LOGO AND DESIGN

FRONT PAGE

I- About Us

Smart Warehouse is a harmonic team formed of 20 student at fourth year of electrical engineering department , majoring in communications and electronics section of Faculty of Engineering, Alexandria University.

Team members are all passionate students with new creative ideas and awareness of their field of study. The project team includes students with a variety of experiences that achieves the harmony among them and covers all needed aspects to maintain a project.

Two of the team students are the superiors of the whole 2016 batch, others have a strong technical backgrounds and visions earned through trainings and practical work of many projects along the years of study, many have also participated in social and civil work that serves the society and the community of the faculty and other students around them which qualified them to have many skills beside technical ones that helped in handling the project issues finalizing it in a well shaped appearance.

Smart **W**arehouse is an application based on Machine to Machine technology in the field of manufacturing. The project helps enhancing an actual applied system in one of the leading companies in business domain, a consumer goods company Procter and Gamble (P&G). The case study is a current system the company uses and the project aims at enhancing and converting it into an automated system.

Team members (alphabetically ordered):

Abdelrahman Mahmoud Zayed

Ahmad Hamdy Mohamed

Ahmed Abdelhamid AlAttar

Ahmed adel mahmoud

Ahmed Adel Saad

Ahmed Hamdy Mahmoud

Ahmed Mohamed Tawfik

Ahmed Mostafa Abdulah

Dalia Ibrahim Mohamed

Dina Sabry Ahmed

Eslam Salah Zaki

Fathy Rafeik Fathy

Hazem Mohamed Kamel

Hossam Hassan Ramadan

Omar Ahmed Medhat

Omar Mohamed Yehia Saad El Welely

Osama Mohamed Gamal El din sultan

Taha Gamal Abd El-Salam

Waleed Mabrouk El-Henawy

Ziad Elsayed Ibrahem

II- Preface

This book shows an application on Machine-to-machine (M2M) communications technology specified in the manufacturing domain, Smart Warehouse. The project takes a human-being based system and turns it to an automated system that applies the same tasks with mainly robots (forklifts) to perform the required system functions; which privileged the system to be more efficient and more secured.

Machine to Machine is a technology when used in industrial domain it's to ensure costs are managed properly and processes are executed efficiently.

At first, we worked on finalizing a whole one robot as a hardware built object with all its characteristics and tools. then we made two more copies of it to add more features to the system dealing multitasking system.

In parallel we were working on the communications and software part that will connect all the robots together and to the system.

At the beginning of our work we depended on using certain techniques and tools, turned out to perform in an unexpected way that didn't met the required level of the project's plan after a long time of hard work on them. So, we had to move to using other techniques instead.

The whole system will be illustrated in the upcoming chapters and all of the failed trials will be moved to the future work part.

III- Acknowledgement

At the very beginning, we would like to thank our Professor, **Dr. Adel Al-Fahhar** for being our project's supervisor. He has been a very helpful illustrator and supported as a lot with many issues we needed to make our project come to reality.

We would love also to thank our team leader **Hazem Kamel**, for taking the lead of the whole project along the year. Also all the gratitude is presented to all **Smart Warehouse** team members, who have been very interactive and supportive to each-other, handling extra tasks and their motivation that was led by their harmony and enthusiasm.

Contents

Chapter 1 "Introduction to Machine to Machine Technology"……17

1. Introduction……18

1.1. Motivation……...19

1.2. Internet of Things (IoT)……...21

1.2.1. IoT Definition…… 21

1.2.2. Iot and M2M……22

1.3. M2M Communication…… …23

1.4. M2M Applications…… …..23

2. System Overview……26

Chapter 2 "System Components"……...29

1. System Components Function……30

1.1. Arduino Mega…….. ……30

1.2. Wheels…….....31

1.3. Ultrasonic Sensor……..31

1.4. Infrared Reflection Sensor……32

1.5. Tags……...32

1.6. RFID Modulator Features……33

1.6.1. Tags Types…….....33

1.6.2. RFID Nature……33

1.7. Lithium Polymer Battery……..34

1.8. IMAX B6 Balance Battery Charger……..35

1.9. Servo Motor……..36

1.10. DC-DC Converter……37

1.11. DC Motors with Encoders……38

1.11.1. Motor and Encoder Funtion…….38

1.12. DC Motor with Driver……..39

1.13. Razor IMU 9 DoF…….39

1.13.1. IMU Definition……39

1.13.2. Degrees of Freedom……40

1.13.3. Inside IMU 9 DoF……40

Chapter 3 "Hardware System" (Robots)……...51

1. Microcontrollers…… …….52

1.1. Microcontroller Selection Criteria……54

2. Automated Guided Vehicle…….54

2.1. AVG Model realization…….55

2.1.1. AVG Microcontroller……...55

2.1.2. AVG Chassis……56

2.1.3. AVG Motors…….....57

2.1.4. AVG Lithium Polymer Battery…….....58

3. Navigation and Localization……...58

3.1. Navigation……58

3.1.1. Methods of Navigation…….....59

3.1.1.1. IMU (9 DoF) + PID Controller……59

3.1.1.2. Line Following…….60

3.2. Localization …….....61

3.2.1. Dead Reckoning……...61

3.2.2. Global Positioning System……...62

3.2.3. Landmarks……62

3.2.3.1. Landmark Options……62

4. Obstacle Avoidance……64

5. System Components Arduino Codes…….....64

Chapter 4 "System Connection" (Network)……73

1. Preamble……75

2. Networks……75

2.1. Network Types……75

2.2. M2M Network Type……75

2.3. Sockets……..76

3. Wireless Connection Protocols Overview……..76

3.1. Bluetooth Wireless Technology……76

3.2. Second Generation Wireless Technology (2G)……77

3.3. Third Generation Wireless Technology (3G)……...77

3.4. Fourth Generation Wireless technology (4G)……...78

3.5. Zigbee Wireless Technology……79

3.6. UWB Wireless Technology……..79

4. Wi-Fi Technology……...82

4.1. Wi-Fi Alliance……..82

4.2. IEEE 802.11 Standard…….....82

4.3. IEEE 802.11X History…….....83

4.4. Wi-Fi Frequencies……84

4.4.1. 802.11a Standard……..86

4.4.2. 802.11b Standard……..86

4.4.3. 802.11g Standard……..86

4.4.4. The more advanced 802.11n Standard……87

4.5. Wi-Fi Frequency Band……87

5. How Wi-Fi Works……..88

5.1. Access Point……..88

5.1.1. Access Point Definition……88

5.1.2. Access Point Utilization……...89

5.1.3. Hotspot Access Point Application……89

6. Data Exchange in Wi-Fi……90

6.1. Data Exchange Phases……..90

6.1.1. Phase I……..90

6.1.2. Phase II……90

6.1.3. Phase II……90

6.2. Exchanged Data Format……91

6.3. MAC Header and Payload……92

6.4. Frame Check Sequence……95

7. ESP8266 Wi-Fi Module……95

7.1. WT8266-SI Wi-Fi Module……...96

7.2. WT8266-SI Module Features……...97

8. Wi-Fi Module Hardware Connection……99

Chapter 5 "Wi-Fi Connection and Configuration"……104

1. Raspberry Pi Kit……105

1.1. Software Connection……..106

1.2. Hardware Connection……111

2. Raspbian Operating System……..111

3. Network Connection Requirements……..112

4. Wi-Fi Network Configuration……..114

4.1. Wi-Fi Configuration Method One……..114

4.2. Wi-Fi Configuration Method One……..118

5. Getting Wi-Fi Network Details……118

6. Adding Wi-Fi Network Details to Raspberry Pi……..119
.

Chapter 6 "Server Side"……..……120

1. Phase one "Single Robot"……123
    1.1. Mapping……..123
        1.1.1. Mapping Versions……..123
    1.2. Shortest Paths Algorithms……..126
        1.2.1. Shortest Path Problem Definition……...127
        1.2.2. Algorithms……128
        1.2.3. Dijkstra's Algorithm……...129
            1.2.3.1. History……...129
            1.2.3.2. Algorithm……..130
            1.2.3.3. Dijkstra's Algorithm Explication……..131
            1.2.3.4. Pseudocode……133
            1.2.3.5. Using a Priority Queue……..135
        1.2.4. Understanding Dijekstra's Algorithm……133
        1.2.5. Dijkstra's Algorithm Implementation Approaches……140
            1.2.5.1. Easy Approach……..141
            1.2.5.2. Dijkstra Approach……141
            1.2.5.3. Modern Approach……142
        1.2.6. Correction Proof of Dijkstra Algorithm……142
        1.2.7. Dijkstra's Algorithm Characteristics……..144
            1.2.7.1. Dijkstra's Algorithm Advantages……..145
            1.2.7.2. Dijkstra's Algorithm Disadvantages……..145
        1.2.8. Dijkstra's Algorithm Applications……..145
    1.3. System Description……146

1.3.1. System Brain……..147

1.3.2. Brain Prospects……...147

1.3.3. Brain Steps of Implementation……...148

1.4. Single Robot Phase Simulation……...149

1.4.1. Python Programming Language……150

1.4.2. System Simulation……150

2. Phase Two "Multi-Robot"……153

2.1. System database……..154

2.1.1. Database ……154

2.1.1.1. DBMS Functions……...154

2.1.2. Database Model……..154

2.1.2.1. Relational Database……...156

2.1.2.2. SQL……...156

2.1.3. Database in Python……156

2.1.3.1. Python DB-API……156

2.1.4. My SQL Database Server……...156

2.1.4.1. My SQL Server Selection……158

2.1.4.2. SQL Connectivity Steps……159

2.1.5. System Database Tables……160

2.2. Multithreading……164

2.2.1. Multithreading Definition……..166

2.2.2. Threads versus Processes……...167

2.2.3. Multithreading Benefits……169

2.2.4. Multithreading Concepts ……...171

2.2.4.1. Concurrency and Parallelism……171

2.2.4.2. Looking at Multithreading Structure……171

2.2.4.3. User Lever Threads……..….172

2.2.4.4. User Level Threads State……..173

2.2.5. Multithreading in Python……...171

2.2.5.1. Stating a New Thread……173

2.2.5.2. The Threading Module……..174

2.2.5.3. Creating Threads Using Threading Module……..175

2.2.5.4. Example……177

2.2.6. Synchronization Threads……..……..178

2.2.6.1. Example……178

2.3. Multi Robot System Description……180

2.3.1. Brain Prospects ……..181

2.3.2. Brain Implementation Steps……...182

2.4. Multi Robot Phase Simulation……183

2.5. Forklifts Collision……...186

Chapter 7 "Future Work"……188

1. Using Raspberry Pi kits instead of Arduino Kits……189
2. Increasing Forklifts Number……...189
3. Zigbee Protocol Communication……189
4. Real Time Monitoring System……....190
5. Razor IMU Usage……...190
6. Swarming Algorithm……..190

References……192

List of figures

Chapter 1

- Figure 1.1: System Overview.

Chapter 2

- Figure 2.1: Arduino Kit

- Figure 2.2: Wheels.

- Figure 2.3: Ultrasonic Sensor.

- Figure 2.4: Infrared Reflection Sensor.

- Figure 2.5: RFID Module.

- Figure 2.6: Lithium Polymer battery.

- Figure 2.7: IMAX B6 Balance Battery Charger.

- Figure 2.8: Servo Motor.

- Figure 2.9: Servo Motor.

- Figure 2.10: LM2596HV DC-DC Converter.

- Figure 2.11: IMU 9 DoF

Chapter 3

- Figure 3.1: DC Motor Arduino Code

- Figure 3.2: Infrared Sensor Arduino Code

- Figure 3.3: QTR Arduino Code

- Figure 3.4: RFID Arduino Code

- Figure 3.5: Servo Motor Arduino Code

- Figure 3.6: Ultrasonic Arduino Code

- Figure 3.7: Ultrasonic Arduino Code

- Figure 3.8: Line Finder Arduino Code

Chapter 4

- Figure 4.1: Wireless Communication Technologies Characteristics.

- Figure 4.2: Graphical Representation of Wi-Fi 2.4 GHz Band.

- Figure 4.3: Wi-Fi Data Exchange.

- Figure 4.4: WT8266-SI Wi-Fi Module.

- Figure 4.5: WT8266-SI Module Hardware Connection

Chapter 5

- Figure 5.1: Edimax Wireless 802.11 b\g\n USB Adaptor.

Chapter 6

- Figure 6.1: Version 1 Map Generated Matrix Code.

- Figure 6.2: First Version Map Simulation.

- Figure 6.3: Second Version Map Simulation.

- Figure 6.4: Dijkstra's Algorithm Practical Case.

- Figure 6.5: Dijkstra's Algorithm Practical Case.

- Figure 6.6: Dijkstra's Algorithm Practical Case.

- Figure 6.7: Dijkstra's Algorithm Implementation Approaches.

- Figure 6.8: Dijkstra's Algorithm Shortest Path Calculation.

- Figure 6.9: Dijkstra's Algorithm Proof of the Claim.

- Figure 6.10: Software System Explanation.

- Figure 6.11: Software System Explanation.

- Figure 6.12: Software System Explanation.

- Figure 6.13: Phase 1 Software System Simulation.

- Figure 6.14: Phase 1 Software System Simulation.

- Figure 6.15: Phase 1 Software System Simulation .

- Figure 6.16: Production Line Database Table.

- Figure 6.17: Robots Database Table.

- Figure 6.18: Flags Database Table.

- Figure 6.19: Difference between a Process and a Thread.

- Figure 6.20: Per Process and Per Thread Items.

- Figure 6.21: Phase 2 Software System Simulation.

- Figure 6.22: Phase 2 Software System Simulation.

- Figure 6.23: Phase 2 Software System Simulation.

Chapter 7

- Figure 7.1: Forklift
- Figure 7.2: Forklift
- Figure 7.3: Forklift
- Figure 7.4: Forklift

List of Tables

Chapter 2

- Table 2.1: System component characteristics and features.

Chapter 4

- Table 4.1: Different Wireless Communication Technologies Comparison

Chapter 1

"Introduction to Machine to Machine Technology"

1. Introduction

Machine to machine (M2M) is a broad label that can be used to describe any technology that enables networked devices to exchange information and perform actions without the manual assistance of humans.

M2M communication is often used for remote monitoring. In product restocking, for example, a vending machine can message the distributor when a particular item is running low.

M2M communication is an important aspect of warehouse management, remote control, robotics, traffic control, logistic services,

supply chain management, fleet management and telemedicine. It forms the basis for a concept known as the Internet of Things (IoT).

Machine to Machine technology can be used in many applications, with different major categories. The possibilities in the realm of M2M can be seen in four major use cases, which are;

- Manufacturing
- Home Appliances
- Health Care Device Management
- Smart Utility Management

Our project will be as an application of the Manufacturing category.

A warehouse is a commercial building for storage of goods. Warehouses are used by manufacturers, importers, exporters,

wholesalers, transport businesses, customs, etc. They are usually large plain buildings in industrial areas of cities, towns and villages.

They usually have loading docks to load and unload goods from trucks. Sometimes warehouses are designed for the loading and unloading of goods directly from railways, airports, or seaports. They often have cranes and forklifts for moving goods, which are usually placed on ISO standard pallets loaded into pallet racks. Stored goods can include any raw materials, packing materials, spare parts, components, or finished goods associated with agriculture, manufacturing and production.

1.1. Motivation

In a built in system used in companies and factories to transport products from production lines to storage places. The forklifts are responsible for this transport process, they are controlled manually and a workman is the one giving instructions to the forklifts to move the objects around the workspace. This process being implemented manually, could be less accurate an inefficient than being done automatically.

The workmen being human beings make them able to be exposed to any sudden action could affect his concentration or affected by any circumstances with a random probability that would influence the system efficiency and may cause delay of a high cost to the company whenever that happens.

The main desire here is to convert the whole system into an automated one, having automated forklifts that do the workman job, sensing the needed objects movements from a certain place to another, taking into

consideration a complete system that allows communication between many forklifts and between a forklift and an object. That would be used using Machine to Machine communication technique. A WiFi connection among forklifts and software system to command the whole objects.

The history of M2M (Machine to Machine) communication touches the year 1995 when Siemens launched a GSM Data module titled 'M1' to enable machines to communicate over wireless networks. By 1997, M2M wireless technology became more established and sophisticated as customized modules were developed and launched for the needs of different vertical markets such as automotive and telematics. Today, M2M data modules are becoming extremely sophisticated with a variety of capabilities including flexible land grid array surface mounting, on-board global positioning (GPS) technology, M2M enabled SIM cards and embedded Java technology. In early 2010, market operators from United States including AT&T, KPN, Rogers, Telcel / America Movil and Jasper Wireless began to work together in the creation of a M2M site, which will serve as a hub for developers in the field of M2M communication electronics. The same year a strategic alliance between Vodafone, Verizon Wireless and nPhase (a joint partnership of Qualcomm and Verizon) was announced to provide global M2M solutions that would enable customers to roll out M2M solutions across Europe and the US.

M2M communication is useful when you have a significant number of devices to communicate with, whether they are mobile or wired interfaces, and when other methods of communication are impractical or otherwise undesirable. The key concerns related with M2M communication are of

addressing and security.

1.2. Internet of Things (IOT)

As stated previously that M2M forms the basis for a concept known as the Internet of Things (IoT), it's necessary to illustrate the concept of (IOT) and declare the difference and the relationship between them both.

1.3.1. IOT definition

The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

A thing, in the Internet of Things, can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low, or any other natural or man-made object that can be assigned an IP address and provided with the ability to transfer data over a network.

IOT has evolved from the convergence of wireless technologies, micro-electromechanical systems (MEMS), micro-services and the Internet. The convergence has helped tear down the silo walls between operational technology (OT) and information technology (IT), allowing unstructured machine-generated data to be analyzed for insights that will drive improvements.

1.3.2. IOT and M2M

Differentiating between IOT and M2M we can be certain that these two concepts do indeed have different meanings. Most conclude that Internet of Things is a broader concept, which will evolve from M2M and other technologies.

We simply put, Machine-to-Machine is where “Machines” use network resources to communicate with remote application infrastructure for the purposes of monitoring and control, either of the “machine” itself, or the surrounding environment. The potential interconnection of smart objects and the way we interact with the environment is what The Internet of Things is envisioned to be, where the physical world will merge with the digital world.

In an attempt to explain the relationship between both concepts, Matt Hatton compares M2M to the plumbing of the Internet of Things. M2M is what provides The Internet of Things with the connectivity that enables capabilities, which would not be possible without it.

M2M with Internet protocols could be considered a subset of the Internet of Things and understood from a more vertical and closed point of view. On the other hand, the Internet of Things encompasses a more horizontal and meaningful approach where vertical applications are pulled together to address the needs of many people.

1.3. Machine to Machine Communication

As previously stated, machine-to-machine communication makes the Internet of Things possible. According to Forbes, M2M is among the fastest-growing types of connected device technologies in the market right now, largely because M2M technologies can connect millions of devices within a single network. The range of connected devices includes anything from vending machines to

medical equipment to vehicles to buildings. Virtually anything that houses sensor or control technology can be connected to some sort of wireless network.

This sounds complex, but the driving thought behind the idea is quite simple. Essentially, M2M networks are very similar to LAN or WAN networks, but are exclusively used to allow machines, sensors, and controls, to communicate. These devices feed information they collect back to other devices in the network. This process allows a human (or an intelligent control unit) to assess what is going on across the whole network and issue appropriate instructions to member devices.

1.4. M2M Applications

M2M technology not only can be used in many different applications concerning a specific category, it's also used in many of our life's fields and can be applied to more applications with different scopes and purposes.

1.4.1. Manufacturing

Every manufacturing environment - whether it’s food processing or general product manufacturing - relies on technology to ensure costs are managed properly and processes are executed efficiently. Automating manufacturing processes within such a fast-paced environment is expected to improve processes even more. In the manufacturing world, this could involve highly automated equipment maintenance and safety procedures.

For example, M2M tools allow business owners to be alerted on their smart-phones when an important piece of equipment needs servicing, so they can

address issues as quickly as they arise. Sophisticated networks of sensors connected to the Internet could even order replacement parts automatically.

1.4.2. Home Applications

IoT already affects home appliance connectivity through platforms like Nest. However, M2M is expected to take home-based IoT to the next level. Manufacturers like LG and Samsung are already slowly unveiling smart home appliances to help ensure a higher quality of life for occupants.

For example, an M2M-capable washing machine could send alerts to the owners' smart devices once it finishes washing or drying, and a smart refrigerator could automatically order groceries from Amazon once its inventory is depleted. There are many more examples of home automation that can potentially improve quality of life for residents, including systems that allow members of the household to remotely control HVAC systems using their mobile devices. In situations where a homeowner decides to leave work early, he or she could contact the home heating system before leaving work to make sure the temperature at home will be comfortable upon arrival.

1.4.3. Health Care Devices

One of the biggest opportunities for M2M technology is in the realm of health care. With M2M technology, hospitals can automate processes to ensure the highest levels of treatment. Using devices that can react faster than a human healthcare professional in an emergency situation make this possible. For instance, when a patient's vital signs drop below normal, an M2M-connected life support device could automatically administer oxygen and additional care until a healthcare professional arrives on the scene. M2M also allows patients to be monitored in their own homes instead of in hospitals or care centers. For

example, devices that track a frail or elderly person's normal movements can detect when he or she has had a fall and alert a healthcare worker to the situation.

