# Automated train track -
# development of features.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Automation Engineering Valkeakoski

Spring 2019

Pavel Kulaev

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Automation Engineering
Valkeakoski

**Author** Pavel Kulaev **Year** 2019

**Subject** Automated train track: renewal and development of features.

**Supervisor(s)** Juha Sarkula

ABSTRACT

The thesis was commissioned by the Automation Engineering Degree Programme at Häme University of Applied Sciences Valkeakoski. The aim of the project was to renew old features and to develop new features for the train track model table present at HAMK Valkeakoski. A renewal of features was necessary for the use of the table by future automation students at the university. The topic for the project was provided by Juha Sarkula at Häme University of Applied Sciences Valkeakoski.

The main part of the project was formed by an introduction of new features through coding and by looking for solutions for an improved tracking of trains moving on the train track table. The difference in price and complexity of these solutions were taken into account. Many possible tracking solutions were examined. Other improvements for the table were discussed as well.

A new solution was developed to meet the requirements of the commissioning party. A program was written for the train track table which allowed the selection of different trains as well as a manual and an automatic movement of a train. Necessary safety features were implemented. Project goals were met in a satisfactory fashion.

**Keywords** Electrical engineering, automation, PLC, locomotive

**Pages** 65 pages including appendices 28 pages

CONTENTS

LIST OF FIGURES

# 1 INTRODUCTION

## 1.1 Project description

The first part of the thesis describes the objectives of the project, the equipment available at the start of the project and problems that had to be solved in order to meet the project goals.

The second part is the theoretical part of the project. It describes in detail the system that was used for the project: communication method between the trains and the controllers, operation of trains, methods of train tracking present at the beginning of the project.

Chapter three showcases the solutions that were researched and discussed as part of the project but were rejected as the main solution. Chapter four is used to showcase the accepted solution of the problem as well as new additions to the system. Chapter five describes the resulting state of the project. Part six talks about setbacks and unexpected issues encountered during the project.

## 1.2 Objectives

The project was provided by Juha Sarkula from Häme University of Applied Sciences (later referred to as HAMK) Valkeakoski. One of the main objectives of the project was the renewal of the train track table system at HAMK Valkeakoski. The system present at the beginning of the project was supposed to be used for practice for the automation students at the university however, multiple issues made the use of the table impractical. The project was started to solve those issues and to add additional functionality to improve the systems potential for practice-based exercises at campus. The thesis also touched different methods for small scale object tracking and use of such systems in combination with modern automation solutions.

After consideration the flowing objectives were established:

- Research the operation of the train track table.
- Develop a new tracking system for the trains or introduce improvements to the existent system.
- Develop new features.
- Write a program to showcase the capabilities of the train track installation.
- Create a robust user interface for the PLC

## 1.3 **Train track table**

A train track table present at HAMK Valkeakoski is a table with miniature track, switch rails, magnetic sensors and a programmable logic controller. The dimensions of the train track table are 404x147 cm. Serial communication was used to send commands to the Twin-Center device which in turn encoded the commands and send them to the train's decoder. Multiple trains could operate on the track at the same time, but no solution was in place to prevent train collision. However, this train track table had a great potential to be used for practical exercises on automation but was riddled with a magnitude of issues. The magnetic sensors were installed in pairs with equal distance between pairs. Because of the position of the magnetic sensors, the actual location of the trains was hard to track, and it was impossible to stop the train at the precise moment it reached the end of the track.

## 1.4 **Issues with tracking**

There are some viable options available for tracking on a small scale.
Plenty of solutions are either unreliable or economically unfeasible. Some companies provide finished solutions for this specific problem, but these solutions were too expensive, and the use of complete solutions went against the spirit of a thesis project. The ways in which small scale tracking can be implemented was examined here and these methods are included into the second chapter of the thesis.

# 2 THEORY

## 2.1 Scale models and their uses

Figure 1. Train track scale model (Railroad Modelling, 2015).

A scale model, as seen in Figure 1, is a model that represents interactions between objects and the world on a smaller scale. Usually only particularly important properties of the objects are translated to the scale model.

### 2.1.1 Practical uses of scale models

Scale models can be used for simulation purposes. Specifically, in the aerospace industry such models are used to test the aerodynamic properties of aeroplane designs. Furthermore, educational institutions often possess scale models for the purpose of showcasing on practice the operations of machinery without the need of the students to visit real life installations. HAMK Valkeakoski possess multiple such models.

### 2.1.2 Use of scaled models in studying automation

At HAMK one can find many examples of scale models. These models allow students to study automation practices on a smaller scale, without inconveniences such as size, price or maintenance which come with the real object.

## 2.2 **Rail transport modelling**

### 2.2.1 History of railroad modelling

According to the NMRA National Model Railroad Association railroad modelling can help with developing planning, development and engineering skills (Why Model? 2015).

The First miniature trains were made in Germany. These models were made of tin and lacked methods of locomotion. The first steam powered train model is credited to Sir Henry Wood from England. Such models were known as “Dribblers”. Later clockwork mechanisms were used in the creation of moving train models. Theodore Marklin, a toy maker from Germany is credited for the production of Europe’s first electrical train set. He also developed the first figure-eight layout (Coulter & Coulter n.d.).

Train modelling received recognition as a popular hobby only after WW2 then technology behind model trains became cheaper and easily available to the general public. Today, international train modelling communities exist that help bring more people into the hobby and serve as hubs for aspiring train modelers to receive guidance on their hobby. Through history many great people were model railroaders such: Winston Churchill, Walt Disney, Frank Sinatra (Why Model? 2015).

### 2.2.2 Powered railroad models

Today, train models are usually powered by low voltage electricity received from the track. First rail models were powered by steam and had very little control functionality. Later clockwork was used to power model trains. These models were controlled by levers on the train itself. (Railroad Modelling, 2015).

First electrical models were used three rails with the middle track providing power to the locomotive through a skid below the model. That system was used because many modern materials were unavailable so most of the model parts were conductive which led to a variety of problems (Railroad Modelling, 2015).

With increasing importance of model accuracy, a new system was adopted, which used only two rails: one providing positive supply and one negative. DC is the most common power supply today but sometimes AC power is also used (Railroad Modelling, 2015).

At first electrical train models used battery power as few houses were wired for electricity at the time. Today battery powered trains are often considered toys by the railway modelling enthusiasts ("Railroad Modelling", 2015).

### 2.2.3 Control systems

The first models which ran on clockwork or steam had no way to change speed or to stop the locomotive during operation thus the models ran until they were out of power. Appearance of electric trains allowed to change the models speed and direction. AC powered models had a mechanism which allowed to change direction when the power was cycled. DC powered models can reverse polarity for the same effect. A model layout can be divided into blocks which allows to operate multiple trains at the same time. Many modern layouts are controlled digitally and use PLC's (Railroad Modelling, 2015).

## 2.3 **Small scale object tracking on a surface**

Tracking objects orientation in space is a relatively easy endeavour - you can find an accelerometer in many modern smartphones. Things get complicated then the position of a small-scale object needs to be tracked in space with adequate precision. When detecting miniature locomotives on a track National Model Railroad Association (NMRA) provides some possible solutions.

### 2.3.1 Optical Detection

Optical detection is used in many applications. Such system can be independent from the track power and locomotive control systems. It can also consistently detect the moment the locomotive crosses the sensor. Doesn't require modifications to be made to the locomotives. That said, the sensor can only detect position when the train crosses it and the detector parts must be aligned to function (Train Detection, 2019).

LIGHT DEPENDENT RESISTOR

In the past, the most used miniature train detecting method used Light Dependent Resistors (LDR). It is a simple part that changes its resistance when exposed to light. To detect locomotives a LED light would be positioned on one side of the track with a LDR on the other side. When the locomotive passes that part of the track it will break the line of sight between the LDR and the LED. Of course, such sensor can be ineffective if heavy light pollution is present (Train Detection, 2019).

PHOTO TRANSISTORS AND IR DETECTOR PAIRS

An IR transmitter and a receiver can be used similarly to the LDR and detect the train, then the train passes between them. This solution is more resistant to light pollution than LDR. The IR detector pair is more precise and has a much longer effective range (Train Detection, 2019).

### 2.3.2 Conductive Detection

Another way of train detection uses the changes in current on different parts of the track to determine locomotive location. It only allows for block detection: detection of which section of the track the rolling stock is in. This method, however, is independent from the light pollution and lacks the requirement of complicated positioning. The current consumption is also higher, and the track must be separated into isolated parts (Train Detection, 2019).

VOLTAGE DROP

A section of the track is connected to power supply through a Diode. When a voltage drop occurs, diode detects current flow in the track thus detection which section of the track the train is in. This solution has one fatal flaw. If the Diode fails an entire section of the track will stop to function (Train Detection, 2019).

CURRENT TRANSFORMER

Power to the track passes through a Current Transformer and a detection circuit is used to measure generated current. This solution does not cause a voltage drop, produces less noise (Train Detection, 2019).

HALL EFFECT CURRENT SENSOR

Power passes through a Hall Effect sensor. A detection circuit is used to measure the response of the sensor. Just like the current transformer method, HE does not cause voltage drop. This sensor is more sensitive to current (Train Detection, 2019).

### 2.3.3 Proximity Detection

Metallic or ultrasonic sensors can be used to detect the locomotive passing over or close to the sensor. Such solution could work independently from the track power and has low amount of errors, although has its own limitations. For example, metallic sensors will have trouble with detecting plastic locomotives unless additions are made for better detection (Train Detection, 2019).

### 2.3.4 Reed sensors

The type of system present at HAMK Valkeakoski at the beginning of the thesis project. The Switches are activated by the magnet on the train. The switches themselves are positioned either between the tracks, under the train or on the sides of the track (Train Detection, 2019).

## 2.4 **Programmable Logic Controller**

A programmable logic controller (PLC) is a device based on a microprocessor that allows engineers and programmers to implement engineering solutions. Programmable memory is used to store instructions. PLC are programmed using specialized control programs that make use of simplified programming languages. This allows engineers with limited programming experience to carry out engineering tasks (Bolton, 2009).

### 2.4.1 History

PLC have originated from relay control systems. Before PLC all automatic control was done by combination of switches, clocks, relays and counters. A lot of complicated wiring was required, and such solutions required a lot of space to accommodate the relays. Implementation of such control systems takes a lot of time. If a change must be introduced the entire physical unit of relays must be rewired. Since relays were used the faults and operational interruptions were common (Bolton, 2009). Two examples of the relays can be seen in Figure 2.

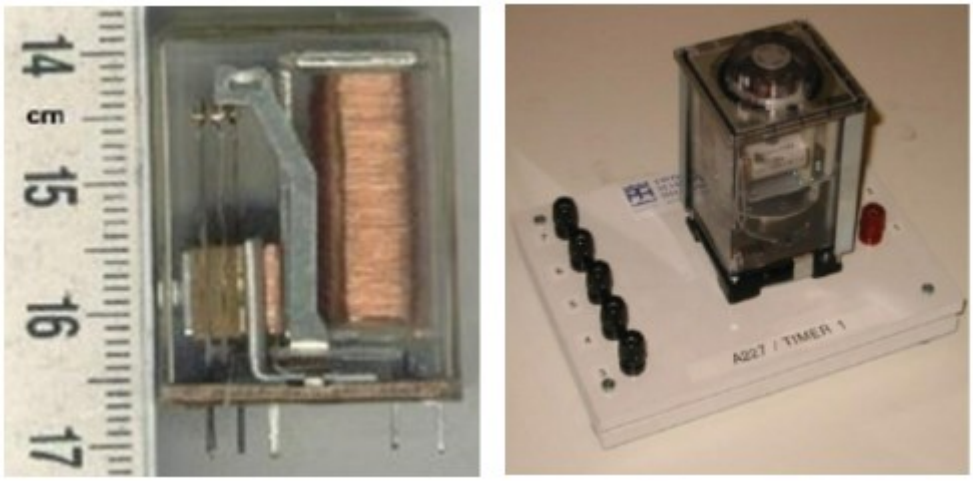


Figure 2. Relay example (Bolton, 2009).

When General Motors were in need for a replacement for relay control systems, the first PLC was developed. They wanted a software-based replacement for the older relay systems. The new solution had to be easy to change, program and had to be astronomically viable. The company to develop the first PLC was Bedford Associates, Inc. Based in Bedford, Massachusetts. It was called MODICON (modular digital controller). The controller used LD programming language, which had similarities with previously used relay diagrams (Bolton, 2009).

When the microprocessors became available in the early 1970s the development of programmable logic controllers continued with renewed vigour. Today, PLC include a variety of premade functions and

development tools. Around 1983 Modbus system was developed by Modicon. This system allowed for efficient communication between multiple PLC units and the ability to position the PLC further from production (Bolton, 2009).

Before the IEC 61131-3 standard was taken into use, the has been a lot of differences between different PLCs made by different PLC producers thus there was a large difference in implemented programming languages. Implementation of IEC 61131-3 finally allowed for greater compatibility between systems (Bolton, 2009).

### 2.4.2 Structure

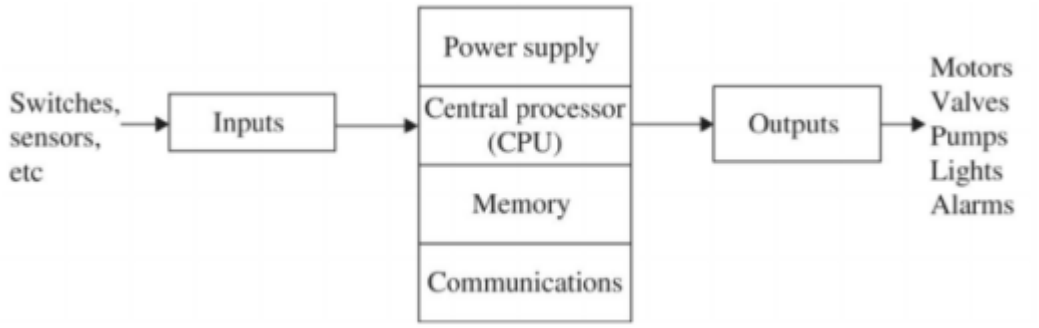


Figure 3. PLC structure (Bolton, 2009).

