Predictive maintenance of a conveyor belt

University of Applied Sciences Upper Austria
2018

Abstract

Normally in the industry we have to do periodic controls of all our equipment to prevent or solve failures and malfunctions. Also with an appropriated and regular maintenance is possible to prolong the useful life of the facilities and nevertheless it will save the company money.

Introduction

There are three types of maintenance or quality control:

1. Preventive: where facilities are inspected and protected before break down or other problems occur.
2. Corrective: where equipment is repaired or replaced after wear, malfunction or break down.
3. Pedictive: which uses sensor data to monitor a system, then continuously evaluates it against historical trends to predict failure before it occurs

In this report we present a case of predictive maintenance of a conveyor belt.

There are many parameters that we can measure in the belt itself as well as in the environment to prevent a malfunction or to estimate the useful life of the belt, for example: vibration, tension, temperature of the belt, temperature and humidity of the environment, fatigue in the rollers, alignment or cleaning.

We decided to measure the temperature of the environment with a special sensor and the velocity of the belt with a rotary encoder. To be able to see the variables throw a computer, we have used the following elements: a raspberry pi 3, a ESP32 chip and CODESYS software.

A brief theoretical framework and a practical part are included below.

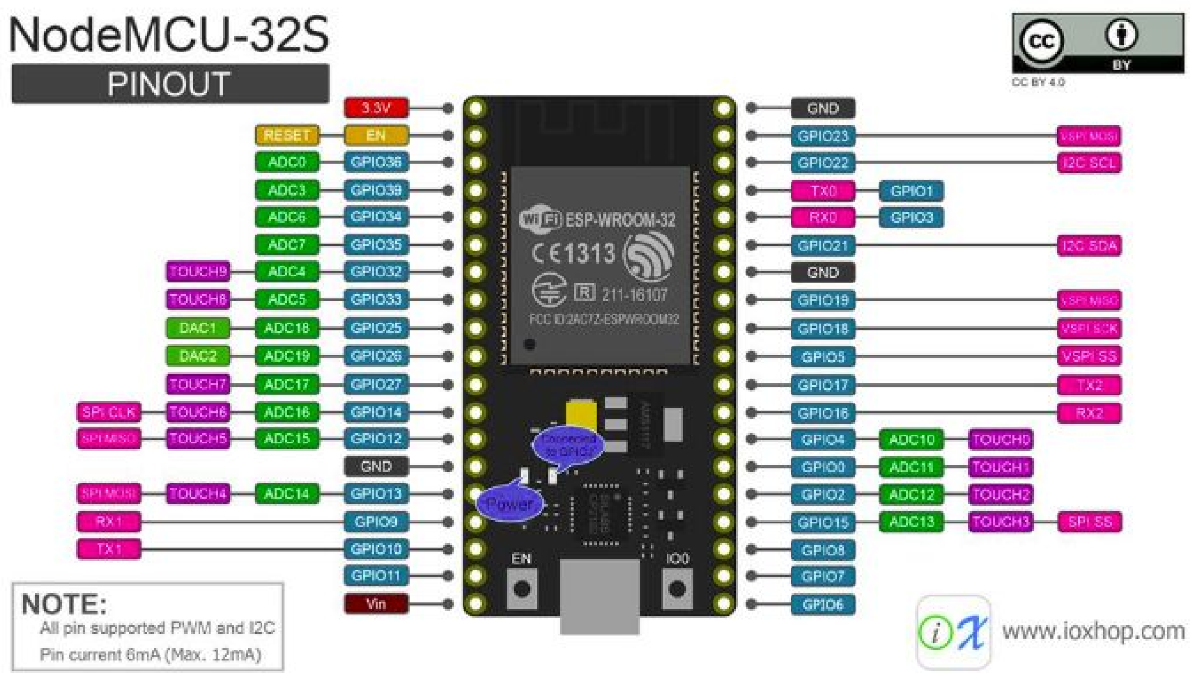
Sources:
https://en.wikipedia.org/wiki/Maintenance_(technical)

Theoretical part

1. ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth.
We decided to work with this chip because the mentioned characteristics, primarily the low cost and the Wifi provision, and doubt to It can be programmed with Arduino IDE. This software makes it easy to write code and we could find a lot of documentation to use ESP32 with Arduino IDE in the following repository:

https://github.com/espressif/arduino-esp32



Sources:
https://en.wikipedia.org/wiki/ESP32

2. CODESYS

CODESYS (an acronym for controller development system) is a development environment for programming controller applications. CODESYS licenses are free of charge and can be installed legally without copy protection on further workstations. There are five PLC programming languages available in CODESYS. They are defined in the international industrial standard IEC 61131-3:

- IL (instruction list)
- ST (structured text) is similar to programming in Pascal or C.
- LD (ladder diagram) enables the programmer to virtually combine relay contacts and coils.
- FBD (function block diagram).
- SFC (sequential function chart) .