1.4.4. Smart Utility management

In the new age of energy efficiency, automation will quickly become the new normal. As energy companies look for new ways to automate the metering process, M2M comes to the rescue, helping energy companies automatically gather energy consumption data, so they can accurately bill customers. Smart meters can track how much energy a household or business uses and automatically alert the energy company, which supplants sending out an employee to read the meter or requiring the customer to provide a reading. This is even more important as utilities move toward more dynamic pricing models, charging consumers more for energy usage during peak times.

A few key analysts predict that soon, every object or device will need to be able to connect to the cloud. This is a bold but seemingly accurate statement. As more consumers, users, and business owners demand deeper connectivity, technology will need to be continually equipped to meet the needs and challenges of tomorrow. This will empower a wide range of highly automated processes, from equipment repairs and firmware upgrades to system diagnostics, data retrieval, and analysis. Information will be delivered to users, engineers, data scientists, and key decision-makers in real time, and it will eliminate the need for guesswork.

Our project Smart Warehouse by M2M technology is an application of the M2M manufacturing section.

2. System Overview:

The system mainly consist of three basic parts:

1- The Brain (server)

2- The Robots (forklifts)

3- The Local Area Network (LAN)

The Brain is the main controller of the system, that give orders to the forklifts depending on the production line priorities, the forklifts status and the database of the whole system that it accesses easily.

The forklifts are the executers of the system, the robots that carry on the production lines' objects up and storing them down to the system shelves upon the server's orders.

The Local Area Network (LAN) is the main network that connects the whole system parts together via Wi-Fi wireless connection.

In our system architecture, we have four production lines (A line, B line, C line and D line), each line generates different products or packages from other lines, and three forklifts working simultaneously.

The whole system can be simply illustrated in the following steps;

1- The robots are initially idle, and await a command from the Brain.

2- If a product exists on a production line, the brain searches for all the idle robots, the production line is represented by a destination end point for the robot.

3- Dijkstra algorithm runs on all idle robots to determine the shortest path each robot has to cover to reach the destination point.

4- The robot having the shortest path is chosen and it's state is switched to "Busy" and the path is sent in the form of an array carrying information about directions and Landmarks the robot has to pass through.
5- The robot receives the "shortest path array" through Wi-Fi.
6- The robot starts executing the "shortest path array" as soon as it receives it. It navigates through the environment with the aid of Line tracking modules and maintains information about its location through RFID tags mounted on the floor and carrying information about the corresponding location
7- The robot reaches its destination, scans the Barcode found on the product and sends it to the brain.
8- The brain receives information about the product, assigns a storing point for that package from its data base and checks the storing location flags.
9- The brain runs Dijkstra algorithm for a second time to recalculate the shortest path to the offload location and sends it to the robot to navigate it's environment back to the offload point
10- The process is always repeated during the warehouse working hours.

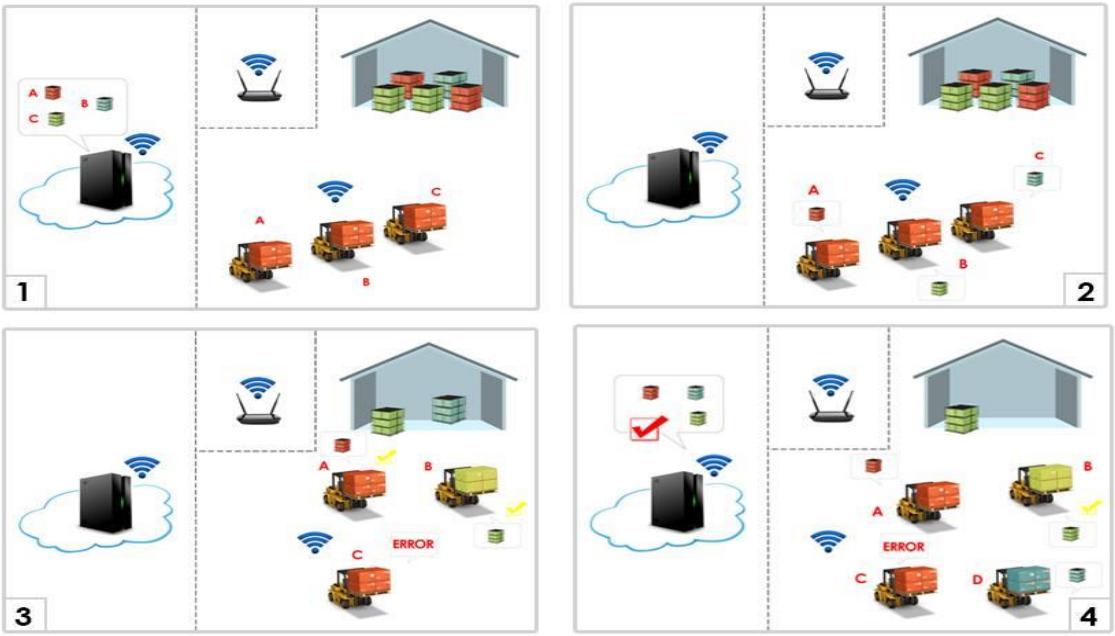


Figure 1.1 System Overview

Chapter 2

"System Components"

In this chapter we will introduce all the system components. Our project prototype consists mainly of three robots (forklifts) with the almost the same characteristics, they should communicate with each other in a way that maintain the system function , avoid collisions and assures high level of security.

We will describe a one typical forklift and the others shall be the same in all characteristics. It will also be illustrated the way of connection among forklifts.

1. System Components Function.

1.1. Arduino Mega

The Arduino Mega is a microcontroller board(kit) based on the ATmega1280. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

It contains everything needed to support the microcontroller.

The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila. See fig. 2.1.

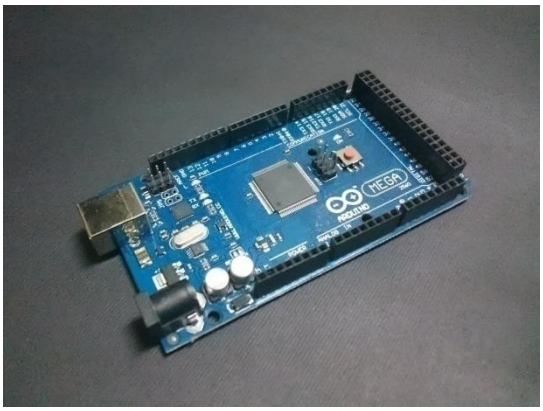


Figure 2.1. Arduino Mega Kit

1.2. Wheels

A **wheel** is a circular component that is intended to rotate on an axle bearing. The wheel is one of the main components of the robots making it able to move and functionalize.

Used wheels have an outer diameter of a 65mm, a width of 28 mm and a brass coupler inner diameter of a 4 mm.

Metal ball caster wheels are also used with the forklifts to make it more controllable while moving. See fig. 2.2.

Figure 2.2. Wheels

1.3. Ultrasonic Sensor

An Ultrasonic sensor detects the distance of the closest object in front of it. It can detect objects along a distance from 3 cm up to 400 cm.

Working by sending out a burst of ultrasound waves and listening for the reflected echo when it bounces off of an object. It pings the obstacles with ultrasound waves.

The Arduino or Genuino board sends a short pulse to trigger the detection, then listens for a pulse on the same pin using the "pulseIn()" function. The duration of this second pulse is equal to the time taken by the ultrasound wave to travel to the object forwardly and come back to the sensor backward. Using the speed of used sound, this time can be converted to distance by a simple calculation. See Figure 2.3.

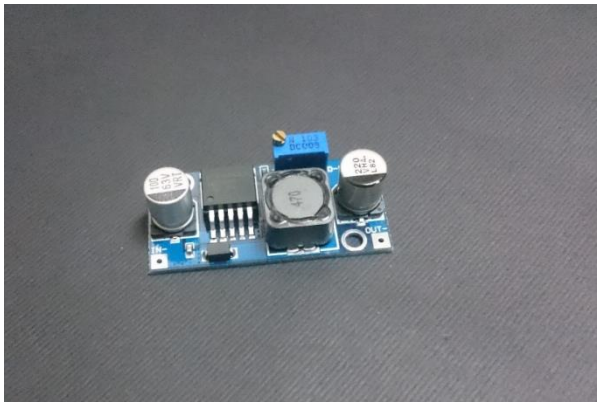


Figure 2.3. Ultrasonic Sensor

1.4. Infrared Reflection Sensor

It detects a specific range between the forklift's arm and the pallet having the required to carry object. See fig. 2.4.

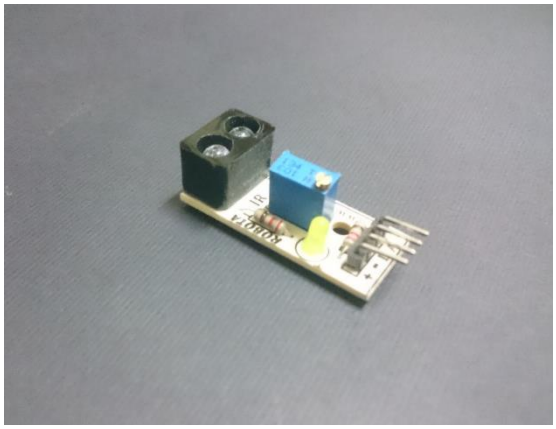


Figure 2.4 Infrared Reflection Sensor

1.5. Tags

Tags are the used indications to have a link with the forklifts. We can consider it as the construction of all possible paths for the forklift to move. Tags help in making specific actions to the forklifts such as making it rotate to the left or to the right.

Mifare-One RFID Tags are used.

1.6. RFID Module Features:

Radio-frequency identification (RFID) is a module uses electromagnetic fields to automatically identify and track the needed tags attached to the required path. The tags contain electronically stored information.

1.6.1. Tags Types

There are two types of tags used;

- Passive tags
- Active tags

Passive tags collect energy from a nearby RFID reader's interrogating radio waves. Where active tags have a local power source such as a battery and may operate at hundreds of meters from the RFID reader.

Unlike a barcode, the tag need not be within the line of sight of the reader, so it may be embedded in the tracked object. RFID is one method for Automatic Identification and Data Capture (AIDC)

Our used tags here are passive.

The RFID has a privilege of having a small dimension which make it easy to embed into your project.

1.6.2. RFID nature

This module is not only RFID 13.5 KHz Read/Write module but also it is NFC module.

NFC (Near Field Technology) is a short-distance radio technology that enables communication between devices that are held close together.

NFC traces its roots in RFID technology and is an open standardized platform technology. It operates at 13.5 MHZ frequency and is widely used like RFID to recognize cards/tags (MIfare Cards / Tags)

The NFC/RFID 13.5 KHz module is based on PN532, which is the most popular NFC chip. This chip is exist in every phone or device that does NFC. Check Fig. 2.5.

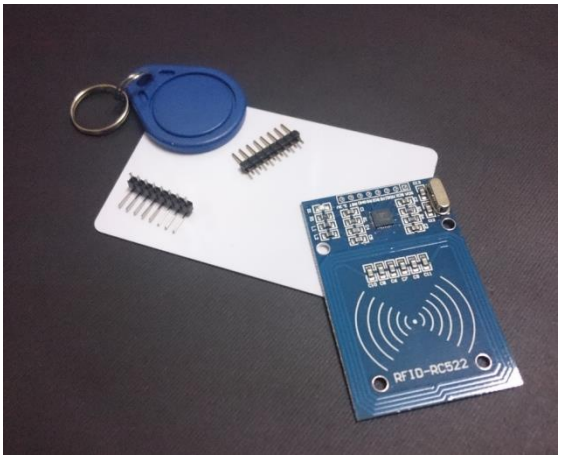


Figure 2.5. RFID Module

1.7. Lithium Polymer Battery

A lithium polymer battery, or more correctly lithium-ion polymer battery (abbreviated variously as **LiPo, LIP, Li-poly** and others), is a rechargeable battery of lithium-ion technology in a pouch format. Unlike cylindrical and prismatic cells, LiPos come in a soft package or pouch, which makes them lighter but also less rigid.

Using a lithium-polymer battery allows the manufacturer to shape the battery in essentially any way they please, as it can be shaped to fit in an array of devices.

In our project, lithium-polymer batteries are used to power up the forklifts.

Figure 2.6. Lithium Polymer Battery

1.8. IMAX B6 Balance Battery Charger

The IMAX B6 is a very advanced charger, able to charge, balance and discharge LiIon, LiPoly, LiFe (A123), NiCd and NiMH batteries. It is microprocessor controlled just like all the best chargers and will balance the individual cells in your Li-XX batteries.

It will charge from 0.1 to 5.0A and is capable of charging packs up to 6S packs. It features input voltage protection so as not to flatten your car battery at the field and will also storage charge you packs for when you are not using them. You can power it with any power supply delivering from 11~18V so it is very flexible. See fig. 2.7.

Figure 2.7. IMAX B6 Balance Charger

The intuitive menu system which means charging and cycling is an easy process and can be done quickly and accurately either in the field, by using a 12V input such as a car battery, or at home with a 12V power supply.

To satisfy mostly any charging requirements for the batteries there are a number of input/output wires connected to the charger to satisfy those needs.

1.9. Servo Motors

Servo Motors are ordinary geared down DC Motors equipped with servo mechanism for precise control of angular position. A servomechanism or servo refers to an error sensing negative feedback which is used to correct the performance of a device. Metal gears allow increase servo motor reliability at high loads, making servo suitable for more critical applications.

See fig. 2.8 and 2.9

Figure 2.8 Servo Motor

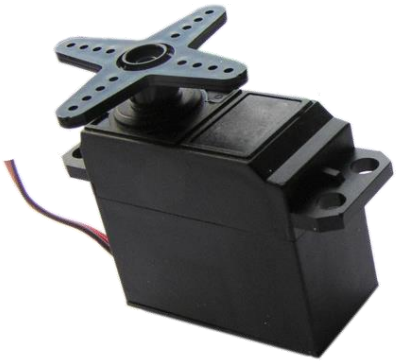


Figure 2.9. Servo Motor

Unlike dc motors, with servo motors we can position the motor shaft at a specific position (angle) using the control signal.

The motor shaft will hold at this position as long as the control signal is not changed. This is very useful for controlling the forklifts (robots) arms, unmanned airplanes control surface or any object required to move at a certain angle and stay at its new position.

The servo motor is mainly used to up and down the forklift's arm carrying and discharging the pallet respectively.

1.10. LM2596HV DC-DC Converter

A servo motor needs around 6 volt charge and a maximum of 3A current to work safely. To adjust the exact needed current and voltages to the servo motors, we use a DC to DC converter. It can also short-circuit with protection limiting self-recovery. See fig. 2.10

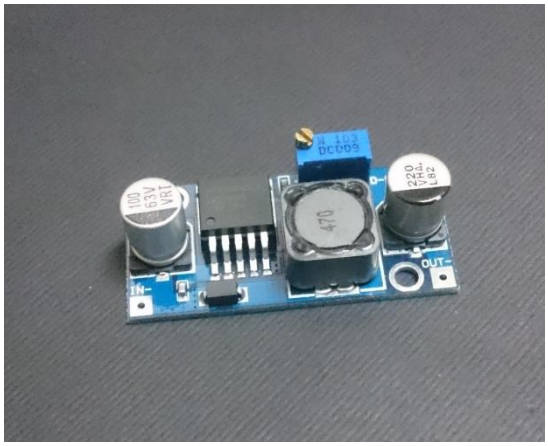


Figure 2.10. LM2596HV DC-DC Converter

1.11. DC motors with encoders

These DC motors are equipped with gear box and encoder. It uses 2 phase or 2 channels (Quadrature) incremental encoders.

Encoder feedback data enables us to control our motor speed and direction efficiently.

DC motors with encoder have many other applications used with robots, its arms, CNC machine and many other applications that require accurate control of speed, direction and position.

Motors can be controlled using a microcontroller or an Arduino kit as well.

1.11.1. Motor and Encoder Function

- **DC Motor**

  DC motor is connected to the forklift wheels, as its used in moving the Forklift and controlling it.

- **Encoder**

The encoder is responsible for detecting the speed of the motor, direction of rotation and the distance it has travelled.

This is all aiming at guaranteeing a stable controlled system. See fig. 2.11

Figure 2.11. DC Motor with Encoder

1.12. DC Motor Driver (10A)

This is a dual motor driver which is designed to drive 2 DC motor with a high current up to 10A continuously and a 30A peak value. The driver works in a voltage range from 5 to 25 Volt. It also includes fast test switch used for driver testing.

We mainly use the DC motor driver to drive the DC Motors with a specific speed as needed in the system.

1.13. Razor IMU 9 Degrees of Freedom (DoF)

1.13.1. IMU Definition

An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers and gyroscopes, sometimes also magnetometers. IMUs are typically used to maneuver aircraft, including unmanned aerial vehicles (UAVs), among many others, and spacecraft, including satellites and landers. Recent developments allow for the production of IMU-enabled GPS devices. An IMU allows a GPS receiver to work when GPS-signals are unavailable, such as in tunnels, inside buildings, or when electronic interference is present**.**

1.13.2. Degrees of Freedom (DoF)

In statistics, the number of degrees of freedom is the number of values in the final calculation of a statistic that are free to vary.

The number of independent ways by which a dynamic system can move, without violating any constraint imposed on it, is called number of degrees of freedom. In other words, the number of degrees of freedom can be defined as the minimum number of independent coordinates that can specify the position of the system completely.

1.13.3. Inside IMU 9 DoF

The 9DOF Razor IMU incorporates three sensors;

- ITG-3200 (MEMS triple-axis gyro)
- ADXL345 (triple-axis accelerometer)
- HMC5883L (triple-axis magnetometer)

For the IMU and in order to provide a nine degrees of inertial measurement. The outputs of the three sensors are processed by an on-board ATmega328 and gives an output over a serial interface.

The IMU is mainly used for localization and orientation of the Forklift.

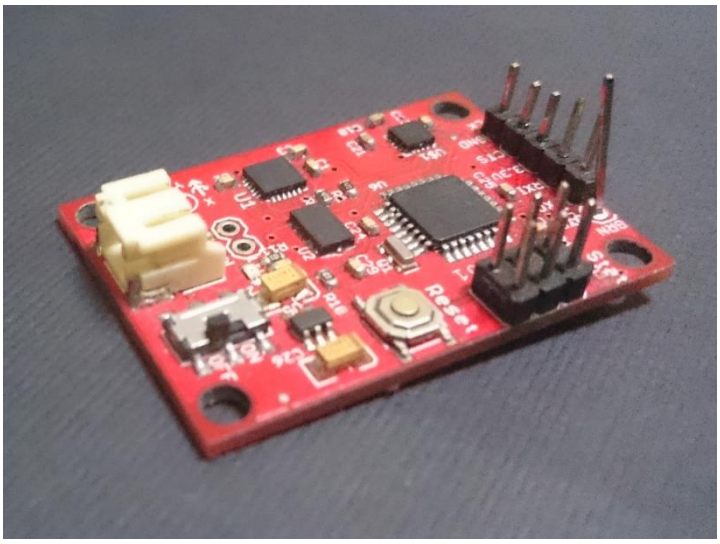


Figure 2.11. IMU 9 DoF

1.14. Power Board

The power board used is usually a Printed Circuit Board (PCB). It's designed in order to protect the components of the forklift from any possible damage can be caused by the high input current or high input voltage using the type of low resistance resistor, fuses.