Structure of a PLC can be seen in Figure 3. Below, is a detailed explanation of the structure.

BUSES

A bus is a combination of wires, which is used for communication between parts of a PLC. Usually PLC have a control, system, address and data buses (Bolton, 2009).

CPU

CPU (Central Processing Unit) carries out calculations and controls how the program operates (Bolton, 2009).

MEMORY

There are three types of memory used by the modern PLC:

- ROM (Read-Only Memory) is used to store system data. EPROM (Erasable Programmable Read-Only Memory) is used then it is important to be able to update the PLC software since the standard ROM data cannot be deleted.
- RAM (Random Access Memory) is used to store programs currently used, since it allows for fast calculations and memory access. A

battery is used to prevent the loss of information from RAM, since it cannot keep data without current.
- Sometimes memory cards can be added to increase available memory (Bolton, 2009).

COMMUNICATION UNIT

Communication unit contains the communication protocols. Some PLC manufacturers provide their own communication solutions for example PROFIBUS, CANbus. More and more manufactures offer solutions for Ethernet protocol which allows for increased speed and low response time (Bolton, 2009).

POWER SUPPLY

PLC manufacturers produce their own power supplies, usually in the form of modules. Usually manufacturers produce 24 V DC, 220 V AC and 120 V AC power supplies however, PLC electronics run at 5 V. PLCs often have 24 V output for powering sensors (Bolton, , 2009)

I/O

I/O blocks allow for PLCs to receive and transmit signals. Inputs and outputs have their own addresses in the system. Many PLC available can process signals on their own (Bolton, 2009).

INPUTS

Although 5 V is the typical PLC voltage input signals use 24 V DC. To protect the PLC optical couplers are used.

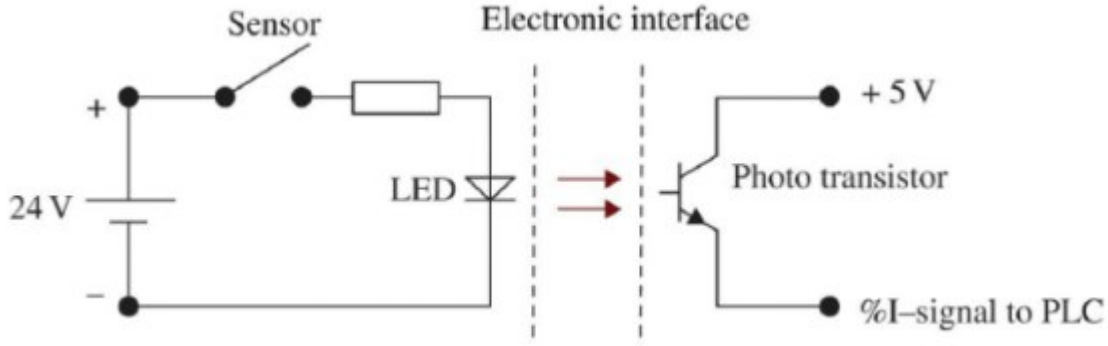


Figure 4. Coupler (Bolton, 2009).

A coupler is made from a LED and a phototransistor. When the sensor signal is HIGH the LED lights up, the light travels to the photo transistor, which triggers HIGH signal in the 5 V circuit. For analog inputs analog to digital converters are used. The signal is converted into binary values.

Usually 16 bit are used (Bolton, 2009). Coupler operation can be seen in Figure 4.

OUTPUTS

There are three main output module versions: relay, triac and transistor outputs.

The least used type is the Triac outputs. They are used if fast AC switching is required. They must be sufficiently protected from overcurrent.

Relay Outputs can be used with both DC and AC at different voltages. This method allows the PLC and the external circuits to be isolated. Then CPU changes the output signal, an external circuit is closed completing the connection (Bolton, 2009). An example of a relay output can be seen in Figure 5.

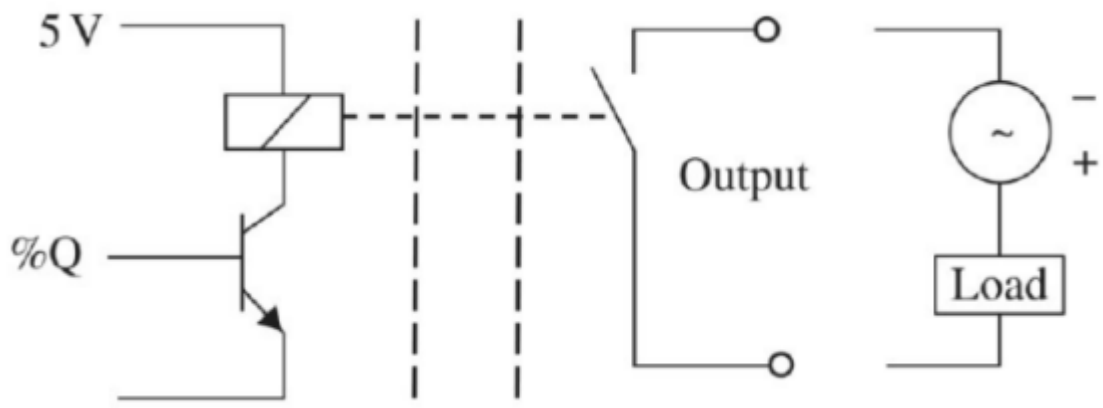


Figure 5. Relay output (Bolton, Programmable Logic Controllers, 2009).

Another type of outputs is the Transistor output as seen in Figure 6. Such outputs cost less than the other types. A transistor is used on order to complete an external circuit. Transistor outputs are faster at switching then the relays. Such outputs can only operate with DC. Fuses must be used to protect these outputs from overloads and wrong polarity (Bolton, 2009).

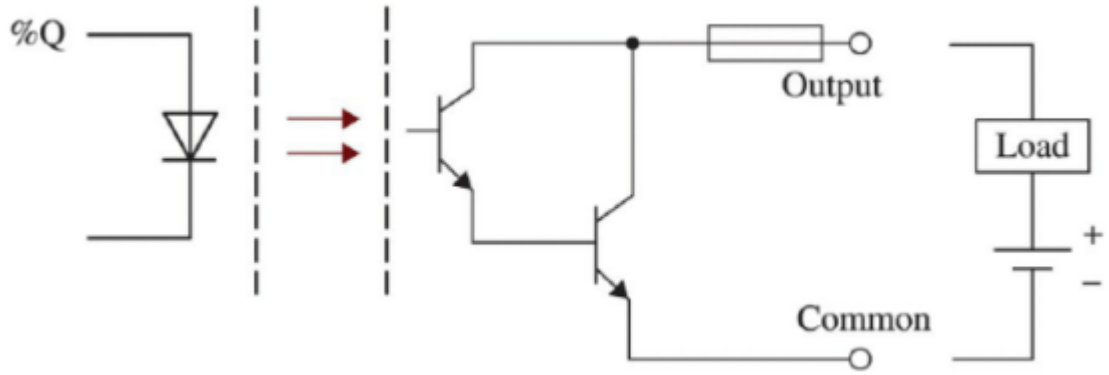


Figure 6. Transistor output (Bolton, Programmable Logic Controllers, 2009).

### 2.4.3 Principle of Operation

PLC operation is very similar to that of a typical PC. One of the main differences is that the PLC is mostly used to control and monitor the process. The PLC must be programmed in order for it to carry out commands. Processes are divided can be divided into different states. Process type dictates the required sensor types (Bolton, 2009).

As seen in Figure 7, during a normal operation of the PLC there are 4 main operations performed contentiously: Internal processing, Input reading, program execution and output update (Bolton, 2009).

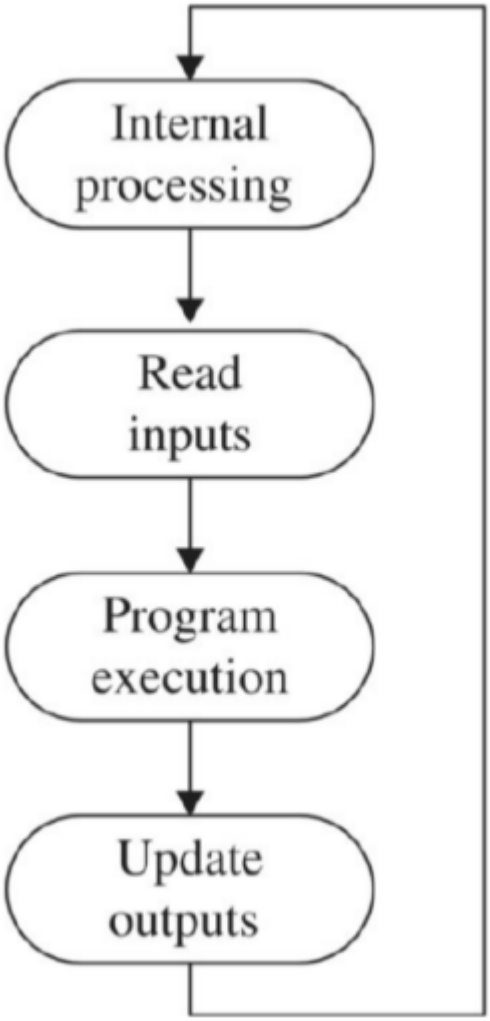


Figure 7. Main PLC operations (Bolton, 2009).

## 2.5 **TwinCat Software**

TwinCat by Beckhoff is a software system that can be used to turn a PC into a real time control system for automation. TwinCat software applicable to most automation solutions. User has access to an array of different programming languages. TwinCat 3 supports the modern advancements in automation and modular structure (The Windows Control and Automation Technology n.d.). This is the software used for the purpose of this thesis.

### 2.5.1 TwinCat XAE

eXtended Automation Engineering (XAE) brings the twincat integration into the Microsoft Visual Studio. Microsoft Visual Studio brings to the table a future-proof platform with plentiful expansion possibilities. A single tool is needed for the development of the software for a solution. This greatly simplifies the process of software engineering (eXtended Automation Engineering XAE n.d.). You can see a detailed comparison between integrated and standard TwinCat software in Figure 8.

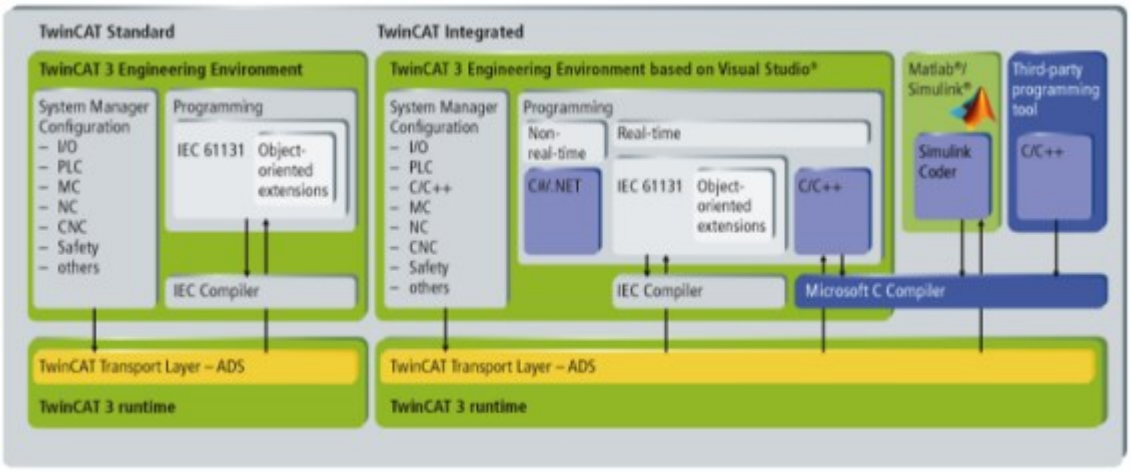


Figure 8. TwinCat operation ("eXtended Automation Engineering XAE").

### 2.5.2 Advantages

TwinCat software offers a number of useful features:

- TwinCat software can be used for both programming and configuration.
- TwinCat is integrated into the Microsoft Visual Studio environment.
- TwinCat systems execute control programs in real-time.
- Logical and sophisticated PC interface.
- Expansive function libraries.
- Easy I/O management.
- Supports both textual and graphical languages such as Structured Text, Function Block Diagram, Ladder Diagram etc (The Windows Control and Automation Technology n.d.).

### 2.5.3 Main functions of TwinCat

Following are the main functions of TwinCat software:

- Automation Device Specification (ADS) is a protocol that controls data transmission, writing and reading of data within TwinCat.
- TwinCat I/O allows for data collection from different fieldbuses. It is possible to use different cyclic task for different fieldbuses.
- TwinCat PLC allows for multiple PLCs to be realized on one CPU. “PROGRAM” type files can be connected to tasks
- TwinCat C++ allows for the execution of programs written in C++ (The Windows Control and Automation Technology n.d.).

## 2.6 **Serial Communication and RS-232 protocol**

### 2.6.1 Serial communication

Two main methods exist for data communication between devices: parallel and serial. Serial communication sends data one bit at a time and thus requires only one connection. This however leads to serial communication being slower than the parallel communication and should only be used if data transfer speed is inconsequential (Blom. n.d.).

### 2.6.2 Baud rate

Baud rate is the speed at which the data is sent through the serial communication. The measuring unit of baud rate is bit per second (bps). For serial communication to function both connected devices must operate at the same baud rate.

### 2.6.3 Data framing

Data sent through the serial communication is organized into a frame consisting of bits. The frame consists of synchronization bits, data bits and parity bits as seen in Figure 9. The main part of this frame is the data being send. It can range from 6 bits to 9 bits. Bits at the start and the end of the frame are called synchronization bits. They are used to recognize the beginning and the start of the frame. Finally, the parity bits are used for simple error handling (Blom, n.d.)


Figure 9. Serial communication message frame (Blom. n.d.).

### 2.6.4 Wiring

Serial communication makes use of two connections Receiver (RX) and a transmitter (TX). TX from one device is connected to the RX from another device.

### 2.6.5 Beckhoff implementation

KL6001 Terminal from beckhoff was used to implement serial communication for the train track installation described in this thesis. This terminal allows for serial communication to be established with RS-232 protocol devices. Beckhoff Twincat 3 Software includes a download-able

serial communication library that is used to set up serial communication through the KL6001 terminal.

