VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Bui Van Thanh

Proactive Maintenance Solution for Industrial Robots

Technology and Communication
2017

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Bui Van Thanh
Title	Proactive Maintenance Solution for Industrial Robots
Year	2018
Language	English
Pages	54
Name of Supervisor	Timo Kankaanpää

This thesis was done based on the requirements provided by Co-Automation Oy to develop an effective maintenance and monitoring solution for industrial robots, which is able to collect and store data, monitor and display alerts using IoT Ticket.

Industrial production requires robots to operate through hours every day, which will eventually cause damage in the robots' parts and may affect the whole operation of the factory. The monitoring process is very time consuming if it is done on the robots one by one. This thesis proposes a solution which helps to improve in monitoring the performance of robots and also give prediction on the time that the robot may break down using the Internet of Thing system or IoT.

IoT has been developed and applied in many different fields. IoT provides better access and control for the users on each device connected to an IoT system. IoT-Ticket is a platform developed by Wapice Ltd which provides a user-friendly interface for the users to access and interact remotely with their devices.

This thesis covers the requirement specifications, analyzing process, planning, implementing and the testing process of the solution. This document will focus mainly on the analyzing phase and the implementing phase.

Keywords IoT, IoT-Ticket, Proactive maintenance

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
PLC	Programmable Logic Controller
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
API	Application Programming Interface
REST	Representational State Transfer
Nm	Newton-meter
MS	Microsoft
SQL	Structured Query Language
CPU	Central Processing Unit
SSMS	SQL Server Management Studio
UI	User interface
ICT	Information and Communications Technology
ms	millisecond

1 INTRODUCTION

1.1 Co-Automation Oy

Figure 1 Co-Automation

"Co-Automation Oy, founded in 2007, is a company in the electrification and automation industry" (Co-Automation LinkedIn). The headquarter of Co-Automation is located in Vaasa, Finland. There are about 25 employees working at Co-Automation, besides that at some projects, there can also be temporary worker. Co-Automation is a channel partner to ABB.

"Co-Automation is a systems supplier of industrial production and packaging cells and lines built around robotics. Over a short time, we have grown into a significant automation supplier for global companies, such as Nokia Networks, ABB and Wärtsilä." (Co-Automation web page, 2017). Their automation systems are designed for engineering industry, patch production industry and food industry.

They are used in assembly, various treatments for pieces, packaging, testing stacking, inspection and sorting.

Co-Automation has three main fields of focus:

- Production automation
- Preliminary design
- Maintenance and Service for industries

1.1.1 Production automation

Automated production improves the effectiveness of equipment in production. With automation, the production can minimize the non-productive employee-related costs, for example sick leaves, holidays, or accelerate the product throughout time by several days. This can result in boosting production volume, improving delivery reliability or improving the quality of products.

“An efficient automation system performs on the factory floor exactly the way described in the quote. It is designed, tested and installed according to schedule. When a cell or production line is activated, it stacks, assembles, tests and packs the required number of pieces in the time promised. This guarantees that the calculated repayment term is achieved.” (Co-Automation web page, 2017)

1.1.2 Preliminary design

Preliminary design can perform the production of real world in 3D designs, which is remarkably convenient for seeking the highest possible productivity. The design may involve an entire factory-size production or a single production line.

At Co-Automation, they use the most up-to-date programs to carry out designs and production simulation in 3D design, such as SolidWork and RobotStudio. Preliminary design involves creating a virtual environment of the customer’s production, production lines and individual manufacturing processes for products.

1.1.3 Maintenance and Service for industries

Co-Automation also provides service for both automating systems provided by their company and production equipment made by other industrial manufacturers.

Their services include:

- Service and repair of ABB robots
- Maintenance of electrical and automation devices, troubleshooting and repair
- Specification of spare part requirements for electrical and automation devices
- Preparation of a service program for electrical and automation devices
- Control systems reprogramming and program changes
- Filming electrical equipment with a thermographic camera
- Mechanical servicing and maintenance service
- Remote support services

1.2 Objectives

The main goal of the project is to create a solution to collect torque data from an ABB robot and store the data in a database. The saved data will be sent to IoT ticket in order to provide a visualized solution for users to keep tracking on robot operating status.

The solution should be able to read torque data from robot axes, there are 6 axes on ABB IRB 1600 robot. Torque data will be sent to Omron NJ101 series PLC and the PLC sends the torque data to MS SQL Server database.

The client from Co-Automation was informed and given clear understanding of the direction of this solution. In the developing process for the solution, the author and the client had many discussions to keep track of the process and to acknowledge as well as handle unexpected issues.

The solution is only a feature running side by side with main function of the robot's program. Therefore, a torque data message sent from this solution to PLC can be different from the actual program, since data handling is not a part of this solution but the actual working robot program of Co-Automation.

1.3 Scope and limitation

The scope of this thesis is to develop a solution that meets the following requirements:

- Read torque values from six axes of the robot
- Send torque value through socket connection to the PLC
- Add max, min, average value on graphs

These requirements required the knowledge about the built-in features of an ABB robot to handle the collected data as well as the suitable API for sending data to IoT-Ticket platform and how to organize data flow in order to display suitable data on IoT-Ticket web interface.

The Omron PLC was completely managed and programmed by Co-Automation, therefore, the details of how the PLC's working in reading and parsing the data sent from the robot and send to database is not possible to be acknowledged.

At the time of this thesis was process, the alarm feature of IoT-Ticket was confirmed not to be working by a member of IoT-Ticket developing team. Since there was a problem on the platform, an alternative solution had to be implemented on the IoT-Ticket. Moreover, due to the actual torque data was not being possible to provide, the data used in this thesis was made based on some test data and modified to meet the requirements of the test on IoT-Ticket and the calculations were based on the basic elements of the actual operations of the robot.

2 BACKGROUND INFORMATION

2.1 Internet of things (IoT) and IoT-Ticket

2.1.1 IoT

According to Wikipedia, “The Internet of things” (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to connect and exchange data. Each thing is uniquely identifiable through its embedded computing system but is able to inter-operate within the existing Internet infrastructure.”

The term “Internet of Things” was created in 1999 by Kevin Ashton from MIT to describe a system where the Internet is connected to the real world via sensors. A “thing” in the term can be any source of data, machine, everyday device which are smart enough to communicate.

Figure 2 Best practices of Internet of Things (Networkcomputing.com)

IoT solutions are being implemented and used in all fields of daily life, in financing, business, industry, medical purpose, etc. The benefits of IoT are undeniably various, but, in general, they can be grouped in these IoT solutions:

- Create the opportunity to monitor, keep track and predict health and performance of people and devices or machines.
- The ability to manage mobile and fixed asset
- Improvement in data broker and finding value from data

2.1.2 IoT-Ticket

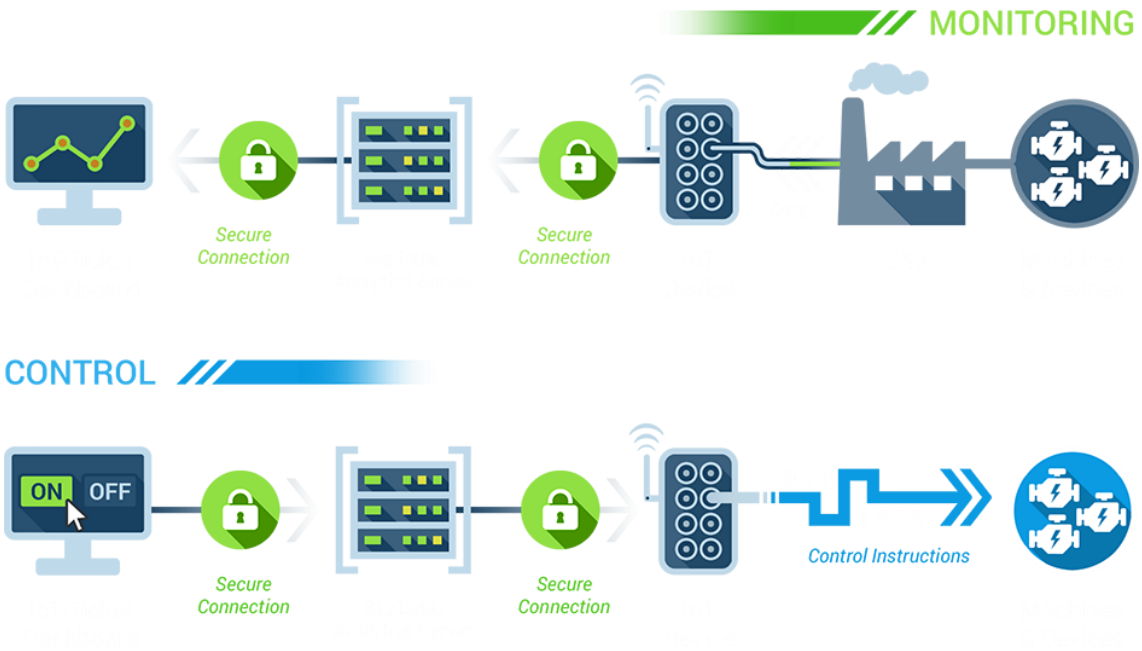

Figure 3 IoT-Ticket platform overview

"IoT-Ticket is a complete Internet of Things (IoT) platform covering data acquisition, reporting, dashboard and analytics. It enables operational efficiency and business model innovation for companies. The platform supports supervisory monitoring, control, automation and advanced reporting functionalities." (IoT-Ticket homepage (2017), iot-ticket.com)

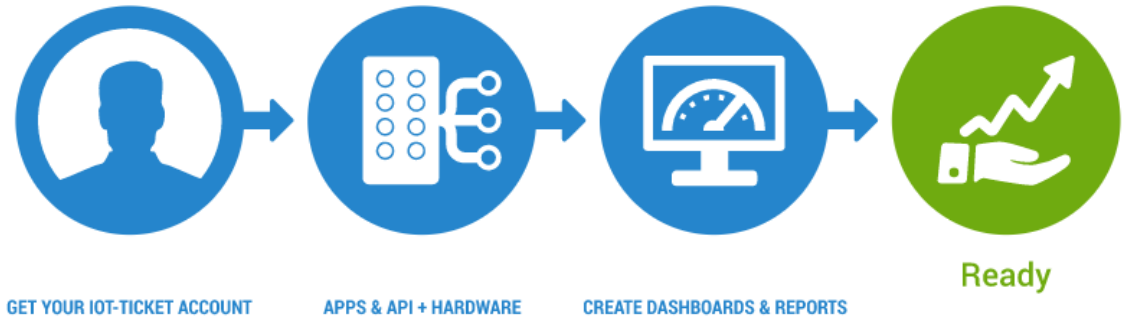

Figure 4 IoT Ticket in 3 steps

IoT-Ticket platform is developed by Wapice Ltd whose headquarters are located in Vaasa, Finland. IoT-Ticket provides Dashboard interface which is a modern, web-based and user friendly for end user. Through dashboards, users can connect, access and interact with remote devices to update their status or get performance reports.

