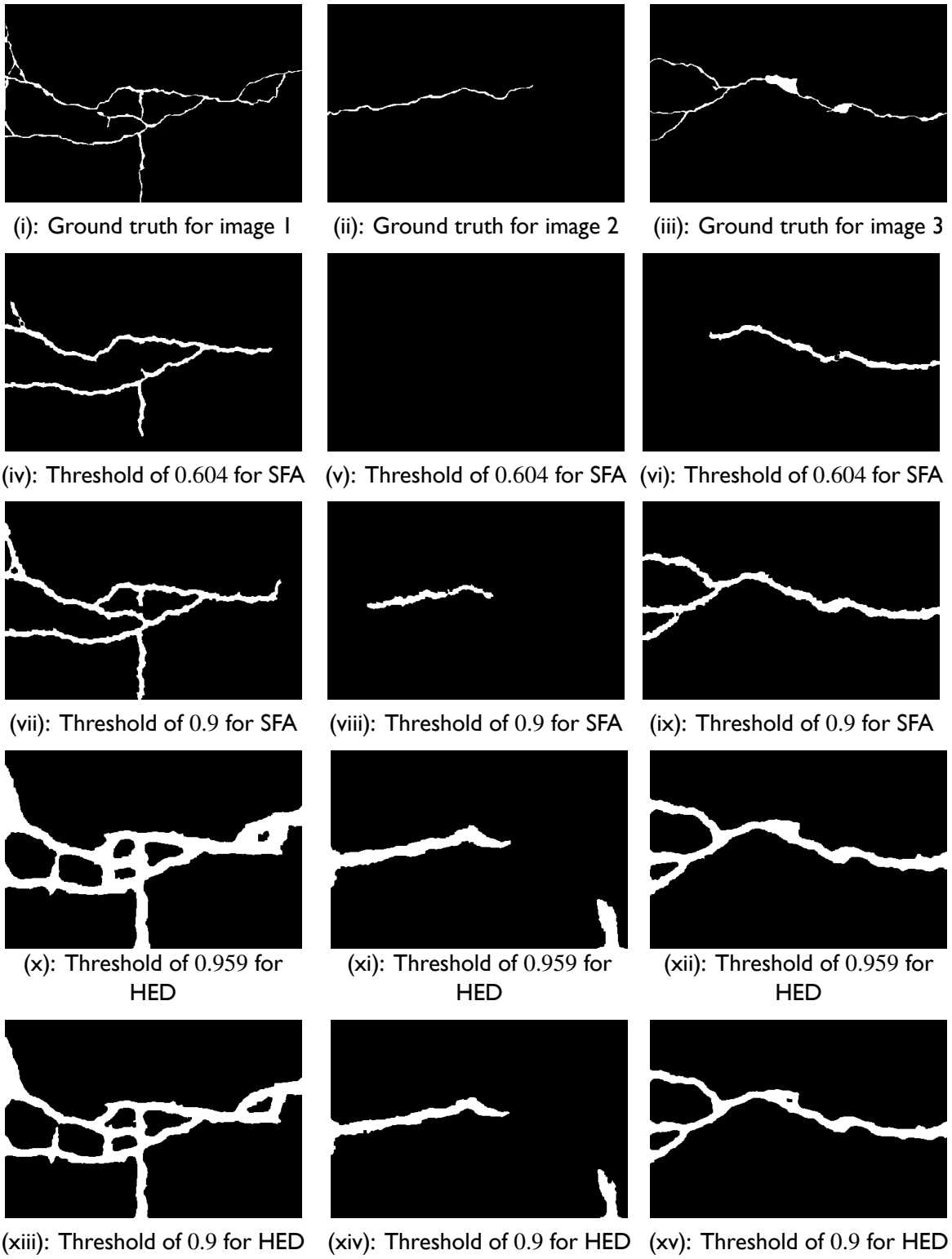
(iv): Threshold of 0.604 for SFA (v): Threshold of 0.604 for SFA (vi): Threshold of 0.604 for SFA

(vii): Threshold of 0.9 for SFA (viii): Threshold of 0.9 for SFA (ix): Threshold of 0.9 for SFA

(x): Threshold of 0.959 for HED (xi): Threshold of 0.959 for HED (xii): Threshold of 0.959 for HED

(xiii): Threshold of 0.9 for HED (xiv): Threshold of 0.9 for HED (xv): Threshold of 0.9 for HED

Figure 5.7: Qualitative results of SFA and HED on three images from the test dataset when different thresholds are applied. A threshold of 0.604 and 0.959 are applied to SFA and HED respectively, A threshold of 0.9 is also applied to match the threshold applied in SFA by (Shi et al., 2016). Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.



(i): Ground truth for image 1 (ii): Ground truth for image 2 (iii): Ground truth for image 3

(iv): Threshold of 0.604 for SFA (v): Threshold of 0.604 for SFA (vi): Threshold of 0.604 for SFA

(vii): Threshold of 0.9 for SFA (viii): Threshold of 0.9 for SFA (ix): Threshold of 0.9 for SFA

(x): Threshold of 0.959 for HED (xi): Threshold of 0.959 for HED (xii): Threshold of 0.959 for HED

(xiii): Threshold of 0.9 for HED (xiv): Threshold of 0.9 for HED (xv): Threshold of 0.9 for HED

Figure 5.8: Qualitative results of SFA and HED on three images from the CF dataset when applying the post-processing step of morphological opening and the thresholds of 0.604 and 0.959 and 0.9 . Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

Next, an alternative evaluation method is considered: the SSIM and S-measure approaches described in Section 5.8.2. These values are plotted for the original SFA and HED implementations as well as the thresholded values and post-processed and thresholded values in Figure 5.9 and Figure 5.10.

The S-measure and SSIM was calculated on a per-image basis for the test dataset results and averaged over the dataset, whereas the F_1 score was obtained for the test dataset results as a whole. Figure 5.9 and Figure 5.10 show that on the whole, the F_1 score has a lower value than S-measure and SSIM, but from a ranking perspective the difference between the F_1 and S-measure is similar for all approaches. In addition, although the application of the post-processing steps in Figure 5.8 gave a better visual appearance of the cracks, the overall F_1 score did not improve and there appears to be a decrease for the values of SSIM and S-measure, although this reduction was minimal for SSIM on the SFA approach. In general, the SSIM had a value over 0.8 for all methods and image thresholds, which shows that all methods gave low amounts of background texture, but otherwise is not useful for discerning between approaches.

For HED, the S-measure and F_1 score decreased when the image was post-processed. This reduction is most probably because the segmented crack is much thicker than the ground truth, as discussed for the qualitative results in Figure 5.8. For SFA, this effect is less prominent, since there are fewer pixels to which the morphological opening is applied. An alternative threshold could be applied to minimise this effect, with the aim of maximising the S-measure after pre-processing rather than the F_1 score before pre-processing.

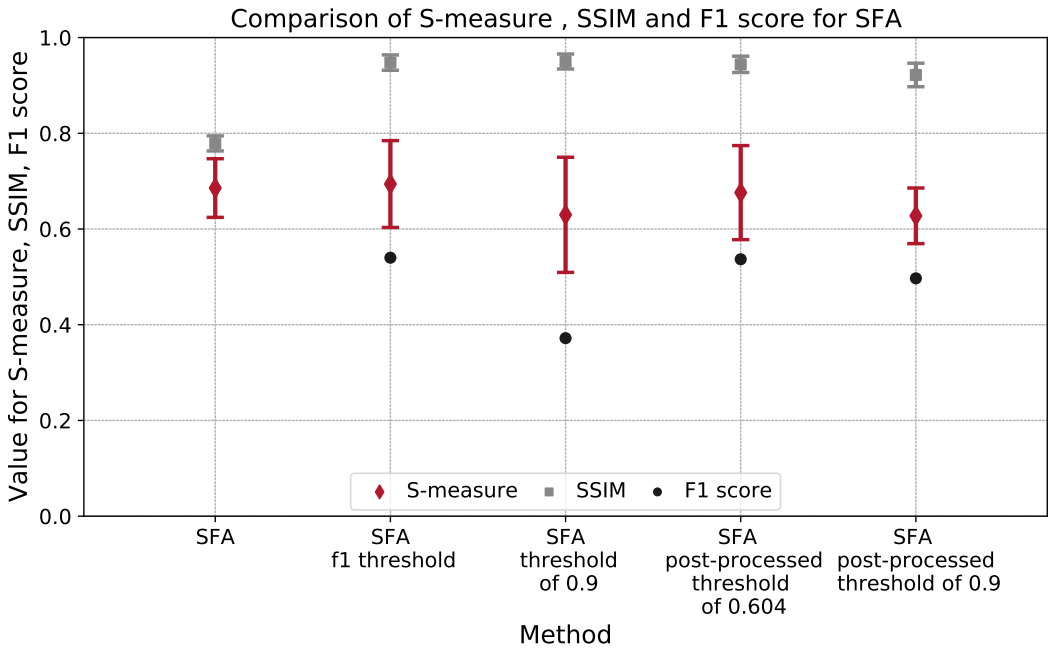


Figure 5.9: A comparison of SSIM, S-measure and F_1 score for the original SFA implementation, and SFA after a thresholding step or post-processing has been applied to the results.

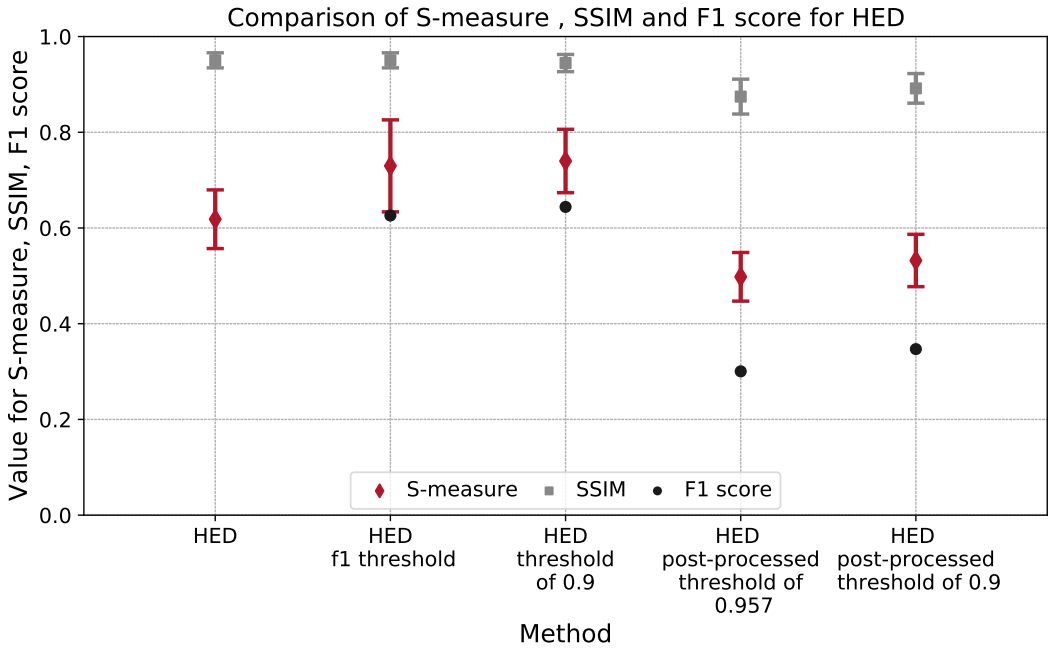


Figure 5.10: A comparison of SSIM, S-measure and F_1 score for HED for the original HED implementation, and HED after a thresholding step or post-processing has been applied to the results.

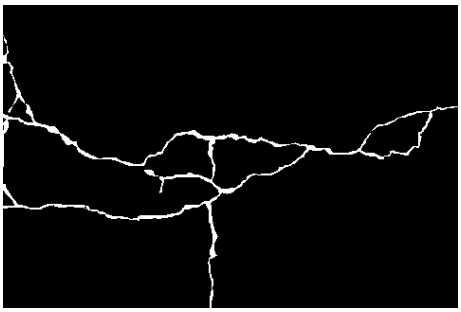
5.10.2 Evaluating the side-outputs of HED

As described in Section 5.3, the HED network has outputs (also referred to as side-outputs) for each set of layers in the network. In this section, the side-outputs are compared to evaluate whether a different configuration of HED could be used for an improved segmentation of cracks, since different details of the cracks are extracted at each side-output (see Figure 5.11). Each of the output layers follows a naming convention of deeply supervised network layer (DSN) from 1–5, which reflects the process of calculating the loss for each of the outputs, of which there are five, and using this loss to train the network. The final output fuses the five output layers and is henceforth referred to as 'fused'. For the same HED network as Section 5.10.1, an example of the qualitative results for the side-outputs for a sample image from the test dataset are shown in Figure 5.11, alongside the original image and corresponding ground truth label. The network is trained and tested on the CF dataset.

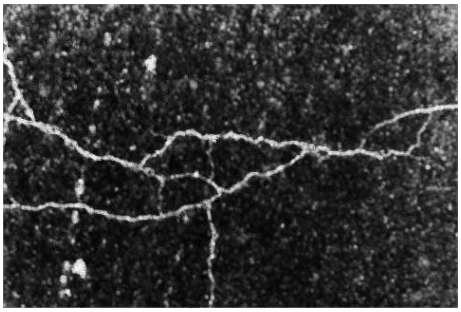
In Figure 5.11, the side-outputs of HED show segmentation of the cracks at different image scales. In Figure 5.11iii, the thickness of the segmented crack appears most similar to the ground truth label, but there is the greatest amount of background noise, when compared to the other side-outputs. Each of the side-outputs from DSN 1–5 shows an increase in the thickness of the segmented crack and a decrease in the amount of background noise. Figure 5.11viii shows the fused result, where the background noise has been removed, but the crack is also thinner than in Figure 5.11vii.

The precision-recall curve is plotted in Figure 5.12, for each of the side-outputs layers and fused layer and the F_1 score where precision is equal to recall is marked. The fused output layer has the greatest precision for all values of recall. The first side-output (DSN1) and the fifth side-output (DSN5) has the lowest F_1 scores for all of the layers. DSN1 probably has a low F_1 score since its output has the greatest amount of background noise, visible in the example output of Figure 5.11iii, and DSN5 may has a low score due to the thickness of the segmented crack, also visible in the example output of Figure 5.11vii, which increases the number of false positive when compared to the ground truth.

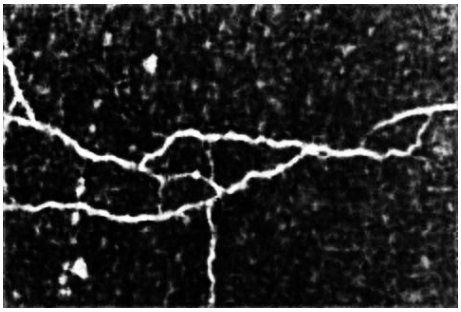
(i): RGB Image 2 of CF dataset



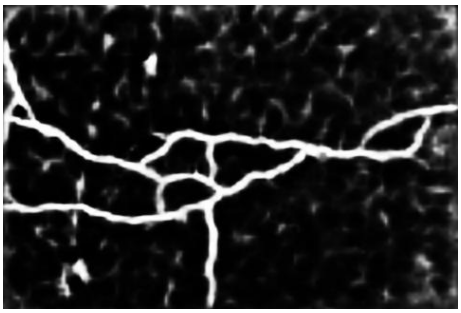
(ii): Ground truth of image 2 of CF dataset



(iii): Result of side output 1 (dsn1) of HED

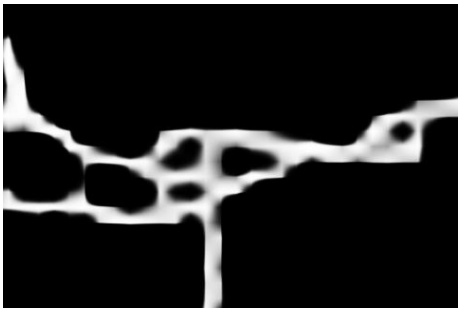


(iv): Result of side output 2 (dsn2) of HED



(v): Result of side output 3 (dsn3) of HED

(vi): Result of side output 4 (dsn4) of HED



(vii): Result of side output 5 (dsn5) of HED

(viii): Result of fused output of HED

Figure 5.11: The qualitative results of the side-outputs of the HED network on the CF dataset. An example image from the CF dataset is shown in 5.11i and the corresponding ground truth label is shown in 5.11ii. Each of the five side outputs is shown in 5.11iii–5.11vii and the fused result is shown in 5.11viii. Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

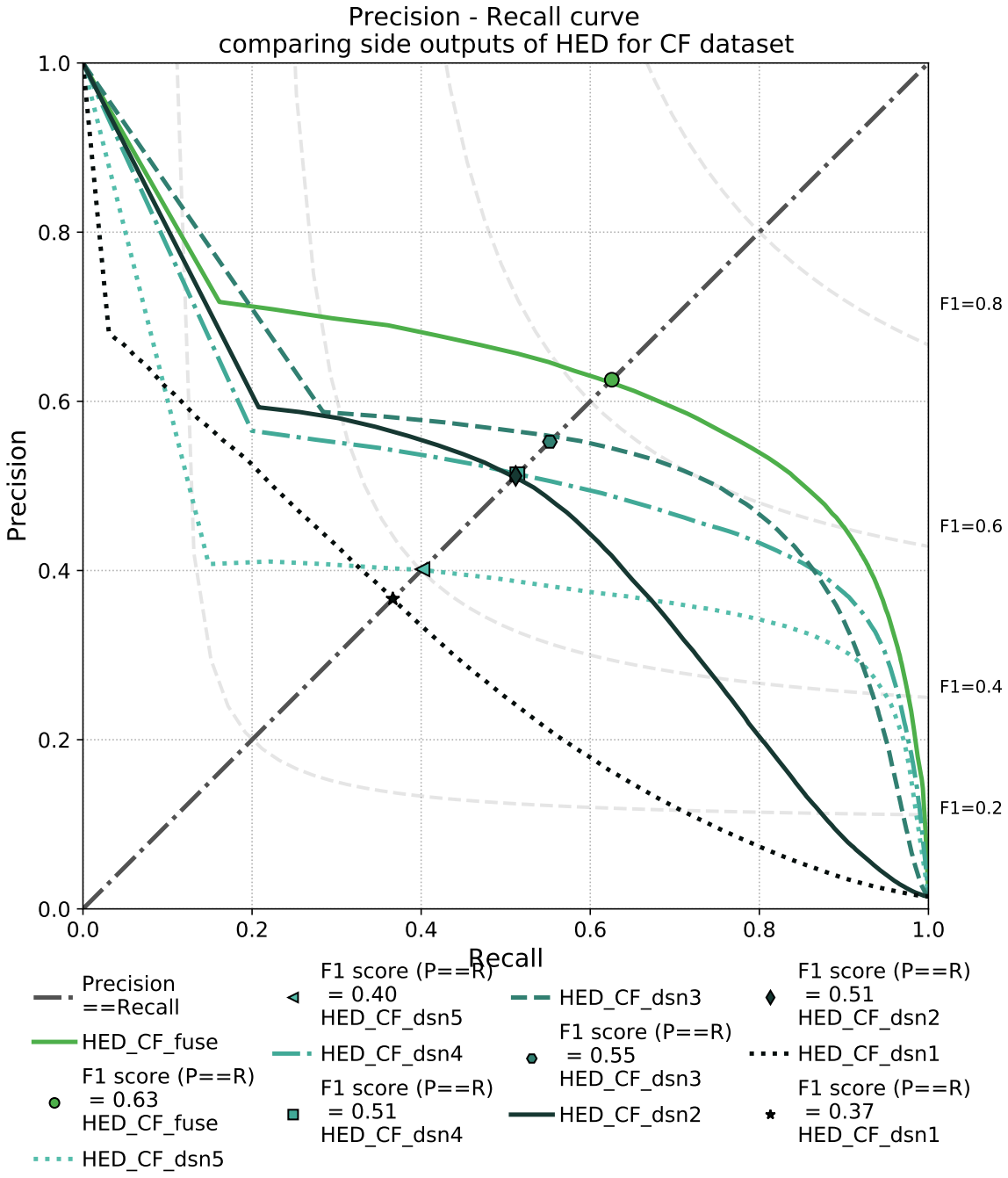


Figure 5.12: A comparison of the precision-recall curves for the side-outputs of the HED network (labelled DSN–5 and fuse). The F_1 score where the precision is equal to the recall is marked for each of the side-outputs and the values given in the legend.

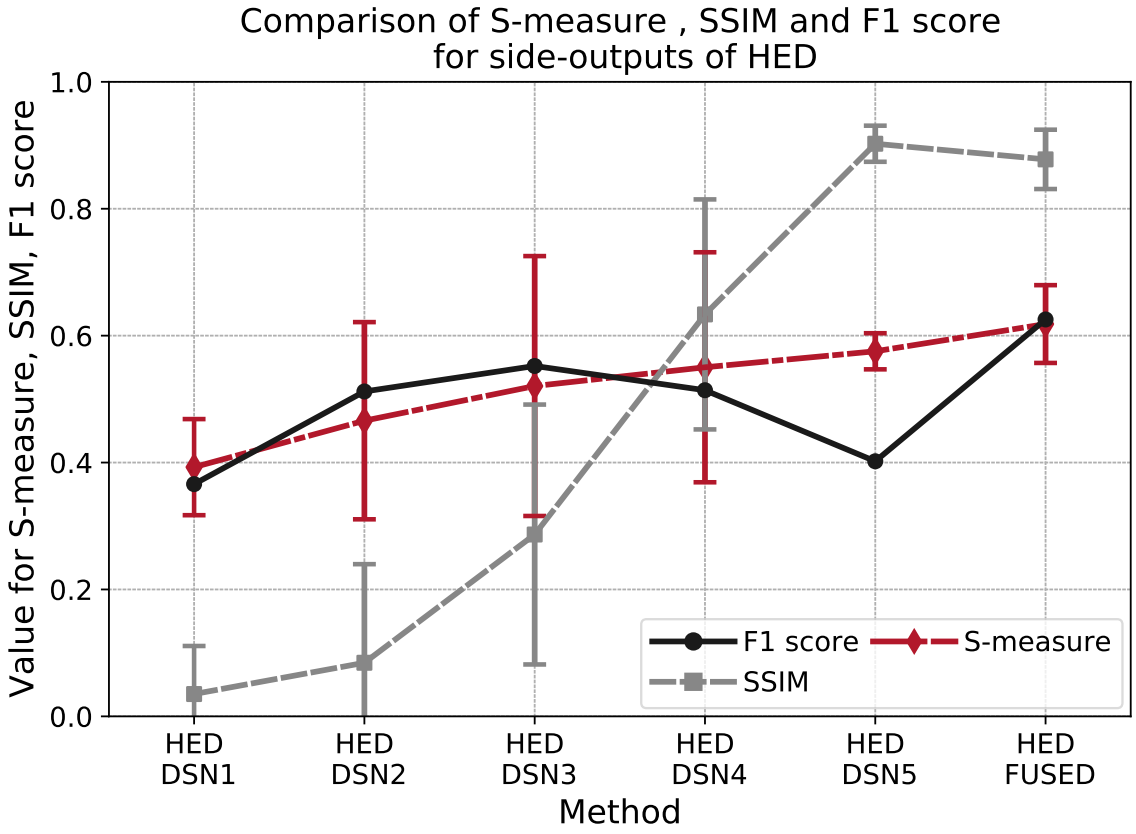


Figure 5.13: A comparison of SSIM, S-measure and F_1 score for the side-outputs of HED.

In addition to the precision-recall curve, the averaged structure measures (SSIM and S-measure) are also plotted in Figure 5.13, alongside the associated F_1 score. Figure 5.13 shows that the average S-measure increases almost linearly as the scale of the image features from the side-outputs increases. The F_1 score is within the error bounds of the S-measure for all outputs except for side-output 5, where the crack is much thicker than the corresponding ground truth. The overlap between F_1 score and S-measure suggests that the ranking of these metrics when benchmarking different methods may be similar. The standard deviation is greater for DSN2–DSN4 for both the S-measure and SSIM. The SSIM changes considerably between the side-outputs, which suggests it is more affected by background texture, since it varied only slightly for the fused outputs of SFA and HED in Figure 5.9 and 5.10 when the majority of background noise had been removed. The S-measure gives the best performance for the fused output, where the F_1 score is also highest. In ad-

dition, the S-measure also appears to be less variable than F_1 score, which suggests that it is more robust to the change in thickness of the crack and also mis-labelling of the ground truth (Fan et al., 2017). It is recommended for future work that the S-measure be used in comparison with the F_1 score for ranking different approaches and datasets to understand the correlation between S-measure and F_1 score and to determine whether S-measure is a better metric for evaluating crack segmentation algorithms.

5.10.3 Cross-dataset testing

To evaluate the effect of the dataset on the results of HED network, the HED network was trained on three additional datasets: the CSSC dataset, a 50:50 split of CF and CSSC with a total of 70 images (henceforth referred to as CF_CSSC_70_Ca or CF_CSSC_70_Cb for testing on the CF or CSSC dataset, respectively) and a 50:50 split of CF and CSSC with a total of 140 images (henceforth referred to as CF_CSSC_140_Da or CF_CSSC_140_Db for testing on the CF or CSSC dataset, respectively). The HED networks trained on these different datasets were then tested on the CF dataset and CSSC dataset. Reference examples of the CF and CSSC dataset are given in Figure 5.14. Qualitative results for cross-dataset testing are given in Figure 5.15 and Figure 5.17 and the precision-recall curves are shown in Figure 5.16 and Figure 5.18.

