Robot Data (VDL Nedcar)

IRC5_Data_Collector : Collecting Data from IRC5 robot controllers
IRC5_Data_Viewer : View Data from IRC5 robot controllers

TABLE OF CONTENTS

1. INTRODUCTION 4

1.1. Application 4

1.2. Problem 5

1.3. Sollution Error! Bookmark not defined.

1.4. System requirements 7

1.5. System structure description 9

2. FUNCTIONAL DISCRIPTION – COLLECT_ABB_IRC5_INFO 14

2.1. Background information 14

2.2. Function diagram 16

3. FUNCTIONAL DISCRIPTION – VIEW_ABB_IRC5_INFO 17

3.1. Background information 17

3.2. Function diagram 19

4. BUILD CONTENTS 20

4.1. ABB_IRC5_Collect 20

VERSION HEADER

Version	Date	Status	Remark
1.0	09-05-2019	Draft	-
1.1	07-06-2019	Update	Completion info on Collector and Viewer applications

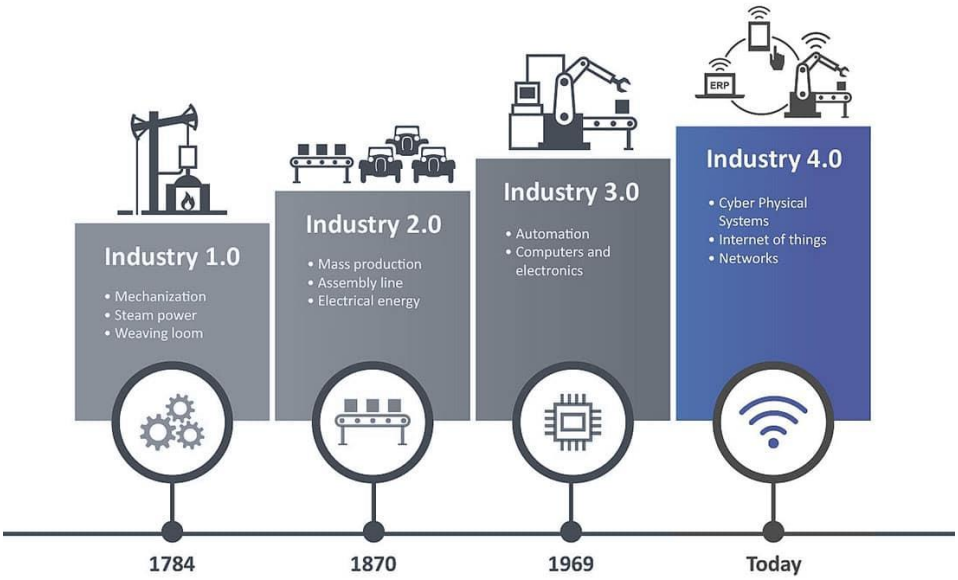
1. Introduction

1.1. Application

A piece of equipment/machinery will, inevitably, break down and stop functioning due to the wear and tear of its sub components/parts.
In order to keep the equipment/machinery running for as long as possible, broken/worn parts need to be repaired or replaced and/or processes need to be reordered.
This practise is called ‘Maintenance’.
Maintenance, as is used by the time of creating this document, comes in 3 distinct levels (in increasing order of usefulness);
-Corrective Maintenance;
Only when a defect is found an appropriate repair is performed.
-Preventive Maintenance;
At certain predetermined intervals maintenance is performed.
-Predictive Maintenance;
Ascertained by theoretical and/or practical checks, maintenance is performed right before a breakdown occurs.
The latter use case we will be focusing on, because this will yield the best results with regards to production up-time as, theoretically, Maintenance will only be performed when it’s duly necessary.

Since equipment is growing in sophistication and becoming ever more intelligent, it is now possible to extract, remotely, vast quantities of data on a variety of quality parameters. This bulk data (also called ‘Big Data’) can be used to extrapolate certain key factors from:
-How much energy and/or material the equipment consumes
-Faults and/or warning messages
-How long the equipment is in stand-by or running-mode
-Process values evolve and/or degrade over time
-Etc.

These key factors can be used to make a theoretical model of the practical equipment situation, and thus allows a prediction to be cast of the current state of the equipment. This ‘interconnected predictive maintenance modelling’ falls under the guise of many names; ‘Industry 4.0’, ‘Internet Of Things (IoT)’, ‘Smart Factoring’, etc. (see picture 1: *Industry 4.0*).



Picture 1: *Industry 4.0*

1.2. Problem

This project is aimed at introducing this afore mentioned ‘remote predictive maintenance modelling’ (from now of referred only as ‘Predictive Maintenance’).
The application field will be set-up for use in VDL Nedcar in the sub-factory ‘Body Manufacturing’.

In the ‘Body Manufacturing’ the uncoated metal framework of a car (also called ‘Body in white’ or ‘Body’ for short), is produced.
This ‘Body’ is an amalgamation/collection of metal subparts (of other subparts) assembled by a large amount of automated systems, including robots.

These stand-alone robots (also referred to as ‘manipulators’) drive much of the physical assembly work in producing the ‘Body’. These robots are performing a variety of heavy duty tasks; welding, moving parts, gluing, screwing, etc.

It is on this collection of robots that this project will be applied upon.

The question that is being posed:
“How can we increase our robot production up-time by reducing unwanted, and possibly, avoidable downtime?”

