DEGREE PROJECT IN MECHANICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

A Practical Approach of an Internet of Robotic Things Platform

ROBERT YOUSIF

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

KTH Industrial Engineering and Management

Master of Science Thesis TRITA-ITM-EX 2018:737

A Practical Approach of an Internet of Robotic Things Platform

Robert Yousif

Approved	Examiner Martin Törngren	Supervisor Jinzhi Lu
	Commissioner ÅF	Contact person Håkan Roos

Abstract

This thesis aims to design and develop a platform based on a novel concept - the Internet of Robotic Things (IoRT) constructed by a robotic platform, an Internet of Things (IoT) platform and cloud computing services. A robotic platform enables hardware abstraction, facilitating the management of input/output between software, mechanical devices and electronic systems. The IoT platform is a global network enabling a massive number of devices known as things to communicate with each other and transfer data over the Internet. Cloud computing is a shared pool of scalable hardware usually provisioned as cloud services by third party cloud vendors. The integration of these concepts constitutes the core of the IoRT platform, as a global infrastructure facilitating robots to interconnect over the Internet utilizing common communication technology. Moreover, the pool of cloud resources shared by the connected robots enables scalable storage and processing power.

The IoRT platform developed in this study constitutes firstly of the Amazon Web Service (AWS) IoT core serving as the IoT platform. Secondly, it incorporates the Robot Operating system (ROS) as the robotic platform and thirdly the cloud services Amazon DynamoDB and AWS Lambda for data storing and data processing respectively.

The platform was evaluated in terms of delays & utilization and visualization capabilities. The platform demonstrates promising result in terms of delays exchanging small packages of data, round-trip delays in order of 50-60ms were obtained between a robot placed in Stockholm and the communication platform AWS IoT placed in Dublin, Ireland. Most of the delay is due to the traveling distance, where a round trip ping between Stockholm and Dublin takes around 50ms. The platforms ability to visualize streaming

data from the robots, enables an operator to visualize selected data from any service in the platform over the Internet in near real-time, with round-trip delays in order of 250-300ms where the data propagates through multiple cloud service. In conclusion, this report illustrates the feasibility of merging two major platforms together: ROS and AWS IoT, and moreover, the accessibility to exploit the power and potential enabled by the modern data centers.

Keywords: Robotics, Internet of Things, Cloud Services, Amazon Web Services, Robot Operating System, Internet of Robotic Things, Real-time processing, Micro-Services

KTH Industrial Engineering and Management

Examensarbete TRITA-ITM-EX 2018:737

Ett praktiskt tillvägagångssätt för en IoT-baserad robotplatform

Robert Yousif

Godkänt	Examinator	Handledare
	Martin Törngren	Jinzhi Lu
	Uppdragsgivare	Kontaktperson
	ÅF	Håkan Roos

Sammanfattning

Avhandlingens syfte är att utforma och utveckla en plattform baserat på konceptet Internet of Robotic Things konstruerat av en robotikplattform, en Internet of Things plattform och molntjänster. En Internet of Things plattform är ett globalt nätverk som tillåter många enheter att kommunicera med varandra och överföra data över Internet. En robotikplattform underlättar kontrollen av in/ut mellan mjukvara, mekaniska enheter och elektroniska system. Molntjänster är en gemensam pool av skalbar hårdvara som vanligtvis erbjuds av tredje parts molnleverantörer. En Internet of Robotic Things plattform är en global infrastruktur som underlättar avancerade robotar att interagera över Internet genom en gemensam kommunikationsteknik, en pool av molntjänster som delas av alla uppkopplade robotar som tillåter skalbar lagring och processorkraft.

Plattformens huvudkomponenter är robotikplattformen Robot Operating System, Internet of Things plattformen AWS IoT Core och molntjänsterna Amazon DynamoDB och AWS Lambda för lagring och databearbetning.

Plattformen evalueras i form av plattformegenskaperna, fördröjningar & funktionstid och visualiseringsförmåga. Plattformen visar lovande resultat i from av fördröjningar mellan två robotar som utbyter data med hjälp av IoT plattformen, där fördröjningarna är begränsade av distanssträckan. Plattformens egenskap att visualisera strömmande data från robotar möjliggör för en operatör att visualisera utvald data från plattformen över internet i realtid.

Acknowledgments

The dedication of this thesis goes to

My supervisor Jinzhi Lu who contributed with wisdom, enthusiasm and feedback and for his advice on systems engineering and communication infrastructure.

My examiner Martin Törngren who contributed with great feedback.

My supervisors Håkan Ros and Azad Nourani at ÅF who gave me this opportunity to work with this project and for welcoming me to ÅF. Their feedback, enthusiasm and bright insights helped a lot throughout the project.

My mother and father for their endless encouragement and support.

Robert Yousif

Contents

Nomenclature

IoT Internet of Things

IoRT Internet of Robotic Things

ROS Robot Operating System

FaaS Function as a Service

AWS Amazon Web Service

OS Operating System

VM Virtual Machine

CPU Control Processing Unit

MQTT Message Queuing Telemetry Transport

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

QoS Quality of Service

NFC Near Field Communication

DDS Data Distribution Service

CAGR Compound Annual Growth Rate

NoSQL Non-relational Standard Query Language

SDK Software Development Kit

API Application Programming Interface

KDD Knowledge Discovery in Databases

RC Remote Controlled

ECU Electronic Control Unit

List of Terminology

- Cloud Computing: A shared pool of scalable hardware provisioned as services [1].
- Micro-Services: A collection of loosely coupled services that work together [2].
- Serverless: A shared pool of hardware dynamically managed by a third party cloud vendor [3].
- Node: A participant in a network.
- Robot: A machine with interconnected sensors, computers and actuators.
- Thing: Passive and interactive edge devices [4].
- On Premises: A computer/server [1].
- Service: A software functionality or a set of software functionalities.
- Composability: A service's ability to be modular and stateless.
  - Modularity: A service's ability to be deployed independently.
  - Stateless: A service's ability to to treat invocations independently.
- Interoperability: A service's ability to exchange data with other services.
- Scalability: A service's ability to scale on demand.
  - Concurrency: A service's ability to scale in number of instances.
  - Elasticity: A service's ability to automatically adapt workloads on demand.
- Near Real-Time: A service's ability to function with a delay that the user sense as immediate. Real-time implies that the delay is controlled and acceptable within a certain range while near real-time underlines that the delay is not guaranteed [5].

Chapter 1

Introduction

1.1 Background

The continuous evolution of robotic, cloud computing and Internet of Things (IoT), triggered by technological advances in hardware, software and communication technologies have led the different converging communities to a beneficial synergy [6]. The communities provide a vast amount of services based on different technologies and distinguishing characteristics. Thus, one might ask, what services do the communities provide and what are the characteristics and technologies enabled by them? Moreover, how do we combine these services?

A robot is essentially one machine utilizing a robotic platform with different interconnected sensors, computers and actuators working together to perform an action. A robotic platform enables hardware abstraction, facilitating interface management between software, mechanical devices and electronic systems. The complexity of an action produced is limited by the processing power available to the robot, a robot dependent on local hardware is therefore restrained in its initial setup in terms of scalability. Migrating the computations of the robot to be processed remotely enables the robots to reduce the weight, cost of hardware and battery consumption. Moreover, it facilitates the availability and storage of the produced data. One solution to remotely process and store data is cloud computing.

Cloud computing refers to a shared pool of scalable hardware provisioned as a cloud service. A cloud service refers to a virtual machine, a function or multiple functions deployed in the cloud i.e micro-services [2]. Micro-services are serverless cloud services managed by a third party cloud vendor such as Amazon Web Service, Google Cloud or Microsoft Azure. Serverless refers to a shared pool of hardware dynamically managed by the cloud vendor, abstracting the hardware from the developer. Using abstracted hardware [7], cloud computing enables a vast amount of scalable processing power and storage capacity to be utilized as centralized functionalities for the robots. Merging the

two areas of robotics and cloud computing construct a concept called cloud robotics [8].

Cloud robotics is a robot connected to the cloud, utilizing the shared pool of resources provided by the cloud. Cloud robotics introduce centralized functionalities, allocation of resources facilitating storage and processing power for multiple robots. However one vital feature missing in cloud robotics is the robot's ability to communicate with other robots and devices over the Internet utilizing a common communication technology [8]. One solution enabling a global communication platform is IoT.

IoT is a global network enabling a cluster of devices known as *things* to communicate with each other and transfer data over the Internet. This enables a communicative platform where all participants communicate utilizing a common communication technology.

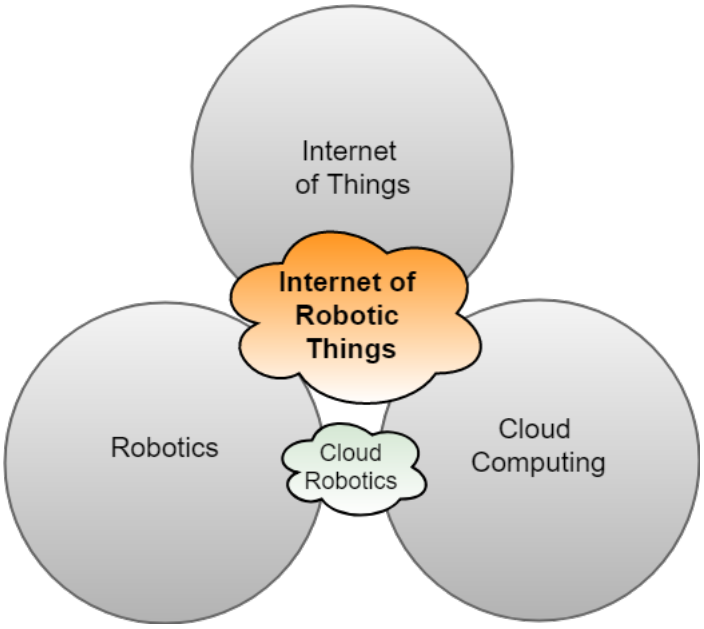

Figure 1.1. Illustration of the IoRT technologies.

Consequently, by merging these communities we aim to elaborate the novel concept, Internet of Robotic Things (IoRT) as illustrated in figure 1.1 [8]. IoRT is a combination of IoT, robotics and cloud computing. IoRT functions as a global infrastructure facilitating advanced robots to interconnect utilizing existing and evolving communication technologies to share interoperable information. Additionally, the IoRT infrastructure function as a shared pool of centralized cloud computing services provided by modern data centers enabling scalable data processing, storage and communication resources for the robots [9].