### 2.6.6 RS-232 protocol

RS-232 allows for serial binary data exchange between devices through connection between data terminals. It initially was used for communication using telephone lines. A modem can convert digital data into a form that can be transmitted over a telephone line. Today the RS-232 protocol is widely used for communication between PC's and compatible devices (The RS-232 protocol, n.d.).

RS-232 Low signal indicate a value 1 or a stop bit while the High signal indicates data value of 0. Example of this signal can be seen in Figure 10.

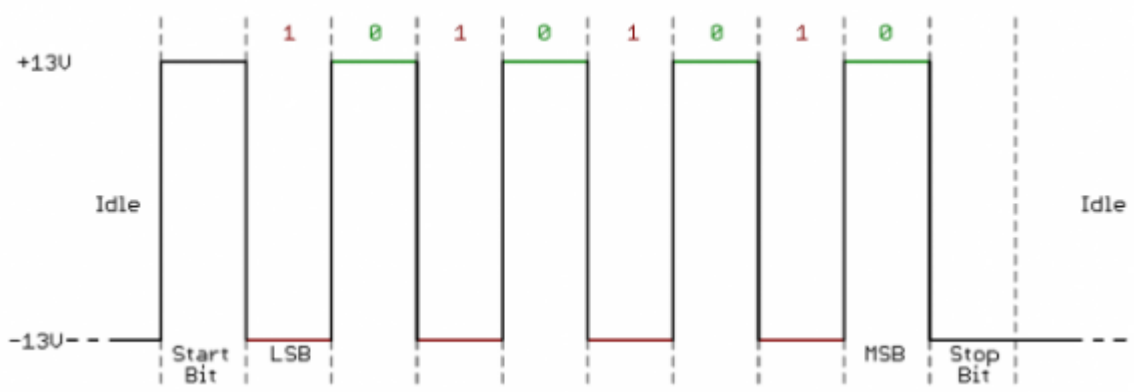


Figure 10. RS-232 frame (Blom, n.d.).

## 2.7 **RFID**

### 2.7.1 History of RFID

First roots of RFID started in the beginning of the 20th century. Soviet physicist Lev Theremin (Leon Theremin) is considered to have created the first simple RFID device. Reflected radio frequency was modulated by a resonator, which in turn, changed shape by the sound waves vibrating a diaphragm. Although it was not an id tag it is widely considered to be a predecessor to the RFID technology (History of RFID n.d.).

That said, the technology used in the creation of RFID has existed since 1935 when the Scottish physicist Sir Robert Alexander Watson-Watt developed radar technology. During World War 2 German pilots began changing the roll angle of their airplanes before returning to base in order to change the reflected radio signal to identify themselves as friendly. An identify friend or foe (IFF) system was then developed for the British military by a research team under Watson-Watt. Every British plane had a transmitter that, when it received a signal from the radar system, would

broadcast a signal back, identifying aircraft as friendly. Modern RFID uses similar basic concept.

The first US RFID patent was received by Mario W. Cardullo on January 23, 1973. The patent was for an active RFID tag that had modifiable memory. It is considered that US government has been working on the RFID systems in secret before that (RFID History n.d.).

### 2.7.2 Working principle

Generally, RFID (Radio Frequency identification) refers to any form of radio communications used for identification. Usually RFID uses a tag or a label on an object. After the tag receives a signal it then returns identification information back to the transmitter. Of course, the system may be much more complicated than that (Thornton & Lanthem, 2006).

### 2.7.3 Types of RFID

Modern RFID system contains a tag (transponder) and a reader (receiver). The reader sends a signal to the tag, receives information and either sends it somewhere else or Internets it and shows it on the interface display. A tag (transponder) can both receive and transmit data. Some types of transponders can do calculations before sending the signal back, some can only receive data and send its own data in reply. Usually an RFID tag contains power supply, an antenna, encoder/decoder and communication control (Thornton & Lanthem, 2006). There two types of RFID: active and passive. Comparison between these RFID types can be seen in Figure 11.

An active tag has power supply, often in a form of a battery. Such tags can both receive and transmit data without the help of the readers antenna. The operating range of such tags is longer then passive tags. This comes with bigger size of the tag and the requirement to change power supply then it runs out, if the tag lacks connection to the power supply (Thornton & Lanthem, 2006).

Unlike the active tag, the passive tag lacks its own power source thus the passive tag must obtain its power from the reader device. It is done using The Near Field principle. The Near Field provides power to the tag.
One issue with this is that the tag must be close to the reader since the passive tag does lacks a power source (Thornton & Lanthem, 2006).

When in radio transmission, electromagnetic is substantially strong to induce an electric field in a coil, such phenomenon is called The Near Field. The range of The Near Field can be calculated using this formula: $r=\lambda/2\pi$ where $\lambda$ is wavelength. If the passive tag is to function it needs to be within this distance from the antenna (Thornton & Lanthem, 2006).

There is also a number of semi-passive tags that are partly powered by the internal power and partly by The Near Field (Thornton & Lanthem, 2006).

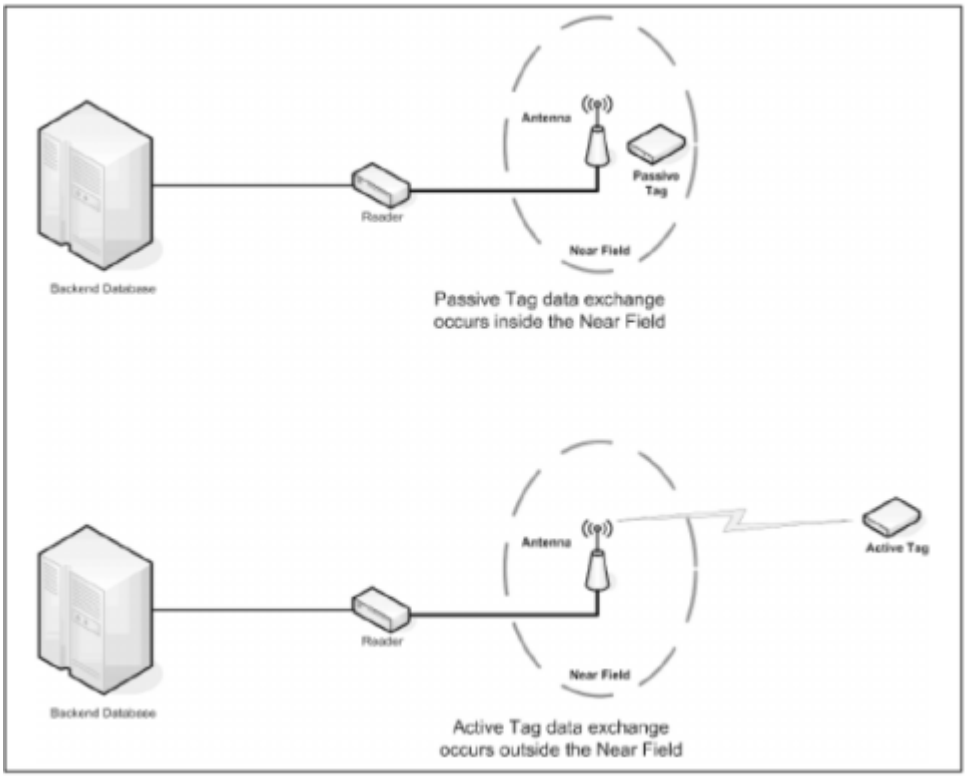


Figure 11. Example of passive and active RFID tags (Thornton & Lanthem, 2006).

### 2.7.4 RFID regulations

RF waves can travel substantially long distances and pass through solid objects. Because of this different RF devices can interfere with each over and at substantial power levels can even be harmful to living tissue (Thornton & Lanthem, 2006).

Countries usually have their own RF standards. Bringing all regions to the same standard can be difficult due to the use of older RF devices in different fields. Different standards allocate frequency range to different applications.

The recent regulation, EN 302-208 has the following features:

- A device should have a listen before talk functionality.
- Effective radiated power levels of 100 mW, 500 mW, and 2 W.
- Before the transmission starts the device must be in listening mode for 5 milliseconds.
- Continuous transmissions should be no longer than 4 seconds Transmission should be within 865-868 MHz.
- 100 ms should pass between transmissions on the same subband or device should transmit on a different frequency.

- Subbands should be at 200 kHz (Thornton & Lanthem, 2006).

### 2.7.5 Applications

RFID has moved a long way from its original uses such as spying and friend-foe identification. Today RFID is used in many peaceful fields such as logistics, inventory monitoring, material management, library systems, shopping, interactive marketing, farming, engineering and many others.

## 2.8 **Reed Switches**

One of the best methods of sensing proximity is the Reed Sensor. A permanent magnet is moving with the object, which proximity to the sensors must be observed. Such object can be a locomotive, a door, a piston etc. Many companies produce reed sensors for different specific operations. Such sensors include a reed switch which is affected by the magnetic field (Sensors & Magnets n.d.).

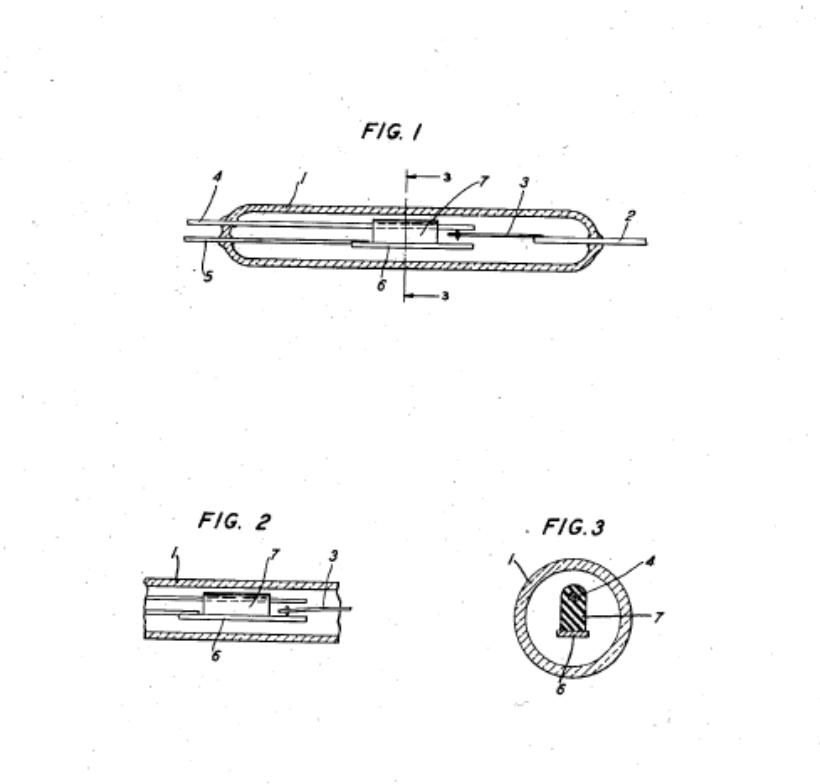


Figure 12. Electromagnetic switch patent (Ellwood, 1941).

Reed switch is a magnetically operated switch developed by Walter B Ellwood in 1940. In the patent it is stated that the electromagnetic switch is a reliable and easily replaceable contact device (Ellwood, 1941).

The switch itself consists of the electrical contacts made from magnetic material, sealed in the tube with either inert gas or vacuum. These

electrical contacts are positioned parallel to each-over, with a small overlap. A third non-magnetic contact is connected to one of the magnetic contacts. Then magnetic force is applied to the circuit, the two magnetic contacts are moved closer and one magnetic contact is moved from the non-magnetic contact to the second magnetic contact (Ellwood, 1941). Figure 12 shows the electromagnetic switch cross section as present in the original patent.

On figure 13 FIG. 1 object number 4 is the non-magnetic contact, object number 3 is the first magnetic contact and object number 6 is the second magnetic contact (Ellwood, 1941).

# 3 ALTERNATIVE SOLUTIONS

## 3.1 Ultrasonic sensor system

GamesOnTrack , a company specializing in indoor positioning systems has number of products that can allow for locomotive monitoring. Their positioning system consists of 3 “receivers”, which with the combination of radio communication between units and ultrasound, can provide the position of the “sender” unit that can be directly connected to the train supply. Their GT-Xconnect unit receives the information from the system and sends it to a PC, which using the provided software shown in Figure 13 can interpret the data into the position information. Although the system works and can, according to the company, detect the position of the locomotive with precision up to 10mm, its expensive and can only calculate the position using the provided software so, in order to use it with a PLC, a separate program must be created. In the end this solution is both expensive and more complicated the necessary thus it was rejected (Indoor GPS, our real time Positioning System n.d.).

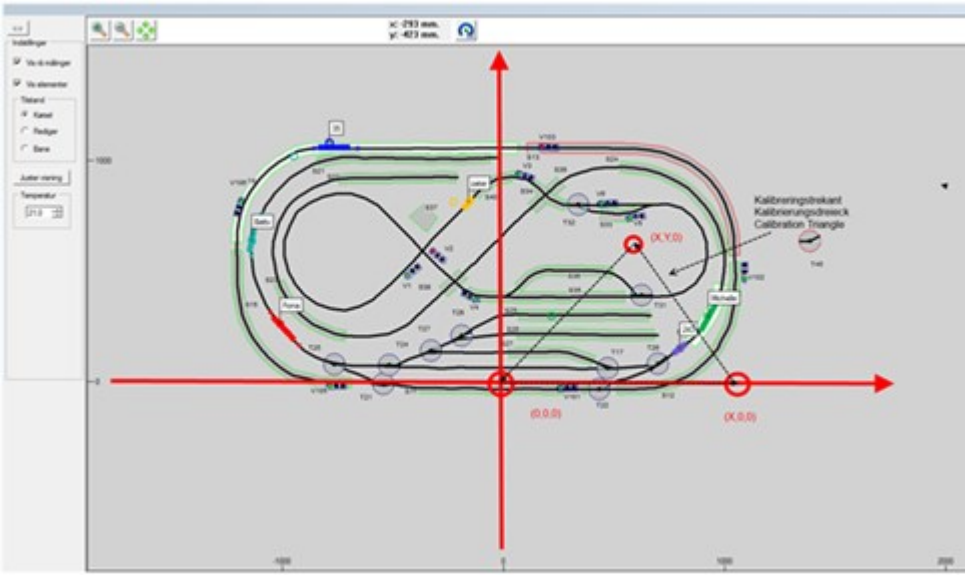

Figure 13. Positioning software (Indoor GPS, our real time Positioning System n.d.).

