

https://dspace.org/

A distributed architecture for unmanned aerial systems based on publish/subscribe messaging and simultaneous localisation and mapping (SLAM) testbed

Saib, Kamil Nuther

2017

https://hdl.handle.net/10539/24843

Nuther Saib, Kamil (2017) A distributed architecture for unmanned aerial systems based on publish/subscribe messaging and simultaneous localisation and mapping (SLAM) testbed, University of the Witwatersrand, Johannesburg, <http://hdl.handle.net/10539/24843>

Downloaded from a DSpace repository. https://dspace.org/

A Distributed Architecture for Unmanned Aerial Systems based on Publish/Subscribe Messaging and Simultaneous Localisation and Mapping (SLAM) Testbed

Kamil Nuther Saib



A Dissertation submitted in fulfilment
for the degree of Master of Science.

School of Computational and Applied Mathematics,
University of the Witwatersrand,
Johannesburg, South Africa.

November 2017

Abstract

The increased capabilities and lower cost of Micro Aerial Vehicles (MAVs) unveil big opportunities for a rapidly growing number of civilian and commercial applications. Some missions require direct control using a receiver in a point-to-point connection, involving one or very few MAVs. An alternative class of mission is remotely controlled, with the control of the drone automated to a certain extent using mission planning software and autopilot systems.

For most emerging missions, there is a need for more autonomous, cooperative control of MAVs, as well as more complex data processing from sensors like cameras and laser scanners. In the last decade, this has given rise to an extensive research from both academia and industry. This research direction applies robotics and computer vision concepts to Unmanned Aerial Systems (UASs). However, UASs are often designed for specific hardware and software, thus providing limited integration, interoperability and re-usability across different missions. In addition, there are numerous open issues related to UAS command, control and communication(C3), and multi-MAVs.

We argue and elaborate throughout this dissertation that some of the recent standard-based publish/subscribe communication protocols can solve many of these challenges and meet the non-functional requirements of MAV robotics applications. This dissertation assesses the MQTT, DDS and TCPROS protocols in a distributed architecture of a UAS control system and Ground Control Station software. While TCPROS has been the leading robotics communication transport for ROS applications, MQTT and DDS are lightweight enough to be used for data exchange between distributed systems of aerial robots. Furthermore, MQTT and DDS are based on industry standards to foster communication interoperability of "things". Both protocols have been extensively presented to address many of today's needs related to networks based on the internet of things (IoT). For example, MQTT has been used to exchange data with space probes, whereas DDS was employed for aerospace defence and applications of smart cities.

We designed and implemented a distributed UAS architecture based on each publish/-subscribe protocol TCPROS, MQTT and DDS. The proposed communication systems were tested with a vision-based Simultaneous Localisation and Mapping (SLAM) system involving three Parrot AR Drone 2 MAVs. Within the context of this study, MQTT and DDS messaging frameworks serve the purpose of abstracting UAS complexity and heterogeneity. Additionally, these protocols are expected to provide low-latency communication and scale up to meet the requirements of real-time remote sensing applications. The most important contribution of this work is the implementation of a complete distributed communication architecture for multi-MAVs. Furthermore, we assess the viability of this architecture and benchmark the performance of the protocols in relation to an autonomous quadcopter navigation testbed composed of a SLAM algorithm, an extended Kalman filter and a PID controller.

Declaration

I, Kamil Nuther Saib, hereby declare the contents of this dissertation to be my own work unless otherwise explicitly referenced. This dissertation is submitted for the degree of Master of Science (Dissertation) at the University of the Witwatersrand, Johannesburg. This work has not been submitted to any other university, or for any other degree.

....

Signature *Date*

Acknowledgement

"Everyone can count the number of seeds in an apple, but only God can count the number of apples in one seed.", quote attributed to a pastor, makes one think of what is and what is not possible in this world. Scientific research and experiments creates new possibilities out of the pursuit of knowledge, however all praise goes to the Knower of everything, Allah the almighty and Muhammad, his messenger (peace be upon him).

Without my beloved wife Shameeah, our son and daughter, Abduli and Huda, who have been my light, strength, motivation and reason in life, this work would not have been possible.

My thanks and humble gratitude to Professor Turgay Celik who has brought much guidance, advice and support to this work. I would also like to thank professor Michael Mitchley who co-supervised and advised some parts of this project. Lastly, my appreciation to the Wits Intelligent Systems Experts (WISE) Research Group, of cross-skilled mathematicians, roboticists, statisticians, data scientists, artificial intelligence modellers, computer vision scientists for their contributions in various ways to this work through learning from them, their questions, debates and advises.



Abduli, 2014.

Contents

Abstract i

Declaration ii

Acknowledgements iii

List of Figures xii

List of Tables xiii

1 Introduction 1

1.1 Background to the problem 2

1.2 Research motivation and objectives 3

1.2.1 Research motivation 3

1.2.2 Aim and objectives 4

1.3 Research context 7

1.4 Research questions 8

1.5 Organisation of the dissertation 9

2 Unmanned Aerial Systems (UASs) 10

2.1 Introduction 10

2.2 Characteristics of UASs 10

2.3 Unmanned Aerial System(UAS) design 11
2.4 Summary 13

3 Micro Aerial Vehicles(MAVs) and Their Applications 15

3.1 Introduction 15
3.2 Micro Aerial Vehicles(MAVs) 15
3.3 MAV applications 18
3.4 Summary 20

4 Distributed UAS Frameworks and Communication Technologies. 21

4.1 Introduction 21
4.2 MAV networks 21
4.3 Distributed UAS frameworks 23
4.3.1 Multi-MAV systems 25
4.4 Distributed robots communication 27
4.4.1 Robot Operating System (ROS) 27
4.4.2 Internet of Things (IoT) publish/subscribe messaging protocols . 29
4.5 Summary 33

5 MAV Autonomous Navigation 34

5.1 Introduction 34
5.2 Robot mapping and localisation 34
5.2.1 Simultaneous Localisation and Mapping (SLAM) 35
5.2.2 Bayes filters and related models 42
5.2.3 Sensors used for state estimation 46
5.3 MAV control 47
5.3.1 MAV control architecture and the PID controller 47

5.4 The autonomous navigation system 50
5.4.1 The monocular SLAM system 50
5.4.2 The extended Kalman filter 54
5.4.3 The PID controller 57
5.5 Summary 57

6 Research Methodology **58**
6.1 Introduction 58
6.2 Research approach and methods 58
6.3 UAS system requirements 60
6.4 System design 62
6.4.1 Distributed communication framework selection 63
6.4.2 TCPROS, MQTT and DDS protocols 65
6.4.3 The communication gateways 65
6.4.4 The autonomous navigation testbed 69
6.5 Data collection and analysis 71
6.6 Summary 72

7 System Implementation and Experiments **74**
7.1 AR Drone MAV platform 74
7.2 System overview 75
7.3 Communication gateways 77
7.3.1 MQTT sender and receiver 77
7.3.2 DDS sender and receiver 78
7.4 The navigation testbed 78
7.4.1 ROS platform and software components 78
7.4.2 EKF data fusion and time delays 80

7.5 Experiments design 82

8 Experimental Results **84**

8.1 MQTT, DDS and TCPROS message transmission performance. 84

8.1.1 Results - MQTT communication latencies 84

8.1.2 Results - DDS communication latencies 88

8.1.3 Results - Network latencies averaged over 10 mins experiments for MQTT and DDS 89

8.1.4 Results - Network latencies of MQTT, DDS and TCPROS in a multi-MAV and constrained bandwidth scenario 90

8.1.5 Results - Frequency and bandwidth utilisation of MQTT, DDS and TCPROS in a multi-MAV scenario and constrained bandwidth . . 91

8.2 EKF-PTAM prediction model accuracy 94

9 Conclusion and Future Work **100**

A Technical Documentation **103**

A.1 A.R. DRONE 2.0 PLATFORM 103

A.2 AR-DRONE IP AND WIFI SETTINGS 104

A.3 NETWORK LAYOUT 105

A.4 MQTT AND DDS GATEWAYS 107

A.5 EKF-SLAM-PID NAVIGATION SYSTEM 109

A.6 EKF CALIBRATION 109

A.7 DATA SET FOR COMMUNICATION GATEWAY AND SLAM NAVIGATION SYSTEM 110

A.8 DDS, TCPROS, MQTT MEASUREMENTS FOR COMMUNICATION LATENCIES FOR ONE, TWO AND THREE DRONES 110

Bibliography **119**

List of Figures

1.1 Conceptual architecture of the proposed sub-systems and communication primitives for a simple scenario of two ground stations and few MAVs. . . 5

1.2 Alternative setup of proposed system, communication primitives and sub-systems, using an on-board messaging client for fully autonomous control. 7

2.1 Functional components of UASs and purposes of each: automation software/hardware to control and program the UAVs, and the communication links to analyse data from UAVs. 12

2.2 A Pixhawk autopilot as illustrated in [1]. 12

3.1 Evolution of MAV platforms and specifications according to [2, 3]. 16

3.2 Examples of unmanned aerial systems remote sensing applications 18

3.3 A glimpse into the future, with the DronePort project, appeared in the United Nations Office for Project Services (UNOPS) annual publication. The concept describes how drones goods transportation could become the centre of community life. 19

4.1 Unmanned aerial systems communication networks. 22

4.2 ROS nodes initialise connection with intervention of the master and exchange data directly using publish/subscribe mechanism. 28

4.3 Three-layered IoT architecture model and design considerations as presented in [4]. 29

4.4 MQTT and DDS communication models. 31

4.5 MQTT-SN architecture and specification (Copyright IBM Corporation 1999, 2013.). Clients can connect to gateways dynamically based on the range or when they become available. Gateways can also help with load sharing. 32

4.6 QoS-controlled data-sharing communication. Subscriptions can specify time and content filters and get only a subset of the data being published [5]. 33

5.1 The essential SLAM algorithm problem [6]. The *True* locations are not known, hence the need to estimate the robot and landmark locations simultaneously. 37

5.2 SLAM process as proposed in [4]. Here, following an observation, the local perception or map has to be associated with the global one in order to update both the robot pose and global map. 38

5.4 PTAM initialisation process. The initialisation process needs a baseline to function properly. To provide this, the camera need to move sideways between the first two keyframes. 40

5.5 PTAM mapping process flow as in [7]. The input is keyframes computed from the camera image frames. 42

5.6 Block diagram of a standard input/output feedback control system. The goal or desired value x_{des} is constantly compared to the measured state x_t, to determine the error e_t, which in turn specifies the controller output u_t based on the control gain K.. 48

5.7 System response using various gains of the PID-controller, see [8]. The black, green and red graphs show the effect of applying different gains K_p, K_i and K_d for a step input. 49

5.8 The drone body (b) and world (w) coordinate systems. Φ, Θ and Ψ are the orientation angles: roll, pitch and yaw respectively. 50

5.9 PTAM tracking from the camera feed. The colours blue, green, yellow, and red denote intensity of the FAST corners or how coarse the feature points were identified at: blue representing the coarsest and red representing the finest patches. 51

5.10 GUI for drone control, PTAM camera-feed/frame tracking(top right) and PTAM drone pose 3D map (bottom right). 54

5.11 GUI for PTAM 3D map - The *Camera positions* are shown as red-white-green world coordinate frames, where-as red crosses denotes landmarks position. 54

6.1 Extract of an example of the communication gateway parametrisation. *subscribed*** ROS topics are in-bound data coming from the navigation system and *published*** MQTT topics are out-bound communication to the control unit. 61

6.2 Distributed MAV communication network. 62

6.3 Logical architecture of the UAS distributed model and middleware platform to communicate with server applications. 63

6.4 MAVs connecting to a load balancer, forwarding all connections to two MQTT brokers and control units. 67

6.5 System architecture for the communication gateways based on MQTT. . 68

6.6 UAS and control units as DDS participants in a scaled, high-availability scenario. 69

6.7 Camera-based navigation system components of the testbed and delays in data communications. 70

6.8 Overview of the three approaches for data collection, involving the navigation system from [9] and the communication agents. 71

7.1 Parrot AR Drone2 MAV platform 75

7.2 Parrot AR Drone2 MAV platform sending and receiving data using the ROS 'ardrone_autonomy' driver. 75

7.3 Implementation of the software components and outline of SLAM algorithm. Here, the extended Kalman filter computes an accurate estimate of the drone's pose and speed at each video frame. 76

7.4 ROS software components. 79

7.5 The Kalman filter takes values from a buffer and must be rolled forward to compensate for delays at which a video frame was received. 81

7.6 System architecture for an experiment setup with 3 MAVs. 82

7.7 Network layout and setup for an experiment with 3 MAVs and the Netgear WiFi router. 83

8.1 Packaging and transmission of video frames (sent at 30 Hz) and IMU sensor data (sent at 200 Hz) published in real-time, with no loss in sending and receiving frequency. 85

8.2 PTAM map of coordinate frames for each drone pose while holding position for 60s. 86

8.3 IMU data messages latencies for MQTT protocol averaged for each experiment with the drone hovering for 60 s. 87

8.4 Image packets latencies for MQTT protocol averaged for each experiment with the drone hovering for 60 s. 87

8.5 IMU data packets latencies for DDS protocol averaged for each experiment with the drone hovering for 60s. 88

8.6 Video frames packets latencies for DDS protocol averaged for each experiment with the drone hovering for 60s. 88

8.7 Navadata frames packets latencies for MQTT and DDS protocol over 10 minutes experiment. 89

8.8 Video frames packets latencies for MQTT and DDS protocol over 10 minutes experiment. 89

8.9 Communication latencies for video frames under multiple scenarios. . . . 91

8.10 **Frequencies** of images received from the master MAV, based on following scenarios: 1) MQTT, DDS, or TCPROS protocol, 2) in a system of one, two or three MAVs, and 3) over a 100 Mbps or Gigabit Ethernet LAN. . 92

8.11 **Bandwidth** utilisation of video frames received from the master MAV, based on following scenarios: 1) MQTT, DDS, or TCPROS protocol, 2) in a system of one, two or three MAVs, 3) over a 100 Mbps or Gigabit Ethernet LAN. 92

8.12 Video streams transmission frequencies in a distributed system with 3 MAVs. 93

8.13 Camera images and PTAM maps for 60 s experiment with the MAV hovering. 94

8.14 Estimated positions of the MAV in non-distributed setup. Communication between MAV driver and navigation system, both running on the same computer (i.e. non-distributed mode). 95

8.15 Estimated positions of the MAV in a distributed setup. Both estimations of the Kalman filter (*filterData*) and pose prediction based on PTAM (*ptamData*), are done on the remote control unit. 96

8.16 MAV flying a square figure. 96

8.17 Experiment results for one MAV flying a square figure 97

8.18 Experiment results for two MAVs, and one of the MAV flying a square figure 98

8.19 Experiment results for three MAVs, and one of the MAV flying a square figure 99

A.1 Parrot AR Drone2 IMU and camera coordinate system: red corresponds x, green to y and blue to z coordinates respectively. 103

A.2 Network setup layouts for a single and two MAVs 106

A.3 WiFi channels analyser for two Parrot AR Drones in university lab . . . 107

List of Tables

2.1 UAS components and services for configuration and orchestration of UAV missions. 11

3.1 Micro Aerial Vehicle (MAV) categories and common capabilities of MAVs available off-the-self [10–12]. 17

4.1 Summary of the layers of the mission-oriented protocol framework as presented in [13]. 26

4.2 Qualitative comparison of MQTT, DDS, CoAP and MQTT according to [14–17] .
M2M: Machine-to-Machine, M2S: Machine-to-Server, S2S: Server-to-server 30

8.1 Efficiency of the communication protocols transfer rates. 93

A.1 Parrot AR Drone2 *navdata* data structure. 104

A.2 Average throughput, frequencies and bandwidth measurements for set of experiments. 111

Chapter 1

Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) have become a popular topic in commercial, military and governmental applications. The technology has evolved to Miniature Aerial Vehicles (MAVs), and has gained major attention in a number of civilian applications. MAVs can fly autonomously within or out of the line-of-sight to serve a variety of purposes.

Due to the overwhelming popularity of MAVs, technology giants like Amazon and Alphabet are currently developing commercial programs to tap into the huge market potential of drones. Small, low cost, commercial autonomous drones are being considered as an important part of the Internet of Things (IoT). The application of this technology includes automated package delivery, precision farming, terrain mapping, 3D modelling, surveillance and security, support of minimally manned installations, and wildlife monitoring, to name only a few. Autonomous drones are also being used in manufacturing and production industries such as oil, gas, mining, and railway etc. Application of autonomous drone is far and wide as compiled by the Canadian Centre for Unmanned Vehicle System, see [18].

This huge burst of popularity and market growth has put pressure on aviation authorities and researchers of this field. Both entities need to address some critical concerns and challenges if they want the MAV technology to provide real benefits to the civilian market. The major concerns focus on control systems, public safety, privacy, and data security. Since most of the drones are managed from their ground control stations, researchers are expected to devise efficient mechanisms to manage MAV missions.

Unmanned Aerial Systems (UASs) are required to address the need of all these applications, and manage drone missions from a ground control station. A UAS, in very general case, consists of the following main components: a UAV, a ground control station (GCS) operated by a "pilot", and data links to transmit control inputs to the aircraft and receive payload/telemetry data from the aircraft. The work presented in [19] elaborated how diverse types of UASs are designed and implemented. Recently, many applications involving multiple drones have been shown. These systems aim to operate with multiple

heterogeneous drones, vastly increasing their complexity over single drone control systems. With more complex system requirements, the architecture needs to be extended with control units and M2M communication.

1.1 Background to the problem

The development of open protocols for communication between control units and MAVs can solve a number of real world problems. Despite the improvements and success of research-driven UAS hardware and software, it is difficult to implement a practical, real-time MAV mission. The complexity lies in: 1) collecting and interpreting data, 2) configuring the embedded avionics, and 3) establishing secure and quick communications with the GCS. MAV autonomous navigation is another complex problem that is a challenge for research community. This involves topics like probabilistic state estimation, linear control, and path planning. Academic research in the robotic labs around the world has been focused on the development of autonomous MAVs. Efficient and reliable network communication is a critical component of enabling robots to navigate without human intervention.

Another major challenge lies in implementation of algorithms for multiple-MAVs missions. With growing advancements in the industrial and commercial application of MAVs, it is becoming essential to develop and implement algorithms that can control different categories of MAVs or multiple-MAVs. Even when cooperation is not needed for some applications, it is still essential to build robust and integrated remote sensing software to bridge the gap between wireless network communications and other sources of information. Hence, the ability to communicate among themselves and with other devices are essential requirements for enabling aerials robots to realise meaningful missions.

Since UASs require a combination of both wired or wireless connections to communicate with MAVs, this domain poses further research challenges. UASs often also comprise of dynamic network topologies with high performance and reliability requirements. Thus, a major challenge lies in choosing the appropriate communication standard to power MAV systems.

Furthermore, MAV platforms and ground control stations generally have limited computation power to execute resource intensive tasks like planning, mapping, and vision-based algorithms. Mobile robots could integrate with intelligent external autonomous robotic systems running on a cloud platform to share and augment the abilities of MAVs. These systems often involve higher volume of data and have a higher data processing rate. Therefore, MAVs require powerful hardware and software combinations. In addition, researchers aim to reduce the operation costs and increase the flight duration of UASs by offloading the algorithm computation tasks that have no stringent real-time requirements.

Integrating the MAVs to a country's National Air Space (NAS) is another major ethical and legal consideration. Research works have highlighted the important challenges

of integrating the autonomous flying robots into the civil airspace, see [20], and this field is expected to be the dilemma for all major aviation authorities. A distributed architecture that uses open standards can propose the required services to enable integration of small UAV swarms into the aviation systems within non-segregated air space (i.e. at an altitude that is not reserved for conventional aircraft). This would involve assurance of a secured communication and proper trust mechanism between a network of distributed agents.

1.2 Research motivation and objectives

1.2.1 Research motivation

This research derives its motivation from the fact that flying robots will play an important role in the 21st century's technological evolution. This is driven by recent emerging advances in MAV mechatronics, autopilots, data transfer rates, and payload. This research investigates the communication architectures within unmanned aerial systems for the advancement of its industrial and commercial application. With the proliferation of wide-area networks and the Internet, there has been a clear trend in various fields to move from confined systems based on a monolithic approach to a distributed architecture. First, a comprehensive literature review is presented. Furthermore, emerging communication protocols in the field of unmanned aerial system are described. To analyse the communication systems, this study uses a robotics navigation system as a testbed, and demonstrates the performance and scalability of an architecture design to route high volume of data in UASs at high frequencies.

Furthermore, this dissertation develops and demonstrates a messaging software architecture that can execute low latency navigation and computer vision algorithms. In the process, it aims to simplify the overall hardware and software complexity as a service oriented platform, hence improving the interoperability and MAVs coordination. A major motivation for such an architecture is to enable the aerial robots offload their heavy computations to modern data centres via open communication protocols. Robotics applications have very strict computational needs to be viable in production. In addition, the computational capability of a flying robot is limited due to payload and power constraints. To answer these challenges, the proposed communication model is designed to offload computation load to a networked distributed system. This strategy to move the control unit to a data centre or cloud infrastructure allows the application to make efficient use of the MAV/GCS hardware resources and limited payload capacity. Furthermore, the use of public and private clouds– which have powerful computational, storage, and communication resources– would reduce maintenance and overhead costs.

This dissertation implements a middleware-based MAV communication and evaluates the proposed architecture with a Simultaneous Localisation and Mapping (SLAM) algorithm. The experimental results are obtained from a simulated in-door setup of a distributed UAS of few MAVs and findings from experiments helped to assess the performance, challenges and viability of the proposed system.

Finally, the intent of this work is to describe a possible architecture which will enable to build applications needed to execute similar low-latency algorithms. Eventually, the same concept can be extended to develop applications that require long range communication, multi-MAVs swarm control, and dense environment mapping.

1.2.2 Aim and objectives

The research aim of this project is to develop a UAS for MAV robotics applications based on emerging standard publish/subscribe communication protocols. The intent is to investigate the reliability, ease, and performance of such protocols for near real-time processing of information.

One of the main objectives is to simplify the data processing capability of MAV missions by using a distributed control architecture. The proposed architecture implements the communication gateways to exchange data between the two main components - the UAS Control Unit (UASCU) and the GCS, as depicted in Fig. 1.1. The GCS and control unit can exchange information using the default protocol of ROS middleware like TCPROS. However, the focus of this study is to evaluate the recent publish/-subscribe messaging protocols as alternative communication standards between the distributed agents. In this respect, TCPROS and two lightweight middleware technologies are tested, namely the MQTT [21] and Data Distribution Service (DDS) [22]. MQTT is a publish/subscribe ISO standard protocol (formerly Message Queue Telemetry Transport) and described as a "lightweight" messaging protocol for IoT sensors and mobile devices. It provides three QoS levels to guarantee delivery of messages between the UAS and the Control Unit. MQTT uses a broker as the back-end component to connect decoupled MQTT clients. DDS, on the other hand, does not need a broker and communicates in a decentralized peer-to-peer model; data is exchanged within a domain space, and accommodates both Machine-to-Machine (M2M) and Machine-to-Ground (M2G) communication. Both MQTT and DDS promise to provide better performance and scalability than TCPROS and conventional SOA protocols.

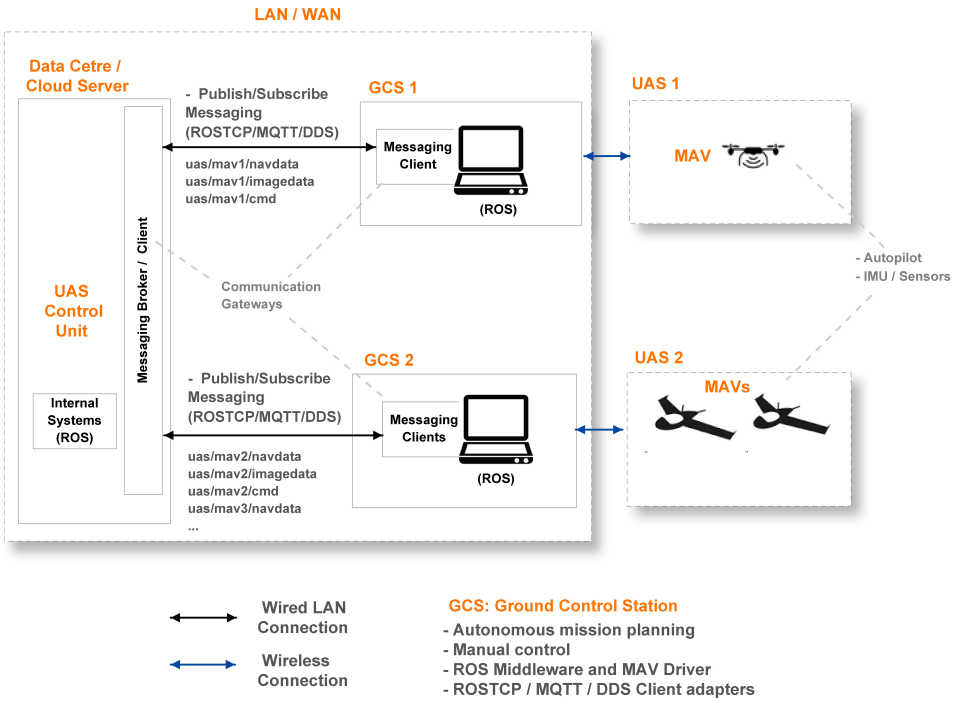


Figure 1.1: Conceptual architecture of the proposed sub-systems and communication primitives for a simple scenario of two ground stations and few MAVs.

MAVs data can be obtained using various kinds of sensors, such as inertial measurement units (IMUs) and real-time image/video capturing devices. In the proposed architecture, the GCS relays data to the UASCU for processing or to expose it to other systems. Hence, this abstracts the complexity of the UAS from the main processing systems. This distributed architecture also allows the possibility of offloading computationally intensive processes from the drone to a control system. Multiple data processing sub-systems can be integrated in the control unit (referred as 'internal systems' in Fig. 1.1).

Primarily, this research study objective is to focus on the *design, development, simulation and testing of a distributed UAS architecture with a vision-based real-time navigation algorithm.* Fig. 1.1 illustrates the following main components of the distributed Unmanned Aerial System (UAS):

1. **The micro aerial vehicle(s) and payloads.**
   The MAV platform consists of an IMU and an on-board computer connected to the on-board sensors. It provides sensor measurements (navigation data, image/video frames, range scanners like lasers and depth cameras) and receives flight commands. It also consists of a radio transmitter and receiver.

2. **UAS messaging client.**
   The base station is designed to act as a client using publish/subscribe standard communication protocol to transfer information between the UASCU server and MAV. It uses a communication pattern that meets the key architecture requirements

like performance, security, and robustness. The data transmission technology over the network (from GCS to UASCU) is based on TCP/IP protocol across a LAN or WAN (Internet) network. The motivation for having the GCS and UASCU within the same local/wide area network or data centre is to benefit from a reliable network, bandwidth and computing power as required by robotics applications.

3. **UAS Control Unit (UASCU) middleware.**
   The UASCU offloads the computationally intensive processes from the UAS to a control unit and sub processing systems. The UASCU can be hosted within a cloud computing server infrastructure with shared services accessible to multiple agents and ability to implement failover mechanisms. Additionally, it allows integration to external systems and multi-MAV control.

One of the primary advantage of a standard messaging protocol is that it eliminates the need to build custom middleware for specific MAVs and missions. A main objective of the system is to use a service-oriented architecture which would allow the control unit and GCS to interact via reliable packet based networks. A service-oriented architecture allows sending commands and receiving payload data across distinct, pre-defined interfaces. For instance, the data from the MAVs in Fig. 1.1 is transmitted through **topics** like *uas/mav2/navdata* and *uas/mav2/image2*.

MQTT and DDS service-based models and event-based programming paradigms can ease the implementation of distributed software through network abstraction, service discovery, authentication, Service Level Agreement (SLA) or Quality of Service (QoS) parameters for reliable data delivery and performance. In addition, the proposed architec; e.g. intelligent data processing systems, web servers, and mobile devices. The driving force towards exploring an open distributed architecture was to ease development and enable a wider range of remote sensing applications. In this respect, the aim behind using standard communication protocols is to provide higher levels of guarantee and security for safe operation of networked MAVs.

The future aim of the proof-of-concept implemented in this research, based on open communication technologies, is to enable the messaging client to be part of the MAV hardware. This will significantly decrease reliance on the GCS. Here, the messaging client could be hosted on an embedded microprocessor platforms like Raspberry Pi, BeagleBone, Arduino or any Linux powered on-board computer. Fig. 1.2 shows a conceptual view of the resulting alternative setup, where the control can be fully-automated through 3G/4G GSM networks or other faster WLAN networks. In this case, the MAV plans and executes the missions through a Control Unit. The GCS becomes an optional component since the control capabilities can be within the UASCU. While the set-up in Fig. 1.2 is meant to be fully autonomous, generally it also has a direct link to the MAV within a given range to enable manual overriding operation, if necessary.

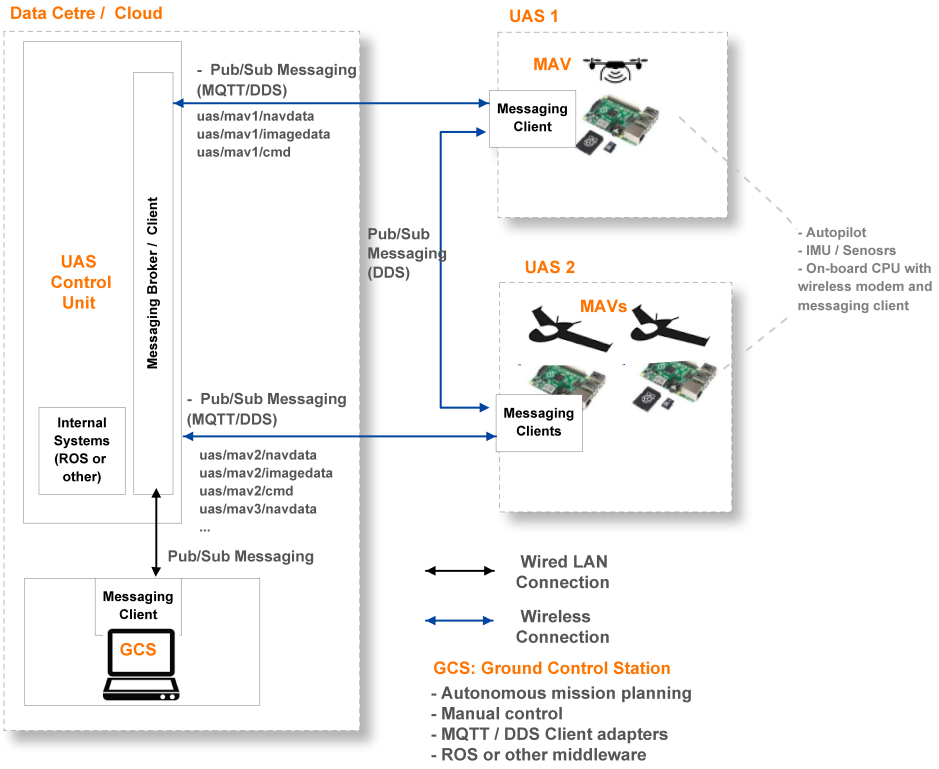


Figure 1.2: Alternative setup of proposed system, communication primitives and subsystems, using an on-board messaging client for fully autonomous control.

Besides moving some GCS functions and messaging client to the embedded MAV processing unit, there are some other notable differences between the proposed architecture implemented in this project (as in Fig. 1.1), and the planned evolution illustrated in Fig. 1.2. In the latter case, there is a greater need to use a platform independent middleware technology built specially for long-range communication, low-latency data transfer, and uses a software client that can be executed on resource-constrained embedded devices. Here, the ROS transport TCPROS is not a suitable candidate as ROS is known to be a resource intensive middleware. Also, M2M messaging is only realisable with DDS as neither MQTT nor TCPROS provide for peer-to-peer communication, needing a broker and master respectively.

1.3 Research context

Real-time tracking and control is essential for some categories of MAV applications where the control software needs to be executed at the GCS or within embedded device OS. On the other hand, for many other applications, processing can be distributed to other systems. In this respect, this research aims at deducing **what are the system constraints and what is the achievable performance in the proposed real-time distributed**

environment when executing a robot mapping algorithm on the UASCU. MAV real-time applications need to run on the same UASCU instance, cluster or private cloud ('Internal Systems') for higher performance. This study proposes an architecture to help cope with the strict latency, bandwidth, and QoS in MAV communication systems. Further, this would eventually provide the ability to bridge the data communication gap between MAV and other systems. Studies in network robotics, based on the cloud architecture paradigm, have discussed the possibilities for MAVs to interact within a distributed architecture [23, 24]. The intention of this research is to present a practical approach of a MAV communication system within wide/large area networks, using messaging technologies based on open standards. Such communication has predominantly been achieved using device specific communication and using purpose built drivers on Robot platforms such as ROS. Hence, we critically argue the need for investigating the use of alternative standard-based protocols.

A key area where this work is relevant is within the context of near real-time MAV robotics applications, and to assess the architecture and messaging protocols for systems in this domain. The application used as testbed in this dissertation is a MAV autonomous navigation system. Navigating in unstructured environments is a complex task which needs fusion of data from multiple sensors and IMUs. SLAM is an example of a similar task where the vehicle must estimate both its location and the map of the environment. It is a widely studied problem for robots to navigate autonomously. In SLAM, the robot starts at an unknown environment and simultaneously maps the environment and locates itself. On similar ground, this research work aims to use an extended Kalman filter (EKF) based SLAM navigation system as testbed with different implementations of the communication architecture and using real-life MAVs as distributed agents.

1.4 Research questions

This study seeks answers for the following research questions:

1. (a) Does a distributed architecture based on publish/subscribe protocols like MQTT and DDS address the complexity of implementing diverse UAS applications?
   (b) Does the distributed architecture proposed based on standard-based messaging also allow easy integration with diverse types of UAS and data processing applications?
2. (a) How does the publish/subscribe protocols MQTT and DDS compares to each other in terms of performance(i.e. communication latency) and bandwidth?
   (b) How does these protocols compares to conventional robotics messaging transport like TCPROS?
3. How well an architecture based on distributed computing and asynchronous communication technologies meet the functional and non-functional requirements of a real-time MAV robotic algorithm?

4. (a) Does the system based on MQTT/DDS provides the required performance, robustness, accuracy, resiliency, and scalability for the MAVs to navigate in an unknown and GPS denied environment?

   (b) And finally, how does the MQTT/DDS/TCPROS variants of the architecture perform in a multi-MAV and constrained bandwidth environment?

1.5 Organisation of the dissertation

Chapter 2 provides a comprehensive literature review on UAS and works related to the design of the different components of UASs' components. In Chapter 3, we survey different MAV platforms and their remote sensing applications. Chapter 4 looks at MAV networks and related distributed communication technologies. We also give an overview of the IoT concept in relation to robotics. Chapter 5 introduces the robot mapping and localisation problem. It explores the Bayesian filtering methods and SLAM algorithms used in autonomous navigation systems. Chapter 6 covers the methodology to design the UAS Control Unit, communication gateways and SLAM navigation testbed. Chapter 7 describes the overall system architecture and software implementation components followed by a description of the experiments carried out. Chapter 8 outlines the results and performance obtained from the distributed UAS architecture along with the accuracy of the SLAM algorithm. Finally, Chapter 9 concludes with the dissertation achievements and proposes future directions for further research.

Chapter 2

Unmanned Aerial Systems (UASs)

2.1 Introduction

This chapter gives an overview about UASs, their characteristics, and their functional design elements. The primary aim of this chapter is to define the components within a UAS and describe the functions of each part of the system.