1.2 Problem Statement

The purpose of this thesis is to create an IoRT platform based on the three technologies: robotics, IoT and cloud computing. To develop a platform that supports the characteristics of IoRT, the following abilities demanded by the system are needed:

1. Distribution of interoperable information between multiple robots utilizing existing and evolving communication technologies.
2. Centralized cloud computing services provided by modern data centers, enabling data processing, storage and communication resources.

The first ability requires a distributed network illustrated in figure 1.2B, a communication technology supporting multiple end devices to communicate with each other. The second ability requires centralized cloud computing functionalities illustrated in figure 1.2A. Combining these enables a simultaneously centralized & distributed network illustrated in figure 1.2C.

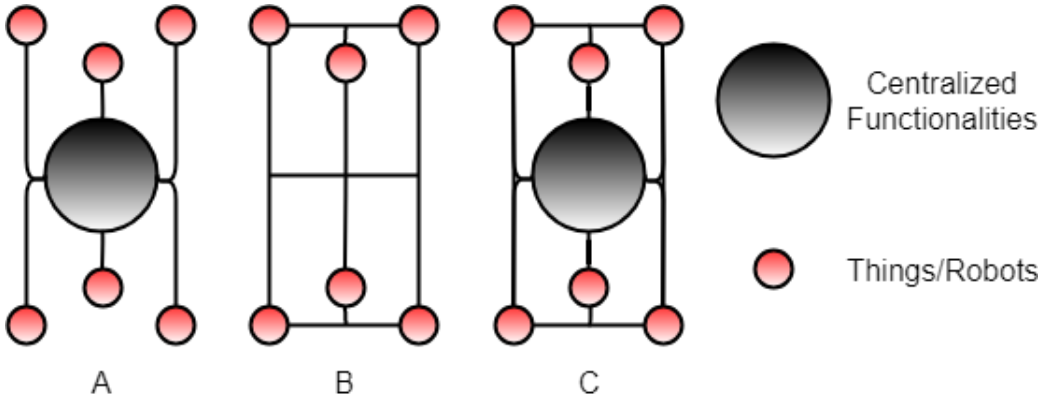

Figure 1.2. Illustration of different networks setups. A: Centralized Network, B: Distributed Network, C: Both Centralized & Distributed Network

A distributed network coordinates multiple robots to share information with each other over a common communication technology enabling all the robots connected to establish connections with each other. A centralized network enables the robots to communicate with centralized functionalities providing the robots to share information, utilize stored information and offload the local hardware.

In this report, we aim to develop an IoRT platform by combining existing technologies, i.e. a combination of services consisting of a robotic platform, an IoT platform and cloud services.

1.2.1 Research Question

The IoRT platform is based on three technologies: IoT, robotics and cloud computing. The practical approach to develop the IoRT platform is a combination of an IoT platform, a robotic platform and cloud services aiming to answer the following research question:

- How is a distributed network designed and developed combining an IoT platform with a robotic platform?
  - Is it feasible to use an IoT platform together with a robotic platform in terms of processing and network delays?
- How are centralized functionalities designed and developed combining cloud computing micro-services with a robotic platform?
  - Is it feasible to store and process sensor data in the cloud utilizing micro-services in comparison with traditional on premises hardware?
  - Is it feasible to visualize streaming data processed with micro-services in near real-time?

1.2.2 Solution Overview

The purpose is to design a platform enabling a distributed network of robots to asynchronously share information, in addition to centralized cloud services enabling visualization, storage and processing power in near real-time.

The proposed approach has the following key components:

- Robot Operating System (ROS) [10]: Open source robotic platform with large software libraries and tools to develop robotic applications.
- Amazon Web Service (AWS) IoT core: An IoT platform enabling IoT devices to connect to other IoT devices using Internet transport protocols.
- Serverless: A shared pool of hardware dynamically managed by a third party cloud vendor.
- Function as a Service (FaaS): Processing, storing, monitoring and visualizing functionalities utilized in the cloud as serverless services.
- Micro-services: Multiple FaaS working together to perform a task.
- Amazon DynamoDB: A FaaS database providing storage.
- AWS Lambda: FaaS computation environment providing processing power.
- Service: A software functionality such as a robotic platform, an IoT platform, Amazon DynamoDB and AWS Lambda.

The choice of services is based on logistic reasons, all the cloud services including AWS IoT

1.3 Delimitations

Delimitations set to narrow down the scope of the thesis:

- There are many different cloud vendors but Amazon Web Service (AWS) is used as the infrastructure for the IoT platform and the micro-services in use for data processing and storage. Other cloud vendors can be used for the same purpose, but they are not covered.
  - AWS IoT core is used as the IoT communication platform as it is the only IoT service provided by AWS, similar services are provided by Google Cloud and Microsoft Azure.
  - Amazon DynamoDB is used for storage as it is the only serverless database on Amazon that supports interconnection to AWS IoT core.
  - AWS Lambda is used for serverless processing power as it is the only serverless computing service that supports interconnection to AWS IoT core.
- ROS is used as the robotic platform as it is utilized on the test equipment.
- The costs of the services used with AWS is not taken into account.
- Security credentials is used during this project but will not be evaluated in this thesis.

1.4 Ethics

In this thesis, the security aspect is left outside the scope. However, security credentials utilized by devices for authentication over the Internet only permit a selected number of services for security reasons. When open source software is used in this project, it is clearly referred to.

1.5 Thesis Outline

Chapter 2 illustrates the research methodology. Chapter 3 includes the frame of reference, the essential theory to understand the rest of the report. Chapter 4 explains the design of the IoRT platform, the specific choices of services. Chapter 5 includes the experimental study, in this chapter the test platform is explained. Chapter 6 includes the results.

Chapter 7 contains the discussion and ethics. Chapter 8 states the conclusion drawn from this thesis and personal reflections for future work on the IoRT platform.

Chapter 2

Research Methodology

Exploratory literature research [11] is applied to explore the existing papers of IoRT and the corresponding techniques. Systems thinking is then applied supporting the development process and finally an experimental setup to verify the functionalities of the IoRT platform.

2.1 Exploratory Literature Research

The literature study includes a qualitative approach using exploratory research [11]. IEEE Xplore is the main source for research papers and articles, moreover online research papers have been studied for the literature research. The process is as follows:

1. Overview of the research area to identify and explore new areas of interest.
2. Map out the key components, researches and related terminology.
3. Investigate and address the gaps in previous platforms.

2.2 Systems Thinking

Systems thinking is a combination of analytic skills about systems used together to create a system of systems with a value greater than the sum of its parts. Moreover, to understand the systems and predict their behaviors in order to achieve the purpose [12]. A systems thinking definition must contain elements, interconnections and a purpose as described in the steps below [12][13].

- **Purpose:** Describing the purpose of the systems.
- **Elements:** The characteristics composed by the systems.

- **Interconnections:** The relation between the elements.

System thinking is used as a framework to design the development process of the platform that satisfies the research questions.

2.2.1 Development Process

The development process is designed utilizing both exploratory research and systems thinking approach and contains 6 steps.

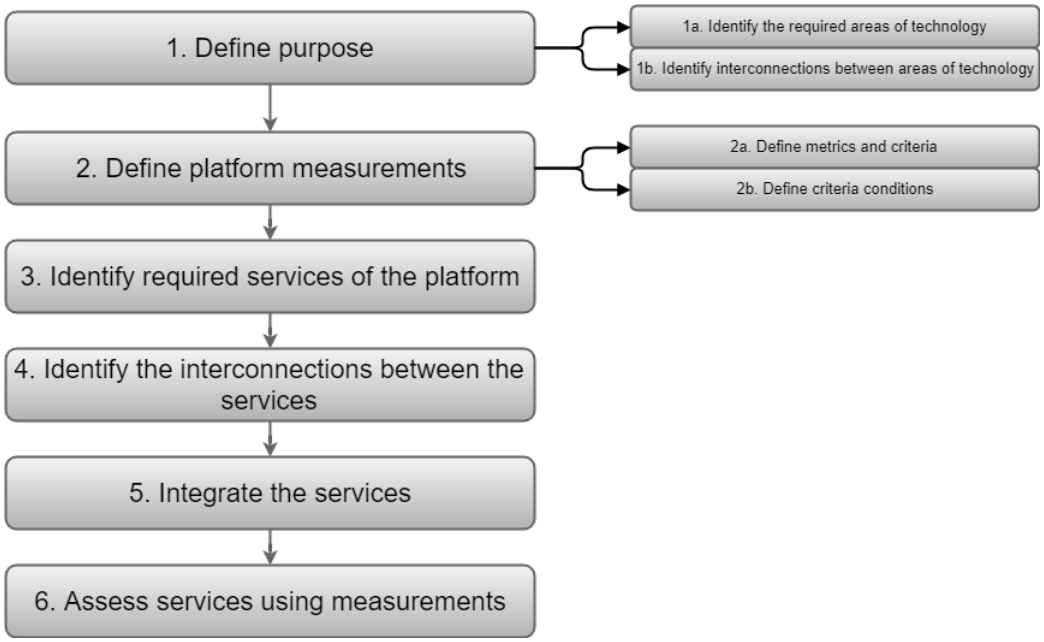

Figure 2.1. Development Process.

1. Describing purposes, what are the characteristics and what are the capabilities?
   (a) Identify the required areas of technology to fulfill the purposes.
   (b) Identify interconnections between the areas of technologies.
2. Define platform measurements based on the characteristics composed by the system.
   (a) Define the metrics (e.g. Elasticity) related to the measurement and the criteria (e.g. Can the service automatically adapt the workload on demand?) which sets the question about the metric.
   (b) Define the criteria conditions (e.g. Yes, No, Limited) that answer the questions about the metrics.

3. Identify required services for the platform based on the characteristics of the system to satisfy the purpose.
4. Identify the interconnections between the services. How do they interact with each other?
5. Integrate the services together utilizing the interconnections identified.
6. Assess the services using defined measurements.

2.3 Experimental Setup

A data analysis process is examined and used as a template to design a platform, the steps of the process is presented in section 4.6. The platform consists of a robot, an IoT platform, computational and storage cloud services, moreover visualization and monitoring services. The experimental setup covers the platforms ability to visualize, process and store data from the robots in near real-time, furthermore the delays between the services used in the platform is measured.

The platform is evaluated with the measurements: functionality, composability, interoperability, scalability and computing time. Every measurement have corresponding metrics and criteria, the criteria of each metric is presented below in table 2.1. The criteria is answered in the results with a criteria condition. The platform is measured with the following measurements and metrics:

- Functionality: The services ability to interact in its surrounding, monitor service metrics, store and process data.
  - Interaction: Can the service interact with its surroundings?
  - Computation: Can the service provide an execution environment?
  - Storage: Can the service store data?
  - Monitoring: Can the service monitor own service metrics and/or other service metrics?
- Composability: The services ability to be modular and stateless.
  - Modularity: Can the service be deployed independently?
  - Stateless: Can the service treat invocations independently?
- Interoperability: The services ability to exchange data with other services in the platform.
  - Pattern: What messaging pattern does the service provide?