Picture 2: *Body Manufacturing with robots*

1.3. Solution

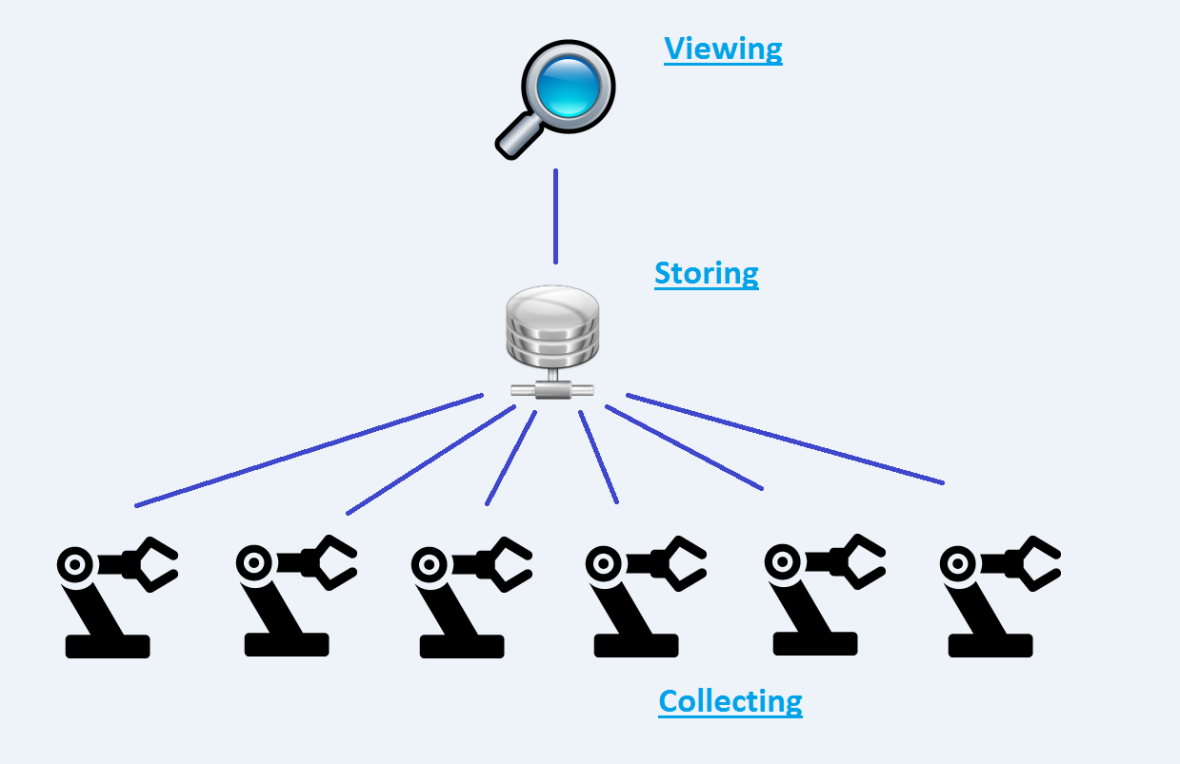
In order to reduce robot downtime the 'Predictive Maintenance' model of Industry 4.0 is used (as described in the introductory Application key note).

The following needs to be accomplished:

- Information needs to be remotely collected
- This collected information needs to be remotely stored in a general location (i.e. server)
- This stored information needs to be remotely viewed, in order to make these predictive plans

This three step model can be applied to the 'Robot Predictive Maintenance Model':

- Collecting fault/warning messages, process values, temperatures, production times, etc.
- Storing robot information on a data server (e.g. SQL server)
- Viewing the stored robot information and make a (automated) maintenance plan



Picture 3: *System diagram*

1.4. System requirements

In order to 'Collect, Store & View' robot data, a constraint is made in the application field;

- Only robot controllers (computer that directs the physical robot manipulator) of the type ABB IRC5 are used.

The following requirements are set:

- A way to remotely connect to a robot controller and retrieve any relevant information
- A storage server to allocate all collected data to of sufficient size (estimated 150GB of information is created annually, further explanation follows in the following paragraphs)
- Access/permission for the robot controller to said storage server
- Access/permission for an engineer to view/modify said storage server

In detail the following has been chosen to fill those requirements;

By using a special SDK (Software Development Kit), distributed by ABB Robotics, we can access all the necessary information via the IRC5 robot controller server. This SKD is written for C# & VB applications. For this project it was decided to use the C# implementation in Microsoft Visual Studio, due to the wide spread use of this programming implementation.

The required data (as described in *1.5. System description*) will be collected from all IRC5 robots in the Body manufacturing via the SDK implementation. Said data will then be send to a data archiving server.

For this project an SQL server (Structured Query Language server) is selected as data server due to there already being an infrastructure for it, and for its easy and widespread use case for it.
The SQL server build is: Microsoft SQL Server 2008.

This part of the system that collects and stores robot information is called the: 'Collect_ABB_IRC5_Info'.

Then said data can then be queried from the SQL server to be viewed and processed locally (office environment).
At this location observation, fault messages and calculation can be performed in order to ascertain (with local static data) a correct Predictive Maintenance plan of all IRC5 robots.

This part of the system for viewing robot data is called the: 'View_ABB_IRC5_Info'.

Robot Data

The following robot data (IRC5) is required to be collected:

Category 1 data [Interval = every day];

- Event log [1000 entries, filtered by essential system fault messages, Timestamp - CategoryID - MessageNumber - Title]

Category 2 data [Interval = every month];

- System Temperature [Celsius]
- CPU Temperature [Celsius]
- Heatsink cooling fan speed [RPM]
- L10H of gearboxes [6+1x axis, expected life rating in hours]
- Actual duty cycles [Hours]
- Last time since servicing [Hours]
- Torque check statistics [6x, 1 per axis. 2 additional if a track is present];
  - Actual tool data at Torque check
    - Tool present [Bool]
    - Tool point XYZ [Millimeter, Millimeter, Millimeter]
    - Tool orientation [Bool,Bool,Bool,Bool]
    - Centre of mass weight [Kilograms]
    - Centre of mass centrepoint XYZ [Milimeter,Millimeter,Millimeter]
    - Inertia point XYZ [Millimeter, Millimeter, Millimeter]
    - Intertia orientation [Bool,Bool,Bool,Bool]
  - Time index of Torque check measurement interval [Milliseconds]
  - Theoretical Torque [Nm]
  - Measured Torque [Nm]
  - Speed setpoint [Millimeters/second]
  - Axis position [Degrees/Millimeters]

Category 3 data [Interval = once per life/project cycle by manual trigger];

