AN INTEGRATED SYSTEM DESIGN FOR BUILDING INSPECTION BY AUTONOMOUS UAVs

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FATİH KÜÇÜKSUBAŞI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
BUILDING SCINCE IN ARCHITECTURE

DECEMBER 2017

Approval of the Thesis:

AN INTEGRATED SYSTEM DESIGN FOR BUILDING INSPECTION BY AUTONOMOUS UAVs

submitted by **FATİH KÜÇÜKSUBAŞI** in partial fulfillment of the requirements for the degree of **Master of Science in Building Science in Architecture Department, Middle East Technical University** by,

Prof. Dr. M. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. T. Elvan Altan _____
Head of Department, **Architecture**

Prof. Dr. Arzu Gönenç Sorguç _____
Supervisor, **Architecture Dept. METU**

Examining Committee Members:

Prof. Dr. Aydan M. Erkmen _____
Electrical And Electronics Engineering Dept., METU

Prof. Dr. Arzu Gönenç Sorguç _____
Architecture Dept., METU

Prof. Dr. Figen Beyhan _____
Architecture Dept., Gazi University

Assoc. Prof. Dr. Ayşe Tavukçuoğlu _____
Architecture Dept., METU

Assist. Prof. Dr. M. Koray Pekeriçli _____
Architecture Dept., METU

Date: 29.12.2017

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: FATİH KÜÇÜKSUBAŞI

Signature :

ABSTRACT

AN INTEGRATED SYSTEM DESIGN FOR BUILDING INSPECTION BY AUTONOMOUS UAVs

Küçüksubaşı, Fatih

M.Sc. in Building Science, Departmant of Architecture

Supervisor: Prof. Dr. Arzu Gönenç Sorguç

December 2017, 103 pages

Inspection of buildings throughout their lifecycle is vital in terms of human safety as the number of structures increases expeditiously. However, it is not easy to perform inspections for all cases. Physical reachability and complexity of the buildings are major problems along with the safety of inspectors during on-site operations. In this context, Unmanned Aerial Vehicles (UAV) have recently shown great performance collecting visual data through autonomous exploration and mapping in building inspection. Yet, the number of studies is limited considering the post processing of the data and its integration with autonomous UAVs. These will enable huge steps onward into full automation of building inspection. In this regard, this work presents a decision making tool for revisiting tasks in visual building inspection by autonomous UAVs. The tool is an implementation of fine-tuning a pretrained Convolutional Neural Network for surface crack detection. It offers an optional mechanism for task planning of revisiting pinpoint locations during inspection. It is integrated to a quadrotor UAV system that can autonomously navigate in GPS-denied environments. The UAV is equipped with onboard sensors and computers for autonomous localization, mapping and motion planning.

Additionally, a Graphical User Interface is developed in order to wrap the high-level features of the system for users. The integrated system is tested through simulations and real-world experiments. The results show that the system achieves crack detection and autonomous navigation in GPS-denied environments for building inspection.

Keywords: Unmanned Aerial Vehicle, Building Inspection, Task Planning, Crack Detection, Autonomous Navigation

ÖZ

OTONOM İHA'LARLA YAPI DENETİMİ İÇİN BÜTÜNLEŞİK BİR SİSTEM TASARIMI

Küçüksubaşı, Fatih
Yüksek Lisans, Yapı Bilimleri, Mimarlık Bölümü
Tez Yöneticisi: Prof. Dr. Arzu Gönenç Sorguç

Aralık 2017, 103 sayfa

Binaların sayısı hızla artarken, insan güvenliği açısından bina denetimi giderek önem kazanmaktadır. Bununla birlikte, her yapı için denetim yapmak kolay değildir. Binaların fiziki erişilebilirliği ve karmaşıklığı, sahadaki işlemler sırasında denetmenlerin güvenliği açısından büyük bir sorundur. Yapı denetiminde otonom keşfetme ve haritalama yoluyla görsel veri toplayabilen İnsansız Hava Araçları (İHA) son yıllarda son derece yüksek performans göstermektedir. Ancak, bu görüntülerin işlenmesi ve otonom İHA'larla entegrasyonu göz önüne alındığında, çalışma sayısı sınırlı kalmaktadır. Bunlar gerçekleştiğinde, yapı denetiminin tam otomasyonuna yönelik önemli adımlar atılmış olacaktır. Bu bağlamda, bu çalışmada otonom İHA'lar ile görsel yapı denetiminde belirlenen konumların yeniden ziyaret edilebilmesi için bir karar destek aracı sunulmaktadır. Önceden eğitilmiş bir Yapay Sinir Ağı'nın yeniden eğitilmesiyle, araç yüzey çatlaklarını tespit edebilmektedir. Noktasal olarak belirlenen konumların denetim sırasında tekrar gözden geçirilebilmesi için görev planlamasında kullanımı isteğe bağlı bir yöntem sunmaktadır ve GPS erişimsiz ortamlarda otonom olarak gezinebilen dört pervaneli bir İHA sistemine entegre edilmiştir.

İHA, otonom konumlama, haritalama ve hareket planlaması için bütünleşik algılayıcılar ve bilgisayarlarla donatılmıştır. Ayrıca, kullanıcıların sistemin üst düzey özelliklerini kontrol edebilmeleri için bir grafik kullanıcı arayüzü geliştirilmiştir. Bütünleşik sistem benzetimler ve deneyler ile test edilmiştir. Sonuçlar sistemin bina denetimi için GPS erişimsiz ortamlarda otonom seyrüseferi ve çatlak algılamayı gerçekleştirdiğini göstermektedir.

Anahtar Kelimeler: İnsansız Hava Aracı, Yapı Denetimi, Görev Planlama, Çatlak Tespiti, Otonom Seyrüsefer

in memory of Aziz Küçüksubaşı

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis supervisor Prof. Dr. Arzu Gönenç Sorguç for her encouragement and advice throughout this study.

I am thankful to my colleagues Serkan Ülgen, Fırat Özgenel, Müge Kruşa, and Gizem Yetiş for their support.

I would also offer my sincere appreciation to METU Design Factory Team.

I would like to thank all those who have served and contributed to science and technology.

Finally, I must express my very profound gratitude to my mother Şükran, my father Nadir and my brother Faruk for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

API: Application Programming Interface

BKPIECE: Bi-directional KPIECE

CNN: Convolutional Neural Network

EKF: Extended Kalman Filter

EST: Expansive Space Trees

GPS: Global Positioning System

GPU: Graphics Processing Unit

GUI: Graphical User Interface

IMU: Inertial Measurement Unit

KPIECE: Kinodynamic Motion Planning by Interior-Exterior Cell Exploration

LIDAR: Light Detection and Ranging Sensor

MAV: Micro Aerial Vehicle

MILP: Mixed-Integer Linear Programming

MPC: Model Predictive Control

PRM: Probabilistic Roadmap

RGB-D: Red Green Blue Depth

ROS: Robot Operating System

RRT: Rapidly Exploring Random Trees

RTAB-Map: Realtime Appearance Based Mapping

SDK: Software Development Kit

SLAM: Simultaneous Localization and Mapping

SRDF: Semantic Robot Description Format

UAV: Unmanned Aerial Vehicle

URDF: Unified Robot Description Format

VTOL: Vertical Take-off Landing

CHAPTER I

INTRODUCTION

1.1 Motivation

Emergence of the word ‘robot’ had waited until the 1920s even though the history of automation can be traced back to ancient times. After the 1950s, robots have been utilized mostly to accomplish relatively simple manufacturing tasks in the earlier stages of robotics technology. The area of use is expanded in time. Many industries have employed industrial robots for automation of many complex tasks as the rapid advances takes place both in the technology and the methodology. Pioneered by space missions, defense industry and healthcare, robotics becomes one of the essential technologies. Furthermore, many distinct types of robots have become a part of domestic activities as they are commercialized with affordable prices like cleaning robots for household uses.

Besides, mobility of robots had become a subject of the contemporary research as the need of achieving in-situ missions emerge. Increasing reachability, working in different environments (from underground to space) lead up mobile robotics. Thus, mobile robots have extended the bounds of applications.

Mobile robots can be classified in terms of the environment in which they take action as ground robots, underwater robots and aerial robots. Especially, the use of aerial robots has radically increased in recent years in many fields such as; photogrammetry, cargo, agriculture, geology, firefighting, forestry etc. The main reason of this increase is their unrestricted motion capability in 3D space. They can operate through -theoretically- unlimited workspace to perform dedicated actions autonomously and/or manually. Additionally, sensors have recently improved their ability of taking actions that are difficult/dangerous for humans such as search and rescue operations or surveying an area after disasters.

Figure 1.1 Examples of aerial vehicles (TRNDLabs, 2014; IMAGINE DRONE, 2017; GENERAL ATOMICS, 2007; Viehmeister-Kerner, A., 2017)

On the other hand, aerial robots can only involve tasks in built-environments as the autonomous navigation extends. Concurrent with the advances in other fields, the number of robotics applications for buildings is rapidly increasing throughout different phases of buildings' lifecycle.

Today, robots stand out in construction, monitoring/inspection, and maintenance phases of buildings since they provide precision, accuracy, repeatability, safety and more. It is now possible to state that these vehicles are keys of the forthcoming era for automation in construction and inspection related tasks. It is evident that the automation of inspection tasks requires prompt action since it is related to human safety. Therefore, the motivation of this study is to put effort on automation of building inspection by autonomous aerial vehicles.

1.2 Problem Statement

Inspection or monitoring of buildings throughout their lifecycle is vital in terms of human safety as the number of structures increases expeditiously. Studies in this field not only have importance in the assessment of various performance indicators, but also in extending the life of buildings. In line with this objective, periodic inspections are essential for residents' safety while overloading, disasters and aging have adverse effect on structural properties. Law enforcements by governments might be an indicator how important building inspection is from the standpoint of safety.

However, it is not easy to perform inspection for all cases. Physical accessibility and complexity of the environment are major problems when the part of structure subjected to inspection (i.e. airshaft, bridge deck) is hard to be reached by humans. High-rise buildings, dams, bridges, chimneys, towers can be given as examples of buildings that are hard to access. Another problem is to carry out the inspection missions without risking the inspectors' safety while working in such environments.

Inspectors generally use special equipment (Figure 1.2) to reach a location of interest and this can lead occupational accidents during operations.

Furthermore, inspection of rapidly increasing building stock is still a labor intensive work. For instance, systematic bridge inspections are done periodically in six years to detect structural cracks (Metni & Hamel, 2007). Even if the number of bridges is considered, inspection demands a very large amount of labor force.

Figure 1.2 Risky conditions in building inspection

Despite the cutting edge-technology, one of the most preferable inspection methods is still based on observing buildings visually which is called visual inspection. It is conducted periodically and approximately 95% of the building inspections are based on visual inspection (Eschmann, Kuo, Kuo & Boller, 2013). Therefore, automation of inspection processes has become requisite since it is safer and easier to bring the view of the location of interest to inspector, than the condition in which he/she sets foot in that location.

In this context, mobile robots have started to succor with their potentials that they showed in other fields such as search and rescue operations, space exploration missions. Eventually, mobile robots have become means to systematic, efficient and safe building inspection such as in Figure 1.3. Ground vehicles had been demonstrated in the early stages as an instrument in inspection. Yet, they are bounded to limited workspaces since they have to be in contact with surfaces; so, aerial vehicles notably increased physical accessibility with their superior degree of freedom in 3D space. It may not be ambitious to state that human operated (manual) inspections via aerial vehicles are already long-standing.

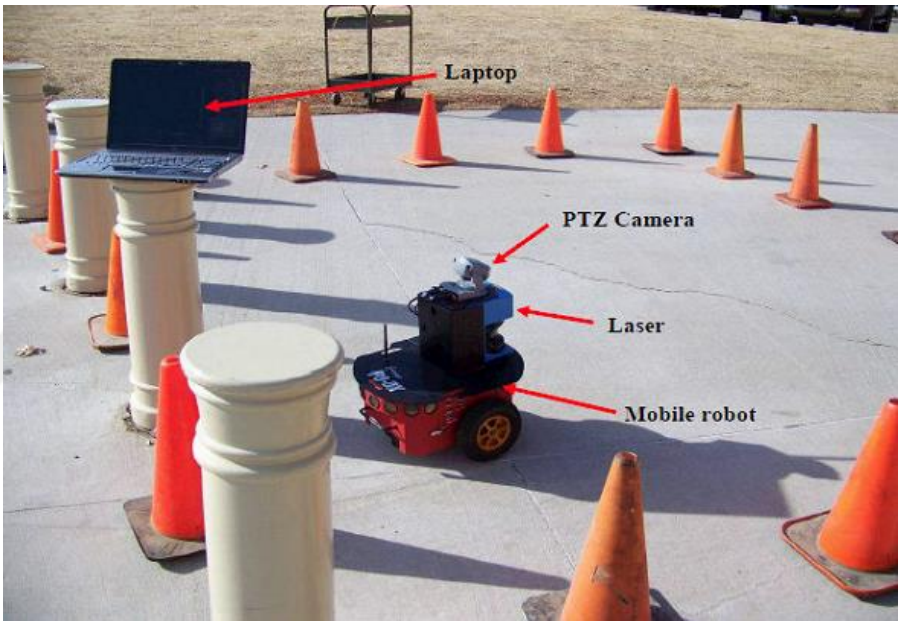

Figure 1.3 A wheeled robotic crack detection system for concrete surfaces (Lim, La & Sheng, 2014)

Today, automation of these manually operated aerial vehicles is one of the main research foci of robotics, in building inspection liberating the process from possible human error in operations, maintain safety and decrease operational costs. In this regard, automation of inspection requires adaptation of Unmanned Aerial Vehicles (UAV) with their potentials to building inspection tasks. In this pursuit, huge steps are taken as examined later in this dissertation.

The robotics community has shown many aspects of autonomous mapping and navigation for building inspection. Most of the efforts have been put on research and development for flight control, localization, mapping and motion planning. Nevertheless, it is a necessity for inspection community to acquire more user-friendly, practical, time and cost effective solutions that are also responding the requirements of inspection operation itself.

Although UAVs have proven autonomous navigation abilities in different environments, there are still steps to take in terms of building inspection. For instance, human inspectors still take part in inspections at least to analyze the obtained data, even if robots are engaged in the processes. Thus, it is important to address the phases after data gathering such as close examination of a particular area. In many cases, it is a necessity to revisit locations on the building to explore/inspect in detail.

Visually locating the defected areas in buildings is not an easy and straightforward task. The textures and visual features make hard the perception and recognition in the problematic areas. Inside of an industrial boiler, face of a dam, envelope of a high-rise or bridge of a deck can be listed as some examples of this scenario. In this case, the UAV should autonomously revisit a pin-point location for a close examination or even maintenance.

In this context, planning of such a revisiting task including post-processing of the obtained data comes into prominence regarding the automation of visual building inspection. To achieve this, the following research questions should be answered:

- Is it possible within state-of-art techniques and technologies for UAVs to revisit predefined locations autonomously in both GPS-denied indoor and outdoor environments?

- Can state of the art machine learning strategies be incorporated in evaluation of the data and decision making to increase efficiency of visual inspections?

- Can mission/task planning be achieved considering the needs of building inspection after imaging/mapping phase?

1.3 Aim and Objectives

Aim of this research is to achieve autonomous navigation and task planning of aerial robots, specifically Vertical Take-Off and Landing (VTOL) UAVs, in GPS-denied confined environments to be able to perform revisiting locations of interest during building inspection operations. In other words, a task planning strategy that enables UAVs to autonomously navigate in different environments while proposing a decision making tool for evaluation of the acquired data. Objectives in order to achieve the aim are listed below. The objectives of autonomous navigation and task planning for revisiting are presented separately for the sake of clarity.

Autonomous navigation of the UAV can be achieved by;

- Real-time localization of the UAV during flight with an accuracy that does not affect the revisiting performance,
- Mapping of the environment in which the UAV operates for global localization and motion planning,
- Motion planning with obstacle avoidance to target locations of revisiting,

Task planning for revisiting pinpoint locations can be achieved by;

- High-level framework in order to wrap the autonomous navigation capabilities for users while specifying pinpoint locations to revisit.
- A graphical interface for users without a priori robotics knowledge in order to plan the task.

Furthermore, a surface crack detection approach from captured images during flights as an assistance for decision making is demonstrated.

1.4 Contributions

It is endeavored to utilize state of the art technology and methodology throughout this study to be able to reduce the risky situations and increase efficiency in visual building inspection. The major contribution of this dissertation can be disclosed as a complete implementation for autonomous building inspection considering not only the mapping phase but also the subsequent close examination phase. An autonomously navigating multirotor UAV with ability of revisiting pinpoint locations is introduced. A user interface is developed to operate high-level functionalities of this system.

In order to demonstrate potential prominent features of the proposed method a pretrained CNN is fine-tuned to be able to facilitate decision making for determining locations for detailed inspection to identify surface cracks. As a result, the proposed approach increments the efforts in automation of building inspection processes.

1.5 Structure of the Dissertation

The subsequent chapters of this dissertation are structured as follows. In Chapter II, a brief overview of related work is presented to be able to apprehend the current state of the art. The proposed system is explained in Chapter III. The methodological track is explained in Chapter IV to reveal assumptions and scientific approach. In Chapter V, the experimental configuration and implementations are explained. The findings in the case studies are presented and discussed in the following chapter. Finally, the dissertation is concluded with highlights and possible future study perspectives revealed.

CHAPTER II

PREVIOUS WORK

Research towards autonomous navigation of multirotor Unmanned Aerial Vehicles (UAV) has been in progress, together with their utilization in building inspection operations. There are works related with UAVs considering both military and civil applications, acknowledged as the current state of the art. In order to comprehend the subject in a well-structured way, the research on autonomous navigation of UAVs can be classified into several subfields as flight control, localization and mapping, planning.

Local (attitude) control of multirotor UAVs for stable and accurate flight has been one of the main foci of the researches. Yet, concentration of this research is on global control of UAVs rather than local control; therefore, relevant subject matters are examined in detail.

2.1 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is an essential concept for autonomy which enables a mobile robot to synchronously estimate its state (position, orientation, velocity etc.) and construct a map (position of landmarks, obstacles) of the environment in which it is located. Although localization and mapping is an interdependent problem, they are examined separately for the sake of clarity.

2.1.1 Localization

Low-cost UAVs often have color cameras (Máthé & Buşoniu, 2015). The acquired images are subjected to computer vision techniques for mapping as well as localization. Taking advantage of these readily available onboard cameras appear practical considering the payload limits of UAVs.

For UAV navigation, there are ongoing studies in the field that are recently focused on visual-inertial localization since inertial measurements acquired from Inertial Measurement Units (IMU) accumulate errors in time that cause drift in estimations (Bachrach, 2012). To overcome this problem, exteroceptive sensors such as laser scanners, cameras, GPS, ultrasonic sensors are utilized (Weiss, 2012).

As in the case of this thesis, visual-inertial sensor fusion is one of the most common methods for state estimation. The underlying reason of this approach is the ability of visual pattern/feature recognition of the state of art methods. Bouvrie (2011) demonstrated visual-inertial sensor fusion using RGB-D sensor and IMU by iterative closest point approach. However, this is computationally demanding approach so, it has drawbacks for onboard computing.

Bloesch *et al.* (2014) presented a method for fusing optical flow and inertial measurements. They reduced the dimensionality of the state space for fast implementation. Loianno, Watterson & Kumar (2016) developed a visual-inertial odometry system using Unscented Kalman Filter on SE(3) in order to obtain singularity-free representation of a rigid body pose.

Máthé & Buşoniu (2015) distinguish the use of computer vision methods in UAV navigation as;

"*In UAV navigation, feature detectors and extractors are often used for object detection; optical flow techniques are used to distinguish motion in a scene; visual servoing is employed to translate image frame motion into UAV displacement; whereas 3D reconstruction methods are exploited for navigation and mapping.*"

State estimation is an important concept for localization of a robot by generally utilizing sensor stream. For multirotor UAVs, state generally composes of position, orientation, linear and angular velocities in 3D space. A common approach is to fuse the measurements from sensors in order to overcome drift and make the estimations more robust while taking advantage of different types of sensors. For this purpose; monocular, stereo and depth (RGB-D etc.) cameras are widely used as exteroceptive sensors for visual ego motion estimation (Shen, Mulgaonkar, Michael & Kumar, 2013; Newcombe *et al.*, 2011; Mur-Artal, Montiel & Tardos, 2015; Fang & Zhang, 2015; Liu, Zhang, Wu & Don, 2014).

Images acquired from cameras are then fused with either IMU, GPS, LIDAR etc. data for global localization. Lots of effort is put on using Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) as the approach for state estimation (Heng, Lee, Fraundorfer & Pollefeys, 2011; Shen *et al.*, 2013).

EKF and UKF are extended versions of the Kalman Filter (Kalman, 1960) to solve nonlinear problems which is the case in quadrotor flights. The variations of Kalman Filter are widely used in UAV navigation applications. EKF's complete picture of operation can be seen in Figure 2.1.

Bloesch, Omari, Hutter & Siegwart (2015) used EKF for fusing monocular visual-inertial odometry by pixel intensity errors of image patches. The work demonstrates the approach by directly employing on a UAV.