2.2 ABB robot

ABB is well known for operating in robotics, heavy electrical equipment, automation technology and power. ABB is now operating in more than 100 countries and employing about 130 000 people. The company has a history lasts more than 130 years and now is a global leader in industrial technology. The headquarters of ABB are now located in Zürich, Switzerland.

In this thesis, the robot used is the IRB 1600 series. The IRB 1600 series has the qualities shown in the table

Table 1 IRB 1600 versions

Robot versions	Handling capacity	Reach	Protection options
IRB 1600 - 6 / 1.2	6 kg	1.2 m	Foundry Plus 2 (IP 67)

IRB 1600 - 6 / 1.45	6 kg	1.45 m	Foundry Plus 2 (IP 67)
IRB 1600 - 10 / 1.2	10 kg	1.2 m	Foundry Plus 2 (IP 67)
IRB 1600 - 10 / 1.45	10 kg	1.45 m	Foundry Plus 2 (IP 67)

2.3 Omron NJ101 series

Omron Corporation is a manufacturer of control equipment, factory automation systems, electronic components, automotive electronics, ticket vending machines and medical equipment. Omron has its base in Kyoto, Japan.

The model of Omron PLC used in this project is the NJ101-1020. This model was released in December 2015. The Database Connection CPU of the 1020 model is equipped with a function to collect and utilize the information at the production sites. The PLC unit can connect directly with a number of databases including MS SQL Server which does not require any special software or middleware. With this unit, users do not need special knowledge about ICT-related programming to construct a system.

The NJ101-1020 was one of the models released in 2015 which were said to be the leverage Big-Data and IoT at production sites. With this model of NJ101, users can, for example, involve Big-Data and IoT into their manufacturing processes easily on some small scale of productivity enhancement or predictive maintenance.

3 RELEVANT TECHNOLOGY

3.1 Java and Maven

According to Wikipedia, “Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA) meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.” As of January 2018, Java is still the most popular programming language in the world, according from Inc., Tiobe and http://pypl.github.io. Java is well-known for programming Android operating system or applications and client-server applications and more.

Maven is a tool that helps in managing software projects by using the concept of POM – project object model. This tool is used to manage projects’ build, reporting or documentation. Maven helps developers by:

- Improving the building process, make it easier
- Providing a uniform build system
- Providing a quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

3.2 MS SQL Server

Microsoft SQL Server is software product of Microsoft. It is a relational data management system which main function is storing and retrieving data as requested by other software applications. MS SQL Server is using T-SQL which is a proprietary extension to the SQL and SQL is used in interacting with rational databases. T-SQL is basically the same as SQL, but since T-SQL uses SQL standard to include new features and new support functions, there are some differences between those two.

MS SQL Server 2017 is the latest version of MS SQL Server released on 2nd October 2017.

3.3 Socket Connection

According to Oracle, a socket is software endpoint that establishes bidirectional communication between a server program and one or more client programs. In socket communication, a server program always requires an available hardware port on the machine where it runs so any client program in the network can connect and communicate with the server. To send data or requests to server, clients must know the server's opening port, or the data will never reach the destination, and the server must be listening on the running port all the time, waiting to receive any incoming data or request.

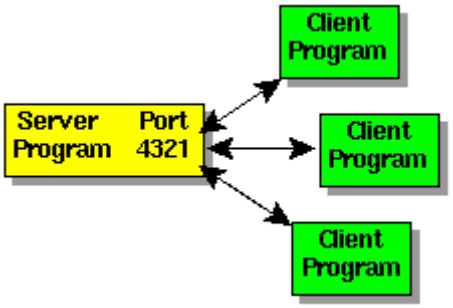

Figure 5 Socket communication illustration

There are two types of Internet protocol (IP) traffic which can be applied in socket communication, they are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is connection oriented – once a connection is established, data can be sent bidirectional, UDP is a simpler, connectionless Internet protocol. Multiple messages are sent as packets in chunks using UDP.

4 TOOL IMPLEMENTATION

4.1 RobotStudio

RobotStudio is a simulation and offline programming software of ABB, which helps programming a robot to be done in a computer. The software allows the development to be done without shutting down the production.

RobotStudio is built on ABB Virtual Controller. This allows developers to have simulations of real robots, actual production in the virtual world using real robot programs and configurations.

In RobotStudio, RAPID is the programming language used to control and program ABB industrial robots. RAPID is a high-level programming language introduced in 1994, whose features include:

- Routine parameters:
    - Procedures - used as a sub-program.
    - Functions - return a value of a specific type and are used as an argument of an instruction.
    - Trap routines - a means of responding to interrupts.
- Arithmetic and logical expressions
- Automatic error handling
- Modular programs
- Multi-tasking

4.2 SQL Server Management Studio (SSMS)

SSMS is a product of Microsoft which is used to manage any SQL infrastructure. SSMS has been widely used to access, administer, configure, manage and develop components of SQL Server, Azure SQL Database, SQL Data Warehouse. The software was first launched with MS SQL 2005. SSMS supports both script editor and graphical tools for users.

SSMS has been supporting the following operating systems: Windows 10/8/8.1/7(SP1), Windows Server 2012/2008 R2. Since November 2016, SSMS can also be used to remotely connect to Linux SQL Server instances.

5 REQUIREMENT ANALYSIS

5.1 Approach

In order to have torque data collected from a robot and monitored on IoT-Ticket web interface, the system was designed as follows:

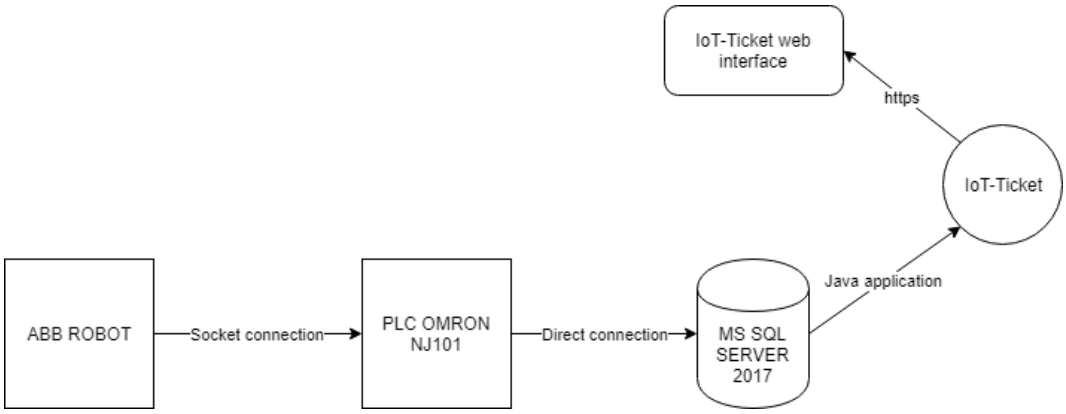

Figure 6 Operating flow chart

In this design, the system can take advantage of socket connection feature of ABB robot to send a data message to PLC. PLC will access directly to MS SQL Server 2017 database to store data. An application using IoT-Ticket API will be developed to update data from database to IoT-Ticket with REST API. IoT-Ticket web interface will be responsible for displaying data as requested.

By following the system design above, table 2 lists out the functions which were considered important to assure that the system will work as expected:

Table 2 List of functions

Reference	**Description**	**Prioritize**
F1	Validate torque value fetched from robot (plan, document, analyse results)	1

F2	Predict torque value in 2 weeks	3
F3	Update robot variables to store torque energy of any number of axes	1
F4	Send data to PLC via socket client	1
F5	Create initial tool for predictive analysis (graph, alarm limits, service need estimation)	2
F6	Change the alarm limit	2

After considering some approaches and discussions, the development of this thesis was arranged as follow: the development was divided into two main sections: the robot side and the software application side. The development of the robot side consisted of functions F1, F3, F4. The development of the software application side consisted of functions F2, F5 and F6. This selection meant that the robot side will take care of the torque data collection and the software application side would be responsible for the data displaying and monitoring. Since on the robot side contained all of the high priority functions, the robot side would be focused on more.

5.2 Robot side

The order of functions running on robot side is F1 ➔ F3 ➔ F4.

