TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

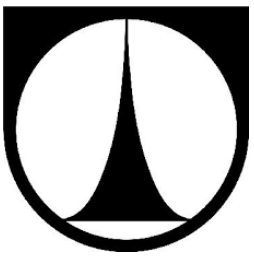
Control System of Smart Factory Model

Master Thesis

Liberec 2020 Backia Siva Vallinayagam

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering



Control System of Smart Factory Model

Master Thesis

Liberec 2020 Backia Siva Vallinayagam

Liberec 2020

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Control System of Smart Factory Model

Master Thesis

Study programme: N2301 Mechanical Engineering

Study branch: Manufacturing Systems and Processes

Author: **Backia Siva Vallinayagam**

Thesis Supervisors: Ing. Radek Votrubec, Ph.D.
Department of Manufacturing Systems and Automation

Liberec 2020

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Master Thesis Assignment Form

Control System of Smart Factory Model

Name and surname:	**Backia Siva Vallinayagam**
Identification number:	S18000460
Study programme:	N2301 Mechanical Engineering
Study branch:	Manufacturing Systems and Processes
Assigning department:	Department of Manufacturing Systems and Automation
Academic year:	**2019/2020**

Rules for Elaboration:

The aim of the diploma thesis is to improve control system of smart factory model.

1. Meet with all components of the smart factory (stacks, vehicles, server and control application), Arduino controllers and Wi-Fi communication.
2. Create working diagrams of each Smart Thing component. Describe the communication relationships between the components.
3. Design new algorithms for more vehicles and more stacks. Optimize the paths of the vehicles.
4. Test new algorithms on real model of factory.

Scope of Graphic Work: as required
Scope of Report: 50
Thesis Form: printed/electronic
Thesis Language: English

List of Specialised Literature:

[1] BEQUETTE, B. Process control: modeling, design, and simulation. Upper Saddle River, N.J.: Prentice Hall PTR, 2003. ISBN 0133536408.

[2] Arduino Learning: Getting Started with Arduino. In: Arduino [online]. 2014 [cit. 2015-01-09]. available from: http://arduino.cc/en/Guide/HomePage

Thesis Supervisors: Ing. Radek Votrubec, Ph.D.
Department of Manufacturing Systems and Automation

Date of Thesis Assignment: November 20, 2019
Date of Thesis Submission: May 20, 2021

L.S.

prof. Dr. Ing. Petr Lenfeld
Dean

Ing. Petr Zelený, Ph.D.
Head of Department

Liberec November 20, 2019

Declaration

I hereby certify, I, myself, have written my master thesis as an original and primary work using the literature listed below and consulting it with my thesis supervisor and my thesis counsellor.

I acknowledge that my bachelor master thesis is fully governed by Act No. 121/2000 Coll., the Copyright Act, in particular Article 60 – School Work.

I acknowledge that the Technical University of Liberec does not infringe my copyrights by using my master thesis for internal purposes of the Technical University of Liberec.

I am aware of my obligation to inform the Technical University of Liberec on having used or granted license to use the results of my master thesis; in such a case the Technical University of Liberec may require reimbursement of the costs incurred for creating the result up to their actual amount.

At the same time, I honestly declare that the text of the printed version of my master thesis is identical with the text of the electronic version up- loaded into the IS/STAG.

I acknowledge that the Technical University of Liberec will make my master thesis public in accordance with paragraph 47b of Act No. 111/1998 Coll., on Higher Education Institutions and on Amendment to Other Acts (the Higher Education Act), as amended.

I am aware of the consequences which may under the Higher Education Act result from a breach of this declaration.

May 11, 2020 Backia Siva Vallinayagam

ACKNOWLEDGEMENT

With immense pleasure and utmost gratitude, I acknowledge the support of my university for providing me this great opportunity to develop my engineering skills for accomplishing this diploma thesis and help me to further develop the knowledge on Industry 4.0 concept with the latest trends used in it.

I am indebted firstly to thank **Ing. Radek Votrubec, Ph.D.,** whose expertise and wide knowledge in the field of study and his encouragement, professional guidance and his time all along the thesis to work better under his supervision.

I would like to thank our Head of the Department **Ing. Petr Zelený Ph.D.,** for giving me this opportunity to perform this thesis and for his great support in every way to pursue our academics.

I gratefully thank my entire department staffs at the Technical university of Liberec for their immense support and insights on the various technologies required to complete my thesis.

Finally, I am grateful to thank my parents and friends, for their unending help and support in all the situations and encouraging my works.

This work was partly supported by the Student Grant Competition of the Technical University of Liberec under the project Optimization of manufacturing systems, 3D technologies and automation No. SGS-2019-5011.

ABSTRACT

The construction of a smart factory involves many complications in installing it. This thesis aims at establishing an enhanced and simplified communication and control system for a smart factory model. A smart factory model is made with components like, vehicle, stacks with beads, server and control application, which, communicates using Wi-Fi communication. A vehicle is navigated in a controlled path of the system, to collect beads from the stacks automatically on getting order from user. The communication relationship between the components was made by Mr. Jansa [10], but had many cons in it. The control system with Arduino worked slower, as those codes were complicated in testing too many conditions in main loop. The communication is possibly made efficient, by drawing working diagrams of automaton for each smart component to split up the conditions checked at stages. New strategies for the communication model are made and tested on the smart factory. Hence, after implicating the work diagrams into the program and using new strategies, simple and comparatively good control system in the smart factory model is exercised.

Keywords:

IOT, Industry 4.0, Smart Factory, Control System.

ABSTRAKT

Konstrukce inteligentní továrny obnáší při instalaci mnoho komplikací. Tato práce si klade za cíl vytvořit vylepšený a zjednodušený komunikační a řídicí systém pro inteligentní model chytré továrny. Model chytré továrny je se skládá z těchto komponent: vozík, zásobníky korálků, server a řídicí aplikace. Jednotlivé komponenty spolu komunikují pomocí Wi-Fi. Vozík se pohybuje pomocí sledování černé čáry mezi zásobníky korálků tak, aby automaticky získal ze zásobníků korálky správných barev podle objednávky od uživatele. Komunikační protokol mezi jednotlivými komponenty továrny, který byl vytvořen panem Jansou [10], měl mnoho nevýhod. Řídicí systém s Arduinem pracoval pomaleji, protože řídicí algoritmus, spočívající v testování příliš mnoho podmínek v hlavní smyčce, byl příliš složitý. Komunikace je zefektivněna použitím struktury automatů. Pro každou komponentu byly navrženy grafy automatu, aby se podmínky testovaly ve fázích. Na inteligentní továrně jsou dale vytvářeny a testovány nové řídicí strategie. Po začlenění pracovních diagramů automatů do programu s využitím několika nových strategií se tedy v modelu chytré továrny podařilo vytvořit jednoduchý a poměrně dobrý řídicí systém.

KLÍČOVÁ SLOVA:

Internet věcí, Průmysl 4.0, chytrá továrna, řídicí system.

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

IOT – Internet of Things

AI – Artificial Intelligence

CPS – Cyber Physical Systems

AGV – Automatic Guided Vehicle

PCB – Printed Circuit Board

LCD – Liquid-Crystal Display

LED – Light Emitting Diode

RFID – Radio Frequency Identification

RISC – Reduced Instruction Set Computer

PWM – Pulse Width Modulation

IR – Infrared

1. INTRODUCTION

The human race is far developed with technologies, in its various field. These developments paves ways to us in communicating and sharing our knowledge between each other. These communications made in real life, is now examined to apply on virtual worlds of machineries, to establish communications between the components of a system to make it smart enough to do its work on its own. As like these ideologies, the world welcomes new industrial inventions to improvise and enhance all sectors of the global industries.

The idea of this thesis is to construct a smart factory prototype with communication system. A communication model made by Mr. Jansa, have many problems in it. The Arduino gets slower in running his code. This is because, he had many conditions to check at each loop and also those conditions were difficult with several logical functions in it. To overcome this, a solution is made using automaton graphs, as the conditions checked in the main loop is split into many states. This gives a better solution on communication model and the disadvantages in Mr. Jansa's control system [10], are eradicated and a good solution is obtained.

The aim is to provide an enhanced complete automation in the model of the "smart factory". The order of beads is received from user and is sent to the server of the model. The server transmits it to the vehicle. The vehicle is loaded with microprocessor, sensors and transceivers for Wi-Fi communication. The vehicle takes the order from the server and moves along the specified path to collect beads from the stacks by communicating with the gates.

The communication model between the vehicle and the stack is optimized using different ways, to ease the information exchange and to reduce the rotations or movement of the vehicle.

This smart factory model can be a solution for automatic stack handling and transportation in factories. This model can also be implemented in large scale in warehouses too, to create uninterrupted work flow. In future, factories can use this type of automated transport to ease its work on material handling.

2. EVOLUTION OF INDUSTRIAL REVOLUTION

In these modern days, industries play the major role in economy than in past times. There are many reasons justifies industrial revolution. Some of those reasons are, capitalism, agricultural revolution, imperialism in European countries, and many. But capitalism said to be a key to drive humans towards industrial revolution. It changes economies to run based on large scale mechanized manufacturing industries. In olden days, it was based on agriculture and products related to it. This revolutions on industries, took its urge and transformed industries into many advanced manners, in several periods. These are classified as versions of industrial revolution as 4 stages, as of now. At present, we are in the fourth industrial revolution, INDUSTRY 4.0.

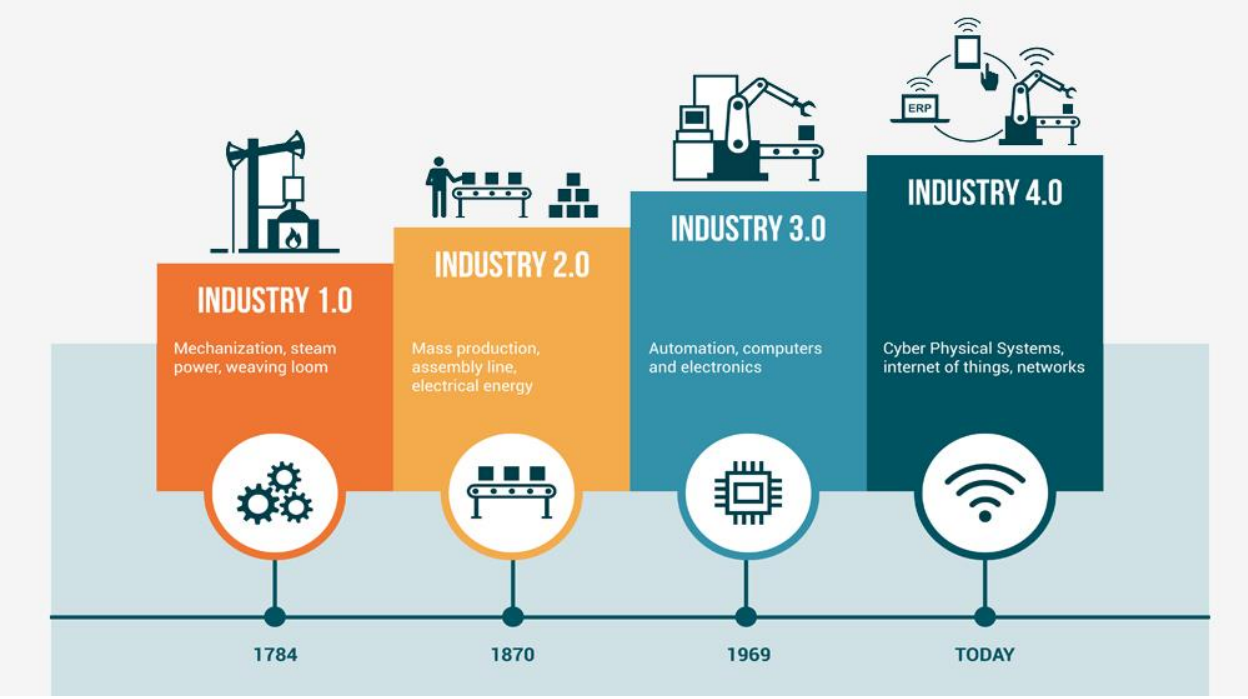


Figure 1. Stages of Industrial Revolution [3]

2.1. Industry 1.0

This was the first industrial revolution, that took place in the world. This revolution is found to be in a confined region, such as, early Britain and very few parts in Europe. This revolution takes place roughly from the mid-18th century to 1830s. This revolution makes the agriculture to industrialize and makes it urban,

by bringing mechanized spinning methods, and many. This revolution makes the pathways for agricultural industry to mechanize. Steam power production, Iron production, are increased in this revolution.

2.2. Industry 2.0

The second revolution witnessed in a period, approximately from 1870-1914. This time, the revolution leads to new inventions to the society to improve technologically. It also confirms the extension of the use and production of electricity, steel, petroleum etc., This period came up with many products which replaces the old ones. This introduces mass production in industries and makes a great change in industrial society. The production industries are designed or introduced with a assembly line in production. This makes an organized way of production, which helps to identify any problem in the line easily and also to sort it out. In later periods and industrial revolutions, this is modernized and improved.

2.3. Industry 3.0

This revolution is basically a digital revolution, because it digitalized the world with the mainframe computers, used semiconductors in computers for improvement, connected people through internet. It took place in 1950s and made the world digitize. The computers are introduced to the production line and in the management of the firms, for its faster performance and the accuracy increase at all stages. The third revolution improvises the Information Technology – IT sectors and made communications ease, throughout the world. In this period, automations in machineries also done. In this revolution, the first Programmable Logic Circuit (PLC) is manufactured.

2.4. Industry 4.0

Now, the world is facing its fourth industrial revolution, which, works on Artificial Intelligence (AI), autonomously running vehicles, Internet Of Things (IOT), Cyber Physical Systems (CPS), etc., It focusses on, advanced phases of the industry

dealing machine learning, real-time data analysis by machines itself, leading to automation in the industries. This operates on improving manufacturing efficiency by improvising the technologies and tools used so far. It modifies the communication relationship between human and machine. This leads to the transfer and communication data between machineries, without human intervention and decides accordingly to function on its own. It also focusses on modifying relationship between supply unit, production unit and the customers. Supply chain management techniques in industries, are modified and many tools are invented to practically make this possible, by this revolution.

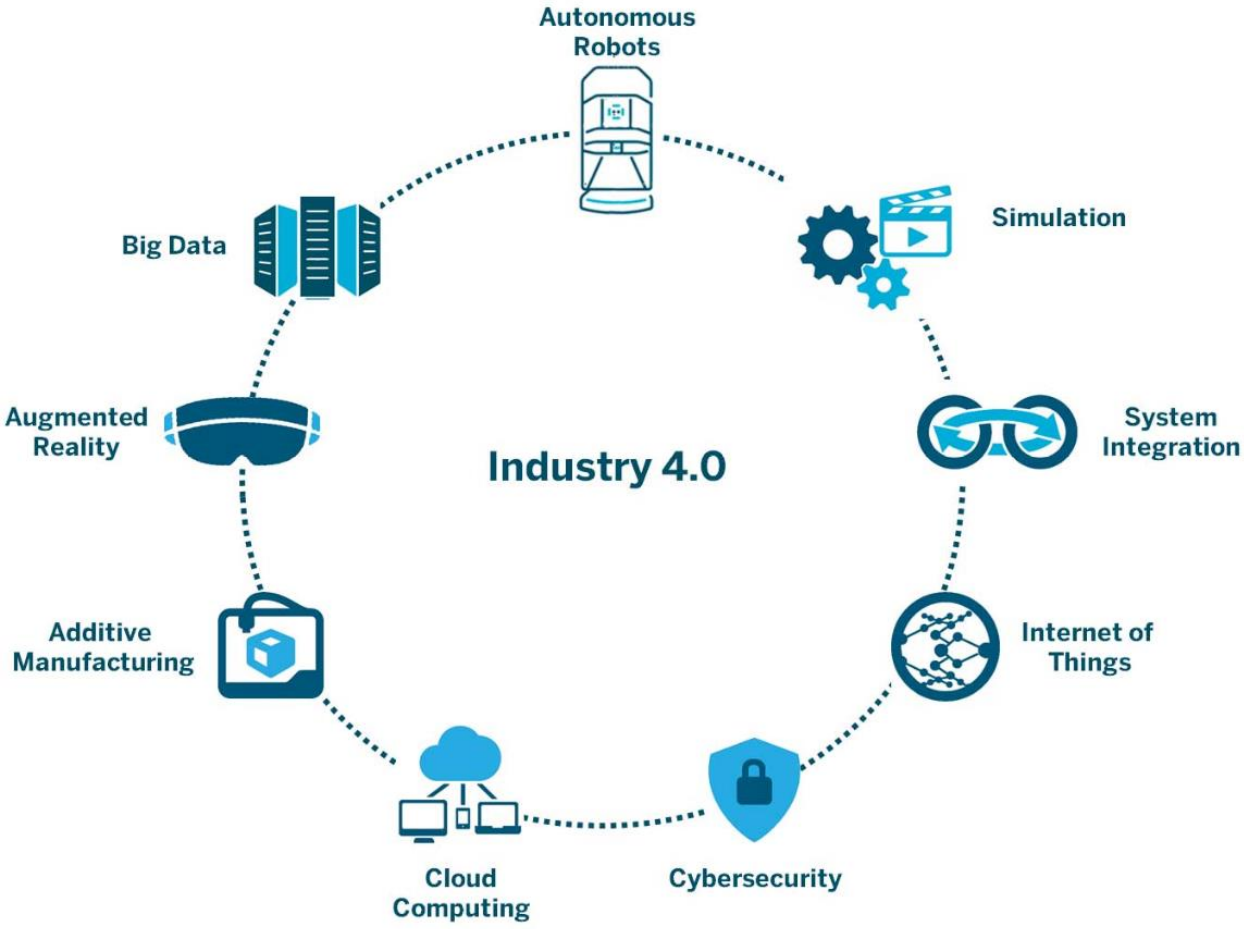


Figure 2. Major Components of Industry 4.0 [8]

There are many technologies, invented, transformed and also modified in this revolution. Among those, nine technologies are in major. They are namely, Internet of Things (IOT), Additive manufacturing, Augmented reality, Autonomous robots, System integration, Simulation, Big data analysis, Cybersecurity, Cloud storage. This adopts many new technologies, to ensure the

smart manufacturing in production and ensure wireless Wi-Fi communication in industries and the warehouses to handle products.

2.4.1. Internet of Things

This idea establishes communications between a physical body, such as sensors, machineries, etc., and the internet. The basic flow of this system is collecting data, collaborate and transfer data, then, analyze the data and take action accordingly.

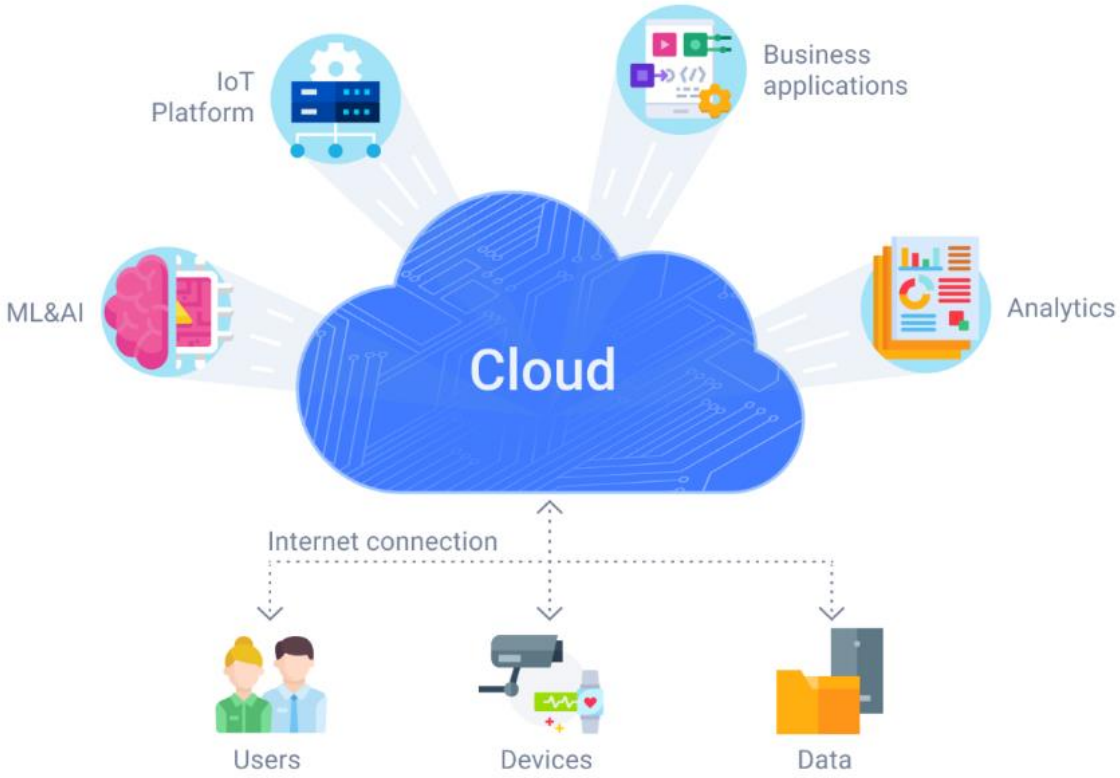


Figure 3. Cloud Based IOT [5]

This system applies on machineries with computing devices or digital machines. They are identified on their unique identity for each component and the data is transferred between themselves without human intervention in the system.

As the Figure 3 depicts, the cloud network (Internet Connection) is used for the communication between all the connected devices and the users. They also handle the data transfers and the analytical calculations and holds the IOT platforms.

2.4.2. Industrial Internet of Things

This is a part of internet of things, which intends to connect machineries, data regarding the machineries and the people. It focusses in making devices, that are intelligent enough to collect, handle the data by themselves. A Cloud network, which handles all data, processes all the collected data and shares the data as required by the service, the network needs. This ensures accessibility of large amount of data in faster speeds and leads to greater efficiencies.

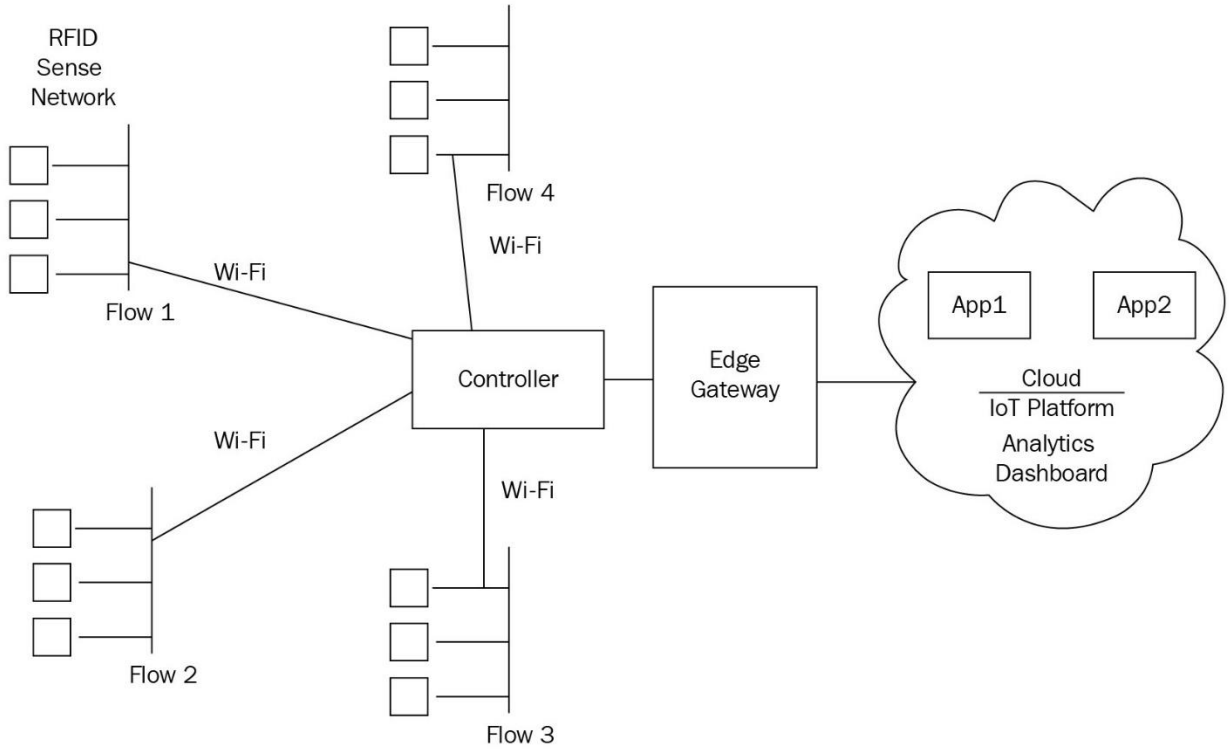


Figure 4. Simple Structure of Industrial IOT [7]

A simple structure of industrial IOT is shown in Figure 4. The system has a host / controller for connecting all the production lines. Each system is connected with the controller using Wi-Fi communication. Every system is a flow of some specific work.