2. Hardware Components Features

Each component of the previously mentioned ones has different characteristics that

allows it fit to the system. You can find all the system components characteristics, and the quantity used in the project showed in table 2.1

Item name	Item description & Characteristics	Quantity
Arduino Mega	- Based on the ATmega1280. - 54 digital input/output pins (of which 14 can be used as PWM outputs). - 16 analog inputs. - 4 UARTs (hardware serial ports). - A 16 MHz crystal oscillator. - USB connection. - power jack. - ICSP header. - A reset button.	4
Wheels	- Outer diameter of wheel: 65mm - Width: 28mm - Brass coupler inner diameter: 4mm	8

Caster Wheels	Dimensions: 25 X 15 - mm	8
Ultrasonic Sensor	- Ultrasonic ranging module. - Power supply: 5V DC. - Quiescent current: <2mA - Effectual angle: <15° - Ranging distance: 2cm – 400 cm - Resolution: 1 cm - Ultrasonic Frequency: 40k Hz	4
Infrared Reflection Sensor	- Voltage: 5V - Max Current: 300 mA - Range: 10mm - 800mm - Type: NPN - Diameter: 17mm - Length: 45mm - Lead: 450mm	4

RFID Module	- 13.56KHz operating frequency. - The NFC/RFID 13.5 KHz module based on PN532, NFC chip. RFID Module Features: - Small dimension and easy to embed. - Support I2C, SPI and HSU (High Speed UART) modes. Easy to change between those them. - Read/write capability. - Support RFID reading and writing. - P2P communication with peers. - NFC with Android phone. - Up to 5cm~7cm reading distance. - On-board level shifter.	4

	- Standard 5V TTL for I2C and UART, 3.3V TTL SPI.	
Tags	- Operating Frequency: 13.5 kHz - Physical Size: 3.5cm*2.8cm - EEPROM storage: 1K Byte	50
Lithium Polymer Battery	- Low weight /size - High power - Suitable for many applications - Capacity: 5200mAh - Configuration: 3S1P / 11.1v / 3Cell - Peak Discharge : 35C - Pack Weight: 413g - Pack Size: 30.6x44x138 - Connector Type: T connector	4

Battery Charger	- IMAX B6 Balance Charger. - Balances and charges Lithium Polymer, LiFe (A123), NiCd and NiMH batteries. - Handles LiPo/LiFe up to 6S and NiMH/NiCd up to 15S .	1
DC-DC Converter (LM2596HV)	- LM2596HV DC-DC 4.5-55V-1.25-3V 3A Adjustable Power Supply Module. - Module Properties: non-isolated step-down module (buck) - input voltage: 4.5-55V - Output voltage: 1.25-30 adjustable - Output Current: Rated 2A.	8
Servo Motors	- Voltage: 6 -7.2V	

	- Speed 0.16 sec/60 degree (6v), 0.14 sec/60degree (7.2v) - Torque : 12 kg.cm (6v), 14 kg.cm (7.2v), Stall torque. - Dimension: 40.8 x20.1x38 mm - Weight: 56g	8
DC Motors with encoders	**<u>DC Motor</u>** - Voltage: 12 V DC - Output Power: 2.2 W - Gear Ratio: 1.75 - Rated Speed: 133 RPM - Rated Torque: 8.8 N.cm - Stall Current: 3A **<u>Encoder</u>** Type: Hall effect -quadrature encoder 5v. - Size: 25 x 54 mm - Weight: 120 g - Shaft Diameter: 4mm	8

DC motor driver (10A Each)	- Bi-directional control for 2 brushed DC motor. - Supports motor voltage ranges from 5V to 25V. - Speed control PWM frequency up to 20KHz. - Supports both locked-antiphase and sign-magnitude PWM operation. - 2 activation buttons for fast test on each channel.	4
Razor IMU 9 DOF	- 9 Degrees of Freedom on a single, flat board: ❖ ITG-3200 - triple-axis digital-output gyroscope. ❖ ADXL345 - 13-bit resolution, ±16g, triple-axis accelerometer ❖ HMC5883L - triple-axis, digital	4

	magnetometer - Outputs of all sensors processed by on-board ATmega328 and sent out via a serial stream - Output pins match up with FTDI Cable - Bluetooth Mate, XBee Explorer - 3.5-16VDC input - ON-OFF control switch and reset switch - Dimensions: 1.1" x 1.6" (28 x 41mm)	
Power Board	**Description**: PCB board **Functionality:** It's designed in order to protect the components of the forklift from damage with the high input current or high input voltage using	4

	fuses.	

Table 2.1. System Components Features

CHAPTER 3

"Hardware System"

(Robots)

An embedded system is a special-purpose system in which the computer is completely encapsulated by the device it controls. Unlike a general-purpose computer, such as a personal computer, an embedded system performs pre-defined tasks, usually with very specific requirements. Since the system is dedicated to a specific task, design engineers can optimize it, reducing the size and cost of the product. Embedded systems are often mass-produced, so the cost savings may be multiplied by millions of items.

1. Microcontrollers:

A microcontroller is a compact microcomputer designed to govern the operation of embedded systems in motor vehicles, robots, office machines, complex medical devices, mobile radio transceivers, vending machines, home appliances, and various other devices. A typical microcontroller includes a processor, a memory and other peripherals.

1.1. Microcontroller Selection Criteria.

There are several constraints to choosing a Microcontroller for a specific application as follows:

1. Peripherals:

If you want your microcontroller to have built in Ethernet, CAN, USB, or even multiple serial ports, many common choices are going to be eliminated. Some peripherals can be handy to have: UARTs, SPI or I2C controllers, PWM controllers, and EEPROM data memory are good examples, even though similar functionality can frequently be implemented in software or external parts.

It's convenient if output pins can supply reasonable amounts of current for driving LEDs or transistors directly; some chips have 5mA or less drive capability.

2. **Memory:**

The size of memory may be an important consideration. Some micros have as few as 256 instructions and 16 bytes of RAM. You can do a lot in that sort of space (believe it or not), but not everything. Moreover, while you can frequently upgrade to a "bigger chip", some chip families have relatively small memory limits imposed by their architecture, so even the biggest chip you can get will have only 8k instructions (for example.) RAM (data memory) is usually scarce in microcontrollers; some algorithms require substantial RAM to be implemented in a straightforward manner, and it may be worthwhile looking for a micro with a lot of RAM (or external RAM expansion capabilities) if that's the sort of thing you had in mind. (For instance, implementing TCP/IP networking protocols in a micro whose total RAM space is less than used by a typical Ethernet packet is likely to be ... interesting.)

3. **Software tools:**

All of the microcontrollers have some level of standard tools (at least an assembler) provided by the manufacturer. Most of them have "Integrated Development Environments" (IDE) that allow integrated use of an editor with the assembler, some compilers, and a simulator (of perhaps limited value). Some have significant additional support from the open source movement.

Tools from manufacturers aren't always great, but you appreciate them more after you've tried to use a microcontroller that doesn't have any such tools available.

4. Cost

Prices of microcontrollers and their modules vary greatly and therefore there is a need to choose the most convenient microcontroller according to the required application.

2. Automated Guided Vehicle (AGV)

An **automated guided vehicle** or **automatic guided vehicle** (AGV) is a mobile robot that follows markers or wires in the floor, or uses vision, magnets, or lasers for navigation. They are most often used in industrial applications to move materials around a manufacturing facility or warehouse.

In **Smart Warehouse** there was a need to model real life AGVs at which they are capable of maintaining some functions, can be summarized in the following points;

1. Receive request from the brain (server)
2. Reach its destination precisely through several Navigation and Localization techniques
3. Scan the Barcode found on the pallet to ensure matching with requested pallet
4. Carry the pallet back to a predetermined location

5. Avoid obstacles found in the AGV's path
6. Swarm with other robots to select which robot has the shortest path

2.1. AGV Model Realization:

As we defied an automated guided vehicle at this section introduction, we should know how to realize this model for real, and how to choose the suitable specified components to maintain it as a real system.

2.1.1. AGV Microcontroller:

Based on previous inputs, it was found that the best microcontroller to suit the AGV application is the **Arduino Mega** Microcontroller.

The Arduino Mega is a microcontroller board based on the ATmega1280. It contains everything needed to support the microcontroller.

All its specifications and function have been illustrated in Chapter 2 "System Components"

2.1.1.1. Arduino Microcontroller Selection:

Arduino Microcontrollers have a range of advantages and capabilities that make it a very useful illustrative kit to learn many things about embedded systems and more many fields. Those privileges also make it the perfect choice for our project. Those advantages includes the following:

I. Ready to Use:

Arduino Microcontroller comes in a complete package form, which includes the 5V regulator, a burner, an oscillator, a micro-controller, serial

communication interface, LED and headers for connections. It's so easy for programming or any other interface connections. Just plug it into USB port to the computer and that's it. It opens the mind for new techniques and ideas to be implemented.

II. Open Source:

The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, it's possible to add AVR-C code directly into the Arduino programs if it's required to.

III. Effortless Functions:

During coding of Arduino, it's easy to notice some Arduino functions, which make the coding much more easier. Another advantage of Arduino is its automatic unit conversion capability, so during debugging, there is no need to worry about the units conversions. By using the all force on the main parts of the projects.

IV. Large Community:

There are many forums present on the internet in which people are talking about the Arduino. Engineers, hobbyists and professionals are making their projects through Arduino. You can easily find help about everything. Moreover the Arduino website itself explains each and every functions of Arduino.

2.1.2. Chassis:

An Acrylic Chassis was carefully designed in order to fit in all the components in place and accommodate durability, strength and lightweight.

Two wheels back wheels of Outer diameter 65mm and Width of 28mm were used alongside two front metal caster wheels of dimensions 25 X 15 mm.

2.1.3. Motors:

In order to run the system and keep it on working with an efficient performance, motors, motor drivers and power boards were selected very carefully to drive the robot efficiently and safely.

As we mentioned before, we have been mainly using DC and servo motors with all the extra needed devices to supply them and the system.

A servomotor is used to elevate/lower load arms of the forklift for a reliable material handling.

- **An Encoder** has been used, that detects the speed of the motor, direction of rotation and the distance it has travelled.
- A dual motor driver is used in order to drive the DC Motors with a required speed.
- **A** PCB Power board had been made in order to protect the AGV components from damage caused by the high input current or high input voltage using fuses.

- A DC-DC converter is used to adjust the desired voltage and current for servomotors, as they need around 6V and maximum current of 3A to work safely. It can also short-circuit with protection limiting self-recovery.

2.1.4. Lithium Polymer Battery

A stable, reliable power source is needed to supply sufficient power for the components.

High quality Lithium battery is used. Its low weight /size and high power make it suitable for many applications.

3. Navigation and Localization:

The ability to go to a specific locations in one's environment in a timely manner is fundamental in creating a fully autonomous mobile robotic system. To do so, a robot not only has to be able to move about its environment, but it also has to identify its location.

If a system has both localization and navigation, we can command it to go to any goal location in the environment by using the former to compute a route and the latter to execute movements.

3.1. Navigation:

Navigation as a definition is given to be a technique for determining one's position accurately on the surface of earth. From that meaning, it's used here as determining the robot's position on any point of the system.

For any mobile device, the ability to navigate in its environment is important. Avoiding dangerous situations such as collisions and unsafe

conditions comes first, but if the robot has a purpose that relates to specific places in the robot environment, it must find those places.

Determining the robot orientation as well as heading is very important so that the robot is driven precisely in a straight line and therefore avoid colliding with the pallets as well as reading the RFID tags successfully and hence determining the exact robot location.

3.1.1. Methods of Navigation.

Mainly we will discuss two main methods of maintaining Navigation in our system, they are;

3.1.1.1. IMU (9DOF)+PID Controller

3.1.1.2. Line Following

3.1.1.1. IMU (9DOF) + PID Controller

A) The 9DOF Razor IMU incorporates three sensors - an ITG-3200 (MEMS triple-axis gyro), ADXL345 (triple-axis accelerometer), and HMC5883L (triple-axis magnetometer) – in order to provide nine degrees of inertial measurement. An on-board ATmega328 and output process the outputs of all sensors over a serial interface.

B) A PID controller continuously calculates an **error value** as the difference between a desired set point and a measured process variable. The controller attempts to minimize the error over time by adjustment of a **control variable.**

- *P* accounts for present values of the error. For example, if the error is large and positive, the control output will also be large and positive.

- *I* accounts for past values of the error. For example, if the current output is not sufficiently strong, error will accumulate over time, and the controller will respond by applying a stronger action.
- *D* accounts for possible future values of the error, based on its current rate of change.

As a PID controller relies only on the measured process variable, not on knowledge of the underlying process, it is broadly applicable by tuning the three parameters of the model, a PID controller can deal with specific process requirements. The response of the controller can be described in terms of its responsiveness to an error, the degree to which the system overshoots a set point, and the degree of any system oscillation. The use of the PID algorithm does not guarantee optimal control of the system or even its stability.

On trying the previous method, results were not as expected due to the accumulation of error over time as well as unstable readings in the HMC5883L (triple-axis magnetometer) in different positions.

3.1.1.2. Line Following

The simplest/most reliable navigation technique. It is a technique where the robot follows a line, either a black line on a white surface or vice-versa.

The QTR-8RC reflectance sensor array is intended as a line sensor, it is a convenient carrier for eight IR emitter and receiver (phototransistor) pairs evenly spaced at intervals of 0.375″ (9.525 mm).

To use a sensor, you must first charge the output node by applying a voltage to its OUT pin. You can then read the reflectance by withdrawing

the externally supplied voltage and timing how long it takes the output voltage to decay due to the integrated phototransistor. Shorter decay time is an indication of greater reflection.

Line Following Advantages.

This measurement approach has several advantages, especially when coupled with the ability of the QTR-8RC module to turn off LED power:

- No analog-to-digital converter (ADC) is required
- Improved sensitivity over voltage-divider analog output
- Parallel reading of multiple sensors is possible with most microcontrollers
- Parallel reading allows optimized use of LED power enable option

The outputs are all independent, but the LEDs are arranged in pairs to halve current consumption. The LEDs are controlled by a MOSFET with a gate normally pulled high, allowing the LEDs to be turned off by setting the MOSFET gate to a low voltage. Turning the LEDs off might be advantageous for limiting power consumption when the sensors are not in use or for varying the effective brightness of the LEDs through PWM control.

Line following is used in our system for reliability as well as simplicity.

3.2. Localization

The current location of a robot can be determined in several very different ways:

3.2.1. Dead Reckoning:

Dead reckoning uses odometry to measure how far the robot moves. Trigonometry and the equations of kinematics are all that is needed to calculate its new position.

This method has 2 requirements:

- It needs a way to determine its initial location.
- Its accuracy usually decreases over time as every measurement has an error and errors accumulate.

Dead reckoning is often used with other methods to improve the overall system accuracy.

3.2.2. Global Positioning System (GPS):

On large terrains GPS can provide more or less accurate coordinates, dead reckoning and the data from its other sensors can fill in the gaps. However on small terrains or indoors the GPS's inaccuracy becomes a problem and the dead reckoning and sensor data becomes the dominant factor in determining the location.

3.2.3. Landmarks:

Placing landmarks is another way to help the robot to navigate through its environment. Landmarks can be active beacons (IR or sound) or passive (reflectors). Using bar code scanners is another possibility.

3.2.3.1. Land Marking Options:

- QR Code
- FRID

QR codes: It's an abbreviated from **Quick Response Code,** it is the trademark for a type of matrix barcode (or two-dimensional barcode) first designed for the automotive industry in Japan. A barcode is a machine-readable optical label that contains information about the item to which it is attached. A QR code uses four standardized encoding modes (numeric, alphanumeric, byte/binary, and kanji) to efficiently store data; extensions may also be used

QR codes could be used for robot localization, however it isn't used in our system as it needs image processing of a great extent which will consume the Arduino's processing power.

RFID: Radio-Frequency Identification (RFID) is the use of radio waves to read and capture information stored on a tag. A tag can be read from up to several feet away and does not need to be within direct line-of-sight of the reader to be tracked.

A RFID system is made up of two parts: a tag or label and a reader. RFID tags or labels are embedded with a transmitter and a receiver. The RFID component on the tags have two parts: a microchip that stores and processes information, and an antenna to receive and transmit a signal. The tag contains the specific serial number for one specific object.

To read the information encoded on a tag, a two-way radio transmitter-receiver called an interrogator or reader emits a signal to the tag using an antenna. The tag responds with the information written in its memory bank. The interrogator will then transmit the read results to an RFID computer program.

RFID is used in our system due to its simplicity as well as its great functionality. The information about the location is stored on the RFID tags and mounted on the floor, RFID reader is mounted on the robot, when the robot passes through a tag it reads the current position information that is stored on that tag.

4. Obstacle Avoidance:

Obstacle avoidance is the task of satisfying some control objective subject to non-intersection or non-collision position constraints. Obstacle avoidance is very important in our system to maintain great safety all over the system.

Ultra sonic sensor is used to detect the distance of the closest object in front of the sensor (from 3 cm up to 400 cm). It works by sending out a burst of ultrasound and listening for the echo when it bounces off of an object. It pings the obstacles with ultrasound. The Arduino or Genuino board sends a short pulse to trigger the detection, and then listens for a pulse on the same pin using the pulseIn () function. The duration of this second pulse is equal to the time taken by the ultrasound to travel to the object and back to the sensor. Using the speed of sound, this time can be converted to distance.

On detecting a nearby object, a software interrupt is triggered to stop the robot immediately and the robot keeps stationary until the object is removed.

5. Components Arduino Codes

5.1. DC Motors

dc_motors | Arduino 1.6.7

File Edit Sketch Tools Help

dc_motors §

```
void RightFwd (int Speed){
  digitalWrite(LeftDir,LOW);
  analogWrite(LeftPWM,Speed);
}
void RightBwd (int Speed){
  digitalWrite(LeftDir,HIGH);
  analogWrite(LeftPWM,Speed);
}
void LeftFwd (int Speed){
  digitalWrite(RightDir,HIGH);
  analogWrite(RightPWM,Speed);
}
void LeftBwd (int Speed){
  digitalWrite(RightDir,LOW);
  analogWrite(RightPWM,Speed);
}
void LeftStop(){
  digitalWrite(LeftDir,LOW);
  digitalWrite(LeftPWM,LOW);
}
void RightStop(){
  digitalWrite(RightDir,LOW);
  digitalWrite(RightPWM,LOW);
}
```

Figure 3.1. DC Motors Arduino Code

// Motor functions used to facilitate the motor driving to get the motor work it take two values

// the first value is the direction if it's forward it will be low and vice versa

// the second value is the speed from 0: 250

5.2. Infrared Sensor

Infrared_sensor | Arduino 1.6.7

File Edit Sketch Tools Help

Infrared_sensor

```
// the main function of this code is to check if there is an
// in front of the IR sencor or not
//if there is an object there ,it will be a reflection to the IR so it
// will read a value and print it
//if there isn't any objects there, it won't be any reflection then the value will be zero
int sensorPin = 7; //analog pin 0
void setup(){
Serial.begin(9600);
}
void loop(){


int val = digitalRead(sensorPin);
Serial.println(val);
//just to slow down the output - remove if trying to
//catch an object passing by
delay(100);
}
```

Figure 3.2. Infrared Sensor Arduino Code

// the main function of this code is to check the distance between the Forklift and the Pallet.

//if there is an object there, it will be a reflection to the IR so it will read a value and print it

//if there isn't any objects there, it won't be any reflection then the value will be zero

5.3. QTR Sensor

```
#include <QTRSensors.h>

// This example is designed for use with eight QTR-1A sensors. These reflectance sensors should be connected to digital inputs 0 to 7.
// The QTR-8A's emitter control pin (LEDON) can optionally be connected to digital pin 2,
// or you can leave it disconnected and change the EMITTER_PIN #define below from 2 to
// QTR_NO_EMITTER_PIN.

// The main loop of the example reads the raw sensor values .
//


#define NUM_SENSORS             8  // number of sensors used
#define NUM_SAMPLES_PER_SENSOR  4  // average 4 analog samples per sensor reading
#define EMITTER_PIN             2  // emitter is controlled by digital pin 2

// sensors 0 through 5 are connected to analog inputs 0 through 5, respectively
QTRSensorsAnalog qtra((unsigned char[]) {0, 1, 2, 3, 4, 5,6,7},
  NUM_SENSORS, NUM_SAMPLES_PER_SENSOR, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];


void setup()
{
  delay(500);
  Serial.begin(9600); // set the data rate in bits per second for serial data transmission
  delay(1000);
}
```

```
void loop()
{
  // read raw sensor values
  qtra.read(sensorValues);

  // print the sensor values as numbers from 0 to 1023, where 0 means maximum reflectance and
  // 1023 means minimum reflectance
  for (unsigned char i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(sensorValues[i]);
    Serial.print('\t'); // tab to format the raw data into columns in the Serial monitor
  }
  Serial.println();

  delay(250);
}
```

Figure 3.3. QTR Sensor Arduino Code

5.4. IFRD

```
void Rfid()

  if (i==0){
 if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {

if(mfrc522.uid.uidByte[0]==64 && mfrc522.uid.uidByte[1]==77 ){
      if(mfrc522.uid.uidByte[2]==19 && mfrc522.uid.uidByte[3]==51){

  RightStop();
  LeftStop();
 buzzer ();
   delay(1000);
i=2;

}
}
}
  }
}
```

Figure 3.4. RFID Arduino Code

// It’s used in localization, used to follow tags which represent the shortest path

// in the previous code , if it found any tag then the robot will be completely stop and turn on the buzzer.

5.5. Servo Motor

```
void up(){
  servoup.write(120); //t7t 80 120 fo2                // sets the servo position according to the scaled value
  delay(15);
 servoback.write(95);// t7t 80 fo2 95
}

void down(){
   servoup.write(90); //t7t 80 120 fo2                // sets the servo position according to the scaled value
  delay(15);
 servoback.write(80);// t7t 80 fo2 95
}


void lift (){
   buzzer ();
      LeftStop();
    RightStop();
    delay(2000);
up();
 delay(1000);
  buzzer ();
 RotRight_180();
RightStop();
LeftStop();
delay(2000);
}



void download(){
  LeftStop();
   RightStop();
   delay(2000);
   down();
   delay(2000);
   i++;
}
```

Figure 3.5. Servo Motors Arduino Code

// it’s used to up and down the forklift arm as shown in the code we have two Fn [Up, Down]

5.6. Ultrasonic Sensor

```
void timerIsr()
{
    trigger_pulse();                                // Schedule the trigger pulses
    stopRobot();

}

void trigger_pulse()
{
      static volatile int state = 0;                // State machine variable

      if (!(--trigger_time_count))                  // Count to 200mS
      {                                             // Time out - Initiate trigger pulse
         trigger_time_count = TICK_COUNTS;          // Reload
         state = 1;                                 // Changing to state 1 initiates a pulse
      }

      switch(state)                                 // State machine handles delivery of trigger pulse
      {
        case 0:                                     // Normal state does nothing
            break;

        case 1:                                     // Initiate pulse
           digitalWrite(trigPin, HIGH);             // Set the trigger output high
           state = 2;                               // and set state to 2
           break;
```

Figure 3.6. Ultrasonic Sensor Arduino Code

```
        case 2:                                     // Complete the pulse
        default:
           digitalWrite(trigPin, LOW);              // Set the trigger output low
           state = 0;                               // and return state to normal 0
           break;
     }
}
// echo_interrupt() External interrupt from HC-SR04 echo signal.
// Called every time the echo signal changes state.
//
// Note: this routine does not handle the case where the timer
//       counter overflows which will result in the occassional error.
void echo_interrupt()
{
  switch (digitalRead(echoPin))                     // Test to see if the signal is high or low
  {
    case HIGH:                                      // High so must be the start of the echo pulse
      echo_end = 0;                                 // Clear the end time
      echo_start = micros();                        // Save the start time
      break;

    case LOW:                                       // Low so must be the end of hte echo pulse
      echo_end = micros();                          // Save the end time
      echo_duration = echo_end - echo_start;        // Calculate the pulse duration
      break;
  }
}
```

Figure 3.7. Ultrasonic Sensor Arduino Code

// it’s used in obstacle avoidance with the robot as it stop the robot if any object suddenly avoid the Forklift path.