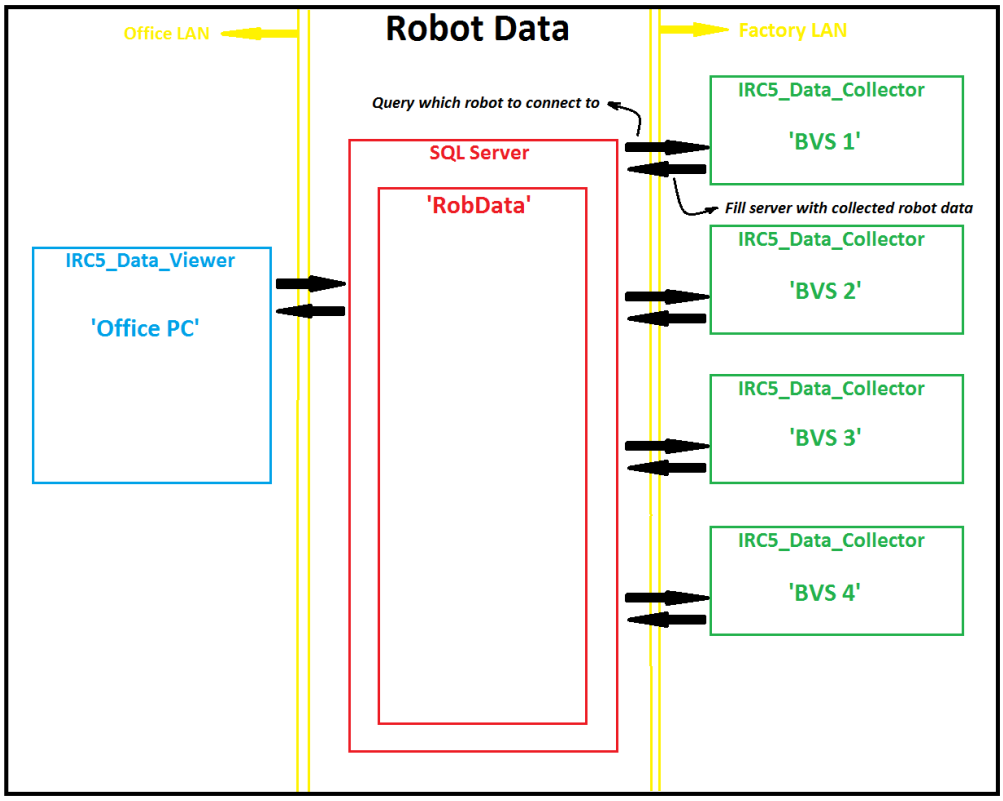
- Controller System ID [Serial number]
- Controller System Name [Name]
- Controller IPAddress [IPv4]
- Controller Robotware Version [Version number]
- Controller Licences [ABB Licence packages]
- Controller Used application [ABB application modules]
- Manipulator Type [Manipulator hardware configuration]
- ---
- -

The bulk of the data resides in the monthly upload of the Category 2 data to the SQL server.
Residing in this Cat2 data, the Torque check statistics the biggest contributor to data size as it consists of about 8000 measuring points.
By practical experiment it is estimated that around ~8MB (megabyte) is produced by one Cat2 data upload (7594kb to be precise).
This means that for the total of 1272 ABB IRC5 controllers present in the ‘Body Manufacturing’ a total of 1272*8 = 10,176GB (gigabyte) per month of data is produced.

Annually this would be 10,176*12 = 122,112GB (gigabyte) of data
for storing the Cat2 data.

1.5. System structure description

Hereby follows an overview of the entire system encompassing the 'Robot Data' project.
The overview displays the *Function Diagram* of the system in its entirety. For the *Function Diagram* of the underlying subcomponents, see the individual chapters '*Functional Description – IRC5_Data_Collector*' and '*Functional Description – IRC5_Data_Viewer*'.