The qualitative results in Figure 5.15 and Figure 5.17 show that training the HED network on the different crack datasets returned positive instances of segmentation of cracks when applied to the test images. However, when testing CSSC_70 on the CF dataset (i.e, CSSC_70_Ba), the crack in Figure 5.15v was not segmented. Conversely, when testing CF_70 (i.e, CSSC_70_Bb) on CSSC the edge at the intersection of a wall and a floor in Figure 5.14ix is incorrectly segmented as a crack in Figure 5.17iii. This example, shows that the CF dataset is too focused on cracks, i.e., there are no (or too few) examples in the training dataset of lines other than cracks from which the model can learn.

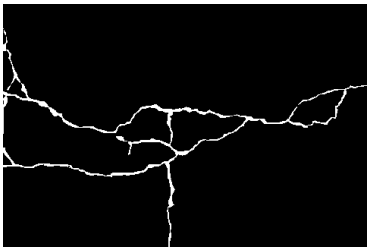
In general, the combination of the two datasets for training increased the amount of background noise that was present in the resulting probability maps. The amount of background noise decreased as the size of the training data increased from 70 images to 140

images, but was still worse than when using only CF or only CSSC. For example, the image Figure 5.17ix shows more of the wall from Figure 5.14ix than in Figure 5.17xii. This result suggests that there may be overfitting of the model when trained only on CSSC or only on CF.

(i): Image 1 of CF dataset

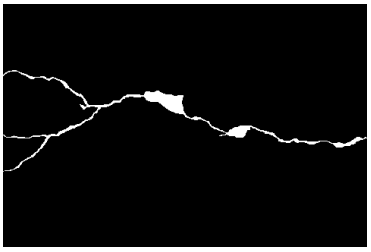
(ii): Image 2 of CF dataset

(iii): Image 3 of CF dataset



(iv): Ground truth for CF image 1

(v): Ground truth for CF image 2

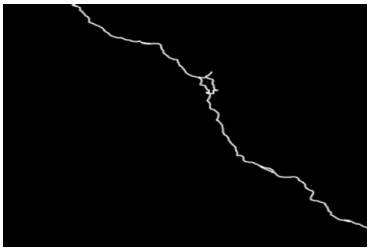


(vi): Ground truth for CF image 3

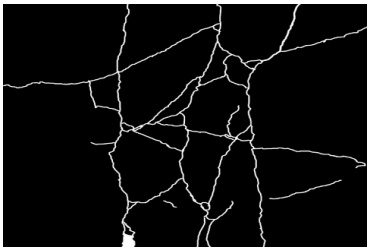
(vii): Image 1 of the CSSC dataset

(viii): Image 2 of the CSSC dataset

(ix): Image 3 of the CSSC dataset



(x): Ground truth for image 1 of the CSSC dataset



(xi): Ground truth for image 2 of the CSSC dataset

(xii): Ground truth for image 3 of the CSSC dataset

Figure 5.14: Reference RGB and ground truth labels of data from the CF dataset (5.14i–5.14vi) and CSSC dataset (5.14vii–5.14xii).Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

In Figure 5.16, the network that had the lowest performance was the CSSC_70_Ba dataset and in Figure 5.18 the network with the lowest performance was the CF_70_Ab dataset. This result is expected, since no data from CF was used to train the CSSC_70 network and no data from CSSC was used to train the CF_70 network: the networks have not been exposed to any data from the opposing datasets during training. Comparing CSSC_70 and CF_70 in Figures 5.15 and 5.17, shows that CSSC_70 has a slightly higher performance when tested on CF (with an F_1 score of 0.567 in Figure 5.16) than the CF_70 network when tested on CSSC (i.e., CF_70_Ab, with an F_1 score of 0.534 Figure 5.18). This result suggests that the CSSC dataset allows better generalisation than the CF dataset to unseen test data, which is expected due to the greater diversity in the training data (e.g., different types of cracks and background texture).

All of the trained networks in Figure 5.18 had a worse performance than in Figure 5.16. This result is likely caused by a larger variation in the types of cracks in the CSSC dataset compared to the CF dataset. Mixing CSSC and CF datasets may give a slight increase in performance for precision at low recall which increased further slightly when the size of the dataset was doubled from CF_CSSC_70_Ca to CF_CSSC_140_Da, see Figure 5.16. However, the F_1 score was not improved when testing on the CF dataset or CSSC dataset.

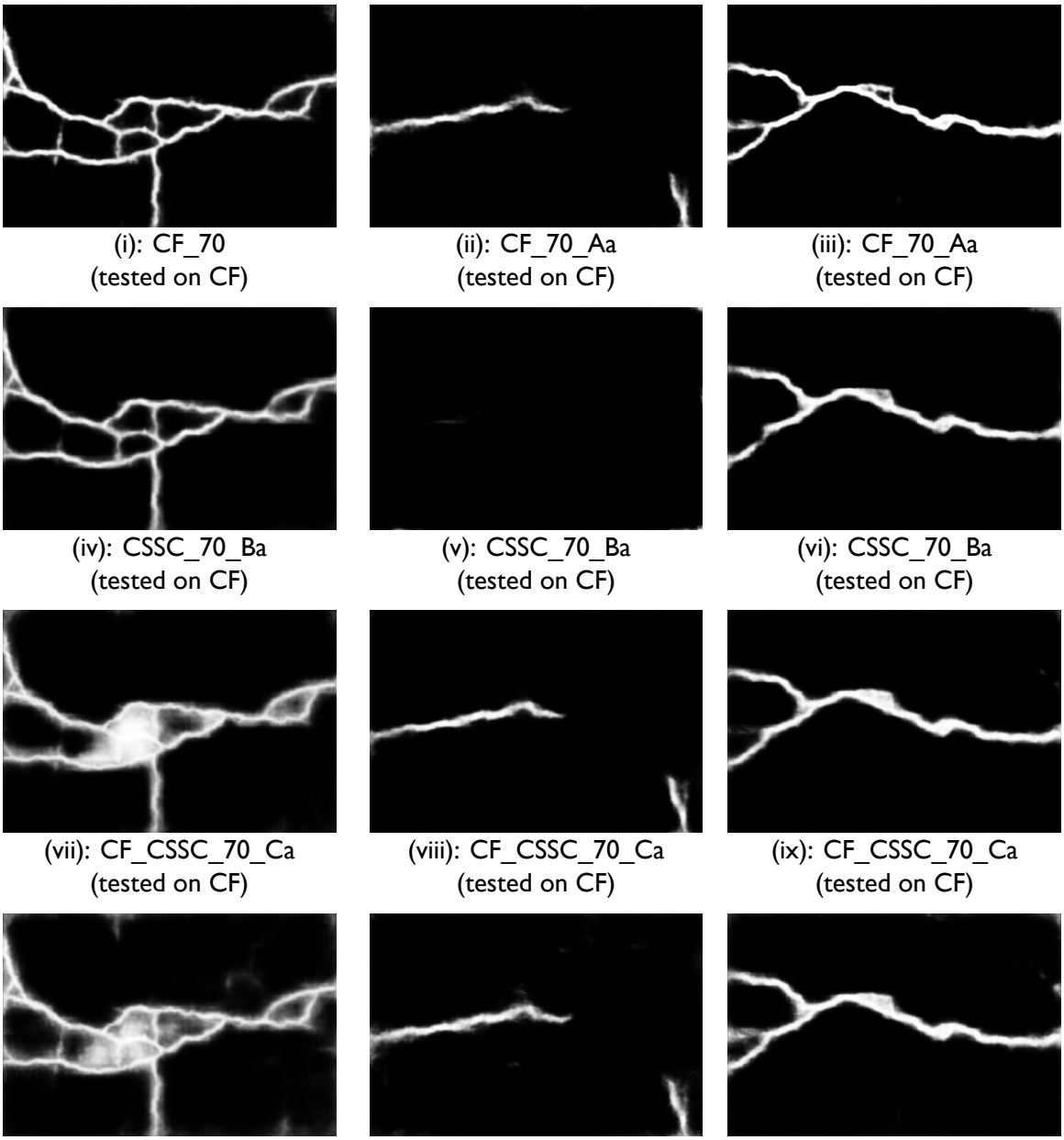


Figure 5.15: A qualitative comparison of the results from different training datasets. The CF_CSSC_70 dataset consists of 35 images from both the CF and CSSC datasets. The CF_CSSC_140 dataset consists of 75 images from both the CF and CSSC datasets. The CF_70 dataset consists of 70 images from the CF dataset. The CSSC_70 dataset consists of 70 images from the CSSC dataset. All networks were tested on the CF dataset. Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

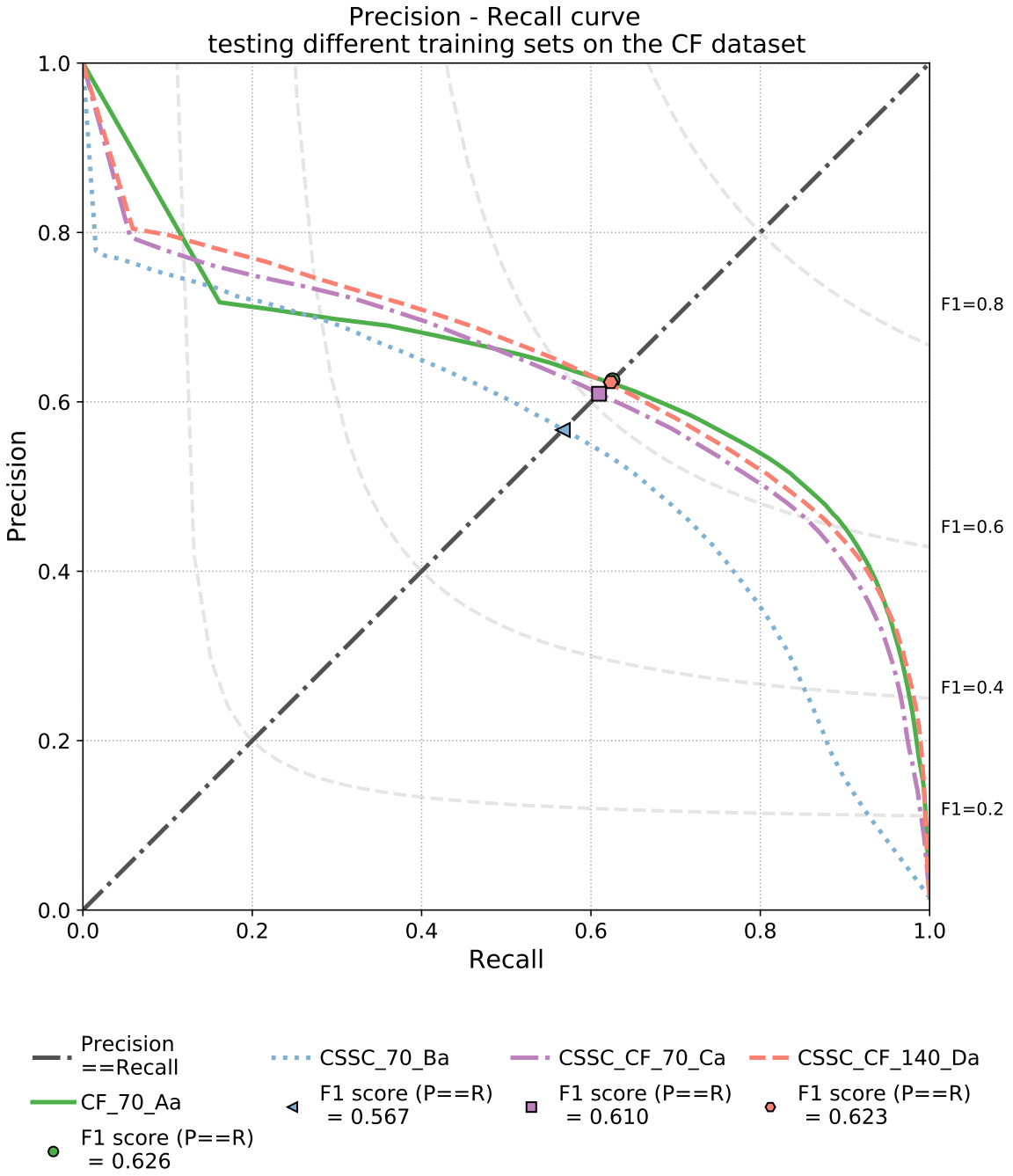
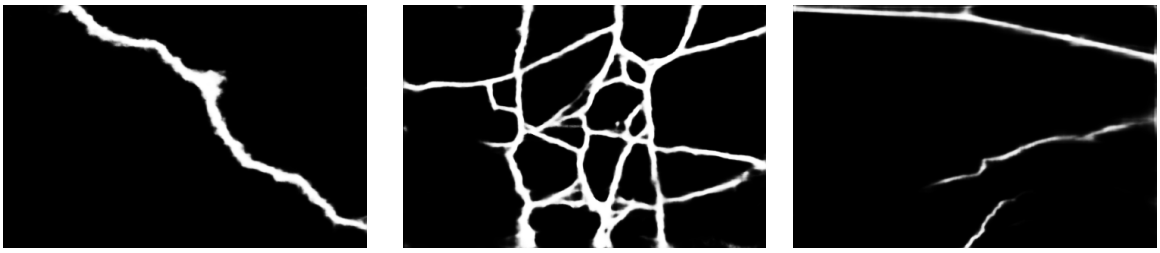


Figure 5.16: A comparison of the precision-recall curve comparing different training datasets for crack segmentation when tested on the CF dataset. The CF and CSSC datasets were combined to give the CSSC_70 and CSSC_140 datasets and are compared with with CSSC_70 and CF_70 datasets.

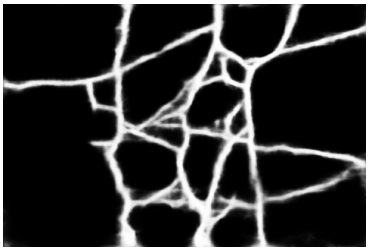


(i): CF_70_Ab (tested on CSSC)

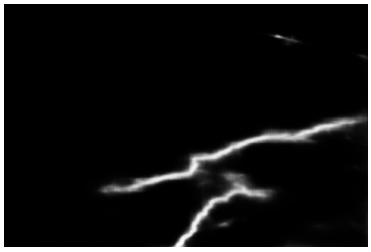
(ii): CF_70_Ab (tested on CSSC)

(iii): CF_70_Ab (tested on CSSC)

(iv): CSSC_70_Bb (tested on CSSC)

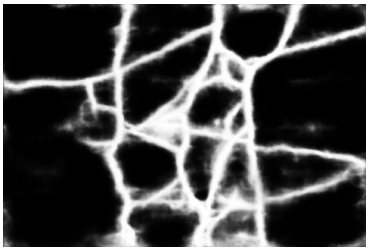


(v): CSSC_70_Bb (tested on CSSC)

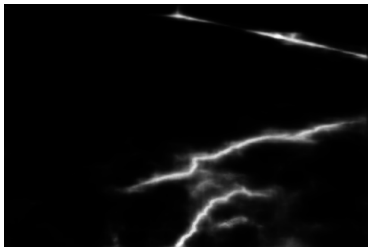


(vi): CSSC_70_Bb (tested on CSSC)

(vii): CF_CSSC_70_Cb (tested on CSSC)

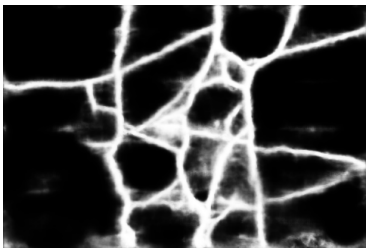


(viii): CF_CSSC_70_Cb (tested on CSSC)

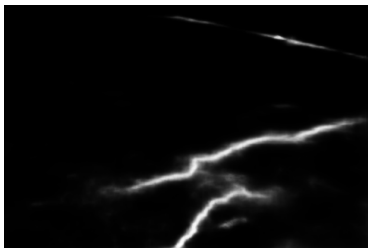


(ix): CF_CSSC_70_Cb (tested on CSSC)

(x): CF_CSSC_140_Db (tested on CSSC)



(xi): CF_CSSC_140_Db (tested on CSSC)



(xii): CF_CSSC_140_Db (tested on CSSC)

Figure 5.17: A qualitative comparison of the results from different training datasets. The CF_CSSC_70 dataset consists of 35 images from both the CF and CSSC datasets. The CF_CSSC_140 dataset consists of 75 images from both the CF and CSSC datasets. The CF_70 dataset consists of 70 images from the CF dataset. The CSSC_70 dataset consists of 70 images from the CSSC dataset. All networks were tested on the CSSC dataset. Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

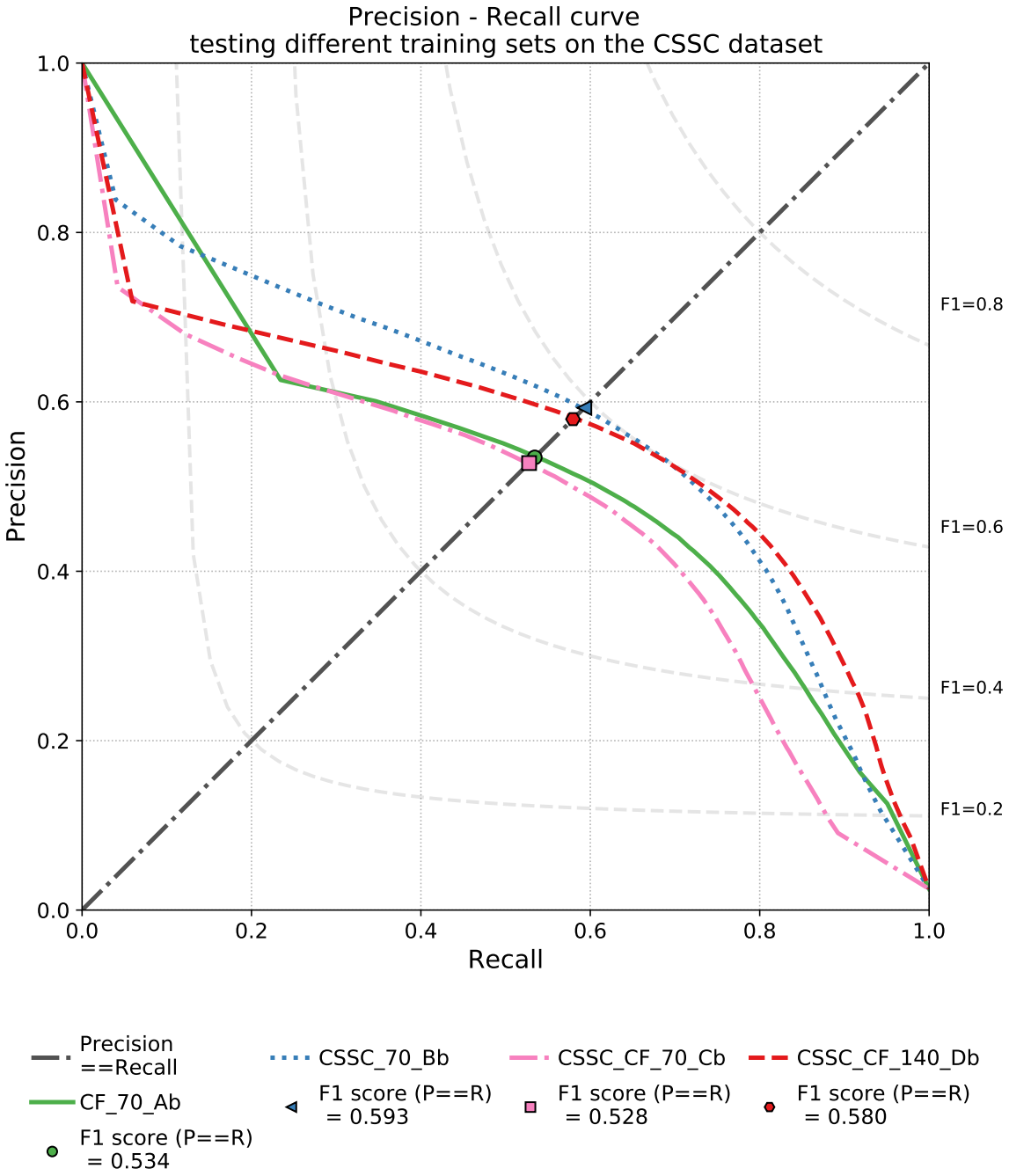
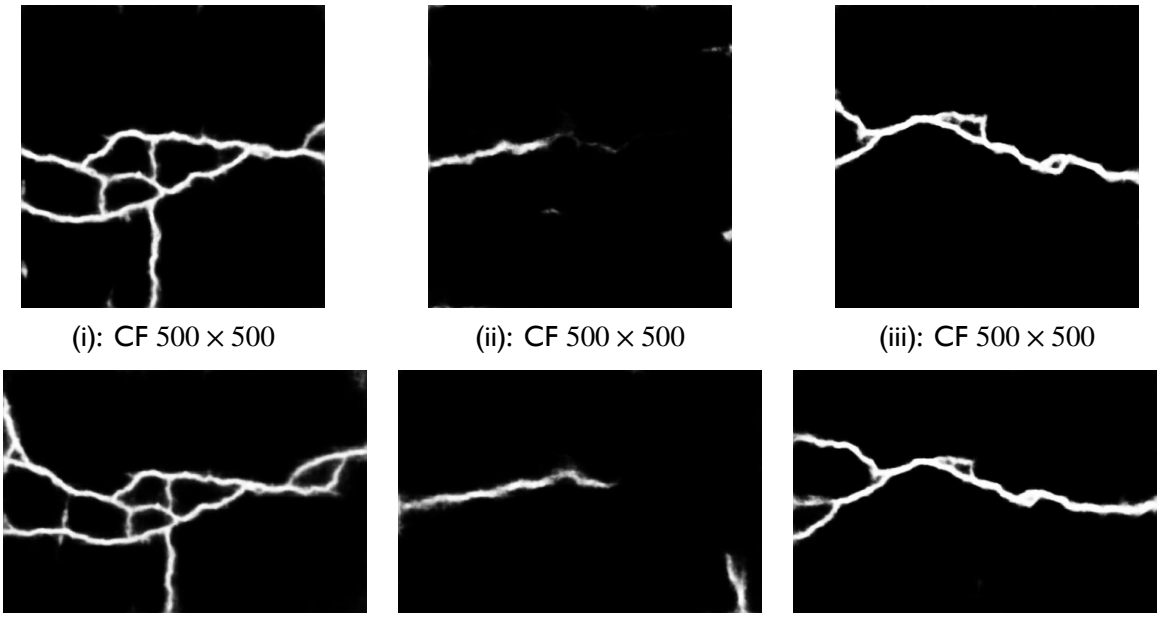


Figure 5.18: A comparison of the precision-recall curve comparing different training datasets for crack segmentation when tested on the CSSC dataset. The CF and CSSC datasets were combined to give the CSSC_70 and CSSC_140 datasets and are compared with with CSSC_70 and CF_70 datasets.

5.10.4 Varying image resolution when training HED

As described in Section 5.3, all of the input images for HED are required to be the same. The image resolution of 480×320 pixels was chosen initially to match the image resolution of the CF dataset and to allow comparison with the results from SFA, which also trained the random forests and SVM classifier using images of 480×320 pixels. However, the original HED implementation uses images of 500x500 pixels. HED was retrained using the same training set from CF, but the images were resized to 500x500 pixels using a script, written in python, that scaled and cropped the images to maintain the aspect ratio of the crack in the original image. The test images were also resized and precision-recall curve for the results for both image resolutions are compared in Figure 5.20 and a qualitative comparison of example results is given in Figure 5.19.



(i): CF 500×500 (ii): CF 500×500 (iii): CF 500×500

(iv): CF 480×320 px (v): CF 480×320 px (vi): CF 480×320 px

Figure 5.19: A qualitative comparison of the results from HED when training and testing on two different image resolutions. The results of the network trained and tested using images scaled to 500×500 are shown in the top row and the results of the network when trained at the original resolution of 480×320 pixels are shown in the bottom row of the Figure. Both sets of images are from the CF dataset. Black pixels correspond to image regions where there are no instances of a crack and white pixels correspond to image regions where there are cracks.