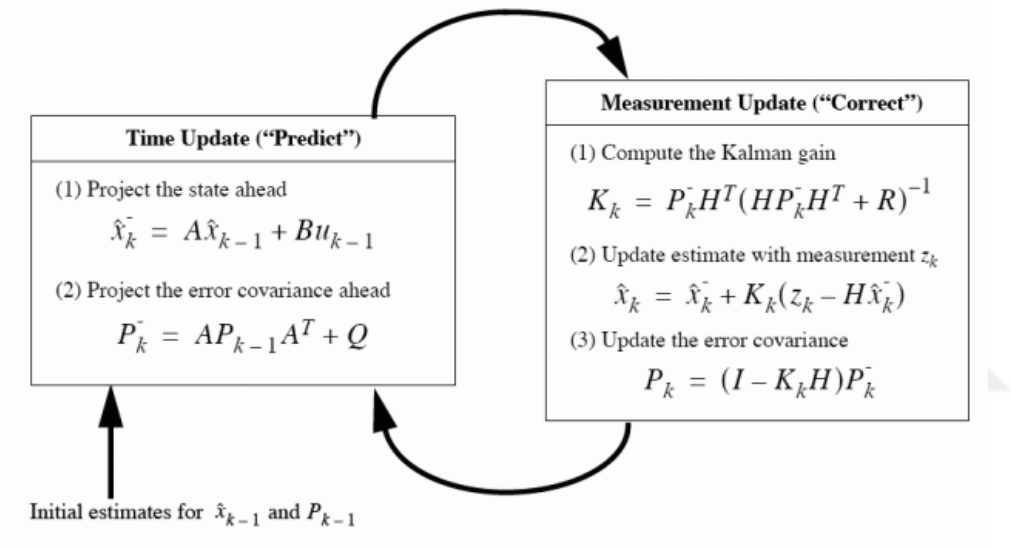

Figure 2.1 Operational diagram of Extended Kalman Filter (Fischer, 2004)

2.1.2 Mapping

As the vehicle moves especially in large environments, errors in the odometry accumulate and cause drifts in state estimation. Therefore, mapping is necessary both for planning and (global) localization relative to the environment. Moreover, it is a helpful step while considering visualization in building inspection. Today, many types of map representations (grid map, point cloud, voxel grid etc.) are employed in applications by robotics community.

Vision-based approaches are generally utilized to extract features from the environment for global loop-closure detection because it is possible and easier to identify distinctiveness via vision (Tapus & Siegwart, 2008). Visual mapping is considerably advantageous considering visual building inspection since the map can also be used for inspection purposes. Therefore, visual mapping approaches are assessed within the scope of this study.

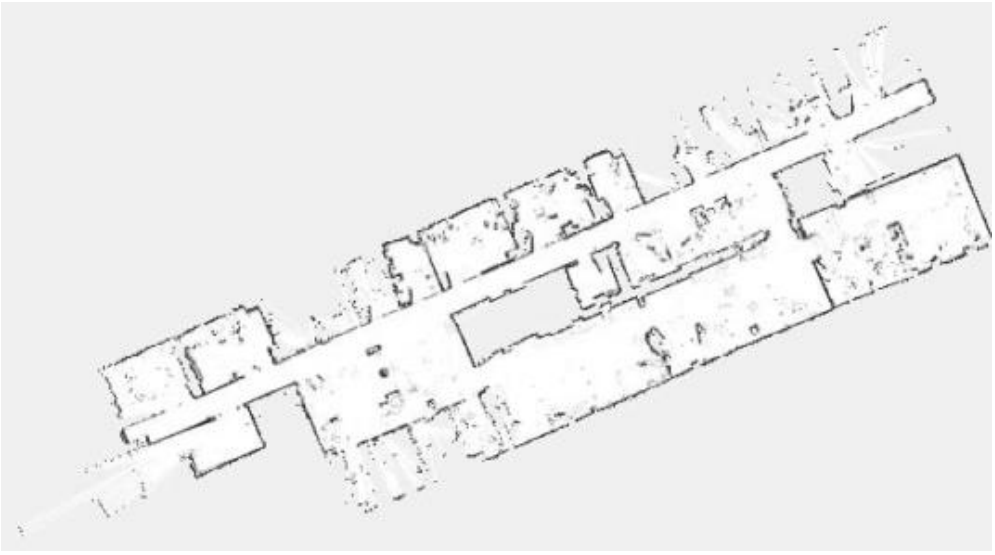

Figure 2.2 An occupancy grid map representation (Stachniss, 2006)

Considering UAVs, the map representation should be 3D rather than 2D as in the case of occupancy grid representation (Figure 2.2). Although 2D occupancy grid maps are useful for many tasks for ground vehicles, reliable path planning and collision avoidance necessitates 3D maps. Two of the most common approaches of 3D representations are point clouds and 3D voxel grids as illustrated in Figure 2.3.

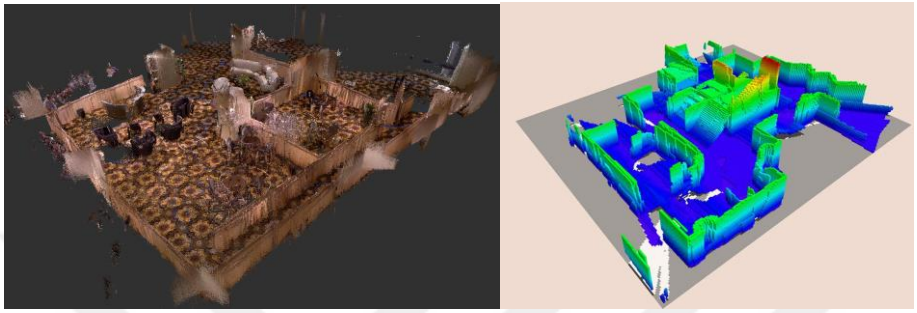

Figure 2.3 Examples of point cloud (left) (MAXED-OUT TEAM, 2014) and 3D voxel grid (right) (Robotic Research Team, 2017)

Another important concept in SLAM is called loop closure that enables minimizing localization error by the help of landmark recognition because of the interconnected character of SLAM problem (Cadena *et al.*, 2016). It helps creating consistent maps. Once a location is revisited, landmarks are identified and then the map is reconstructed to minimize localization and/or mapping errors.

In SLAM applications, bag-of-words (Sivic, & Zisserman, 2003) is a common approach for visual mapping (Cummins & Newman, 2008; Botterill, Mills & Green, 2011; Konolige *et al.*, 2010; Nister & Stewenius, 2006; Angeli, Filliat, Doncieux & Meyer, 2008). The bag-of-words uses representation of images with visual words taken from so-called vocabulary. Local feature descriptors are extracted and quantized into vocabulary in order to compare with the words in the vocabulary later on. This provides efficiency and speed especially in large datasets (environments) which are the general case in UAV navigation.

A research using appearance-based (visual) localization and mapping is proposed by Labbé and Michaud (2014) to solve kidnapped robot problem and/or multi-session mapping with an online global loop-closure detection method. They released an open source standalone software package named RTAB-Map for SLAM community. This graph-based SLAM system considers memory management for online processing and is suitable to use with stereo or RGBD camera configuration for both indoor and outdoor environments without a dependency of time and size. Hence, this is a suitable method both for applicability and robustness in the realm of this thesis. Figure 2.4 illustrates the loop-closure detection algorithm in RTAB-Map.

```
1: time ← TIMENOW()                    ▷ TIMENOW() returns current time
2: I_t ← acquired image
3: L_t ← LOCATIONCREATION(I_t)
4: if z_t (of L_t) is a bad signature (using T_bad) then
5:     Delete L_t
6: else
7:     Insert L_t into STM, adding a neighbor link with L_{t-1}
8:     Weight Update of L_t in STM (using T_similarity)
9:     if STM's size reached its limit (T_STM) then
10:        Move oldest location of STM to WM
11:    end if
12:    p(S_t|L^t) ←Bayesian Filter Update in WM with L_t
13:    Loop Closure Hypothesis Selection (S_t = i)
14:    if S_t = i is accepted (using T_loop) then
15:        Add loop closure link between L_t and L_i
16:    end if
17:    Join trash's thread             ▷ Thread started in TRANSFER()
18:    RETRIEVAL(L_i)                                  ▷ LTM → WM
19:    pTime ← TIMENOW() − time                        ▷ Processing time
20:    if pTime > T_time then
21:        TRANSFER()                                  ▷ WM → LTM
22:    end if
23: end if
```

Figure 2.4 Loop-closure detection pseudo algorithm of RTAB-Map (Labbe & Michaud, 2013).

2.2 Planning

Motion and/or path planning for UAVs in cluttered (especially indoor) environments is relatively challenging problem than the case of ground robots when one considers 3D workspace that they should operate. The main objective of a path planning problem is generally to compute a complete collision-free trajectory while dealing with geometric and physical constraints. Optimality, on the other hand, is a common goal to achieve and might be considered for path length and/or time wise optimization.

In the paper submitted by Yang, Qi, Xiao and Yong (2014), a decent review of 3D path planning approaches for UAVs is presented. They classified the algorithms in five categories namely; Sampling Based Algorithms, Node Based Optimal Algorithms, Mathematic Model Based Algorithms, Bio-inspired Algorithms, Multi-fusion Based Algorithms. According to the authors, sampling based algorithms sample the workspace as nodes, then connect the nearest nodes, or depth-first search strategy etc. The next step is to search for an optimal and complete path generally in an accepted metric range. This type of algorithms is easy to implement and can be used for static and online (real-time) planning. Node based algorithms take nodes into account while disregarding regular mapping formation. In general, they deal with node information that translates distance into calculation weight. Then, a global optimal path is searched. These approaches are suitable to combine with other methods and can be used for online planning. Multi-fusion based algorithms are combination of different algorithms' advantageous parts in order to find global optimal solutions. These algorithms are dealing with challenges that a single algorithm may not manage. For instance, artificial potential field algorithm cannot achieve an optimal global solution without navigation function or any other approach. The summary of aforementioned algorithms may be seen in Table 2.1.

Table 2.1 Summary of path planning algorithms by Yang *et al.* (2014)

Method	Elements of each method	Time Complexity	S/D Environment	Real-time
Sampling Based Algorithms	Voronoi [14], RRT [15], PRM[3], K-PRM [11], S-PRM [10], Visibility Graphs [38], Corridor Map[16], DDRRT [17], RRT* [18]	$0(n\log n) \le T \le 0(n^2)$	S and D	On-line
Node Based Algorithms	Dijkstra's Algorithms [4], A* [3], D*[6], LPA [19], Theta* [5], Lazy Theta* [20], D*-Lite [21], Harmony Search [22]	$0(m\log n) \le T \le 0(n^2)$	S and D	On-line
Mathematic Model Based Algorithms	Optimal Control [22], Mixed-Integer Linear Programming [23], Binary Linear Programming[24], Non-linear Programming [27,28]	Depending on the polynomial equation	S and D	Off-line
Bio-inspired Algorithms	NN [30], genetic algorithm [31], memetic algorithm[32], particle swarm optimization [33], ant colony optimization[34], shuffled frog leaping algorithm[35].	$T \ge 0(n^2)$	S	Off-line
Multi-fusion Based Algorithms	VVP [36], PRM Node based optimal algorithms [3], GIS-MCDA algorithms [40], visibility graph Node based optimal algorithms [38], visibility graph Geodesics algorithm[39]	$0(n\log n) \le T$	Depending on the algorithm	On-line

It is aimed to have real-time onboard planning in this work so, Mathematic Model Based Algorithms and Bio-inspired Algorithms are beyond the scope of this research since they are offline algorithms. As a matter of fact, planning in confined 3-dimensional areas is complicated since a robot is restricted in the spaces between structural components, or where it is obstructed. To be able to overcome these type of problems, global path planning strategies built upon sampling based planning are frequently utilized to provide feasible paths.

Mathe and Buşoniu (2015), in their detailed review, discussed that rapidly-exploring random tree (RRT) algorithm reduces computational cost. They also examined model predictive control (MPC) schemes by addressing several works integrated with other approaches such as mixed-integer linear programming (MILP) and Dubins curves. This work is important in the implementation phase of this thesis since RRT and RRT* are predominately tested.

In their work, Bircher *et al.* (2015) proposed an algorithm for outdoor inspection path planning for complex 3D structures. Contrary to finding a minimal set of viewpoints in the Art Gallery Problem (AGP) (Toth, O'Rourke & Goodman, 1997), their algorithm samples the waypoints so that full coverage is ensured with a short connecting path. A triangular mesh representation of the building is already available for the algorithm to compute the desired path. Although this study demonstrates a full-proof application, it only deals with data acquisition and planning.

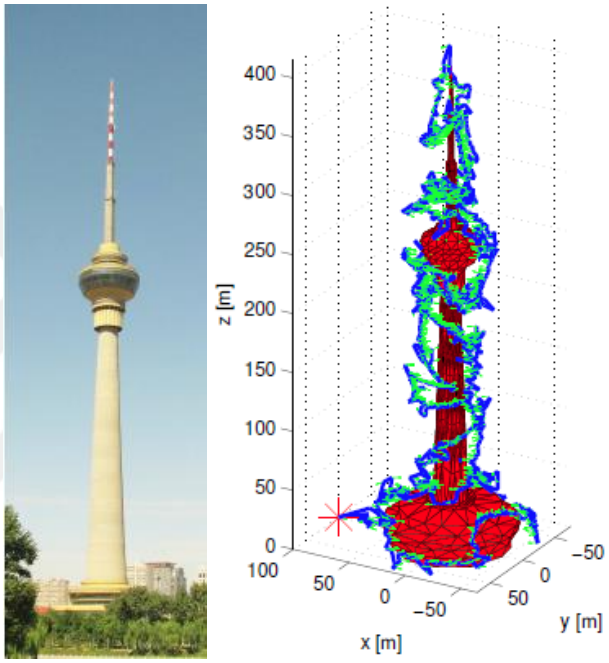

Figure 2.5 The 405m high Central Radio & TV Tower in Beijing inspected by the planning approach proposed by Bircher *et al.* (2015)

2.3 Building Inspection Using UAVs

After robotics community achieved autonomous navigation of UAVs, these vehicles are implemented to utilize complex tasks such as building inspection. In their work, Moranduzzo and Melgani (2014) emphasized that;

"*Unmanned aerial vehicles can reach inaccessible and dangerous areas and thus avoid endangering the lives of people. Traditional monitoring techniques require manual inspections of structures which commonly are expensive and time-consuming…*"

During the last decade, a huge effort is put on employing aerial robots in building inspection/monitoring. In 2007, Metni and Hamel presented a micro helicopter system using computer vision approaches to be able to inspect bridges. Their focus is visual servoing for local control of this system.

Figure 2.6 Mono camera integrated helicopter developed for bridge inspection (Metni & Hamel, 2007)

A year later, Rathinam *et al.* (2008) presented a vision-based tracking system in order to monitor infrastructures such as pipelines, roads etc. via fixed-wing type UAV. Andre and Simoes, in 2009, developed a manually operated airship for monitoring purposes. Serrano (2011) introduced a UAV system for inspecting culverts utilizing GPS, LIDAR and IMU. The data acquired from these sensors are fused to estimate the state enabling autonomous outdoor navigation. Although these works are valuable in terms of demonstrating different use cases of robots in building inspection, they focus on navigation and data acquisition.

Figure 2.7 Image of the quadrotor mounted on a cart for data collection (Serrano, 2011)

Eschmann *et al.* (2013) reported a UAV system equipped with different sensors to be able to navigate semi-autonomously with GPS-guided control as well as manually controlled by an operator. The pictures taken by the system are stitched to construct 2D maps and processed for crack inspection when the vehicle comes back to the ground station.

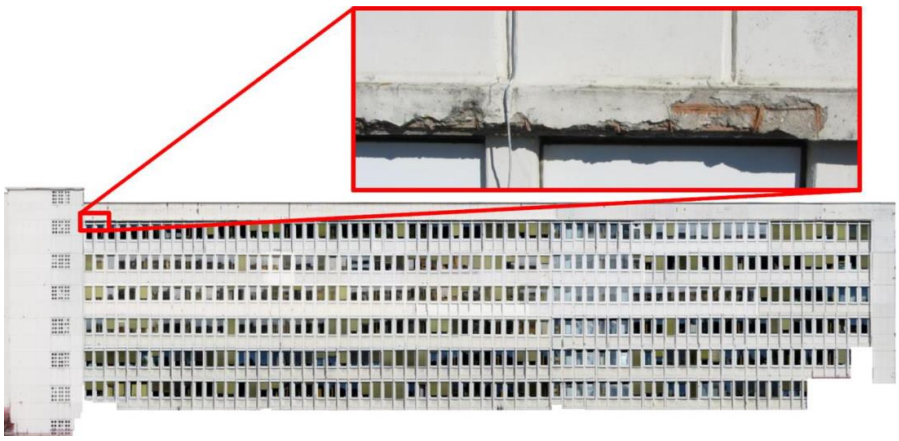

Figure 2.8 Digital facade reconstruction based on images (Eschmann *et al.*, 2013)

Moranduzzo and Melgani (2014) presented a methodology to monitor the changes due to corrosion damages on industrial plants by using UAV. Images acquired at different instances are aligned through geometric transformation to highlight the changes above a threshold which is automatically determined by assuming damages that have usually different aspects with respect to the surrounding structures.

Nikolic *et al.* (2013) demonstrated a quadrotor MAV integrated with a stereo camera configuration that can explore GPS-denied environments. They validated the system by autonomous flights inside an industrial boiler. The study is important since it demonstrates autonomous indoor navigation for inspection as in the case of this thesis.

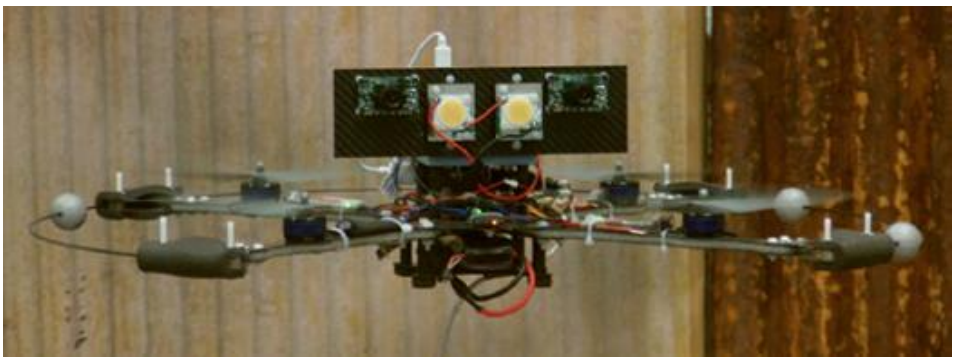

Figure 2.9 Stereo camera integrated quadrotor MAV for industrial boiler inspection developed by Nikolic *et al.* (2013)

Moreover, Høglund's quadrotor system with IMU and monocular imaging for inspection of wind turbines (2014); Araar, Aouf and Dietz's power pylon detection approach (2015); manually operated UAV for 3D map reconstruction by Omari, Gohl, Burri, Achtelik and Siegwart (2014); the works of Winkvist, Rushforth and Young (2013) and Özaslan, Shen, Mulgaonkar, Michael and Kumar (2015) can be listed as further examples of UAV implementations for inspection purposes. Although, these studies constitute a basis for the present study, they mostly contribute in terms of autonomous navigation for building inspection.

In summary, the use of UAVs in the inspection phases of buildings is demonstrated in data (i.e. images, point clouds) acquisition in the literature. However, secondary examination (revisiting) is important in periodic inspections in terms of time efficiency. The number of researches on this context is limited in the literature. Therefore, revisiting locations of interest for detailed inspection by UAVs is aimed in this study to be able to extend the studies.

CHAPTER III

OVERVIEW OF THE SYSTEM

In this study, the proposed approach is to expand the area of use of the unmanned aerial vehicles for building inspection/monitoring. The aim of this research is to achieve autonomous navigation and task planning of UAVs both in GPS-denied and/or outdoor environments to be able to perform automated building inspection operations. This task planning strategy enables UAVs to autonomously navigate in different environments while proposing a decision making tool for evaluation of the acquired data.

State of the art technology and methodology are employed to be able to reduce the risks and increase efficiency in visual building inspection. The major contribution of this dissertation can be stated as a complete implementation for autonomous building inspection considering not only the mapping phase but also subsequent phases such as revisiting a defected location.

An autonomously navigating quadrotor UAV with ability of revisiting pinpoint locations is introduced. SLAM using onboard visual-inertial sensor fusion to explore the environment in which the UAV is located; motion planning with obstacle avoidance and geometric-visual reconstruction of the environment are implemented to accomplish the aim.

A CNN is trained to identify cracks to facilitate high-level decision making for determining locations that would be revisited. A graphical user interface is developed to wrap the functionalities.

In order to validate and verify the methods by testing, a commercial quadrotor UAV is integrated with onboard sensors. Software is implemented by both using open source libraries and packages supported by the community and by developing the new ones in the scope of this study. Figure 3.1 presents a schema of the software architecture for the overall system. All the computations for autonomous navigation are done onboard apart from GUI that runs image classifier on a ground station computer. The software developed in the scope of this work is open-source and can be reached online[1].