## 3.2 **RFID**

Another possible solution is to replace the magnetic sensors with RFID tags and readers. Passive RFID tags can be placed on the bottom of the train and the readers could be installed underneath the track.

Although, today passive RFID tags can be extremely cheap, some could cost below one USD, the transmitters can be quite expensive. In addition, small transmitters that can fit below the track have a severe lack of range. Comparing to the magnetic sensors, RFID system could distinguish between different locomotives.

That said, to even maintain coverage comparable to the magnetic sensor system, around 400 USD worth of RFID readers will be required, and substantial modifications need to be applied to the track table, requiring drilling the table. Combined with the lack of meaningful improvement on the existing system, this solution was deemed enviable.

## 3.3 **Increase in the number of magnetic sensors**

There is a simple way to increase the coverage of the monitoring system: increasing the number of sensors. However, there are multiple issues with that option. More I/modules must be added to the control box and there is only room for one more, two if one output module is replaced with an input module. This solution also requires drilling more holes into the table.

### 3.4 Re-positioning of available sensors

At the beginning of the project the magnetic switches are positioned in pairs on both sides of the track. That positioning was excessive and prevented accurate locomotive speed calculation. As an option the sensor pairs can be moved apart and positioned 2-3 cm from each over. This will allow to calculate the trains speed then it passes the gap.

### 3.5 Image processing

A camera can be positioned above the table and background subtraction technique can be used to find the position of the trains on the track.
This solution requires the camera to be positioned high enough in order to cover the entire table and the ceiling in HAMK Valkeakoski prevent the camera from being positioned high enough. Multiple cameras can potentially be used to remedy that, and the image combined using software. Although this technique can provide accurate locomotive position information it can be resource heavy and have a substantial delay.

## 4 FINAL SOLUTION

The train track was a low priority installation in HAMK thus more expensive solutions were unviable.

After consideration the “Re-positioning of the available sensors” became the accepted solution. Two sensors were re-positioned as a proof of concept, an automation program and visualization were created to showcase the features of the train track.

Before the process could take place, the necessary documentation was gathered and research was done on the basic operation principles of existing equipment, an outline of expected features was drafted.

### 4.1 Rail model in HAMK

Train model in HAMK Valkeakoski is an advanced scaled installation. It consists of the track table and the I/O box.

The table has the following properties:

- Three tables, 135 cm x146 cm each, connected together with a track on top.
- The distance between rails is 2 cm.
- There is a total of 46 ASSEMtech magnetic switches positioned on both sides of the track.
- Electric locomotive with a decoder.

- Twin-Center device.

The I/O box contains the following:

- EK1100 EtherCat coupler
- IMO DPS-1-060-24 Power supply
- Switches
- KL6001 serial interface
- KL1002 2-channel digital input terminal 24 V DC
- 10x KL1408 8-channel digital input terminal 24 V DC
- 6x KL2114 4-channel digital output terminal 24 V DC
- KL9010 End terminal

In addition, a Beckhoff based PLC CPU is used to process and store the program.

### 4.1.1 Control method

Fleischmann Twin-Center control unit is used to control the trains. As name suggests two trains can be controlled at the same time on the same track. Usually miniature trains are controlled using variable D.C. voltage. The higher the voltage the faster the motor runs. Because of that all trains on the track will move with the same speed at the same time. However, the Twin-Center uses digital train control so trains can operate independently. The track is under 20 volts A.C. voltage. It is both used to power the trains and to control the train systems. Commands are stored within the Twin-Center controller and are sent using digital impulses to the locomotives. Decoders within the trains receives the impulses and interpret the commands. Each piece of equipment connected to the system has its own address and will only carry out the commands directed at this address. Therefore, multiple trains can operate on the same track independently (Fleischmann Twin-Center 6802 Operating Instruction Manual, n.d.). The layout of the Twin-Center system can be seen in Figure 14.

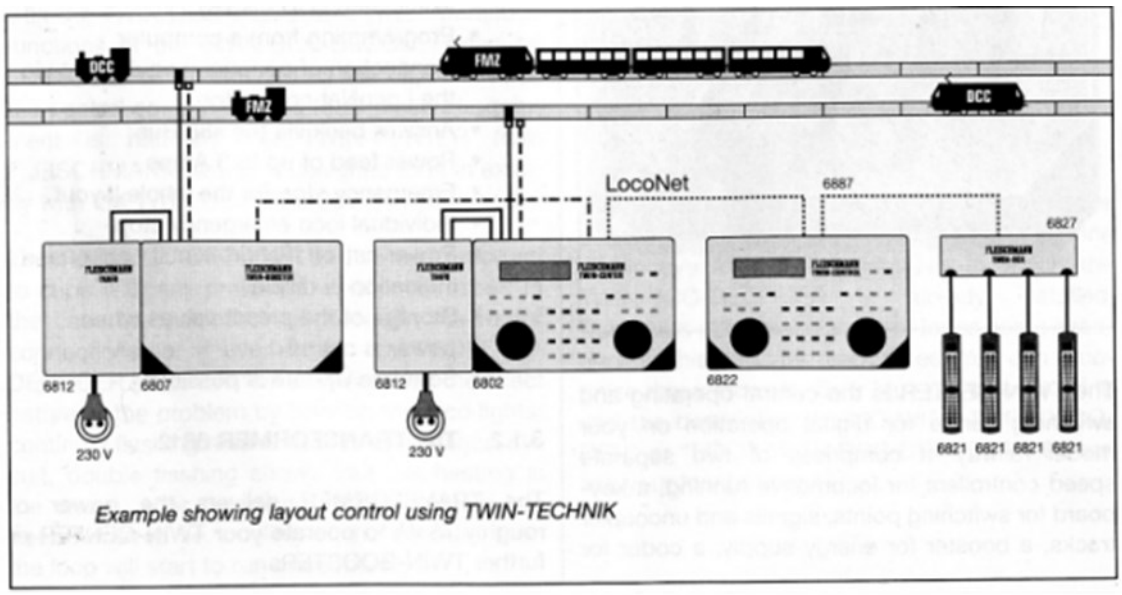


Figure 14. Twin-Center (Twin-Center Operating Instruction Manual, n.d.).

### 4.1.2 I/O box internals

The following are the descriptions of the I/O box terminals:

- EK1100 EtherCat coupler must be the first terminal.
- IMO DPS-1-060-24 is a DIN-rail power supply with one output operating at 24V/60W. Cooled by air convection.
- KL6001 serial interface terminal allows for communication with devices that use RS232 interface.
- KL1002 2-channel digital input terminal transmits signals to the PLC. It has 2 LED lights signifying the state of the input channels.
- KL1408 8-channel digital input terminal transmits signals to the PLC. It contains 8 channels and 8 LED lights signifying the state of the input channels. Power contacts are looped.
- KL2114 4-channel digital output terminal connects the PLC to the actuators. Similarly, to the input terminal it has LED lights showcasing the status of the outputs. This terminal contains 4 channels.
- Any terminal assembly must end with a KL9010 End terminal. It allows for the data exchange between the bus terminals and couplers.

Full I/O list can be found in Appendix 1.

### 4.1.3 Beckhoff PLC

To store and run the created program, a Beckhoff based PLC CP6606 was used. The above-mentioned PLC is shown if Figure 15 and is designed to be installed on the front of control cabinets. It can be used for a variety of engineering solutions. The PLC has an integrated seven-inch display with

touch screen. It makes use of an ARM Cortex-A8 1 GHz Processor. CP6606 is powered by 24 V DC power supply. Microsoft Windows Embedded Compact 7 is used as an operating system for the PLC.


Figure 15. Beckhoff based PLC

### 4.1.4 Present issues and challenges

Train track table came with a number of issues:

- All sensors were doubled on both sides of the track. This was unnecessary and lowered the sensor coverage.
- Because all sensor pairs were so far apart calculating the train speed accurately was impossible.
- A part of the track was not connected properly and might have caused a short circuit if the train passed over that part.

## 4.2 **Implementation**

Sensors 36 and 38 were re-positioned as the proof of concept for the system that can detect the speed of the moving train using the available sensors. Sensors were moved further apart allowing the program to use the distance between them and the system time to establish the speed. The repositioned sensors are shown in Figure 16.

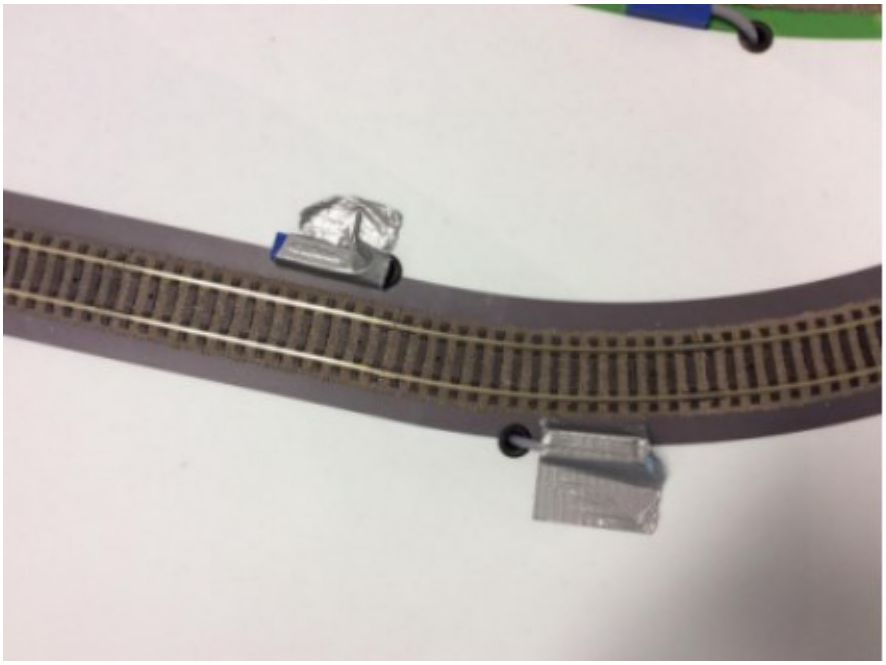

Figure 16. Repositioned pair of sensors.

Following that the program was written to showcase the capabilities of the train track as well as to add new features to the control method. The program now allows for the train to be operated manually or automatically. All switches can be operated separately.

## 4.3 **Advantages**

The solution has its drawbacks though there are also clear advantages.

- No extra installations are necessary. This solution uses the devices that are already part of the installation.
- No financial investment is needed.
- Since major changes to the table such as drilling or cutting were prohibited many solutions became non-viable. Re-positioning of the sensors requires no irreversible changes made to the table itself.

## 4.4 **Disadvantages**

Although there was a clear economical advantage of the above-mentioned method, there was also an array of issues and drawbacks:

- Each train possess two magnets which activate the reed switches. That needs to be taken into account during the coding process.
- Accurate position of the train could only be established at the location of the sensor.
- If the trains starting location was between the sensor and one of the two intersections, requiring a change of polarity, then the program would be incapable of changing the polarity unless the train crossed one of the sensors beforehand.

- Sensors detected the train with a slight delay which could lead to an overshoot.
- Due to the length of the sensor cables repositioning of the sensor was not possible everywhere thus only one pair of sensors was repositioned.

## 4.5 **Coding process**

The presentation code for the program consists of two program files, five function blocks and a visualization. Two PLC tasks are required for the handshake between the train control device and the PLC to function properly.

The following requirements for the program were established by the thesis supervisor:

- It must be possible for the operator to run the train manually using visualization. The operator must be able to change the speed of the train during operation.
- The train must be able to freely traverse the areas where the track polarity must be changed to prevent short circuit.
- The train speed that is acquired from the sensors must be displayed on the visualization screen.
- It must be possible to use visualization to switch between two trains that can be operated by the Twin-Center device.
- It must be possible to operate all the available switches using the visualization.
- An automated program must be written to allow for an automatic movement of the train between the train stations.
- Visualization must contain the information on the switch position.

Before the code could be implemented the location of the sensors and the switches had to be recorded. A schematic was created using a third-party program Simple Computer Aided Railway Modeller (SCARM). The schematic can be seen in Figure 17.

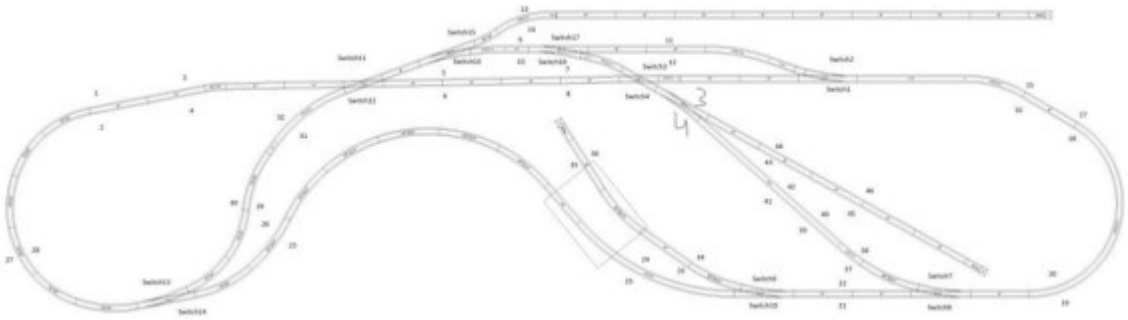

Figure 17. Track schematic.

Following that, part of the program responsible for manual operation was written and necessary safety features preventing die-railing added to the

solution. Thirdly, paths for the trains were programmed and visualization elements created for convenience of operation. Finally, additional features such as track intersection indicators were added.

### 4.5.1 Libraries used

The following libraries were used for the purpose of this project. Libraries with the Visu prefix are used for visualization elements used in the project. Tc2_SerialCom library contained the necessary functions and data types for the serial communication. The rest of the libraries contained functions used for the rest of the program. Figure 18 contains a list of references.