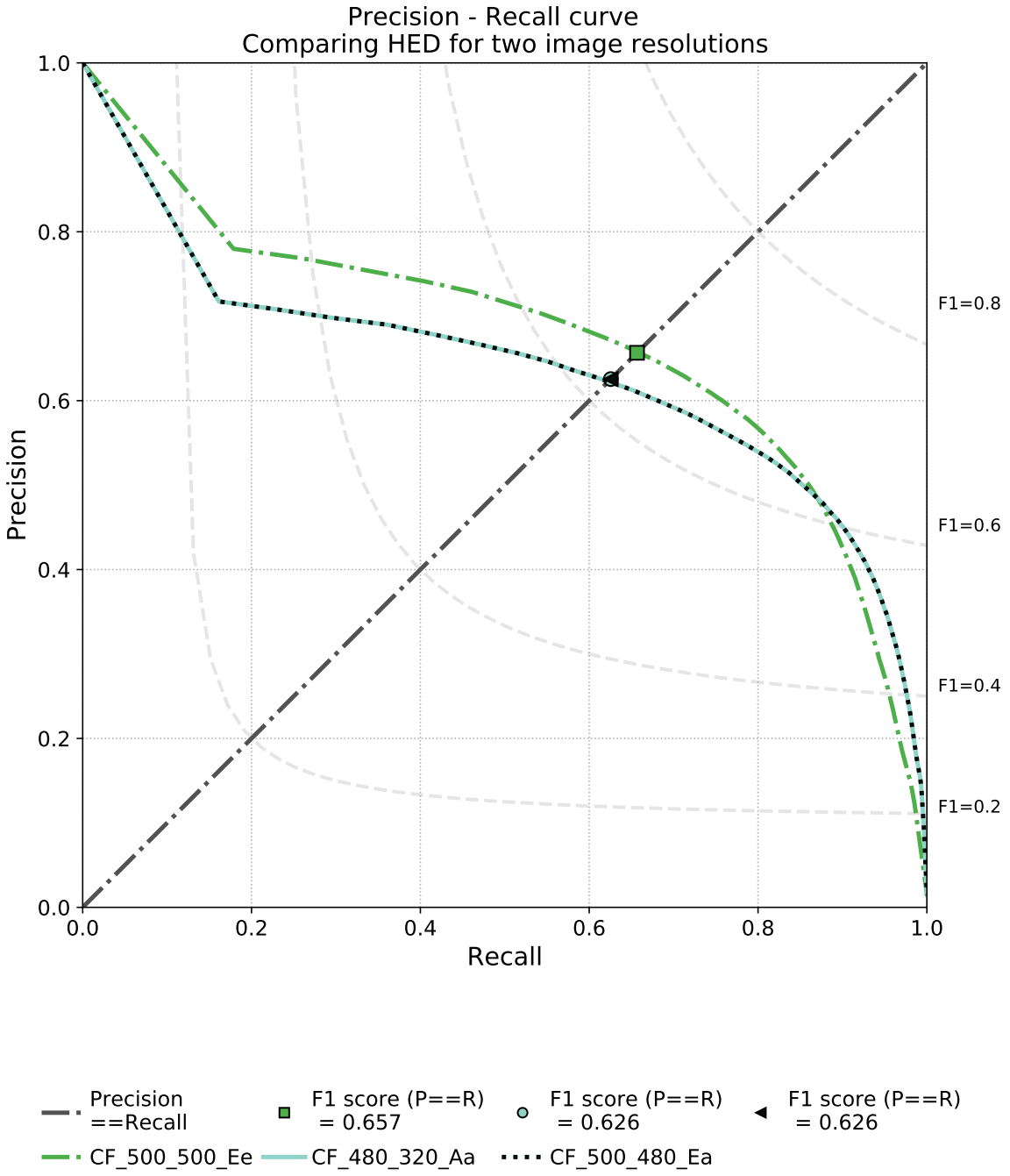


Figure 5.20: Comparison of the precision-recall values for the results of the fused output of the HED network for the CF dataset for image resolutions of 480×320 pixels and 500×500 pixels. The F_1 score where the precision is equal to the recall are marked for each method and the values given in the legend.

Changing the input image resolution improved the F_1 score from 0.626 to 0.657 and the values of precision at low recall. One potential reason for the increase in performance, is shown in Table 5.5. Table 5.5 shows that by resizing the images of the CF dataset from 480×320 pixels to 500×500 pixels for training and testing the HED network, the number of pixels as a ratio the background pixels increases by almost $50\,\%$. One possible reason for the improvement in the results for the HED network is due to increase in the number of positive instances of a crack in the training set for no change in the background texture in the image. As shown in Figure 5.19, one consequence of resizing the images is that some of the cracks have been cropped away. However, the CF dataset focuses the images around the crack, hence the majority of pixels that were removed were background pixels, which increases the percentage of pixels containing cracks relative to the image. Increasing the scale of the CF dataset images increased the probability that a pixel is a crack and details of the results. This is apparent by the increased contrast of the cracks in Figures 5.19i–5.19iii when compared to Figures 5.19iv–5.19vi and the increased detail seen in Figure 5.19iii when compared to Figure 5.19vi.

Table 5.5: The percentage of pixels corresponding to cracks in the CF dataset for an image resolution of 480×320 pixels and 500×500 pixels. The percentage of pixels classified as a crack is calculated as a ratio to the number of non-crack pixels in the image.

Dataset Name	Data type	Averaged percent crack (%)	Standard deviation (%)
CF 480×320 px	Crack	1.43	0.95
CF 500×500 px	Crack	2.15	0.95

5.11 Incorporating features from blood vessels for the segmentation of cracks

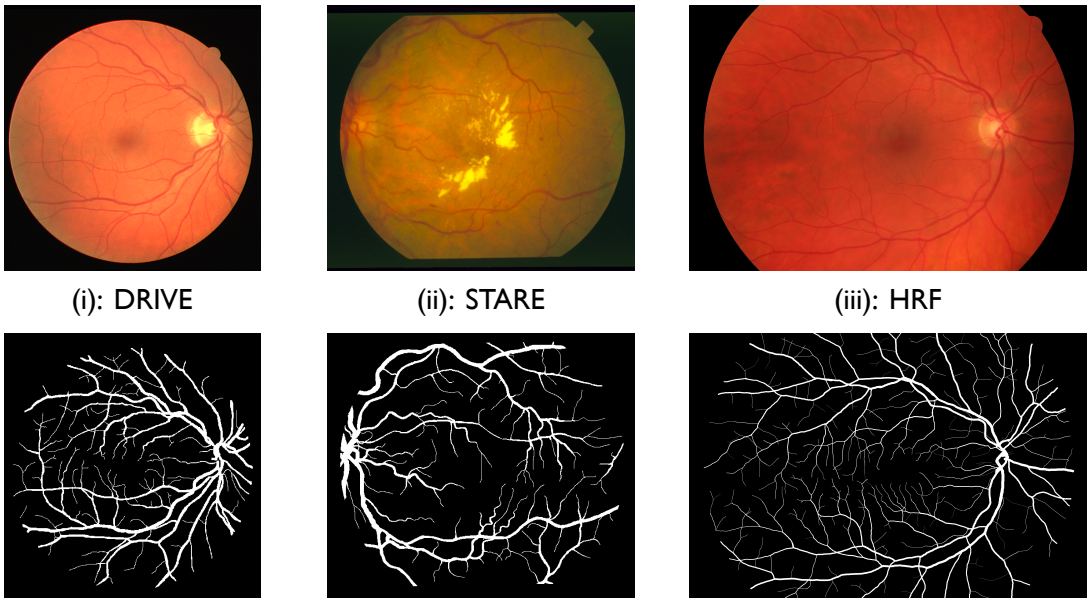
In Section 5.10 methods from the medical segmentation of blood vessels from images of the retina were used to improve the methodology for evaluating the segmentation of cracks from images. However, there may be other useful information that can be utilised for the purpose of segmenting cracks. For example, there are many visual similarities between the branching patterns made by blood vessels and cracks. In the book 'The Self-Made Tapestry: Pattern Formation in Nature' Philip Ball describes some of the physical reasoning behind the patterns that are visible in nature, such as the branching patterns of mineral dendrites in crystals (Ball, 1999). These patterns can often be described mathematically using fractals. The term fractal was coined by Benoit Mandelbrot (Mandelbrot, 1982) to describe both the irregularity and fragmented shapes of patterns in nature (Mandelbrot, 1982). A key feature of fractals is the property of scale invariance, where zooming in or out of a section of the shape gives the shape as a whole and self-similarity.

Not all branching structures are fractals, but, as demonstrated by Ball, cracks can be described by fractal models and there formation has similarities to the propagation of lightning (Ball, 1999). Both the retinal blood vessels and cracks can be described using their fractal dimension. The fractal dimension describe how densely packed a branching structure is (Ball, 1999; Mandelbrot, 1982), where a fractal dimension of 1 is close to a straight line and a fractal dimension of 2 is a densely branching structure.

To compare the fractal dimension of cracks and retinal blood vessels, some datasets of the retina are required. Images of the retina are collected when diagnosing patients by looking for narrowing of the retinal blood vessels or discolouration of the optic nerve when diagnosing diseases such as Coat's disease and diabetes (Staal et al., 2004; Hoover, 2000). Three commonly used datasets in retinal segmentation and the properties of these datasets is outlined in Table 5.6.

Table 5.6: A summary of recent retinal datasets showing the datasets described above with the associated image resolutions and number of images for each dataset.

Dataset name	Dataset type	image resolution (pixels)	Total number of images at original resolution
DRIVE	Retina	565×584	40
STARE	Retina	700×605	20
HRF	Retina	3504×2336	45



(i): DRIVE (ii): STARE (iii): HRF

(iv): DRIVE ground truth (v): STARE ground truth (vi): HRF ground truth

Figure 5.21: Example RGB images (5.21i–5.21iii) and corresponding ground truth labels (5.21iv–5.21vi) from the retinal blood vessels datasets. The ground truth labels were manually annotated. where the white pixels denote a pixel containing a blood vessel and the black pixels denote the background.

The fractal dimension of a shape can be calculated using a box-counting method. Box-counting overlays a grid over an image and counts the number of pixels of the shape of interest in the boxes of the grid whilst the grid-size is decreased over subsequent iterations. Box-counting is used to determine how the detail of the shape changes with decreasing

scale. The change in detail (number of relevant pixels) is plotted against the change in scale of the grid, and the slope of the logarithmic regression of these values is the fractal dimension.

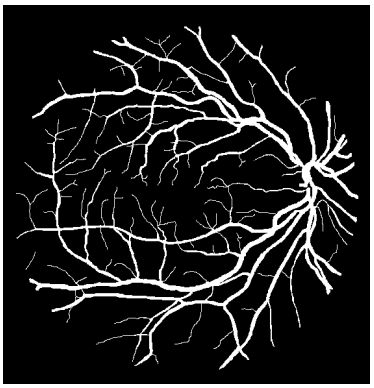
Several authors have used the fractal properties of the retinal blood vessels and of cracks for image segmentation and characterisation. For example, the authors of Salari and Ouyang (2012) use fractal-inspired approach for thresholding images containing cracks. Similarly, Ventura et al. (2015) compare the fractal dimension of the retinal blood vessels in patients with and without a disease with the aim of creating a classifier for the disease.

For context, the fractal dimension of the images in Figure 5.22 are given. The box-counting algorithm used to obtain the fractal dimension of these images was implemented using the fractal analysis plugin FracLac[3] to the image analysis software called ImageJ[4]. Since the HRF dataset has a larger image resolution than DRIVE or STARE, the images were cropped to match the image resolution of the CF dataset.

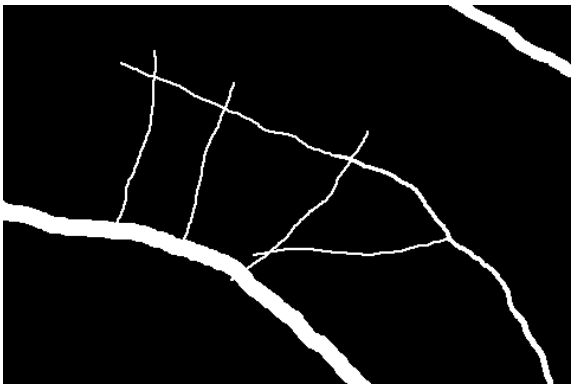
Both cracks and the retina have a fractal dimension between 1 and 2 (Ventura et al., 2015). The full image of the retina in Figure 5.22i has a fractal dimension of $D = 1.6463$ with a standard deviation 0.0323. This result aligns with fractal dimension of 1.7 reported in (Ball, 1999). For the images of cracks in Figure 5.22iii and 5.22iv, the fractal dimensions are $D = 1.17(+/-0.0282)$ and $D = 1.14(+/-0.0136)$. These values show that the cracks have some self-similarity, but are not as dense as the retinal image. Furthermore, when the high resolution retina image is cropped it also has a fractal dimension of $D = 1.2306(+/-0.0148)$. This value is much closer to the value for images of cracks and suggest that there may be some features that are worth using for training the HED network. The patches of HRF appear visually similar to features you might expect in crack detection, such as being linear across an image and branching into smaller sections.

In the previous section, the performance of the segmentation of cracks was improved by increasing the image resolution. When the image resolution was increased, the proportion of the number of cracks containing pixels to background increased by $50\,\%$. The authors of HED (Xie and Tu, 2017) address the class imbalance between pixels by introducing a

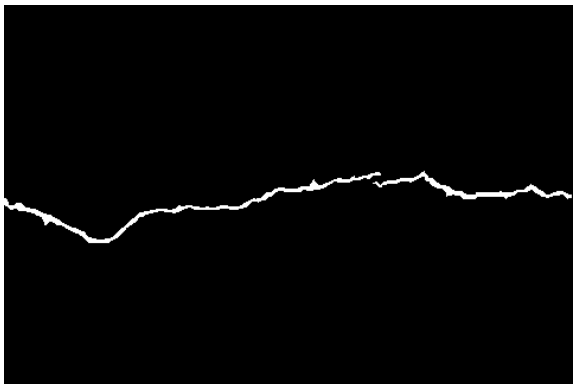
[3]https://imagej.nih.gov/ij/plugins/fraclac/ date last accessed:01/02/19
[4]https://imagej.nih.gov/ date last accessed:01/02/19



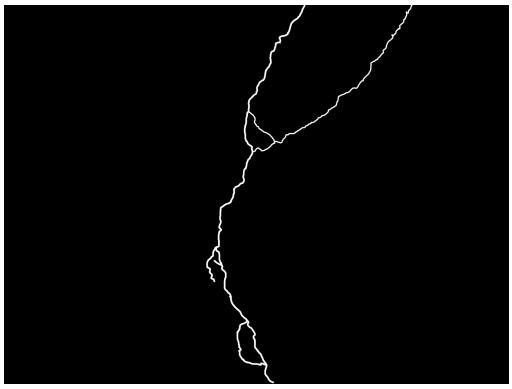
(i): DRIVE: $D = 1.6463 + / - 0.0323$



(ii): HRF: $D = 1.2306 + / - 0.0148$



(iii): CF: $D = 1.17 + / - 0.0282$



(iv): CSSC: $D = 1.14 + / - 0.0136$

Figure 5.22: Example ground truth images from datasets of the DRIVE and HRF retinal datasets (5.22i and 5.22ii) and from the CF and CSSC datasets of cracks in concrete material (5.22iii and 5.22iv).

term in the loss function used when training the networks (see Section 5.3). When training the Deep Retinal Image Understanding (DRIU) network (a variation of HED) using DRIVE and STARE images (Maninis et al., 2016) state that there is an expected imbalance where only $10\,\%$ of pixels in the image will contain a positive example (i.e., the pixel contains an example of a retinal vessel), the remaining pixels should be classified as negative.

In Table 5.7 the number of pixels that are classed as a blood vessel in proportion to the background is shown. The number of positive pixels in the HRF patches are around six times greater than in the CF dataset. As the features for eyes and crack are similar, the addition of HRF images may improve the segmentation of cracks during cross-dataset training and testing. It should be noted, that the increase in the number of pixels for the CSSC dataset did not improve the results, since the standard deviation of the percent of

pixels in the image is much greater for CSSC than CF, it is likely that the results did not improve due to size of some of the cracks included in the training dataset, see Figure 5.23 or due to the noisy nature of the dataset (discussed further in Section 5.12).

Figure 5.23: An example of an image and associated ground truth label in the CSSC dataset which contains a crack that has a high percentage of image pixels containing cracks.

The number of positive examples of pixels containing cracks in the DRIVE dataset is $51.83\,\%$ which is a five-fold increase over the HRF patches. For comparison, the HED network has been used for the segmentation of blood vessels from the DRIVE dataset, and achieved a reported F_1 score of 0.796 (Maninis et al., 2016). Since the number of examples of pixels containing blood vessels is greater than the number of pixels containing positive examples of cracks, it may be that introducing examples of eye images into the crack dataset may increase the amount of available data for training for crack images. In this section, the images from the cropped HRF dataset was used to test whether the addition of images of the retina can improve the training and generalisation of the crack dataset and HED. The cropped HRF dataset was used over the DRIVE dataset due to the visual similarities to cracks. Example patches for HRF are shown in Figure 5.24.

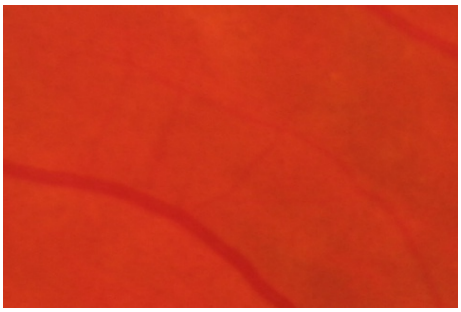
Table 5.7: The percentage of pixels corresponding to cracks or blood vessels in different datasets. The percentage of pixels classified as a crack is calculated as a ratio to the total number of pixels in the image.

Dataset Name	Data type	Averaged percent crack / retina (%)	Standard deviation (%)
CF	Crack	1.41	0.69
CSSC	Crack	5.02	4.68
DRIVE	Retina	51.83	17.18
HRF (cropped)	Retina	9.46	5.54

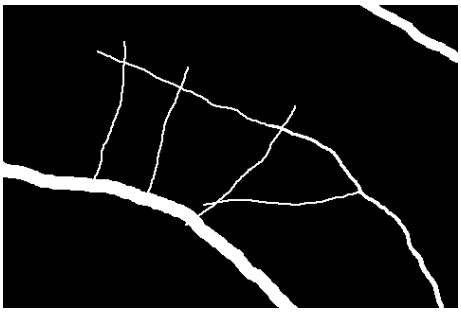
Similarly to the approach used for cross-dataset training and testing of crack datasets in Section 5.10.3, mixed datasets were generated for training the HED network. These datasets consisted of a combination of HRF patches and images from the CF dataset, both at an image resolution of 480×320 pixels. The mixed datasets were composed of a $50:50$ split of HRF patches and images from the CF dataset. The first mixed dataset had a total of 70 images, and will be referred to henceforth as CF_HRF_70_Fa for the networks tested on the CF dataset and CF_HRF_70_Fb for the networks tested on the CSSC dataset, respectively. The second mixed dataset had a total of 140 images and will be referred to henceforth as CF_HRF_140_Ga for the networks tested on the CF dataset and CF_HRF_140_Gb for the networks tested on the CSSC dataset, respectively. These datasets were tested on the CF and CSSC datasets; an overview of these experiments is given in Table 5.8. The qualitative results are shown in Figure 5.25 and the precision-recall results are shown in Figure 5.26. For reference, the results of the cross-dataset testing from Section 5.10.3 are also included in the precision-recall plot (Figure 5.26).

Table 5.8: A summary of the experiments performed by the combined dataset of cracks and eyes, with the method that is tested and the number of images that are used for training and testing.

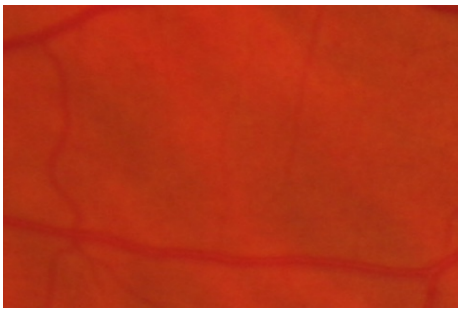
Dataset label	Dataset for training	Dataset for testing	Num. images training	Num. images testing	Method
CF_HRF_70_Fa	CF+HRF	CF	70	48	HED
CF_HRF_70_Fb	CF+HRF	CSSC	70	48	HED
CF_HRF_140_Ga	CF+HRF	CF	140	96	HED
CF_HRF_140_Gb	CF+HRF	CSSC	140	96	HED



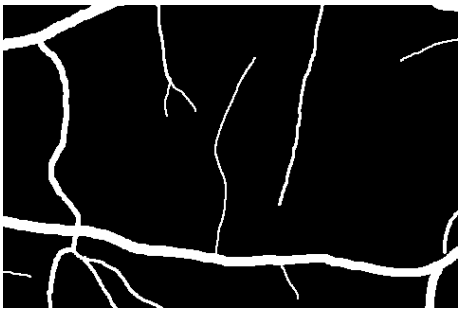
(i): Image 1 of HRF training dataset



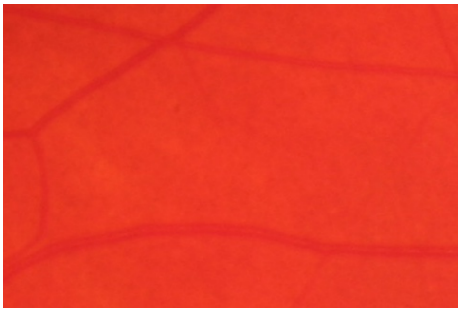
(ii): Ground truth of image 1 of HRF training dataset



(iii): Image 2 of HRF training dataset

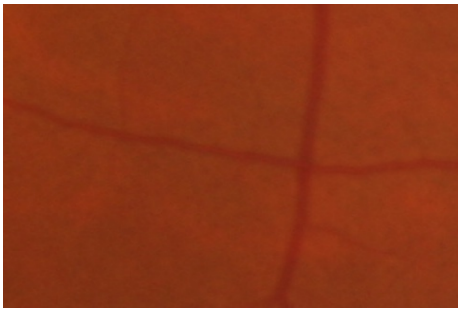


(iv): Ground truth of image 2 of HRF training dataset

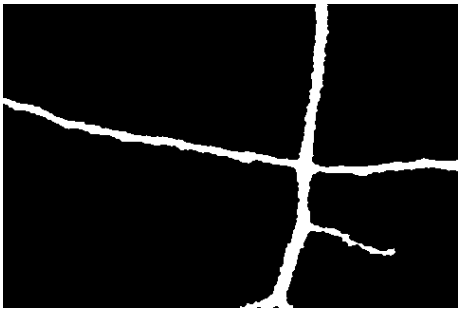


(v): Image 3 of HRF training dataset

(vi): Ground truth of image 3 of HRF training dataset



(vii): Image 4 of HRF training dataset



(viii): Ground truth of image 4 of the HRF training dataset

Figure 5.24: Example images from the cropped HRF dataset with respective ground truth labels, where the white pixels denote a pixel containing a blood vessel and the black pixels denote the background. Images 5.24ii,5.24iii,5.24v,5.24vii have been edited to improve the visibility of the retinal blood vessels for this example.

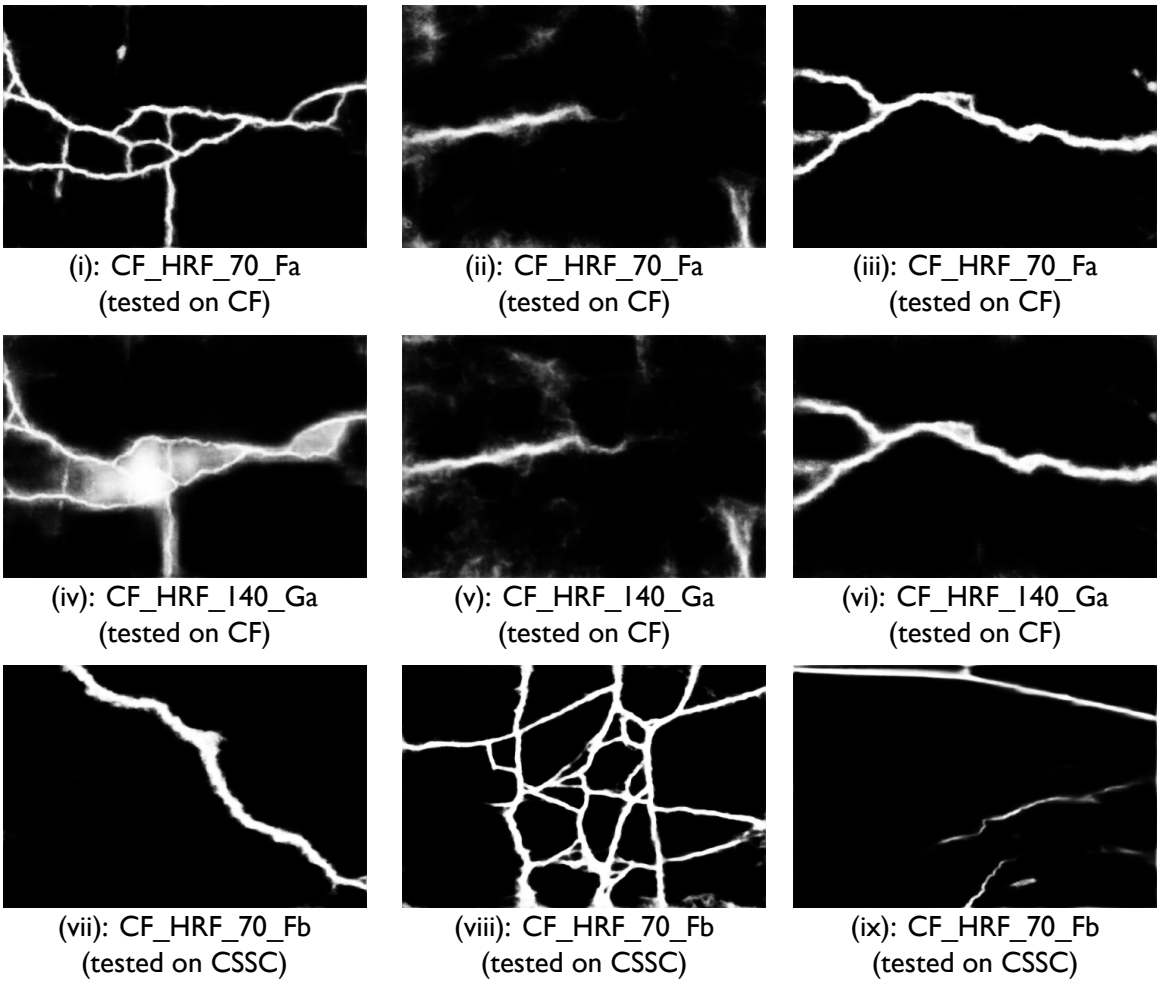


Figure 5.25: A qualitative comparison of the results from different training datasets. The CF_HRF_70 dataset consists of 35 images from both the CF and HRF datasets. The CF_HRF_140 dataset consists of 70 images from both the CF and HRF datasets. The CF_70 dataset consists of 70 images from the CF dataset. The CSSC_70 dataset consists of 70 images from the CSSC dataset. All networks were tested on the CF dataset. The white pixels denote a pixel containing a crack and the black pixels denote the background.