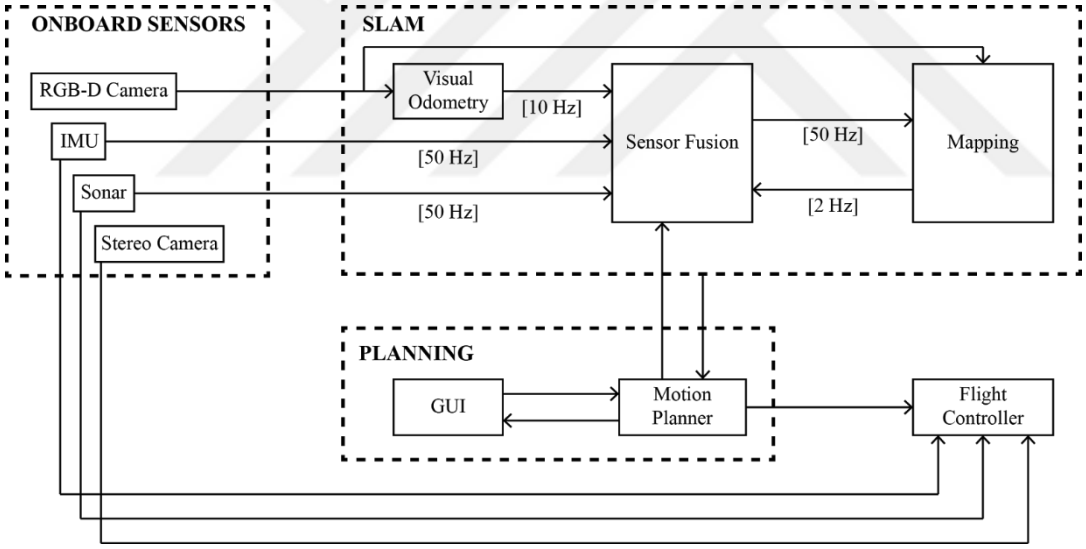

Figure 3.1 Schematic of the high-level software architecture

[1] https://github.com/fatihksubasi/orko

Low-level (attitude and velocity) control of the vehicle is achieved by the flight controller of the quadrotor platform. SLAM and motion planning strategies are implemented with modified open source software. RGB-D camera is used as the main source for the visual odometry and mapping processes. It is fused with onboard IMU and ultrasonic sensor for state estimations. The details of the autonomous navigation framework can be found in the next chapter.

Besides, a CNN is deployed as an image classifier for the crack detection on concrete surfaces. It presents an optional support mechanism for task planning of revisiting locations during inspection. It is built on top of autonomous navigation capability of the UAV with a user interface.

A GUI is developed to launch high-level planning of the tasks for flights. It runs on a ground station computer and demonstrates the capabilities of the system and enables users to use it without a priori knowledge of robotics. Figure 3.2 illustrates its workflow. It wraps functional callbacks for planning, motion control and other features for visualization purposes such as live video stream.

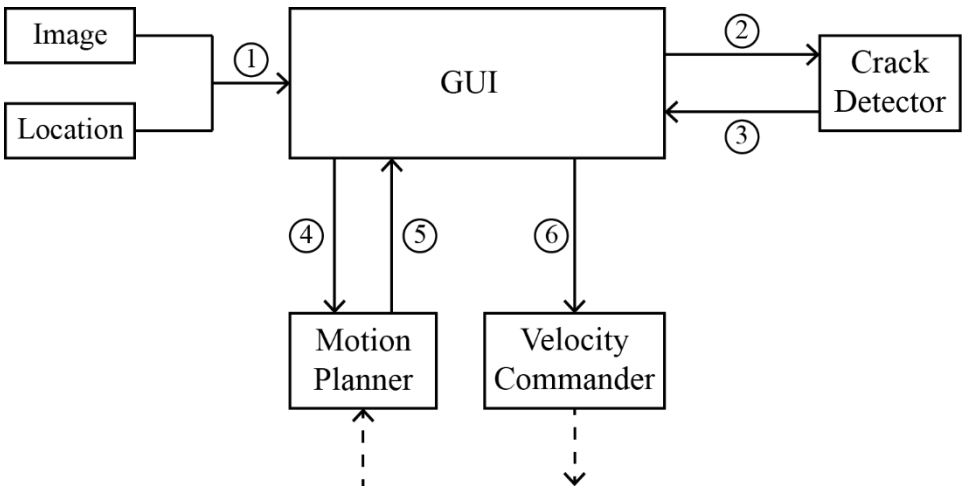

Figure 3.2 Workflow of the GUI developed for task planning

The step-by-step workflow of the proposed task planning approach is presented below as illustrated in Figure 3.2:

1. The GUI shows the images captured during flight for the user to choose as a revisiting location. Since these images are previously matched with locations, each image corresponds a location (position and orientation) in the algorithm.
2. (Optional) If the user decides to utilize the crack detection approach that is introduced as a support for decision making of locations to revisit, the crack detector processes the images.
3. If Step-2 is executed, the GUI visualizes the new set of images in which cracks are identified.
4. After a location for revisiting is determined by its corresponding image, the GUI sends this goal to the motion planner.
5. It receives an obstacle-free optimal trajectory as the form of waypoints if available.
6. An algorithm calculates the required velocities between these waypoints for the UAV to cover the path. If the task is executed or there is no feasible motion plan available, the GUI reports feedback.

CHAPTER IV

DEVELOPMENT AND INTEGRATION OF THE SYSTEM

In this chapter, the theoretical background is addressed and the materials and the methods used in the implementation of the system are presented. Hardware components and features of the system, followed by software integration and development are explained.

4.1. Hardware

The hardware used in the research is presented in this section. It includes the UAV platform, sensors, onboard computer and ground station employed both in simulations and real-world tests.

4.1.1 UAV

DJI Matrice 100 (2014) developer platform is used as the aerial platform. It is a vertical take-off and landing (VTOL) quadrotor vehicle with reconfigurable hardware installation capability. The UAV meets the requirements of this research since it has an onboard low-level flight controller that handles attitude control.

Its Robot Operating System (ROS) integrated software development kit (SDK) (2017) enables attitude, velocity and position control of the quadrotor. Moreover, the SDK sends onboard sensor stream (IMU, GPS etc.) over ROS.

Figure 4.1 DJI Matrice 100 (2014)

4.1.2 Sensors

An onboard visual sensing system composed of five units of low resolution stereo camera and one processor named Guidance (Zhou *et al.*, 2015) is integrated. The processor makes obstacle avoidance possible by fusing stereo camera data, IMU and ultrasonic sensors. It also works seamlessly with N1 flight controller of Matrice 100.

Microsoft Kinect v1 (2010) is utilized both for mapping and localization as the onboard RGB-D sensor. Kinect is a widely used and open sourced hardware. It is composed of an RGB camera and infrared depth camera with 43° vertical by 57° horizontal field of view at 30 frames per second.

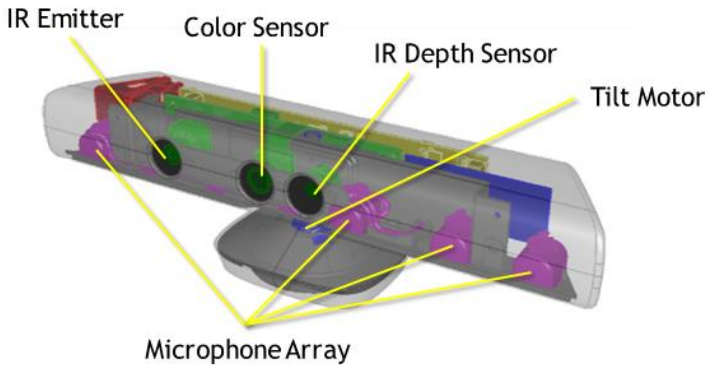

Figure 4.2 Microsoft Kinect components (Microsoft, n.d.)

A low cost, lightweight step-down voltage regulator is integrated to reduce the voltage supplied by Matrice 100 power source (20-26.1 V) so that Kinect's input voltage requirement (12 V) can be satisfied.

4.1.3 Onboard Computer

Onboard computer is one of the most important components for fully-autonomous UAVs since all the computations regarding SLAM and planning tasks are done onboard.

DJI Manifold (2015) is a lightweight onboard computer that is compatible with Matrice 100 platform. Its power consumption is relatively low (~15 W) which is a critical specification in terms of increasing flight time of a limited power-supplied UAV. Manifold has a quad-core, 4-plus-1 ARM processor, NVIDIA Kepler-based GeForce graphics processor, 2GB memory with customized version of Linux Ubuntu 14.04LTS. It is also equipped with a wireless connection chip and antennas for communication purposes.

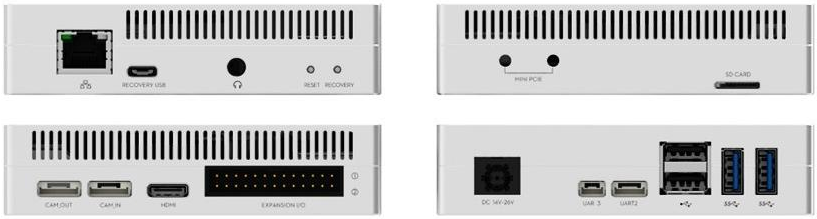

Figure 4.3 DJI Manifold (2015)

4.1.4 Ground Station

Two ground station computers are utilized in order to perform simulations and communicate with the onboard computer during testing phases. For simulations, a desktop workstation which has 16 core Intel Xeon CPU at 3.5GHz, NVIDIA Quadro M4000 GPU and 64GB memory. It operates with Linux Ubuntu 14.04 LTS operating system. Another computer with quad-core Intel Core i7 CPU at 2.7GHz and 6GB memory is used in testing phases as the ground station so that the communication over ROS can be achieved. The latter also runs the developed interface that is explained in the next sections.

Figure 4.4 Fully integrated UAV used in the implementation of this research

4.2 ROS

Most of the implementation throughout this study is done using Robot Operating System. Therefore, an overview of ROS is presented in this section. ROS is a widely-used open source middleware for robotic software applications. It presents easy-to-integrate tools, libraries and conventions developed by the robotics community.

"*...ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work…*" (Open Source Robotics Foundation, 2007)

Furthermore, ROS enables distributed and modular software design that increases flexibility. It offers inter-process communication (message passing) at the lowest level so that one does not need to reimplement this type of a time-consuming basic task. The ROS communication protocol supports:

- Publish/subscribe anonymous message passing
- Data recording
- Request remote calls (services)
- Distributed parameter system
- Message definitions for robots
- Robot geometry library
- Robot description language
- Diagnostics
- State estimation
- Localization, Mapping
- Navigation

ROS also provides several tools for debugging, plotting, visualization and simulation. Its distributed architecture comprises of master, nodes, topics, messages, services and parameters. Master helps nodes to find each other. Nodes are executables that communicate with each other. Nodes can publish or subscribe to topics which are buses to exchange messages. Messages are ROS data type protocols. Services provide a different way for nodes to communicate. When a node runs as a server, it can be sent requests and received responses rather than streaming messages as in the case of topics.

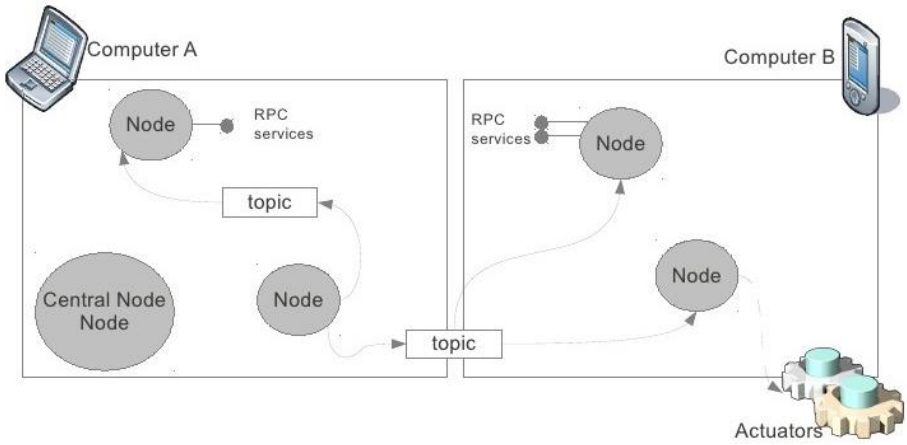

Figure 4.5 Example of distributed ROS Network (Blasco, 2012)

In consideration of these features, ROS Indigo on Ubuntu 14.04 LTS is used during the implementation phase of this work. ROS packages and nodes referred through this dissertation have been employed and developed to be able to take the advantage of modular software elements presented by the robotics community. The software architecture is deployed using ROS framework.

Table 4.1 presents a brief of the software used in this research. The table shows the software that are uniquely developed in the scope of this work along with available open source software that are adopted.

Table 4.1 Summary of the software packages used in this work

Software Module	**Available Open Sources**	**Developments in the Scope of This Work**
Local (attitude) control	- Velocity control service of DJI SDK	- Development of a node to translate the velocity commands for the UAV (Appendix A)
Visual odometry	- RGB-D odometry node of RTAB-Map	- Development of a node is for setting visual odometry to the output of the sensor fusion in case that visual odometry gets lost
Sensor fusion	- EKF node of robot_localization package	- Downward facing ultrasonic sensor for indoor height estimations
Mapping	- Mapping node of RTAB-Map	
Motion planning	- MoveIt! with OMPL backend - Tonioni's (2013) approach for quadrotor motion planning in MoveIt!	- Modification of the physical model of the UAV for collision detection in motion planning - Development of the interoperability of the motion planner and the task planner (Appendix B)
Crack detection	- Pretrained InceptionV3 CNN model	- Fine-tuning the CNN using Keras with TensorFlow backend - Development of an testing algorithm for detecting cracks in the images (Appendix E)
Task Planning		- Development of a node for matching the images with their corresponding locations (Appendix D)

Software Module	Available Open Sources	Developments in the Scope of This Work
		- Development of an algorithm for revisiting locations of interest. It sends the goal to the motion planner and computes the velocities from the corresponding trajectory. It runs as the backend of the GUI
GUI		- Development of the original GUI (Appendix C)

4.3 Localization and Mapping

Localization of a mobile robot in the environment it operates is one of the fundamentals of autonomous navigation. This requires a representation (map) of the environment for the robot to localize itself relative to this representation. In this section, a collection of tools, algorithms and approaches are presented in order to overcome SLAM problem in the scope of this research. The problem is addressed in two fundamental concepts as state estimation and mapping, respectively.

4.3.1 State Estimation

The state comprises of relative position, orientation and linear-angular velocities in 3D considering UAVs. Global state estimation in an environment is essential both for localization and control of the UAV.

Transformation between coordinate frames can be stated as one of the essentials of state estimation so that relative pose (position and orientation) can be computed. The required transformations for autonomous navigation in this implementation can be seen in Figure 4.6. Real-time conversions are computed using tf package of ROS.

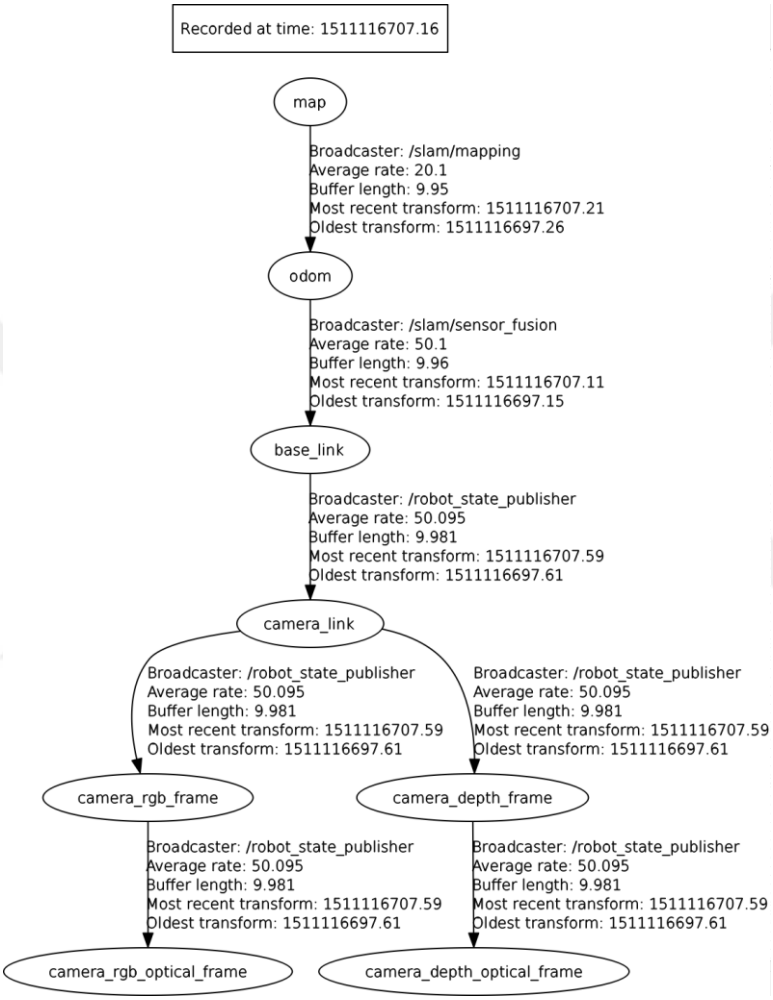

Figure 4.6 ROS tf tree of the implementation

The ‘map’ frame is the frame where representation of environment of interest placed and it is the reference frame. The ‘odom’ frame, on the other hand, is the localization (odometry) frame calculated using the measurements from the sensors. ‘base_link’ frame denotes the quadrotor UAV’s base. Thus, the transformation between base_link and camera_link is static since camera is rigidly attached to the vehicle. The transformations between ‘map’ and ‘odom’ frames are calculated and broadcasted at approximately 20 Hz by RTAB-Map ROS node (2013) which is denoted by ‘/slam/mapping’. The transformations between ‘odom’ and ‘base_link’ are computed by robot_localization (2013) ROS node. These two nodes are clarified in the next sections.

As emphasized in the previous chapter, inertial measurements are generally insufficient for global (position) control of UAVs because of drift phenomenon. Therefore, it is necessary for the system to navigate utilizing other sensors to fuse the data. In the extent of this research, cameras are employed as visual sensors so that visual-inertial state estimation can be obtained. In this context, computing visual odometry is necessary.

4.3.1.1 Visual Odometry

Odometry is the data estimating change in position over time and visual odometry is a computer vision technique to estimate the state using sequential images. Visual odometry outperforms especially in GPS-denied environments. A classification of visual odometry can be addressed by the camera types as monocular and stereo. There is an ongoing trade-off between those. However, both monocular and stereo approaches have advantages and disadvantages over each other.

In monocular visual odometry, motion scale is unobservable while stereo approach presents metric results. On the other hand, for relatively long distances, stereo odometry results in a similar manner with the monocular case. Thus, stereo approach is a decent choice for relatively near-field image capturing as in the case of building inspection.

In the implementation of this work, a consumer grade RGB-D camera is employed for visual odometry since depth information requirement of stereo approach is already satisfied so that no extra computation cost needed.

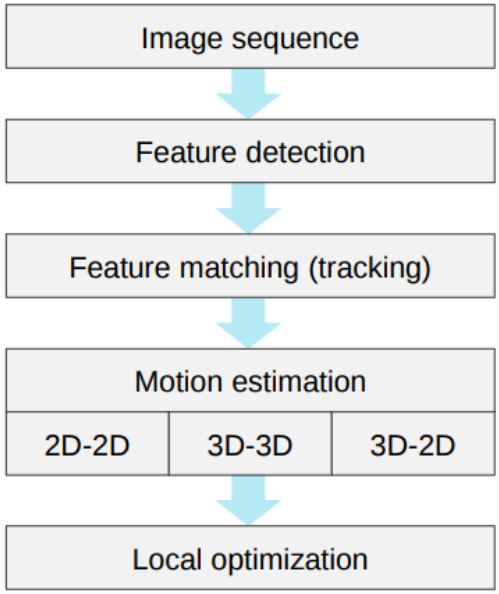

Figure 4.7 General visual odometry pipeline (Scaramuzza, n.d.)

In the present study, RTAB-Map's visual odometry node is utilized since it is easy-to-integrate and robust. The node supports both RGB-D and stereo cameras. It provides Frame to Map and Frame to Frame odometry strategy options. Additionally, pose estimation strategy can be selected as either Kalman or Particle filtering.

After the image rectification and the depth registration, RGB and depth images captured by the onboard camera are fed to the visual odometry node at an adaptive frequency so that computational cost is minimized. Transformations are also predicted based on previous motion as an extra layer for consistency. The state initialized aligning with ground to correct the camera angle and height. FAST and BRIEF feature detection algorithms are combined because it is computation-wise efficient than the other available algorithms in RTAB-Map such as SURF and SIFT (Labbe, 2014).

The visual odometry data obtained by this method is used as a complement to inertial measurements acquired from the onboard IMU. In the next section, this sensor fusion framework is described.

4.3.1.2 Sensor Fusion

Elmenreich (2002) defines sensor fusion as:

"*... is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually.*"

As discussed earlier, state estimation using only inertial measurements are prone to drift. On the other hand, orientation estimations (especially roll and pitch angles) of visual odometry have poor performance than inertial measurements. Thus, visual sensor data is fused with inertial measurements to make estimations more robust and reliable. Visual-Inertial navigation principles are applied throughout this work. For sensor fusion purposes, a ROS node named robot_localization is employed. The node has applications of both Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). However, EKF approach is preferred in this implementation due to its lower computational cost. The node presents many parameters to configure the filter as needed.

In the extent of this study, the robot_localization node broadcasts the state estimates (position, orientation and velocities) to the ROS network at 50 Hz in the implementation. It is also responsible for transformations between 'odom' and 'base_link' frames. Linear velocities (in x and y directions) from visual odometry and orientations (roll and pitch) from inertial measurements are used.

Yaw angle computed by fusing the yaw rate measurements coming from both of these sensors since gyros are prone to have error in absolute yaw information because of the perpendicular direction of gravity to the yaw direction. Yaw angle of IMUs are generally prone to have error (Neto, Mendes & Moreira, 2015).

In z direction (height), downward facing Guidance stereo kit's ultrasonic sensor is utilized since it provides more accurate estimates in this direction than visual odometry. This is demonstrated in the next chapters. Figure 4.8 shows the corresponding data from which the filter is fused.