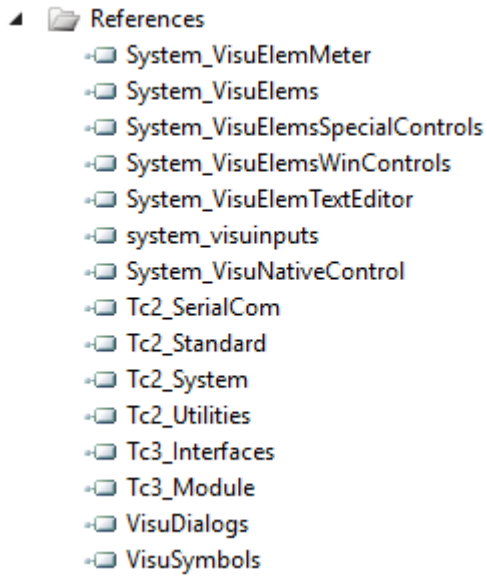


Figure 18. Used libraries.

### 4.5.2 Program and Function Blocks

In this chapter the most crucial parts of the code and their functions are presented. The “MAIN” program contains the function calls, most parts of the code responsible for the visualization and the code that switches the polarity of the track then necessary.

```
64  IF I_bTrain_Go THEN
65  GOTRAIN(
66      s_StringSpeed:=s_StringSpeed ,
67      i_StringTrainSelect:= ,
68      i_sDirection:=i_Direction ,
69      fbSend_Go:= ,
70      I_bTrain_Go:=I_bTrain_Go,
71      TxBuffer:=TxBufferKL ,
72      RxBuffer:=RxBufferKL);
73  END_IF
```

Figure 19. GO function call in MAIN.

In the above mentioned program the most important part is a function call responsible for the “GO” function block. This function call can be seen in figure 19. A Boolean variable “I_bTrain_Go” is set to FALSE after a single data frame is sent through the serial communication and is set to TRUE if the communication instruction changes. This ensures that duplicate frames are not sent as otherwise the important communications will be delayed.

The code shown in Figure 20 ensures that the train can move through the areas of the train track where the change in polarity is required to prevent short circuit.

```
IF Term5_7 OR Term5_3 OR Term7_1 OR Term8_2 OR Term6_8 OR Term5_6 THEN
    bSwitch19:=FALSE;
    bSwitch20:=FALSE;
    //b_pArea:=FALSE;
END_IF

IF Term7_3 OR Term9_6 OR Term9_8 OR Term9_3 OR Term7_6 OR Term4_8 OR Term4_5 OR Term4_2 OR Term9_7 OR Term5_2 THEN
    bSwitch19:=TRUE;
    bSwitch20:=TRUE;
END_IF
```

Figure 20. Polarity change.

The “SerialCom” program is responsible for the implementation of the serial communion between the PLC and the Twin-Center device. Figure 21 shows the standard functions used for serial communication with a necessary setting for the Twin-Center device.

```
fbKL6001Config(
    Execute:= bKL6001ConfigExe,
    Mode:= SERIALLINEMODE_KL6_5B_STANDARD,
    Baudrate:= 9600, //9600
    NoDatabits:= 8, //8
    Parity:= 0,
    Stopbits:= 2,
    Handshake:= 1, //HANDSHAKE_RTSCTS
    ContinousMode:= FALSE,
    pComIn:= ADR(stIn_KL6001),
    pComOut:= ADR(stOut_KL6001),
    SizeComIn:= SIZEOF(stIn_KL6001),
    Done=> ,
    Busy=> ,
    Error=> bKL6001ConfigError,
    ErrorId=> eKL6001ConfigErrorID );
IF NOT fbKL6001Config.Busy AND NOT bKL6001ConfigError THEN
    bKL6001ConfigExe := FALSE;
    fbKL6001Ctrl(
    Mode:= SERIALLINEMODE_KL6_5B_STANDARD,
    pComIn:= ADR(stIN_KL6001),
    pComOut:= ADR(stOut_KL6001),
    SizeComIn:= SIZEOF(stIn_KL6001),
    Error=> bKL6001CtrlError,
    ErrorID=> eKL6001CtrlErrorID,
    TxBuffer:= TxBufferKL,
    RxBuffer:= RxBufferKL);
END_IF
```

Figure 21. Serial communication configuration.

Function block “GO” is used for compiling and sending of instructions to the Twin-Center. The message must be five characters long. In Figure 22 the function which handles the sending of the data string can be seen.

```
//Here is the function block that sends a string to the train.
    stringSent := StringSelect;
    fbSend_Go(  SendString:= stringSent,
            TXbuffer:= TxBuffer,
            Busy=> bSendBusy_Go,
            Error=> eSendErrorID);
i_RealDir:= i_sDirection;
i_RealSpeed:=i_VisuSpeed;
I_bTrain_Go:=FALSE;
//This resets the send string command to make sure only one string is sent
```

Figure 22. Function for sending the data string to the Twin-Center device.

Message is sent to the decoder in the train. It uses special syntax to establish the address of the train, type of command and the speed. For example, in message “V3V07” character “V” establishes that this command will change the direction of the train. The opposite direction is set by

replacing characters “V” with “R”. Number “3” is the address of the train and “07” is the expected speed of the train. Due to settings selected in the Twin-Center “00” would mean full stop and “20” would be the maximum safe speed for the train.

Function block “Paths” is designed to control the automatic movement of the train between the stations. The paths a separated into the basic paths and the combined paths. Examples of these paths can be seen in Figures 23 and 24. Basic paths are the shortest possible paths between two stations. The combined paths are the paths that consist of multiple basic paths. Below are the examples of the basic path and the combined path.

```
IF b_aPath[1, 2] THEN
    i_Direction:= i_Direction2;
    IF b_aSwitchState[6] THEN
        b_aProgSwitch[6]:=TRUE;
    END_IF
    IF Term9_4 OR Term9_7 THEN
        i_Direction:= 0;
        b_aPath[1, 2]:= FALSE;
    END_IF
END_IF
```

Figure 23. Train path from station one to station two.

```
IF b_aPath[1, 3] THEN
    CASE i_aCase[0] OF
    0:
    b_aPath[1, 2]:=TRUE;
    i_aCase[0]:= i_aCase[0]+1;
    1:
    IF NOT b_aPath[1, 2] THEN
        b_aPath[2, 3]:=TRUE;
        i_aCase[0]:= i_aCase[0]+1;
    END_IF
    2:
    IF NOT b_aPath[2, 3] THEN
        b_aPath[1, 3]:=FALSE;
        i_aCase[0]:=0;
    END_IF
    END_CASE
END_IF
```

Figure 24. Train path from station one to station three.

In the Path 1-3 the train follows the path 1-2 first then continues using the path 2-3 once it reaches the station two.

Function block “StationN” detects the station at which the train has stopped. This information is used in the “Paths” function block and the visualization.

Function block “Switch” operates the rail switches and the two cross-rails. It is important that the switches are switched off after the track is operated because they are rated for lower operational voltage. In the current set up if the switches are left on, they will eventually burn out.

Two switches are used to operate one track intersection thus it is necessary for the program to remember the position of the track. Figure 25 contains the operational code for one track intersection.

```
IF b_aProgSwitch[1] THEN
    IF b_aSwitchState[1] THEN  // Switch 1 & 2
        CASE i_StepA[0] OF

        0:

        i_StepA[0]:=i_StepA[0]+1;

        1:
        i_StepA[0]:=i_StepA[0]+1;
        2:
        TP1(IN:= TRUE, PT:=T#500MS , Q=>bSwitch1  , ET=> );
        TON1(IN:= TRUE , PT:=T#500MS , Q=> b_CasA[0] , ET=> );
        IF b_CasA[0] THEN
        TP1(IN:= FALSE);
        TON1(IN:=FALSE);
        b_CasA[0]:=FALSE;
        i_StepA[0]:=i_StepA[0]+1;
        END_IF
        3:
        b_aSwitchState[1]:=FALSE;
        i_StepA[0]:=0;
        b_aProgSwitch[1]:=FALSE;

        END_CASE

    ELSE
        CASE i_StepA[0] OF

        0:

        i_StepA[0]:=i_StepA[0]+1;

        1:
        i_StepA[0]:=i_StepA[0]+1;
        2:
        TP1(IN:= TRUE, PT:=T#500MS , Q=> bSwitch2 , ET=> );
        TON1(IN:= TRUE , PT:=T#500MS , Q=> b_CasA[0] , ET=> );
        IF b_CasA[0] THEN
        TP1(IN:= FALSE);
        TON1(IN:=FALSE);
        b_CasA[0]:=FALSE;
        i_StepA[0]:=i_StepA[0]+1;
        END_IF
        3:
        b_aSwitchState[1]:=TRUE;
        i_StepA[0]:=0;
        b_aProgSwitch[1]:=FALSE;

        END_CASE

    END_IF
END_IF
```

Figure 25. Switch operation.

Since the position of the switches may change at the time that the PLC is off, code was introduced to allow the operator to reset the position of the switches as seen in the Figure 25.

```
5   IF BtR THEN
6       FOR i:=1 TO 9 DO
7           b_aSwitchState[i]:=TRUE;
8           IF b_aSwitchState[i] THEN
9           b_aProgSwitch[i]:=TRUE;
10          END_IF
11      END_FOR
12      b_SwitchR:=FALSE;
13  END_IF
```

Figure 26. Switch reset function.

Function block “LocalTime” collects the local system time and uses it in conjunction with the sensor data to establish the speed of the train. The solution also contains a single global variable list “IO”.

## 4.6 **Visualisation and operation**

Clear and easy to read interface is just as important as functionality of a solution. After a system is installed it will be operated by people with little to no contact to the engineers that installed the equipment. Thus, a lacking interface may lead to a severe productivity loss. A good interface must possess all necessary functions for a correct operation of a device or an installation while maintaining simplicity.

The creation of the visualization started with the schematic of the train track. It included the location of all the track intersections and sensors present on the track table. Full visualization can be seen in Figure 27.

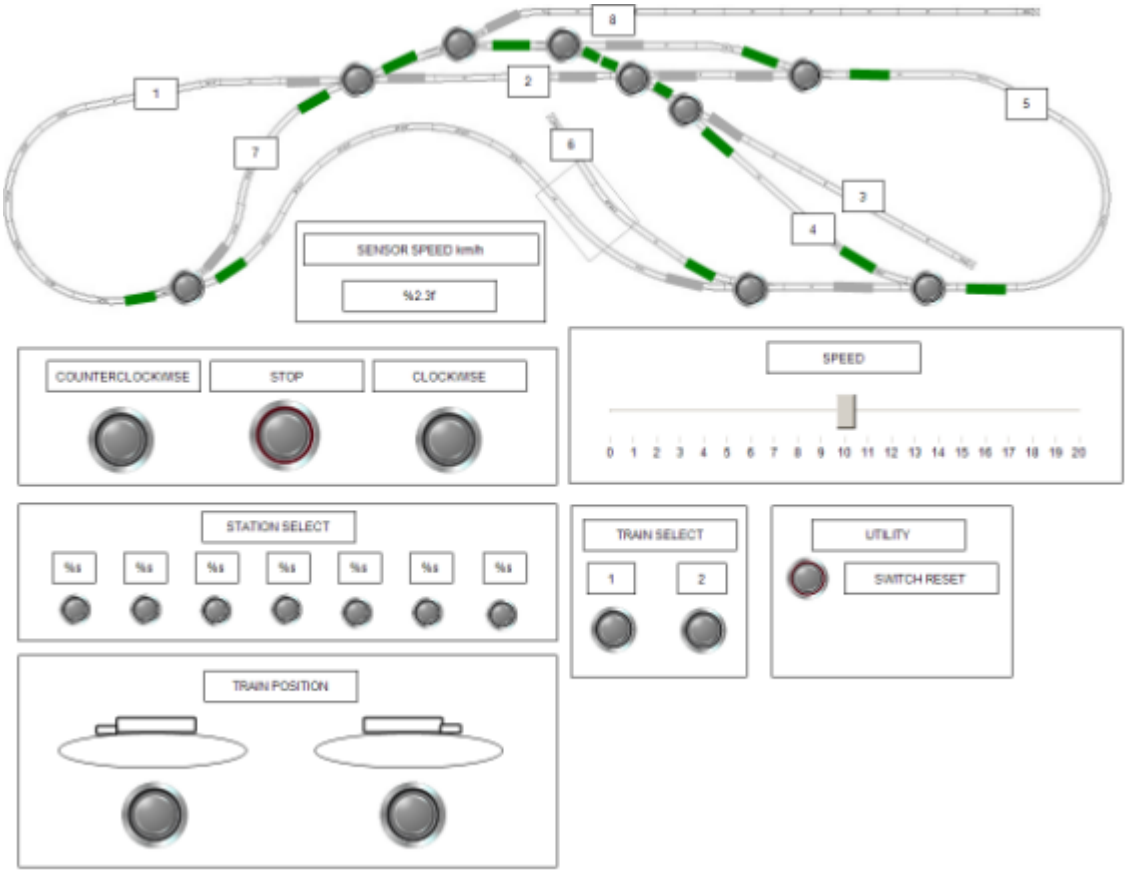


Figure 27. Visualization.

The Visualization is split into parts according to their functions.

Train track schematic contains the locations of the train stations that change colour to indicate if the train is currently present at the station. After the program recognizes the initial station containing the train, station number boxes can be pressed to initiate automatic movement of the train towards that station. Round buttons are used to operate the switches. Green and grey rectangles indicate the current position of the switches.

Below the systematic is the speed display according to the sensor. If the sensors are not currently receiving speed information, this display will stay invisible.

Display containing three buttons designated as “COUNTERCLOCKWISE”, “STOP” and “CLOCKWISE” as well as nearby display containing a “SPEED” slider, are used for the manual operation of the train. If “STOP” is pressed all current path programs and movement are immediately ceased.

The “STATION SELECT” display creates an alternative method to select the automatic station path. The availability of the stations depends on the current location of the train. Measures are taken to prevent the operator from choosing a new path while the previous one is in progress.

Next to the “STATION SELECT” screen, the “TRAIN SELECT” screen allows the operator to choose which train will be controlled by the program.

