OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG MB ILM FAKULTÄT FÜR MASCHINENBAU INSTITUT FÜR LOGISTIK UND MATERIALFLUSSTECHNIK

Digital Engineering Project

On

IMPLEMENTATION OF COMMUNICATION PROTOCOLS IN INDUSTRY 4.0 LEARNING LABORATORY

By

Chaitanya Sankaramanchi	**225820**
Elvis Joseph	**226214**
Naveen Kumar Pesaru	**230060**
Srivamsi Malladi	**229795**
Sri Girish Tangirala	**230061**

Under Supervision of

Dr.-Ing. Tobias Reggelin

M.Sc. Vasu Dev Mukku

Abstract

In the present industrial ecosystem, the demand from customers is continuously changing. To meet these requirements, the logistics industry should adopt various end to end supply chain solutions. The goal is to achieve coordination and transparency in all material flow operations. This can be achieved by constantly modifying the factory layouts according to production requirements. The standalone and modular factory components can be used for implementing dynamic factory layouts efficiently. The communication between these factory modules also plays an important role in the production network.

In this project, completely standalone and modular factory components are developed, each with an independent controller. The factory layouts are developed with production, storage and dispatch areas. The control flow in the factory layout is developed by simulating a virtual model in Visual Components and realised as a physical layout implementing different communication protocols.

Contents

List of Figures

List of Tables

List of Equations

1. Introduction

1.1 Motivation

Industry 4.0 concepts can be used in Logistics and material flow industries to improve the supply chain. The evolution of the Internet of things, cyber physical systems helps in development of modular and standalone material handling components for dynamical factory layouts. The existing factory layouts are static and run on a single controller. This makes the customization of layouts difficult and time consuming. To meet customer requirements the production plants should adapt their production process for a sustainable business. This is leading to development of dynamic and decentralised factory layout, which are flexible for customisations. These modular components use wireless medium for communications.

The aim of the project is to make the components in Industry 4.0 Learning laboratory modular and standalone such that they can be customised for different material flow operations. The modularity focuses on using independent controllers, power supply and communication channels. The current scope is also to implement communication protocols such as LiFi (Light Fidelity), Bluetooth, NRF (Radio Frequency), Infrared, UART (Universal Asynchronous Receiver Transmitter). The operations such as production, storage and dispatch are developed with appropriate communication protocol and these are interconnected using UART communication. The components are designed in a way to be able to use any of the mentioned communications.

1.2 Objectives

Based on the aim discussed in previous section, the objectives of the project are:

- Conceptual Model
- Assembling and testing of individual factory components
- Integrating communication protocols
- Developing and simulating Virtual model
- Realising the virtual model into physical model

1.3 Organization of project

Chapter 1 gives the introduction of the project in detail with motivation and objectives.

Chapter 2 gives literature review on Industry 4.0, Industry 4.0 Learning Laboratory, Simulation and communication protocols.
Chapter 3 describes the implementation of the project in detail with hardware and software setup, simulation, design of factory modules, implementation of communication protocols in factory modules, realisation of factory layout from conceptual and simulated model.
Chapter 4 summarizes the project and future work is discussed.

2. Literature Survey

2.1 Industry 4.0

Industrial revolution which began in late 1700s was the main cause for development of factories. The trends which took place with different generations of industrial revolution helped in increasing the terms productivity, quality, flexibility and time for development of products. The first industrial revolution focused on mechanisation and usage of steam power. The second industrial revolution introduced the term “factory” which is the cause for mass production on assembly lines powered by electrical energy. The internal combustion engine was produced at this time which is the main source for automobiles. The third industrial revolution can be considered an important phase due to advancements in telecommunications, electronics and computers. The programmable logic controllers (PLC) and robots helped in development of automation. The fourth industrial revolution introduced the terms smart factories, cyber physical systems. The components of Industry 4.0 are shown in Figure 2.1. The technologies on this revolution on development of products considering occupational safety, ergonomically adapted workstations, training and collaboration in production networks, protecting the environment by using energy efficient methods.

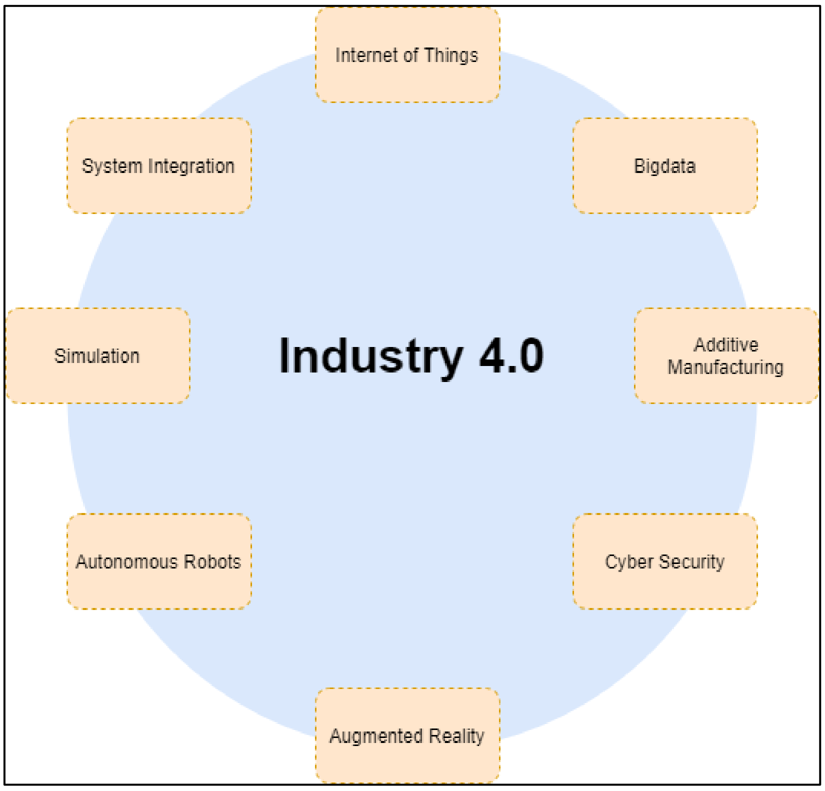


Figure 2.1 Industry 4.0 Technologies

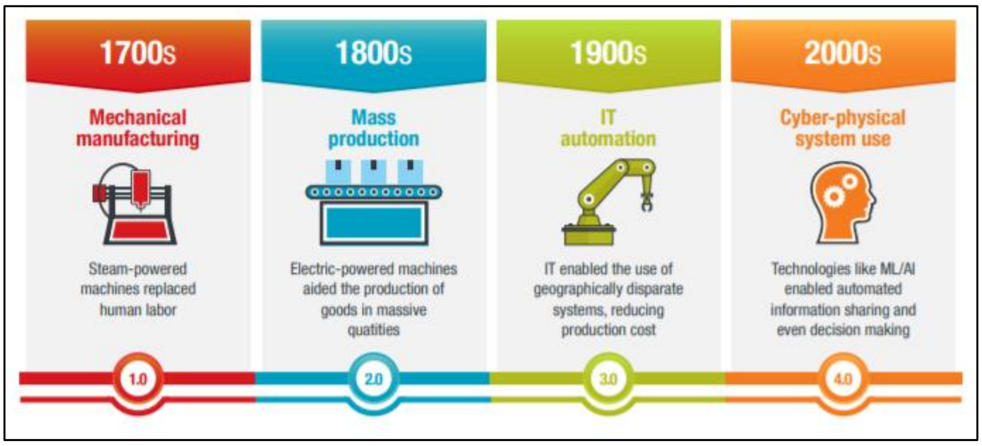

1700s
Mechanical manufacturing
Steam-powered machines replaced human labor
1.0
1800s
Mass production
Electric-powered machines aided the production of goods in massive quatities
2.0
1900s
IT automation
IT enabled the use of geographically disparate systems, reducing production cost
3.0
2000s
Cyber-physical system use
Technologies like ML/AI enabled automated information sharing and even decision making
4.0

Figure 2.2 Revolutions in Industry [1]

2.2 Industry 4.0 Learning Laboratory

Institute of Logistics and Material flow (ILM) department of Otto von Guericke University, Magdeburg, Germany in cooperation with Fraunhofer Magdeburg initiated the Industry 4.0 Learning Laboratory. The principle element of the laboratory is modular, plug and play factory modules, which proves the concept of implementing dynamic system structures. Currently each module is equipped with a LiFi transceiver and the master module has an RFID reader. Each module is programmed with a unique ID. The modules communicate using LiFi Communication. Different factory prototypes are implemented using conveyors, turntables and a slider by assigning unique device ID to each module, testing the maximum communication distance and reliability of LiFi communication.

2.3 RIOT OS:

RIOT is an open source OS developed to run on low-end [2] devices, especially microcontrollers and processors for developing Embedded and IoT systems. The prominent features of RIOT OS are as follows:

- Very lightweight operating system,
- Consumes lesser system resources like RAM, ROM and even power,
- Support for C, C++,
- Support for several libraries and APIs, especially Arduino libraries and
- Multithreading support.

2.4 Simulation

Engineering applications and systems have entered an extremely sophisticated era led by technology. Engineers involve themselves in a critical challenge while designing, implementing, managing and optimizing these complex systems according to the constantly changing products of the future. The same technological advancement that complicates the engineering application also provides a solution to enhance the system engineering and how its tasks can be augmented with a quantitative approach.

Simulation modelling provides a platform to solve realistic problems efficiently and safely. It gives a very important scheme of analysis which can be easily verified, understood and communicated. Across industries and other platforms, modelling through simulation gives efficient solutions by providing clearer insights into the complex systems. Simulation can be used in experimenting and representing a system digitally. Contrast to the physical modelling which is usually implementing a scaled copy of a system, simulation involves computer-oriented design and makes use of mathematical analysis. Simulating using software gives a dynamic environment for the computer model analysis during run, also facilitating views in 3D or 2D. The use of simulation is increasingly becoming predominant when conducting experiments on a real system, but often it becomes impossible or impractical and might be a cause of cost and time factors.

Modelling a system can be divided into three main categories. Analytical Modelling, Simulation Modelling and Hybrid modelling [3]. A Simulation Model is descriptive in nature which makes use of software modules or sub-models that signify a complex real-world system. It can be a discrete time-based simulation or continuous in nature. The simulation modelling uses a detailed representation of a system and variety of components through software modules. It describes the system behaviour of an engineering model by replicating the performance and functionality for a set of observations. Simulation modelling is a time-consuming application, it is majorly useful when the system components and configuration is known and could be estimated. The major steps of modelling a simulation involves problem formulation, collection of data and analysis, forming the model, program coding, model validation, error finding, design experiments and result analysis.

2.4.1 Simulations in Industry 4.0

Simulation methods have been used in the supply chain management (SCM) context. The primitive approaches involved modelling the different stages of the supply chain as a single simulation, whereas the distributed supply chain management facilitated in coordinated implementation of the current existing models using the distributed simulation middleware. Implementation of Industry 4.0 and cyber physical systems have created “smart factory” which introduced customized configuration for product assembly lines. The role of simulation in designing, experimenting and prototyping the implementation of a number of interconnected Industry 4.0 components can overcome computational and information disclosure problems within the physical model. [4]

In this project, simulation modelling has been used to design, analyse, and portray the implementation of the physical model. The simulation software Visual Components 4.0 was used for simulating the project model.

2.4.2 Visual Components 4.0

Visual Components is a leading developer of 3D manufacturing simulation software and solutions. It makes manufacturing design and simulation technology easy to use and accessible to manufacturing organizations of all sizes. The Visual Components Software was used to design and virtually validate the model of this project. The features of the software can be used for layout planning, performance calculation, realistic visual experience using virtual reality features, virtual commissioning and PLC validation.

The manufacturing simulation was helpful for:

- Designing a smarter solution
- Getting a predictable performance
- Speeding up the concept Design and
- Validating the changes virtually

2.5 Literature Review on Communication Modules

2.5.1 UART

UART is one of the most basic serial communication protocols in electronics. It requires two wires connected between two communicating devices, one for transmission and one for reception. The transmitting wire (Tx) of one device is connected to the receiving wire (Rx) and vice versa. Data is transmitted in the form of a series of signal pulses.

The serial communication pins of an Arduino Uno are D0 and D1 and they are labelled as Tx and Rx respectively. When the Arduino is connected to a computer through a USB cable, these pins are connected to a virtual serial port on the computer. When the Arduino is not

communicating with a computer, these two pins can be used to communicate with another device which supports UART communication.
The rate at which bits of data are transmitted is called Baud Rate. One of the most common baud rates for UART, especially while used in tandem with Arduino, is 9600 bps. Examples of other common baud rates are 4800, 19200, 38400, and 115200 bps. Two devices communicating through UART must be configured to use the same baud rate. The maximum baud rate that a system can handle depends mostly on the length of the wires of the UART transmission, in addition to other minor factors. Two different Arduino boards connected with two long wires may not be able to support high rates such as 115200 bps, thus limiting the speed of communication.

2.5.2 NRF

The NRF24L01+ Wireless module by Nordic Semiconductor is mainly used for low-power applications as it ensures reliable, two-way RF communication between multiple controllers. It uses GFSK modulation for data transmission and it is designed to operate in a 2.4GHz frequency band. [5]
NRF24L01+ uses Enhanced ShockBurst protocol (ESB), which enables the implementation of ultra-low power and high-performance communication with inexpensive host microcontrollers. The operating voltage of the module is 1.9V to 3.6V, and the logic pins are 5V tolerant. It can be directly connected to an Arduino or any 5V logic microcontroller without using any logic level converter.
The NRF24L01+ module communicates using a 4-pin SPI interface with a maximum data rate of 10 Mbps. A Master-Slave model of asynchronous communication method is implemented among two modules. All these parameters can be configured using software SPI.
The maximum operating current is 13.5 mA and the expected communication range is 100 meters in the line-of-sight.
Two versions of the module are available, the NRF24L01+ and the NRF24L01+ PA/LNA. The former uses an onboard antenna, due to which the communication range is lesser, and the signal is weaker, especially indoors where they might be interfered by walls. The latter consists of a RFX2401C chip which comes equipped with a Power Amplifier (PA), the Low-Noise Amplifier (LNA) and the transmit-receive switching circuitry. Due to these circuits and an additional antenna, the range is much larger and can reach upto 1000m in open air.

The Power Amplifier merely boosts the signal that is being transmitted from the module, whereas the Low-Noise Amplifier receives the incoming signal through the antenna and amplifies it to a level which can be processed by a controller.

The NRF module can transmit and receive data through one of 125 frequency channels. Two modules communicating with each other need to be on the same channel. Each channel occupies 1MHz frequency spacing, although it is good practice for two independent channels to be at least 2MHz apart to avoid signal overlapping at maximum data transmission speed.

The NRF module can simultaneously receive information from 6 other modules. Each physical RF channel can be logically divided into 6 parallel data channels called Data Pipes. Each data pipe has its own address and can be configured. Conversely, the module can only transmit to one channel at a time.

2.5.3 LiFi

Li-Fi (Light Fidelity) is a wireless communication technology which utilizes light to transmit data between devices. The term was first introduced by Harald Haas during a 2011 TEDGlobal talk in Edinburgh.

In technical terms, Li-Fi is a light communication system that is capable of transmitting data at high speeds over the visible light, ultraviolet, and infrared spectrums. In its present state, only LED lamps can be used for the transmission of visible light.

In terms of its end use, the technology is similar to Wi-Fi, the key technical difference being that Wi-Fi uses radio frequency to induce a voltage in an antenna to transmit data, whereas Li-Fi uses the modulation of light intensity to transmit data. Li-Fi can theoretically transmit at speeds of up to 100 Gbit/s. Li-Fi's ability to safely function in areas otherwise susceptible to electromagnetic interference (e.g. aircraft cabins, hospitals, military) is an advantage. The technology is being developed by several organizations across the globe [6].

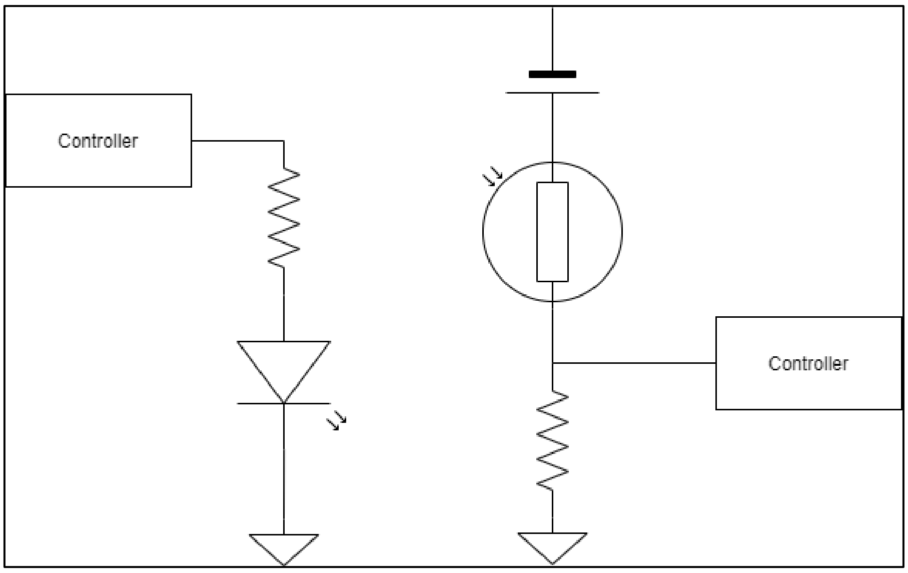


Figure 2.3 LiFi Transceiver Circuit Block Diagram

Li-Fi is a derivative of optical wireless communications (OWC) technology, which uses light from light-emitting diodes (LEDs) as a medium to deliver network, mobile, high-speed communication in a similar manner to Wi-Fi. The Li-Fi market was projected to have a compound annual growth rate of 82% from 2013 to 2018 and to be worth over $6 billion per year by 2018. However, the market has not developed as such and Li-Fi remains with a niche market, mainly for technology evaluation.

Visible light communications (VLC) works by switching the current to the LEDs off and on at a very high speed, too quick to be noticed by the human eye, thus, it does not present any flickering. Although Li-Fi LEDs would have to be kept on to transmit data, they could be dimmed to below human visibility while still emitting enough light to carry data. This is also a major bottleneck of the technology when based on the visible spectrum, as it is restricted to the illumination purpose and not ideally adjusted to a mobile communication purpose. Technologies that allow roaming between various Li-Fi cells, also known as handover, may allow a seamless transition between Li-Fi. The light waves cannot penetrate walls which translates to a much shorter range, and a lower hacking potential, relative to Wi-Fi. Direct line of sight is not necessary for Li-Fi to transmit a signal; light reflected off walls can achieve 70 Mbit/s.

Li-Fi has the advantage of being useful in electromagnetic sensitive areas such as in aircraft cabins, hospitals and nuclear power plants without causing electromagnetic interference. Both Wi-Fi and Li-Fi transmit data over the electromagnetic spectrum, but whereas Wi-Fi utilizes radio waves, Li-Fi uses visible, ultraviolet, and infrared light. While the US Federal Communications Commission has warned of a potential spectrum crisis because Wi-Fi is close to full capacity, Li-Fi has almost no limitations on capacity. The visible light spectrum is 10,000 times larger than the entire radio frequency spectrum. Researchers have reached data rates of over 224 Gbit/s, which was much faster than typical fast broadband in 2013. Li-Fi is expected to be ten times cheaper than Wi-Fi. Short range, low reliability and high installation costs are the potential downsides.