5.7. Line Finder

```
void back() {
  int k=1;
  int val;
 while(k==1){
   buzzer ();
    LeftBwd(52);
    RightBwd(50);
    val = digitalRead(sensorPin);
while (val==1){
   buzzer ();
      LeftBwd(52);
    RightBwd(50);
      val = digitalRead(sensorPin);
while (val==0){
   buzzer ();
    RightStop();
  LeftStop();
  delay(2000);
  val=2;
  k=2;
    Serial.println(val);
}
}

  }


}
```

Figure 3.8. Line Finder Arduino Code

// it’s used in the backward motion for the robot, as it give a feedback for the controller.

CHAPTER 4

"System Connection"

(NETWORK)

The key components of an M2M system include sensors, RFID, a Wi-Fi or cellular communications link and autonomic computing software programmed to help a networked device interpret data and make decisions.

This chapter will concentrate on the networking of the system. The network specifications and techniques to get the system main objects connected which mainly means the forklifts.

1. Preamble

It is required to connect the server (Raspberry pi) with the forklifts (Arduino) using a wireless connection in order to cover the large area of the warehouse as the wired connection with Ethernet cables has less privileges being exposed to be cut or damaged as well as its limited coverage ability for the needed area.

For our network connection we use Wi-Fi protocol, instead of any other communication protocol because we can assign a static Internet Protocol (IP) address to each robot (forklift) which enables the user to send the required data or task to each robot without broadcasting it to all the other robots. Which maintain the system's stability.

Another reason for choosing Wi-Fi protocol is, if there is an extension in the area of warehouse and we need to cover an added space to the system, we can just use wireless repeaters to cover a larger area without changing any configuration in the server or the robots.

2. Networks

A network is a group of two or more computer systems, intelligent ends or devices linked together.

2.1. Network Types

There are many types of computer networks, including the following:

- **local-area networks (LANs):** The computers or devices are geographically close together (that is, in the same building).
- **Wide-area networks (WANs):** The computers or devices are farther apart and are connected by telephone lines or radio waves.
- **Campus-area networks (CANs):** The computers or devices are within a limited geographic area, such as a campus or military base.
- **Metropolitan-area networks MANs):** A data network designed for a town or city.
- **Home-area networks (HANs):** A network contained within a user's home that connects a person's digital devices.

2.2. M2M Network Type

M2M technologies can connect millions of devices within a single network. The range of connected devices includes anything from vending machines to medical equipment to vehicles to buildings. Virtually anything that houses sensor or control technology can be connected to some sort of wireless network.

Essentially, M2M networks are very similar to LAN or WAN networks, but are exclusively used to allow machines, sensors, and controls, to communicate. These devices feed information they collect back to other devices in the network. This process allows a human (or an intelligent control unit) to assess what is going on across the whole network and issue appropriate instructions to member devices.

2.3. Sockets

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

3. Wireless Connection Protocols Overview

Wireless connection can be implemented using many protocols. Each one has specific characteristics that match each application under usage. Before getting closer to the chosen protocol for this project we shall give an overview to some of those protocols. That shall help choosing the suitable protocol knowing precisely what the application needs.

3.1. Bluetooth Protocol

Bluetooth is a wireless technology standard for exchanging data over short distances using short-wavelength UHF radio waves

in the ISM band from fixed and mobile devices, and building personal area networks (PANs).

Bluetooth is a global 2.4 GHz personal area network for short-range wireless communication. Device-to-device file transfers, wireless speakers, and wireless headsets are often enabled with Bluetooth.

3.2. Second Generation Protocol (2G)

2G (or 2-G) is short for second-generation wireless telephone technology. Second-generation 2G cellular telecom networks were commercially launched on the GSM standard in Finland by Radiolinja (now part of Elisa Oyj) in 1991.[1] Three primary benefits of 2G networks over their predecessors were that phone conversations were digitally encrypted; 2G systems were significantly more efficient on the spectrum allowing for far greater mobile phone penetration levels; and 2G introduced data services for mobile, starting with SMS text messages. 2G technologies enabled the various mobile phone networks to provide the services such as text messages, picture messages, and MMS (multimedia messages). All text messages sent over 2G are digitally encrypted, allowing for the transfer of data in such a way that only the intended receiver can receive and read it.

3.3. Third Generation Protocol (3G)

3G, short form of third generation, is the third generation of

mobile telecommunications technology. This is based on a set of standards used for mobile devices and mobile telecommunications use services and networks that comply with the International Mobile Telecommunications-2000 (IMT-2000) specifications by the International Telecommunication Union. 3G finds application in wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV. 3G telecommunication networks support services that provide an information transfer rate of at least 200 kbit/s. Later 3G releases, often denoted 3.5G and 3.75G, also provide mobile broadband access of several Mbit/s to smart phones and mobile modems in laptop computers. This ensures it can be applied to wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV technologies.

3.4. Fourth Generation (4G)

4G, short for fourth generation, 4G is the short name for fourth-generation wireless, the stage of broadband mobile communications that will supercede the third generation (3G)
A 4G system must provide capabilities defined by ITU in IMT Advanced. Potential and current applications include amended mobile web access, IP telephony, gaming services, high-definition mobile TV, video conferencing, 3D television, and cloud computing. Two 4G candidate systems are commercially deployed: the Mobile WiMAX standard (first used in South Korea in 2007), and the first-release Long Term Evolution (LTE)

standard (in Oslo, Norway, and Stockholm, Sweden since 2009).

3.5. Zigbee Protocol

ZigBee is a mesh network specification for low-power wireless local area networks (WLANs) that cover a large area. ZigBee was designed to provide high data throughput in applications where the duty cycle is low and low power consumption is an important consideration. (Many devices that use ZigBee are powered by battery.) Because ZigBee is often used in industrial automation and physical plant operation, it is often associated with machine-to-machine (M2M) communication and the Internet of Things (IoT).

ZigBee is based on the Institute of Electrical and Electronics Engineers Standards Association's 802.15 specification. It operates on the IEEE 802.15.4 physical radio specification and in unlicensed radio frequency bands, including 2.4 GHz, 900 MHz and 868 MHz. The specifications are maintained and updated by the ZigBee Alliance.

3.6. UWB

Ultra wideband (also known as UWB or as digital pulse wireless) is a wireless technology for transmitting large amounts of digital data over a wide spectrum of frequency bands with very low power for a short distance. Ultra wideband radio not only can carry a huge amount of data over a distance up to 230 feet at very low power (less than 0.5 milliwatts), but has the ability to carry

signals through doors and other obstacles that tend to reflect signals at more limited bandwidths and a higher power. Ultra wideband can be compared with another short-distance wireless technology, Bluetooth, which is a standard for connecting handheld wireless devices with other similar devices and with desktop computers.

As shown in table 4.1. a simple comparison illustrating the differences among Bluetooth, UBW, Zigbee and WiFi technologies. Also figure 4.2. Shows the relationship between transmission time and data payload size for the four mentioned technologies

TYPICAL SYSTEM PARAMETERS OF THE WIRELESS PROTOCOLS

Standard	Bluetooth	UWB	ZigBee	Wi-Fi
IEEE Spec.	802.15.1	802.15.3	802.15.4	802.11a/b/g
Max data rate (Mbit/s)	0.72	110*	0.25	54
Bit time (μ s)	1.39	0.009	4	0.0185
Max data payload (bytes)	339 (DH5)	2044	102	2312
Max overhead (bytes)	158/8	42	31	58
Coding efficiency+ (%)	94.41	97.94	76.52	97.18
* Unapproved 802.15.3a.	+ Where the data is 10K bytes.			

Table 4.1. Wireless Communication Technologies Differences

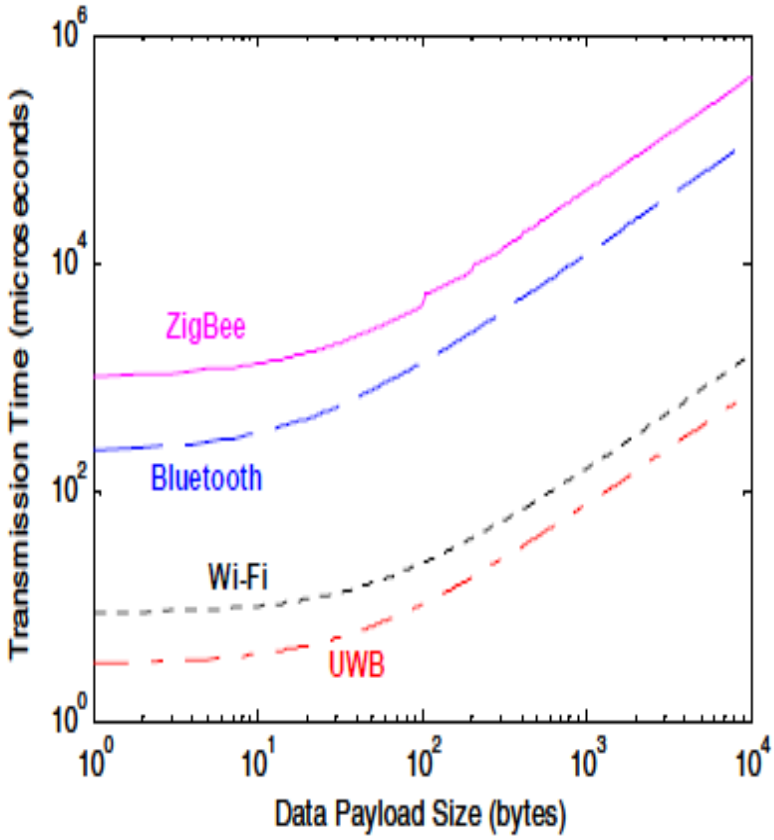


Figure 4.1. Wireless Communication Technologies Characteristics

From the above read sections it's obvious that the Bluetooth wireless communication technology could be the closest to the selected technology for the project "Wi-Fi". Both technologies stand as wireless communication protocols and even share some frequencies under the ISM band. However, significant numbers of differences are there between these two technologies in terms of range, power consumption, applications etc. Maximum operating range for Bluetooth is 10m while that for Wi-Fi is around 100m.Bluetooth provides easy pairing up between devices and consume less power but tradeoffs are made in terms of security and data exchange speeds. On the other hand, Wi-Fi set-ups are

complicated and consumer higher amount of power but are secure and provide higher data exchange speeds.

4. Wi-Fi Technology.

Wi-Fi is the name of a popular wireless networking technology that uses radio waves to provide wireless high-speed Internet and network connections. A technology that allows electronic devices to connect to a wireless LAN (WLAN) network, mainly using the 2.4 gigahertz (12 cm) UHF and 5 gigahertz (6 cm) SHF ISM radio bands. A WLAN is usually password protected, but may be open, which allows any device within its range to access the resources of the WLAN network.

A common misconception is that the term Wi-Fi is short for "*wireless fidelity*," however this is not the case. Wi-Fi is simply a trademarked phrase that means *IEEE 802.11x*.

4.1. Wi-Fi Alliance

Wi-Fi Alliance is an organization made up of leading wireless equipment and software providers with the missions of certifying all 802.11-based products for interoperability and promoting the term Wi-Fi as the global brand name across all markets for any 802.11-based wireless LAN products.

While all 802.11a/b/g products are called Wi-Fi, only products that have passed the Wi-Fi Alliance testing are allowed to refer to their products as "Wi-Fi Certified" (a registered trademark).

Products that pass are required to carry an identifying seal on their packaging that states "Wi-Fi Certified" and indicates the radio frequency band used (2.5GHz for 802.11b or 11g, 5GHz for 802.11a)

The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network" (WLAN) product based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards. However, the term "Wi-Fi" is used in general English as a synonym for "WLAN" since most modern WLANs are based on these standards. "Wi-Fi" is a trademark of the Wi-Fi Alliance. The "Wi-Fi Certified" trademark can only be used by Wi-Fi products that successfully complete Wi-Fi Alliance interoperability certification testing.

4.2. IEEE 802.11 Standard

IEEE 802.11 is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication in the 900 MHz and 2.4, 3.6, 5, and 60 GHz frequency bands. They are created and maintained by the Institute of Electrical and Electronics Engineers (IEEE) LAN/MAN Standards Committee (IEEE 802). The base version of the standard was released in 1997, and has had subsequent amendments. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand. While each amendment is officially revoked when it is incorporated in the latest version of the standard .

Across all variations of 802.11, maximum achievable throughputs are given either based on measurements under ideal conditions or in the layer-2 data rates. This, however, does not apply to typical deployments in which data is being transferred between two endpoints, of which at least one is typically connected to a wired infrastructure and the other endpoint is connected to an infrastructure via a wireless link.

4.3. IEEE 802.11x History

In 1971, ALOHAnet connected the Hawaiian Islands with a UHF wireless packet network. ALOHAnet and the ALOHA protocol were early forerunners to Ethernet, and later the IEEE 802.11 protocols, respectively.

A 1985 ruling by the U.S. Federal Communications Commission released the ISM band for unlicensed use. These frequency bands are the same ones used by equipment such as microwave ovens and are subject to interference.

In 1991, NCR Corporation with AT&T Corporation invented the precursor to 802.11, intended for use in cashier systems. The first wireless products were under the name WaveLAN.

The Australian radio-astronomer Dr John O'Sullivan with his colleagues Dr Terrence Percival AM, Mr Graham Daniels, Mr Diet Ostry, Mr John Deane developed a key patent used in Wi-Fi as a by-product of a Commonwealth Scientific and Industrial

Research Organisation (CSIRO) research project, "a failed experiment to detect exploding mini black holes the size of an atomic particle". In 1992 and 1996, CSIRO obtained patents for a method later used in Wi-Fi to "unsmear" the signal.

The first version of the 802.11 protocol was released in 1997, and provided up to 2 Mbit/s link speeds. This was updated in 1999 with 802.11b to permit 11 Mbit/s link speeds, and this proved to be popular.

In 1999, the Wi-Fi Alliance formed as a trade association to hold the Wi-Fi trademark under which most products are sold.
The term Wi-Fi, commercially used at least as early as August 1999, was coined by brand-consulting firm Interbrand Corporation. The Wi-Fi Alliance had hired Interbrand to determine a name that was "a little catchier than 'IEEE 802.11b Direct Sequence'". Phil Belanger, a founding member of the Wi-Fi Alliance who presided over the selection of the name "Wi-Fi", also stated that Interbrand invented Wi-Fi as a play on words with hi-fi, and also created the Wi-Fi logo.

The Wi-Fi Alliance used the "nonsense" advertising slogan "The Standard for Wireless Fidelity" for a short time after the brand name was invented, leading to the misconception that Wi-Fi was an abbreviation of "Wireless Fidelity". The yin-yang Wi-Fi logo indicates the certification of a product for interoperability.

Non-Wi-Fi technologies intended for fixed points, such as Motorola Canopy, are usually described as fixed wireless. Alternative wireless technologies include mobile phone standards, such as 2G, 3G, 4G or LTE.

The name is often written as WiFi or Wifi, but these are not approved by the Wi-Fi Alliance.

4.4. Wi-Fi Frequencies

A wireless network will transmit at a frequency level of 2.4 GHz or 5GHz to adapt to the amount of data that is being sent by the user. The 802.11 networking standards will somewhat vary depending mostly on the user's needs.

4.4.1. 802.11a Standard

This standard will transmit data at a frequency level of 5GHz. The Orthogonal Frequency-Division Multiplexing (OFDM) used enhances reception by dividing the radio signals into smaller signals before reaching the router. You can transmit a maximum of 54 megabits of data per second.

4.4.2. 802.11b Standard

That will transmit data at a frequency level of 2.4GHz, which is a relatively slow speed. You can transmit a maximum of 11 megabits of data per second.

4.4.3. 802.11g Standard

It will transmit data at 2.4GHz but can transmit a maximum of 54 megabits of data per second as it also uses an OFDM coding.

4.4.4. The more advanced 802.11n Standard

This standard can transmit a maximum of 140 megabits of data per second and uses a frequency level of 5GHz.

4.5. Wi-Fi Frequency Band

802.11b, 802.11g, and 802.11n-2.4 utilize the 2.400–2.500 GHz spectrum, one of the ISM bands. 802.11a and 802.11n use the more heavily regulated 4.915–5.825 GHz band. These are commonly referred to as the "2.4 GHz and 5 GHz bands" in most sales literature. Each spectrum is sub-divided into channels with a center frequency and bandwidth, analogous to the way radio and TV broadcast bands are sub-divided.

The 2.4 GHz band is divided into 14 channels spaced 5 MHz apart, beginning with channel 1, which is centered on 2.412 GHz. The latter channels have additional restrictions or are unavailable for use in some regulatory domains as shown in figure 4.2.

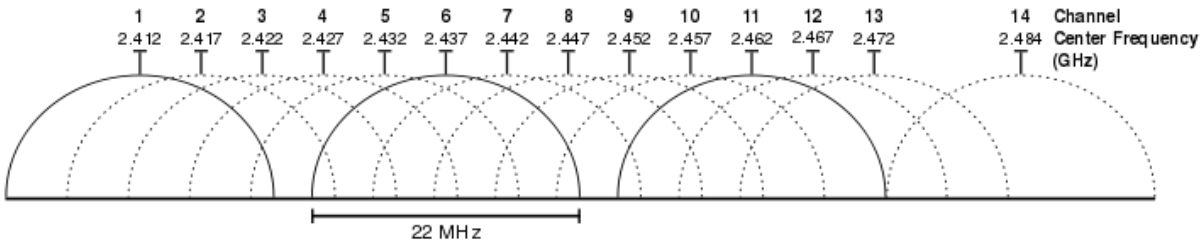


Figure 4.2. Graphical representation of Wi-Fi channels in the 2.4 GHz band

The channel numbering of the 5.725–5.875 GHz spectrum is less intuitive due to the differences in regulations between countries. These are discussed in greater detail on the list of WLAN channels.

5. How Wi-Fi Works

Wi-Fi works with no physical wired connection between sender and receiver by using radio frequency (RF) technology -- a frequency within the electromagnetic spectrum associated with radio wave propagation. When an RF current is supplied to an antenna, an electromagnetic field is created that then is able to propagate through space.

The cornerstone of any wireless network is an access point (AP). The primary job of an access point is to broadcast a wireless signal that computers can detect and "tune" into. In order to connect to an access point and join a wireless network, computers and devices must be equipped with wireless network adapters.

5.1. Access Point

5.1.1. Access Point Definition

In a wireless local area network (WLAN), an access point is a station that transmits and receives data (sometimes referred to as a transceiver). An access point connects users to other users within the network and also can serve as the point of interconnection

between the WLAN and a fixed wire network. Each access point can serve multiple users within a defined network area; as people move beyond the range of one access point, they are automatically handed over to the next one. A small WLAN may only require a single access point; the number required increases as a function of the number of network users and the physical size of the network.

5.1.2. Access Point Utilization

Typical corporate use involves attaching several WAPs to a wired network and then providing wireless access to the office LAN. The wireless access points are managed by a WLAN Controller which handles automatic adjustments to RF power, channels, authentication, and security. Furthermore, controllers can be combined to form a wireless mobility group to allow inter-controller roaming. The controllers can be part of a mobility domain to allow clients access throughout large or regional office locations. This saves the clients time and administrators overhead because it can automatically re-associate or re-authenticate.

5.1.3. Hotspot Access Point Application

A hotspot is a common public application of WAPs, where wireless clients can connect to the Internet without regard for the particular networks to which they have attached for the moment. The concept has become common in large cities, where a combination of coffeehouses, libraries, as well as privately owned

open access points, allow clients to stay more or less continuously connected to the Internet, while moving around. A collection of connected hotspots can be referred to as a lily pad network.

6. Data Exchange in Wi-Fi.

As shown in fig. 4.3. Data exchange in Wi-Fi can be summarized into three phases.

6.1. Data Exchange Phases

6.1.1. Phase I

Phase one starts when data is prepared for transmission; it is encoded; changed into frames. Digital signals are sent in frames for better quality of service (QoS). The frequency for data transmission is also chosen depending upon the technique used to send the signals wirelessly.

6.1.2. Phase II

Phase two begins where data is transmitted in the air as the medium of wave transmission.

6.1.3. Phase III

When the data is received, phase three starts, then data is also decoded, acknowledged and then used.

All of these phases apply some of the popular digital communications spread spectrum techniques for signal multiplexing (FHSS, Infrared, OFDM etc.), make use of security methods (WEP, WPA). Let's find out the technical insides of the Wi-Fi legacy.