In the “UTILITY” display the switch select button allow the operator to reset the position of all the switches. This button must be operated before the use of the program may commence. Between the operation sessions the position of the intersections may change and since track switches lack sensors indicating their position, the program must record the current position of the sensors in the software. This is the main reason for above mentioned feature.

The final display “TRAIN POSITION” is used to mirror the controls in case the train is turned around by the operator. The train possess no sensors that could indicate the directional position of the locomotive thus the operator must distinguish the facing of the train by looking at the train and choosing its correct position manually in the visualization.

# 5 CHALLANGES ENCOUNTERED

During the development process multiple issues were encountered and are listed here.

## 5.1 Serial communication

Serial communication, especially the protocol used for the purpose of this research project, generally lacks the response time of a parallel communication method. Thus, care must be taken to ensure that only important communications are sent to prevent a delay in communication. This issue was resolved in the software.

## 5.2 Second train

Although in the current configuration control the of a second train using the Twin-Center device is possible, that functionality is coincidental. The second train is a device from a different generation of Fleichman solutions and uses a different syntax. A longer message must be sent to control the second train and neither serial communication configuration nor the current Twin-Center device allow for it. Further developments must be made to ensure the correct operation of the second train. However, such development requires further investment into the track table.

## 5.3 Sensor delay

There is a substantial delay between the reed switch activation and the receival of the signal by the PLC. Because of this fact, if the train speed is

too high, a serious overshoot of the station may occur. This issue can be fixed in the program by limiting the maximum speed of the train during the automatic operation.

### 5.4 **Visualization upload to PLC**

A problem was encountered during an attempt to upload the current visualization to be used with the PLC touch display. The visualization is too complex and will cause watchdog errors wehn an upload to PLC is initiated. Thus, the visualization exists only in the Twin-Cat software window and not in the embedded PLC display.

## 6 **CONCLUSION**

The objective of the project was accomplished successfully, within a reasonable timeframe. The purpose was to familiarize the author with the development of software solutions using the Twin-Cat software, and to familiarize the author with a research and development process.

The finalized solution is simple and easily reversible if the need arises. A PLC is required to run the solution and a compatible PC is necessary to upload the program to the PLC as well as to access the visualization interface. Software interface is engineered to allow an operator to interact with the program after minimal orientation. The project consists of a TwinCat solution and the changes made to the train track table. Features requested by the commissioner were developed in full.

Object detection on a small scale poses a continuous challenge to automation engineers of different specialties. Indeed, a single solution can be used for a multitude of tasks, but a task may be found where any generic solution may be lacking and a separate, specialized solution must be developed. Operational and installation costs must be taken into consideration when choosing the correct solution for a task.

Due to limitations in funding and physical properties of the train table system the accepted solution was less complex then initially planned. The solution allows a detection of the train speed at a single point of the track that contains repositioned sensors. It serves as proof of concept for the developed solution.

Extensive knowledge of digital communication and scale model operation was gained over the duration of the thesis. Overall, all project milestones were reached, and valuable experience was gained by the author. The commissioner has accepted the project as complete.

## REFERENCES


Blom, J. (n.d.). Serial Communication. Retrieved April 17, 2019, from https://learn.sparkfun.com/tutorials/serial-communication/all

Bolton, W. (2009). Programmable Logic Controllers (5th ed.). Elsevier Science & Technology.

Coulter, L. (n.d.). History of model trains. (B. Coulter, Ed.). Retrieved from http://www.o-gauge.com/Model_Train_History.htm

Ellwood, W. B. (1941). U.S. Patent No. US2264746A. Washington, DC: U.S. Patent and Trademark Office. Retrieved from https://patents.google.com/patent/US2264746

EXtended Automation Engineering XAE. (n.d.). Retrieved March 30, 2019, from https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_overview/4275768971.html&id=5986876456246657686

Fleischmann Twin-Center 6802 Operating Instruction Manual [PDF]. (n.d.). Fleischmann.

History of RFID. (n.d.). Retrieved December 5, 2018, from https://www.globalventurelabels.com/history-of-rfid/

Indoor GPS, our real time Positioning System. (n.d.). Retrieved September 9, 2018, from http://www.gamesontrack.com/satellites.html

Railroad Modeling. (2015, August 14). Retrieved October 25, 2018, from https://dccwiki.com/Railroad_Modeling

RFID History. (n.d.). Retrieved December 6, 2018, from https://www.electronics-notes.com/articles/connectivity/rfid-radio-frequency-identification/development-history.php

Sensors & Magnets. (n.d.). Retrieved February 5, 2019, from https://standexelectronics.com/products/reed-sensors-magnets/

The RS-232 protocol. (n.d.). Retrieved March 5, 2019, from https://www.omega.co.uk/temperature/z/rs232standard.html

The Windows Control and Automation Technology. (n.d.). Retrieved March 30, 2019, from https://www.beckhoff.com/twincat/

Thornton, F., & Lanthem, C. (2006). RFID Security (1st ed.). Elsevier Science & Technology Books.

Train Detection. (2019, March 1). Retrieved March 18, 2019, from https://dccwiki.com/Train_Detection

Why Model? (2015, January 9). Retrieved October 24, 2018, from https://www.nmra.org/beginner/why-model

Appendix 1

I/O list

| Serial Communication I/O | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| Inputs | | | | | |
| | Terminal | Channel | Variable | Device N. | |
| | KL6991-S | 1 | stIn_KL6001 | | |
| | | | | | |
| Outputs | | | | | |
| | Terminal | Channel | Variable | Device N. | |
| | KL6991-S | 1 | stOut_KL6001 | | |
| | | | | | |
| I/O | | | | | |
| | | | | | |
| Inputs | | | | | |
| | Terminal | Channel | Variable | Device N. | |
| | KL1408(1) | 1 | Term4_1 | 43 | |
| | | 2 | Term4_2 | 44 | |
| | | 3 | Term4_3 | 39 | |
| | | 4 | Term4_4 | 40 | |
| | | 5 | Term4_5 | 42 | |
| | | 6 | Term4_6 | 41 | |
| | | 7 | Term4_7 | 15 | |
| | | 8 | Term4_8 | 16 | |
| | KL1408(2) | 1 | Term5_1 | 11 | |
| | | 2 | Term5_2 | 12 | |
| | | 3 | Term5_3 | 22 | |
| | | 4 | Term5_4 | 21 | |
| | | 5 | Term5_5 | 37 | |
| | | 6 | Term5_6 | 38 | |
| | | 7 | Term5_7 | 20 | |
| | | 8 | Term5_8 | 17 | |
| | KL1408(3) | 1 | Term6_1 | 18 | |
| | | 2 | Term6_2 | 19 | |
| | | 3 | Term6_3 | 14 | |
| | | 4 | Term6_4 | 45 | |
| | | 5 | - | - | |
| | | 6 | - | - | |
| | | 7 | Term6_7 | 25 | |
| | | 8 | Term6_8 | 30 | |

| | | | | | |
|---|---|---|---|---|---|
| | KL1408(4) | 1 | Term7_1 | 26 | |
| | | 2 | Term7_2 | 29 | |
| | | 3 | Term7_3 | 4 | |
| | | 4 | Term7_4 | 3 | |
| | | 5 | Term7_5 | 31 | |
| | | 6 | Term7_6 | 32 | |
| | | 7 | Term7_7 | 2 | |
| | | 8 | Term7_8 | 1 | |
| | KL1408(5) | 1 | Term8_1 | 27 | |
| | | 2 | Term8_2 | 28 | |
| | | 3 | - | - | |
| | | 4 | - | - | |
| | | 5 | Term8_5 | 36 | |
| | | 6 | Term8_6 | 35 | |
| | | 7 | Term8_7 | 23 | |
| | | 8 | Term8_8 | 24 | |
| | KL1408(6) | 1 | Term9_1 | 34 | |
| | | 2 | Term9_2 | 33 | |
| | | 3 | Term9_3 | 13 | |
| | | 4 | Term9_4 | 7 | |
| | | 5 | Term9_5 | 5 | |
| | | 6 | Term9_6 | 6 | |
| | | 7 | Term9_7 | 8 | |
| | | 8 | Term9_8 | 10 | |
| | KL1408(7) | 1 | - | - | |
| | | 2 | Term10_2 | 9 | |
| | | 3 | - | - | |
| | | 4 | - | - | |
| | | 5 | - | - | |
| | | 6 | - | - | |
| | | 7 | - | - | |
| | | 8 | - | - | |
| | KL1408(8) | 1 | - | - | |
| | | 2 | - | - | |
| | | 3 | - | - | |
| | | 4 | - | - | |
| | | 5 | - | - | |
| | | 6 | - | - | |
| | | 7 | - | - | |
| | | 8 | - | - | |
| | KL1408(9) | 1 | - | - | |
| | | 2 | - | - | |
| | | 3 | - | - | |
| | | 4 | - | - | |
| | | 5 | - | - | |
| | | 6 | - | - | |
| | | 7 | - | - | |
| | | 8 | - | - | |

| Outputs | | | | | |
|---|---|---|---|---|---|
| | Terminal | Channel | Variable | Device N. | Intersection/Relay pair N. |
| | KL2114(1) | 1 | bSwitch1 | 1 | 1 |
| | | 2 | bSwitch2 | 2 | 1 |
| | | 3 | bSwitch3 | 3 | 2 |
| | | 4 | bSwitch4 | 4 | 2 |
| | KL2114(2) | 1 | bSwitch5 | 5 | 3 |
| | | 2 | bSwitch6 | 6 | 3 |
| | | 3 | bSwitch7 | 7 | 4 |
| | | 4 | bSwitch8 | 8 | 4 |
| | KL2114(3) | 1 | bSwitch9 | 9 | 5 |
| | | 2 | bSwitch10 | 10 | 5 |
| | | 3 | bSwitch11 | 11 | 6 |
| | | 4 | bSwitch12 | 12 | 6 |
| | KL2114(4) | 1 | bSwitch13 | 13 | 7 |
| | | 2 | bSwitch14 | 14 | 7 |
| | | 3 | bSwitch15 | 15 | 8 |
| | | 4 | bSwitch16 | 16 | 8 |
| | KL2114(5) | 1 | bSwitch17 | 17 | 9 |
| | | 2 | bSwitch18 | 18 | 9 |
| | | 3 | bSwitch19 | 19 | Polarity |
| | | 4 | bSwitch20 | 20 | Polarity |
| | KL2114(6) | 1 | - | - | |
| | | 2 | - | - | |
| | | 3 | - | - | |
| | | 4 | - | - | |

Appendix 2

MAIN program

```
IF NOT b_Busy THEN
IF b_VisuSwitchL THEN
        CASE b_dirChanged OF
        0:
        i_Direction := 1;
        I_bTrain_Go:=TRUE;
        1:
        i_Direction := 2;
        I_bTrain_Go:=TRUE;
        END_CASE
END_IF
IF b_VisuSwitchR THEN
        CASE b_dirChanged OF
        0:
        i_Direction := 2;
        I_bTrain_Go:=TRUE;
        1:
        i_Direction := 1;
        I_bTrain_Go:=TRUE;
        END_CASE
END_IF
END_IF

IF b_VisuSwitchS THEN
        i_Direction := 0;
        I_bTrain_Go:=TRUE;
END_IF
// Changes button color if moving
CASE b_dirChanged OF
        0:
        IF i_direction=1 THEN
        b_aVisuSwitchCOLOR[1]:=TRUE;
        ELSE
        b_aVisuSwitchCOLOR[1]:=FALSE;
        END_IF
        IF i_direction=2 THEN
        b_aVisuSwitchCOLOR[2]:=TRUE;
        ELSE
        b_aVisuSwitchCOLOR[2]:=FALSE;
        END_IF
        1:
        IF i_direction=1 THEN
        b_aVisuSwitchCOLOR[2]:=TRUE;
        ELSE
```

```
                b_aVisuSwitchCOLOR[2]:=FALSE;
                END_IF
                IF i_direction=2 THEN
                b_aVisuSwitchCOLOR[1]:=TRUE;
                ELSE
                b_aVisuSwitchCOLOR[1]:=FALSE;
                END_IF
END_CASE

//checks if speed or direction changes in order to activate the send
function
s_StringSpeed := INT_TO_STRING (i_VisuSpeed) ;
b_SpeedChange:= i_VisuSpeed <> i_RealSpeed; //This updates the speed
b_DirChange:= i_Direction <> i_RealDir;
IF b_SpeedChange OR b_DirChange THEN
                I_bTrain_Go:=TRUE;
END_IF
//Send function call
IF I_bTrain_Go THEN
GOTRAIN(
                s_StringSpeed:=s_StringSpeed ,
                i_StringTrainSelect:= ,
                i_sDirection:=i_Direction ,
                fbSend_Go:= ,
                I_bTrain_Go:=I_bTrain_Go,
                TxBuffer:=TxBufferKL ,
                RxBuffer:=RxBufferKL);
END_IF


StatN(i_sDirection:=i_Direction);
//Paths function caöö
Paths1(i_SwitchSelect:=i_SwitchSelect , i_Direction:=i_Direction ,
StringSpeed:=s_StringSpeed );

Switch1(i_SwitchN:= i_SwitchSelect);
//Used for station select buttons
IF b_Button1 THEN
                b_aProgSwitch[1]:=TRUE;
END_IF

IF b_Button2 THEN
                b_aProgSwitch[2]:=TRUE;
END_IF

IF b_Button3 THEN
                b_aProgSwitch[3]:=TRUE;
END_IF
```

```
IF b_Button4 THEN
		b_aProgSwitch[4]:=TRUE;
END_IF

IF b_Button5 THEN
		b_aProgSwitch[5]:=TRUE;
END_IF

IF b_Button6 THEN
		b_aProgSwitch[6]:=TRUE;
END_IF

IF b_Button7 THEN
		b_aProgSwitch[7]:=TRUE;
END_IF

IF b_Button8 THEN
		b_aProgSwitch[8]:=TRUE;
END_IF

IF b_Button9 THEN
		b_aProgSwitch[9]:=TRUE;
END_IF
//Changes switch color
FOR i:=1 TO 9 DO
		IF b_aSwitchState[i] THEN
		b_aSwitchColor[i]:=TRUE;
		ELSE
		b_aSwitchColor[i]:=FALSE;
END_IF
END_FOR
LocalTime(seconds=> , milliseconds=> );

IF NOT b_Busy THEN	//prevents the train from running off track
		IF Term5_4 THEN
			b_aStopT[1]:=TRUE;
		END_IF

		IF Term4_1 THEN
			b_aStopT[2]:=TRUE;
		END_IF

		IF Term10_2 OR Term9_5 OR Term7_5 OR Term7_3 THEN
			b_aStopT[3]:=TRUE;
		END_IF

		IF b_aStopT[1] THEN //Station 6 stop
```

```
                        IF Term9_2 THEN
                                i_Direction:=0;
                                b_aStopt[1]:=FALSE;
                        END_IF
                END_IF

                IF b_aStopT[2] THEN //Station 3 stop
                        IF Term6_4 THEN
                                i_Direction:=0;
                                b_aStopt[2]:=FALSE;
                        END_IF
                END_IF

                IF b_aStopT[3] THEN //Station 8 stop
                        IF Term9_3 THEN
                                i_Direction:=0;
                                b_aStopt[3]:=FALSE;
                        END_IF
                END_IF
END_IF
//Color change for train select
CASE i_TrainVisuSelect OF
                0:
                StringTrainSelect := '3';
                b_aTrainVisuCOLOR[1]:=TRUE;
                b_aTrainVisuCOLOR[2]:=FALSE;
                1:
                StringTrainSelect := '4';
                b_aTrainVisuCOLOR[2]:=TRUE;
                b_aTrainVisuCOLOR[1]:=FALSE;
END_CASE
IF b_aTrainVisuSelect[1] THEN
                i_TrainVisuSelect:=0;
END_IF
IF b_aTrainVisuSelect[2] THEN
                i_TrainVisuSelect:=1;
END_IF
IF Term5_7 OR Term5_3 OR Term7_1 OR Term8_2 OR Term6_8 OR
Term5_6 THEN
                bSwitch19:=FALSE;
                bSwitch20:=FALSE;
                //b_pArea:=FALSE;
END_IF
IF Term7_3 OR Term9_6 OR Term9_8 OR Term9_3 OR Term7_6 OR
Term4_8 OR Term4_5 OR Term4_2 OR Term9_7 OR Term5_2 THEN
                bSwitch19:=TRUE;
                bSwitch20:=TRUE;
END_IF
```