2.5.4 Bluetooth

The HC-05 is a Serial Port Protocol (SPP) module, which is designed for full-duplex wireless serial data transmission. It consists of a 2.4 GHz radio transceiver and uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology.

The operating voltage is between 1.8 to 3.6V for the on-board chip, although the voltage regulator added in the module allows the use of 5V power supply. Likewise, the Tx Pin supplies 5V, but the Rx pin is only compatible with 3.3V logic. A potential divider circuit has to be implemented if the Rx pin needs to be connected to an Arduino, because the Serial communication pins of Arduino implement 5V logic.

The operating current is 30mA and the expected range is less than 100m. It uses Frequency-Hopping Spread Spectrum (FHSS) and works using a USART circuit. The supported baud rates for HC-05 modules are 9600, 19200, 38400, 57600, 115200, 230400 and 460800 bps.

An LED is present on the module for the indication of the connection state. When the master and slave modules are connected, the LED blinks once every two seconds, and when they are disconnected, the LED blinks twice per second.

The default baud rate, master/slave mode and other such parameters can be configured using the AT command mode by connecting the HC-05 to a computer.

2.5.5 IR

IR radiation is simply light that we cannot see, which makes it great for communication. IR sources are all around us. The sun, light bulbs, or anything with heat is very bright and is in the IR spectrum. In a device such as a TV remote, an IR LED is used to transmit information to the TV. The IR signal is modulated, which enables it to distinguish the necessary signal from the ambient noise. Modulating a signal is similar to assigning a pattern to data, so that the receiver knows to listen. [7]

A common modulation scheme for IR communication is something called 38kHz modulation. There are very few natural sources that have the regularity of a 38kHz signal, so an IR transmitter sending data at that frequency would stand out among the ambient IR. 38kHz modulated IR data is the most common, but other frequencies can be used.

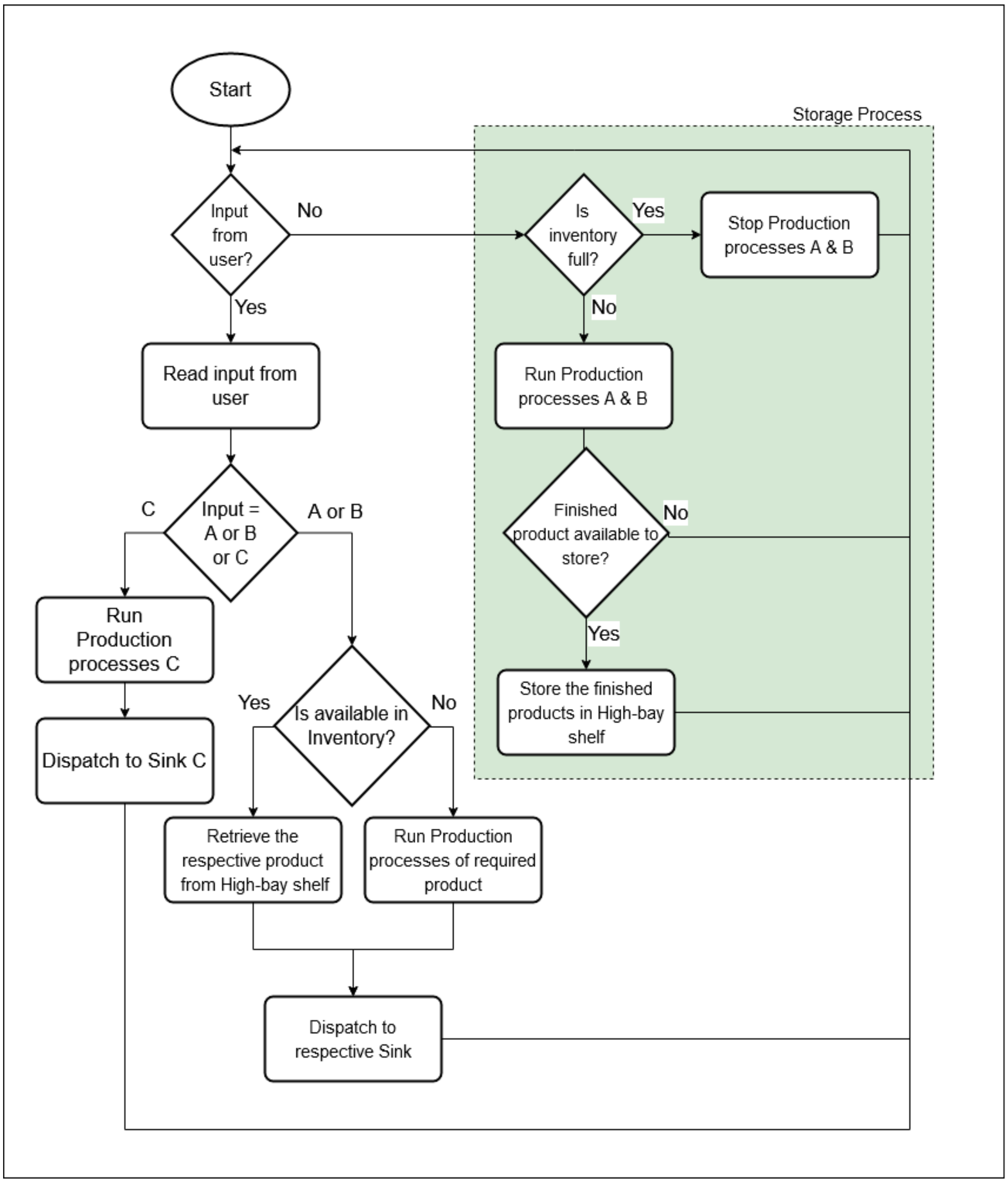
When a key is pressed on a remote, the transmitting IR LED will blink very quickly for a fraction of a second, transmitting encoded data to the appliance. Each pulse is turned on and off at a frequency of 38kHz. If an oscilloscope is connected to the TV remote's IR LED, a signal similar to the one above can be observed. This modulated signal is exactly what the receiving system sees. However, the point of the receiving device is to demodulate the signal and output a binary waveform that can be read by a microcontroller. The OUT pin of the TSOP382 with the wave from above produces a periodic rectangular signal.

By controlling the spacing between the transmitted modulated signals, the waveform can be read by an input pin on a microcontroller and decoded as a serial bit stream. An Arduino or other microcontroller can be connected to either end of the system to transmit data (left side) or receive data (right side).

3. Implementation

3.1 Conceptual Model

Several factory floor plans have been designed to include the basic industry functions such as production, storage of products, and dispatch on customer request. The products are categorized into two types, make-to-order and make-to-stock. Several plans would be possible with different combinations of the given components, two of which are discussed below.



Start
Storage Process
Input from user?
No
Yes
Is inventory full?
Yes
No
Stop Production processes A & B
Read input from user
Run Production processes A & B
Input = A or B or C
C
A or B
Finished product available to store?
No
Yes
Run Production processes C
Is available in Inventory?
Yes
No
Store the finished products in High-bay shelf
Dispatch to Sink C
Retrieve the respective product from High-bay shelf
Run Production processes of required product
Dispatch to respective Sink

Figure 3.1 Flowchart of the factory

3.1.1 First Factory Floor Plan

The factory design is mainly divided into three areas, namely Production area, Storage area and Dispatch area. As the names suggest, the Production area is where the products are manufactured. They are forwarded to the Storage area for storing till a customer requests for that product. Finally, when a customer requests for a particular type of product, it is dispatched by the Dispatch area to the customer.

The initially proposed plan is as shown in Figure 3.2

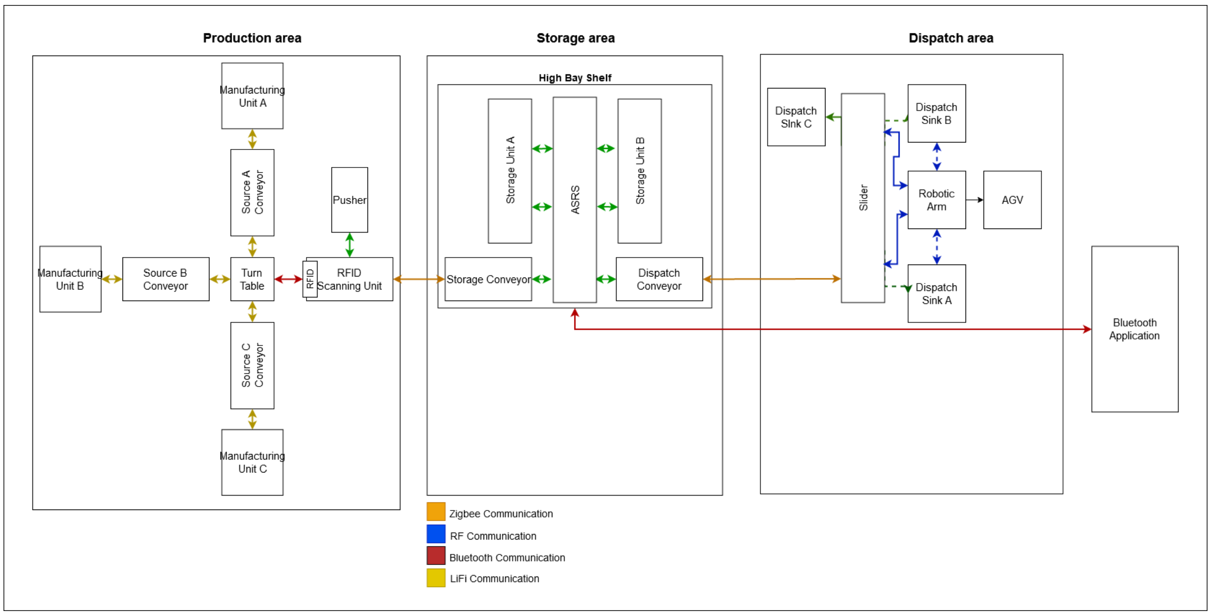


Figure 3.2 First Factory Floor Plan

3.1.2 Second Factory Floor Plan

Due to the limitations of the number of the factory components of each type, availability and compatibility of electronic components corresponding to the proposed communication protocols and the area limitations of the laboratory, the design had to be modified. Detailed descriptions of each area can be found in Section 3.5.

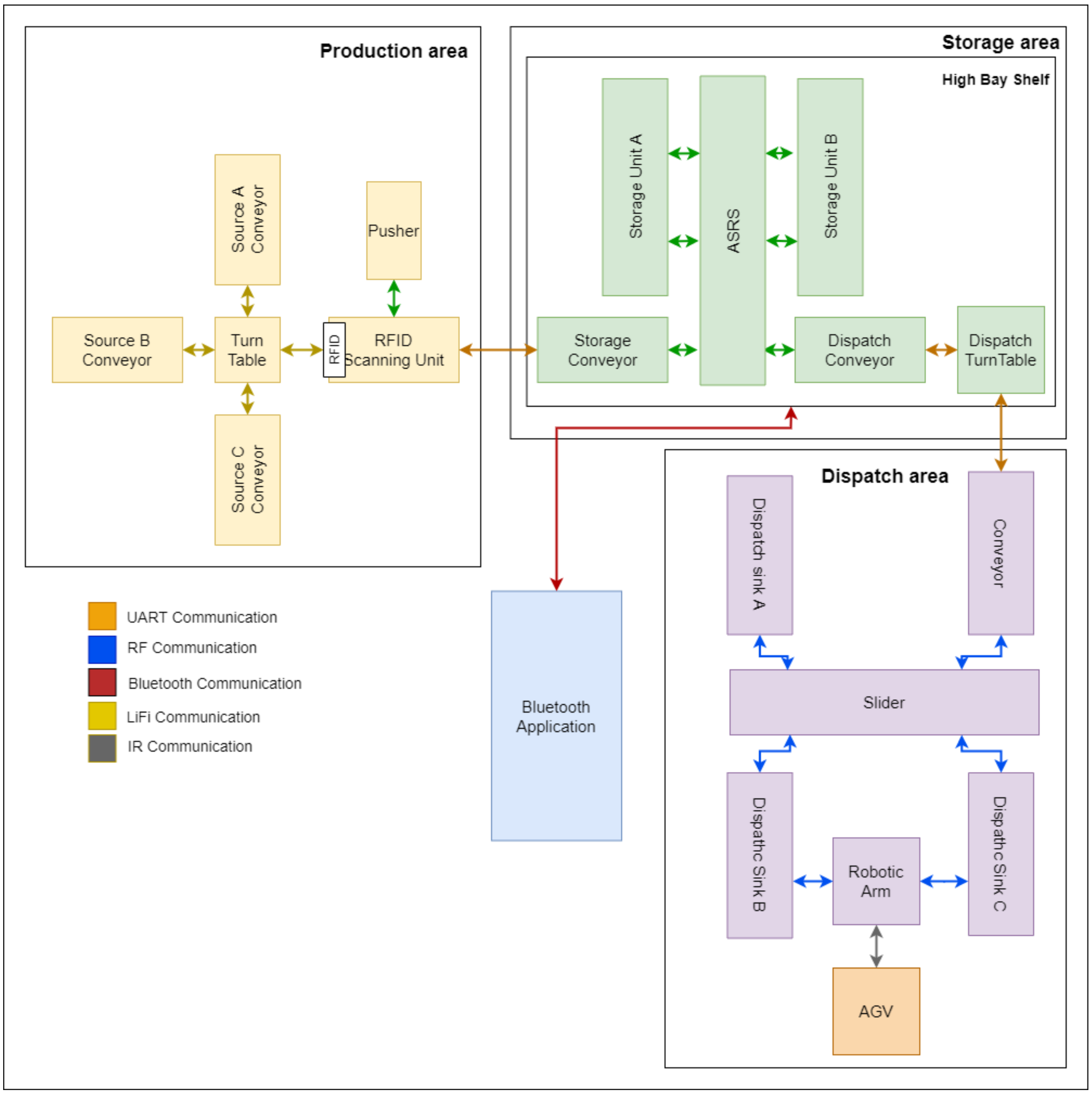


Figure 3.3 Second Factory Floor Plan

3.2 Assembling and testing individual factory components

3.2.1 Hardware setup

The Industry 4.0 learning laboratory has different material flow components such as conveyor, turntable, slider, pusher, high bay shelf, robotic arm, scanner and Automated Guided Vehicle (AGV). These modules must be interfaced with controllers and communication transceivers to use them in material flow operations. These factory modules can be powered with 9-24 V supply [8].

The conveyor systems are used to transport the materials with minimal effort. It is a mechanical structure which uses a belt for transmission. The belt is driven using a motor and the conveyor has two obstacle detection sensors at its ends which depicts the position of materials.

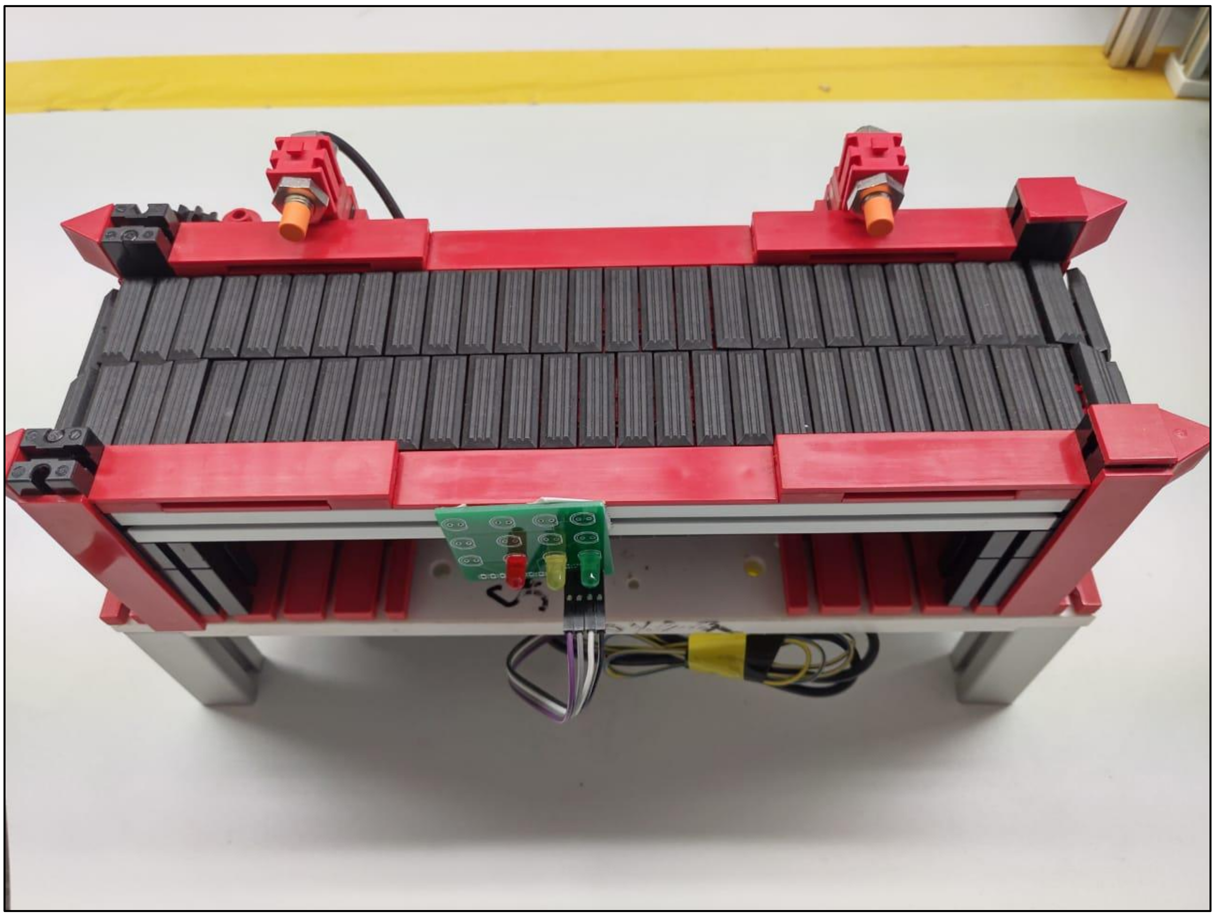


Figure 3.4 Conveyor

The turntable system is used as an intermediate between a maximum of 4 conveyors. The turntable structure has two motors of which one is for linear movement of material and the other motor is for rotating the whole platform. The rotation movement can be detected using two end stops (limit switches) It also has an obstacle detection sensor to detect the presence of the material on the turntable.