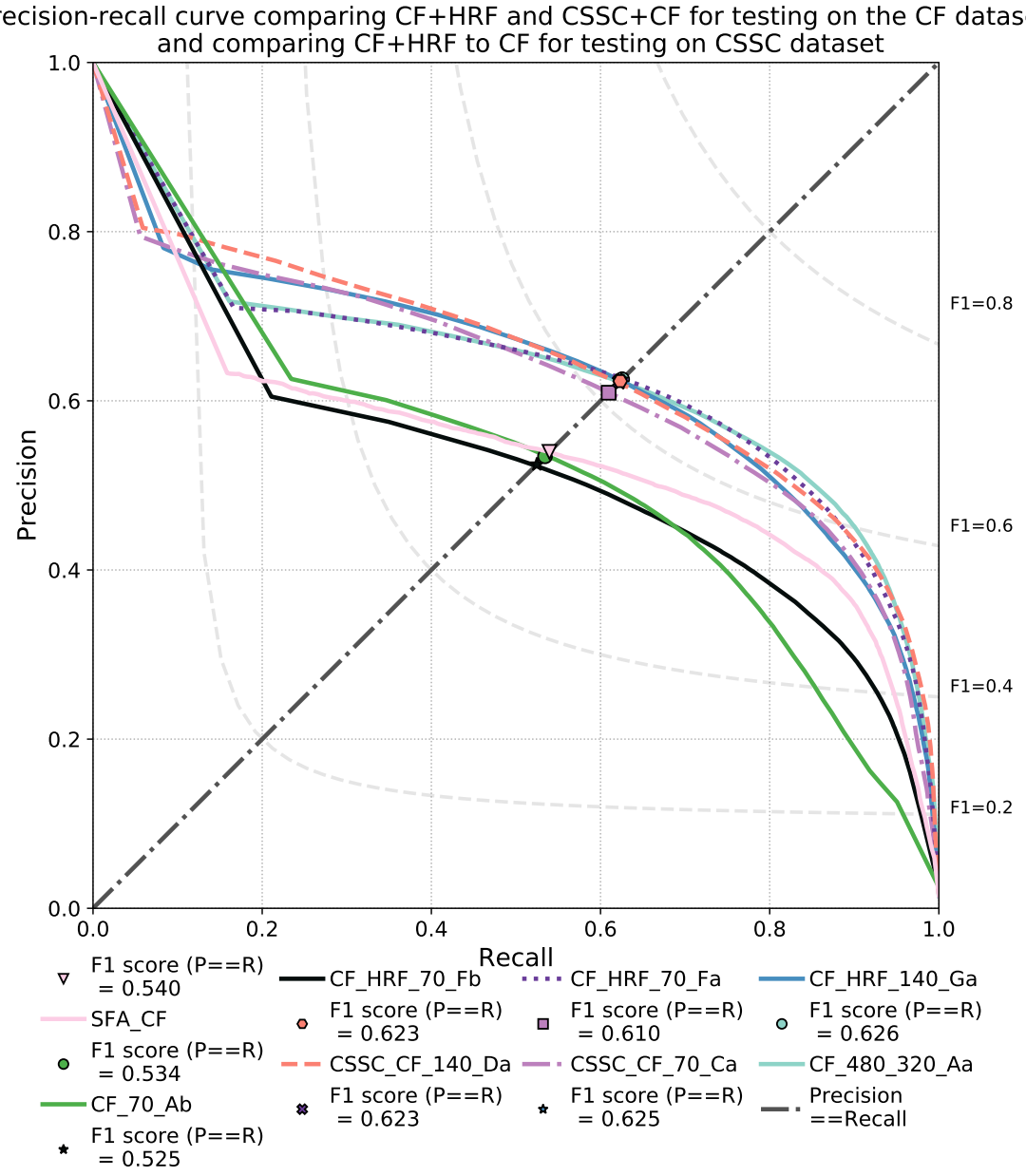


Figure 5.26: Comparison of the precision-recall values for the results of the fused output of the HED network for datasets that include images of both cracks and blood vessels. All of the combined datasets are tested on the CF dataset. The F_1 score where the precision is equal to the recall are marked for each method and the values given in the legend.

Again, the combination of datasets led to an increase in the segmented background noise, particularly for the increased amount of HRF patches in the combined dataset (compare Figures 5.25i–5.25iii and Figures 5.25iv–5.25vi). Similarly for the datasets compared in Section 5.10.3, the networks trained on eyes also mis-segment pixels that belong to a join in the floor and wall as a crack, see Figures 5.25ix.

The CF_HRF_70 dataset was also tested on the CSSC dataset to observe whether the addition of the features from blood vessels would give an acceptable performance on a dataset that it has not been exposed to. This result is shown as the black dotted line marked CF_HRF_70_Fb in Figure 5.26. However, this result is as poor as CSSC_70 when tested on the CF dataset (light blue dotted line marked CSSC_70 in Figure 5.16). Therefore the addition of the blood vessel features has not significantly increased the performance of the HED network for generalisation on a new dataset.

Overall, the precision recall of the trained using HRF patches combined with the CF datasets (CF_HRF_70 and CF_HRF_140) are very similar to the performance of the networks trained using CSSC and CF datasets (CF_CSSC_70 and CF_CSSC_140). The performance for both CF_CSSC_140 and CF_HRF_140 is marginally better than for CF_CSSC_70 and CF_HRF_70 at low recall values when tested on the CF dataset, but not when tested on the CSSC dataset (CF_CSSC_140 in Figure 5.17), which suggests overfitting of the HED network due to the uniformity of the CF dataset.

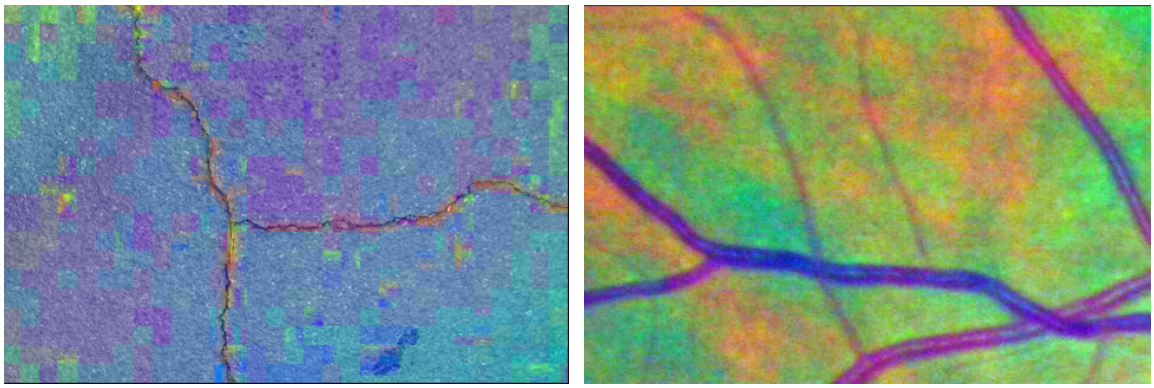
5.12 Discrepancies in dataset quality

One reason that the results may not have improved further is due to the difference in quality of the available datasets. The CF and CSSC dataset both contain artefacts that are not present in the HRF dataset. These artefacts can be emphasised by applying the Matlab implementation of the decorrelation stretch algorithm[5] to the CF and HRF patches (see Figure 5.27). Decorrelation stretch acts to accentuate the differences between the channels (i.e., red, green and blue) of a multi-channel image. The artefacts visible as a chess-board pattern in Figure 5.27 are most likely due to jpeg compression of the images in the CSSC and CF dataset, which is unavoidable for the CSSC dataset as the images were collated from an online image search, but may also be due to the quality of the camera that was used to collect the data. In contrast, there are no artefacts present in the HRF images, which were collected under more rigorous laboratory conditions.

It is also possible that the background texture of the retina is different to the texture of asphalt or concrete in images, and this feature may be the cause of some of the 'fuzzy' artefacts that appear in Figure 5.25. However, due to the block-like artefacts in the crack images it is difficult to discern how much an effect the difference is caused by the lack of block-like artefacts in the retina images rather than the difference between the visual background texture.

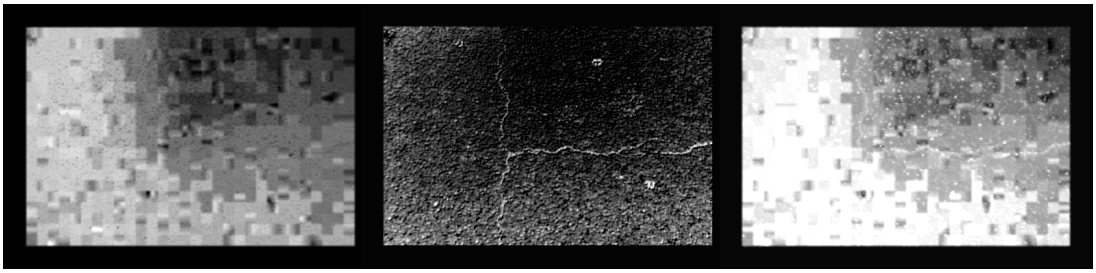
Furthermore, the effect on the training processes caused by the artefacts is unknown. Figure 5.28 displays activation maps from layers of the HED network, which shows example filters applied to the network during the testing process. Although these filters may not be the ones used to produce the results shown in this chapter, Figure 5.28 shows that these compression artefacts are also visible as a chess-board pattern during the training process and may even be the reason for the blocky nature of the cracks segmented by DSN5 (see Figure 5.28xv and Figure 5.11vii in Section 5.10.2).

[5] https://uk.mathworks.com/help/images/ref/decorrstretch.html (Date accessed: 01/02/2019)



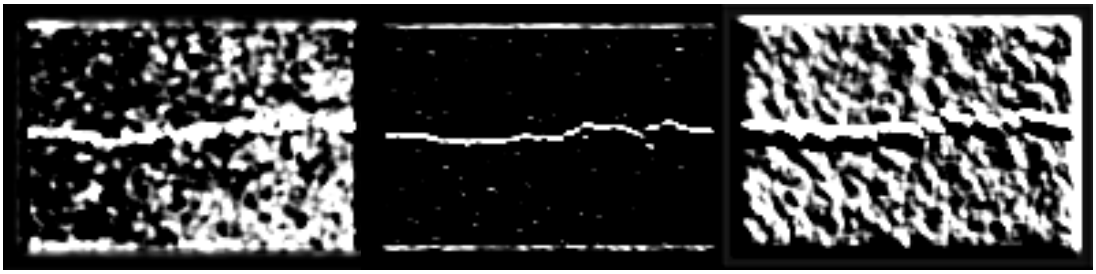
(i): CF image with decorrelation stretch (ii): HRF image with decorrelation stretch

Figure 5.27: Decorrelation stretch algorithm applied to an example image from the CF dataset (5.27i) and HRF dataset (5.27ii). Artefacts are visible as a chess-board pattern when decorrelation stretch is applied to the CF dataset; these artefacts are not visible in the HRF image and are suggestive of jpeg compression artefacts.

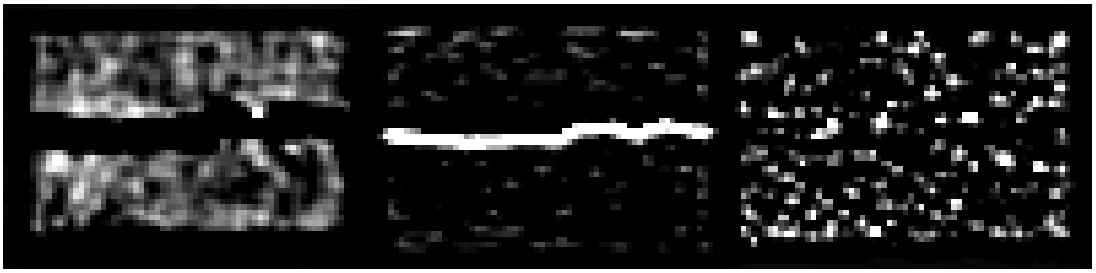


(i): A filter from layer one (ii): A filter from layer one (iii): A filter from layer one

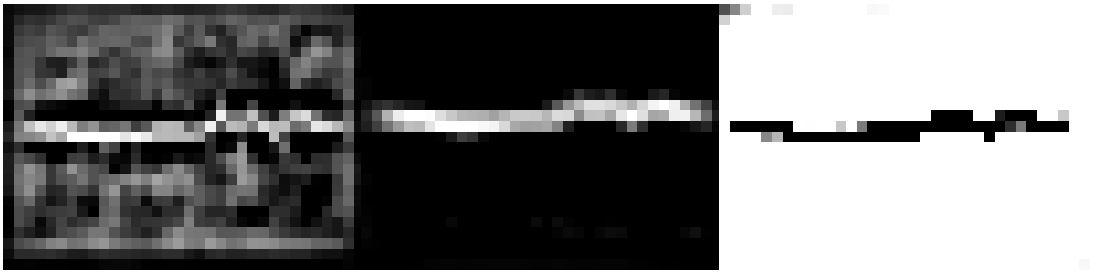
(iv): A filter from layer two (v): A filter from layer two (vi): A filter from layer two



(vii): A filter from layer three (viii): A filter from layer three (ix): A filter from layer three



(x): A filter from layer four (xi): A filter from layer four (xii): A filter from layer four



(xiii): A filter from layer five (xiv): A filter from layer five (xv): A filter from layer five

Figure 5.28: Comparison of the activation of layers inside the HED network when trained on images from the CF dataset. Each column shows an example filter from inside a layer of the network. The filters were chosen to show the block-like artefacts, see Figure 5.27.

5.13 Summary and scope for future work

In this chapter, a review of the current literature for computer vision based segmentation of cracks was performed. The literature showed that there is an increasing interest in the automation of segmentation of cracks from images, particularly using deep learning approaches. However, there are many inconsistencies in the approaches used for training and evaluating the deep learning approaches, with the use of different metrics making the comparison of such approaches difficult. These inconsistencies show the need for agreement in the methods used for bench-marking new approaches.

The work in this chapter made steps towards this goal by using existing datasets to compare a deep learning approach and an approach that used random forests and a SVM classifier. Using these datasets, evaluation of segmentation approaches were performed. The main comparisons took the form of precision-recall analysis, which has foundations in the fields of object recognition and medical segmentation (e.g., Everingham et al. (2010) and Xie and Tu (2017) and Maninis et al. (2016)). Using this method allows justification for the choice of the threshold that is applied to the grey-level probability maps, rather than using arbitrary thresholds, as reported in the current literature. Qualitative examination of the results was also performed. In addition, two other approaches were introduced for evaluation (SSIM and S-measure), which focused on the structure of the results, and compared to the F_1 score (a pixel-based evaluation metric that combines precision and recall). It is recommended that future research benchmarks the results from literature using the metrics presented in this chapter to better understand the correlation between these metrics, if any, and the usefulness of these metrics for real-world testing.

The network used in this chapter (referred to as HED) had a better performance than the approach that used random forests and an SVM classifier (referred to as SFA) for all precision-recall metrics. The authors of SFA (Shi et al., 2016) defined thresholds to remove pixels with low confidence from the resulting probability maps, but with very little justification as to why this threshold was suitable. In contrast, the evaluation methods used in this chapter constitute a more consistent means for choosing this threshold value.

The application of post-processing joined gaps in the segmented cracks and may be useful for emphasising cracks that have low confidence in the probability maps. It is recommended that the threshold for post-processing be chosen carefully, since too high a threshold increased the recall of pixels labelled as cracks, and reduced the performance for all metrics.

The side-outputs of the HED network showed the detection of features at different image scales, with more background noise being removed in the deeper layers of the network. The S-measure results for these side-outputs increased in a linear fashion. Overall, the fused output of HED gave the best-performing result, rather than the one of the side-outputs. However, in general there may be benefit to further exploration of the effect of varying network architecture. It is recommended that future work should evaluate different deep learning architectures (e.g., U-Net Ronneberger et al. (2015)) to understand the effect of different architectures on the segmentation of crack features and to compare with other network architectures that have been used for crack segmentation.

Cross-dataset evaluation was used to test the generalisation of the trained network to other datasets. Overall, combining the different crack datasets did not lead to an increase in the performance for segmenting cracks. Future work should examine these results further to ensure there is no over-fitting of the deep-learning networks. Increasing the resolution of the images used for training did improve the performance of the HED network, perhaps due to the increase in the number of positive examples of the crack as a proportion of the training data.

Datasets for segmentation of blood vessels were also used to test whether the number of positive samples could be boosted using the visually similar features of the blood vessels and cracks, such as the branching patterns formed by both. The results of training on a mixture of data from eyes and cracks were very similar to the results from cross-dataset testing using datasets of cracks and there was no increase in performance by adding the images containing blood vessels.

In addition, the datasets from crack detection were found to have compression artefacts. The affect of these artefacts on the results should be explored in future research. It is

proposed that the community needs to develop a dataset that can be used for comparison and bench-marking new approaches.

In the case of poor quality datasets, further research into the use of similar features for training should be explored, with suggestions for using satellite data for the segmentation of rivers or synthetic cracks created from simulated data.

Although the datasets used in this chapter are not from bridge environments, the methods explored here are applicable to crack detection in the concrete structure surrounding bridge bearings, which is required as part of the visual inspection of bridge bearings (see Chapter 1). Questions for future work include the applicability of the methods discussed in this chapter for use in the inspection environment, by testing the trained model on real-world data.

Chapter 6

Summary and conclusions

Two main research topics were considered in this thesis. The first research topic focused on the development of a robotic platform for testing localisation and mapping approaches; the second topic focused on the use of computer vision methods for detection of cracks in concrete. Both of these research areas contribute to the automated and robotic inspection of a bridge, with a specific focus on the bridge bearings. The success and limitations of this thesis in addressing these research topics are discussed in this chapter.

6.1 Robotic localisation and mapping for inspection environments

Investigate and identify existing sensors and methods for localisation and mapping that could be used for navigating in a real inspection environment.

In Chapter 1, the current literature for robotic bridge inspection was reviewed. This review highlighted the aspiration for automated bridge inspection. This aspiration is due to an increasing demand for inspection of bridges, partly because of an increasing number of bridges (Sutter et al., 2018), but also due to ageing infrastructure with some components, such as bridge bearings, requiring frequent inspection (Freire et al., 2014). Consequently, the literature demonstrates growing interest in developing robotic tools for inspection of

bridges. Although fully automated robotic bridge inspection was stated as an ambition, with many researchers developing robotic platforms, most of the current and previous research focuses only on platform development or data collection, with little focus on developing the navigation algorithms that are necessary to achieve this aim.

To achieve autonomous inspection, methods for mapping and/or localising in an environment are required. In other robotics applications, methods for simultaneous localisation and mapping (SLAM) or localisation-only methods (where the map is determined a priori) are used. In this thesis, different sensors and algorithms were reviewed for SLAM and localisation-only approaches. A review of the literature also informed the choice of robotic platform and a ground-based platform was selected over other configurations due to its stability and maturity in the wider robotics literature. However, the methods reviewed in this thesis are not platform specific and could potentially be used to aid other approaches and platforms developed in the literature.

Develop a robotic platform for the purpose of testing localisation and mapping algorithms in a real bridge environment.

Many of the robotic platforms that exist in the literature are tested only in a laboratory environment. A real bridge environment was chosen to complement a laboratory environment for the experimental environments for the studies in this thesis. One difference between a typical bridge environment and other settings in robotics research is the size constraints that are imposed. This factor limited the size of the robotic platform and the sensors that could be used for testing in this thesis. A 2D LiDAR, monocular camera and stereo camera were chosen alongside existing methods for SLAM and localisation that had been implemented in urban or outdoor environments.

Use existing tools that are commonly used in robotics research for inspection applications (e.g., the Robot Operating System).

After reviewing robotic applications in other disciplines where inspection was required, the Robot Operating System (ROS) was selected to aid the implementation of data collection,

localisation and mapping algorithms and motion of the robotic platform. In general, ROS provided many useful features that were invaluable in this project. However, the impact of the failures of a system like ROS require further research in real-world environments. In addition, it should be noted that many of the available algorithms for navigation, path planning and localisation from ROS were designed for use in indoor environments on larger robots. As a result, autonomous motion of the robotic platform in this research was not implemented and should be considered as an area of future research, with a particular focus on navigation and path-planning in confined areas.

In Chapter 2, existing SLAM and localisation techniques were evaluated in a laboratory and a bridge environment. Hector SLAM was selected for use with the 2D LiDAR scan. In general, Hector SLAM performed better than the localisation-only approaches, because it takes the current environment into account and is not affected by changes, since the map is created as the robot moves around the environment. In contrast, AMCL uses a pre-existing map; in this thesis, a mapping approach that can make use of existing surveying data was presented for use with AMCL. The maps created using this method showed comparable performance to when AMCL was tested using a map created using data from Hector SLAM.

Develop a mapping approach that can make use of existing surveying data.

The method for map creation used data that was commonly collected by researchers in the literature review; namely 3D point cloud data. In particular, two types of data collection were considered for point cloud generation. The first was a 3D reconstruction method known as Structure-from-Motion, which extracts key features in photographs to generate a point cloud. In addition, terrestrial LiDAR data was collected, the output of which is also a 3D point cloud. They key difference between SfM and terrestrial LiDAR data is that the SfM data typically has no global scale, unless some reference measurements to provide scale are supplied. However, SfM is typically a lower cost solution to generating the point clouds than terrestrial LiDAR and they can be generated as a secondary task. In this thesis, scale was provided to the SfM point clouds using measurements from the reference point cloud data, but past research has also used measurements from the bridge environment, CAD

drawings, or referenced (i.e., photogrammetry). The SfM point cloud was validated against a terrestrial LiDAR point cloud.

After the point clouds were collected, they were cropped to obtain a region in the area of interest, this reduced map was then converted into an occupancy map. In this thesis, the assumption was made that the surface of the bearing enclosure was reasonably level, and that points can be extracted from a region around the height of the 2D LiDAR on the robotic platform. The generalisation of the method used in this thesis should be tested in different bridges.

Test existing state-of-the-art localisation and mapping techniques in a real environment and compare to laboratory experiments.

In the laboratory environment, local and global implementations of AMCL were tested; global AMCL allows localisation without a known starting position, whereas local AMCL needs an initial position guess in the map. For global AMCL, the errors in the calculated position in the lab environment were as large as $75\,\mathrm{cm}$ and were sustained over several time steps. For the local implementation of AMCL, the discrepancy with respect to the reference trajectory was below 3cm at all times. However, the repeatability of initialising the robot from a known start position and the maximum threshold for this error in the initial position should be considered as a topic of future research.

Both Hector SLAM and AMCL are 2D methods. In this thesis, limited studies were conducted into methods for 3D. One approach for 3D mapping–ORB SLAM–was evaluated using both a monocular and stereo camera. However, the stereo implementation was found to have problems relating to the available frame rates of the camera. The monocular camera performed better than stereo but a method for correctly scaling the resulting trajectory is required. Further research might consider the development of expanding the work in this thesis from 2D to 3D.

In the bridge environment, a maximum error of 10cm from the reference trajectory was chosen to evaluate the performance of the algorithms. In general, Hector SLAM performed better than the AMCL localisation methods; However, the errors for all methods remained

below the defined error limit of 10cm. There are potential advantages to using each of the methods in an inspection environment. Since Hector SLAM creates the map each time, it is more robust to changes in the environment; however, the robot has to explore regions of the environment to know if it is possible to travel there. For an inspection application, there may be some utility in defining the map in advance to enable a human operator to define regions of interest. The work in Chapter 4 made steps towards creating a combined approach using the trajectory information from both Hector SLAM and AMCL. It is recommended that future research evaluate this approach further and consider the use of additional sensors.

Use existing data from bridge surveys to develop a simulation environments in order to test and prototype in different bridge configurations.

In Chapter 3 and Chapter 4 the robotic platform was developed for use in simulation using the open-source simulation environment called Gazebo (Koenig and Howard, 2004). Gazebo can be used to simulate the dynamics of an environment and provides plugins for simulation of sensors with appropriate noise models (Koenig and Howard, 2004). Gazebo has been used by many robotics researchers but is not commonly used for inspection environments, perhaps since there is little research to show how the simulations correlate to use in real-world environments.