Appendix 3

SerialCom program

```
//Serial communication
fbKL6001Config(
        Execute:= bKL6001ConfigExe,
        Mode:= SERIALLINEMODE_KL6_5B_STANDARD,
        Baudrate:= 9600, //9600
        NoDatabits:= 8, //8
        Parity:= 0,
        Stopbits:= 2,
        Handshake:= 1, //HANDSHAKE_RTSCTS
        ContinousMode:= FALSE,
        pComIn:= ADR(stIn_KL6001),
        pComOut:= ADR(stOut_KL6001),
        SizeComIn:= SIZEOF(stIn_KL6001),
        Done=> ,
        Busy=> ,
        Error=> bKL6001ConfigError,
        ErrorId=> eKL6001ConfigErrorID );
IF NOT fbKL6001Config.Busy AND NOT bKL6001ConfigError THEN
        bKL6001ConfigExe := FALSE;
        fbKL6001Ctrl(
        Mode:= SERIALLINEMODE_KL6_5B_STANDARD,
        pComIn:= ADR(stIN_KL6001),
        pComOut:= ADR(stOut_KL6001),
        SizeComIn:= SIZEOF(stIn_KL6001),
        Error=> bKL6001CtrlError,
        ErrorID=> eKL6001CtrlErrorID,
        TxBuffer:= TxBufferKL,
        RxBuffer:= RxBufferKL);
END_IF
```

Appendix 4

GO function block

```
StringDirectionR := 'R';
StringDirectionV := 'V';
StringDirectionR2 := 'R0';
StringDirectionV2 := 'V0';

i_Speed := 9;

s_StringSpeed1:= STRING_TO_INT (s_StringSpeed);

b_SpeedLow := i_VisuSpeed <= i_Speed;
//Combines the data command
CASE i_sDirection OF
0:

        StringCon := CONCAT(StringDirectionV,StringTrainSelect);
        StringCon1 := CONCAT(StringCon,StringDirectionV);
        stringSelect := CONCAT(StringCon1, '00');
1:

        IF b_SpeedLow THEN
        StringCon := CONCAT(StringDirectionR,StringTrainSelect);
        StringCon1 := CONCAT(StringCon,StringDirectionR2);
        stringSelect := CONCAT(StringCon1, s_StringSpeed);
        ELSE
        StringCon := CONCAT(StringDirectionR,StringTrainSelect);
        StringCon1 := CONCAT(StringCon,StringDirectionR);
        stringSelect := CONCAT(StringCon1, s_StringSpeed);
        END_IF
2:

        IF b_SpeedLow THEN
        StringCon := CONCAT(StringDirectionV,StringTrainSelect);
        StringCon1 := CONCAT(StringCon,StringDirectionV2);
        stringSelect := CONCAT(StringCon1, s_StringSpeed);
        ELSE
        StringCon := CONCAT(StringDirectionV,StringTrainSelect);
        StringCon1 := CONCAT(StringCon,StringDirectionV);
        stringSelect := CONCAT(StringCon1, s_StringSpeed);
        END_IF
END_CASE


//Here is the function block that sends a string to the train.
        stringSent := StringSelect;
```

```
                fbSend_Go(   SendString:= stringSent,
                                                TXbuffer:= TxBuffer,
                                                Busy=> bSendBusy_Go,
                                                Error=> eSendErrorID);
i_RealDir:= i_sDirection;
i_RealSpeed:=i_VisuSpeed;
I_bTrain_Go:=FALSE;
//This resets the send string command to make sure only one string is sent



//Echo communication
fbReceive(
                Prefix:= ,
                Suffix:= ,
                Timeout:= ,
                Reset:= ,
                StringReceived=> bStringReceived,
                Busy=> bReceiveBusy,
                Error=> eReceiveErrorID,
                RxTimeout=> bReceiveTimeout,
                ReceivedString:= sReceivedString,
                RXbuffer:= RxBuffer);
IF bStringReceived THEN
                sLastReceivedString := sReceivedString;
END_IF
```

Appendix 5

Switch function block example

```
//Switch reset button
IF BtR THEN
        FOR i:=1 TO 9 DO
                b_aSwitchState[i]:=TRUE;
                IF b_aSwitchState[i] THEN
                b_aProgSwitch[i]:=TRUE;
                END_IF
        END_FOR
        b_SwitchR:=FALSE;
END_IF
//Switches
//1:
IF b_aProgSwitch[1] THEN
        IF b_aSwitchState[1] THEN  // Switch 1 & 2
                CASE i_StepA[0] OF

                0:

                i_StepA[0]:=i_StepA[0]+1;

                1:
                i_StepA[0]:=i_StepA[0]+1;
                2:
                TP1(IN:= TRUE, PT:=T#500MS , Q=>bSwitch1  ,
ET=> );
                TON1(IN:=  TRUE  ,  PT:=T#500MS  ,  Q=>
b_CasA[0] , ET=> );
                IF b_CasA[0] THEN
                TP1(IN:= FALSE);
                TON1(IN:=FALSE);
                b_CasA[0]:=FALSE;
                i_StepA[0]:=i_StepA[0]+1;
                END_IF
                3:
                b_aSwitchState[1]:=FALSE;
                i_StepA[0]:=0;
                b_aProgSwitch[1]:=FALSE;

                END_CASE

        ELSE
                CASE i_StepA[0] OF

                0:
```

```
                        i_StepA[0]:=i_StepA[0]+1;

                        1:
                        i_StepA[0]:=i_StepA[0]+1;
                        2:
                        TP1(IN:= TRUE, PT:=T#500MS , Q=> bSwitch2 ,
ET=> );
                        TON1(IN:= TRUE , PT:=T#500MS , Q=>
b_CasA[0] , ET=> );
                        IF b_CasA[0] THEN
                        TP1(IN:= FALSE);
                        TON1(IN:=FALSE);
                        b_CasA[0]:=FALSE;
                        i_StepA[0]:=i_StepA[0]+1;
                        END_IF
                        3:
                        b_aSwitchState[1]:=TRUE;
                        i_StepA[0]:=0;
                        b_aProgSwitch[1]:=FALSE;

                        END_CASE

                END_IF
END_IF
```

Appendix 6

Paths function block example

```
//Button for direction change
IF b_adirChangedBUTTON[1] THEN
        b_dirChanged:=0;
END_IF
IF b_adirChangedBUTTON[2] THEN
        b_dirChanged:=1;
END_IF
CASE b_dirChanged OF
0:
i_Direction2:=2;
i_Direction1:=1;
b_adirChangedCOLOR[1]:=TRUE;
b_adirChangedCOLOR[2]:=FALSE;
1:
i_Direction2:=1;
i_Direction1:=2;
b_adirChangedCOLOR[2]:=TRUE;
b_adirChangedCOLOR[1]:=FALSE;
END_CASE


//Detects if the train is busy
FOR y:=1 TO 8 DO
        FOR i:=1 TO 8 DO
                IF b_aPath[y, i] THEN
                        b_Busy:=TRUE;
                END_IF
        END_FOR
END_FOR
IF b_VisuSwitchS THEN
        FOR x:=1 TO 8 DO
                FOR z:=1 TO 8 DO
                        b_aPath[x, z]:=FALSE;
                END_FOR
        END_FOR
        FOR u:=0 TO 40 DO
                i_aCase[u]:=0;
        END_FOR
END_IF
IF b_Busy THEN
        i_VisuSpeed:=5;
END_IF
IF i_Direction=0 THEN
        b_Busy:=FALSE;
```

```
END_IF

//Visu limits
CASE i_CaseStatN OF

        0:

        1:
        IF NOT b_Busy THEN
        IF b_aStatSwitch[0] OR b_aTrackBSwitch[2] THEN
                b_aPath[1, 2]:=TRUE;

        END_IF
        IF b_aStatSwitch[1] OR b_aTrackBSwitch[3] THEN
                b_aPath[1, 3]:=TRUE;

        END_IF
        IF b_aStatSwitch[2] OR b_aTrackBSwitch[4] THEN
                b_aPath[1, 4]:=TRUE;

        END_IF
        IF b_aStatSwitch[3] OR b_aTrackBSwitch[5] THEN
                b_aPath[1, 5]:=TRUE;

        END_IF
        IF b_aStatSwitch[4] OR b_aTrackBSwitch[6] THEN
                b_aPath[1, 6]:=TRUE;

        END_IF
        IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
                b_aPath[1, 7]:=TRUE;

        END_IF
        IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
                b_aPath[1, 8]:=TRUE;

        END_IF
        END_IF
        2:
        IF NOT b_Busy THEN
        IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[2, 1]:=TRUE;

        END_IF
        IF b_aStatSwitch[1] OR b_aTrackBSwitch[3] THEN
                b_aPath[2, 3]:=TRUE;

        END_IF
```

```
IF b_aStatSwitch[2] OR b_aTrackBSwitch[4] THEN
                b_aPath[2, 4]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[5] THEN
                b_aPath[2, 5]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[6] THEN
                b_aPath[2, 6]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
                b_aPath[2, 7]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
                b_aPath[2, 8]:=TRUE;

END_IF
END_IF
3:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[3, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
                b_aPath[3, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[4] THEN
                b_aPath[3, 4]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[5] THEN
                b_aPath[3, 5]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[6] THEN
                b_aPath[3, 6]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
                b_aPath[3, 7]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
```

```
                b_aPath[3, 8]:=TRUE;

END_IF
END_IF
4:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[4, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
                b_aPath[4, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[3] THEN
                b_aPath[4, 3]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[5] THEN
                b_aPath[4, 5]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[6] THEN
                b_aPath[4, 6]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
                b_aPath[4, 7]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
                b_aPath[4, 8]:=TRUE;

END_IF
END_IF
5:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[5, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
                b_aPath[5, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[3] THEN
                b_aPath[5, 3]:=TRUE;
```

```
END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[4] THEN
        b_aPath[5, 4]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[6] THEN
        b_aPath[5, 6]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
        b_aPath[5, 7]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
        b_aPath[5, 8]:=TRUE;

END_IF
END_IF
6:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
        b_aPath[6, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
        b_aPath[6, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[3] THEN
        b_aPath[6, 3]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[4] THEN
        b_aPath[6, 4]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[5] THEN
        b_aPath[6, 5]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[7] THEN
        b_aPath[6, 7]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
        b_aPath[6, 8]:=TRUE;

END_IF
```

```
END_IF
7:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[7, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
                b_aPath[7, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[3] THEN
                b_aPath[7, 3]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[4] THEN
                b_aPath[7, 4]:=TRUE;

END_IF
IF b_aStatSwitch[4] OR b_aTrackBSwitch[5] THEN
                b_aPath[7, 5]:=TRUE;

END_IF
IF b_aStatSwitch[5] OR b_aTrackBSwitch[6] THEN
                b_aPath[7, 6]:=TRUE;

END_IF
IF b_aStatSwitch[6] OR b_aTrackBSwitch[8] THEN
                b_aPath[7, 8]:=TRUE;

END_IF
END_IF
8:
IF NOT b_Busy THEN
IF b_aStatSwitch[0] OR b_aTrackBSwitch[1] THEN
                b_aPath[8, 1]:=TRUE;

END_IF
IF b_aStatSwitch[1] OR b_aTrackBSwitch[2] THEN
                b_aPath[8, 2]:=TRUE;

END_IF
IF b_aStatSwitch[2] OR b_aTrackBSwitch[3] THEN
                b_aPath[8, 3]:=TRUE;

END_IF
IF b_aStatSwitch[3] OR b_aTrackBSwitch[4] THEN
                b_aPath[8, 4]:=TRUE;
```

```
                END_IF
                IF b_aStatSwitch[4] OR b_aTrackBSwitch[5] THEN
                                b_aPath[8, 5]:=TRUE;

                END_IF
                IF b_aStatSwitch[5] OR b_aTrackBSwitch[6] THEN
                                b_aPath[8, 6]:=TRUE;

                END_IF
                IF b_aStatSwitch[6] OR b_aTrackBSwitch[7] THEN
                                b_aPath[8, 7]:=TRUE;

                END_IF
                END_IF
END_CASE

//Train Paths
IF b_aPath[1, 2] THEN
                i_Direction:= i_Direction2;
                IF b_aSwitchState[6] THEN
                                b_aProgSwitch[6]:=TRUE;
                END_IF
                IF Term9_4 OR Term9_7 THEN
                                i_Direction:= 0;
                                b_aPath[1, 2]:= FALSE;
                END_IF
END_IF

IF b_aPath[1, 3] THEN
                CASE i_aCase[0] OF
                0:
                b_aPath[1, 2]:=TRUE;
                i_aCase[0]:= i_aCase[0]+1;
                1:
                IF NOT b_aPath[1, 2] THEN
                                b_aPath[2, 3]:=TRUE;
                                i_aCase[0]:= i_aCase[0]+1;
                END_IF
                2:
                IF NOT b_aPath[2, 3] THEN
                                b_aPath[1, 3]:=FALSE;
                                i_aCase[0]:=0;
                END_IF
                END_CASE
END_IF
IF b_aPath[1, 4] THEN
                CASE i_aCase[1] OF
```

```
                0:
                b_aPath[1, 2]:=TRUE;
                i_aCase[1]:= i_aCase[1]+1;
                1:
                IF NOT b_aPath[1, 2] THEN
                        b_aPath[2, 4]:=TRUE;
                        i_aCase[1]:= i_aCase[1]+1;
                END_IF
                2:
                IF NOT b_aPath[2, 4] THEN
                        b_aPath[1, 4]:=FALSE;
                        i_aCase[1]:=0;
                END_IF
                END_CASE
END_IF

IF b_aPath[1, 5] THEN
                CASE i_aCase[2] OF
                0:
                b_aPath[1, 2]:=TRUE;
                i_aCase[2]:= i_aCase[2]+1;
                1:
                IF NOT b_aPath[1, 2] THEN
                        b_aPath[2, 5]:=TRUE;
                        i_aCase[2]:= i_aCase[2]+1;
                END_IF
                2:
                IF NOT b_aPath[2, 5] THEN
                        b_aPath[1, 5]:=FALSE;
                        i_aCase[2]:=0;
                END_IF
                END_CASE
END_IF

IF b_aPath[1, 6] THEN
                CASE i_aCase[3] OF
                0:
                b_aPath[1, 5]:=TRUE;
                i_aCase[3]:= i_aCase[3]+1;
                1:
                IF NOT b_aPath[1, 5] THEN
                        b_aPath[5, 6]:=TRUE;
                        i_aCase[3]:= i_aCase[3]+1;
                END_IF
                2:
                IF NOT b_aPath[5, 6] THEN
                        b_aPath[1, 6]:=FALSE;
                        i_aCase[3]:=0;
```

```
                END_IF
                END_CASE
END_IF

IF b_aPath[1, 7] THEN
                CASE i_aCase[27] OF
                0:
                b_aPath[1, 2]:=TRUE;
                i_aCase[27]:= i_aCase[27]+1;
                1:
                IF NOT b_aPath[1, 2] THEN
                                b_aPath[2, 7]:=TRUE;
                                i_aCase[27]:= i_aCase[27]+1;
                END_IF
                2:
                IF NOT b_aPath[2, 7] THEN
                                b_aPath[1, 7]:=FALSE;
                                i_aCase[27]:=0;
                END_IF
                END_CASE
END_IF

IF b_aPath[1, 8] THEN
                i_Direction:= i_Direction2;
                IF NOT b_aSwitchState[8] THEN
                                b_aProgSwitch[8]:=TRUE;
                END_IF
                IF NOT b_aSwitchState[6] THEN
                                b_aProgSwitch[6]:=TRUE;
                END_IF
                IF Term9_3 THEN
                                i_Direction:= 0;
                                b_aPath[1,8]:=FALSE;
                END_IF
END_IF
```