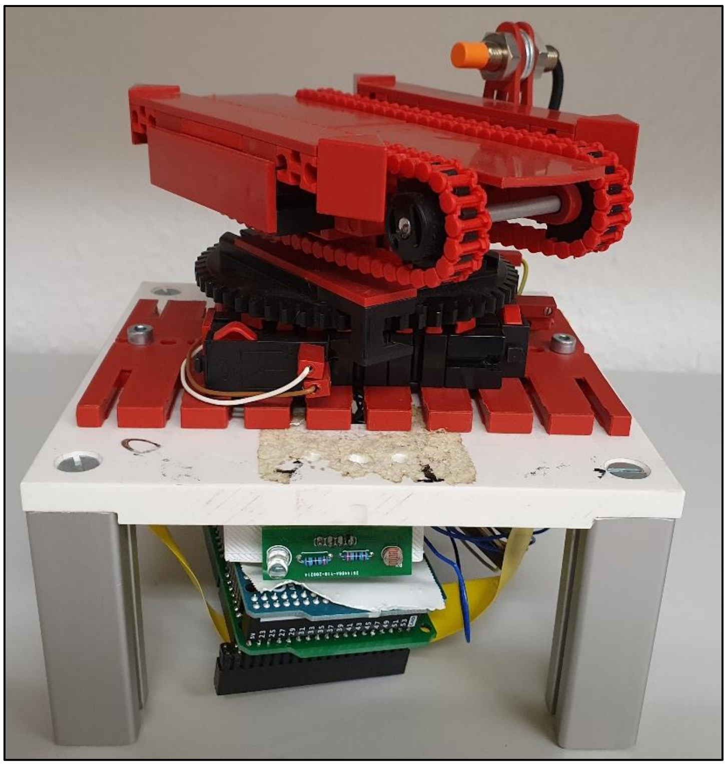


Figure 3.5 Turntable

The slider can be used as an intermediate between maximum of 4 conveyors where two conveyors can be at a farther distance. The slider is designed with two motors which operate linearly but in different axes. The movement of the platform can be controlled using one of the motors and constrained with the help of end stops (limit switches) on each end. It also has an obstacle detection sensor to detect the presence of the material on the slider.

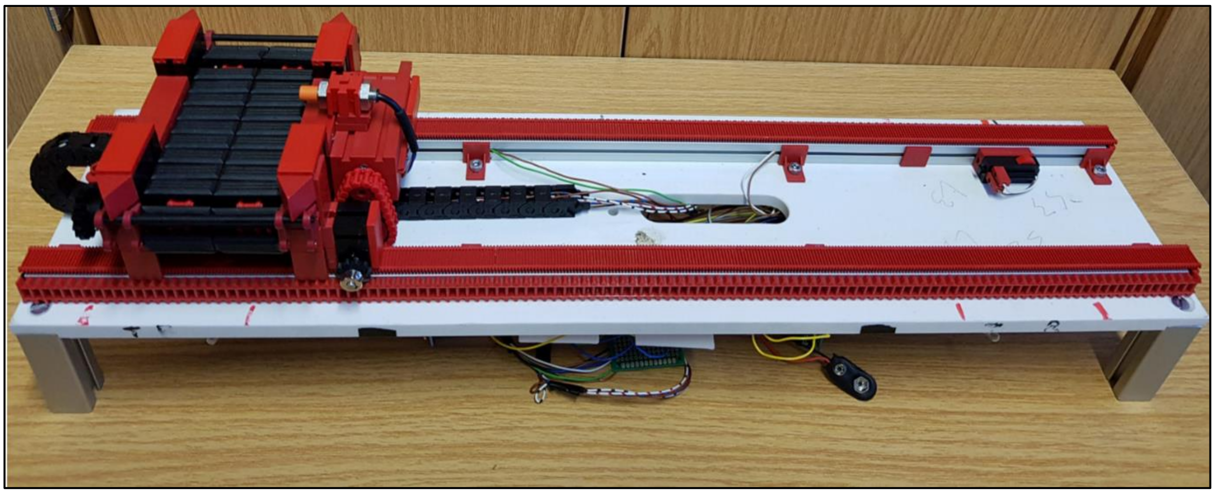


Figure 3.6 Slider

Pusher module is used for transporting the materials with different dimensions. In the current project pusher is used to segregate the damaged materials out of the production process. The pusher has a motor drive which can push the material, once detected with the aid of the obstacle detection sensor on it, to segregate them. It has two end stops for controlling the position of the pusher.

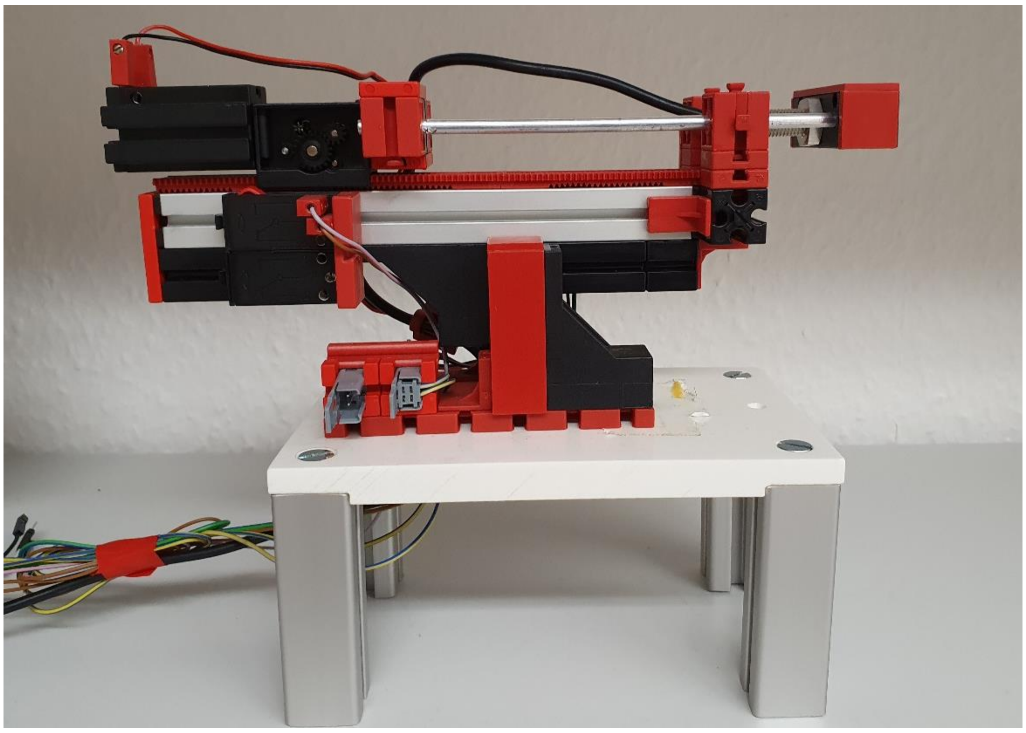


Figure 3.7 Pusher

High bay shelf is used for storage of products in the material flow process. It used conveyor technology to perform the storage and retrieval tasks from the pre-allocated storage spaces. The module has motors, obstacle sensors and limit switches which are used for serving the storage and retrieval needs in the process flow.

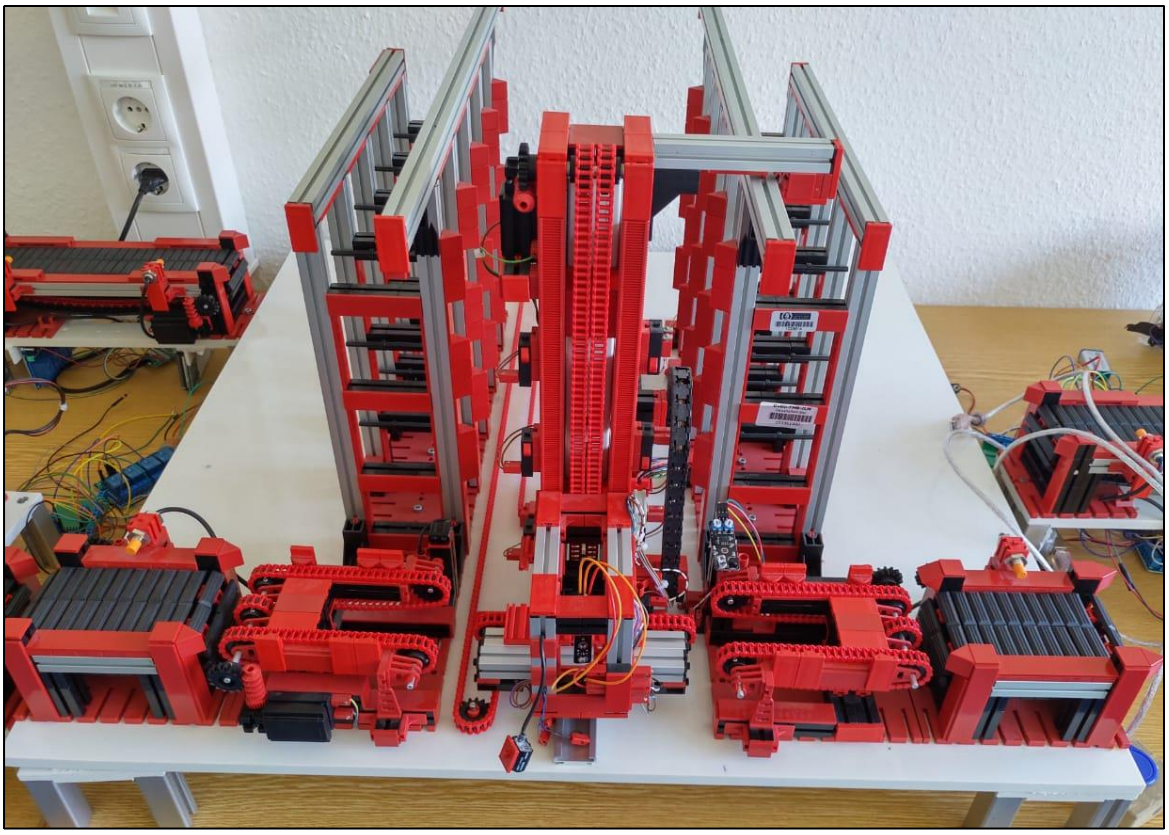


Figure 3.8 High Bay Shelf

Automated Guided Vehicle: An Automated Guided Vehicle (AGV) is used to automate the warehouse process. It can transport the products from one location to another by following a given path. It has an in-built controller (Arduino Leonardo), IR receiver to facilitate communication and a set of IR transmitter and receiver sensors using which it detects the path to follow and motors which provide the locomotion.

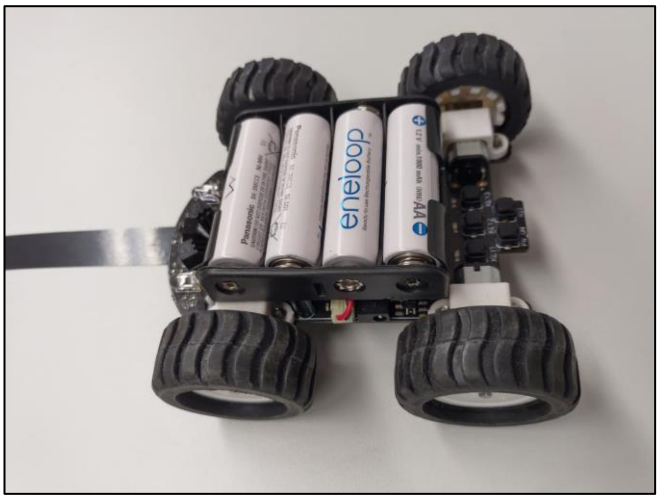


Figure 3.9 AGV

A scanner is used to identify the products by either scanning the QR code or the RFID on the material. The scanner used for the current project is MFRC522 RFID reader.

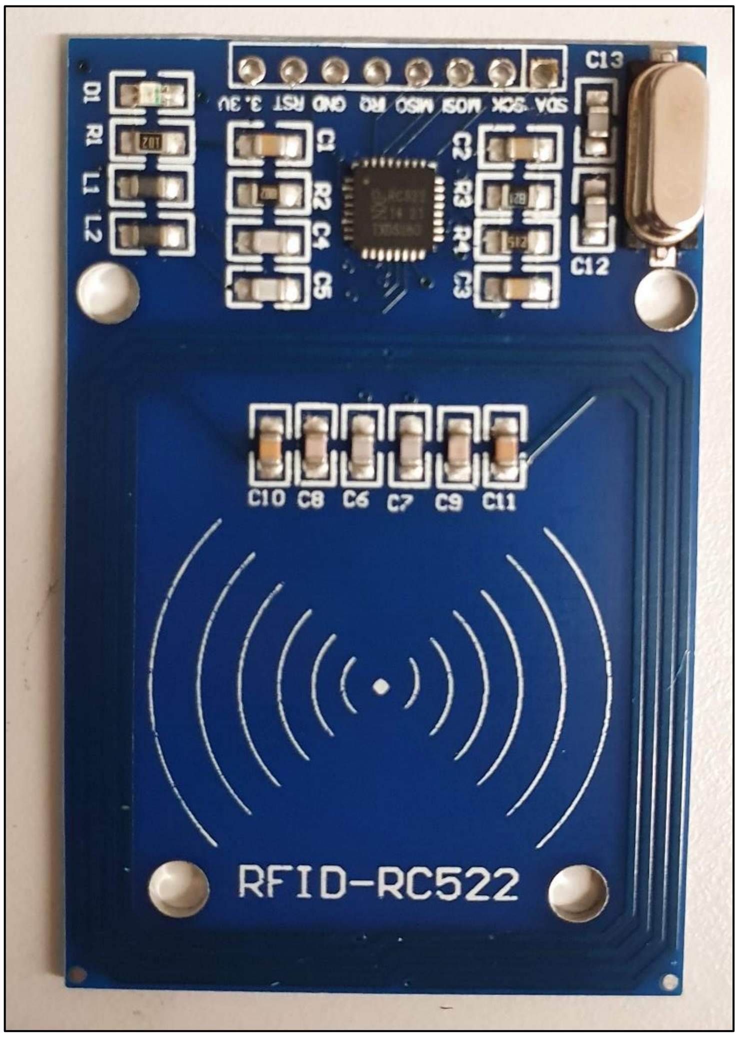


Figure 3.10 RFID Scanner

The robotic arm used is called TinkerKit Braccio which is a 6 DoF robotic arm and can be controlled using an Arduino with the compatible shield that comes along with the arm. The arm is as shown in Figure 3.11 and its specifications are as shown in the Table 3.1

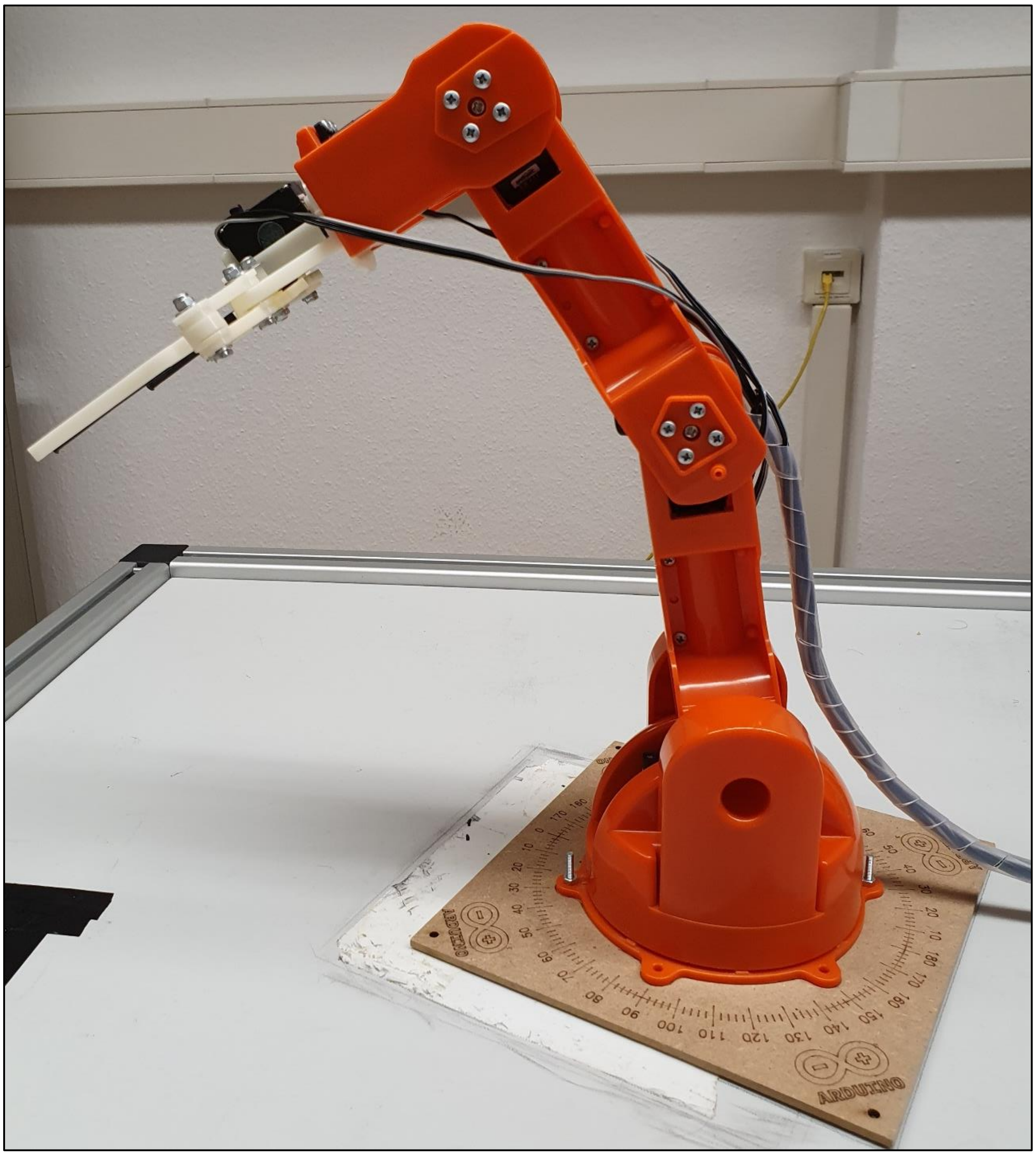


Figure 3.11 Robotic Arm

Axes	6	
Load capacity	Maximum weight of 150g at 32cm of operating distance Maximum weight of 400g at minimum configuration	
Motors	2 x SR 431	4 x SR 311
Torque	12.2 kg-cm @ 4.8V 14.5 kg-cm @ 6.0V	3.1 kg-cm @ 4.8V 3.8 kg-cm @ 6.0V

Speed	0.20 sec/60° @ 4.8V 0.18 sec/60° @ 6.0V	0.14 sec/60° @ 4.8V 0.12 sec/60° @ 6.0V
Rotation angle	180°	
Power	5V DC 4000 mA	

Table 3.1 Specifications of Robotic Arm

The six links of the robot are named base, shoulder, elbow, vertical wrist, rotatory wrist and gripper. Every joint has their own joint limits based on the Braccio arm construction. According to the datasheet [9], the limits of each joint are as shown in Table 3.2

Joint	**Rotation Angle range**
M1: Base	0 - 180
M2: Shoulder	15 - 165
M3: Elbow	0 - 180
M4: Vertical Wrist	0 - 180
M5: Rotatory Wrist	0 - 180
M6: Gripper	10 (Gripper open) - 73 (Gripper close)

Table 3.2 Joint Angle Limits of Robotic Arm

The six motors attached to the six links of the arm must be connected to the provided pins on the shield as shown in Figure 3.12. Each servo motor has a power supply (VCC), a ground (GND) and a signal pin which must be provided with a square wave whose duty cycle, when varied, changes the rotation angle of the servo.