In this thesis, existing data from bridge surveys was used to develop simulation environments and the experiments from the bridge environment were repeated in this simulation environment. In particular, the maps created using SfM and LiDAR data for AMCL were tested again using data collected using the simulated laser scanner in the simulated environment, and these results were compared to the results from Hector SLAM in the simulated environment. In general, the results in simulation gave similar errors to the results from AMCL in the bridge environment, but the errors for Hector SLAM were more like the laboratory environment. The use of simulation could be used to expand the possibilities for testing the localisation, mapping and navigation algorithms in different bridge configurations and modifications to the platform configuration, including weight, types of wheels or tracks

and is recommended for future research. Furthermore, the results in Chapter 4 showed that the addition of different sensors can be tested, such as wheel odometry sensors that were not implemented on the real robotic platform. Further tests and validation of the robot model in simulation should be performed with reference to laboratory and real-world environments. In addition, the work in this thesis could be compared to alternative methods for creating simulated environments, such as using CAD models.

Finally, Cadena et al. (2016) highlights an open research question concerning the degradation of sensor performance and the effect on SLAM reliability, particularly for safety-critical applications. One of the key challenges highlighted by Cadena et al. (2016) is the need for long-term testing of SLAM algorithms and associated hardware to validate the robustness of these methods. Many SLAM methods are also fragile in complex environments since SLAM is based on the assumption that the environment is static, which is rarely the case. It is recommended that future research focuses on the long-term assessment of robotic sensors and navigation methods for inspection environments.

6.2 Deep learning for crack segmentation

Test the effectiveness of methods from computer vision and deep-learning for the segmentation of cracks in images of concrete.

The second part of this thesis focused on inspection. The main requirements for bridge bearing inspection include detection of changes in geometry of the bearing or detection of deterioration of the surrounding structure, including cracking, crushing and build-up of debris. The work in this thesis focused on the use of computer vision and deep-learning methods for the detection of cracks in concrete. The data used in this thesis centred around existing datasets that were collected using RGB monocular cameras. RGB cameras were also the most reported sensor to be used onboard the robotics platforms, which makes this data and approach extendable to the real applications.

Initially, an existing approach for crack segmentation using a machine learning method was compared to a deep learning approach that has been previously been used for edge

detection (Xie, 2015; Xie and Tu, 2017). Most of the approaches for crack detection in images from the literature use a CNN-based deep learning approach to crack detection, with network architectures that have been previously used for object recognition and classification. As a result, the majority of available datasets for cracks in concrete comprise of images cut into patches; the patches are then labelled as true or false instances of a crack, rather than each pixel. In this thesis, the disadvantages of using patched-based approaches were discussed; the main disadvantage is the increase in the number of false positives that are detected by the classifier due to the similarity of the regions containing the crack and the background texture of the image. The deep learning approach used in this thesis (known as HED) takes a whole image as an input and learns features at different image scales, which are then combined to segment the crack from the image. However, there are relatively few datasets of cracks in concrete with a corresponding ground truth label at a pixel level. Only one other approach in recent literature was found to be using a related approach, but comparisons could not be made with this approach since their dataset and model implementation was not available.

Identify evaluation metrics from the literature and use these metrics for the evaluation of the machine learning and deep learning methods for crack segmentation.

The methods for evaluation of crack segmentation are inconsistently presented in the literature with the reported metrics including precision, recall, F_1 score and the reported values for these metrics ranging from averaged to maximum values across the dataset and on a per-image instance. Occasionally, machine learning methods from the broader literature were followed and receiver-operator characteristic curve or precision-recall curves were produced. Overall, this inconsistency makes it difficult to properly benchmark the differences in the methods reported in the literature. In this thesis, the difference between these metrics was discussed, and the case for discounting metrics that include the background as a true positive was presented due to the biases these pixels introduce. Taking these factors into account, the precision-recall curve, with F_1 metric was chosen as the

main tool for evaluation of the methods in this thesis and compared to additional metrics that accounted for the structure of the crack.

An existing approach called SFA was chosen for comparison to HED. SFA combined random forests and a SVM classifier to segment cracks at a pixel level. HED performed better than SFA when evaluated using the precision-recall curve with F_1 metric and a qualitative examination of the resulting grey-level images. The post-processing methods implemented by SFA were also tested in this thesis and improved the visual appearance of the cracks through morphological filtering. However, these post-processing steps did not improve the results using the pixel-based or structure-based evaluations. The performance of the S-measure method increased in a linear fashion when side-outputs of the HED network were examined, which when examined visually, showed the increasing definition of the segmented crack. Otherwise, the S-measure correlated well with the ranking of the F_1 metric. It is recommended that future research should focus on these metrics and perform correlation studies with other methods reported in the literature.

Identify and compare different datasets and the effects of training and testing using a mixture of these datasets on the chosen methods.

Generalisation of the HED network was tested through cross-dataset evaluation. Overall, varying the datasets- through increasing the number of training images and mixing the dataset with other images of cracks- appeared to have little impact on the results. Some increase in performance was found by changing the resolution of the input images. It was hypothesised that this increase in performance was due to the cracks accounting for a greater proportion of the pixels of the image when the resolution was increased.

This hypothesis was examined further by introducing images collected for the diagnosis of diseases in the retinal blood vessels. The cropped images of the retina had a comparable fractal dimension to the images of the cracks, since both have a branching structure. However, the images from the retinal dataset had a greater number of pixels that could be counted as a positive instance by the classifier by approximately a factor of 10. By introducing the images from the retinal dataset, it was thought that there would be an improvement

in the training of the networks for crack segmentation, since the network would be exposed to a greater amount of positive data examples.

The deep-learning network was then re-trained on images from the retinal database and the cracks database and compared using a precision-recall curve. The results from training HED using images of cracks and of the retina were comparable to the cross-dataset testing when two different datasets of cracks were combined. The qualitative results showed that increasing the number of retinal images increased the amount of background noise that was included in the segmentation. However, the HED networks trained using both images of the retina and images of cracks performed just as well as the network trained on images of just cracks. When tested on a dataset of cracks previously unseen in the training of the networks, all networks gave a comparable performance to the original SFA implementation.

Overall, there appears to be an upper limit to the achievable performance of the network, which was only overcome in this thesis by increasing the image resolution. The outcome of the deep learning networks appears to be sensitive to the input dataset, which may be indicative of overfitting of the model to the dataset, but also perhaps due to the difference in quality of the crack datasets compared to the retinal datasets. It is recommended that different network architectures should be compared to the results in this thesis.

The crack forest (CF) dataset was chosen as the main dataset for training and testing the HED network as it has been used in other approaches in the literature. However, the results in Chapter 5 showed that this dataset was affected by compression artefacts, which were also apparent when the model weights of the network were examined. The effect of dataset on the deep-learning results is unknown and further experimental work is required to better understand this problem. Overall, this thesis highlighted the need for a consistent dataset and for consistent methods of evaluation to allow better comparison between researchers.

Since the deep-learning approaches in this thesis performed better than the state of the art, it is recommended that real-world testing be performed, through controlled laboratory experiments and real world testing and in a variety of bridge environments.

6.3 Scope for future work

In this work, potential solutions for the mapping and localisation of robotic platforms in inspection environments were investigated. The next stages of this research could expand on the use of the simulated environment developed in Chapter 4 to refine the robotic platform and the combined localisation approach that was developed in this thesis. Complementary sensors and long-term testing, both in simulation and in the real environment, could be utilised to create a robust and reliable inspection system. In addition, research of path-planning in confined spaces and testing the methods from this work in different bridge configurations may be required.

The state of the art for deep-learning for crack segmentation in concrete and asphalt materials was also presented. This research could be expanded by implementing the model developed in Chapter 5 onto the robotic platform for evaluation in the real environment. The data collected on this platform could be used to improve the existing datasets from the literature, and to create a benchmark for similar inspection applications, but may also be sensitive to the detection of edges in the environment, and hence datasets with edges that may be found in real environments should be used for training. The data collected in Chapter 4 and Chapter 5 could also be used for other inspection applications, such as calculating changes in bearing geometry or using point cloud data for detecting anomalies over time, to further aid visual inspection of the bridge bearings.

References

Abdel-Qader, I., Abudayyeh, O., and Kelly, M. E. (2003). "Analysis of Edge-Detection Techniques for Crack Identification in Bridges". In: *Journal of Computing in Civil Engineering* 17.4, pp. 255–263. DOI: 10.1061/(ASCE)0887-3801(2003)17:4(255) (see p. 143).

Agnisarman, S., Lopes, S., Chalil Madathil, K., Piratla, K., and Gramopadhye, A. (2019). "A survey of automation-enabled human-in-the-loop systems for infrastructure visual inspection". In: *Automation in Construction* 97.November 2018, pp. 52–76. DOI: 10.1016/J.AUTCON.2018.10.019 (see pp. 5, 15, 21, 137).

Agüero, C., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., Gerkey, B., Paepcke, S., Rivero, J., Manzo, J., Krotkov, E., and Pratt, G. (2015). "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response". In: *IEEE Transactions on Automation Science and Engineering* 12.2, pp. 494–506 (see p. 118).

Ahmad, N., Ghazilla, R., Khairi, N., and Kasi, V. (2013). "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* 1.2, pp. 256–262. DOI: 10.12720/ijsps.1.2.256-262 (see p. 26).

Akutsu, A., Sasaki, E., Takeya, K., Kobayashi, Y., Suzuki, K., and Tamura, H. (2017). "A comprehensive study on development of a small-sized self-propelled robot for bridge inspection". In: *Structure and Infrastructure Engineering* 2479.October, pp. 1–12. DOI: 10.1080/15732479.2016.1236132 (see pp. 7, 13).

Aldea, E. and Le Hégarat-Mascle, S. (2015). "Robust crack detection for unmanned aerial vehicles inspection in an a-contrario decision framework". In: *Journal of Electronic Imaging* 24.6, p. 061119. DOI: 10.1117/1.JEI.24.6.061119 (see pp. 166, 174).

Alom, M. Z., Hasan, M., Yakopcic, C., Taha, T. M., and Asari, V. K. (2018). "Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation". In: *Cornell Computing Research Repository* abs/1802.06955. eprint: 1802.06955 (see p. 158).

Aria, M. and Akbari, R. (2013). "Inspection, condition evaluation and replacement of elastomeric bearings in road bridges". In: *Structure and Infrastructure Engineering* 9.9, pp. 918–934. DOI: 10.1080/15732479.2011.638171 (see pp. 1, 2).

Arth, C., Pirchheim, C., Ventura, J., Schmalstieg, D., and Lepetit, V. (2015). "Instant Outdoor Localization and SLAM Initialization from 2.5D Maps". In: *Transactions on Visualization and Computer Graphics* 21.11, pp. 1309–1318. DOI: 10.1109/TVCG.2015.2459772 (see p. 23).

Baimas, N. and McClean, J. (1998). "Mancunian Way Bearing Replacements". In: *Proceedings of the Institution of Civil Engineers - Municipal Engineer*. Vol. 127. 3, pp. 124–131. DOI: 10.1680/imuen.1998.30988 (see p. 3).

Ball, P. (1999). *The Self-Made Tapestry: Pattern Formation in Nature*. 7. New York, NY, USA: Oxford University Press, Inc., p. 421. DOI: 10.1119/1.880339 (see pp. 195, 197).

Bechar, A. and Vigneault, C. (2016). "Agricultural robots for field operations: Concepts and components". In: *Biosystems Engineering* 149, pp. 94–111. DOI: 10.1016/j.biosystemseng.2016.06.014 (see p. 14).

Behrooz, M., Yarra, S., Mar, D., Pinuelas, N., Muzinich, B., Publicover, N. G., Pekcan, G., Itani, A., and Gordaninejad, F. (2016). "A self-sensing magnetorheological elastomer-based adaptive bridge bearing with a wireless data monitoring system". In: *Proceedings of SPIE - The International Society for Optical Engineering*. Vol. 9803, p. 98030D. DOI: 10.1117/12.2218691 (see p. 2).

Behzadian, B., Agarwal, P., Burgard, W., and Tipaldi, G. D. (2015). "Monte Carlo localization in hand-drawn maps". In: *International Conference on Intelligent Robots and Systems*, pp. 4291–4296. DOI: 10.1109/IROS.2015.7353985 (see p. 82).

Burgard, W., Derr, A., Fox, D., and Cremers, A. (1998). "Integrating global position estimation and position tracking for mobile robots: the dynamic Markov localization approach". In: *Proceedings of the 1998 International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications*. Vol. 2, pp. 730–735. DOI: 10.1109/IROS.1998.727279 (see p. 48).

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: Towards the Robust-Perception Age". In: *Transactions on Robotics* 32.6, pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754 (see pp. 32, 42, 60, 82, 218).

Cha, Y. J., Choi, W., Suh, G., Mahmoudkhani, S., and Büyüköztürk, O. (2018). "Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple

Damage Types". In: *Computer-Aided Civil and Infrastructure Engineering* 33.9, pp. 731–747. DOI: 10.1111/mice.12334 (see p. 12).

Cha, Y.-J., Choi, W., and Büyüköztürk, O. (2017). "Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks". In: *Computer-Aided Civil and Infrastructure Engineering* 32.5, pp. 361–378. DOI: 10.1111/mice.12263 (see pp. 10, 141, 142, 165).

Chambon, S. and Moliard, J. (2011). "Automatic road pavement assessment with image processing: Review and comparison". In: *International Journal of Geophysics* 2011. DOI: 10.1155/2011/989354 (see p. 143).

Chen, S., Rice, C., Boyle, C., and Hauser, E. (2011). "Small-Format Aerial Photography for Highway-Bridge Monitoring". In: *Journal of Performance of Constructed Facilities* 25.2, pp. 105–112. DOI: 10.1061/(ASCE)CF.1943-5509.0000145 (see pp. 6, 9, 13, 83).

Chen, S., Truong-hong, L., Keeffe, E. O., Laefer, D. F., and Mangina, E. (2018). "Outlier detection of point clouds generating from low cost UAVs for bridge inspection". In: *The Sixth International Symposium on Life-Cycle Civil Engineering* January 2019, pp. 1969–1975 (see p. 12).

Chen, T. (2017). "Factors in Bridge Failure, Inspection, and Maintenance". In: *Journal of Performance of Constructed Facilities* 31.5, 04017070 (online document). DOI: 10.1061/(ASCE)CF.1943-5509.0001042 (see p. 3).

Cho, J. S., Park, J. C., Gil, H. B., Kim, H. J., and An, H. H. (2014). "Computer Vision Techniques for Bridge Bearing Condition Assessment using Visual Inspection Photographs". In: *IABSE Symposium Report*. Vol. 102. 9, pp. 2697–2704. DOI: 10.2749/222137814814070253 (see p. 1).

Cho, K., Kim, H. M., Jin, Y. H., Liu, F., Moon, H., Koo, J. C., and Choi, H. R. (2013). "Inspection robot for hanger cable of suspension bridge: Mechanism design and analysis". In: *Transactions on Mechatronics* 18.6, pp. 1665–1674. DOI: 10.1109/TMECH.2013.2280653 (see pp. 6, 11, 13).

Chong, Z. J., Qin, B., Bandyopadhyay, T., Ang, M. H., Frazzoli, E., and Rus, D. (2013). "Synthetic 2D LIDAR for precise vehicle localization in 3D urban environment". In: *Proceedings of the International Conference on Robotics and Automation*, pp. 1554–1559. DOI: 10.1109/ICRA.2013.6630777 (see p. 23).

Chu, K. W., Lee, W. S., Cheng, C. Y., Huang, C. F., Zhao, F., Lee, L. S., Chen, Y. S., Lee, C. Y., and Tsai, M. J. (2013). "Demonstration of lateral IGBTs in 4H-SiC". In: *Electron Device Letters* 34.2, pp. 286–288. DOI: 10.1109/LED.2012.2230240 (see p. 160).

Cox, I. J. and Wilfong, G. T. (1990). *Autonomous Robot Vehicles*, p. 462. DOI: 10.1007/978-1-4613-8997-2 (see p. 32).

Cubero-Fernandez, A., Rodriguez-Lozano, F. J., Villatoro, R., Olivares, J., and Palomares, J. M. (2017). "Efficient pavement crack detection and classification". In: *Eurasip Journal on Image and Video Processing* 2017.1. DOI: 10.1186/s13640-017-0187-0 (see pp. 140, 143, 145).

Darby, P. and Gopu, V. (2018). *Bridge Inspecting with Unmanned Aerial Vehicles*. https://digitalcommons.lsu.edu/transet_pubs/14/. Accessed: 2019-02-01 (see p. 7).

Davis, J. and Goadrich, M. (2006). "The Relationship Between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233–240. DOI: 10.1145/1143844.1143874 (see pp. 164, 165).

Droeschel, D., Schwarz, M., and Behnke, S. (2017). "Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner". In: *Robotics and Autonomous Systems* 88, pp. 104–115. DOI: 10.1016/j.robot.2016.10.017 (see p. 61).

Dueñas, F. M. (2015). "Stereo Visual SLAM for Mobile Robots Navigation". PhD thesis. University of Málaga (see p. 39).

Durrant-Whyte, H. and Bailey, T. (2006). "Simultaneous localization and mapping (SLAM)". In: *Robotics and Automation Magazine* 13.2, pp. 99–116. DOI: 10.1109/MRA.2006.1638022 (see pp. 27–29, 33, 34, 37).

Ellenberg, A., Kontsos, A., Moon, F., and Bartoli, I. (2016a). "Bridge deck delamination identification from unmanned aerial vehicle infrared imagery". In: *Automation in Construction* 72, pp. 155–165. DOI: 10.1016/j.autcon.2016.08.024 (see p. 9).

Ellenberg, A., Kontsos, A., Moon, F., and Bartoli, I. (2016b). "Bridge related damage quantification using unmanned aerial vehicle imagery". In: *Structural Control and Health Monitoring* 23.9, pp. 1168–1179. DOI: 10.1002/stc.1831 (see pp. 9, 13, 14, 77, 83).

BS-EN-1337 (2003). *British-European Standard 1337: Structural Bearings – Part 10 Inspection and maintenance* (see pp. 4, 12, 13, 137).

Eschmann, C. and Wundsam, T. (2017). "Web-Based Georeferenced 3D Inspection and Monitoring of Bridges with Unmanned Aircraft Systems". In: *Journal of Surveying Engineering* 143.3, pp. 1–10. DOI: 10.1061/(ASCE)SU.1943-5428.0000221 (see pp. 9, 13).

Everett, H. R. (1995). *Sensors for Mobile Robots: Theory and Application*. Natick, MA, USA: A. K. Peters, Ltd. (see pp. 21, 22, 25, 26).

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). "The PASCAL Visual Object Classes (VOC) Challenge". In: *International journal of computer vision* 88.2, pp. 303–338 (see p. 209).

Fabresse, L. (2018). "Evaluation of Out-of-the-box ROS 2D SLAMs for Autonomous Exploration of Unknown Indoor Environments". In: *International Conference on Intelligent Robotics and Applications*. August. DOI: 10.1007/978-3-642-40852-6 (see p. 60).

Fan, D., Cheng, M., Liu, Y., Li, T., and Borji, A. (2017). "Structure-Measure: A New Way to Evaluate Foreground Maps". In: *Proceedings of the International Conference on Computer Vision*, pp. 4558–4567. DOI: 10.1109/ICCV.2017.487 (see pp. 166–168, 184).

Fei-Fei, L., Deng, J., and Li, K. (2010). "ImageNet: Constructing a large-scale image database". In: *Journal of Vision* 9.8, pp. 1037–1037. DOI: 10.1167/9.8.1037 (see p. 145).

Filatov, A., Filatov, A., Krinkin, K., Chen, B., and Molodan, D. (2018). "2D SLAM quality evaluation methods". In: *Conference of Open Innovation Association, FRUCT*, pp. 120–126. DOI: 10.23919/FRUCT.2017.8250173 (see p. 60).

Flach, P. A. (2015). "Precision-Recall-Gain Curves : PR Analysis Done Right". In: *Proceedings of Advances in Neural Information Processing Systems*. Vol. 28, pp. 1–9 (see pp. 164, 165).

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pp. 343–349. DOI: 10.1.1.2.342 (see p. 48).

Freire, L. and de Brito, J. (2006). "Relationship between bearings type and their most common anomalies". In: *Proceedings of the 3rd International Conference on Bridge Maintenance, Safety and Management - Bridge Maintenance, Safety, Management, Life-Cycle Performance and Cost*, pp. 205–206. DOI: 10.1201/b18175 (see pp. 1, 2).

Freire, L., de Brito, J., and Correia, J. (2014). "Management system for road bridge structural bearings". In: *Structure and Infrastructure Engineering* 10.8, pp. 1068–1086. DOI: 10.1080/15732479.2013.788524 (see pp. 1–5, 16, 213).

Fuentes-Pacheco, J., Ruiz-Ascencio, J., and Rendón-Mancha, J. (2012). "Visual simultaneous localization and mapping: a survey". In: *Artificial Intelligence Review* 43.1, pp. 55–81. DOI: 10.1007/s10462-012-9365-8 (see pp. 34, 39, 40).

Galvez-López, D. and Tardos, J. (2012). "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *Transactions on Robotics* 28.5, pp. 1188–1197. DOI: 10.1109/TRO.2012.2197158 (see p. 46).

Gibb, S., La, H., and Louis, S. (2018). "A Genetic Algorithm for Convolutional Network Structure Optimization for Concrete Crack Detection". In: *Congress on Evolutionary Computation*, pp. 1–8 (see pp. 10, 142).

Gibb, S., Le, T., La, H., Schmid, R., and Berendsen, T. (2017). "A Multi-functional Inspection Robot for Civil Infrastructure Evaluation and Maintenance". In: *International Conference on Intelligent Robots and Systems*, pp. 2672–2677 (see pp. 7, 16, 78).

Giubilato, R., Chiodini, S., Pertile, M., and Debei, S. (2018). "An Experimental Comparison of ROS-compatible Stereo Visual SLAM Methods for Planetary Rovers". In: *International Workshop on Metrology for AeroSpace*. September. DOI: 10.1109/MetroAeroSpace.2018.8453534 (see p. 61).

Goodfellow, I., Bengio, Y., and Courville, A. (2015). "Deep Learning". In: *Nature Methods* 13.1, pp. 35–35. DOI: 10.1038/nmeth.3707 (see pp. 147, 148, 160, 161).

Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W. (2010). "A tutorial on graph-based SLAM". In: *Intelligent Transportation Systems Magazine* 2.4, pp. 31–43. DOI: 10.1109/MITS.2010.939925 (see pp. 29, 35, 36).

Großmann, A. and Poli, R. (1999). "Robust mobile robot localisation from sparse and noisy proximity readings". In: *Proceedings of IJCAI-99 Workshop on Reasoning with Uncertainty in Robot Navigation (RUR-99)*. Vol. 37, pp. 1–18 (see p. 22).

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, L., Wang, G., Cai, J., and Chen, T. (2015). "Recent Advances in Convolutional Neural Networks". In: *Pattern Recognition* 77, pp. 354–377. DOI: 10.3389/fpsyg.2013.00124 (see pp. 147, 148).

Guo, S., Gao, Y., Wang, K., and Li, T. (2018). "Deeply supervised neural network with short connections for retinal vessel segmentation". In: *Cornell Computing Research Repository* abs/1803.03963 (see pp. 158, 166).

Hallermann, N. and Morgenthal, G. (2014). "Visual inspection strategies for large bridges using Unmanned Aerial Vehicles (UAV)". In: *Bridge Maintenance, Safety, Management and Life Extension*. July. DOI: 10.1201/b17063-96 (see pp. 7, 9, 13, 77, 78, 118).

Heinrich, M., Sperl, A., Mittmann, U., and Henkel, P. (2018). "Reliable multi-GNSS real-time kinematic positioning". In: *Proceedings - International Symposium Electronics in Marine*. September, pp. 103–108. DOI: 10.23919/ELMAR.2018.8534600 (see p. 25).

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments". In: *The International Journal of Robotics Research* 31.5, pp. 647–663. DOI: 10.1177/0278364911434148 (see p. 24).

Hiasa, S., Karaaslan, E., Shattenkirk, W., Mildner, C., and Catbas, F. (2018). "Bridge Inspection and Condition Assessment Using Image-Based Technologies with UAVs". In: *Structures Congress 2018: Bridges, Transportation Structures, and Nonbuilding Structures - Selected Papers from the Structures Congress 2018*. Vol. 2018-April, pp. 217–228. DOI: doi:10.1061/9780784481332.020 (see pp. 9, 13, 77, 78).

Hidalgo, F. and Braunl, T. (2015). "Review of underwater SLAM techniques". In: *ICARA 2015 - Proceedings of the 2015 6th International Conference on Automation, Robotics and Applications*, pp. 306–311. DOI: 10.1109/ICARA.2015.7081165 (see p. 34).

Hoeke, L., Singh, P., Moser, R., Kahn, L., and Kurtis, K. (2009). "Degradation of steel girder bridge bearing systems by corrosion". In: *National Association of Corrosion Engineers* (see p. 2).

Hoffer, N., Coopmans, C., Dorafshan, S., Maguire, M., Hoffer, N. V., and Coopmans, C. (2017). "Challenges in Bridge Inspection Using Small Unmanned Aerial Systems : Results and Lessons Learned". In: *International Conference on Unmanned Aircraft Systems*, pp. 1722–1730. DOI: 10.1109/ICUAS.2017.7991459 (see pp. 6, 7, 9, 13).

Hoover, A. (2000). "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response". In: *Transactions on Medical Imaging* 19.3, pp. 203–210. DOI: 10.1109/42.845178 (see pp. 144, 195).

Hoskere, V., Narazaki, Y., Hoang, T., and Spencer, B. (2018). "Vision-based Structural Inspection using Multiscale Deep Convolutional Neural Networks". In: *Cornell Computing Research Repository*. eprint: 1805.01055 (see p. 12).