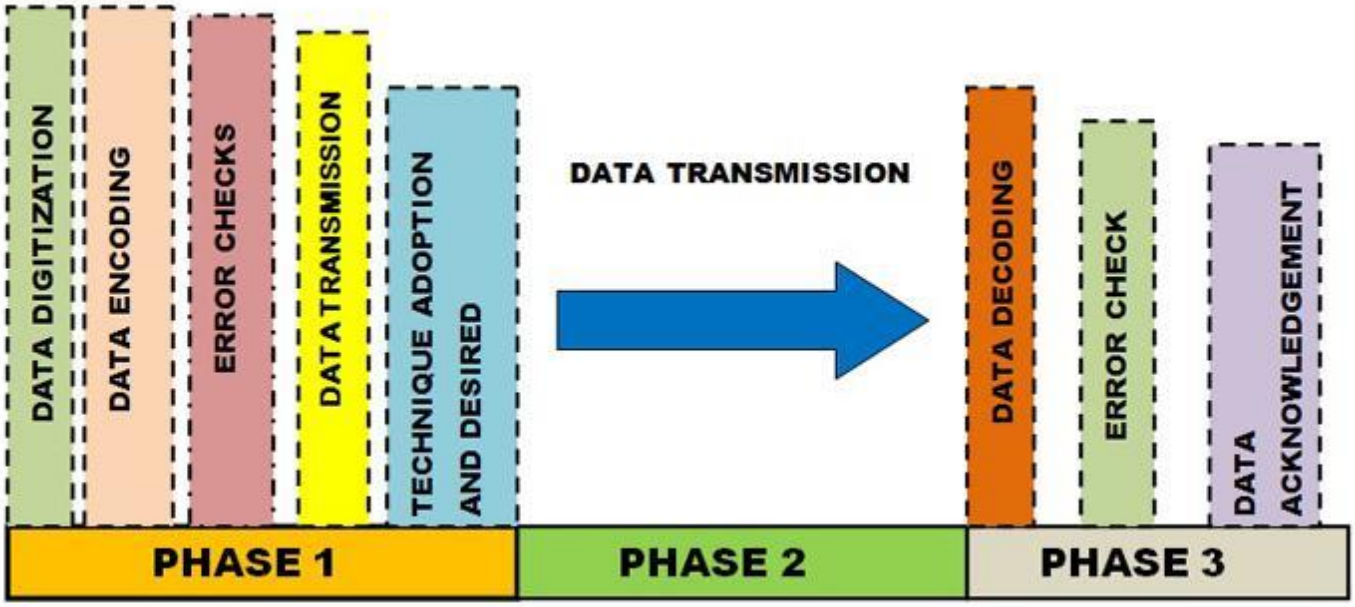


Figure 4.3. Wi-Fi Data Exchange

6.2. Exchanged Data Format

Data exchanged and transmitted through Wi-Fi is in the form of datagram.

The datagrams are called frames. Current 802.11 standards specify frame types for use in transmission of data as well as management and control of wireless links.

Frames are divided into very specific and standardized sections. Each frame consists of a MAC header, payload, and frame check sequence (FCS). Some frames may not have a payload.

6.3. MAC Header and Payload

The first two bytes of the MAC header form a frame control field specifying the form and function of the frame. This frame control field is subdivided into the following sub-fields:

1. **Protocol Version**

   Two bits representing the protocol version. Currently used protocol version is zero. Other values are reserved for future use.

2. **Type**

   Two bits identifying the type of WLAN frame. Control, Data, and Management are various frame types defined in IEEE 802.11.

3. **Subtype**

   Four bits providing additional discrimination between frames. Type and Sub Type are used together to identify the exact frame.

4. **ToDS** and **FromDS**

   Each is one bit in size. They indicate whether a data frame is headed for a distribution system. Control and management frames set these values to zero. All the data frames will have one of these bits set. However communication within an Independent Basic Service Set (IBSS) network always set these bits to zero.

5. **More Fragments**

The More Fragments bit is set when a packet is divided into multiple frames for transmission. Every frame except the last frame of a packet will have this bit set.

6. **Retry**

Sometimes frames require retransmission, and for this there is a Retry bit that is set to one when a frame is resent. This aids in the elimination of duplicate frames.

7. **Power Management**

This bit indicates the power management state of the sender after the completion of a frame exchange. Access points are required to manage the connection, and will never set the power-saver bit.

8. **More Data**

The More Data bit is used to buffer frames received in a distributed system. The access point uses this bit to facilitate stations in power-saver mode. It indicates that at least one frame is available, and addresses all stations connected.

9. **Protected Frame**

The Protected Frame bit is set to one if the frame body is encrypted by a protection mechanism such as Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA), or Wi-FI

Protected Access II (WPA2).

10. **Order**

This bit is set only when the "strict ordering" delivery method is employed. Frames and fragments are not always sent in order as it causes a transmission performance penalty.

The next two bytes are reserved for the Duration ID field. This field can take one of three forms: Duration, Contention-Free Period (CFP), and Association ID (AID).

An 802.11 frame can have up to four address fields. Each field can carry a MAC address. Address 1 is the receiver, Address 2 is the transmitter, Address 3 is used for filtering purposes by the receiver.

<u>The remaining fields of the header are:</u>

11. The Sequence Control field is a two-byte section used for identifying message order as well as eliminating duplicate frames. The first 4 bits are used for the fragmentation number, and the last 12 bits are the sequence number.
12. An optional two-byte Quality of Service control field that was added with 802.11e.

The payload or frame body field is variable in size, from 0 to 2304 bytes plus any overhead from security encapsulation, and contains information from higher layers.

6.4. Frame Check Sequence

The Frame Check Sequence (FCS) is the last four bytes in the standard 802.11 frame. Often referred to as the Cyclic Redundancy Check (CRC), it allows for integrity check of retrieved frames. As frames are about to be sent, the FCS is calculated and appended. When a station receives a frame, it can calculate the FCS of the frame and compare it to the one received. If they match, it is assumed that the frame was not distorted during transmission.

7. ESP8266 Wi-Fi Module.

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to our WiFi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. This module also comes pre-programmed with an AT command set firmware, meaning, it's easily simple to hook this up to the Arduino device and get about as much WiFi-ability as a WiFi Shield offers. The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

7.1. WT8266-S1 Wi-Fi Module

As shown in fig. 4.4. WT8266-S1 FCC /CE / RoHS / Wi-Fi Module based on ESP8266 is a low consumption, high performance embedded Wi-Fi network control module designed by Wireless-Tag. It can meet the IOT application requirements in smart power grids, building automation, security and protection, smart home, remote health care etc.

The module's core processor ESP8266 integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor with smaller package size and 16 bit compact mode, main frequency support 80 MHz and 160 MHz, support RTOS, integrated Wi-Fi MAC / BB / RF / PA / LNA, on-board PCB antenna.

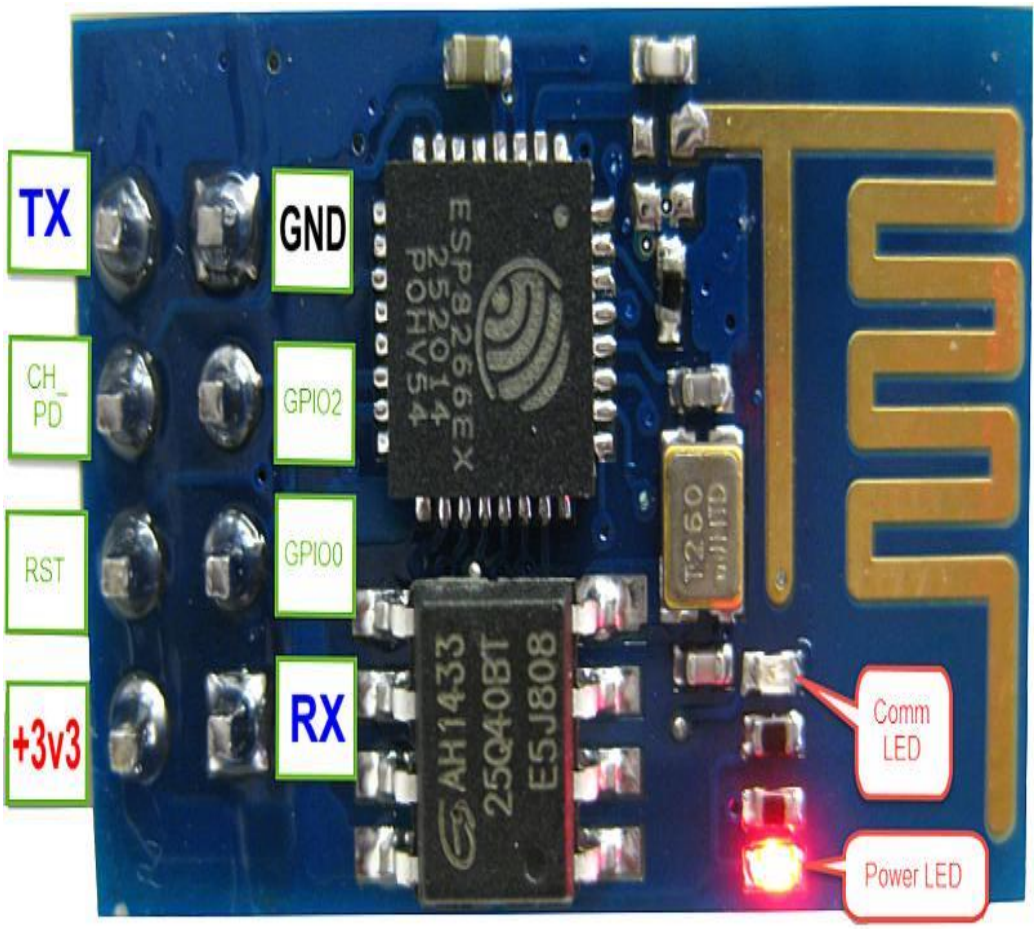


Fig. 4.4. WT8266-S1 Wi-Fi Module

7.2. WT8266-S1 Module Features.

Stated below will be the features of Wi-Fi module of ESP8266 family, used in the system.

- Operating Voltage: 3.3V
- Operating Temperature：-40°C -85°C
- CPU Tensilica L106
- RAM 36KB（Available）
- Flash 32Mbit

System:

- 802.11 b/g/n
- Integrated Tensilica L106 ultra-low power 32-bitmicro MCU, with 16 bit compact mode, main frequency support 80 MHz and 160 MHz, support RTOS.
- WIFI 2.4 GHz,support WPA/WPA2
- Ultra-Small 18.6mm*15.0mm
- Integrated 10 bit high precision ADC
- Integrated TCP/IP Stack
- Integrated TR switch, balun, LNA, Power amplifier and matching network
- Integrated PLL、Regulator and power source management components, +20 dBm output power in 802.11b mode
- Supports antenna diversity
- Deep sleep current < 20uA, Power down leakage current < 5uA
- It can use an application processor SDIO 2.0、SPI、 UART
- Polymerization of STBC、 1x1 MIMO、 2x1 MIMO A-MPDU 、A-MSDU and 0.4 sguard interval
- Wake up within 2ms, connect and transfer data packets
- Standby power consumption < 1.0mW (DTIM3)
- Support AT remote upgrades and cloud OTA upgrade
- Support STA/AP/STA+AP operation modes

Feature summary:

- Distance 280meters--3 Bars, 350meters no prolem. Up to 400 meters
- Industrial temp grade
- CE, FCC, RoHs compliance
- AT command
- Transparent transmission
- Ultra-low power consumption <20uA @ deep sleep mode
- 32M flash

8. Wi-Fi Module Hardware Connection

The hardware connections required to connect to the ESP8266 module are fairly straight-forward but there are a couple of important items to note related to power, should be considered wile connecting.

- The ESP8266 requires 3.3V power–do not power it with 5 volt
- ESP8266 needs to communicate via serial at 3.3 V not having a tolerant inputs of 5V. So, a level converter is needed while communicating with a 5V microcontroller like most Arduino kits.

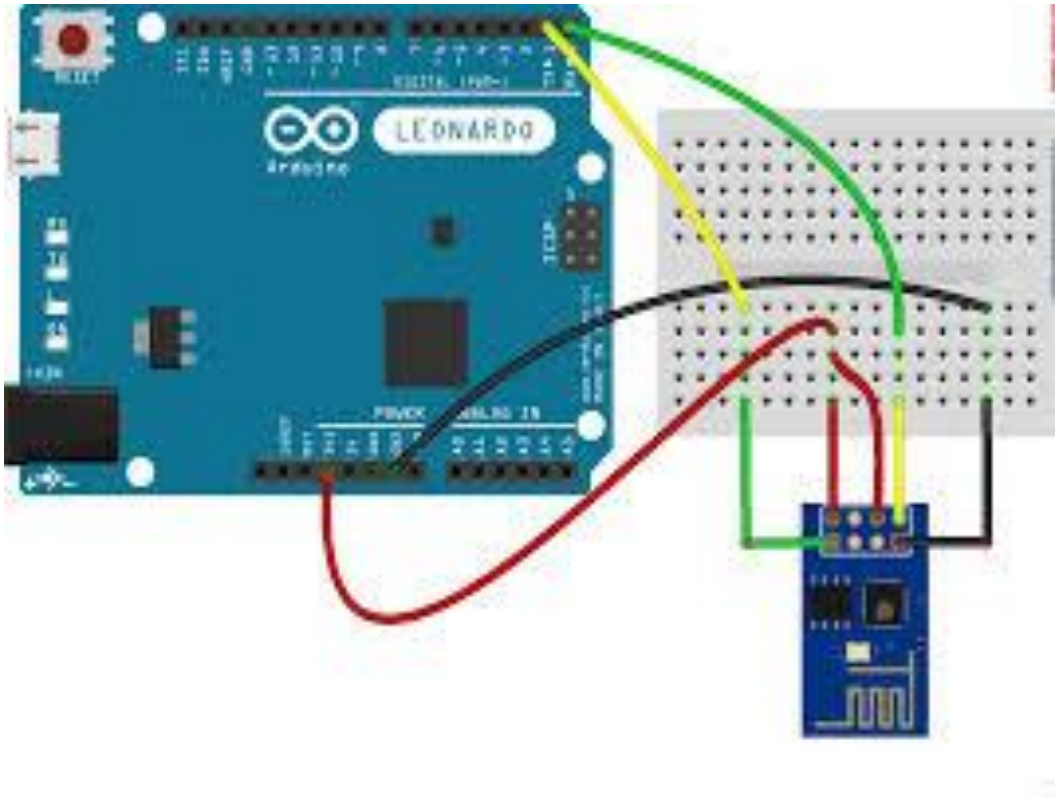


Figure 4.5. ESP8266 Module Hardware Connection.

8.1. Arduino Code Description.

```
void setup() {
  // put your setup code here, to run once:
  Serial3.begin(115200);
  Serial.begin(115200);
  pinMode(13, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  char temp;
  String msg;
  data[];
while (Serial3.available() > 0)
  {

    temp = Serial3.read();
    delay(50);
    msg += String (temp);
    Serial.println(msg);


    parseCommand(msg);
     data = msg.substring(msg.indexOf("#"));
    Serial.println(data);




    Serial.println(data);


  }


  }
  void parseCommand(String com ) {

}
```

```
// Import required libraries
#include<ESP8266WiFi.h>
// WiFi parameters
const char* ssid = "wifi";
const char* password = "hk1234567";
// Host
const char* host = "192.168.1.4";
void setup() {
  // Start Serial
  Serial.begin(115200);
  delay(10);

  // We start by connecting to a WiFi network
  Serial.println();
  Serial.println();
  //Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
 Serial.println(WiFi.localIP());
}
int value = 0;
void loop() {
int x=3;




  //Serial.print("Connecting to ");
  //Serial.println(host);
  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 8080;
  if (!client.connect(host, httpPort)) {
    //Serial.println("connection failed");
    return;
  }
  // This will send the request to the server
  client.print(String("GET /") + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(1000);
```

However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible.

CHAPTER 5

"WI-FI Network Connection and Configuration"

1. Raspberry Pi Kit.

The **Raspberry Pi** is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation with the intent to promote the teaching of basic computer science in developing countries. The Foundation of Raspberry Pi provides Debian and Arch Linux ARM distributions for download, and promotes Python as the main programming language of Raspberry Pi, the language we use to code the kit with support for BBC BASIC (via the RISC OS image or the Brandy Basic clone for Linux). C, C++, PHP, Java, Perl, Ruby, Squeak Smalltalk and more programming languages are also available.

In order to have better performance and high speed for transmitting and receiving data between the server and the robots, we used Raspberry Pi kits to establish the required Wi-Fi connection of the system. We configured the Raspberry Pi on the robots and on the server as well to make the Wi-Fi communication connection coded in python programming language to establish communication session between them .

Connection with raspberry pi is slightly different from Arduino connection, as Raspberry Pi runs raspbian operating system and can establish connection with different ways using system configuration or by running codes to make the connection. So, as mentioned previously, we may use much more programming languages than python .

The Raspberry Pi is a popular choice when developing IoT

products. It offers a complete Linux server with a tiny platform at an incredibly low price.

Actually, the Raspberry Pi is so well-known to IoT that the company has featured several Internet of Things projects on their site. It's also Easily programmed and trouble-shoted

1.1. Software Connection

It is required to connect the server (Raspberry pi) with the robots (Arduino) with wireless connection in order to cover the large area of the warehouse as the wired connection with Ethernet cables could be cut and can cover only limited area opposing the wireless connection.

We have two different codes. A code for the server kit and another for the robot or the forklift.

1. Server code :

For the kit on the server. Its function is to transmit and receive data from robots , then put data in the database of the server, upon this data the server will send instructions to the robots to take actions

```
import socket

from _socket import AF_INET
from _socket import SOCK_STREAM

s=socket.socket(AF_INET,SOCK_STREAM)

#host=socket.gethostbyname("www.google.com")
```

```
s.connect(("192.168.1.6",7122))  # make it as tuple (host,80)
message="123,456,789,101112"

s.sendall(message)

data=s.recv(1000)
print(data)
s.close()
```

Here in this code we import the library of "socket" which enables the connection between any two IPs as if they are sockets.

* s.connect(("192.168.1.6",7122))

At the argument, we assign the IP address that the message will be sent to. At the second argument it's assigned the port number needed to be used.

* message="123,456,789,101112"

At this command we put the message we want to send “position , tasks , error , target and destination “

* s.sendall(message)

Is sending the message command.

* data=s.recv(1000)

Is an acknowledgment message sent by the client indicating that the message has been received.

* s.close()

The socket is closed.

2. Client and I2C code :

For the Robots kit. Its function is to transmit and receive data to and from the server .

```
importsocket

importsmbus

importtime

bus= smbus.SMBus(1)



# This is the address we setup in the Arduino Program

address= 0x04



defwriteNumber(value):

bus.write_byte(address, value)

# bus.write_byte_data(address, 0, value)
```

```
return-1


defreadNumber():

number= bus.read_byte(address)

# number = bus.read_byte_data(address, 1)

returnnumber

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

s.bind(("",61))

s.listen(5)


whileTrue:


conn,addr=s.accept()

data=conn.recv(1000)

if not data:

Break


printdata

var=[]

   x=[]
```

```
conn.close()

   n=data.count(',')+1

data=data.split(',')

printn

fori in range (0,n):

x=int(data[i])

var= x

if not var:

continue

time.sleep(1)

writeNumber(var)

print"RPI: Hi Arduino, I sent you ", var

time.sleep(1)
```

* s.bind((",7122))

At this command, the raspberry pi listens to any IP using the port in the second argument.

* s.listen(5)

Here, the raspberry pi listens to only 5 users at a time.

* while (true):

We check if there is any data received from the server. If yes, the code will enter to the while loop. If else, the code shall close the connection (socket).

* conn,addr=s.accept()
* data=conn.recv(1000)

Here we accept the received request and permit the reception of 1000 characters.

* conn.sendall(data)

At this command, the client sends an acknowledgment message indicating the reception of the message.

1.2. Hardware Connection

We used Wi-Fi dogle which is a USB Wi-Fi module that can be easily attached to the raspberry pi and provide Wi-Fi connectivity to it.

2. Raspbian operating system

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run.

3. Network Connection Requirements

To start building our network we will state some prerequisites and equipment that would be needed.

For establishment we will need the following;

- A Raspberry Pi.
- A USB WiFi Adapter (we used the Edimax – Wireless 802.11b/g/n nano USB adapter – its small and cheap).
- A SD Card flashed with the Raspbian OS.
- Access to the Raspberry Pi kit either via a keyboard and a monitor or remotely.

Figure 5.1. Edimax – Wireless 802.11b/g/n nano USB adapter

Connection Steps

- We Plug the USB WiFi adapter into one of the free USB ports on the Raspberry Pi.

- Power up the Raspberry Pi –at this point the WiFi adapter does not work yet. We need some other means of being able to control the Raspberry Pi either via a keyboard or remotely using a wired network connection.
- After booting and logging-in make sure that the Raspberry Pi found your new wireless adapter.
- To look at which peripherals the operating system found when booting run the following command:

  dmesg | more

- You can use the spacebar to scroll down a page at a time – towards the end you will see something similar to the following lines:

  [3.282651] usb 1-1.2: new high-speed USB device number 4 using dwc_otg
  [3.394810] usb 1-1.2: New USB device found, idVendor = 7392, idProduct = 7811
  [3.407489] usb 1-1.2: New USB device strings: Mfr = 1, Product = 2, SerialNumber = 3
  [3.420530] usb 1-1.2: Product: 802.11n WLAN Adapter

  This means that the operating system recognized the USB WiFi Adapter using one of the built-in drivers (you can return to the terminal by pressing “q”).