The approach on the robot side is to start collecting maximum, minimum and average values of torque data while the robot is operating. The initial frequency of the collection is fifty times/second, which means there are 20 ms between each record. However, it is not wise to keep the collection working all the time, since there are a lot of idle time between tasks when the robot waits for new instructions from PLC. The results collected in the idle period of the robot will critically affect the calculation of torque values to get the desired results. Therefore, the robot

needs to have a signal to initiate the collecting data and whether to start the calculation when the physical tasks start running and stop when the tasks are done. The following flow chart demonstrates process:

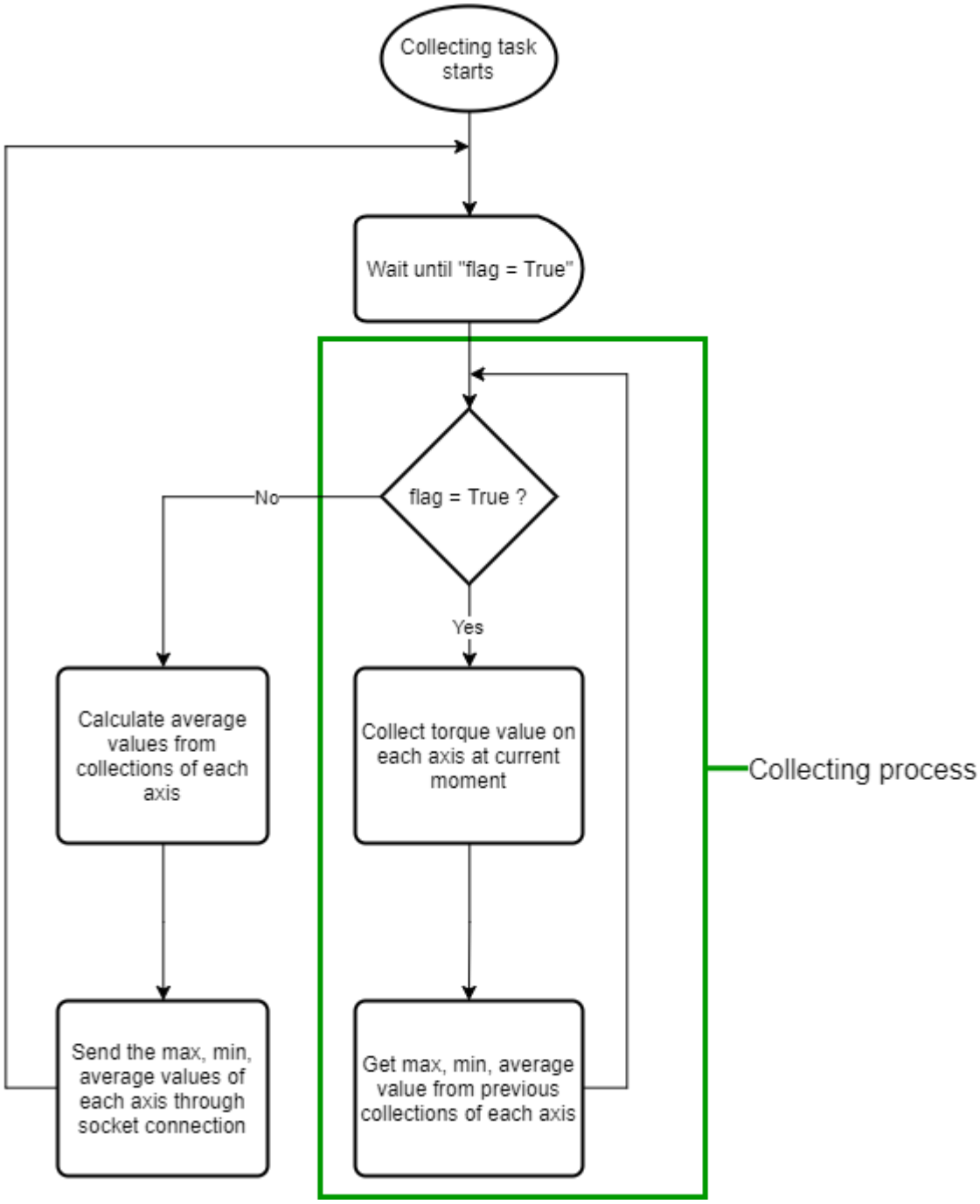

Figure 7 Collecting task diagram

The collecting task is planned to start as soon as the robot starts up. Its purpose to wait for any motion task to begin and then start the data collecting process. The

“flag” is the signal variable which indicates when the collection should begin and stop. The “flag” is globally accessible from all the others task within the entire robot program. This signal variable should be embedded in motion tasks where the robot will perform physical interaction with the surrounding environment, therefore returning the actual torque values. “flag” will be set “False” as default when the robot starts up. The motion tasks start will update “flag” to “True” which also starts the data collecting process as shown above. The collecting process will not stop until the motion task is done and “flag” is changed to “False” again by that event. The period of time when “flag” changes from “False” to “True” is the idle period of the robot.

In the collecting process, the task will also point out the maximum, minimum and average values from the previous collections. When the collecting process has stopped, those collections will be calculated to get the average values of torque. When all the processes are done, the final results will be sent to PLC through socket connection.

In the robot side development, the advantages of having socket connection supported and built-in function in getting torque data is clearly remarkable.

5.3 Software application side

The application plays its role between the database and IoT-Ticket server. Its main purpose it to check and update the data on IoT-Ticket server if the timestamps of data between database and IoT-Ticket do not match. The application was written using Java as the programming language. Figures 9 describes the process of the application:

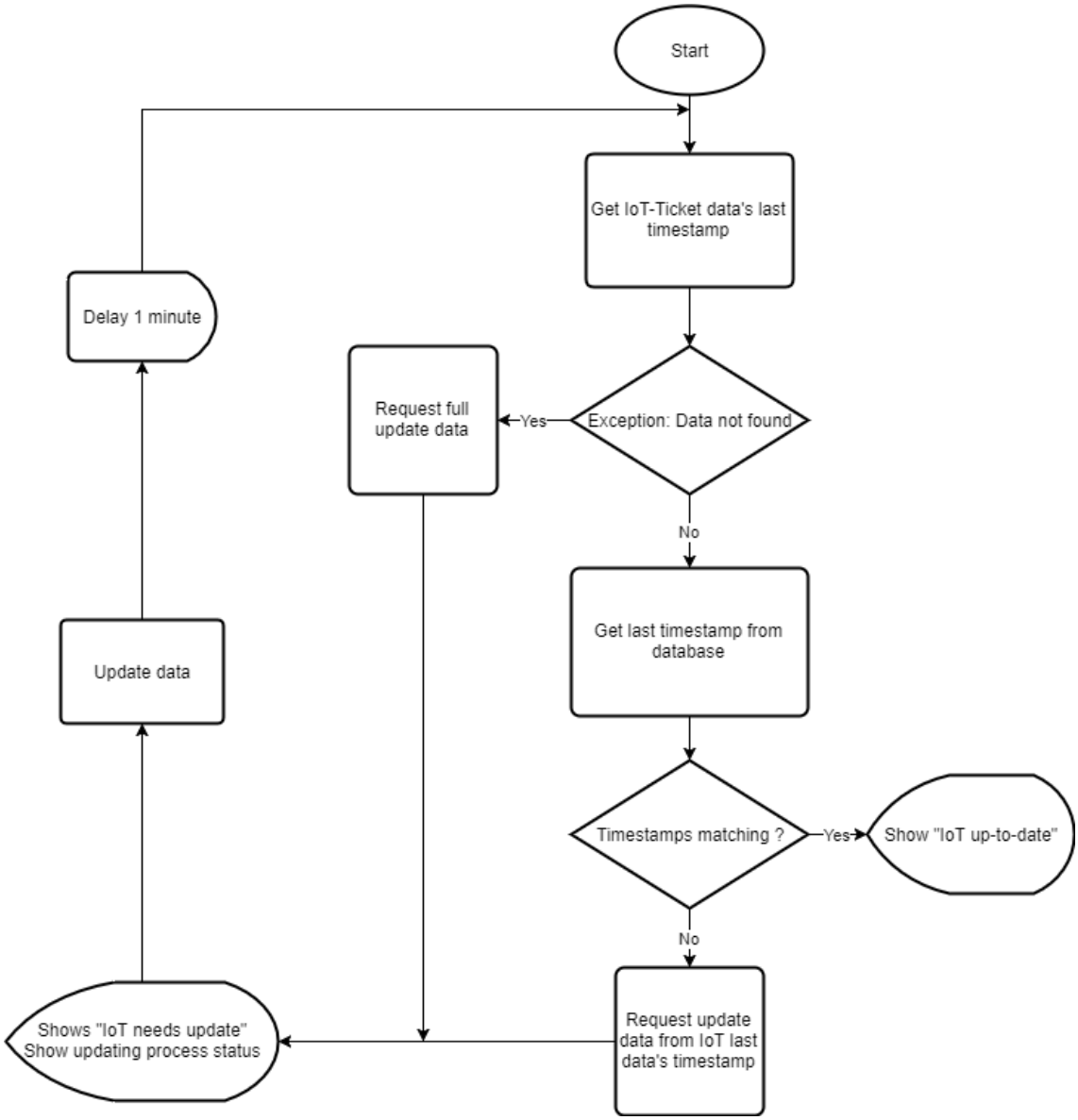

Figure 8 Application diagram

The application is designed to run continuously as its responsibility is to update whenever there is new data in database. IoT-Ticket API developed by IoT-Ticket developing team was recommended for developing custom application. The API provides a secured RESTful connection to Big Data Analytics Server of IoT-Ticket.

5.4 Database

The MS SQL Server should run on a local machine in LAN, so the PLC and the application can access easily. The purpose of the database is to store all the data collected from the robot. The data stored in database will be string and date time type of data. Therefore, it requires only simple configuration.

5.5 IoT-Ticket web interface

In this section, F2, F5, F6 will perform their mission with IoT-Ticket. As F5 is the one with the highest priority, F5 needs to be done before F6 can be implemented, and F2 is the lowest, this part of analysis will proceed as order F5, F6 and F2.

IoT-Ticket supports web interface where users can design and organize their own dashboards. IoT-Ticket will display all data including the collected data from the robot, alert limit, alert notification and it will predict the future torque values of each axis.

As this is the last phase where data is displayed on screen, data must be available on the server of IoT-Ticket before any further action can be done and this should be done by the application. To display data on IoT-Ticket, a dashboard needs to be created at the beginning of the implementation.

Initially, to create a tool for predictive analysis, charts are required. As there are six axes and six sets of data for each axis, there should be six charts showing on IoT-Ticket web interface and each chart contains three series of data: “Max”, “Min” and “Avg” for average. Line charts seemed to be the best for the requirement since it lets the users to see the change through time of the series of data. Chart feature of IoT-Ticket supports alarm function. If there is a value that crosses the limit, an alarm will flash on the screen of the user or, in this thesis, an alternative solution was to display an alert text which contains the time when the alarm function was triggered, or the value crossed the alarm limit. This is where the F5 was analyzed for a solution.

For F6, which is to make the alarm adjustable to be done, F5 is absolutely required to be done before, since the alarm function exists inside chart feature. To create an alarm limit on a chart in IoT-Ticket, a value needs to be assigned to the chart, from there the chart will know where to display the limit.

For the prediction of future values, which is function F2, IoT-Ticket will do the calculation following a pre-set guide on its dataflow editor. The results will be calculated based on the general change in the value of the latest working week until the current day. To calculate the change in value of torque, the following formula would give the result:

$$\Delta torque = \frac{torque_{current\ day} - torque_{1\ week\ ago}}{number\ of\ performed\ tasks}$$

This is assuming that the performed tasks were identical. Δtorque is the general change in torque value of an ABB robot over the previous working week. From there, the value of torque in the next two weeks will be calculated as:

$$torque_{future} = \Delta torque * number\ of\ tasks\ in\ two\ weeks$$

Assuming the number of tasks in every week remain the same. These two formulas basically describe the goal of F2.