Jahanshahi, M., Shen, W., Mondal, T., Abdelbarr, M., Masri, S., and Qidwai, U. (2017). "Reconfigurable swarm robots for structural health monitoring: a brief review". In: *International Journal of Intelligent Robotics and Applications* 1.3, pp. 287–305. DOI: 10.1007/s41315-017-0024-8 (see pp. 5, 7, 11).

Jalón-Monzón, A., De León, C. G. R., Alvarez-Múgica, M., Méndez-Ramírez, S., Hevia-Suárez, M. Á., and Escaf-Barmadah, S. (2016). "RESCUER: Development of a Modular Chemical, Biological, Radiological, and Nuclear Robot for Intervention, Sampling, and Situation Awareness". In: *Journal of Field Robotics* 33.7, pp. 931–945. DOI: 10.1002/rob.21588 (see p. 14).

Javadnejad, F., Gillins, D. T., Higgins, C. C., and Gillins, M. N. (2017). "BridgeDex : Proposed Web GIS Platform for Managing and Interrogating Multiyear and Multiscale Bridge-Inspection Images". In: *Journal of Computing in Civil Engineering* 31.6, p. 04017061. DOI: 10.1061/(ASCE)CP.1943-5487.0000710 (see pp. 5, 83).

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). "Poisson Surface Reconstruction". In: *Symposium on Geometry Processing* (see p. 120).

Khaloo, A., Lattanzi, D., Cunningham, K., Dell'Andrea, R., and Riley, M. (2018). "Unmanned aerial vehicle inspection of the Placer River Trail Bridge through image-based 3D modelling". In: *Structure and Infrastructure Engineering* 14.1, pp. 124–136. DOI: 10.1080/15732479.2017.1330891 (see pp. 9, 13).

Kim, C., Son, H., Hwang, N., and Kim, C. (2014). "Rapid and automated determination of rusted surface areas of a steel bridge for robotic maintenance systems". In: *Automation in Construction* 42, pp. 13–24. DOI: 10.1016/j.autcon.2014.02.016 (see pp. 6, 12, 83).

Klein, G. and Murray, D. (2007). "Parallel Tracking and Mapping for Small AR Workspaces". In: *International Symposium on Mixed and Augmented Reality*, pp. 1–10. DOI: 10.1109/ISMAR.2007.4538852 (see p. 61).

Koenig, N. and Howard, A. (2004). "Design and Use Paradigms for Gazebo , An Open-Source Multi-Robot Simulator". In: *International Conference on Intelligent Robots and Systems*, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727 (see pp. 15, 118, 217).

Kohlbrecher, S. (2016). *Gazebo cannot handle collada files as collision mesh*. https://bitbucket.org/osrf/gazebo/issues/2072/gazebo-cannot-handle-collada-files-as. Accessed: 2019-02-01 (see p. 119).

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., von Stryk, O., and Klingauf, U. (2014). "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots". In: *RoboCup 2013: RoboCup 2013: Robot World Cup*. Vol. XVII, pp. 624–631. DOI: 10.1007/978-3-662-44468-9_58 (see pp. 43, 61, 80).

Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U. (2011). "A flexible and scalable SLAM system with full 3D motion estimation". In: *Proceedings of the International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160. DOI: 10.1109/SSRR.2011.6106777 (see pp. 43, 44, 58, 61, 80).

Kooi, S. (2013). *Using VisualSFM and Meshlab for creating 3-D models Guidelines for using SfM in field archaeology*. http://www.academia.edu/10190411/Using_VisualSFM_and_Meshlab_for_creating_3-D_models_Guidelines_for_using_SfM_in_field_archaeology. Accessed: 2019-02-01 (see p. 120).

Kramer, J. and Scheutz, M. (2007). "Development environments for autonomous mobile robots: A survey". In: *Autonomous Robots* 22.2, pp. 101–132. DOI: 10.1007/s10514-006-9013-8 (see p. 68).

Kulich, M., Lhotský, V., and Přeučil, L. (2017). "Practical Aspects of Autonomous Exploration with a Kinect2 sensor". In: *Cornell Computing Research Repository*, pp. 2–3 (see p. 24).

Kurtzer, G., V, S., and Bauer, M. (2017). "Singularity: Scientific containers for mobility of compute." In: *PLoS ONE* 12.5. DOI: 10.1371/journal.pone.0177459 (see pp. 80, 160).

La, H. M., Lim, R. S., Basily, B., Gucunski, N., Yi, J., Maher, A., and Romero, F. A. (2013). "Autonomous Robotic System for High-Efficiency Non-Destructive Bridge Deck Inspection and Evaluation". In: *Proceedings of the International Conference on Automation Science and Engineering*, pp. 1053–1058. DOI: 10.1109/CoASE.2013.6653886 (see pp. 10, 78).

La, H., Gucunski, N., Kee, S., Yi, J., Senlet, T., and Van Nguyen, L. (2014). "Autonomous robotic system for bridge deck data collection and analysis". In: *International Conference on Intelligent Robots and Systems*, pp. 1950–1955. DOI: 10.1109/IROS.2014.6942821 (see pp. 7, 10).

La, H., Lim, R., Basily, B., Gucunski, N., Yi, J., Maher, A., Romero, F., and Parvardeh, H. (2013). "Mechatronic Systems Design for an Autonomous Robotic System for High-Efficiency Bridge Deck Inspection and Evaluation". In: *Transactions on Mechatronics*. Vol. 18. 6, pp. 1655–1664. DOI: 10.1109/TMECH.2013.2279751 (see pp. 6, 7, 10, 78).

Labbe, M., Michaud, F., and Labb, M. (2014). "Online global loop closure detection for large-scale multi-session graph-based SLAM". In: *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 2661–2666. DOI: 10.1109/IROS.2014.6942926 (see p. 61).

Lattanzi, D. and Miller, G. (2015). "3D Scene Reconstruction for Robotic Bridge Inspection". In: *Journal of Infrastructure Systems* 21.2, p. 04014041. DOI: 10.1061/(ASCE)IS.1943-555X.0000229 (see pp. 12, 13, 83, 119).

Lattanzi, D. and Miller, G. (2017). "Review of Robotic Infrastructure Inspection Systems". In: *Journal of Infrastructure Systems* 23.3, e–page: 04017004. DOI: 10.1061/(ASCE)IS.1943-555X.0000353 (see pp. 5–8, 16, 65, 77, 137).

Le Cun, Y., Boser, B., Denker, J., Howard, R., Habbard, W., Jackel, L., and Henderson, D. (1990). "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2*, pp. 396–404 (see p. 147).

Le, T., Gibb, S., Pham, N., La, H., Falk, L., and Berendsen, T. (2017). "Autonomous robotic system using non-destructive evaluation methods for bridge deck inspection". In: *Proceedings of the International Conference on Robotics and Automation*, pp. 3672–3677. DOI: 10.1109/ICRA.2017.7989421 (see pp. 6, 10, 83).

Leberl, F., Irschara, ., Pock, T., Meixner, P., Gruber, M., Scholz, S., and Wiechert, A. (2010). "Point Clouds: Lidar versus 3D Vision". In: *Photogrammetric Engineering and Remote Sensing* 76.10, pp. 1123–1134. DOI: 0099-1112/10/7610–1123 (see p. 51).

Lee, B., Shin, D., Seo, J., Jung, J. D., and Lee, J. (2012). "Intelligent bridge inspection using remote controlled robot and image processing technique". In: *Proceedings of the* 28^{th} *International Symposium on Automation and Robotics in Construction* (see pp. 8, 11).

Lee, C., Xie, S., and Gallagher, P. (2015). "Deeply-Supervised Nets". In: *Cornell Computing Research Repository*, arXiv:1409.5185 (see p. 150).

Lee, J. H., Lee, J. M., Kim, H. J., and Moon, Y. S. (2008). "Machine vision system for automatic inspection of bridges". In: *Proceedings of the 1st International Congress on Image and Signal Processing*, pp. 363–366. DOI: 10.1109/CISP.2008.672 (see p. 12).

Lee, M., Hanczor, M., Chu, J., He, Z., Michael, N., and Whittaker, R. (2018). "3-D Volumetric Gamma-ray Imaging and Source Localization with a Mobile Robot". In: *Cornell Computing Research Repository* abs/1802.06072 (see pp. 14, 77).

Li, G., Zhao, X., Du, K., Ru, F., and Zhang, Y. (2017). "Recognition and evaluation of bridge cracks with modified active contour model and greedy search-based support vector machine". In: *Automation in Construction* 78, pp. 51–61. DOI: 10.1016/j.autcon.2017.01.019 (see p. 140).

Li, H., Song, D., Liu, Y., and Li, B. (2017). "Automatic Pavement Crack Detection by Multi-Scale Image Fusion". In: *Transactions on Intelligent Transportation Systems* 99, pp. 1–12 (see pp. 138, 143, 155).

Li, L., Sun, L., Guobao, N., and Shengguang, T. (2014). "Automatic pavement crack recognition based on BP neural network". In: *International Conference on Computer and Electrical Engineering*, pp. 11–22 (see pp. 12, 13, 139).

Li, Z., Zhu, C., and Gold, C. (2004). *Digital Terrain Modeling: Principles and Methodology*. CRC Press (see p. 118).

Lim, R., La, H., Shan, Z., and Sheng, W. (2011). "Developing a crack inspection robot for bridge maintenance". In: *Proceedings of the International Conference on Robotics and Automation*, pp. 6288–6293. DOI: 10.1109/ICRA.2011.5980131 (see pp. 7, 10, 138).

Liu, D., Dissanayake, G., Miro, J. V., and Waldron, K. J. (2014). "Infrastructure Robotics: Research Challenges and Opportunities". In: *Proceedings of the 31st International Symposium on Automation and Robotics in Construction*, p. 1 (see p. 16).

Liu, J., Miao, X., and Yuan, Y. (2012). "The rail bridge bearing monitoring system base on FBG". In: *Proceedings of the 4th International Conference on Electronics, Communications and Networks*. DOI: 10.1117/12.968607 (see p. 2).

Long, J., Shelhamer, E., and Darrell, T. (2015). "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440 (see p. 150).

Luque-Vega, L. F., Castillo-Toledo, B., Loukianov, A., and Gonzalez-Jimenez, L. E. (2014). "Power line inspection via an unmanned aerial system based on the quadrotor helicopter". In: *Proceedings of the Mediterranean Electrotechnical Conference - MELECON*. IEEE, pp. 393–397. DOI: 10.1109/MELCON.2014.6820566 (see p. 24).

Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., and Omata, H. (2018). "Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone". In: *Cornell Computing Research Repository* abs/1801.09454, pp. 4–6. eprint: 1801.09454 (see p. 141).

Maity, S., Saha, A., and Bhowmick, B. (2018). "Edge SLAM: Edge Points Based Monocular Visual SLAM". In: *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*. Vol. 2018-Janua, pp. 2408–2417. DOI: 10.1109/ICCVW.2017.284 (see p. 103).

Maizuar, M., Zhang, L., Miramini, S., Mendis, P., and Thompson, R. G. (2017). "Detecting structural damage to bridge girders using radar interferometry and computational modelling". In: *Structural Control and Health Monitoring* 24, e–record 1985. DOI: 10.1002/stc.1985 (see p. 2).

Makantasis, K., Protopapadakis, E., and Doulamis, A. (2015). "Deep Convolutional Neural Networks for Efficient Vision Based Tunnel Inspection". In: *International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 335–342. DOI: 10.1109/ICCP.2015.7312681 (see p. 140).

Mandelbrot, B. B. (1982). *The Fractal Geometry of Nature*. WH freeman New York (see p. 195).

Maninis, K.-k., Pont-tuset, J., Arbel, P., and Gool, L. V. (2016). "Deep Retinal Image Understanding". In: *Medical Image Computing and Computer-Assisted Intervention* (see pp. 147, 198, 199, 209).

Markom, M., Aniza, S., Shukor, A., Adom, A. H., Sulino, E., Muslim, M., and Shakaff, A. Y. (2016). "Indoor Scanning and Mapping using Mobile Robot and RP Lidar". In: 3.1, pp. 42–47 (see p. 23).

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics". In: *Proceedings of the International Conference on Computer Vision*. July (see p. 145).

Mazumdar, A. and Asada, H. (2009). "Mag-Foot: A steel bridge inspection robot". In: *Proceeding of the International Conference on Intelligent Robots and Systems*, pp. 1691–1696. DOI: 10.1109/IROS.2009.5354599 (see p. 7).

McLoughlin, B., Pointon, H., McLoughlin, J. P., Shaw, A., and Bezombes, F. (2018). "Uncertainty characterisation of mobile robot localisation techniques using optical surveying grade instruments". In: *Sensors* 18.7. DOI: 10.3390/s18072274 (see p. 16).

Meeussen, W. (2010). *REP 105: Coordinate Frames for Mobile Platforms*. http://www.ros.org/reps/rep-0105.html (YEAR last accessed: 01/02/19) (see p. 70).

Menendez, E., Victores, J. G., Montero, R., Martínez, S., and Balaguer, C. (2018). "Tunnel structural inspection and assessment using an autonomous robotic system". In: *Automation in Construction* 87.November 2016, pp. 117–126. DOI: 10.1016/j.autcon.2017.12.001 (see p. 15).

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., and Stryk, O. von (2012). "Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo". In: *Simulation, Modeling, and Programming for Autonomous Robots*, pp. 400–411 (see p. 118).

Micheletti, N., Chandler, J. H., Lane, S. N., Prosdocimi, M., Calligaro, S., Sofia, G., Dalla Fontana, G., Tarolli, P., Schenk, T., Micheletti, N., Chandler, J. H., and Lane, S. N. (2015). "Structure from Motion (SfM) Photogrammetry Photogrammetric heritage". In: *Department of Civil and Environmental Engineering and Geodetic Science, The Ohio State University* 2.14, pp. 1–12. DOI: 10.1002/esp.3767 (see p. 51).

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence* 68.2, pp. 593–598. DOI: 10.1.1.16.2153 (see p. 39).

Moore, T. and Stouch, D. (2016). "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, pp. 335–348. DOI: 10.1007/978-3-319-08338-4_25 (see p. 48).

Morena, M., Manhães, M., Sebastian, A., Voss, M., Douat, L. R., and Rauschenbach, T. (2016). "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation". In: *Oceans 2016*. IEEE. DOI: 10.1109/oceans.2016.7761080 (see pp. 119, 122).

Moutarlier, P. (1989). "Stochastic Multisensory Data Fusion For Mobile Robot Location And Environment Modeling". In: *Proceedings of the International Symposium on Robotics Research* (see p. 32).

Mur-Artal, R., Montiel, J., and Tardos, J. (2015). "ORB-SLAM : a Versatile and Accurate Monocular SLAM System". In: *Transactions on Robotics* 31.5, pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671 (see pp. 45–47, 61, 80, 102, 105).

Mur-Artal, R. and Tardos, J. (2017). "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *Transactions on Robotics* 33.5, pp. 1255–1262. DOI: 10.1109/TRO.2017.2705103 (see pp. 45, 46, 102, 105).

Murphy, K. (2000). "Bayesian map learning in dynamic environments". In: *Advances in Neural Information Processing Systems* 12, pp. 1015–1021. DOI: 10.1.1.21.3240 (see pp. 22, 37).

Murphy, R. R., Steimle, E., Hall, M., Lindemuth, M., Trejo, D., Hurlebaus, S., Medina-Cetina, Z., and Slocum, D. (2011). "Robot-assisted bridge inspection". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 64.1, pp. 77–95. DOI: 10.1007/s10846-010-9514-8 (see p. 11).

Nayyeri, F. and Hou, L. (2016). "Foreground-–background separation technique for crack detection". In: *Computer-Aided Civil and Infrastructure Engineering*, pp. 1–14. DOI: 10.1111/mice.12428 (see pp. 138, 144).

Neves, F., Sobreira, H., Campos, D., and Morais, R. (2015). "Towards a Reliable Monitoring Robot for Mountain Vineyards". In: *International Conference on Autonomous Robot Systems and Competitions*, pp. 37–43. DOI: 10.1109/ICARSC.2015.21 (see p. 118).

Nhat-Duc, H., Nguyen, Q. L., and Tran, V. D. (2018). "Automatic recognition of asphalt pavement cracks using metaheuristic optimized edge detection algorithms and convolution neural network". In: *Automation in Construction* 94.July, pp. 203–213. DOI: 10.1016/j.autcon.2018.07.008 (see p. 141).

Niemierko, A. (2016). "Modern Bridge Bearings and Expansion Joints for Road Bridges". In: *Transportation Research Procedia* 14, pp. 4040–4049. DOI: 10.1016/j.trpro.2016.05.501 (see p. 1).

Nilsson, F. (2008). *Thermal Cameras*. CRC Press, p. 110. DOI: 10.1201/9781420061574 (see p. 25).

Nitsche, M., Pire, T., Krajník, T., Kulich, M., and Mejail, M. (2014). "Monte Carlo localization for teach-and-repeat feature-based navigation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8717 LNAI, pp. 13–24. DOI: 10.1007/978-3-319-10401-0_2 (see p. 61).

Noh, Y., Koo, D., Kang, Y., Park, D. G., and Lee, D. H. (2017). "Automatic crack detection on concrete images using segmentation via fuzzy C-means clustering". In: *Proceedings of the International Conference on Applied System Innovation: Applied System Innovation for Modern Technology*, pp. 877–880. DOI: 10.1109/ICASI.2017.7988574 (see pp. 2, 13, 143).

Nüchter, A., Bleier, M., Schauer, J., and Peter Janotta (2018). "Continuous-Time SLAM—Improving Google's Cartographer 3D Mapping". In: *Latest Developments in Reality-Based 3D Surveying and Modelling*, pp. 53–73 (see p. 23).

Oh, J., Lee, A., Oh, S., Choi, Y., Yi, B., and Yang, H. (2007). "Design and Control of Bridge Inspection Robot System". In: *International Conference on Mechatronics and Automation*, pp. 3634–3639 (see p. 8).

Open Source Robotics Foundation (2014). *Import Meshes*. http://gazebosim.org/tutorials?tut=import_mesh&cat=build_robot. Accessed: 2019-02-01 (see p. 119).

Özgenel, Ç. F. and Sorguç, A. G. (2018). "Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings". In: *Proceedings of the 35th International Symposium on Automation and Robotics in Construction* (see pp. 142, 143, 145, 146).

Panboonyuen, T., Jitkajornwanich, K., Lawawirojwong, S., Srestasathiern, P., and Vateekul, P. (2017). "Road segmentation of remotely-sensed images using deep convolutional neural networks with landscape metrics and conditional random fields". In: *Remote Sensing* 9.7, pp. 1–19. DOI: 10.3390/rs9070680 (see p. 144).

Parkinson, B. and Gilbert, S. (1983). "NAVSTAR: Global positioning system—Ten years later". In: *Proceedings of the IEEE* 71.10, pp. 1177–1186. DOI: 10.1109/PROC.1983.12745 (see p. 25).

Pauly, L., Peel, H., Luo, S., Hogg, D., and Fuentes, R. (2017). "Deeper Networks for Pavement Crack Detection". In: *Proceedings of the 34th International Symposium on Automation and Robotics in Construction*. DOI: https://doi.org/10.22260/ISARC2017/0066 (see p. 141).

Peel, H., Morgan, G., Peel, C., Cohn, A., and Fuentes, R. (2016). "Inspection robot with low cost perception sensing". In: *Proceedings of the 33rd International Symposium on Automation and Robotics in Construction*. Vol. 33. DOI: 10.22260/ISARC2016/0008 (see p. 22).

Pham, N. and La, H. (2016). "Design and implementation of an autonomous robot for steel bridge inspection". In: *54th Annual Allerton Conference on Communication, Control, and Computing*. Vol. 10. January, pp. 556–562. DOI: 10.1109/ALLERTON.2016.7852280 (see pp. 6, 8, 10, 13, 77, 78).

Philippe, M. and Chatila, R. (1990). "An Experimental System for Incremental Environment Modelling by an Autonomous Mobile Robot". In: *Experimental Robotics I*, pp. 327–346. DOI: 10.1007/BFb0042528 (see p. 32).

Ponnu, G., George, J., and Skovira, J. (2016). *Real-time ROSberryPi SLAM Robot*. Masters of Engineering Design Project (see pp. 45, 61, 64, 80).

Portugal, D., Cabrita, G., Gouveia, B. D., Santos, D. C., and Prado, J. (2015). "An autonomous all terrain robotic system for field demining missions". In: *Robotics and Autonomous Systems* 70, pp. 126–144. DOI: 10.1016/j.robot.2015.02.013 (see p. 118).

Post, M. A., Bianco, A., and Yan, X. T. (2017). "Autonomous Navigation with ROS for a Mobile Robot in Agricultural Fields". In: *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*. Vol. 2, pp. 79–87. DOI: 10.5220/0006434400790087 (see p. 15).

Prasanna, P., Dana, K. J., Gucunski, N., Basily, B. B., La, H. M., Lim, R. S., and Parvardeh, H. (2016). "Automated Crack Detection on Concrete Bridges". In: *Transactions on Automation Science and Engineering*. Vol. 13. 2, pp. 591–599. DOI: 10.1109/TASE.2014.2354314 (see pp. 2, 13, 139, 143, 165).

Protopapadakis, E., Stentoumis, C., Doulamis, N., Doulamis, A., Loupos, K., Makantasis, K., Kopsiaftis, G., and Amditis, A. (2016). "Autonomous robotic inspection in tunnels". In: *Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* 3.5, pp. 12–19. DOI: 10.5194/isprsannals-III-5-167-2016 (see pp. 6, 15).

Qian, W., Xia, Z., Xiong, J., Gan, Y., and Guo, Y. (2014). "Manipulation Task Simulation using ROS and Gazebo". In: *International Conference on Robotics and Biomimetics*. IEEE, pp. 2594–2598 (see p. 118).

Quigley, M., Berger, E., and Ng, A. Y. (2007). "STAIR : Hardware and Software Architecture". In: *AAAI 2007 Robotics Workshop, Vancouver, BC*, pp. 31–37. DOI: 10.1.1.88.7903 (see p. 68).

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software* (see p. 68).

Ratsamee, P., Kriengkomol, P., Arai, T., Kamiyama, K., Mae, Y., Kiyokawa, K., Mashita, T., Uranishi, Y., and Takemura, H. (2016). "A hybrid flying and walking robot for steel bridge inspection". In: *International Symposium on Safety, Security and Rescue Robotics*, pp. 62–67. DOI: 10.1109/SSRR.2016.7784278 (see pp. 7, 11, 13).

Ridao, P., Carreras, M., Ribas, D., and Garcia, R. (2010). "Visual Inspection of Hydroelectric Dams Using an Autonomous Underwater Vehicle". In: *Journal of Field Robotics* 27.6, pp. 759–778 (see p. 16).

Rohde, J., Jatzkowski, I., Mielenz, H., and Z, J. M. (2016). "Vehicle Pose Estimation in Cluttered Urban Environments Using Multilayer Adaptive Monte Carlo Localization". In: *Proceedings of the International Conference on Information Fusion* (see pp. 61, 80).

Ronneberger, O., Fischer, P., and Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention, Springer, LNCS, Vol.9351: 234–241, 2015*, pp. 1–8. DOI: 10.1007/978-3-319-24574-4_28 (see p. 210).

Rossow, M. (2006). "Section 9: Inspection of Bridge Bearings (BIRM)". In: *Federal Highway Administration Bridge Inspector's Reference Manual*. Chap. 9, pp. 9.1.1–9.1.36 (see p. 2).

Ryan, T. W., Eric Mann, J., Chill, Z. M., and Ott, B. T. (2012). *Bridge Inspector's Reference Manual* (see pp. 1, 2).

Sacks, R., Kedar, A., Borrmann, A., Ma, L., Brilakis, I., Hüthwohl, P., Daum, S., Kattel, U., Yosef, R., Liebich, T., Barutcu, B. E., and Muhic, S. (2018). "SeeBridge as next generation bridge inspection: Overview, Information Delivery Manual and Model View Definition". In: *Automation in Construction* 90.May 2017, pp. 134–145. DOI: 10.1016/j.autcon.2018.02.033 (see pp. 5, 83).

Sakagami, T. (2015). "Remote nondestructive evaluation technique using infrared thermography for fatigue cracks in steel bridges". In: *Fatigue and Fracture of Engineering Materials and Structures* 38.7, pp. 755–779. DOI: 10.1111/ffe.12302 (see p. 12).

Salari, E. and Ouyang, D. (2012). "An image-based pavement distress detection and classification". In: *International Conference on Electro Information Technology*. DOI: 10.1109/EIT.2012.6220706 (see p. 197).

Sanchez-Cuevas, P., Heredia, G., and Ollero, A. (2017). "Multirotor UAS for bridge inspection by contact using the ceiling effect". In: *International Conference on Unmanned Aircraft Systems, ICUAS 2017*, pp. 767–774. DOI: 10.1109/ICUAS.2017.7991412 (see p. 9).

Sato, Y. (2018). "Crack Detection on Concrete Surfaces Using V-shaped Features". In: *The World of Computer Science and Information Technology Journal* 8.1, pp. 1–6 (see pp. 12, 139, 143).

Shi, Y., Cui, L., Qi, Z., Meng, F., and Chen, Z. (2016). "Structured Forests". In: *Transactions on Intelligent Transportation Systems*. Vol. 17. 12, pp. 1–12. DOI: 10.1109/TITS.2016.2552248 (see pp. 138, 140, 153–155, 164, 170, 172, 174, 176, 209).