  - API/Protocol: What communication protocol or API (Application Programming Interface) do the service support?
  - Payload: What payload is used by the service?
- Scalability: The services ability to scale on demand.
  - Concurrency: Can the service scale in number of instances?
  - Elasticity: Can the service automatically adapt workloads on demand?
- Computing Time: Network latency and cloud processing time.
  - End-to-end delays: How much time it takes for a message to reach its destination?
  - Utilization time: How much time it takes for a process to be utilized?

The table below includes the measurements with their corresponding metrics and criteria.

Table 2.1. Measurement, Metrics & Criteria

Measurement	**Metrics**	**Criteria**
Functionality	Interaction	Yes or No
	Compute	Yes, No or partially possibly with simple operations
	Storage	Not possible, Databases or Files
	Monitoring	No, Monitor own metrics or monitor multiple service metrics
Composability	Modularity	Yes or No
	Stateless	Yes or No
Interoperability	Pattern (section 3.3.1)	Publish/Subscribe, Push, Pull, Request/Respond
	API/Protocol (section 4.5)	MQTT, TCPROS/UDPROS, IoT Rules, DynamoDB Streams, AWS SDK, AWS MQTT SDK
	Payload	JSON [14], ROS Message
Scalability	Concurrency	Unlimited, Limited or No
	Elasticity	Yes or No
Computation Time	End-to-end delays	Time
	Utilization time	Time

Chapter 3

Frame of Reference

This chapter presents the theoretical framework divided into six sections. The first two sections; cloud computing and cloud robotics. The third and fourth section; communication technologies and Internet of things. The forth section is about the architecture IoRT and the last section includes the literature review.

3.1 Cloud Computing

Cloud computing is an emerging field providing a shared pool of scalable hardware provisioned as cloud services by modern data centers. The offered services can roughly be divided into four different types shown in the figure 3.1 below [1]:

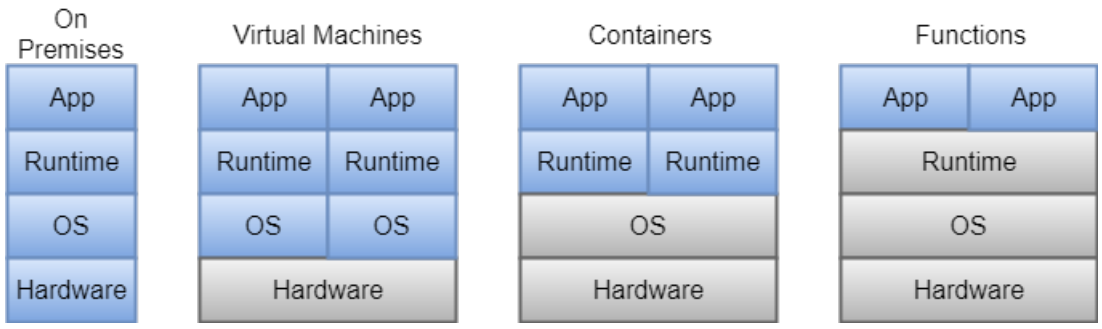

Figure 3.1. Different cloud types where the gray layers are shared [1].

The whole stack consists of the hardware, operating system (OS), runtime and application. The hardware corresponds to the physical servers and the network infrastructure. The OS includes a kernel plus additional software, the kernel is the intermediary between the software and the hardware. The runtime environment refers to an environment where

the software is executed, when executed the software can access the kernel and send instructions to the hardware. App, short for application is the software.

On Premises

On premises includes the whole stack, the hardware, OS, runtime environment and the applications. A manual process that is hard to scale and expensive to initiate.

Virtual Machines

Virtual machines (VM) excludes the hardware, where cloud vendors maintain a shared pool of hardware. Here the developer maintains the OS, the runtime and the applications. This enables the developer to easily scale to multiple VM:s, however this approach is often a waste of resources as each VM runs a full copy of an OS.

Containers

Containerized infrastructure evolved as a result of the heavy overhead incorporated in VMs. Containers are striped version of VM:s, making it possible for multiple isolated applications to access the same kernel. The developer packs the software with all dependencies and libraries in a container and deploy it to the cloud. This is achieved with a container engine such as Docker [15] or Kubernetes [16]. Containers facilitates the distribution of instances and introduces concurrency and modularity. However, the developer still needs to handle the runtime environment, dedicating memory and CPU to the application.

Functions

Functions as a service or FaaS is an emerging paradigm where all the resources except the application are maintained by the cloud vendor, even the runtime environment. Developers do not need to spend any resources until execution. Moreover, the cloud vendor provides an execution environment enabling elasticity, concurrency and modularity facilitating the needs to develop, manage and operate the applications [3].

3.2 Cloud Robotics

Cloud robotics refers to a robot utilizing cloud computing where the robots takes advantage of the centralized functionalities provided by the cloud. Cloud robotics have emerged from the limitations of networked robotics, limited by physical constraints

where all systems are based locally on the robots. A networked robot is proposed according to IEEE Robotics & Automation Society [17] referring to a device connected to a communication infrastructure such as the Internet and is divided into two types [17]:

1. Tele-operated, where there is a human supervisor sending and receiving information through the network. The mars rover is a good example of a tele-operated system, where humans tells the robot what to do, and the robot feeds back information.
2. Autonomous machine to machine infrastructure, where the robots and sensors exchange data through the network.

Cloud robotics offers a new form of efficient robotic systems [18] that rely on cloud computing for processing power, storage and scalability. Although cloud robotics profits from all available cloud services, it brings a lot of challenges described below [19] [20]:

1. Network latency: Time to send and receive data over the network.
2. Cloud processing: Time to process and analyze the data in the cloud.
3. Standardization: No homogeneous standard for a common communication technology.

To make a cloud solution feasible, the network latency and cloud processing time should be less than a solution without a cloud. However there are advantages [21] [22], such as:

- The cloud infrastructure introduces concurrency, elasticity and modularity.
- Most of the computational work can be utilized in the cloud in order to save both power and weight on the remote robot.
- Scalable pool of hardware that adapts dynamically.

Cloud robotics makes it possible for multiple robots to utilize centralized functionalities. However, one vital feature in a fast and available system is to have a distributed network, a platform that utilizes a common communication technology enabling all the robots to establish a communication.

3.3 Communication Technology

Communication technology refers to the use of technology to exchange information between communicating participants. This section presents the messaging patterns and the communication protocols.

3.3.1 Messaging Pattern

Messaging patterns defines how two opposing services establish a connection and communicate with each other. The messaging patterns are split up into synchronous and asynchronous events as illustrated in figure 3.2

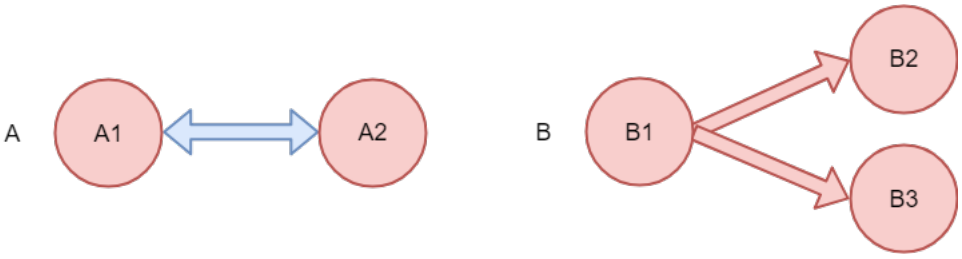

Figure 3.2. Examples of pipelines using A: Synchronous and B: Asynchronous messaging patterns.

- Synchronous: In this case A1 either request information from A2 where A2 will respond back or A2 pull information from A1 and respond back to A1. In either case the process requires a response, therefore A1 is blocked until the instance is complete.
- Asynchronous: In this case B1 can publish multiple messages to the subscribers B2 and B3, here B1 do not care how B2 and B3 handles the incoming messages, moreover B1 do not require an answer for the messages. B1 can publish multiple messages to the same receiver without waiting for the previous message to complete.

Publish/Subscribe

A publish-subscribe messaging pattern utilizes asynchronous communication and is often used in robotic applications. The main advantage is that nodes publish or subscribe to a topic/channel with no direct communication to other nodes. A node is a participant in the network that communicate through defined topics/channels with the ability to publish and subscribe to multiple topics/channels simultaneously. The publisher sets the frequency and the subscriber triggers its callback function every time a message is received [23].

Request/Respond

Request/response messaging pattern is widely used in both robotics and the world wide web. A basic pattern where one service requests some data whereas the second service respond to that request. This pattern is mostly implemented as synchronous events but can also be used asynchronously [24].

Push & Pull

Push and pull are two different messaging patterns often used by cloud services [25]. Push is an asynchronous event and pull is a synchronous event. In figure 3.3 A1 and A2 illustrates two services where A1 is the publisher and A2 is the subscriber. In the first scenario A2 pushes a message to A4 each time A2 receives a message on a specified topic/channel, the push is therefore invoked by the service A2. The asynchronous push enables A2 to push multiple messages concurrently to A4 or other services independently.

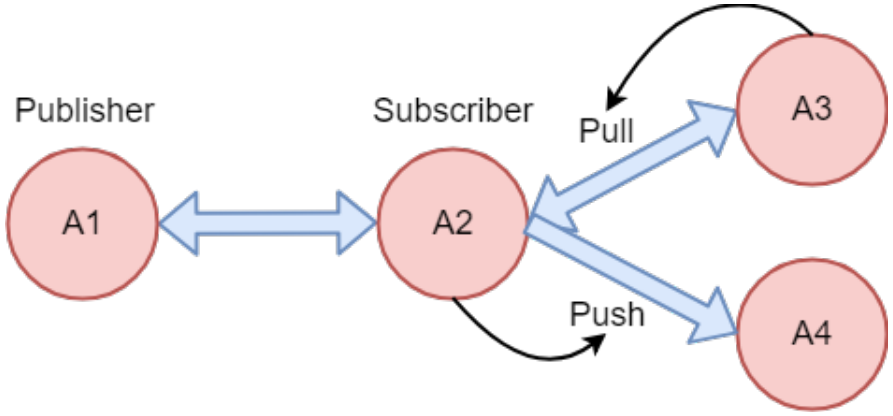

Figure 3.3. An illustrative figure of push and pull messaging patterns. The blue arrows illustrate the data flow and the black arrows illustrates the messaging pattern.

