Group Report Template

Stefan-Daniel Horvath*, Hi IM Micahel*, Student 3*, Student 4*, Student 5*,
University of Southern Denmark, SDU Software Engineering, Odense, Denmark
Email: * {Stefan-Daniel Horvath,student2,student3,student4,student5}@student.sdu.dk

Abstract—
Index Terms—Keyword1, Keyword2, Keyword3, Keyword4, Keyword5

I. Introduction and Motivation

The advancement of technology is at an all-time high and the possibilities using technology are expanding rapidly. Several industries are benefitting using assembly robots, self-driving forklifts, etc. People, machines, and products are directly related to each other. The term "Industry 4.0" describes the intelligent networking of machines and processes for industry with the help of information and communication technology [?]. The principles of Industry 4.0 result in several benefits, including the possibility of flexibility within production, ability to quickly change a production line for a different task, and many more [?].

This concept introduces the need for many different systems to be able to communicate in one way or another, which is one of the big challenges Industry 4.0 is faced with [?]. Therefore, for a system to successfully adhere to Industry 4.0 principles it must be capable of communicating between subsystems and various types of hardware within the system.
Industry 4.0 might originally have been coind for the manufacturing industry, but the potential of the concept is not limited to this industry alone. There is great potential for farming to adapt to the principles also. By incorporating the power of real-time data, IoT (internet of things) and analytics, farmers can respond to more nuanced changes in the environnment and the health of the animals. This not only has great benefits for the productivity of the farm, but also has benefits for a more sustainable farming industry.
In this paper the intent of the project is to create a system that can handle various aspects of production, including monitoring biometrics, managing feed, and handling orders in a fully automated livestock farming system.

Therefore this paper designs and implements an architecture based on a smart livestock farming system. The system is evaluated by its ability to adhere to stated quality attributes. The motivation behind designing such a system is based on the need for interoperability between different systems, which in its current state in farming is one of the major challenges. The lack of communication between these technologies hampers their synergistic use by farmers [?]. Through our work, we aim to address these challenges and contribute to the seamless integration of diverse technologies.

The structure of the paper is as follows. Section II outlines the research question and the research approach. Section III describes similar work in the field and how our contribution fits the field. Section IV-A presents a use case of the system, which gives us a better representation of what the system is able to do. The use case serves as input to specify QA requirements for the system IV-B. Section V introduces the proposed software architecture design for the system. Section VI evaluates the proposed architecture on tests conducted on the system and analyzes the results against the stated QA requirement.

II. Problem and Approach

Problem. Productivity in livestock farming har big implications on the environment. While traditional practices have served the industry, their capacity to adapt to rapid changes remian limited. The Industry 4.0 framework offers a solution through its emphasis on intelligent networking and interoperability across systems. By adapting such a framework, efficiency in farming operations can be significantly improved, leading to higher productivity and potentially higher living environment for livestock. Moreover, this could also benifit the principles of sustainable farming. By addressing the challenges of system integration and real-time response to environmental and health variables, the industry could unlock the potential for a holistic, automated livestock farming model that is scalable and future-ready.

Research questions:

1) How can different architectures support the stated production system requirements?
2) Which architectural tradeoffs must be taken due to the technology choices?

Approach. The following steps are taken to answer this paper's research questions:

1) Develop the overall architecture of the system.

a) Identify the services and servers that are needed to support the production system.
b) Identify the usecases that the system should support.
c) Identify the Quality Attributes that the system should support and how they are prioritized.
d) Identify the non-functional requirements that the system should support.

2) Research the technologies that could be used to support the system and the tradeoffs that are made by using them.
3) Develop a prototype of the system.
4) Evaluate the prototype based on the Quality Attributes that are identified in step 1.
5) Analyze the results and answer the research questions.

III. Related work

This section addresses existing knowledge and contributions by examining Industry 4.0 and its current use within the livestock and agricultural domain. In total, eight papers are investigated using a systematic literature review.