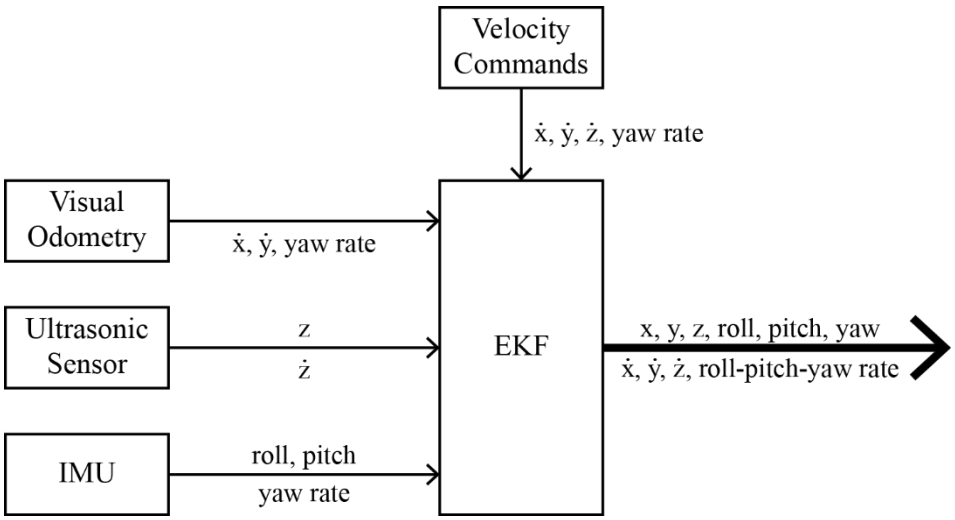

Figure 4.8 Sensor fusion dataflow

In case that visual odometry gets lost (i.e. because of poor feature correspondence or lighting), a node (odometry_correction) is created so that visual odometry is instantaneously set to the filter's estimation. Even if this approach may cause drift in position estimation, it helps the robot gets long-term autonomy if visual odometry recovers in short-term.

4.3.2 Mapping

While visual-inertial odometry only aims to the local consistency, the complete SLAM framework ensures the global consistency. This requires mapping with loop-closure detection so that errors accumulated in odometry can be compensated. Several map representation applications are already investigated in the previous chapter. Here, a graph-based mapping approach is used presented along with extended implementations considering the requirements of the objectives.

In this thesis, a graph-based mapping approach is employed with RTAB-Map's (Real-Time Appearance-Based Mapping) ROS node. It has an "*approach based on an incremental appearance-based loop closure detector. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environments are always respected*." (Labbé, n.d.)

RTAB-Map is also advantageous in online large-scale and long-term operations since it has a loop-closure detection approach based on a memory management mechanism. Thus, it is convenient to use RTAB-Map for building inspection operations as building inspection generally requires large-scale maps. It is also easy-to-implement using ROS and proven for RGB-D mapping.

In this work, RTAB-Map is also used for mapping purposes. It subscribes the visual-inertial odometry information at 50 HZ broadcasted by sensor fusion node and publishes registered point cloud data of the environment. Combination of FAST and BRIEF feature detection algorithms as in visual odometry is also adopted in mapping for consistency.

Rectified RGB images and registered depth images captured by the onboard camera are then fed into the mapping node at 2 Hz for loop closure detection and point cloud generation. 'map' to 'odom' coordinate frame transformations are calculated by this node so that 'map' to 'base_link' transformations eventually are linked. This also implies that a complete SLAM pipeline is accomplished after these steps.

Furthermore, an ROS node presented in Appendix D is developed in order to match images with poses (position and orientation) where the UAV had been visited during inspection for possible revisiting in future missions. This algorithm subscribes to both visual-inertial odometry and RGB image topics. Then, it matches them using Approximate Time Synchronizer message filter of ROS in a predefined period. The images saved with their corresponding poses in favor of the task planner that is described in the next sections.

4.4 Planning

SLAM processes described in the previous section manage how to estimate both the state of the UAV and the environment in which it operates. Moreover, it should be noted that low-level (attitude) control of the vehicle is out of scope of this implementation since it is handled by the onboard controller (N1 controller) of the employed UAV. Thus; stable control of roll, pitch and yaw axes is already achieved.

On the other hand, planning of movement of the vehicle with obstacle avoidance during building inspection tasks is the main objective of this work along with the planning of the task itself. Thus, the high-level planning problem boils down to two-fold quests as; path (trajectory) planning considering motion constraints and task/mission planning.

4.4.1 Motion Planning

A path planning strategy in 3D space considering the objectives of this study should be taken into account along with the motion and geometric constraints of the UAV. Obstacle avoidance while in action, on the other hand, is important in terms of safety of both humans and the robot.

In this context, a motion planning framework, Moveit!, is adopted in this work since it is easy-to-integrate using ROS and provides flexibility in terms of several path planning strategies in application such as; Rapidly Exploring Random Trees (RRT), Probabilistic Roadmap (PRM), Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE), Expensive Space Trees (EST) and their variations.

"Moveit! is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains." (Sucan & Chitta, 2011)

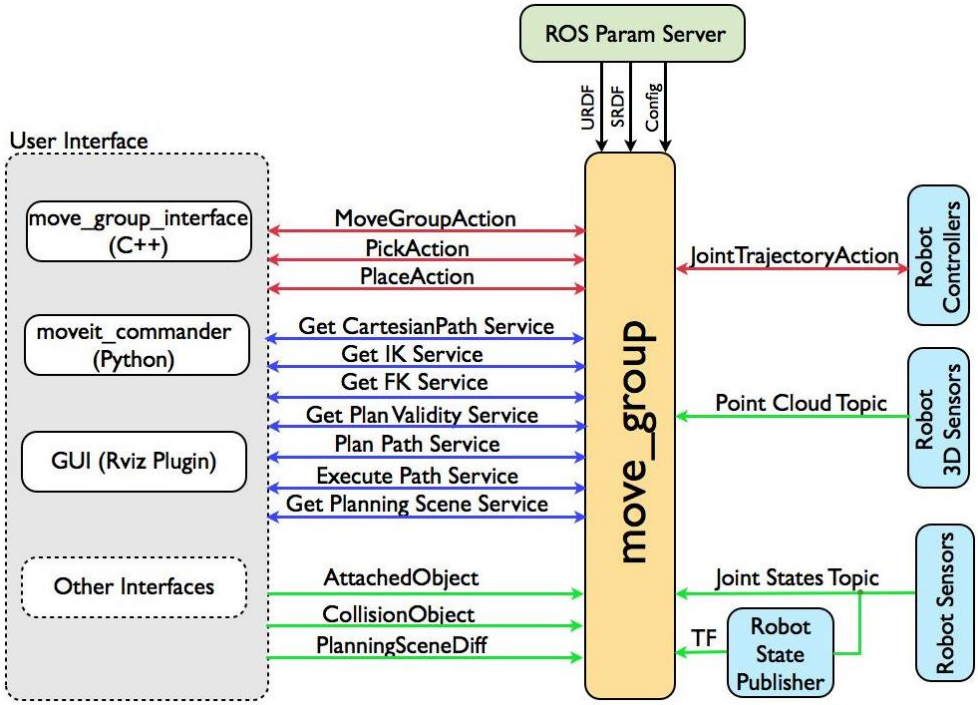

Figure 4.9 High-level system architecture for the primary ROS node (move_group) of MoveIt! (Sucan, & Chitta, n.d.)

Figure 4.9 shows the system architecture of the ROS node of MoveIt!. It subscribes point cloud (map) topic in order to plan a collision-free paths. The joint states topic in this implementation is the visual-inertial state estimation of the quadrotor since the planning is computed between base_link and map_link. Robot State Publisher broadcasts static transforms between camera_link and base_link. The ROS node looks for URDF (Unified Robot Description Format) and SRDF (Semantic Robot Description Format) files that contain the robot's physical parameters including a mesh representation. This data is used in search of collision-free paths along with the 3D voxel grid representation of the environment. MoveIt! also presents a multi-domain interface for users.

MoveIt! uses Open Motion Planning Library (OMPL) (Sucan & Chitta, 2012) as the back-end for motion planning. OMPL is an open-source motion planning library. It provides sampling-based motion planners such as RRT, PRM, KPIECE and their variations.

Moreover, benchmarking of these planners available for a set of planning problems as described in Moll, Sucan and Kavraki's research (2015). It is valuable to determine the most convenient planner for the requirements of such a problem as in this work by the help of this benchmark results. Figure 4.10 shows the benchmarking results of available OMPL planners in MoveIt! package. It can be claimed that EST planner outperforms for the given abstract problem. However, as Tonioni (2013) stated, RRT* gives the best performance considering optimality and motion constraints for a quadrotor UAV motion planning problem. The performance of these planning algorithms is evaluated in detail in the case studies section of the dissertation.

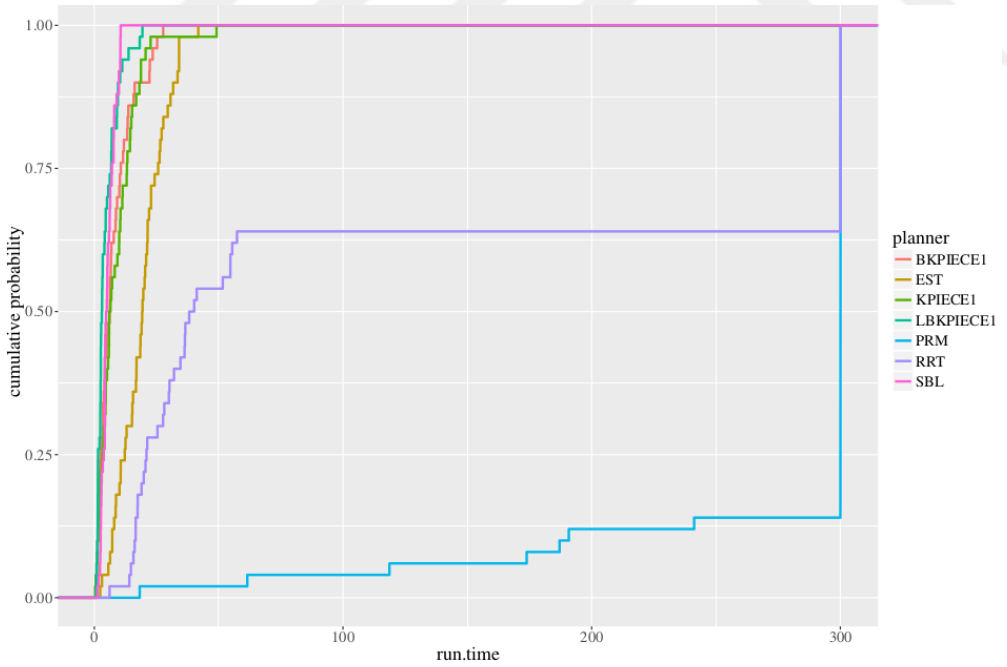

Figure 4.10 Benchmarking results of available OMPL planners in MoveIt! (Moll, Sucan & Kavraki, 2015)

On the other hand, MoveIt! is originally a mobile manipulation software package that only supports robot arms and not suitable for a quadrotor motion planning problem. Tonioni (2013) solves this problem by defining a floating joint between ground (map_link) and quadrotor base (base_link). This solution enables Moveit! to plan collision-free paths for an aerial robot, so; it is adopted in this work.

In the implementation, the motions are planned assuming the environments as static, even though real-time obstacle avoidance takes place. This assumption as the motion plans are computed in static buildings.

After planning motions, move_group node provides the trajectory as set of waypoints. Another node ('velocity_controller'), which is developed in the scope of this work, computes the velocity commands for the vehicle to pursue. The algorithm in the node simply divides the trajectory segments between waypoints to the time that is computed by the move_group node. The broadcasted velocities, then, moves the UAV since DJI SDK allows velocity control of the vehicle in body or ground frame over ROS.

Furthermore, real-time obstacle avoidance is achieved using the onboard sensors (Guidance modules) by the help of flight controller's native capability since they are placed top, bottom, front, back, right and left of the UAV. The vehicle instantaneously stops if it encounters an obstacle along the trajectory, which was not available in mapping session. Thus, another layer of safety is added.

4.4.2 Task Planning

The major contribution of this work stands in task planning of an autonomous navigating mobile robot considering building inspection/monitoring. The task planning approach addresses a task planner which might help performing repetitive revisiting locations of interest during building inspection operations. Along with the autonomous exploration capability that the robotics community provided, it presents a complete autonomous framework for application of aerial robots in building inspection. The proposed revisiting approach produces effective solutions in the case of revisiting a desired pin-point location either at the time of first inspection or at a later time to be able to have a second look and/or maintenance purposes.

Although numerous tasks can be listed in inspection/monitoring phases, this study focuses on one of them in order to verify the proposed approach. In this context, identifying surface cracks is an exemplary task in which professionals aim frequently during visual building inspection operations. Therefore, the cracks from captured images are identified with an algorithm as the backbone of task planner. Moreover, a graphical user interface is developed in order to present these capabilities in a more user-friendly way.

4.4.2.1 Crack Detection

Although crack detection is not the main aim of this study. It is implemented as a demonstration of a decision making tool for revisiting defected locations during building inspection since crack detection is one of the most common objectives in building inspection. For example, systematic bridge inspections are done periodically in six years to detect cracks (Metni & Hamel, 2007).

Non-automated crack detection techniques depend on human workers. However, these are labor intensive, human error-prone, subjective and require expertise. Thus, research has focused on computer vision techniques for crack detection.

The traditional computer vision techniques generally consist of two steps: hand engineered feature extraction and classification of the features. But one main disadvantage of these methods is that they are generally not able to generalize crack detection task in real-world conditions due to diversity in surface texture (i.e. brick, pavement) or variation in lighting (Pauly et al., 2017). Moreover, these techniques fail in the presence of non-crack features such as joints (Gopalakrishnan et al., 2017).

In this context, Convolutional Neural Networks (CNN) are one of the most commonly used architectures since they can overcome most of the contemporary challenges in crack detection (Pauly et al., 2017). They are getting more accurate and robust for image classification in recent years. Hence, CNNs are prominent architectures for surface crack detection.

A convolutional neural network (CNN) consists of one or more convolutional layers (usually a sub-sampling step) and then accompanied by fully connected layers. The architecture of a CNN is designed so that the 2D architecture of images can be used. It is managed through local connections and associated weights followed by pooling. CNNs are easy to train and apply through open source libraries and implementations. On the other hand, training a CNN from scratch is rather hard and time consuming since it requires large datasets. Hence, a common practice, namely Transfer Learning, has emerged. In Transfer Learning, a pretrained network on a large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories) is used as an initialization or a fixed feature extractor for the newly created network.

Fine-tuning is one of the methods in Transfer Learning. This strategy is especially useful when relatively small datasets are available. Therefore, this technique is suitable for the implementation of interest since there is no large dataset available. Hence, it is employed as a complementary of CNN in this study.

Training the CNN

In the scope of this dissertation, Transfer Learning approach is adopted in order to identify surface cracks from acquired images. This approach is suitable in the realm of this study since there is no large datasets available in terms of number of images. It is also appropriate in the sense of time effectiveness because pretrained networks already form a basis for training.

In this regard, InceptionV3 (Szegedy et al., 2016) network model with ImageNet weights is fine-tuned in the present study since it has relatively high performance in top-1 validation accuracy than most of the top scoring single-model architectures (Figure 4.11).

Fine-tuning is achieved using Keras with TensorFlow backend for the implementation. “Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.” (2014). Graphics Processing Units (GPU) are utilized in the training sessions to be able to reduce the time spent. A modified version (Varga, 2016) of ‘Fine-tune Inception v3 on a new set of classes’ example is used for training of the CNN.

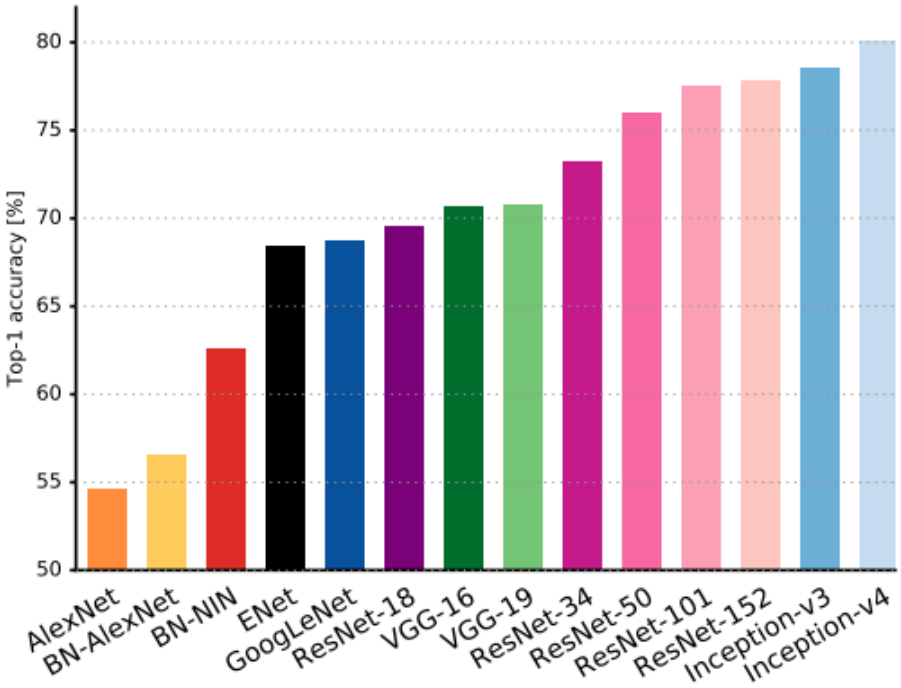

Figure 4.11 Single-crop top-1 validation accuracies for top scoring single-model architectures (Canziani, Paszke & Culurciello, 2016).

Before the training, two classes are determined as ‘Crack’ and ‘NonCrack’ for the classifier. The dataset for training the network is collected from different buildings in Middle East Technical University campus and contains 582 images with cracks and 458 images without cracks (Figure 4.12). The network had poor performance on brick wall images in the first implementation. Therefore, 64 images of brick walls

are added to the ‘NonCrack’ dataset so that the network is able to identify brick texture. Data augmentation function of Keras is used to increase the number of data. The images in the dataset is augmented by flipping horizontally and shifting width and height by 0.125.

Figure 4.12 Sample of images used in training of the CNN (images with cracks at left, images without cracks at right)

Figure 4.13 illustrates the training and the validation accuracies over each epoch. The training accuracy of the model jumps over 90% after 5 epochs. After 20 epochs, the training and the validation accuracies converge to approximately 98%. Figure 4.14 shows the training and the validation losses over each epoch. The losses converge to 0.05 after 20 epochs. These results clearly show that using InceptionV3 as the pretrained CNN is suitable for such a crack detection application.

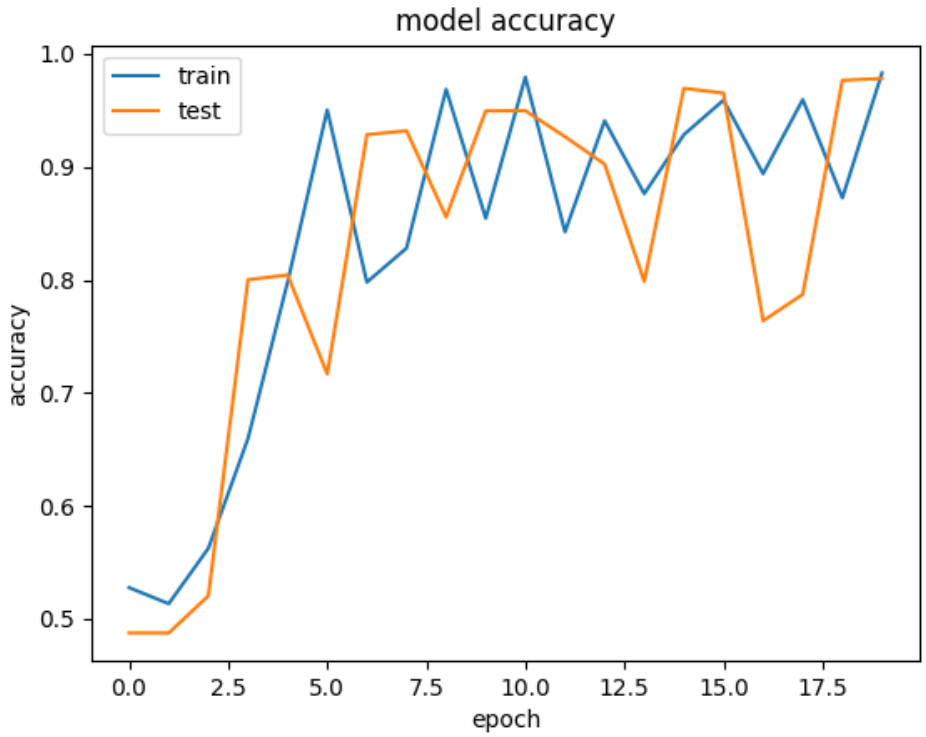

Figure 4.13 Model accuracy vs. epoch number in training and validation sets

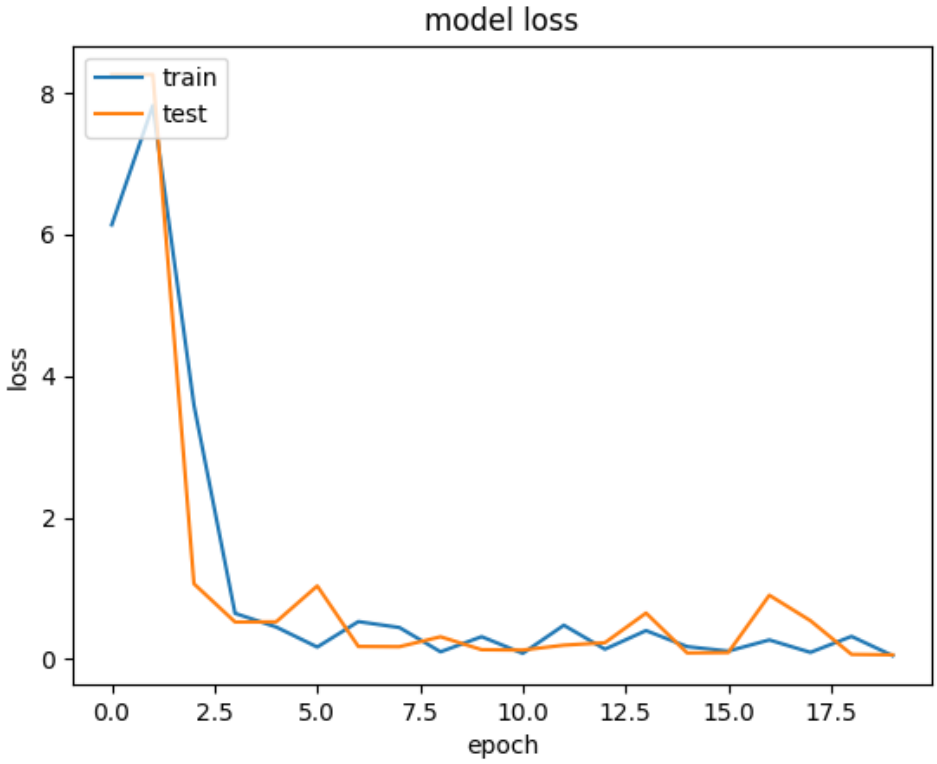

Figure 4.14 Model loss vs. epoch number in training and validation sets

Validation

After the training session, cross validation of the model is conducted by processing a dataset that contains different images than the training dataset. The cross validation dataset consists of 64095 images in total. 19368 of these have surface cracks while there are no cracks in the rest. The fine-tuned model accurately predicts 62417 from the 64095 image. The accuracy is 97.382% in the cross validation.