Each work flow is monitored by the controller. This virtual world is connected with the IOT cloud platform as any software application. Any of the gateways are used to bridge between the controller and the application. Using the application, the real world gives information to the virtual world, which is necessary for all work flows.

2.4.3. Cyber-Physical Systems

While talking about the fourth industrial revolution, it is mandatory to talk about the CPS. This system works seamlessly to achieve a greater integration between the physical world with the cyber connections. Computing is made to increase efficiencies of the modern physical entities. This helps in making the infrastructure of a inter physical and cyber controlled activity. It satisfies the system with a futuristic need of safety, security and sustainability.

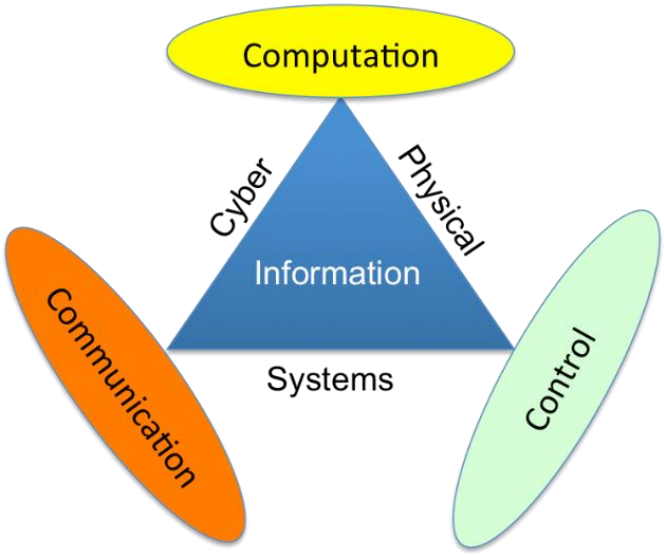


Figure 5. Structure of Cyber-Physical Systems [9]

The structure of CPS is managing the information of the computation, control and communication, as mentioned in Figure 5. Generally, these systems, which are also called as Network Controlled Systems (NCS), has a feedback for correcting itself. This feedback-controlled system is automated along with the cyber connectivity and the local networking system. This helps in rectifying its errors by the same system, without any intervention of humans.

The physical entities, in major, mentioned in this system are the embedded sensors, communication system and the actuating drives. These systems play a major role in the advancement of many fields including health care, sustainable development, environment, transportation and many fields. Anything can be made smart, i.e., vehicles, factory production lines, medical care units, home appliances or in sometimes the home itself.

3. SMART FACTORY MODEL AND ITS COMPONENTS

A model of a smart factory is made as a production line with four gates, an automatic guided vehicle (AGV), and a server to mediate the communication between all components. Each component has its own independent control system. This small model represents the simple production line of a smart factory would be. In a standard fixed path, the vehicle moves for its stock of beads, from the gates having the stacks. The vehicle receives beads, that are ordered from the server, through the Blynk application in smart phone.

3.1. Smart Factory

The fourth industrial revolution INDUSTRY 4.0 paves the path of smart factory in the society. A smart factory isn't a simple digitization and automation. it is the place where, digital world meets physical world and makes operations on it. This is a system, which is highly flexible system in times and can make decisions on its own, regarding the situation.

This system eliminates the centralized decision on every parts of the factory. Each individual automated production line, takes its own decisions as per the necessity. In an automated warehouse, many automatic guided vehicles run to fetch its necessary products.

Each individual vehicle takes decision based on the real time conditions, on its own, without human intervention. The embedded systems play a major role in these technological improvements and will lead to the conversion of all factories smart enough. Machine learning also plays a major role in making it. In few years, self-intelligent co-robots could be friends of people in production line and help them in many ways.

This merges the real world into the virtual world, because higher levels of automation are needed to build a smart factory. The implementing of cyber-physical system in the production, in a major extent, leads to supervising of production process automatically by the machine itself. This makes the work of humans easier and reduce human errors at many stages.

3.2. Sections of System

The smart factory model has some sections in it, which will communicate each other and form an automated work flow transportation. It has an automated vehicle, travelling in a specified path. Four gates are in the path of this vehicle, where, the vehicle can collect the beads on its necessity. The server plays an important role of receiving and distributing information between the devices. The user interface used for this system is the Blynk application in Android smart phone, connected in common Wi-Fi setup.

3.2.1. Automatic Guided Vehicle

The main component on this smart factory model is the vehicle used for collecting the beads. It is designed in such a way that; it follows the black stripped pathway as its way. To make this in real, some sensors are used and drives are used to drive the vehicle. This vehicle is made to function, based on the circuits, installed and programmed on the Printed Circuit Board (PCB) using Arduino Mega 2560.

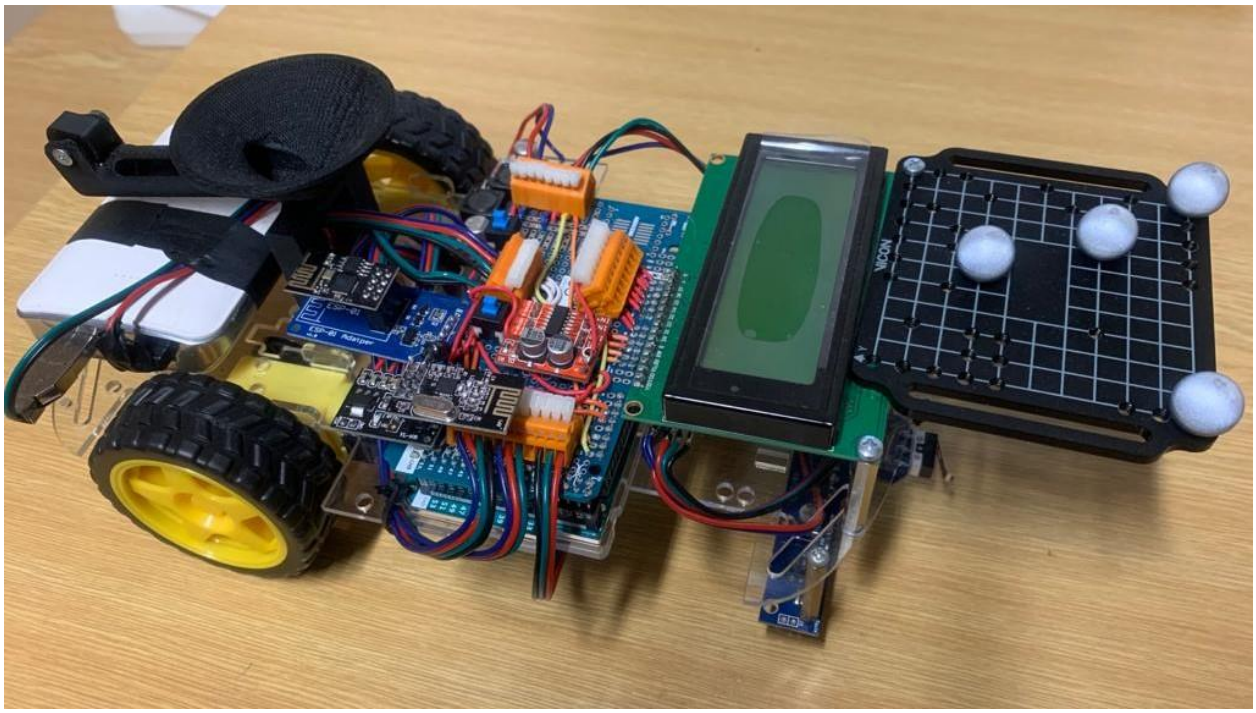


Figure 6. Automatic Guided Vehicle (AGV) [Source: Own]

The vehicle has a Double H-Bridge driver, to drive the two servo motors for each of its wheel. It has a channel tracking sensor module to track the black tape on the floor, to follow the correct path. It has a battery as a power source to operate. The

PCB of the vehicle is attached with two Wi-Fi modules, ESP8266 with nRF24L01 transceivers, each one for each purpose.

One transceiver is for the communication of the Blynk application to the server, and the other one is for the communication between the vehicle and the gates with stacks. The vehicle is also provided with an 20X4 LCD screen to print the information, while communicating with the server and stack. There is a provision in vehicle, for collecting and holding the beads.

3.2.1.1. Construction of Vehicle

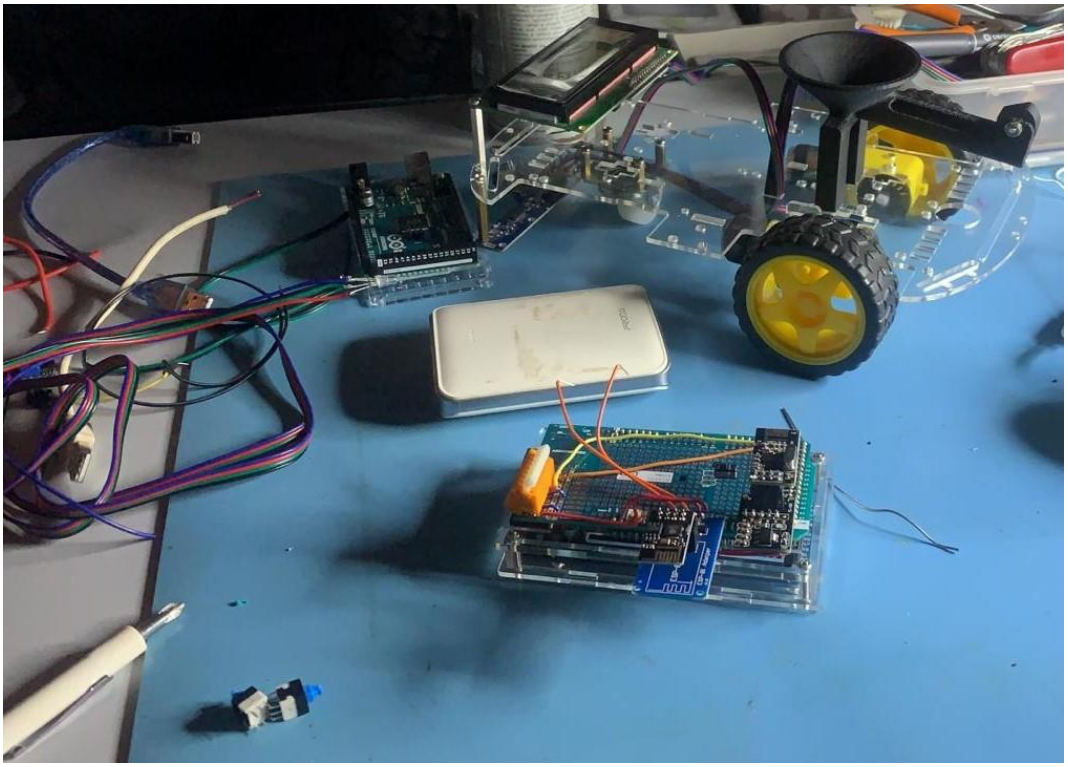


Figure 7. Vehicle Before Construction [Source: Own]

The vehicle is constructed on a plain polymer frame, which holds all components on its top and is placed on the wheels. First, the frame is placed over the servo motors. The movement of the vehicle is controlled by the Double H-Bridge driver. This driver is a double bridge and so it has two 16V operating bridges. So, this driver controls the two servo drives attached with the two wheels of the vehicle. The angular displacement of both the wheels are calculated by the microcontroller, and the H-Bridge drives the wheels. When the vehicle needs to travel and take a turn to right side, the left wheel of the vehicle needs to cover a greater angular displacement, so that the vehicle turns to its right side. This ensures that the

vehicle moves in the correct path. Then, a caster wheel is placed in the center of the frame, as a support wheel to rotate. This support wheel ensures the stability of the vehicle and supports the rear wheels to take a relaxed turn.

A battery is used as the power source of the vehicle and is fixed at the back side of the vehicle. The PCB is used to infuse the components to process into it and is attached to the Arduino board for processing. The PCB is used to reduce the usage of the wires in the construction. This ensures the uninterrupted contact between the pins and so the equipment is electronically stable and free of loose connections as in those with pin wires.

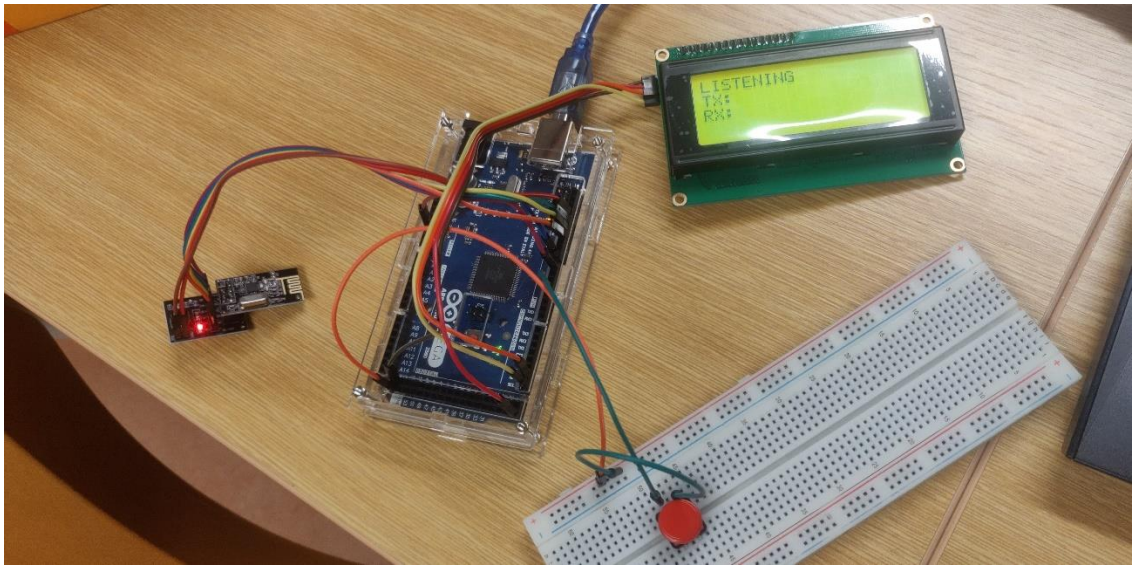


Figure 8. Testing Transceivers [Source: Own]

Two transceivers are installed to the vehicle for communication. Transceivers are the components which are capable of receiving as well as transmitting the messages. One of those is to install communication between the vehicle with the server. The another one is to make communication with the vehicle to stacks. The first one receives order from the server and confirms the arrival of the order. The second transceiver is to make communication with the gates of stacks to collect the color of the beads, mentioned in the order.

An LCD (20*4) screen is placed in the front top of the vehicle to display the communications between the components. This screen is programmed to print the necessary communications. The channel tracking sensors are placed in the front of the vehicle, so that it can guide the vehicle with front way. A small cone shaped

collector of beads is set at the top of the vehicle, so that the vehicle can easily collect the beads from the gates easily.

The transceivers are programmed to transmit and receive data. Using Arduino Mega microcontroller, a button, an LCD screen, and a transceiver, a small connection is made and the program is tested for its effectiveness, as given in Figure 8. The listening and receiving of data are based on the conditions that are imposed to the system for its functioning.

3.2.1.2. Construction of PCB in Vehicle

All the components necessary for the operation of the vehicle are programmed in the PCB and is attached to the Arduino Mega microcontroller. The PCB gives rigid connections to the pins and avoids the interruptions in process. The transceivers in the PCB are provided with step-down transformers, and is connected from Nrf24l01 and the input of ESP8266. The Tx, Rx, pins of the transceivers are connected to the Tx and Rx pins of microcontroller. The Vcc pins of all components are connected to 5V power supply of Arduino board, and the Gnd pins are connected to the Gnd of the microcontroller.

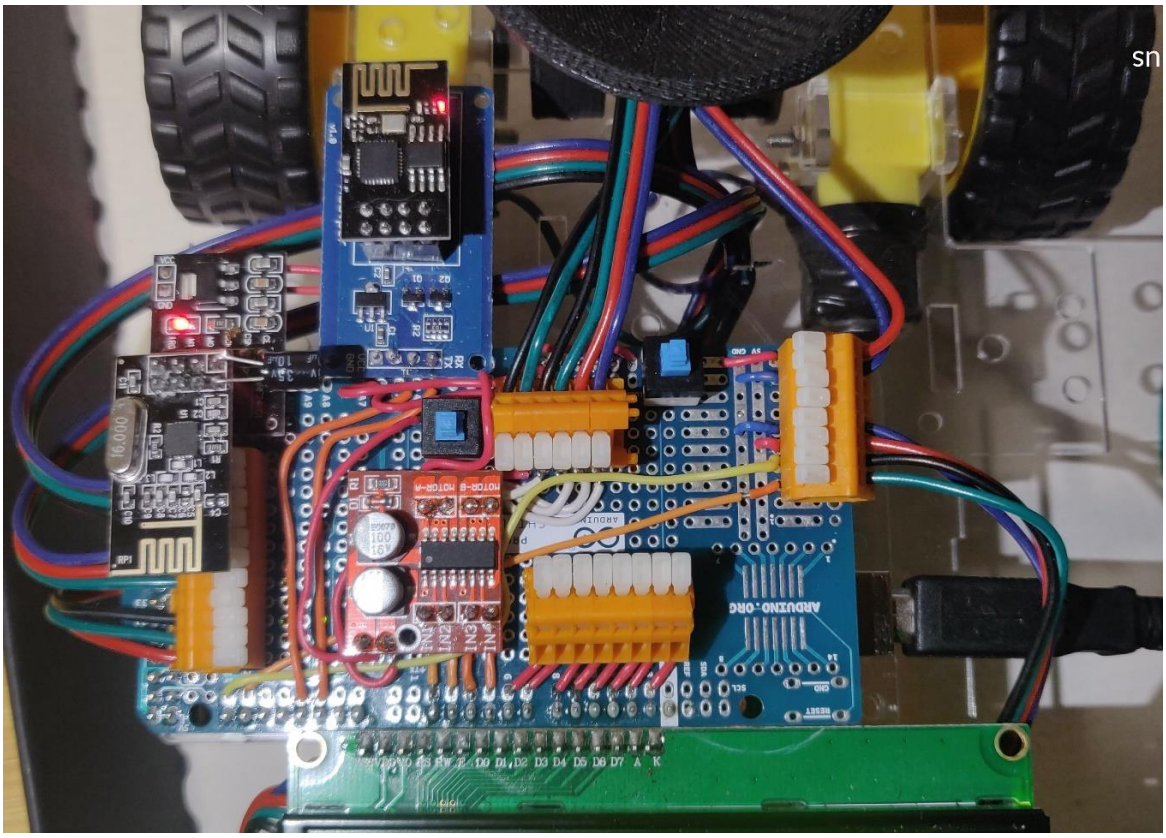


Figure 9. PCB Installed in Vehicle [Source: Own]

The Double H-Bridge driver has four inputs for two motors to run. These four pins are connected with digital inputs of the Arduino boards. The first two inputs are for motor A and the next two inputs corresponds to motor B. The input values are can be modified to make the vehicle run smooth. There are two series of connectors are used for connecting the Double H-Bridge driver with the motor A and motor B. These connectors connect the pins from the driver on one side and the servo drive on the other side.

Using the third connector, the SDA (Serial Data) and SCL (Serial Clock) pins of the I2C module connected with LCD screen is connected to the digital input of microcontroller. The I2C module with PCF8574 chip eases the connection of LCD screen to Arduino board. Another connector connects the channel tracking sensors with Arduino. A reset button is installed using digital pins to reset the microcontroller fed program.

3.2.2. Gates with Stacks

There are four gates in the system, which has stack of beads with a certain color of beads in each gate.

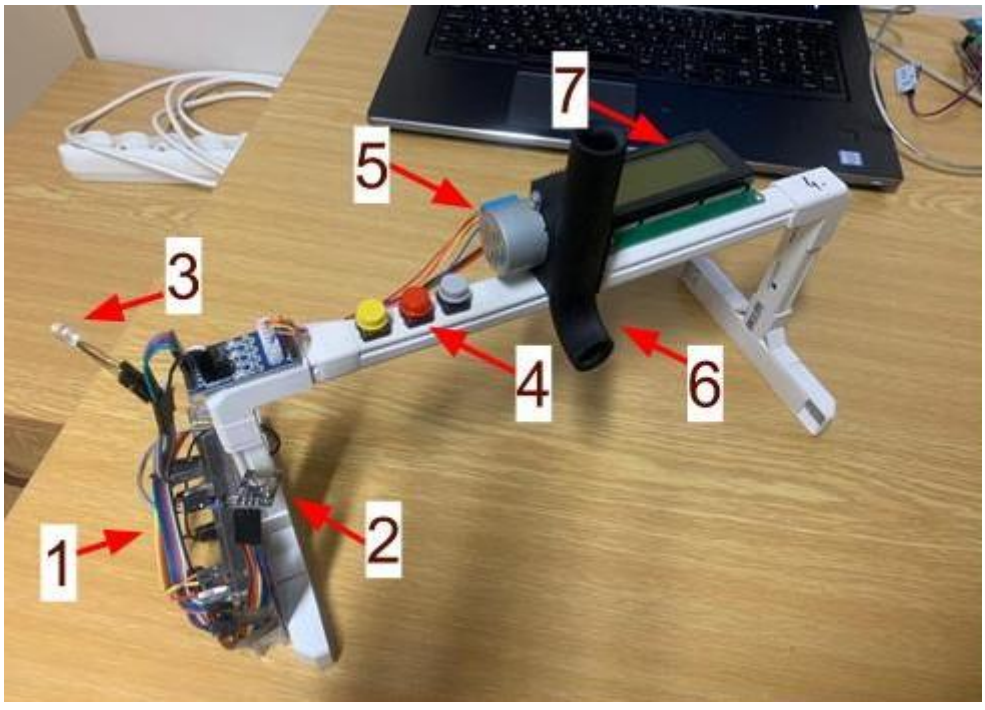


Figure 10. Gate with Stack [Source: Own]

1) Arduino Mega 2560 2) Wi-Fi module NRF24L01 3) LED 4) Buttons
5) Stepper motor 6) Ball magazine 7) Display

These gates are provided with microcontroller Arduino Mega 2560, Wi-Fi module, stepper motor, LCD screen, LED and also buttons. The Wi-Fi module ensures the gate to communicate with the vehicle. Gates only communicate with the vehicle and not connected with the server. The stepper motor is used for releasing the beads to the vehicle. The Arduino Mega 2560 board processes and also controls all the functions to be done by the gate.

The RFID module is set in all the gates and the information about the particular gates is sensed by the vehicle. This is used to identify the color of the beads in that particular gates, by the vehicle. This enhances the program to be dynamic, so that, the vehicle need not to be reprogrammed all times.

when the vehicle moves in a different path, the vehicle is not sure that, it moves through the same order of the gates or not. In this situation the vehicle should be reprogrammed manually to identify its path and the order of the gates it passes through. To overcome this, the RFID sensor is used. This sensor communicates the information about the gates to the vehicle. So, the vehicle is not in a need of reprogramming for each path manually. This makes the system react automatically to the change in the environment without any interventions.

3.2.3. Server

Each system needs a host for controlling and monitoring a loop of process. The host guides and helps as a bridge between the components of the system and the user.

The server makes possible for the communication of the real world to the virtual components. The crucial part of all communications is done by the server. It communicates with all the components in major. The server serves as intermediator and so, receives the order from the user and instructs the vehicle with its order of the beads.

The server for this system is built over an PCB for compactness and rigidity. It is also given with reset button to reset the program fed into the microcontroller. It is provided with an LCD screen to show off its communications with the Blynk application and also with the vehicle. The LCD screen is also connected with I2C for easy connection with microcontroller.