2.2 Characteristics of UASs

A UAS is an intelligent system of the UAVs, and recent research from academia and industry have demonstrated its importance for scientific applications. Literature on UAS cover primarily medium to large UAVs for commercial and military applications. *UAS is defined as a system whose components includes the necessary equipment, network, and personnel to control an unmanned aircraft*, see [25]. C3 (Command Control and Communication) is the main system within a UAS, besides the other activities like personnel for the mission, shelter facilities, launch, and recovery [26]. The survey in [26] categorises UAS C3 into two broad sets: LOS (within radio frequency Line-of-Sight) and BLOS (Beyond Line-of-Sight). In addition, each group have other subcategories of systems, some specific or common to both group of C3. Example of subsystems include command and control, autonomy, data links, and Air Traffic Control (ATC). BLOS can also mean the systems that are beyond radio frequency or BVLOS (Beyond Visual Line-of-Sight), this in turn implies that the UAV could be connected using longer range frequencies or cellular data network. BVLOS navigation requires precise mission planning and additional autonomous features.

In fact, designing, developing, and operating a UAS covers various other research disciplines in aeronautics; a detailed argument can be found in [19]. Table 2.1 summarises key UAS concerns in the category of UAVs that fly within non-segregated airspace. The features outlined are generally required for mission configuration and orchestration of

most typical MAV application.

Payload	Awareness services	Flight services	Mission services	Data Links
Actuators	Conflict detection	Virtual autopilots	Data Storage	Protocols
Radars	Sense and avoid	Path planning	Mission management and monitoring	Bandwidth
Image sensors		Contingency management		Frequency usage
Sensor data acquisition		Flight monitoring	Real time data processing	Link loss procedures
		Air traffic control		

Table 2.1: UAS components and services for configuration and orchestration of UAV missions.

2.3 Unmanned Aerial System(UAS) design

UASs are systems of UAVs composed of multiple other sub-systems responsible for specific set of functions. The UAV is equipped with navigation/sensing equipment and payloads. An on-board radio or modem transmits data to the ground station. There is also a need for manual launch/recovery/control of the aircraft. Ground control station includes software and hardware for configuring, planning missions, programming behaviours, and monitoring progress of a given task or mission. In [19, 25] an elaborate background on UASs is given, in addition to all aspects related to the design and development of UAV missions. Recently, the use of MAVs to enable robotics and remote sensing applications have attracted researchers' attention.

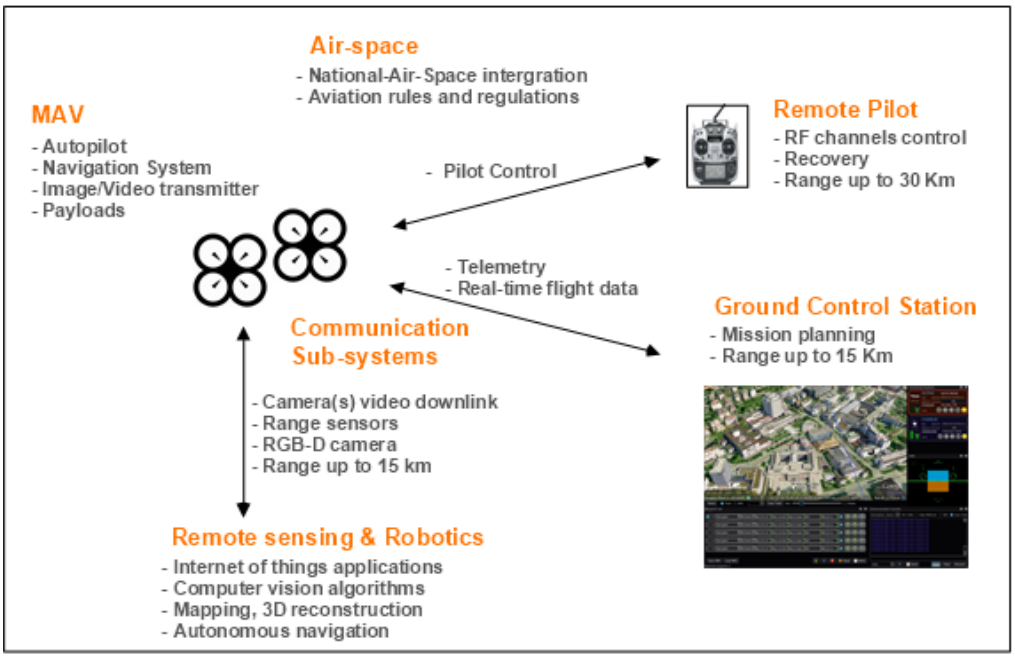


Figure 2.1: Functional components of UASs and purposes of each: automation software/hardware to control and program the UAVs, and the communication links to analyse data from UAVs.

Fig. 2.1 presents a general abstraction of each component of UASs and its features. The following points provide a detailed description, current trend, and example from literature for each of the functional components abstracted hereafter:

1. **UAV** is the main component, which houses the on-board **flight controller, payloads, and sensors**. UAVs can carry a wide range of payload for various purposes; for example, typical sensors are used to gather visual data, survey an area, or capture data from other mobile/non-mobile sensors during a mission. Examples of payloads includes image/video camera, infrared/thermal camera, and Light Detection and Ranging (LiDAR) device. Beside the traditional mechanical control, a flight controller or autopilot (e.g. the 'Pixhawk autopilot', as shown in Fig. 2.2) serves as the 'brain' of the UAV. It is used to enable autonomous navigation, and it can be programmed to connect multiple sensors or communication modules, see [1].



Figure 2.2: A Pixhawk autopilot as illustrated in [1].

2. The **Ground Control Station (GCS)** provides the human control interfaces with

help of a mission planning software. Mission operators use a GCS to plan and communicate directions, as well as other desired flight behaviours which enable UAVs to perform an assigned task. Hence, the GCS is often used to configure missions and monitor the UAVs during an operation. QGroundControl [27] is an example of an open source GCS with a graphical mission planner and multi-MAV visualization. It can connect to several devices' autopilots like the Pixhawk, ArduPilotMega, and Parrot AR Drone 2.0.

3. The **Communication sub-systems** include the communication components and links between the command and control of a drone. These technologies facilitate the transmission of different types of sensor data from the aircraft to the GCS and vice-versa. This is achieved through a wireless transmission communication infrastructure and networking communication standards. Some example of long range radio frequencies used by MAVs are: the IEEE standards like WiFi 802.11n, 802.15.4 personal-area network (PAN) radio standard, LTE-A, ZigBee wireless technology. In more complex scenarios, the communication can extend to one or more MAVs with MAV-to-MAV and MAV-to-ground data interchange links.

4. **Remote sensing application** is the software that allows data from the UAVs to be collected, processed, and analysed. This enables intelligent use cases.

   The research presented in [28–30] describe examples of MAV applications which involve data capture such as terrain mapping, security operations, ocean and wild-life monitoring. These systems mainly need to capture the sensory data by flying over a GPS defined area. Afterwards, the sensory data is sent for processing/analysing.

   The UAV research community has also shown growing interests in **Robotic sensing**, imaging, and navigation techniques. For instance, navigation systems in structured or unstructured environments were reported in [31, 32]. In [9], the results involved navigating in GPS-denied environments using a camera and IMU sensors. In [33], the navigation employed four cameras in stereo mode. Other practical robotics applications include sense and avoid, and tracking of external landmarks, as described in [34, 35].

   UAVs can also have robotic arms and actuators which can interact with the environment. These type of UAV systems can perform complex tasks like package delivery, repair works, and object manipulation, as shown in [36]. **Cooperative control** of a fleet of UAVs is another important area of UAV robotics applications which is getting major attention in recent years. This direction involves studies in Machine-to-Machine (M2M) communication. For example, the work presented in [37, 38] explored co-ordination frameworks of UAVs and leader-follower algorithms.

2.4 Summary

This chapter provided the reader with a brief description of a UAS and its various components. Furthermore, this chapter described each part of the UAS system with suitable

examples and related studies, thus, presenting a general view of the modern UAS. Specific attention was drawn toward the communication technologies and remote sensing applications as this dissertation focuses on these constituents.

Chapter 3

Micro Aerial Vehicles(MAVs) and Their Applications

3.1 Introduction

This chapter narrows down the discussion from UAVs to MAVs. In this respect, this research assesses the capabilities of some new and common MAVs available off-the-shelf, and gives an overview of their technical specifications. This part extends from the concept of UAS to show how the MAV system can serve multiple applications in real-life scenarios.

3.2 Micro Aerial Vehicles(MAVs)

Though UAVs dates back to the 1920s, MAVs like quadcopters appeared quite late, around 2004. This delay is attributed to the software/hardware complexity and electronic stability issues. Recently, the capabilities of MAVs have progressed exponentially, as shown in Fig. 3.1. Moreover, MAV technology is reshaping how human beings use their airspace. UAVs are being built for military, governmental and commercial applications. Most of the manufacturers like DJI and 3D Robotics develop only a specific category of drone, while others like Boeing constructs drones for both military and commercial applications. The technology media has publicised the capabilities of the civilian UAVs, highlighting the impressive impact it can have on people's daily lives. The increase in the demand of drone applications is pushing UAV manufacturers to shift towards developing a low cost commercial device.

MAVs are a smaller form of UAVs, typically weighing less than 25Kg [39] with a wing-spanning less than 1.5m on an average. They are categorized into the following:

- Fixed-wing;

- Multi-copters; and
- Hybrids of fixed-wing and multi-copters.

Multi-copter MAVs are common because they are easy to launch and recover. They are suitable for flights range of less than 10 km with load less than 2 kg. This demonstrates rapid, frequent, low cost transport capabilities of MAVs in difficult terrains such as valleys and forests. Fig. 3.1 shows the 3DR-X8 multi-copter, which is an example of a robust MAV platform. It can be used for tasks like aerial mapping and mission planning. On the other hand, fixed-wing MAV are suitable for demonstrating long range flights ranging up to 50 km.

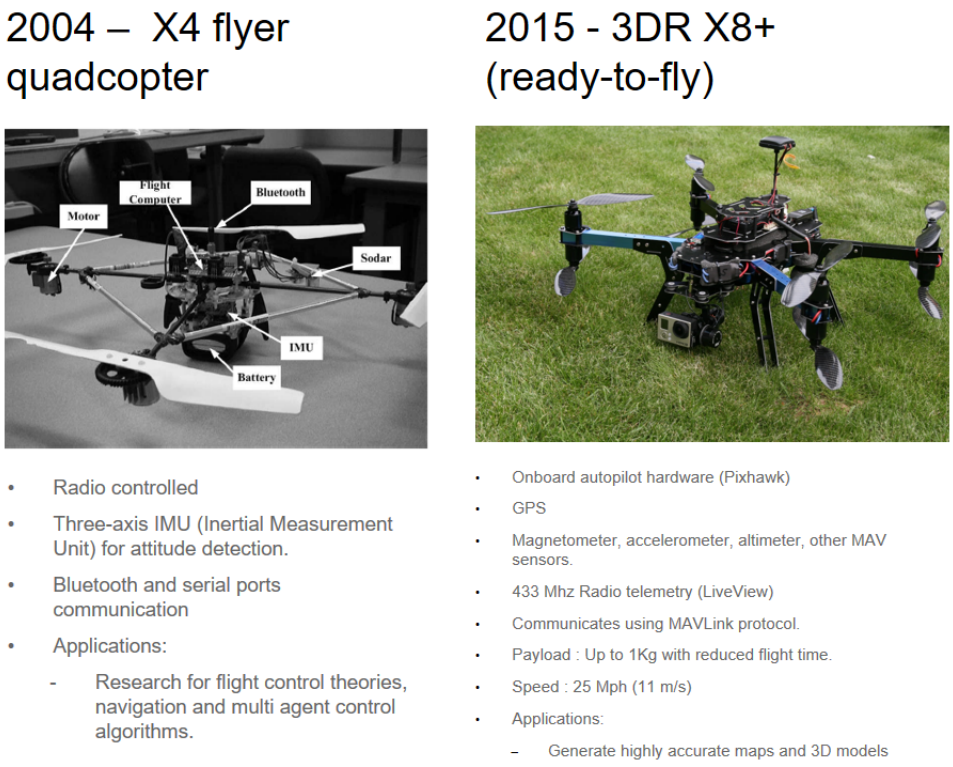


Figure 3.1: Evolution of MAV platforms and specifications according to [2, 3].

MAV capabilities have been enhanced rapidly over the past decade. Modern day MAVs comprise of on-board sensors like visual sensors (monocular or stereo), range sensors, depth sensors, thermal sensors, and infra-red sensors. Table 3.1 presents a brief survey on the most common characteristics of the new MAVs. An important feature of an advanced MAV is the fact that "Telemetry and command" bandwidth is considered separate from "Payload" (video camera, and lasers) bandwidth. For instance, [39] discussed the telemetry communication could be accommodated by a 56 Kbps link, while payload bandwidth would require at least 8 Mbps for good quality video. Additionally, [39] also describes the following communication techniques:

- Wireless Local Area Network (WLAN) 802.11 standards based on 2.4 GHz band; and
- Coded Orthogonal Frequency Division Multiplexing ("COFDM") relying on 5 to 24 GHz band.

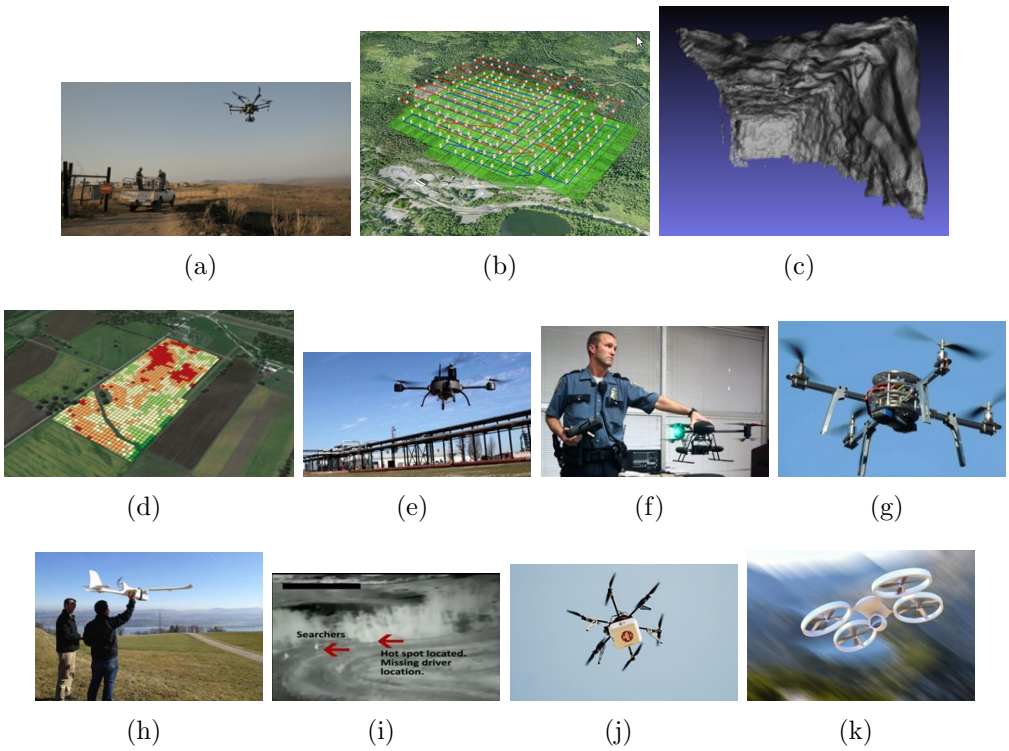
A major concern highlighted is the fact that systems for payload communication are degraded by Doppler frequency shift, and this limits the speed of the MAV. As described in [39], Orthogonal QPSK (Quadrature Phase Shift Keying) is a technique that can alleviate the Doppler effect.

Type	 Rotor, Multi-rotor, Hybrid (VTOL), Fixed wing
Size	< 25 Kg
Cost	$1-15,000 USD for drone, cameras, and support equipment
Communication	• *6-20 RF channels of control, using for e.g. GPRS Modem and Modbus protocol (Telemetry and command)* • *WLAN (2.4 Ghz Wi-Fi), 3G-4G, Video link (payload such as video camera and photography)*
Control	• Line of sight command and control • Programmable altitude and GPS Waypoint BVLOS (beyond-visuals line of sight) Autonomous Flight • "First Person View" long range real time control
Capability	• Stabilized HD video (1080p),12-24MP still photo, infrared/thermography • Programmable camera control and "Region of Interest" specification • Multiple switchable cameras with live and recorded downlink
Range	• Duration: 10-60 min • Speed: 0- 100 Km/h (30 mps) • Altitude: 1-2000 m, but limited by local laws • Control range: 150m – 50Km

Table 3.1: Micro Aerial Vehicle (MAV) categories and common capabilities of MAVs available off-the-self [10–12].

This study used the Parrot AR Drone2 MAV platform, which runs an on-board Linux computer with a WiFi interface for communication. The Parrot AR Drone2 MAV has multiple sensors, e.g. pressure, ultrasound, allowing precise control and stability of the drone. It also has an optional GPS device for supporting the semi-autonomous flight. The system is equipped with a front and bottom facing camera, which can stream navigation data and video over WiFi to an iOS or Android device. The Parrot drones have been used extensively in areas of MAV robotics research

Recently, there has been a surge in the market of nano-sized micro aerial vehicles like the nano and pico quad-copters. The smaller size and mass of a nano-sized aerial vehicle makes it possible to accelerate and manoeuvre in tighter spaces. Researchers have explored formation control and swarm control techniques to provide coordination among the MAVs, as well as planning and executing of group tasks. In particular, the work presented by [40] achieved remarkable results with pico-quadrotors. However, this is fairly a new area for real-life application, and it is driven by latest advances in the field of microelectronics, battery capacity, payload, and sensory capabilities.



(a) (b) (c)

(d) (e) (f) (g)

(h) (i) (j) (k)

Figure 3.2: Examples of unmanned aerial systems remote sensing applications: a) Wild life monitoring [28]; b) Mapping and surveying [29]; c) Mine mapping and 3D-Reconstructions [41]; d) Precision agriculture [42]; e) Minimally manned installations inspection [43]; f) Law enforcement [30]; g) Disease prevention [44]; h) Climate monitoring [45]; i) Search and rescue [46]; j) Disaster recovery and emergency [47]; and k) IoT and scientific research.

3.3 MAV applications

The introduction and motivation into this work show that UAVs are enabling progressive and disruptive applications. The UAV industry has a wide scope and a crucial role to play in the near future. This research narrowed down the focus on MAVs, which are a special form of miniature UAVs. Fig. 3.2 lists down some MAV applications that have received higher attention in recent years. In many smart city scenarios, MAVs are used regularly, filling the gaps in cases like 3D (Dull, Dirty and Dangerous) operations.

MAVs have been widely used in filming and surveillance activities; but, many companies now use them for delivery services and medical aid. Therefore, MAVs provide both commercial and social value to our community. They can also be used in the health-care transport systems of under-developed and complicated regions. Matternet [48] is very good example of a company extensively testing MAVs to deliver medical supplies and other goods. The MAVs perform frequent flights and carry packages of up to 1 kg over 20 km to hard-to-reach rural clinics.

In September 2015, Foster&Partners, a global studio for architecture, design, and

engineering revealed plans to work on a pilot drone port in the Rwanda. The 'Droneport project', depicted in Fig. 3.3, offered to address the lack of infrastructure in Africa and provide UAS facility to support goods transportation and community life.



Figure 3.3: A glimpse into the future, with the DronePort project, appeared in the United Nations Office for Project Services (UNOPS) annual publication [49]. The concept describes how drones and goods transportation could become the centre of community life.

E-Commerce delivery system in mega cities provides another long-term opportunity for using the MAVs. Nevertheless, at present state, this technology faces many challenges. Some major challenges include lack of sufficient autonomy and lack of efficient communication system for flying MAVs in city and rural environments. Another area where MAVs are extensively used is for the Dull, Dirty and Dangerous (3D) scenarios where autonomous systems are used in places dangerous for human beings to access.

UASs and MAVs are disruptive technologies for many industries, but also present many threats to the infrastructure they operate in. In order for MAVs to be used in smart-city scenarios, the current city infrastructure has to provide for public safety and data security. Authorities need to consider a lot of other factors to unlock the growth of small transport UAVs. Presently, the following are the main regulatory barriers:

- Public safety;
- Airspace safety;
- Wireless communication licences; and

- Licence to operate civilian UASs.

'The Law Library of Congress', a global law agency has surveyed UAS regulations of twelve countries including the European Union in April 2016, see [50]. They assess extensively the legislative proposals and provisions for UAVs with respect to its weights, altitudes and types of use. Survey shows that the integration of UAVs into the National Air Space (NAS) is a major concern. The EU and many other countries have developed a regulation road map to enable the 'Remote Piloted Aircraft Systems' within their airspace. Other governments are also considering to roll out regulations as early as possible to build infrastructure towards harnessing the benefits of MAVs.

3.4 Summary

This chapter gave a general categorisation of MAVs based on their features and specifications. In addition, this research compiled a collection of UAS remote sensing applications specifically leveraging MAVs and showing the breadth of opportunities for smart cities and villages. These applications have specific technical requirements in terms of network, communication infrastructure and algorithmic data processing. These specific challenges, and needs, do justify the need for further research, in order to get a deeper understanding of the UAS architecture. Discussion in the chapter also highlighted the research requirements to advance, support, and enable technologies around MAVs.

Chapter 4

Distributed UAS Frameworks and Communication Technologies.

4.1 Introduction

The first part of this chapter surveys existing literature on distributed UAS frameworks that applies the concept of centralised control. The chapter starts by discussing the organisation of a MAV's network. The few works found on distributed UAS relate mainly to larger UAVs and more complex UAS. Extending on these, the relevant concepts that can be applied to civilian MAVs and simpler UAS are highlighted. In the second part of this chapter, the relevant middleware and communication frameworks that can be applied to the field of robotics are discussed. This chapter evaluates the possible communication technologies and protocols, which can meet the needs of networked MAVs. In this respect, the transport layer of ROS and the IoT protocols are discussed.

4.2 MAV networks

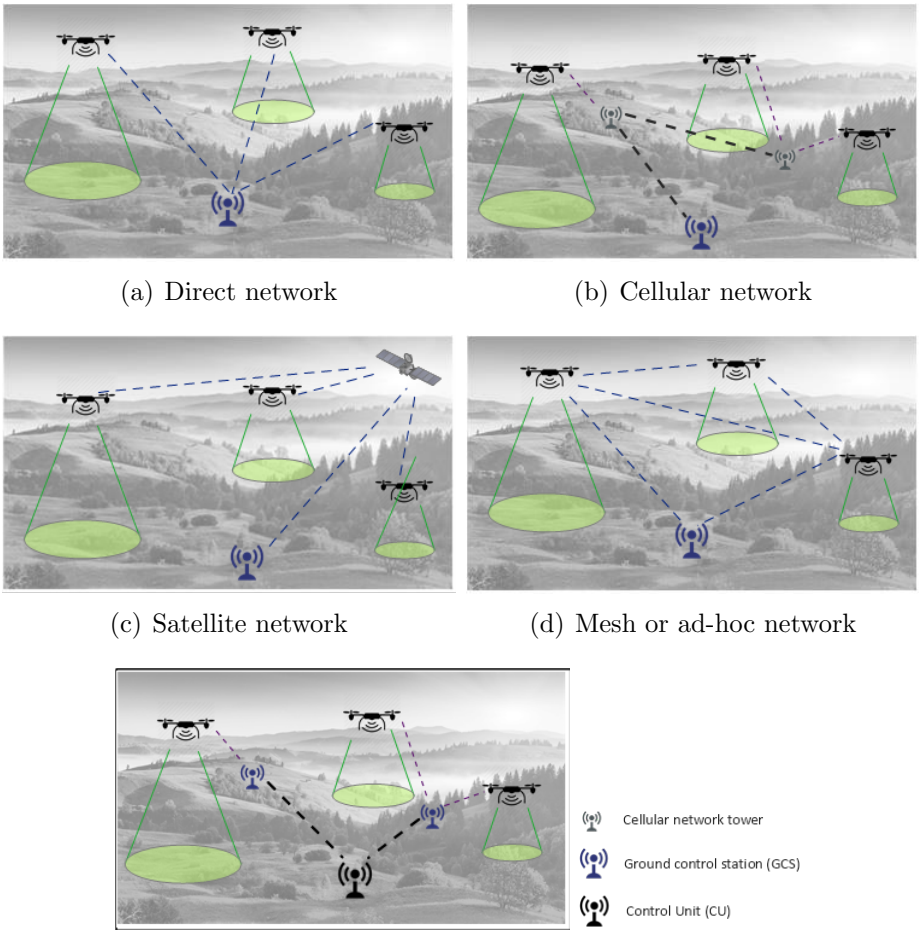
There are two broad categories of network topology that can be applied to autonomous UAVs:

1. Infrastructure-based networks consist of nodes that require a central access point. For instance, a single MAV connected to a base station through a wireless network. Consequently, for the infrastructure-based network the information of each MAV is processed by a control unit. The use of satellite communication is another example of infrastructure network. The majority of UAS applications employ infrastructure-based network with fewer MAVs where air-to-air connection is either not feasible or not needed. However, such multi-MAV systems might restrict the MAVs to cooperate in several tasks. Another drawback is the control unit can be the source

of the central point of failure.

2. Infrastructure-less networks involve point-to-point, ad-hoc formation of networks between MAVs. In this case, the nodes do not require a central access point. For infrastructure-less networks, cooperation between MAVs helps in building a complex application. In a recent work, [51] investigated an ad-hoc vehicular network known as 'Flying Ad-hoc Network', which in general does not rely on pre-existing infrastructure. It is formed spontaneously as the nodes connect, and addresses collaboration between the MAVs and the control stations.

Fig. 4.1 depicts the layouts of MAV communication infrastructure which are based on one or a combination of the aforementioned concepts. The systems in a) Direct, b) Cellular, c) Satellite, and d) Mesh, show the common multi-MAV communication network layouts which transfer data using the radio frequencies and other long-range wireless networks; whereas e) Distributed networks architecture, shows a network topology that relays data from the GCS to the control unit.



(a) Direct network (b) Cellular network

(c) Satellite network (d) Mesh or ad-hoc network

(e) Distributed network

Figure 4.1: Unmanned aerial systems communication networks.

MAVs can form more complex networks based on mesh and star topology. Consequently, engineers face specific challenges to operate emerging UAS networks. In [52], the important issues in UAV communication networks are highlighted. These networks are characterised as slow, unreliable and sometimes subject to dynamic organisation. In an interesting study, [53] discussed the dynamic routing for flying ad-hoc networks, whereas [54] and [55] presented different frameworks for formation of fixed-wing MAVs and UAV swarms respectively. According to [52], the routing demands of UAV networks will certainly go beyond the needs of the current 'Vehicular Ad-hoc Networks'. While wireless networks of MAVs present many opportunities, it involves other challenges related to wireless transmission links, data transfer rate, communication range, and embedded device capabilities. In [56], different network topologies– including ad-hoc multihop networks– were analysed to investigate the factors of UAV system which influences the data transfer rate and range.

This dissertation motivates for a distributed UAS architecture and considers that many of the future UASs will depend on architecture decisions which can address the network and communication challenges mentioned. The discussion which follows focuses on the communication technologies that could facilitate deployment and management of network services. One way to try to achieve this is through the use of flexible and robust communication middleware and messaging protocols. The same problems can be looked at from a network routing perspective; in this case, the dynamic networks of robot agents can use the concept of Software Defined Networking (SDN) to manage network behaviour programmatically via open interfaces. The aim in the latter case would be to address the challenges in network communication and to abstract low-level network components. In addition, SDN has the ability to increase security and availability in MAV networks. The work presented in [57] demonstrated an application of SDN and OpenFlow, in conjunction with the Data Distribution Service.

Another concept which is also closely linked to distributed UAS is the concept of multi-MAV control or swarming. Even if this work does not cover swarming, the literature on this subject is of importance as multi-MAV cooperation is a logical extension to the distributed framework and messaging infrastructure presented. Future trends and some studies in the last decade have been centred around the control and networking of multiple MAVs. [58] showed what MAV capabilities could be used to form automated routes and network of different types in conjunction with ground, sea and other air vehicles. The authors give a deeper insight on vehicular networks and multi-hop communication requirements of multi-UAV coordination.

4.3 Distributed UAS frameworks

Since this research applies the distributed computing paradigm to UAVs, it is important to survey the frameworks developed in this field and related technologies which are applicable to a network of smaller UAVs. The most distinctive feature of a distributed UAS is the decoupling between the MAV messaging client and the control unit. A centralised

control unit is hence essential to achieve distributed control of MAVs.

The Cloud paradigm is a key technology that can bring many benefits to distributed UAS architectures. Cloud robotics architectures has attracted a growing body of literature, in addition to an undeniable commercial interest. Cloud robotics is driven by the possibility for distributed mobile agents to be part of a broader ecosystem of Internet applications in order to access a huge knowledge base. The other main advantage is the capability to leverage the extensive computing resources available on the cloud. A decentralised UAS control unit can be deployed on more than one server instances and configured based on the computing power requirements of the applications and bandwidth needed. The architecture of the cloud can include components like a load balancer, which distributes workloads to a more reliable system. Use of the disaster recovery mechanisms like active-active or active-passive configurations of server instances could increase availability and reliability of UASs. Auto-scaling is yet another important feature provided by cloud-based services, allowing instances to be provisioned as and when the load increases. This research does not utilise a clould framework; however the distributed UAS concept can be easily extended to any private or public cloud infrastructure.

There have been few studies that describe the concept of distributed UAV control in the military domain of larger and more complex UAVs and UGVs(Unmanned Ground Vehicles). However, there is very limited information on implementations of distributed UASs in the civilian domain. Therefore, more detail is needed on the distributed architecture and messaging infrastructures relevant to MAVs. This dissertation assessed the characteristics and approaches of several contributions from academia on distributed UAS and their respective communication mechanism. These contributions can be roughly categorised into the following:

- **A middleware based** UAS and simulation framework was presented in [59]. The autonomous distributed system was based on the commLibX/ServiceX middleware, which is a specific and non-mainstream middleware technology;
- The CORBA **Service Oriented Architecture(SOA)** communication standard (developed in 1991) was used to design a distributed software architecture for an aerial robotic system (WITAS project) in the research work presented in [60]. Additionally, the work in [61] proposed a layered architecture model and service abstraction of interconnecting systems with UAVs;
- In [62,63], their results were based on SOA frameworks where the focus was mainly on embedded avionics and low-level electronics components' communication. In [63], a 'Virtual Autopilot Service' was proposed. This system interacts with mission and flight services like flight planning, mission management/monitoring, and contingency management. These works require a customised software/hardware components and they are not practical for off-the-shelf quadrotors and fixed-wing aircraft that come with predefined communication standards;
- **A publish/subscribe messaging pattern** framework for UAS operations was published in a report by the US Department of Defence, see [64, 65]. The middle-

ware based architecture uses the Data Distribution Service (DDS) protocol to address UAVs integration within military applications. This framework was meant to address the integration challenges between different UAVs, as well as other ground systems. Civilian MAVs have a distinct set of requirements; nevertheless, they also share similar integration needs as the control architectures discussed above; hence, motivating for a distributed architecture and centralised unit of communication; and

- **Integration within the National Air Space(NAS)** is one critical area regrouping various challenges and concerns for integration of MAVs into segregated and non-segregated air space. For instance, [20] investigated various challenges UAS faces to access non-segregated airspace. The authors also gave a description of regulations and design of interacting components of a system-of-systems involving UAVs. [66] outlined a reference software architecture with key functionalities such as control, conflict detection, and situation awareness for integration of UAVs.

4.3.1 Multi-MAV systems

As discussed, multi-MAV systems depend on a solid distributed framework for the C3 (command, control and communication). Another interesting feature of multi-MAV systems is the ability to cooperate and coordinate actions to perform an operation. This feature is often know as 'swarming' of MAVs; it involves a MAV operating as the master and communication backbone between other MAVs. A common scenario is when the master MAV is always connected to the GCS and coordinate with all the agents to perform a mission. A multi-agent network involving MAVs poses many challenges related to the underlying communication and networking between the agents. Building reliable communication among the multi-MAV systems and network is hence imperative to designing a UAS.

Many smart-city applications of networked robots are time-sensitive; hence, they require instantaneous or minimal delay in communication and data transfer. Examples of such algorithms are localisation and mapping algorithms, used for navigating and mapping unknown or GPS-denied environments. Another example is an object detection application where the robots react dynamically based on the available sensor data. Thus, it can be concluded that efficient communication among multi-MAV systems is essential for building real-time and near real-time applications when there are strict latency requirements.

Other applications within smart cities that involve MAVs, do not have strict requirements for communication lag or data delay. A delay-tolerant network (DTN) paradigm in [67] has been used to show how MAVs re-connect sensors to a network in network-constrained scenarios. The drones essentially act as relays in the network. DTN is presented to address the limitations of telecommunication infrastructure to support mobile aerial vehicles and sensor communication. The study in [67] also identifies potential bottlenecks with the Wide Area Networks (WANs) to handle large number of nodes and

data.

The collaboration, coordination, mission planning and control of multiple robots is no simple task. The framework presented in [13] describes a layered protocol and architecture to support coordination among the members of a MAV team and with Unmanned Ground Vehicles (UGVs). The system enables robots with different abilities to contribute to complete a high-level task. The protocol layers consist of an application layer, which is designed to cater for the specific requirements of managing and prioritising tasks among the heterogeneous robots. Search and rescue operations during calamity or disaster are examples of practical instances where this could be applied. Another example of mission that uses collaboration are the missions of finding and recognising a target. These missions may be executed and coordinated using a team of aerial robots.

Protocol Layer	Purpose	Examples and technologies proposed
Access layer	Radio technologies for communication between robots.	Adaptation module (to provide standard access to framework) 3G, 4G, Wi-fi, and others.
Internetworking layer	Ad-hoc network communication .to support the application-level protocol.	AirGround protocol with following properties: *neighbour discovery*, *leader election support*, *distributed task assignment.*
Object-to-Object layer	Cooperative execution of a task.	*Unicast & Multicast* network transmissions of messages.
Application Layer	Mission and tasks configuration.	Missions: *Area exploration*, *Search person* Tasks: *Search target*, *Reach target*, *Follow Target.*

Table 4.1: Summary of the layers of the mission-oriented protocol framework as presented in [13].

Table 4.1 highlights salient features of the communication protocol presented in [13] and the responsibilities of each layer of the architecture stack. For instance, within a team of MAVs, air-to-air communications paves the opportunity to apply the leader-follower algorithms to navigate drones. The control node primarily acts as the mission controller and dispatches information to the team of drones. The work in [58] proposed a multi-UAV network architecture for cooperative Air-to-air (A2A) and Air-to-ground (A2G) communications. The study suggested the use of UAVs as relays to support ground vehicular networks. The authors applied the concept of multi-hop networking and evaluate the challenges, performance and reliability of A2A, A2G, A2A-A2G (multi-hop) with respect to the IEEE 802.11a specification for sensor data transmission and ZigBee for control. The device specific hardware and software characteristics often hampers the mechanism to implement drone-to-drone communication. Furthermore, the architecture and application requirements for designing mission with air-to-air coordination is by far more demanding and difficult to achieve without custom built units.

4.4 Distributed robots communication

MAV applications rely heavily on the underlying communication frameworks. For example, UASs require robust telecommunication capability based on cellular network or radio tower. In the previous section, the distributed architecture and some of the communication aspects of UASs and multi-MAV systems have been discussed which could involve both wireless and wired networks. In this section, we focus deeper into the technologies and protocols of the communication. A UAS requires handling real-time situations. This is driving the need for more reliable, low-footprint communication frameworks involving MAVs within the context of mobile robots. Previous published works have focused on the cooperative control, task planning, reliability, and fault tolerance of the communication. However, only few research works have focused on improving the distributed robotic systems' communication. Therefore, we assess in more detail the relevant robotics and alternative communication technologies.

4.4.1 Robot Operating System (ROS)

The advent of advanced embedded devices like autonomous robots has added more complex requirements to real-time control and monitoring of robotics applications. ROS open-source middleware has been the most common development platform for large scale robotics software. It has been built to operate on multiple computers distributed in a network. Each node can function independently in a hybrid peer-to-peer architecture after initiating the connection with the help of the *Master* node, see [68].