Additionally, a testing algorithm (Appendix E) is developed considering the needs of this application. First, it loads the trained model with its weights. Then, it processes the images captured during flight whether cracks are present or not. Images classified as 'Crack' are moved to another directory in ground station for the GUI to visualize so that it helps making decisions for selecting locations to revisit.

It is important to state that the surface crack detection is not the main aim of this study. It is rather a demonstration of a decision making tool for revisiting defected locations during building inspection. Hence, it is prone to be developed more in future studies.

4.4.2.2 Graphical User Interface

Kivy (2011) is employed for the development of the Graphical User Interface. Kivy is an open source, cross-platform Python library that is widely used for GUI development. The GUI has three tabs; namely, Home, Mapping, and Revisiting. In Home screen, it offers high-level commands for the UAV such as; takeoff, landing. Moreover, the system including all the ROS nodes can be launched by a switch. A toggle button provides live video stream from onboard camera. Instructions for usage of the GUI settles in the home screen in order to give a brief for users. Figure 4.15 shows the Home screen of the GUI.

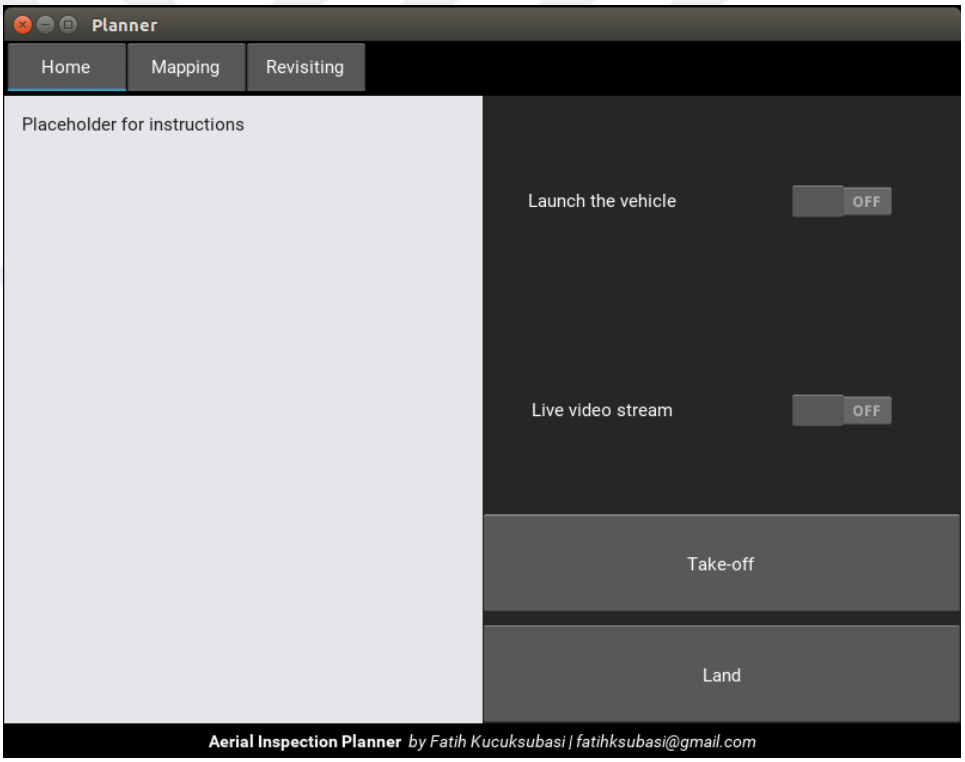

Figure 4.15 Home screen of the developed GUI

There are two main modes of the Aerial Inspection Planner application as mapping and revisiting; although, only revisiting mode is implemented since exploration (mapping) missions are out of the scope of this work. In the Revisiting screen (Figure 4.16), the captured images appear in the sequence of the vehicle's route during flight. A button calls the function to process all images captured for crack detection with the approach described when it is pressed. After this operation, the images with detected cracks reappear in the screen. This provides a first screening before users may determine a location for revisiting. The images can be explored and viewed by the help of a slider or directly inserting the number of the image of interest. Picking an image out is picking the corresponding (goal) position for revisiting. 'Start Mission' button triggers the callback functions of motion planning and flight controller; thus, sends the vehicle to the goal. 'Stop Mission' button, which is a sort of emergency button, stops the vehicle immediately and it hovers at that location.

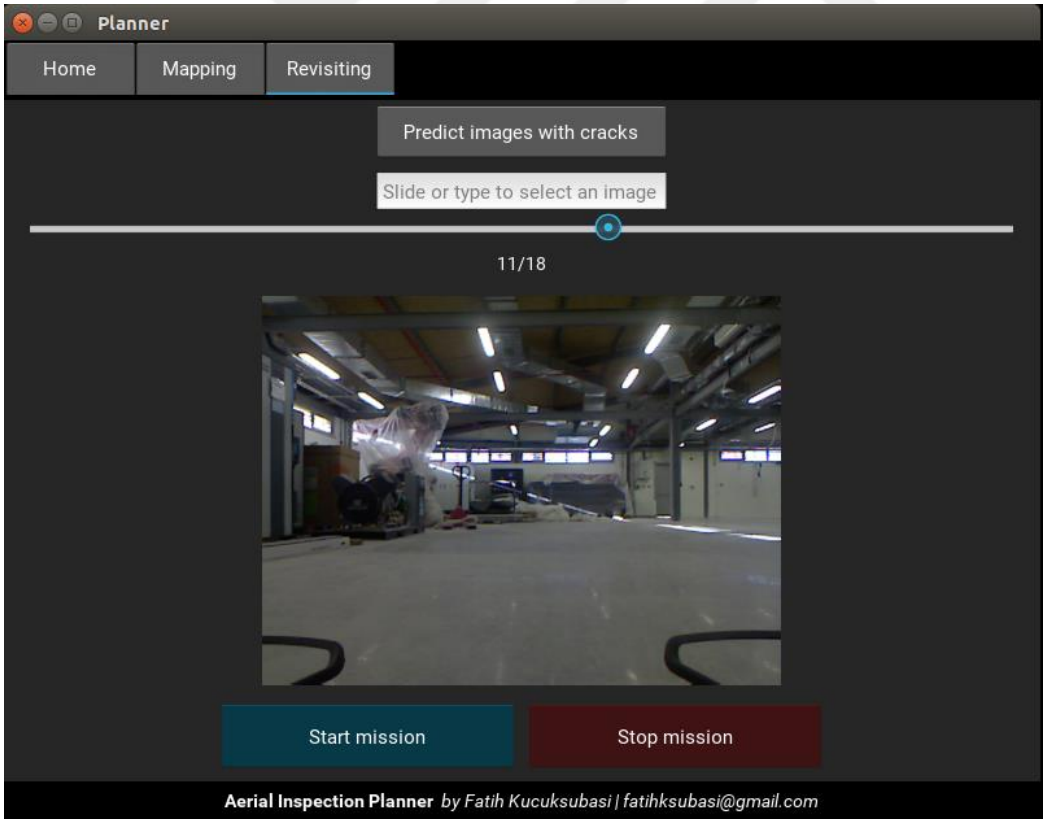

Figure 4.16 Revisiting screen of the developed GUI

CHAPTER V

CASE STUDIES

This chapter presents the results obtained through tests of the proposed system. Computer simulations and real-world tests are conducted in order to validate and verify the hardware and the methods used. The findings are compared ground truth information as long as it is available.

5.1 Simulations

In this study, simulations are carried out to confirm the localization and mapping performance of the system. In this regard, the test environment constructed in simulations is developed to be similar to the environment in which the system actually works. For this purpose, Gazebo simulator which can natively communicate with ROS is used to construct an indoor environment. The constructed environment mimicking an indoor space is employed for visual-inertial navigation of the system to analyze mapping and state estimation performances, respectively.

Figure 5.1 shows the indoor environment used in simulations. Two connected spaces in the environment is enclosed by 3x1x3 m (height x width x length) brick walls which have important texture to test. In this way, the effects on the localization and mapping performance of repetitive monotone patterns can be easily observed because the loop-closures are harder when repetitive patterns are present in the environment.

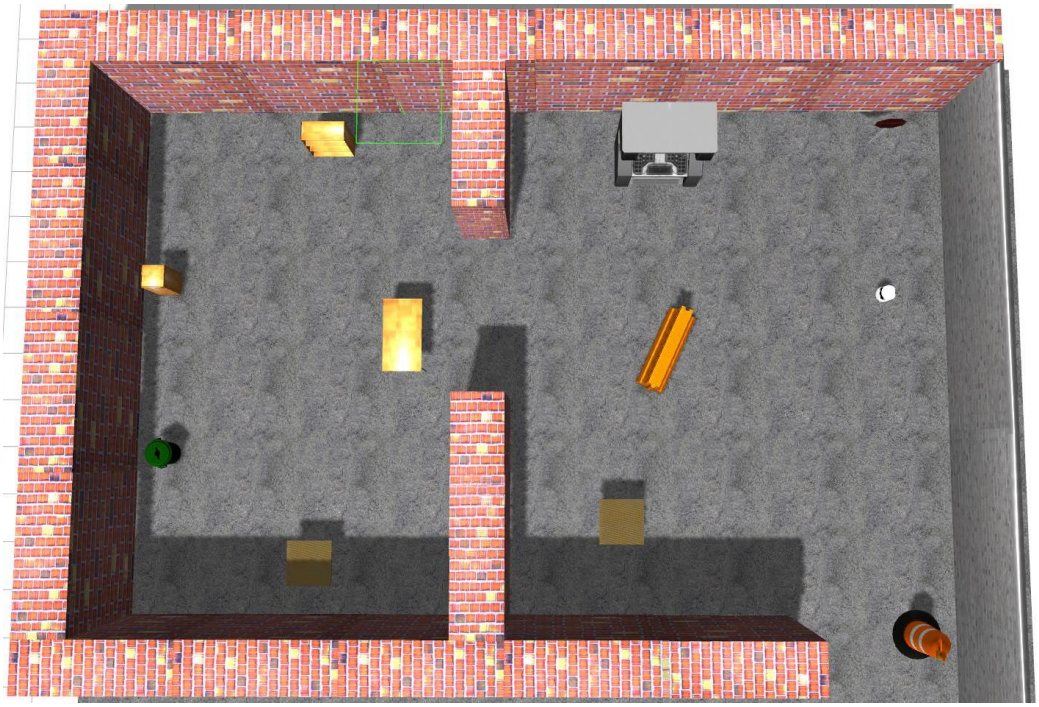

Figure 5.1 Indoor environment used in the simulations

On the other hand, since there is no GPS data in the simulations, the system must be completely dependent on visual-inertial state estimation. The visual data is taken from the model obtained by modeling Kinect, so it is a reliable simulation in this sense. The hector_quadrotor ROS package (Meyer, Sendobry, Kohlbrecher, Klingauf & Von Stryk, 2012) is used for the flight controller model. As in the system that is actually integrated, the flight controller here is also driven by velocity commands. hector_quadrotor package also provides IMU data and UAV's ground truth state estimates. The ground truth data is useful in evaluating the results later.

5.1.1 Mapping

After the simulation environment is established, the tests are performed to evaluate the performance of the mapping approach. Although mapping is not the main objective of what is proposed in this thesis, it is defined as a prerequisite for revisiting a defected location since revisit is a secondary detailed inspection phase. For this reason, exploration of the environment in which any location will be revisited is necessary for detection of defect locations. The tests also form a basis for real-world experiments.

First, an exploration (of the environment) session is conducted by manually operating the quadrotor UAV by covering the space, a 3D voxel grid map is constructed with the mapping approach (RTAB-Map) as discussed in the previous chapter. This map can be seen in Figure 5.2.

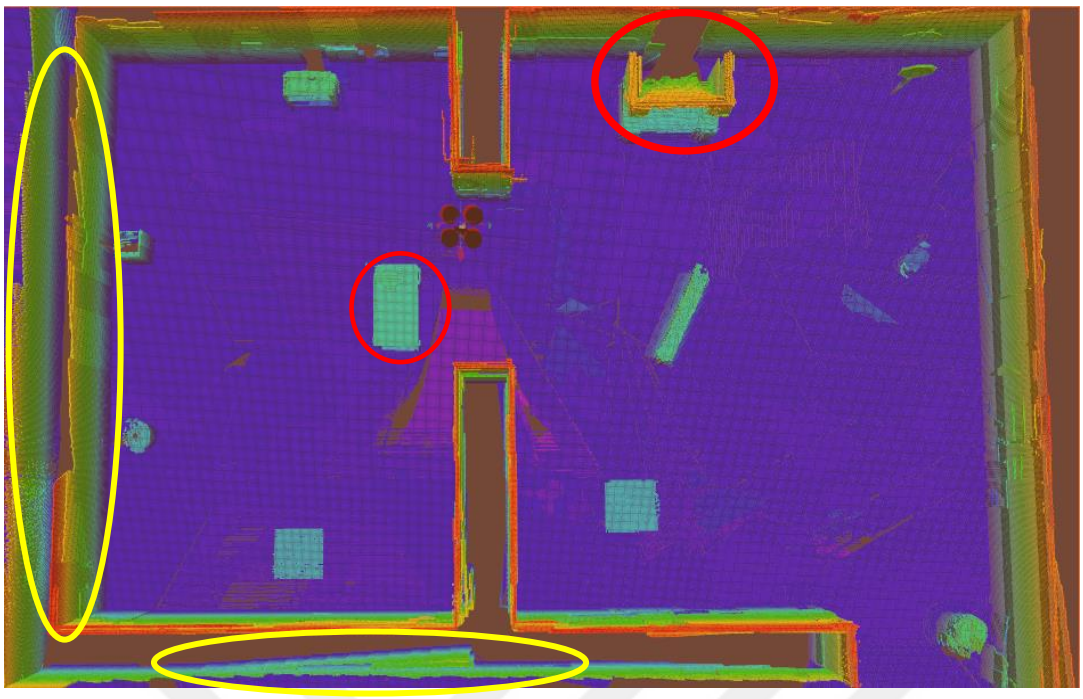

Figure 5.2 3D voxel grid map created after the mapping session

The performance of the mapping is checked by comparing it with the original environment in the simulations. The simulation environment (ground truth map) and the reconstructed map by the mapping approach in this work can be seen in Figure 5.3. Two different evaluations have been considered, global consistency and local accuracy are evaluated, respectively.

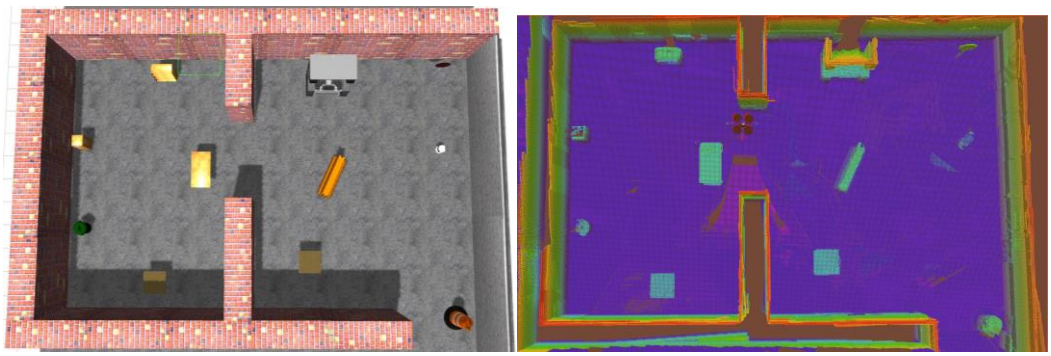

Figure 5.3 The simulation environment (left) and the reconstructed map in the test (right)

For the assessment of global consistency, the reconstructed map is compared to the original map. In this context, it is seen that the map overlaps with the environment created in simulation (Figure 5.3). All surfaces and objects have been successfully reconstructed. However, in some places it is observed that the point cloud cannot be completed. As illustrated in Figure 5.2, this problem occurs in regions marked in yellow. In addition, the point cloud of the region behind the object marked with red is not obtained due to being out of side.

Local accuracy is assessed by comparing the measured values of a point cloud of an object to the actual values in the simulation environment. The table in the simulation shown by red circle in Figure 5.2 is selected for this operation. The width and the depth measurements of this table are used. The lengths required to be 1.5 x 0.8 x 1.0 m [width x depth x height] have been reconstructed as 1.52 x 0.82 x 0.99 m.

5.1.1.1 Discussion

In the map reconstructed through simulations, there are misalignments due to point cloud stitching that is marked by yellow in Figure 5.2 although the dimensions of the map are quite accurate in terms of autonomous navigation purposes. This may be due to the fact that some fields do not coincide the field of view (FOV) during mapping. For this reason, the upper parts of the brick walls may not have been reconstructed as point clouds.

In conclusion, the resolution and accuracy of the 3D map obtained in simulations is sufficient for autonomous navigation. The risk of crashing is very low given the system's obstacle avoidance feature. This leads to the conclusion that the mapping process is successful in accordance with the requirements of this study.

5.1.2 State Estimation

The second test is performed to evaluate the state estimation performance of the system in the same simulation environment used in the mapping tests. As explained in the previous sections, state estimation is crucial for autonomous navigation of the UAV. On the other hand, it has an additional importance in the scope of this thesis for revisiting a pinpoint location which is the main aim because it is one of the basic requirements for the UAV to be able to identify precisely the locations visited and then relocate them.

In order to evaluate the state estimation performance of the system, the position (global x-y-z) and orientation (yaw) estimates are compared with the corresponding ground truth values. The ground truth values are obtained from the simulation environment. Visual odometry and visual-inertial odometry results are plotted along with ground truth values (Figure 5.5, 5.6, 5.7, 5.8). The estimates are closely following the actual measurements during the route covered in the simulation (Figure 5.4).