An optional toolkit enables the user to create his own visualization elements.

Another characteristic that make it interesting for us is that we can extract data throw OPC UA, a protocol that makes it easy to establish connection between several PLCs.

Sources:

https://en.wikipedia.org/wiki/CODESYS

3. Modbus

Modbus is a serial communication protocol to make communication possible between automation devices. It consists in two parts: the application layer and the underlying networking. The application layer is the core of the protocol and determines many of the design constraints. The Modbus application layer is implemented as a request-response, master-slave design for transmitting single-point data across various networking layers.
There is a client, or more commonly called master, that generates a request and waits on the response and a slave device that analyze the requests from the master, handles them, and then returns a response.
Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems.

We used Modbus because is a *de facto* standard communication protocol, meaning that is very popular with years of proven reliability and because it is an open protocol that we can use for free. Specifically we used MODBUS TCP/IP that is a variant of the MODBUS family. It covers the use of MODBUS messaging in an ‘Intranet’ or ‘Internet’ environment using the TCP/IP protocols.

Sources:
https://www.google.com/url?q=https://www.wisdomjobs.com/e-university/modbus-interview-questions.html&sa=D&ust=1544530024419000&usg=AFQjCNExstZyoy4YNBz_bZf3IZu0_76bCw
http://www.ni.com/white-paper/52135/en/

4. OPC UA

OPC Unified Architecture (OPC UA) is a machine to machine communication protocol for industrial automation developed by the OPC Foundation.
We used to transfer data from the Raspberry Pi to CODESYS. An advantage is that is relatively easy establishing connection with a PLC or between several PLCs.

Sources:
https://en.wikipedia.org/wiki/OPC_Unified_Architecture

5. Raspberry PI 3

A Raspberry Pi is a “single-board” computer, meaning a very simple one with just one circuit board, with wireless LAN and Bluetooth connectivity. It has the size of a credit card and costs less than $40.
The Raspberry Pi can run simple programs and they can be designed to do a nearly limitless variety of things. The Raspberry can be programmed using a variety of programming languages (included Python) and has the ability to run multiple programs.
The technical characteristics are:

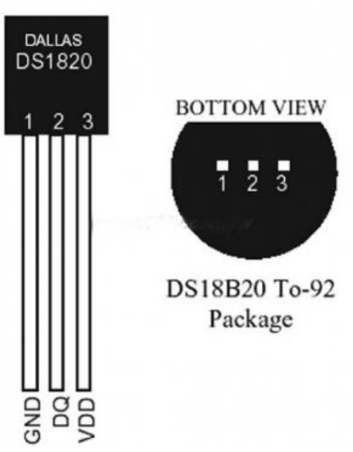
- 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU (BCM2837)
- 1GB RAM (LPDDR2 SDRAM)
- On-board wireless LAN - dual-band 802.11 b/g/n/ac (CYW43455)

- On-board Bluetooth 4.2 HS low-energy (BLE) (CYW43455)
- 4 x USB 2.0 ports
- 300 Mbit/s ethernet
- 40 GPIO pins
- Full size HDMI 1.3a port
- Combined 3.5mm analog audio and composite video jack
- Camera interface (CSI)
- Display interface (DSI)
- microSD slot
- VideoCore IV multimedia/3D graphics core @ 400MHz/300MHz

Sources:
https://thepihut.com/products/raspberry-pi-3-model-b-plus

6. DS18B20 Temperature Sensor

The DS18B20 is a digital sensor that communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor.
The sensor has just three pins.



- Operating range temperature: -55ºC to 125ºC
- Accuracy +/-0.5 ºC (between the range -10ºC to 85ºC)

The DS18B20 is also available in waterproof version but in our case we used the normal version.

Sources:

https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

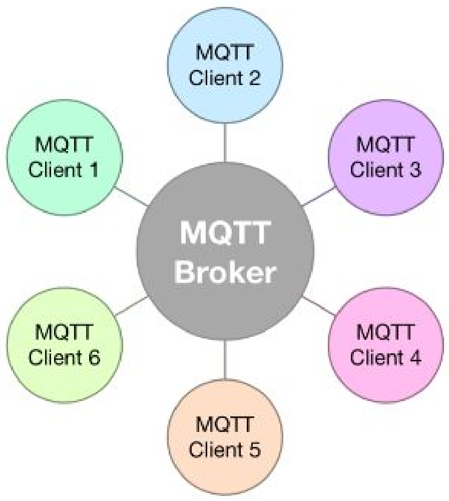
7. Node-RED

Node-RED is an open source visual editor that allows programmers to rapidly interconnect physical i/o, cloud based systems, databases and most API's in any combination. Node-RED can be run on most modern computer system included Windows, Mac, Linux or even lightweight computers like Raspberry Pi or industrially hardened appliances like ARQ.
Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.
JavaScript functions can be created within the editor using a rich text editor.