ROS provides for two topic transport protocols:

1. **TCPROS** is a simple, reliable and ordered communication stream. With TCP, lost packets are resent.
2. **UPPROS** is a transport protocol in ROS where packets are not guaranteed to arrive or can be duplicated. It has the advantage of low latency and often suitable for lossy WiFi or GSM modem audio/video transmission.

ROS achieves a distributed communication using asynchronous message passing mechanism over *Topics*. RPC-style synchronous communication is also possible between nodes via a *Service*. Nodes and nodelets serve as application modules with two primary communication mechanisms based on XML-RPC (Remote Procedure Call) and publish/subscribe protocols. Fig. 4.2 shows the information flow of the aforementioned communication mechanisms inherent to ROS.

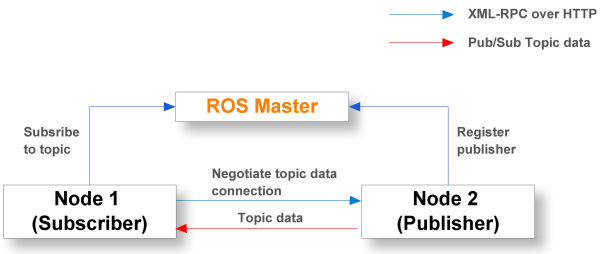


Figure 4.2: ROS nodes initialise connection with intervention of the master and exchange data directly using publish/subscribe mechanism.

ROS disadvantages

ROS has been a state-of-the-art operating system in the research domain and comes with many advantages such as being modular, loosely coupled, multi-language, and able to communicate between ubiquitous device components or process information in parallel. However, one of the main drawback of ROS is that it is not a real-time framework [69]. A real-time software provides a set of guarantees within the time that certain operations take to complete, e.g. inter-process communication, process scheduling and Input/Output. While ROS can be coupled with a real-time OS or process, it only provides a 'best-effort' timing during robot operations. Secondly, ROS is only really compatible with Unix-based platforms. Third, ROS is often deemed too heavy to run embedded and resource constrained devices. Lastly, ROS is unable to work efficiently in network restrained environment with limited sources.

To address these disadvantages, ROS is planned to undergo a major upgrade to "ROS2" [70] and utilise the Data Distribution Service (DDS) as transport middleware. With the growing popularity of robots and their use in commercial domain, an upgrade to ROS is mandatory. ROS2 is still in Beta version and aims to target multiple DDS middleware implementations, multi-platform systems, and various programming languages.

The main reasons given by ROS2 contributors [70] for starting a parallel release of ROS2 rather than just improving on ROS1 can be summarised as follows:

- Support for multi-robot systems involving unreliable networks;
- Support for real-time control; and
- Cross-platform support.

4.4.2 Internet of Things (IoT) publish/subscribe messaging protocols

Robots have evolved much in the last decade and consequently the middleware and communication technologies need to keep pace with this evolution. The emerging robotics applications consist of ubiquitous services consumable from applications over the Internet. This has given way to the "Internet of Robotics Things" concept that aims to provide access to the shared resources like processes, memory, big data, and machine learning, as described in [71]. While, the preferred communication pattern is Publish/Subscribe, there are several other requirements of the IoT that have given rise to messaging protocols with a richer set of features to support various levels of quality assurance, as well as meet the needs of both robots and networks. Furthermore, the IoT applications and services can consist of a number of mobile agents such as autonomous robots, cars, and drones. Therefore, researchers from academia, industry, and government institutes are giving special attention to messaging technologies for the vehicular communication networks of autonomous vehicles.

The works focused on IoT in [16, 17] describe multiple abstraction of the IoT layers and architecture models. The three-layered model, summarised in Fig. 4.3, is a high-level model generalised from literature on IoT architectures. IoT often uses low-energy Wireless Sensor Networks (WSNs); consequently, such wireless communication needs to manage a trade-off between data rate, range, and energy consumption.

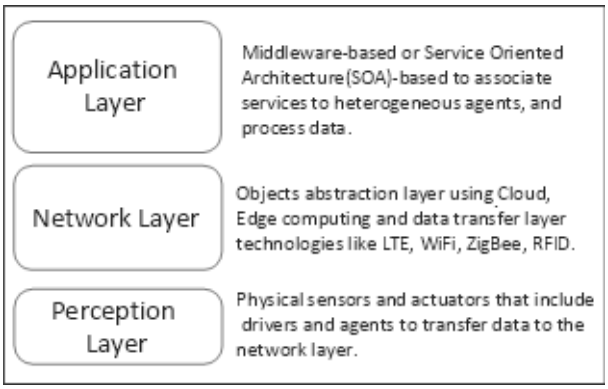


Figure 4.3: Three-layered IoT architecture model and design considerations as presented in [4].

Publish/Subscribe messaging protocols

Publish/Subscribe (Pub/Sub) messaging technologies are considered a derivative of the Message Oriented Middleware (MOM) system communication infrastructure between distributed systems. The protocols based on Pub/Sub have been used extensively in enterprise applications (e.g. IBM MQ), in research platforms like ROS as described in previous section, and many others. However, the proliferation of the IoT provided a rapid boost to the number and quality of these messaging technologies.

The competition and complexity of the IoT protocols has brought along many challenges. Engineers face major difficulty in achieving convergence and flexibility because of the wide range of IoT application architectures. From a technical perspective, the major difficulty lies in defining the architecture and standards that should govern the IoT system of systems. Researchers are working to address logical concerns such as security and privacy issues. The major operational concerns include connectivity and scalability of the IoT systems. A successful IoT architecture should be a loosely coupled system. Additionally, the communication should be asynchronous and provide recovery mechanisms in the event of failures or high load.

Standardisation of IoT Pub/Sub communication protocols has been a recurrent topic within stakeholders and governing bodies like the European Commission, see [72]. In [73] a comparative study was done by Prismtech; a company specialised in providing system solutions for Internet of Things applications. The study assesses DDS, MQTT, AMQP, JMS, REST, and CoAP IoT communication standards with respect to requirements of industrial IoT applications. In another interesting work, [14] used the event-based programming paradigm of MOM protocol AMQP for self-management automation systems. In another work by [15], the evaluation of two common IoT protocols namely AMQP and MQTT over unstable and mobile networks is presented.

Table 4.2 outlines a comparative study of relatively important features for each of these IoT standard protocols. The assessment also shows limitations of these technologies to support some types of use cases.

	Transport	Paradigm	Connectivity	Auto-discovery	Security	Fault Tolerance
MQTT	*TCP/IP*	*Publish/Subscribe*	*M2S*	*No*	*TLS*	*Using load balancer.*
DDS	*UDP/IP and TCP/IP (Unicast + Multicast)*	*Publish/Subscribe, Request/Reply*	*M2M M2S S2S*	*Yes*	*TLS, DTS, DDS Security*	*Decentralised*
CoAP	*UDP/IP*	*Request/Reply (REST web service)*	*M2M*	*Yes*	*DTLS*	*Decentralised*
AMQP	*TCP/IP*	*Point-to-point message exchange*	*M2M M2S S2S*	*No*	*TLS*	*Implicit and explicit*

Table 4.2: Qualitative comparison of MQTT, DDS, CoAP and MQTT according to [14–17] .
M2M: Machine-to-Machine, M2S: Machine-to-Server, S2S: Server-to-server

The characteristics of MOM communication standards protocols like MQTT, CoAP, AMQP and DDS are very compelling for robots to communicate within a network. This is because some of these protocols are primarily designed to be lightweight and run on constrained networks (other than HTTP) and devices with low computation power and memory. Furthermore, these protocols are open standards providing asynchronous publish-subscribe communication, but have very subtle differences that must be assessed before being selected as possible implementation choice for UAS communication. These protocols also provide hierarchical topic naming scheme that would allow access to send

and collect data to and from a MAV.

One of the main feature of IoT protocols is the various levels of assurance when messages are delivered over unreliable networks; this is referred to as Quality of Service (QoS) levels of the communication technology. HTTP protocol is established for client-server communication on the Internet. However, it does not have neither a broadcast mechanism nor a built-in support for QoS. HTTP also requires a web-server to accept data, thus, requiring more computation, which in turn is a scarce resource for robots and embedded systems. Both MQTT and DDS are very lightweight, with minimal headers, packet routing information and data packaging semantics. This research work compares the **MQTT and DDS** communication standards. Both protocols have a small footprint, making them applicable to low-powered devices, however differ in implementation and semantics.

MQTT (Message Queue Telemetry Transport)

MQTT [21] is a protocol for M2M and IoT applications with the following key features:

- 256 Mb maximum message size;
- Uses TCP/IP for fragmentation, transmission ordering and reliability;
- Reliable storage and forward mechanism; and
- Network efficiency, since there is no polling from the server.

MQTT, as its name implies, is suitable for transporting telemetry data from sensors. Being lightweight, it is suitable for WSNs (Wireless Sensor Networks) and wired TCP/IP networks. There is notable differences in the functionalities of MQTT and DDS, as depicted in Fig. 4.4. MQTT uses a Publish/Subscribe mechanism using a broker; therefore, it is optimised for applications running on distributed servers. MQTT can also achieve M2M communication using the classic model i.e. communicating via a central server, in this case the communication broker.

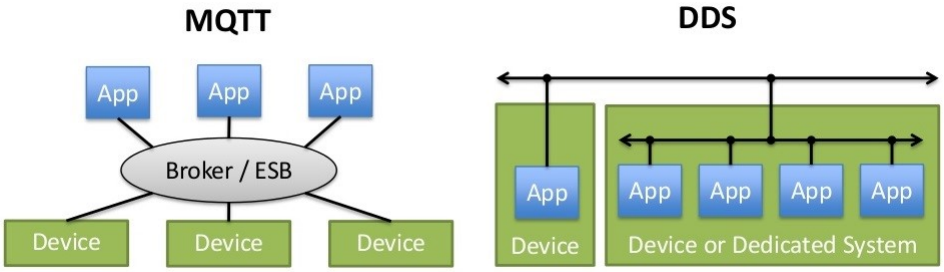


Figure 4.4: MQTT and DDS communication models.

MQTT for Sensor Networks (MQTT-SN):
While MQTT assumes a TCP/IP network and takes the advantages of TCP without the need to reinvent the wheel, MQTT-SN does not require the connection oriented transport provided by TCP. It is adapted for wireless networks based on UDP. ZigBee[1].

Besides the fact that MQTT-SN makes MQTT possible across a non-conventional network, there are interesting features of MQTT-SN, e.g. auto-discovery and registration of topics that can enable nodes to form a dynamic network topology. These features can allow mobile sensor platforms like MAVs to form networks as they move into the range of other gateways. However, the MQTT-SN specification has not been standardised like MQTT. Furthermore, MQTT-SN is mainly designed for small messages and networks with higher risk of link failures, hence, it might only be useful for telemetry/air-to-air links, rather than for high rate of larger image packets size coming from vision sensors.

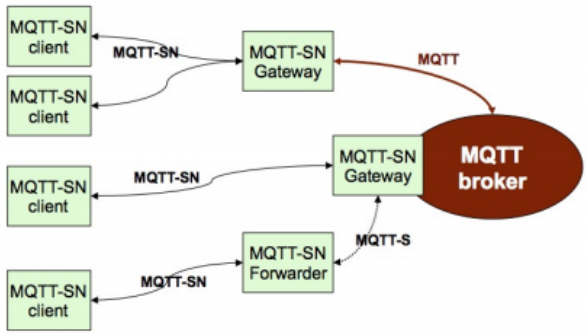


Figure 4.5: MQTT-SN architecture and specification (Copyright IBM Corporation 1999, 2013.). Clients can connect to gateways dynamically based on the range or when they become available. Gateways can also help with load sharing.

Data Distribution Service (DDS)

DDS is a feature-rich standard that adopts a data-centric approach to alleviate development efforts. Fig. 4.6 gives a good overview of the DDS framework according to the standard specifications. The implementation of the DDS framework can be found in few air and ground vehicle operations, like the UAS Control Segment for military UAVs, see e.g. [65]. The decentralised protocol eliminates the need for a centralised message broker. While MQTT uses essentially the machine to server communication pattern, DDS focuses on M2M messaging model; hence, it is optimised for distributed processing of data. Furthermore, it provides important additional properties such as:

- M2M Peer-to-peer communication between devices with different hardware and networking capabilities;

[1]ZigBee specification is an appealing type of Wireless Personal Area Networks (WPANs) used by low-power devices, which require higher speed and range of data transfer. It is based on the 802.15.4 IEEE wireless network communications protocol. Zigbee networks can use XBee radios to transmit data

- Very high and predictable performance to scale for large-scale deployments;
- Data-Readers and Data-Writers are automatically matched using **Dynamic Discovery**; and
- Rich set of QoS to control existential, temporal and spatial properties of data.

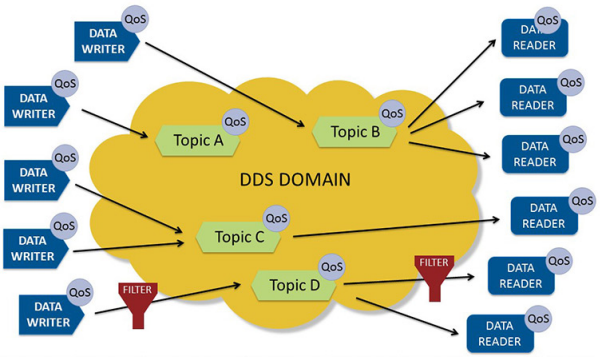


Figure 4.6: QoS-controlled data-sharing communication. Subscriptions can specify time and content filters and get only a subset of the data being published [5].

4.5 Summary

This chapter gave a formal description on the distributed UAS with respect to other prevalent types of networks. It also presented works specifically relating to the civilian UAS involving MAVs, and proposed the need to consider broader literature on the UAS control systems. The chapter concludes by discussing features of client-server technologies and communication frameworks available for designing the architecture of MAVs. The characteristics of IoT networks and the features of publish/subscribe protocols were also surveyed.

Chapter 5

MAV Autonomous Navigation

5.1 Introduction

This chapter presents an overview of components of an autonomous navigation system for a MAV. The discussion aims to introduce the navigation system implemented in [9]. This system comprises of a setup that consists of an extended Kalman filter, a monocular SLAM algorithm and a PID controller. This setup makes it possible for the robot to navigate in an unknown environment. First, this chapter gives an overview of the state-of-the art visual SLAM and filtering techniques. Then, the chapter introduces the algorithms used in implementation of the system.

5.2 Robot mapping and localisation

Researchers define the problem to map an unknown environment and simultaneously computing robot localisation as **Simultaneous Localisation and Mapping (SLAM)**. In this section, we discuss on the two tightly coupled concepts of robot navigation on which the SLAM problem is based: mapping and localisation. Often associated to chicken-or-egg problem, a map is critical for estimating the robot location and inversely a pose estimate is needed to build a map.

The robot's motion, execution, and planning depends on its ability to create a model; in other words, to formulate a map of the surrounding environment. Therefore, environment mapping by the robot is vital for autonomous navigation. Clearly, the environment perceived by the robot needs interpretation; there are many available techniques to enable this representation, such as 3D points from sensors like the laser range-finder or visual sensors like monocular or stereo cameras. Robot mapping has been extensively researched in the field of mobile robotics. Basically, a robot creates a topology or geometric map using its on-board sensors. Subsequently, the resultant map is used by the robot to navigate the area. The map can also be used as a working environment. The

mapping problem can be defined as the ability of the robot to gradually discover and map its environment. The on-board sensors are utilised to perceive points of interest, which are then integrated and related to each other.

The robot ensures consistency in integration of all the spacial points by precisely computing its new position with respect to the previous point. Therefore, localisation is necessary to accurately estimate the robot state variables in an unknown environment and relate its location to new points of interest. The work presented in [4] proposed the following techniques to achieve localisation:

- **Global Positioning System** (GPS) is an external method to localise a robot; however, often GPS is not available or not precise enough;
- **Odometry** or **Inertial Measurement Unit** (IMU) provides incremental motion displacement; however, this tool is subjected to drift over time and accumulation of errors; and
- **Feature recognition** is a recognition technique, where landmarks are identified and then used to determine object's translation.

5.2.1 Simultaneous Localisation and Mapping (SLAM)

From the perspective of a computer vision specialist, SLAM finds its roots from the Structure From Motion (SFM) algorithms. While the focus of SFM is popular mainly for **offline** 3D reconstruction of scenes, SLAM tries to build upon the same concept of multi-view geometry of SFM to reconstruct a 3D model based on visual or range sensors. The key difference in SLAM is the need for the robot to determine the camera position in **real-time**.

SLAM forms the basis of most autonomous navigation systems. It is crucial for a range of indoor and outdoor, air and underwater applications for both manned and unmanned vehicles. These systems face the same problem of simultaneously building a model of the environment on one hand, and estimating the position of the robot on the other hand. These two operations, that is, building the environment and estimating position, are inseparable. In other words, these operations are tightly coupled.

To navigate autonomously in an unknown environment, the SLAM algorithm needs to associate the data coming from the robot's sensors to a global map, which is continuously updated during the learning process. The general idea is quite straight forward: basically, the map of the environment is generated, and then it is used to re-estimate the position of the robot after regular time interval. Following this straight forward procedure, two questions are answered by the robots: 1) What does the world looks like?, and 2) Where is the robot?

The mathematical framework for SLAM is well established. However, there exists

many open issues related to complexity of dealing with large, dynamic environments and data association.

The general probability distribution for a SLAM system is represented by Equation 5.1. This process is described using the following variables: the **vehicle state** x_k, for all time steps k along its path given by the set $\mathrm{X}_{0:k} = \{\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2, .., \mathrm{x}_k\}$. Besides the state, the **map** m is also estimated during each iteration of the SLAM process. The new state and map m are dependent on the landmark observations $\mathrm{Z}_{0:k}$ and history of control inputs $\mathrm{U}_{0:k}$, as well as the initial state of the vehicle x_0.

$$p(\mathrm{x}_k, \mathrm{m} \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k}, \mathrm{x}_0) \tag{5.1}$$

There are essentially two forms of SLAM: full and online, see [74]. Equation 5.1 refers to the **full SLAM**; it estimates the posterior over the full robot path. The **online SLAM** only estimates the current pose of the robot x_k, rather than the entire path. Online SLAM algorithms are incremental and derived from the Markovian model. The *Markov chain* is a stochastic process, where the state x_k is assumed to be complete in its temporal evolution, i.e. it is conditioned by the previous state and control. Similarly, the measurement probability is conditioned on the current state.

This work focuses on the **online SLAM**. The aim is to **compute an estimate of the vehicle's location** x_k **at time step** k **and a map of the environment** m. Hence, a joint posterior over these two variables are computed by applying Bayes theorem.

Probabilistic SLAM methods like the Extended Kalman Filter (EKF)-based SLAM, calculate the probability of the robot location and landmark positions using data from prior robot location, controls, and landmark observations. The mathematical models for the filtering-based SLAM system have been widely researched and presented in academic works.

The following **mathematical notations** are used in this section and in the subsequent chapters to represent system variables and coordinates:

- x_k represents the state vector describing the location and orientation of the MAV;
- m is the map representation of the environment, composed of landmarks;
- u_k represents the control vector to move the robot from state x_{k-1} to x_k;
- z_k represents the measurement vector of a landmark at time step k;
- $\mathcal{W}, \mathcal{C}, \mathcal{K}$: calligraphic upper-case letters are used to denote coordinate frames. $\mathcal{W}$ is the world coordinate frame. $\mathcal{C}$ is the camera frame and $\mathcal{K}$, the keyframe coordinate which is camera-centred;
- M, N italic capital letters are used to denote sets, where M is the collection of landmarks on the map and N, the set of keyframes during the SLAM process;

Fig. 5.1 illustrates the essential SLAM problem for a robot moving on a path while sensing landmarks as presented in [6]. Both the trajectory of the robot and the location of all landmarks are estimated in real-time, without the need to have a prior knowledge of the location.

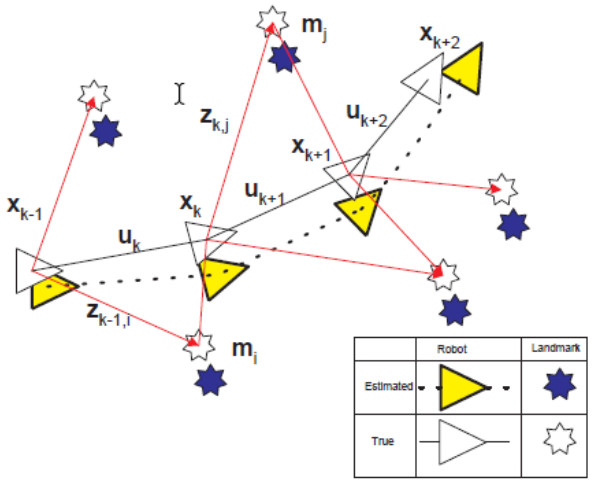


Figure 5.1: The essential SLAM algorithm problem [6]. The *True* locations are not known, hence the need to estimate the robot and landmark locations simultaneously.

Visual SLAM tracks the motion of a camera/robot while it moves in a natural environment, where image features serve as landmarks. The need for the features to be distinctive and recognisable from different viewpoints is an essential dependency for efficient feature tracking. In Fig. 5.1, at the starting point, it is assumed that the robot's uncertainty is zero. If the robot makes one observation at each step, the following can be used to describe the sequence of operations for each frame:

- Predict how the robot has moved;
- Measure by observing new features; and
- Update of the internal representations;

The first action of prediction is based on the motion model and vehicle kinematics. This activity logically increases the uncertainty of the robot position. In the subsequent step, the robot uses its camera to observe surrounding features. The new positional uncertainty results from the combination of the measurement error and the robot pose uncertainty. Finally, the robot updates its state representation, gradually correlating the features observed to the map and estimating the robot position.

1. A **state transition model**, represented in Equation 5.2, to obtain the new state, given the known vehicle location for the previous time step and control applied.

$$p(\mathrm{x}_k \mid \mathrm{x}_{k-1}, \mathrm{u}_k) \tag{5.2}$$

2. An **observation model** of the sensor readings, given the vehicle location and location of landmarks in the map are known, represented in Equation 5.3.

$$p(\mathrm{z}_k \mid \mathrm{x}_k, \mathrm{m}) \tag{5.3}$$

A key part of SLAM is its ability to define the *robot pose* relative to an external coordinate system, after perceiving features in its environment. Hence, in probabilistic robotics, the concept of belief is a probability distribution over the true state variable x_k. The posterior probability, before incorporating a measurement, is known as the **prediction step**; this is represented by Equation (5.4). Subsequently, the **measurement update step** after a measurement or observation is being made is described via Equation (5.5). The SLAM algorithm (excluding the map update here for simplicity) is then implemented recursively in a two-step sequential prediction and update steps.

$$\overline{bel}(\mathrm{x}_k) = p(\mathrm{x}_k \mid \mathrm{z}_{1:k-1}, \mathrm{u}_{1:k}) \tag{5.4}$$

$$bel(\mathrm{x}_k) = p(\mathrm{x}_k \mid \mathrm{z}_{1:k}, \mathrm{u}_{1:k}) \tag{5.5}$$

The overall SLAM process shown in Fig. 5.2 needs to address various sub-problems namely: managing uncertainties, estimating state, and associating data. As discussed, solutions to the SLAM problem rely on a probabilistic framework; the robot pose correction and map fusion are modelled with Gaussian noise variables to represent and handle uncertainties. In this respect, the Kalman filter is used for linear systems and the extended version of the Kalman filter is applied on non-linear systems.

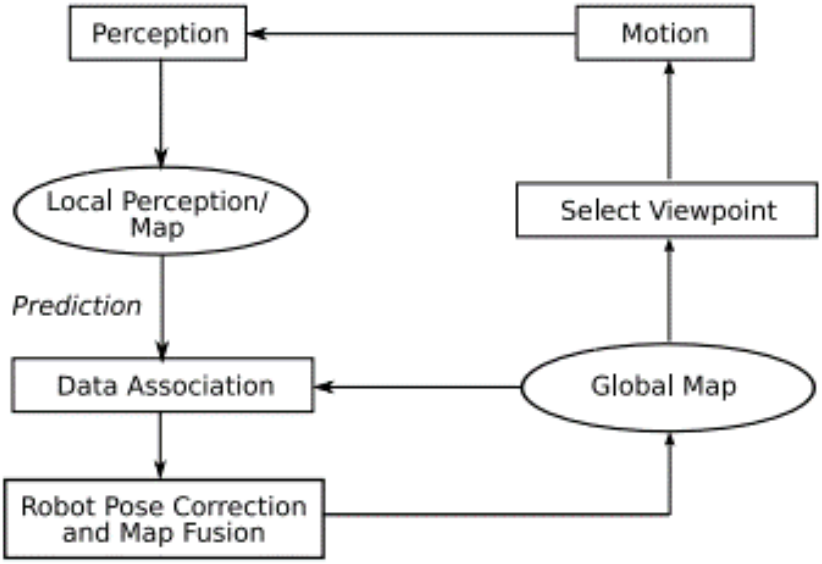


Figure 5.2: SLAM process as proposed in [4]. Here, following an observation, the local perception or map has to be associated with the global one in order to update both the robot pose and global map.

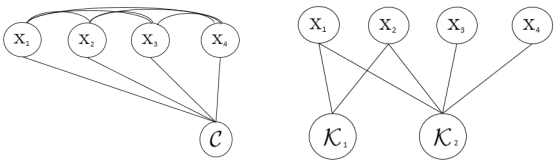
The **EKF-SLAM** is based on non-linear Bayes filtering and considered a well-known algorithm to implement SLAM. The results obtained from the prediction and the update of the location posteriors and map are approximated using an EKF. It is worth mentioning that the EKF technique have one main disadvantage, which is the quadratic increase in the computation complexity associated with the number of landmarks. This is due to the fact that landmarks and positions of the vehicle are jointly computed; therefore, the co-variance matrix of x_k is updated for addition of each new landmark. Obviously, this limits the number of landmarks and its ability to navigate in large environments. This limitation, however, was addressed using the so-called Rao-Blackwellised particle filter and **FastSLAM** as reported in the algorithm presented in [75].

Keyframe-based SLAM algorithm

The aforementioned discussion shows that tracking and mapping are two important inter-related processes in visual SLAM. Prior to key-frame based methods, EKF-SLAM and FastSLAM were widely used methods to update the current camera position and location of all landmarks. In these cases, besides increased computation complexity, bad data association lead to catastrophic consequences; to overcome this limitation new robust mechanisms were developed which would prevent corruption of the map.

The Keyframe-based methods like the **Parallel Tracking and Mapping (PTAM)** is based on a monocular SLAM system, and has been argued to perform better than filtering-based SLAM methods in certain situations, see [76]. Keyframe-based methods execute the two processes of tracking and mapping in two separate threads. Instead of using the Bayesian filtering method of EKF-SLAM, keyframe-based SLAM relies on an efficient Bundle Adjustment (BA) step for pose and map representation. The visual representation in Fig. 5.3 compares the map for both methods.

Referring to Fig. 5.3, in the case of a) filtering-based, the map is a fully connected graph of landmarks, whereas in the case for b) keyframe-based, a selected number of correlations between landmark positions and keyframes are maintained as well as the uncertainties.



(a) (b)

Figure 5.3: a) EKF-based SLAM uses a filtering based technique where all tracked points are fully correlated among each other; b) keyframe-based SLAM methods links keyframes to a set of landmarks and links serve to denote observations.

The entire algorithm for keyframe-based SLAM can be represented into the following

three steps: 1) Initialisation, 2) Tracking, and 3) Mapping. In the SLAM algorithm which is both keyframe-based and vision-based, the robot starts at an assumed location; therefore, it requires the initialisation stage. An initial map localises the robot in the unknown environment. This initial map is obtained by using a stereo technique that baselines the two frames with respect to a translation movement as shown in the Fig. 5.4.

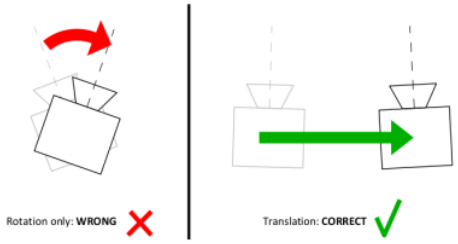


Figure 5.4: PTAM initialisation process. The initialisation process needs a baseline to function properly. To provide this, the camera need to move sideways between the first two keyframes.

After initialisation, the algorithm tracks and maps landmarks, and updates the robot pose in continuous loop by identifying new landmarks and incrementing the map. In this process, the robot pose is updated by tracking the 3D point features in each acquired image frame. Another way of representing the tracking process is through the association of the localisation to the camera position $\mathcal{C}$, with respect to landmarks x. The Bundle Adjustment (BA) process, which is described shortly after, performs a batch optimisation of observations.

The initialisation and tracking processes include generation of the image pyramid of the keypoints in each frame, containing four levels of intensity at different resolutions. The keypoints are identified using the FAST algorithm corner detectors. FAST corner detectors use a technique where the keypoints can be identified at various scales. The keypoints identified at different pyramid levels. This method makes the system more robust to scale changes as well as to resolution changes due to motion blur, i.e. erratic shifts and rotations in camera frame. This robustness is owned to the state of low resolutions, which in turn reduces the impact of motion blur.

In the tracking process, the camera position is also computed based on the perspective-n-point (PnP) technique and on the 3D-to-2D coordinate mapping. The **camera pose and re-projection** process transformation from the global coordinate $\mathcal{W}$ to camera coordinate $\mathcal{C}$ is given by (5.6) and (5.7), where:

- p_{jC} is the camera coordinate of the jth point on the map m, in the form $p_{jC} = (x, y, z, 1)^T$;
- E_{CW} is a square matrix denoting the transformation from frame W to C. It is composed of a rotation and translation component;
- p_{jW} represents the coordinate of a patch in the world coordinate frame;

- u_i and v_i represents the image point coordinates projected from the camera frame; and
- $CamProj$ is the camera projection function to calculate image points.

$$p_{jC} = E_{CW} p_{jW} \tag{5.6}$$

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = CamProj(E_{CW} p_{iW}) \tag{5.7}$$

Using the new pose location estimated by (5.6) and (5.7), an accurate location of the camera is computed at a higher resolution of the pyramid. If tracking is lost, the system will attempt to re-initialise the process and compute a new pose estimate.

After each successful observation which tracked a set of keypoints for successive frames during the PTAM process, the local perception or map has to be associated with the global one. This is done in order to update both the robot pose and global map. The mapping process as depicted in Fig. 5.5, is the process that maps the landmarks and optimises the relationships. This step is important since more landmarks are continuously added. Throughout this process, the*maximum likelihood* approach can be used to minimize the error between landmarks and camera locations. This minimization technique is based on the so-called Levenberg-Marquardt iterative technique as proposed in [77], which is also known as *global bundle adjustment*. The main disadvantage of this technique however, is that it is computationally unfeasible to do this for all landmarks and camera positions. In order to overcome this disadvantage, the concept of *local bundle adjustment* was proposed. In this case, a subset of the keyframes is matched with new keyframes using epipolar search. If a match is found, the landmark is not added; but if no matches are found, the landmark is added to the map and to the corresponding camera location. The keyframe-based SLAM process includes further refinements, such as outlier rejection of landmark observations.

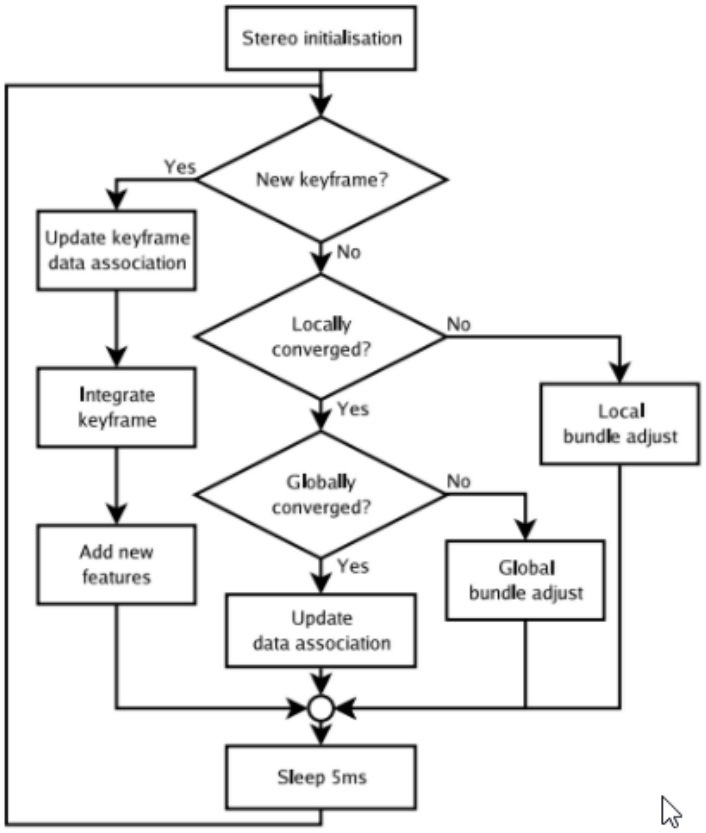


Figure 5.5: PTAM mapping process flow as in [7]. The input is keyframes computed from the camera image frames.

5.2.2 Bayes filters and related models

As discussed earlier, SLAM requires a probabilistic framework to manage uncertainties in estimating and mapping the robot's position. Given the past measurements and sent controls, the **Bayes filter** is a general framework for recursive state estimation and represents belief of the robot by a posterior distribution over the state of the environment and the robot. The Bayes filter aim is to get an accurate estimate of the system state variables by using the data from multiple sensors and knowledge of the system dynamics.

The Bayes filters estimates the state x of a system at time t, given by:

- observations z; and
- actuating variables u over previous time steps.

This is represented by the following probability distribution:

$$p(\mathrm{x} \mid \mathrm{z}, \mathrm{u}).$$

In practice, this eventually involves application of Bayes rule to define a recursive algorithm to integrate sensor observations and control data. Applying Markov assumption, where the current 'state of the world' is known, there is no requirement to handle all the previous states and only previous state of the system at time $t-1$ is needed. The posterior distribution of the state of the Bayes Filter is represented as $p(\mathrm{x}_t \mid \mathrm{u}_t, \mathrm{x}_{t-1})$.

The Kalman Filter (KF) and Extended Kalman Filter (EKF) are the two common variants of Bayes filtering techniques employed in the field of robotics. Both the KF and EKF are members of the *parametric filters* family, described in [78]. The Kalman Filter is an optimal estimator for linear systems whose noise is modelled by a Gaussian distribution, i.e. the noise is assumed to be a multivariate normal distribution, parametrised using the *moments parametrisation* mean μ and covariance Σ.

The Kalman filter algorithm can be summarised as follows:

1. Prediction step:
$$\overline{bel}(\mathrm{x}_t) = \int p(\mathrm{x}_t \mid \mathrm{u}_t, \mathrm{x}_{t-1}) bel(\mathrm{x}_{t-1}) d\mathrm{x}_{t-1}.$$
The term $p(\mathrm{x}_t \mid \mathrm{u}_t, \mathrm{x}_{t-1})$ is actually the *motion model*, denoting how the robot moves from time t to $t-1$. The predicted belief is given by integrating the motion model and the previous belief of the system $bel(\mathrm{x}_{t-1})$.

2. Correction step:
$$bel(\mathrm{x}_t) = \eta\, p(\mathrm{z}_t \mid \mathrm{x}_t) \overline{bel}(\mathrm{x}_t).$$
The sensor or observation model is represented by $\eta\, p(\mathrm{z}_t \mid \mathrm{x}_t)$ used to estimate the corrected belief x_t, from the prior predicted belief, where η is the normalisation constant to obtain a probability density distribution.