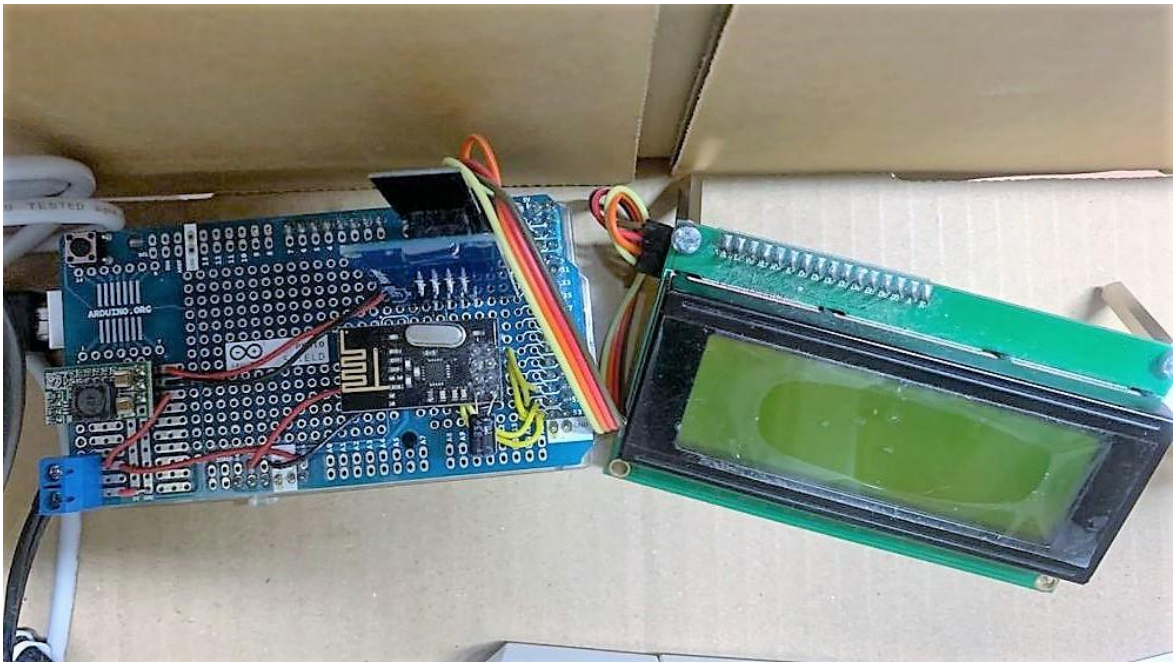


Figure 11. Server [Source: Own]

The server for this smart factory model, has two Wi-Fi modules, one for communicating with the Blynk application in the mobile, to receive order of beads, and the another one to communicate and transfer order to the vehicle. This makes it possible for the order to transmit from the Blynk application to the vehicle.

3.2.4. Blynk Application

Blynk is an open platform of IOT, which provides free cloud network to connect devices in cloud and control them. It is also a platform to connect applications with, things, people, data, with a cloud network.

Through Wi-Fi, Ethernet, LTE or 2G – 4G, the hardware devices are connected with the cloud network. This application uses Blynk library to connect the hardware through any of the above-mentioned forms. As it is an open source cloud, it can run in our local environment too. This makes us easier to use this application to order the beads to the vehicle.

Using this, the equipment can be controlled remotely and multiple devices can also be managed. This supports over 400 hardware modules and works in all smartphones, with Android OS version 4.2+ or IOS version 9+.

In this application, a new project is created, which works on Arduino Mega 2560. The project is named as Smart Factory and five buttons are added to the project. The buttons are for Red, Green, Blue, Clear and Finish. These buttons are created

as push buttons and its respective pins are configured in the project. This can be used to provide order to the server.

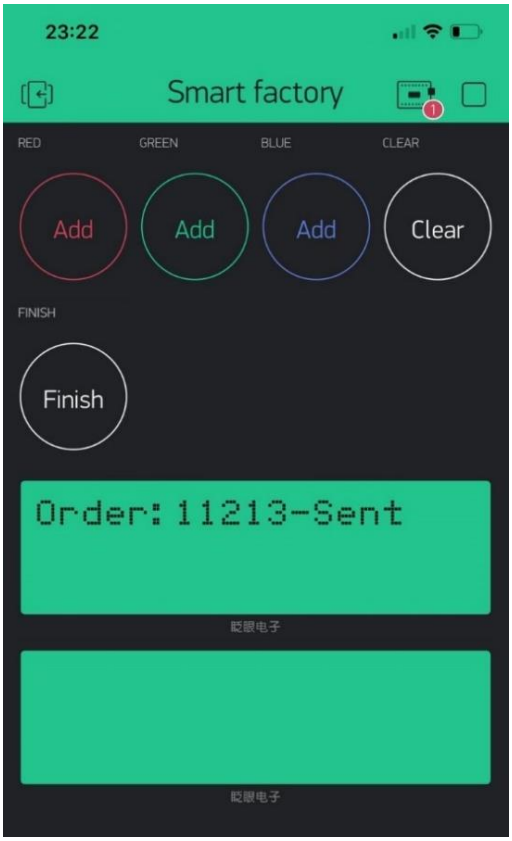


Figure 12. Screenshot of Blynk Application [Source: Own]

The screenshot of this application, depicting the system of smart factory, with an order placed and sent to the server of the system is shown in Figure 12. The order and its status of sent are always displayed, when the order is made.

3.2.4.1. Ordering Procedure

In Blynk application, numbers correspond to the color of the beads to be ordered. The number 1 corresponds to the red colored bead and the number 2 corresponds to the green colored bead and the number 3 corresponds to the blue colored bead. If the order is over and if no other beads are needed, 0 is used. The order is limited to five, because, the limit of the order array is set to five.

First, the user should open the Blynk application in the mobile phone and connect it with the local Wi-Fi connection. Then, based the correspondences of the color of the beads and its numbers, necessary color of the beads is ordered by clicking on the buttons. This educates the Blynk application, with the order necessary by the user. Then, this order can be communicated with the server for further process.

3.3. Components in The System

The smart factory model is set by assembling the above explained systems. These systems are constructed with the help of many individual electronical components as well as sensors, microprocessors, buttons, stepper and servo motors, and many. These components serve a specific purpose for the system and contributes its part of work to the system. They are chosen with a particular task to do on the system, and are as follows.

3.3.1. Arduino

Electronic microcontroller boards, which can read inputs, process it and produce an output. These boards get inputs from physical electronic components such as, sensors, buttons, or any device, and also as message from software like Blynk application. It gives its output, as running physical components like, lighting led, run servo motors or stepper motors, switching spray nozzles, etc., and also transmits messages or information through modules to software for processing. These microcontrollers, process the information by input devices to decide to act on output device. They work on the conditions, which are programmed on the boards. The programs are made using Arduino programming language and is made in Arduino software called IDE. These programs with conditions are fed into the boards through USB cables from computers.

3.3.1.1. Arduino Mega 2560

Figure 13. Arduino Mega 2560[12]

This microcontroller board is one of the Arduino microcontroller boards, which have its own specification in it. It is an updated version of Arduino mega board, which have more advantages over Arduino mega board. This board have 54 digital input or output pins and 16 analog inputs. It has 4 Universal Asynchronous Receiver / Transmitter, for the data transmission. The smart factory model needs this to process many Transceivers. So, this microcontroller is used in this model, for the server and also for the vehicle, which operates two Transceivers.

The Figure 14 provides the top view of the Arduino board, showing its digital and analog pins, power pins, AT mega 2560 processor, power supply port, USB interface, reset button and power LED.

3.3.2. Transceiver (Esp8266 & Nrf24l01)

This ESP8266 in Figure 14, is a supporting component of the microcontroller, and serves as an input module for Arduino microcontroller board. It works at 3.3V power supply, so the voltage that Arduino operates, differs from it. To overcome this, the voltage is step down to 3.3V, when the component needs to for use.

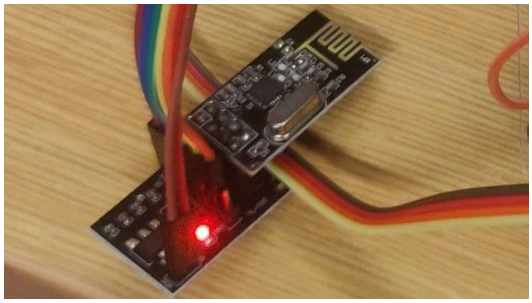


Figure 14. Esp8266 & Nrf24l01 [Source: Own]

It is used for application of IOT in real life i.e., it establishes Wi-Fi communication for the machines to communicate. This module is also a microcontroller chip, which has 32-bit RISC processor in it. This processor is created by Tensillica company. It functions on a Real Time Operating System (RTOS). The main advantage of using this module is its low power consumption.

3.3.3. Channel Tracking Sensor

The 5-channel line tracking sensor is used in the system. It works on 5V power supply and have five output pins from S1, S2, S3, S4, S5, and one from bump switch is CLP digital and another from IR obstacle Near digital. These pins are soldered and connected using connectors to the vehicle's microcontroller.

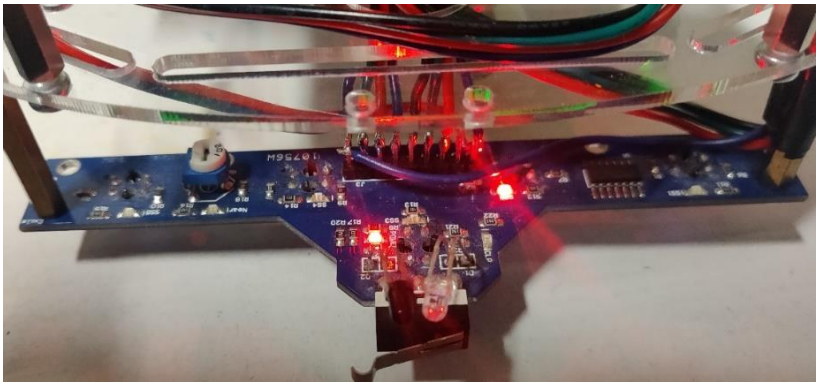


Figure 15. Channel Tracking Sensor [Source: Own]

3.3.4. Double H-Bridge Driver

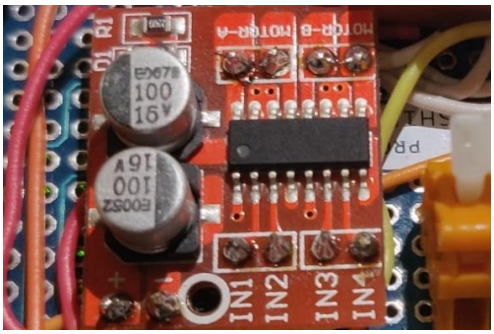


Figure 16. H-Bridge [Source: Own]

The Figure 16 shows the Double H-Bridge driver, used for controlling the two servo motors of the vehicle. The two pins, in1 and in2 act as input pins and are used for controlling the spinning of the motor A. The two pins, in3 and in4 act as input pins and are used for controlling the spinning of the motor B. The ENA and ENB pins enable PWM signal for motor A and motor B respectively.

3.3.5. RFID Sensor

The RFID sensors are placed in all gates to give the information about the stack to the vehicle. The SDA, SCK, and RST pins are connected to the Arduino as input pins and reset pins. The MISO and MOSI pins are used sending data to vehicle.

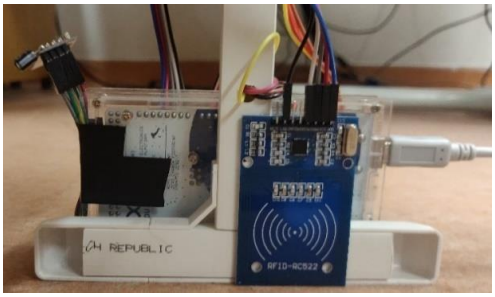


Figure 17. RFID Sensor [Source: Own]

4. AUTOMATONS

To represent the work flow, a simple graphical way of representing a flow of work, based on conditions are to be developed. The automaton graph is such representation and helps in distributing the conditions to various states. Based on conditions, the decision on the flow of work is decided to proceed. These diagrams also help us to check the interruptions in the work flow, due to any circumstances, in an ease way.

By these diagrams, the work is split up into many parts and the conditions for the work flow is checked using many states. This way of splitting of the conditions, programing and solving solutions gets easier.

These diagrams have three parts namely,

1. States / Nodes
2. Conditions
3. Action

4.1. State / Nodes

It is the finite state of an automaton. It represents the stage, where, the flow of program is actually in. Basically, this is denoted by circles, and the work flows from one state to another, during the execution.

4.2. Conditions

There are the conditions checked at every state, and the flow of work is decided to proceed. There are many conditions checked at each state of the automatons. When, any one of the conditions is satisfied, the program proceeds to next state.

4.3. Action

These are the work done during the flow in the diagram. When a condition is satisfied, in a state, the work flows to next stage. But, before moving to next stage, the work done in the system is called action. These actions include all works, starting from switching on an LED diode to the complex work on transmitter or receiver of signals and controlling any devices.

5. FUNCTIONING OF SMART FACTORY MODEL

5.1. Server Functioning - Working of buttons

The server is the host of whole system and transfers the order of beads. To make sense to it, the working of the buttons is to be organized. Using Blynk application, a system is created with five buttons.

Each of these buttons' notices to a unique function. They are used to place order of the color beads from the stack. One button represents red color, other one represents green color, next one represents blue color. These three buttons are for ordering the colors, whereas, the next two is for clearing the order and finishing the order respectively.

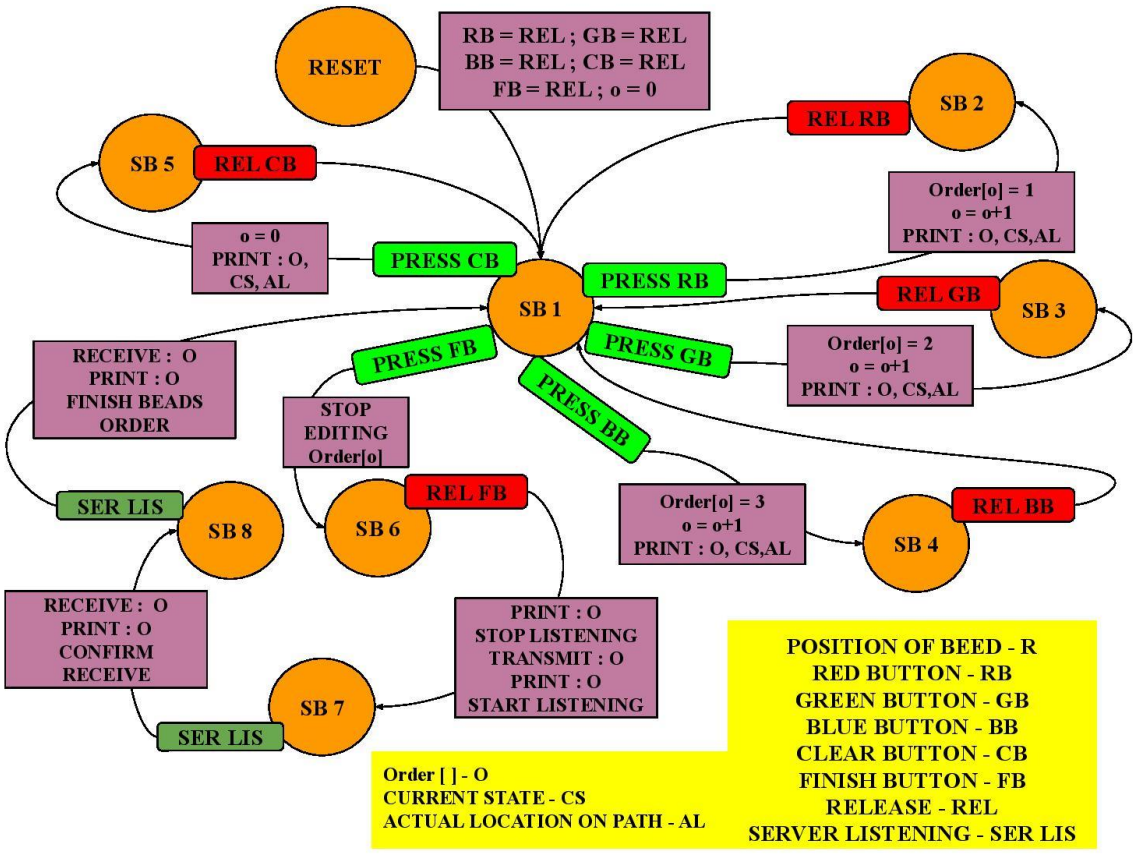


Figure 18. Automaton of State of Buttons [Source: Own]

Using a simple graphical representation of automaton, its functioning during the order is explained. The automaton graph tells us clearly about the conditions, which are need to be checked at each state of the buttons are specified, and its respective action needed to be done if the conditions are satisfied are mentioned

in a clear manner. The mentioned automaton graph in Figure 18, is made for the state of the buttons. This must be understood along with the Figure 12.

From reset, the buttons regarding to red, green, blue, clear and finish button is set to release. The order array is set to zero. The state of the button is now at first state.

Here, at this first state, the buttons are waiting for the user to order the beads based on its color needed. If the user presses the button for red color bead, then, three actions are performed. Those three actions are,

1. The first cell of the order array is set to the number corresponding with the color of the bead ordered. In this case it is "1" as it corresponds to red color.
2. Then, the cursor for input is set to the next cell of the array for taking order for next cell.
3. Next to it, the current state of buttons, actual location path and the order array is printed on the serial monitor of the Arduino software.

At this time, the state of buttons reaches to next state and waits for the release of the same button. After releasing the button, it jumps to the first state for receiving next order in next cell for next bead. The above mentioned three actions are done in circulation for all the red, green and blue buttons alone. The "clear" and "finish" buttons work different. This process continues till the user finishes the order by pressing the finish button.

When the clear button is pressed, the order array is made empty, by replacing all cells of the order array with zeros, and it waits for the releasing. When the button is released, it falls to the first state of buttons. When the finish button is pressed, these loops in receiving order from the user is stopped and remaining cells in order array is set to zero.

After releasing the finish button, the Blynk application stops receiving the order prints the order to serial monitor and transmits the order to the vehicle. Again, the server starts listening. The vehicle receives the order and proceeds for collecting the beads. The receiver of the data is confirmed after state seven. After the vehicle's confirmation, the order is ended and this process remain same for the state of buttons.

5.1.1. Programming of Server

The server is programmed in order to receive the order from the user and transfer to the vehicle.

Initializing Variables for Server

The necessary input variables of the buttons are initialized to zero, with no order. These variables make sense as, each of the variable serves a purpose. The mentioned colored buttons with red, green, and blue are initiated as bR, bG, bB. The clear and finish buttons are initiated as bClear and bFinish. The state of the buttons is as Sb, and the main server state as Ss.

```
int s=0;   //step variable
int Sb=1;  //state of buttons automaton
int Ss=1;  //state of main server automaton
int lastc2=0;
        //variables to control blynk buttons
int bR=0;
int bG=0;
int bB=0;
int bClear=0;
int bFinish=0;
```

Figure 19. Initial Variables of Server [Source: Own]

Function "void setup()"

```
void setup() {
  pinMode(12, INPUT_PULLUP);
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.clear();
  printLcd(0,"Server-initializing");
  printLcd(1,"TX:");
  printLcd(2,"RX:");
  EspSerial.begin(ESP8266_BAUD);
  delay(1000);
  Blynk.begin(WIFI_AUTH, wifi, WIFI_SSID, WIFI_PASSWORD);
  //timer.setInterval(1000L, sendSensor);
  myRadio.begin();              //WIFI NRF
  myRadio.setChannel(0x60);
  myRadio.setPALevel(RF24_PA_MAX);
  //myRadio.setDataRate( RF24_250KBPS );
  printLcd(0,"Server-order beads");
  printLcd3();
  sendMessageToPhone3();
}
```

Figure 20. Server "Void Setup()" Function [Source: Own]

From the Figure 20, the setup of the module ESP8266 is used for the Wi-Fi communication is programmed. This is setup as primary action and done in the

setup part itself. The "EspSerial.begin" command starts the work of the component and begins it.

This setup needs SSID and its corresponding PASSWORD for the access of the network. The local network's ID and its corresponding password is provided in the program, to give access to the Arduino to use it. Then the server is made to listen to the Blynk application.

Function "void loop()"

```
void loop() {
  Blynk.run();
  if (Ss==1){ SbAutomaton();}
  }
```

Figure 21. Server "void loop()" Function [Source: Own]

The loop function calls the timer of the function and then calls the "void SbAutomaton()" to execute the program.

Functions in Server Program

```
int readCodeFromWifi() {
  while (myRadio.available()) {
    myRadio.read(&dataRecieve, sizeof(dataRecieve));}
}
void sendMessageToPhone(String message) {
  lcdPhone.clear();
  lcdPhone.print(0, 0, message); }
void sendMessageToPhone3() {
  lcdPhone.clear();
  lcdPhone.print(0, 0, "Order:"+String(Order[0])+String(Order[1])+
  String(Order[2])+String(Order[3])+String(Order[4])); }
BLYNK_WRITE(V0) {bR = param.asInt();}
BLYNK_WRITE(V1) {bG = param.asInt();}
BLYNK_WRITE(V2) {bB = param.asInt();}
BLYNK_WRITE(V3) {bClear = param.asInt();}
BLYNK_WRITE(V4) {bFinish  = param.asInt();}
```

Figure 22. Functions in Server Program [Source: Own]

The function "readCodeFromWifi()" reads the code, when data is received. For sending message to phone and to print the order to phone, two functions are created. The 'V0', 'V1', 'V2', 'V3', 'V4', are made to sync with Blynk.

State Sb1 in "void SbAutomaton()"

The Figure 23 shows the first state of buttons and the actions corresponding to the buttons of three colors and the clear and finish buttons are separately fed. If red

button is pressed, the respective next state 2 is assigned and the order is set to 1 for first one and increases it by 1. This is done for all three buttons for red, green and blue. The condition," bclear == 1" is made true, all the cells are infused with 0. In all these processes, "sendMessageToPhone3()" function is called and the phone LCD is printed with the order, as in Figure 12. Only when, the finish button is pressed, it jumps to the state 6 and does all these actions.

```
void SbAutomaton(){
  if (Sb==1) {if (bR==1) {Sb=2;
                          Order[o]=1;
                          if (o<4) o=o+1;
                          printLcd3();
                          sendMessageToPhone3();  }
              if (bG==1) {Sb=3;
                          Order[o]=2;
                          if (o<4) o=o+1;
                          printLcd3();
                          sendMessageToPhone3();  }
              if (bB==1) {Sb=4;
                          Order[o]=3;
                          if (o<4) o=o+1;
                          printLcd3();
                          sendMessageToPhone3();  }
              if (bClear==1) {Sb=5;
                              Order[0]=0;
                              Order[1]=0;
                              Order[2]=0;
                              Order[3]=0;
                              Order[4]=0;
                              o=0;
                              printLcd3();
                              sendMessageToPhone3();  }
              if (bFinish==1) {Sb=6;  }
              }
```

Figure 23. State Sb1 of State of Buttons [Source: Own]

State Sb6 in "void SbAutomaton()"

The state S6 is reached, when the 'Finish' button is pressed and released, the order is printed on the phone screen using "lcdPhone.print()" command. Then the server stops listening using "myRadio.stopListening()" and transmits the order array to the vehicle with "dataTransmit." Command. Also, Tx, Rx, Val are transmitted using same command. The variable 'c' also transmitted with an increment.Now, the server opens the writing pipe with "myRadio.openWritingPipe(addresses[0])" and writes the data to transmit. Then the transmitted order is printed on LCD screen using "lcd.print" mentioned in the Figure 24.