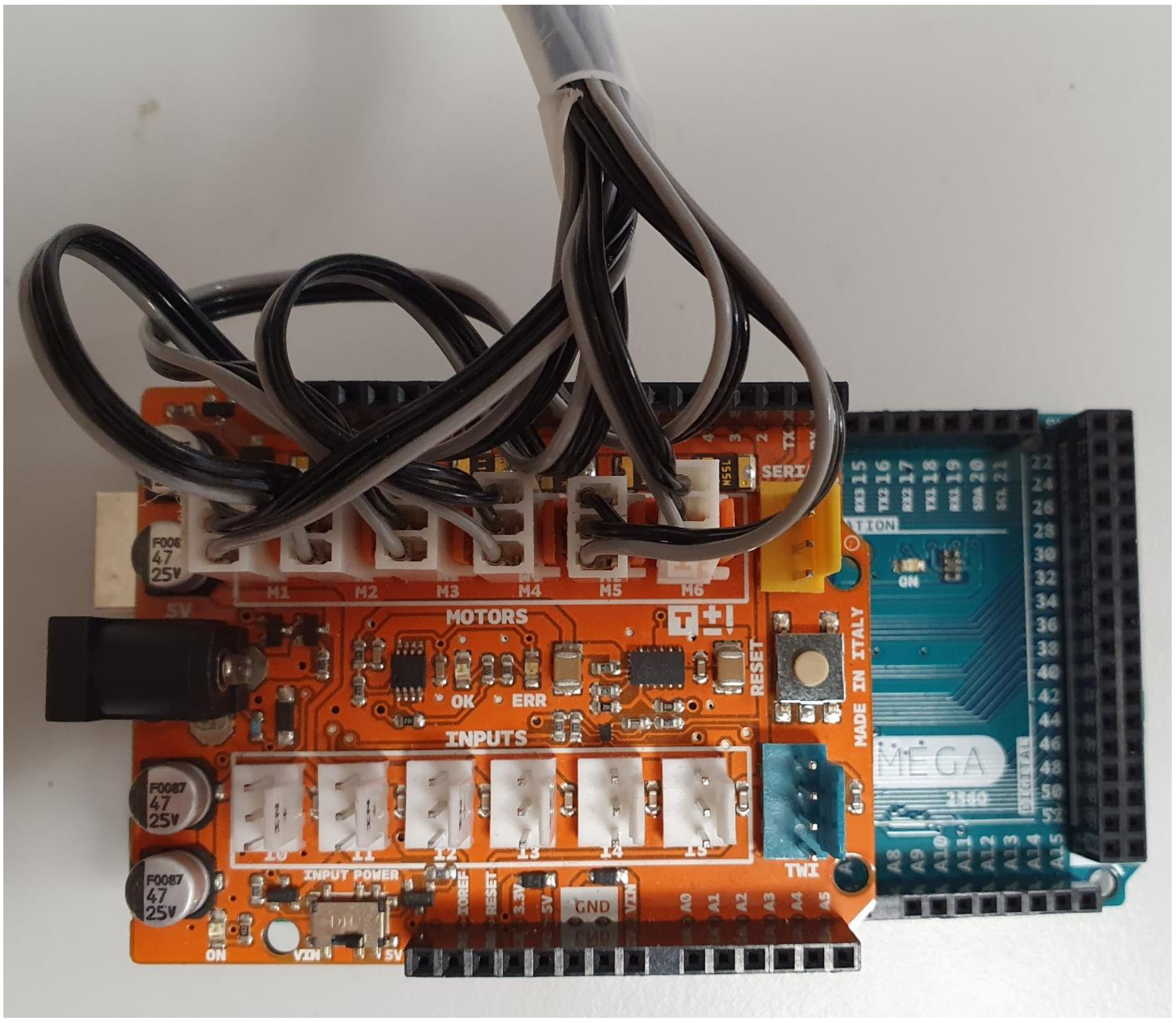


Figure 3.12 Robotic Arm Arduino mountable Shield

Controller: Arduino

Arduino is an open source prototyping platform which has easily usable hardware and software. It is a low-cost microcontroller development board which has open source and extensible software with a simple and clear programming environment. Arduino boards have microcontrollers from 8-bit which run on low power and can be used for various IOT applications. Arduino boards can be adapted for several functions in various products. There are many variants of Arduino boards namely, Arduino Uno, Nano, Pro mini, Mega, Leonardo etc. In the current project we have used three variants of Arduino boards i.e. Arduino Uno, Arduino Mega 2560, Arduino Leonardo.

Specification	Arduino Uno	Arduino Mega 2560	Arduino Leonardo
Controller	AT Mega 328p	AT Mega 2560	ATMega32U4
Digital I/O	14	54	20
PWM	6	14	7
Analog Inputs	6	16	12
UART	1	4	1
SPI	1	1	1
I2C	1	1	1
Flash Memory	32 KB	256 KB	32KB
EEPROM	1KB	4KB	1KB
Clock Speed	16MHz	16MHz	16MHz
Input Voltage	7-12V	7-12V	7-12V
Operating Voltage	5V	5V	5V

Table 3.3 Specifications of Arduino Variants

Circuit Design:

The factory modules conveyor, slider, pusher, turntable contain the same electronic components like sensors, motors and limit switches. So the control system has an Arduino Mega as a processing unit and relays to drive the motors. A printed circuit board is developed to connect the components without complexity. The PCB is used as a bridge between controller and relays, sensors, limit switches. The PCB for high bay shelf is also developed considering the same factors.

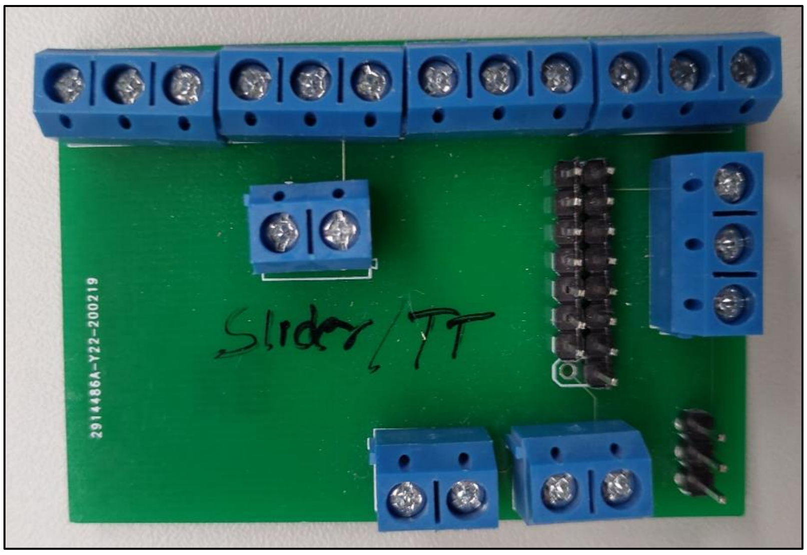


Figure 3.13 PCB First Version

The problem with this design is that factory components cannot justify the purpose of standalone and modularity. So another version of PCB is designed which can be a plug and play shield on Arduino Mega. It has an on board L298N motor driver, so the need of using a relay module is minimised. The PCB has an on-board slot to directly plug in the wireless communication transceivers. The board also has slots for 2 motors, 2 sensors, 2 limit switches, 4 LiFi transceivers, 1 NRF transceiver, 1 Bluetooth module and 1 ESP8266 WiFi module so that each PCB can be used for either conveyor, turntable, slider or pusher.

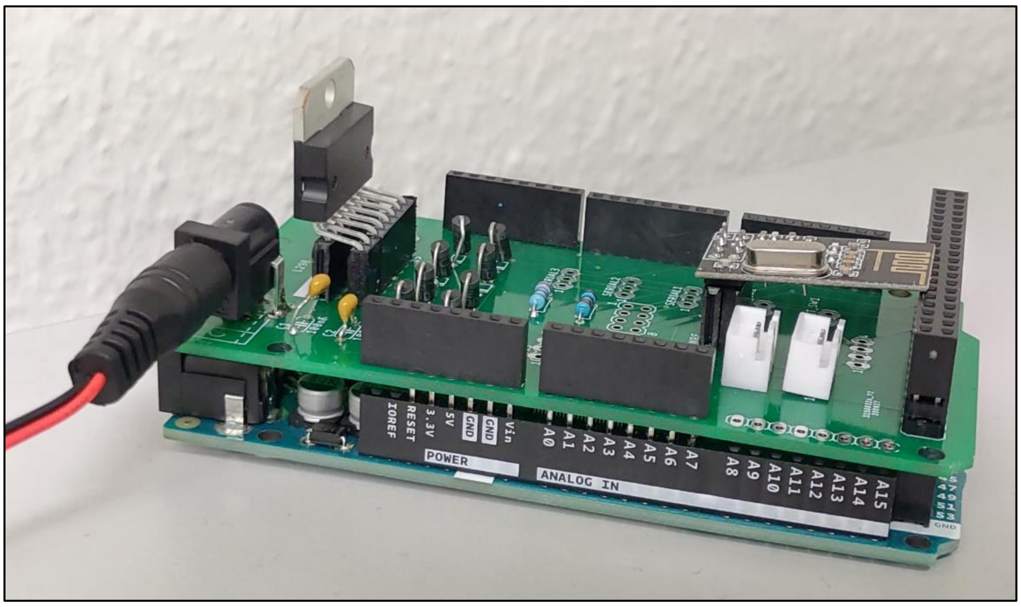


Figure 3.14 PCB Second Version

The PCB for the high bay shelf is also developed in the same process, with the similar interfaces to wireless transceivers just as described in the above section. The high bay shelf has more motors and sensors than other factory components. So the PCB is designed with 4 L298N motors drivers to connect 8 motors and additional slots to connect limit switches of storage shelves. Multiple power slots are also added to the PCB to enable the high power motors to operate at maximum capacity. For this purpose, a 24V supply port has been added, in addition to the 9V Arduino power supply.

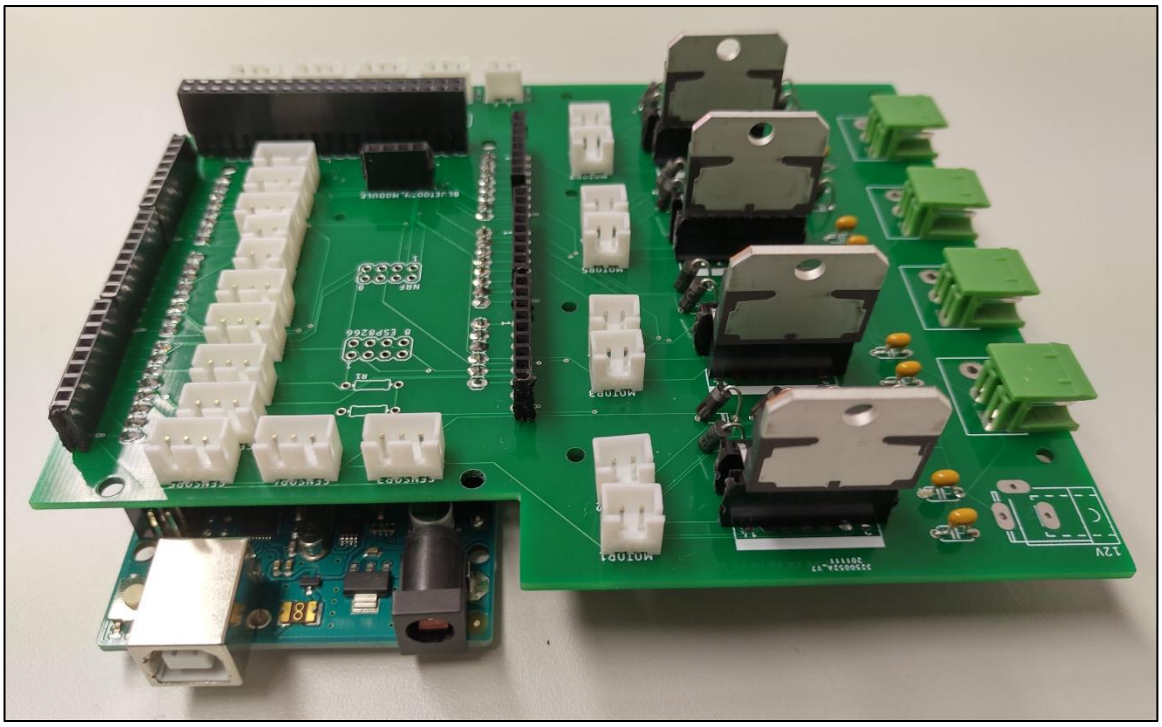


Figure 3.15 High Bay Shelf PCB Second Version

3.2.2 Developing control logic

Proportional-Integral-Differential (PID) controller [10] is a control loop mechanism used to control a system where the control engineer defines the desired behavior or output value (Setpoint) of the system to be controlled and the current state or output value (Process output) is given by the feedback sensors. The current output values are used to calculate the deviation (error) from the desired value. This deviation is minimized by controlling the input (Process variable) of the system. The amount of change needed in the input, for obtaining the desired output is calculated by varying the proportional, integral and differential gains in the controller as shown in Equation 3.1

$$\boxed{\begin{aligned} Error &= CurrentValue - Setpoint \\ Correction &= (Kp * Error) \\ &\quad + (Ki * AccumulatedError) \\ &\quad + (Kd * (Error - PreviousError)) \end{aligned}}$$

Equation 3.1 Correction in PID

Where

- The term Kp is Proportional constant which contributes to the correction due to Proportional block is directly proportional to the error.

$$Correction(Kp) = Kp * Error$$

- The term Ki is Integral constant which contributes to the correction due to the accumulation of the error over time.

$$Correction(Ki) = Ki * AccumulatedError$$

- The term Kd is Differential constant which contributes to the correction due to the difference in the error from the previous time instance and the present time instance.

$$Correction(Kd) = Kd * CurrentError - PreviousError$$

For an AGV, the output variable that can be measured is its deviation from the line it should follow. The position of the AGV is determined by the IR transceivers. The input variable that can be used to control the output is the steering or the turning direction of the AGV which is in turn controlled by the motor speeds. Two methods were tried to implement the PID controller by using these input and output variables.

The 4WD MiniQ Robot [11] which is used as an AGV consists of five pairs of IR Transceivers that are connected to Analog inputs and Motors are connected to the PWM output pins of Arduino Leonardo.

The ATmega32U4 microcontroller present on Arduino Leonardo uses 10-bit analog-to-digital converter (ADC) and 8-bit Pulse-width modulation (PWM), the analog values from the sensors ranges from 0-1023 and the PWM inputs to the motors ranges from 0-255.

The analog readings from the sensors when it is on the black line and in the white space is taken to calculate the mean, which is further used as an input for the PID controller.

Sensor	Analog reading on Black-line	Analog reading on white space	Mean
Left most	480	745	613
left	550	975	763
middle	340	820	580
Right	700	970	835
Right most	935	982	959

Table 3.4 Sensor Readings

The pin declarations for sensors and motors is made as follows:

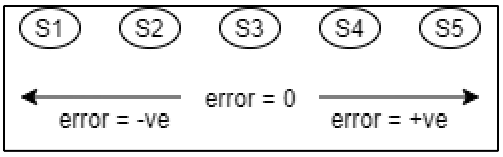


Figure 3.16 Error with respect to sensor positions

a. First approach:

The first approach takes the analog readings of the IR transceivers to calculate the deviation (error) of the AGV from the desired position on the line. This is done by first reading the analog values from all five transceivers. The error is then calculated by the weighted sum of these values as shown in the Equation 3.2. So, suppose the transceivers read a value of 200 when on black line and 900 otherwise, if the AGV's second transceiver from the left is on the black line, then the readings from all the transceivers from left to right would be 900, 200, 900, 900, 900. And hence, the error value would be +700. Similarly, if the AGV's fifth transceiver from the left is on the black line, then the readings from left to right would be 900, 900, 900,900, 200, and the error value would be -1400.

$$\boxed{Error = (2 * S1) + (1 * S2) + (0 * S3) - (1 * S4) - (2 * S5)}$$

Equation 3.2 Error from weighted sum of sensor readings

Where, S0 is the analog reading of the first sensor from the left, S1 is second from the left and so on.

So, more positive the error, more is the position of the AGV towards the right of the line and zero indicates that the AGV is positioned correctly. So the setpoint for this approach would be error = 0 and the PID controller must be designed such that the error is always maintained at zero by varying the speeds of the left and right motors which change the direction the AGV moves towards. But this approach might not work effectively when the transceivers do not read the same value when placed on black like when first transceivers reads 700 when placed on black and 800 otherwise while the third transceivers reads 200 when placed on black, but reads 500 when placed on white area. One possibility to avoid this issue would be to normalize the readings from all the transceivers by mapping the upper value (when on black line) and the lower value (when on white area) to fixed values for all the transceivers.

b. Second approach:

In this approach, we first read the analog values of each transceiver when placed on black line and on white area. Then, the average of the high and low values is decided as the threshold (T) for each transceiver so that if the value read by a transceiver is lesser than the threshold set for that particular transceiver, then it can be decided that the transceiver is on the black line, else on the white area. The resulting decision of whether the transceivers are on black line or on the white area, we can decide the error as shown in the Table 3.5

Sensor readings					**Error**
S1 (Left most)	**S2** (Left)	**S3** (Center)	**S4** (Right)	**S5** (Right most)	
< T (Black)	> T (White)	> T (White)	> T (White)	> T (White)	-4
< T (Black)	**< T (Black)**	> T (White)	> T (White)	> T (White)	-3
> T (White)	**< T (Black)**	> T (White)	> T (White)	> T (White)	-2
> T (White)	**< T (Black)**	**< T (Black)**	> T (White)	> T (White)	-1
> T (White)	> T (White)	**< T (Black)**	> T (White)	> T (White)	0
> T (White)	> T (White)	**< T (Black)**	**< T (Black)**	> T (White)	1
> T (White)	> T (White)	> T (White)	**< T (Black)**	> T (White)	2

> T (White)	> T (White)	> T (White)	**< T (Black)**	**< T (Black)**	3
> T (White)	> T (White)	> T (White)	> T (White)	**< T (Black)**	4

Table 3.5 Error values based on position of AGV on the line

So, in this approach, the error is more negative when the line is more towards the left of the AGV, more positive when the line is more towards the right side of the AGV and zero when the line is exactly at the center of the AGV as shown in Table 3.5

The second approach is used in this project as the normalization technique shown in the first approach was not much effective when implemented.

After calculating the error, the corresponding compensation in motor speeds is to be calculated so that the line is always at the center of the AGV. For example, when the black line is on the left side of the AGV, error is negative, then the compensation must be such that the AGV turns towards the left side. This is done by slowing down the left motor and speeding up the right motor. The left motor can be rotated in the reverse direction to turn more angle in a shorter time. To achieve this logic, the following formulae are used to determine the speeds of the left and the right motors.

$$Speed_{left} = Speed_{base} - Correction$$
$$Speed_{right} = Speed_{base} + Correction$$

Equation 3.3 Speed Correction due to PID

Where $\text{Speed}_{\text{left}}$ is the left motor speed,

$\text{Speed}_{\text{right}}$ is the right motor speed,

$\text{Speed}_{\text{base}}$ is the base motor speed

Since we are using a PID controller, the correction is calculated using the formula shown in Equation 3.4

$$Correction = (Kp * error) + (Ki * \Sigma\ error) + (Kd * \Delta\ error)$$

Equation 3.4 Total Correction in PID Controller

As discussed earlier, if the error is positive, the AGV must be turned left by slowing down the left motor and speeding up the right motor. The same can be observed in the speed equations Equation 3.3 as correction is positive for positive error and negative for negative error.