In the case of the linear Kalman filter, the following is assumed:

1. Firstly, the prediction and the states are modelled by
$$p(\mathrm{x}_t \mid \mathrm{u}_t, \mathrm{x}_{t-1}) \tag{5.8}$$
$$\mathrm{x}_t = A_t \mathrm{x}_{t-1} + B_t \mathrm{u}_t + \epsilon_t \tag{5.9}$$
where A_t and B_t are the system and input matrices, respectively, x_t and x_{t-1} are the state vectors, u_t is the control vector, and ϵ_t represents independent Gaussian noise, i.e. $\epsilon_t \sim \mathcal{N}(0,\, Q_t)$. is a multivariate normal distribution with 0 mean and covariance Q_t.

2. The second assumption of the linear Kalman filter is that the measurement probability are represented by
$$p(\mathrm{z}_t \mid \mathrm{x}_t) \tag{5.10}$$

$$z_t = C_t x_t + v_t \tag{5.11}$$

,

where z_t is the measurement vector, C_t is the output matrix describing how to map state x_t to observations z_t. Lastly, v_t represents the Gaussian noise distribution where $v_t \sim \mathcal{N}(0,\, R_t)$. and R_t represent the covariance.

3. The final assumption of the linear Kalman filter is that the initial belief, $bel(x_0)$ has to be normally distributed. The aforementioned assumptions help to ensure that the posterior belief $bel(x_t)$ is also Gaussian at any time t.

On the other hand, the EKF approach is applied to non-linear systems, with the state and observation vectors given by (5.12) and (5.13). These two equations are the non-linear versions of equations (5.9) and (5.11) respectively. ϵ_t and v_t represent **process noise vectors**, with 0 mean and a certain covariance matrix recomputed at every step.

$$x_t = f(u_{t-1}, x_{t-1}) + \epsilon_{t-1} \tag{5.12}$$

$$z_t = h(x_t) + v_t \tag{5.13}$$

In the case of **the EKF** model, the belief $bel(x_t)$ of the robot at time t, is no longer expected to be Gaussian [78]. Therefore, the functions f and h need to be linearised using the first order *Taylor Expansion*. This enables the EKF to calculate a Gaussian approximation of the true belief of the robot, represented by the means and covariances. The steps and mathematical equations involved for the EKF are discussed below.

For EKF-based SLAM algorithms, once the extraction of landmarks and the data association from SLAM are executed, the filtering process below is applied. This process can be described in the following three steps based on the inputs (x_{t-1}, P_{t-1}, u_t, and z_t), where P_{t-1} is the covariance for the previous belief of both the robot pose and landmarks:

1. **Odometry update**:
   Let $\hat{x}_{i|j}$ be the estimate of x_i using the observation information up to and including time j. Then, the current state is estimated using the odometry data. This is given by the *motion model* defined for the vehicle dynamics. The following two equations are used to model this process:

$$\hat{x}_{t|t-1} = f(x_{t-1|t-1}, u_{t-1}). \tag{5.14}$$

$$P_{t|t-1} = F_{t-1} P_{t-1|t-1} F_{t-1}^T + Q_{t-1}. \tag{5.15}$$

   As is the case for the Kalman filter, the first steps of the EKF is prediction of the next state to obtain $\overline{bel}(x)$ from the previous state x_{t-1}, covariance P_{t-1} and controls u_t. The covariance P represents the degree of correlation between the variables at discrete time steps. It contains the variances of robot and landmarks positions.

F is the state transition model. The state transition matrix is a partial derivative (from first order Taylor expansion), known as the *Jacobian*.

The following Equation (5.16) is an example of a motion model f for a robot modelled by location variables (x, y and ψ)^T, with respect to a global world coordinate frame.

$$f(x,u) = \begin{pmatrix} x + (cos(\psi)\dot{x} - sin(\psi)\dot{y})\delta_t \\ y + (sin(\psi)\dot{x} + cos(\psi)\dot{y})\delta_t \\ \psi + \dot{\psi}\delta_t \end{pmatrix} \tag{5.16}$$

The model is obviously non-linear because of the trigonometric nonlinear functions; the Jacobian F is then given by the following differentiating function:

$$F = \frac{\partial f(\mathrm{x}, \mathrm{u})}{\partial \mathrm{x}} \tag{5.17}$$

2. **Observation update**:
   This step consists of computing a new state from the landmarks observation and tracking process. This process is also called the *innovation process* and determines the difference between the estimated robot position and the new robot position.

   Each landmark observation is also associated with a certain uncertainty. This eventually involves the following steps; first the innovation residual y_t and covariance residual S_k are computed. Then, the $KalmanGain$ K_t is calculated as follows:

$$K_t = P_{t|t-1} H_t^T S_t^{-1}. \tag{5.18}$$

   The Kalman gain represents the degree of uncertainty in the predicted belief and observations. H_t is the Jacobian matrix for the observation model, while I is the identity matrix. The updated state estimate and covariance estimate are computed using the Kalman gain, as defined in equations (5.19) and (5.20):

$$\hat{\mathrm{x}}_{t|t} = \hat{\mathrm{x}}_{t|t-1} + K_t y_t. \tag{5.19}$$

$$P_{t|t} = (I - K_t H_t) P_{t|t-1}. \tag{5.20}$$

   This part is also called the *measurement update*, which incorporate the sensor(s) reading to update the state mean and covariance.

3. **Landmark update**:
   The estimated location of new landmarks are added to the current list of landmark locations. The correlations between previously observed landmarks and new landmarks are also stored.

A key point to note on the computation complexity of the EKF, is that the computational complexity of the prediction step is linear with respect to the number of landmarks, whereas it is quadratic with respect to the number of landmarks in the observation step.

The full steps and proofs of the KF and EKF algorithm in robotics navigation are beyond the scope of this study. They are explained in great detail in [78].

The EKF has been a common state estimation technique used in the field of robotics. This is primarily because of its computation efficiency. The particle filter on the other hand, the computational time grows exponentially with respect to the dimension of the state vector x_t. The limitation is the degree of non-linearity at the point of approximation using Taylor expansion. Bayes filters can be implemented using different techniques such as the Unscented Kalman Filter, Information Filter, Histogram Filter, and increasingly popular Particle Filter. Each technique makes certain assumptions on the robot initial belief, state transition, and measurement probabilities. The following properties are key in evaluating and choosing a specific technique:

- Computational complexity;
- Accuracy of the approximation; and
- Ease of implementation.

We can classify three main filtering approaches applied to SLAM algorithms, as well as several variants for each one of them [78]:

1. **Kalman Filter** and **Extended Kalman Filter** approaches are suitable for feature-based map. The difference being for *online SLAM* only the current estimate of the robot is maintained, whereas in *full SLAM* the full path of the robot is computed recursively [78] [79].
2. **Particle Filter** based approaches are similar to the parametric filters (KF/EKF). It uses a sampling approach based on Rao-Blackwellized, montecarlo or other random number generation techniques. As discussed in the previous section, Fast-SLAM uses a particle filter to provide estimations on the robot locations.
3. **Graph based** optimisation techniques maintains a graph of nodes of each robot pose, as well as constraints between the nodes [80].

5.2.3 Sensors used for state estimation

State estimation for MAVs can be achieved by using the following kind of sensors. This section describes filtering methods applied in conjunction with these sensors:

1. **Monocular sensor:**
   The work in [81] demonstrated a MATLAB implementation of fusing vision and IMU data using EKF; however, it was only a simulation based demonstration, and

it might be difficult to apply for practical systems in real-time control. Alternatively, [82] and [83] implemented state estimation using the Monocular SLAM; however, the first research work only used a KF, arguing that the MAV platform already provides some levels of noise filtering, while the second research used an EKF implementation on the ROS platform.

2. **RGBD cameras** and **Range scanners like Lidars**:
   In [84] an RGB-D camera and laser scanners were used to show autonomous indoor environment mapping and generation of occupancy grid 3D maps. The work in [85] and [86] also showed similar methods for 3D dense mapping using RGB-D sensors.

3. **Kinect sensors:**
   The Microsoft Kinect sensing device and Kinect fusion library were used in [87] to localise a MAV and perform complex manoeuvring like gripping to a vertical wall.

4. **Stereo cameras:**
   Stereo cameras method was presented in the work of [88]. The presented method employed a Kalman-filter approach, assigning white Gaussian noise levels to 2D image acquired from stereo 3D images from two cameras. Later, [89] showed a stereo and motion fusion algorithm yielding a 6D representation of mobile robots positioning and motion.

5.3 MAV control

Control theory involves addressing the problem of controlling a set of variables of a dynamic system. The system set values, e.g. position and orientation, are controlled by adjusting one or more of the system actuating variables. Existing literature provides impressive results on the control techniques of MAVs. For example, [90] showed two agile quadcopters juggling with a ball, whereas [91, 92] are interesting publications on precise and aggressive control of MAVs and nano-UAVs. While the core motion of the aerial robot is based on the control architecture techniques, these types of precise movements are often achieved using external fixed camera systems like the Vicon.

This work employs a PID controller in the closed-loop. This approach is widely used in autonomous robot navigation consisting of SLAM algorithms and Kalman filters.

5.3.1 MAV control architecture and the PID controller

An aerial vehicle platform utilises multiple control techniques, applied based on the mechanics and aerodynamics of the vehicle's body. These can be summarised as the following stacked layers of controls:

- Motor speed controllers generating forces to move the MAV in a 3D space;

- Attitude control and state estimation, i.e. system controls are based on the motor speed and voltages applied to the quadcopter;
- Position control and localisation modules; and
- Way-points and trajectory following.

The motion process model is used within the Kalman filter to estimate the state of a moving robot after a certain time interval δ_t, based on the control u_t applied. At this level of the control architecture stack, calculating the control signal u, to reach a goal state or position, is the most common problem of input/output feedback control loops.

Fig. 5.6 shows a block diagram of a control system. Here, the controller output u_t is derived to influence the system state based on the previous state x_{t-1}. The error e_t is constantly updated based on the difference between the sensor measurements and desired values. In this example, a simple linear model was used to program the controller using the past state, control command, and a constant error in the system ϵ_t.

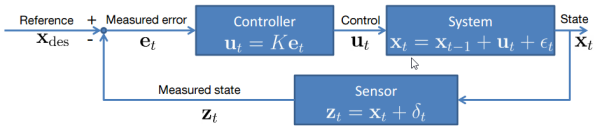


Figure 5.6: Block diagram of a standard input/output feedback control system. The goal or desired value x_{des} is constantly compared to the measured state x_t, to determine the error e_t, which in turn specifies the controller output u_t based on the control gain K..

The PID controller is a very common feedback control loop to influence the states of dynamic systems. The output $u(t)$ of the PID controller is given by the following Equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \tag{5.21}$$

where,

- t is the current time;
- K_p is the tuned parameter for proportional gain;
- K_i is the tuned parameter for integral gain;
- K_d is the tuned parameter for derivative gain;
- $e(t) = SP - PV(t)$ represents the error, calculated as the difference between the setpoint SP and process variable $PV(t)$; and

- τ is the variable of integration, from time 0 to the present time t.

The tuned coefficients K_p, K_i and K_d are obtained experimentally by trial and error or other heuristic methods. The aim is to determine the coefficients that minimises the error $e(t)$.

The value of the gain K_p influences the performance and robustness to noise of the system being controlled. The time delay in the system is another factor that requires consideration when applying control; incorrect account of delays can cause overshooting or oscillation to reach a desired value. For modelling the rigid body kinematics and dynamics, a differential component D is often needed to smooth the rate at which a body approaches a desired location i.e. it helps to dampen the oscillations due to errors. An integral term I can be added to compensate for steady state errors, such as the gravitational force and other disturbances in the system.

Fig. 5.7 shows how the PID controller parameters affects the response of a generic system for a step input. When the integrator gain is too large ($K_i = 2$), this causes overshooting or oscillation. On the other hand, when this gain is too low ($K_i = 0.5$), the control takes a long time to converge. In this example, the three controls ($K_p = 1, K_i = 1, K_d = 1$) were applied together, creating a fairly optimal tuning for the system.

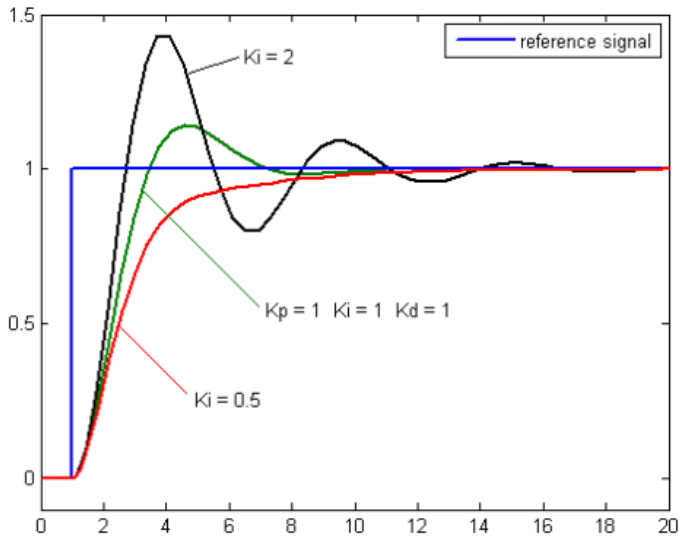


Figure 5.7: System response using various gains of the PID-controller, see [8]. The black, green and red graphs show the effect of applying different gains K_p, K_i and K_d for a step input.

The Parrot AR Drone2 uses a vector of four control command parameters for motion control as outlined in [83, 93]. The motion control vector is defined in (5.22) and expect values for the roll Θ and pitch Φ angles, as well as yaw rotational speed $\dot{\Psi}$ and vertical speed $\dot{z}$.

The controller running on-board uses the orientation angles and linear velocities values to adjust the motors speeds accordingly.

$$u = (u_\Theta, u_\Phi, u_{\dot{\Psi}}, u_{\dot{z}})^T \in [-1.0, +1.0]^4 \tag{5.22}$$

where,

- u_Θ represents the control for roll angle;
- u_Φ represents the control for pitch angle;
- $u_{\dot{\Psi}}$ represents the control for yaw angular velocity; and
- $u_{\dot{z}}$ represents the control for linear velocity along the z axis.

5.4 The autonomous navigation system

This section combines the methods defined so far to describe the setup of a system capable of navigating a MAV autonomously. Each subsection can be viewed as practical implementation of the aforementioned methods as used by the testbed system from [9]. The testbed uses the monocular SLAM algorithm PTAM from [7], an extended Kalman filter and a PID controller.

5.4.1 The monocular SLAM system

The state of the drone is a key vector as it is used by both the SLAM and Kalman filter. Fig. 5.8 shows both the drone and world coordinate systems. PTAM is a keyframe-based method for estimating the pose of the MAV, and it needs to translate from the drone coordinates to the world coordinate system.



Figure 5.8: The drone body (b) and world (w) coordinate systems. Φ, Θ and Ψ are the orientation angles: roll, pitch and yaw respectively.

As discussed in section 5.2.1, online SLAM estimates the posterior of the robot given by (5.23), which is obtained by integrating out past poses. x_t is the state at discrete time steps and m the map which is composed of a set of tracked features. In this case, the correspondences c_t is made explicit. In the traditional EKF-SLAM approach, maps are 'feature-based' [78], whereas PTAM maps keep a subset of previous observations related to *keyframes*. Each *keyframe* consists of a set of features or point landmarks and their positions in 3D space. At every step, each feature-set (keyframe) tracked are corresponding to previous features using a maximum likelihood probabilistic technique.

$$p(\mathrm{x}_t, m, c_t \mid \mathrm{z}_{1:t}, u_{1:t}) \tag{5.23}$$

The PTAM system has been defined in section 5.2.1 as a keyframe-based SLAM method. The system was first presented in [7]. Here, each keyframe is built using FAST corner detector as shown in Fig. 5.9. The initial map is based on a stereo technique using five-point algorithm and using RANSAC to estimate the essential matrix (relation between two images viewpoints) and triangulate the base map. Then, in a continuous loop new keyframes are tracked and added to the map using a method known as *bundle adjustment*, described in details through section 5.2.1.

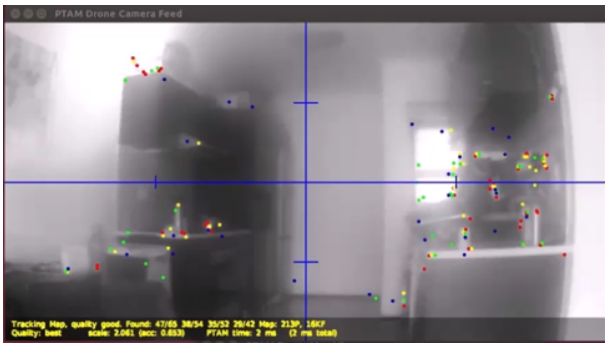


Figure 5.9: PTAM tracking from the camera feed. The colours blue, green, yellow, and red denote intensity of the FAST corners or how coarse the feature points were identified at: blue representing the coarsest and red representing the finest patches.

PTAM pose estimates

The visual SLAM system provides 6 degrees of freedom (DoF) pose estimations. However, PTAM pose tracking does not contain scale estimation. In [9] and [82], different methods for scale estimation using drift-free altitude measurements from the sonar were shown.

The video frames from the MAV monocular camera serve as an input to the PTAM implementation from [7]. The output of this process is a 6 DoF pose estimate of the MAV in the Euclidean space $\mathbb{R}^3$. The original PTAM implementation, like other monocular SLAM algorithms cannot determine the scale of the scene and the global coordinate.

The scale of the map is calculated by generating 3D sample pairs within each time

interval. It is based on two factors i.e. first, it depends on the distances measured by the SLAM system, and second, it depends on the available navigation sensors data i.e. integral over the horizontal speeds. As described in [9], the ultrasound height measurements are used to estimate a scaling factor utilising a maximum likelihood approach. This is required in order to translate the PTAM pose to the global pose. The resulting Equation 5.24, where the offset o is adapted for each value of the scaling factor λ is given by:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{world} = \lambda \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{ptam} + o \tag{5.24}$$

The MAV pose (combination of position and orientation) from PTAM is re-estimated at each video frame, as well as the scaling factor λ and offset o. Since the PTAM provides 6 DoF pose, the roll, pitch and yaw values are treated as direct observations, resulting in the following vector:

$$\begin{bmatrix} x \\ y \\ z \\ \Phi \\ \Theta \\ \Psi \end{bmatrix}_{ptam}$$

PTAM observation model

The state vector of the drone state is obtained by applying observation functions to the direct measurements of the drone sensors. There are two distinct observation sources, namely the PTAM and IMU, hence the need for two observation functions:

- $h_{ptam}(\mathrm{x})$, represents the PTAM observation function; and
- $h_{imu}(\mathrm{x})$ is the IMU observation function.

As discussed in section 5.2.2, for the normal Kalman Filter algorithm, the measurement vector is modelled mathematically by (5.25):

$$\mathrm{z}_t = H_t \mathrm{x}_t + v_t \tag{5.25}$$

where:

- $\mathrm{z}_t \in \mathbb{R}^3$ represent the measurement vector;
- H_t is a correction feedback matrix;

- x_t is the true state at time t; and
- The process noise is normally distributed and represented by v_t.

For the extended Kalman Filter, the matrix multiplication $H_t\mathrm{x}_t$ to compute the predicted measurement is replaced by function $h(\mathrm{x}_t)$ in (5.26):

$$\mathrm{z}_t = h(\mathrm{x}_t) + v_t, \tag{5.26}$$

where, h is non-linear mapping function, and hence will be different for each state, requiring a first order Taylor series expansion to linearise the subsequent changes in values over time.

The measurement vector for the SLAM source is represented as:

$$\mathrm{z}_{ptam} \in \mathbb{R}^3,$$

defined with respect to the drone body coordinate system. To obtain the measurement vector z_{ptam}, the observation function $h_{ptam}(\mathrm{x}_t)$ maps the current state to the expected measurement. This is described first by the equation (5.27) representing the 3D drone body positions, roll, pitch and yaw measurements; and second, by the equation (5.28) which denotes the mapping function to obtain the 6 DoF state vector with respect to the global coordinate frame.

$$h_{ptam}(\mathrm{x}) = (x_b, y_b, z_b, \Phi_b, \Theta_b, \Psi_b)^T \tag{5.27}$$

$$\begin{aligned} \mathrm{z}_{ptam} &= h_{ptam}(\mathrm{x}) + v_{ptam} \\ &= f(E_{DC}E_{C,t}) \end{aligned} \tag{5.28}$$

,

where,

- $h_{ptam}(\mathrm{x_t})$ is the observation from the PTAM, with the added noise v_{ptam};
- E_{DC} is a transformation matrix from camera to drone coordinate; and
- $E_{C,t}$ is the camera pose from the PTAM.

At each successful observation which tracked a set of points during the SLAM process from one frame to the other, the local perception or map is associated with the global one to update both the robot pose and global map. Fig. 5.10 and Fig. 5.11 shows the generated 3D PTAM map and the tracking points on the image respectively. The red crosses on the PTAM map are used to denote the landmarks already tracked as more are added over time.

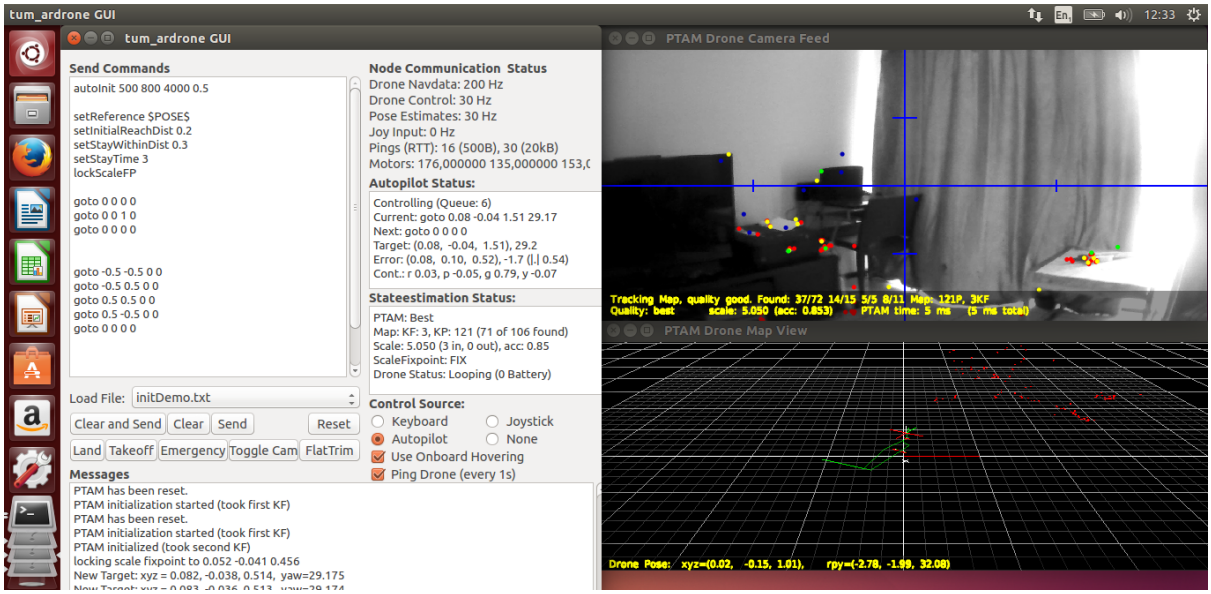


Figure 5.10: GUI for drone control, PTAM camera-feed/frame tracking(top right) and PTAM drone pose 3D map (bottom right).

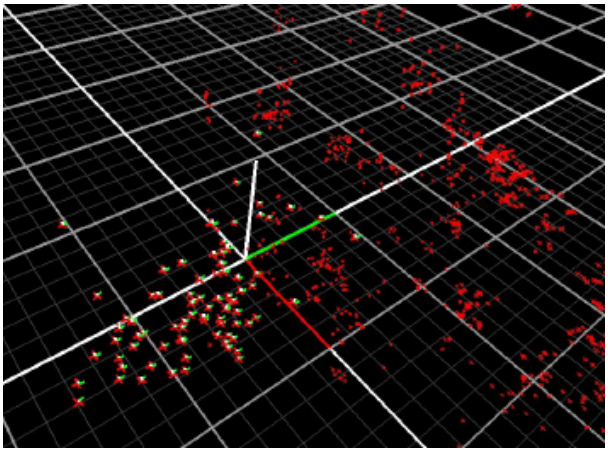


Figure 5.11: GUI for PTAM 3D map - The *Camera positions* are shown as red-white-green world coordinate frames, where-as red crosses denotes landmarks position.

5.4.2 The extended Kalman filter

The EKF estimates the drone state based on estimates provided by PTAM and IMU sensor measurements. As discussed in section 5.2.2 and in the previous subsection, this will require visual SLAM and sensor observation models, state transition model and control models to be defined, as well as time synchronisation among sensors.

The observation model calculates the expected measurement vector z_t based on the current state vector x_t. In the previous part, PTAM provided a measurement vector z_{ptam} based on the optical tracking from the observation function $h_{ptam}(x)$ ((5.27)). A

similar model was used for the IMU sensor, defining a measurement vector z_{imu} from an observation function h_{imu}.

The functions h_{imu} and h_{ptam} give the state space variables of the MAV x_t defined in (5.29):

$$x_t := (x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi})^T \in \mathbb{R}^{10} \tag{5.29}$$

The Parrot MAV sends sensor values, referred to as *navdata*, which are received at regular time interval of approximately 5 ms. The following velocities and orientation variables, obtained from the observation models, define the state of the EKF:

1. **MAV position:** x, y and z world coordinates; x, y positions are obtained from PTAM, whereas the height is measured using the ultrasonic altimeter which also provides for stabilised motion. However, this sensor value is not entirely reliable because of its height limitation (6 m) and it is subject to error over uneven ground [94]. Nevertheless, it provides the relative height measure of the MAV, used in [9], together with a maximum likelihood estimator to calculate the scale of the scene.
2. **MAV velocities:** $\dot{x}$, $\dot{y}$, $\dot{z}$. The Parrot AR Drone IMU computes the horizontal speed using a 60 FPS bottom-facing camera. The camera recognises the texture of the ground using an on-board optical-flow algorithm. However, current technical manuscripts do not provide insights o the method for on-board speed calculation [95]. From simple experiments, it was observed that the speed value (magnitude) deviates from the true value if the texture of the ground is poor.
3. **MAV orientation:** angles for roll, pitch and yaw Φ, Θ, Ψ, and yaw rotational speed $\dot{\Psi}$. These values are direct observations from the gyroscope and require to little pre-processing.

The state vector of the Kalman filter is given by (5.30). This vector consists of the global-coordinates (x, y, z), the velocities $(\dot{x}, \dot{y}, \dot{z})$, the roll Φ, the pitch Θ, the yaw angle Ψ, and the yaw rotational speed $\dot{\Psi}$.

$$x_t := (x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi})^T \tag{5.30}$$

Following from the methods described, the **state transition function** for a complete state update $x(t) + f(x(t), u(t)) \rightarrow x(t + \delta_t)$ is defined by:

$$\begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix} + \delta_t \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x}(\mathrm{x}) \\ \ddot{y}(\mathrm{x}) \\ \ddot{z}(\mathrm{x},\mathrm{u}) \\ \dot{\Phi}(\mathrm{x},\mathrm{u}) \\ \dot{\Theta}(\mathrm{x},\mathrm{u}) \\ \dot{\Psi} \\ \ddot{\Psi}(\mathrm{x},\mathrm{u}) \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix} \tag{5.31}$$

This state transition model essentially predicts the state for the next time step δ_t. The velocities are directly obtained from the IMU, whereas the accelerations in x and y directions is approximated based on the variables from the state x. The horizontal acceleration of the drone is given by $(\ddot{x}, \ddot{y})^T$, and it is approximated in relation to the different forces acting on the drone. This is based on the fact that a quadcopter accelerates by tilting its body in response to greater thrust forces applied by the rotors.

The drone's horizontal acceleration $\ddot{x}(\mathrm{x})$ and $\ddot{y}(\mathrm{x})$, are then given by Equations (5.32) and (5.33), correspondingly:

$$\ddot{x}(\mathrm{x}) = c_1(c_2(cos\Psi sin\Phi cos\Theta - sin\Psi sin\Theta) - \dot{x}) \tag{5.32}$$

$$\ddot{y}(\mathrm{x}) = c_1(c_2(-sin\Psi sin\Phi cos\Theta - cos\Psi sin\Theta) - \dot{y}) \tag{5.33}$$

where, c_1 to c_2 are the model constants of the EKF. These constant are determined and calibrated experimentally based on a simulation method as described in Appendix A.6. Here, the drone is assumed to behave in the same manner in both the x and the y direction.

The Kalman filter generates accurate state estimates of the MAV's pose after each video frame using these aforementioned models, together with the PID controller, i.e. with respect to the state and control commands applied. The control model parameters for $\dot{\Phi}, \dot{\Theta}, \ddot{\Psi}$ and $\ddot{z}$ are discussed in the next section 5.4.3.

The EKF has two other main purposes. First, it is used for initialising image tracking because SLAM needs a prior map to integrate new information. Second, the EKF is used as substitute for initialisation when the device loses tracking. Tracking is considered lost when the roll and pitch measures have high deviations, and in this case the keyframes are not added to the global map.

5.4.3 The PID controller

The PID controller is applied to the MAV system and uses the quadrotor rigid body kinematics to generate the control commands required to moving the system into a desired location. Control commands are generated by the extended Kalman filter at a rate of 100 Hz (commands per second). These commands are predicted based on the expected pose of the drone.

As presented in section 5.3, the Parrot AR Drone 2 uses the following vector of control commands to influence the motion of the vehicle. $(\overline{\Phi}, \overline{\Theta}, \overline{\dot{\Psi}}, \overline{\dot{z}})^T$. The following functions and variables are used for modelling the influence of a control command with respect to the current state:

$$\begin{aligned} \dot{\Phi}(\mathrm{x}, u) &= c_3(c_4\overline{\Phi} - \Phi) \\ \dot{\Theta}(\mathrm{x}, u) &= c_3(c_4\overline{\Theta} - \Theta) \\ \ddot{\Psi}(\mathrm{x}, u) &= c_5(c_6\overline{\dot{\Psi}} - \dot{\Psi}) \\ \ddot{z}(\mathrm{x}, u) &= c_7(c_8\overline{\dot{z}} - \dot{z}) \end{aligned} \tag{5.34}$$

where, c_3 to c_8 are the constants of the EKF which are determined and calibrated experimentally based on a simulation method described in Appendix A.6.

5.5 Summary

This chapter gave an overview of the SLAM problem and discussed the various approaches to solve it. In addition, this chapter compared the filter-based and keyframe-based techniques.

This chapter also provided a formal definition to the SLAM problem and the Kalman filter. PTAM was discussed as an improvement over the well-known EKF-SLAM method. However, PTAM was originally designed for small spaces and augmented reality applications; hence, it does not scale well in large environments.

Then, this chapter described MAV control techniques and how a PID controller is employed in a closed-loop to regulate dynamic systems.

Summarising all previous results, the last section explained how the methods have been implemented for the autonomous EKF-PTAM-PID system used as testbed in this dissertation.

Chapter 6

Research Methodology

6.1 Introduction

This chapter presents the proposed steps and methods of the main study of this dissertation. The experiments conducted are based on an empirical approach which collects data from a real-world MAV system. This work implemented a **distributed UAS prototype** and tested it in an indoor laboratory setup, which comprises of networked computers and MAVs to simulate a distributed UAS communication architecture, and the navigation testbed consisting of a SLAM algorithm, coupled with a probabilistic filtering method.

This chapter starts by describing the parts of the target systems, followed by a discussion on key requirements of the distributed architecture. Furthermore, this chapter also discusses different metrics which were applied to benchmark the communication, such as the performance and observation on the stability of the SLAM navigation system. Finally, this chapter describes the usage of the navigation testbed presented in Section 5.4, namely the EKF-PTAM-PID system.

6.2 Research approach and methods

This work uses an experimental approach to test the research hypothesis. Initial research hypothesis states that a distributed UAS architecture can provide the required communication technology for an autonomous navigation sub-system. In this respect, a UAS control architecture needs to address several technical issues. This research makes an effort towards addressing some of the key technical challenges related to the implementation of distributed autonomous systems of MAVs. First, the experiments test the feasibility and quality of operating the navigation system under the proposed constraints. Second, the performance and scalability of the communication are analysed in a controlled environment, simulating a distributed scenario constituting of more than one MAV flying with a set of on-board sensors.

The proposed system design abstracts the complexity of the UAS by hosting the autonomous navigation sub-system on a server. This decoupled client-server architecture of an autonomous system aims to address issues related to its scalability, computing power, performance, and reliability requirements. While there are many benefits of implementing a distributed UAS architecture, the control unit adds another layer of communication. This additional layer of communication requires the development of additional data transfer software components, in order to enable communication between the GCS and control unit. In this context, first, the overall system is tested with ROS as the end-to-end communication and application middleware, with TCPROS used as the default messaging protocol. Second, this work assesses two main protocols employed in communication technologies; particularly, the MQTT and DDS. Furthermore, there is the need to carefully account for additional delays induced in the system, by the distributed communication infrastructure.

This research proposes the use of the autonomous navigation system implementation from [9] as a testbed. The study in [9] consisted of a set of experiments with a low-cost quadcopter and a ground-based laptop to show a visual SLAM system. This mechanism localises the MAV by tracking landmarks in the environment. In this research, the navigation sub-system is executed within the UAS control unit; hence, from the context of an autonomous navigation system, this research work describes the details of an UAS architecture and communication components. Consequently, the research method involved the following steps to evaluate the system:

- The TCPROS, MQTT and DDS protocols were evaluated for the latencies in data communication;
- Comparative view of different implementations of the communication gateway were analysed;
- Communication latencies were measured for different scenarios such as constrained bandwidth and number of MAVs; and
- Overall stability of EKF-PTAM-PID navigation system was evaluated based on the quality of the sensor input data;
- The SLAM algorithm accuracy was assessed with the EKF managing the delays and sensor uncertainties.

It is worth mentioning that this thesis primarily uses experiments and critically analyses the experimental data obtained from the prototype system. The overall system architecture is a simulation of a real-life UAS architecture for low-latency robotics applications. This can also be categorised as the hardware-in-the-loop simulation with MAVs connecting to a Ground Control Station and system evaluated under different scenarios.

6.3 UAS system requirements

This section discusses the system components required for designing the proposed distributed architecture. The design of UAS can vary based on the size, capability and number of MAVs. In addition, the type of missions which include range, and types of controls (direct or distributed) also defines appropriate UAS design. Areas of particular focus include the network transmission and type of payload which have an influence on the system design. This work focuses on defining the **UAS requirements of real-time and near real-time autonomous systems to accomplish remote sensing missions**.

In order to provide viable real-life solutions, aerial robotic systems need to meet certain strict architecture requirements. Major operational concerns, related to UAS, include redundancy, fail-over mechanisms, scalability, performance, and security. If these concerns are left unattended, they can cause catastrophic failures. In the following section, we discuss system architecture concerns, describing the selection criteria and principles adopted to meet these requirements.

Performance:
A *resource constrained network* is characterised as the network that limits the data processing capability of a device. Bandwidth and throughput are two major factors that influence the network performance. UAS comprises of a mix of wireless and wired networks. Wireless networks are known to have limited resource in terms of energy and bandwidth; which affects the rate at which data is received from the devices. This leads to the need for communication between the ground stations and control unit to be a high-bandwidth link. Additionally, the messaging protocols need to be able to provide high throughput based on the total payload size arriving at each time-step.

Scalabililty:
A scalable architecture allows seamless addition of modular functionality or agents to the system. This provides an additional processing power or addresses the high load of data. Scalability is also required when the same information needs to be shared with several consumers. Both MQTT and DDS were designed to provide for scalable communication at different scales.

Persistence:
The storage of data is another requirement of MAV applications that often arise, that is in other words the ability to store high volume of data generated from different devices. The data at the control unit can be stored either on physical network storage, disk storage, or in-memory databases.

Computational power:
A system made of one or more robots produces large amount of data and processing all of this knowledge in a central system requires powerful computation capability based on the type of algorithms. For instance, the filtering technique used in robot applications can have significant impact on the required computational complexity.