Shiau, Y., Huang, C., Wang, M., and Zeng, J. (2008). "Discussion of Pot Bearing for Concrete Bridge". In: *Proceedings from the 25th International Symposium on Automation and Robotics in Construction*, pp. 213–223. DOI: 10.22260/ISARC2008/0033 (see p. 2).

Shibasaki, N., Ikeda, C., and Sakano, C. (2016). "Evaluation of bridge bearing performance based on the field measurement". In: *Maintenance, Monitoring, Safety, Risk and Resilience of Bridges and Bridge Networks - Proceedings of the 8th International Conference on Bridge Maintenance, Safety and Management*, pp. 870–877 (see p. 2).

Siciliano, B. and Khatib, O. (2016). "Springer handbook of robotics". In: (see pp. 27, 29–33, 36–38, 41).

Siegwart and Nourbaksh (2011). *Introduction to Autonomous Mobile Robots*. MIT press (see pp. 23, 25, 26, 36, 39).

Sim, R., Elinas, P., Griffin, M., and Little, J. J. (2005). "Vision-based SLAM using the Rao-Blackwellised Particle Filter". In: *Workshop on Reasoning with Uncertainty in Robotics*. Vol. 14, pp. 9–16. DOI: 10.1631/jzus.A071361 (see p. 39).

Simonyan, K. and Zisserman, A. (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556*, pp. 1–14. DOI: 10.1016/j.infsof.2008.09.005 (see pp. 144, 146, 148, 150, 158, 159).

Smith, R., Self, M., and Cheeseman, P. (1987). "Estimating Uncertain Spatial Relationships in Robotics". In: *International Conference on Robotics and Automation* (see p. 32).

Sommer, A. M., Nowak, A. S., and Thoft-Christensen, P. (1993). "Probability-Based Bridge Inspection Strategy". In: *Journal of Structural Engineering* 119.12, pp. 3520–3536. DOI: 10.1061/(ASCE)0733-9445(1993)119:12(3520) (see p. 5).

Spuler, T., Meng, N., and Moor, G. (2015). "Life-cycle costs of bridge bearings – Key considerations for bridge designers and owners". In: *Multi-Span Large Bridges - Proceedings of the International Conference on Multi-Span Large Bridges, 2015*, pp. 743–750. DOI: 10.1201/b18567-95 (see p. 3).

Staal, J., Abràmoff, M., Niemeijer, M., Viergever, M., and Van Ginneken, B. (2004). "Ridge-based vessel segmentation in color images of the retina". In: *Transactions on Medical Imaging* 23.4, pp. 501–509. DOI: 10.1109/TMI.2004.825627 (see pp. 144, 195).

Su, T. (2013). "Application of Computer Vision to Crack Detection of Concrete Structure". In: *International Journal of Engineering and Technology* 5.4, pp. 457–461. DOI: 10.7763/IJET.2013.V5.596 (see pp. 138, 143).

Sutter, B., Lelevé, A., Pham, M. T., Gouin, O., Jupille, N., Kuhn, M., Lulé, P., Michaud, P., and Rémy, P. (2018). "A semi-autonomous mobile robot for bridge inspection". In: *Automation in Construction* 91.May 2017, pp. 111–119. DOI: 10.1016/j.autcon.2018.02.013 (see pp. 5, 8, 11, 16, 77, 83, 137, 213).

Takada, Y., Ito, S., and Imajo, N. (2017). "Development of a Bridge Inspection Robot Capable of Traveling on Splicing Parts". In: *Inventions* 2.3, p. 22. DOI: 10.3390/inventions2030022 (see pp. 6, 11).

Thrun, S., Burgard, W., and Fox, D. (2005). "Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)". In: The MIT Press (see pp. 48, 49, 61, 82).

Toldo, R., Gherardi, R., Farenzena, M., and Fusiello, A. (2015). "Hierarchical structure-and-motion recovery from uncalibrated images". In: *Computer Vision and Image Understanding* 140, pp. 127–143. DOI: 10.1016/j.cviu.2015.05.011 (see p. 51).

Torrey, L. and Shavlik, J. (2009). "Transfer Learning". In: *Handbook of Research on Machine Learning Applications*, pp. 1–22. DOI: 10.1016/j.jbi.2011.04.009 (see p. 146).

van Diggelen, F. and Enge, P. (2015). "The World's first GPS MOOC and Worldwide Laboratory using Smartphones". In: *Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation*, pp. 361–369 (see p. 25).

Van Nguyen, L., Gibb, S., Pham, H., and La, H. (2018). "A Mobile Robot for Automated Civil Infrastructure Inspection and Evaluation". In: *International Symposium on Safety, Security, and Rescue Robotics*, pp. 1–6. DOI: 10.1109/SSRR.2018.8468642 (see pp. 6, 7, 10, 13, 77).

Van Rijsbergen, C. (1979). *Information Retrieval*. 2nd. Butterworth-Heinemann (see p. 165).

Varadharajan, S., Jose, S., Sharma, K., Wander, L., and Mertz, C. (2014). "Vision for road inspection". In: *Winter Conference on Applications of Computer Vision*, pp. 115–122. DOI: 10.1109/WACV.2014.6836111 (see pp. 6, 137, 139, 143).

Ventura, E., Costa, L., and Nogueira, R. A. (2015). "Fractal, multifractal and lacunarity analysis applied in retinal regions of diabetic patients with and without nonproliferative diabetic retinopathy". In: *Fractal Geometry and Nonlinear Analysis in Medicine and Biology* 1.3, pp. 112–119. DOI: 10.15761/FGNAMB.1000118 (see p. 197).

Victores, J., Martínez, S., Jardón, A., and Balaguer, C. (2011). "Robot-aided tunnel inspection and maintenance system by vision and proximity sensor integration". In: *Automation in Construction* 20.5, pp. 629–636. DOI: 10.1016/j.autcon.2010.12.005 (see p. 6).

Vivet, D., Checchin, P., and Chapuis, R. (2013). "Localization and mapping using only a rotating FMCW radar sensor". In: *Sensors (Switzerland)* 13.4, pp. 4527–4552. DOI: 10.3390/s130404527 (see p. 22).

Vu Dung, C. and Duc Anh, L. (2019). "Automation in Construction Autonomous concrete crack detection using deep fully convolutional neural network". In: *Automation in Construction* 99.December 2018, pp. 52–58. DOI: 10.1016/j.autcon.2018.11.028 (see pp. 143, 145, 164).

Ward, P., Manamperi, P., Brooks, P., Mann, P., Kaluarachchi, W., Matkovic, L., Paul, G., Quin, P., Pagano, D., Liu, D., Waldron, K., and Dissanayake, G. (2014). "Climbing robot for steel bridge inspection: design challenges". In: *Proceedings for the Austroads*, pp. 1–13 (see pp. 11, 13).

Westoby, M. J., Brasington, J., Glasser, N. F., Hambrey, M. J., and Reynolds, J. M. (2012). "'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications". In: *Geomorphology* 179, pp. 300–314. DOI: 10.1016/j.geomorph.2012.08.021 (see p. 118).

Xie, S. (2015). "Holistically-Nested Edge Detection University of California , San Diego". In: *International Conference on Computer Vision (ICCV)*, pp. 1395–1403. DOI: 10.1109/ICCV.2015.164 (see pp. 147, 150, 219).

Xie, S. and Tu, Z. (2017). "Holistically-Nested Edge Detection". In: *International Journal of Computer Vision* 125.1-3, pp. 3–18. DOI: 10.1007/s11263-017-1004-z (see pp. 147, 150, 151, 158–161, 197, 209, 219).

Yanagihara, M., Matsuzawa, T., Kudo, M., and Turker, T. (2000). "Replacement of Bearings in the Golden Horn Bridge, Turkey". In: *Structural Engineering International* 10.2, pp. 121–123. DOI: 10.2749/101686600780557857 (see p. 1).

Yang, C., Wen, M., Chen, Y., and Kang, S. (2015). "An Optimized Unmanned Aerial System for Bridge Inspection". In: *Proceeding of the 32nd International Symposium on Automation and Robotics in Construction*. DOI: 10.22260/ISARC2015/0084 (see pp. 6, 9, 13, 77, 78).

Yang, F., Zhang, Y. D., and Zhu, Y. J. (2016). "Road Crack Detection Using Deep Convolutional Neural Network". In: *International Conference on IImage Processing (ICIP)* October 2017. DOI: 10.1109/ICIP.2016.7533052 (see pp. 140, 141, 143).

Yang, L., Li, B., Li, W., Liu, Z., Yang, G., and Xiao, J. (2017). "Deep Concrete Inspection Using Unmanned Aerial Vehicle Towards CSSC Database". In: *Proceedings of the International Conference on Intelligent Robots and Systems* (see pp. 12, 141, 143, 155, 164, 165).

Yang, X., Li, H., Yu, Y., Luo, X., Huang, T., and Yang, X. (2018). "Automatic Pixel-Level Crack Detection and Measurement Using Fully Convolutional Network". In: *Computer-Aided Civil and Infrastructure Engineering* 33.12, pp. 1090–1109. DOI: 10.1111/mice.12412 (see p. 142).

Yeum, C. and Dyke, S. (2015). "Vision-Based Automated Crack Detection for Bridge Inspection". In: *Computer-Aided Civil and Infrastructure Engineering* 30.10, pp. 759–770. DOI: 10.1111/mice.12141 (see p. 13).

Yokoyamma, S. and Matsumoto, T. (2017). "Development of an automatic detector of cracks in concrete using machine learning". In: *The 3rd International Conference on Sustainable Civil Engineering Structures and Construction Materials - Sustainable Structures for Future Generations*, pp. 1250–1255. DOI: 10.1016/j.proeng.2017.01.418 (see p. 141).

Zaffar, M., Ehsan, S., Stolkin, R., and Maier, K. M. (2018). "Sensors, SLAM and Long-term Autonomy: A Review". In: *Cornell Computing Research Repository* abs/1807.01605. DOI: arXiv:1807.01605v1 (see pp. 21, 23).

Zhang, A., Wang, K. C. P., Asce, M., Fei, Y., Liu, Y., Tao, S., Chen, C., Li, J. Q., and Li, B. (2018a). "Deep Learning – Based Fully Automated Pavement Crack Detection on 3D Asphalt Surfaces with an Improved CrackNet". In: *Journal of Computing in Civil Engineering* 32.5, pp. 1–14. DOI: 10.1061/(ASCE)CP.1943-5487.0000775. (see pp. 142, 144, 145, 172).

Zhang, A., Wang, K. C. P., Fei, Y., Liu, Y., Chen, C., Yang, G., Li, J. Q., and Yang, E. (2018b). "Automated Pixel-Level Pavement Crack Detection on 3D Asphalt Surfaces with a Recurrent Neural Network". In: *Computer-Aided Civil and Infrastructure Engineering* 34.3, pp. 213–229. DOI: 10.1111/mice.12409 (see p. 142).

Zhou Wang and Alan C. Bovik (2004). "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *Transactions on Image Processing* 13.4, pp. 1–14. DOI: 10.1109/TIP.2003.819861 (see pp. 166, 167).

Zitnick, C. L. and Dollár, P. (2015). "Fast Edge Detection Using Structured Forests". In: *IEEE transactions on pattern analysis and machine intelligence*. Vol. 37, pp. 1558–1570 (see pp. 158, 161, 166).

Appendix A

Simulation parameters

A.1 Defining the robot URDF

The parameters below were used in simulation, but can also be used for the real robot in order to utilise tools in ROS, such as motion path planners.

```
<?xml version='1.0'?>

<robot name="rosbot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:include filename="$(find␣rosbot_description)/urdf/body/body.
      gazebo" />
  <xacro:macro name="body" params="parent␣baseHeight␣baseWidth">
  <link name='base_link'>
    <pose>0 0 0.1 0 0 3.142</pose>
    <inertial>
      <mass value="5"/>
      <origin xyz="0␣0␣0.04" rpy="␣0␣0␣0"/>
      <inertia
      ixx="0.01" ixy="0.01" ixz="0"
      iyy="0.01" iyz="0.01"
      izz="0.01"
      />
    </inertial>

    <collision name='collision'>
      <origin xyz="0␣0␣0.05" rpy="0␣0␣0"/>
      <geometry>
        <box size="0.19␣0.12␣0.045"/>
      </geometry>
    </collision>

    <visual name='base_link_visual'>
      <origin xyz="-0.085␣-0.06␣0.02" rpy="␣0␣0␣0"/>
      <geometry>
         <mesh filename="package://rosbot_description/meshes/
             diddy_box_small.dae"  scale="0.1␣0.1␣0.1"/>
```

```
      </geometry>
    </visual>

  </link>
  <joint name="top_joint" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <parent link="${parent}"/>
    <child link="top"/>
  </joint>
  <link name='top'>
    <pose>0 0 0 0 0 0</pose>

    <inertial>
      <mass value="0.01"/>
      <origin xyz="0 0 0.1" rpy=" 0 0 0"/>
      <inertia
      ixx="0.0" ixy="0" ixz="0"
      iyy="0.0" iyz="0"
      izz="0.0"
      />
    </inertial>
    <visual name='top'>
      <origin xyz="-0.085 -0.06 0.05" rpy=" 0 0 0"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/
            diddy_box_upper.dae" scale="0.1 0.1 0.1" />
      </geometry>
    </visual>
  </link>

  <link name="middle_left_wheel">
    <collision name="collision">
      <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>2</mu>
            <mu2>2</mu2>
          </ode>
        </friction>
      </surface>
    </collision>

    <visual name="middle_left_wheel_visual">
      <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
      <mass value="0.5"/>
```

```
    <inertia
    ixx="0.001" ixy="0.0" ixz="0.0"
    iyy="0.001" iyz="0.0"
    izz="0.001"/>
  </inertial>
</link>

<link name="front_left_wheel">
  <collision name="collision">
    <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
    <geometry>
      <mesh filename="package://rosbot_description/meshes/wheel.dae"
          scale="0.00075 0.00075 0.00075" />
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>2</mu>
          <mu2>2</mu2>
        </ode>
      </friction>
    </surface>
  </collision>
  <visual name="front_left_wheel_visual">
    <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
    <geometry>
      <mesh filename="package://rosbot_description/meshes/wheel.dae"
          scale="0.00075 0.00075 0.00075" />
    </geometry>
  </visual>
  <inertial>
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <mass value="0.5"/>
    <inertia
    ixx="0.001" ixy="0.0" ixz="0.0"
    iyy="0.001" iyz="0.0"
    izz="0.001"/>
  </inertial>
</link>

<link name="middle_right_wheel">
  <collision name="collision">
    <origin xyz="0 0 0" rpy="0 0 1.5707"/>
    <geometry>
      <mesh filename="package://rosbot_description/meshes/wheel.dae"
          scale="0.00075 0.00075 0.00075" />
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>2</mu>
          <mu2>2</mu2>
        </ode>
      </friction>
    </surface>
  </collision>
  <visual name="middle_right_wheel_visual">
    <origin xyz="0 0 0" rpy="0 0 1.5707"/>
```

```
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
      <mass value="0.5"/>
      <inertia
      ixx="0.001" ixy="0.0" ixz="0.0"
      iyy="0.001" iyz="0.0"
      izz="0.001"/>
    </inertial>
  </link>

  <link name="front_right_wheel">
    <collision name="collision">
      <origin xyz="0 0 0" rpy="0 0 1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>2</mu>
            <mu2>2</mu2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="front_right_wheel_visual">
      <origin xyz="0 0 0" rpy="0 0 1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
      <mass value="0.5"/>
      <inertia
      ixx="0.001" ixy="0.0" ixz="0.0"
      iyy="0.001" iyz="0.0"
      izz="0.001"/>
    </inertial>
  </link>

  <link name="rear_left_wheel">
    <collision name="collision">
      <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
      <surface>
        <friction>
          <ode>
```

```
            <mu>2</mu>
            <mu2>2</mu2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="rear_left_wheel_visual">
      <origin xyz="0 0 0" rpy="0 0 -1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
      <mass value="0.5"/>
      <inertia
      ixx="0.001" ixy="0.0" ixz="0.0"
      iyy="0.001" iyz="0.0"
      izz="0.001"/>
    </inertial>
  </link>

  <link name="rear_right_wheel">
    <collision name="collision">
      <origin xyz="0 0 0" rpy="0 0 1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>2</mu>
            <mu2>2</mu2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="rear_right_wheel_visual">
      <origin xyz="0 0 0" rpy="0 0 1.5707"/>
      <geometry>
        <mesh filename="package://rosbot_description/meshes/wheel.dae"
            scale="0.00075 0.00075 0.00075" />
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
      <mass value="0.5"/>
      <inertia
      ixx="0.001" ixy="0.0" ixz="0.0"
      iyy="0.001" iyz="0.0"
      izz="0.001"/>
    </inertial>
  </link>

  <joint type="continuous" name="front_left_wheel_hinge">
    <origin xyz="0.082 0.065 -0.005" rpy="0 0 0"/>
```

```
    <child link="front_left_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

  <joint type="continuous" name="front_right_wheel_hinge">
    <origin xyz="0.083 -0.065 -0.005" rpy="0 0 0"/>
    <child link="front_right_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

  <joint type="continuous" name="middle_left_wheel_hinge">
    <origin xyz="0.0 0.065 -0.005" rpy="0 0 0"/>
    <child link="middle_left_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

  <joint type="continuous" name="middle_right_wheel_hinge">
    <origin xyz="0.0 -0.065 -0.005" rpy="0 0 0"/>
    <child link="middle_right_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

  <joint type="continuous" name="rear_left_wheel_hinge">
    <origin xyz="-0.083 0.065 -0.005" rpy="0 0 0"/>
    <child link="rear_left_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

  <joint type="continuous" name="rear_right_wheel_hinge">
    <origin xyz="-0.083 -0.065 -0.005" rpy="0 0 0"/>
    <child link="rear_right_wheel"/>
    <parent link="${parent}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
  </joint>

 </xacro:macro>

</robot>
```

A.2 Additional robot parameters for simulation

The parameters below adapt the robot description above for the simulation of the differential drive controller.

```
<?xml version="1.0"?>
<robot>
  <gazebo>
    <plugin name="skid_steer_drive_controller" filename="
        libgazebo_ros_skid_steer_drive.so">
      <updateRate>20.0</updateRate>
      <robotBaseFrame>base_link</robotBaseFrame>
      <wheelSeparation>0.075</wheelSeparation>
      <wheelDiameter>0.065</wheelDiameter>
      <torque>0.8</torque>
      <leftFrontJoint>front_left_wheel_hinge</leftFrontJoint>
      <rightFrontJoint>front_right_wheel_hinge</rightFrontJoint>
      <rightMiddleJoint>middle_right_wheel_hinge</rightMiddleJoint>
      <leftMiddleJoint>middle_left_wheel_hinge</leftMiddleJoint>
      <leftRearJoint>rear_left_wheel_hinge</leftRearJoint>
      <rightRearJoint>rear_right_wheel_hinge</rightRearJoint>
      <!--max_velocity>0.1</max_velocity-->
      <topicName>cmd_vel</topicName>
      <commandTopic>cmd_vel</commandTopic>
      <broadcastTF>true</broadcastTF>
      <odometryTopic>odom_gazebo</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <covariance_x>0.000100</covariance_x>
      <covariance_y>0.000100</covariance_y>
      <covariance_yaw>0.010000</covariance_yaw>
    </plugin>
  </gazebo>

  <gazebo reference="base_link">
    <material>Gazebo/RustySteel</material>
  </gazebo>
  <gazebo reference="top">
    <material>Gazebo/Chrome</material>
  </gazebo>
  <gazebo reference="front_left_wheel">
    <material>Gazebo/Black</material>
  </gazebo>
  <gazebo reference="front_right_wheel">
    <material>Gazebo/Black</material>
  </gazebo>
  <gazebo reference="middle_left_wheel">
    <material>Gazebo/Black</material>
  </gazebo>
  <gazebo reference="middle_right_wheel">
    <material>Gazebo/Black</material>
  </gazebo>
  <gazebo reference="rear_left_wheel">
    <material>Gazebo/Black</material>
  </gazebo>
</robot>
```

A.3 Defining the 2D LiDAR in simulation

Below is an example of the parameters used to define the 2D LiDAR in the simulation environment. These parameters were based on the parameters of the real 2D LiDAR sensor in Chapter 3

```
<robot>
 <gazebo reference="rplidar">
    <sensor type="ray" name="head_rplidar_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>false</visualize>
      <update_rate>40</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>720</samples>
            <resolution>1</resolution>
            <min_angle>-3.14159265</min_angle>
            <max_angle>3.14159265</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.2</min>
          <max>30.0</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>
      <plugin name="gazebo_ros_head_rplidar_controller" filename="
          libgazebo_ros_laser.so">
        <topicName>scan</topicName>
        <frameName>laser</frameName>
      </plugin>
    </sensor>
  </gazebo>
</robot>
```

Appendix B

Parameter sweep for training parameters for HED network

B.1 Varying base learning rate and step-size

As described in Chapter 5, parameters are chosen for training the deep-learning network by performing tests on a validation set. First, the base learning rate and the step size at which the learning rate is reduced were varied. The results are shown in Figures B.1– B.3. When the base learning rate is set too high, i.e., above $1e-7$, the model does not converge (Figure B.1). Figure B.3 also shows that for a reduced base learning rate of $1e-8$, when the learning rate is reduced too frequently (e.g., for a step-size of 5000) then there are large oscillations in the loss. It is likely that the learning rate has been reduced too much and the model is learning slowly.

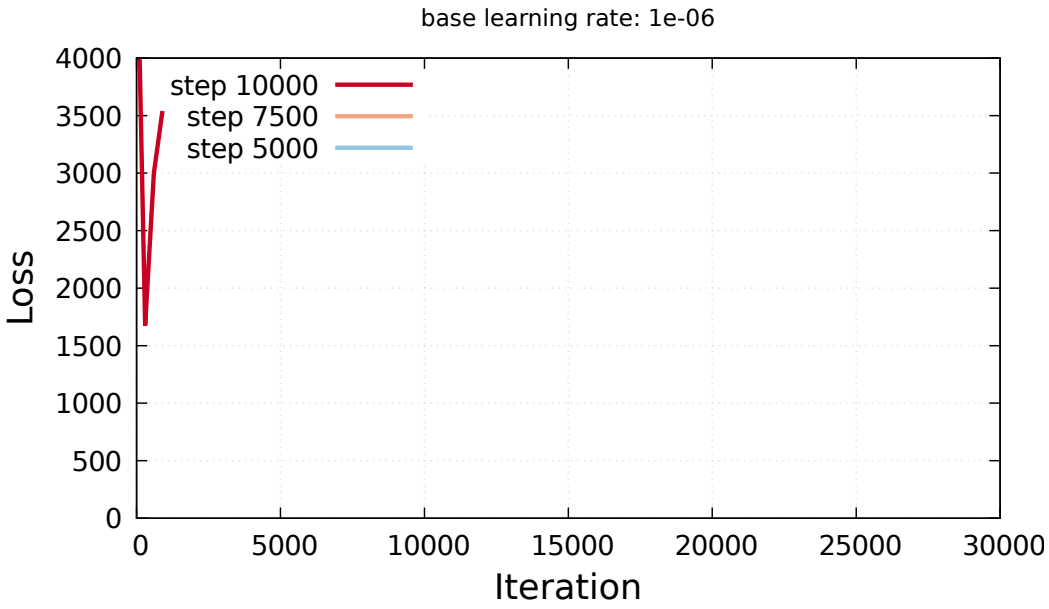


Figure B.1: Comparing training loss against number of training iterations for a base-learning rate of 1e-06. The learning rate is reduced at every 5000, 7500 or 10000 training iterations, but the model does not converge as the base-learning rate is too high.

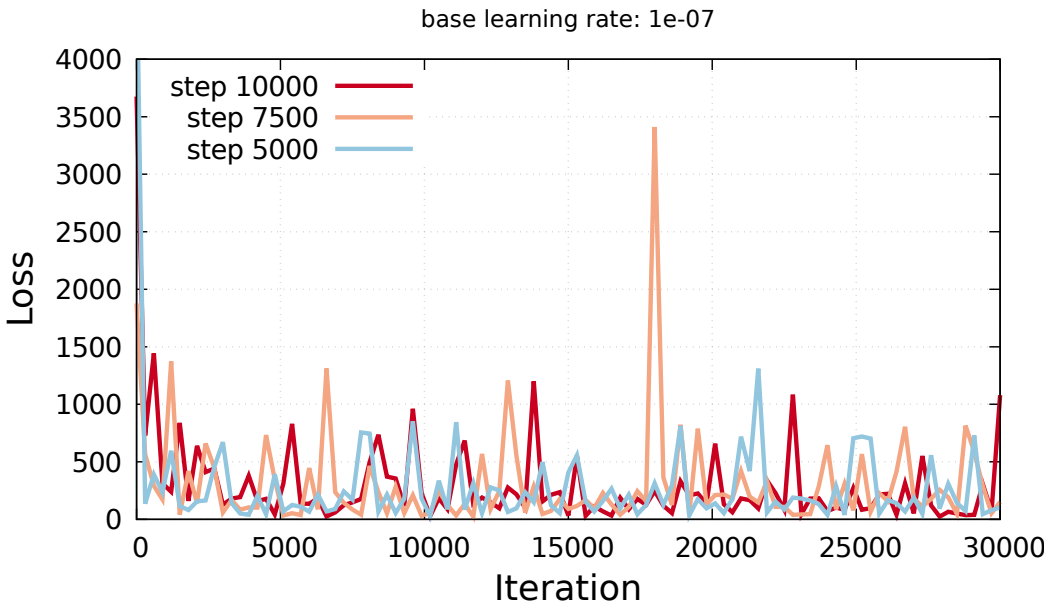


Figure B.2: Comparing training loss against number of training iterations for a base learning rate of 1e-07. The learning rate is reduced at every 5000, 7500 or 10000 training iterations.