- All that is left is to configure your WiFi connection.

4. Wi-Fi Network Configuration

4.1. Method One

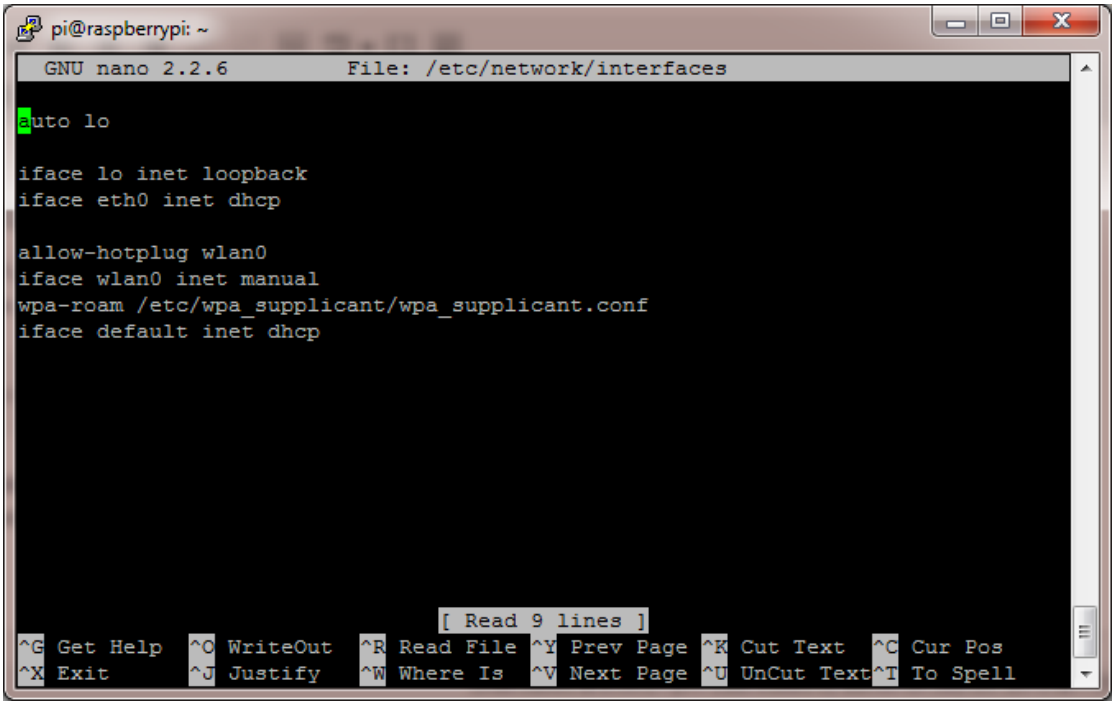
1. On the Raspberry Pi (and on Linux in general) we configure the network settings in the file “/etc/network/interfaces”.

2. This file can be edited using the following command:

sudo nano /etc/network/interfaces

3. This will open the file in an editor called **nano**. It is a very simple text editor that is easy to approach and use; even for users who are not familiar to a linux based operating systems would just use the arrow keys.

4. After opening the file in **nano** it will appear a screen like this:



5. To configure the wireless network, we shall modify the file such that it looks like the following:

auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0

auto wlan0

iface wlan0 inet dhcp

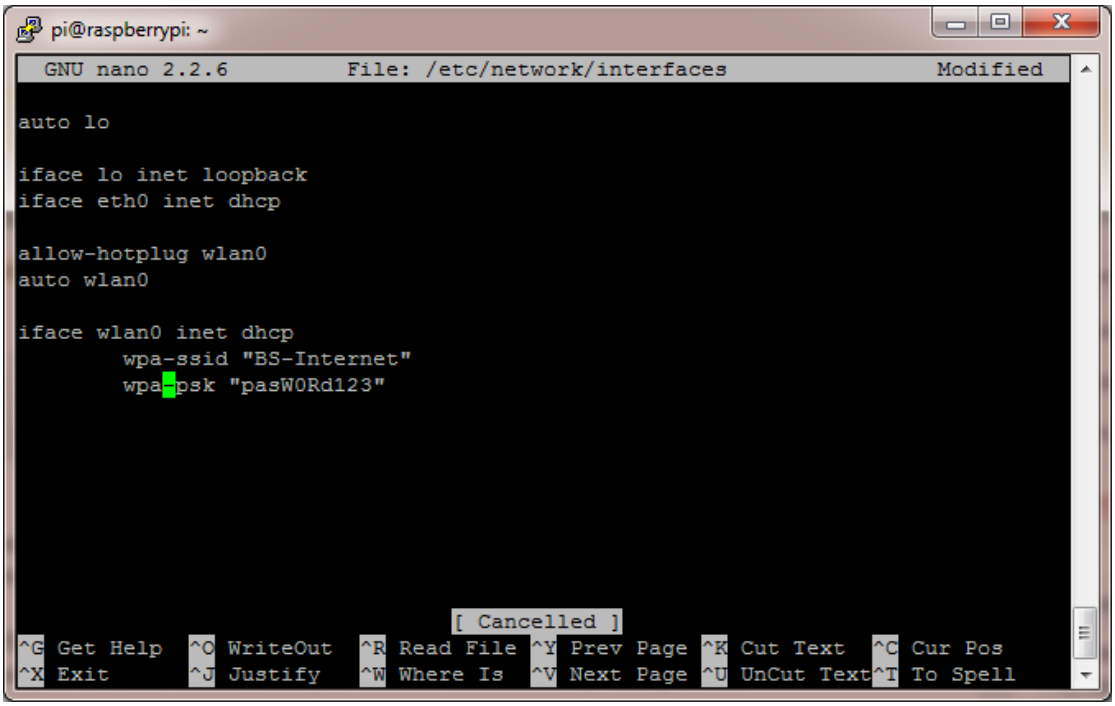
wpa-ssid "Your Network SSID"

wpa-psk "Your Password"

6. We will need to put our own SSID and password into the appropriate places.

(For any reason if we can't remember the network name, we just pull up our phone as it will most likely display the name in the settings or the network settings screens).

7. After editing the file you should see something like the following:



8. To save the file press *Ctrl+O* this will write the file to the disk – afterwards you can exit **nano** by pressing *Ctrl+X*.

If **nano** asks if you want to *Save modified buffer?* We press "Y" followed by hitting *enter* to confirm the filename.

9. At this point everything is configured – all we need to do is reload the network interfaces. This can be done by running the following command

sudo service networking reload

(*warning:* if you we are connected using a remote connection it will disconnect now)

10. After reloading the network interface - (and re-connecting to the pi if you are using a remote connection) – we can now check the status of our Wi-Fi connection by running:

ifconfig

11. The result should look something like this:

wlan0 Link encap:Ethernet HWaddr 80:1f:02:aa:12:58

inet addr:192.168.1.8 Bcast:192.168.1.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:154 errors:0 dropped:173 overruns:0 frame:0

TX packets:65 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:32399 (31.6 KiB) TX bytes:13036 (12.7 KiB)

When we see a valid IP address under “inet addr” we can now disconnect the network cable .

4.2. Method Two

Setting WiFi up via the command line

This method is suitable if you don't have access to the graphical user interface normally used to set up WiFi on the Raspberry Pi. It's especially suitable for use with a serial console cable if you don't have access to a screen or wired Ethernet network. Note also that no additional software is required; everything you need is already included on the Raspberry Pi.

5. Getting Wi-Fi network details

To scan for Wi-Fi networks, use the command "sudo iwlist wlan0 scan". This will list all available Wi-Fi networks, along with other useful information.

Look out for:

1. ESSID:"testing". This is the name of the WiFi network.
2. IE: IEEE 802.11i/WPA2 Version 1. This is the authentication used; in this case it's WPA2, the newer and more secure wireless standard which replaces WPA. This guide should work for WPA or WPA2, but may not work for WPA2 enterprise. You'll also need the password for the WiFi network. For most home routers this is located on a sticker on the back of the router. The ESSID (ssid) for the network in this case is testing and the password (psk) is testing Password.

6. Adding the Network Details to the Raspberry Pi

1. Open the wpa-supplicant configuration file in nano:

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

2. Go to the bottom of the file and add the following:

```
network={
    ssid="The_ESSID_from_earlier"
    psk="Your_wifi_password"
}
```

In the case of the example network, we would enter:

```
network={
    ssid="testing"
    psk="testingPassword"
}
```

3. Now save the file by pressing **Ctrl+X** then **Y**, then finally press **Enter**.
4. At this point, wpa-supplicant will normally notice a change has occurred within a few seconds, and it will try and connect to the network. If it does not, either manually restart the interface with "sudo ifdown wlan0" and "sudo ifup wlan0", or reboot your Raspberry Pi with "sudo reboot".

   You can verify if it has successfully connected using "ifconfig wlan0".

If the "inet addr" field has an address beside it, the Pi has connected to the network.

- If not, check if the password and ESSID are correct.

CHAPTER 6

"Software System"

(Server)

As we mentioned before, our system mainly consists of three main parts:

1- The Brain (server)
2- The Robots (forklifts)
3- The Local Area Network (LAN)

In our system architecture, we have four production lines (A line, B line, C line and D line), each line generates different products or packages from other lines.

The brain has a full view for the whole system; it knows the map of the warehouse, the robots' state (idle or busy), the number of products in each production line, the priority of each production line, and the storing locations for each product line.

The brain also checks if there is any product on the production line needs any function, and this is by the server accessing the production line's database looking for number of product on each line, choosing the specific production line according to number of products and production line’s priority.

In our work, we split the project into two main phases:

1- Phase1 (Single Robot System)
2- Phase2 (Multiple Robots)

1. Phase (1) "Single Robot System"

1.1. Mapping

The main aim of mapping is to simulate the actual warehouse to ease the process of the project during the simulation. And as to represent any plane we use matrices.

With the use of rows and columns, it's easy to define a number of points, where every point has its own notation so, we can call it anytime.
Every point has many neighboring points, mainly all points have four neighboring points but for the edge points of the matrix which have only three neighboring points.
Between each two points there is a weight, and this is used in calculating the shortest path using a certain algorithm that is explained in details later on to this chapter.

The weight represents how mush these paths can cost the robot to pass by it, so to give a priory to one path over another, we give it a lower weight.

1.1.1. Maps Versions

There were mainly two versions of the map used in the system,

1. First Version
2. Second Version

1. First Version:

The first map version was just a test to try our shortest path algorithms, checking if it's valid and effective or not.

We generated a matrix automatically with the code attached with dimensions of 18*18, then we defined a start and end (Destination) points, then we changed the weight of some paths, and let the code calculate the shortest path.

As shown in figure 6.1. the code used to generate the matrix

```
def matrix(n, m):
    z = []
    x = [[]]
    for l in range(0, n-1):
        x.append([])
    # generate a list of Nodes
    for i in range(0, n):
        for j in range(0, m):
            z.append((i, j))
    # convert the list into array
    for k in range(0, m):
        x[k] = z[k*m:k*m+m]

    return x
```

```
def matrix(n, m):

    x = [[]]
    z = []

    for l in range(0, n-1):
        x.append([])

    for i in range(0, n):
        for j in range(0, m):
            z.append(1)

    for k in range(0, n):
        x[k] = z[k*m:k*m+m]
    return x
```

Fig. 6.1. Used code to generate the matrix of first version of mapping system

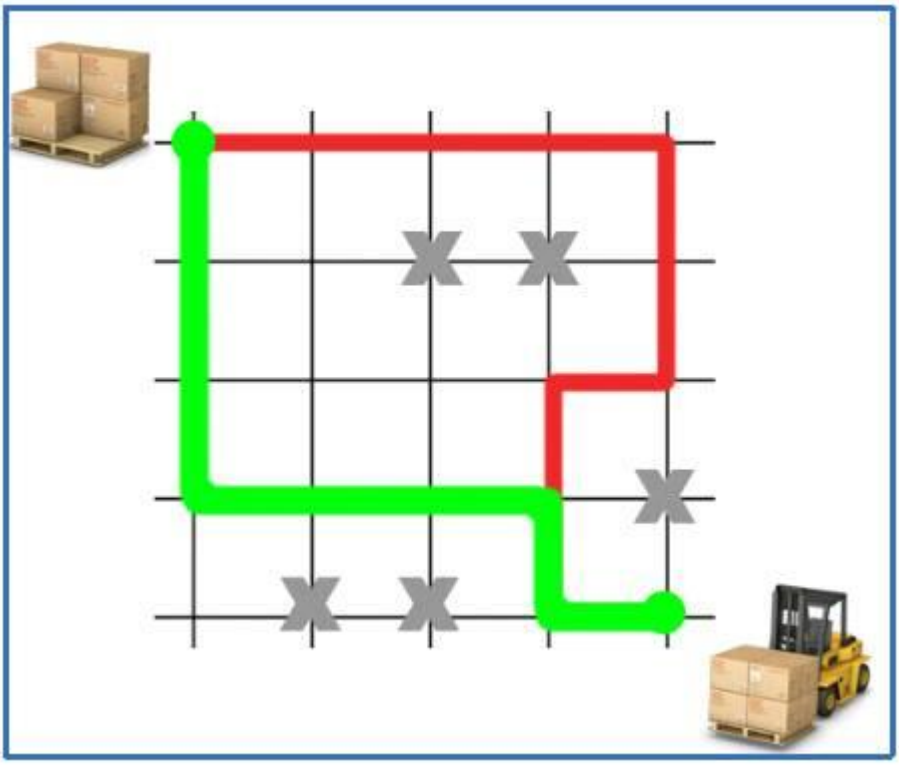


Figure 6.2. First Version Mapping Simulation

2. Second Version:

The second version of the map is an enhanced version, we took into consideration the racks places that the forklifts store the products in, the paths and how to grow as close as possible with actual real version of the warehouse. This is also is to make the process more easy to our robot.

We divided the map into 90 points, 16 of them are for the racks which obviously can't be used, another 74 points which are available to use.

But in this version we used on off weights instead of increasing and decreasing the weights, so initially the weights are all with the

value of 1 and if any path is closed for any reason (maintenance - another robot) it's weight value turns to 0. See fig. 6.3.

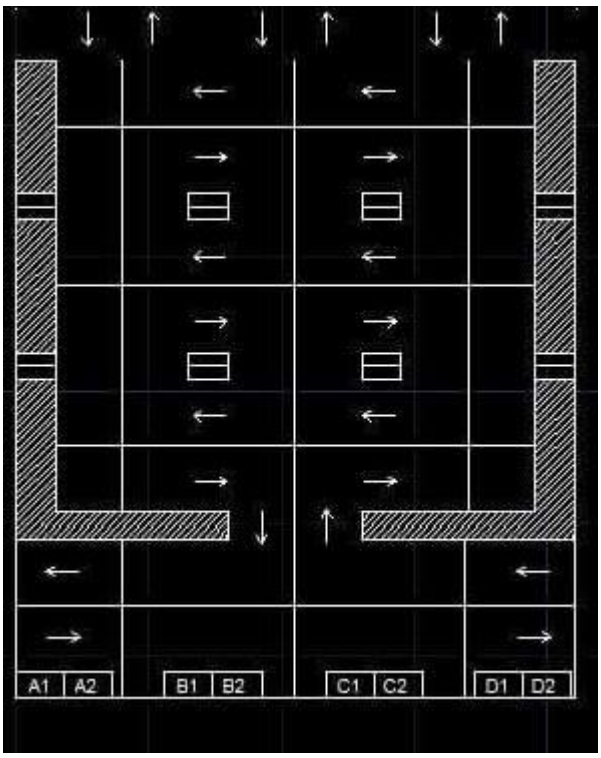


Figure 6.3. Second Version Mapping Simulation

To generate such a matrix we had to generate it manually, so we defined each single, point its neighboring points, created a complete graph for our map which can be used to simulate the real warehouse with all of its possible cases.

1.2. Shortest Path Algorithms

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. The problem of finding the shortest path between two intersections on

a road map (the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment) may be modeled by a special case of the shortest path problem in graphs.

1.2.1. Shortest Path Problem Definition.

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. It is defined here for undirected graphs; for directed graphs the definition of path requires that consecutive vertices be connected by an appropriate directed edge. Two vertices are adjacent when they are both incident to a common edge. A path in an undirected graph is a sequence of vertices $P = (v1, v2, . . .,vn) \in V \times V \times . . . \times V$ such that vi is adjacent to vi+1 for $1 \leq i< n$. Such a path P is called a path of length $n - 1$ from v1 to vn . (The vi are variables; their numbering here relates to their position in the sequence and needs not to relate to any canonical labeling of the vertices.) Let ei,j be the edge incident to both vi and vj . Given a real-valued weight function $f : E \rightarrow R$, and an undirected (simple) graph G , the shortest path from v to v ′ is the path P = (v1, v2, . . . , vn) (where v1 = v and vn = v ′ ∑) that over all possible n minimizes the sum n−1 i=1 f(ei,i+1). When each edge in the graph has unit weight or $f : E \rightarrow \{1\}$, this is equivalent to finding the path with fewest edges.

The problem sometimes is also called the single-pair shortest path problem, and to distinguish it from the following variations:

- **The single-source shortest path problem**:

  Is the problem in which we have to find shortest paths from a source vertex v to all other vertices in the graph.

- **The single-destination shortest path problem:**

  Is the problem in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v.

  This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

- **The all-pairs shortest path problem:**

  Is the problem in which we have to find shortest paths between every pair of vertices v, v' in the graph.

  These generalizations have significantly more efficient algorithms than the simplistic approach of running a single pair shortest path algorithm on all relevant pairs of vertices.

1.2.2. Shortest Paths Problem Algorithms

We can summarize the most important algorithms for solving short path problems in the following lines.

1. **Dijkstra's algorithm**

   It solves the single-source shortest path problem.

2. **Bellman–Ford algorithm**

   It solves the single-source problem if edge weights may be negative

3. **A* search algorithm**

   It solves for single pair shortest path using heuristics to try to speed up the search.

4. **Floyd -Warshall algorithm**

   It solves all pairs shortest paths.

5. **Johnson's algorithm**

   It solves all pairs shortest paths, and may be faster than Floyd-Warshall on sparse graphs.

6. **Viterbi algorithm**

   It solves the shortest stochastic path problem with an additional probabilistic weight on each node.

1.2.3. Dijkstra's Algorithm:

As mentioned above, Dijkstra's Algorithm is one of the most important methods to slove the shortest paths problems, and it's used mainly to solve the single source shortest paths problems. This is the Algorithm we are using here to our software system runner.

1.2.3.1. Dijkstra's Algorithm History

Dijkstra thought about the shortest path problem when working at the Mathematical Center in Amsterdam in 1956 as a programmer to demonstrate capabilities of a new computer called ARMAC. His objective was to choose both a problem as well as an answer that would be produced by computer, that non-computing people could understand. He designed the shortest path

algorithm and later implemented it for ARMAC for a slightly simplified transportation map of 64 cities in the Netherlands (64, so that 6 bits would be sufficient to encode the city number). A year later, he came across another problem from hardware engineers working on the institute's next computer: minimize the amount of wire needed to connect the pins on the back panel of the machine. As a solution, he re-discovered the algorithm known as Prim's minimal spanning tree algorithm (known earlier to Jarník, and also rediscovered by Prim). Dijkstra published the algorithm in 1959, two years after Prim and 29 years after Jarník.

1.2.3.2. Algorithm Method.

Let the node at which we are starting from be called the **initial node**. Let the **distance of node** Y be the distance from the **initial node** to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

❖ **Steps:**

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current.
3. Mark all other nodes unvisited.
4. Create a set of all the unvisited nodes called the unvisited set.
5. For the current node, consider all of its unvisited neighbors and calculate their tentative distances.

6. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
   For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6 + 2 = 8.
   If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
7. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. And a visited node will never be checked again.
8. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
9. The algorithm has finished. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

1.2.3.3. Algorithm Explication

Dijkstra's Algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and other network related protocols. For a given source vertex (node) in the graph, the algorithm finds the

path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra`s algorithm can be used to find the shortest route between one city and all other cities.

Dijkstra`s algorithm works by solving the sub problem k, which computes the shortest path from the source to vertices among the k closest vertices to the source. For the Dijkstra's algorithm to work it should be directed- weighted graph and the edges should be non-negative. If the edges are negative, then the actual shortest path cannot be obtained. At the kth round, there will be a set called Frontier of k vertices that will consist of the vertices closest to the source andthe vertices that lie outside frontier are computed and put into New Frontier. The shortestdistance obtained is maintained in sDist[w]. It holds the estimate of the distance from s to w.

Dijkstra`s algorithm finds the next closest vertex by maintaining the New Frontier vertices in a priority-min queue. The algorithm works by keeping the shortest distance of vertex v from the source in an array, sDist. The shortest distance of the source to itself is zero. sDist for all other vertices is set to infinity to indicate that those vertices are not yet processed. After the algorithm finishes the processing of the vertices sDist will have the shortest distance of vertex w to s. two sets are maintained Frontier and New Frontier which helps in the processing of the algorithm. Frontier has k vertices which are closest to the

source, will have already computed shortest distances to these vertices, for paths restricted up to k vertices. The vertices that resides outside of Frontier is put in a set called New Frontier.