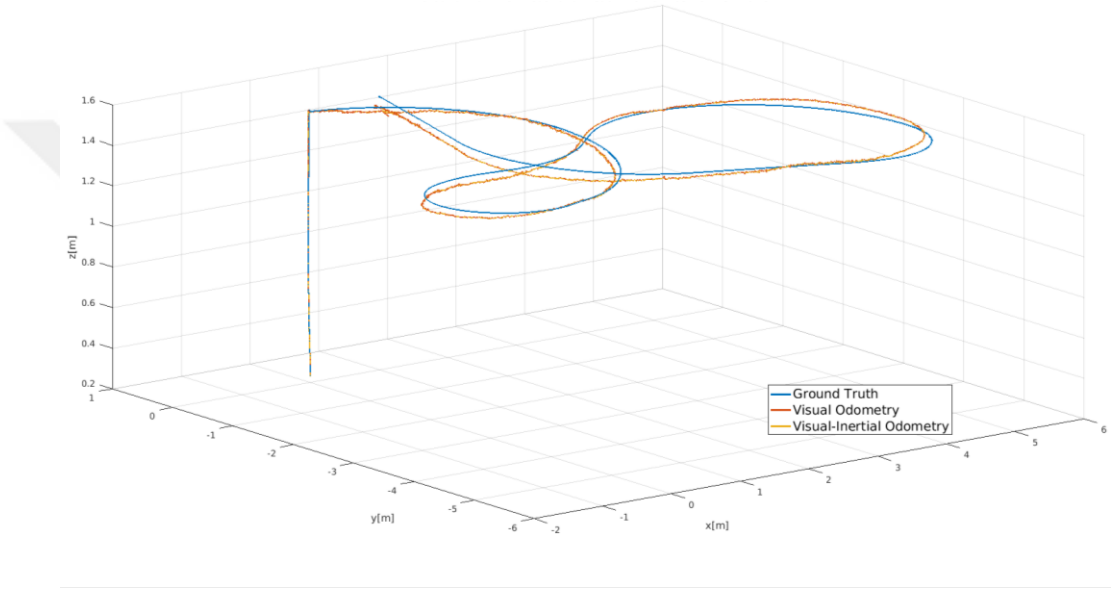

Figure 5.4 Ground truth trajectory vs. Estimated trajectories

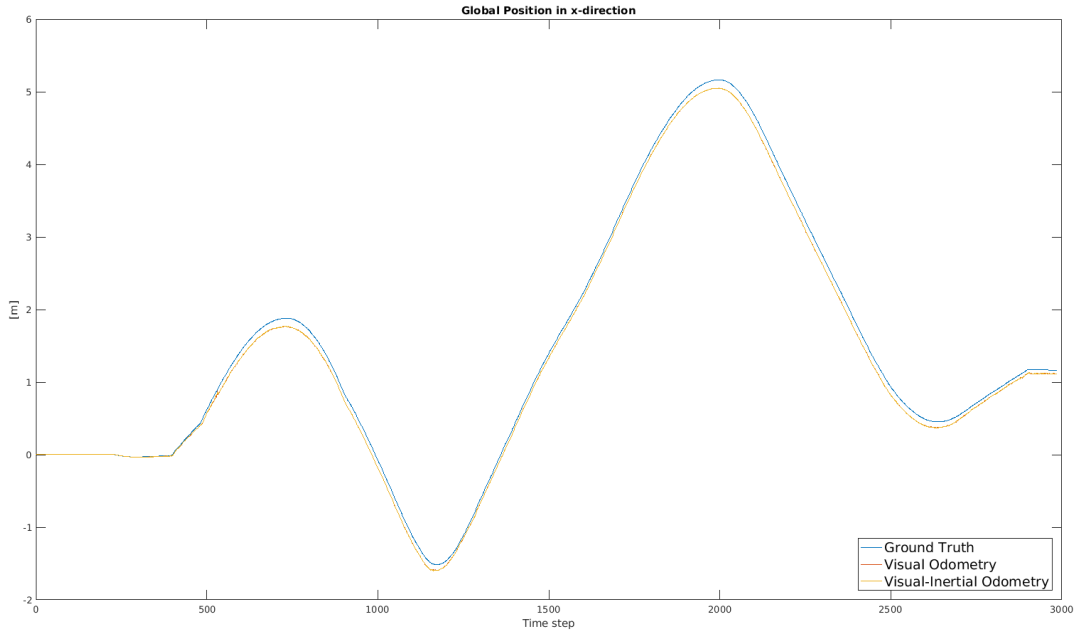

Figure 5.5 Ground truth vs. Visual-inertial position estimation in global x-direction

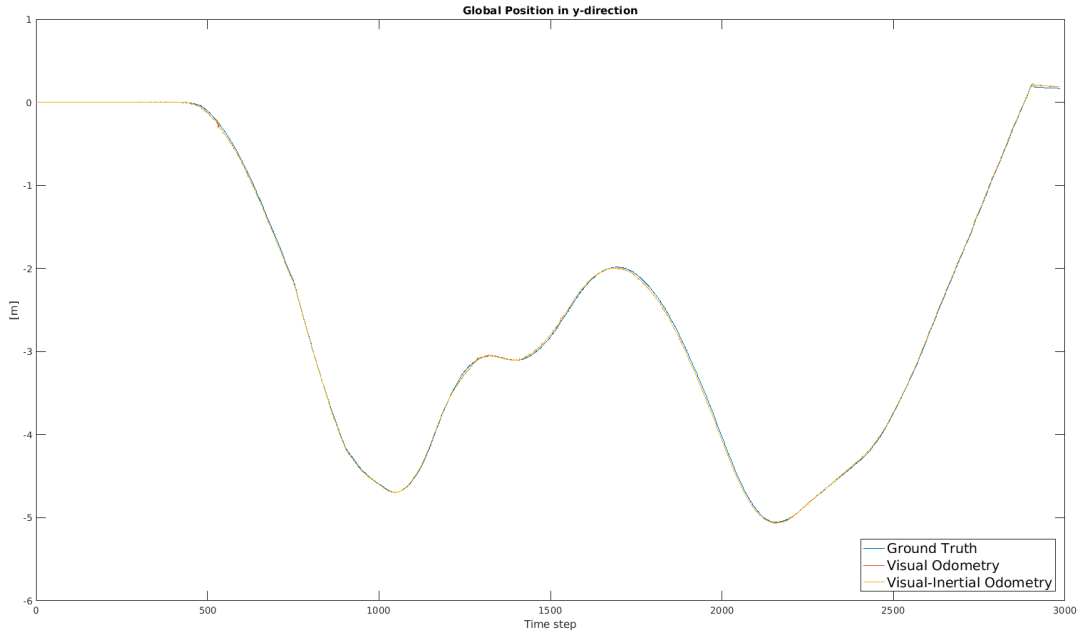

Figure 5.6 Ground truth vs. Visual-inertial position estimation in global y-direction

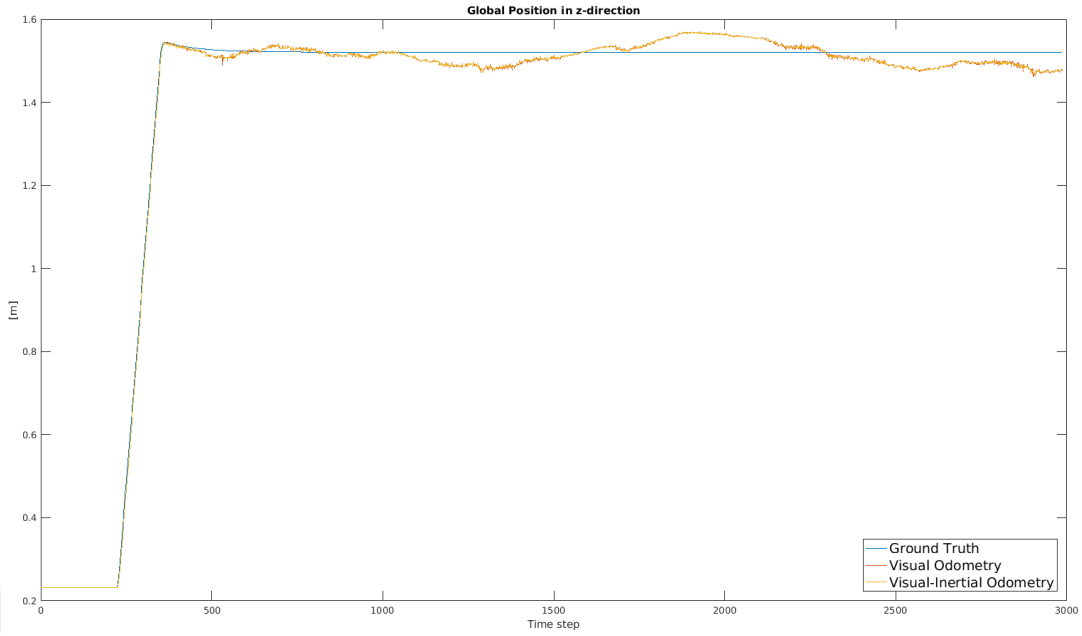

Figure 5.7 Ground truth vs. Visual-inertial position estimation in global z-direction

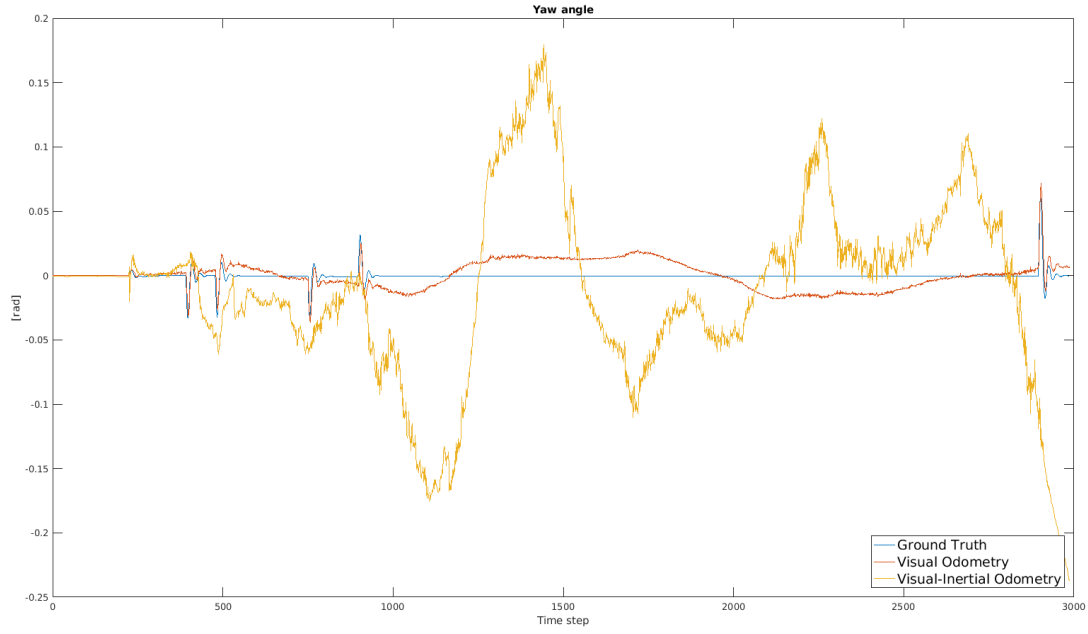

Figure 5.8 Ground truth vs. Visual-inertial yaw angle estimation

5.1.2.1 Discussion

The importance of visual-inertial state estimation for GPS-denied environments is described in the previous chapters. The results obtained in the tests conducted in this context are presented comparing the ground truth values.

The maximum deviations (errors) in state estimates are presented in Table 5.1 in order to comprehend in a clearer way. It can be observed that visual-inertial odometry has superior performance than visual odometry as expected. The values in the table show that the maximum error along the estimated trajectory is under 0.1 m. This value is relatively negligible considering the building scale so, the performance of the state estimation can be evaluated as sufficient in terms of building inspection.

Table 5.1 Maximum errors in state estimations

	Visual Odometry Max. Error	Visual-Inertial State Estimation Max. Error
Global x-direction	0.146157218669 m	0.14588655685 m
Global y-direction	0.0648243906416 m	0.0521918003553 m
Global z-direction	0.0580570380627 m	0.0546632381688 m
Yaw angle	0.237239904855 rad	0.022748689915 rad

According to the table, although the maximum deviations in the x, y and z directions are close to each other, the errors in the yaw angle are slightly dramatic compared to them. This expected behaviour shows the importance of fusing inertial measurements with visual odometry since yaw angle of IMUs are generally prone to have error due to the fact that the gravity measured by accelerometers cannot be used to help to estimate it (Neto, Mendes & Moreira, 2015). Fortunately, visual-inertial sensor fusion improves the yaw estimations as well.

Simulations enable evaluation of SLAM performance of the proposed approach before conducting real-world experiments. It can be verified that the visual-inertial approach is valid and the employed open source software gives sufficient results regarding revisiting operations for building inspection.

5.2 Real-World Test: Indoor Experiments

The second test case is conducted indoor to be able to verify the fully integrated system. After verifying the state estimation and mapping performances in simulations, experiments are performed in a GPS-denied environment for evaluation of the integrated system.

The experiments are conducted in the workshop of Design Factory in Middle East Technical University that can be seen in Figure 5.9. Although the same software architecture with simulations is adopted during indoor experiments, there are additional steps as listed below for physical world experiments in order to satisfy the hardware requirements.

- The first step is to install freenect package as the driver of the RGB-D camera in order to process depth image registration and RGB image rectification. Then, the other nodes that subscribe the visual data can access the processed images.

- The second step is to develop a node that translates the planned velocity commands for the velocity control format of the DJI SDK. The simplified ROS network diagram of the integrated system can be seen in Figure 5.10.

Figure 5.9 Workshop of METU Design Factory where indoor tests are conducted

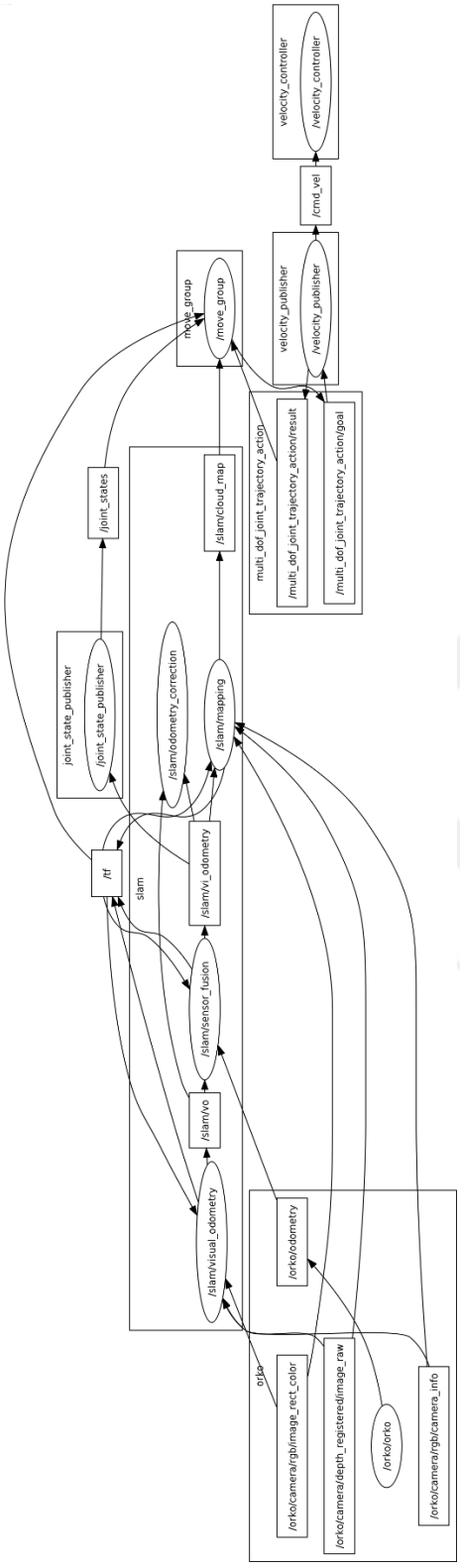

Figure 5.10 Dataflow of the implemented ROS network

The industrial space where the experiments are conducted is appropriate for testing performances of the state estimation and the planning strategies of the system since repetitive elements (i.e. aluminum joineries) are placed. There is also small number of patterns/landmarks on the floor of the space so that the possibility of failure in state estimations is higher. These two complications make the case more compelling in terms of stability and robustness of the SLAM.

5.2.1 Mapping

The first phase of this test case is mapping phase. A low resolution point cloud representation is presented in Figure 5.11. The environment is partially mapped since it is sufficient for the evaluation of the system. For mapping of the environment, the UAV is covered a trajectory (Figure 5.11). The trajectory is determined to explore the part of the environment in which the motions are planned (Figure 5.12). Additionally, the trajectory contains overlapping positions so that loop closures can be detected.

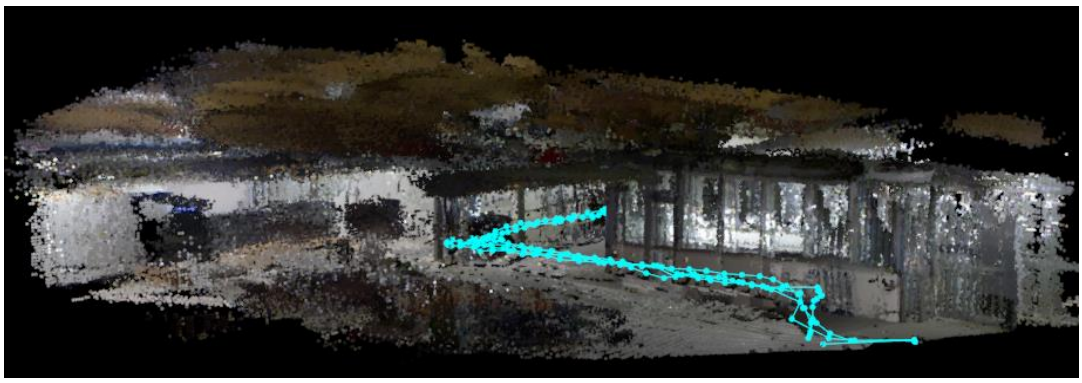

Figure 5.11 The reconstructed map of the test environment

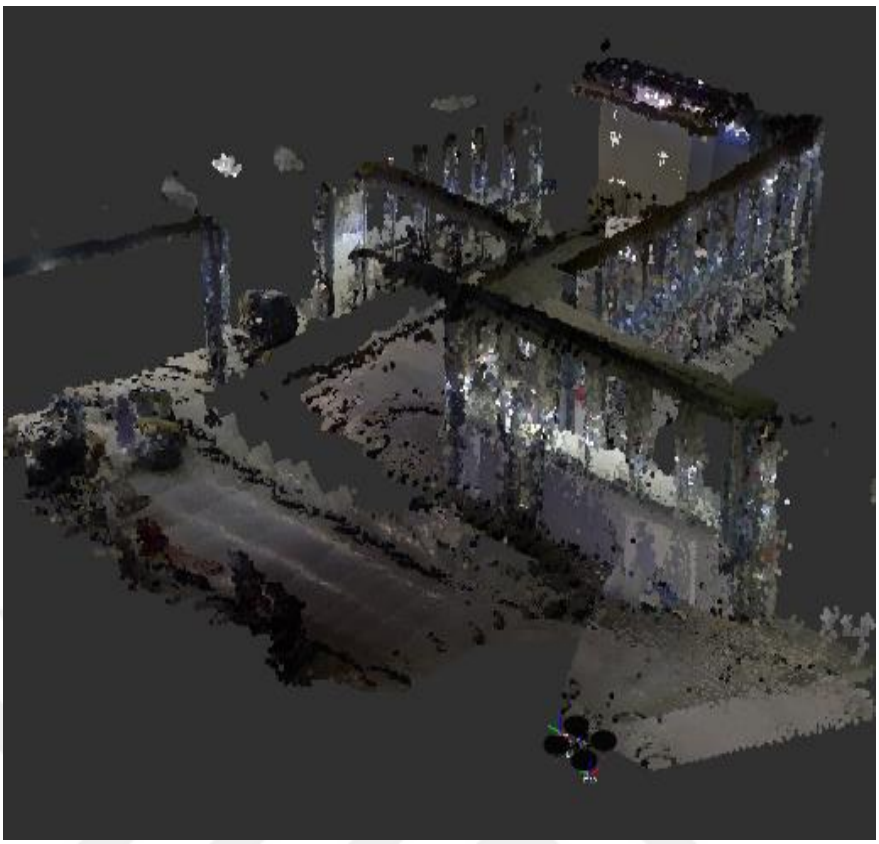

Figure 5.12 The part of the map in which motions are planned

It can be noted that the algorithm matches the images to their corresponding locations at 2 Hz in this test. For further studies, it is recommended to adjust this frequency to available hardware considering the FOV of the camera and flight speed. The frequency should be adjusted so that it ensures any image frame coincides with its adjacent frame when a constant velocity of the UAV is set. A simpler solution might be to not set a frequency in the algorithm. This provides that the system acquires as many images as it can in terms of computational power although this causes a large number of data that should be post processed.

5.2.1.1 Discussion

In real-world conditions, lighting conditions of the environment is critical in terms of visual odometry and mapping. In this context, RGB-D cameras are sensitive to sunlight since they are generally equipped with infrared sensors. It is observed that direct or indirect sunlight might affect the visual data. Stereo or monocular cameras can be considered as alternatives to overcome this issue. However, they demand more computational resource since depth should be computed onboard.

The maps are reconstructed as point clouds and 3D voxel maps (octomap). The mapping resolution is adjustable in the method used but it is highly dependent on computational power. The resolution of the map in the simulations is higher than the one in real-world experiments because the ground station has more powerful than the onboard computer in terms of computation. It is possible to increase the resolution of the maps by offline post-processing the data gathered on a ground station computer.

5.2.2 Crack Detection

After the mapping, the integrated system is tested. The test case is demonstration of a revisiting task. For this purpose, the images acquired during the mapping phase are processed with the image classifier developed to detect cracks on walls of the environment.

The node that is developed for matching the images with their corresponding locations acquires 20 images (Figure 5.13) in mapping phase. 13 of these images are replaced with images that contain cracks (Figure 5.14) since there are no surface cracks available in the test environment.

The registered locations (positions) are kept same as in mapping but only the images are changed. In this way, the developed task planning pipeline as well as the crack detection approach can be tested.