The second scenario, A3 is triggered each time A2 receives a message. A2 in this case can be a database logging data as it is published by A1. For every new event logged in the database, A3 pulls the message, processes it and send it back to the database.

3.3.2 Communication Protocol

A communication protocol is a set of rules defining how two opposing services transmit information. The most common communication protocols supported by IoT platforms is pure MQTT (Message Queuing Telemetry Transport), MQTT over WebSockets and HTTP (Hypertext Transfer Protocol) illustrated in figure 3.4.

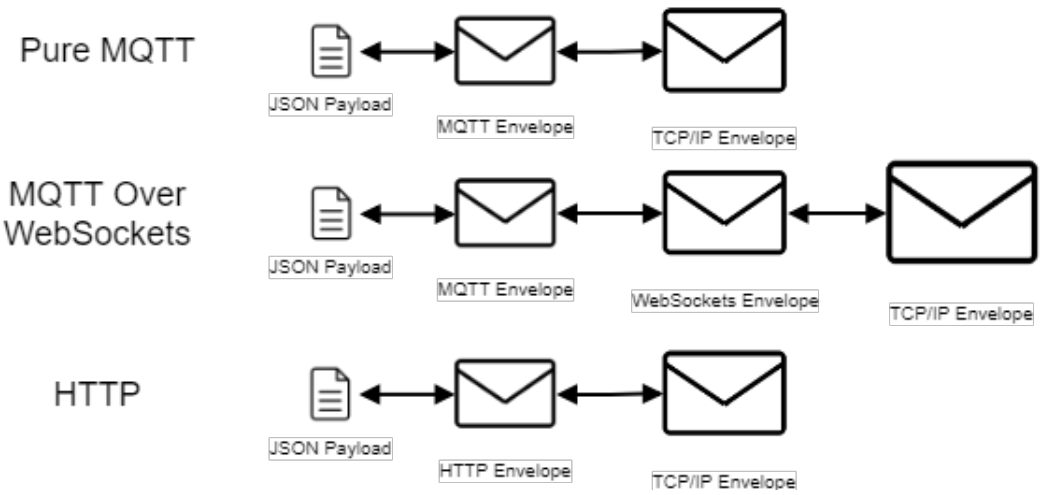

Figure 3.4. An illustrative figure of the communication protocols.

The payload is in JSON [14] (JavaScript Object Notation) text format which is light weight, easy to read and language independent. The envelopes is an illustration of the message packaging.

MQTT

MQTT is a communication protocol utilizing a publish-subscribe messaging pattern that works on top of the TCP/IP protocol. MQTT is a lightweight machine to machine/IoT protocol with advantages such as short response time, high throughput, low battery consumption and Quality of Service (QoS). MQTT offers three different levels of QoS [26].

- Level 0: Only the TCP protocol handshake, no extra QoS.
- Level 1: Arrive at least once, but the message can arrive more than once.
- Level 2: Arrive only once, guarantees that the message will only reach its destination once.

WebSockets

The WebSocket protocol [27] enables a full-duplex communication between a client and a host over TCP connection. MQTT over WebSockets enables the MQTT protocol to communicate with web clients making any browser-based technology compatible with other MQTT nodes.

HTTP

HTTP is a document-centric request/response method, often compared with MQTT as the slower alternative. However, it is the most popular and widely used protocol mostly for world wide web applications, HTTP offers no QoS [28].

3.4 Internet of Things

IoT is described according to The International Telecommunication Union as a global infrastructure for the information society. IoT enables interoperability for multiple devices known as things to communicate through the Internet, where services interconnect (physical and virtual) things, based on existing information and communication technologies [4]. The largest IoT platforms is AWS IoT Core [29], Microsoft Azure IoT Hub [30] and Google Cloud IoT Core [31].

3.5 Internet of Robotic Things

Internet of Robotic Things (IoRT) is a new emerging concept, a combination of IoT and cloud robotics, where cloud robotics is a combination of robotics and cloud computing. IoRT utilizes existing communication technologies and cloud computing services provided by cloud vendors, to share, process and store interoperable information. The architecture is divided into five layers illustrated in figure 3.5 and described in short below [32]:

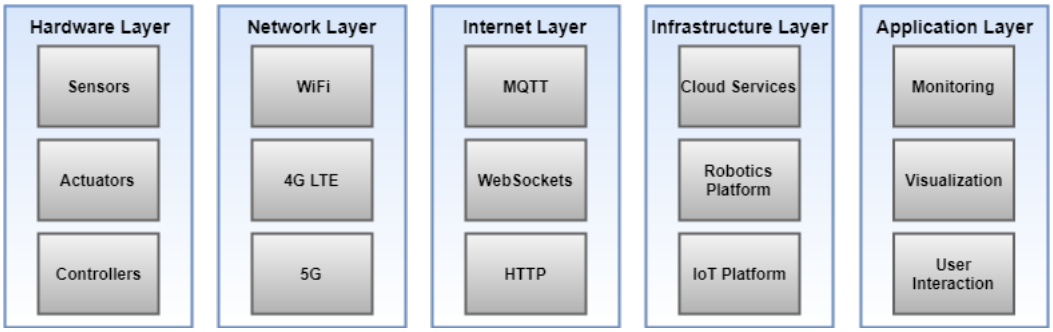

Figure 3.5. An illustrative figure of all the layers presented [32].

Hardware Layer The first layer includes the peripherals of the architecture, from low level to high level sensors and actuators. Here all the information is gathered to the next layer.

Network Layer The second layer includes the networking protocols for communication, Near Field Communication (NFC), Bluetooth, WiFi, 3G, 4G/LTE are all included in the network layer.

Internet Layer The third layer includes the transport protocols needed to send and receive messages, such as MQTT 4.3, TCPROS 4.2 , Data Distributed Service (DDS) [33], WebSockets and HTTP. Most of these protocols are light weight and energy efficient suited for robotic applications.

Infrastructure Layer The fourth and most vital layer in the architecture, including the robotic platform, IoT platform and the cloud services.

Application Layer The last layer is used to propagate the user experience, including user interaction, monitoring and visualization.

3.6 Literature Review

The IoRT market is expected to be valued at USD 21.44 Billion by 2022, with a compound annual growth rate (CAGR) of 29,7% between 2016 and 2017 [34]. The rapid evolution in the fields of cloud computing, IoT, robotics, machine learning, big data and networking have opened up the possibilities for developers to combine and integrate the different fields in various ways [21],[35],[36]. The combining possibilities are close to endless and new emerging software are presented constantly.

Rapyuta [37] is an open source cloud robotics framework, that enables the robots to move heavy computation into the cloud. A container based platform that provides connectivity to RoboEarth [38]. Roboearth main features are store and share information, providing scalable storage to be shared leveraging the experience of other robots. The communication between robots and Rapyuta is implemented based on full-duplex Websockets [27] [39].

DAvinCi (Distributed Agents with Collective intelligence) is another serverless open source cloud robotics framework, that enables a system of heterogeneous robots to communicate with each other through the cloud [40]. A robot in the DavinCi environment either communicate locally through publish/subscribe messaging patterns or with the cloud utilizing HTTP request/response messaging pattern, making the framework slow in terms of distributed communication between the robots over the Internet. However, the main purpose is to introduce centralized functionalities for a network of robots.

Another open source framework used in robotic applications is Spacebrew [41]. Spacebrew facilitate the connection between remote objects utilizing a publish/subscribe messaging pattern, using WebSockets as its communication protocol [42] [43]. However, Spacebrew do not provide any centralized functionalities.

IoRT is still a novel concept and the previous research is limited. Very few works explore robotic frameworks with both centralized cloud functionalities together with practice of the art IoT platforms for distributed communication. Moreover, no research is found combining the IoT functionalities of the giant cloud vendor Amazon together with the most frequently applied open source robotic framework ROS.

Chapter 4

IoRT Platform Design

This chapter explains the design of the IoRT platform consisting of a robotic platform, an IoT platform, cloud services and their corresponding interconnections. The last section introduces the data analysis process.

4.1 Overview

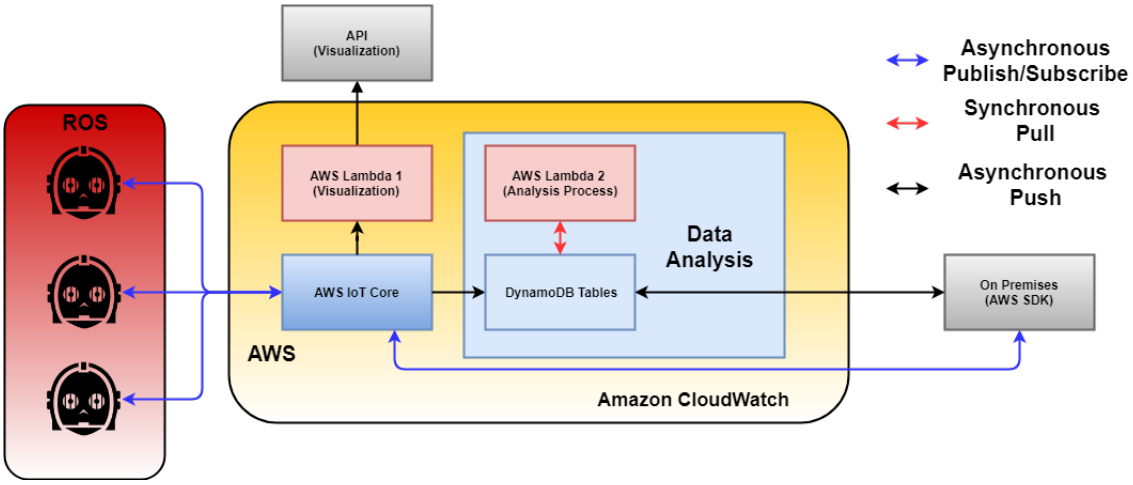

Figure 4.1. Service Oriented Test Platform.

Figure 4.1 illustrates a setup of the IoRT platform composed of a robotic platform, cloud services and an IoT platform. ROS is utilized as the robotic platform and is displayed in the figure as multiple robots connected together. The centralized functionalities consist of cloud services provided by AWS where DynamoDB is used for data storage and AWS Lambda used for data processing. The distributed network consists of the IoT platform

AWS IoT Core and is the link between all the robots over the Internet and additionally functioning as the bridge connecting the robot to the centralized functionalities.

4.2 Robot Operating System

ROS is an open source robotic platform widely applied for robotics development [10] as ROS utilizes a publish-subscribe messaging pattern. In every ROS network there is a master node managing the initiation of the publish/subscribe communication, this enables the nodes to advertise the topics/channels they are set to publish and subscribe to. A node is a participant in the ROS network that share its information with other nodes using defined topics/channels. The ROS master establishes a peer-to-peer connection between the publishing and subscribing nodes upon initiation, and continuously listen for new advertisements. The connection between the nodes is setup with an agreed transport protocols such as TCPROS [44] or UDPROS [45]. Figure 4.2 illustrates the connection establishment.