Then, the server opens the reading pipe by the "myRadio.openReadingPipe(1, addresses [0])" command and starts listening to the phone. Then the state is set to 7. The intermediate states S2, S3, S4, S5, checks for the red, green, blue and clear buttons to release and, when it satisfies, the state is set to S1.

```
if (Sb==6) {if (bFinish==0) {
   printLcd(0,"Server-order sent");
   lcdPhone.clear();
   lcdPhone.print(0, 0, "Order:"+String(Order[0])+
   String(Order[1])+String(Order[2])+String(Order[3])+
   String(Order[4])+"-Sent");
   myRadio.stopListening();      //WIFI NRF TRANSMIT
   dataTransmit.Order[0]=Order[0];
   dataTransmit.Order[1]=Order[1];
   dataTransmit.Order[2]=Order[2];
   dataTransmit.Order[3]=Order[3];
   dataTransmit.Order[4]=Order[4];
   dataTransmit.c=dataTransmit.c+1;
   dataTransmit.tx=1;
   dataTransmit.rx=2;
   dataTransmit.val=2;
   myRadio.openWritingPipe(addresses[0]);
   myRadio.write(&dataTransmit, sizeof(dataTransmit));
   myRadio.openReadingPipe(1, addresses[0]);
   myRadio.startListening();
   printLcd(0,"Server-listening");
   Sb=7;
   }
 }
```

Figure 24. State Sb6 of State of Buttons [Source: Own]

States Sb7 and Sb8 in "void SbAutomaton()"

After the state Sb6, the state is made to set to 7. Here, when the network is available, the server receives the transmitted messages and prints the messages on the LCD screen and then confirms receiving of messages and sets the state to Sb8. In state Sb8, the order is received from the phone and receives the order and finishes the order and prints the order is finished. Then the state is set to Sb1 and the loop continues.

```
if (Sb==7) {while (myRadio.available()) {
            myRadio.read(&dataRecieve, sizeof(dataRecieve));}
            if (lastc2<dataRecieve.c&&dataRecieve.rx==1) {
            lastc2=dataRecieve.c;

            printLcd(0,"Server-conf received");
            delay(3000);
            printLcd(0,"Server-waiting end");
            Sb=8;
            }
           }
if (Sb==8) {while (myRadio.available()) {
            myRadio.read(&dataRecieve, sizeof(dataRecieve));}
            if (lastc2<dataRecieve.c&&dataRecieve.rx==1) {
            //Prints Received order
            printLcd(0,"Server-order finished");
            delay(3000);
            printLcd(0,"Server-order beads");
            Sb=1;
           }
       }
```

Figure 25. States Sb7 and Sb8 of State of Buttons [Source: Own]

5.2. Functioning of Gates with Stacks

The gates with stacks of beads are an important section in the system there are four gates installed, with the red, green and blue beads. They are installed with many parts including Arduino, as mentioned in Figure10. These Arduinos are programmed in such a way to listen to the vehicle and communicates with it. Over communication, if the vehicle's necessary bead is possessed by that stack, then, the gates operates the stepper motor in a way that it releases the bead for the vehicle.

The MISO and MOSI pins of RFID sensors connected to this Arduino, sends information about the color of the bead the gates have, to the vehicle. This enhances the program as this brings out a greater advantage that, the vehicle need not have the path in memory and also the order of the gates consecutively. The color of the beads in the gates is provided to the vehicle and it stores that color and compares the order with this stored color.

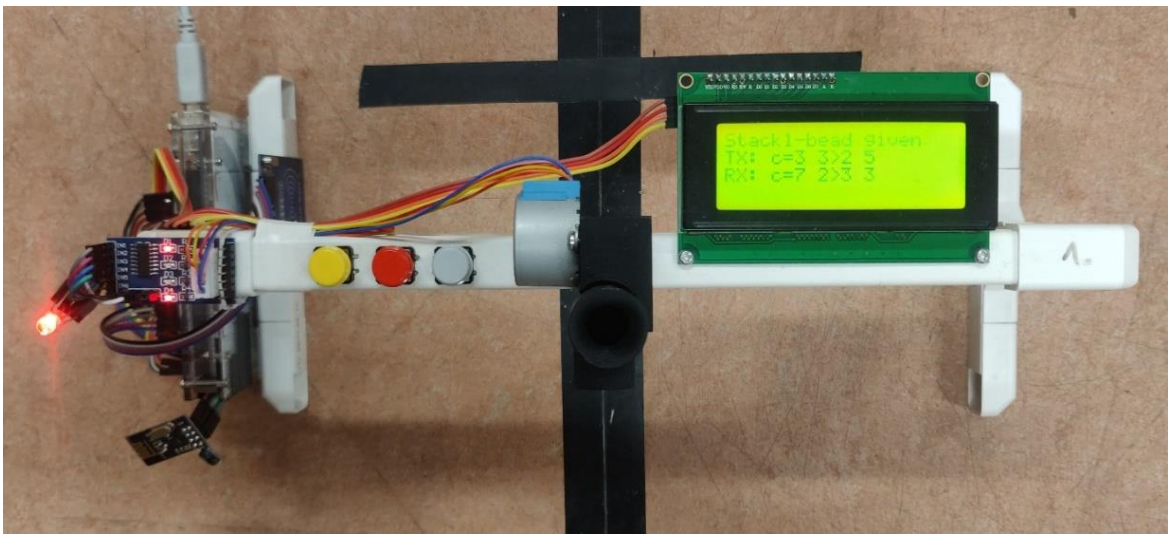


Figure 26. Gate in Operation [Source: Own]

The gates are also attached with an LED of the color of the bead it stacks. The Figure 23 gives an idea about the gates, while functioning. The LCD screen on gates provides information about releasing the beads and the values of TX and RX. As like this gate, four gates are placed in the path of the vehicle.

The gates are placed in a way that, the ball magazine is exactly above the black strip, which the channel tracking sensors tracks. This helps the vehicle on collection of beads from gates.

5.2.1. Programming of Stacks

The stacks are programmed in such a way that it functions to the requirements of the system mentioned above. They respond to the vehicle and releases the beads, when necessary.

Initializing Variables for Stacks

```
int GateN=4;
byte addresses[][6] = {"0"};
struct package {
  //char text[20]="00000";
  int Order[5]={0,0,0,0,0};
  int c=1;
  int tx=GateN+2;
  int rx=0;
  int val=0;
};

typedef struct package Package;
Package dataRecieve;
Package dataTransmit;
int OLDvariable = 0;
int S=1;
int lastc1 = 0;
int Sb=1;
bool calib;
bool color;
int LED;
const int in1 = 23;
const int in2 = 25;
const int in3 = 27;
const int in4 = 29;
int vstepper = 1;
```

Figure 27. Initial Variables of Stacks [Source: Own]

In the starting of the program, the necessary variables are initiated before the setup process. The Figure 24, shows the important initial variables necessary for programming. A structure 'package' is setup with a cleared order array and the Tx, Rx, are programmed. For establishing the Wi-Fi communication for receiving and transmitting the data, under "typedef struct package Package", the "Package dataRecieve" and "Package dataTransmit" are initiated.

The state is initiated with 1, and the Boolean variables 'calib' and 'color' are initiated. The in1, in2, in3, in4, represents the pins of stepper motor and so its respective pins connected in microcontroller are fed. The 'vstepper' provides the velocity of the stepper motor.

Function "void setup()"

As in Figure 25, the 'pinMode' is used to configure, the pins to act as input pins or as output pins. The in1, in2, in3, in4, pins of stepper motor are set as output pins and so receives the instruction and acts accordingly. To calibrate the buttons, pins 5, 6, 7, 8, are used to enable internal pullup resistors using 'INPUT_PULLUP' in 'pinMode'. The LCD screen is initiated and is printed with 'tx', 'rx', 'GateN' and "initializing". Then, the GateN is assigned with the respective LED, as per conditions. The Wi-Fi communication channel is initiated and the stack is set listening and the LCD screen is printed with "Listening".

```
void setup()
{
  Serial.begin(9600);
  delay(1000);
  pinMode(LED, OUTPUT);        //RGB LED
  pinMode(in1, OUTPUT);        //STEPPER
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(8, INPUT_PULLUP);    //CALIBRATE BUTTON
  pinMode(7, INPUT_PULLUP);
  pinMode(6, INPUT_PULLUP);
  pinMode(5, INPUT_PULLUP);
  lcd.init();                  // LCD
  lcd.backlight();
  lcd.clear();
  printLcd(0,"Stack"+String(GateN)+"-initializing");
  printLcd(1,"TX:");
  printLcd(2,"RX:");
  if (GateN==1) LED=13;        //12=green
  if (GateN==2) LED=12;        //11=blue
  if (GateN==3) LED=11;        //13=red
  if (GateN==4) LED=12;
  if (GateN==5) LED=13;
  analogWrite(LED, 20);        //RGB LED
  myRadio.begin();             //WIFI NRF
  myRadio.setChannel(0x60);
  myRadio.setPALevel(RF24_PA_MAX);
  //myRadio.setDataRate( RF24_250KBPS );
  myRadio.openReadingPipe(1, addresses[0]);
  myRadio.startListening();
  printLcd(0,"Stack"+String(GateN)+"-listening");
}
```

Figure 28. Stack "void setup()" Function [Source: Own]

State S1 in "void loop ()"

The stack is always maintained in state 1. The stack will always be listening, except the time in transmitting. When the vehicle is available, as in Figure 26, the stack is made to receive the data and compares with the 'lastc1' variable. If that condition satisfies, using "printLcd()" function, the information of which bead is asked and with a delay of 3 seconds, the bead given is also printed. Then the "bead()" function is called to run. Then the LED in the stack is made to blink by "analogWrite(LED, 250)" for glowing and "analogWrite(LED, 10)" to dim the LED, with a delay time of about 1 second.

Within this period of time, the bead function rotates the stepper motor as necessary, and bead is released from the stack. then, the LCD screen in the gate is printed with the stack number and mentioning the bead given as shown in Figure29.

Then the gate stops listening by "myRadio.stopListening()" and transmits tx=GateN+2, rx=2, val=5, and 'c' is incremented and transmitted. After transmitting the values, it writes using "myRadio.openWritingPipe(addresses[0])" and then starts listening.

```
void loop()
 {calib=digitalRead(7);         //CALIBRATE BUTTON
  if (!calib) rotationCW();
  if (S==1) {
      while (myRadio.available()) myRadio.read( &dataRecieve, sizeof(dataRecieve) );
      if (lastc1<dataRecieve.c&&dataRecieve.rx==GateN+2&&dataRecieve.val==3)
       {lastc1=dataRecieve.c;
        printLcd(2,"RX: c="+String(dataRecieve.c)+" "+String(dataRecieve.tx)+">"+
        String(dataRecieve.rx)+" "+String(dataRecieve.val));
        printLcd(0,"Stack"+String(GateN)+"-bead asked");  delay(3000);
        printLcd(0,"Stack"+String(GateN)+"-giving bead");
        bead();
        analogWrite(LED, 250);    //RGB LED intensity HI
        delay(750);
        analogWrite(LED, 10);     //RGB LED intensity LO
        delay(750);
        analogWrite(LED, 250);    //RGB LED intensity HI
        delay(750);
        analogWrite(LED, 10);     //RGB LED intensity LO
        printLcd(0,"Stack"+String(GateN)+"-bead given");  delay(3000);
        myRadio.stopListening();              //WIFI NRF TRANSMIT
        dataTransmit.c=dataTransmit.c+1;
        dataTransmit.tx=GateN+2;
        dataTransmit.rx=2;
        dataTransmit.val=5;
        myRadio.openWritingPipe(addresses[0]);
        myRadio.write(&dataTransmit, sizeof(dataTransmit));
        printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"+
        String(dataTransmit.rx)+" "+String(dataTransmit.val));

        myRadio.startListening();
        S=1;
       }
      }
```

Figure 29. State S1 of Stack [Source: Own]

Function "void bead()"

As in the Figure 30, the full rotation of the stepper motor of 360deg is split into 8 portions and the "rotationCW()" function is made run for 64deg, using 'for' loop. This function is used and called for delivering the beads to the vehicle.

```
void bead() {
//rotation 360 = 512 steps
  for(int i=0;i<(64);i++){
    rotationCW();
  }
}
```

Figure 30. Function "void bead()" [Source: Own]

Functions "void rotationCW()" and "void rotationAntiCW()"

The function 'bead', calls the "rotationCW()" function in its loop for many times. This function in Figure 31, gives the order of calling the step function for the clockwise and anti-clockwise rotation. This calls all the eight step functions in the order mentioned in Figure 28 to rotate the stepper motor clockwise. The reverse calling of these step functions, makes the motor rotate anti-clockwise direction.

```
void rotationCW() {
  step1();
  step2();
  step3();
  step4();
  step5();
  step6();
  step7();
  step8();
}
```

```
void rotationAntiCW() {
  step8();
  step7();
  step6();
  step5();
  step4();
  step3();
  step2();
  step1();
}
```

Figure 31. Rotation of Stepper Motor [Source: Own]

Functions "void step1()" and "void step2()"

```
void step1(){
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  delay(vstepper);
}
```

```
void step2(){
  digitalWrite(in1, HIGH);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  delay(vstepper);
}
```

Figure 32. Functions Step1 and 2 for Rotation [Source: Own]

The step functions called in the "rotationCW()" function are mentioned in Figure 32. There are 8 step functions in the program, which gives the pins in1, in2, in3, in4, with high and low current with a delay of the velocity of motor 'vstepper' in end. The high and low of the power in all steps are shown below.

Inputs of Stepper Motor in All 8 Steps

	in1	in2	in3	in4
step1	High	Low	Low	Low
Step2	High	High	Low	Low
Step3	Low	High	Low	Low
Step4	Low	High	High	Low
Step5	Low	Low	High	Low
Step6	Low	Low	High	High
Step7	Low	Low	Low	High
Step8	High	Low	Low	High

Figure 33. Inputs of Stepper Motor in All Steps [Source: Own]

The Figure 30 shows inputs received by all the four pins, pins in1, in2, in3, in4, in all steps 1 to 8 are shown. Based on the mentioned inputs from Figure, the step functions are created as in Figure 30.

5.3. Vehicle Functioning

The vehicle is the crucial part of the system, as the vehicle is only responsible for obtaining the correct colored beads and bring it to the correct position. The functioning of vehicle is set to have three states, and so it acts in such a way the states implies. An automaton graph is made to make this process easier and to increase the effectiveness.

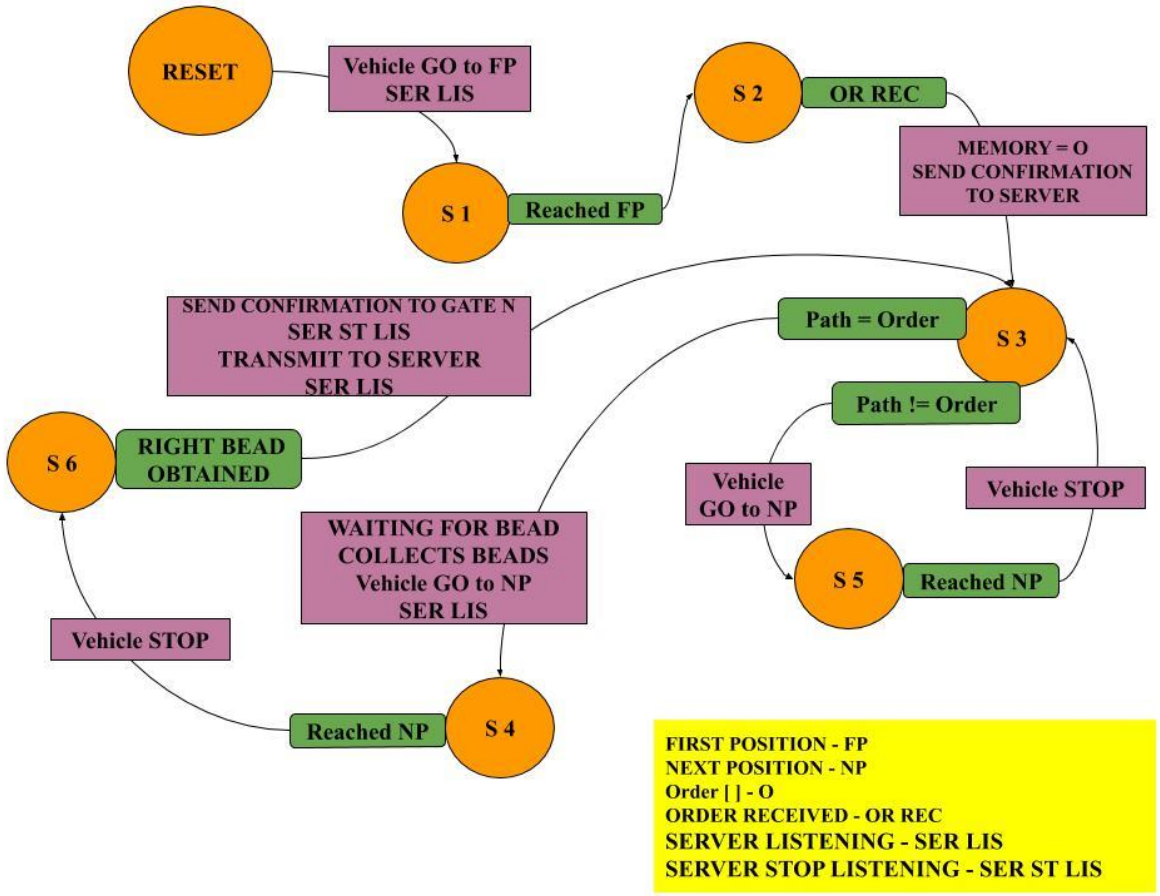


Figure 34. Automaton of Functioning of Vehicle [Source: Own]

The automaton graph in Figure 31, describes the functioning of the vehicle in the smart factory model. When the system starts functioning, the vehicle is listening to the server for receiving the order from the server.

After receiving the order, the memory is fed with it. When the order is same with stack, it collects beads and stops listening and transmits confirmation to Gate N or the vehicle moves to next position. And this is done till collecting all beads that are ordered. The form of checking the order with the gates are made using many strategies to optimize receiving of beads.

5.4. Optimization of Vehicle Movement

There are four gates in the system, through which, the vehicle can pass through. Each gate has a stack of beads with its respective colors. The vehicle is designed in a way to follow a specific path upon a black label, using position sensor. While moving along the path, the vehicle collects beads from each gate and fulfills the order from stack.

During this process, the vehicle follows a pattern to check with the necessary beads, from the order, and to collect it from the gates. This pattern of vehicle needs to be optimized, in order to improve the way, the vehicle collects the beads.

This step by step optimization, improvises the efficiency of the system. This optimization process involves three strategies as it improves the efficiency of the movement of vehicle in receiving the ordered beads in a steadily increasing manner. Each strategy, has its own improvements. The main aim of implementing these strategies is to optimize the moving pattern used by the vehicle.

5.4.1. Strategy 1

This is the basic strategy used for the movement, and is very simple to proceed. The vehicle checks with each of the cell in the order array, and passes to the next position. Here, the first cell of the array is taken into account, and the vehicle is made to move from one stack to other for collecting the corresponding order made in the first cell of the array.

Only when the order of the first cell is collected, the second cell of the array is taken into account, and the vehicle moves from one stack to other to collect it and goes on, till collecting all the beads from the order.

5.4.1.1. Work Flow Description

First, the vehicle moves to first position. Then, the vehicle asks for the bead in the first cell of the Order array to the first stack. If the necessary color bead in the first cell is in that stack, the bead is obtained from the stack and then the vehicle moves to next position. If not, the vehicle moves to next position and asks for the same bead in the first cell, to the next gate also, and goes on.

Only, after getting that bead from any of the gates, the second cell of the order is asked to the gates, and receives the beads one after one, as in the order array. This continues, till the vehicle collects all the necessary beads from the gates. Thus, all the ordered beads are obtained by the vehicle and goes to idle position.

5.4.1.2. Hierarchy of Commands

With reference to the ideology of using strategy 1, and the work flow explained above, the commands are created. These commands provide information about the things the vehicle needs to concentrate at each stage.

The vehicle follows the commands as orders and does it step by step. These commands are used for making the automaton graph of this strategy, which is necessary to build the Arduino program with split condition checks. The command flow based on strategy 1 is as follows.

1) Reset.
2) Go to initial position.
3) Use Position sensor, to find path.
4) If the position reached, Receive color of the gate.
5) Compare Order [i] and received color of the gate.
    a) If same, Obtain bead, i<=i+1, and Go to next position.
    b) If they are dissimilar, Go to next position.
    c) If Order [i], i == 6, OR If Order [i] == 0, jump to point 7.
6) Jump to point 3.
7) Go to initial position.

5.4.1.3. Work Diagram with Strategy 1

Based on the flow of commands, the automaton graph is to be designed to help with a good programming. The commands play an important role in splitting of states and the conditions to be checked at states. The actions to be performed, when the conditions satisfy, are derived from the commands.

The automaton graph in Figure 32, gives a clear idea about the vehicle movement using strategy 1. There are two states, where the conditions are checked. The state S2 checks for the conditions to get idea of receiving the beads or to leave the position. Whereas, the state S3 is to check with channel tracking sensor pins, that, the vehicle reached next position.

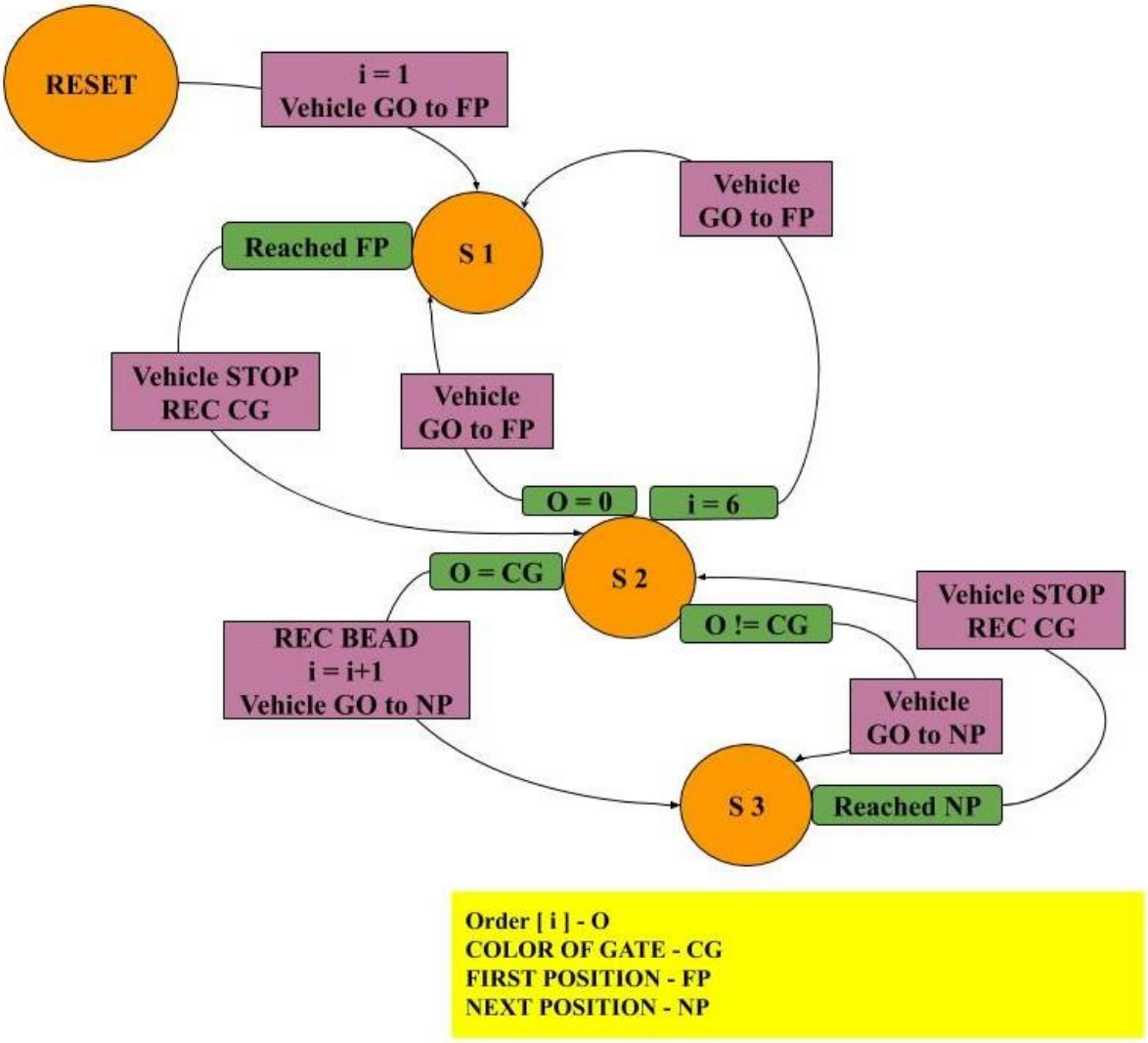


Figure 35. Automaton of vehicle on strategy 1 [Source: Own]

The state 1 helps in making the vehicle move to the first stack. The second state is the condition of the vehicle in each stack. It checks for the color of the bead in stack with the cell of the order received, and if the necessary bead color is same as of the gate the vehicle stands, the bead is received by the vehicle and the next cell of the order array is taken into account and made to move to next position. If it doesn't match, the vehicle is made to move to next position.