The Kp, Ki and Kd values vary from system to system and also based on the inputs and the desired output (Setpoint) of the system. Generally, to obtain the suitable PID parameters, the transfer function of the system is used, but because we do not know the transfer function of the AGV, we shall use a manual tuning algorithm.

In manual tuning algorithm, first all the parameters are set to zero and the base motor speed ($\text{Speed}_{\text{Base}}$) is set to half of the maximum speed of the AGV. Then Kp is increased until there the AGV moves with oscillation on the line. Kp is fixed at this point and Ki is slowly increased until the AGV stops oscillating after a desired amount of time when placed on the line with offset. Too high Kp can cause the AGV to oscillate alot, thereby making it miss the line and become unstable. Then, fix Kp and Ki at these values and increase Kd until the stabilizations time is under the acceptable limit.

Several values for Kp, Ki and Kd are tried with different base motor speed values. The best results with the fastest motor speeds, at 240, were observed with 120, 0.1 and 120 as Kp, Ki and Kd values respectively. But at this speed, due to inertia, the AGV might not be able to turn sharply and might not go as desired at junctions. Also, the product placed on the AGV might fall off. So, the optimum speed of 100 is chosen with Kp, Ki and Kd values as 50, 0.1 and 50 respectively. Also, if Ki values is non-zero, as it contributes to correction due to the accumulation of error, if the AGV goes off the line due to some technical problems, the correction due to Integral factor keeps increasing continuously and so, the AGV is observed to become highly unstable. For this reason, Ki is set to zero while Kp and Kd are set to 50 each for the motor base speed of 100. The closed loop control system of the PID controller looks as shown in Figure 3.17 PID Controller Block DiagramFigure 3.17

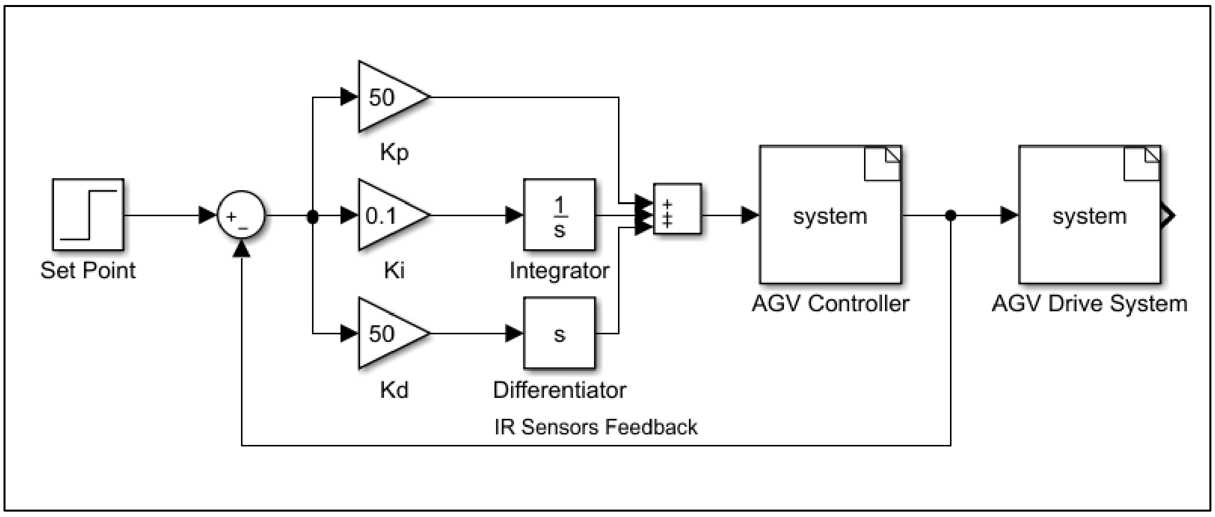


Figure 3.17 PID Controller Block Diagram

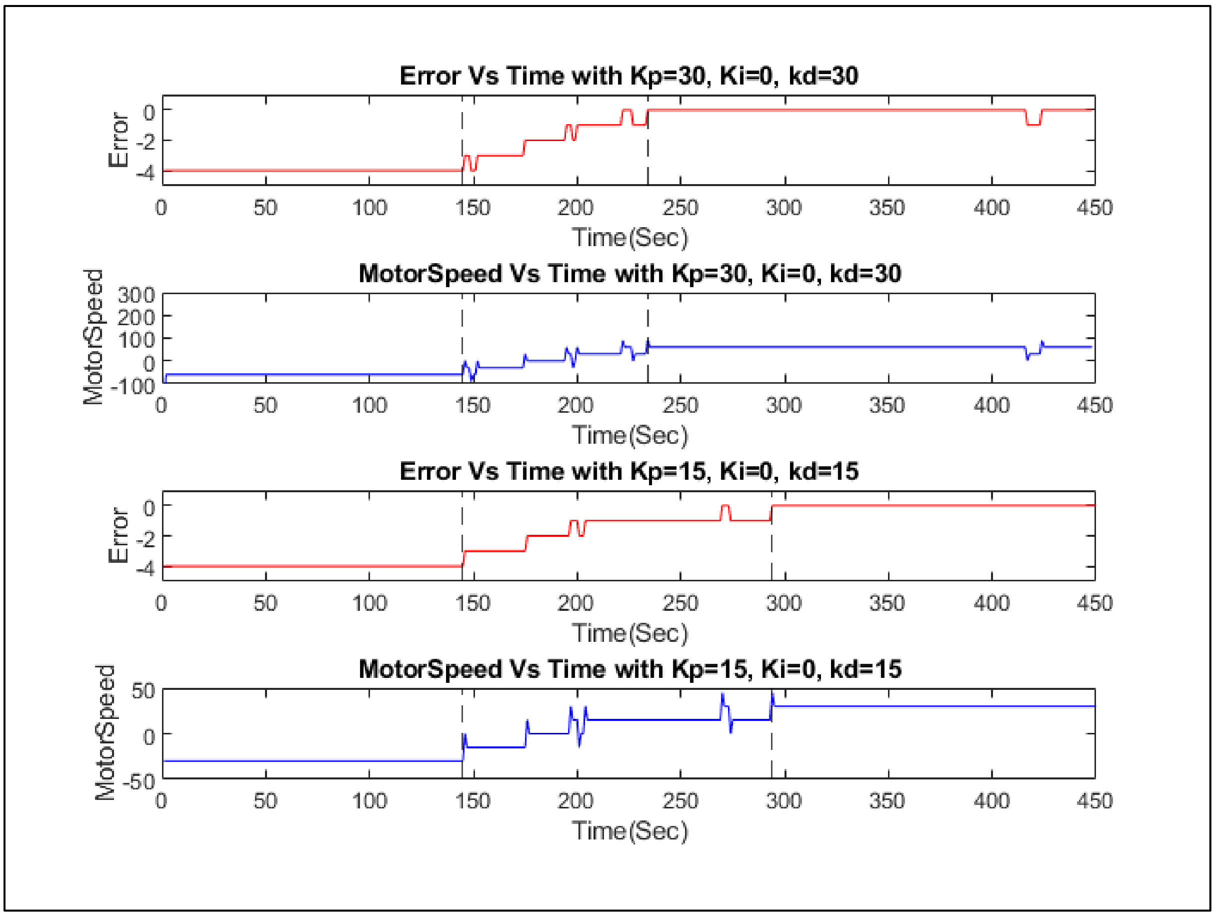


Figure 3.18 Comparison of PID controller with different parameters

Figure 3.18 shows error values and corresponding correction values for two different combinations of Kp, Ki and Kd with a base motor speed of 60. The first two graphs are plotted for Kp, Ki and Kd values of 30, 0 and 30 respectively while the second set of graphs are plotted when Kp, Ki and Kd are set to 15, 0 and 15. For testing the PID controller and plotting the graph, the AGV is placed with the line on the leftmost sensor which is why the error value is initially -4 in the figure. At the first dotted vertical line, the AGV starts moving such that the line is at it's center and so we can observe the error reaching zero at the second dotted vertical line. Also, it can be clearly observed that the AGV takes less time when Kp, Ki and Kd are 30, 0 and 30 respectively when compared to that of Kp, Ki and Kd set to 15, 0 and 15 respectively.

The following pin declarations are to be made to use the motors and the sensors on the 4WD MiniQ Robot

```
void setup() {
  pinMode(5, OUTPUT);//control speed(Left motors)
  pinMode(12, OUTPUT);//control direction of rotation(Left motors)
  pinMode(6, OUTPUT);//control speed(Right motors)
  pinMode(7, OUTPUT);//control direction of rotation(Right motors)

  pinMode(A0, INPUT);// IR sensor 1
  pinMode(A1, INPUT);// IR sensor 2
  pinMode(A2, INPUT);// IR sensor 3
  pinMode(A3, INPUT);// IR sensor 4
  pinMode(A4, INPUT);// IR sensor 5
}
```

The generated **Correction factor** is applied to the motors as follows:

```
analogWrite(5, baseMotorSpeed + correction);// speed control Left                    motors
digitalWrite(12, HIGH);                              // clockwise rotation

analogWrite(5, baseMotorSpeed + correction);
digitalWrite(12, LOW);                              // anti-clockwise rotation



analogWrite(6, baseMotorSpeed - correction); // speed control of Right motors
digitalWrite(7, HIGH);                               // clockwise rotation

analogWrite(6, baseMotorSpeed - correction);
digitalWrite(7, LOW);                               // anti-clockwise rotation
```

Braccio Arm:

A library named *Braccio.h* [12], provided along with the Tinkerkit Braccio Robot is used for easier implementation of the robotic arm. It must be used along with the library named *Servo.h* [13] which is used to control the rotation angle of the servo motor by varying the duty cycle of the PWM signal provided to the servo motor. Both these libraries can be installed from Arduino Library Manager in Arduino IDE or can be found in their respective git repositories.

The *Braccio.h* library consists of two important functions, namely *Braccio.begin()* and *Braccio.ServoMovement()*. *Braccio.begin()* must be called at the beginning of the program. But before that, the motors of the arm must be defined as *Servo* objects as shown in Table 3.6 so that these corresponding motors are assigned to appropriate PWM pins of the Arduino controller. Default connections can be found in the documentation of the *Braccio.h* library [12]. Once the motor outputs and the arm are initialized, the movement of the arm can be performed using the function *Braccio.ServoMovement()* which takes seven input arguments. First input argument to it is the delay in milliseconds which is applied after every degree change in the joint angles of the arm. Integer values between 20 and 30 milliseconds, including 20 and 30 are allowed in this parameter. The next six arguments are integer values of the desired angles of each axis of the arm in the order, from base to gripper. Only the values shown in the table above (Table 3.2) are allowed for respective join angles.

A sample code for moving the arm with all the joints from one extreme angle to the other extreme angles (from Table 3.2) is shown below.

```
#include <Braccio.h>
#include <Servo.h>

Servo base;
Servo shoulder;
Servo elbow;
Servo wrist_ver;
Servo wrist_rot;
Servo gripper;

void setup() {
        Braccio.begin();
}

void loop() {
                                  //delay, M1, M2,  M3,  M4,  M5, M6
        Braccio.ServoMovement(20,   0,  15, 180, 170,   0, 73);
        delay(1000);

        Braccio.ServoMovement(20,  180, 165,   0,   0, 180, 10);
        delay(1000);
}
```

Table 3.6 Basic Program for Robotic Arm

So the code shown in Table 3.6 declares all the six servo motors with the specified names and initializes the robot using *Braccio.begin()* function in the *setup()*. In the *loop()*, every axis of the robot is rotated from one extreme angle value to the other extreme value with 20 milliseconds delay between each degree change in the join angles and one second delay after reaching one of the specified configurations. Similarly, any joint angles in the allowed range as shown in Table 3.2 can be specified and the robot gets set to the specified configuration.

High Bay Shelf

The basic logic for storage of a palette in a specified location and retrieval from the same location is developed and tested. The sensors on each conveyor are continuously read and based on its output the conveyor's movement is controlled until it reaches storage and retrieval belt. Once the palette is inside the belt it moves forward and stops until the switch at the specified rack is detected. The belt is moved up or down depending on whether the palette should be retrieved or stored respectively for a given location. Once the storage and retrieval algorithm for one location is tested then the logic is developed for all the locations of the high bay shelf.

```
#define sensor_conveyor1 4  //sensor signal on D4(conveyor towards production area)
#define sensor_conveyor2 10 //sensor signal on D10
#define sensor_asrs      44 //sensor signal on D44
#define sensor_turntable 28 //sensor signal on D28

#define turntable_limit_Switch1 20 // limit switch 1 on D20
#define turntable_limit_Switch2 21 // limit switch 2 on D21

#define high_shelf_limit_Switch1 27 //limit switch on D27 towards the open end
#define high_shelf_limit_Switch2 47 //limit switch on D47
#define high_shelf_limit_Switch3 45 //limit switch on D45
#define high_shelf_limit_Switch4 43 //limit switch on D43
#define high_shelf_limit_Switch5 41 //limit switch on D41 towards the conveyors

#define Y_Left_Top    33    // limit switch on D33 ASRS_YAxis_Left side Top Limit Switch
#define Y_Left_Middle 31    // limit switch on D31 ASRS_YAxis_Left side Middle Limit Switch
#define Y_Left_Bottom 29    // limit switch on D29 ASRS_YAxis_Left side Bottom Limit Switch
#define Y_Right_Top    38    // limit switch on D38 ASRS_YAxis_Right side Top Limit Switch
#define Y_Right_Middle 48    // limit switch on D48 ASRS_YAxis_Right side Middle Limit Switch
#define Y_Right_Bottom 40    // limit switch on D40 ASRS_YAxis_Right side Bottom Limit Switch
#define X_Left 37     // Limit switch on D37 ASRS_XAxis_Left limit switch
#define X_Middle 35   // Limit switch on D35 ASRS_XAxis_Middle limit switch
#define X_Right 2     // Limit switch on D2 ASRS_XAxis_Right limit switch

#define motor1_p 8  // motor1 + on D8  Conveyor1 towards production area
#define motor1_n 9  // motor1 - on D9
#define motor2_p 12 // motor2 + on D12 Conveyor 2
#define motor2_n 11 // motor2 - on D11
#define motor3_p 30  // motor3 + on D30 ASRS X Axis Motor
#define motor3_n 32  // motor3 - on D32
#define motor4_p 42  // motor4 + on D42 Conveyor towards dispatch area
#define motor4_n 46  // motor4 - on D46
#define motor5_p 22  // motor5 + on D22 Turntable linear motor
#define motor5_n 23  // motor5 - on D23
#define motor6_p 3   // motor6 + on D3  Turntable rotation motor
#define motor6_n 13  // motor6 - on D13
#define motor7_p 24  // motor7 + on D24 ASRS Y Axis Motor
#define motor7_n 26  // motor7 - on D26
#define motor8_p 34  // motor8 + on D34 ASRS Z Axis Motor
#define motor8_n 36  // motor8 - on D36
```

Scanner

MFRC 522 RFID reader is used to scan the tags (on palettes), this reader is interfaced using Serial Peripheral Interface (SPI) communication to the Arduino. The RFID reader scans the ID on the tag and sends the signal through the SPI channel to Arduino Mega.

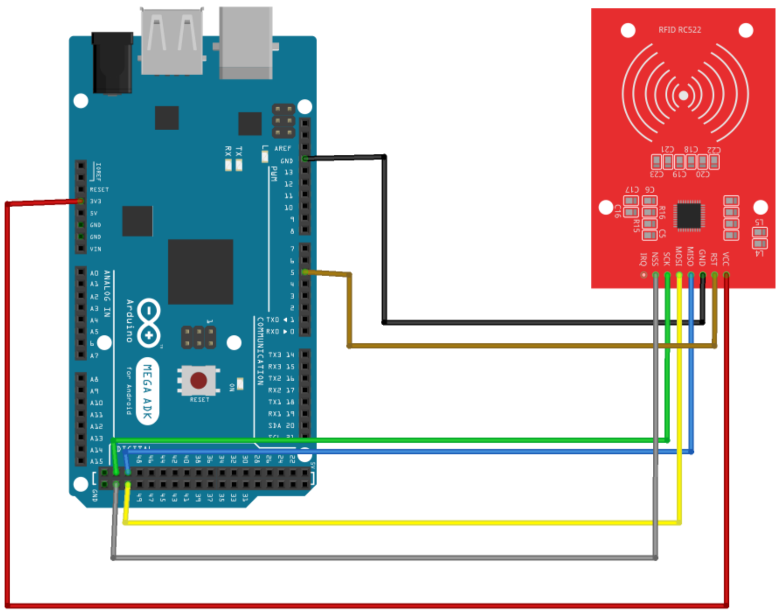


Figure 3.19 RFID Scanner - Arduino Interface Circuit

The following header files and function must be used to declare the RFID reader

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 53
#define RST_PIN 5
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

The control logic for detecting RFID tags:

```
if ( ! mfrc522.PICC_IsNewCardPresent()) // Look for new cards
  {
    return;
  }

  if ( ! mfrc522.PICC_ReadCardSerial())  // Select one of the cards
  {
```

```
  return;
 }
```

Factory Modules:

The PCB for conveyor, turntable, slider and pusher is a plug-and-play shield on Arduino Mega 2560. The device serves as standalone and can be powered with a 9-12V adapter.

The following pin declarations must be used for shield for sensors and motors.

```
void setup()
{
    pinMode(sensor1,INPUT); //sensor1 signal on D2
    pinMode(sensor2,INPUT); //sensor2 signal on D3

    pinMode(limit_Switch1,INPUT_PULLUP); //limit switch 1 on D30
    pinMode(limit_Switch2,INPUT_PULLUP); //limit switch 2 on D32

    pinMode(motor1_p,OUTPUT);  // motor1 + on D6
    pinMode(motor1_n,OUTPUT);  // motor1 - on D9
    pinMode(motor2_p,OUTPUT);  // motor2 + on D10
    pinMode(motor2_n,OUTPUT);  // motor2 - on D11
}
```

3.3 Integrating communication protocols

3.3.1 Hardware setup

LiFi:

Since the LiFi module must be a transceiver module, there needs to be a transmitter and a receiver related hardware or circuit. The transmitter is merely an LED whose positive and negative terminals are connected to a digital pin and Ground pin of a controller respectively. A resistor, based on the Digital I/O voltage levels and the current ratings of the LED, generally of the value 220 or 270 Ohms for 5V Digital I/O voltage level of the controller and a current rating of 25mA to 35mA, is added in series to the LED between the digital pin and the ground pin of the controller to limit current as shown in theFigure 3.20 LiFi module - Arduino Interface Circuit Figure 3.20. A Light dependent resistor (LDR) is used as the receiver whose one terminal is connected to VCC of the controller and the negative terminal is connected to the ground pin with a resistor in series, generally a 220 or 270 ohms based on the current ratings of the LDR and the I/O logic voltage of the controller, for limiting current. The terminal of the LDR which is connected to one of the terminals of the current limiting resistor is connected to one of the analog pins of the controller, A0 in this case. This circuit acts as a voltage divider and the voltage read by the analog pin varies based on the variable resistance of the LDR as its resistance depends on the light that falls on its surface.