6 IMPLEMENTATION

6.1 Robot program

To implement the collecting task into actual working robot, one of the working robot programs being used at Co-Automation was used to implement the task.

To collect the data while the robot is working, the collecting task was required to run simultaneously with other motion tasks. To do that, the multitasking feature of ABB robots was utilized to be able to operate more than two tasks at the same time.

The task was named “T_TorqueCollect” in order to match the name model in the program. Figure 9 show the configuration of *T_TorqueCollect*

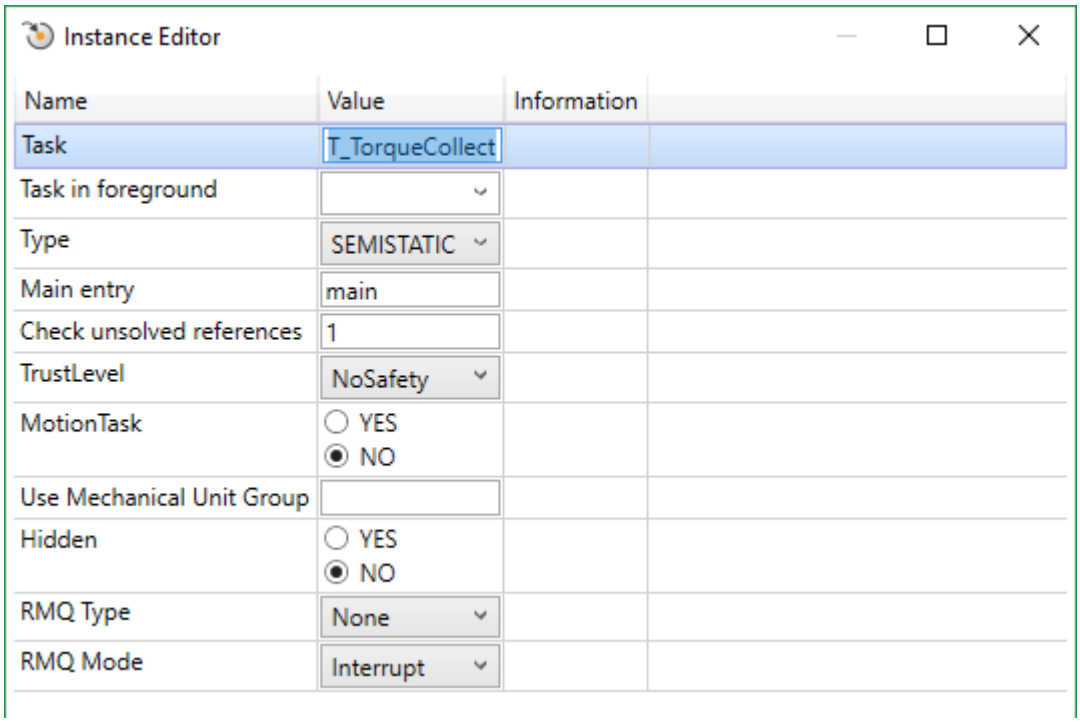

Figure 9 T_TorqueCollect configuration

The type of the task is set “SEMISTATIC” to ensure that the task will be initiated when the robot is starting up. The “Main entry” field indicates the main function where the task should start reading for the instruction, in this case, the default is “main” function. Since this task is a background task and its purpose is only to collect and send data, therefore, there should be no interruption to the whole program except a warning on the controller in case of an error showing up. Therefore, “TrustLevel” field is set to “NoSafety”. “Motion Task” selected “No” option, which indicates this task is not going to perform any physical movement or interaction to the actual working environment. All other fields are left as default.

The RAPID program had the global variable “flag”, which can be accessed by other functional tasks. The “flag” is a Boolean variable which lets *T_TorqueCollect* when to start the collecting process and when to stop. The collecting process has a *while* loop, which will run non-stop until “flag” is turned back to “FALSE” value. There is another *for* loop run six times in each cycle of the *while* loop to collect torque data from each of the six axes and to retrieve maximum and minimum values. In each cycle of the collecting process, there is a delay of 20 ms which prevents the possibility of the collecting process overload the CPU of the robot. The collecting process takes advantage of the built-in function *GetMotorTorque()* of RAPID language. The function is used to read the current torque on the desired axis of the robot. By inputting between the parenthesis a number value from 1 to 6 as a parameter of *GetMotorTorque()* function, the value returned is the torque of the desired axis with unit “Nm”.

After the collecting process, there is the calculation for average value of each axis. Once all the values are calculated, all of them will be parsed and merged into one single string. The PLC receives only string values and there should be only one message sent at a time and therefore the value parsing is necessary.

The message sent from the robot is a string type message, which has a maximum of 80 characters in a single string message. In order to include all eighteen values in single string, it requires some parses of values from numeric type to string type.

Torque values will be initially multiplied by 100 in order to maintain the accuracy after the decimal separator and then parsed into string by using function *NumToStr()* with zero digits after the decimal separator to keep each string value at maximum four characters. The robot program also checks for those string values that have under four characters. If a string value found has less than four characters, digit "0" as a string will be automatically added into the string value. This method ensures that all eighteen string values of torque values from six axes will always be at four-character state that it will be easy for the process of receiving messages and storing them into database. After the eighteen values have been parsed into strings, eighteen string values will be merged into one single string, which has 72 characters occupied in total. The last eight characters will be used for the motor ID and the robot ID of the robot, four characters are for the motor ID and the last four characters are for the robot ID.

This rule helps reduce the chance that the message string exceeds 80-character limit of RAPID programming language. This is just an alternative solution while the actual method of sending message through socket connection at Co-Automation can handle this case. Due to the limited time and the availability of the robot, the implementation of the messaging method takes time and therefore the alternative solution was put into use. The PLC was programmed to handle the string message and send all value strings to database.

The code below is the program for the collecting process on ABB robot.

```
MODULE Collect
    VAR socketdev socket;
    CONST num axis_no := 6;
    Const num arraysize:=10;
    VAR num array{axis_no,arraysize};
    VAR num maxVal{6};
    VAR num minVal{6};
    VAR num avgVal{6};
    VAR num sum{6};
    VAR num i:=1;
    VAR num count:=0;
```

```
VAR num torque;
VAR num length;

VAR string robotID := "1600";
VAR string motorID := "moto";

PERS bool flag;
PERS string packet{18};

!3 values per axis = 18 values in total!
PERS string packetStr:="";



PROC main()
    VAR string val;
    waituntil flag=TRUE;
    packetStr:="";
    count:=0;
    FOR i FROM 1 TO 6 DO
        maxVal{i}:=0;
        minVal{i}:=999;
        sum{i}:=0;
    ENDFOR
    !!!!!Collecting data!!!!!
    WHILE flag DO
        FOR no FROM 1 TO axis_no DO
            torque:=Abs(GetMotorTorque(no));
            sum{no}:=sum{no}+torque;
            IF torque>maxVal{no} THEN
                maxVal{no}:=torque;
            ENDIF
            IF torque<minVal{no} THEN
                minVal{no}:=torque;
            ENDIF
        ENDFOR
        i:=i+1;
        IF i>arraysize THEN
            i:=1;
        ENDIF
        count:=count+1;
        waittime 0.02;
    ENDWHILE
    !!!!!End collecting data!!!!!
```

```
        !!!!!Calculating Max,Min,Avg values!!!!!
        FOR no FROM 1 TO axis_no DO
            avgVal{no}:=sum{no}/count;

            packet{no*3}:=NumtoStr(avgVal{no}*100,0);
            packet{no*3-1}:=NumtoStr(minVal{no}*100,0);
            packet{no*3-2}:=NumtoStr(maxVal{no}*100,0);
            !=======================================!
        ENDFOR
        !!!!!End of calculation!!!!!

        FOR i FROM 1 TO 18 DO
            IF Strlen(packet{i}) < 4 THEN
                FOR j FROM 1 TO 4-Strlen(packet{i}) DO
                    packet{i} := "0" + packet{i};
                ENDFOR
            ENDIF
        ENDFOR

        FOR i FROM 1 TO 18 DO
          packetStr:=packetStr+packet{i};
        ENDFOR

        packetStr:=packetStr+motorID+robotID;

        !!!!!Sending value!!!!!
        SocketCreate socket;
        SocketConnect socket, "192.168.1.212",3000;
        SocketSend socket \Str:=packetStr;
        !!!!!End send!!!!!

        SocketClose socket;

    ENDPROC
ENDMODULE
```

6.2 Software application

To have an application which can handle the connection between MS SQL Server database and IoT-Ticket, a Java application was written to take care of that. By using Maven, the work to manage dependencies was much easier.

In the application, there are three dependencies used:

- IoT-Ticket API: for handling and interacting with IoT-Ticket server
- SqlJBDC4: for establishing connection with MS SQL Server 2017
- Persistence API: for the management for persistence and object/relational mapping from database into Java application.

At the beginning, the application was built based on Spring framework. However, since there were not many functions needed in this application, a basic Java application was an appropriate solution for reducing the building time, the size of the final executable file and the time to initiate the application. After every 1 minute, the application will check the status of data and display whether the date timestamps are matching or need to be updated.

Figure 10 shows the relation between classes in the Java software application.