Over the past years, large parts of the industry have been undergoing the revolution to Industry 4.0, integrating higher levels of technology into the lifecycle of products. Order systems, production environments and shipment as well as maintenance of products became more automated and integrated different, interconnected smart systems. However, the primary sector, mainly the agricultural part of industry, has not kept up with the rapid changes to modern production standards [?]. This project plan to design an Industry 4.0 architecture for a prototype system for the livestock farming domain. For that, the project begins with evaluating which needs and challenges are already reported on for similar systems in the domain and find existing suggestions for designing such systems. In detail, a small but concise systematic literature review is performed considering the following research questions:

- What functional requirements are considered important for agricultural industry 4.0 systems in existing literature?
- Do suggestions for parts of the architecture and technologies of systems exist that support the requirements of agricultural industry 4.0 systems?
- What are current challenges within agricultural industry 4.0 systems in use?

To answer these questions, an examination of the scientific reports ragarding Industry 4.0 concepts and software in the livestock farming and agricultural context is conducted. For the process of identifying appropriate research, a semi-automated search strategy as well as snowball techniques after finding promising starting point papers. This search is conducted across a spectrum of academic digital databases, including MPBI, ProQuest, DOAJ, Google Scholar, and ScienceDirect. A set of inclusion criteria was utilized with the purpose of gathering appropriate articles, reviews and case studies published in English within a timeline of the last 7 years. This approach is designed to collate a comprehensive and current body of knowledge, providing a solid foundation for the study.

The keyword search focused on a list of keywords and key phrases related to the research questions. That list included the words Industry 4.0, agriculture, agriculture 4.0, farming, livestock, and any related phrases. Evaluating the search results, the ones which were applicable to the above-mentioned criteria and related to the research questions and challenges were chosen (excluding some papers with a focus on machine learning, since they were not related to the architectural parts of software design). Additionally, ensuring that the papers present a balanced view by considering various perspectives and supporting their arguments with evidence from diverse sources, such as references and/or observed data was important. It was also checked that the authors have taken care to present the information objectively, avoiding bias in their analysis.

To report similar or different aspects covered in the different papers, it was determined three categories covered in most of the papers. These categories are “Functional requirements and quality attributes”, “Architectural aspects” and “Existing challenges/problems”. For each of them, an overview of the most important points made is given below.

A. Functional requirements and quality attributes

The most common quality attribute for agriculture 4.0 systems throughout the list of papers is interoperability of different subsystems. All the listed papers either mention this attribute directly or refer to middleware products, which are the heart of interoperability between different subsystems. Similarly, flexibility of the system, for example being able to add new subsystems or change parts of the software, is addressed as an important attribute in several papers [?], [?], [?], [?]. Another common attribute is scalability, which papers [?], [?], [?] directly address. In the context of a farm there is a possibility of it growing over time, it is therefore important that the system capable of scaling and can adapt to larger amounts of data and the added workload. A few of the papers [?], [?] also mentions security as an important attribute of large-scale systems, so that the data and system can’t be tampered with. While there were little contradicting opinions on attributes, different papers had different focus points. A few attributes

worth mentioning include component availability [?], reconfigurability [?] or accuracy of systems [?].

B. Architectural aspects

Several of the papers formulate guidelines or opinions on different architectural aspects of agricultural 4.0 systems. To address the above mentioned interoperability requirement, most papers [?], [?], [?], [?], [?], [?] directly focus on middleware components that are used to orchestrate and abstract between different subsystems. In this context, they also mention well-defined interfaces for the communication between different components. Another directly addressed aspect is the virtualization [?], [?] of different components in the actual physical farm, so that they can be included in the software ecosystem. While different explicit technologies are addressed by different papers (Docker, Kafka, etc.), a more abstract technological term across different papers is "IoT" [?], [?], [?], [?]. Since many systems in the agricultural context rely on sensor data and physical devices (robots, vehicles), the inclusion of IoT devices is necessary for agriculture 4.0 systems.

C. Existing challenges/problems

Several of the papers [?], [?], [?], [?], [?], [?], [?] come to the conclusion that many technologies exist that try to make livestock farming smarter and more efficient, but the problem with the technologies is that often coordination problems occur making the interoperability difficult to achieve. A gap in this is the need for common communication protocols or orchestration middleware that these technologies can share so data can be used for shared interests. Additional reported challenges exist within the integration of 4.0 systems to the actual farms, which include high financial costs and missing expertise for technological equipment and systems [?], [?], [?]. To counteract some of these issues, there could be governmental policies, which are at this point still undeveloped [?]. Another reported issue might be bandwidth problems in rural areas, which would restrict large interconnected systems with many components [?], [?].