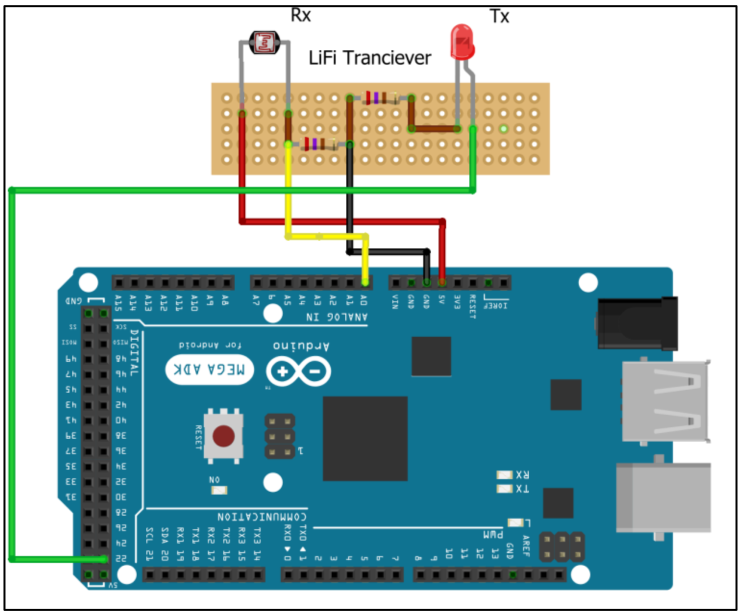


Figure 3.20 LiFi module - Arduino Interface Circuit

A PCB was designed and printed as shown in Figure 3.21 based on the same circuit shown in Figure 3.20. In the depicted view i.e. with the transmitter or LED on the left, the description of the pins on the PCB and their connection to the controller are mentioned in Table 3.7.

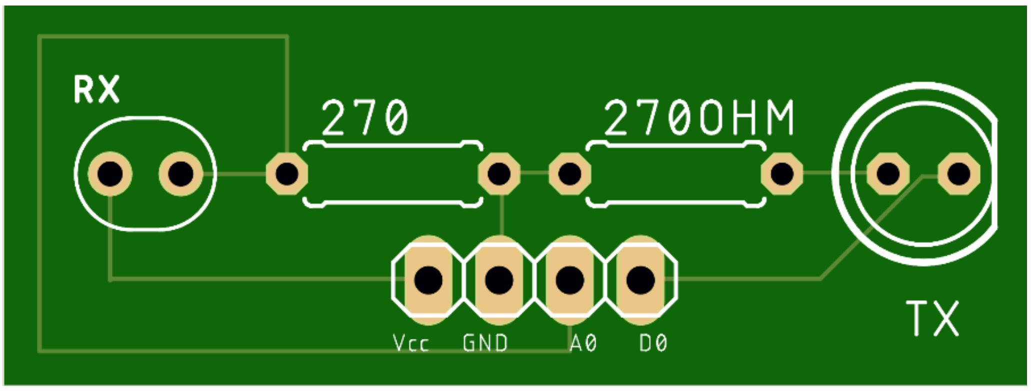


Figure 3.21 PCB for LiFi module

Pin number (from left)	Pin description	Connection to controller
1	LED positive	Digital pin (22)
2	LDR output	Analog pin (A0)
3	Ground	Ground (GND)
4	LDR VCC	VCC

Table 3.7 LiFi - Arduino Connections

Bluetooth

The HC-05 module uses a USART circuit for full-duplex communication. The Enable pin is used to toggle between data mode and AT command mode. The Tx and Rx pins are used for serial communication and the VCC pin is used to power the circuit with a 5V supply. The GND pin is connected to the system ground. For USART serial communication with a controller, the Tx pin is connected to the Rx of the controller, and vice versa. A State pin is internally connected to an onboard LED and can be used for feedback from the module.

The PCB designed for the factory modules consists of a 6-pin connection port to incorporate the HC-05 module. It can be directly plugged onto the PCB. The names of the pins and their connections can be found in Table 3.8.

HC-05 pin	Arduino
VCC	5V
GND	GND
Tx	Rx
Rx	Tx
EN/KEY	5V/GND
State	Unconnected

Table 3.8 HC-05 - Arduino Connections

Infrared (IR):

An IR Transmitter is connected to Arduino Uno as shown in Figure 3.22. The positive end of the IR LED is connected to PWM pin 3 via a 330 Ohm resistor and the negative end to the ground (GND) pin on the Arduino Uno.

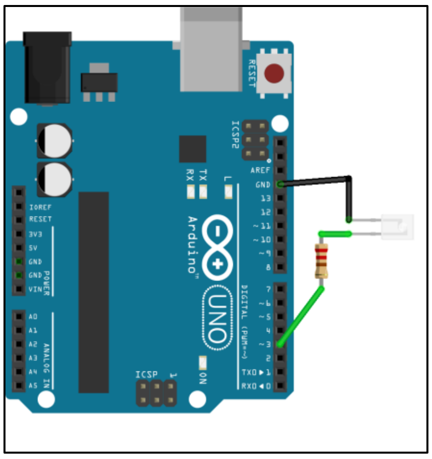


Figure 3.22 IR - Arduino Interface Circuit

The IR Receiver is located on the AGV (Automated Guided vehicle) which is connected to pin D17 (SS/RXLED) on built-in board Arduino Leonardo of the 4WD MiniQ Robot [11].

NRF:

The NRF24L01+ modules use SPI as an interface with the Arduino. The supply voltage is 3.3V which is readily available on the Arduino Mega, eliminating the need for a potential divider circuit. The Chip Enable and Chip Select pins on the module can be initialized using software and the Arduino pins used in this setup are 7 and 8 respectively. The MOSI and MISO pins are used to configure the master/slave mode of the module and are connected to pins 51 and 50 on the Arduino respectively. The Serial Clock pin needs to be supplied with the clock pulses, the time period of which will be the communication rate between the NRF module and the Arduino. It is connected to pin 52 on the Arduino.

The PCB designed for the factory modules consists of an 8-pin connection port to incorporate the NRF module. It can be directly plugged onto the PCB. The names of the pins and their connections can be found in the table below.

NRF24L01+ pin	**Arduino pin**

VCC	3.3V
GND	GND
CE	7
CSN	8
MISO	50
MOSI	51
SCK	52

Table 3.9 NRF24L01+ - Arduino Connections

3.3.2 Developing control logic

LiFi: Arduino logic, RIOT parallel threading

The main idea for transmitting a message using LiFi is to blink the LED, which is the transmitter of a LiFi Transceiver, according to the bits of the binary message that is to be transmitted and read the blinks from the other LiFi module using the LDR. For this to happen, since the controllers are discrete systems and have certain clock frequency they work with, we need to either synchronize the controller with the transmitter circuit with the controller of the receiver circuit or use asynchronous transmission techniques that can retrieve the clock from the message signals. So, Manchester coding is used for this purpose, as clock signal can be retrieved from the message signal encoded using Manchester technique.

Initially, the logic was planned to be implemented on Arduino with Arduino Programming Language [14] to keep the programming environment consistent throughout the Factory layout. But in certain situations, for certain factory modules, more than one LiFi transceiver might be needed in which case Full-duplex communication cannot be achieved with the given specification even with the use of the internal Timers and Interrupts of the controllers. When one transceiver is transmitting data, the controller is busy blinking the LED throughout the duration of transmission and cannot be able to receive any message from the other transceivers, or even its own receiver. The solution for this is to implement parallel threading technique where each transceiver reception and transmission codes run on a thread. As a result, all the transceivers can work independently and parallelly. For this reason, RIOT OS is implemented with the chosen Arduino controller and the techniques and software algorithm shown in [] (Add Vasu thesis reference) (TBA) were used.

Bluetooth and UART

The HC-05 module can be configured with the standard Serial library functions for communicating with an Arduino. The same functions are used for UART communication using the Serial ports of the Arduino. The function signatures and descriptions are provided in the table below.

Function	Description
begin(baudRate)	Starts the operation of the module with the specified baud rate
available()	Checks whether payload is available for reading
read()	Reads the available payload
write(payload)	Writes the specified payload to the controller

Table 3.10 UART functions

The operation always starts by calling the *begin()* function. In the read mode of operation, the *available()* function is first called in a loop to halt the controller processing until data is available. The *read()* function is then called to read the payload and store for processing. In the write mode, the *write()* function is used to write data back to the controller in the case of the HC-05 module, or to another device or controller in the case of UART communication.

IR:

IR communication takes place by blinking of the IR-LED according to the Transmission Bits at a certain frequency. *IRRemote.h* library [15] is used for sending and receiving messages using the IR Transmitter and IR receiver respectively. The message decode type can be one of the following types: NES, SONY, RC5 and RC6.

IR Transmitter:

The message to be transmitted is created. A pin is fixed for transmission depending on the timer the library is using. Here the IR Transmitter is connected to the PWM pin 3 on Arduino Uno. A transmit object is created using the *IRsend* class and *irsend.sendSony(message)* sends the given message.

```
#include <IRremote.h>

IRsend irsend;

void setup()
{
     pinMode(3,OUTPUT); // IR Transmitter is connected to PWM pin 3
}

void loop() {
   irsend.sendSony(0xa80, 12); //(message, numBits)
}
```

IR Receiver:

On the other end, the IR Receiver on the AGV is connected to pin 17 of the Arduino Leonardo. *irrecv.enableIRIn()* function initiates the receiving operation, *irrecv.decode(&results)* decodes the message received and *results.value* is used to access the received message.

```
#include <IRremote.h>

int RECV_PIN = 17;

IRrecv irrecv(RECV_PIN);

decode_results results;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn();  // Start the receiver
 }
void loop() {
   if (irrecv.decode(&results)) {        // Decode received message
      Serial.println(results.value, HEX); // Print Message received
   }
   results.value = 0;
   irrecv.resume();      // Receive the next message
}
```

Note: The *Servo.h* (PWM) and *IRRemote.h* (IR) libraries use the same timer by default. So, necessary changes are to be done in the library files to make sure PWM and IR use different timers, when using them simultaneously.

NRF:

The NRF24L01+ modules communicate using frequency channels called Data Pipes, which are distinguished from each other by means of logical addresses. As there are six devices which use NRF in this factory layout, the NRF modules have each been given a specific address. The Arduino libraries for NRF, available for free on GitHub [5], have been used to establish the connection and achieve communication among the NRF modules.

The following functions from the RF24 Arduino library are used for developing the control logic for the NRF communication.

Name	**Description**
begin()	Starts the operation of the chip
openReadingPipe(*number*, *address*)	Opens the data pipe for reading from the corresponding pipe *number* and *address*
openWritingPipe(*address*)	Opens the data pipe for writing to the corresponding *address*
startListening()	Start listening on the pipes opened for reading
stopListening()	Stops listening for data on the pipes
available()	Tests whether there are bytes available to read
read(*buffer*, *length*)	Reads the payload from the available pipe into the *buffer* for the specified payload *length*

write(*buffer*, length)	Writes data through the *buffer* into the open writing pipe

Table 3.11 NRF Arduino functions

The *begin()* function is called once in the setup of the Arduino. The *read()* function is always preceded by the *startListening()* function and similarly, the *write()* function is always called after the *stopListening()* function. Upto 5 data pipes can be opened for simultaneous reading, which are numbered 1-5 in the *openReadingPipe()* function, and the writing pipe is always numbered 0 by default. The writing of payload always corresponds to the latest open data pipe, while the reading of payload can occur through any of the open reading pipes. In case of multiple open reading pipes, it is a good practice to check the pipe number on which data is available by providing an optional address parameter using a variable in the *available()* function, which changes the value of the address variable to that of the pipe.

A few optional functions have been used for reliability, which are *setPALevel()* which is used to set the power amplifier level to one of four values and *setRetries()* which is used to set the delay and count for the number of retries upon failed transmission.

3.4 Developing and simulation Virtual model

3.4.1 Visual Components

Design aspects of Virtual Model

The Conceptual Model was designed in Visual Components [16] in three separate sub models. Each sub model features three separate functions of the system

Production Area: This simulation model features the functionality of product creation and transport using conveyor models and product feeders.

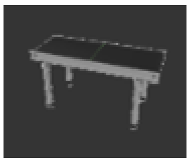
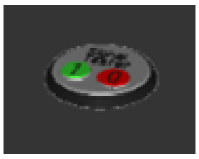
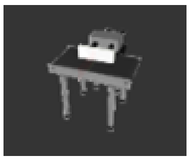
Object Name	Library	Functionality
Sensor Conveyor 	Conveyors	Transports the product from one end to the other. Detects products at the sensor location
Turntable Conveyor	Conveyors	Distributes products with a rotating turntable to various user-defined outputs
Advanced Feeder	Feeder	Feeder to produce batches and distributions of products with different product IDs and properties.
Signal Tester 	Misc	Use as a tester to fire signals or as a listener to print incoming signal values.
Pusher Conveyor 	Conveyors	Pushes the predefined products based on the product ID to other lines

Table 3.12 Factory Components used in Simulation

Operation Logic:

- The Boolean “True” signal from the Signal Tester activates the Advanced Feeder
- The Advanced Feeder manufactures product types on receiving signal “True” and passes the product on to the Sensor Conveyor
- The Sensor Conveyor detects the product and sends the signal to the Turntable module
- The turntable actuates according to the signal detection from the sensor conveyor to its respectable ports to transport the products.
- The products from three different manufacturing ports are transported to the fourth port which carries the product to next phase of the layout
- The Pusher Conveyor is used to remove the defective product which is identified according to product ID in the simulation is pushed out of the conveyor line. Rest of the products are transported to the next phase or the High bay shelf.
- Signals feature of the Visual components is used to link signal tester with advanced feeder and the Sensor Conveyors to the Turntable Conveyor

Note: Signals property from the Visual Components software is used to allow components with wireless interfaces or ports to be remotely connected within the conceptual layout.

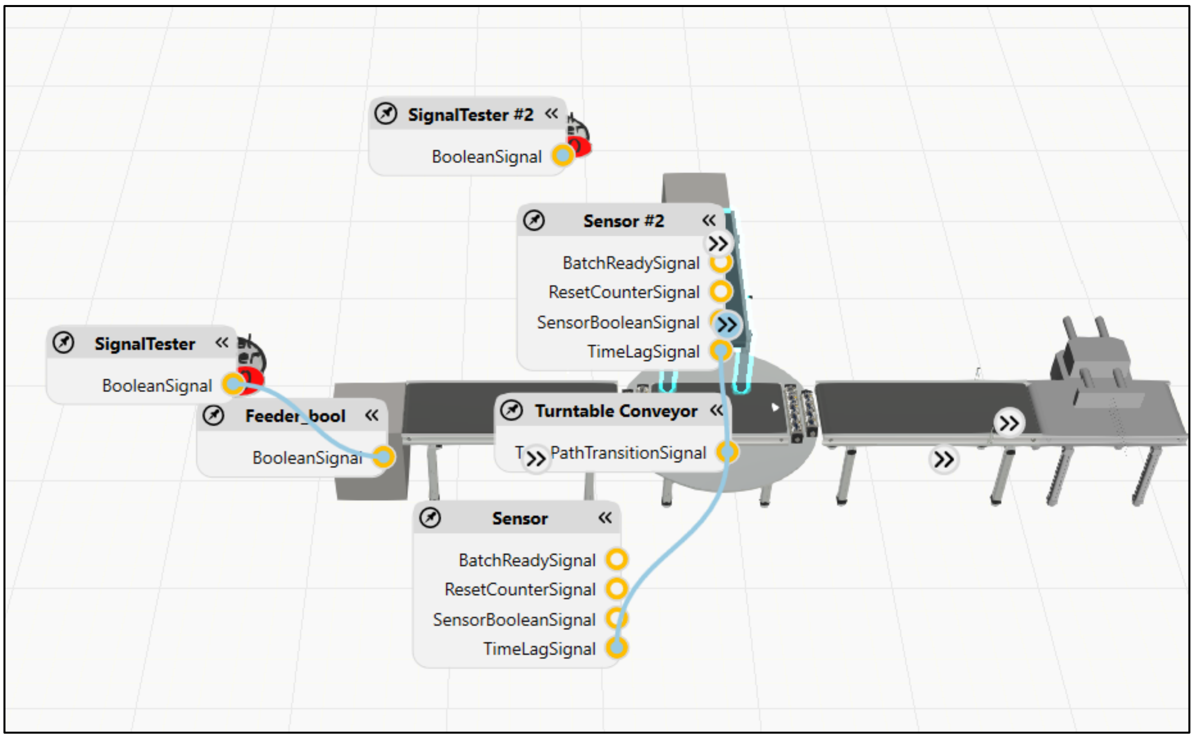


Figure 3.23 Connected Signals links in Production Area

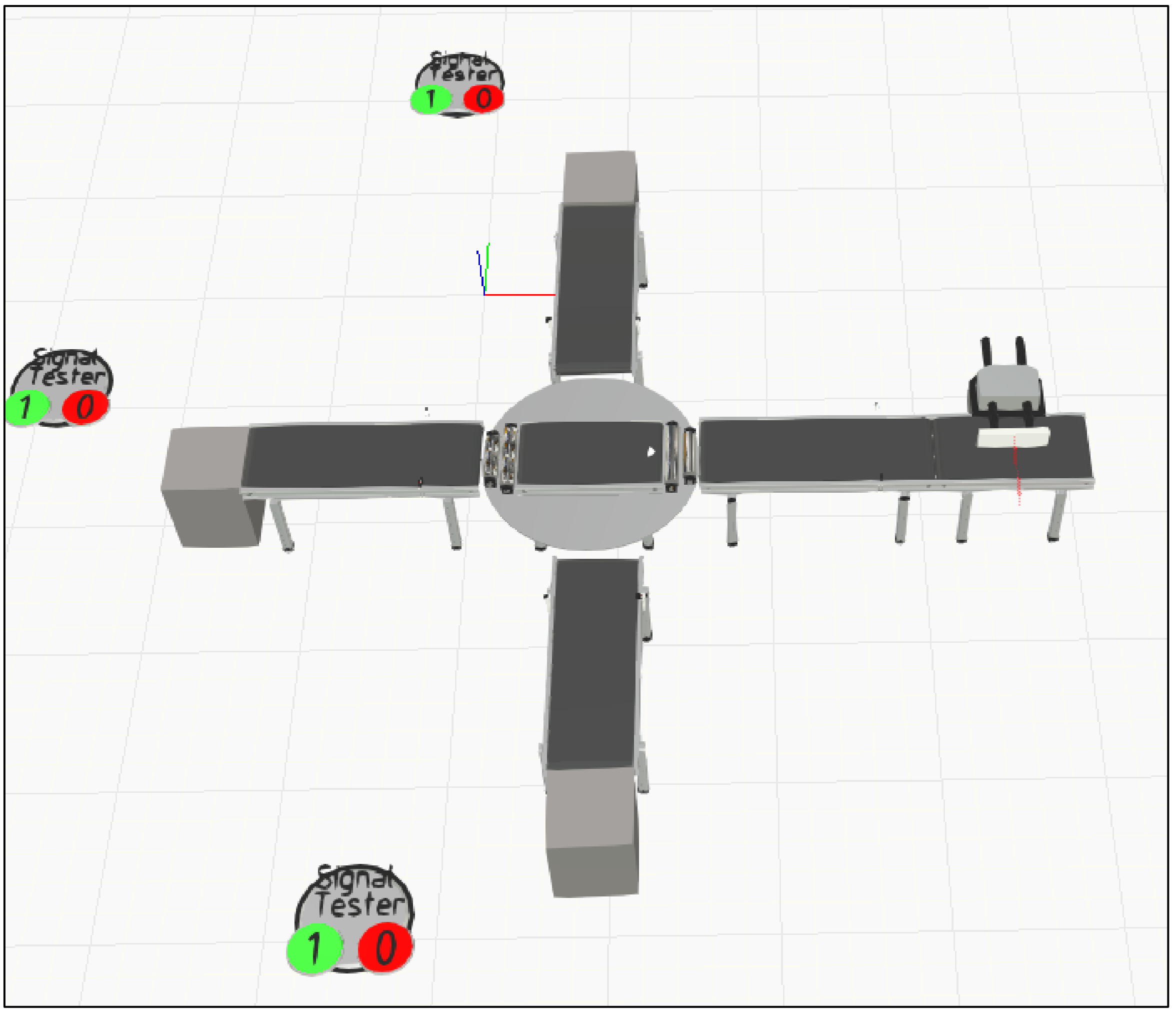


Figure 3.24 Production Area Conceptual Layout

Storage Area

This conceptual layout model features the functionality of product storage and dispatch using components such as input, output control feeders, crane and racks for storage, retrieval, transport functionality.