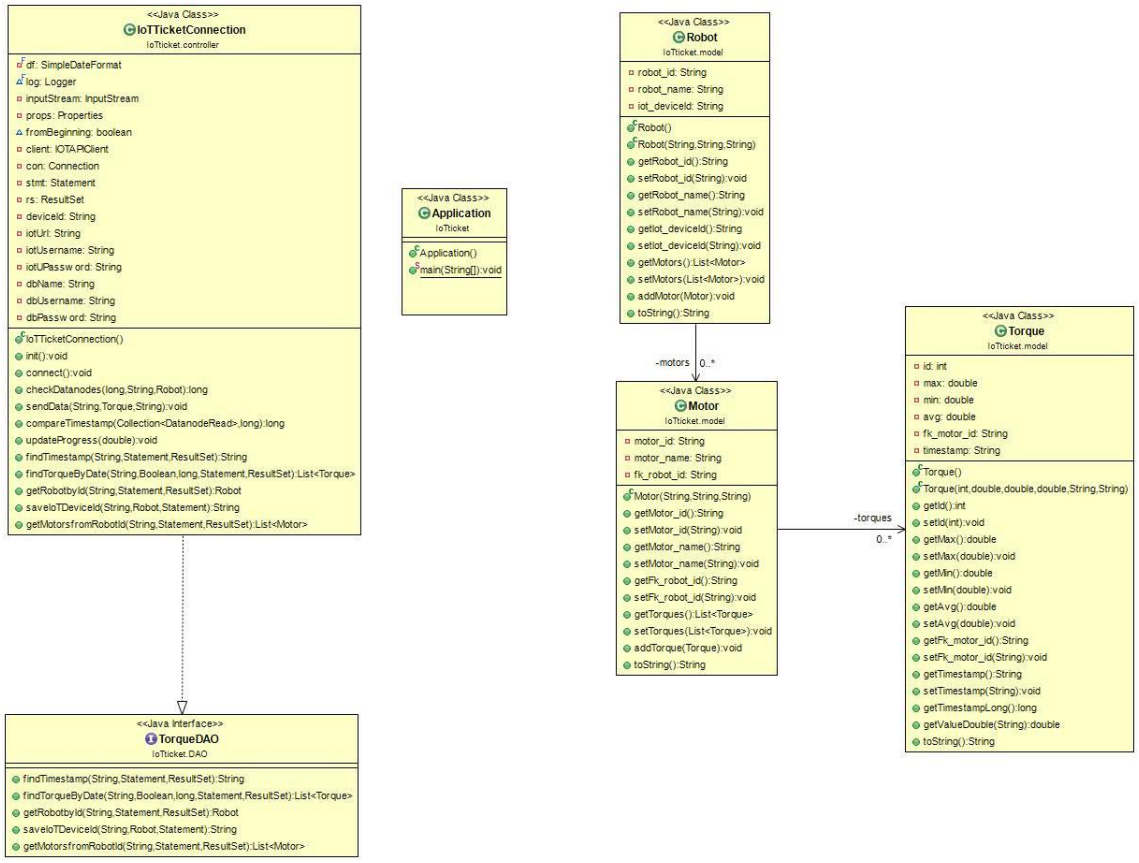

Figure 10 Class diagram

The application will initially prompt for the robot ID before continuing but if the ID is not found in the database, the application will keep asking for a valid ID. Next, the application will update the IoT device ID, if it does not exist in database, and the structure of the robot with the motor ID is acquired from the database. The application will keep updating data on the designated robot with the time interval of one minute. In the process, the application will check for the match between the latest timestamp of data on IoT-Ticker server and in the database of the designated robot. If there is not a match, the application will start the updating process and display the process, otherwise the application will wait for another minute before re-checking data. If the application decides to update new data to IoT-Ticket serv-

er, *sendData()* function will handle the update by sending three different types of value: maximum, minimum and average with names “Max”, “Min” and “Avg” with the number of motor and the end of the data tag’s name. The number of motor is the last character of the motor ID. Figure 11 describes the process of the application.

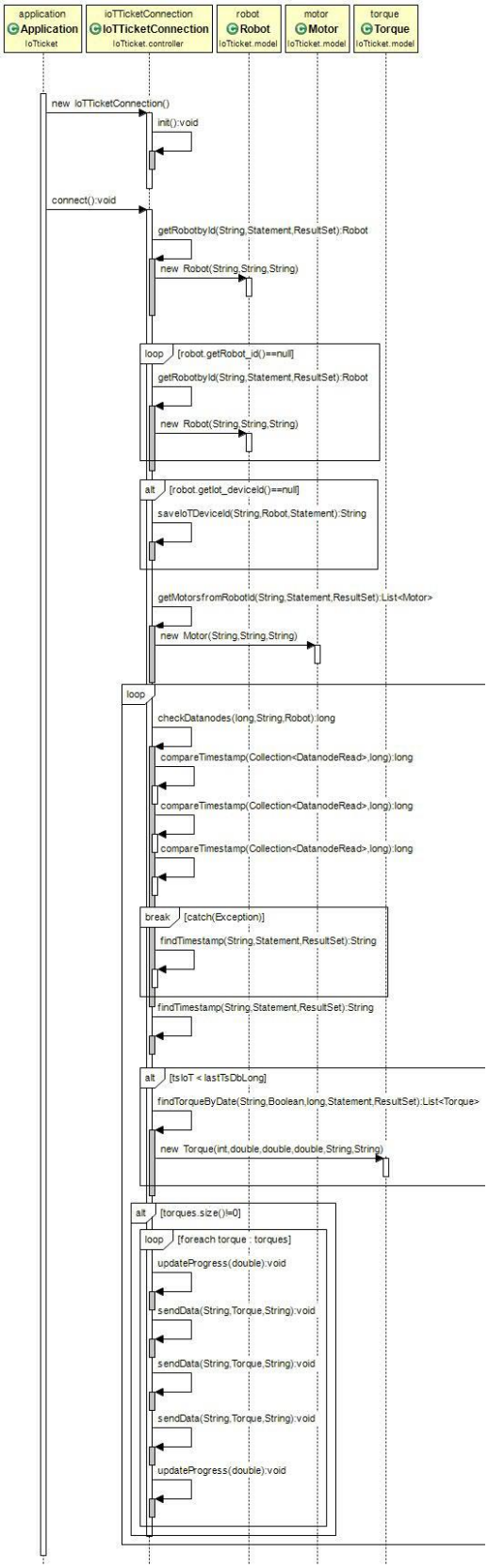
application
Application
IoTicket
ioTTicketConnection
IoTTicketConnection
IoTicket.controller
robot
Robot
IoTicket.model
motor
Motor
IoTicket.model
torque
Torque
IoTicket.model
new IoTTicketConnection()
init():void
connect():void
getRobotbyId(String,Statement,ResultSet):Robot
new Robot(String,String,String)
loop
[robot.getRobot_id()==null]
getRobotbyId(String,Statement,ResultSet):Robot
new Robot(String,String,String)
alt
[robot.getIot_deviceId()==null]
saveIoTDeviceId(String,Robot,Statement):String
getMotorsfromRobotId(String,Statement,ResultSet):List<Motor>
new Motor(String,String,String)
loop
checkDatanodes(long,String,Robot):long
compareTimestamp(Collection<DatanodeRead>,long):long
compareTimestamp(Collection<DatanodeRead>,long):long
compareTimestamp(Collection<DatanodeRead>,long):long
break
[catch(Exception)]
findTimestamp(String,Statement,ResultSet):String
findTimestamp(String,Statement,ResultSet):String
alt
[tsIoT < lastTsDbLong]
findTorqueByDate(String,Boolean,long,Statement,ResultSet):List<Torque>
new Torque(int,double,double,double,String,String)
alt
[torques.size()!=0]
loop
[foreach torque : torques]
updateProgress(double):void
sendData(String,Torque,String):void
sendData(String,Torque,String):void
sendData(String,Torque,String):void
updateProgress(double):void

Figure 11 Sequence diagram

6.3 Database

To serve the purpose of storing data, a new database called “Torque” was created in SQL Server database by using SSMS. A new schema was created the Torque database called “torque_value”, which contains the structure of the data table. The structure of the database is shown in Figure 12:

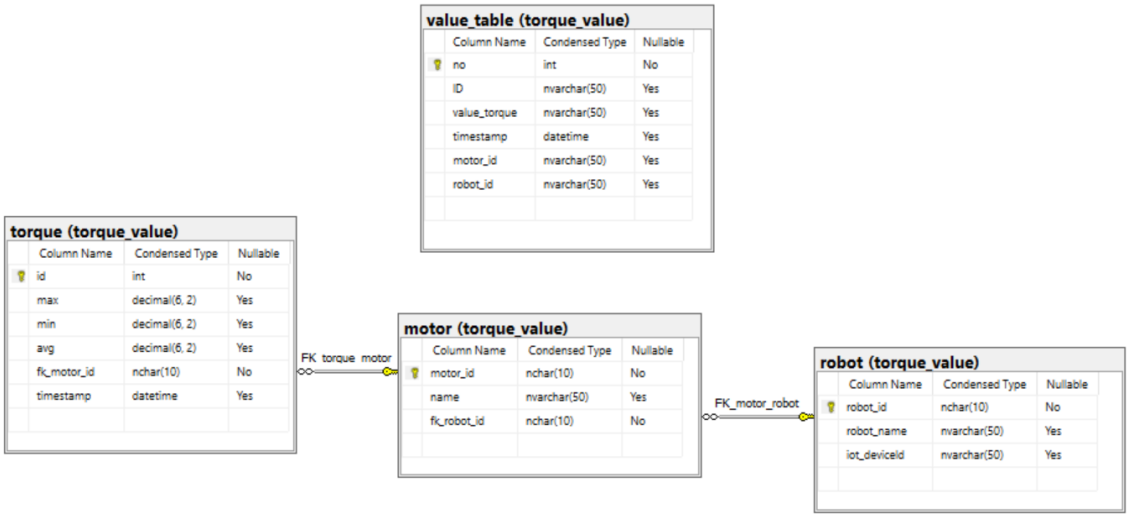

Figure 12 Structure of database

The database consists of four tables: “values_table”, “robot”, “motor”, “torque”. The “value_table” was initially designed based on the customer’s requirements, which is only used to save the data sent from the PLC. However, the structure of “value_table” is not ideal for managing and accessing data. Therefore, a trigger was created to call for a stored procedure every time there is a new row of data saved into “value_table”.

The stored procedure will take the data from each column but ignore the “no” column from “value_table” table, which is only used as an identifier for each row of data. The data from “motor_id” and “robot_id” will be inserted into the same

name columns in “robot” table and “motor” table. In “value_table” table, many different rows of data can have the same “motor_id” or “robot_id”, and since the value for “motor_id” column in “motor” table and “robot_id” column in “robot” table must be unique, therefore, the stored procedure will distinctly select the values of “motor_id” and “robot_id” from “value_table”.

In “robot” table, “robot_name” column and “iot_deviceId” column can be null. These two fields can be filled later, especially the “iot_deviceId” can be filled while running the software application for this thesis.

The column “name” in “motor” table is also nullable. However, “fk_robot_id” is not possible the be left null. The purpose of this column is to identify the relationship between “motor” table and “robot” table, “robot” has “one-to-many” relationship with “motor” since one robot can have many motors. The value inserted into “fk_robot_id” does not have to be unique but it has to exist in “robot” table at “robot_id” column, this is the constraint for the foreign key.