Sources:
https://nodered.org/
https://www.youtube.com/watch?v=3AR432bguOY&t=492s

8. MQTT

MQTT (Message Queuing Telemetry Transport) is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport.
An MQTT system consists of clients communicating with a server, often called a "broker". A client may be either a publisher of information or a subscriber. Each client can connect to the broker.



Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. If a broker receives a topic for which there are no current subscribers, it will discard the topic unless the publisher indicates that the topic is to be retained. Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics.

Sources:
https://en.wikipedia.org/wiki/MQTT

9. Rotary encoder LPD3806

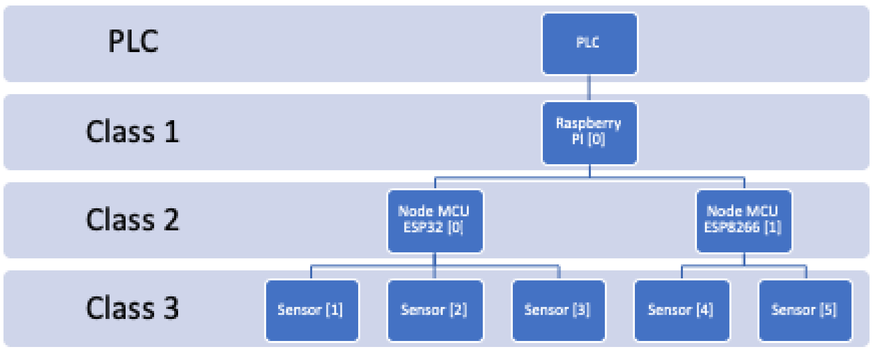
An encoder is an electrical mechanical device that converts linear or rotary displacement into digital or pulse signals. The most popular type of encoder is the optical encoder, which consists of a rotating disk, a light source, and a photo detector (light sensor). The disk, which is mounted on the rotating shaft, has patterns of opaque and transparent sectors coded into the disk. As the disk rotates, these patterns interrupt the light emitted onto the photo detector, generating a digital or pulse signal output. An incremental encoder generates a pulse for each incremental step in it's rotation. Although the incremental encoder does not output absolute position, it can provide high resolution at an acceptable price. The most common type of incremental encoder uses two output channels (A and B) to sense position. By monitoring both the number of pulses and the relative phase of signals A and B, we can track both the position and direction of rotation.

Specifications:
Green = A phase; white = B phase; red = Vcc; black = GND.
Note: AB two -phase outputs must not be connected directly to VCC, otherwise, they will burn the output transistor.
Power source: DC 5-24V
Maximum mechanical speed: 5000 rev / min.
Performance: 400 pulses / rev.

Sources:
https://www.pc-control.co.uk/incremental_encoders.htm

Practical part

1. System architecture



Devices are chained into three classes.

In the first class is the only one main device, the device is Raspberry PI, which acts like a master to class two and three. From class one is possible to obtain data from devices in class two and three.

In the second class are microcontrollers, which are capable to obtain data from various range of sensors.

In the third class are sensors, as a proof of concept we made an example with sensors with temperature sensor and rotary encoder.

Communication protocols

Communication direction	Wired/Wireless	Connection protocol	Data transfer protocol
Class 1 -> PLC	Both	IEEE 802.11 (Wi-Fi) or IEEE 802.3 (Ethernet)	OPC UA

Class 1 -> Class 2	Wireless	IEEE 802.11 (Wi-Fi)	Modbus over TCP/IP, MQTT
Class 2 -> Class 3 (Node MCU ESP8266)	Wired	UART, GPIO, I2C, I2S, SDIO, ADC and SPI	Custom or sensor based
Class 2 -> Class 3 (Node MCU ESP32)	Both	Bluetooth, UART, GPIO, I2C, I2S, SDIO, ADC and SPI	Custom or sensor based

Sources:
ESP 32 Datasheet
esp32_datasheet_en.pdf
ESP 8266 Datasheet
https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