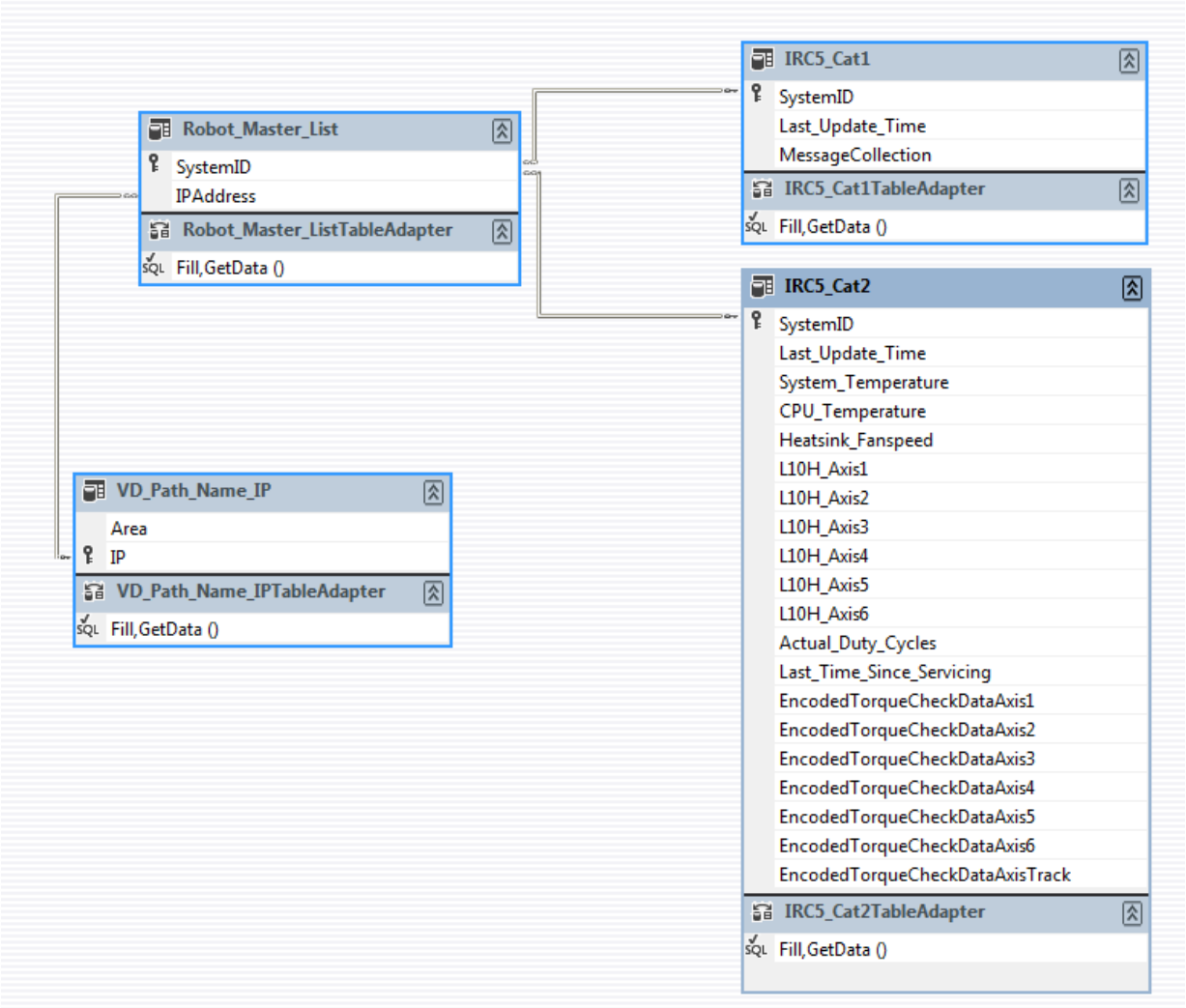


Picture 4: *System structure diagram*

The following is the data structure inside the SQL database:

- Database: Robot_Data; houses all relevant information for all the robots
  - Table: Robot_Master_List; is a unique lookup table for all current controllers
  - Table: VD_Path_Name_IP; is a temporary lookup table for VersionDog path name
  - Table: IRC5_Cat1; houses all fetched Category 1 data
  - Table: IRC5_Cat2; houses all fetched Category 2 data

A database diagram of above information is found in the picture below (picture 5: *Database diagram*)



Picture 5: *Database diagram*

These underlying tables have the following relations;

- Robot_Master_List has a primary key on SystemID
- VD_Path_Name_IP has a foreign key on IP to table Robot_Master_List, IPAddress
- IRC5_Cat1 has a foreign key on SystemID to table Robot_Master_List, SystemID
- IRC5_Cat2 has a foreign key on SystemID to table Robot_Master_List, SystemID

Table 'Robot_Master_List' properties:

- Robot_Master_List has a primary key on SystemID

Column Name	Data Type	Allow Nulls
SystemID	char(15)	☑
IPAddress	char(16)	☑
		☐

Picture 6: *Table 'Robot_Master_List'*

Table ‘VD_Path_Name_IP’ properties:

- VD_Path_Name_IP has a foreign key on IP to table Robot_Master_List, IPAddress

Column Name	Data Type	Allow Nulls
Area	nvarchar(255)	☑
IP	nvarchar(255)	☑
		☐

Picture 7: *Table ‘VD_Path_Name_IP’*

Table ‘IRC5_Cat1’ properties:

- IRC5_Cat1 has a foreign key on SystemID to table Robot_Master_List, SystemID

Column Name	Data Type	Allow Nulls
SystemID	char(15)	☑
Last_Update_Time	datetime	☑
MessageCollection	text	☐
		☐

Picture 8: *Table ‘IRC5_Cat1’*

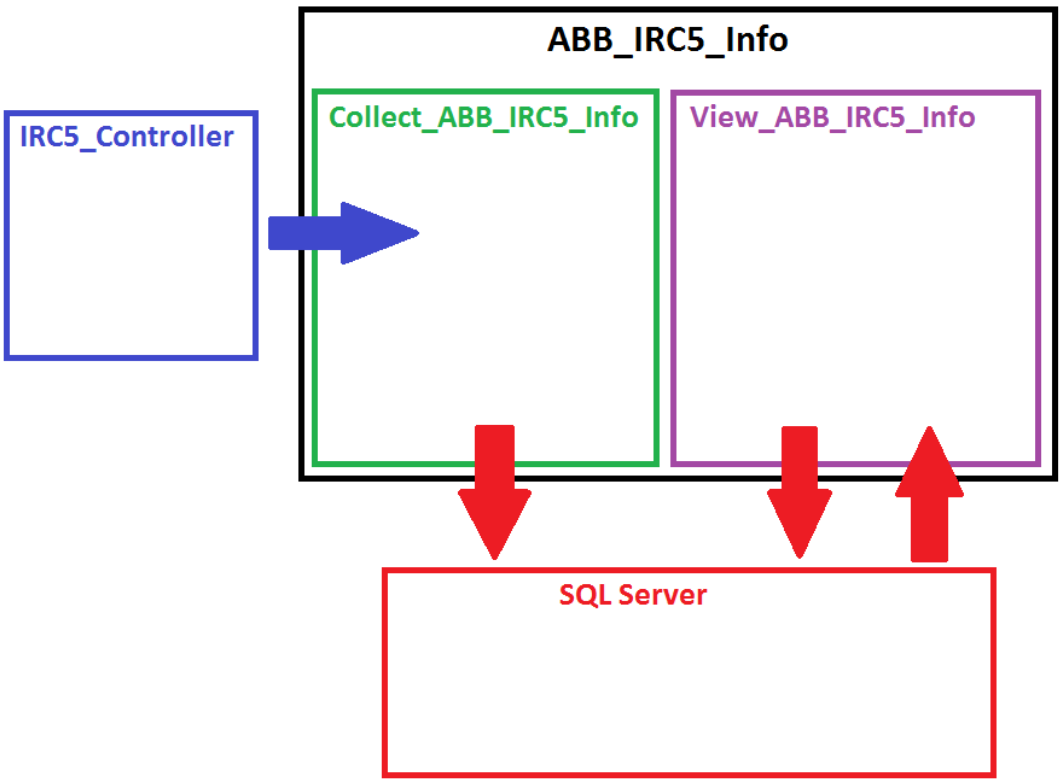
Table ‘IRC5_Cat2’ properties:

- IRC5_Cat2 has a foreign key on SystemID to table Robot_Master_List, SystemID
- The info of the ‘EncodedTorqueCheckDataAxisx/Track’ is XML serialized data for easier loading and writing within the program

Column Name	Data Type	Allow Nulls
SystemID	char(15)	☑
Last_Update_Time	datetime	☑
System_Temperature	smallint	☑
CPU_Temperature	smallint	☑
Heatsink_Fanspeed	smallint	☑
L10H_Axis1	int	☑
L10H_Axis2	int	☑
L10H_Axis3	int	☑
L10H_Axis4	int	☑
L10H_Axis5	int	☑
L10H_Axis6	int	☑
Actual_Duty_Cycles	int	☑
Last_Time_Since_Servicing	int	☑
EncodedTorqueCheckDataAxis1	text	☑
EncodedTorqueCheckDataAxis2	text	☑
EncodedTorqueCheckDataAxis3	text	☑
EncodedTorqueCheckDataAxis4	text	☑
EncodedTorqueCheckDataAxis5	text	☑
EncodedTorqueCheckDataAxis6	text	☑
EncodedTorqueCheckDataAxisTrack	text	☑
		☐

Picture 9: *Table ‘IRC5_Cat2’*

The system responsible for collecting the robot data, called ‘Collect_ABB_IRC5_Info’, is going to be running on the same program as the system responsible for viewing the collected data from the storage server, called ‘View_ABB_IRC5_Info’.
The common program/runtime of both these subsystems is called ‘ABB_IRC5_Info’.
Each of the two subcomponents communicate on their own with the SQL server and/or robot controller as is shown in the picture diagram below (Picture 9: *Program Diagram*).