To insert data to “torque” table, the stored procedure will be based on the value pattern of “ID” column from “value_table”. The “ID” column has the value from 1 to 18 as a string value. The reason to have eighteen values for the “ID” column is because the data sent from the robot to the PLC contains eighteen values which are maximum, minimum and average of each motor on each axis as there are six axes so altogether eighteen values in total. The data the PLC received is in the following order:

Max_1, Min_1, Avg_1, Max_2, Min_2, Avg_2, …, Max_6, Min_6, Avg_6

(the numbers indicate the numbers of axes on ABB robot)

From this order, the data can be translated to: ID “1” “2” and “3” means maximum, minimum and average value of motor 1, ID “4” “5” and “6” means maximum, minimum and average value of motor 2 and so on. The third column “value” of data table also takes strings as values. The reason is that the working robot

program and PLC are configured and programed to send and receive data under string format. From this order, the torque data will be inserted one-by-one into “max”, “min” and “avg” column in “torque” table with the same ID of the motor that these torque values belong to. The motor ID will be shown in “fk_motor_id” column where it can identify the relationship between “motor” and “torque” is also “one-to-many” since one motor can have several records. While inserting value from “value_table” to “torque”, the stored procedure also return the value of torque from string to numerical form by parsing the string value to numerical value and divide the number by 100, with 2 digits of fraction.

The last column “timestamp” takes the time when the data is taken into database which is also inserted from “value_table”. The data of this column is datetime, which is as “YYYY-MM-dd hh:mm:ss.[.n*]”, following the configured time zone in the PLC. The format of datetime type in SQL Server has the following element ranges:

- YYYY is four digits from 1753 through 9999 that represent a year.
- MM is two digits, ranging from 01 to 12, that represent a month in the specified year.
- DD is two digits, ranging from 01 to 31 depending on the month, that represent a day of the specified month.
- hh is two digits, ranging from 00 to 23, that represent the hour.
- mm is two digits, ranging from 00 to 59, that represent the minute.
- ss is two digits, ranging from 00 to 59, that represent the second.
- n* is zero to three digits, ranging from 0 to 999, that represent the fractional seconds.

The “ID” column of “torque” is just an identifier for each row of data, it is an incremental column, which increase by 1 every time there is a new row of data inserted.

In order to create a trigger called “stored_proc” on “value_table” which execute a stored procedure to other tables, the code below was used to trigger on every insert event of “value_table”:

```
USE [Torque]
GO
/****** Object:  Trigger [torque_value].[stored_proc]
Script Date: 02-Mar-18 8:24:48 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:		<Author,,Name>
-- Create date: <Create Date,,>
-- Description:	<Description,,>
-- =============================================
CREATE TRIGGER [torque_value].[stored_proc]
   ON  [torque_value].[value_table]
   AFTER insert
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

    -- Insert statements for trigger here
	insert into torque_value.robot(robot_id) select distinct
robot_id from torque_value.value_table
	where
		not exists(select robot_id from torque_value.robot
robot
		where

	torque_value.value_table.robot_id=robot.robot_id)
	insert into torque_value.motor(motor_id,fk_robot_id) se-
lect distinct motor_id,robot_id from torque_value.value_table
	where
		not exists (select motor_id, robot_id from
torque_value.motor motor
		where

	torque_value.value_table.motor_id=motor.motor_id and

	torque_value.value_table.robot_id=motor.fk_robot_id
```

```
            )

      declare @t table(
            id nvarchar(50),
            value_torque nvarchar(50),
            motor_id nvarchar(50),
            robot_id nvarchar(50),
            timestamp datetime
      )
      insert into @t
      select vt.id,vt.value_torque,
vt.motor_id,vt.robot_id,vt.timestamp from
torque_value.value_table vt
      inner join (
            select max(value_torque) as val,motor_id,
max(timestamp) as latest from torque_value.value_table group
by motor_id
      ) tm on vt.motor_id=tm.motor_id and
vt.timestamp=tm.latest

      insert into
torque_value.torque(max,min,avg,fk_motor_id,timestamp)
      select con-
vert(decimal(6,2),t1.max)/100,convert(decimal(6,2),t2.min)/100
,convert(decimal(6,2),t3.avg)/100,t1.motor_id, t1.timestamp
from
            (select motor_id,robot_id,timestamp,value_torque as
max from @t where id in ('1','4','7','10','13','16')) t1
            left join
            (select motor_id,robot_id,timestamp,value_torque as
min from @t t2 where id in ('2','5','8','11','14','17')) t2
            on t1.motor_id=t2.motor_id and
t1.robot_id=t2.robot_id and t1.timestamp=t2.timestamp
            left join
            (select motor_id,robot_id,timestamp,value_torque as
avg from @t t3 where id in ('3','6','9','12','15','18')) t3
            on t2.motor_id=t3.motor_id and
t2.robot_id=t3.robot_id and t2.timestamp=t3.timestamp
            order by motor_id
END
```

6.4 IoT-Ticket web interface

The first thing to do on IoT-Ticket before connecting a device to the server is to create a new device slot on IoT-Ticket server. The list of devices can be found on IoT-Ticket EnterpriseManager page. The most important detail to notice was the

device ID when creating new device, since that is the only identifier needed for future interaction between the local devices and the IoT-Ticket server. The new created device was named “ABB Robot Monitor” which contains the information shown in Figure 13:

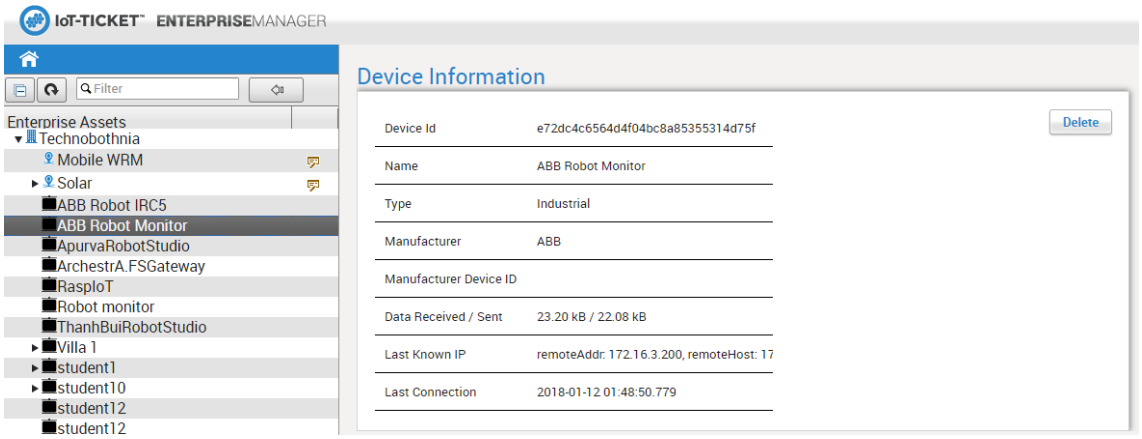

Figure 13 IoT-Ticket device information

After the device was successfully created, the Java application was initiated to start the update process to send data from local database to IoT-Ticket server, the time to complete the process depends a lot on the amount of data to be sent from local database, the Java application has an indicator where the status of the updating process is shown, and users can see it. After the update was done, there was a list called “Datanodes” where all the names of items are shown with their latest values. The “Datanodes” list has a limit of 20 items maximum, which is still more than 18 items total for 6 axes. This list provides all the data needed for the implementation on IoT-Ticket web interface.

Before creating a user-friendly web interface for monitoring, the device needed a dashboard to display data by clicking “Create Dashboard” and input the name of the dashboard to be created.

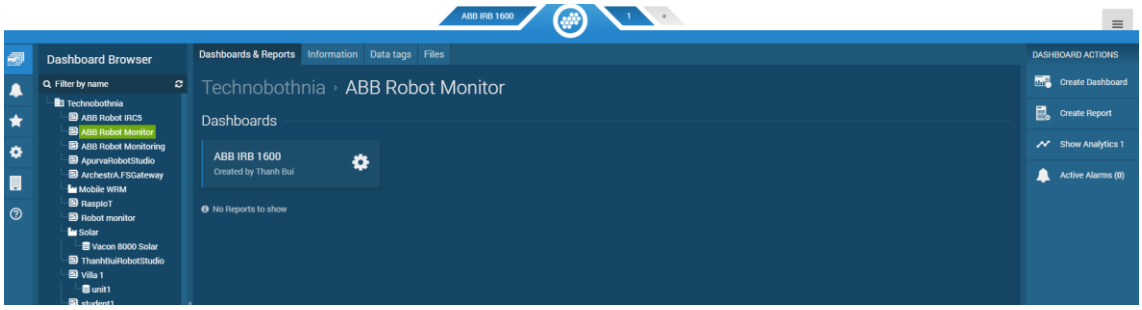

Figure 14 Dashboard browser

In this thesis, a dashboard for the IRB 1600 robot was created called “ABB IRB 1600” as shown in figure 14. The “Dashboard Browser” also shows data tags, whose values was retrieved from datanodes list to use in the user-interface implementation, along with the values and latest timestamp when the values were updated.

In order to create an interface suitable for users to keep track on devices, the dashboard needed to be designed and edited by going to edit page via the control panel of IoT-Ticket by choosing “Edit” as shown is Figure 15.

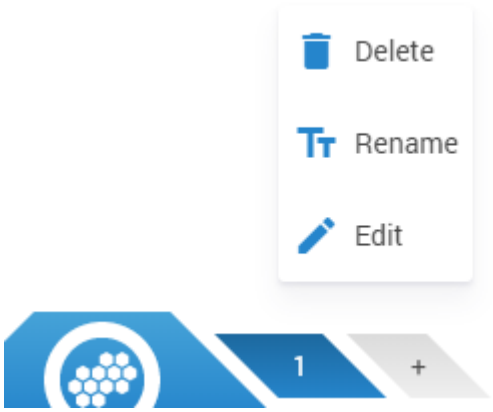

Figure 15 Accessing edit page on IoT-Ticket

The “Interface designer” was a white blank, but it provided a UI elements panel which contains many different items for different purposes.