When reaching next position, vehicle stops on that position, and receives the color of the gate and continues with state S2. When the value of order array is '0' or the number of cells is '6', the vehicle moves to first position and stops.

5.4.1.4. Programming of Vehicle with Strategy 1

The vehicle is programmed based on the automatons in Figure 31 and Figure 32 in such a way that it follows the strategy 1.

Initializing Variables for Vehicle

The Figure 36, shows the variables that are given as inputs such as, 'FSpin', 'CSpin', 'LSpin', 'LLSpin', 'RSpin', 'RRSpin', 'FSSpinn', are initialized as integer "int". they are given with pins that they are connected. These are the pins in the channel tracking sensor, attached in the front of the vehicle. The variables, 'LMF', 'RMF', 'LMB', 'RMB', are initialized as Boolean "bool" as they give instructions for the left and right motor to move forward and backward.

```
int FSpin  = 34; //LINE front
int CSpin  = 40; //LINE central
int LSpin  = 42; //LINE left
int LLSpin = 44; //LINE left left
int RSpin  = 38; //LINE right
int RRSpin = 36; //LINE right right
int FSSpin = 46; //LINE front switch

bool FS;         //LINE front
bool CS;         //LINE central
bool LS;         //LINE left
bool LLS;        //LINE left left
bool RS;         //LINE right
bool RRS;        //LINE right right
bool FSS;        //LINE front switch

int LMFpin = 5;  //left motor forward
int RMFpin = 3;  //right motor forward
int LMBpin = 4;  //left motor backward
int RMBpin = 2;  //right motor backward
int LMpwm = 0;   //pwm left motor
int PMpwm = 0;   //pwm right motor
bool LMF ;       //left motor forward
bool RMF ;       //right motor forward
bool LMB ;       //left motor backward
bool RMB ;       //right motor backward
int vel = 100;   //velocity of the vehicle 0-255
```

Figure 36. Initialization of Necessary Variables [Source: Own]

The pins for the red, green and blue are assigned with their corresponding vales. The velocity of the vehicle should be from 0 to 255, and is here set as 100. The pins for the variables, 'LMFpin', 'RMFpin', 'LMBpin', 'RMBpin', are assigned initially in the code.

Function "printLcd"

The function 'printLcd' in Figure 37, is used to print data on LCD screen. Two input variables 'line' and 'message' are given to this function to operate. The code sets the cursor in the specified location in screen and then prints the blank space. Then, in the given 'line', the given 'message' is printed.

The function 'ReadLine' reads the values of the pins 'FSpin', 'CSpin', 'LSpin', 'LLSpin', 'RSpin', 'RRSpin', 'FSSpinn', where, except, the front 'FSpin' and the front switch 'FSSpinn', all other pins are inverted.

```
void printLcd(int line, String message){
  //prints message at N line
  lcd.setCursor(0, line);
  lcd.print("                    ");
  lcd.setCursor(0, line);
  lcd.print(message);
  }

void ReadLine() {
  FS =   digitalRead(FSpin);  //LINE front
  CS =  !digitalRead(CSpin);  //LINE central
  LS =  !digitalRead(LSpin);  //LINE left
  LLS = !digitalRead(LLSpin); //LINE left left
  RS =  !digitalRead(RSpin);  //LINE right
  RRS = !digitalRead(RRSpin); //LINE right right
  FSS =  digitalRead(FSSpin); //LINE front switch
  }
```

Figure 37. Code of 'printLcd' And 'ReadLine' Functions [Source: Own]

Function "Going"

```
void Going() {
  ReadLine();
  if ((!LS)&&(!CS)&&(!RS)){analogWrite(LMFpin,0);
                           analogWrite(LMBpin,0);
                           analogWrite(RMFpin,0);
                           analogWrite(RMBpin,0);}

  else if (( LS)&&( CS)&&( RS)){analogWrite(LMFpin,vel);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,vel);
                                analogWrite(RMBpin,0);}
  else if ((!LS)&&( CS)&&(!RS)){analogWrite(LMFpin,vel);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,vel);
                                analogWrite(RMBpin,0);}
  else if ((!LS)&&( CS)&&( RS)){analogWrite(LMFpin,vel);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,vel-30);
                                analogWrite(RMBpin,0);}
  else if ((!LS)&&(!CS)&&( RS)){analogWrite(LMFpin,vel-30);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,0);
                                analogWrite(RMBpin,0);}
  else if (( LS)&&( CS)&&(!RS)){analogWrite(LMFpin,vel-30);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,vel);
                                analogWrite(RMBpin,0);}
  else if (( LS)&&(!CS)&&(!RS)){analogWrite(LMFpin,0);
                                analogWrite(LMBpin,0);
                                analogWrite(RMFpin,vel-30);
                                analogWrite(RMBpin,0);}

  }
```

Figure 38. Code of Function 'Going' [Source: Own]

This 'Going' function plays the main role in the movement of vehicle. First, the vehicle reads the values of the pins using 'ReadLine' function as in Figure 37. The Boolean variables 'LS' (left), 'CS' (central), and 'RS' (right), are checked for various possibilities of input as in Figure 38.

By checking those possibilities, the vehicle can adjust the wheels of the motor with its velocity ('vel') in the 'LMFpin', 'LMBpin', 'RMFpin' and 'RMBpin' pins. The left and right motors are adjusted by its velocity and in necessary places, the velocity is reduced with '30'.

Function "Stopping"

The 'Stopping' function, reads the values of the pins using 'ReadLine' function as shown in Figure 39. Then the pins, 'LMFpin', 'RMFpin', 'LMBpin' and 'RMBpin' are written '0', so that the wheels stop rotating, making the vehicle stop.

The vehicle needs to understand the color of the beads with its corresponding numbers. To make this possible, the function 'LedOrder' is created. If the condition "Order[o]==1" is true, then, 'LEDRpin' is set '1' and the other pins "LEDGpin" and "LEDBpin" are set to zero. As like the previous case, for the "Order[o]==2" the pin "LEDGpin" is set '1'. And for "Order[o]==3" pin "LEDBpin" is set '1'. If the "Order[o]==0", the all three pins are set to '1'.

```
void Stopping() {
  ReadLine();
  analogWrite(LMFpin, 0);
  analogWrite(RMFpin, 0);
  analogWrite(LMBpin, 0);
  analogWrite(RMBpin, 0);
  }

void LedOrder() {
  if (Order[o]==1){digitalWrite(LEDRpin,1);
                   digitalWrite(LEDGpin,0);
                   digitalWrite(LEDBpin,0);}
  if (Order[o]==2){digitalWrite(LEDRpin,0);
                   digitalWrite(LEDGpin,1);
                   digitalWrite(LEDBpin,0);}
  if (Order[o]==3){digitalWrite(LEDRpin,0);
                   digitalWrite(LEDGpin,0);
                   digitalWrite(LEDBpin,1);}
  if (Order[o]==0){digitalWrite(LEDRpin,1);
                   digitalWrite(LEDGpin,1);
                   digitalWrite(LEDBpin,1);}
  }
```

Figure 39. Code of 'Stopping' And 'LedOrder' Functions [Source: Own]

Function "OrderFinished"

When the order is over, the "OrderFinished" function is made to run. This first calls "LedOrder" function in Figure 40. Then prints "Vehicle-finished" in LCD screen and checks in 'While' loop for 'p==0'. If it satisfied, condition "(! ((LLS)&&(RRS)))" is checked and if it is true, the function "Going" is called.

Otherwise, the variable 'p' is incremented and checked for 'p==5'. If this satisfies, 'p=0' is set and proceeds.

Now, the vehicle is made to stop listening and transmits 'tx=2' 'rx=1' 'val=3' and 'c', incremented by 1. This is made by opening writing pipe and transmitting these data. Then these transmitted values are printed in the screen. The "Stopping ()" function is made to run to stop the vehicle. Then the vehicle is given a greater delay period.

```
void OrderFinished()
     {
      LedOrder();
      printLcd(0,"Vehicle-finished");
      while(p==0){
                  while (!((LLS)&&(RRS))) {Going();}
                  p=p+1;if (p==5) {p=0;}
                 }
      myRadio.stopListening();      //WIFI NRF TRANSMIT
      dataTransmit.c=dataTransmit.c+1;
      dataTransmit.tx=2;
      dataTransmit.rx=1;
      dataTransmit.val=3;
      myRadio.openWritingPipe(addresses[0]);
      myRadio.write(&dataTransmit, sizeof(dataTransmit));
      lcd.setCursor (3,1);
      lcd.print("                ");
      lcd.setCursor (3,1);
      //lcd.print(dataTransmit.text);
      lcd.print(" c=");
      lcd.print(dataTransmit.c);
      lcd.print(" ");
      lcd.print(dataTransmit.tx);
      lcd.print(">");
      lcd.print(dataTransmit.rx);
      lcd.print(" ");
      lcd.print(dataTransmit.val);
      Stopping();
      delay(10000000);
      }
```

Figure 40. Code of Function 'OrderFinished' [Source: Own]

Configuration of Input & Output pins

```
void setup() {
  pinMode(FSpin,   INPUT);
  pinMode(CSpin,   INPUT);
  pinMode(LSpin,   INPUT);
  pinMode(LLSpin,  INPUT);
  pinMode(RSpin,   INPUT);
  pinMode(RRSpin,  INPUT);
  pinMode(FSSpin,  INPUT);

  pinMode(LMFpin,  OUTPUT);
  pinMode(RMFpin,  OUTPUT);
  pinMode(LMBpin,  OUTPUT);
  pinMode(RMBpin,  OUTPUT);
  pinMode(LEDRpin, OUTPUT);
  pinMode(LEDGpin, OUTPUT);
  pinMode(LEDBpin, OUTPUT);
```

Figure 41. Configuration of Input & Output pins [Source: Own]

As in the Figure 41, the motor pins 'FSpin', 'CSpin', 'LSpin', 'LLSpin', 'RSpin', 'RRSpin', 'FSSpinn', are set as inputs using "pinMode", as they are the sources of information to the vehicle. The pins, 'LMFpin', 'RMFpin', 'LMBpin', 'RMBpin', 'LEDRpin', 'LEDGpin', 'LEDBpin', are given outputs from the program to the components of vehicle to function.

Function "void setup()"

```
Serial.begin(9600);
lcd.init();                  //LCD init or begin
lcd.backlight();             //LCD
printLcd(0,"Vehicle-initializing");
printLcd(1,"TX:");
printLcd(2,"RX:");
printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+
String(Order[1])+String(Order[2])+String(Order[3])+
String(Order[4])+" o"+String(o)+"p"+String(p));
/* // demo*/  while (!((LLS)&&(RRS))) {Going();}
Stopping();
p=0;
myRadio.begin();                  //WIFI NRF
myRadio.setChannel(0x60);
myRadio.setPALevel(RF24_PA_MAX);
//myRadio.setDataRate( RF24_250KBPS );
myRadio.openReadingPipe(1, addresses[0]);
myRadio.startListening();
S=1;
// S=2;LedOrder();  //demo
printLcd(0,"Vehicle-listening");
printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+
String(Order[1])+String(Order[2])+String(Order[3])+
String(Order[4])+" o"+String(o)+"p"+String(p));
```

Figure 42. Things Done in Setup Function [Source: Own]

In setup, after configuring all pins for inputs and outputs, serial monitor is initiated to operate using "Serial.begin(9600)" command as in Figure 42. Then LCD screen is initiated and the backlight is made glow. The 0th, 1st and 2nd lines of the LCD screen is printed with "Vehicle-initializing", "TX:", "RX:", respectively and the 3rd line is printed with the order array sent to the vehicle.

Then, "LLS" and "RRS" are checked for same and the vehicle is made run using "Going()" function and stopped in its first position using "Stopping()" function. Now the Wi-Fi communication is initiated and the vehicle starts its listening line using, "myRadio.openReadingPipe(1, addresses[0])" and starts listening by the function "myRadio.startListening()". Then the state of vehicle is set to 1. Again, the LCD screen is printed with the "Vehicle-listening" and the order array.

Function "void loop()"

```
void loop() {
  if (0==1) { ReadLine();                                          //LINE test
              Serial.print(FSS);Serial.print(FS);Serial.print("-");
              Serial.print(LLS);Serial.print(LS);Serial.print(CS);Serial.print(RS);Serial.println(RRS);
            }
  if (0==1) { digitalWrite(LEDRpin,1);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,0);delay(1000); //LED test
              digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,1);digitalWrite(LEDBpin,0);delay(1000);
              digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,1);delay(1000);
            }
  if (0==1) { Stopping();analogWrite(LMFpin,150);delay(500);     //MOTOR test
              Stopping();analogWrite(RMFpin,150);delay(500);
              Stopping();analogWrite(LMBpin,150);delay(500);
              Stopping();analogWrite(RMBpin,150);delay(500);
            }
  if (0==1) { Serial.println(millis()-lasttime);   //time of last loop
              lasttime=millis();
            }
  if (0==1) { Going();
              if ((LLS)&&(RRS)) {Stopping();delay(1000); }       //simple following test with stops on lines
            }
```

Figure 43. Testing Components of System [Source: Own]

In the loop, the components are tested first for its functioning as shown in Figure 43. Using "ReadLine()" function, the variables are read by the vehicle, and is made to print in serial monitor, using "Serial.print()", for checking the inputs of those variables. The LED lights are checked for glowing one after one by "digitalWrite(LEDRpin,1)" with a delay of 1 second.

Then the motor is tested by assigning values for 'LMFpin', 'RMFpin', 'LMBpin', 'RMBpin', with the value of 150, one after one with a delay of 0.5 second between one another. The function "Stopping()" is also used after testing of each pins. Using "millis()" function, the time for the loop is made and printed in the serial monitor. The vehicle is also tested for its stopping in lines.

In the state S1 of strategy 1 from the Figure 44, the vehicle is made listen inside 'while' loop and the condition for moving to next loop is checked with 'if' loop, and when it satisfies, the received order is written to the memory as like "Order[0]=dataRecieve.Order[0]" for all cells of order and "c". then the order is set '0'. Then this received 'c' 'tx' 'rx' 'val' and all cells of order.

State S1 of Strategy 1 in "void loop ()"

Then in the first line of LCD screen is printed with "Vehicle-order obtained" and after a delay of 3 seconds, "Vehicle-conf sending" is printed. Now the vehicle stops listening and starts transmitting, then the 'tx=2' 'rx=1' 'val=2' and 'c' is

incremented by 1 and transmitted. This is done by opening the 'writing pipe' and" myRadio.write(&dataTransmit, sizeof(dataTransmit))". Then in the second line of LCD screen, all these transmitted values of the variable along with the variable is printed. Then the state is set to 2.

```
if (S==1) { //initial state - listening
  while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));} //WIFI NRF listening
  if (/*lastc1<dataRecieve.c&&*/dataRecieve.rx==2&&dataRecieve.tx==1&&dataRecieve.val==2) {
    lastc1=dataRecieve.c;
    Order[0]=dataRecieve.Order[0];
    Order[1]=dataRecieve.Order[1];
    Order[2]=dataRecieve.Order[2];
    Order[3]=dataRecieve.Order[3];
    Order[4]=dataRecieve.Order[4];                    //write in the order to memory
    o=0;
    printLcd(2,"RX: c="+String(dataRecieve.c)+" "+String(dataRecieve.tx)+">"+String(dataRecieve.rx)+
    " "+String(dataRecieve.val)+" "+
    String(Order[0])+String(Order[1])+String(Order[2])+String(Order[3])+String(Order[4]));
    printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+String(Order[2])+
    String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
    printLcd(0,"Vehicle-order obted");  delay(3000);
    printLcd(0,"Vehicle-conf sending");               //send confirmation to server
    myRadio.stopListening();                          //WIFI NRF TRANSMIT
    dataTransmit.c=dataTransmit.c+1;
    dataTransmit.tx=2;
    dataTransmit.rx=1;
    dataTransmit.val=2;
    myRadio.openWritingPipe(addresses[0]);
    myRadio.write(&dataTransmit, sizeof(dataTransmit));
    printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"+
    String(dataTransmit.rx)+" "+String(dataTransmit.val));
    S=2;
    printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+String(Order[2])+
    String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
    LedOrder();
    printLcd(0,"Vehicle-going");
    } //end of condition
} //end of S==1
```

Figure 44. State S1 of Strategy 1 [Source: Own]

The order obtained is printed on the fourth line of the LCD screen and the "LedOrder()" function is called, so that the system understands with the color of the beads and its corresponding values. After the end of this state, the actions for next state is performed.

State S2 of Strategy 1 in "void loop ()"

In the state S2, mentioned in Figure 45, the vehicle is started to move with "Going ()" function and then it is made to stop using" Stopping ()" by checking "(LLS)&&(RRS)" in 'if' condition. Under this loop, the variable 'p' is incremented and checked it for reaching '5'. If it reaches '5', the variable 'p' is set to '0' again.

This is the limitation of the order array. If the path equals the order, the vehicle stops listening and also prints "Vehicle-asking bead" in the LCD screen.
Then it transmits address of the Vehicle (sender), address of the Gate N (receiver). To do this, the vehicle opens the writing pipe and write the data transmit to the receiver.

```
if (S==2) {Going();                    //state S2 - following the line
   if ((LLS)&&(RRS)) {Stopping();
                      p=p+1;
                      if (p==5) {p=0;}
   if (Path[p]==Order[o]){ printLcd(0,"Vehicle-asking bead");
                           myRadio.stopListening();       //WIFI NRF TRANSMIT
                           dataTransmit.c=dataTransmit.c+1;
                           dataTransmit.tx=2;             //adress of vehicle - sender
                           dataTransmit.rx=Order[o]+2;    //adress of GateN - receiver
                           dataTransmit.val=3;
                           myRadio.openWritingPipe(addresses[0]);
                           myRadio.write(&dataTransmit, sizeof(dataTransmit));
                           printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"+
                           String(dataTransmit.rx)+" "+String(dataTransmit.val));
                           myRadio.startListening();
                           S=3;                           //automaton - arrow to S3 - waiting for the bead
                           printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+
                           String(Order[2])+String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
                           }
   else {printLcd(0,"Vehicle-wrong bead");
         delay(2000);
         S=2;                                             //automaton - arrow to S2 - continue to next stack
         printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+String(Order[2])+
         String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
         printLcd(0,"Vehicle-going");
         }
          }                                               //end of condition
           }                                              //end of S==2
```

Figure 45. State S2 of Strategy 1 [Source: Own]

Then these transmitted data are printed in the LCD screen. Again, the vehicle starts listening and the state is set to '3'. If the path doesn't equal the order, the vehicle prints "" on its LCD screen and the state is again set to '2' and the zeroth lone is printed with "Vehicle-going" and the last line of LCD screen, the order array, path are also printed.

State S3 of Strategy 1 in "void loop ()"

In the state S3, the vehicle is made listening, and three conditions to obtain correct bead mentioned in Figure 46, "dataRecieve.rx==2 && dataRecieve.tx == Order[o]+2 && dataRecieve.val == 5", are checked for true to proceed with the actions of next state. Once the conditions are satisfied, the values with variables 'c' 'tx' 'rx' 'val' and all cells of order are printed on the third line of LCD screen.

Then in zeroth line of LCD screen, “Bead obtained” is printed, and after a delay of 3 seconds, “Vehicle-conf sending” is printed and confirmation is done by stopping the listening of vehicle and transmits the ‘tx=2’ ‘rx= Order[o]+2’ ‘val=2’ and the incremented ‘c’, by opening the ‘writing pipe’ and” myRadio.write(&dataTransmit, sizeof(dataTransmit))”.

```
if (S==3) { //state - listening and waiting for obtain the right bead
    while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));}   //WIFI NRF listening
    if (/*lastc1<dataRecieve.c&&*/dataRecieve.rx==2&&dataRecieve.tx==Order[o]+2&&dataRecieve.val==5) {
      lastc1=dataRecieve.c;
      printLcd(2,"RX: c="+String(dataRecieve.c)+" "+String(dataRecieve.tx)+">"+String(dataRecieve.rx)+" "+
        String(dataRecieve.val)+" "+String(Order[0])+String(Order[1])+
        String(Order[2])+String(Order[3])+String(Order[4]));
      printLcd(0,"Bead obtained");  delay(3000);
      printLcd(0,"Vehicle-conf sending");               //action to S2 - send confirmation to GateN
      myRadio.stopListening();                          //WIFI NRF TRANSMIT
      dataTransmit.c=dataTransmit.c+1;
      dataTransmit.tx=2;
      dataTransmit.rx=Order[o]+2;
      dataTransmit.val=2;
      myRadio.openWritingPipe(addresses[0]);
      myRadio.write(&dataTransmit, sizeof(dataTransmit));
      printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"+String(dataTransmit.rx)+
        " "+String(dataTransmit.val));                  // end of actions on arrow to S2
      S=2;                                              //arrow to S2
      Order[o]=0;
      printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+String(Order[2])+
        String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
      o=o+1;
      LedOrder();
      printLcd(0,"Vehicle-going");

      if ((Order[0]==0)&&(Order[1]==0)&&(Order[2]==0)&&(Order[3]==0)&&(Order[4]==0)) {OrderFinished();}
      } //end of condition
  } //end of S==3
} //end of loop
```

Figure 46. State S3 of Strategy 1 [Source: Own]

Then the state is set to S2 and the “Order[o]” is set to zero. Now the order is displayed in the LCD screen and the position of necessary bead in order array is incremented with 1. The function, “LedOrder()” is called to notify the relation between the color and its respective numbers in values to the vehicle. Then the five conditions “Order[0]==0) && (Order[1]==0) && (Order[2]==0) && (Order[3]==0) && (Order[4]==0” are checked using ‘if’ condition. If it satisfies the condition, the “OrderFinished()” is called to function.

This way, the vehicle is programmed to function as per the strategy 1. All the other functions are as per mentioned in the basic structure of programming the vehicle. This programming is optimized step by step using further strategies.

5.4.2. Strategy 2

The second strategy improves in approach of checking the order with the stacks. In this strategy, the vehicle is made to check the next cell of the order array, to receive the beads of same stack if ordered. First the vehicle checks the first cell of the order array with each stack and finds the match. Then, the vehicle checks the cells of the order array one after one. When the necessary bead mismatches with the stack, the vehicle goes to the next stack. In each stack, the vehicle checks for the beads in order array of consecutive cells for same color beads in that stack.
This enhances the situation, that, the vehicle checks for the order of many consecutive cells in one stack, if can. So, the vehicle movement is minimized to some extent from using strategy 1.

5.4.2.1. Work Flow Description

At first, the vehicle moves to first position and asks for the bead in the first cell of the Order array. It checks the first cell with the stacks and find the matching stack and receives the bead. Then, in the same position, the consecutive cells are examined for matches and if matches, the beads are obtained.
When they mismatch, the vehicle moves to next position and continue with the same process. This continues till the vehicle receives all the ordered beads are obtained.

5.4.2.2. Hierarchy of Commands

Based on the description of the work flow process of strategy 2, the commands of the flow process are developed. This instructs the vehicle with the list of commands to be followed, on the process of receiving the beads. The list of commands based on the strategy 2 is as follows.

1) Reset.
2) Go to initial position.
3) Use Position sensor, to find path.
4) If the position reached, Receive color of the gate.
5) Compare Order [i] and received color of the gate.
    a) If same, Obtain bead, i<=i+1, jump to point 5.
    b) If they are dissimilar, Go to next position.

c) If Order [i], i == 6, jump to point 7.

d) If Order [i] == 0, Count <= 5, jump to point 7.

6) Jump to point 3.

7) Count the number of beads obtained.

a) If Count == 5, jump to point 8.

b) If Count < 5, Go to next position, jump to point 3.

8) Go to initial position.