Reusability:
In contrast to the traditional customised systems for a specific class application, a distributed multi-agent system model is generally based on reusable software components and framework. This ensures ease of implementation and maintenance of the software. The standard messaging protocols MQTT and DDS provide brokers and client libraries across multiple programming languages and communication between different processing units. This work adopts a generic approach for the gateway, allowing in-bound and out-bound channels of communication to be parametrisable in a *config file*. Fig. 6.1 shows an example of a parameter file for the control unit.

```
7   <node name="mqttCmdVelReceiver" pkg="ros_mqtt_bridge_cpp" type="mqttCmdVelReceiver"
8       <param name="loop_rate" value="200" />
9
10      <!-- Make sure the topics published are subscribed with the exact name -->
11
12      <param name="subscribedMqttTopic_land" value="/mqtt/land"/>
13      <param name="publishedRosTopic_land" value="/ardrone/land"/>
14
15      <param name="subscribedMqttTopic_reset" value="/mqtt/reset"/>
16      <param name="publishedRosTopic_reset" value="/ardrone/reset"/>
17
18      <param name="subscribedMqttTopic_takeoff" value="/mqtt/takeoff"/>
19      <param name="publishedRosTopic_takeoff" value="/ardrone/takeoff"/>
20
21      <param name="subscribedMqttTopic_cmdvel" value="/mqtt/cmd_vel"/>
22      <param name="publishedRosTopic_cmdvel" value="/cmd_vel"/>
23
```

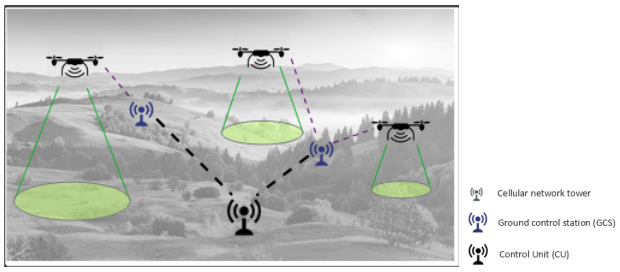
Figure 6.1: Extract of an example of the communication gateway parametrisation. *subscribed*** ROS topics are in-bound data coming from the navigation system and *published*** MQTT topics are out-bound communication to the control unit.

Delays:
Delays to transmit information is prevalent in dynamic systems, and they can cause oscillations, overshoots, or undershoots of the system behaviour. For example, a control system exhibits oscillations when an action is applied with a delayed or outdated information to define a target state. In a dynamic MAV system, delay or lag can result in an unstable system response, which directly affects motion control and response to real-time path planning. For example, delays occur in quadrotors due to network transmission, mechanical movement, and interference. Thus, it is often critical to effectively manage or account for information delays.

In a system that needs to control the motion of a MAV, wireless delays actually vary as the quadrotor flies or is impacted by interference. These disturbances are partly addressed by the Kalman filter and by the PID controller, which is employed in the closed-loop of the navigation system.

6.4 System design

Distributed computing is a model where a network of computers passes messages to trigger action, send request, or post information. The service-oriented pattern allows services to be defined in an interface that can be consumed by nodes in the network. Fig. 6.2 gives a high-level view of a distributed network of MAVs, with a communication framework allowing sharing of data.



(a) Distributed network

Figure 6.2: Distributed MAV communication network.

First, this study defines the logical architecture and system components required for the distributed UAS. The "UAS Control Unit" software platform is the central server running the navigation algorithms of the system architecture. The UAS control unit is based on the publish/subscribe messaging technology, and proposes a generic framework that allows mission control of current and emerging UAS programs. Fig. 6.3 is a logical design of the overall system, which illustrates examples of data processing modules.

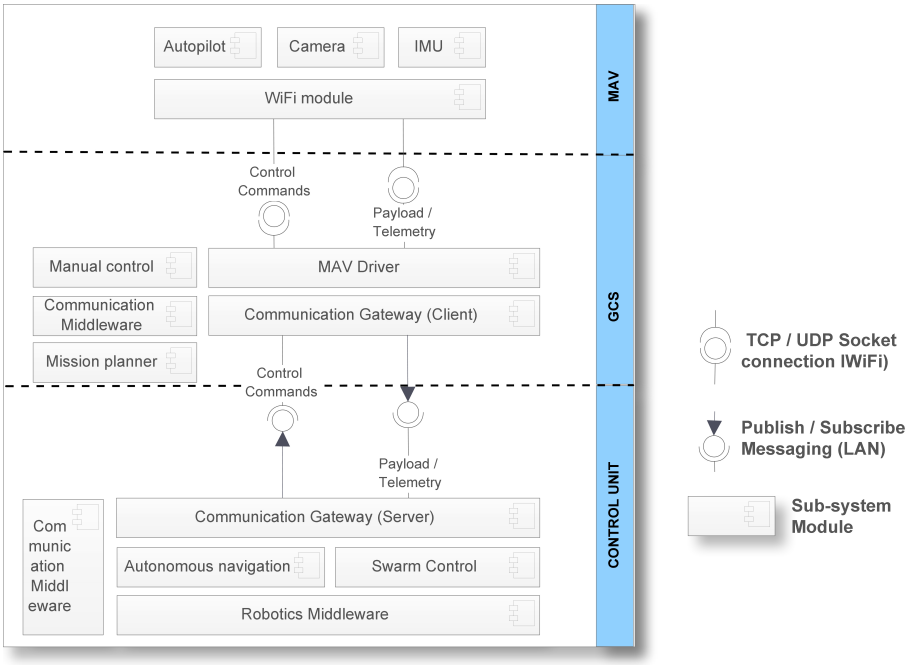


Figure 6.3: Logical architecture of the UAS distributed model and middleware platform to communicate with server applications.

In this research, ROS forms the basis of the middleware that connects directly to the control unit's gateway interface. This interface is kept independent from the software technology and operating system, such that other sub-systems can easily send and receive data. The overall system consists of two communication gateways: one on the client side and another at the control unit. These gateways are the main components of this distributed system with following functionalities:

- To act as a 'dummy' client that communicate with the MAV and bridges the connection to the control unit, making use of standard protocols and technologies; and
- To provide a standard set of service interfaces of raw MAVs data and control commands to the control unit.

Designing the communication gateways requires an understanding of the possible communication technologies and frameworks. The following section addresses this concern by describing the communication technology utilised in this research.

6.4.1 Distributed communication framework selection

The choice of communication framework depends on the type of Command, Control, and Communication: the C3. As illustrated in the survey published by [26], the following factors can influence choice of the communication mechanism:

- LOS (Line-of-sight) v/s BVLOS (Beyond-visual-line-of-sight) missions;
- Wireless network type, reliability, and bandwidth;
- Data encoding, security, and protocol;
- Standards and regulations; and
- M2M, M2G co-ordination requirements.

For commercial off-the-shelf (COTS) MAV platforms, the MAV specifications determine how much access does a developer has to the avionic components, payload data, and actuators. Furthermore, autopilots and smart equipment need to be accompanied with the proper library and MAV device driver SDK to enable the capability of receiving and sending data in the MAVs. This helps to develop software based on smart autopilots to solve real-life problems. For example, the Parrot drones uses the MAVLink protocol to transmit telemetry and image streams. MAVLink [96], which stands for Micro Air Vehicle Communication Protocol, is very lightweight, header-only message marshalling library for micro air vehicles. Initially released in 2009, MAVLink currently supports ten autopilots and can be used under the LGPL license. This protocol is primarily designed to facilitate integration with mission planning software like QGroundControl**??**, and it acts as a message marshalling library. Hence, it is not considered to be appropriate for distributed communication of data.

Another consideration to select the right framework is the nature of the communication between the distributed agents. For **multiple-MAV** scenarios, the MAV platform need to transfer specific data, e.g. its location, motion commands, tasks assignments, or basically just acts as a relay of sensor data, for example to form star and mesh networks. Formation of a distributed network depends on the communication links and application protocols. Two prevailing approaches are the use of Service-Oriented-Architecture (SOA) technologies such as CORBA and Remote Procedure Call (RPC). Both technologies were used in the work published in [60, 61]. SOA approaches like CORBA, REST, and SOAP use the *pull* model for inter-system communication, whereas modern systems essentially leverage the message-oriented-middleware (MOM) pattern based on asynchronous *push* mechanism that trigger event listeners.

The most recent studies that successfully used specifically designed middleware and protocols for UAV communication were presented in [13, 97]. These studies offer several benefits such as fully decoupled client/server configuration, asynchronous modes, and full duplex capabilities among others. However, in these architectures, the autopilot and other on-board avionics, as well as ground control station software were abstracted through a service bus. The communication gateways enable both M2M (mav-to-mav) or M2G (mav-to-ground) scenarios, showing a fully decentralised network of cooperating hardware and software. Similar approaches were found beneficial to distribute tasks, to communicate, and to perform collaborative missions using the multi-UAVs configuration.

6.4.2 TCPROS, MQTT and DDS protocols

TCPROS was introduced in section 4.4.2 as the underlying transport protocol between ROS distributed nodes. The navigation testbed used in this dissertation is built on ROS platform, but was developed to run on a single computer. With some simple configurations, it is easy to separate the MAV driver and core navigation system components on two computers. As discussed, TCPROS supports TCP/IP, which as result benefits from the TCP retry mechanism if packets are lost. This in turn ensures reliability on Ethernet network; however, UDP would be a better choice for wireless networks. One of the drawbacks mentioned is how to establish the connection, as an RPC synchronous call has to be made to the ROS master. Subsequently, TCPROS publish/subscribe asynchronous calls are direct between nodes. Besides, there are no additional QoS parameters that can be set. Finally, the obvious drawback of TCPROS is that ROS is needed on both sender and receiver.

This study looks at recent message-oriented-middleware (MOM) frameworks MQTT and DDS as alternatives. These service-based communication models emerged recently and they address the needs of IoT applications. Furthermore, this study explores benefits of these frameworks such as easier configuration of mission and lower cost of UAS operation. It is important to highlight that the communication gateway acts as a lightweight bridge and transmits bi-directional data. It can run from a laptop, a smart phone, or a MAV on-board computer, such as a raspberry-pi.

As explained in section 4.4.2, MQTT and DDS are standard-based protocols, implementing the distributed architecture pattern. They also offer the possibility to implement more robust communication failure mechanisms. In addition, they enable interoperability between the heterogeneous agents through standard implementation at the application and protocol level. Since MQTT and DDS have a smaller footprint, they have better bandwidth usage than competing protocols. They also maintain the session open whenever required by the process, thus greatly minimising resource requirements for IoT devices. These characteristics are essential for remote sensing missions, where sensor data arrives at a high rate and volume.

The level of QoS features ensure some level of reliability and assurance of delivery when required. Once again, a compromise is made based on the overhead that higher QoS levels will bring to the sending/receiving data rate and performance. A final aspect of designing the communication components of the system is the availability and scalability of services at the control unit; these are two major factors to enabling safe operation of UASs in real-life scenarios.

6.4.3 The communication gateways

The communication gateway should adhere to the following criteria:

1. It should be a lightweight component that bridges the communication to and from

the MAV.

2. It must efficiently handle high rate and volume of data in both unicast or broadcast transmission modes.

There is no significant processing and logic at the gateway; however, it is meant to fulfil multiple purposes as listed below:

- Address and route data;
- Enhance efficiency of computation and minimise its cost;
- Provide proper redundancy mechanisms if ever a link is down;
- Synchronise the time-series data;
- Time-stamp the data; and
- Encrypt/decrepit or compress information.

The system developed in this research was based on two options for the gateways, namely the MQTT and DDS communication gateways. Hereafter, we describe the gateway implementations for each option in more detail.

MQTT communication gateways

As defined in section 4.4.2, the **MQTT broker** centralise data processing between the publisher and subscriber agents. For the MQTT implementation of the gateway, the following MQTT brokers have been considered to provide for the CPU performance, scalability, and lowest latency possible:

1. Mosquitto [98].
2. HiveMQ [99].
3. JoramMQ [100].

A brief literature review indicates that **Mosquitto broker** is better than HiveMQ and JoramMQ to manage the high load of messages. A benchmark study was conducted by Scalagent [101], a software vendor of distributed technologies, where Mosquitto is shown to have gradual increase for a constant CPU usage with a high number of connected clients (100,000). The same benchmark study shows that the message latency for high number of publishers was better compared to other message brokers. Although our tests showed better latency for JoramMQ, with high number of messages (300 messages/s), JoramMQ frequently experienced out-of-memory after short intervals. On the other hand, results

from our experiment indicates that the Mosquitto broker could handle 1000 messages per second (from 2 MAVs).

MQTT brokers like Mosquitto can connect multiple brokers together with the *bridge* feature to increase scalability of overall system and allow separate systems to operate on the same data. In this configuration, messages are synchronised across all the brokers. This bridge also has a *round_robin* option that defines the behaviour of the bridge on a connection failure and provides a list of alternative IP addresses.

Some MQTT brokers like HiveMQ [99] also support clustering of MQTT brokers. The brokers can be connected in a setup as described in Fig. 6.4 through a TCP 'load balancer' to provide a scaled environment. In this scenario, if one MQTT broker or the control unit is not available, the available brokers can reroute the traffic to another available processing unit.

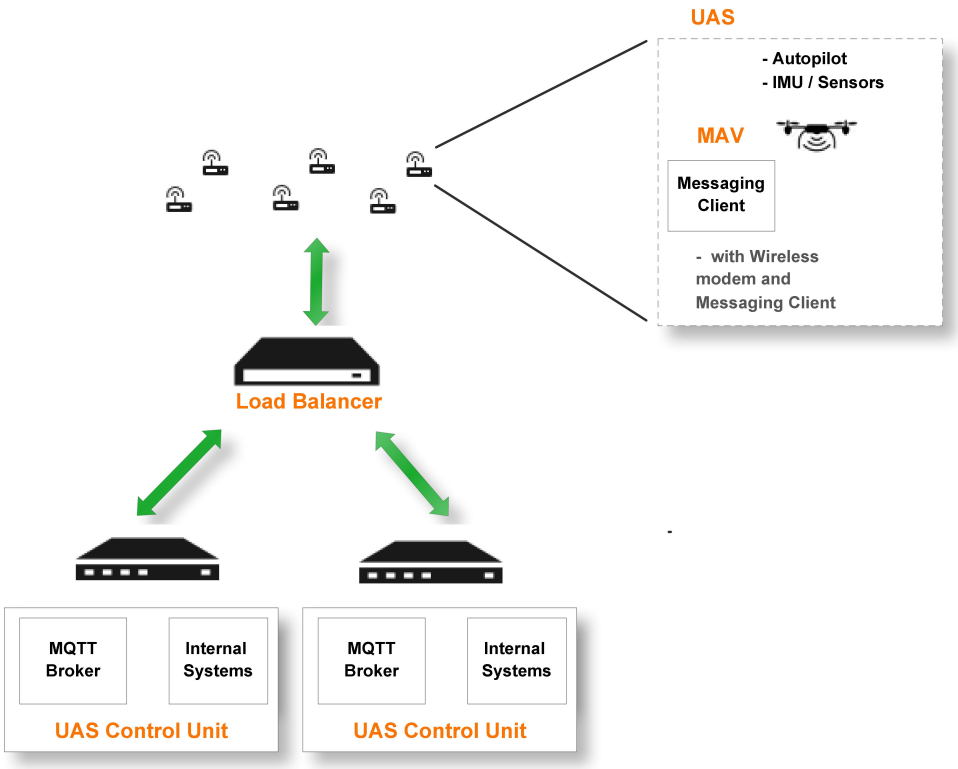


Figure 6.4: MAVs connecting to a load balancer, forwarding all connections to two MQTT brokers and control units.

The MQTT pub/sub protocol addresses data according to topics rather than physical network location. A topic consists of multiple levels separated by a forward slash at each level. Wildcards are used to allow the subscribers to subscribe to more than one topic by using '+' for a single-level wildcards or '#' for multi-level wildcards. Hereafter, we present a few examples of topic nomenclatures for transmitting data streams in the proposed UAS:

- *uas/mav2/imagedata* and *uas/mav2/navdata* - Topics published from a GCS
- *uas/mav2/+* - Subscriber subscribes to all topics one level higher than *uas/mav2*
- *uas/#* - Subscriber subscribes to all topics at any level higher than *uas*

MQTT uses Transport Layer Security (TLS) and Secure Sockets Layer (SSL) to provide a secure communication channel among the distributed agents. A broker helps establish a hand-shake at connection and authentication of the client. There is certainly a cost involved to enable security for a constrained resource messaging client, which in turn requires additional CPU usage. Hence, this additional overhead does impact the rate of data transfer in a high frequency real-time system.

Fig. 6.5 illustrates the high level system architecture based on the MQTT implementation. The distributed communication is based on the communication gateway, message broker, and MQTT pub/sub mechanism. The communication software components abstract the network topology and address contents by data semantics, instead of physical network addresses.

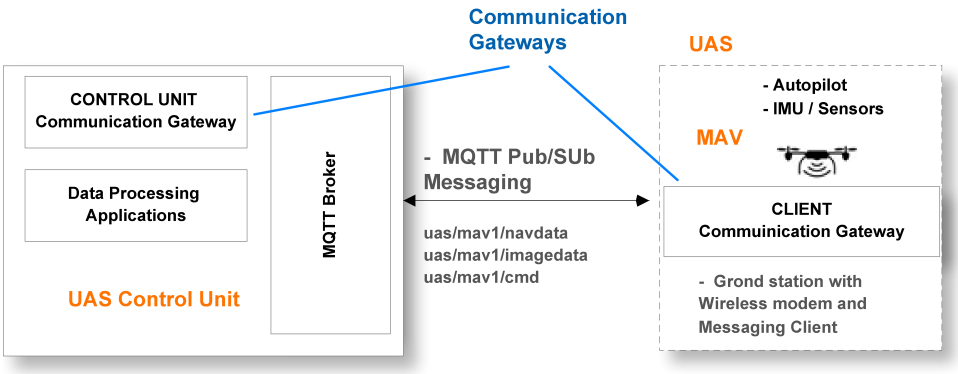


Figure 6.5: System architecture for the communication gateways based on MQTT.

DDS communication gateways

As previously mentioned, DDS is a high-performance pub/sub communication protocol which implements a fully distributed peer-to-peer communication. It is data-centric, meaning routing of data between publishers and subscribers are based on a distributed data model and does not rely on an external Database Management System (DBMS). All communication happens within a *domain*. Topics can be segmented based on data, hence the middleware provides *DDS partitions* within the domain. The high-level message routing and system components for the DDS communication gateways are the same as in Fig. 6.5, except there is no need for a message broker in DDS.

Fig. 6.6 shows a a scaled architecture with multiple MAV agents. DDS participants and data processing systems can be organised in a certain layout to minimise failure

points, and also to prevent communication bottlenecks. While M2M messaging is possible with DDS, this project implements only the M2G communication. In this case, the control unit is used to centralise processing of information.

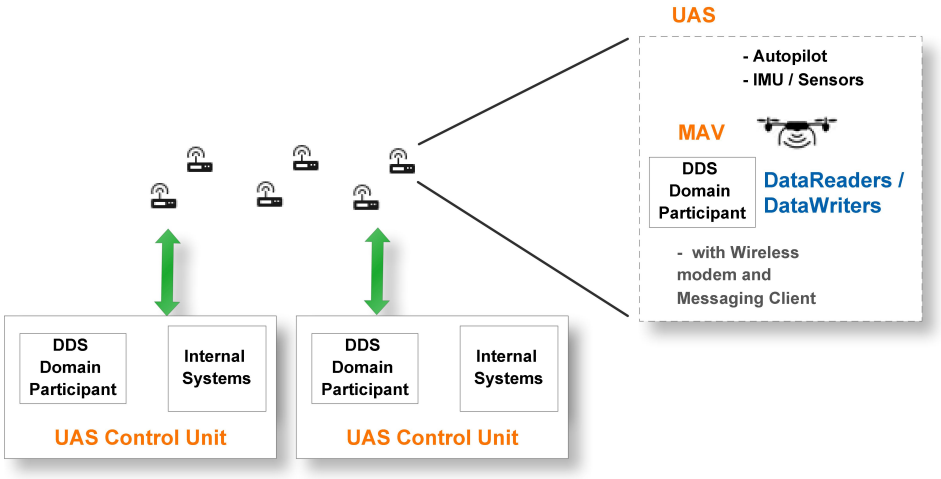


Figure 6.6: UAS and control units as DDS participants in a scaled, high-availability scenario.

6.4.4 The autonomous navigation testbed

So far, this chapter described the characteristics and motivation behind the sub-systems and communication pattern. To benchmark the communication architecture, this work uses the autonomous MAV EKF-based SLAM navigation system described in [9]. This section uses the system model described in section 5.4 and outlines the data processing components of the architecture. This navigation system has the capability to locate a flying robot in an environment deprived of a reliable and precise GPS. To achieve this, the MAV uses its on-board sensors for odometry and localisation using distinguishable landmarks. The autonomous quadcopter navigation testbed helps to evaluate the system performance, and more specifically accuracy of the SLAM system in a distributed computational environment.

Many navigation systems are camera-based or hybrid, i.e. uses a combination of other sensors. The work presented in [9] uses the on-board front-facing 320×240 resolution camera of the Parrot ARDrone2, the ultrasound altimeter and IMU data to predict pose of the quadcopter. Fig. 6.7 shows the components of the system, as well as the frequencies of transferring streaming data.

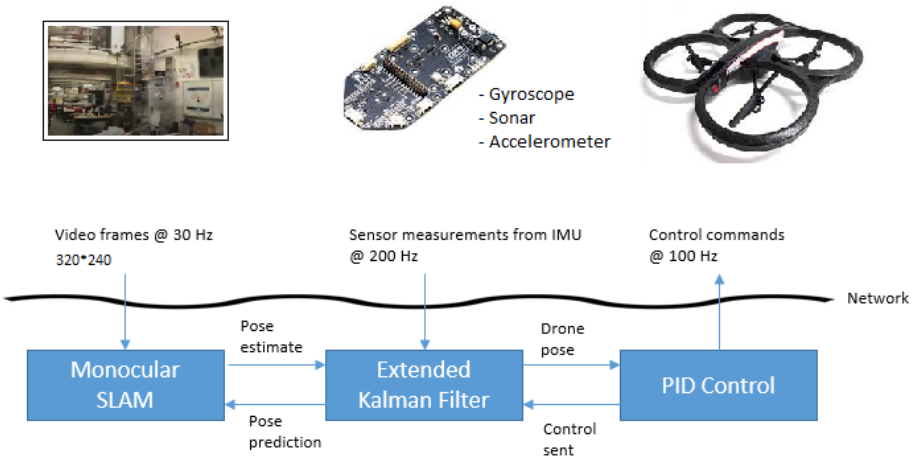


Figure 6.7: Camera-based navigation system components of the testbed and delays in data communications.

The quadcopters front camera captures images which are then converted to keyframes. Then, the absolute pose of the drone is estimated by using the visual tracking methods of PTAM. This tracking method is a monocular SLAM algorithm from [7] explained previously throughout section 5.4.1. The system also aims to cater for various delays present due to network communication, and temporary loss of visual tracking. With the help of visual and inertial sensors on the drone, the testbed uses the monocular SLAM tracking, an extended Kalman filter, and a PID controller for autonomous navigation in unknown environment. The purpose of each software component are described below:

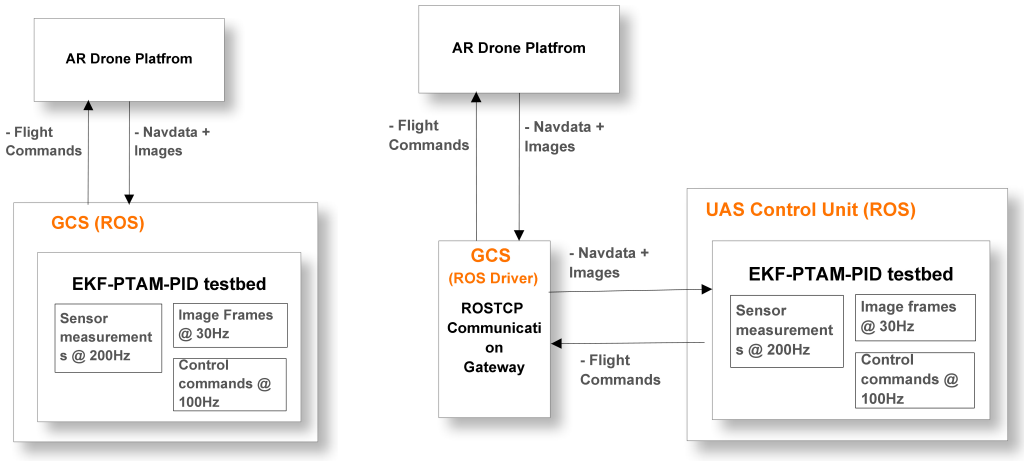
- **Parallel Tracking and Mapping:**
  The PTAM algorithm receives image frames at 30 Hz. The Kalman filter calculates predication-based states for the MAV, and this information is subsequently fed into the monocular SLAM for initial camera tracking and recovery upon loss of tracking. The IMU Kalman filter predictions also allows the SLAM to discard tracked frames when it measures high deviation in the pitch or roll angle. This prevents corruption of the map.

- **Extended Kalman Filter:**
  The EKF computes an estimate of drone's state variables, i.e. the pose and speeds. This estimation is based on the measurements provided by the sensors of the PTAM and IMU. The estimation is also useful to compensate for delays due to off-board processing through buffering of navigation and visual data.

- **PID controller:**
  The controller is used to direct the drone to set value by generating a set of control signals. The control signal is the weighted sum of the three terms, i.e. proportional, integral, and derivative gains.

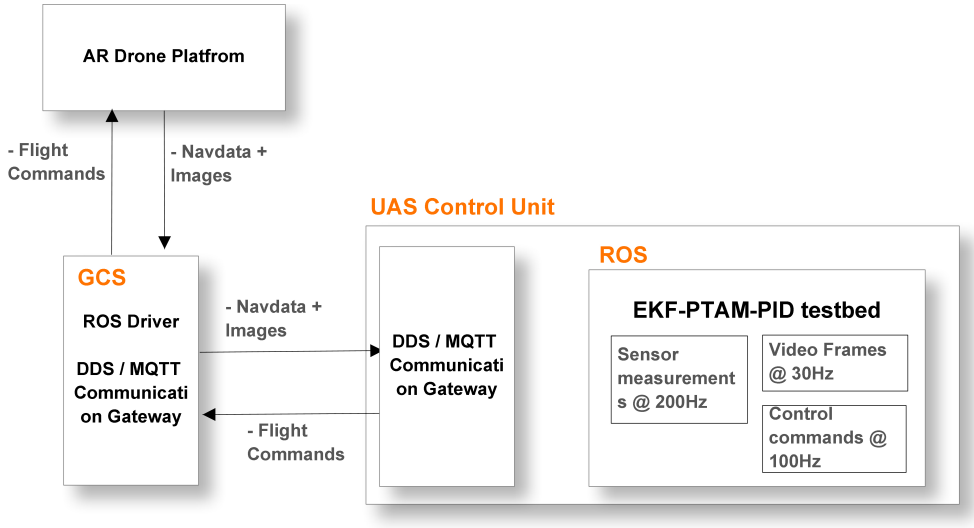
6.5 Data collection and analysis

The previous section described the navigation sub-systems, which are used as a testbed to evaluate the algorithm robustness in a distributed, constrained, and/or multi-robot environment. In this respect, this work exposes the system to the following test scenarios illustrated in Fig. 6.8:



(a) Non-distributed navigation system

(b) Distributed navigation system with ROS only



(c) Distributed navigation system with MQTT and DDS gateways

Figure 6.8: Overview of the three approaches for data collection, involving the navigation system from [9] and the communication agents.

- First, the sub-system is evaluated in a non-distributed mode and directly uses the ROS autonomous navigation sub-system as described in Fig. 6.8 (a). Each node runs specific software component on the same GCS computer;
- Second, the sub-system is tested in distributed setup with ROS nodes communicat-

ing using TCPROS. The data exchange takes place via ROS nodes and ROS topics on the GCS and control unit. This is depicted in Fig. 6.8 (b); and

- In the third scenario, the navigation system is executed with two added communication gateways and the UAS control unit is executed on a distributed server, implementing a bi-directional communication mechanism using MQTT and DDS protocols as shown in Fig. 6.8 (c).

The SLAM-EKF-PID parameters are calibrated and extended with additional parameters to account for the different delays induced by the distributed communication. This tuning helps in monitoring the system performance, accuracy of the state variables prediction model, and most importantly, allows to deduce both quantitatively and qualitatively the operability of the navigation system under the given constraints.

The research methods in this study were split into the following steps:

1. The navigation system was executed in a non-distributed setup. In other words, the processing was done on a GCS laptop directly connected to the quadcopter.

   This step involves data collection for estimating the state prediction accuracy using the monocular SLAM algorithm.

   The next chapter describes the experimental setup and software components of the system as implemented in [9].

2. The system architecture was designed and implemented for the distributed UAS with a special emphasis on the communication protocols and gateways.

   The frequency of the TCPROS, MQTT and DDS frame rate were used for indicating and measuring the performance and throughput of the communication links.

3. Furthermore, the latencies associated with the pub/sub LAN communication and WiFi communication were computed and used by the Kalman filter for computing time-synchronised control commands.

4. This research also compared the TCPROS, DDS and MQTT Round Trip Time communication for the distributed system, and assessed the factors influencing performance of the communication.

5. Finally, the behaviour and accuracy of the autonomous navigation system were analysed and compared with respect to the additional delays and slight adaptations to the EKF-based SLAM system.

6.6 Summary

This chapter summarised the methods associated with this research work. In this respect, first this chapter defined the key requirements of a distributed MAV architecture and

proposed a design involving publish/subscribe protocols MQTT and DDS. Then, the chapter assessed the main features and attributes of these two protocols with a particular focus on its relevance to low-latency navigation algorithms running on a distributed server (referred to as the *control unit*).

Then, the mathematical models for the ROS implementation of a camera-based quadcopter navigation as in [9] are presented: the extended Kalman filter, Parallel Tracking and Mapping, and PID controller.

The last section of this discussion presented the overall approach and methods for data collection and analysis using ROSTCP, MQTT and DDS.

Chapter 7

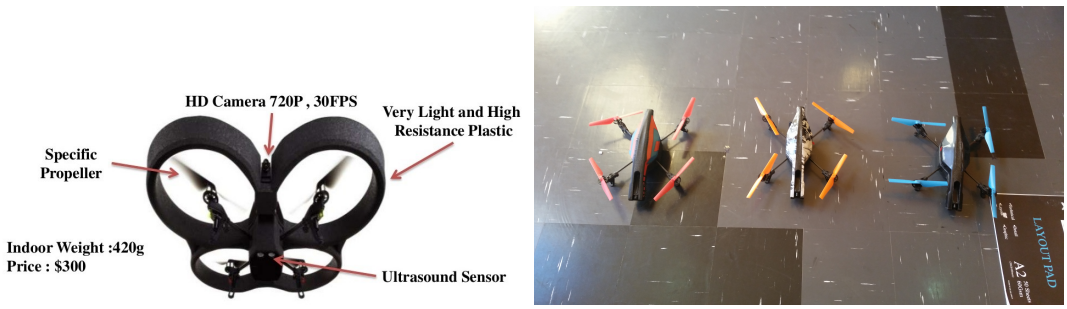
System Implementation and Experiments

This chapter describes the software components and implementation of the architecture proposed. The focus is on the ROS driver for the MAV, messaging communication gateways, and EKF-PTAM-PID navigation testbed.

7.1 AR Drone MAV platform

Many off-the-shelf MAVs are available with open System Development Kits (SDKs). These MAV systems can be used to experiment and simulate real-life scenarios. This research work used the Parrot AR Drone 2.0 for conducting the experiments and simulations. (Fig. 7.1 (a)) shows various components of the Parrot AR Drone 2.0. It runs a Linux operating system with USB, WiFi interfaces and multiple sensors; e.g. pressure, ultrasound, and gyroscope. These features and functionality of the drone makes it possible to maintain stability and precise control. The drone has an optional GPS device for executing semi-autonomous flight. Additionally, the drone is equipped with a front and bottom facing camera, which streams video over WiFi to iPhone or Android mobile devices. Other MAV platforms may consist of a pair of radio for telemetry and video data analogue receiver, making decoding of signals more complex. Fig. 7.1 (b) shows three Parrot drones used during the practical experiments of this dissertation.

The AR Drone Autonomy ROS driver from [102] provides communication interface abstraction to and from the MAV as illustrated in Fig. 7.2. Appendix A.1 outlines the specific sensor values and format of the data accessible from the AR Drone 2.0 platform.



(a) Parrot AR Drone2 specifications and on-board sensors.

(b) Parrot AR Drones used for experiments.

Figure 7.1: Parrot AR Drone2 MAV platform

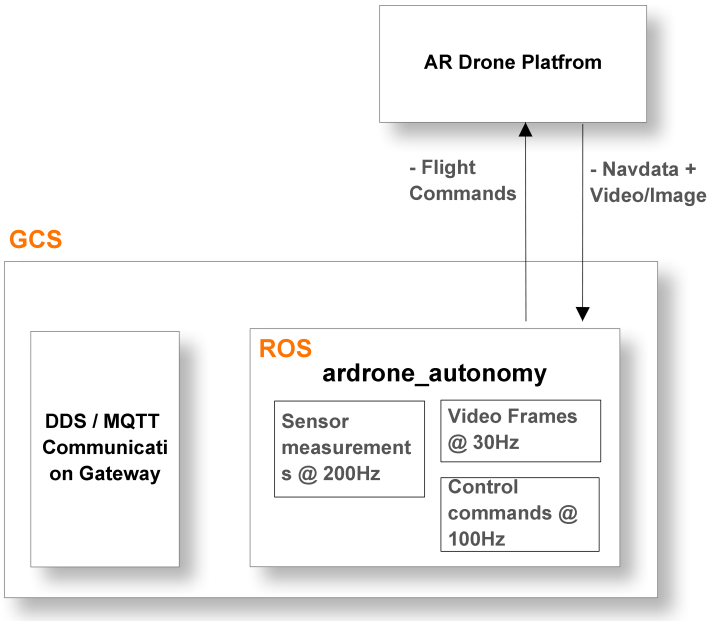


Figure 7.2: Parrot AR Drone2 MAV platform sending and receiving data using the ROS 'ardrone_autonomy' driver.

7.2 System overview

This section describes the architecture and implementation of the end-to-end systems: the UAS and the control unit. Fig. 7.3 shows all the components, software, and different parts of the proposed architecture. Subsequently, the implementation approach of the architecture and interacting components are outlined; namely the communication gateways and navigation system.