Figure 16 UI elements panel

In this dashboard page, to achieve the goals of this thesis, the following UI items were used to create a functional page:

- Chart element: to display the trends of data on each axis, there will be 6 charts
- Label: the title of the page, which is “ABB IRB 1600 MONITOR:
- Input label: to display alert or prediction of future values.

Each chart will have one label for showing name, one input label to display alert, one for the prediction of value in two weeks and one for the estimation of time remaining before alert. All these items were aligned to be in groups, so they can show clearly data of each axis from the IRB 1600.

To assign any data or any function, the “Data-flow editor” provides an interface where users can connect values of data tags to UI elements using mouse by dragging. To simply add data of maximum, minimum, average values and alert limit to a chart, the connection between data tags, chart items and functions can be done as in Figure 17:

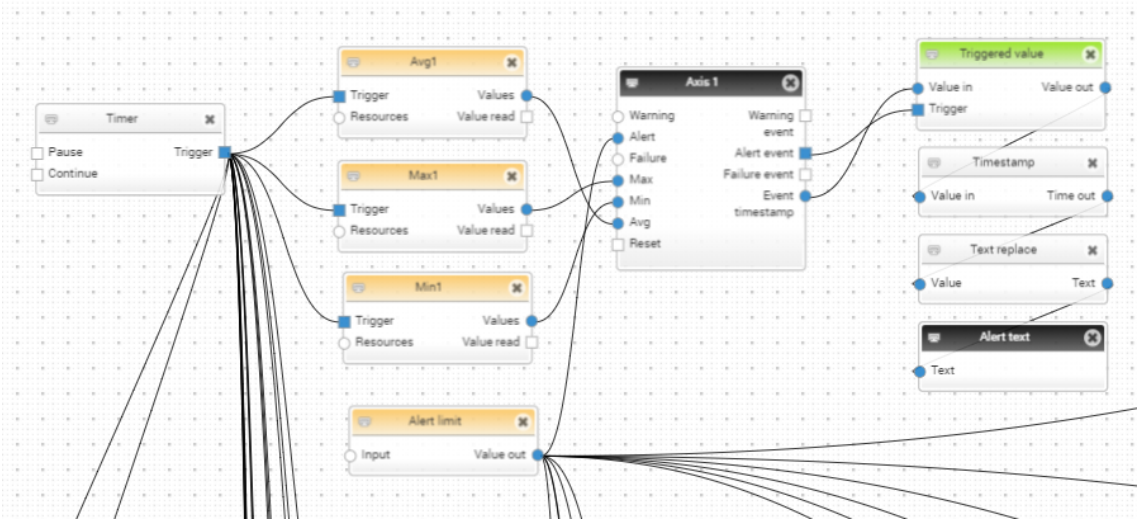

Figure 17 Connect data to chart (Axis 1)

The “Timer” helps updating the values of each data tag, it was set to trigger all the tags connected every 1 second. Each data tag connects to the match series name, “Max”, “Min” or “Avg”, on the chart item, in the Figure 16 is the chart of axis 1. To assign alert limit to the chart, setting value to a constant called “Alert limit”, the name can be changed as wished, and connect to “Alert” node on the chart. Once the value crosses the limit, it will trigger the “Alert event” and display the “Alert text”. Since the value of “Alert limit” can be changed anytime, it makes the limit of alert adjustable.

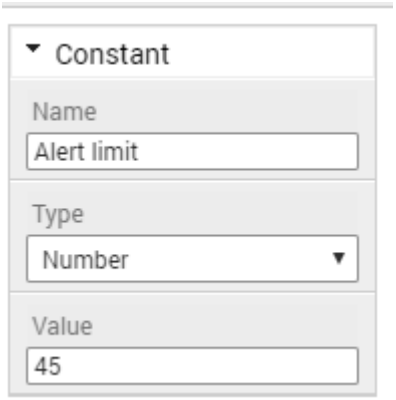

Figure 18 Adjust alert limit by changing "Value" field

To calculate the prediction of values and estimation time before reaching the alert limit of each axis, calculation blocks which have mathematical functions were put into used. Even though the formulas seem simple but putting them into a graphical user-friendly view is not as simple, especially when it requires a lot from the performance of the computer to display all the details in the editor.

Figure 19 Data flows and calculations

In the estimation of the remaining time before reaching alert limit of data, the flow will decide whether to display the number of remaining days or tasks, if the number of tasks is less than the average number of tasks in one day, the dashboard will display the number of remaining tasks below the chart, otherwise it will show the number of days.

After everything had been arranged and put into place, the dashboard looked like in Figure 20 without any data:

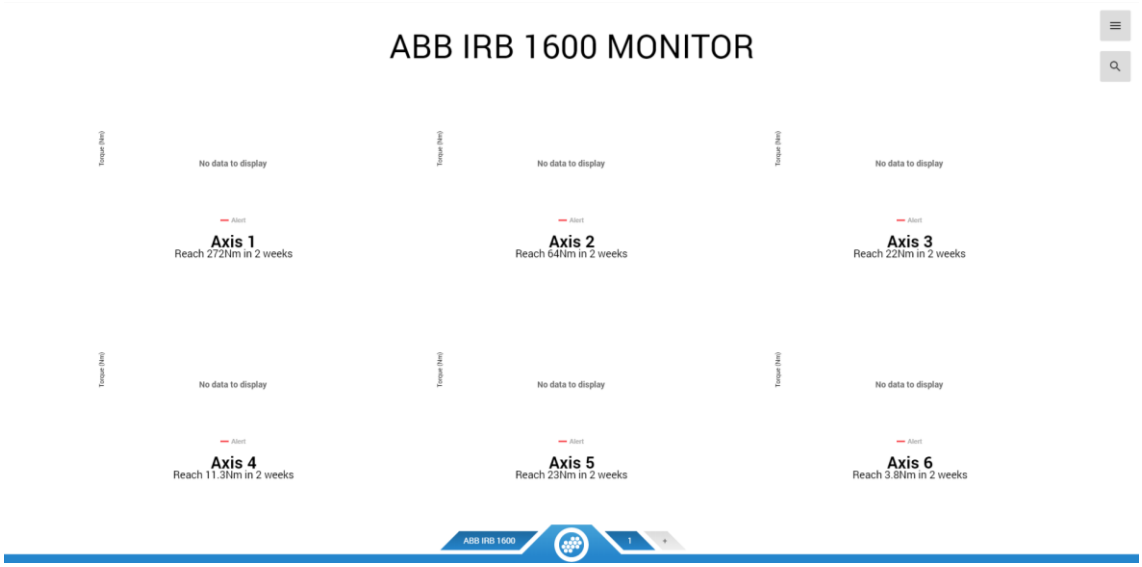

Figure 20 Dashboard of IRB 1600 – without data

7 TESTING AND RESULTS

7.1 Robot test

The robot test included the calculation accuracy test and main test.

The calculation test was to decide if the logic in the robot program was working properly to collect data and calculate accurately. To do that, a test task was implemented in the robot to collect a number of torque values and send those values along with the results. The test task was collecting 10 torque values on each axis while running the test and calculate th8e maximum, minimum and average from those values. After that, the robot will send two messages, one contains a total of 60 values uncalculated, the other contains 18 calculated values. Figure 21 shows the 661th message, which contained 60 torque values collected from the robot.

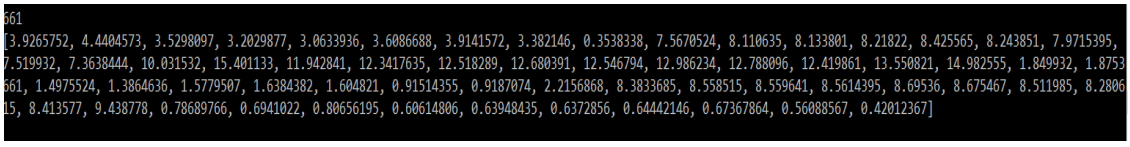

Figure 21 60 collected test torque values

Figure 22 shows the 662nd message which contained 18 calculated values after the collection, the “0.0” was not counted since it was the default value if there are no value sent to the value slot after the 18th.

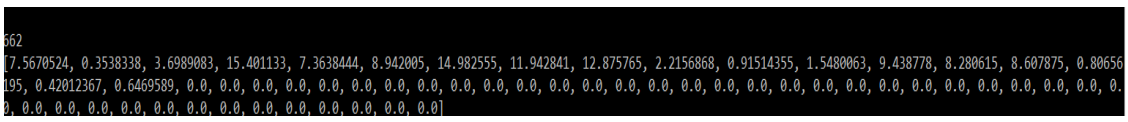

Figure 22 Calculated test values

After receiving the messages, all the values were taken into manual calculation and proved that the 662th message contained the correct values and in the right order as expected.

After the calculation test was the main test. This test was made to test if the socket feature of the ABB robot can handle the collected values after parsed into a string. The robot was programmed to perform identical movements in 50 cycles and collect the torque value on each axis after every cycle. The result should be almost or exactly the same to each other. This test also pointed out if the “flag” variable was working properly as every time the cycle ends, there will be a message sent. To test the result, software SocketTest was used to receive the messages sent from the robot. Figure 23 shows the messages sent throughout the second test on the robot.

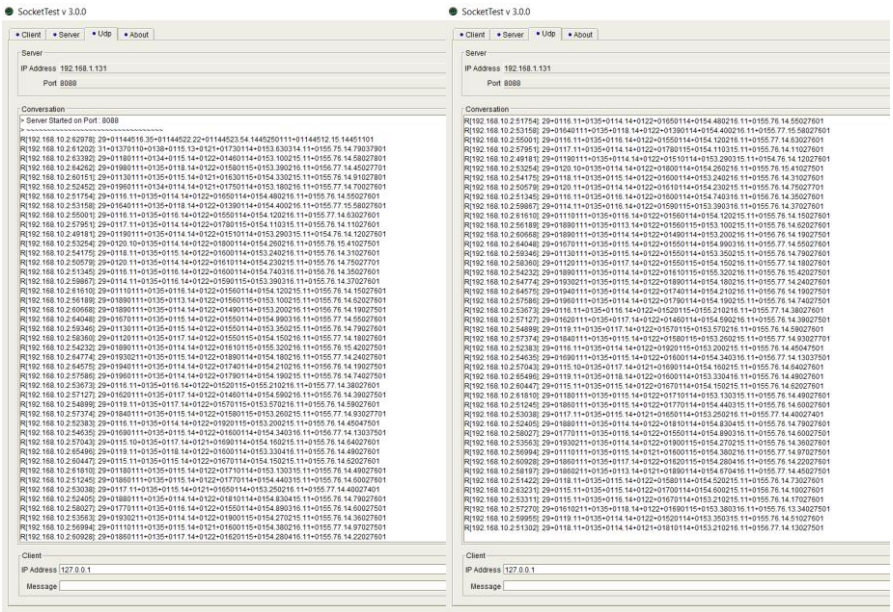

Figure 23 Received test messages

As the robot was programmed to run 50 cycles, there were 50 messages sent and received on SocketTest, which means the “flag” variable work as expected. All the values received were almost identical, which proved that the function *GetMo-*

torTorque was working properly and parsing from numbers to single string was also performing fine that the socket feature was able to send messages.