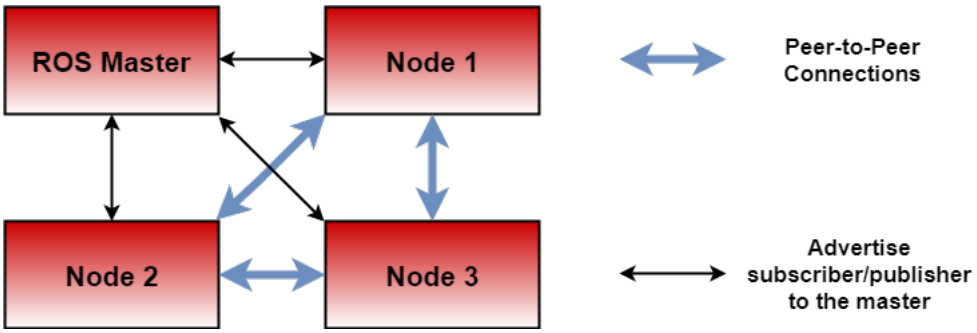

Figure 4.2. Illustrative figure of the connection establishment.

4.3 AWS IoT Core

AWS IoT Core [46] is a serverless AWS IoT platform enabling bidirectional communications for devices to securely interact with other devices. According to AWS [46], the platform has the ability to scale on demand supporting billions of devices, moreover trillions of messages.

AWS IoT core supports publish/subscribe messaging pattern between all the things in the network where MQTT is used as the communication protocol, moreover MQTT over WebSockets is supported enabling any service using WebSockets to connect to the network of things. IoT core also supports HTTP where clients post data to any topic/channel, the post uses request/response messaging pattern where the request message contains the data [25][47].

4.4 Cloud Services

This section presents the cloud services presented in the overview.

4.4.1 AWS Lambda

AWS Lambda [48] is a serverless computing service enabling lambda functions to run in the cloud as FaaS. AWS lambda has its ability to run multiple invocations of the same application at once. The two core components are the lambda functions and the event source [49]. The function is a custom script that is invoked creating an event, the event source is the service that triggers the function.

An event source can be either an AWS service or custom applications. Connecting AWS lambda to other services is done through event source mapping. Event source mapping enables mapping between the event sources and the lambda function and can be either stream-based (Synchronous Pull Invocation) or non stream-based (Asynchronous Push Invocation).

Lambda functions enable to adjust the compute resource, the amount of memory dedicated to the function and the maximum execution time. The maximum execution time sets the limit of the allowed runtime, if the function exceeds this time it is terminated. The memory allocated is proportional to the CPU power of the function, the memory available is between 128 MB and 3008MB.

4.4.2 Amazon DynamoDB

Amazon DynamoDB [50] is a serverless NoSQL (non-relational Standard Query Language) database service. DynamoDB stores data in items, each item consists of a partition key, a sort key and attributes. The partition key and sort key makes the primary key, the primary key values must be unique for every item stored in the database as the identity of each item is based on the primary key. An example illustrated in figure 4.3 where the table is used as a time-series with X,Y,Z as arbitrary data. One main benefit using a NoSQL database is that the attributes are dynamic, facilitating storage from different sources using their own attribute IDs.

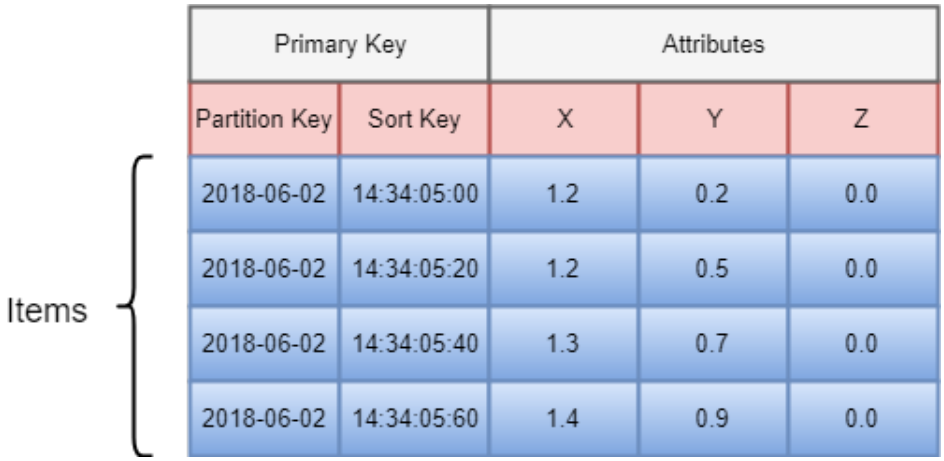

Figure 4.3. Illustration of a DynamoDB table.

DynamoDB supports query and scan operations. Query operations must specify a partition key value, the sort key is optional. However the query can only filter either the partition key or the sort key. Scan operations can scan all the data without a partition key value, furthermore filter out specific parts of the table e.g. X>1.2 [51].

4.4.3 Amazon CloudWatch

Amazon CloudWatch [52] enables the developer to monitor the AWS resources. All resources used by AWS gathered in one place facilitates the development and debugging of the system. Moreover, monitoring and operational data can be collected and visualized.

4.5 Interconnection Design

The interconnections transfer the information between the services on the platform. The connectors utilize publish/subscribe messaging pattern, synchronous pull and asynchronous push messaging patterns.

4.5.1 ROS & AWS IoT Core

Both ROS and AWS IoT utilizes a publish/subscribe messaging pattern making both services inter-operable, the topics/channels must be defined both in ROS and in AWS IoT. Each publisher defines the topic/channel to publish to and the subscribers subscribe to a topic/channel of choice. Figure 4.4 illustrates the workflow of a ROS message and is explained in detail under the figure.

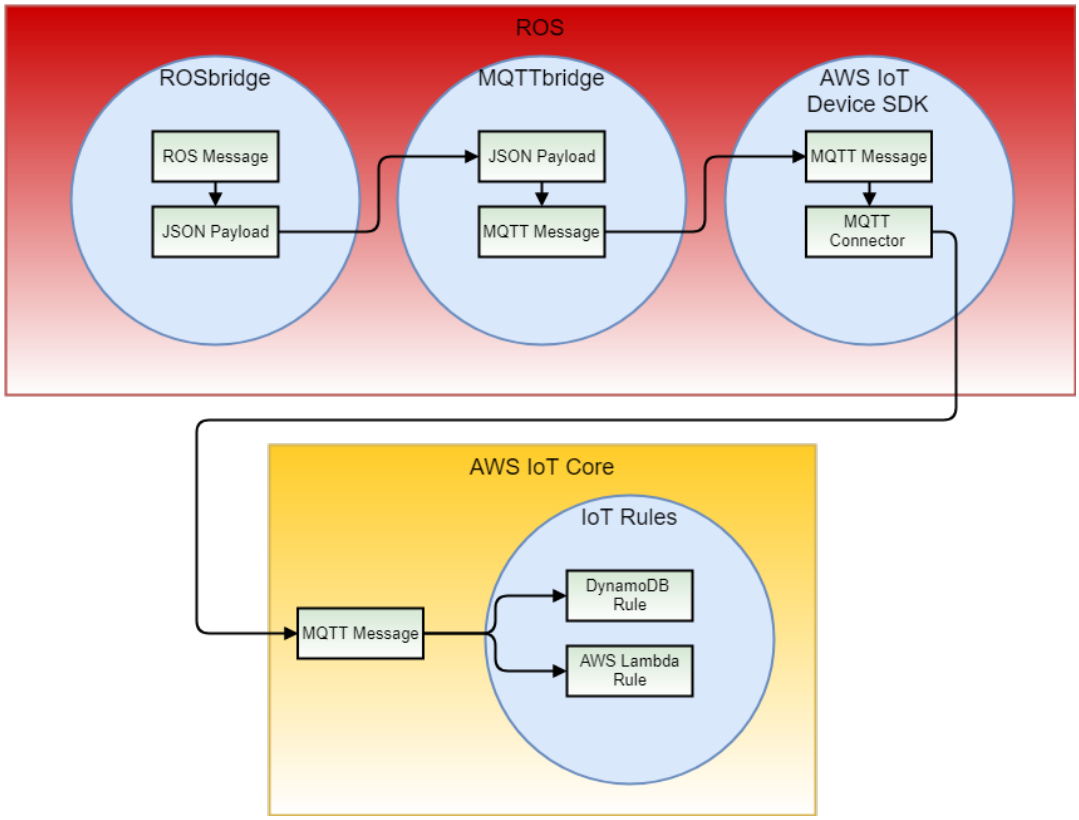

Figure 4.4. Workflow of a ROS message between the local hardware (red) and the cloud (yellow).

ROSbridge [53] is used to convert ROS messages to JSON format. Converting the payloads makes the ROS and MQTT messages compatible, MQTTbridge [54] built on top of ROSbridge makes it possible for a bidirectional message flow between ROS and MQTT topics/channels as it places the JSON payload inside an MQTT message. AWS IoT Device SDK for python [55] provided by AWS is a secure connector between any python application and AWS IoT Core topic/channel through the MQTT protocol, to establish a publish/subscribe messaging transportation. MQTTbridge and AWS IoT Device SDK for Python is combined by bydottck13 [56] and enables ROS to work bidirectional with IoT Core. Merging the two publish/subscribe services together creating a heterogeneous distributed network. A modified version of bydottck13 repository made by the author is used [57].

4.5.2 AWS IoT Rules

Rules are APIs for the IoT core and provides interoperability for the IoT devices and other AWS services [58]. More accurately the rules enables IoT core topics/channels to interact with other AWS services as they are triggered as illustrated in figure 4.4.

Messages on any topic/channel can be asynchronously pushed to a DynamoDB table or asynchronously invoke a lambda function with the Lambda Rule. It is also possible to republish messages to an AWS IoT topic/channel using IoT rule republish [59] enabling simple computational operations.

4.5.3 DynamoDB & AWS Lambda

DynamoDB supports synchronous invocations through DynamoDB streams [60]. These streams enable AWS lambda to pull the stream upon new records in the DynamoDB table, invoking a lambda function. The invocations are synchronous using a request/response pattern enabling near real-time processing of incoming data, furthermore the synchronous invocations makes it possible to write new data in a DynamoDB table in one process.