Picture 9: *Program Diagram*

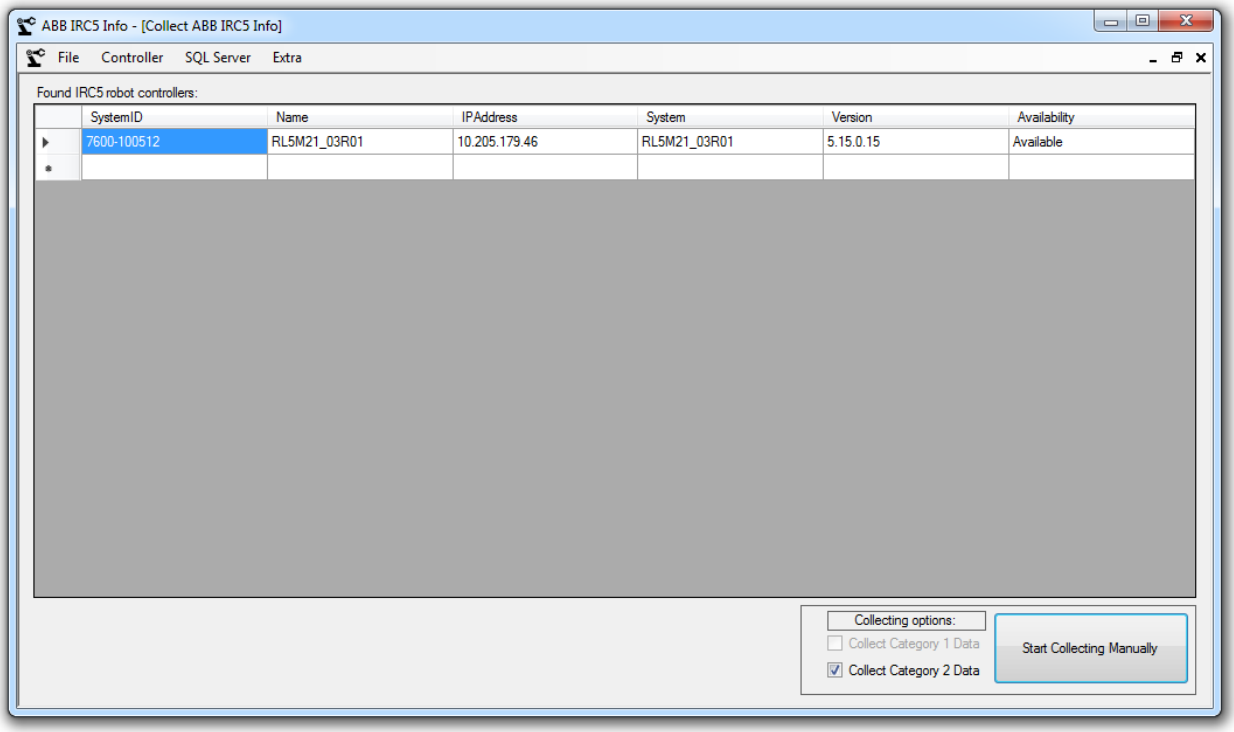
2. Functional Description – Collect_ABB_IRC5_Info