2. Raspberry Pi

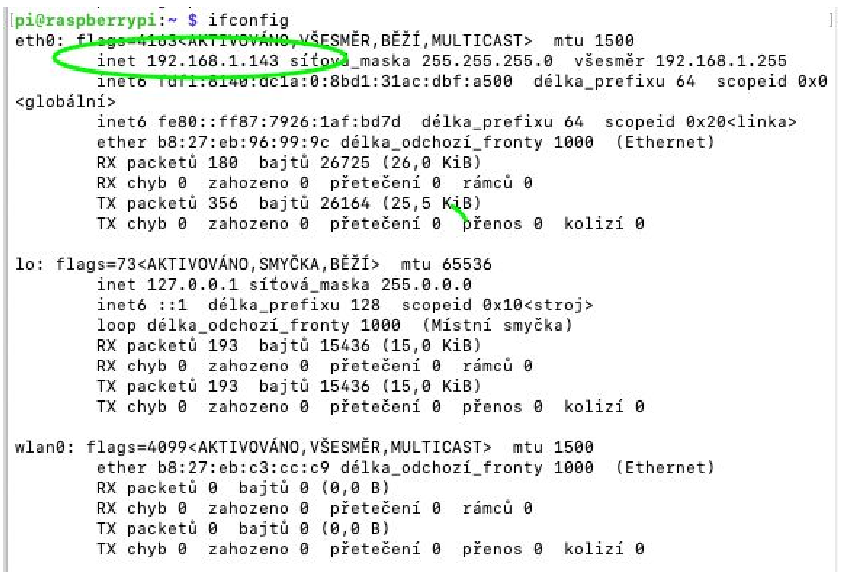
Installation of Raspbian OS

Before installation of CODESYS Control for Raspberry Pi SL is necessary to have Raspberry Pi with image of Raspbian OS on SD card.
How to achieve flashing of Raspbian OS is written on this page: https://github.com/raspberrypi/documentation/blob/master/installation/installing-images/README.md
Those steps are done on PC.
After completing the flashing of image we recommend to:

1. attach USB keyboard, mouse and HDMI monitor to Raspberry Pi,
2. insert SD card with flashed Raspbian image,
3. connect Raspberry Pi over Ethernet to local network,
4. plug in micro-usb connector with power,
5. follow command on the screen.

After the boot of Raspberry Pi it is recommended to change the password, tutorial could be found there:

https://github.com/raspberrypi/documentation/blob/master/linux/usage/users.md#change-your-password
Next step is to determine the IP-address of the device in local network, which will be needed in next step of installation.



```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<AKTIVOVÁNO,VŠESMĚR,BĚŽÍ,MULTICAST>  mtu 1500
        inet 192.168.1.143 síťová_maska 255.255.255.0  všesměr 192.168.1.255
        inet6 fdf1:8140:dc1a:0:8bd1:31ac:dbf:a500  délka_prefixu 64  scopeid 0x0
<globální>
        inet6 fe80::ff87:7926:1af:bd7d  délka_prefixu 64  scopeid 0x20<linka>
        ether b8:27:eb:96:99:9c délka_odchozí_fronty 1000  (Ethernet)
        RX packetů 180  bajtů 26725 (26,0 KiB)
        RX chyb 0  zahozeno 0  přetečení 0  rámců 0
        TX packetů 356  bajtů 26164 (25,5 KiB)
        TX chyb 0  zahozeno 0  přetečení 0  přenos 0  kolizí 0

lo: flags=73<AKTIVOVÁNO,SMYČKA,BĚŽÍ>  mtu 65536
        inet 127.0.0.1 síťová_maska 255.0.0.0
        inet6 ::1  délka_prefixu 128  scopeid 0x10<stroj>
        loop délka_odchozí_fronty 1000  (Místní smyčka)
        RX packetů 193  bajtů 15436 (15,0 KiB)
        RX chyb 0  zahozeno 0  přetečení 0  rámců 0
        TX packetů 193  bajtů 15436 (15,0 KiB)
        TX chyb 0  zahozeno 0  přetečení 0  přenos 0  kolizí 0

wlan0: flags=4099<AKTIVOVÁNO,VŠESMĚR,MULTICAST>  mtu 1500
        ether b8:27:eb:c3:cc:c9 délka_odchozí_fronty 1000  (Ethernet)
        RX packetů 0  bajtů 0 (0,0 B)
        RX chyb 0  zahozeno 0  přetečení 0  rámců 0
        TX packetů 0  bajtů 0 (0,0 B)
        TX chyb 0  zahozeno 0  přetečení 0  přenos 0  kolizí 0
```

1. Open terminal and type command ifconfig,
2. write down the inet address of eth0 device, the position is circled in the picture.