5.4.2.3. Work Diagram with Strategy 2

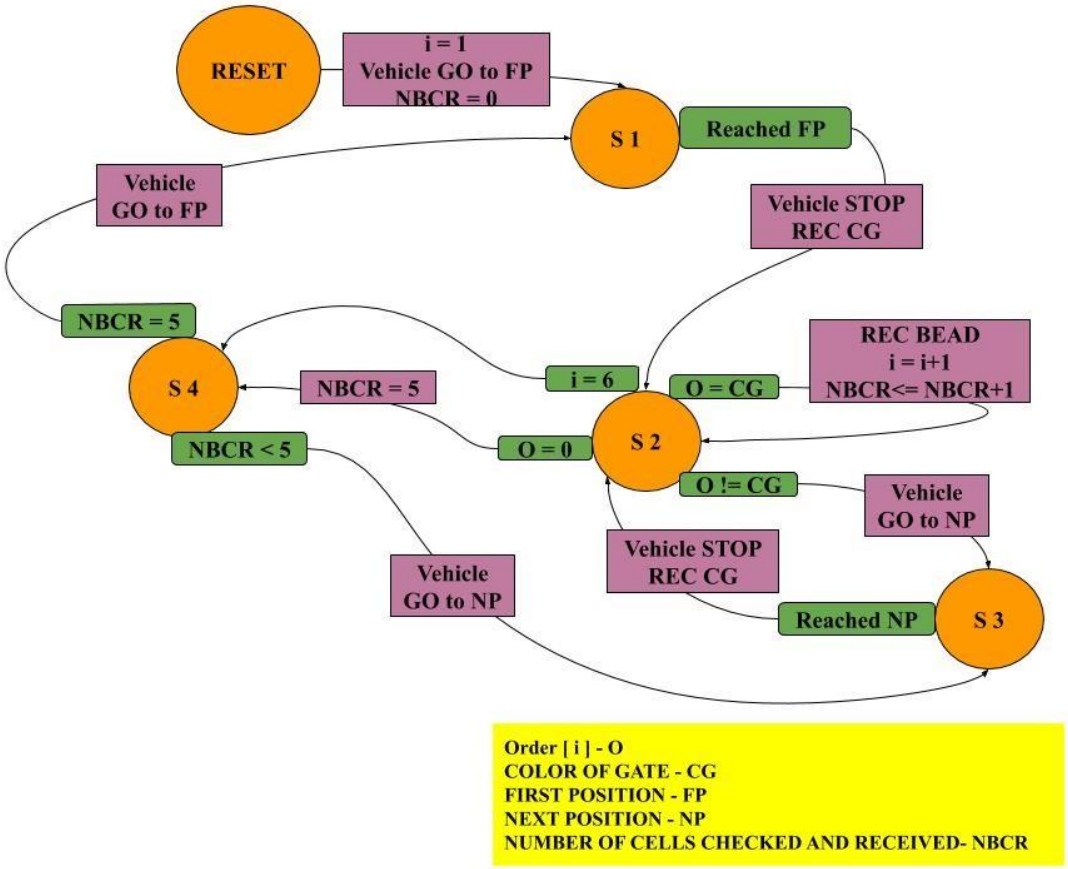


Figure 47. Automaton of vehicle on strategy 2 [Source: Own]

With the reference of the list of commands created and mentioned above, the automaton graph is created. This automaton in Figure 47, has split states, which checks for the similarity of consecutive cells of order array in each state. When it mismatches, the state jumps to S2. In this state, the vehicle is made to decide to move to next position or to check with next cell also. If the vehicle needs to checks with further cells, the state S3 facilitates it. At any state, when the variable 'i' of the order reaches the number 6, the vehicle checks with the number of beads

received by the vehicle. If the obtained beads reached five, it makes the loop close and the vehicle is instructed to move to initial position.

Then the loop is made infinite as it circulates in S1, S2 and S4. Thus, the vehicle stands in the first position and waits for reset and to function from the beginning.

5.4.2.4. Programming of Vehicle with Strategy 2

The Figure 47 shows the improvement in receiving the ordered beads. All the functions used for programming the vehicle with strategy 1 is same. The states, which are designed in a way to implement the strategy 2 in the movement of the vehicle, are modified in such a way that, it serves the purpose. The programmed states S1 and S2 are same as used in the program of strategy 1.

The change of code in the state S3 alone mentioned in APPENDIX D. In the APPENDIX C, this state S3 is alone replaced by APPENDIX D to get the full code of vehicle using strategy 2.

State S3 of Strategy 2 in "void loop ()"

The state S3 to respond in a way to implement strategy 2 is given in Figure 48. When getting the bead, in the state S3, the order is incremented and checked for the next cell to match with the same gate. If they are dissimilar, then the vehicle is made to go to the next gate and perform these actions repeatedly. In state S3, the vehicle listens and wait for the bead to obtain and receives beads from the gates. After receiving the beads, that particular cell of the order array, is set to '0' by the command, "Order[o] = 0;" after this, the "o" of the Order[o], is incremented with 1 and an "if else" condition is performed.

The condition "Path[p]==Order[o]" is checked here, and if it is true, the state is again set to S3, to obtain bead and again check for next cell. If it becomes false in any situation, the state is set to S2, i.e., the vehicle moves to next position and do the same. In state S3, the order is always checked for '0'. When the order array becomes '0', the function, "OrderFinished()" is called to stop the program.

Thus, by making these changes with the strategy 1 based program, the vehicle follows the strategy 2 process. All the other functions, used for programming strategy 1 are included as such in the strategy 2 also.

```
if (S==3) { //state - listening and waiting for obtain the right bead
    while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));} //WIFI NRF listening
    if (/*lastc1<dataRecieve.c&&*/dataRecieve.rx==2&&dataRecieve.tx==Order[o]+2&&dataRecieve.val==5)
      {
      lastc1=dataRecieve.c;
      printLcd(2,"RX: c="+String(dataRecieve.c)+" "+String(dataRecieve.tx)+">"+String(dataRecieve.rx)+
      " "+String(dataRecieve.val)+" "+String(Order[0])+String(Order[1])+String(Order[2])+
      String(Order[3])+String(Order[4]));
      printLcd(0,"Bead obtained");  delay(3000);
      printLcd(0,"Vehicle-conf sending");
      myRadio.stopListening();                                  //WIFI NRF TRANSMIT
      dataTransmit.c=dataTransmit.c+1;
      dataTransmit.tx=2;
      dataTransmit.rx=Order[o]+2;
      dataTransmit.val=2;
      myRadio.openWritingPipe(addresses[0]);
      myRadio.write(&dataTransmit, sizeof(dataTransmit));
      printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"
      +String(dataTransmit.rx)+" "+String(dataTransmit.val));
      Order[o]=0;
      o=o+1;
      // end of actions on arrow to S2
      if (Path[p]==Order[o]){ S=3;
                              }
      else {S=2;                                                //arrow to S2
      printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+
      String(Order[2])+String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
      LedOrder();
      printLcd(0,"Vehicle-going");
      }
      if ((Order[0]==0)&&(Order[1]==0)&&(Order[2]==0)&&(Order[3]==0)&&(Order[4]==0)) {OrderFinished();}
      } //end of condition
  } //end of S==3
```

Figure 48. State S3 of Strategy 2 [Source: Own]

5.4.3. Strategy 3

This strategy has far improvement than past two strategies, in optimizing the movement of the vehicle. The vehicle, when reaching the first stack, checks all the cells of the order array and obtain as many beads of that color needed, as in count in the array.

5.4.3.1. Work Flow Description

The Vehicle is made to move to the first position. Vehicle verifies the received bead color of the gate with all the cells in the order array. When the color of the bead in gate matches with that in the cell of the order array, the bead is collected and again checks for next cell of the array.

This way, all cells are checked for the color of bead in the gate. If all cells are checked, the number of beads collected is cross checked. If the beads received is lesser than five, the vehicle moves to next position. The bead color of this gate is checked with all the other cells of order array, and the necessary beads are received. This continues, till the vehicle collects all the necessary beads from the

gates. If the order has zero in it, it means the finish of necessary beads. So, in that situation, the vehicle is claimed to the first position by registering the obtained bead to the value '5'.

5.4.3.2. Hierarchy of Commands

The description gives clear idea of the flow of conditions to be checked on all the states. The commands for creating automatons are made in a hierarchical order of happening. This helps to create automaton graph and to cross check with the conditions that are to be checked in every states. The list of commands following strategy 3 are as follows.

1) Reset.
2) Go to initial position.
3) Use Position sensor, to find path.
4) If the position reached, Receive color of the gate.
5) Compare Order [i] and received color of the gate.
    a) If same, Obtain bead, i<=i+1.
        i) If Order [i] < 6, i<=i+1, jump to point 5.
    b) If they are dissimilar, i<=i+1.
        i) If Order [i], i < 6, i<=i+1, jump to point 5.
    c) If Order [i] == 0, Count <= 5, jump to point 6.
    d) If Order [i] == 6, jump to point 6.
6) Count the number of beads obtained.
    a) If Count == 5, jump to point 7.
    b) If Count < 5, Go to next position, jump to point 3.
7) Go to initial position.

5.4.3.3. Work Diagram with Strategy 3

As per the listed commands, automaton graphs are created to split the checking of conditions in various states. Here, from Figure 49, the state S2 checks with the cells of the order array for zero or the same color of the gate or different from the color of the gate. If it has same color, the bead is received from the gate and the next cell is taken into account for comparison. If it is dissimilar, the next cell is taken into account for comparing. If it shows zero, it is considered as all the beads are collected and the vehicle is made to go to its initial position.

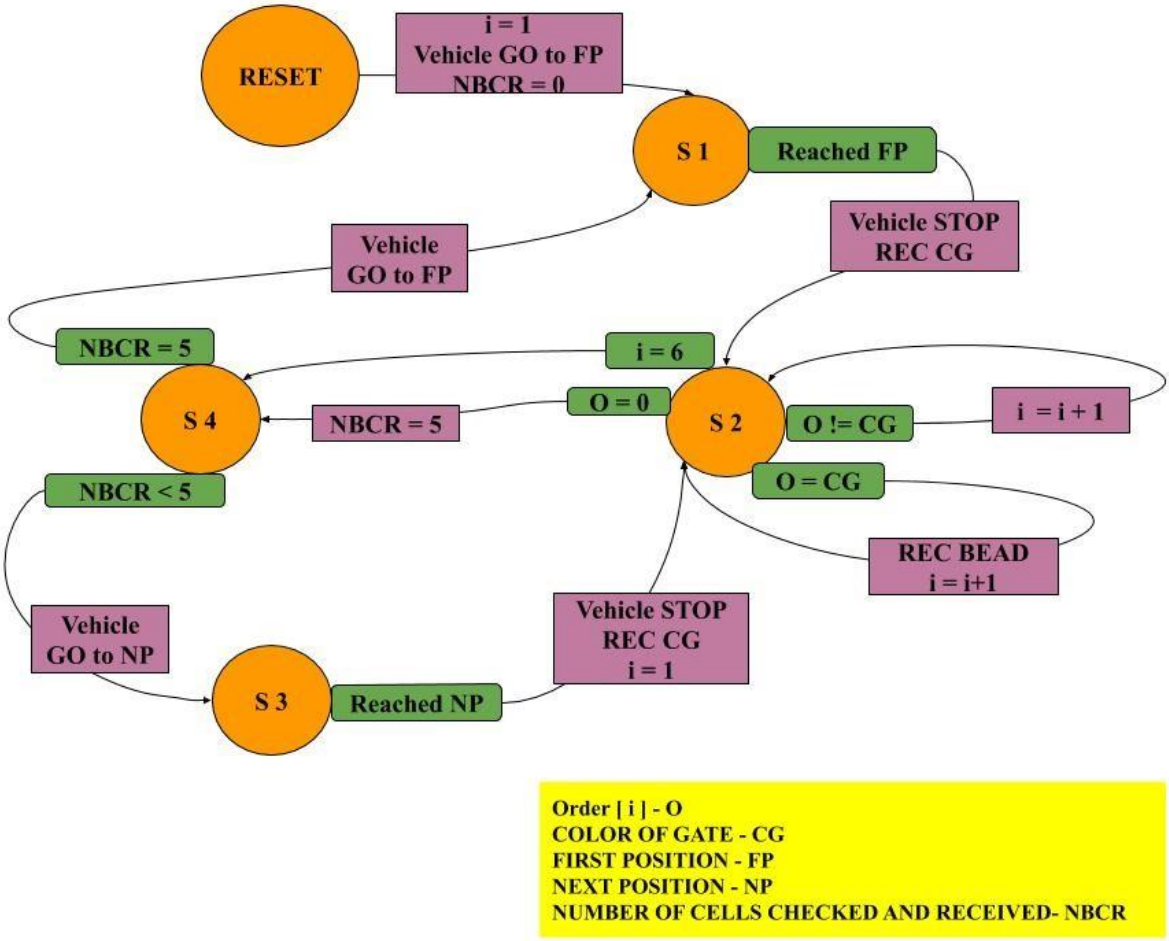


Figure 49. Automaton of vehicle on strategy 3 [Source: Own]

The state S2 also checks for the checking of cells not to exceed the value '6', as the order array limit is '5'. If it reaches '6', the state S4 is checked for the number of beads received. If the received beads are lesser than '5', then the vehicle is made to go to next position and made the cells to check as '1'. Then the process of state S2 continues till the number of beads received is '5', or the order becomes zero.

5.4.3.4. Programming of Vehicle with Strategy 3

The Figure 49 shows the improvement of the strategy 3 from strategy 2 in receiving the beads. All the functions used in programming the vehicle with strategy 1 is same for strategy 3 also. The programmed states S1 and S3 are same as used in the program of strategy 2. The state S2 and S3 are modified and state S4 is created.

The change of code in the states S2, S3 and the new state S4 are mentioned in APPENDIX E. In the APPENDIX C, the states S2 and S3 are replaced and state S4 is added with APPENDIX E to get the full code of vehicle using strategy 3.

State S2 of Strategy 3 in "void loop ()"

To transfer the program to function based on strategy 3, changes have been made on the program designed to function on strategy 1. There is a small change made in the state S2, which creates a bigger difference in the program.

The condition, "Path[p] == Order[o]" is checked in 'if' loop, and the actions are performed accordingly. When this condition fails, using 'else' statement, the state is set to S4, which is created to implement the strategy 3. The state S4 is given in Figure 52. This is made clear in the below mentioned Figure 50. This change

```
if (S==2) {Going();                    //state S2 - following the line
           if ((LLS)&&(RRS)) {Stopping();
                              p=p+1;
                              if (p==5) {p=0;}
                              if (Path[p]==Order[o]){ printLcd(0,"Vehicle-asking bead");
                                 myRadio.stopListening();              //WIFI NRF TRANSMIT
                                 dataTransmit.c=dataTransmit.c+1;
                                 dataTransmit.tx=2;           //adress of vehicle - sender
                                 dataTransmit.rx=Order[o]+2;  //adress of GateN - receiver
                                 dataTransmit.val=3;
                                 myRadio.openWritingPipe(addresses[0]);
                                 myRadio.write(&dataTransmit, sizeof(dataTransmit));
                                 printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+
                                 ">"+String(dataTransmit.rx)+" "+String(dataTransmit.val));
                                 myRadio.startListening();
                                 S=3;
                                 printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+
                                 String(Order[2])+String(Order[3])+String(Order[4])+
                                 " o"+String(o)+"p"+String(p));
                                 }
                            else {printLcd(0,"Vehicle-wrong bead");
                                  delay(2000);
                                  S=4;                          //automaton - arrow to S4
                                  printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+
                                  String(Order[2])+String(Order[3])+String(Order[4])+
                                  " o"+String(o)+"p"+String(p));
                                   }
                             } //end of condition
          } //end of S==2
```

Figure 50. State S2 of Strategy 3 [Source: Own]

State S3 of Strategy 3 in "void loop ()"

The Figure 51 shows the changes made from the state S3 of strategy 2 to state S3 of strategy 3. In this state, the beads are received from the stacks and then that particular cell of the order is set to '0'. Then the state of the vehicle is set to S4. So that the state S4 checks with all the cells of the order in single position. Here, the next cell of the array is not introduced in any of the states S1, S2 or S3. It is done in the state S4 as shown in Figure 52.

```
if (S==3) {               //state - listening and waiting for obtain the right bead
    while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));}
    if (/*lastc1<dataRecieve.c&&*/dataRecieve.rx==2&&dataRecieve.tx==Order[o]+
    2&&dataRecieve.val==5) {
      lastc1=dataRecieve.c;
      printLcd(2,"RX: c="+String(dataRecieve.c)+" "+String(dataRecieve.tx)+">"+
      String(dataRecieve.rx)+" "+String(dataRecieve.val)+" "+String(Order[0])+
      String(Order[1])+String(Order[2])+String(Order[3])+String(Order[4]));
      printLcd(0,"Bead obtained");  delay(3000);
      printLcd(0,"Vehicle-conf sending");
      myRadio.stopListening();                                  //WIFI NRF TRANSMIT
      dataTransmit.c=dataTransmit.c+1;
      dataTransmit.tx=2;
      dataTransmit.rx=Order[o]+2;
      dataTransmit.val=2;
      myRadio.openWritingPipe(addresses[0]);
      myRadio.write(&dataTransmit, sizeof(dataTransmit));
      printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"
      +String(dataTransmit.rx)+" "+String(dataTransmit.val));
      Order[o]=0;
      S=4;                                                      //arrow to S4
      if ((Order[0]==0)&&(Order[1]==0)&&(Order[2]==0)&&(Order[3]==0)&&(Order[4]==0))
      {OrderFinished();}
      } //end of condition
    } //end of S==3
```

Figure 51. State S3 of Strategy 3 [Source: Own]

State S4 of Strategy 3 in "void loop ()"

The Figure 49 shows the state S4 of the vehicle to perform the major action of strategy 3. In the state S4, the cell of the order array is incremented with '1'. To restrict the checking of non-existing cells, the value of 'o' is checked for reaching '5'. When the cell reaches '5', the cell of order array is set to '0' and the state is given as S2 and so the vehicle moves to next position and its loop continues.

```
if (S==4){
  o=o+1;
  if(o==5){ o=0;
            S=2;
            printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+
            String(Order[1])+String(Order[2])+String(Order[3])+
            String(Order[4])+" o"+String(o)+"p"+String(p));
            LedOrder();
            printLcd(0,"Vehicle-going");
  }
if (Path[p]==Order[o]){
      S=3;
    }
else{ S=4;  }
}
```

Figure 52. State S4 of Strategy 3 [Source: Own]

The condition "Path[p] == Order[o]" is checked here, and if it is true, the state is set to S3. If this is not satisfied, again the state is registered as S4 to check with the other cells of the array. This satisfies the vehicle on functioning based on strategy 3.

6. SMART FACTORY MODEL WITH MORE VEHICLES

This smart factory model functions with a single vehicle on the line. It is to be improved in such a way that the system to function with many vehicles on it. To function with many vehicles, the server and the vehicle are to be modified in such a wat that they respond correctly to the system.

6.1. Modifications in Vehicle

In construction of the vehicle, the vehicle is provided with two more components to avoid accidents between vehicles. The “Infrared Sensor” and “Mechanical Front Switch” are fixed in front of the vehicle to identify another vehicle or any obstacle before it. If the vehicle finds any obstacles using these components, the vehicle is made to stop, and the accident is avoided.

The vehicle is also initiated with variables like, “bool Free”, to register that the vehicle is free from work, and is available to take order, and “int Battery” to check with the battery power level, to be assigned with work, by the server.

6.1.1. Obstacle Protection

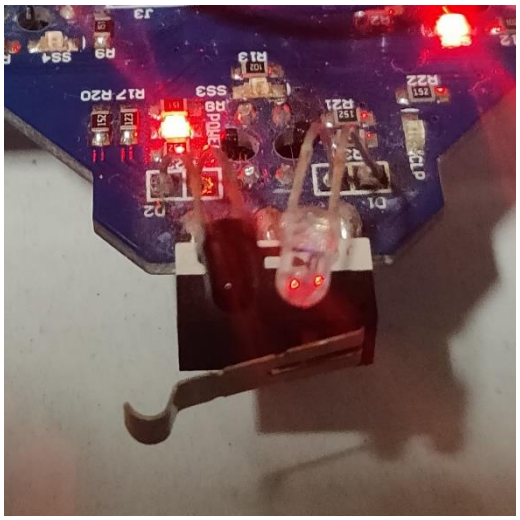


Figure 53. IR Sensor and Mechanical Front Switch [Source: Own]

These two components are checked for its response, when the vehicle is made to move. When these two components respond as ‘0’, the vehicle is free to move and the servo motor pins can be powered to move the vehicle. If any one of these components responds ‘1’, the vehicle is made to “stop” all of the sudden to avoid accident with any obstacle. These modifications are inserted to the system, by

installing these components to the vehicle and writing the list of commands to be followed by the vehicle, when it is made free to move.

6.1.2. Work Flow Description for Accident Free Movement

When the vehicle is ordered to move, it checks with the two component's return value to take decision on moving. The variable, "CAN_GO" decides to move or stop the vehicle. Only, at times, the variable, "CAN_GO" is '1', the vehicle is allowed to move. At any instant of time, the variable "CAN_GO" is '0', the vehicle is stopped.

Infrared Sensor	Mechanical Front Switch	CAN_GO
0	0	1
0	1	0
1	0	0
1	1	0

Figure 54. Decision of "CAN_GO" Variable [Source: Own]

The components "Mechanical Front Switch" and "IR Sensor" are checked for its possibilities and the variable "CAN_GO" is set. The Figure 53, gives clear idea about its response to set the variable "CAN_GO" with '0' or '1'.

6.1.3. Hierarchy of Commands

Based on the description and Figure 53, clear idea of the conditions to be followed are understood and the commands followed are written.

1) Reset.
2) Check return value of MFS
   a) If MFS == 1, CAN_GO <= 0, jump to point 3.
   b) If MFS == 0, jump to point 3.
3) Check return value of IRS
   a) If IRS == 1, CAN_GO <= 0, jump to point 3.
   b) If IRS == 0, jump to point 4.
4) Check return value of IRS

a) If MFS == 1, CAN_GO <= 0, jump to point 3.

b) If MFS == 0, CAN_GO <= 1, jump to point 3.

6.1.4. Work Diagram for Checking Sensors

The Figure 54, gives the automaton for the vehicle to decide the variable "CAN_GO". The automaton is designed in such a way that, only when the two components responds with '0', the variable "CAN_GO" is set '1'. This is a closed automaton and runs when the vehicle runs the function "Going()". This assists the vehicle to move eradicating accidents.

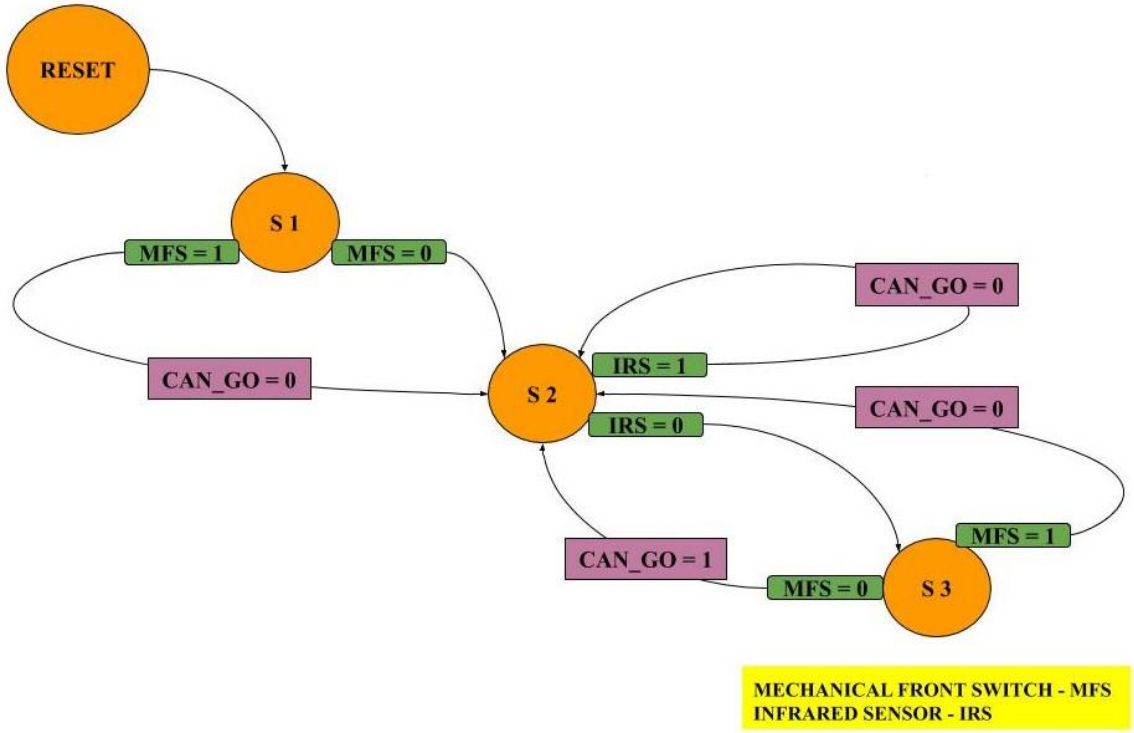


Figure 55. Automaton for obstacle protection of vehicle [Source: Own]

In the first state S1, the "Mechanical Front Switch" is checked for '0' or '1'. If the switch returns '1', the variable "CAN_GO" is set to '0' and "IR Sensor" is checked for '0' or '1'. If it is '1' again, the variable "CAN_GO" is set to '0' and "IR Sensor" is again checked for '0' or '1'. In this state S2, the "IR Sensor" is checked for its response. If "IR Sensor" returns '0', the "Mechanical Front Switch" is checked and only if it also returns '0', the variable "CAN_GO" is set to '1'.