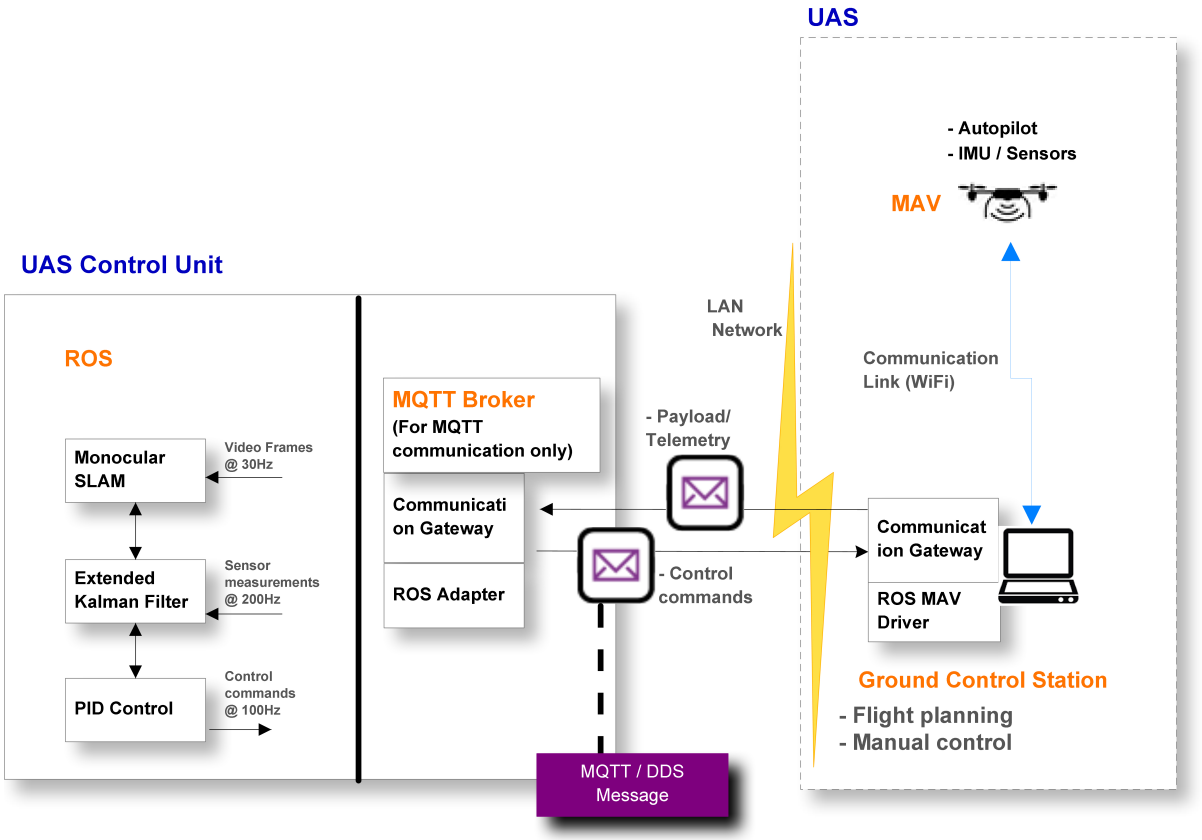


Figure 7.3: Implementation of the software components and outline of SLAM algorithm. Here, the extended Kalman filter computes an accurate estimate of the drone's pose and speed at each video frame.

The overall software platform illustrated in Fig. 7.3 consists of the following system components and sub-components:

1. Ground Unit: The ground unit comprises of a C/C++ implementation of the AR Drone2 ROS driver. This implementation enables communication and control of the drone. This part also consists of the client communication gateway, built on top of MQTT mosquitto broker package and DDS RTI Connext library from [103]. The client adapter publishes payload (camera images) and telemetry (IMU, sonar navigation data) to the interested data subscribers. This part of the system also monitors the WiFi delay and transmits this information to the control unit; thus, helping to keep account of the latency in the wireless network.

2. Control unit: The UAS control unit comprises of the following sub-components:

   (a) The MQTT broker which runs as a background process and uses TCP port to accept messages. Mosquitto works as a messaging middleware and handles the communication to the UAS and exposes a set of standard UAV communication API interfaces in the form of topics. For the DDS implementation option, the

control unit does not need a broker and communicates as a domain participant subscribing and publishing data.

(b) The communication gateway of the control unit is a service API that exposes data to other systems internally (or externally). The internal system in this case is the ROS connecting to the messaging gateway. This setup provides a clean interface for sending image/IMU data and receiving control commands. Hence, the service API across topics makes data available within the same co-located data centre.

(c) Another part of the UAS control unit is the ROS platform itself, which is often considered middleware. This is due to the nature of services the ROS provides, that is in other words, the ROS helps in inter-nodal communication. Furthermore, the ROS provides the building blocks for the EKF-SLAM-PID to exchange information over ROS topics, see Section 5.4 for further details.

3. Testbed: The testbed for the architecture, i.e. the monocular SLAM camera-based navigation system running on the ROS middleware. The ROS nodes communicates with the service interface of the UAS control unit in 2(a). The software of the whole system is referenced in more details within Appendix A.5 for interested readers.

7.3 Communication gateways

7.3.1 MQTT sender and receiver

Most message-oriented-middleware hub and spoke protocols have client libraries available for various programming languages. For example, MQTT client implementations exist for Java, JavaScript, Python, C, C++, Lua, and C#. This work experimented with several client implementations. In Appendix A.4, we give a brief description about the findings and issues encountered for each. The MQTT C++ communication gateway uses a better serialisation method than the python implementation, and the JAVA implementation had a rather mediocre performance.

C++ *libmosquitto* from [104] library has been the preferred MQTT client implementation. It is compatible with the *ardrone_autonomy* [102] ROS package and provides the right object-oriented and thread mechanism to handle high rate and volume of data over the TCP connections. As discussed in the previous section, the ROS Driver is responsible for exchanging and interpreting MAV data. At the GCS, the *libmosquitto* acts as a bridge between ROS driver and other MQTT subscribers.

The QoS configuration for the MQTT sessions can be set to 0 (the minimal level) and guarantees best effort delivery; it is often called fire-and-forget. The messages are time-stamped at the sender-end to help communication latency computation among the distributed computers. Appendix A.4 outlines the source code and implementation details of the MQTT gateway.

7.3.2 DDS sender and receiver

The DDS implementation uses the RTI Connext DDS messaging library (see [103]) and corresponding academic licence. RTI DDS is an implementation of the OMG DDS standard to "foster interoperability and an open architecture" which meets the robust requirements of the Industrial Internet of Things (IIoTs). The DDS implementation also guarantees low-latency and real-time QoS distributed data-oriented communication. In addition, the Connext DDS framework provides tools to accelerate system integration, testing, and debugging.

For DDS, the processor architecture needs specification for compiling and generating data types. The following two code building architectures were tested:

- i86Linux2.6gcc4.1.1
- x64Linux2.6gcc4.1.1

Additionally, an Interface Definition Language (IDL) file need to be defined when the public data structures are described. The IDL file help generates project stubs for facilitating the messaging between publishers and subscribers. The *Navdata* and *Image* frames types are marshalled and un-marshalled automatically. QoS allows to use convenient pre-made profiles such as *BestEffort* and *Streaming*.

In general, the DDS implementation has a steeper learning curve than MQTT which consists of simpler verbs to communicate. However, the DDS configuration was seamless as no broker was involved, and the services were auto-discovered while interacting within the same *Domain*. Appendix A.4 describes the source code and licence details of the DDS gateway.

7.4 The navigation testbed

7.4.1 ROS platform and software components

The software components of the ROS platform were developed using the *ROS catkin* package builder on Ubuntu 14.04 OS. Fig. 7.4 shows that ROS architecture. The software details of the navigation system were derived from [9], which was extended and used as the test platform. Appendix A.5 gives more details on the software components of the navigation system.

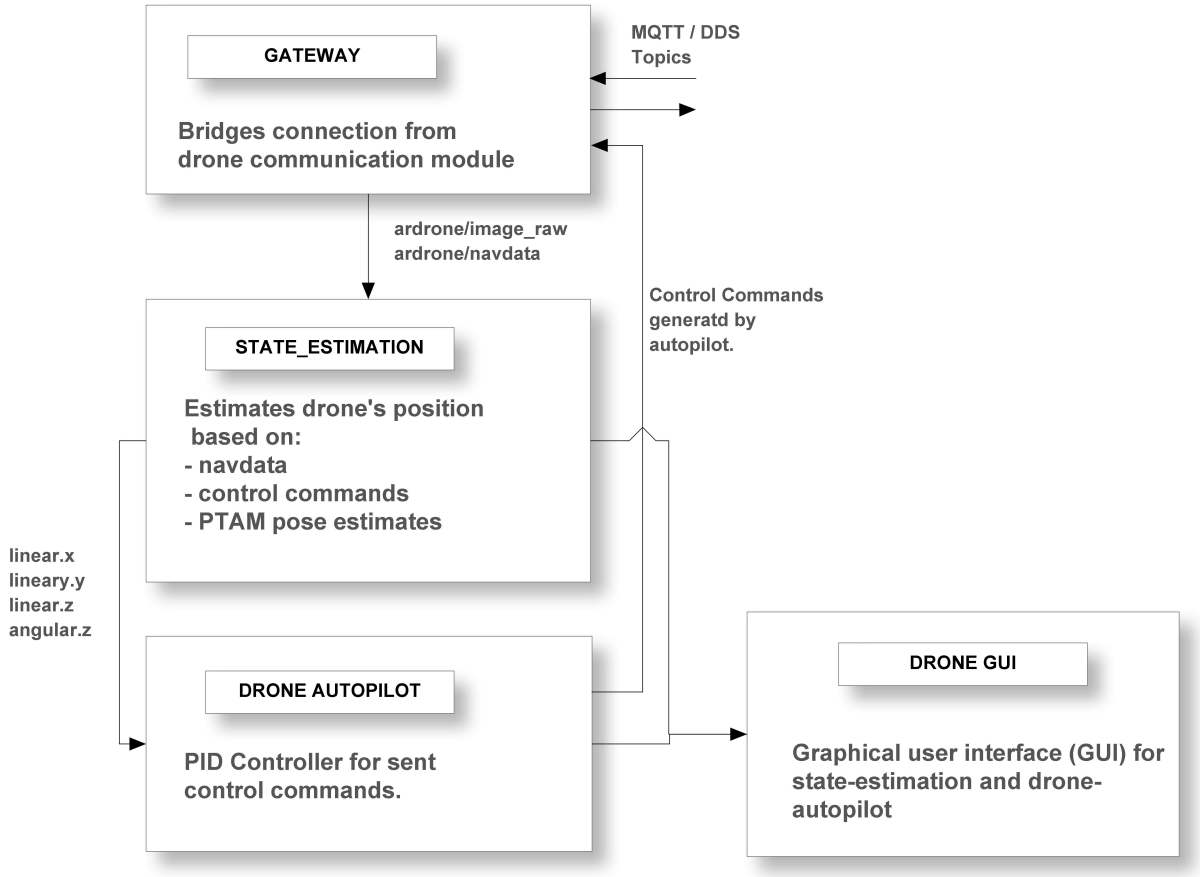


Figure 7.4: ROS software components.

ROS defines topic names and a 'namespace' for the communication among the ROS nodes. The following points give a description of each ROS node as depicted in Fig. 7.4:

1. The MQTT or DDS gateway bridging the MQTT and DDS messages across topics to the ROS messages.

2. The state-estimation node, which also includes the PTAM. There are three key inputs to this node: a) IMU sensor measurements, b) Optical pose estimates from PTAM, and c) Previously sent control commands. Key outputs are the PID controller signals for the x, y, z position and the yaw angle.

3. The drone-autopilot node is used for controlling, simple route planning, and sending control commands. The node also sends commands to toggle the emergency, flat-trim (after a crash), and reset the state.

4. The Graphical User Interface (GUI) for status information and user actions.

7.4.2 EKF data fusion and time delays

In summary the following models are inherent to the inner workings of the EKF state estimation process:

1. State transition model: This was previously defined in section 5.4.2 and (5.31). The state transition model defines the complete state transition for each time interval, propagating the state x_t to $\mathrm{x}_{t+\delta t}$.
2. Observation model: There are two sources of information namely the PTAM tracking module and the IMU sensor measurements. Consequently, the algorithm utilises two observation functions as described in section 5.4.2. The observation model calculates the expected measurement based on the estimated state variables of the MAV.
3. Control model: The PID controller for all the four parameters of the drone state variables i.e. the position in x, y, z, and the yaw orientation Ψ.

The delay compensation in the distributed system is another key aspect underlying the functioning of the Kalman filter; it takes into consideration dynamic time-spans for the wireless LAN and pub/sub communication mechanism. The total Round Trip Time (RTT) latency for the communication is given by 7.1. The values of the WiFi RTT, DDS RTT and MQTT RTT latencies are re-established after each time interval.

$$RTT_{total} = RTT_{wifi} + RTT_{pubsub} \tag{7.1}$$

Two measures for the RTT were calculated based on two packet sizes of 0.5 Kb and 20 Kb respectively, as shown in (7.2) and (7.3). This is done at one second interval within the course of the SLAM-EKF iterations. These values are then used by the Kalman filter for delay compensation, and for synchronising data inputs and outputs:

$$RTT_{total,0.5Kb} = RTT_{wifi,0.5Kb} + RTT_{pubsub,0.5Kb} \tag{7.2}$$

$$RTT_{total,20Kb} = RTT_{wifi,20Kb} + RTT_{pubsub,20Kb} \tag{7.3}$$

The indicative delays of the measured and experimentally determined time-spans of each packet were on average between 20 ms to 50 ms for DDS and MQTT. The next chapter presents the detailed results of the DDS delays and MQTT delays for actual data packets; in addition, the following chapter illustrates how DDS and MQTT compare for *navdata* and *image* packets.

Fig. 7.5 shows an end-to-end iteration of the Kalman filter when one image frame is received, and incorporation of the IMU data from a buffer. The delays computed in (7.2) and (7.3) are continuously fed back to the filter. Then the delay values help to determine

the approximate time taken to apply the generated commands ($t_{control_applied}$). This is done after every 10 ms (approximately) upon receiving an image frame; similar process is carried out until it receives the next image frame.

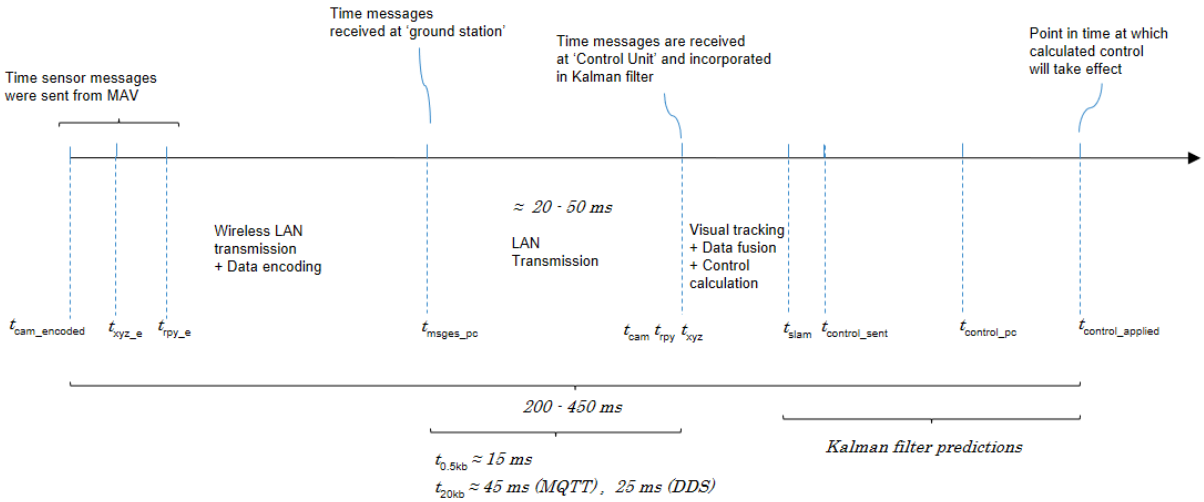


Figure 7.5: The Kalman filter takes values from a buffer and must be rolled forward to compensate for delays at which a video frame was received.

The following points provide a brief description of each time interval utilised in Fig. 7.5:

- $t_{cam_encoded}$**:** The time stamp of one image frame was encoded from the MAV. t_{xyz_e} and t_{rpy_e} are the estimated time, the velocities, and roll, pitch, and yaw values were encoded;
- t_{msges_pc}**:** The time stamp IMU and image frame are received at the ground station;
- t_{cam}**:** The time stamp an image gets incorporated into the EKF;
- t_{xyz}**:** The delay for the MAV velocities values, from the time it is computed by the IMU to the time it reaches the EKF-PTAM system;
- t_{rpy}**:** The direct roll, pitch, and yaw measurements from the MAV gyroscope. These values in principle require very little processing. So, delay is assumed to be less than velocities and image frames;
- t_{slam}**:** The time required for image feature tracking and EKF-SLAM pose estimates;
- $t_{control_sent}$, $t_{control_pc}$, and $t_{control_applied}$: The time spans for a control command to be generated at the control unit from the EKF, in order to reach the ground station and the time the control takes effect on the MAV, respectively.

7.5 Experiments design

All the experiments conducted in this work are based on the system architecture illustrated in Fig. 7.6. The network layout consists of a single or multi-MAV connected to one GCS. The MAV connects to the GCS through a WiFi router, which is bridged to a LAN network consisting of the GCS and the control unit.

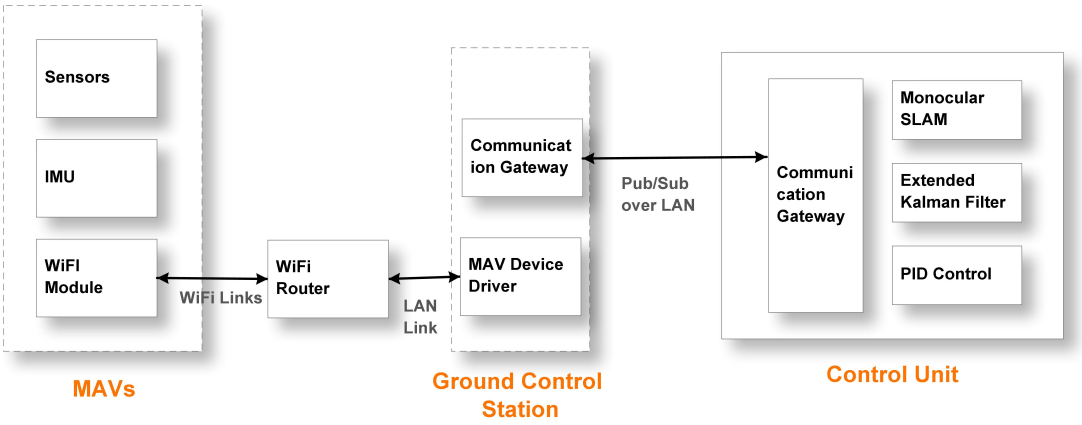


Figure 7.6: System architecture for an experiment setup with 3 MAVs.

The specifications of the GCS and control unit standard computers were as follows:

- Control unit: Intel i7 3.40 GHz with 4 CPU cores and 12 GB RAM.
- GCS: Intel core i5 3.20 GHz with 2 CPU cores and 8GB of RAM.

The MAV WiFi network becomes highly unstable when other nearby WiFi networks interferes with it. Appendices A.2 and A.3 describe the issues and solutions of the MAV WiFi networks. The resulting network layout and setup in Fig. 7.7 was the one which provided the best WiFi performance and hence enabled a multi-MAV setup. This is primarily due to the powerful Netgear 1 GHz dual-core processor WiFi router.

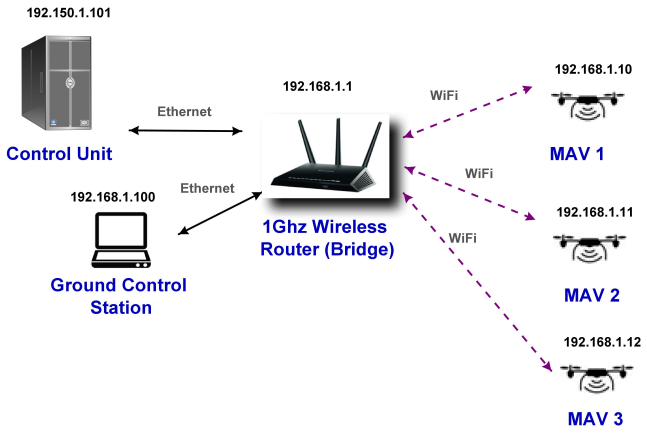


Figure 7.7: Network layout and setup for an experiment with 3 MAVs and the Netgear WiFi router.

The following videos show some experiments carried out for evaluation or data collection:

- https://youtu.be/n5JWWjqPFl0
- https://youtu.be/-3RH47MQn-c

In the first video, the experiments were conducted with one MAV, the GCS client and the control unit server. These essentially focused on the frame rate and performance of the pub/sub mechanism, as well as the navigation system parameters.

The second video shows experiments with only one GCS computer serving as client for two and three MAVs. The navigation system is hosted on the control unit and consists of a GUI which broadcasts control information to subscribers for each MAV at the GCS. For instance, a 'take-off' command will start all the drones. Similarly, in the other direction, the control unit subscribes to data from all the MAVs. However, the autonomous SLAM system was executed for only one MAV, i.e. it only considered the video and IMU input data from one MAV. Another SLAM process can be started, but this will depend on the number of available processing cores. The experiments essentially show that with multiple MAVs, the video feeds for each MAV is successfully received at the control unit with no visible loss in quality or loss in frame frequency.

Chapter 8

Experimental Results

8.1 MQTT, DDS and TCPROS message transmission performance.

As discussed in the previous chapter, the EKF-PTAM navigation system compensates for the delays in the system by using the Round-trip time (RTT) latency of the pub/sub messaging. The performance in terms of network latency was measured between the source and destination nodes. The EKF computes the latency for explicit MQTT/DDS messages of different sizes (0.5 kb and 20 kb) sent from the control unit to the GCS, as described in Section 7.4.2. In the aforementioned case, the time from the source was used for initiating a 'Request' message, and the 'Response' message time was computed using the same clock. As discussed earlier, the RTT values vary between 20 ms and 50 ms.

8.1.1 Results - MQTT communication latencies

In addition to the aforementioned, this research computed the unidirectional delay of actual MAV data packets in order to obtain a realistic latency for the communication from the GCS to the control unit. This end-to-end latency computation takes into account the duration between the time stamp (at the ground station before processing) and delivery (at the control unit) of IMU data and video frame packets. Unlike RTT 'ping' messages described, these latency values give the real-time duration that includes marshalling/unmarshalling, transmission, and ROS processing. To obtain an accurate time duration in milliseconds between the sending and receiving nodes, it is important to synchronise the computer clocks of each communicating node with the time of a common server. The nodes are accurately synced using the Linux Network Time Protocol (NTP) Linux server and client configuration. This setting provides a minimum offset, between 10 to 30 ms, of the clocks.

Fig. 8.1 shows a screenshot from one of the experiment carried out during this research

work. It shows two visualisation nodes at the control unit and the ground station with MQTT senders and receivers.

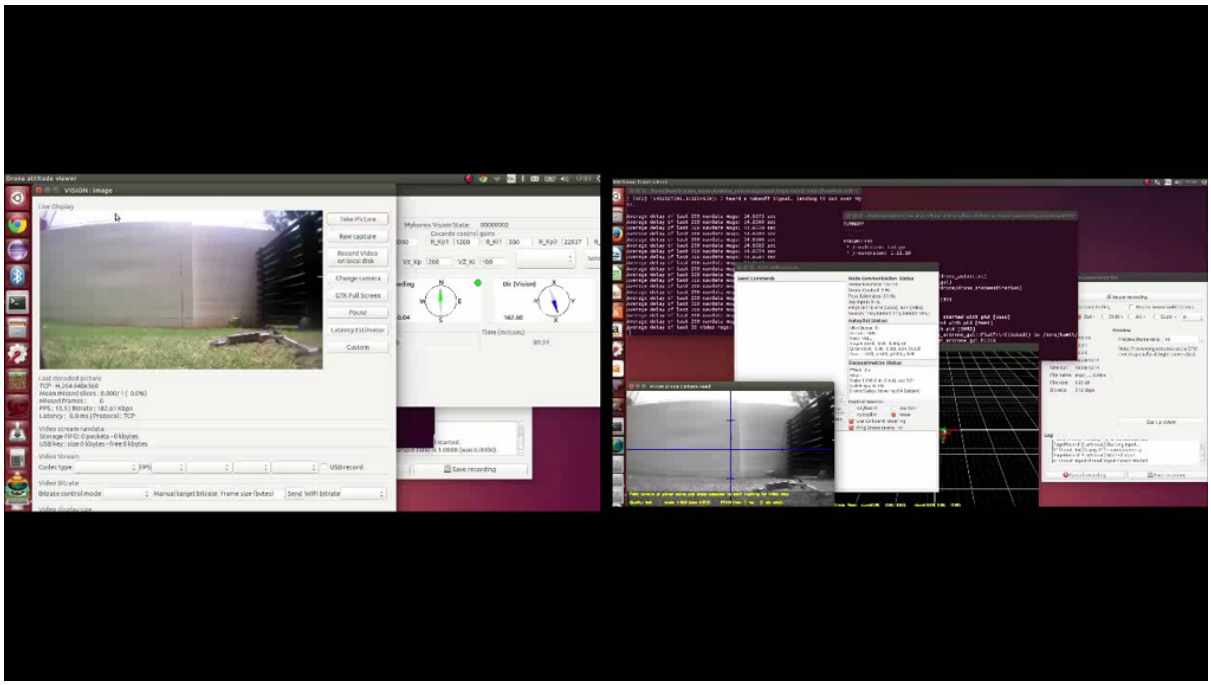


Figure 8.1: Packaging and transmission of video frames (sent at 30 Hz) and IMU sensor data (sent at 200 Hz) published in real-time, with no loss in sending and receiving frequency.

The experimental data to compute the communication latencies were collected from a quadrotor which was allowed to hover for one minute. Fig. 8.2 depicts an example of the PTAM map for such an experiment. The following points describe the experimental setup conditions:

- IMU packet size = 0.3 kb, sent at 200 Hz;
- Image packet size = 691 kb, sent at 30 Hz;
- Average latencies for 20 experiments;
- Average latency of last 200 messages for navdata, and last 30 messages for images; and
- Ethernet LAN of 1 Gbps.

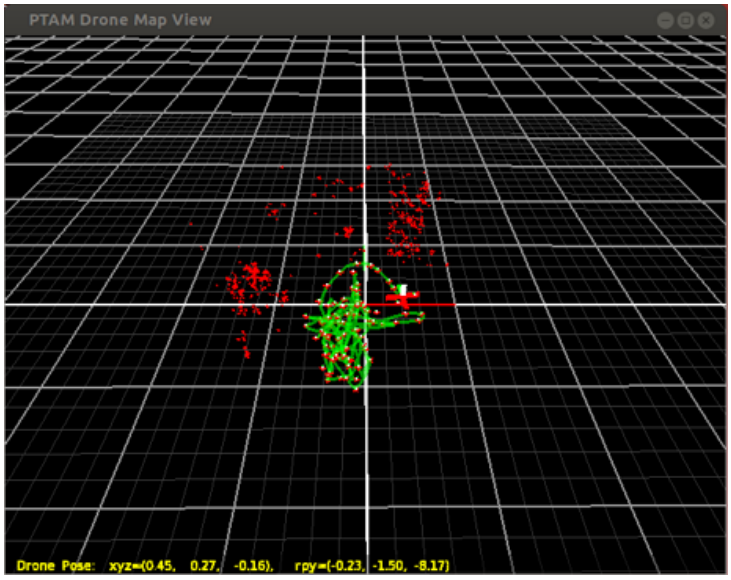


Figure 8.2: PTAM map of coordinate frames for each drone pose while holding position for 60s.

Fig. 8.3 shows the experimental results for MQTT latency. Each box-and-whisker plot represents spread of the average latency for samples of 200 navigation data messages during one experiment. The median value for one plot was obtained using (8.1).

$$\frac{1}{n}\sum_{i=1}^{n}\mu_{imu,i} \tag{8.1}$$

where,

- i is the sample interval for each set of 200 IMU message packets;
- $\mu_{imu,i}$ is the mean IMU message latency for set i; and
- n is the total number of sample during a 60 s experiment.

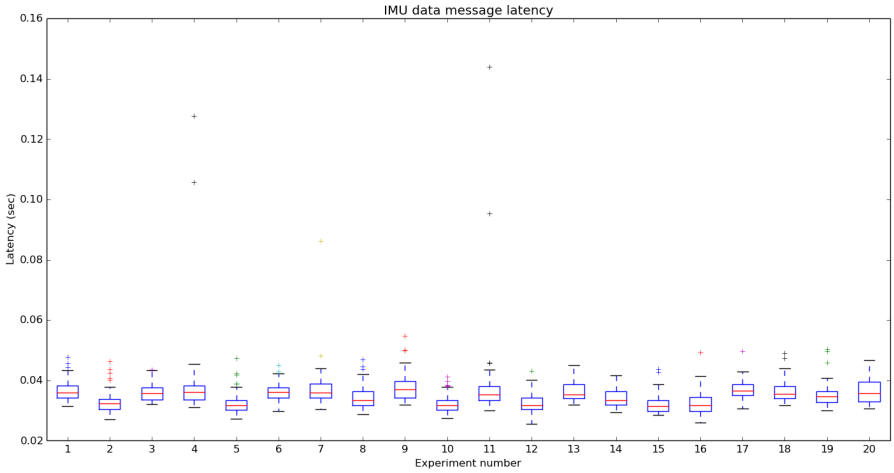


Figure 8.3: IMU data messages latencies for MQTT protocol averaged for each experiment with the drone hovering for 60 s.

Fig. 8.4 shows the message transmission latencies for camera image frames.

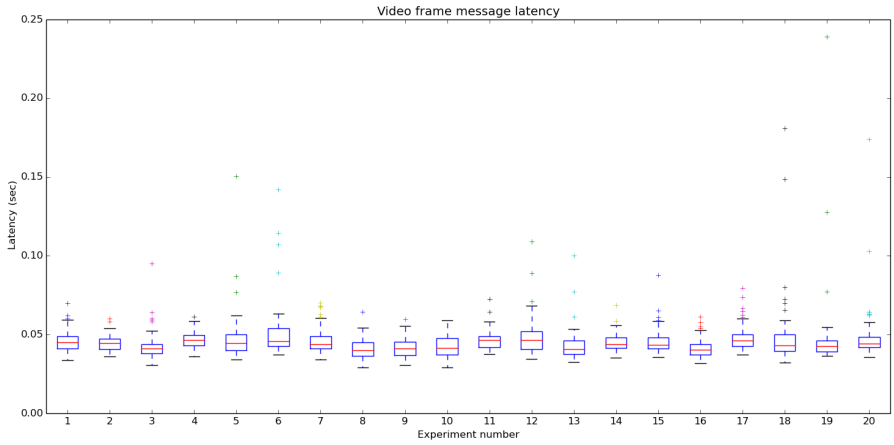


Figure 8.4: Image packets latencies for MQTT protocol averaged for each experiment with the drone hovering for 60 s.

The results show that over a distributed network, the gateway operations experience low delays (40 ± 20 ms) and smooth communication for messages involving image and navigation data transfers. Figures 8.3 and 8.4 show that the MQTT frame transmission

latencies contain few outliers and peaks, but these were much less than the WiFi delays. The results show that there were no bottlenecks encountered in the end-to-end data transfer. The proposed configuration guarantees smooth transmission rate for large size image packets sent at high frequencies with moderate computation and memory usage.

8.1.2 Results - DDS communication latencies

The DDS transmission latencies were computed in a similar way as calculated for MQTT. The DDS implementation of the communication gateways consists of Data-Writers and Data-Readers for image, IMU, and control commands message packets. Figures 8.5 and 8.6 show the average latencies of 20 experiments for IMU and image data respectively.

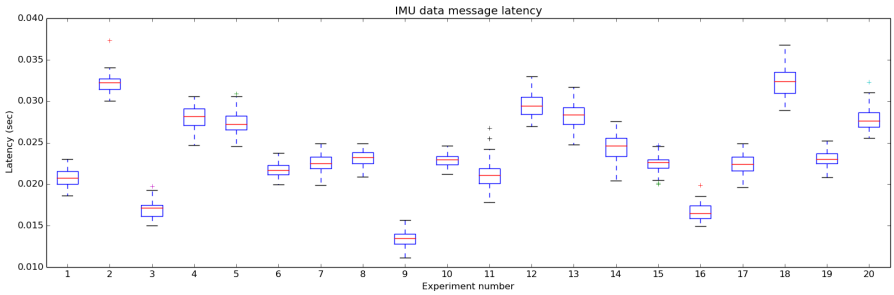


Figure 8.5: IMU data packets latencies for DDS protocol averaged for each experiment with the drone hovering for 60s.

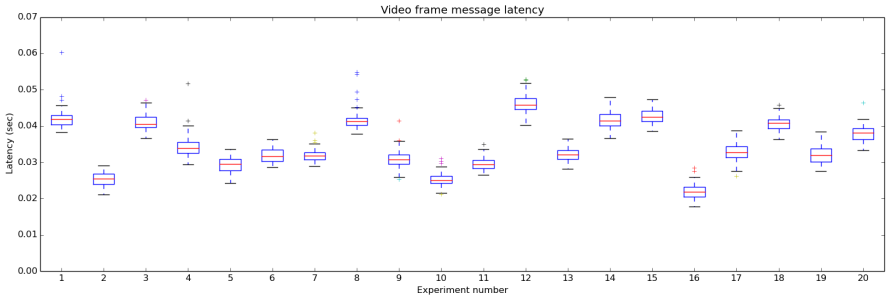


Figure 8.6: Video frames packets latencies for DDS protocol averaged for each experiment with the drone hovering for 60s.

The results show that compared to MQTT, DDS has much less outliers and high peaks with message latencies lying in the range of 25 ± 10 ms for IMU messages and 35 ± 10 ms for images. The decrease in latency over a series of experiments can probably be attributed to the CPU clock steadily drifting by a few ms.

8.1.3 Results - Network latencies averaged over 10 mins experiments for MQTT and DDS

Longer experiments of 10 mins intervals were conducted to assess the stability and performance of the communication infrastructure. The communication infrastructure comprised of the sender and receiver communication gateways. The results were obtained by computing an average for 10 experiments of the message transmission latencies for both MQTT and DDS. This work compared the average latency values of the last 200 navdata and 30 images received for each experiment throughout a 10 mins flight duration. These experiments were conducted within close range indoor environment. Fig. 8.7 shows the plot of average latencies of navdata packets. It clearly shows that the DDS implementation performs slightly better than MQTT. Fig. 8.8 depicts the transmission latencies for larger image packets. There is less difference in terms of average time; however, MQTT shows higher peaks.

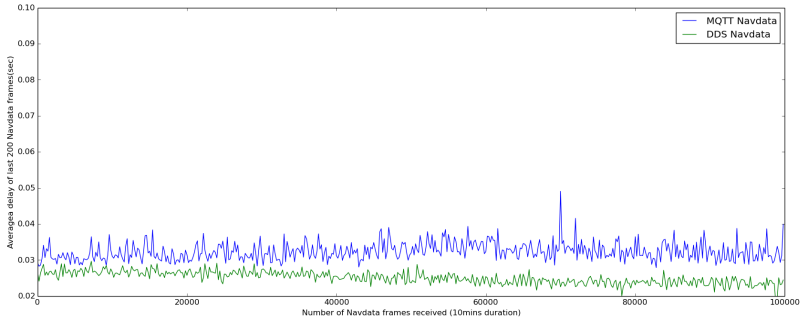


Figure 8.7: Navadata frames packets latencies for MQTT and DDS protocol over 10 minutes experiment.

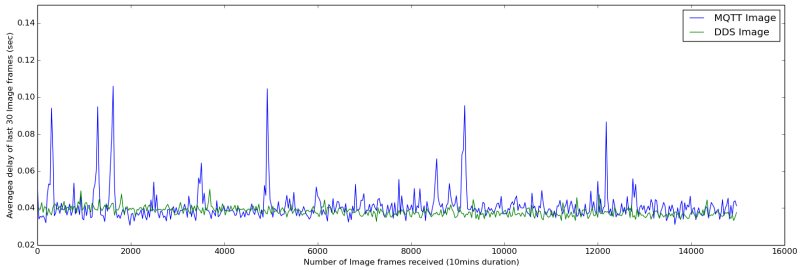


Figure 8.8: Video frames packets latencies for MQTT and DDS protocol over 10 minutes experiment.

A very interesting finding during the experiments comes from a particular situation where the sending PC experienced reduction in its processing power. Subsequently, the MQTT client at the ground station was unable to send messages at a constant delay and suffered from a considerably low sending rate for both navdata and image. This

can be attributed to reduced processing power due to increase in the CPU temperature. In such a situation, the latencies of the messages increased indefinitely by 1 second for each frame. This heavily delayed sending of messages, creating a bottleneck. The images arrived with a clear gradual delay and appeared much later at the control unit receiver.