Name: Collect_ABB_IRC5_Info
Type: Sub System
Version: 1.0

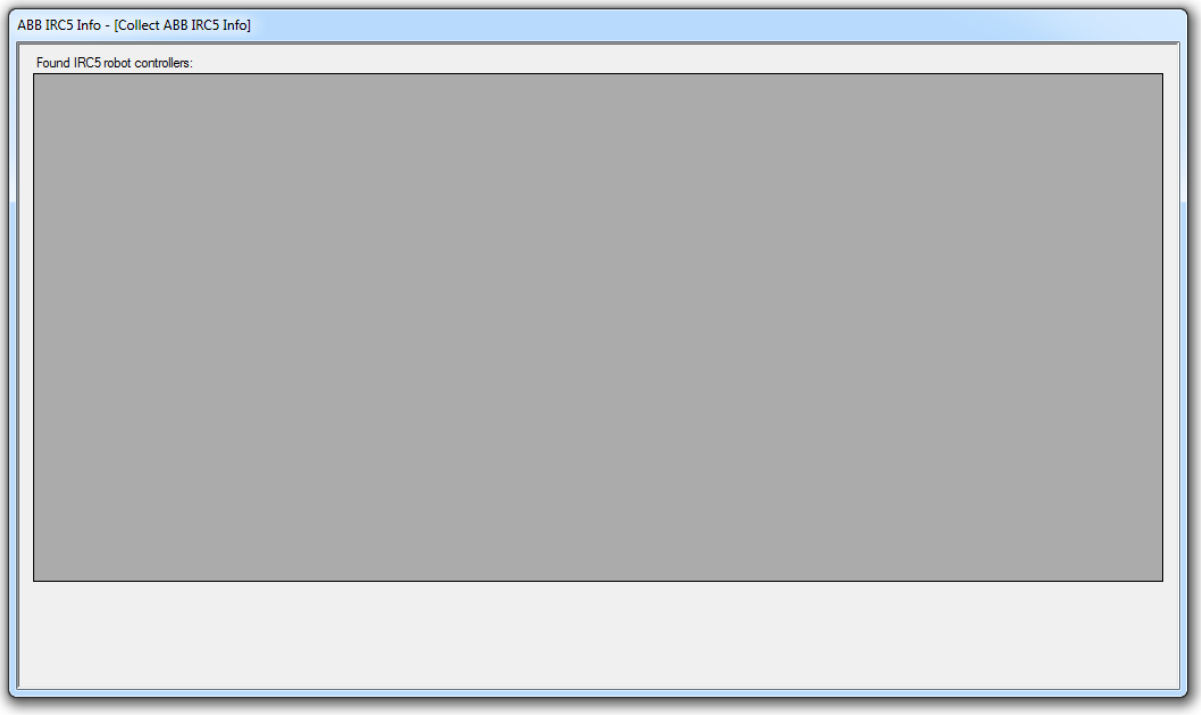
2.1. Background information

The program ‘Collect_ABB_IRC5’ is used to collect Cat1,2 robot data and send it to the SQL server. Currently this can only be executed manually. In the future it is desired for this to be run automatically at a fix interval (once per day for Cat1 data, once per month for Cat2 data). When this application runs in the ‘Factory_Lan’ (see Picture 4: *System structure diagram*) only the automatic collector will be shown. All other functionalities will be hidden/blocked from the user as to avoid (accidental) misuse.

To access the ‘Collect_ABB_IRC5’ while ‘CollectorMode’ is inactive you must navigate to [Controller] -> [Collect_ABB_IRC5].



Picture 10: *Collector in manual mode*



Picture 10: *Collector in automatic mode*

In order to accomplish this, the program uses the following custom C# Classes:

- SQL_IO: For sending/receiving SQL data in .Net DataTable format
- IRC5_IO: For connecting, sending and receiving IRC5 controller data
- XML_IO: For en/decoding TorqueCheckData classes to send to the SQL server
- TestNetworkConnection: For pinging robotcontrollers in a given subnet for availability
- DataTable_IO: For creating custom DataTable in use for SQL queries

2.2. Function diagram

For manual & automatic collection of IRC5 data the following functional diagram is followed:

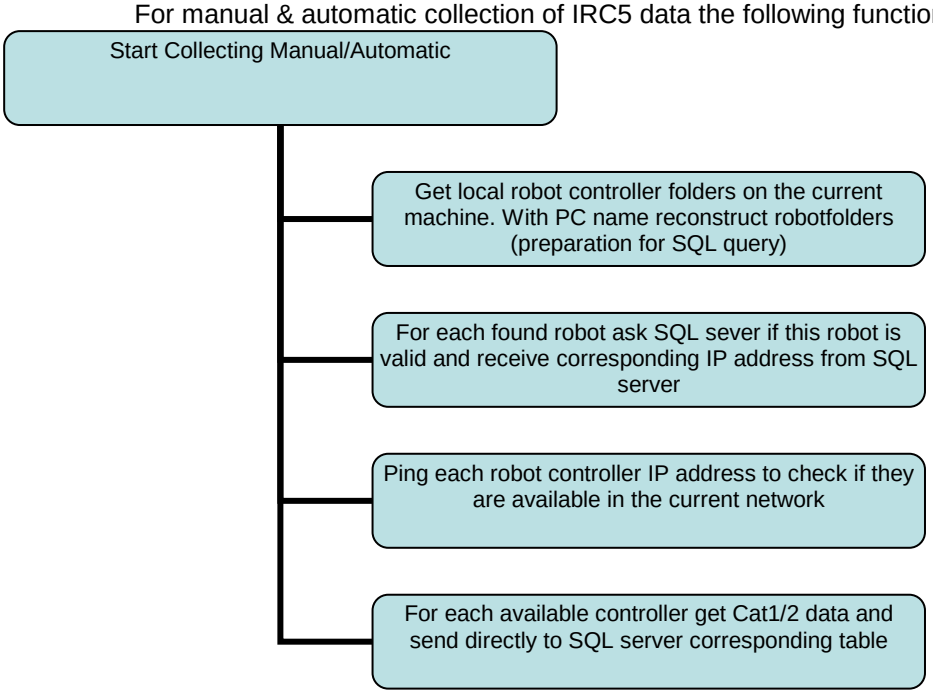


Diagram 1: *Collector function diagram*

If it is desired to run the collector without accessing a robot controller (i.e. dry run operation) then this can be accomplished by changing the program setting 'EmulationActive' to value 'True'.
If it is desired to run the collector without accessing and checking the local PC name then the program setting 'SimulationActive' can be set to value 'True'.

When the collector runs into a situation where you are either emulating and/or simulating then, because there is no reference data, you will be prompted to provide the necessary data (controller list and/or Cat2 data).

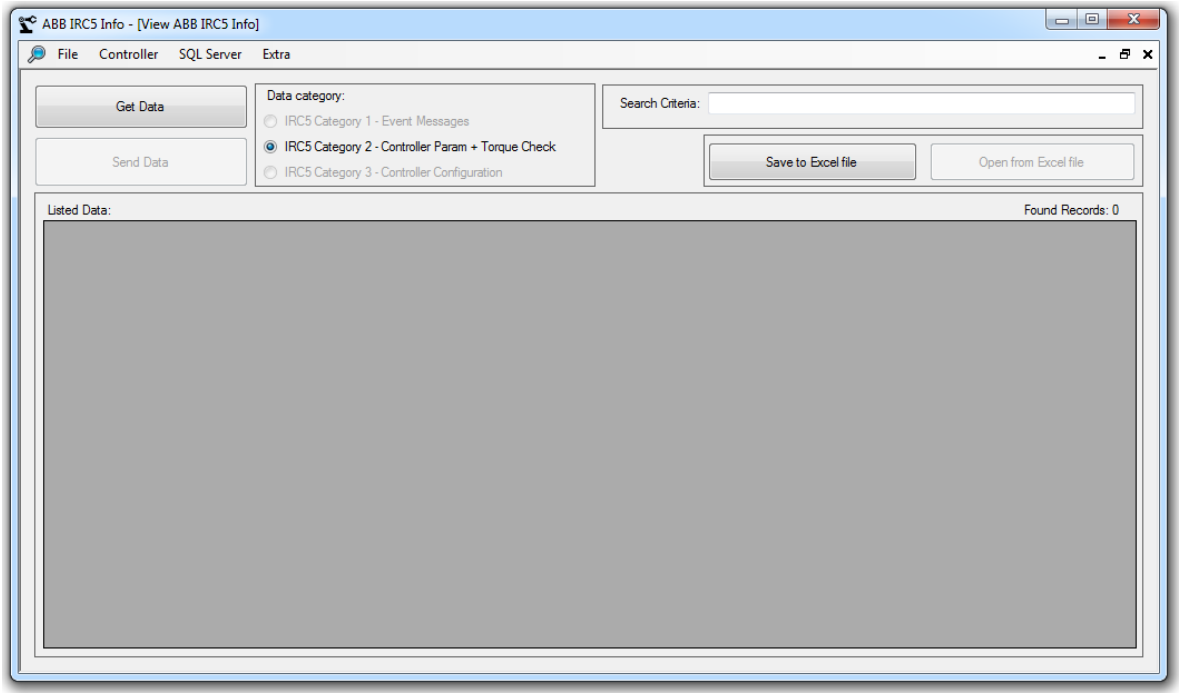
During automatic run no error/fault messages are generated when the application runs into a fault. It will simply default to leaving out the information and/or skipping the program.
Incomplete data delivery to the SQL server is responsible for the supervising engineer.
If separate fault handling/checking is requested/required on the 'View_ABB_IRC5_Info' then this can be appended in the future.