Push & Pull

As explained in section 4.4.1 an event source can be either stream-based or non stream-based. Stream-based models pulls the stream when it detects a new record, invoking a lambda function. Non stream-based models invokes a lambda function every time it pushes a message. Scaling behavior differs between the two event sources in how they handle elasticity [61]. Stream-based event sources process each shard event in sequence, this makes the number of lambda functions running concurrently dependent on the number of shards [61]. The number of shards depends on the partitioning of the stream. If a stream is split up into 100 shards, the maximum number of lambda functions running concurrently is 100. Non stream-based event sources invokes a lambda function for each event and support concurrency up to the account limit. The DynamoDB read/write elasticity limit is set by the developer as operations per second and can be set to dynamically change according to the load.

4.5.4 AWS Software Development Kit

AWS Software Development kit (SDK) [62] for python called Boto3 is used to facilitate the development of the platform. It enables low-level direct service access and object-oriented API:s. A session is established between the AWS services and any python application.

4.6 Data Analysis

Data analysis is a process that includes different techniques and approaches, and the representation of these techniques can differ very much depending on the approach and application. Data mining is a well known analysis technique, one process known as the

Knowledge Discovery in Databases (KDD) described in the book Data Mining: Concepts and Techniques [63] explains that data mining can be recognized both as a synonym to KDD and as an essential part of the KDD process. The KDD process described in the book:

1. Data Cleaning: Remove noise and inconsistent data
2. Data Integration: Combining multiple data sources
3. Data Selection: Selecting data relevant to the analysis
4. Data Transformation: Summery or aggregations operations to transform the data appropriate for data mining.
5. Data Mining: Use of intelligent methods to extract data patterns.
6. Pattern Evaluation: Identifying interesting patterns representing knowledge of the system.
7. Knowledge Presentation: Visualization and knowledge representation techniques to present mined knowledge.

The process is illustrated in figure 4.5, the steps will not be evaluated in depth as it is outside the scope. However, the process is used as a template for the case study to test the potential of the IoRT platform.

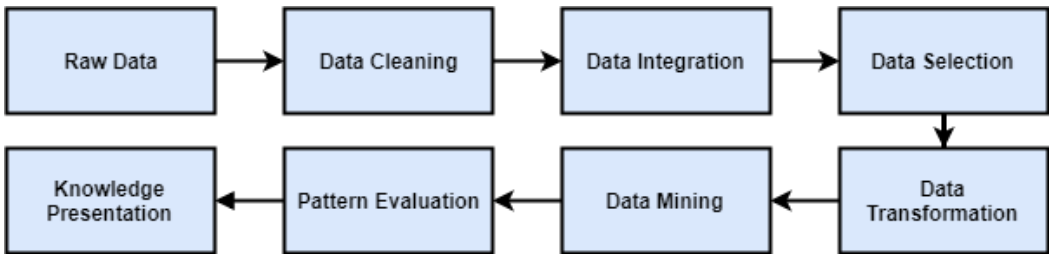

Figure 4.5. Data analysis process.

Chapter 5

Experimental Setup

In this chapter, a data analysis process is used as a template to construct an IoRT platform with practical settings utilizing the 5 layers presented in section 3.5. The experimental setup includes a small RC (Remote Controller) car running ROS, generating sensor data as the IoRT platform handles storage, processing and visualization in near real-time.

5.1 Hardware Layer

The hardware layer consists of an RC car. The RC car runs a NVIDIA Jetson TX1 Developer Kit with Linux distribution Ubuntu 16.04 on top of it with an electric speed control (ESC) unit generating sensor data in real-time.

5.2 Network & Internet Layer

The network layer consists of a wireless network interface controller to establish a communication between the robot and the Internet. The networking standard used is the IEEE 802.11ac. Publish-subscribe messaging pattern is used for the robot to communicate locally on the ROS network and remotely with the IoT platform. MQTT is utilized as the communication protocol between the RC car and the cloud as illustrated in figure 4.4.

5.3 Infrastructure Layer

The infrastructure layer consists of a robotic platform, an IoT platform and cloud services. The robotic platform ROS is used on the RC car and handles the communication between

the nodes locally on the car. The IoT platform AWS IoT Core handles the communication between the RC car and the cloud. AWS Lambda and DynamoDB represents the cloud services handling the centralized functionalities; storage and data processing.

5.4 Application Layer

The application layer consists of the monitoring services and the visualization applications.

PubNub

PubNub [64] provide near real-time publishing/subscribing pattern APIs, to enable other third-party applications to connect to PubNub [65]. Microsoft Power BI support PubNub APIs for near real-time communication making PubNub the bridge between the AWS services and the visualization tool Microsoft Power BI. PubNub utilize its own agnostic approach in choice of protocol, moreover PubNub also supports MQTT [66].

Microsoft Power BI

Microsoft Power BI [67] provides various services to analyze and visualize business intelligence. Microsoft Power BI visualize the near real-time data-stream of any topic/channel available on PubNub with a set choice of charts [68]. Microsoft power BI supports visualization on any web browser, moreover on applications for Android and IOS devices.

5.5 The Test Platform

The test platform is used to utilize the micro-services, moreover measure the utilization time and the network delays.

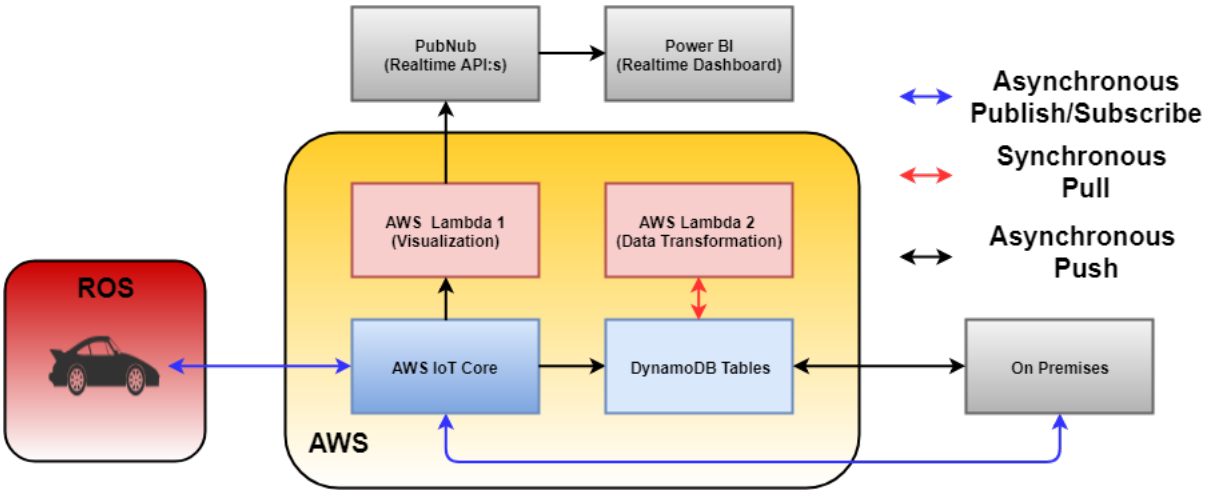

Figure 5.1. Service Oriented Test Platform.

A ROS node subscribes to the raw sensor data from the VESC node, transforms and publishes the data to an IoT Core topic/channel using the connector in section 4.5.1. Here the data goes two ways using the IoT rules in section 4.5.2; Lambda 1 and DynamoDB tables.

Lambda 1 receives the push from IoT Core, selects a couple of data points, initiates a connection to PubNub and pushes it to a PubNub channel. Microsoft Power BI streams to the channel and visualizes the data on a dashboard in near real-time.

The storage of data is asynchronously pushed to DynamoDB table where it is stored as raw data in JSON format. The data uploaded to DynamoDB can instantly be downloaded by any device or service for further implementations.

Lambda 2 is synchronously invoked every time the DynamoDB table receives a new record through the DynamoDB streams connector in section 4.5.3. Lambda 2 polls the DynamoDB table upon new entry and then returns the same value to a new table illustrating a simple data transformation.

On premises is a computer running Windows 10 with an Intel I7 processor of the 7th generation and a wireless network interface controller with the networking standard IEEE 802.11ac. The computer utilizes the AWS SDK boto3 to query and scan the data stored in DynamoDB, moreover connect to IoT core.

Chapter 6

Results

In this chapter, the results of the experimental study are presented. First section includes the measurements of functionality, composability, interoperability and scalability. The second section presents the delays of the platform and answers the first research question (RQ1). The third section presents the utilization time, visualization capabilities and answers the second research question (RQ2).

6.1 Platform Abilities

The results presented below are acquired through both exploratory literature research and experiments.

Functionality

Functionality is the services ability to interact in its surrounding, monitor service metrics, store and process data. The metrics are:

- Interaction: Can the service interact with its surroundings?
- Computation: Can the service provide an execution environment?
- Storage: Can the service store data?
- Monitoring: Can the service monitor own service metrics and/or other service metrics?

The criterion conditions:

Table 6.1. The Criterion condition of the Functionality Metrics

Service	Interaction	Compute	Storage	Monitoring
ROS	Yes	Yes	Files	Own Metrics
AWS IoT	No	Partially	No	Own Metrics
Lambda	No	Yes	No	Own Metrics
DynamoDB	No	No	Database	Own Metrics
CloudWatch	No	No	No	Multiple Service Metrics

The functionality measurement in table 6.1 demonstrates the flexibility in employing computational operations. AWS IoTs criterion condition for the metric compute is set to partially, this is because the platform only supports simple computing operations and not software packages i.e. analysis libraries. ROS and Lambda supports any kinds of computations including software packages, however Lambda is restricted to a disk capacity of 512MB.

Interoperability

Interoperability is the services ability to exchange data with other services in the platform. The metrics are:

- Pattern: What messaging pattern does the service provide?
- API/Protocol: What communication protocol or API (Application Programming Interface) do the service support?
- Payload: What payload is used by the service?

The criterion conditions:

Table 6.2. The Criterion Conditions of the Interoperability Metrics

Service	Pattern	API/Protocol	Payload
ROS	Publish/Subscribe	ROSTCP/ROSUDP, AWS SDK, AWS MQTT SDK	ROS Messages
AWS IoT	Publish/Subscribe, Push	MQTT, IoT Rules	JSON
Lambda	Push & Pull	AWS SDK, DynamoDB Streams	JSON
DynamoDB	-	-	JSON

ROS supports pure MQTT to external topics/channels through the AWS MQTT SDK, however the native ROS communication do not support pure MQTT. Therefore, the messages transferred between the two platforms must be repacked before they are utilized

by the local robot. The repacking occurs locally on the robot as illustrated in figure 4.4. Both ROS and AWS IoT utilizes publish/subscribe messaging pattern making the two platforms synchronized.