Installation of CODESYS Control for Raspberry Pi SL

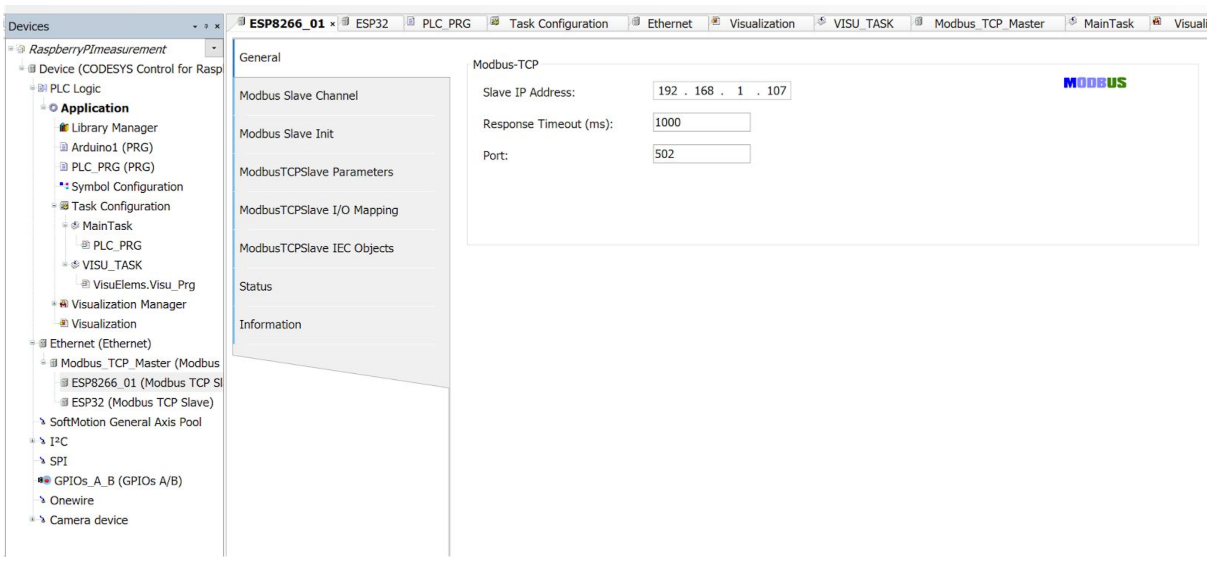
Those steps are done from a PC with Windows, since there is no other supported OS by CODESYS Development System.

1. Make a registration in Codesys Store. This step is necessary to have an ability of downloading free software in Codesys Store. Link to the registration is following:
   https://store.codesys.com/customer/account/login/
2. Download CODESYS Development System V3 from:
   https://store.codesys.com/codesys.html
3. Download CODESYS Control for Raspberry Pi SL from:
   https://store.codesys.com/codesys-control-for-raspberry-pi-sl.html
4. Install CODESYS Development System V3 on the computer.
5. The installation of CODESYS Control for Raspberry Pi SL to the Raspberry Pi could be done according to this tutorial: https://www.youtube.com/watch?v=6FPf3RHWyeU.

Codesys source code

Source code for Codesys could be found in our repository to see in chapter Repository. In Codesys simply open the project in the folder.

Codesys asks MCUs for data over Modbus and visualize them. It is necessary to configure appropriate IP Address for each MCU in this window.



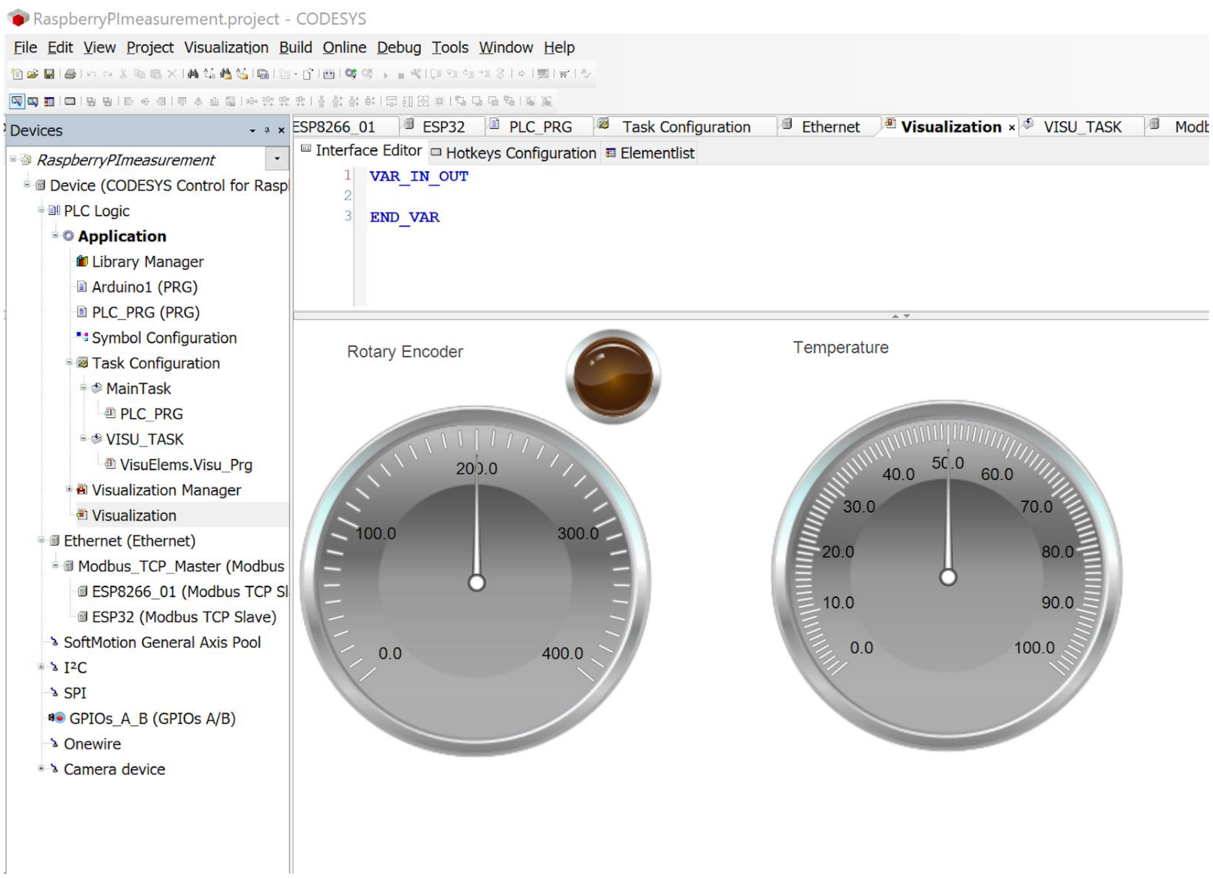
RaspberryPImeasurement
- Device (CODESYS Control for Rasp
  - PLC Logic
    - Application
      - Library Manager
      - Arduino1 (PRG)
      - PLC_PRG (PRG)
      - Symbol Configuration
      - Task Configuration
        - MainTask
          - PLC_PRG
        - VISU_TASK
          - VisuElems.Visu_Prg
      - Visualization Manager
      - Visualization
  - Ethernet (Ethernet)
    - Modbus_TCP_Master (Modbus
      - ESP8266_01 (Modbus TCP Sl
      - ESP32 (Modbus TCP Slave)
  - SoftMotion General Axis Pool
  - I²C
  - SPI
  - GPIOs_A_B (GPIOs A/B)
  - Onewire
  - Camera device