All contributions provide valuable knowledge about Industry 4.0 and its usage within modern agriculture and livestock farming. However, a notable gap in existing literature is the absence of real-world examples of successful or unsuccessful implementations of livestock farming systems that apply Industry 4.0 principles. Developing practical examples can help provide a clearer understanding of how to solve the challenges of Industry 4.0 within the livestock farming domain.

IV. Use Case and Quality Attribute Scenario

This Section introduces the use cases and the specified QASes. The QASes are developed based on the use cases.

A. Use cases

To better grasp the ability of the system, several use cases have been constructed. The use cases assists to identify systems and subsystems needed to implement the system. Its important to also consider the quality of the system that goes beyond just functionality. This will be refered to as quality attributes [?].

1) Automatic feed ordering: The first use case outlines an automatic order placement for more feed, which is triggered by low feed stock levels. This use case contains three actors: A feed distributor, a control system and a feed vendor. Events in this use case is the feed distributor continously monitors feed levels and when the stock reaches below a certain amount it triggers a signal which gets sent to the control system. The control system recieves the signal and creates the order for more feed, which it then forwards to the vendor system. The vendor recieves order and checks for availability of feed for the incoming order. In case of resource shortage the vendor notifies the control system, so that the farmer can take action accordingly. If resource are available the vendor schedules packaging of the feed and makes sure it is shipped to the farm and the control panel is notified of the order completion. From the time that the feed distributor recieves a signal of low stock the whole sequence must be finished within a resonable timeframe.

From this use case it is possible to derive several systems already for the management of livestock feed. First off for the feed distributor to know how much feed is currently in the silo on the farm, it needs information on the current stock. This could be done with a weight sensor in the silo but implmentations could varie. The feed distributor is then designed to act upon the stock of the silo reaches below a certain point.

The use case also describes a control system which handles the signal from the feed distributor and the creation of the order for the vendor, the vendor in this case is an external system that is not included in the prototybe system.

2) Detect changes in livestock conditions: This use case describes a scenario that can alert the farmer of the wellbeings of the livestock. This use case contains four actors: Sensors, a health analyzer, a control system and the farmer. Eevnts in this use case is first of initiated by continously data being generated by sensors that are either attached to the livestock or its surroundings. This data is being transmitted into a storage system which then can be used to analyze the health of the livestock. The health analyzer will then analyze the sensor data and if the data suggests e.g. a disease or other forms of the livestocks condition being worsened the system will

transmit a warning to the control panel. The control panel can be accessed by the farmer and the alert from the health analyzer will be able to tell the farmer which actions are required. The system should be able to handle the addition of new livestock, this will result in more sensors being added which should be completed with no downtime of the system. In case of added load by the addition of more sensors, the system should be able to handle the increased load by dynamically scaling the storage.

From this use case more systems was identified to be implemented in the system. The control panel is already mentioned from the previous use case. A system for the sensors are needed to transmit the data in to some form of storage system. The sensors will be generating a great amount of data, so the system should be able to handle a high throughput of data and also store a large amount of it. This storage system can then be accessed by the health analyzer, which should be able to able to alert the farmer and also inform of the necessary actions.

B. Quality attribute scenarios

To define success criteria for the system, several quality attribute scenarios have been created. These are used for setting up a scenario in which one of the quality attributes of the system gets addressed [?]. This is helpful for when doing testing on the system later. A template for defining QASes is used for the creation of these scenarios. The templates captures a scenario through Source, Stimulus, Artifact, Environment, Response and Response measure to which the requirement should apply [?].
For the system in this paper there is specified three main requirements:

1) Production software must be able to exchange and coordinate information to execut a prodcution and change production.
2) Production software must run 24/7
3) Production software must be continously deployable

These three requirements can be directly translated to quality attributes. The first one refers to interoperability, the second availability and the third is described by deployability.

1) Interoperability: Table I is a scenario for the interoperability quality attribute, which referes to the ability of two or more systems to be able to exchange and use information [?]. For the interoperability QAS, the scenario described is the ability of the system to be able to react to low feed stock and therefore make sure the necessary communication is carried out with other systems, this includes both the external vendor but also the control system. The Source in this scenario is the feed distributor system, which triggers the Stimulus, in this case the indication of low feed stock. The stimulus arrives at the Artifact (the control system), which is responsible for creating the order for feed. The Environment assumes the condition of the component / system, which in this case is assumed to be under normal operation. The Response describes a process that handles the stimulus, so the system should be able to successfully create and forward the order. To be able to determine if the requirement from the scenario is satisfied the Response measure is specified, which in this scenario specifies that the whole process should be transmitted and acknowledged within a certain timeframe (in this case 5 seconds).