Object Name	**Library**	**Functionality**
ASRS-Infeeder	ASRS-Library	Input terminal connecting material flow from conveyors to the storage system.

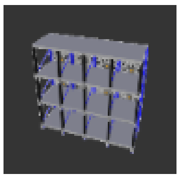
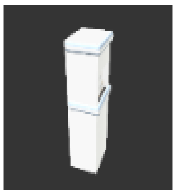
ASRS-Outfeeder	ASRS-Library	Output terminal connecting material flow from the ASRS system to output conveyor.
ASRS-Crane	ASRS-Library	ASRS-Crane handles products between in-feeders, out-feeders and high racks.
ASRS-ProcessRack 	ASRS-Library	Rack with the inbuilt process time. When a part is placed on the rack a process is started and when the process is done, the product is automatically called to be retrieved.
ASRS-Controller 	ASRS-Library	ASRS-Controller is the main logic control component for an ASRS-setup.

Table 3.13 Libraries and Factory Components Used in Simulation

Operation Logic

- The products from the Turntable Conveyor are transported to the Storage area also known as High bay shelf. The infeeder component recognises the incoming product which is transported to the ASRS Crane for storage and retrieval functions.
- Two ASRS Process Racks are attached to the ASRS crane on either side as shown in the figure below. The crane facilitates storage of the products to its consecutive rack spaces.
- The incoming products are identified using their unique Product IDs assigned using the Advanced Feeders from the production area.

- The immediate products are not stored to the process rack, rather they are directly transported to the next section.
- The products that are stored to the rack are retrieved using the ASRS-Crane
- The retrieved products from the crane are recognised at the ASRS Outfeeder connected to the crane which then transport it to the conveyor and then to the dispatch area
- The entire control sequence of the High bay shelf is assisted using ASRS Controller logic and interface functionality of the Visual Components software.
- Connect the Infeeder and Outfeeders to the Main Infeeders and Main Outfeeders interface of the ASRS-Controller.
- Connect the crane with the ASRS-Controller and place it in the scene close to the racks and in/outfeeders. Connecting from the controller side to crane is recommended and not crane to the controller.
- ASRS Controller controls and communicates with the ASRS-Infeeders, Outfeeders, Racks and Cranes. Connect those components with the ASRS Controller by using the Interfaces in this controller component.

Note: Interfaces features of the Visual Components software allow to access the special operating features of different components within the layout.

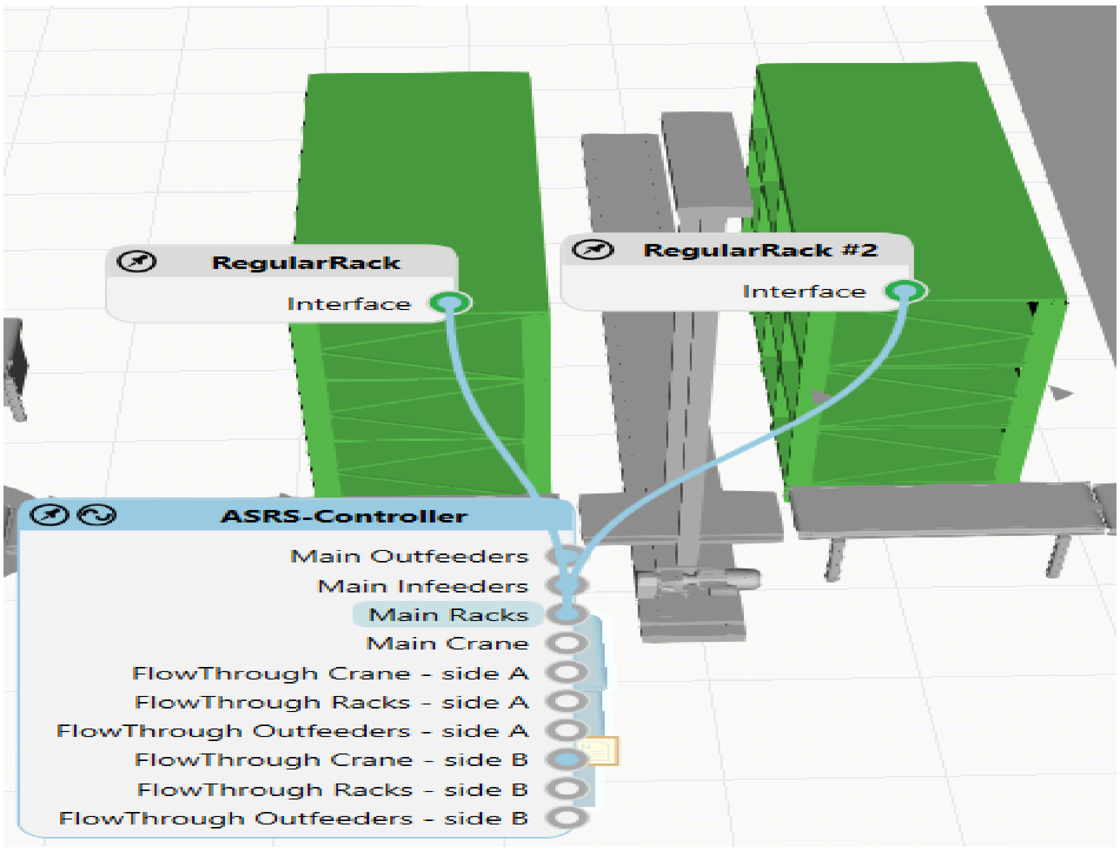

RegularRack
Interface
RegularRack #2
Interface
ASRS-Controller
Main Outfeeders
Main Infeeders
Main Racks
Main Crane
FlowThrough Crane - side A
FlowThrough Racks - side A
FlowThrough Outfeeders - side A
FlowThrough Crane - side B
FlowThrough Racks - side B
FlowThrough Outfeeders - side B

Figure 3.25 Interface connections connecting the high bay shelf components

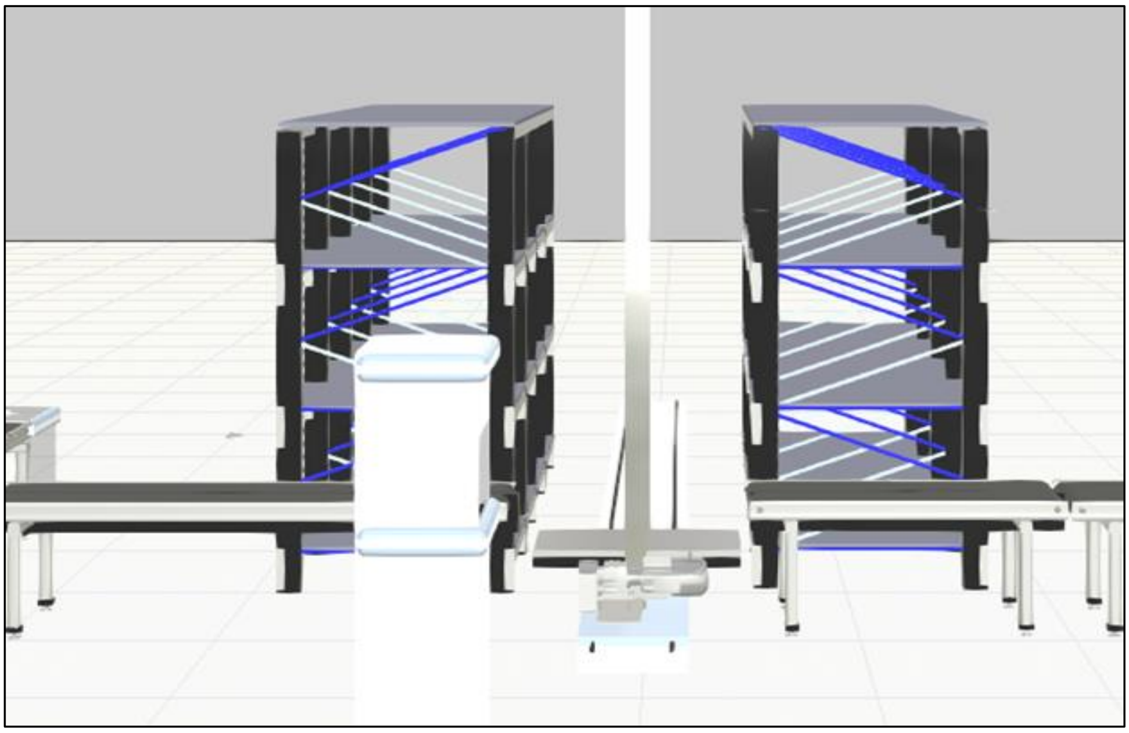
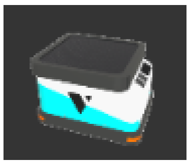
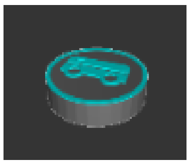
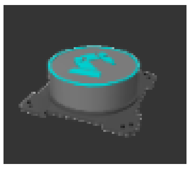
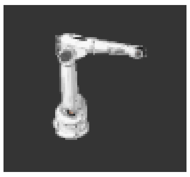
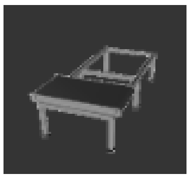
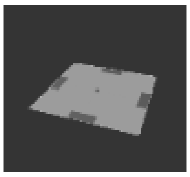


Figure 3.26 Interface connections connecting the high bay shelf components

Dispatch Area

This third section of the layout model features the functionality of product dispatch using components such as AGV, Robotic Arm, Conveyors to different dispatch routes.

Component Name	Library	Functionality
Mobile Robot Resource 	Process Resources	Mobile robot resources can utilize pathway components to avoid obstacles and to transport the dispatched products from the conveyors

Mobile Robot Transport Controller 	Process Transport Controllers	All functional resources need to have a dedicated transport controller. After assigning resources to a transport controller, the controller takes control of their actions.
Robot Transport Controller 	ASRS-Library	Robot controller for Process Model work flow. Use this as a transport controller in layouts with other Process Modelling components.
Generic Articulated Robot 	Robots	Generic and parametric Visual Components Articulated Robot. Used to place the dispatched the products on the AGV.
Shuttle Conveyor 	Conveyors	Distributes products with a moving shuttle to various user-defined outputs at the same height
Pathway Area 	Process Resources	Allowed pathway area for the resources. Facilitate the movement path for the AGV.

Operation Logic

- The products from the High bay shelf are transported to the dispatch area through the out-feeder controller.
- The products are then transported to different dispatch end points using the shuttle conveyors.

- The immediate dispatch products are identified at the dispatch endpoint. The articulated robot picks the products and places it on the AGV for transport.
- The path area component defines the area in which the AGV can operate.
- The AGV drops the products to the dispatch area and returns to its collection point for the next dispatched product.

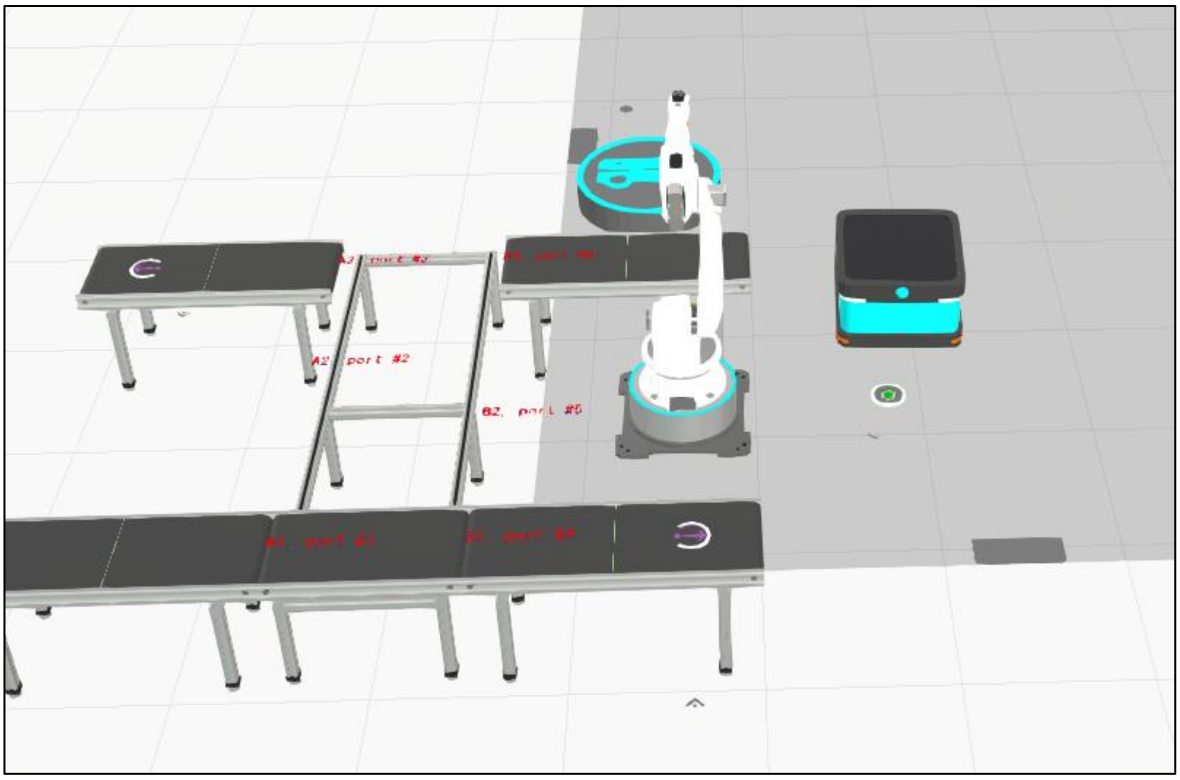


Figure 3.27 Conceptual design of Dispatch Area

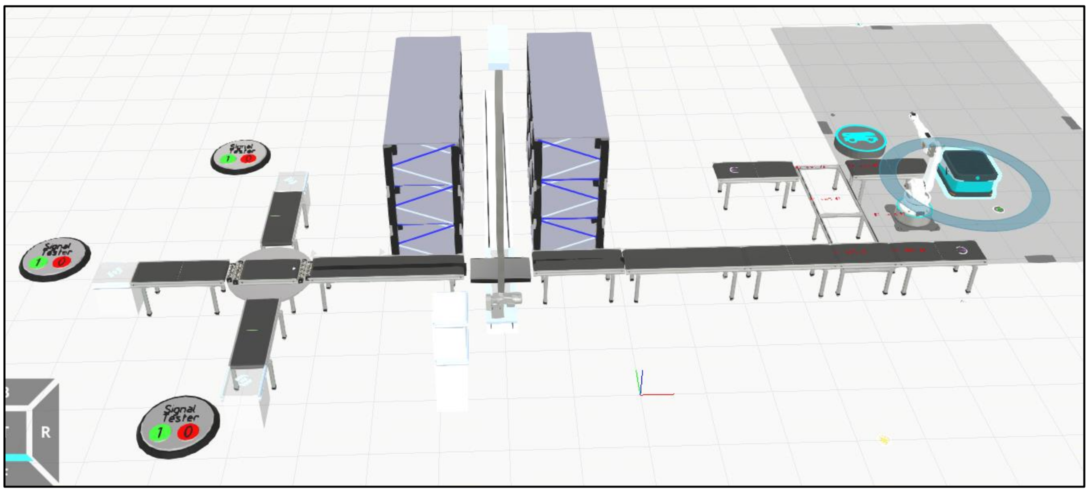


Figure 3.28 Design of Factory Floor Plan

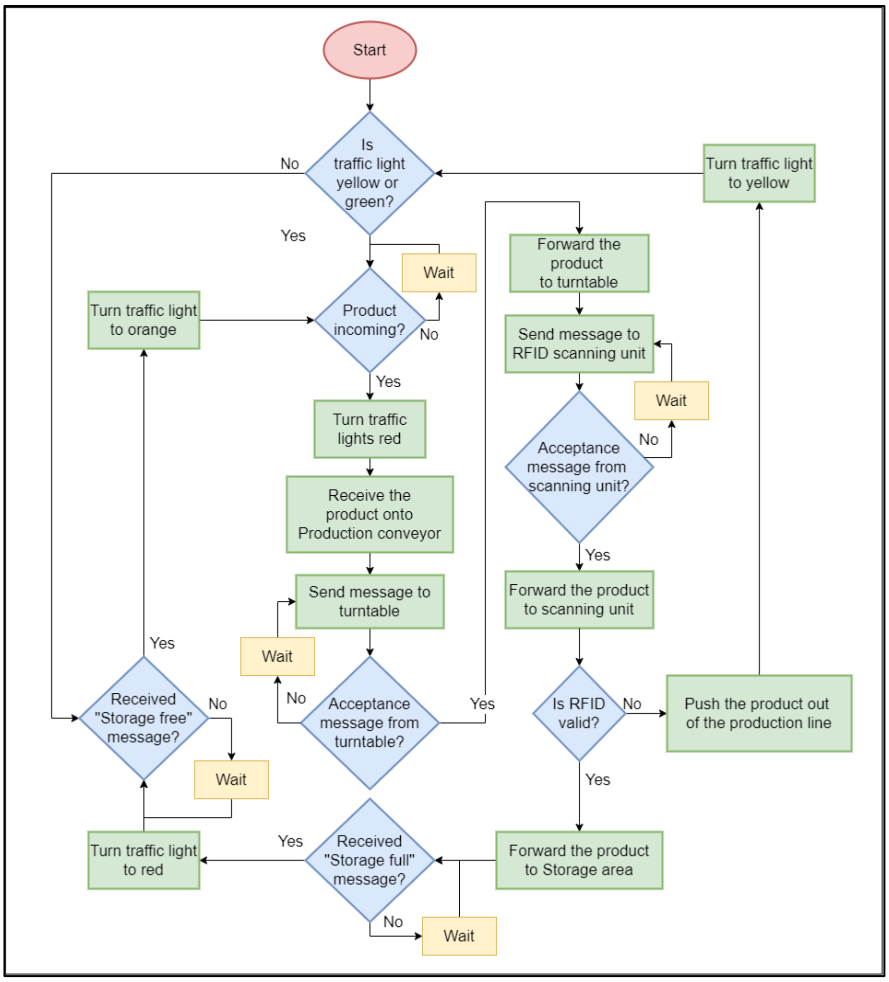
3.5 Realizing Virtual Model into a Physical Layout

3.5.1 Production area

This area is where the production of the products takes place. There are three conveyors as shown in the Figure 3.29 representing production to three types of products, namely A, B and C. Product types A and B are produced continuously and the corresponding conveyors accept products whenever they arrive. But product type C is an immediate dispatch product which means that it is only produced and forwarded to the corresponding conveyor only upon customer’s request.