The state S3, checks for "Mechanical Front Switch" and if it returns '1', the variable "CAN_GO" is set to '0' and then "IR Sensor" is checked for its response in state S2.

6.2. Modifications in Server

When the system is installed with many vehicles, the server has to respond with the system. The server receives the order of necessary beads from Blynk application. It should assign a vehicle to carry the order. To assign the order, the vehicle should be selected based on some conditions. By implementing and checking necessary conditions, the vehicle to which the order to be given is selected. This model assigns two conditions to check with the availability of the vehicle and gives the order.

6.2.1. Work Flow Description of Assigning Orders

The server receives order from the user and needs to assign this work of collecting the beads to any of the vehicles. To assign the work, the server needs to have a work flow of checking necessary conditions and assign the order with the vehicle. Here, the criteria of assigning orders to vehicle are,

- The vehicle should be free from any orders.
- The battery power is higher than 20%.

The vehicle, possessing the above-mentioned values are checked in the order from vehicle 1. In this way, orders are assigned to the vehicles one after another. These criteria of choosing the vehicle can be modified as per the needs of the factory.

6.2.2 Hierarchy of Commands

Based on the criteria mentioned in description, the commands to implement this process on server are created. An array variable 'Vehicle Array Address', "VAA[p]" is created to store the address of vehicle. When the conditions are satisfied, the Order[i] is sent to the vehicle of the stored address in VAA[p].

1) From old state, server is listening, jump to point 3.
2) If order received, p=0, Send address to VAA[p], Server Listening.
3) If order received, jump to point 5.
4) Check the value of "VEHICLE FREE"
    a) If VF == 1, jump to point 6.
    b) If VF == 0, p=p+1, jump to point 7.
5) Check the value of "BATTERY"
    a) If BT > 20%, Send Order[i] to VAA[p], Server Listening, jump to point 8.

b) If BT < 20%, p=p+1, jump to point 7.

6) Check value of "p"

a) If p == 3, p=0, jump to point 8.

b) If p < 3, jump to point 8.

7) Check value of "p"

8) If p < 3, p=0, Send address to VAA[p], Server Listening, jump to point 4.

9) Continue with Next Old State.

6.2.3. Work Diagram for Distributing Orders

The Figure 56, gives the part of automaton need to include in the work flow of server to modify the server, to assign the Order[i] to the vehicle, which satisfies the condition. It is made possible by the array variable VAA[p]. This variable stores the address of the vehicles. When the conditions are satisfied with the vehicles, the vehicle with its address stored in VAA[p] is assigned with the order.

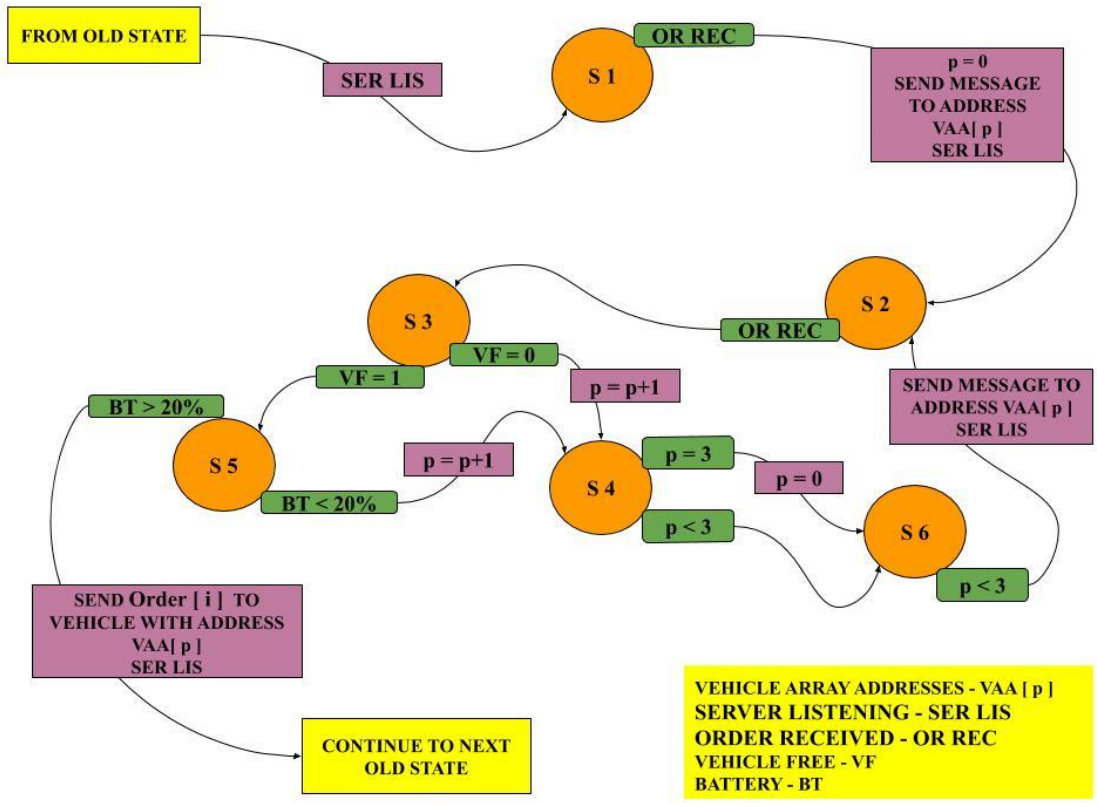


Figure 56. Part of Server Automaton to Identify Free Vehicles [Source: Own]

Based on the addresses of the vehicle, it is identified for free and the battery power, and the server assigns that vehicle with that address, with the order[i]. Thus, based on required conditions, the Order[i] is distributed by the server.

7. CONCLUSION

The primary aim of thesis is to create an improved control system of a smart factory model with simplified codes for all components of the system, using various modules and sensors in connection with the Arduino Mega2560 as the microcontroller of all components. This was a prototype of smart factory unit to check its response of control systems.

To build this system, a server was created with two transceivers on a PCB, and programmed based on the structure of automaton graphs. The server was made to listen the order of beads from the user, through mobile application and transmits to the vehicle with confirmation of vehicle on receiving the order.

The stacks are reprogrammed in such a way that it split the conditions check by drawing automaton for the stacks.

A vehicle was made in such a way, that it communicates with the components of system, using Wi-Fi module to collect the ordered beads. The vehicle was attached with necessary sensors for its accurate movement.

This control system was an improved model as it uses automatons to split checking the conditions, and eradicates checking of many conditions in main loop. This makes the system to respond faster. This control system was also made testing three possible strategies on optimizing the vehicle movement. This control system has many applications in factories for its different purposes. It can be implemented in smart industries to transport objects by the vehicle, without human interference.

Scope of the Thesis:

The control system of the smart factory can be aimed to have new functions as per the specific requirements of the factory unit.

- As educational model, it can be used for implementing and testing of new ideas to improve the system.
- As, many wireless networks available, we can use any of those, in this control system and use it in diverse applications that factories need.
- It can be improvised further to transport components all over the factory and even people around the factory.

REFERENCES

[1] Hozdić, Elvis. (2015). Smart factory for industry 4.0: A review. International Journal of Modern Manufacturing Technologies. 7. 28-35.

[2] Wang, Shiyong & Wan, Jiafu & Li, Di & Zhang, Chunhua. (2016). Implementing Smart Factory of Industrie 4.0: An Outlook. International Journal of Distributed Sensor Networks. 2016. 1-10. 10.1155/2016/3159805.

[3] The Industrial Revolution: From Industry 1.0 to Industry 4.0. (2019, October 10). Retrieved from https://www.seekmomentum.com/blog/manufacturing/the-evolution-of-industry-from-1-to-4

[4] T. G. Pareek, Raksha Padaki, Anuradha Iyer, and Priya G, "App Based Device Controlling System," *www.ijarcs.info*, 2017. [Online]. Available: http://www.ijarcs.info/index.php/Ijarcs/issue/view/64. [Accessed: 13-Feb-2020]

[5] What is IoT and Should Your Company Be on Board? (n.d.). Retrieved from https://www.kaaproject.org/what-is-iot-for-business

[6] Mir, Arsheen & Rajguru, Swarnalatha. (2018). Implementation of an industrial automation system model using an Arduino. Journal of Engineering Science and Technology. 13. 4131-4144.

[7] {{metadataController.pageTitle}}. (n.d.). Retrieved from https://subscription.packtpub.com/book/business/9781788832687/1/ch01lvl1sec15/industrial-iot-deployment-architecture

[8] Melanson, T. (2018, November 7). Industry 4.0 and Mobile Robots - Connecting Islands of Automation. Retrieved from https://aethon.com/mobile-robots-and-industry4-0/

[9] Nikam. (2019, September 18). Cyber-Physical System Market: Will China be Able to Surpass Western Europe in Terms of Growth? Retrieved from https://communalnews.com/cyber-physical-system-market-will-china-be-able-to-surpass-western-europe-in-terms-of-growth/

[10] Jansa, M. (2019, April 29). Výukový model řízení výroby s principy Průmyslu 4.0. Retrieved from https://dspace.tul.cz/handle/15240/153958

[11] Barrett, Steven F. "Arduino microcontroller processing." Synthesis Lectures on Digital Circuits and Systems 8.4 (2013): 1-513. Kumar RH, Roopa AU, Sathiya DP. Arduino ATMEGA-328 microcontroller. International journal of innovative research in electrical, electronics, instrumentation and control engineering. 2015;3(4):27-9.

[12] *, N. (2019, August 1). Arduino Mega 2560 Board: Specifications, and Pin Configuration. Retrieved from https://www.elprocus.com/arduino-mega-2560-board/

APPENDICES INDEX

Appendix A – Program of Server

```
#define BLYNK_PRINT Serial                                  // blynk
#include <ESP8266_Lib.h>                                    // blynk
#include <BlynkSimpleShieldEsp8266.h>                       // blynk
#include <SPI.h>                                            // WIFI
#include "RF24.h"                                           // WIFI
#include <Wire.h>                                           // LCD
#include <LiquidCrystal_I2C.h>                              // LCD
#define EspSerial Serial2                                   // blynk
#define ESP8266_BAUD 115200                                 // blynk
LiquidCrystal_I2C lcd(0x27, 20, 4);                         // LCD 20x4
ESP8266 wifi(&EspSerial);                                   // blynk
BlynkTimer timer;
WidgetLCD lcdPhone(V15);
RF24 myRadio(49, 53);                                       // WIFI
byte addresses[][6] = {"0"};                                //WIFI
struct package {
  //char text[20]="00000";
  int Order[5]={0,0,0,0,0};
  int c=1;
  int tx=0;
  int rx=0;
  int val=0;  };
typedef struct package Package;                             // WIFI
Package dataRecieve;
Package dataTransmit;
const char WIFI_AUTH[] = "e25c2bec146c4ffb842df4216e8ea372";       //WIFI Settings
const char WIFI_SSID[] = "RadekV";
const char WIFI_PASSWORD[] = "Hotspot76";
int s=0;                                                    //step variable
int Sb=1;                                                   //state of buttons automaton
int Ss=1;                                                   //state of main server
automaton
int lastc2=0;                                               //variables to control blynk
buttons
int bR=0;
int bG=0;
int bB=0;
int bClear=0;
int bFinish=0;
int o=0;                                                    // position of bead in order
int Order[5]={0,0,0,0,0};                                   // order
int p=0;
                                                            // position on the path
void setup() {
  pinMode(12, INPUT_PULLUP);                                // button for stepping
  Serial.begin(9600);                                       // LCD settings
  lcd.init();                                               // lcd.begin();
  lcd.backlight();                                  // begin or init depends on LCD library
```

```
  lcd.clear();
  printLcd(0,"Server-initializing");
  printLcd(1,"TX:");
  printLcd(2,"RX:");
  EspSerial.begin(ESP8266_BAUD);                    // Wifi settings
  delay(1000);
  Blynk.begin(WIFI_AUTH, wifi, WIFI_SSID, WIFI_PASSWORD);
  //timer.setInterval(1000L, sendSensor);   //timer of blynk
  myRadio.begin();                                  // Wifi settings and listening
  myRadio.setChannel(0x60);                         //WIFI NRF
  myRadio.setPALevel(RF24_PA_MAX);
  myRadio.setDataRate( RF24_250KBPS );
  printLcd(0,"Server-order beads");
  printLcd3();
  sendMessageToPhone3();
}

void loop() {
  Blynk.run();                                  //timer.run();   //timer of blynk
  if (Ss==1){                                   //Ss1-creating order on the phone
        SMofSb();                               //SerialMonitor: debugging blynk buttons and
Sb
        SbAutomaton();       }
  }

int readCodeFromWifi() {                    // listens wifi and puts recieved code to
dataRecieve
  while (myRadio.available()) {
    myRadio.read(&dataRecieve, sizeof(dataRecieve));      }
}

void sendMessageToPhone(String message) {               // sends message to phone
  lcdPhone.clear();
  lcdPhone.print(0, 0, message);
}

void sendMessageToPhone3() {                            // sends message line3 to phone
  lcdPhone.clear();
  lcdPhone.print(0, 0, "Order:"+String(Order[0])+String(Order[1])+String(Order[2])+
  String(Order[3])+String(Order[4]));
}

void printLcd(int line, String message) {               //prints message at N line
  lcd.setCursor(0, line);
  lcd.print("                ");
  lcd.setCursor(0, line);
  lcd.print(message);
}

void printLcd3() {
  printLcd(3,"Ss"+String(Ss)+" Order:"+String(Order[0])+String(Order[1])+
  String(Order[2])+String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
```

```
}

BLYNK_WRITE(V0) {       bR = param.asInt();          }       // sync with Blynk
BLYNK_WRITE(V1) {       bG = param.asInt();          }
BLYNK_WRITE(V2) {       bB = param.asInt();          }
BLYNK_WRITE(V3) {       bClear = param.asInt();      }
BLYNK_WRITE(V4) {       bFinish  = param.asInt();    }

void SMofSb(){                      // SerialMonitor: debugging blynk buttons and Sb
  delay(50);
  Serial.print("Order:");
  Serial.print(Order[0]);
  Serial.print(Order[1]);
  Serial.print(Order[2]);
  Serial.print(Order[3]);
  Serial.print(Order[4]);
  Serial.print("-Buttons:");
  Serial.print(bR);
  Serial.print(bG);
  Serial.print(bB);
  Serial.print(bClear);
  Serial.print(bFinish);
  Serial.print("-State:");
  Serial.println(Sb);   }

void SbAutomaton(){
  if (Sb==1) {if (bR==1) {Sb=2;
                    Order[o]=1;
                    if (o<4) o=o+1;
                    printLcd3();
                    sendMessageToPhone3();
                      }
             if (bG==1) {Sb=3;
                    Order[o]=2;
                    if (o<4) o=o+1;
                    printLcd3();
                    sendMessageToPhone3();
                      }
             if (bB==1) {Sb=4;
                    Order[o]=3;
                    if (o<4) o=o+1;
                    printLcd3();
                    sendMessageToPhone3();
                      }
             if (bClear==1) {Sb=5;
                       Order[0]=0;
                       Order[1]=0;
                       Order[2]=0;
                       Order[3]=0;
                       Order[4]=0;
                       o=0;
                       printLcd3();
```

```
                    sendMessageToPhone3();          }
          if (bFinish==1) {Sb=6;        }
          }
if (Sb==2) {if (bR==0) {Sb=1;}        }
if (Sb==3) {if (bG==0) {Sb=1;}        }
if (Sb==4) {if (bB==0) {Sb=1;}        }
if (Sb==5) {if (bClear==0) {Sb=1;}  }
if (Sb==6) {if (bFinish==0) { printLcd(0,"Server-order sent");
                lcdPhone.clear();
                  lcdPhone.print(0, 0, "Order:" + String(Order[0]) + String(Order[1]) +
                  String(Order[2]) + String(Order[3]) + String(Order[4])+"-Sent");
                  myRadio.stopListening();                          //WIFI NRF TRANSMIT
                dataTransmit.Order[0]=Order[0];
                dataTransmit.Order[1]=Order[1];
                dataTransmit.Order[2]=Order[2];
                dataTransmit.Order[3]=Order[3];
                dataTransmit.Order[4]=Order[4];
                dataTransmit.c=dataTransmit.c+1;
                dataTransmit.tx=1;
                dataTransmit.rx=2;
                dataTransmit.val=2;
                myRadio.openWritingPipe(addresses[0]);
                myRadio.write(&dataTransmit, sizeof(dataTransmit));
                lcd.setCursor (3,1);
                lcd.print("                ");
                lcd.setCursor (3,1);
                //lcd.print(dataTransmit.text);
                lcd.print(String(dataTransmit.Order[0]));
                lcd.print(String(dataTransmit.Order[1]));
                lcd.print(String(dataTransmit.Order[2]));
                lcd.print(String(dataTransmit.Order[3]));
                lcd.print(String(dataTransmit.Order[4]));
                lcd.print(" c=");
                lcd.print(dataTransmit.c);
                lcd.print(" ");
                lcd.print(dataTransmit.tx);
                lcd.print(">");
                lcd.print(dataTransmit.rx);
                lcd.print(" ");
                lcd.print(dataTransmit.val);                  //automaton-action listen after
                                                                  RESET
                myRadio.openReadingPipe(1, addresses[0]);
                myRadio.startListening();
                printLcd(0,"Server-listening");
                Sb=7;
                }
          }
if (Sb==7) {while (myRadio.available()) {
          myRadio.read(&dataRecieve, sizeof(dataRecieve));}
          if (lastc2<dataRecieve.c&&dataRecieve.rx==1) {
          lastc2=dataRecieve.c;
          lcd.setCursor (3,2);
```

```
            lcd.print("                ");
            lcd.setCursor (3,2);
            lcd.print(String(dataRecieve.Order[0]));
            lcd.print(String(dataRecieve.Order[1]));
            lcd.print(String(dataRecieve.Order[2]));
            lcd.print(String(dataRecieve.Order[3]));
            lcd.print(String(dataRecieve.Order[4]));
            lcd.print(" c=");
            lcd.print(dataRecieve.c);
            lcd.print(" ");
            lcd.print(dataRecieve.tx);
            lcd.print(">");
            lcd.print(dataRecieve.rx);
            lcd.print(" ");
            lcd.print(dataRecieve.val);
            printLcd(0,"Server-conf received");
            delay(3000);
            printLcd(0,"Server-waiting end");
            Sb=8;
           }
 if (Sb==8) {
                  while (myRadio.available())
                  {
            myRadio.read(&dataRecieve, sizeof(dataRecieve));
                  }
            if (lastc2<dataRecieve.c&&dataRecieve.rx==1)
                  {
            lastc2=dataRecieve.c;
            lcd.setCursor (3,2);
            lcd.print("                ");
            lcd.setCursor (3,2);
            lcd.print(String(dataRecieve.Order[0]));
            lcd.print(String(dataRecieve.Order[1]));
            lcd.print(String(dataRecieve.Order[2]));
            lcd.print(String(dataRecieve.Order[3]));
            lcd.print(String(dataRecieve.Order[4]));
            lcd.print(" c=");
            lcd.print(dataRecieve.c);
            lcd.print(" ");
            lcd.print(dataRecieve.tx);
            lcd.print(">");
            lcd.print(dataRecieve.rx);
            lcd.print(" ");
            lcd.print(dataRecieve.val);
            printLcd(0,"Server-order finished");
            delay(3000);
            printLcd(0,"Server-order beads");
            Sb=1;
           }
      }
   }
}
```

Appendix B – Program of Stacks

```
int GateN=4;
#include <SPI.h>                          //WIFI NRF
#include "RF24.h"                         //WIFI NRF
//RF24 myRadio(9,10);                             // UNO      //WIFI NRF
RF24 myRadio(9,53);                               // Mega     //WIFI NRF
byte addresses[][6] = {"0"};                      //WIFI NRF
struct package {
  //char text[20]="00000";
  int Order[5]={0,0,0,0,0};
  int c=1;
  int tx=GateN+2;
  int rx=0;
  int val=0;     };
typedef struct package Package;                   //WIFI NRF
Package dataRecieve;                              //WIFI NRF
Package dataTransmit;                             //WIFI NRF
#include <Wire.h>                                 //LCD
#include <LiquidCrystal_I2C.h>                    //LCD
LiquidCrystal_I2C lcd(0x27,20,4);         //LCD 20x4
int OLDvariable = 0;
int S=1;
int lastc1 = 0;
int Sb=1;
bool calib;
bool color;
int LED;                                          //RGB LED
const int in1 = 23;                               //PINS OF STEPPER
const int in2 = 25;
const int in3 = 27;
const int in4 = 29;
int vstepper = 1;                                 //VELOCITY OF STEPPER

void setup()
{
  Serial.begin(9600);
  delay(1000);
  pinMode(LED, OUTPUT);                           //RGB LED
  pinMode(in1, OUTPUT);                           //STEPPER
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(8, INPUT_PULLUP);               //CALIBRATE BUTTON
  pinMode(7, INPUT_PULLUP);               //CALIBRATE BUTTON
  pinMode(6, INPUT_PULLUP);               //CALIBRATE BUTTON
  pinMode(5, INPUT_PULLUP);               //COLOR BUTTON
  lcd.init();                                     // LCD
  lcd.backlight();
  lcd.clear();
  printLcd(0,"Stack"+String(GateN)+"-initializing");
  printLcd(1,"TX:");
```

```
  printLcd(2,"RX:");
  if (GateN==1) LED=13;                          //13=red
  if (GateN==2) LED=12;                          //12=green
  if (GateN==3) LED=11;                          //11=blue
  if (GateN==4) LED=12;                          //12=green
  if (GateN==5) LED=13;
  analogWrite(LED, 20);                  //RGB LED
  myRadio.begin();                               //WIFI NRF
  myRadio.setChannel(0x60);
  myRadio.setPALevel(RF24_PA_MAX);
  //myRadio.setDataRate( RF24_250KBPS );
  myRadio.openReadingPipe(1, addresses[0]);
  myRadio.startListening();
  printLcd(0,"Stack"+String(GateN)+"-listening");
}

void loop(){
  calib=digitalRead(7);                  //CALIBRATE BUTTON
  if (!calib) rotationCW();
  if (S==1) {
     while (myRadio.available()) myRadio.read( &dataRecieve, sizeof(dataRecieve) );
     if (lastc1<dataRecieve.c&&dataRecieve.rx==GateN+2&&dataRecieve.val==3)
         {
       lastc1=dataRecieve.c;

         printLcd(2,"RX: c=" + String(dataRecieve.c) + " " + String(dataRecieve.tx) + ">" +
         String(dataRecieve.rx) + " " + String(dataRecieve.val));

       printLcd(0,"Stack"+String(GateN)+"-bead asked");
       delay(3000);
       printLcd(0,"Stack"+String(GateN)+"-giving bead");
       bead();

       analogWrite(LED, 250);          //RGB LED intensity HI
       delay(750);
       analogWrite(LED, 10);           //RGB LED intensity LO
       delay(750);
       analogWrite(LED, 250);          //RGB LED intensity HI
       delay(750);
       analogWrite(LED, 10);           //RGB LED intensity LO
       printLcd(0,"Stack"+String(GateN)+"-bead given");
       delay(3000);
       myRadio.stopListening();                        //WIFI NRF TRANSMIT
       dataTransmit.c=dataTransmit.c+1;
       dataTransmit.tx=GateN+2;
       dataTransmit.rx=2;
       dataTransmit.val=5;
       myRadio.openWritingPipe(addresses[0]);
       myRadio.write(&dataTransmit, sizeof(dataTransmit));
         printLcd(1,"TX: c=" + String(dataTransmit.c) + " " + String(dataTransmit.tx) +
         ">" + String(dataTransmit.rx) + " "+ String(dataTransmit.val));
```

```
        myRadio.startListening();

       S=1;
          }
          }
   if (S==2) {   S=3;    }
   if (S==3) {   S=1;    }
}

void printLcd(int line, String message) {                    //prints message at N line
  lcd.setCursor(0, line);
  lcd.print("                  ");
  lcd.setCursor(0, line);
  lcd.print(message); }

void bead() { for(int i=0;i<(64);i++)   {   rotationCW();   } }   //rotation 360 = 512 steps

void rotationCW() { step1(); step2(); step3(); step4(); step5(); step6(); step7(); step8(); }

void rotationAntiCW() {step8();step7();step6();step5();step4();step3(); step2(); step1();}

void step1(){  digitalWrite(in1, HIGH);      digitalWrite(in2, LOW);
               digitalWrite(in3, LOW);       digitalWrite(in4, LOW);
               delay(vstepper);        }

void step2(){  digitalWrite(in1, HIGH);      digitalWrite(in2, HIGH);
               digitalWrite(in3, LOW);       digitalWrite(in4, LOW);
               delay(vstepper);        }

void step3(){  digitalWrite(in1, LOW);       digitalWrite(in2, HIGH);
               digitalWrite(in3, LOW);       digitalWrite(in4, LOW);
               delay(vstepper);        }

void step4(){  digitalWrite(in1, LOW);       digitalWrite(in2, HIGH);
               digitalWrite(in3, HIGH);      digitalWrite(in4, LOW);
               delay(vstepper);        }

void step5(){  digitalWrite(in1, LOW);       digitalWrite(in2, LOW);
               digitalWrite(in3, HIGH);      digitalWrite(in4, LOW);
               delay(vstepper);        }

void step6(){  digitalWrite(in1, LOW);       digitalWrite(in2, LOW);
               digitalWrite(in3, HIGH);      digitalWrite(in4, HIGH);
               delay(vstepper);        }

void step7(){  digitalWrite(in1, LOW);       digitalWrite(in2, LOW);
               digitalWrite(in3, LOW);       digitalWrite(in4, HIGH);
               delay(vstepper);        }

void step8(){  digitalWrite(in1, HIGH);      digitalWrite(in2, LOW);
               digitalWrite(in3, LOW);       digitalWrite(in4, HIGH);
               delay(vstepper);        }
```