Composability & Scalability

Composability is the services ability to be modular and stateless. Scalability is the services ability to scale on demand. The metrics are:

- Modularity: Can the service be deployed independently?
- Stateless: Can the service treat invocations independently?
- Concurrency: Can the service scale in number of instances?
- Elasticity: Can the service automatically adapt workloads on demand?

The criterion conditions:

Table 6.3. The Criterion Conditions of the Composability & Scalability Metrics

Service	Modular	Stateless	Concurrency	Elasticity
ROS	Yes	Yes	Limited	No
AWS IoT	Yes	Yes	Unlimited	Yes
Lambda	Yes	Yes	Unlimited	Yes
DynamoDB	Yes	Yes	No	Limited

DynamoDBs criteria condition for the metric elasticity is set to limited. This is due to the DynamoDB tables utilizing stream-based events, these events are limited by the partitioning of the stream as explained in section 4.5.3. However, the DynamoDB tables utilizing non stream-based events automatically adapt the workload on demand; read/write operations. ROS criteria condition on concurrency is set to limited as the number of nodes and topics/channels is dependent on the ROS master as explained in section 4.2.

6.2 Delays

The research question answered in this section is the following:

- RQ1: How is a distributed network designed and developed combining an IoT platform with a robotic platform?
  - RQ1.1: Is it feasible to use an IoT platform together with a robotic platform in terms of processing and network delays?

Details of the design and development process of the distributed network is presented in Chapter 4 and 5. The feasibility of the IoT platform is measured with end-to-end delays, where the round-trip is the time it takes for a package to go back and forth, including the utilization time. The RC car is placed in Stockholm using WiFi, the IoT platform and AWS Lambda managed by AWS are placed in Dublin. Publishing frequencies from the robot spanning from 1 to 50 Hz generated the same result.

Table 6.4. Round trip delays

Service	Round-Trip (ms)	Size (KB)	Frequency (Hz)
ROS - IoT - AWS Lambda 1 (128MB)	150-200	0.5	1-50
ROS - IoT - IoT Rule Republish	100-110	0.5	1-50
ROS - IoT	50-60	0.5	1-50

The communication pattern publish/subscribe is utilized both by ROS and AWS IoT making the transmission of messages fast and synchronized. The delays between the IoT platform placed in Dublin and a robot utilizing ROS placed in Stockholm is around 25-30ms according to the results (one-way-trip). The delays almost doubles for ROS - IoT - IoT Rule Republish, the reason for the extra delays is due to internal transmissions within the AWS infrastructure. Similar results for ROS - IoT - AWS Lambda 1 (128MB), with a delay of 150-200ms for a round-trip including the utilization time of 20-40ms for Lambda 1 (128MB). The extra delay is a result of the transmissions between AWS IoT and AWS Lambda, the reason for this could be the different communication patterns utilized by the services.

The proposed platform demonstrated that it is feasible to combine ROS with AWS IoT for time sensitive application in terms of network delays. Taking into account that a ping round-trip from Stockholm to Dublin takes approximately 50ms [69], the latencies for a round trip between ROS and IoT shows that the majority of the delays consists of network latencies due to the traveled distance. By choosing services in regions closer to robot, the delays can be further decrease. However, the time at which this report is written, the closest placement of the Amazon services is Dublin, Ireland.

6.3 Utilization time & Visualization

The research question answered in this section is the following:

- RQ2: How are centralized functionalities designed and developed combining cloud computing micro-services with a robotic platform?
  - RQ2.1: Is it feasible to store and process sensor data in the cloud utilizing micro-services in comparison with traditional on premises hardware?
  - RQ2.2: Is it feasible to visualize streaming data processed with micro-services in near real-time?

Details of the design and development process of the centralized functionalities are presented in Chapter 4 and 5. RQ 2.1 and RQ2.2 are answered in the following two subsections.

RQ2.1

The feasibility to store and process sensor data is measured with the utilization time; the service ability to utilize different operations presented in table 6.5 and 6.6. Table 6.5 shows the utilization time of Lambda 1, Lambda 2 and ROS.

Table 6.5. Utilization Time

Service	Utilization Time (ms)
AWS Lambda 1 (128MB)	20-40
AWS Lambda 1 (256MB)	10-15
AWS Lambda 1 (512MB)	8-12
AWS Lambda 2 (128MB)	12
ROS	2

The table below shows the utilization time of query and scan operations with the DynamoDB database.

Table 6.6. DynamoDB Query & Scan operations

DynamoDB Query & Scan Table	Utilization (s)	Message Size (MB)	Data-points
AWS Lambda Query 128MB	20,6	4,95	14368
AWS Lambda Query 256MB	10,4	4,95	14368
AWS Lambda Query 512MB	5,5	4,95	14368
AWS Lambda Scan 256MB	24,7	4,95	14368
AWS Lambda Scan 512MB	12,9	4,95	14368
On premises Query	6,1	4,95	14368
On premises Scan	14,1	4,95	14368

Lambda 1 and Lambda 2 (AWS Lambda Stream) displays the time for Lambda to initiate the function and execute it. The execution of the function is explained in section 5.5. ROS displays the utilization time to run one cycle for the ROS node described in section 5.5.

Increasing the Lambda function compute resources displays similar improvements of the utilization time for Lambda Query, Lambda Scan and Lambda 1. In table 6.6 the increment from 256MB to 512MB overcame the performance of the on premises computer. These operations demonstrate the services ability to query and scan 14369 data points to be used for further analysis.

The results demonstrate that it is feasible to store and process sensor data in near real-time in the cloud utilizing micro-services. Moreover, these results displays that the bottleneck for both queries and scans is the available processing power. The FaaS Lambda

produces faster response then the on premises computer running a 7th generation Intel Core i7.

RQ2.2

Figure below demonstrates visualization of raw sensor data in near real-time with Microsoft Power BI.

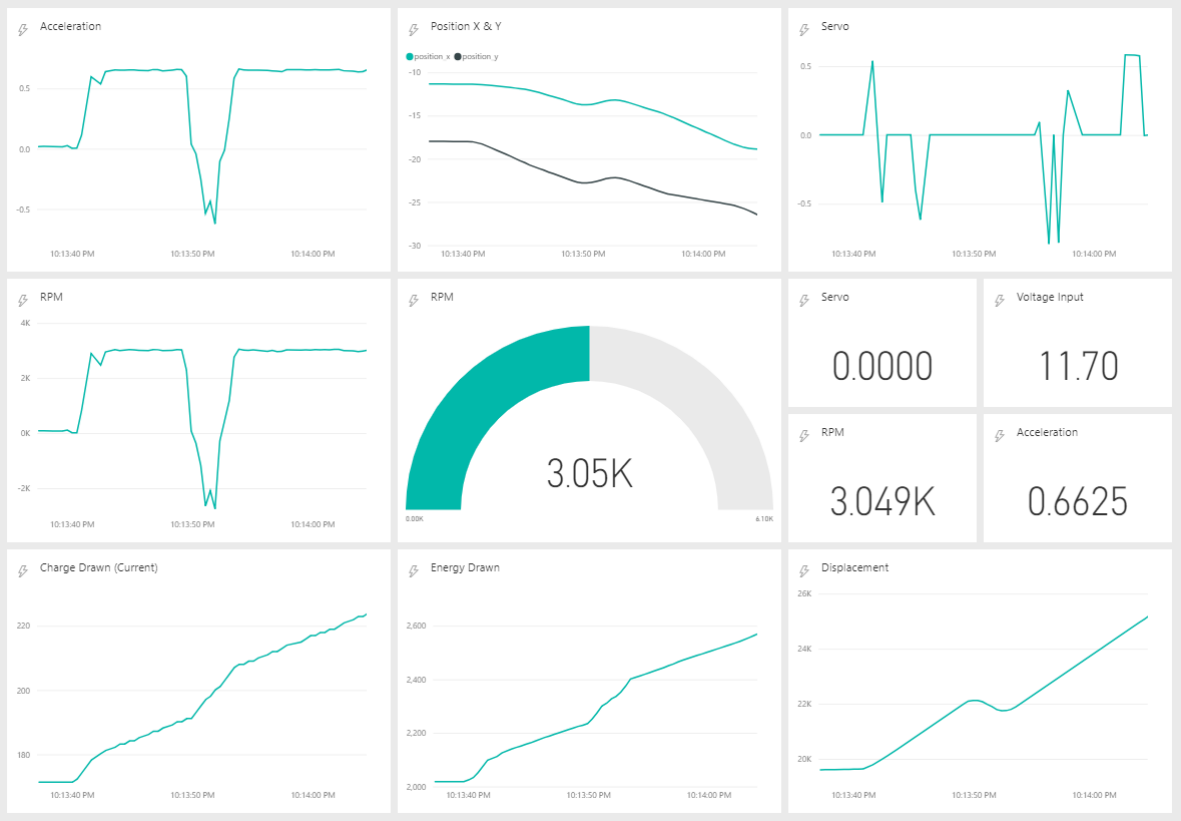

Figure 6.1. A snapshot of Microsoft Power BI dashboard on a web-browser.

The round-trip delays from ROS to PubNub is around 250-300ms, the delays between PubNub and Microsoft Power BI is not measured due to limitations in the Microsoft Power BI to publish back the messages. However, one can certainly assume that a one-way-trip between ROS and Microsoft Power BI is less then 300ms, moreover the delays are unnoticeable for the human eye.

This experimental setup uses AWS Lambda to process the data which utilizes an asynchronous push messaging pattern. However, both PubNub and AWS IoT utilizes a publish/subscribe messaging pattern resulting in messaging pattern alterations, as a consequence this could slow down the process. Both PubNub and ROS utilizes the

publish/subscribe messaging pattern, enabling ROS nodes to directly connect to PubNub to reduce the delays. However, this solution contradicts the purpose of this thesis which is to offload the end devices.

Based on the results, it is feasible to visualize streaming data processed with micro-services in near real-time. Furthermore, both PubNub and Microsoft Power BI are serverless services, making them a part of the micro-service cluster.

Chapter 7

Discussion

This chapter evaluates the platform performance based on the results: delays & utilization, platform abilities and visualization.

The performance of the IoRT platform varies in terms of delays and utilization time depending on different factors. One contributor to the delays between the services is the network latency introduced due to the traveling distance. Another contributor to the latencies is the communication pattern shifts between the services.

The message delay for a round-trip between ROS and AWS IoT is around 50-60ms, according to [70] which displays inter-region ping latencies between AWS regions. A round-trip from the region AWS Dublin named eu-west-1 to the region AWS Frankfurt named eu-central-1 takes around 50ms. Considering that the RC car was placed in Stockholm and the AWS IoT core located in Dublin, the comparison between the two trips shows promising results.