TABLE I: Interoperability quality attribute scenarios

Portion of scenario	Value
Source	Feed distributor system.
Stimulus	A signal from the feed distributor indicating low feed stock.
Artifact	Control system.
Environment	Normal operation during low feed stock.
Response	The control system successfully creates and forwards an order to the external vendor system.
Response measure	The order is accuratly transmitted and the external vendor acknowledges it wihtin 5 seconds.

TABLE II: Availability quality attribute scenarios

Portion of scenario	Value
Source	High sensor data throughput.
Stimulus	The message bus experiences an overload, causing the loss of a partition.
Artifact	Message bus.
Environment	Overloaded condition.
Response	Due to the replication factor, the system ensures that the lost partition's data is still available.
Response measure	The system maintains data integrity and availability through replication, even when a partition is lost.

2) Availability: Table II is a scenario for the availability quality attribute, which refers to the systems ability to be ready to carry out a task, but it also covers the systems ability to mask or repair faults [?]. For the availability QAS, the scenario described is the systems ability to handle added load from the throughput of sensor data. The system should from the added load be able to circumvent the loss of data or the system becoming degraded in performance. The Source is coming from the sensor data throughput, which as described by the Stimulus causes loss of a partition within the Artifact (the message bus). The Environment is assumed to be in an overloaded condition due to the amount of data causing the loss of a partition in the message bus. The Response handles the stimulus by applyin tactics or

patterns to address the availability of the system in the case of a partition loss. The Response measure indicates the handling of the fault should result in no data lost and the system to be with no downtime.

TABLE III: Deployability quality attribute scenarios

Portion of scenario	Value
Source	Increased livestock population.
Stimulus	Addition of more sensors to the livestock monitoring system.
Artifact	Sensor system.
Environment	Deployment of additional sensors.
Response	The system allows plug-and-play functionality, allowing quick and straight forward integration of additional sensors.
Response measure	New sensors can be deployed and be operational without downtime or configuration challenges.

3) Deployability: Table III is a scenario for the deployability of the system, which refers to the ability to continously update a system without the need for downtime [?]. The idea is that integration can be quickend and development can push out updates to the production system as fast as possible. For the deployability QAS, the scenario described highlights the systems ability to increase the amount sensors in the system. The system should be able to handle the added sensors without the need for downtime in the system. The Source is the increased livestock population which triggers the Stimulus which is the addition of sensors. This is handled within the Artifact (the sensor system), and the Environment is the deployment of the additional sensors. The Response describes the systmems ability to allow plug-and-play functionality, that allows quick and easy integration of additional sensors. The Response measure implies the process should be completed with no system downtime and configuration challenges.

V. The solution

This section outlines the proposed technological solution for the medium-sized cattle farm. The design integrates various hardware and software components such as sensors, Java-based subsystems, message bus for middleware communication and Big data server for storing data. The whole purpose of this solution is to address the unique needs of the farm's operations which include animal care, feed management, and sales.

A. System Overview

The proposed system integrates biometric and feed monitoring, automated feeding, and resource management, ensuring efficiency and sustainability in farm operations. It is designed to manage around 100 animals, utilizing 200 to 300 sensors for various monitoring purposes such as monitoring the heart rate of cows, temperature or humidity of silos. In order for the system to be successful, continuous interoperability among various departments and their Java-based subsystems need to exist. Therefore, a robust middleware communication system facilitated by a central message bus was chosen to be used as the optimal architecture.

Figure (ref) depicts the initial architecture that was drafted for the system. A few of these systems are not present in the final system implementation, but there will be enough implementation to show how the system adheres to the required quality attributes which are specified in the quality attribute scenarios from section IV-B.

B. Middleware and Communication

For an Industry 4.0 system the middleware is the center of attention. From related work knowledge was obtained that highlighted the need for interoperability between different systems, this is where the message bus comes into play. Apache Kafka [?] has been selected for real-time data handling and as a central component of the message bus due to its distinct advantages in large-scale applications. Kafka offers high throughput and low latency, making it ideal for managing huge streams of sensor data in real-time. Furthermore, Kafka's ecosystem, including Kafka Connect, enables the efficient processing of big data, an essential requirement for the project.