Figure 5.13 Several images acquired during mapping phase of the indoor experiments

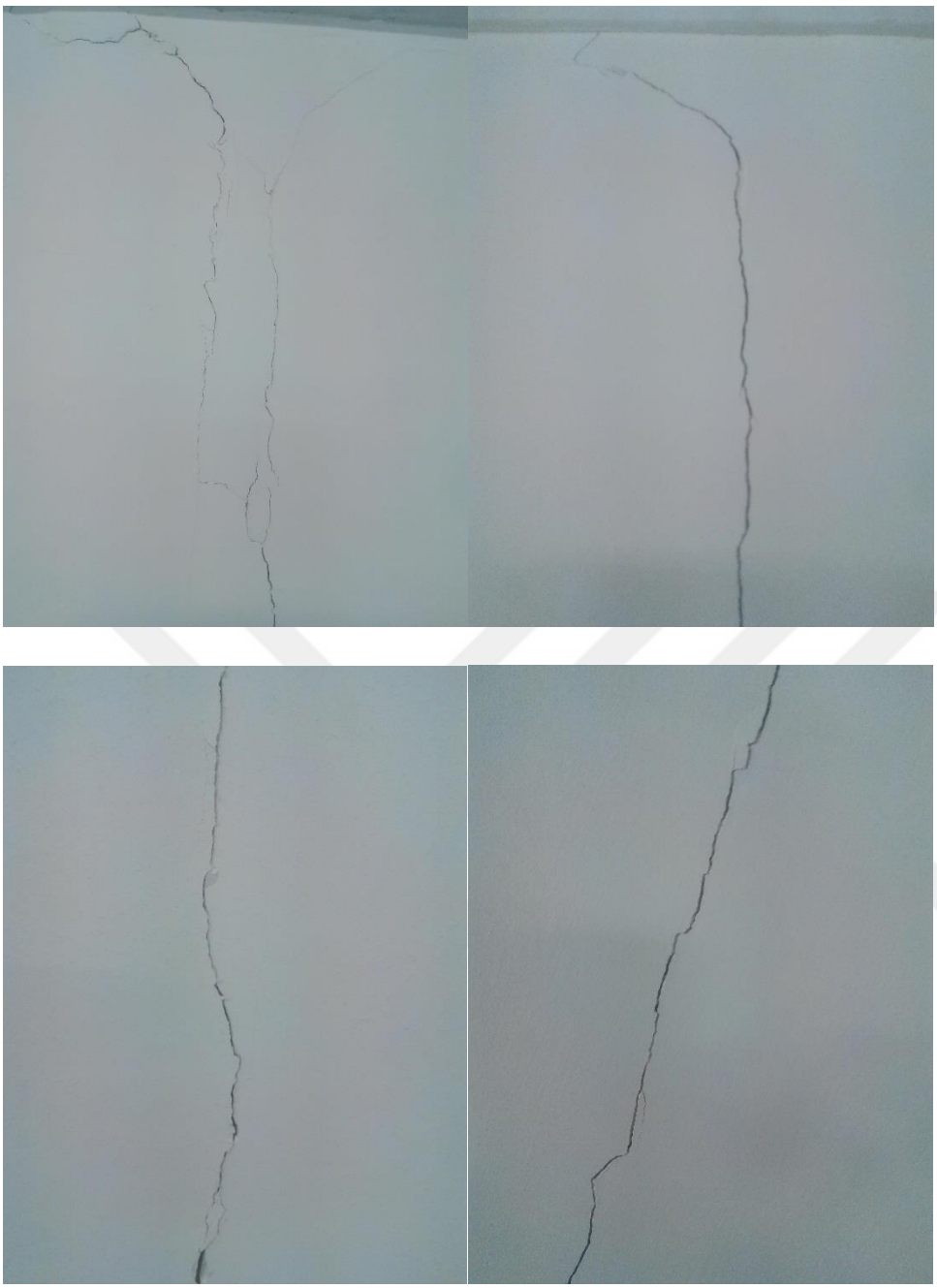

Figure 5.14 Examples of images containing cracks that are replaced with acquired images

Then, these 20 images are fed into the CNN in order to identify the cracks. After processing, the CNN classifies 16 of the images as cracks although the actual cracks exist on 13 of the data. Several other elements such as windows and radiator mislead the CNN as in the extra 3 images which can be seen in Figure 5.15. However, the other 13 images are those which have cracks.

Figure 5.15 The images that are misclassified as containing cracks by the CNN

5.2.3 Motion Planning

After the crack detection, one of these crack images that corresponds to the location shown in Figure 5.16 is selected as the goal position for revisiting. The goal position is selected so that the most complicated motion plan possible in the test environment can be achieved.

Because increasing the number of obstacles and creating narrow passages challenge sampling based motion planning algorithms. Thus, the distance between the start and the goal positions is set to be as away as possible in the map. Moreover, the walls and the windows exist between them as obstacles so that the UAV should takeoff and move around the junction of the two walls for obstacle avoidance during motion.

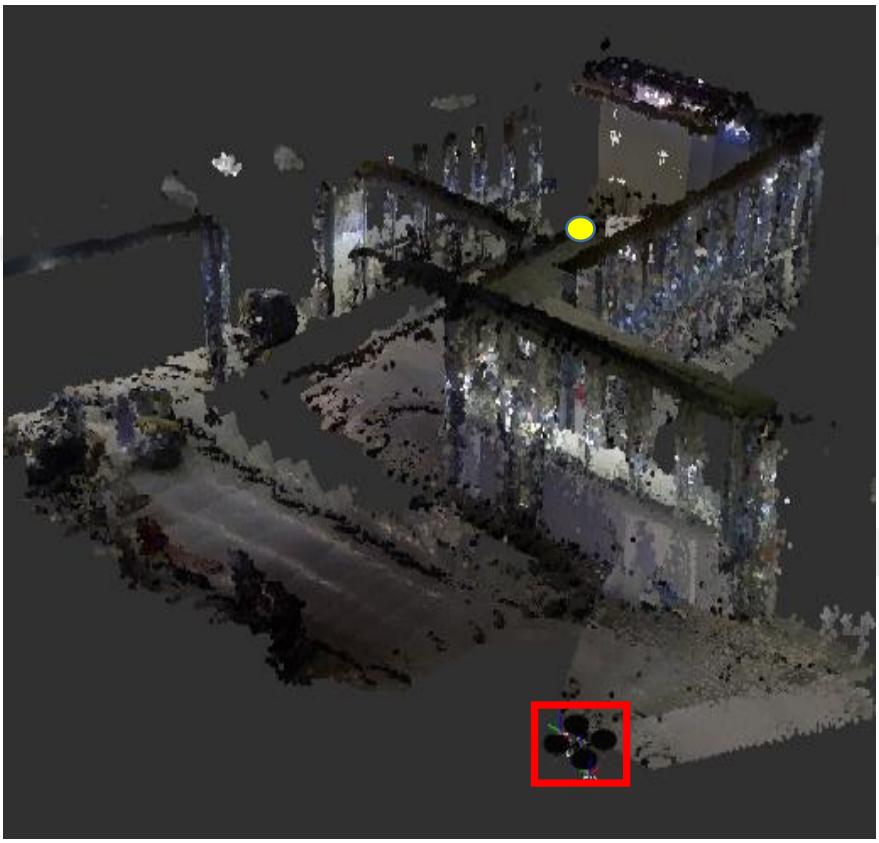

Figure 5.16 Start and goal positions of the motion planning problem. (The red rectangle encapsulates the start position while the yellow circle indicates the goal)

Once the goal location is determined by the GUI, it is necessary to plan the path and the motion. The criteria for motion planning can be listed as:

- A complete path including the start and the goal positions,
- An obstacle-free path,
- An optimal path in terms of length,
- A path computed within the specified time (20 sec.),
- A plan considering the motion constraints of the UAV (i.e. not exceeding roll and pitch limits that may cause overturn and crash)

Considering these criteria, several path planning algorithms available in MoveIt! are tested for the motion planning problem between start and goal positions. PRM*, RRT and RRT* algorithms compute solutions while EST, SPL, LBKPIECE, PRM, BKPIECE algorithms are not able to solve the task.

The solution of the PRM* algorithm (Figure 5.17) is not acceptable in terms of both the optimality and the motion constraints since it requires large roll degrees in the motion that may cause overturn. The trajectory planned by RRT (Figure 5.18) has a sudden jump in the motion which is not possible for the UAV to execute. On the other hand, RRT* (Figure 5.19) computes a trajectory that satisfies the criteria. The trajectory is collision-free and smooth as well as optimal in terms of length. Therefore, the motion planner is set to use RRT* as the main algorithm in the plans.

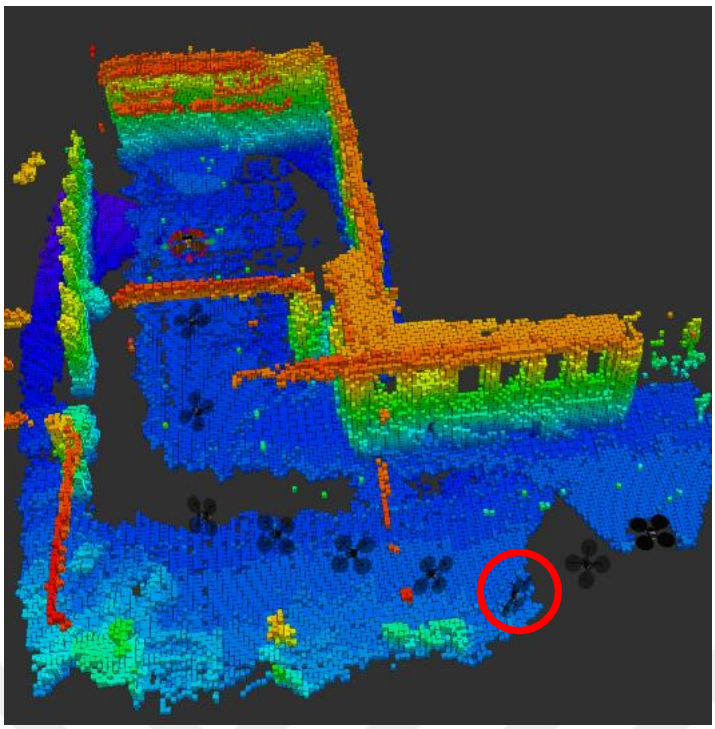

Figure 5.17 Path planned by PRM*. (Red circle shows the large roll angle that may cause overturn)

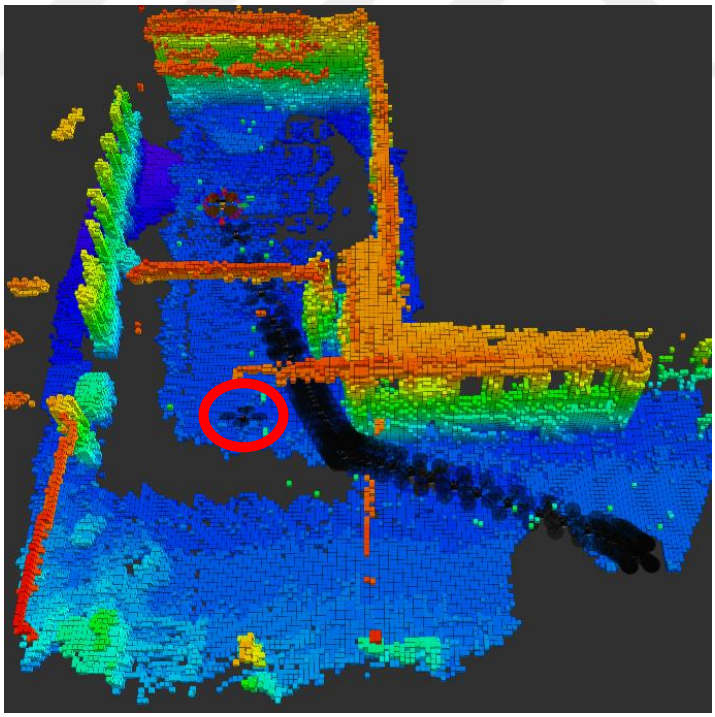

Figure 5.18 Path planned by RRT. (The sudden jump is shown by a red circle)

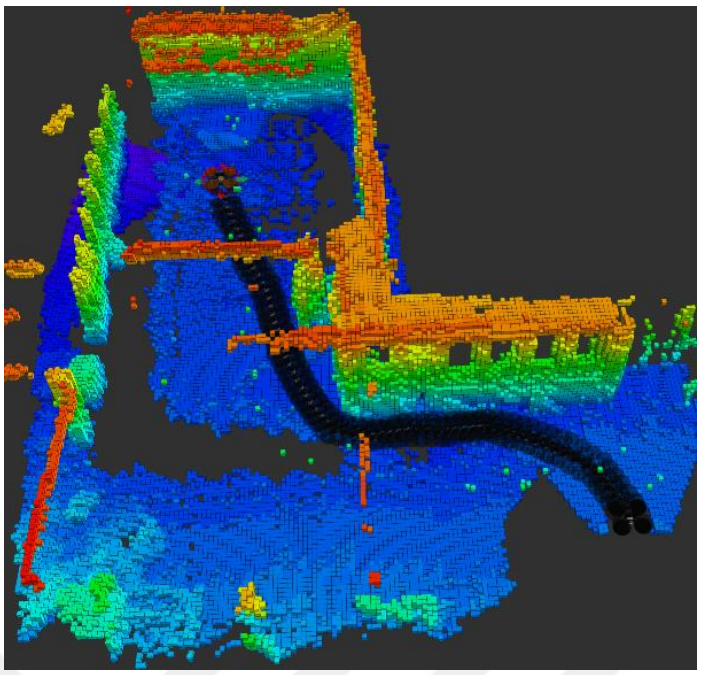

Figure 5.19 Path planned by RRT* (The path is collision-free, smooth and optimal in length)

After the trajectory has been calculated according to the criteria, the UAV has taken action to follow the waypoints and the task can be executed by the end of this operation.

5.2.3.1 Discussion

This section presents the analysis of the motion planning performances of the system. It is found out that fusing the measurements from ultrasonic sensor for height estimation (in z-direction) gives better results than the respective data of visual odometry. However, this might lead significant errors for relatively high altitudes since ultrasonic sensors have range limits (i.e. 20 m for this case).

In the motion plans, sampling based path planning algorithms are evaluated. Optimality objective in motion planning is determined as the path length so that the planner should find the obstacle-free shortest path available. Among the tested planning algorithms, only RRT* gives an acceptable solution considering geometric constraints and the optimality objective as Tonioni (2013) claimed. Thus, RRT* is selected as the motion planning algorithm for further applications.

The revisiting accuracy is strongly dependent on the hardware since a commercial onboard flight controller is used in the present work. Therefore, performance analysis of the revisiting accuracy could not be conducted due to the hardware limitations in this study.

Consequently, it can be stated that autonomous navigation and planning performance of the UAV has promising results considering building inspection operations.

CHAPTER VI

CONCLUSION

In this study, autonomous navigation and planning of unmanned aerial vehicles are investigated in the pursuit of automation of building inspection/monitoring operations. An integrated system is designed in order to address the further requirements of inspection missions by UAVs. While the robotics community has mostly focused on exploration (mapping) of the buildings, this work puts effort on revisiting a damaged location. For this purpose, surface cracks are identified as structural defects to be detected since crack detection is one of the common objectives during building inspection.

In this context, the related work is overviewed and background of the methodology is presented. Then, the materials and methods used in the implementation is described in detail. In the implementation, a commercial quadrotor platform is equipped with onboard sensors and computers. Visual-inertial sensor fusion approach is adopted for state estimations. Onboard RGB-D camera, IMU and ultrasonic sensor are utilized in SLAM processes. Although there are numerous sensors (LIDAR, sonar etc.) used by the robotics community, visual cameras are preferred in this study since they are also utilized for visual inspection. All the computations except high-level task planning are achieved by onboard computer.

Furthermore, a decision-making tool for task planning is developed in order to address revisiting locations of interest after exploration of the environment. It is built on top of autonomous navigation and motion planning ability of the UAV. It can be claimed that this approach should increase the efficiency of task planning in building inspections considering crack detection. A GUI is developed in order to wrap the functionality of the system. The GUI presents high-level commands

After the system integration, validation and verification of the system is tested through simulations and real-world experiments. The results of mapping, state estimation and planning are evaluated in the previous chapter. The overall system demonstrates promising performance regarding its application of building inspection processes. It is important to remark that the maps constructed in the experiments are for autonomous navigation; so, the accuracy and resolution are adequate for localization and planning. More accurate maps can be reconstructed for applications that demand high-quality visuals. Although the performance of the crack detection approach is sufficient, it is highly dependent on training datasets. Therefore, larger and diverse datasets can increase reliability.

Consequently, this work contributes to the full automation of building inspection and achieves an integrated system that offers a decision making tool integrated to autonomously navigating UAVs.

In future, this study can be extended for a multi-agent system that achieves the missions in a more efficient way. The presented task planning strategy can also be extended to identify different types of defects. The Graphical User Interface can be enhanced to visualize defected locations on a 3D model leading to a more user-friendly interface.

REFERENCES

Andre, M., & Simoes, O. (2009). *Development of an Aerial Robot for Inspection and Surveillance* (Master's Thesis, Universidade de Aveiro), pp. 1–82. Retrieved from papers2://publication/uuid/5AAF07A5-72AA-448B-8554-150BB10DD804

Angeli, A., Filliat, D., Doncieux, S., & Meyer, J. A. (2008). Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, *24*(5), 1027-1037.

Araar, O., Aouf, N., & Dietz, J. L. V. (2015). Power pylon detection and monocular depth estimation from inspection UAVs. *Industrial Robot: An International Journal*, *42*(3), 200–213. https://doi.org/Doi 10.1108/Ir-11-2014-0419

Bircher, A., Alexis, K., Burri, M., Oettershagen, P., Omari, S., Mantel, T., & Siegwart, R. (2015). Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (pp. 6423-6430). IEEE.

Blasco, P. I. (2012). *ROS distributed architecture*. Retrieved from https://www.slideshare.net/pibgeus/21-distributed-architecture-deploymentinstrospection

Bloesch, M., Omari, S., Fankhauser, P., Sommer, H., Gehring, C., Hwangbo, J., ... & Siegwart, R. (2014). Fusion of optical flow and inertial measurements for robust egomotion estimation. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on* (pp. 3102-3107). IEEE. https://doi.org/10.1109/IROS.2014.6942991

Bloesch, M., Omari, S., Hutter, M., & Siegwart, R. (2015). Robust visual inertial odometry using a direct EKF-based approach. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (pp. 298-304). IEEE. https://doi.org/10.1109/IROS.2015.7353389

Botterill, T., Mills, S., & Green, R. (2011). Bag- of- words- driven, single-camera simultaneous localization and mapping. *Journal of Field Robotics*, *28*(2), 204-226.

Bouvrie, B. (2011). Improving RGBD Indoor Mapping with IMU data, 66. Retrieved from http://www.es.ewi.tudelft.nl/msc-theses/2011-DesBouvrie.pdf

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … & Leonard, J. J. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, *32*(6), 1309–1332.

Canziani, A., Paszke, A., & Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications, 1–7. Retrieved from http://arxiv.org/abs/1605.07678

Cha, Y.-J., Choi, W., & Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. Computer-Aided Civil and Infrastructure Engineering, 32(5), 361–378. https://doi.org/10.1111/mice.12263

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

DJI (2014). *Matrice 100*. Shenzhen, China. Retrieved from https://www.dji.com/matrice100/info#specs

DJI (2015). *Manifold.* Shenzhen, China. Retrieved from http://www.dji.com/manifold/info#specs

DJI (2017). *Onboard SDK*. Shenzhen, China. Retrieved from https://developer.dji.com/onboard-sdk/

Elmenreich, W. (2002). An introduction to sensor fusion. *Vienna University of Technology, Austria*. Retrieved from http://www.vmars.tuwien.ac.at/documents/intern/805/elmenreich_sensorfusionintro.pdf

Eschmann, C., Kuo, C. M., Kuo, C. H., & Boller, C. (2013). High-resolution multisensor infrastructure inspection with unmanned aircraft systems. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, (2), 125-129. https://doi.org/10.5194/isprsarchives-XL-1-W2-125-2013

Fang, Z., & Zhang, Y. (2015). Experimental Evaluation of RGB-D Visual Odometry Methods. *International Journal of Advanced Robotic Systems*, *12*(3), 26. https://doi.org/10.5772/59991

Fischer, R. B. (2004, January 15). Kalman Filter [Digital image]. Retrieved from http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/WELCH/kalman-146.gif

Gopalakrishnan, K., Khaitan, S. K., Choudhary, A., & Agrawal, A. (2017). Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection. Construction and Building Materials, 157, 322–330. https://doi.org/10.1016/j.conbuildmat.2017.09.110

GENERAL ATOMICS. (2007). *MQ-9B RPA.* Retrieved from http://www.ga-asi.com/mq-9b

Google Brain Team. (2015). *TensorFlow*. Retrieved from https://www.tensorflow.org

Guo, C. X., & Roumeliotis, S. I. (2013, November). IMU-RGBD camera navigation using point and plane features. In *Intelligent Robots and Systems*

(IROS), 2013 IEEE/RSJ International Conference on (pp. 3164-3171). IEEE. https://doi.org/10.1109/IROS.2013.6696806

Heng, L., Lee, G. H., Fraundorfer, F., & Pollefeys, M. (2011). Real-time photo-realistic 3d mapping for micro aerial vehicles. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* (pp. 4012-4019). IEEE. https://doi.org/10.1109/IROS.2011.6048818

Høglund, S. (2014). *Autonomous inspection of wind turbines and buildings using an UAV* (Master's thesis, Institutt for teknisk kybernetikk). https://doi.org/http://hdl.handle.net/11250/261286

Howard, A., Koenig, N., & Open Source Robotics Foundation. (2013). *Gazebo*. Retrieved from http://gazebosim.org/

IMAGINE DRONE. (n.d.). *EMERGENCIAS Y RESCATE*. Retrieved from http://imaginedrone.es/emergencias-y-rescate/

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of basic Engineering*, *82*(1), 35-45. https://doi.org/10.1115/1.3662552

Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566-580.

Keras Applications. (n.d.). Retrieved from https://keras.io/applications/#usage-examples-for-image-classification-models

Kivy Organization. (2011). *Kivy*. Retrieved from https://kivy.org

Konolige, K., Bowman, J., Chen, J. D., Mihelich, P., Calonder, M., Lepetit, V., & Fua, P. (2010). View-based maps. *The International Journal of Robotics Research*, *29*(8), 941-957.

Labbé, M. (2013). *Rtabmap_ros*. Retrieved from http://wiki.ros.org/rtabmap_ros

Labbé, M. (n.d.). *RTAB-Map*. Retrieved from http://introlab.github.io/rtabmap/

Labbe, M., & Michaud, F. (2013). Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, *29*(3), 734-745.

Labbe, M. (2014). Kinect odometry comparison: SIFT, SURF, FAST/BRIEF and ICP Retrieved from https://www.youtube.com/watch?v=3xggFmS5-1s

Lim, R. S., La, H. M., & Sheng, W. (2014). A robotic crack inspection and mapping system for bridge deck maintenance. *IEEE Transactions on Automation Science and Engineering*, *11*(2), 367–378.