1.2.3.4. Dijkstra's Algorithm Pseudocode

In the following algorithm, the code u ← vertex in Q with min dist[u], searches for the vertex *u* in the vertex set *Q* that has the least dist[*u*] value. length (*u*, *v*) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes *u* and *v*. The variable *alt* on line 17 is the length of the path from the root node to the neighbor node *v* if it were to go through *u*. If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path. The *prev* array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:             // Initialization
6          dist[v] ← INFINITY                  // Unknown distance from source to v
7          prev[v] ← UNDEFINED                 // Previous node in optimal path from source
8          add v to Q                          // All nodes initially in Q (unvisited nodes)
9
10     dist[source] ← 0                        // Distance from source to source
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]    // Source node will be selected first
14         remove u from Q
15
16         for each neighbor v of u:           // where v is still in Q.
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]:               // A shorter path to v has been found
19                 dist[v] ← alt
20                 prev[v] ← u
21
22     return dist[], prev[]
```

With this *alt* path. The prev array is populated with a pointer to the "next-hop" node on the source

- graph to get the shortest route to the source.

- If we are only interested in a shortest path between vertices *source* and *target*, we can terminate the search after line 13 if *u* = *target*. Now we can read the shortest path

```
1  S ← empty sequence
2  u ← target
3  while prev[u] is defined:                  // Construct the shortest path with a stack S
4      insert u at the beginning of S         // Push the vertex onto the stack
5      u ← prev[u]                            // Traverse from target to source
6  insert u at the beginning of S             // Push the source onto the stack
```

from *source* to *target* by reverse iteration:

Now sequence *S* is the list of vertices constituting one of the shortest paths from *source* to *target*, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between *source* and *target* (there might be several different ones of the same length). Then instead of storing only a single node in each entry of prev [] we would store all nodes satisfying the relaxation condition. For example, if both *r* and *source* connect to *target* and both of them lie on different shortest paths through *target* (because the edge cost is the same in both cases),

then we would add both *r* and *source* to prev [*target*]. When the algorithm completes, prev [] data structure will actually describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm was run with some starting node, then every path from that node to any other node in the new graph will be the shortest path between those nodes in the original graph, and all paths of that length from the original graph will be present in the new graph. Then to actually find all these shortest paths between two given nodes we would use a path finding algorithm on the new graphs, such as depth-first search.

1.2.3.5. Using a priority queue

A min-priority queue is an abstract data type that provides 3 basic operations:

1. add_with_priority ()
2. decrease_priority()
3. extract_min()

As mentioned earlier, using such a data structure can lead to faster computing times than using a basic queue.
Notably, Fibonacci heap (Fredman&Tarjan 1984) or Brodal queue offer optima implementations for those 3 operations. As the algorithm is slightly different, we mention it here, in pseudo-code as well:

```
function Dijkstra(Graph, source):
    dist[source] ← 0                                    // Initialization

    create vertex set Q

    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY                          // Unknown distance from source to v
            prev[v] ← UNDEFINED                         // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                              // The main loop
        u ← Q.extract_min()                            // Remove and return best vertex
        for each neighbor v of u:                       // only v that is still in Q
            alt = dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist[], prev[]
```

Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only source; then, inside the if alt <dist[v] block, the node must be inserted if not already in the queue (instead of performing a decrease_priority operation).

Other data structures can be used to achieve even faster computing times in practice.

1.2.4. Dijkstra Algorithm Practical Case.

To understand the way Dijkstra's algorithm works more specifically, showing a practical case would make the picture more clear.

Single-source shortest-paths:

- **Step 1**

Using Figure 6.4. we are going to trace the graph from vertex A to all other accessible vertices. We will consider vertex A to be our origin. Before we begin, notice that we are using a directed weighted graph. We can tell this because the edges contain arrows showing the direction of travel they will allow. The graph is weighted because each edge is assigned a non-negative numerical value. Starting at A, we look for vertices that we can reach. Here we can reach B (cost 50), C (cost 30), D (cost 100), and F (cost 10).

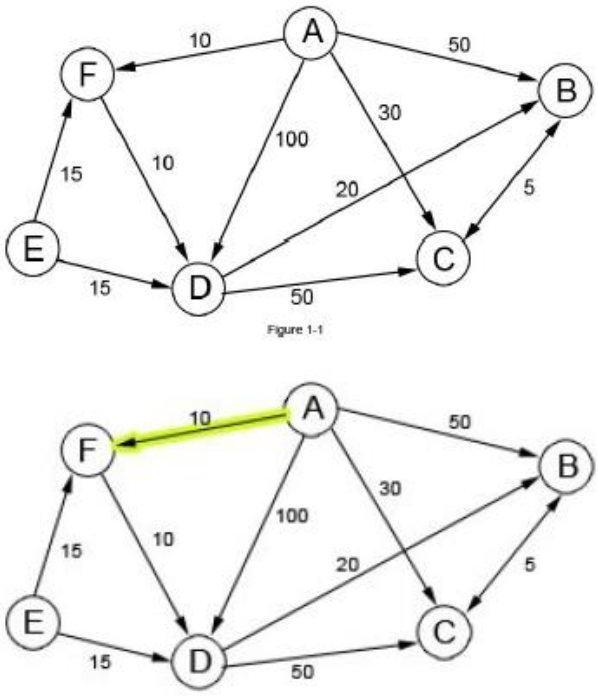


Figure 6.4.

- **Step 2**

In the prior illustration we were able to identify that from vertex A, vertex F was the F vertex with the shortest path or least value. So now given A and F, we need to find the next path of least cost, so we evaluate the differences at F and add them to our table on the right. From If we cannot get to B, however we can get to B from A as indicated from the previous step. Therefore, we bring the 50/A form line 1 of the table, down to line 2. The same event will follow through. See fig. 6.5. and fig. 6.6.

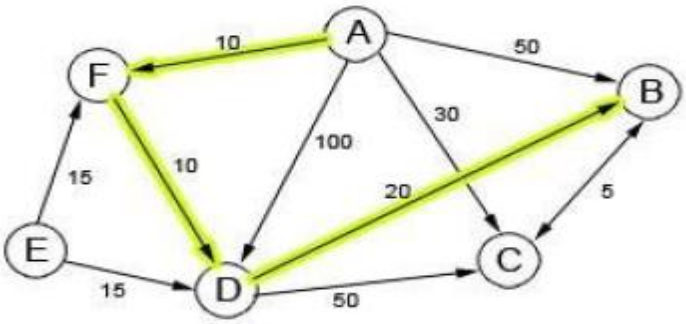


Figure 6.5.

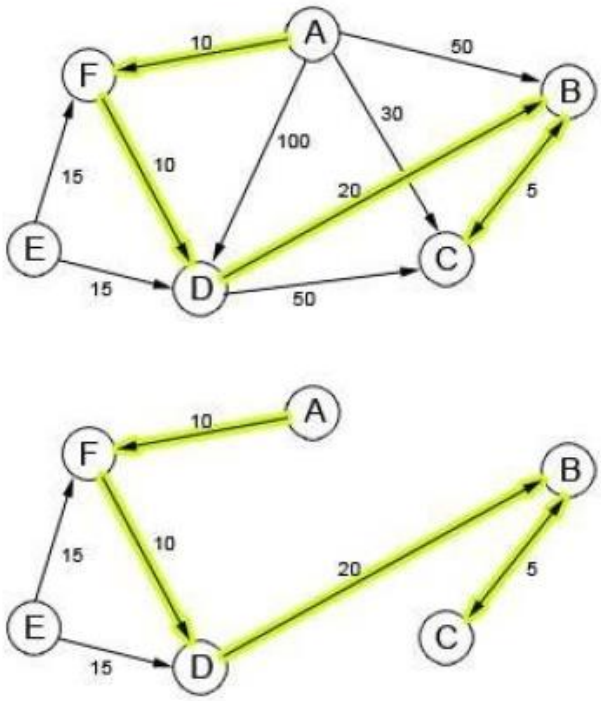


Figure 6.6.

- **Step**3

Now that vertices A, F, and D have been identified as the path with least cost, we now look for the next vertex from D or F, A and B if available as well. From D, we can reach vertices B and C with costs of 20 and 50 respectively. Therefore, we will choose B since its cost is lower than C.

- **Step4**

So far we have identified vertices A, F, D, and B as the shortest path. We now need to account for vertex C. We could have traveled to C from A several steps earlier, but the cost of going to C was far greater than the cost to travel to F at that time. Since A, D, and B are in our generated path, and each of these vertices can go to D, we need to look at each individual edge and the cost assigned. A to C has a cost of 30, D to C has a cost of 50, and B to C has a cost of 5. As a result, we will choose edge BC at a cost of 5.

- **Step 5**

Our final graph of Dijkstra Algorithm will look like the data shown above. The final traceable path from origin A is A-F-D-B-C with E remaining at infinity since the only edges that originate from E point away at either direction making it impossible to reach E.

1.2.5. Dijkstra's Algorithm Implementation Approach:

There are three approaches that Dijkstra's Algorithm can be implemented with.

- **Easy**
- **Dijkstra**

- **Modern**

1.2.5.1. Easy

Initialize S to s, dist[s] to 0, dist[v] to for all other v Repeat until S contains all vertices connected to s

- Find v-w with v in S and w in S’ that minimizes dist[v] + weight[v-w]
- Relax along that edge • add w to S Idea 1 (easy): Try all edges Total running time proportional to VE

1.2.5.2. Dijkstra

Initialize S to s, dist[s] to 0, dist[v] to for all other v Repeat until S contains all vertices connected to s

- Find v-w with v in S and w in S’ that minimizes dist[v] + weight[v-w]
- relax along that edge
- Add w to S Idea 2 (Dijkstra) : maintain these invariants • for v in S, dist[v] is the length of the shortest path from s to v.
- For w in S’, dist[w] minimizes dist[v] + weight[v-w]. Two implications
- Find v-w in V steps (smallest dist[] value among vertices in S’)
- Update dist[] in at most V steps (check neighbors of w) Total running time proportional to V 2

1.2.5.3. Modern:

Initialize S to s, dist[s] to 0, dist[v] to for all other v Repeat until S contains all vertices connected to s

- Find v-w with v in S and w in S' that minimizes dist[v] + weight[v-w]
- Relax along that edge
- Add w to S Idea 3 (modern implementations)
- For all v in S, dist[v] is the length of the shortest path from s to v
- Use a priority queue to find the edge to relax

  Total running time proportional to E lg E

	sparse	dense
easy	V^2	EV
Dijkstra	V^2	V^2
modern	E lg E	E lg E

Figure 6.7. Dijksta's Algorithm implementation approaches

1.2.6. Correctness Proof of Dijkstra's Algorithm:

Proof is by induction on (S).

Base Case: S = f v_0 g The cost of the shortest path from v_0 to v_0 is 0 by definition, i.e., D[v] = 0. For all other vertices v: a path which lies entirely within S = f v_0 g except for the endpoint, v itself consists only of one edge. Therefore, the cost of that path is: D[v] = f (v_0; v) where f is the weight function for the graph. (1)
Notice that the definition of D in equation (1) is consistent with the claim stated in sub-section x1.2.

General Case: By the inductive hypothesis, the values of D[v] is the cost of the shortest path for all v 2 S. Let w denote the vertex in V --S such that D[w] is
minimal. The situation is illustrated in below, fig. 6.8.

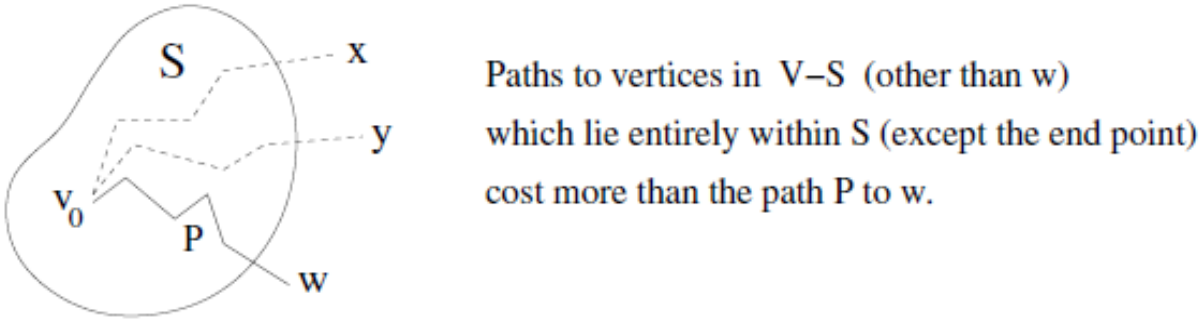


Figure 6.8. Dijkstra's Algorithm Shortest Path Calculation

Claim: D[w] is the cost of the shortest path from v_0 to w (with no restrictions on the possible path).

Proof of the Claim: (by contradiction) Suppose there is a shorter path P 0 to w. The path P0 must go to some vertex (other than w)

outside of S and then proceed to w.4 The situation is illustrated in the following figure 6.9.

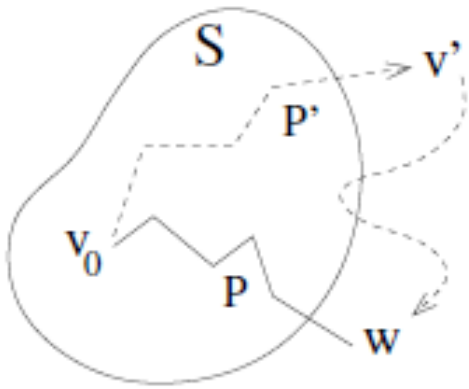


Figure 6.9. Dijksta's Algorithm Proof of the Claim

Let v_0 be the first vertex outside of S along the path P 0. Observe that the cost of path P 0 to w is higher than the partial path along P 0 to v_0. This contradicts the way in which w was chosen. I.e., if the path P 0 exists, Dijkstra's algorithm would have chosen v_0 instead of w.

The update step for D[v] considers the fact that in the newly expanded set S, the paths from v_0 to vertices x still outside of S may go to w, and then proceed to x.

1.2.7. Dijkstra's Algorithm characteristics

1.2.7.1. Dijkstra's Algorithm Advantages

- Once it has been carried out you can declare the least weight path to all permanently labeled nodes.
- You don't need a new diagram for each pass. Dijkstra's algorithm has an order of n^2 so it is suitable enough to use for relatively large problems.

1.2.7.2. Dijkstra's Algorithm Disadvantages

- The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.
- Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

1.2.8. Dijkstra's Algorithm Applications

- Traffic information systems use Dijkstra`s algorithm in order to track the source and destinations from a given particular source and destination
- In the Link-state routing protocol approach, using Dijkstra`s algorithm each router calculates the shortest path to each network and enters this information into the route table.

- OSPF- Open Shortest Path First, used in Internet routing. It uses a link-state in the individual areas that make up the hierarchy. The computation is based on Dijkstra`s algorithm which is used to calculate the shortest path tree inside each area of the network.
- Maps
- **Robot navigation.**
- Texture mapping
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Subroutine in advanced algorithms.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Network routing protocols (OSPF, BGP, RIP).

1.3. Single Robot System Description

As we mentioned before, the single robot phase mainly considers the interaction between the system's main executer, which is the forklift, and the production line that holds all the objects the system is implemented upon moving and carrying.

1.3.1 System Brain

As we knew from the whole system's overview, that the brain

is one of the main three basic posts of the system.

The brain is mainly the controller of the system, as it has a full view for the whole framework; it knows the map of the warehouse, the forklift's state whether it's idle, so it's ready to receive orders from the server to implement a specific function, or it's busy which means it's currently working on a previously sent task to it, when the brain waits for its state to come back again to idle for it to have another task to proceed.

The brain is also aware of the number of the products on each production line, the priority of each production line, and the storing locations for each product line.

Every production line in our system has its own database, through which the brain checks if there is any product on the production line, looking for number of product in each production line, choosing the production line according to number of products and production line's priority.

1.3.2 Brain Prospects

Initially, the robot is placed in a specific location to avoid any traffic jams in the warehouse.

As the brain always checks the database to take the actions of giving specific orders to the forklifts, there are three different probable cases allowed to the brain concerning this matter.

And they are as follow;

I. **If there are no products on the production lines.**

At this case, the brain gives an order to the robot to remain idle until further announcement.

II. **If the brain found products on the production lines, but the robot is found to be busy doing a task.**

At this case the brain waits for the robot to be idle again or to finish its task.

III. **If there are products on the production lines and the robot is found to be idle.**

At this case the brain goes through its main function which we mainly talk about in this section.

1.3.3 Brain Steps of Implementation:

The main steps that the brain follows to take a decision upon each of the probable prospects that can be found in the system, are illustrated as follow;

1- The brain hires the robot by changing the state of robot to be "busy".

2- The brain specifies the product line where there are products on need to be stored, this product line is specified by a point in the map we work on, so we can consider this point as an end point to the robot to reach.

3- The brain runs Dijkstra algorithm for the first time to find the shortest path from the robot initial point to the point of the product location.

4- When the robot reaches the specified production line, the brain gives an order to the robot to upload the product or the package from the production line.
5- According to the product or the package, the brain knows the corresponding point for storing this package from its database.
6- Then brain checks the flags of storing location, and chooses one of storing location according to its priority.
7- Also it changes the flag of the chosen storing location to be “0”, indicating that this place is to be filled with a package.
8- Now the brain assigns the robot with a task to carry the product to the appropriate storing location which is done by running Dijkstra algorithm for the second time to find the shortest path to the specified storing location.
9- At the end of the second path, or at the place of storage, the brain gives an order to the robot to download the package it carries in the specified point.
10- After the robot finishes its task, the brain returns the state of robot to “idle”.
11- The robot returns back to its specific location, also by applying Dijkstra algorithm to find the shortest path, unless the brain hires it to do another task.
12- Finally, the process is always repeated during the warehouse work hours.

1.4 Single Robot Phase Simulation:

Our system architecture of the forklifts and the production lines, and as we mentioned the main role of the brain at coordinating all

of orders among the system objects, this section will illustrate the software implementation of this system.

In our system, the brain implementation is done using **Python** programming language on both Windows operating system on laptops, and Linux (specifically the Raspian) operating system on the Raspberry Pi 2.

1.4.1. Python Programming Language.

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including **object-oriented**, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library which make it a very suitable choice for our software system.

1.4.2. System Simulation

At first we target to simulate the system to test it before setting up the communication with the forklifts.

With the aid of the Pygame library [1], we managed to simulate the system architecture successfully. Pygame is a cross-platform library designed to make it easy to write multimedia software, such as games, in Python. Pygame requires the Python language and SDL multimedia library. It can also make use of several other popular libraries.

The system simulation can be clearly illustrated in the following steps;

1. Firstly, we set up the background of the warehouse as shown in figure 6.13. , the background consists of a grid of the map points we are working on, those points also present the RF IDs which the forklifts walk on detecting a point in the map. Also the background consists of the outline of the warehouse, the production lines (uploading area) and the racks (downloading area).

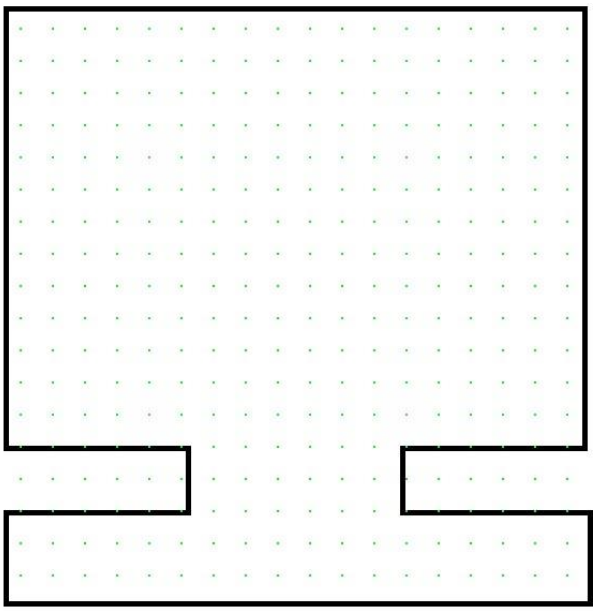


Figure 6.13.

2. Then, we wanted to implement the movement of the robot in the background through its uploading path from its random start position which is assumed to be any point in the grid to the production lines A, B, C or D depending on the robot task which is specified by the system as we illustrated at previous sections. We want also to implement the movement through the downloading path from the production lines to the according racks. So we took those two paths with their end points as system output to be an input to the simulation process or function, to simulate the movement of the robot to the specific storing point besides the storing process as shown in figure 6.14.

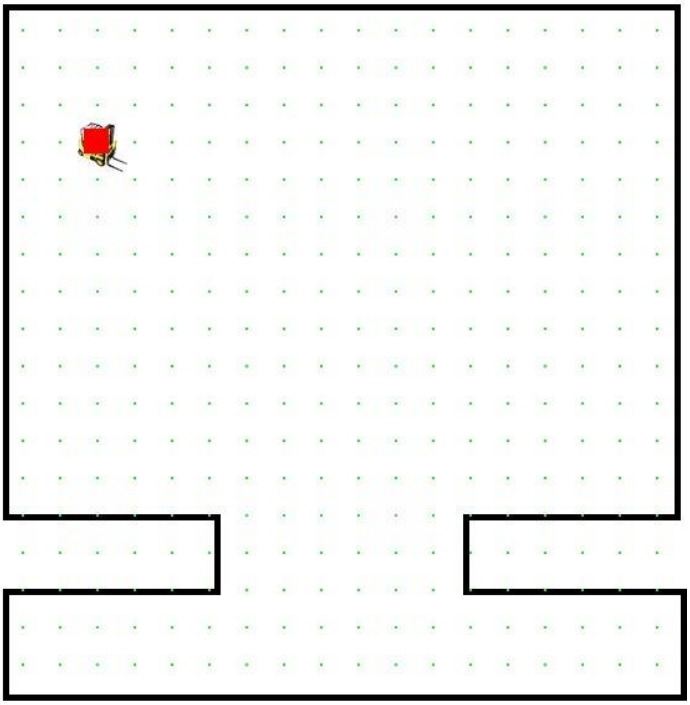


Figure 6.14.

3. After storing one object taken from one of the production lines (A, B, C or D), the system does not stop as there are more objects on the same production line or on other ones. So we apply the system

again on the current working robot. By applying the system, we mean to see what object has to be stored in order to specify the robot task or in other words the end point of the robot then the path the robot should take for both uploading and downloading. By simulating this system, we see the final storing process as shown in figure 6.15. for 4 different packages.

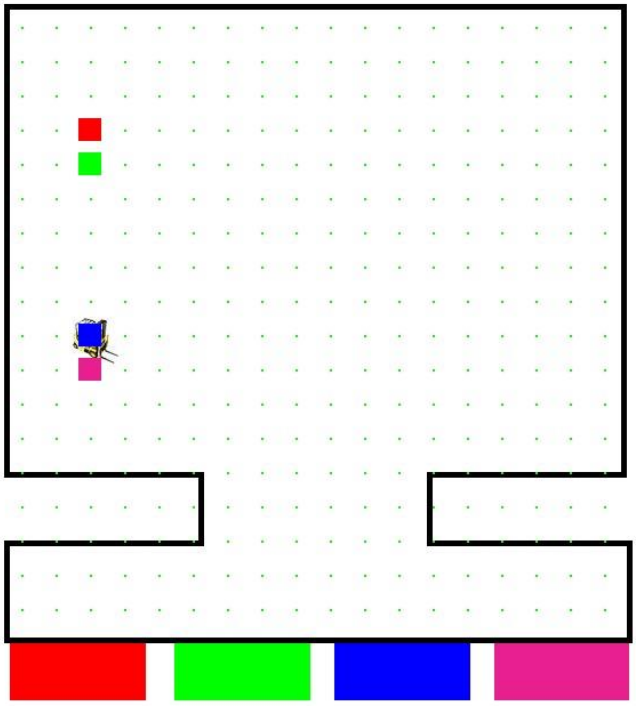


Figure 6.15.

2. Phase Two "Multi Robots"

2.1. System Database

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. It is an efficient integration of data which is structured to facilitate data-processing operations.

2.1.1. Database Management System

Access to these data is usually provided by a "database management system" (**DBMS**) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database. The **DBMS** provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.

2.1.1.1. DBMS Functions

Existing DBMSs provide various functions that allow management of a database and its data which can be classified into three main functional groups:

1. Data definition – Creation, modification and removal of definitions that define the organization of the data.
2. Update – Insertion, modification, and deletion of the actual data.
3. Retrieval – Providing information in a form directly usable or for further processing by other applications. The retrieved data may be made available in a form basically the same as it is stored in the database or in a new form obtained by altering or combining existing data from the database.

2.1.2. Database Models

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in

which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.

2.1.2.1. Relational database:

Relational database is a computer database model in which all data is stored in Relations which are tables with rows and columns. Each table is composed of records (called Tuples) and each record is identified by a field (attribute) containing a unique value. Every table shares at least one field with another table in three different relationships;

- One to one
- One to many
- Many to many

These relationships allow the database user to access the data in almost an unlimited number of ways, and to combine the tables as building blocks to create complex and very large databases.

Originally, databases were flat. This means that the information was stored in one long text file, called a tab delimited file.

Each entry contains multiple pieces of information (fields) about a particular object or person grouped together as a record. The text file makes it difficult to search for specific information or to create reports that include only certain fields from each record.

Related information is gathered by a sequential search through the entire file.

A relational database allows to easily find specific information. It also allows to sort based on any field and generate reports that contain only certain fields from each record. Relational databases use tables to store information. The standard fields and records are represented as columns (fields) and rows (records) in a table.

The arrangement of data in columns provides a quick comparison of information. The relational database model takes advantage of this uniformity to build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database.

2.1.2.2. Structured query language (SQL):

The standard user and application program interface to a relational database is the structured query language (SQL). SQL statements are used both for interactive queries for information from a relational database and for gathering data for reports. It is designed over relational algebra and tuple relational calculus.

2.1.3. Database in Python

The Python standard for database interfaces is the Python DB-API (database application programming interface). Most Python database interfaces adhere to this standard.

2.1.3.1. Python DB-API

The Python database API (DBAPI) provides a 'database neutral' programming interface for a wide range of database servers such as MySQL, PostgreSQL, SQLite, MS-SQL and Oracle, and others, based on the structured query language SQL.

SQL is a language called Structured Query Language used to connect to and interact with relational database systems to provide queries to manipulate data, via an interface (software) that facilitates communications between database management system and a python program.

2.1.4. MySQL Database Server

MySQL is the most database server used. MySQLdb is the interface for connecting to a MySQL database server to Python.

MySQL is a relational database system, which is a logical representation of data that allows the relationships between the data. MySQL is the world’s most popular open source database, enabling the cost-effective delivery of reliable, high-performance and scalable Web-based and embedded database applications.

2.1.4.1. MySQL Server Selection

As mentioned before, that The Python database API (DBAPI) provides a 'database neutral' programming interface for a wide range of database servers such as MySQL, PostgreSQL, SQLite, MS-SQL and Oracle, and many others, based on the structured query language SQL. There are many reasons upon which we chose to work with MySQL database server. Which are;

1. High performance
2. Strong data protection
3. Open source

1. High Performance

Whether the intended application is a high-speed transactional processing system or a high-volume web site that services a billion queries a day, MySQL can meet the most demanding performance expectations of any system. MySQL is undeniably quick. It is a true multi-threaded database server that excels at retrieving information. MySQL was originally designed for the purpose of querying information at an amazingly fast pace. With high-speed load utilities, distinctive memory caches, full text indexes, and other performance-enhancing mechanisms.

2. Strong Data Protection

Because guarding the data assets of corporations is the number one job of database professionals, MySQL offers exceptional security features that ensure absolute data protection. In terms of database authentication, MySQL provides powerful mechanisms for ensuring only authorized users have entry to the database server, with the ability to block users down to the client machine level being possible.

3. **Open Source**

MySQL is an open-source product released under the GPL (GNU Public License). It is free; it is usually available on shared hosting packages and can be easily set up in a Linux, UNIX or Windows environment. If a web application requires more than database, requires load balancing or sharing, it is easy to set up maybe instances of the database requiring only the hardware costs, as opposed to commercial databases that would require a single license for each instance.

2.1.4.2. SQL connectivity steps:

As we mentioned before that we chose the SQL database server, the following steps will illustrate the connectivity steps of this database system**.**

1. Import database module
2. Connect to the database

- The DB API describes a connection object that accesses the database.
- For SQLite no host, user and password are necessary to get a connection, the database is just a single file.

3. Acquire cursor

- The connection object is used to create the cursor object, which manipulates and retrieves data.
- With connection.cursor()

4. Execute SQL statement.

- With execute(), executemany() or executescript():

5. Fetch the results.

- With fetchone(), fetchmany() or fetchall()

6. Commit changes.

- With connection.commit()

7. Close connection.

- Use connection.close()

2.1.5. System Database Tables

For our system, there are the production lines, the robots, and the tags that need database tables. This section is illustrating each one of them.

1. **Production Lines Tables**

The table consists of three fields: the production line, its location and number of objects required to be transported to the storage area as shown in figure 6.16.

Table: Lines

	prod_line	num	loc
	Filter	Filter	Filter
1	A1	2	(0,0)
2	A2	2	(0,1)
3	B1	2	(0,5)
4	B2	0	(0,6)
5	C1	1	(0,11)
6	C2	1	(0,12)
7	D1	0	(0,16)
8	D2	2	(0,17)

Figure 6.16. Production Line Database Table

The system starts by checking number of objects in all production lines to select the line to be served; the priority is given to the one with more objects. After determining the object, the number of the total objects on that line is decremented by one and its location is retrieved also from table.

2. Robots Database Tables.

The table of Robots consists of three fields: the robot, its current location and its status as shown in figure 6.17.

Robot status is given to be "0" if the robot is busy, and set to be "1" if it's idle.

Table: Robots

	robot	location	status
	Filter	Filter	Filter
1	1	(7,2)	1
2	2	(1,0)	1
3	3	(8,2)	0

Figure 6.17. Robots Database Table

Now, an object is ready to be transported so it is required to select a robot to do this job. The selection process starts by checking the status of all robots, if the robot is on task or idle. The next step is comparing between the free robots to choose the suitable one for that task, the comparison is based on the path between the robot and the object. The start point of this path is the location of the robot, which is retrieved also from Robots table, and the end point is the location of the object (line). The priority is for the one with the shortest path to the object. After determining the robot to do the job, its status is cleared to indicate that it is on task and not ready to take any other tasks. The status is set again to idle after finishing the job.

3. Flags Database Tables

Flags table represents the points of storage area. For each flag we define its location and its status as shown in figure 6.18.

Table: Flags

	flag	status	graph	location
	Filter	Filter	Filter	Filter
1	f01	1	{(7, 2): {(6, 2): 1,...	(7,2)
2	f02	1	{(7, 3): {(8, 3): 1,...	(7,3)
3	f03	1	{(7, 14): {(6, 14):...	(7,14)
4	f04	1	{(7, 15): {(8, 15):...	(7,15)
5	f05	1	{(8, 2): {(7, 2): 1,...	(8,2)
6	f06	1	{(8, 3): {(9, 3): 1,...	(8,3)
7	f07	1	{(8, 14): {(7, 14):...	(8,14)
8	f08	1	{(8, 15): {(9, 15):...	(8,15)
9	f09	1	{(13, 2): {(12, 2):...	(13,2)
10	f10	1	{(13, 3): {(14, 3):...	(13,3)
11	f11	1	{(13, 14): {(12, 1...	(13,14)
12	f12	1	{(13, 15): {(14, 1...	(13,15)
13	f13	1	{(14, 2): {(13, 2):...	(14,2)
14	f14	1	{(14, 3): {(15, 3):...	(14,3)
15	f15	1	{(14, 14): {(13, 1...	(14,14)

Figure 6.18. Flags Database Table

Now, the forklift is ready to transport the object but the destination is not known yet.

For each production line, there are certain racks in the storage area to store its objects, these racks are defined by the flags. So, once the object to be transported is selected and the robot to do the task is chosen, the robot goes to the object and scans its barcode. Based on the barcode scanned, the corresponding flags are known. But the aim now is to select a specific flag. The selection is based on the status: the flags are monitoring in a sequential method, the first empty flag found will be the destination of the transport path and its location is retrieved from the table. The status of this flag is then cleared to indicate that this point is closed.

2.2. Multi-Threading

In a networked world, it is common practice to share resources among multiple users. Therefore application programs designed to be deployed in a network should be designed to serve multiple users requests simultaneously. Even on desktop computers, users typically run multiple applications and carry out some operations in the background (e.g., printing) and some in the foreground (e.g., editing) simultaneously. With the increasing popularity of multi-core processors, it is common to see even desktop and laptop computers with an ability to carry out multiple tasks concurrently. To meet these requirements, modern programming languages and operating systems are designed to support the development of applications containing multiple activities that can be executed concurrently.

Modern operating systems hold more than one activity (program) in memory and the processor can switch among all to execute them. This simultaneous occurrence of several activities on a computer is known as multitasking. For example, you can open a fi le using MS Word and you can work on MS Access for creating your database. Two applications, MS Word and MS Access, are available in memory and the processor (by means of the operating system) switches to the application you are actively working on. Here two different applications are made to run concurrently by the same processor. The operating system supports multitasking in a cooperative or preemptive manner. In

cooperative multitasking each application is responsible for relinquishing control to the processor to enable it to execute the other application. Earlier versions of operating systems followed cooperative multitasking. Modern operating systems such as Windows 95, Windows 98, Windows NT, and Windows 2000 support preemptive multitasking. In the preemptive type multitasking, the processor is responsible for executing each application in a certain amount of time called a timeslice. The processor then switches to the other applications, giving each its timeslice. The programmer is relieved from the burden of relinquishing control from one application to another. The operating system takes care of it.

A single processor computer is shared among multiple applications with preemptive multitasking. Since the processor is switching between the applications at intervals of milliseconds, you feel that all applications run concurrently. Actually, this is not so. To have true multitasking, the applications must be run on a machine with multiple processors. Multitasking results in effective and simultaneous utilization of various system resources such as processors, disks, and printers.

2.2.1. Multithreading Definition

To understand multithreading, the concepts process and thread must be understood. A process is a program in execution. A process may be divided into a number of independent units known as threads. A thread is a dispatchable unit of work. Threads are light-weight processes within a process. A process is a collection of one or more threads and associated system resources. The difference between a process and a thread is shown in fig 6.19. A process may have a number of threads in it. A thread may be assumed as a subset of a process.

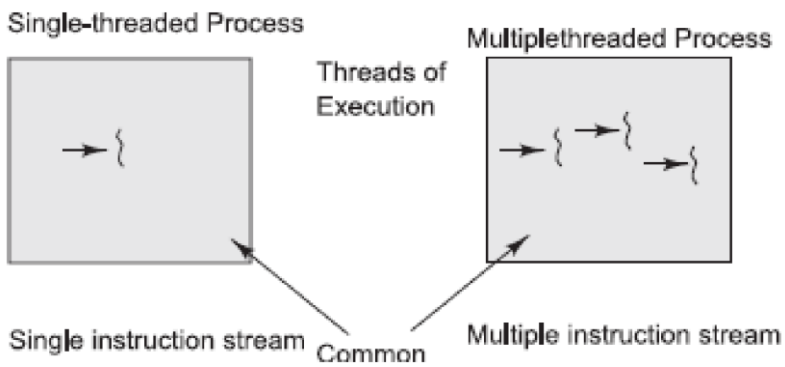


Figure 6.19. The difference between a thread and a process

If two applications are run on a computer (MS Word, MS Access), two processes are created.

Multitasking of two or more processes is known as process-based multitasking. Multitasking of two or more threads is known as thread-based multitasking. The concept of multithreading in a programming language refers to thread-based multitasking. Process-based multitasking is totally controlled by the operating

system. But thread-based multitasking can be controlled by the programmer to some extent in a program.

The concept of context switching is integral to threading. A hardware timer is used by the processor to determine the end of the timeslice for each thread. The timer signals at the end of the timeslice and in turn the processor saves all information required for the current thread onto a stack. Then the processor moves this information from the stack into a predefined data structure called a context structure. When the processor wants to switch back to a previously executing thread, it transfers all the information from the context structure associated with the thread to the stack. This entire procedure is known as context switching.

2.2.2. Threads Versus Processes

The differences between a thread and a process, can be clearly illustrated in the following points;

- A program in execution is often referred as process. A thread is a subset (part) of the process.
- A process consists of multiple threads. A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.
- A process is sometime referred as task. A thread is often referred as lightweight process.

- A process has its own address space. A thread uses the process's address space and share it with the other threads of that process.

- A thread can communicate with other thread (of the same process) directly by using methods like wait(), notify(), notifyAll(). A process can communicate with other process by using inter-process communication.
- New threads are easily created. However the creation of new processes require duplication of the parent process.
- Threads have control over the other threads of the same process. A process does not have control over the sibling process, it has control over its child processes only.

See fig. 6.20. It also illustrates the illustrates the items per process and per thread.