The distributed nature of Kafka uses multiple brokers, which for Kafka means that it has several servers to recieve and send data. This helps avoid a single point of failure and ensures system reliability. Kafka uses multiple tactics to achieve it's fault tolerance, one being replication of data across multiple brokers, which ensures if a broker is lost no data will be lost [?]. This behavior is needed for the QAS described for availability in the system. Besides affecting the QAS, Kafka also has great support for a variety of different systems and languages [?], which will support the interoperability of the system.

The main downsides of using Kafka is the technological complexity and lack of management and monitoring tools. This presents a challenge and necessitates additional effort in system administration. Despite these challenges, Kafka's performance, scalability, and the ability to integrate with various technologies such as Hadoop for big data needs, make it the most suitable choice for the system.

C. Subsystem Integration

Each department's Java-based subsystem, operates through the middleware message bus, carrying out specific functions that are essential for each department needs.

The choice of using Java-based subsystems can be descirbed by several factors. First and foremost Java is an object-oriented programming language which makes for easier implementation as it allows for breaking the code into smaller bits [?]. Another important feature of Java is the ability to run on any machine regardless of the operating system or hardware, this also applies for being to run on any type of device. This supports the interoperability of the system as Java is able to connect and communicate with a wide variety of systems. It also supports deployability as the deployment of a Java-based system requires no modifciation regardless of where its running. Java also provides automatic garbage collection, which helps it manage memory effeciently [?]. This supports the availability of the system by reducing the likelihood of memory errors.

Even though Java supports several of the requirements for the system, it also has it's tradeoffs. Just as automatic garbage collection is a plus, it also affects the performance as its a continous process running in the background, whereas in languages like C or C++ they are not hampered by this [?].

D. Sensor Implementation

The farm's operations are monitored by an array of sensors that perform various functions. Cattle health sensors monitor vital signs such as heart rate and body temperature, providing critical data for early illness detection. Environmental sensors monitor conditions within the livestock's farm, including temperature and humidity levels within the fields, the barns and silos. Silo weight sensors monitor and measures the amount of feed stored, and feed weight sensors keep track of distribution of feed to the cattle farm.

All sensors are connected to a central middleware message bus, which use Kafka to help manage the real-time collection and streaming of sensor data. This design allows the immediate ingestion of data direct into HDFS, where it can be processed and analyzed. Kafka's distributed nature and streaming capabilities enable the system to handle the influx of data from potentially hundreds of sensors without loss of information or delay, ensuring that the most current data is always available for decision-making processes.

E. Database Management

A relational database using MySQL is employed across the Java-based subsystems, providing a consistent framework for data management. The use of MySQL which is a relational database makes sense for the most of the subsystems in the architecture, because here it is not necessary to store live data but more historical data. MySQL also supports some of the quality attributes for the system. Just like Java, MySQL is platform independent [?], which means it supports deployability by requiring no modification regardless of where it is running. The main disadvantage of MySQL is the poor performance when exposed to high loads [?]. Looking at the specified architecture there is another system for storing the sensor data, which will handle the larger amounts of data better.

F. Bio-monitoring Livestock System

1) Hardware: Sensors are attached to each animal for monitoring temperature and heart rate, ensuring 24/7 health surveillance.

2) Software: A Big Data architecture is employed, leveraging Kafka for robust data ingestion into HDFS. Kafka serves as a high-throughput distributed messaging system, capable of handling the streams of sensor data in real-time. It is chosen for its fault tolerance, durability, and ability to process data streams as continuous flows, which is critical for timely and accurate monitoring of livestock health.

3) Functionality: The system operates continuously, utilizing Machine Learning to analyze the data and detect early signs of illness, notifying in time the animal care department. Kafka's immediate and efficient processing of data ensures speed and time efficiency to the system.

G. Feed System

1) Hardware: Automated feeders equipped with sensors to control and monitor feed distribution.

2) Software: Java-based control system to manage feeding operations, ensuring consistent and accurate feeding.

3) Functionality: Real-time 24/7 monitoring with error detection and alerting mechanisms for immediate handling of issues.

H. Inbound System

1) Hardware: Feed silos equipped with scales and humidity sensors.