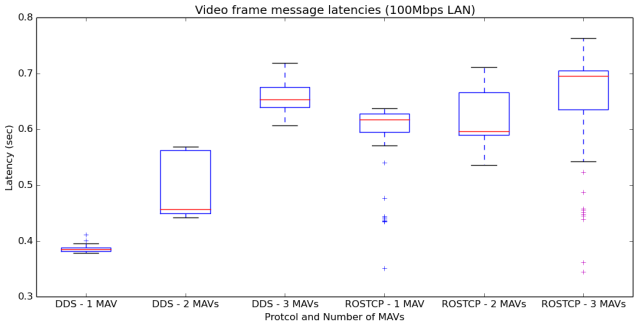
A major finding from this research shows that the gateway performed much better with DDS in such a situation. While the delay of sending navdata and image did increase due to the lower processing power at times (from 40 to around 60 ms); this delay did not keep on increasing (no bottleneck) and the sending rate lowered, but remained constant. Hence, a constant higher delay but smooth rate makes the video stream fluid. In this situation, even if an excessive higher delay can cause failure of the EKF-SLAM; a constant rate is important if the delay is within acceptable range for tractable computable solutions.

8.1.4 Results - Network latencies of MQTT, DDS and TCPROS in a multi-MAV and constrained bandwidth scenario

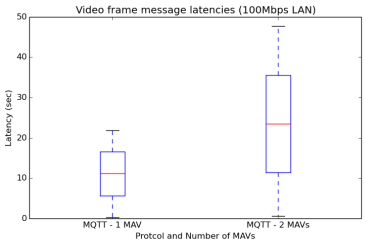
The primary focus of this work is to evaluate MQTT and DDS with a high frequency and high bandwidth robotics system. Hence, it is very interesting to investigate how both of these protocols compares to TCPROS. In addition, while the aforementioned results showed promising results in terms of functionality in a high bandwidth network for one MAV, it would be much more interesting to observe the behaviour of each protocol in the following two scenarios:

- Multi-MAV setup of up to three drones; and
- Bandwidth constrained environment of 100 Mbps LAN setup compared to a Gigabit (1 Gbps) network setup.

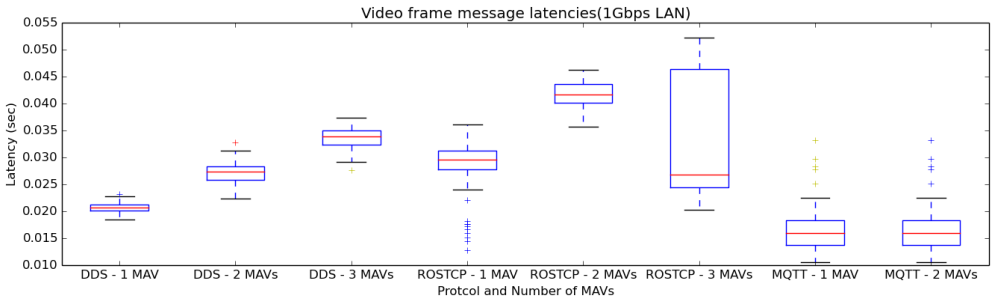
The results illustrated in Fig. 8.9 show how each protocol performs for a 100 Mbps and Gigabit Ethernet connection, with one to three MAVs. We discuss the implications of these results on the system in more detail in the next section. The major findings from the experiments is the fact that the latencies are much higher– between 400 and 700 ms– for DDS and TCPROS, over a 100 Mbps LAN network. MQTT version of the communication gateway suffered from bottlenecks over a 100 Mbps LAN network, causing latencies to increase from few 100 milliseconds to over 45 s. Also, it kept rising by approximately 1 s for every few packets. It can also be observed that DDS showed smaller average latencies than TCPROS.



(a) Video frames latencies for DDS and TCPROS on a 100 Mbps LAN



(b) Video frames latencies for MQTT on a 100 Mbps LAN



(c) Video frames latencies for DDS, TCPROS and MQTT on a 1 Gbps LAN

Figure 8.9: Communication latencies for video frames under multiple scenarios.

8.1.5 Results - Frequency and bandwidth utilisation of MQTT, DDS and TCPROS in a multi-MAV scenario and constrained bandwidth

Any of the following factors: 1) having more MAVs, or 2) less bandwidth in the network, eventually creates a communication bottleneck. Therefore, the latency of communicating each data packet under either a multi-MAV or constrained bandwidth logically has a direct impact on the frequency of the data communication, as well as the bandwidth consumed by each message topic.

The experimental results in Fig. 8.10 and Fig. 8.11 are hereby presented to investigate the aforementioned challenge. The first graph describes how the frequency varies under different scenarios, while the second one investigates the bandwidth utilisation under the same constraints. Further details on the measurements, number of experiments and data collection can be found in Appendix A.8. Surprisingly, DDS performed better than TCPROS in a scenario considering constrained bandwidth of 100 Mbps LAN. The system

for TCPROS doesn't consist of additional communication gateway components and hence it was expected to perform better than both other protocols, but it wasn't the case as shown from the performed experiment. DDS also performed slightly better than MQTT in a low bandwidth network. On the other hand, in the situation with high bandwidth of a Gigabit network, all the protocols seem to perform fairly well. A slightly better performance of TCPROS is noted in this case. MQTT communication system failed for two and three MAVs; this can be attributed either to the protocol limitations, bottleneck at the broker or inefficient buffering mechanism of the gateway code implementation.

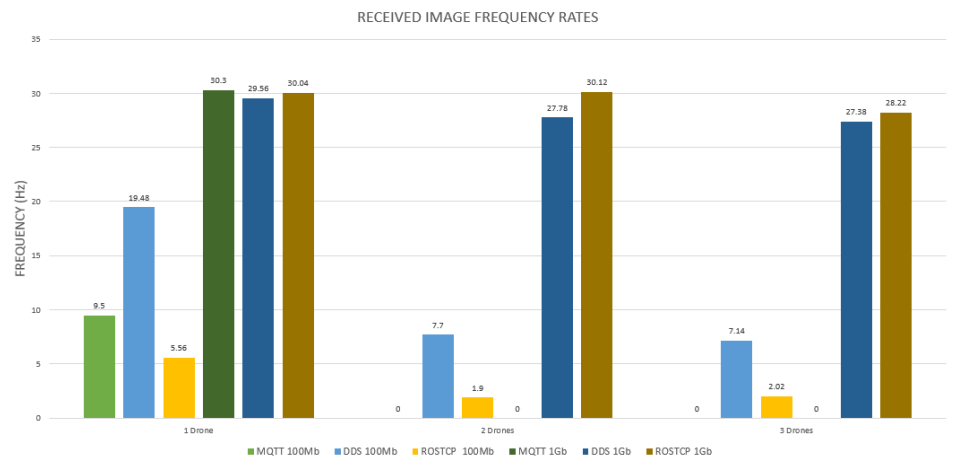


Figure 8.10: **Frequencies** of images received from the master MAV, based on following scenarios: 1) MQTT, DDS, or TCPROS protocol, 2) in a system of one, two or three MAVs, and 3) over a 100 Mbps or Gigabit Ethernet LAN.

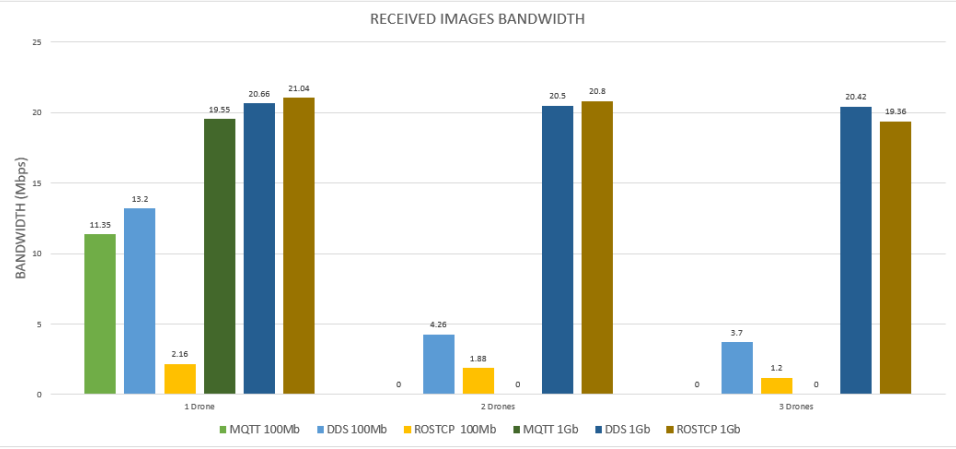


Figure 8.11: **Bandwidth** utilisation of video frames received from the master MAV, based on following scenarios: 1) MQTT, DDS, or TCPROS protocol, 2) in a system of one, two or three MAVs, 3) over a 100 Mbps or Gigabit Ethernet LAN.

The overall efficiency with respect to the **sending rate** is described in table 8.1. The same data from Appendix A.8 is utilised for each protocol under the conditions

described above. The efficiency value for each cell is the percentage of successful video frames transferred. A decrease in efficiency is either due to lost, dropped or delayed frames resulting from a communication bottleneck. The data were collected within a few seconds interval between the sender and receiver, then averaged over multiple experiments. This explains why some of the percentages are slightly above 100%. The latter can just be interpreted as no losses in frequency occurred between the sending and receiving end. In general, the conclusion is similar to the first graph where only receiving frequencies were evaluated; i.e. DDS is better than TCPROS and MQTT for constrained network, whereas MQTT performed rather poorly in multi-MAV and under low bandwidth.

	MQTT	DDS	ROSTCP	MQTT	DDS	ROSTCP
	100Mb	100Mb	100Mb	1Gb	1Gb	1Gb
1 Drone	43%	67%	19%	102%	99%	101%
2 Drones	0	27%	6%	0	93%	100%
3 Drones	0	17%	7%	0	98%	97%

Table 8.1: Efficiency of the communication protocols transfer rates.

As the number of MAV increase, the number of packets transmitted also increase on the network, as illustrated in Fig. 8.12. Hence, the throughput, i.e. rate of successful message delivery in a network, is decreased. If more data is transmitted than the available bandwidth, loss or delay of the packets will occur. As shown, the messaging solution utilised and overheads of the message packet headers is a main factor impacting data transmission performance in a crowded network. There are however other possible solutions to this problem: 1) decreasing the frequency of the data transfer, or 2) decreasing the quality of the stream, e.g. by compression. Network configurations like Maximum Transmission Unit (MTU) can also affect the rate of data transfer.

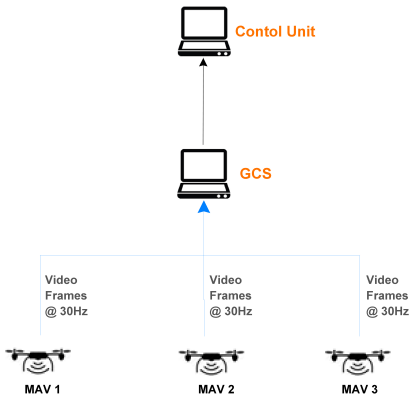


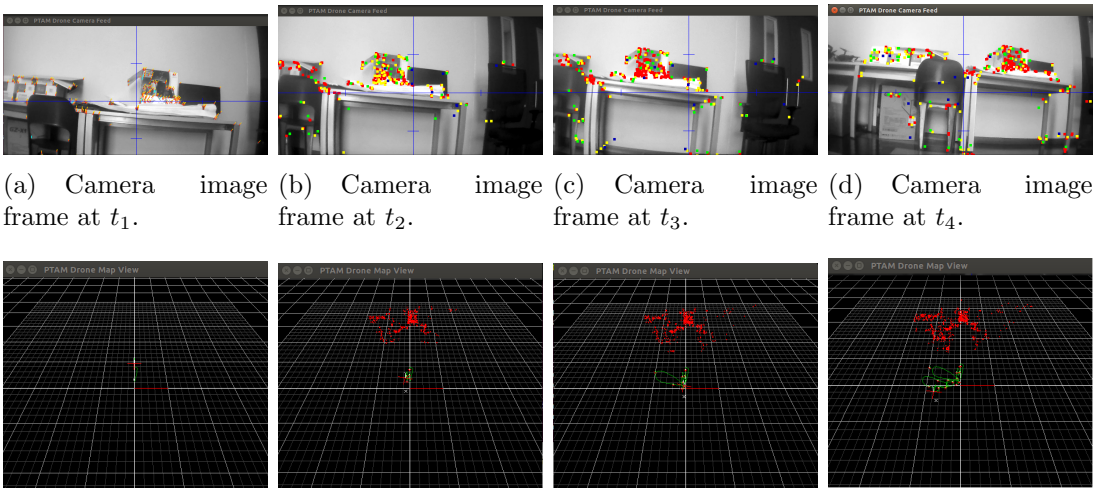
Figure 8.12: Video streams transmission frequencies in a distributed system with 3 MAVs.

The data transport frequency and latencies have a significant impact on the EKF-PTAM navigation testbed. Recall from section 7.4.2, the Kalman filter accounts for delays and predicts ahead of time what is the expected position of the drone. A prediction is done every 10 ms based on the IMU measurements and as soon as a video frame is received, at approximately every 55 ms, a better estimate is obtained based on the SLAM. As

described in the distributed setup, the difference between the point in time at which the image is taken and the time at which the corresponding control need to be applied (based on the SLAM-EKF prediction) ranges between 200 and 450 ms. Hence, it is important for the EKF to receive a constant supply, in other words frequency, of video frames at a minimal delay. This helps to preventing this gap from becoming too large, as the prediction will be invalidated by the time the control reaches the drone. The system architecture described from [9] was not meant to handle large delays and low frequencies. If either the frequency drops from 30 Hz to less than 20 Hz or the delays exceed one second, the system experienced serious failures. This situation occurred for MQTT when the system consisted of more than one MAVs. The communication system failed; the frequency (frames per second) was almost zero for these experiments, which in turn led consequently to the failure of SLAM.

8.2 EKF-PTAM prediction model accuracy

This section assesses the state prediction accuracy of the autonomous control system. In the first experiment, the control objective was to hold the position of the quadcopter after takeoff. The steps of the hovering experiment are depicted in Fig. 8.13 and the overall duration of flight was 60 s. Figures 8.13(a) and 8.13(e) show the initialisation step. Within the time interval $\tau = [10\,\text{s}, 15\,\text{s}]$, the drone was pushed from its hovering position to test the prediction model(Figures 8.13(c), 8.13(d), 8.13(g) and 8.13(h)).



(a) Camera image frame at t_1. (b) Camera image frame at t_2. (c) Camera image frame at t_3. (d) Camera image frame at t_4.

(e) PTAM 3D map at t_1. (f) PTAM 3D map at t_2. (g) PTAM 3D map at t_3. (h) PTAM 3D map at t_4.

Figure 8.13: Camera images and PTAM maps for 60 s experiment with the MAV hovering.

The first experiment was done using a non-distributed MAV and navigation system setup, using default TCPROS as the messaging mechanism between the MAV driver and the navigation system components. Fig. 8.14 represents the Kalman filter measurements

for the experiment where the drone was expected to hold a position after being forcefully pushed away. The plot shows accuracy of the prediction model with respect to the horizontal position x.

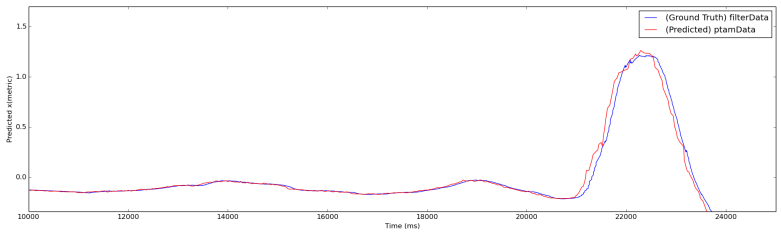


Figure 8.14: Estimated positions of the MAV in non-distributed setup. Communication between MAV driver and navigation system, both running on the same computer (i.e. non-distributed mode).

Fig. 8.14 consists of the following two sets of data collected from the same Kalman filter for one experiment with the GCS and the navigation system running on a local computer:

- The state of the filter without predicting ahead which is based on all available IMU information. The data is represented in *blue*. This is considered the ground truth and is obtained without predicting ahead; hence, approximating the direct IMU reading, referred to as *filterData*.
- The Kalman filter predictions of the drone pose based on the PTAM tracking is represented in *red*. These measurements are referred to as *ptamData*. It should be noted that a major purpose of the state prediction model is to compensate for the delays in the system and send control command signals based on the predicted position.

In Fig. 8.14, when the push actually happens, the Kalman filter could not predict the next state accurately, which is clearly noticeable in the delay between the red and blue lines. The Kalman filter predicts the state based on the best available information i.e. IMU sensors measurements, tracked keyframe and previous control signals. The predication happens within a 10 ms interval, while a new keyframe (used for PTAM pose estimation) is incorporated into the Kalman filter at each interval occurring at 55 ms. Hence, the first experiment above is used to show the 'best-case' of the navigation system allowing to establish a baseline of a working system to compare against a distributed setup.

In a distributed mode, the delays for messaging transmission (including packaging, transmission, marshalling/unmarshalling, protocol hand-shakes, etc.) will add up and influence the ability to estimate the current state and consequently predict the next one. Additionally, the time taken for the control signals to be applied to the MAV is also affected. The other aspects that influence the delays are: 1) the bandwidth of the

network, 2) throughput and processing power of the sender and receiver, as well as 3) bandwidth/processing power of any routers within the network.

Fig. 8.15 shows a similar experiment of the MAV holding a specific position. This time with the MQTT messaging components and delay compensation mechanism. It can be observed that the time taken to recover the target position slightly increased due to the added delays brought into by the messaging system. Nevertheless, the performance is still acceptable for proper functioning of the EKF and ability to execute the SLAM. It can be observed that the state prediction model is also jagged with MQTT. The noticeable delay to recover the correct position (red and blue with larger gap) was still confined within good range. This is an important feature to evaluate qualitatively the functioning of the EKF-SLAM-PID system and assess the delay compensation.

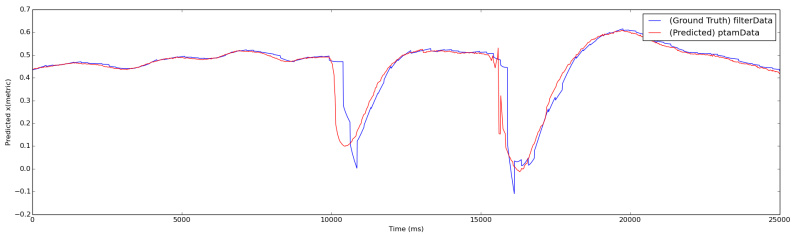
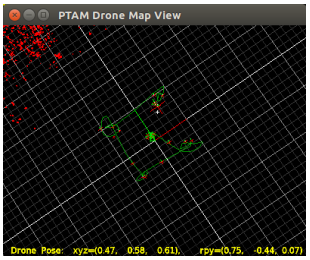
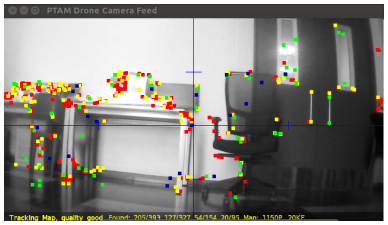


Figure 8.15: Estimated positions of the MAV in a distributed setup. Both estimations of the Kalman filter (*filterData*) and pose prediction based on PTAM (*ptamData*), are done on the remote control unit.

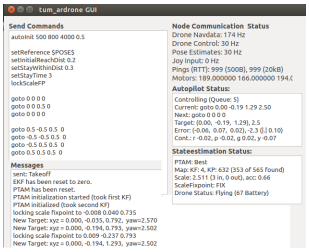
The following set of experiments focused on the DDS version of the gateway. The DDS gateways demonstrated high stability and performance. In the remaining experiments, the control objective focused on servo control mechanism in which the MAV tracks a reference trajectory, using way-points configured through the user interface. Fig. 8.16 outlines the inputs and outputs from one of these experiments. Fig. 8.16(a) shows the EKF trajectory obtained using PTAM tracking. The target trajectory was a 50 cm × 50 cm square. As illustrated in Fig. 8.16(a), it can be easily observed that the measured trajectory from the EKF was within acceptable deviation from the target.



(a) PTAM map for flying a square figure.



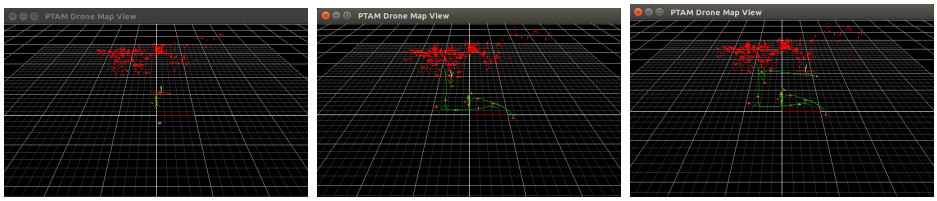
(b) Camera input.



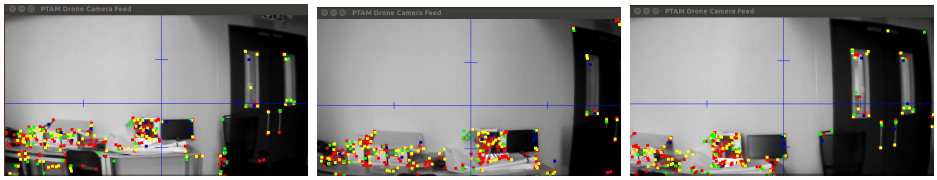
(c) GUI showing the way-points commands and navigation system log messages.

Figure 8.16: MAV flying a square figure of 50 cm × 50 cm.

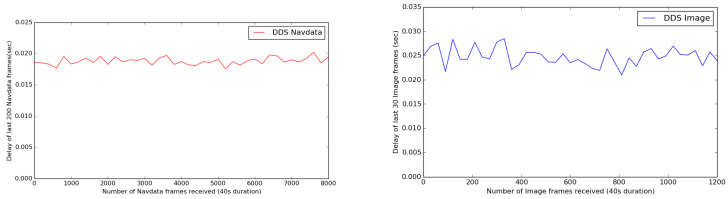
Extending on the previous experiment of flying a specific path, the system was evaluated using one, two, and three MAVs. Fig. 8.17 shows screenshots of the outputs for the EKF trajectory and image frames at various time steps t_1, t_2, and t_3. Fig. 8.17(g) and Fig. 8.17(h) are plots of the measured DDS navigation data and image packets latencies recorded during this experiment.



(a) PTAM map at t_1. (b) PTAM map at t_2. (c) PTAM map at t_3



(d) Image frame at t_1. (e) Image frame at t_2. (f) Image frame at t_3

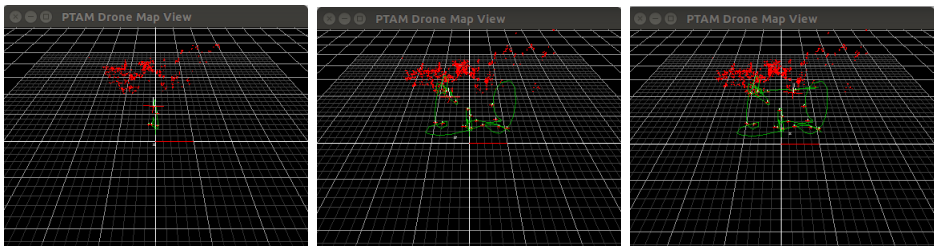


(g) DDS navdata message latencies. (h) DDS image message latencies.

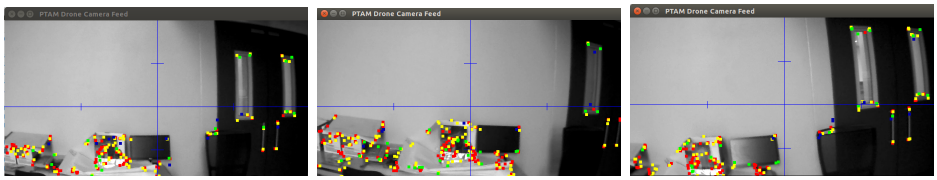
Figure 8.17: Experiment results for one MAV flying a square figure.

The subsequent experiments also consisted of the square trajectory; however, they used two and three MAVs this time, with each of the MAV sending/receiving data to/from the control unit. In these experiments, the second and third MAVs were not flying, but they were kept in the landing position, while the first MAV navigated along the set way-points. Although tests have been conducted with more than one MAV flying along the specific way-points, it was difficult for this experiment to orchestrate the precise motions of two and three MAVs flying together. This difficultly arose due to environmental constraints, associated with the limited space available in the lab.

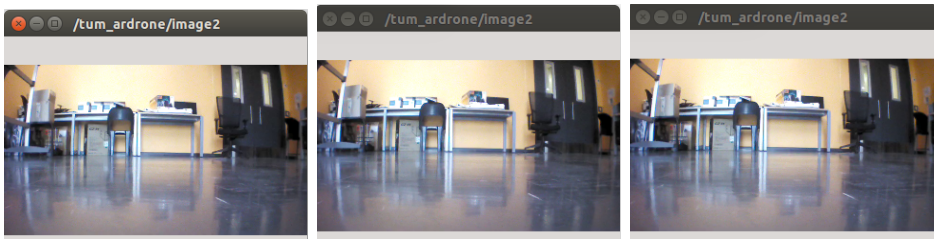
Fig. 8.18 shows the outputs at different time steps for performing the same square figure, but this time with two MAVs. The second row shows output from camera of MAV 1, whereas the third row shows output from the camera of MAV 2. It can be observed that DDS delays were slightly larger (around 5 to 10 ms) than the previous experiment with one MAV.



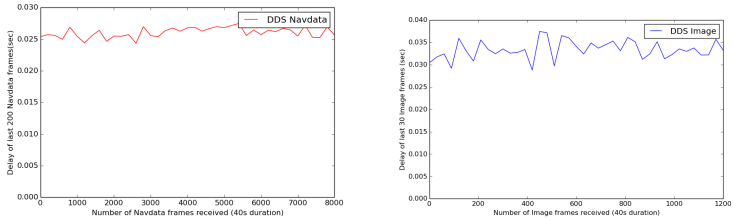
(a) PTAM map at t_1. (b) PTAM map at t_2. (c) PTAM map at t_3



(d) Image frame from MAV 1 at t_1. (e) Image frame from MAV 1 at t_2. (f) Image frame from MAV 1 at t_3



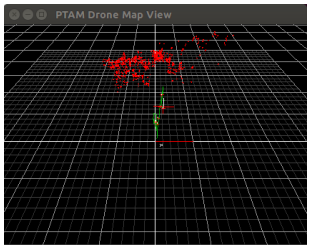
(g) Image frame from MAV 2 at t_1. (h) Image frame from MAV 2 at t_2. (i) Image frame from MAV 2 at t_3



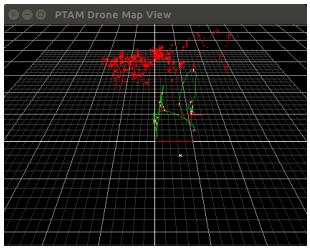
(j) DDS navdata message latencies. (k) DDS image message latencies.

Figure 8.18: Experiment results with two MAVs, and one of the MAV flying a square figure.

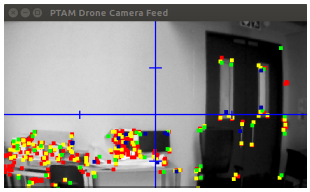
Fig. 8.19 shows the outputs at different time steps for performing the same square figure with three MAVs. The second, third, and fourth rows show outputs from the camera of MAV 1, MAV 2 and MAV 3 respectively. The DDS delays were fairly similar to the previous experiment with two MAVs; hence, there is no major bottleneck in the communication. However, a small drop in the transmission rate –by around 10 frames/sec for navdata– was observed for the configuration of 2 and 3 MAVs. The performance is still within acceptable range for the Kalman filter requirements.



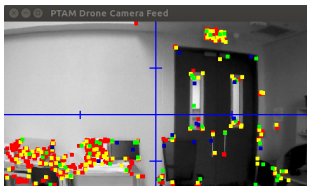
(a) PTAM map at t_1.



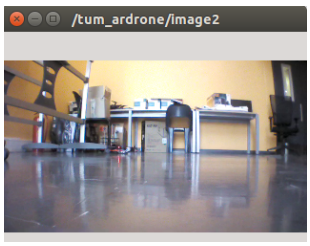
(b) PTAM map at t_2.



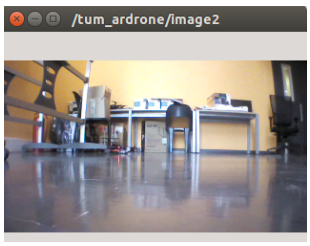
(c) Image frame from MAV 1 at t_1.



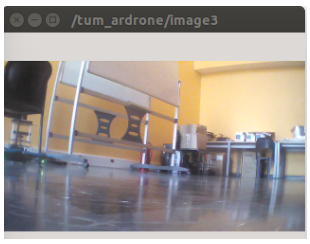
(d) Image frame from MAV 1 at t_2.



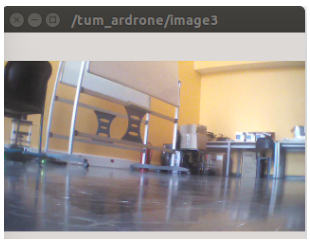
(e) Image frame from MAV 2 at t_1.



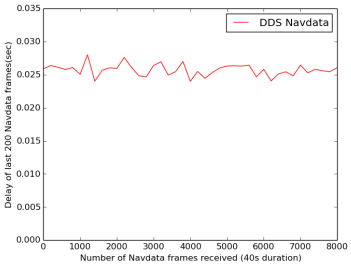
(f) Image frame from MAV 2 at t_2.



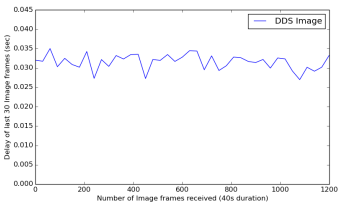
(g) Image frame from MAV 3 at t_1.



(h) Image frame from MAV 3 at t_2.



(i) DDS navdata message latencies.



(j) DDS image message latencies.

Figure 8.19: Experiment results with three MAVs, and one of the MAV flying a square figure.

Chapter 9

Conclusion and Future Work

This study outlined the capabilities of existing communication technologies and their limitations with regards to the implementation of UASs. This research motivates the need for alternative communication technologies to meet performance, resiliency, scalability, heterogeneity, and quality of service aspects of multi-MAV systems. The major findings of this dissertation are based on insights from the practical system realisation and various real-time experiments.

This research work proposed an architecture building blocks of a distributed UAS with MQTT and DDS protocols. This architecture is motivated by the fact that MQTT and DDS are strong options to facilitate hub-spoke model distributed communication. For MQTT, a broker is needed at the data centre, whereas DDS has proven to be slightly more efficient, and definitely proposed a richer set of QoS.

MQTT and DDS also provided very good abstraction at multiple layers of the network OSI model, which can be used to enable larger remote sensing missions of more complex MAVs. It is strongly believed that the implementation of the communication using standard messaging frameworks makes it much easier to deploy and reconfigure MAV missions. This is supported by the fact that both MQTT and DDS are based on standards with multiple available libraries on different languages. Based on our experimental setup, the DDS protocol in particular provided better performance and robustness as compared to MQTT and TCPROS in network constrained environment. On the other hand, MQTT implementation did not yield good results in environments with two and three MAVs in both constrained and high bandwidth environments. So, based on our experiments, DDS would be the recommended protocol for constrained environments. In contrast, DDS and TCPROS were both very reliable within a high bandwidth network of 1 Gbps. To the best of our knowledge, this is the first study in literature that applies and compares MQTT, DDS and TCPROS in the context of a vision-based MAV autonomous navigation application. It is strongly believed that the findings from the experiments of a distributed architecture would enable to implement and further explore single or multi-MAV navigation with other state-of-the-art SLAM algorithms.

Another reason to recommend DDS is because it provides self discovery of the services, and supports both MAV-to-MAV (M2M) and MAV-to-Ground (M2G) communication without the need for a broker. On the protocol side, instead of having to bridge different protocols for M2G and Ground-to-Control Unit, it would be ideal if the communication gateway is integral or being operated on an on-board CPU of the MAV. Under such a situation, DDS would be the preferred option as this would enable M2M/M2G communication, depending on how the network is linked. On the other hand, DDS packets would be able to flow across wireless and/or wired LAN networks, depending on the network topology and set-up, simplifying to a large extent communication across heterogeneous nodes. Long-range wireless networks are much more challenging, with the need to employing a model which defines how an increasing (or decreasing) range impacts the quality and rate of data transmission, based on the facilities the wireless technology provides. The design considerations of DDS shows that it can provide a better solution for the aforementioned problem as well.

Another main contribution of this project is the evaluation of an implemented testbed for an autonomous navigation system based on an extended Kalman Filter (EKF), Simultaneous Localisation and Mapping(SLAM) and a PID controller. The navigation system consumes data and controls a quadcopter at high data transfer rates in real-time. Through experimentation and insights collected during implementation of an aerial navigation system, it became clear that the real-time constraints and the adopted SLAM approach depend on a variety of factors, in addition to the types of applications. There are of course various other forms of SLAM algorithms and alternative approaches that have been presented by the research community and platforms like the OpenSlam [105]. The use of non-parametric filters such as the particle filter is another interesting SLAM approach providing ability to process dense maps. SLAM has evolved to a more efficient and larger scale mapping with higher-end sensors and ability to perform 'loop-closure' by detecting the places which were previously visited. The Parallel tracking and Mapping (PTAM) solution utilised is limited by the number of key frames added to the scene. The map, composed of N key frames has a computation complexity of $O(N^3)$, and hence does not scale efficiently for an increased size of the map.

A further contribution of this dissertation is assosicated with the findings related to the rate and bandwidth of the communication for the time critical vision-based algorithm. For an application using a frame-to-frame processing like SLAM, the data transfer rate is dependent on several factors proven by this study. However, there are other factors that would need to be considered, such as for example the type of environment, speed of the vehicle, capability of sensors, among many others. The thesis contributions enable to apply the architecture concepts and extend this work to alternative algorithms that can run on super computers to meet the needs of newer aerial systems.

An obvious extension of this work could be to evaluate a multi-MAV distributed architecture in a specific real-life scenario with a large scale SLAM, distributed SLAM (DSLAM) or computer vision algorithm with real-time/near real-time data processing needs. The concept of fusing multiple sensors using distributed SLAM for a swarm of cooperating aerial robots is a relatively unexplored field. This situation gets much more

complicated and interesting in a multi-MAV scenario as there is need to manage and orchestrate actions concurrently based on the MAVs motions or possibly interactions with the environment. The purpose of the distributed system proposed is primarily to enable multi-MAV applications. While, a functioning multi-MAV distributed architecture was implemented with three MAVs, managing the communication requirements in real-life would present other major challenges. This study showed that the frequency and bandwidth of the communication is crucial for an EKF-based SLAM system. However, this study could be extended to a dynamic solution where the frequency of each MAV is adapted to make optimal use of the available bandwidth supported by the network. Alternatively, a solution could be built where the frequency is equally shared among all the MAVs.

Another possible future direction could aim to look at an application framework based on the functional programming paradigm for aerial robots communication. This would focus on a different aspect rather than focusing on the lower levels of the OSI model for robots and MAV communication. This direction is motivated based on the assumption that technologies in machine communication will continue to evolve in a non-standard way. Along the same lines, multi-robots need to bridge connections from unconventional networks to conventional networks and control units. Therefore, focusing on a higher-level application model with a basis of a scalable communication framework can lead to state-of-the-art real life M2M - M2G MAV applications. Further work on the architecture model, illustrated throughout this dissertation will focus on the development of fusing data from multi-agent systems and implementation of a decentralised scalable cooperative network of MAVs.

Finally, an interesting direction would be related to employing Artificial Intelligence within the scope of scalable robot communication. For example, [106] compared, in terms of scalability and reliability, two functional programming languages: ROS and Erlang. The authors argue that an increase in the process scalability by a factor of 3.5 was possible using Erlang. This improvement was owned to the lower memory consumption associated with the face tracking experiment. Elixir is another interesting functional programming language that can be used, with capabilities to enable Artificial Neural Networks and multi-core parallel computing. Generally speaking, Elixir abstracts the complexity of building fault-tolerant applications. Such capabilities are becoming the de-facto requirements for future systems of intelligent single, few or swarm autonomous MAVs. Therefore, this research area could be a possible extension to this dissertation.