General | Modbus Slave Channel | Modbus Slave Init | ModbusTCPSlave Parameters | ModbusTCPSlave I/O Mapping | ModbusTCPSlave IEC Objects | Status | Information

Modbus-TCP

Slave IP Address:	192 . 168 . 1 . 107
Response Timeout (ms):	1000
Port:	502

For more Modbus configuration possibilities see developer manual. https://faq.codesys.com/display/CDSFAQ/Modbus+master+slave+communication+over+Ethernet
With Modbus we map variables from MCU to variables in PLC. In source code we just recognize rotary encoder rotation direction and then visualize with WebVisu.



```
VAR_IN_OUT

END_VAR
```

Manual how to start OPC-UA server could be found there:
https://www.codesys.com/products/codesys-runtime/opc-ua-server.html

Sources:
CODESYS Control for Raspberry Pi SL Datasheet

https://store.codesys.com/attachments/files/download/id/6934/

MQTT broker installation and configuration

For using MQTT protocol we need to install MQTT broker. We decided to use Mosquitto. Installation could be done by opening terminal and typing following commands:

```
wget -O - http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key | apt-key
add -
apt-get update && apt-get install mosquitto
```

Those are taken from http://repo.mosquitto.org/debian/readme.txt.
Mosquitto now runs permanently on our Raspberry PI.

Node-red configuration

We use Node-Red to take data from MCUs over MQTT and to visualize them. There is an option to bring data to OPC-UA with external libraries.

Node-red comes with default Raspbian installation. In general, it is good to start it with boot of device. This could be achieved by opening terminal and using following commands:

```
sudo systemctl enable nodered.service
sudo reboot
```

For more configuration see the official manual:
https://nodered.org/docs/hardware/raspberrypi

The configuration server is available on http://localhost:1880. If you are using directly on the Raspberry Pi then this address is fine. When using on another computer replace the "localhost" with IP address of Raspberry Pi, which could be found according to a manual at the beginning of chapter.

For using our source code it is needed to install extension for visualization called Node-Red Dashboard. This could be done by typing following commands to console:

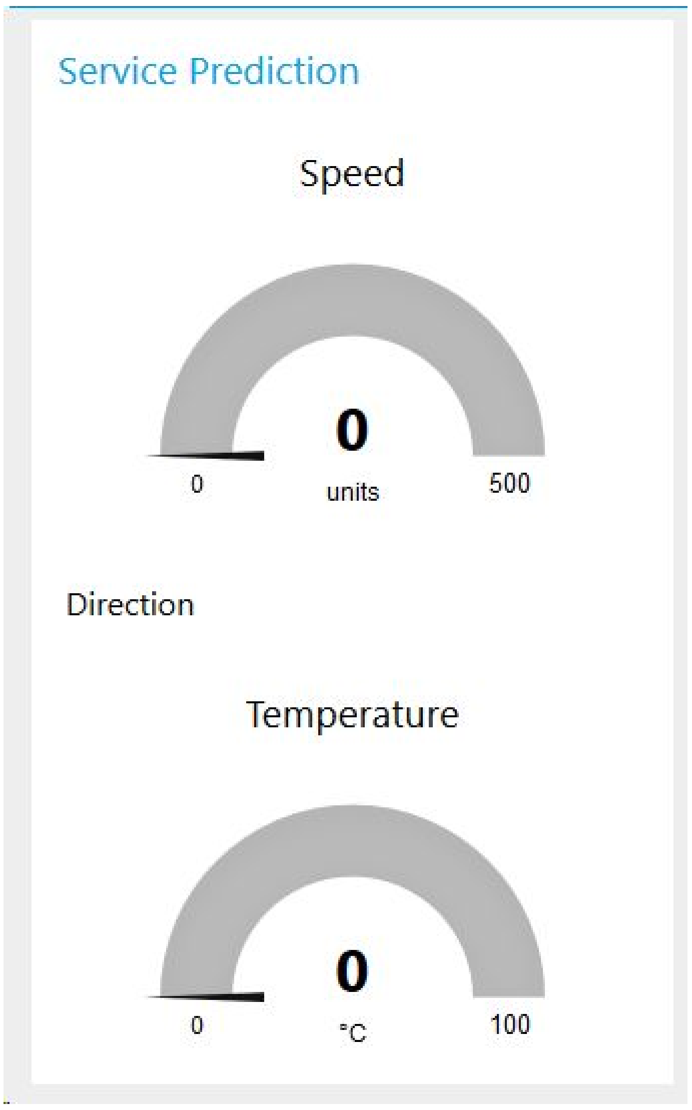
```
sudo apt-get install npm
cd ~/.node-red
npm i node-red-dashboard
```

Source code location could be found in Repository chapter.

It could be imported from file by copying and using import form in Node-Red according to this manual: https://nodered.org/docs/user-guide/editor/workspace/import-export.

Deploy the code by pressing red Deploy button in top right corner.

Type to the browser http://localhost:1880/ui address to see the visualization. The output should look like this.


Service Prediction

Speed

Direction

Temperature

3. Node MCU ESP8266/ESP32

In general there are several ways how to programme microcontrollers from Espressif company. There is a list of those mentioned by the manufacturer:

1. use official SDK provided by Espressif,
2. use Arduino framework,
3. Mongoose OS,
4. MicroPython.

In purpose of simplicity and fast development we use Arduino framework.

For Arduino framework there are two possible IDEs - Arduino IDE, PlatformIO IDE. Since the second one provides auto-complete for coding and in general we consider it as more comfortable we use it. Code is compatible with Arduino IDE.

Instalation tutorial could be found there:
https://docs.platformio.org/en/latest/ide/vscode.html#installation.
For introduction to PlatformIO IDE we recommend this website as a starting point:
https://docs.platformio.org/en/latest/ide/vscode.html#quick-start.

Sources:
Available programming environments
https://www.espressif.com/en/products/software/esp-sdk/overview
Official SDK getting started
https://www.espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_en.pdf
Bridge to Arduino reference to PlatformIO
https://github.com/esp8266/Arduino/blob/master/README.md
Installation of Arduino IDE with support of ESP8266 framework
https://arduino-esp8266.readthedocs.io/en/latest/installing.html

Source code description

Where to locate the source code for MCU could be found in Repository chapter.

The source code for microcontroller is using the folder structure according to PlatformIO IDE defaults, it is different from Arduino IDE defaults. In general all libraries could be located in lib subfolder, source code could be found in src subfolder. The main source file is named main.cpp, following code references are meant into this file.

Configuring the device

We made our sketch compatible with different MCUs - ESP8266 and ESP32. To configure for current needs open platformio.ini file and uncomment lines which corresponds to currently used device. For complete documentation of configuration files follow this link: https://docs.platformio.org/en/latest/projectconf.html#projectconf.

General functionality of our sketch is the following:
- Connect to Wi-Fi network,
- possibly launch server for Modbus and connect to MQTT server,
- start receiving data from sensors, interpret this data,
- transmit them using communications protocols.

We also make a sketch universal as much as possible, that means it is possible to select communication protocol and the sensor/actuator.
Currently supported communication protocols are:

- Modbus over TCP/IP,
- MQTT.

Currently supported sensors are:

- thermometer DS18B20,
- rotary encoder LDP3806.

This configuration could be handled on first lines of main.cpp file by simple commenting/uncommenting those lines:

```
// Configuration of devices in use. Comment appropriate defines, when the device is not in use.
#define DEVICE_ROTARY_ENCODER
#define DEVICE_THERMOMETER_DS18B20

// Configuration of communication protocols in use. Comment appropriate defines, when the protocol is not in use.
#define COMMUNICATION_MODBUS
#define COMMUNICATION_MQTT
```

This particular configuration will lead to using both devices and protocols. If any of sensor is not properly attached or any of protocols is not properly configured, then the whole process might not work properly.