2) Software: Java and MySQL manage data capturing and processing, facilitating efficient feed utilization and ordering.

3) Functionality: Regular monitoring and data capture optimize feed management and resource planning.

I. Outbound System

Developed in Java, this MVC application manages customer orders, linking them with laboratory samples for efficient tracking and processing.

J. Hardware and Software Synergy

The sensor data is being mimicked by a Java application, that can simulate live sensor data. This allows for the posibility of evaluating on the systems capabilities in regards to data throughput.

K. Integration with Hadoop Distributed File System

The integration of Kafka with the Hadoop Distributed File System (HDFS) is a necessary decision to manage the large volumes of data generated by the biometric and environmental sensors. HDFS provides a reliable and scalable storage solution, capable of handling vast amounts of data while ensuring data integrity and accessibility. This integration allows for efficient data processing, storage, and retrieval, which are crucial for the real-time monitoring and analysis required in the bio-monitoring livestock system. HDFS supports several of the quality attributes required by the system. The main reasoning behind the use of it is because of the limited storage possibilites by MySQL, in HDFS it is possible to store extremely large amounts of data. HDFS enhances the availability by implementation of fault tolerance. As Kafka can replicate the data, same goes for HDFS, which means that it replicates the data across several nodes (physical computers) in a cluster (a group of physical computers) [?]. When a node fails no data is therefore lost, this refers to the tactic voting which defines replication as a way of detecting faults [?]. HDFS is also able to scale horizontally which allows for a simple way to add more capacity for storing data. HDFS is located within the Apache ecosystem which provides a handful of different technologies for different use cases [?], this improves the interoperability of the system as HDFS is able to communicate with various different technologies. As for the deployability of HDFS it is built on Java, which as mentioned can run on any system or hardware.

When using HDFS it is important to consider if the files that is being stored are not too small. This is one of the main disadvantages of HDFS, it has very poor handling of small files, in that case it is better to look at MySQL. HDFS also has its own processing when trying to read the data called MapReduce, which if the files are really large it can last quite a while for it to finish a MapReduce job. The architecture circumvents this by the use of Apache Hive. Hive allows the system to read the files with sensor data into a table and then use HiveQL which is an SQL like query language to then query the data [?]. The one downside that is provided by Hive is it doesnt handle real-time data querying, so in this system Hive is for looking at historical data [?]. Therefore the need for live data from the sensors is obtained from an intermediate data storage in the message bus (Kafka).

L. Containerization using Docker

For running the system it was containerized using Docker. Docker is a piece software that allows an open platform for developing, shipping and running applications [?]. It packages up code and its dependencies, which ensures that the applications run quickly and reliably from one environment to another [?]. For the architecture proposed in the solution, each system has its own container and with all the systems containerized, it is possible to perform tests on the system using a single machine. Containerization helps isolate each system and therefore reducing conflicts between systems. Docker makes use of a few tactics to adhere to these principles that are connected to the deployability QA. For managing the deployment pipeline it makes use of the Script deployment command tactic which describes a script can do all the steps necessary for executing the deployment of the system as a whole [?]. For managing the deployed system it also makes use of the Pacakge dependencies which essentially describes how it packages the system, that leads to containerization [?].

M. Discussion on Trade-offs

The selection of Kafka and HDFS, while beneficial for handling big data and ensuring scalability, faces trade-offs. The complexity of Kafka gives rise to the need of complicated technological knowledge and more detailed system management. Additionally, the start setup and maintenance of the Hadoop ecosystem require crucial resources and expertise. These trade-offs and challenges are justified by the long-term benefitial existence of a scalable and flexible system capable of adapting to the evolving farming needs. As for the use of Docker, some of the tradeoffs from Kafka and HDFS are also relevant here. This includes the inital added complexity for setting it up and in general expertise within the domain is needed for setup [?]. The tradeoffs are justified by the positives Docker provides which outweighs the negatives.

VI. Evaluation

This Section describes the evaluation of the proposed design. Section VI-A introduces the design of the experiment to evaluate the system. Section VI-B identifies the measurements in the system for the experiment. Section VI-C describes the pilot test used to compute the number of replication in the actual evaluation. Section VI-D presents the analysis of the results from the experiment.

A. Experiment design

B. Measurements

C. Pilot test

D. Analysis

VII. Future work

VIII. Conclusion