3. Functional Description – View_ABB_IRC5_Info

Name: View_ABB_IRC5_Info
Type: Sub System
Version: 1.0

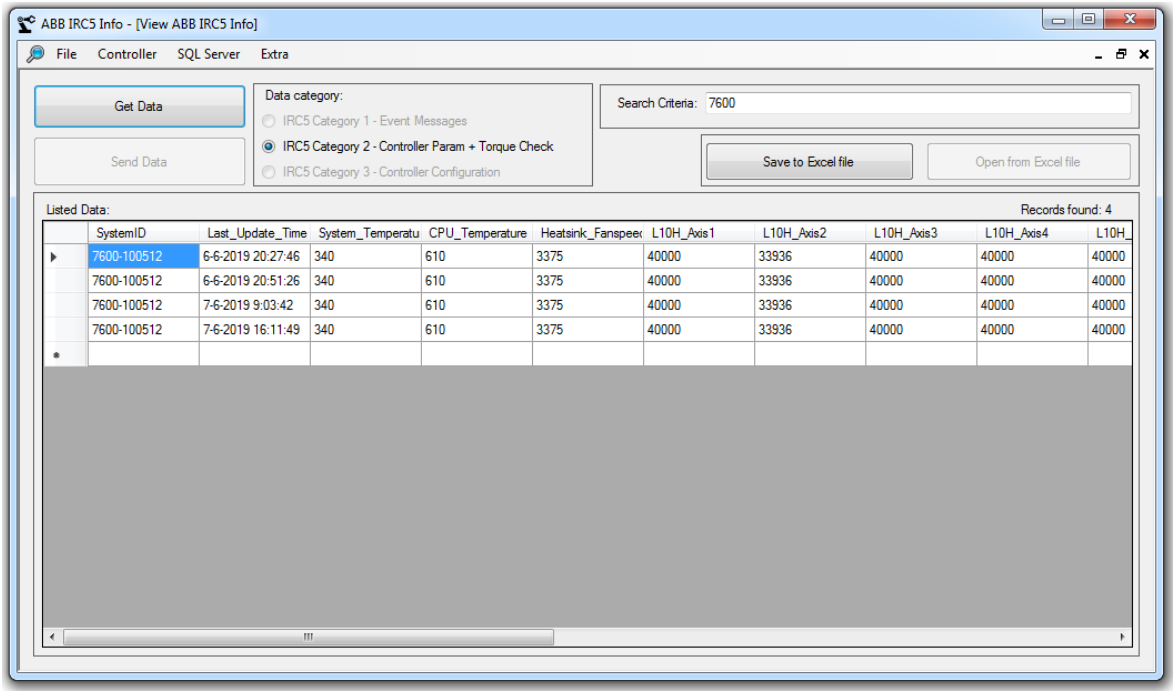
3.1. Background information

The program 'View_ABB_IRC5' is used to view and analyse data stored in the database 'RobotData'. As of now the viewable data is only the Cat2 information.
The analytics of the robot data, in order to create a 'predictive maintenance plan', is not required/requested. If needed this can be appended in the future.