Appendix A

Technical Documentation

A.1 A.R. DRONE 2.0 PLATFORM

The AR Drone2 off-the-shelf platform coordinate system is described in Figure A.1. Some technical specifications of the AR.Drone include:

- CPU: 468MHz ARM9 embedded micro-controller
- RAM: 128MB
- Interfaces: USB(For GPS module) and Wi-Fi 802.11b/g
- Front camera: VGA sensor with 93 lens Vertical camera: 64 lens, recording up to 60fps

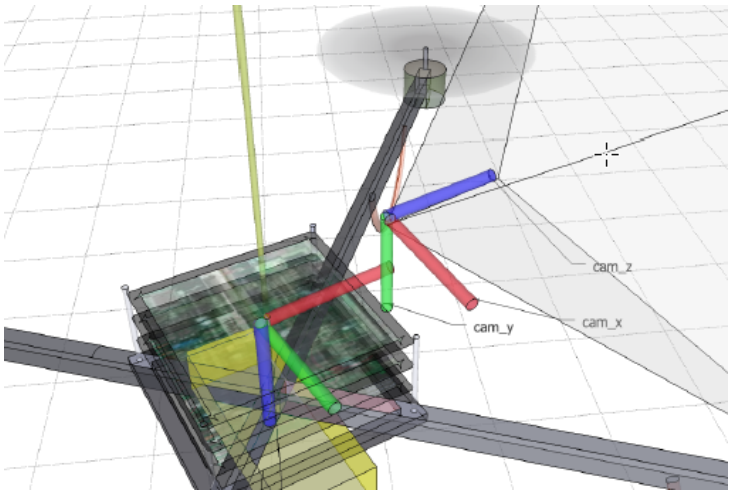


Figure A.1: Parrot AR Drone2 IMU and camera coordinate system: red corresponds x, green to y and blue to z coordinates respectively.

Navdata is one of the most important data structure from the MAV. It consists of Euler angles, linear velocities, accelerations and altitude values as described in Table A.1.

navdata	type	Description	Unit
batteryPercent	float32	0 to 100	%
rotX	float32	left/right tilt	°
rotY	float32	forward/backward tilt	°
rotZ	float32	orientation,yaw	°
altd	float32	estimated altitude	m
vx	float32	linear x velocity	m/s
vy	float32	linear y velocity	m/s
vz	float32	linear z velocity	m/s
accx	float32	body x acceleration	m/s^2
accy	float32	body y acceleration	m/s^2
accz	float32	body z acceleration	m/s^2
gyrox	float32	angle rate about x axis	$°/s$
gyroy	float32	angle rate about y axis	$°/s$
gyroz	float32	angle rate about z axis	$°/s$
tm	float32	Time stamp from ardrone	sec
header	Header	ROS header[1]	

Table A.1: Parrot AR Drone2 *navdata* data structure.

A.2 AR-DRONE IP AND WIFI SETTINGS

The Parrot AR-Drone broadcasts an unsecured WiFi after starting up. The drone is assigned the static IP of 192.168.1.1 and clients are given a DHCP address after connecting to the drone's in-built WiFi. However, this default mechanism has been modified to enable multiple AR-Drones to connect to a network.

Note: This is not advised and can void the warranty of the Parrot AR Drone 2.0, however was necessary for experiments involving multiple MAVs.

The following steps have been taken to change the drone's IP and make it connect to a self-managed WiFi network router:

1. The WiFi router is configured with IP 192.168.1.1 and its settings are modified to broadcast an unsecured WiFi network (for e.g. witsdnet).

2. Telnet into the Parrot AR-Drone and a file named *wifi.sh* was created. The script file should be made executable. The contents of the file are as follows:
killall udhcpd

ifconfig ath0 down
iwconfig ath0 mode managed essid witsdnet
ifconfig ath0 192.168.1.11 netmask 255.255.255.0 up

3. The file need to be executed from the PC as follows:
   echo "./wifi.sh" | telnet 192.168.1.1
   After executing this script, the WiFI connection to PC will be dropped and the drone will connect to the WiFi router instead.

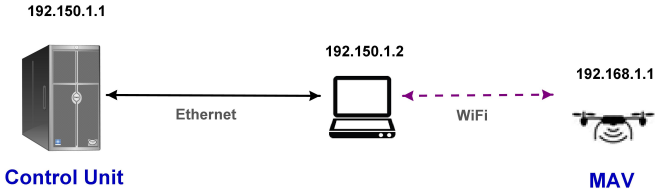
A.3 NETWORK LAYOUT

The network configuration can be the source of biggest instability for experiments involving multiple MAVs using both Ethernet and wireless WiFi networks. Figure A.2 shows the different network layouts that have been experimented throughout this project. The first layout (Figure A.2(a)) involves a point-to-point connection between the client(Ground Station) and server (Control Unit). The Ground station is connected through the WiFi LAN to the MAV. This worked fine as the P2P Ethernet card link provided high bandwidth, not involving any router.

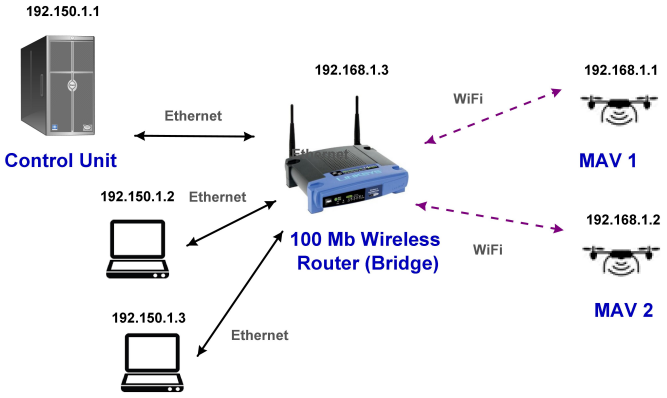
Figure A.2(b) involved the use a Linksys Wireless-G Broadband Router. The parrot AR Drone emits a WiFi signal for clients to connect to, so the router need to be configured to act as the client using replacement firmware Tomato[1]). However with this setup, the 100Mb Wireless router caused a severe bottleneck of messages, due to limited bandwidth to transmit video and navdata streams. A 1Gb wireless router would have worked fine, but was not available.

All the project experiments and data collection were conducted using setup described in Figure A.2(c). This involved a simple 1Gb switch able to cater for the high load of data between the clients and server.

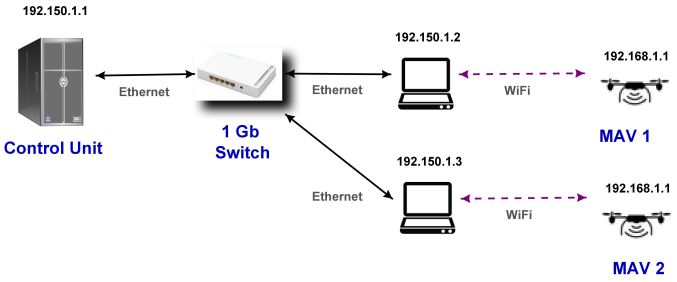
[1]http://www.polarcloud.com/tomato



(a) Simple one MAV setup with peer-to-peer connection of PC and Control Unit.



(b) Two MAVs connected via a Wireless router as bridge

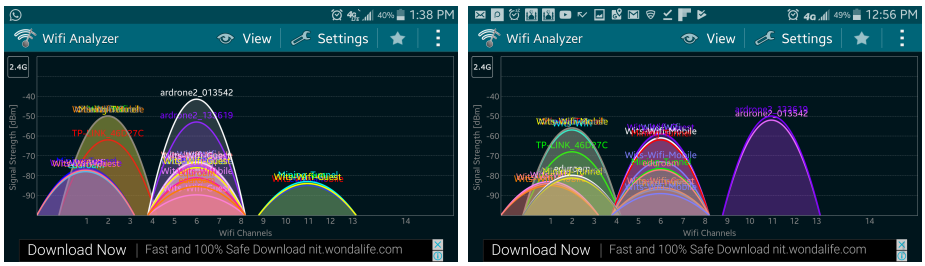


(c) Two MAVs connected via an Ethernet switch

Figure A.2: Network setup layouts for a single and two MAVs.

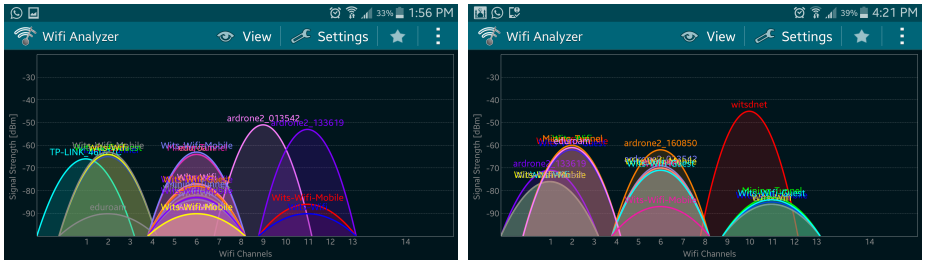
A major source of instability is the quality of WiFi connection, which is heavily influenced by other nearby WiFi networks and the bandwidth they use. This decreased range and caused intermittent connection to the MAV. To alleviate the problem, the WiFi channel of the parrot drone using a shell script[2]. Figure A.3 shows the outputs and effect of changing the WiFi channels (a)-(c), as well as using a WiFi router in (d).

[2]https://www.drone-forum.com/forum/viewtopic.php?t=2291



(a) Two Parrot AR Drones with original WiFi channel settings.

(b) Both drones WiFi channels changed to 11



(c) One drone with WiFi channel changed to 9, and second drone with WiFi channel changed to 11.

(d) Three drones connected to router's WiFi 'witsdnet'. No change to channels required.

Figure A.3: WiFi channels analyser for two Parrot AR Drones in university lab.

A.4 MQTT AND DDS GATEWAYS

This part will give an overview of the MQTT and DDS libraries used by the gateway at the ground station and control unit. It will describe the software libraries and technical designs for each publish/subscribe protocol. All software has been implemented on Ubuntu 14.04 LTS (Trusty) release and ROS Indigo. A notable difference between the two communication paradigm, is the fact that MQTT requires a broker, whereas, DDS has a broker-less architecture and multicasts messages in M2M mode.

A few MQTT implementations were tested throughout this project. The following describe the issues and findings for each one of them:

1. Java:
   A simple MQTT client was implemented using the Java Paho library[3] and the YADrone[4] open-source AR drone project. However, the implementation gave unsatisfactory results and consumed high memory. This was just done as a proof-of-concept, available here[5].

[3]https://eclipse.org/paho/clients/java/
[4]https://vsis-www.informatik.uni-hamburg.de/oldServer/teaching//projects/yadrone/
[5]https://github.com/kamilsss/yadrone

2. JavaScript:
   The HiveMQ JavaScript client library[6] is an interesting test to show how a web browser client can send messages to a cloud hosted MQTT broker. In this test, a simple web page with the MQTT JavaSript client was used to consume and render telemetry and image being sent from the drone to the cloud server. The MAV data could be visualised in real-time on the web dashboard implementation[7]. Note that this proof-of-concept did not include bi-directional data transmission.

3. C:
   An MQTT C client was tested based on the AR Drone SDK test client developed by Parrot [8]. This implementation gave better performance than the Java and JavaScript client implementation results. Furthermore, it demonstrated a successful transmission of high-rate and volume of MAV messages using MQTT.

4. Python:
   The python implementation was forked from an existing ROS-MQTT bridge implementation[9]. The project used the generic configurable MQTT and ROS topics, which is easy to parametrise, however, it consisted of a limited number of data types. This base was used to implement a version of the gateway for *Navdata* and *Image* communication[10].

5. C++:
   C++ *libmosquitto* library[11] has been the preferred MQTT implementation, compatible with the *ardrone_autonomy*[12] ROS package and provides the right object-oriented and thread mechanism to handle high-rate and volume of data over the TCP connections.

The implementation for the MQTT gateway can be found here with code documented:

https://github.com/kamilsss/mqtt_gateway_cpp

The implementation for the DDS gateway can be found here, with code documented:

https://github.com/kamilsss/dds_gateway_cpp

The DDS library used was RTI Connext DDS Professional. An academic licence was obtained from 'REAL-TIME INNOVATIONS, INC' with following details:

- LAC(LICENSE ACKNOWLEDGEMENT CERTIFICATE) #: 17-001250
- Date: May 8, 2017

[6]http://www.hivemq.com/blog/mqtt-client-library-mqtt-js
[7]https://github.com/kamilsss/unmandio
[8]Parrot AR Drone SDK: Hosted on https://github.com/kamilsss/ARDroneSDK2
[9]http://wiki.ros.org/mqtt_bridge
[10]https://github.com/kamilsss/mqtt_bridge_python
[11]https://mosquitto.org/man/libmosquitto-3.html
[12]http://wiki.ros.org/ardrone_autonomy

- RTI Customer ID: WITS01-UP
- Customer: University of Witwatersrand
- Project: Drones Communication System
- Project Type: Research Project
- Site: South Africa

The MQTT implementation is based on open-source mosquitto v1.4.10 broker and libmosquitto client library.

As discussed, the MQTT implementation consists of a buffering mechanism that allowed the rate of reading and writing data to topics to be controlled using a parameter. In the case of DDS, this was not required, however the implementation was not fully tested with variable data rates.

A.5 EKF-SLAM-PID NAVIGATION SYSTEM

The ROS Indigo distribution has been used. The ROS nodes for the navigation algorithm and communication gateways have been compiled under this version of ROS and might need to be adapted for other ROS versions.

The TUM Ardrone project implemented on ROS in [9] was adapted for the purpose of this research. The original version of the code implementation and documentation can be found here:

http://wiki.ros.org/tum_ardrone

A fork and update to the TUM implementation for the purpose of this project is hosted on this repository:

https://github.com/kamilsss/tum_ardrone

A.6 EKF CALIBRATION

Parameters c1 to c8 for the EKF can be calibrated using a recorded ROS Bag as a simulator. The simulator can be run using the following command:

rosbag play -l simulatorFlight.bag

Two state-estimation nodes need to be started, one with control grains OFF and one using PTAM. ROS dynamic reconfigure tool is able to show graphically and plot the time

series values during the simulation. The parameters of the system that can be changed are editable. Each parameter c_i need to be calibrated in order to have a best aligned graph.

The following summarise the commands to be executed:
rosrun tum_ardrone drone_stateestimation __name:=drone_stateestimationn2 /ardrone/predictedPose:=/ardrone/predictedPose2 (Node with remapped topic names and output)
rosrun tum_ardrone drone_stateestimation (Node with PTAM 'ON')
rosrun rqt_reconfigure rqt_reconfigure (ROS Dynamic reconfigure GUI)
rqt_plot /ardrone/predictedPose/dx,/ardrone/predictedPose2/dx (Plot to compare alignment of graphs based on parameter calibration)

A.7 DATA SET FOR COMMUNICATION GATEWAY AND SLAM NAVIGATION SYSTEM

The data sets for experiments can be found here:

https://github.com/kamilsss/data_sets_ros_logs

A.8 DDS, TCPROS, MQTT MEASUREMENTS FOR COMMUNICATION LATENCIES FOR ONE, TWO AND THREE DRONES

The measurements below are for Image data only. Fig. A.2 list the values recorded and are based on either 100 Mb LAN or 1Gigabit LAN setups, on one of the three protocols: DDS, TCPROS or MQTT, and for one to three drones. 'DDS1' refers to DDS latency for the master drone with one drone and DDS as communication mechanism. 'DDS2' refers to DDS latency and bandwidth for the master drone, with two drones in the system. 'DDS3' is an experiment with three drones in the system. The same applies for the other protocols.

The 'Sent' frequency or bandwidth is the rate at which data is sent from the GCS. The 'Received' is the rate at which images are received over the topic at the control unit.

Each data point (cell) in the table is an average over five experiments of 60 seconds.

	Throughput (Router statistics)				Frequency (Hz)				Bandwidth (Mb/s)			
	100 Mb		1Gb		100 Mb		1Gb		100 Mb		1Gb	
	Sent (Mbps)	Received (Kbps)	Sent (Mbps)	Received (Kbps)	Sent	Received	Sent	Received	Sent	Received	Sent	Received
DDS1	11.7	8	21.2	14.4	29	20.1	30.4	29	21.4	12.1	20.8	21
DDS1	11.7	8	20.8	15.2	30	14.5	29.3	30	21.4	14.1	20.6	20.2
DDS1	11.7	8.4	20.5	13.1	29	22.1	29.6	30.2	20.9	15.6	21.4	19.7
DDS1	11.7	8.2	20.6	14.7	28.2	19.4	29.7	29.6	21.2	11.2	22.2	21
DDS1	11.7	8	20.3	14.5	29	21.3	29.9	29	21.4	13	21.8	21.4
DDS2	11.7	8	42	29	28	8.5	30.6	29	19.4	5.9	21.2	20.3
DDS2	11.7	8.4	63	43.4	28	9	29.1	20.8	19.2	5.9	21.1	20.4
DDS2	11.7	8.5	42	29.2	29.3	8.5	29	30	19.9	0.6	21.2	20.8
DDS2	11.7	8.3	83	58	28.1	4.3	30	29	19.8	3	21.4	20.4
DDS2	11.7	8.2	42	29	30	8.2	30	30.1	21.3	5.9	21.3	20.6
DDS3	11.7	8.4	61	43	28	5.7	30	27	16	3.9	21.1	19.5
DDS3	11.7	8	61.2	41	26.1	3.7	30	30	14	4	21	20.5
DDS3	11.7	8.4	60.4	45	28.2	4.6	29.1	29.7	14.3	2.6	21.2	20.6
DDS3	11.7	8.2	58.1	41.2	29	4.4	30.2	28.1	15	4.4	21.5	21.1
DDS3	11.7	8	60.4	44	27.2	5	28.5	29.6	14.1	3.6	21.2	20.4
ROSTCP1	11.7	8.4	63	43	30	4.6	30.5	30.5	21.2	2	21.2	21.4
ROSTCP1	11.7	8.6	61.2	41.2	30	7.1	30.5	30.4	21.3	0.7	21.3	21.2
ROSTCP1	11.7	8.1	60.2	40	30.2	4.6	28	29.2	21.1	3.5	21.1	21
ROSTCP1	11.7	8	45.4	45	29.5	5.5	29.5	29.9	20.4	2.5	20.4	20.6
ROSTCP1	11.7	8.2	58	43.2	29.2	6	30.1	30.2	20.3	2.1	21.2	21
ROSTCP2	11.7	8	85	58	30	1.7	30	30	21	2	21	21
ROSTCP2	11.7	8	85.2	59	30.1	2	29.9	30	21	1.6	21.4	20.6
ROSTCP2	11.7	8.1	85	59	30	2.7	30	30.5	21.2	1.3	21	20.8
ROSTCP2	11.7	8.2	84.4	56	30.5	1.3	30.2	30.1	20.9	2.3	20.6	20.4
ROSTCP2	11.7	8.4	84.4	57	30	1.8	30	30	20.9	2.2	21.1	21.2
ROSTCP3	11.7	8.5	100	75	29.2	1.9	29	29	20.4	1.3	20.2	19.2
ROSTCP3	11.7	8.6	86	73	29	2.2	28.4	26	19.8	1.4	19.6	19.3
ROSTCP3	11.7	8	102	73	27	2.8	28.3	28	19.6	1.4	19.5	19.3
ROSTCP3	11.7	8	101	78	25	1.6	30	29	17.4	1.3	20.2	20
ROSTCP3	11.7	8.2	93	72	25.1	1.6	29.2	29.1	17.3	0.6	20.4	19
MQTT1	11.7	8.3	21	14	30	6	30.7	30.6	21	11.7	21.3	21.1
MQTT1	11.7	8.3	21	15	30	16	30.5	30.3	21.2	11.7	21	21.3
MQTT1	11.7	8.4	23.4	13.2	28.2	12	27.3	29.6	20.5	10.5	21	22
MQTT1	11.7	8.2	24	13.4	27	14.4	30.2	30	20.2	10.6	21.2	19
MQTT1	11.7	8	24.2	14.1	29	13	29	30	20.4	11	21.1	18
MQTT2	FAILED											
MQTT2	FAILED											
MQTT2	FAILED											
MQTT2	FAILED											
MQTT2	FAILED											

Table A.2: Average throughput, frequencies and bandwidth measurements for set of experiments.

Bibliography

[1] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "THE PIXHAWK OPEN-SOURCE COMPUTER VISION FRAMEWORK FOR MAVS," pp. 13–18, 2012.

[2] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, "The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)," in *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, vol. 2. IEEE, 2004, pp. 12—-E.

[3] "3DR-X8 Specifications," 2016. [Online]. Available: https://3dr.com/wp-content/uploads/2017/03/X8-M-Spec-Sheet-v3.pdf

[4] M. E. Jefferies and W. K. Yeap, "Robotics and Cognitive Approaches to Spatial Mapping," *Robotics and Cognitive Approaches to Spatial Mapping*, vol. 38, no. January 2008, pp. 1–5, 2008. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-75388-9$\delimiter"026E30F$nhttp://www.springerlink.com/content/c3q1577r20602x59/

[5] "Omg dds standard," accessed: 29-Mar-2016. [Online]. Available: http://portals.omg.org/dds/what-is-dds-3/

[6] H. Durrant-Whyte and T. Bailey, "1simultaneous localisation and mapping (slam): Part i the essential algorithms," 2006.

[7] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[8] jsturm, "Lecture notes on feedback control, course Autonomous navigation for flying robots," 2015. [Online]. Available: http://jsturm.de/publications/data/lecture_4_part_4.pdf

[9] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 2815–2821. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6385458$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6385458

[10] "3D Robotics," 2015. [Online]. Available: https://3dr.com/

[11] "Sensefly," 2016. [Online]. Available: https://www.sensefly.com/drones/ebee.html

[12] "DJi," 2015. [Online]. Available: http://www.dji.com/

[13] P. Pace, G. Aloi, G. Caliciuri, and G. Fortino, "A Mission-Oriented Coordination Framework for Teams of Mobile Aerial and Terrestrial Smart Objects," *Mobile Networks and Applications*, pp. 1–18, 2016.

[14] S. Gusmeroli, S. Piccione, and D. Rotondi, "Iot@ work automation middleware system design and architecture," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–8.

[15] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 931–936.

[16] L. Tan and N. Wang, "Future internet: The internet of things," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 5. IEEE, 2010, pp. V5–376.

[17] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[18] "Uas applications," Mar 2015. [Online]. Available: http://www.ccuvs.com/industry/uas-applications/

[19] R. Austin, *Unmanned Aircraft Systems - UAVS Design, Development and Deployment*, 2010.

[20] K. Ramalingam, R. Kalawsky, and C. Noonan, "Integration of Unmanned Aircraft System (UAS) in non-segregated airspace: A complex system of systems problem," in *2011 IEEE International Systems Conference, SysCon 2011 - Proceedings*, 2011, pp. 448–455.

[21] "Mqtt standard pub/sub messaging protocol)." [Online]. Available: http://mqtt.org/

[22] "Data distribution service(dds) standard middleware protocol)." [Online]. Available: http://portals.omg.org/dds/

[23] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: Architecture, challenges and applications," *IEEE Network*, vol. 26, no. 3, pp. 21–28, 2012.

[24] G. Ermacora, A. Toma, S. Rosa, B. Bona, M. Chiaberge, M. Silvagni, M. Gaspardone, and R. Antonini, "A Cloud Based Service for Management and Planning of Autonomous UAV Missions in Smart City Scenarios," in *Modelling and Simulation for Autonomous Systems*, 2014, vol. 8906, pp. 20–26. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-13823-7{_}3

[25] S. G. Gupta, M. M. Ghonge, and P. M. Jawandhiya, "Review of Unmanned Aircraft System (UAS)," *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 2, no. 4, pp. 1646–1658, 2013. [Online]. Available: http://ijarcet.org/wp-content/uploads/IJARCET-VOL-2-ISSUE-4-1646-1658.pdf

[26] R. S. Stansbury, M. A. Vyas, and T. A. Wilson, "A survey of UAS technologies for command, control, and communication (C3)," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 54, pp. 61–78, 2009.

[27] "Qgroundcontrol)," accessed: 19-Mar-2015]. [Online]. Available: http://qgroundcontrol.org/

[28] "Wild life monitoring," 2016. [Online]. Available: http://theconversation.com/satellites-mathematics-and-drones-take-down-poachers-in-africa-36638

[29] "Mapping and surveying," 2016. [Online]. Available: https://www.aibotix.com/industries/surveying-and-mapping

[30] "Law enforcement," 2016. [Online]. Available: http://theconversation.com/satellites-mathematics-and-drones-take-down-poachers-in-africa-36638

[31] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor MAV," in *IEEE International Conference on Intelligent Robots and Systems*, 2012, pp. 4557–4564.

[32] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2010, pp. 21–28.

[33] K. Schauwecker and A. Zell, "On-board dual-stereo-vision for autonomous quadrotor navigation," in *2013 International Conference on Unmanned Aircraft Systems, ICUAS 2013 - Conference Proceedings*, 2013, pp. 333–342.

[34] S. Lange, N. Sunderhauf, and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," in *Proceedings of the International Conference on Advanced Robotics*, 2009, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=5174709$\delimiter"026E30F$nhttp://www.dis.uniroma1.it/lmarchetti/private/quadrotor/lange-vision-based-onboard-approach-landing-position-control-UAV-gps-denied-environments.pdf

[35] D. Eberli, D. Scaramuzza, S. Weiss, and R. Siegwart, "Vision based position control for MAVs using one single circular landmark," in *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 61, no. 1-4, 2011, pp. 495–512.

[36] D. I. Montufar, F. Munoz, E. S. Espinoza, O. Garcia, and S. Salazar, "Multi-UAV testbed for aerial manipulation applications," in *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*, 2014, pp. 830–835.

[37] E. Asmare, A. Gopalan, M. Sloman, N. Dulay, and E. Lupu, "A mission management framework for unmanned autonomous vehicles," *MobileWireless Middleware, Operating Systems, and Applications*, pp. 222–235, 2009.

[38] J. Allred, A. B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni, "Sensorflock: an airborne wireless sensor network of micro-air vehicles," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007, pp. 117–129.

[39] J. Barnard, "Small UAV Command, Control and Communication Issues," in *Communicating with UAV's, 2007 IET Seminar on*, 2007, pp. 75–85.

[40] "Kumar robotics," accessed: 24-Mar-2016]. [Online]. Available: http://www.kumarrobotics.org/research/

[41] S. Edwards, "3d reconstruction using Kingfu and RGBD mapping," 2016.

[42] "Precision agriculture," 2016. [Online]. Available: http://dronelife.com/2014/12/30/5-actual-uses-drones-precision-agriculture-today/

[43] "Minimally manned installations inspection ," 2016. [Online]. Available: http://www.nytimes.com/2013/02/16/technology/rise-of-drones-in-us-spurs-efforts-to-limit-uses.html?pagewanted=all&_r=0

[44] "Disease prevention," 2016. [Online]. Available: http://www.bgr.in/news/microsofts-drones-to-catch-mosquitoes-and-help-stop-epidemics/

[45] "Climate monitoring," 2016. [Online]. Available: http://www.trust.org/item/20130902130214-y9f47/

[46] "Search and rescue," 2016. [Online]. Available: https://www.dronesetc.com/blogs/news/drones-being-used-for-search-and-rescue

[47] "Disaster recovery and emergency," 2016. [Online]. Available: http://www.theguardian.com/global-development/2015/jul/28/drones-flying-in-the-face-of-disaster-humanitarian-response

[48] "Matternet," accessed: 24-Mar-2016]. [Online]. Available: https://mttr.net/

[49] "DronePort project," 2016. [Online]. Available: http://www.fosterandpartners.com/news/archive/2016/09/norman-foster-on-the-droneport-project-featured-in-unops-publication/

[50] "Drone regulations," accessed: 24-Mar-2016]. [Online]. Available: https://www.loc.gov/law/help/regulation-of-drones/regulation-of-drones.pdf

[51] I. Bekmezci, O. K. Sahingoz, and a. Temel, "Flying Ad-Hoc Networks (FANETs): A survey," pp. 1254–1270, 2013.

[52] L. Gupta, R. Jain, and G. Vaszkun, "Survey of Important Issues in UAV Communication Networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1123–1152, 2016.

[53] S. Rosati, K. Kruzelecki, G. Heitz, D. Floreano, and B. Rimoldi, "Dynamic Routing for Flying Ad Hoc Networks," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2015. [Online]. Available: http://arxiv.org/abs/1406.4399$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7069210

[54] M. Varga, M. Basiri, G. H. M. Heitz, and D. Floreano, "Distributed Formation Control of Fixed Wing Micro Aerial Vehicles for Uniform Area Coverage," {*IEEE*}/{*RSJ*} {*I*}*nternational* {*C*}*onference on* {*I*}*ntelligent* {*R*}*obots and* {*S*}*ystems*, vol. 669, no. 1, 2015.

[55] R. Purta, M. Dobski, A. Jaworski, G. Madey, F. Hall, and N. Dame, "A Testbed for Investigating the UAV Swarm Command and Control Problem Using DDDAS," 2005.

[56] M. Asadpour, B. den Bergh, D. Giustiniano, K. Hummel, S. Pollin, and B. Plattner, "Micro aerial vehicle networks: An experimental analysis of challenges and opportunities," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 141–149, 2014.

[57] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications," *Communications Magazine, IEEE*, vol. 53, no. 9, pp. 48–54, 2015.

[58] Y. Zhou, N. Cheng, N. Lu, and X. S. Shen, "Multi-UAV-Aided Networks: Aerial-Ground Cooperative Vehicular Networking Architecture," *IEEE Vehicular Technology Magazine*, vol. 10, no. 4, pp. 36–44, 2015.

[59] E. A. Marconato, D. F. Pigatto, K. R. Branco, and L. H. C. Branco, "LARISSA: Layered architecture model for interconnection of systems in UAS," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on.* IEEE, 2014, pp. 20–31.

[60] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman, "A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation," in *Control.* Springer, 2004, p. 233. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.2399{&}rep=rep1{&}type=pdf

[61] A. H. Goktogan and S. Sukkarieh, "Distributed simulation and middleware for networked UAS," in *Unmanned Aircraft Systems.* Springer, 2008, pp. 331–357.

[62] J. L. Rubio, "Service Oriented Architecture for Embedded (Avionics) Applications," Ph.D. dissertation, Universitat Polit{é}cnica de Catalunya, 2011.

[63] E. Pastor, C. Barrado, E. Santamaria, J. Lopez, and P. Royo, *An open architecture for the integration of UAV civil applications.* Citeseer, 2009.

[64] U. S. o. A. Department of Defence, "UAS Control Segment (UCS) Mission and Objectives." [Online]. Available: https://ucsarchitecture.org/pages/mission-and-objectives{#}objectives

[65] ——, "Open business model for Unmanned Aircraft Ground Control Stations," Department of defence, United States of America, Tech. Rep., 2014. [Online]. Available: https://info.publicintelligence.net/DoD-UAS-OBM.pdf

[66] C. W. Heisey, A. G. Hendrickson, B. J. Chludzinski, R. E. Cole, M. Ford, L. Herbek, M. Ljungberg, Z. Magdum, D. Marquis, A. Mezhirov, and Others, "A Reference Software Architecture to Support Unmanned Aircraft Integration in the National Airspace System," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 41–55, 2013.

[67] V. Cerf, S. Burleigh, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," pp. 1–35, 2007.

[68] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[69] "Robot operating system (ros) wiki," accessed: 24-Mar-2016]. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[70] "Ros2 wiki," accessed: 24-Mar-2016]. [Online]. Available: https://github.com/ros2/ros2/wiki

[71] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[72] European Commission, "Research and Innovation — Internet of Things — Digital Single Market." [Online]. Available: https://ec.europa.eu/digital-single-market/en/research-innovation-iot

[73] A. Foster, "Messaging technologies for the industrial internet and the internet of things," *PrismTech Whitepaper*, 2015.

[74] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, 2005.

[75] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Aaai/iaai*, 2002, pp. 593–598.

[76] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[77] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[78] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[79] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Springer handbook of robotics*. Springer, 2008, pp. 871–889.

[80] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[81] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of intelligent & robotic systems*, vol. 61, no. 1, pp. 287–299, 2011.

[82] I. Sa, H. He, V. Huynh, and P. Corke, "Monocular vision based autonomous navigation for a cost-effective mav in gps-denied environments," in *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*. IEEE, 2013, pp. 1355–1360.

[83] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision*. Springer, 2014, pp. 834–849.

[84] S. Shen, N. Michael, and V. Kumar, "Autonomous indoor 3d exploration with a micro-aerial vehicle," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 9–15.

[85] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for rgb-d cameras," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3748–3754.

[86] M. Meilland and A. I. Comport, "On unifying key-frame and voxel-based dense visual slam at large scales," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3677–3683.

[87] A. Kalantari, K. Mahajan, D. Ruffatto, and M. Spenko, "Autonomous perching and take-off on vertical walls for a quadrotor micro air vehicle," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4669–4674.

[88] S. Lee and Y. Kay, "A kalman filter approach for accurate 3-d motion estimation from a sequence of stereo images," *CVGIP: Image Understanding*, vol. 54, no. 2, pp. 244–258, 1991.

[89] U. Franke, C. Rabe, H. Badino, and S. Gehrig, "6d-vision: Fusion of stereo and motion for robust environment perception," in *Joint Pattern Recognition Symposium*. Springer, 2005, pp. 216–223.

[90] M. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *IEEE International Conference on Intelligent Robots and Systems*, 2011, pp. 5113–5120.

[91] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," in *Springer Tracts in Advanced Robotics*, vol. 79, 2014, pp. 361–373.

[92] Y. Mulgaonkar, B. Araki, J. S. Koh, L. Guerrero-Bonilla, D. M. Aukes, A. Makineni, M. T. Tolley, D. Rus, R. J. Wood, and V. Kumar, "The flying monkey: A mesoscale robot that can run, fly, and grasp," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, 2016, pp. 4672–4679.

[93] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl, "Ar-drone as a platform for robotic research and education," in *International Conference on Research and Education in Robotics*. Springer, 2011, pp. 172–186.

[94] "Parrot ar drone details," accessed: 12-Apr-2015. [Online]. Available: https://en.wikipedia.org/wiki/Parrot_AR.Drone

[95] S. Piskorski, N. Brulez, P. Eline, and F. DHaeyer, "Ar. drone developer guide," *Parrot, sdk*, vol. 1, 2012.

[96] "Mavlink protocol," accessed: 19-Mar-2015]. [Online]. Available: http://qgroundcontrol.org/mavlink/start

[97] J. López, P. Royo, C. Barrado, and E. Pastor, "Applying marea middleware to uas communications," in *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*, 2009, p. 1914.

[98] "Mosquitto broker," accessed: 25-May-2015. [Online]. Available: https://mosquitto.org/

[99] "Hive mq broker," accessed: 25-May-2015. [Online]. Available: http://www.hivemq.com/

[100] "Joram mq broker," accessed: 25-May-2015. [Online]. Available: http://joram.ow2.org/

[101] "Scalaagent mqtt brokers benchmark," accessed: 25-May-2015. [Online]. Available: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

[102] "Ros ar drone autonomy driver," accessed: 29-Mar-2016. [Online]. Available: http://wiki.ros.org/ardrone_autonomy

[103] "Rti connext dds," accessed: 29-Mar-2016. [Online]. Available: https://www.rti.com/products

[104] "libmosquitto mqtt library," accessed: 29-Mar-2016. [Online]. Available: https://mosquitto.org/man/libmosquitto-3.html

[105] "Open slam," accessed: 14-Mar-2016. [Online]. Available: http://openslam.org/

[106] A. Lutac, N. Chechina, G. Aragon-Camarasa, and P. Trinder, "Towards reliable and scalable robot communication," 2016.