After the messages were successfully sent, the PLC would take them and parse into separate values then send the new values to SQL Server.

7.2 IoT-Ticket test

The goals of the IoT-Ticket test were to ensure the following things:

- The Java application was working properly with IoT-Ticket API
- The IoT-Ticket web interface displayed correct data and alert if there was any limit crossed.

When the dashboard of IRB 1600 was initially created, it means there was no data of IRB 1600 stored in IoT device. Therefore, no data would be found, and the device would need to be updated with data from the database. The application was initiated and displayed the message shown in Figure 24:

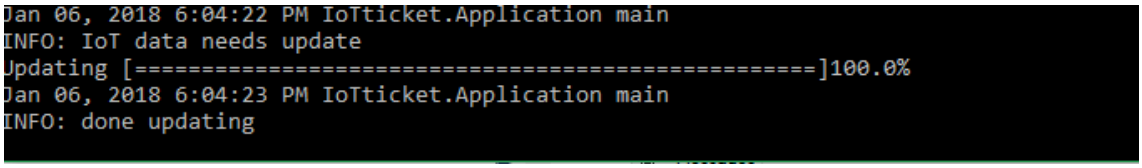

Figure 24 "Updating" message

The application found that "ABB Robot Monitor" was empty and it started the update process with the existing data in the database. After the update, every 1 minute, the console displayed the messages Figure 25:

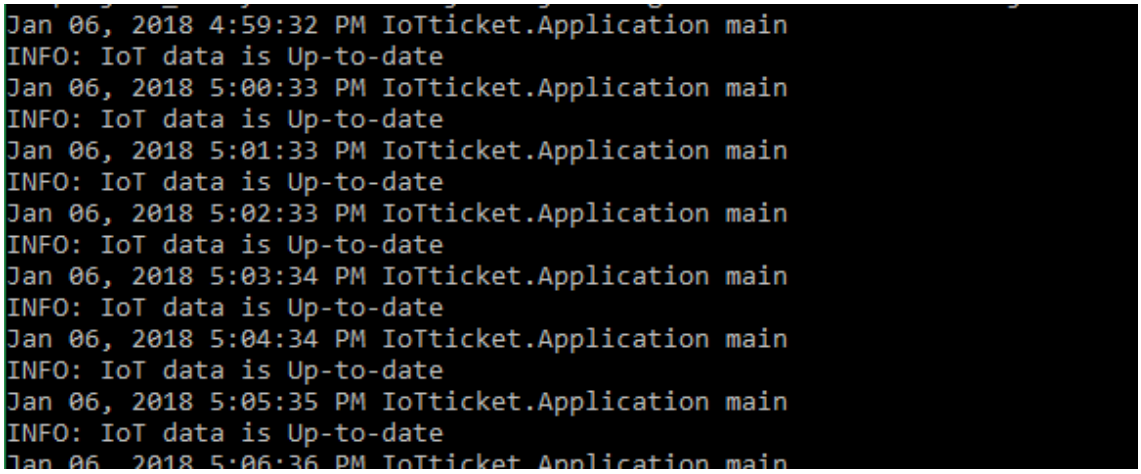

Figure 25 "Up-to-date" message

After the latest set of data was updated, the application still checked for a match between the local machine and IoT-Ticket server. The data at the two places matched each other, as a result, there was no need for any update. As the local machine was sending data to IoT-Ticket server, the datanodes list of IoT-Ticket should be receiving new data with the latest timestamps and values. Figure 26 shows the list of datanodes in IoT-Ticket EnterpriseManager which received the latest values from local machine.

Datanodes

Name	Value	Unit	Time
Avg1	31	Nm	11.01.2018 19:00:00.000
Avg2	18	Nm	11.01.2018 19:00:00.000
Avg3	14	Nm	11.01.2018 19:00:00.000
Avg4	1.5	Nm	11.01.2018 19:00:00.000
Avg5	9.6	Nm	11.01.2018 19:00:00.000
Avg6	0.8	Nm	11.01.2018 19:00:00.000
Max1	110	Nm	11.01.2018 19:00:00.000
Max2	44	Nm	11.01.2018 19:00:00.000
Max3	22	Nm	11.01.2018 19:00:00.000
Max4	7.3	Nm	11.01.2018 19:00:00.000
Max5	15	Nm	11.01.2018 19:00:00.000
Max6	2.2	Nm	11.01.2018 19:00:00.000
Min1	5.2	Nm	11.01.2018 19:00:00.000
Min2	5.5	Nm	11.01.2018 19:00:00.000

Figure 26 Datanodes list

The dashboard should also receive the data and see the data items as data tags, which would be showing on dashboard panel.

Dashboards & Reports | Information | Data tags | Files

Filter by name

Name	Value	Unit	Timestamp
Avg1	31	Nm	2018-01-11 21:00:00.000
Avg2	18	Nm	2018-01-11 21:00:00.000
Avg3	14	Nm	2018-01-11 21:00:00.000
Avg4	1.5	Nm	2018-01-11 21:00:00.000
Avg5	9.6	Nm	2018-01-11 21:00:00.000
Avg6	0.8	Nm	2018-01-11 21:00:00.000
Max1	110	Nm	2018-01-11 21:00:00.000
Max2	44	Nm	2018-01-11 21:00:00.000

Figure 27 Data tag list

IoT-Ticket server received the data. At this point, the Java application was working correctly.

After receiving data, IoT-Ticket dashboard should be showing data sent from database. As values of axis 1 was made to cross the alert limit and some other axes also have increase in their values, the dashboard should be showing alert above the chart of axis 1 and estimation in values on other axis. If there were no changes throughout the time, no estimation would be shown. The dashboard was showing as in the Figure 28.

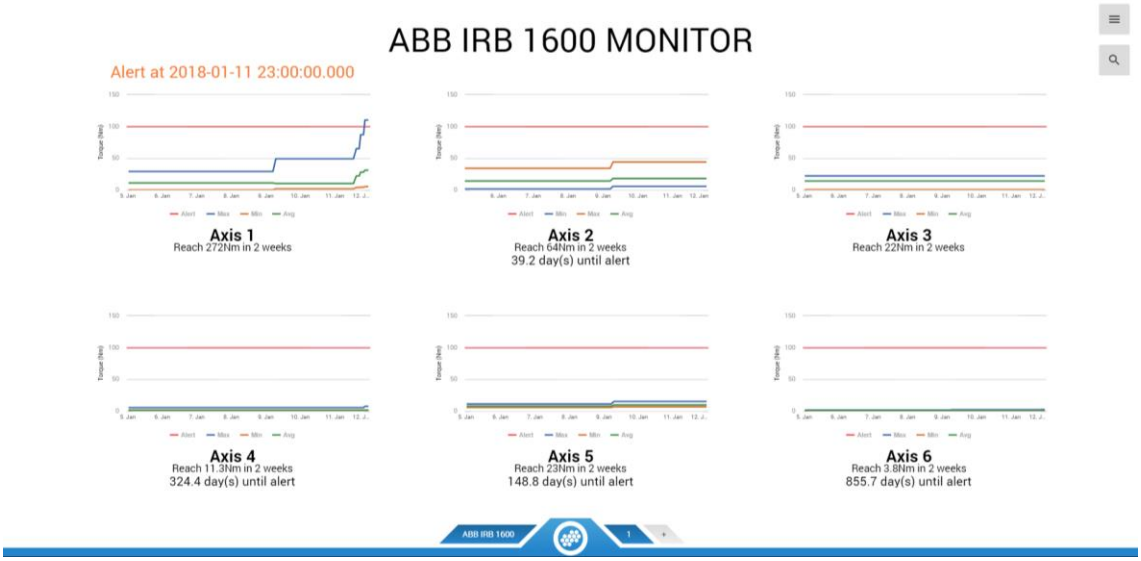

Figure 28 Dashboard of IRB 1600 - with data

The alert event was working fine by displaying the time when the value of axis reached the limit. After comparing between the data on dashboard and database and ensure the calculation of the estimation on IoT-Ticket was correct, the IoT-Ticket was a pass.

8 CONCLUSION

The goal of this thesis was to create a predictive tool for the company Co-Automation to keep track on the performance of their robots and foresee the time to have suitable actions to provide maintenance to their robots as well as prevent the chance of breaking down robot can interrupt the whole production process.

From personal point of view, the analysis phase was done well by being able to design proper working flow charts and decide which technology to use base on the initial requirements from the client. The implementation was done without any major difficulty. The most challenging phase was the testing phase due to the availability of the robot and lack of actual data for the predictive calculation. The thesis was a proof-of-concept project currently supporting only one robot. Future development based on this thesis can improve the system to monitor multiple robots at the same time or a better formula for more precise predictions.

For the conclusion, the thesis has created a possibility for Co-Automation to implement IoT into their flow of work, manufacturing and production to improve their services and open more possibilities of implementing IoT into industries.

REFERENCES

About Co-Automation: from https://www.co-automation.fi

About IoT:

- Wikipedia
- Internet of Things for dummies

Co-Automation services: (2017) Retrieved from Co-Automation Servicing page:

http://www.co-automation.fi/servicing

TCP and UDP comparison (2017): Retrieve from

https://www.diffen.com/difference/TCP_vs_UDP

Best practices of Internet of Things (2015): Retrieve from

https://www.networkcomputing.com/internet-things/7-best-practices-iot-security/1776945892

About Maven: Retrieve from

https://maven.apache.org

Socket communication illustration: Retrieve from

http://www.oracle.com/technetwork/java/socket-140484.html