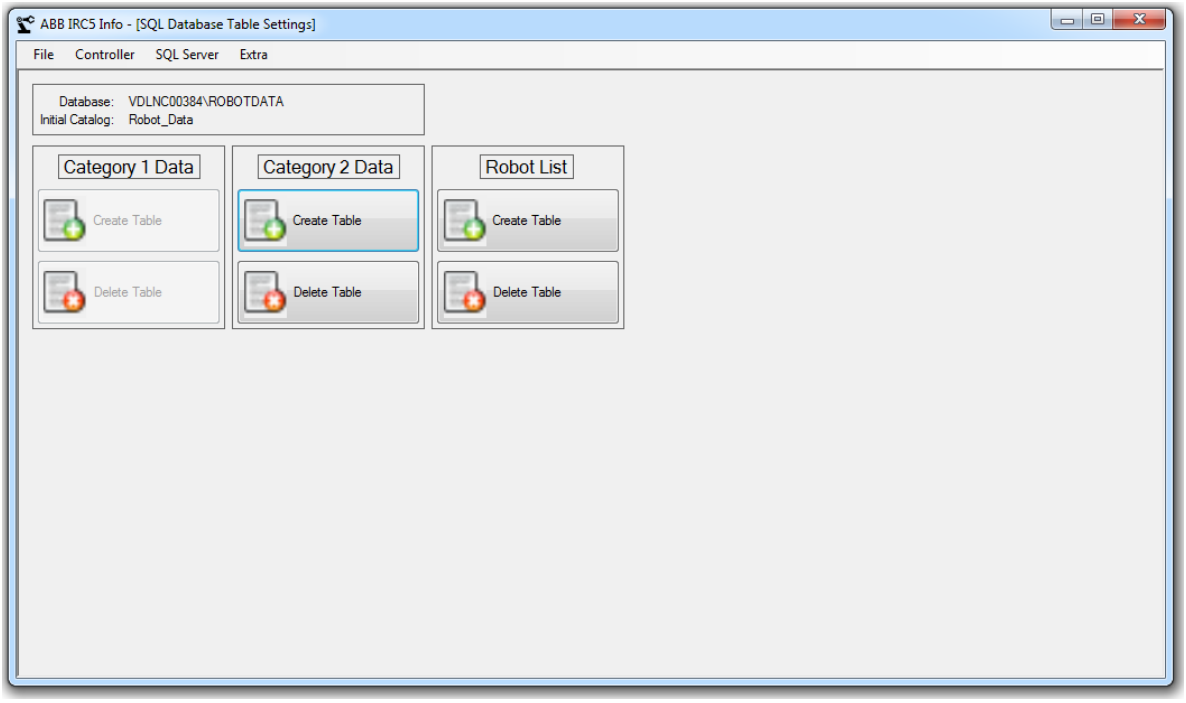
Picture 11: *View Cat2 Data*

In the search criteria the desired robot with the SystemID tag can be filtered out. Doing so will yield the following result shown in 'Picture 12: *View Cat2 Data searched*'



Picture 12: *View Cat2 Data searched*

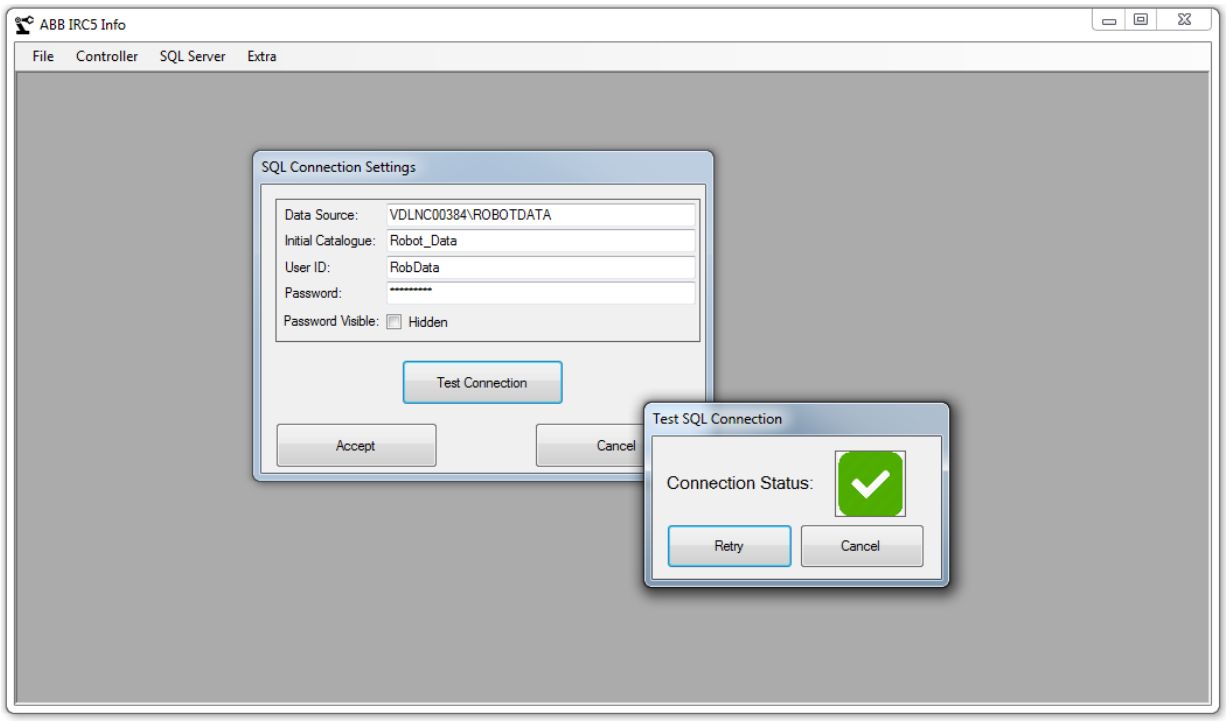
If it is deemed necessary to create and/or delete certain tables inside the SQL Database ‘Robot_Data’, then the function under tab [SQL Server] -> [Table Settings] can be called.
Creating a table will also create all column definitions, including datatypes.



Picture 13: *SQL Table settings*

If for some reason the SQL database credentials/connection settings change in the future, it is possible to change the SQL login data within the program.
Navigate to [SQL Server] -> [Connection Settings] and you will be prompted to change the SQL credentials/connection settings. These settings are saved within the program itself.

Also one could try and test if the entered credentials/connection settings are valid via the ‘Test Connection’ button.



Picture 14: *SQL Connection settings and testing*

In order to accomplish this, the program uses the following custom C# Classes:

- SQL_IO: For sending/receiving SQL data in .Net DataTable format
- XML_IO: For en/decoding TorqueCheckData classes to send/recieved from the SQL server
- TestNetworkConnection: For pinging the SQL servers availability
- DataTable_IO: For creating custom DataTable in use for SQL queries
- Settings: For storing program settings
- Excel_IO: For archiving queried table to Excel format (currently not supported)

3.2. Function diagram

None; non-linear program.

4. Build contents

4.1. ABB_IRC5_Collect

The program ‘ABB_IRC5_Info’ uses the following class list;

- SQL_IO: For sending/receiving SQL data in .Net DataTable format
- XML_IO: For en/decoding TorqueCheckData classes to send/recieved from the SQL server
- IRC5_IO: For connecting, sending and receiving IRC5 controller data
- TestNetworkConnection: For pinging the SQL servers availability
- DataTable_IO: For creating custom DataTable in use for SQL queries
- Settings: For storing program settings
- Excel_IO: For archiving queried table to Excel format (currently not supported)

The program ‘ABB_IRC5_Info’ uses the following references;

- ABB.Robotics.Controllers.PC
- Microsoft.CSharp
- System
- System.Core
- System.Data
- System.Data.DataSetExtensions
- System.Deployment
- System.Drawing
- System.Net.Http
- System.Windows.Forms
- System.Xml
- System.Xml.Linq

This application was build with the following software versions:

- Microsoft Visual Studio 2017 under Framework 4.6.1.
- MS SQL Server 2008
- Microsoft SQL Server Management Tool 18
- ABB IRC PC SDK 6.08
- Microsoft Windows 7 & Windows 10