Traffic light system for each production conveyor is implemented indicating the status of the conveyor. Three colours of lights are used, namely red, orange and green. Red indicates that the conveyor cannot accept any products currently which can be used to indicate that the production area is already busy in transporting another type product, orange indicates that the conveyor is idle and can accept a product from the production system anytime and lastly, green indicates that the conveyor is requesting for a product from the production system which might be because a customer has ordered for that particular type of product.

Also, an RFID scanning unit is included in the production system. It is used to identify the products that have invalid or no RFID tag to them and then separate them from the production line using a pusher.



Start
Is traffic light yellow or green?
No
Yes
Turn traffic light to yellow
Wait
Turn traffic light to orange
Product incoming?
No
Yes
Turn traffic lights red
Receive the product onto Production conveyor
Send message to turntable
Wait
No
Acceptance message from turntable?
Yes
Forward the product to turntable
Send message to RFID scanning unit
Wait
No
Acceptance message from scanning unit?
Yes
Forward the product to scanning unit
Is RFID valid?
No
Push the product out of the production line
Yes
Forward the product to Storage area
Received "Storage full" message?
No
Wait
Yes
Turn traffic light to red
Received "Storage free" message?
No
Wait
Yes

Figure 3.29 Flowchart of Production Area

3.5.2 Storage Area

High bay shelf with 3 types of products:

The storage area acts as a bridge between the customer and the factory layout, and as a bridge between the production and dispatch areas. The storage area is connected to the Android application using Bluetooth, and to the other areas of the factory layout by means of a wired connection, implementing UART.

There are three types of products realized in this project. They have been named as type A, type B and type C. Products of type A and type B can be stored in the high bay shelf and hence can be produced without the customer's request. Type A products are stored on the right side of the high bay shelf, and type B products are stored on the left side. Type C products are only produced on the customer's request and are sent through the storage to the dispatch area without being stored in the high bay shelf.

There are three types of processes involving the storage area.

1. Type A or B without customer request

2. Type A or B with customer request
3. Type C direct dispatch

When a product of type A or B reaches the production area without the customer's request, it transports it to the storage area with an appropriate message indicating that the product is to be stored. The storage and retrieval belt receives the product and transports it to the high bay shelf, where it is stored at one of the rows on the right side for type A or the left side for type B product. An appropriate message is sent to the production area when the storage area is full, so that no more products are sent for storage.

When a customer requests for a type A or type B product, a corresponding product is retrieved from the high bay shelf and transported to the dispatch area. If a product of the requested type is unavailable in the storage, a message is conveyed to the production area, requesting the production of the product. It is then transported through the storage area into the dispatch area without being stored.

When the customer requests a product of type C, a message is sent to the production area, requesting the production of the product. When the product reaches the storage area, it is directly transported to the dispatch area.

3.5.3 Dispatch Area

The Dispatch Area receives products from the storage and transports them to the respective sinks. It consists of a conveyor to initially receive the product from storage, and a slider which transports it to one of three conveyors depending on the type of product. Products of type A and B are picked up by a robotic arm and placed on an Automated Guided Vehicle (AGV), which takes the product to one of two locations depending on the type of product. A product of type C is transported directly to a conveyor which deposits it at the sink.

NRF:

When the initial conveyor receives the product, it also receives information about the type of product. This is conveyed to the slider along with the product. There are three different conveyors adjacent to the slider, one for each type of product, and the slider moves to the corresponding location depending on the information received, regarding the type of product.

Although all components in the dispatch area are wirelessly connected using NRF24L01+ modules, each component is programmed to communicate only with an adjacent module, to avoid interference among the different modules.

Each module is always in one of two states, a read state or a process state. In the read state, the module waits for an incoming message. In the process state, it transports the product and transmits messages to other modules. When a module receives an acknowledgment from an adjacent module that the product has been received, it reverts to its original waiting state.

Robotic arm

A robotic arm is located in between the two conveyors which dispatch the products of type A and type B. These conveyors are parallel to each other. One of their extremes is in line with the rest position of the robotic arm and are 180° apart in terms of the robotic arm angle. When a product reaches either of these two end points of the conveyor, it stops, and a message is transmitted to the robotic arm regarding the type of product. The robotic arm then moves to the corresponding location. It picks up the product and places it on the AGV, and a message is transmitted to the AGV regarding the type of the product.

AGV with IR

The AGV is parked near the Robotic arm in its idle state. When the Robotic arm picks up the Product from one of the conveyors at the end of the Dispatch Section (A or B) and places it on the AGV, a message about the type of the product (A or B) is sent to the AGV through IR communication. The AGV carries the product to the corresponding dispatch sink based on the received message and returns to the parking area and waits for the next product to be dispatched.

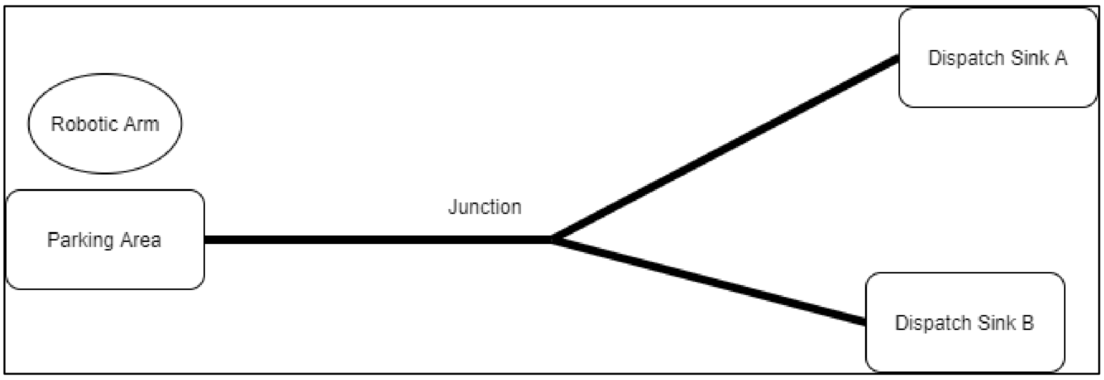


Figure 3.30 AGV Path in Dispatch Area

```
void loop() {

  if (destinationSink == '0') {
    if (irrecv.decode(&results)) {      // Decode received message
      if (results.value == 0x45555550) {
        destinationSink = 'a';              // Product type A
      }
      else if (results.value == 0xF124B0B6) {
        destinationSink = 'b';            // Product type B
      }
      irrecv.resume();                  // Receive the next message
    }
  }
```

Once the message is received by the AGV, it starts moving towards the junction where it takes the path according to the product type. If the product type is ‘A’, AGV takes the path towards the Dispatch Sink A, or if the product type is ‘B’ it takes the path that leads to the Dispatch Sink B.

3.5.4 Android application

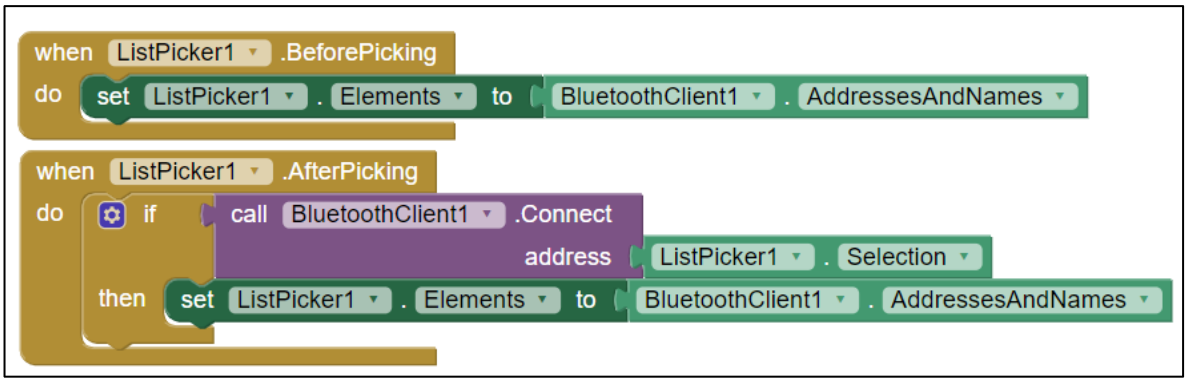
An android application is developed to connect with the physical layout and order the desired products. The user interface is developed using MIT App Inventor web application [17]. The actions in the Android application are developed in the block page of the web application. The MIT App inventor is a user-friendly tool which provides the option to drag and drop the components for the user interface. The following components were used for developing the user interface.

Component	**Description**
Horizontal Arrangement	Used to add and display components from left to right
Table Arrangement	Used to add and display components in a tabular form
Button	Used to identify the click to perform an action
Label	Used to display text
List Picker	Used to display list which enables user to select one
Bluetooth Client	Used to connect to Bluetooth devices using Serial Port Profile
Firebase DB1	Used to communicate with firebase cloud to store and retrieve data

Table 3.14 Android Application Components

The android application communicates with the physical components via Bluetooth channel and the records of the orders are stored in firebase cloud, which can be used for further development of a digital twin.

The Android application uses the Bluetooth of a smartphone to connect with the module on the physical layout. The application displays Bluetooth devices available as a list and the user needs to select the right device and pair them. The logic for connection is shown in Figure 3.31

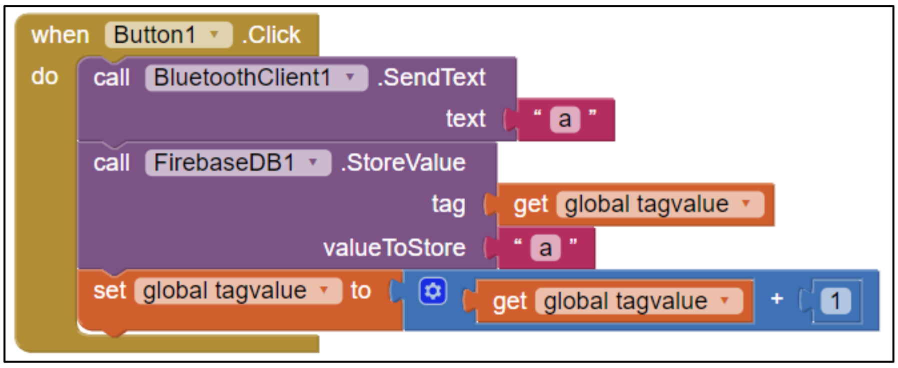


```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to BluetoothClient1 . AddressesAndNames

when ListPicker1 .AfterPicking
do  if    call BluetoothClient1 .Connect
                            address  ListPicker1 . Selection
    then  set ListPicker1 . Elements to BluetoothClient1 . AddressesAndNames
```

Figure 3.31 Logic for Bluetooth device connection

The user can order the products using buttons after successful connection of Bluetooth. Upon button click an appropriate message is transmitted from the application. The message is also stored in the firebase cloud in a secured database. The logic for button clicks is shown in Figure 3.32



```
when Button1 .Click
do  call BluetoothClient1 .SendText
                         text  " a "
    call FirebaseDB1 .StoreValue
                          tag  get global tagvalue
                 valueToStore  " a "
    set global tagvalue to  get global tagvalue + 1
```

Figure 3.32 Logic for button click

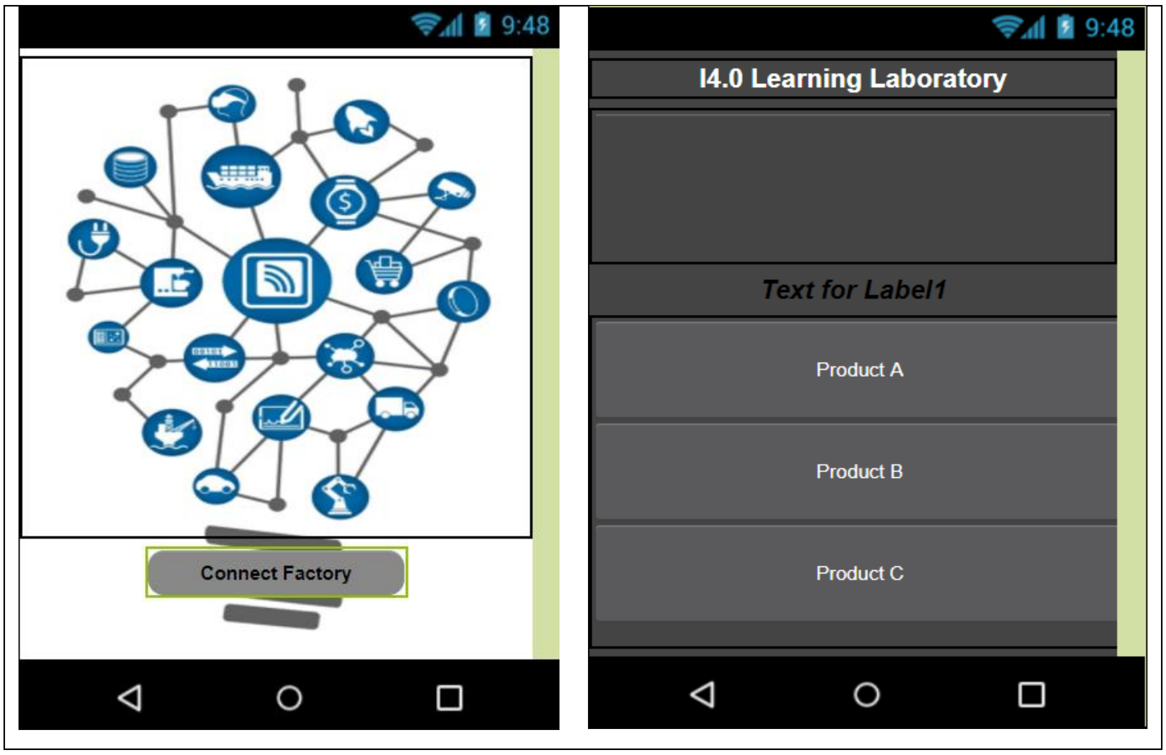


Figure 3.33 Android Application User Interface

3.5.5 Integration

A UART circuit implemented using a three-wire interface between the individual areas of the factory layout is responsible for the communication between them. The Tx pin of the last controller in the production area is connected to the Rx pin of the storage controller, and vice versa. The Ground pins of both modules are connected to each other for electrical neutrality. Similarly, the Tx and Rx pins of the first conveyor in the dispatch area are also connected to the Rx and Tx pins of the storage controller, albeit to a different Serial port, taking advantage of the fact that the Arduino Mega controller has three extra Serial ports besides the one connected to the computer via USB. The Bluetooth module is connected to the third Serial port in the storage controller, as it also uses a UART circuit for communication.

4. Conclusion

4.1 Summary

The aim of the project is to make the components in Industry 4.0 Learning laboratory modular, standalone and implement LiFi, NRF, Bluetooth, IR and UART communication protocols such that they can be customised for different material flow operations. This is achieved in different steps, initially by making a conceptual model, then assembling and testing individual factory modules, which includes designing circuit boards and developing control logics for each individual component. The next stage is testing the communication protocols with transceivers interfaced to Arduino Mega 2560. In the next step, a virtual model is developed based on the conceptual model in Visual Components software. The idea is to understand and test the control flow through simulation before implementing on the physical layout. In the final stage, the physical layout is implemented based on the virtual model. The final physical layout has three individual areas namely production area, storage area and dispatch area. The production area is implemented with LiFi communication, the dispatch area is implemented with NRF and IR communications. The three individual areas are connected by UART wired communication. An android application is developed to order the products and it communicates to physical layout using Bluetooth communication.

4.2 Future Work

In the current scope, a physical layout is developed from the virtual model. In future a digital twin can be developed for the physical layout using an efficient data exchange medium so that different factory layouts can be tested and evaluated before deploying on the physical world and real-time tracking of products can be achieved.

In the current implementation, a continuous power supply is given to all the factory components. This can be further improved as the power supply can also be connected using hub switches. Briefly, the power to dispatch section is only required after successful completion of production or when the storage area receives an order from customer, so a hub switch with relays can be used to trigger power at dispatch section.

3D printing product lines can be developed by extending the current implementation. The Android application should be replaced with a slicing software so that customers can design a product on their own and can order them. The product after being printed can be used with the current layout for dispatching.

References

[1] https://documents.trendmicro.com/assets/white_papers/wp-threats-to-manufacturing-environments-in-the-era-of-industry-4.pdf.

[2] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt and M. Wählisch, "RIOT: An open source operating system for low-end embedded devices in the IoT.," *IEEE Internet of Things Journal,* vol. 5, no. 6, p. 4428–4440, 2018.

[3] R. Krishnamurthi and A. Kumar, Modeling and Simulation for Industry 4.0, 2019.

[4] K. Katsaliaki and N. Mustafee, Distributed Simulation of Supply Chains in the Industry 4.0 Era: A State of the Art Field Overview, 2019.

[5] Arduino Developers, https://github.com/nRF24.

[6] https://en.wikipedia.org/wiki/Li-Fi.

[7] https://learn.sparkfun.com/tutorials/ir-communication/ir-communication-basics.

[8] https://www.staudinger-est.de/produkte/foerdertechnik/.

[9] https://www.sparkfun.com/products/15293.

[10] https://en.wikipedia.org/wiki/PID_controller#:~:text=A%20proportional%E2%80%93integral%E2%80%93derivative%20controller,applications%20requiring%20continuously%20modulated%20control..

[11] https://wiki.dfrobot.com/4WD_MiniQ_Complete_Kit__SKU_ROB0050_.

[12] Arduino Developers, Available: https://store.arduino.cc/tinkerkit-braccio-robot.

[13] M. Margolis. https://github.com/arduino-libraries/Servo.

[14] Arduino Developers, https://www.arduino.cc/reference/en/.

[15] https://github.com/z3t0/Arduino-IRremote.

[16] https://academy.visualcomponents.com/.

[17] D. Barreiro. https://appinventor.mit.edu/explore/library.

[18] A. Martino. Available: https://github.com/arduino-libraries/Braccio.

[19] O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes., "Operating systems for low-end devices in the internet of things: a survey.," *IEEE Internet of Things Journal,* vol. 3, no. 5, p. 720–734, 2015.