Liu, C., Zhang, S., Wu, H., & Don, R. (2014). A Flying Robot Localization Method Based on Multi-sensor Fusion. *International Journal of Advanced Robotic Systems*, 1. https://doi.org/10.5772/58927

Loianno, G., Watterson, M., & Kumar, V. (2016). Visual inertial odometry for quadrotors on SE(3). In *Robotics and Automation (ICRA), 2016 IEEE International Conference on* (pp. 1544-1551). IEEE. https://doi.org/10.1109/ICRA.2016.7487292

Máthé, K., & Buşoniu, L. (2015). Vision and Control for UAVs: A Survey of General Methods and of Inexpensive Platforms for Infrastructure Inspection. *Sensors*, *15*(7), 14887–14916. https://doi.org/10.3390/s150714887

MAXED-OUT TEAM. (2014). 3D Cloud [Digital image]. Retrieved from https://github.com/introlab/rtabmap/wiki/IROS-2014-Kinect-Challenge

Metni, N., & Hamel, T. (2007). A UAV for bridge inspection: Visual servoing control law with orientation limits. *Automation in Construction*, *17*(1), 3–10. https://doi.org/10.1016/j.autcon.2006.12.010

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., & Von Stryk, O. (2012). Comprehensive simulation of quadrotor uavs using ros and gazebo. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots* (pp. 400-411). Springer, Berlin, Heidelberg.

Microsoft (n.d.). *Kinect for Windows Sensor Components and Specifications* [Digital image]. Retrieved from https://msdn.microsoft.com/en-us/library/jj131033.aspx

Moll, M., Sucan, I. A., & Kavraki, L. E. (2015). Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization. *IEEE Robotics and Automation Magazine*, *22*(3), 96–102.

Moore, T. (2013). *Robot_localization*. Retrieved from http://wiki.ros.org/robot_localization

Moranduzzo, T., & Melgani, F. (2014). Monitoring Structural Damages in Big Industrial Plants With UAV Images, 4950–4953. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6947606

Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, *31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

Neto, P., Mendes, N., & Moreira, A. P. (2015). Kalman Filter-Based Yaw Angle Estimation by Fusing Inertial and Magnetic Sensing. In Lecture Notes in Electrical Engineering (Vol. 321 LNEE, pp. 679–688). https://doi.org/10.1007/978-3-319-10380-8_65

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., ... & Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on* (pp. 127-136). IEEE. https://doi.org/10.1109/ISMAR.2011.6092378

Nikolic, J., Burri, M., Rehder, J., Leutenegger, S., Huerzeler, C., & Siegwart, R. (2013, March). A UAV system for inspection of industrial facilities.

In *Aerospace Conference, 2013 IEEE* (pp. 1-8). IEEE. https://doi.org/10.1109/AERO.2013.6496959

Nister, D., & Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on* (Vol. 2, pp. 2161-2168). IEEE.

Oleynikova, H., Burri, M., Lynen, S., & Siegwart, R. (2015, September). Real-time visual-inertial localization for aerial and ground robots. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (pp. 3079-3085). IEEE. https://doi.org/10.1109/IROS.2015.7353802

Omari, S., Gohl, P., Burri, M., Achtelik, M., & Siegwart, R. (2014). Visual industrial inspection using aerial robots. In *Applied Robotics for the Power Industry (CARPI), 2014 3rd International Conference on* (pp. 1-5). IEEE.

Open Source Robotics Foundation (2007). *ROS.* Willow Garage, Menlo Park, California, USA. Retrieved from http://www.ros.org/about-ros/

Özaslan, T., Shen, S., Mulgaonkar, Y., Michael, N., & Kumar, V. (2015). Inspection of penstocks and featureless tunnel-like environments using micro UAVs. In *Field and Service Robotics* (pp. 123-136). Springer International Publishing.

Pauly, L., Peel, H., Luo, S., Hogg, D., & Fuentes, R. (2017). Deeper Networks for Pavement Crack Detection. In Proceedings of the 34th ISARC. 34th International Symposium in Automation and Robotics in Construction (pp. 479–485). https://doi.org/10.22260/ISARC2017/0066

Physical and Biological Computing Group. (n.d.). *Planner Arena*. Retrieved from http://plannerarena.org

Project ONEIROS. (2014). *Keras*. Retrieved from https://keras.io/

Rathinam, S., Kim, Z. W., & Sengupta, R. (2008). Vision-Based Monitoring of Locally Linear Structures Using an Unmanned Aerial Vehicle1. *Journal of Infrastructure Systems*, *14*(1), 52–63. https://doi.org/10.1061/(ASCE)1076-0342(2008)14:1(52)

Robotic Research Team (n.d.). *3D-Mapping* [Digital Image]. Retrieved from http://rrt.fh-wels.at/sites/robocup/mapping.html

Scaramuzza, D. (n.d.). *VO Flow Chart* [Digital image]. Retrieved from http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf

Serrano, N. E. (2011). *Autonomous quadrotor unmanned aerial vehicle for culvert inspection* (Doctoral dissertation, Massachusetts Institute of Technology).

Shen, S., Mulgaonkar, Y., Michael, N., & Kumar, V. (2013). Vision-based state estimation for autonomous rotorcraft MAVs in complex environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (pp. 1758-1764). IEEE. https://doi.org/10.1109/ICRA.2013.6630808

Sivic, J., & Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision* (Vol. *2*, pp. 1470-1478). IEEE.

Stachniss, C. (2006). Ubremen-cartesium [Digital image]. Retrieved from http://cres.usc.edu/radishrepository/view-one.php?name=ubremen-cartesium

Sucan, I. A. & Chitta, S. (2011). *MoveIt!* Retrieved from http://moveit.ros.org

Sucan, I. A., Moll, M., & Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82.

Sucan, I., & Chitta, A. (n.d.). Planning Scene [Digital image]. Retrieved from http://moveit.ros.org/documentation/concepts/

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 2818–2826). IEEE.

Tapus, A., & Siegwart, R. (2008). Topological SLAM. In *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems* (pp. 99-127). Springer Berlin Heidelberg.

Tonioni, A (2013). *Study and implementation of algorithms for 3d reconstruction of the environment and the navigation of autonomous aircraft* (Master's thesis, Alma Mater Studiorum - Computer Science Engineering Faculty - University of Bologne).

Toth, C. D., O'Rourke, J., & Goodman, J. E. (Eds.). (1997). *Handbook of discrete and computational geometry*. CRC press. Boca Raton, FL, USA.

TRNDLabs. (2014). *SKEYE Nano 2*. Retrieved from https://www.trndlabs.com/product/skeye-nano-2/

Varga, D. (2016). *Keras Finetuning*. Retrieved from https://github.com/danielvarga/keras-finetuning

Viehmeister-Kerner, A. (n.d.). *Logistics and Transport*. Retrieved from https://www.multirotor.net/en/applications/logistics-and-transport

Weiss, S. M. (2012). Vision based navigation for micro helicopters (Doctoral dissertation, Eidgenössische Technische Hochschule (ETH) Zurich). https://doi.org/10.3929/ethz-a-007344020

Windows (2010.). *Kinect v1*. Redmond, Washington, USA. Retrieved from https://developer.microsoft.com/en-us/windows/kinect/hardware

Winkvist, S., Rushforth, E., & Young, K. (2013). Towards an autonomous indoor aerial inspection vehicle. *Industrial Robot: An International Journal*, *40*(3), 196-207.

Yang, L., Qi, J., Xiao, J., & Yong, X. (2014). A literature review of UAV 3D path planning. In *Intelligent Control and Automation (WCICA), 2014 11th World*

Congress on (pp. 2376-2381). IEEE. https://doi.org/10.1109/WCICA.2014.7053093

Zhou, G., Fang, L., Tang, K., Zhang, H., Wang, K., & Yang, K. (2015). Guidance: A Visual Sensing Platform for Robotic Applications. *CVPR2015 Workshop*. Retrieved from http://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W12/papers/Zhou_Guidance_A_Visual_2015_CVPR_paper.pdf

Appendix A: The Velocity Controller Algorithm

```
import rospy, time
from dji_sdk.srv import AttitudeControl, SDKPermissionControl, DroneArmControl, DroneTaskControl
from geometry_msgs.msg import TwistStamped

class FlightController:
        def __init__(self):
                self.srv = rospy.ServiceProxy('/dji_sdk/attitude_control', AttitudeControl, persistent=True)
                self.vel_sub = rospy.Subscriber('/command/twist', TwistStamped, self.callback)

        def callback(self, msg):
                vx = msg.twist.linear.x
                vy = msg.twist.linear.y
                vz = msg.twist.linear.z
                yaw_rate = msg.twist.angular.z
                if self.srv(0x40|0x00|0x08|0x00|0x01, vx, vy, vz, yaw_rate):
                        time.sleep(0.02)

if __name__ == '__main__':
        rospy.init_node('flight_controller')
        try:
                rospy.wait_for_service('/dji_sdk/sdk_permission_control')
                sdk_srv = rospy.ServiceProxy('/dji_sdk/sdk_permission_control', SDKPermissionControl)
                sdk_srv(1)
        except rospy.ServiceException, e:
                rospy.logerr("Service call failed: %s"%e)
        try:
                rospy.wait_for_service('/dji_sdk/drone_task_control')
                task_srv = rospy.ServiceProxy('/dji_sdk/drone_task_control', DroneTaskControl)
        except rospy.ServiceException, e:
                rospy.logerr("Service call failed: %s"%e)
        try:
                rospy.wait_for_service('/dji_sdk/attitude_control')
                fc = FlightController()
                rospy.spin()
        except rospy.ServiceException, e:
                rospy.logerr("Service call failed: %s"%e)
        finally:
                #landing and release control
                task_srv(task=6)
                sdk_srv(0)
```

Appendix B: The Revisit Motion Planning Algorithm

```
#!/usr/bin/env python

import sys, copy
import rospy
import moveit_commander
from tf import transformations as trf
from geometry_msgs.msg import TwistStamped

class RevisitPlanner:
	moveit_commander.roscpp_initialize(sys.argv)

	def __init__(self, matchings):
		self.matchings = matchings
		self.cmd = TwistStamped()
		self.cmd.header.frame_id = "world"
		self.vel_pub = rospy.Publisher('/command/twist', TwistStamped, queue_size=100)

	def takeoff(self):
		self.cmd.header.stamp = rospy.Time.now()
		self.cmd.twist.linear.x = self.cmd.twist.linear.y = self.cmd.twist.angular.x = self.cmd.twist.angular.y = self.cmd.twist.angular.z = 0
		self.cmd.twist.linear.z = 0.1
		self.vel_pub.publish(self.cmd)
		rospy.sleep(0.1)
		rospy.loginfo("Take-off")
		self.stop()

	def stop(self):
		self.cmd.header.stamp = rospy.Time.now()
		self.cmd.twist.linear.x = self.cmd.twist.linear.y = self.cmd.twist.linear.z = 0
		self.cmd.twist.angular.x = self.cmd.twist.angular.y = self.cmd.twist.angular.z = 0
		self.vel_pub.publish(self.cmd)

	def publish_translation(self, point, previous_point, dt):
		self.cmd.header.stamp = rospy.Time.now()
		self.cmd.twist.linear.x = (point.translation.x-previous_point.translation.x)/dt
		self.cmd.twist.linear.y = (point.translation.y-previous_point.translation.y)/dt
		self.cmd.twist.linear.z = (point.translation.z-previous_point.translation.z)/dt
		self.cmd.twist.angular.x = self.cmd.twist.angular.y = self.cmd.twist.angular.z = 0
		self.vel_pub.publish(self.cmd)
		rospy.sleep(dt)

	def publish_rotation(self, rotation_angle, dt):
		self.cmd.header.stamp = rospy.Time.now()
		self.cmd.twist.linear.x = self.cmd.twist.linear.y = self.cmd.twist.linear.z = 0
		self.cmd.twist.angular.x = self.cmd.twist.angular.y = 0
		self.cmd.twist.angular.z = rotation_angle/dt
		self.vel_pub.publish(self.cmd)
		rospy.sleep(dt)

	def quat_to_yaw(self, quat):
		quat = (quat.x, quat.y, quat.z, quat.w)
		euler = trf.euler_from_quaternion(quat)
		return euler[2]

	def specify_goal(self, goal_id):
		for i in range(len(self.matchings)):
			if goal_id == self.matchings[i][0]:
				goal = self.matchings[i]
				self.motion_planner(goal)
				return True
		else:
			rospy.loginfo("Image selected could not be found :(")
```

```
def motion_planner(self, goal):
        group = moveit_commander.PlanningSceneInterface()
        group = moveit_commander.MoveGroupCommander("quadrotor")
        group.set_planner_id("RRTstarkConfigDefault")
        group.allow_replanning(True)
        group.allow_looking(False)
        group.set_num_planning_attempts(10)
        group.set_start_state_to_current_state()
        group.set_planning_time(10.0)
        group.set_workspace([-7.5, -10, -0.5, 7.5, 3, 8])
        desired_pose = group.get_current_joint_values()
        desired_pose[0] = goal[1]
        desired_pose[1] = goal[2]
        desired_pose[2] = goal[3]
        desired_pose[3] = goal[4]
        desired_pose[4] = goal[5]
        desired_pose[5] = goal[6]
        desired_pose[6] = goal[7]
        group.set_joint_value_target(desired_pose)
        plan = group.plan()
        points = plan.multi_dof_joint_trajectory.points
        for i in range(len(points)):
                if i is not 0:
                        previous_time = points[i-1].time_from_start.to_sec()
                        current_time = points[i].time_from_start.to_sec()
                        dt = current_time-previous_time
                        self.publish_translation(points[i].transforms[0], points[i-1].transforms[0], dt)
                self.stop()
        rotation_angle = self.quat_to_yaw(points[len(points)-1].transforms[0].rotation) - self.quat_to_yaw(points[0].transforms[0].rotation)
        dt = points[len(points)-1].time_from_start.to_sec() - points[0].time_from_start.to_sec()
        self.publish_rotation(rotation_angle, dt)
        self.stop()
```

Appendix C: The Graphical User Interface Algorithm

```
#!/usr/bin/env python

import subprocess
subprocess.Popen('roscore')

import os
import rospy
import roslaunch
from revisit_planner import RevisitPlanner

import kivy
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.popup import Popup
from kivy.uix.progressbar import ProgressBar
from kivy.base import EventLoop

global directory, img_path, matchings
def image_locater():
    directory = os.path.dirname(os.path.realpath(__file__))
    img_path = directory + '/images/'

    try:
        if os.listdir(directory + '/images/predicted')[0].endswith('.png') or os.listdir(directory + '/images/predicted')[1].endswith('.png'):
            img_path = directory + '/images/predicted/'
    except:
        pass

    with open(img_path + 'image_pose_matchings.txt') as f:
        matchings = []
        for line in f:
            line = line.split()
            if line:
                line = [float(i) for i in line]
                matchings.append(line)
    return directory, img_path, matchings

directory, img_path, matchings = image_locater()

class Mapping(BoxLayout):
    def __init__(self, *args, **kwargs):
        super(Mapping, self).__init__(*args, **kwargs)
        self.matching_node = roslaunch.core.Node('orko', 'image_pose_matching.py')
        self.launch = roslaunch.scriptapi.ROSLaunch()
        self.launch.start()
        self.process = None

    def mapping_mode(self, bool):
        try:
            pass
        except BaseException:
            rospy.logwarn('Mapping warning')

    def match_image_to_pose(self, bool):
        try:
            if bool:
                self.process = self.launch.launch(self.matching_node)
            else:
                self.process.stop()
        except BaseException:
            pass
```

```
class Revisiting(BoxLayout):
    imagenum = len(matchings)
    img_path = img_path

    def __init__(self, *args, **kwargs):
        super(Revisiting, self).__init__(*args, **kwargs)
        self.rp = RevisitPlanner(matchings)
        self.octomap_node = roslaunch.core.Node('orko', 'octomap_loader.py')
        self.octomap_launch = roslaunch.scriptapi.ROSLaunch()
        self.octomap_launch.start()
        self.octomap_process = None
        self.progress = 0.0
        self.progress_bar = ProgressBar(max=1.0, value=self.progress)

    def select_goal(self, image_id):
        try:
            self.rp.specify_goal(image_id)

        except BaseException:
            rospy.logwarn('Mission cannot be executed')

    def stop_robot(self):
        self.rp.stop()

    def load_octomap(self):
        pass

    def predict_cracks(self):
        from crack_detection import test
        test(model_prefix='brick', test_path=directory+'/images')
        directory, img_path, matchings = image_locater()

class MainMenu(TabbedPanel):
    def __init__(self, *args, **kwargs):
        super(MainMenu, self).__init__(*args, **kwargs)
        self.uuid = roslaunch.rlutil.get_or_generate_uuid(None, False)
        roslaunch.configure_logging(self.uuid)
        self.launch = roslaunch.parent.ROSLaunchParent(self.uuid,
            ["~/catkin_ws/src/orko/launch/orko.launch"])
        self.streaming_node = roslaunch.core.Node('rqt_image_view', 'rqt_image_view',
            args='/camera/rgb/image_raw', name='video_stream')
        self.streaming_launch = roslaunch.scriptapi.ROSLaunch()
        self.streaming_launch.start()
        self.streaming_process = None

    def launch_ros(self, bool):
        try:
            if bool:
                self.launch.start()
            else:
                self.launch.shutdown()
        except BaseException:
            rospy.logwarn('The launch file cannot start')

    def live_stream(self, bool):
        try:
            if bool:
                self.streaming_process = self.streaming_launch.launch(self.streaming_node)
            else:
                self.streaming_process.stop()
        except BaseException:
            rospy.logwarn('Live stream cannot start')

class PlannerInterface(BoxLayout):
    pass

class PlannerApp(App):
    def build(self):
        EventLoop.ensure_window()
        return PlannerInterface()

if __name__ == "__main__":
    try:
        rospy.init_node('gui')
        PlannerApp().run()
    finally:
```

Appendix D: The Image-Location Matching Algorithm

```
#!/usr/bin/env python

import sys
import rospy
import cv2
import message_filters
from cv_bridge import CvBridge
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Image
from geometry_msgs.msg import Pose

class ImagePoseMatcher:
        def __init__(self):
                self.bridge = CvBridge()
                self.image_sub = message_filters.Subscriber('/orko/camera/rgb/image_rect_color', Image)
                self.odom_sub = message_filters.Subscriber('/slam/vi_odometry', Odometry)
                self.ts = message_filters.ApproximateTimeSynchronizer([self.image_sub, self.odom_sub], queue_size=10, slop=0.2)
                self.image_id = 1
                self.ts.registerCallback(self.callback)
                rospy.spin()

        def callback(self, img, odom):
                image = self.bridge.imgmsg_to_cv2(img, desired_encoding='passthrough')
                cv2.imwrite('~/images/'+str(self.image_id)+'.png', image)
                x = odom.pose.pose.position.x
                y = odom.pose.pose.position.y
                z = odom.pose.pose.position.z
                q1 = odom.pose.pose.orientation.x
                q2 = odom.pose.pose.orientation.y
                q3 = odom.pose.pose.orientation.z
                q4 = odom.pose.pose.orientation.w
                file.write("%d %f %f %f %f %f %f %f \n" % (self.image_id, x, y, z, q1, q2, q3, q4))
                self.image_id = self.image_id + 1
                rospy.sleep(0.5)

if __name__ == '__main__':
    try:
        file = open("image_pose_matchings.txt", "w")
        rospy.init_node('image_pose_matching')
        matcher = ImagePoseMatcher()
    finally:
        file.close()
```

Appendix E: The Image Classification Algorithm for Crack Detection

```
#!/usr/bin/env python

import os, shutil
import numpy as np
import json
from keras.models import model_from_json
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input

directory = os.path.dirname(os.path.realpath(__file__))

with open(directory +'/images/image_pose_matchings.txt') as f:
    matchings = []
    for line in f:
        line = line.split()
        if line:
            line = [float(i) for i in line]
            matchings.append(line)

def load_model(prefix):
    with open(prefix+".json") as json_file:
        model_json = json_file.read()

    model = model_from_json(model_json)
    model.load_weights(prefix + ".h5")

    with open(prefix+"-labels.json") as json_file:
        tags = json.load(json_file)

    return model, tags

def test(model_prefix='brick', test_path=directory+'/images'):
            file = open(test_path + '/predicted/image_pose_matchings.txt', 'w')
            target_path = test_path + '/predicted/'
            count = crack_num = 0

            print "Loading the model"
            model, tags = load_model(model_prefix)

            print "Testing starts"
            for i in xrange(1, len(os.listdir(test_path))):
                        try:
                                    img_path = test_path + '/' + str(i) + '.png'
                                    img = image.load_img(img_path, target_size=(224, 224))
                                    x = image.img_to_array(img)
                                    x = np.expand_dims(x, axis=0)
                                    x = preprocess_input(x)
                                    pred = model.predict(x)
                                    pred_class = np.argmax(pred)
                                    count += 1
                                    if not pred_class: #since crack class is 0
                                                crack_num += 1
                                                orig_name = target_path + str(i) + '.png'
                                                target_name = target_path + str(crack_num) + '.png'
                                                shutil.copy(img_path, target_path)
                                                os.rename(orig_name, target_name)
                                                file.write('%d %f %f %f %f %f %f %f \n' % (crack_num, matchings[i-1][1],
                                                            matchings[i-1][2], matchings[i-1][3], matchings[i-1][4], matchings[i-1][5],
                                                            matchings[i-1][6], matchings[i-1][7]))

                        except BaseException, e:
                                    print e

            print "Number of images tested: ", count
            print "Number of predicted images with cracks: ", crack_num
            file.close()
```