The following subchapters will be concerned about details of implementing concrete devices and protocols.

Wi-Fi

Connection to Wi-Fi network is meant to work in client mode to a certain access point with obtaining an IP address from DHCP server.
For proper usage the following variables should be configured:

```
const char* ssid = "";
const char* password = "";
```

SSID stands for wireless network name and password for a key to network. We made all test with WPA2 encryption and we recommend to use this one.

The libraries for Wi-Fi are different between ESP32 and ESP8266, but luckily they share the same API.

After successful connection to a Wi-Fi network the MCU will print obtained IP address to a serial line. In purpose of stable IP address we recommend to add MCU into static map list in the DHCP configuration on router/access point.

For more about Wi-Fi configuration on ESP8266 this link could be followed: https://arduino-esp8266.readthedocs.io/en/2.5.0-beta2/esp8266wifi/readme.html.

Thermometer DS18B20

The default wiring could be found in repository. This thermometer could be powered in two modes - parasite power mode or external supply power mode. Our sketch and wiring scheme are in external power supply mode. The number of data pin is configured on those lines in sketch. Please mind as comment says that on these MCUs a digital port number, which is printed on PCB may not match a GPIO port number, which should be written to the sketch. Look into pinout schematics for concrete board.

```
// Data wire is plugged into port D2, whis is equal to GPIO 4
const int oneWireBusPin = 4;
```

This sensor has configurable resolution from 9 up to 12 bits. When increasing the resolution the conversion time increases significantly. The resolution could be configured there:

```
// Resolution of the thermometer could be in range 9 - 12
const int thermometerDS18B20Resolution = 9;
```

The library gives us a possibility to wait for conversion with blocking the processor thread or with non-blocking scenario, which we use. We just periodically ask for temperature conversion and when the conversion is done, we just pick it up. For more look into examples in Arduino Temperature Control Library, which could be located in lib subfolder.

Source: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Rotary Encoder LDP3806

For rotary encoder we attach channel A and B to digital pins of MCU, the wiring scheme could be found in repository. The source code could be found in file RotaryEncoder.h, following code references are meant into this file. Whole code is inspired by document published on http://www.vikipedialabs.com/learn.html.

Both channels should be attached to MCU pins with internal pull-up resistor and interrupt capability. For discovering this capability see MCUs documentation. To define the GPIO number for each channel edit following variables.

```
// specify GPIO pins, to which are channel A and B plugged in
const int encoder0PinA  = 25;
const int encoder0PinB  = 26;
```

We store the rotation to a variable of type long. When the movement in clockwise direction is recognized the value is increased by one, for the movement in counter-clockwise orientation the value is decreased by one.

MQTT

This protocol is implemented as a client to a MQTT broker. In current configuration it publishes all data periodically, which is not the best way, because the aim of this protocol is to push data only when they are needed.

You have to configure the broker address to the sketch in main.cpp.

```
// Adjust the mqtt server adress
const char* mqtt_server = "192.168.1.127";
```

Default port of broker is 1883. Communication is not encrypted. It is also good to adjust topics, which are used for appropriate devices.

```
#ifdef DEVICE_THERMOMETER_DS18B20
// Adjust mqtt topics
const char* mqttTopicDS18B20Encoder = "/Temperature/";
#endif
#ifdef DEVICE_ROTARY_ENCODER
const char* mqttTopicRotaryEncoder = "/RotaryEncoder/";
#endif
```

For more look into examples in PubSubClient library, which could be located in lib subfolder.

Modbus

This protocol creates on MCU a small server on port 502 and make it available to any possible client. We make data available in Holding registers. To configure on which address will be data available edit following lines.

```
// Modbus Registers Offsets (0-9999)
const int RotaryEncoderRegister = 0;
const int TemperatureSensorDS18B20Register = 0;
```

To update data in register we use following function.

```
mb.Hreg(RotaryEncoderRegister, RotaryEncoder::encoderPos.asInt[0]);
```

Because both types for temperature sensor and rotary encoder use as a storage 32-bit variables and modbus registers are 16-bit we have to use union types to access the same memory space with different variables type. There is example for temperature sensor.

```
// In purpose of sending data over modbus we introduce this union as a data
storage.
union TemperatureDataStorage
{
 float dataAsFloat;
 uint16_t dataAsInt[sizeof(float)/2];
}temperatureDS18B20;
```

For more look into examples in modbus-esp8266 library, which could be located in lib subfolder.

4. Repository

Source code and all other support files could be found at:
https://github.com/petrVavra/Industrial-Project-Work---Service-prediction.

The structure of source code is the following:

- MicrocotrollerSketch
  - sketch written in PlatformIO IDE handling the MCU level,
- Codesys
  - sketch written in Codesys for Raspberry PI,
- Node-Red
  - flow for Node-Red
- documentation
  - includes a link to this documentation,
  - includes circuit schematics.