Appendix 7

StationN function block

```
//Detects the current station
IF Term7_7 OR Term7_8 THEN
        i_CaseStatN:= 1;
        b_aStationN[1]:=TRUE;
END_IF

IF Term9_4 OR Term9_7 THEN
        i_CaseStatN:= 2;
        b_aStationN[2]:=TRUE;
END_IF

IF Term6_4 THEN
        i_CaseStatN:= 3;
        b_aStationN[3]:=TRUE;
END_IF

IF Term4_4 THEN
        i_CaseStatN:= 4;
        b_aStationN[4]:=TRUE;
END_IF

IF Term5_8 OR Term6_1 THEN
        i_CaseStatN:= 5;
        b_aStationN[5]:=TRUE;
END_IF

IF Term8_5 OR Term8_6 THEN
        i_CaseStatN:= 6;
        b_aStationN[6]:=TRUE;
END_IF

IF Term7_5 OR Term7_6 THEN
        i_CaseStatN:= 7;
        b_aStationN[7]:=TRUE;
END_IF

IF Term9_3 THEN
        i_CaseStatN:= 8;
        b_aStationN[8]:=TRUE;
END_IF
//Resets the station visualization
IF b_Busy OR i_sDirection<>0 THEN
        FOR y:=1 TO 8 DO
                b_aStationN[y]:=FALSE;
```

```
                END_FOR
END_IF


//Station visualization text
CASE i_CaseStatN OF

                0:

                1:
                IF b_Busy THEN
                s_aStatB[0]:= ' ' ;
                s_aStatB[1]:= ' ' ;
                s_aStatB[2]:= ' ' ;
                s_aStatB[3]:= ' ' ;
                s_aStatB[4]:= ' ' ;
                s_aStatB[5]:= ' ' ;
                s_aStatB[6]:= ' ' ;
                ELSE

                s_aStatB[0]:= '2' ;
                s_aStatB[1]:= '3' ;
                s_aStatB[2]:= '4' ;
                s_aStatB[3]:= '5' ;
                s_aStatB[4]:= '6' ;
                s_aStatB[5]:= '7' ;
                s_aStatB[6]:= '8' ;
                END_IF
                2:
                IF b_Busy THEN
                s_aStatB[0]:= ' ' ;
                s_aStatB[1]:= ' ' ;
                s_aStatB[2]:= ' ' ;
                s_aStatB[3]:= ' ' ;
                s_aStatB[4]:= ' ' ;
                s_aStatB[5]:= ' ' ;
                s_aStatB[6]:= ' ' ;
                ELSE

                s_aStatB[0]:= '1' ;
                s_aStatB[1]:= '3' ;
                s_aStatB[2]:= '4' ;
                s_aStatB[3]:= '5' ;
                s_aStatB[4]:= '6' ;
                s_aStatB[5]:= '7' ;
                s_aStatB[6]:= '8' ;
                END_IF
                3:
```

```
IF b_Busy THEN
s_aStatB[0]:= ' ' ;
s_aStatB[1]:= ' ' ;
s_aStatB[2]:= ' ' ;
s_aStatB[3]:= ' ' ;
s_aStatB[4]:= ' ' ;
s_aStatB[5]:= ' ' ;
s_aStatB[6]:= ' ' ;
s_aStatB[6]:= ' ' ;
ELSE

s_aStatB[0]:= '1' ;
s_aStatB[1]:= '2' ;
s_aStatB[2]:= '4' ;
s_aStatB[3]:= '5' ;
s_aStatB[4]:= '6' ;
s_aStatB[5]:= '7' ;
s_aStatB[6]:= '8' ;
END_IF
4:
IF b_Busy THEN
s_aStatB[0]:= ' ' ;
s_aStatB[1]:= ' ' ;
s_aStatB[2]:= ' ' ;
s_aStatB[3]:= ' ' ;
s_aStatB[4]:= ' ' ;
s_aStatB[5]:= ' ' ;
s_aStatB[6]:= ' ' ;
ELSE

s_aStatB[0]:= '1' ;
s_aStatB[1]:= '2' ;
s_aStatB[2]:= '3' ;
s_aStatB[3]:= '5' ;
s_aStatB[4]:= '6' ;
s_aStatB[5]:= '7' ;
s_aStatB[6]:= '8' ;
END_IF
5:
IF b_Busy THEN
s_aStatB[0]:= ' ' ;
s_aStatB[1]:= ' ' ;
s_aStatB[2]:= ' ' ;
s_aStatB[3]:= ' ' ;
s_aStatB[4]:= ' ' ;
s_aStatB[5]:= ' ' ;
s_aStatB[6]:= ' ' ;
ELSE
```

```
s_aStatB[0]:= '1' ;
s_aStatB[1]:= '2' ;
s_aStatB[2]:= '3' ;
s_aStatB[3]:= '4' ;
s_aStatB[4]:= '6' ;
s_aStatB[5]:= '7' ;
s_aStatB[6]:= '8' ;
END_IF
6:
IF b_Busy THEN
s_aStatB[0]:= ' ' ;
s_aStatB[1]:= ' ' ;
s_aStatB[2]:= ' ' ;
s_aStatB[3]:= ' ' ;
s_aStatB[4]:= ' ' ;
s_aStatB[5]:= ' ' ;
s_aStatB[6]:= ' ' ;
ELSE

s_aStatB[0]:= '1' ;
s_aStatB[1]:= '2' ;
s_aStatB[2]:= '3' ;
s_aStatB[3]:= '4' ;
s_aStatB[4]:= '5' ;
s_aStatB[5]:= '7' ;
s_aStatB[6]:= '8' ;
END_IF
7:
IF b_Busy THEN
s_aStatB[0]:= ' ' ;
s_aStatB[1]:= ' ' ;
s_aStatB[2]:= ' ' ;
s_aStatB[3]:= ' ' ;
s_aStatB[4]:= ' ' ;
s_aStatB[5]:= ' ' ;
s_aStatB[6]:= ' ' ;
ELSE

s_aStatB[0]:= '1' ;
s_aStatB[1]:= '2' ;
s_aStatB[2]:= '3' ;
s_aStatB[3]:= '4' ;
s_aStatB[4]:= '5' ;
s_aStatB[5]:= '6' ;
s_aStatB[6]:= '8' ;
END_IF
8:
```

```
                IF b_Busy THEN
                s_aStatB[0]:= ' ' ;
                s_aStatB[1]:= ' ' ;
                s_aStatB[2]:= ' ' ;
                s_aStatB[3]:= ' ' ;
                s_aStatB[4]:= ' ' ;
                s_aStatB[5]:= ' ' ;
                s_aStatB[6]:= ' ' ;
                ELSE

                s_aStatB[0]:= '1' ;
                s_aStatB[1]:= '2' ;
                s_aStatB[2]:= '3' ;
                s_aStatB[3]:= '4' ;
                s_aStatB[4]:= '5' ;
                s_aStatB[5]:= '6' ;
                s_aStatB[6]:= '7' ;
                END_IF

END_CASE
```

Appendix 8

LocalTime function block

```
//Gets local system time
LocalTime(
        sNetID:= ,
        bEnable:= TRUE,
        dwCycle:= 1,
        dwOpt:= ,
        tTimeout:= ,
        bValid=> ,
        systemTime=> ,
        tzID=> );
LocalFileTime:=Tc2_Utilities.SYSTEMTIME_TO_FILETIME(LocalTime.systemTime);
seconds := (SHL(DWORD_TO_ULINT(LocalFileTime.dwHighDateTime), 32) + DWORD_TO_ULINT(LocalFileTime.dwLowDateTime)) / 10000000 - 11644473600;
milliseconds := (SHL(DWORD_TO_ULINT(LocalFileTime.dwHighDateTime), 32) + DWORD_TO_ULINT(LocalFileTime.dwLowDateTime)) / 10000 - 11644473600000;
//term88 -> 87
curTime:= ULINT_TO_LREAL(milliseconds);
T1(CLK:=Term8_7 , Q=> T1T);
T2(CLK:=Term8_8 , Q=> T2T );
//Records system time then passing sensor
IF T2T THEN
        Time1:=curTime;
END_IF

IF T1T THEN
        Time2:=curTime;
END_IF

Time3:=Time1-Time2;
b_nZero:= Time3  <> 0;
//Calculates speed
IF b_nZero THEN
Time4:=ABS(Time3);
SpeedP:= 5/Time4;
Speed:= SpeedP*36;
b_VisuInvisu:=FALSE;
END_IF
//Turns the sensor visu invisible if necissary
IF Term5_3 OR Term5_4 OR Term7_1 OR Term6_7 OR NOT b_nZero THEN
        b_VisuInvisu:=TRUE;
END_IF
```

Appendix 9

Global Variables

```
//Train_GO : GO;

        //b_SwitchTheRails : BOOL;
        b_Busy: BOOL;
        b_VisuSwitchL : BOOL;
        b_VisuSwitchS : BOOL;
        b_VisuSwitchR : BOOL;
        b_aVisuSwitchCOLOR : ARRAY[1..2] OF BOOL;


        i_VisuSpeed : INT;
        i_RealSpeed : INT;
        i_RealDir : INT;
        Speed: REAL;
        i_TrainVisuSelect:INT;
        b_aTrainVisuSelect: ARRAY[1..2] OF BOOL;
        b_aTrainVisuCOLOR: ARRAY[1..2] OF BOOL;
        b_VisuInvisu: BOOL;

        StringTrainSelect: STRING;

        b_dirChanged: INT;
        b_adirChangedBUTTON: ARRAY[1..2] OF BOOL;
        b_adirChangedCOLOR: ARRAY[1..2] OF BOOL;


        RxBufferKL : ComBuffer;
        TxBufferKL : ComBuffer;

        i_CaseStatN : INT;
        i_CaseStatN2 : INT;
        b_SwitchReset : BOOL;
        b_aStatSwitch : ARRAY [0..6] OF BOOL;
        b_aTrackBSwitch : ARRAY [1..8] OF BOOL;
        s_aStatB : ARRAY [0..6] OF STRING(1);
        b_aStationN : ARRAY [1..8] OF BOOL;
        b_aProgSwitch : ARRAY [1..9] OF BOOL;
        b_aSwitchState : ARRAY [1..9] OF BOOL;
        b_aSwitchColor : ARRAY [1..9] OF BOOL;
```