AWS Lambda has a disk capacity restriction to 512MB, this can be a problem when installing libraries with large software packages. However, the IoT platform enables connectors to other cloud based services; virtual machines, containerized infrastructure and other services provided by the cloud vendor that do not have these restrictions.

The near real-time visualization showed promising results in terms of delays even though the end-to-end transmission included alterations in communication pattern. Furthermore, Microsoft Power BI supports multiple devices to visualize the dashboard simultaneously, on any web browser or on the application supported by Windows, Android and IoS.

Chapter 8

Conclusion

8.1 Conclusion

This paper proposes a practical approach of an IoRT platform. Three main technologies are investigated and applied: robotics, cloud services and IoT. Additionally, a data analysis process is investigated and used as a template to design the test platform.

In conclusion, this approach makes three main contributions:

- A platform combining the robotic platform ROS and IoT platform IoT Core that distributes information over the Internet utilizing a common communication pattern.
- A platform combining the robotic platform ROS and centralized cloud services: AWS Lambda and DynamoDB, that stores and processes sensor data in near real-time in the cloud utilizing micro-services.
- A platform that visualizes streaming data from the micro-services in near real-time.

Given these results, this approach proves that it is feasible utilizing this IoRT platform to distribute information, store data, process data and visualize data from multiple robots in near real-time.

8.2 Future Work

- **Implement a Complete Analysis Process Utilizing the Micro-services**

  In the current implementation of the experimental setup, the analysis process is only used as a template to evaluate the cloud service capabilities. As an analysis process was developed, there was not enough time to implement it, furthermore distribute it over several lambda function.

- **Connect more Robots**

  Test the capabilities of the platform connecting more robots to the distributed network.

- **Test the Platform with Larger message sizes**

  Load the distributed network with larger message size, i.e. transmitting a camera feed and measure the delays.

- **Implement File Storage in the cloud**

  Utilize Amazon S3 buckets to store raw files in the cloud. The AWS framework provides APIs for DynamoDB, AWS IoT and AWS Lambda to directly connect to the S3 buckets.

References

[1] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms", in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2017, pp. 162–169.

[2] *Microservices, What are microservices?*, `http://microservices.io/`, Accessed: 2018-06-30.

[3] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: Extending serverless computing to the edge of the network", in *Proceedings of the 10th ACM International Systems and Storage Conference*, ACM, 2017, p. 28.

[4] *The International Telecommunication Union internet of things global standards initiative*, `https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`, Accessed: 2018-03-01.

[5] *Near Real Time*, `http://wiki.c2.com/?NearRealTime`, Accessed: 2018-11-10.

[6] P. Simoens, M. Dragone, and A. Saffiotti, "The internet of robotic things: A review of concept, added value and applications", *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, p. 1 729 881 418 759 424, 2018.

[7] *Configuring Lambda Functions*, `https://en.wikipedia.org/wiki/Hardware_abstraction`, Accessed: 2018-07-28.

[8] P. P. Ray, "Internet of robotic things: Concept, technologies, and challenges", *IEEE Access*, vol. 4, pp. 9489–9500, 2016.

[9] *The Internet of Robotic Things (IoRT), definition, market and examples i-scoop*, `https://www.i-scoop.eu/internet-of-things-guide/internet-robotic-things-iort/`, Accessed: 2018-05-14.

[10] *ROS*, `http://www.ros.org/`, Accessed: 2018-06-06.

[11] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects", in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 1.

[12] R. D. Arnold and J. P. Wade, "A definition of systems thinking: A systems approach", *Procedia Computer Science*, vol. 44, pp. 669–678, 2015.

[13] R. Edson, "Systems thinking. applied. a primer", *Asyst Institut Oct*, vol. 8, 2008.

[14] *Introducing JSON*, https://www.json.org/, Accessed: 2018-07-04.
[15] *Docker*, https://www.docker.com/, Accessed: 2018-05-28.
[16] *Kubernetes*, https://kubernetes.io/, Accessed: 2018-05-29.
[17] *Networked Robots ieee robotics and automation society*, http://www.ieee-ras.org/technical-committees/117-technical-committees/networked-robots/146-networked-robots, Accessed: 2018-05-04.
[18] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges", *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.
[19] 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.
[20] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud networked robotics", *IEEE Network*, vol. 26, no. 3, 2012.
[21] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues", *IEEE Access*, vol. 4, pp. 2797–2807, 2016.
[22] L. Riazuelo, J. Civera, and J. Montiel, "C 2 tam: A cloud framework for cooperative tracking and mapping", *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.
[23] *Publish Subscribe pattern*, https://en.wikipedia.org/wiki/Publish-subscribe_pattern, Accessed: 2018-07-04.
[24] *Request Response*, https://en.wikipedia.org/wiki/Request-response, Accessed: 2018-07-04.
[25] *Invocation Types*, https://docs.aws.amazon.com/lambda/latest/dg/invocation-options.html, Accessed: 2018-07-04.
[26] O. Standard, "Mqtt version 3.1. 1", *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, vol. 1, 2014.
[27] *The WebSocket Protocol*, https://tools.ietf.org/html/rfc6455, Accessed: 2018-07-02.
[28] *Hypertext Transfer Protocol*, https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, Accessed: 2018-07-04.
[29] *AWS IoT Core*, https://aws.amazon.com/iot-core/, Accessed: 2018-07-03.
[30] *Azure IoT Hub*, https://azure.microsoft.com/en-us/services/iot-hub/, Accessed: 2018-07-03.
[31] *Google Cloud IoT Core*, https://cloud.google.com/iot-core/, Accessed: 2018-07-03.
[32] P. P. Ray, "Internet of robotic things: Concept, technologies, and challenges", *IEEE Access*, vol. 4, pp. 9489–9500, 2016.
[33] *DDS Portal*, http://portals.omg.org/dds/, Accessed: 2018-06-12.
[34] *Internet of Robotic Things Market by Component (Sensor, Power, Control), Service (Professional, Managed), Platform (Device, Application, Network), Software (Analytics, Data, Security, Monitoring, Bandwidth), Application - Global Forecast to*

2022, https://www.researchandmarkets.com/research/73mzpl/internet_of, Accessed: 2018-07-03.
[35] L. Wang, M. Liu, and M. Q.-H. Meng, "Real-time multisensor data retrieval for cloud robotic systems", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 507–518, 2015.
[36] D. M. Lofaro and A. Asokan, "Low latency bounty hunting and geographically adjacent server configuration for real-time cloud control", in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 5277–5282.
[37] *Rapyuta, A Cloud Robotics Platform*, http://rapyuta.org/, Accessed: 2018-06-28.
[38] *Roboearth*, http://roboearth.ethz.ch/, Accessed: 2018-07-02.
[39] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, 2015.
[40] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots", in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 3084–3089.
[41] *Spacebrew*, http://docs.spacebrew.cc/, Accessed: 2018-07-27.
[42] A. B. M. Pereira, R. E. Julio, and G. S. Bastos, "Rosremote: Using ros on cloud to access robots remotely", in *Robot Operating System (ROS)*, Springer, 2019, pp. 569–605.
[43] *Spacebrew Getting Started*, http://docs.spacebrew.cc/gettingstarted/, Accessed: 2018-07-27.
[44] *TCP ROS*, http://wiki.ros.org/ROS/TCPROS, Accessed: 2018-06-06.
[45] *UDP ROS*, http://wiki.ros.org/ROS/UDPROS, Accessed: 2018-06-06.
[46] *AWS IoT Core*, https://aws.amazon.com/iot-core/, Accessed: 2018-06-01.
[47] *POST (HTTP)*, https://en.wikipedia.org/wiki/POST_(HTTP), Accessed: 2018-07-05.
[48] *AWS Lambda*, https://aws.amazon.com/lambda/, Accessed: 2018-06-01.
[49] *AWS Invoking Lambda Functions*, https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-functions.html, Accessed: 2018-06-03.
[50] *Amazon DynamoDB*, https://aws.amazon.com/dynamodb/, Accessed: 2018-06-01.
[51] *Query and Scan the Data*, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.NodeJs.04.html, Accessed: 2018-07-28.
[52] *Amazon CloudWatch*, https://aws.amazon.com/cloudwatch/, Accessed: 2018-07-28.
[53] *ROSbridge*, http://wiki.ros.org/rosbridge_suite, Accessed: 2018-06-01.
[54] *mqtt bridge*, https://github.com/groove-x/mqtt_bridge, Accessed: 2018-06-06.
[55] *aws iot sdk device sdk python*, https://github.com/aws/aws-iot-device-sdk-python, Accessed: 2018-06-06.
[56] *aws mqtt brdige*, https://github.com/bydottck13/aws_mqtt_bridge, Accessed: 2018-06-06.

[57] *modified aws mqtt brdige*, https://github.com/robertoooo/ROS_AWS_IoT.git, Accessed: 2018-06-10.

[58] *Rules for AWS IoT*, https://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html, Accessed: 2018-06-01.

[59] *Republish Action*, https://docs.aws.amazon.com/iot/latest/developerguide/republish-rule.html, Accessed: 2018-07-28.

[60] *Capturing Table Activity with DynamoDB Streams*, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html?shortFooter=true, Accessed: 2018-06-06.

[61] *AWS Understanding Scaling Behavior*, https://docs.aws.amazon.com/lambda/latest/dg/scaling.html, Accessed: 2018-06-03.

[62] *AWS SDK for Python (Boto3)*, https://aws.amazon.com/sdk-for-python/, Accessed: 2018-06-04.

[63] J. Han, M. Kamber, and J. Pei, "Data mining: Concepts and techniques (the morgan kaufmann series in data management systems)", *Morgan Kaufmann*, 2000.

[64] *PubNub*, https://www.pubnub.com/, Accessed: 2018-07-27.

[65] *What is PubNub?*, https://support.pubnub.com/support/solutions/articles/14000046386-what-is-pubnub-, Accessed: 2018-07-27.

[66] *PubNub Supported Protocols*, https://support.pubnub.com/support/solutions/articles/14000043529-what-transports-are-supported-and-what-are-the-fallbacks-, Accessed: 2018-07-27.

[67] *Microsoft Power BI*, https://powerbi.microsoft.com/en-us/, Accessed: 2018-07-27.

[68] *Create Realtime Charts and Graphs with Microsoft Power BI*, https://www.pubnub.com/tutorials/microsoft-power-bi/streaming-business-data-to-dashboards/, Accessed: 2018-07-27.

[69] *Global Ping Statistics*, https://wondernetwork.com/pings, Accessed: 2018-07-28.

[70] *AWS Inter-Region-Latency*, https://www.cloudping.co/, Accessed: 2018-07-28.