Appendix C – Program of Vehicle with Strategy 1

```
#include <Wire.h>                          //LCD
#include <LiquidCrystal_I2C.h>             //LCD
#include <SPI.h>                           //WIFI NRF
#include "RF24.h"                          //WIFI NRF
LiquidCrystal_I2C lcd(0x27, 20, 4);        //LCD (adress,size)
RF24 myRadio(49, 53);                      //WIFI NRF (MEGA)
byte addresses[][6] = {"0"};               //WIFI NRF

struct package {                           //WIFI NRF (data in message)
  //char text[20]="00000";
  int Order[5]={0,0,0,0,0};
  int c=1;
  int tx=2;
  int rx=0;
  int val=0;
};

typedef struct package Package;            //WIFI NRF
Package dataRecieve;                       //WIFI NRF
Package dataTransmit;                      //WIFI NRF

int FSpin  = 34;                           //LINE front
int CSpin  = 40;                           //LINE central
int LSpin  = 42;                           //LINE left
int LLSpin = 44;                   //LINE left left
int RSpin  = 38;                   //LINE right
int RRSpin = 36;                   //LINE right right
int FSSpin = 46;                   //LINE front switch

bool FS;                           //LINE front
bool CS;                           //LINE central
bool LS;                           //LINE left
bool LLS;                          //LINE left left
bool RS;                           //LINE right
bool RRS;                          //LINE right right
bool FSS;                          //LINE front switch

int LMFpin = 5;                    //left motor forward
int RMFpin = 3;                    //right motor forward
int LMBpin = 4;                    //left motor backward
int RMBpin = 2;                    //right motor backward
int LMpwm = 0;                     //pwm left motor
int PMpwm = 0;                     //pwm right motor
bool LMF ;                         //left motor forward
bool RMF ;                         //right motor forward
bool LMB ;                         //left motor backward
bool RMB ;                         //right motor backward
int vel = 100;                     //velocity of the vehicle 0-255

int LEDRpin = 30;                  //LED red   30
```

```
int LEDGpin = 28;                    //LED green 28
int LEDBpin = 26;                    //LED blue  26

unsigned long lasttime=millis();     //only for time of last loop

int S=0;                             //state of vehicle´s main automaton and arrow to S0
int lastc1 = 0;                      //aux var for wifi comm
int o=0;                             // actual position of needed bead in order
int Order[5]={0,0,0,0,0};            // order
//int Order[5]={1,3,2,3,1};          // demo order
int p=0;
int Path[5]={0,1,2,3,2};             // order

void setup() {
  pinMode(FSpin,  INPUT);
  pinMode(CSpin,  INPUT);
  pinMode(LSpin,  INPUT);
  pinMode(LLSpin, INPUT);
  pinMode(RSpin,  INPUT);
  pinMode(RRSpin, INPUT);
  pinMode(FSSpin, INPUT);
  pinMode(LMFpin, OUTPUT);
  pinMode(RMFpin, OUTPUT);
  pinMode(LMBpin, OUTPUT);
  pinMode(RMBpin, OUTPUT);
  pinMode(LEDRpin, OUTPUT);
  pinMode(LEDGpin, OUTPUT);
  pinMode(LEDBpin, OUTPUT);

  Serial.begin(9600);
  lcd.init();                                          //LCD init or begin
  lcd.backlight();                                     //LCD
  printLcd(0,"Vehicle-initializing");
  printLcd(1,"TX:");
  printLcd(2,"RX:");
  printLcd(3,"S"+String(S)+"
Order:"+String(Order[0])+String(Order[1])+String(Order[2])+String(Order[3])+String(O
rder[4])+" o"+String(o)+"p"+String(p));
  /*demo*/  while (!((LLS)&&(RRS))) {Going();}        //the vehicle goes to first position
  Stopping();                                          //automaton - action STOP
  p=0;
  myRadio.begin();                                           //WIFI NRF
  myRadio.setChannel(0x60);
  myRadio.setPALevel(RF24_PA_MAX);
  //myRadio.setDataRate( RF24_250KBPS );
  myRadio.openReadingPipe(1, addresses[0]);                  //automaton   -   action
listen
  myRadio.startListening();
  S=1;                                                       //automaton - arrow S0 to
S1
  // S=2;LedOrder();                                         //demo
  printLcd(0,"Vehicle-listening");
```

```
  printLcd(3,"S"+String(S)+"
Order:"+String(Order[0])+String(Order[1])+String(Order[2])+String(Order[3])+String(O
rder[4])+" o"+String(o)+"p"+String(p));
}

void loop() {
  if (0==1) { ReadLine();                                          //LINE test
            Serial.print(FSS);Serial.print(FS);Serial.print("-");

Serial.print(LLS);Serial.print(LS);Serial.print(CS);Serial.print(RS);Serial.println(RRS);
          }
  if (0==1) {                                                      //LED test
    digitalWrite(LEDRpin,1);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,0);
delay(1000);

digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,1);digitalWrite(LEDBpin,0);delay(1000)
;

digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,1);delay(1000)
;
          }
  if (0==1) { Stopping();analogWrite(LMFpin,150);delay(500);     //MOTOR test
            Stopping();analogWrite(RMFpin,150);delay(500);
            Stopping();analogWrite(LMBpin,150);delay(500);
            Stopping();analogWrite(RMBpin,150);delay(500);
          }
  if (0==1) { Serial.println(millis()-lasttime);                   //time of last loop
            lasttime=millis();          }
  if (0==1) { Going();
     if ((LLS)&&(RRS)) {Stopping();delay(1000);}   //simple following test with stops on
lines
          }

  if (S==1) {                                      //initial state – listening //WIFI NRF listening
    while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));}
          if   (/*lastc1<dataRecieve.c&&*/dataRecieve.rx==2   &&   dataRecieve.tx==1   &&
          dataRecieve.val==2)
          {                                        //condition to S2 - if order recieved
     lastc1=dataRecieve.c;
     Order[0]=dataRecieve.Order[0];
     Order[1]=dataRecieve.Order[1];
     Order[2]=dataRecieve.Order[2];
     Order[3]=dataRecieve.Order[3];
     Order[4]=dataRecieve.Order[4];          //action to S2 - write in the order to memory
     o=0;
     printLcd(2,"RX: c=" + String(dataRecieve.c) + " " + String(dataRecieve.tx) + ">" +
String(dataRecieve.rx) + " " + String(dataRecieve.val) + " " + String(Order[0]) +
String(Order[1]) + String(Order[2]) + String(Order[3]) + String(Order[4]));
     printLcd(3,"S" + String(S) + " Order:" + String(Order[0]) + String(Order[1]) +
String(Order[2]) + String(Order[3]) + String(Order[4]) + " o" + String(o) + "p"+String(p));
     printLcd(0,"Vehicle-order obted");  delay(3000);
```

```
    printLcd(0,"Vehicle-conf sending");          //action to S2 - send confirmation to
server
    myRadio.stopListening();                     //WIFI NRF TRANSMIT
    dataTransmit.c=dataTransmit.c+1;
    dataTransmit.tx=2;
    dataTransmit.rx=1;
    dataTransmit.val=2;
    myRadio.openWritingPipe(addresses[0]);
    myRadio.write(&dataTransmit, sizeof(dataTransmit));
        printLcd(1,"TX: c=" + String(dataTransmit.c) + " " + String(dataTransmit.tx) + ">"
        + String(dataTransmit.rx) + " " + String(dataTransmit.val));
    S=2;                                         //arrow to S2
        printLcd(3,"S"+String(S)+" Order:" + String(Order[0]) + String(Order[1]) +
        String(Order[2]) + String(Order[3]) + String(Order[4]) + " o" + String(o) + "p" +
        String(p));
    LedOrder();
    printLcd(0,"Vehicle-going");
    }                                            //end of condition
 }                                               //end of S==1//the vehicle goes to first position

 if (S==2) {Going();                             //state S2 - following the line
         if ((LLS)&&(RRS)) {Stopping();
                        p=p+1;
                        if (p==5) {p=0;}
                        if (Path[p]==Order[o]){
                          printLcd(0,"Vehicle-asking bead");
                          myRadio.stopListening();          //WIFI NRF TRANSMIT
                          dataTransmit.c=dataTransmit.c+1;
                          dataTransmit.tx=2;                //adress of vehicle - sender
                          dataTransmit.rx=Order[o]+2;       //adress of GateN - receiver
                          dataTransmit.val=3;
                          myRadio.openWritingPipe(addresses[0]);
                          myRadio.write(&dataTransmit, sizeof(dataTransmit));
                          printLcd(1,"TX: c=" + String(dataTransmit.c) + " " +
                          String(dataTransmit.tx) + ">" + String(dataTransmit.rx) + " " +
                          String(dataTransmit.val));
                          myRadio.startListening();
                          S=3;                          //automaton - arrow to S3 - waiting for the
bead
                          printLcd(3,"S" + String(S) + " Order:" + String(Order[0]) +
                          String(Order[1]) + String(Order[2]) + String(Order[3]) +
                          String(Order[4]) + " o" + String(o) + "p" + String(p));
                          }
                     else {printLcd(0,"Vehicle-wrong bead");
                           delay(2000);
                           S=2;                         //automaton - arrow to S2 - continue to next
stack
                          printLcd(3,"S" + String(S) + " Order:" + String(Order[0]) +
                          String(Order[1]) + String(Order[2]) + String(Order[3]) +
                          String(Order[4]) + " o" + String(o) + "p" + String(p));
                           printLcd(0,"Vehicle-going");
                           }
```

```
                    }                                   //end of condition
        }                                               //end of S==2

if (S==3) {                             //state - listening and waiting for obtain the right
bead
   while (myRadio.available())
        {myRadio.read(&dataRecieve, sizeof(dataRecieve));}      //WIFI NRF listening
if ( /*lastc1<dataRecieve.c&&*/ dataRecieve.rx==2 && dataRecieve.tx==Order[o]+2 &&
dataRecieve.val==5)
{                                       //condition of S3 - confirmation that the right bead was
obtained
    lastc1=dataRecieve.c;
        printLcd(2,"RX: c=" + String(dataRecieve.c) + " " + String(dataRecieve.tx) + ">" +
        String(dataRecieve.rx) + " " + String(dataRecieve.val) + " " + String(Order[0]) +
        String(Order[1]) + String(Order[2]) + String(Order[3]) + String(Order[4]));
    printLcd(0,"Bead obtained");  delay(3000);
    printLcd(0,"Vehicle-conf sending");                 //action to S2 - send confirmation to
GateN
    myRadio.stopListening();                            //WIFI NRF TRANSMIT
    dataTransmit.c=dataTransmit.c+1;
    dataTransmit.tx=2;
    dataTransmit.rx=Order[o]+2;
    dataTransmit.val=2;
    myRadio.openWritingPipe(addresses[0]);
    myRadio.write(&dataTransmit, sizeof(dataTransmit));
        printLcd(1,"TX: c=" + String(dataTransmit.c) + " " + String(dataTransmit.tx) + ">"
        + String(dataTransmit.rx) + " " + String(dataTransmit.val));
                                                        // end of actions on arrow to S2
    S=2;                                                 //arrow to S2
    Order[o]=0;
        printLcd(3,"S" + String(S) + " Order:" + String(Order[0]) + String(Order[1]) +
        String(Order[2]) + String(Order[3]) + String(Order[4]) + " o" + String(o) +
        "p"+String(p));
    o=o+1;
    LedOrder();
    printLcd(0,"Vehicle-going");
if    ((Order[0]==0)&&(Order[1]==0)&&(Order[2]==0)&&(Order[3]==0)&&(Order[4]==0))
{OrderFinished();}
    }                                                   //end of condition
  }                                                     //end of S==3
}                                                       //end of loop

void printLcd(int line, String message) {               //prints message at N line
  lcd.setCursor(0, line);
  lcd.print("                ");
  lcd.setCursor(0, line);
  lcd.print(message);
}

void ReadLine() {
  FS =   digitalRead(FSpin);                            //LINE front
  CS =  !digitalRead(CSpin);                            //LINE central
```

```
  LS =  !digitalRead(LSpin);                    //LINE left
  LLS = !digitalRead(LLSpin);                   //LINE left left
  RS =  !digitalRead(RSpin);                    //LINE right
  RRS = !digitalRead(RRSpin);                   //LINE right right
  FSS =  digitalRead(FSSpin);                   //LINE front switch
}

void Going() {
  ReadLine();
  if ((!LS)&&(!CS)&&(!RS)) {analogWrite(LMFpin,0);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,0);
                            analogWrite(RMBpin,0);
                           }
  else if (( LS)&&( CS)&&( RS)) {analogWrite(LMFpin,vel);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,vel);
                            analogWrite(RMBpin,0);
                           }
  else if ((!LS)&&( CS)&&(!RS)) {analogWrite(LMFpin,vel);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,vel);
                            analogWrite(RMBpin,0);
                           }
  else if ((!LS)&&( CS)&&( RS)) {analogWrite(LMFpin,vel);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,vel-30);
                            analogWrite(RMBpin,0);
                           }
  else if ((!LS)&&(!CS)&&( RS))    {analogWrite(LMFpin,vel-30);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,0);
                            analogWrite(RMBpin,0);
                           }
  else if (( LS)&&( CS)&&(!RS)) {analogWrite(LMFpin,vel-30);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,vel);
                            analogWrite(RMBpin,0);
                           }
  else if (( LS)&&(!CS)&&(!RS)) {analogWrite(LMFpin,0);
                            analogWrite(LMBpin,0);
                            analogWrite(RMFpin,vel-30);
                            analogWrite(RMBpin,0);
                            }
  }

void Stopping() {
  ReadLine();
  analogWrite(LMFpin, 0);
  analogWrite(RMFpin, 0);
  analogWrite(LMBpin, 0);
  analogWrite(RMBpin, 0);   }
```

```
void LedOrder() {
  if
(Order[o]==1){digitalWrite(LEDRpin,1);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,
0);}
  if
(Order[o]==2){digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,1);digitalWrite(LEDBpin,
0);}
  if
(Order[o]==3){digitalWrite(LEDRpin,0);digitalWrite(LEDGpin,0);digitalWrite(LEDBpin,
1);}
  if
(Order[o]==0){digitalWrite(LEDRpin,1);digitalWrite(LEDGpin,1);digitalWrite(LEDBpin,
1);}
  }

void OrderFinished()                                    // all beads obtained,  if (Order[0]==0)
  {
                LedOrder();
                printLcd(0,"Vehicle-finished");
                while(p==0) {
                                while (!((LLS)&&(RRS))) {Going();}
                                p=p+1;
                                if (p==5) {p=0;}
                                }

                myRadio.stopListening();                //WIFI NRF TRANSMIT
                dataTransmit.c=dataTransmit.c+1;
                dataTransmit.tx=2;
                dataTransmit.rx=1;
                dataTransmit.val=3;
                myRadio.openWritingPipe(addresses[0]);
                myRadio.write(&dataTransmit, sizeof(dataTransmit));

                lcd.setCursor (3,1);
                lcd.print("               ");
                lcd.setCursor (3,1);
                //lcd.print(dataTransmit.text);
                lcd.print(" c=");
                lcd.print(dataTransmit.c);
                lcd.print(" ");
                lcd.print(dataTransmit.tx);
                lcd.print(">");
                lcd.print(dataTransmit.rx);
                lcd.print(" ");
                lcd.print(dataTransmit.val);

                Stopping();
                delay(10000000);
  }
```

Appendix D – Changes in Program of Vehicle using Strategy 2

```
if (S==3) {
   while (myRadio.available()) {
                                        myRadio.read(&dataRecieve, sizeof(dataRecieve));
                                          }
        if (/*lastc1<dataRecieve.c&&*/ dataRecieve.rx == 2&&dataRecieve.tx == Order[o]
        + 2 && dataRecieve.val == 5)
        {
    lastc1=dataRecieve.c;

        printLcd(2,"RX: c=" + String(dataRecieve.c) + " " + String(dataRecieve.tx) + ">" +
        String(dataRecieve.rx) + " " + String(dataRecieve.val) + " " + String(Order[0]) +
        String(Order[1]) + String(Order[2]) + String(Order[3]) + String(Order[4]));

    printLcd(0,"Bead obtained");
    delay(3000);
    printLcd(0,"Vehicle-conf sending");      //action to S2 - send confirmation to GateN

    myRadio.stopListening();                 //WIFI NRF TRANSMIT
    dataTransmit.c=dataTransmit.c+1;
    dataTransmit.tx=2;
    dataTransmit.rx=Order[o]+2;
    dataTransmit.val=2;

    myRadio.openWritingPipe(addresses[0]);
    myRadio.write(&dataTransmit, sizeof(dataTransmit));
    printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"
    +String(dataTransmit.rx)+" "+String(dataTransmit.val));

    Order[o]=0;
    o=o+1;

    if (Path[p]==Order[o])
        {
      S=3;
    }
    else {
        S=2;                                                    //arrow to S2
    printLcd(3,"S"+String(S)+" Order:"+String(Order[0])+String(Order[1])+
    String(Order[2])+String(Order[3])+String(Order[4])+" o"+String(o)+"p"+String(p));
    LedOrder();
    printLcd(0,"Vehicle-going");
    }

if ((Order[0]==0) && (Order[1]==0) && (Order[2]==0) && (Order[3]==0) &&
(Order[4]==0))
        {
        OrderFinished();
        }
    }                                                           //end of condition
 }                                                              //end of S==3
```

Appendix E – Changes in Program of Vehicle using Strategy 2

```
if (S==2) {Going();                  //state S2 - following the line
   if ((LLS)&&(RRS)) {Stopping();
       p=p+1;
    if (p==5) {p=0;}
    if (Path[p]==Order[o]){
                              printLcd(0,"Vehicle-asking bead");
                              myRadio.stopListening();       //WIFI NRF TRANSMIT
                              dataTransmit.c=dataTransmit.c+1;
                              dataTransmit.tx=2;             //adress of vehicle - sender
                              dataTransmit.rx=Order[o]+2;  //adress of GateN - receiver
                              dataTransmit.val=3;
                              myRadio.openWritingPipe(addresses[0]);
                              myRadio.write(&dataTransmit, sizeof(dataTransmit));
                                printLcd(1,"TX:  c=" + String(dataTransmit.c) + " " +
                                String(dataTransmit.tx) + ">" + String(dataTransmit.rx) + "
                                " + String(dataTransmit.val));
                              myRadio.startListening();
                              S=3;              //automaton - arrow to S3 - waiting for the bead
                                printLcd(3,"S"+String(S) + " Order:" + String(Order[0]) +
                                String(Order[1]) + String(Order[2]) + String(Order[3]) +
                                String(Order[4]) + " o" + String(o) + "p" + String(p));
                             }
                       else {
                              printLcd(0,"Vehicle-wrong bead");
                              delay(2000);
                              S=4;                          //automaton - arrow to S4
                                printLcd(3,"S"+String(S) + " Order:" + String(Order[0]) +
                                String(Order[1]) + String(Order[2]) + String(Order[3]) +
                                String(Order[4]) + " o" + String(o) + "p"+String(p));
                              printLcd(0,"Vehicle-going");
                              }
                       }                    //end of condition
        }                                   //end of S==2

if (S==3) {                                 //state - listening and waiting for obtain the right bead
                                            //WIFI NRF listening
   while (myRadio.available()) {myRadio.read(&dataRecieve, sizeof(dataRecieve));}
        if (/*lastc1<dataRecieve.c&&*/dataRecieve.rx == 2 && dataRecieve.tx == Order[o]
        + 2 && dataRecieve.val==5)
        {
     lastc1=dataRecieve.c;

        printLcd(2,"RX: c=" + String(dataRecieve.c) + " " + String(dataRecieve.tx) + ">" +
        String(dataRecieve.rx) + " " + String(dataRecieve.val) + " " + String(Order[0]) +
        String(Order[1]) + String(Order[2]) + String(Order[3]) + String(Order[4]));

     printLcd(0,"Bead obtained");
     delay(3000);
     printLcd(0,"Vehicle-conf sending");      //action to S2 - send confirmation to GateN
```

```
    myRadio.stopListening();                    //WIFI NRF TRANSMIT
    dataTransmit.c=dataTransmit.c+1;
    dataTransmit.tx=2;
    dataTransmit.rx=Order[o]+2;
    dataTransmit.val=2;
    myRadio.openWritingPipe(addresses[0]);
    myRadio.write(&dataTransmit, sizeof(dataTransmit));

    printLcd(1,"TX: c="+String(dataTransmit.c)+" "+String(dataTransmit.tx)+">"
    +String(dataTransmit.rx)+" "+String(dataTransmit.val));

    Order[o]=0;
    S=4;                                        //arrow to S4
        if ((Order[0]==0) && (Order[1]==0) && (Order[2]==0) && (Order[3]==0) &&
        (Order[4]==0))
        {
        OrderFinished();
        }
        }       //end of condition
  }             //end of S==3

if (S==4){
  o=o+1;
  if(o==5){ o=0;
              S=2;
        printLcd(3,"S"+String(S) + " Order:" + String(Order[0]) + String(Order[1]) +
        String(Order[2]) + String(Order[3]) + String(Order[4]) + " o" + String(o) + "p" +
        String(p));
        LedOrder();
        printLcd(0,"Vehicle-going");
  }
if (Path[p]==Order[o]){
     S=3;
    }
else { S=4;  }
}
```