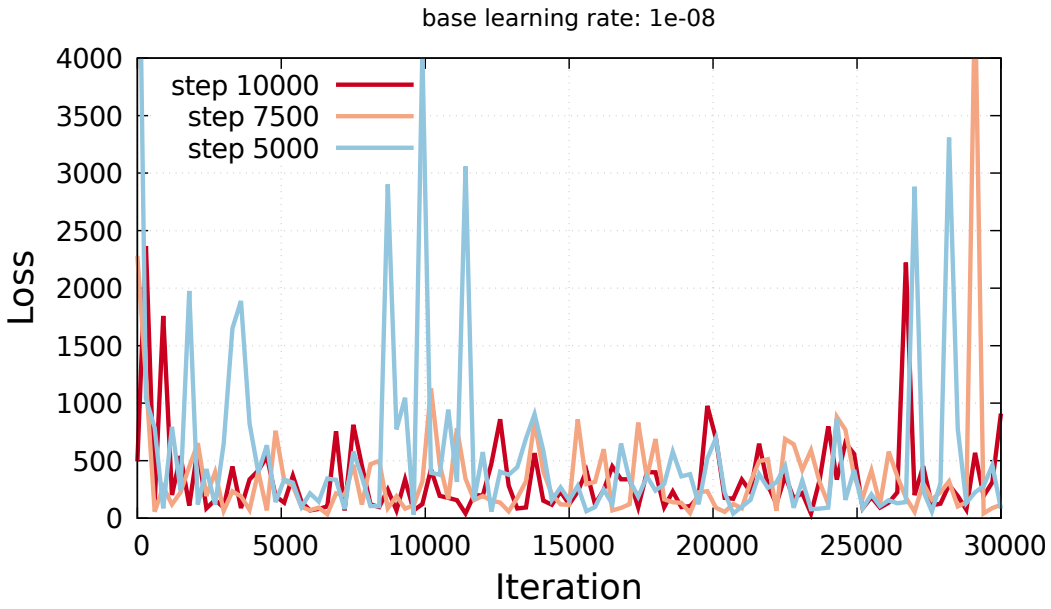


Figure B.3: Comparing training loss against number of training iterations for a base-learning rate of 1e-08. The learning rate is reduced at every 5000, 7500 or 10000 training iterations.

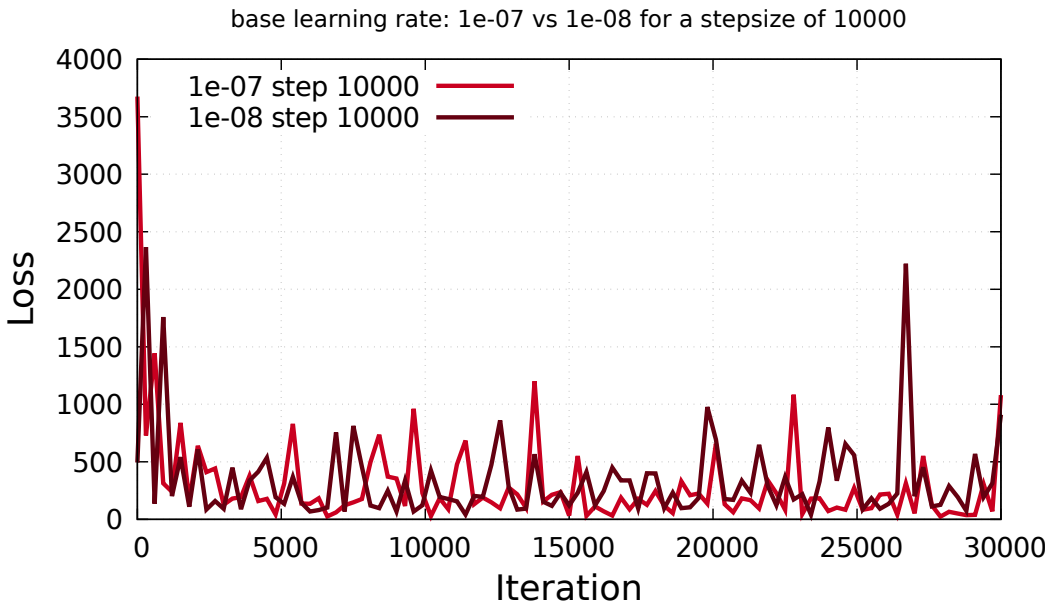


Figure B.4: A comparison of the training loss against number of training iterations for a base-learning rate of 1e-07 and 1e-08. The learning rate is reduced every 10000 training iterations.

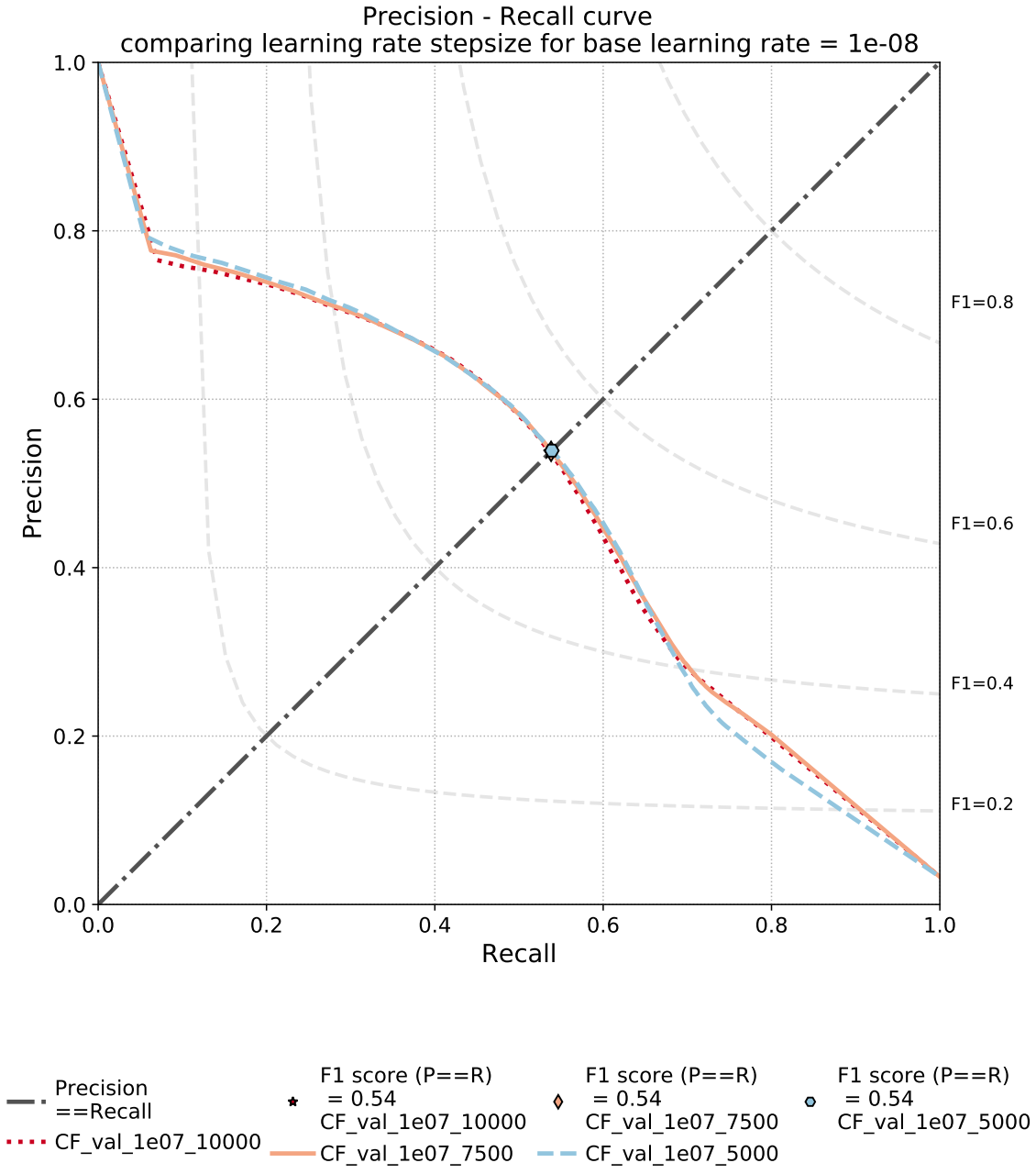


Figure B.5: A comparison of the precision-recall curves for a base learning rate of 1e-07 for learning rate steps of 5000,7500 and 10000.

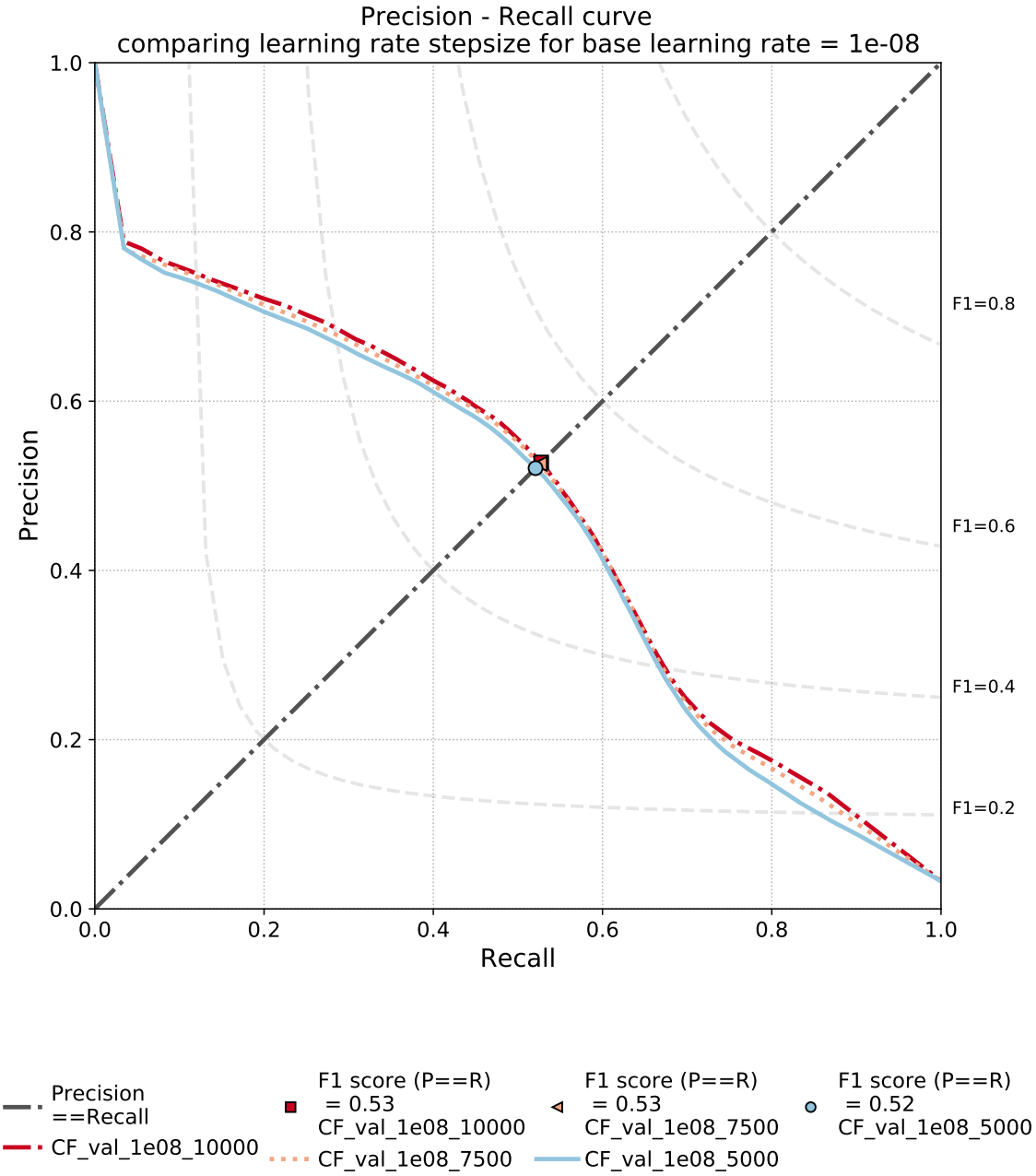


Figure B.6: A comparison of the precision-recall curves for a base learning rate of 1e-08 for learning rate steps of 5000,7500 and 10000.

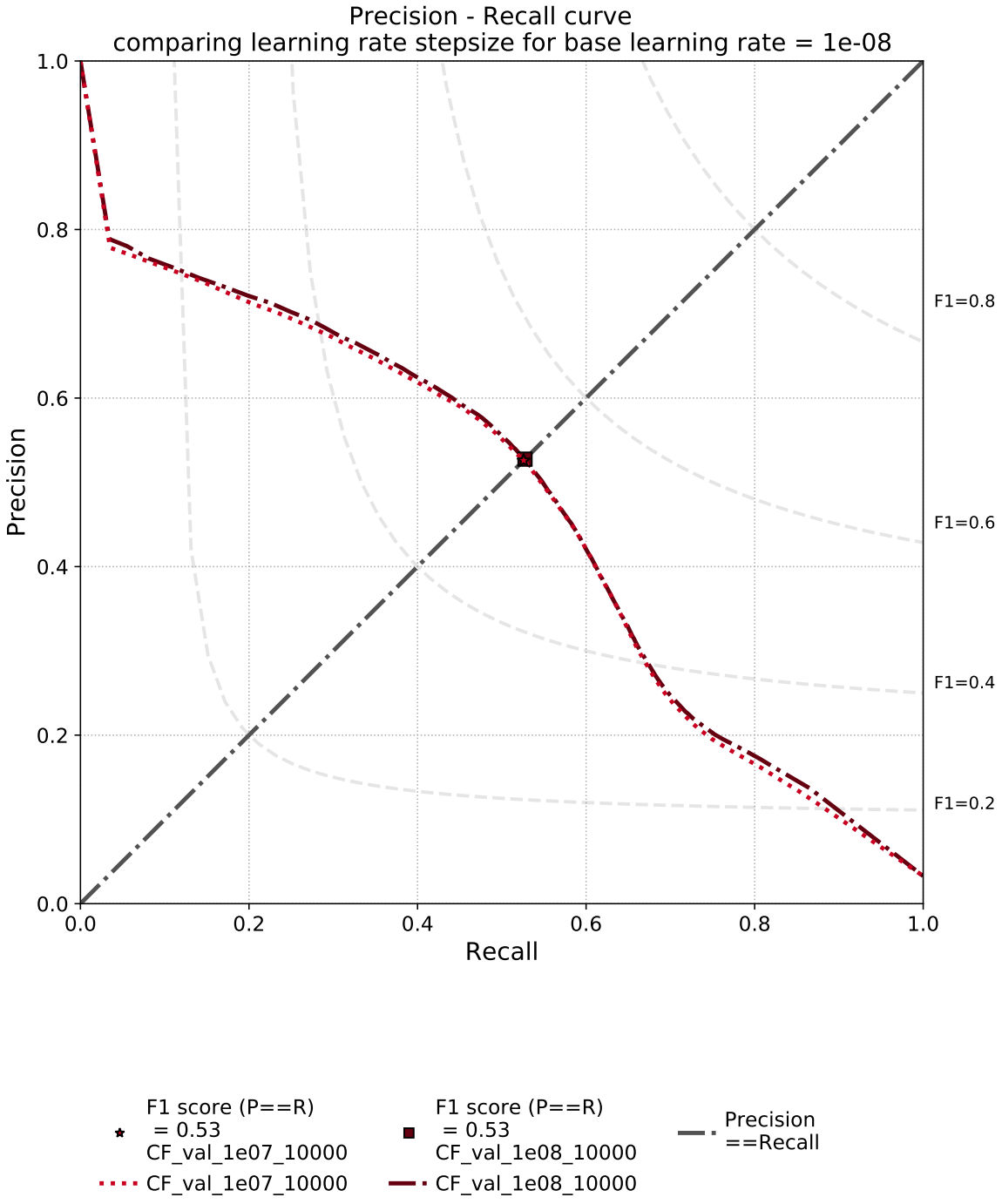


Figure B.7: A comparison of the precision-recall curves for a base learning rate of 1e-08 and 1e-07 for a learning rate step of 10000.

B.2 Varying weight decay

In addition to varying the base learning rate and the step at which the learning rate was reduced, the weight decay parameter is also varied. In the following plots, the weight decay for training the networks was varied for both a base learning rate of 1e-07 and 1e-08.

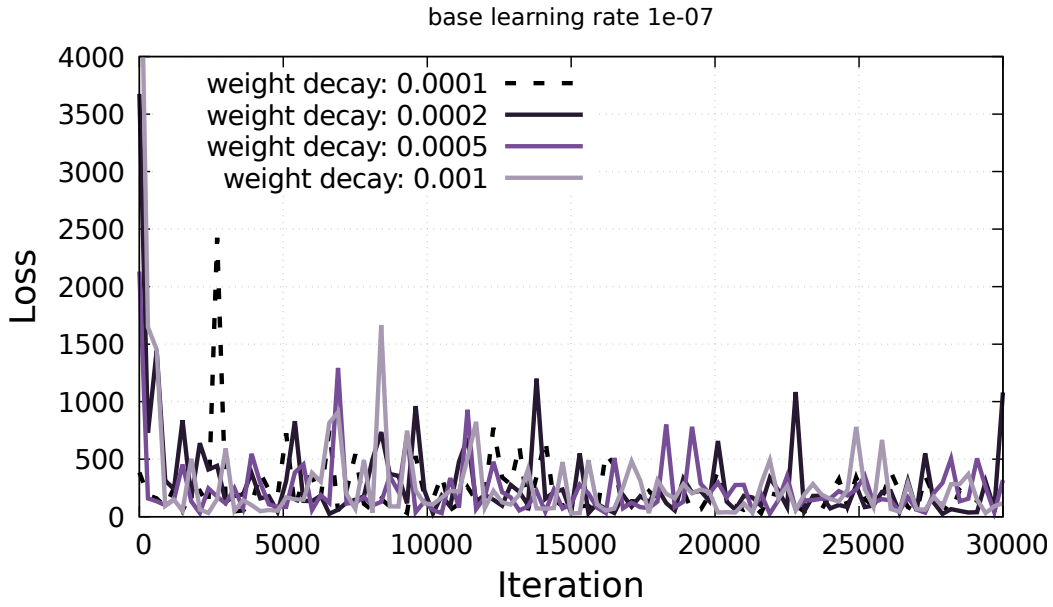


Figure B.8: A comparison of the training loss against number of training iterations for a base-learning rate of 1e-07 for different values of weight decay. The learning rate is reduced every 10000 training iterations.

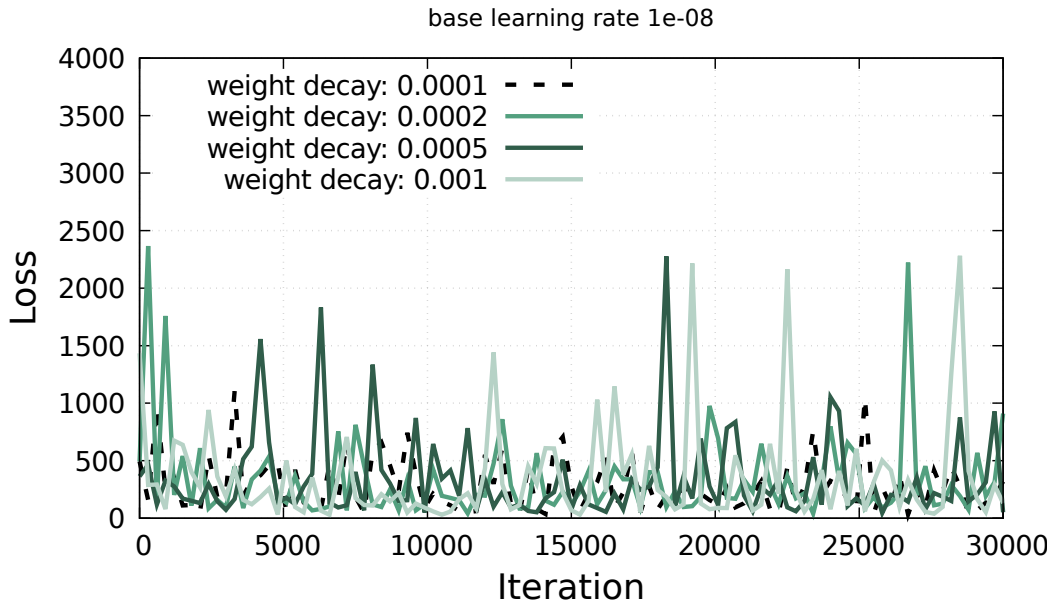


Figure B.9: A comparison of the training loss against number of training iterations for a base-learning rate of 1e-08 for different values of weight decay. The learning rate is reduced every 10000 training iterations.

B.3 Varying number of training iterations

Finally, the number of training iterations is considered. The precision-recall curve is plotted for different points in the training process and compared in Figure B.10. Figure B.10 showed that above 5000 training iterations there was no change in the precision-recall curve, and hence there is no performance gain in terms of F_1 score to be made by training the network for longer periods.

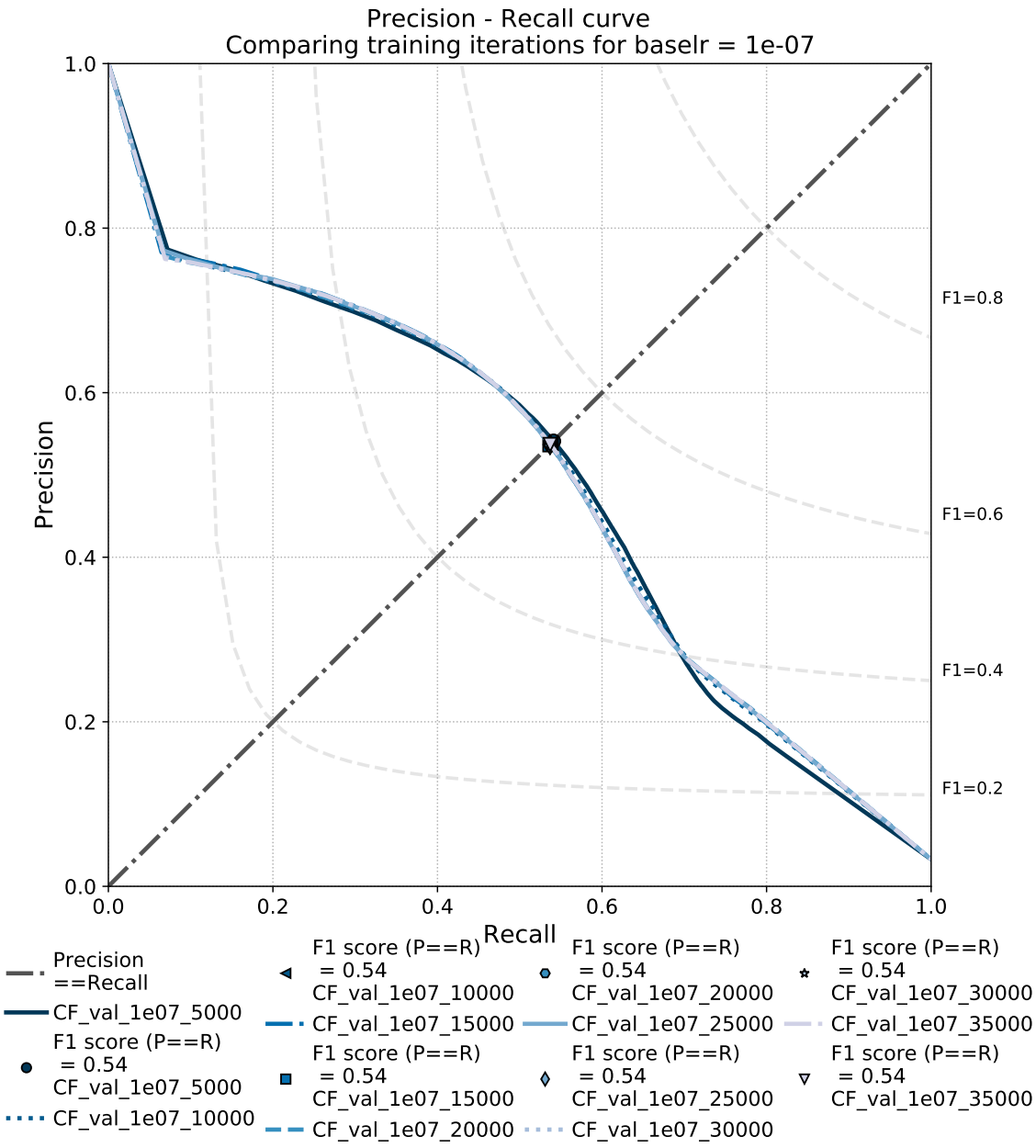


Figure B.10: A comparison of the precision-recall curves for different training iterations for a base learning rate of 1e-07, a learning step-size of 10000 and a weight-decay of 0.002.