```
Per process items              | Per thread items
-------------------------------|----------------
Address space                  | Program counter
Global variables               | Registers
Open files                     | Stack
Child processes                | State
Pending alarms                 |
Signals and signal handlers    |
Accounting information         |
```

Figure 6.20. Per Process and Per Thread Items

2.2.3. Multithreading Benefits

Multithreading has many benefits that make it suitable and easy to use. Those benefits are;

1. **Improving Application Responsiveness**

Any program in which many activities are not dependent upon each other can be redesigned so that each activity is defined as a thread. For example, the user of a multithreaded GUI does not have to wait for one activity to complete before starting another activity.

2. **Using Multiprocessors Efficiently**

Typically, applications that express concurrency requirements with threads need not take into account the number of available processors. The performance of the application improves transparently with additional processors.

Numerical algorithms and numerical applications with a high degree of parallelism, such as matrix multiplications, can run much faster when implemented with threads on a multiprocessor.

3. **Improving Program Structure**

Many programs are more efficiently structured as multiple independent or semi-independent units of execution instead of as a single, monolithic thread. Multithreaded programs can be more

adaptive to variations in user demands than single-threaded programs.

4. **Using Fewer System Resources**

Programs that use two or more processes that access common data through shared memory are applying more than one thread of control.

However, each process has a full address space and operating environment state. Cost of creating and maintaining this large amount of state information makes each process much more expensive than a thread in both time and space.

In addition, the inherent separation between processes can require a major effort by the programmer. This effort includes handling communication between the threads in different processes, or synchronizing their actions.

5. **Combining Threads and RPC**

By combining threads and a remote procedure call (RPC) package, you can exploit nonshared-memory multiprocessors, such as a collection of workstations. This combination distributes your application relatively easily and treats the collection of workstations as a multiprocessor.

For example, one thread might create additional threads. Each of these children could then place a remote procedure call, invoking a procedure on another workstation. Although the original thread has merely created threads that are now running in parallel, this parallelism involves other computers.

2.2.4. Concurrency and Parallelism

In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution.

In the same multithreaded process in a shared-memory multiprocessor environment, each thread in the process can run concurrently on a separate processor, resulting in parallel execution. When the process has fewer or as many threads as the number of processors, the threads support system in conjunction with the operating environment ensure that each thread runs on a different processor. For example, in a matrix multiplication that has the same number of threads and processors, each thread and each processor computes a row of the result.

2.2.5. Looking at Multithreading Structure

Traditional UNIX already supports the concept of threads. Each process contains a single thread, so programming with multiple processes is programming with multiple threads. But, a

process is also an address space, and creating a process involves creating a new address space.

Creating a thread is less expensive than creating a new process because the newly created thread uses the current process address space. The time that is required to switch between threads is less than the time required to switch between processes. A switch between threads is faster because no switching between address spaces occurs.

Communication between the threads of one process is simple because the threads share everything, address space, in particular. So, data produced by one thread is immediately available to all the other threads.

In Solaris 9 and earlier Solaris releases, the interface to multithreading support was through specific subroutine libraries. The libraries are libpthread for POSIX threads, and libthread for Solaris threads. Multithreading provides flexibility by decoupling kernel-level and user-level resources. In the current release, multithreading support for both sets of interfaces is provided by the standard C library.

2.2.5.1. User-Level Threads

Threads are the primary programming interface in multithreaded programming. Threads are visible only from within the process, where the threads share all process resources like address space, open files, and so on.

2.2.5.2. User-Level Threads State

The following state is unique to each thread.

_ Thread ID

_ Register state, including PC and stack pointer

_ Stack

_ Signal mask

_ Priority

_ Thread-private storage

Threads share the process instructions and most of the process data. For that reason, a change in shared data by one thread can be seen by the other threads in the process. When a thread needs to interact with other threads in the same process, the thread can do so without involving the operating environment.

2.2.6. Multithreading in Python

2.2.6.1. Starting a New Thread

To spawn another thread, you need to call following method available in thread module:

thread.start_new_thread (function, args[, kwargs])

This method call enables a fast and efficient way to create new threads in both Linux and Windows. The method call returns immediately and the child thread starts and calls function with the passed list of args. When function returns, the thread terminates.

2.2.6.2. The Threading Module:

The threading module exposes all the methods of the thread module and provides some additional methods:

- threading.activeCount(): Returns the number of thread objects that are active.
- threading.currentThread(): Returns the number of thread objects in the caller's thread control.
- threading.enumerate(): Returns a list of all thread objects that are currently active.

In addition to the methods, the threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows:

- run(): The run() method is the entry point for a thread.
- start(): The start() method starts a thread by calling the run method.
- join([time]): The join() waits for threads to terminate.
- isAlive(): The isAlive() method checks whether a thread is still executing.
- getName(): The getName() method returns the name of a thread.
- setName(): The setName() method sets the name of a thread.

2.2.6.3. Creating Thread Using Threading Module

To implement a new thread using the threading module, you have to do the following –

- Define a new subclass of the Thread class.
- Override the __init__(self [,args]) method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.

Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run()method.

2.2.6.4. Example

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
```

```
    def __init__(self, threadID, name,
counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name


def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print "%s: %s" % (threadName,
time.ctime(time.time()))
        counter -= 1


# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)


# Start new Threads
thread1.start()
thread2.start()
```

```
print "Exiting Main Thread"
```

When the above code is executed, it produces the following result –

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013


Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

2.2.7. Synchronization Threads

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the Lock() method, which returns the new lock.

The acquire(blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.

If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The release() method of the new lock object is used to release the lock when it is no longer required.

2.2.7.1. Example

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):

   def __init__(self, threadID, name, counter):
       threading.Thread.__init__(self)
       self.threadID = threadID
       self.name = name
       self.counter = counter
   def run(self):
```

```
        print "Starting " + self.name
        # Get lock to synchronize threads
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # Free lock to release next thread
        threadLock.release()


def print_time(threadName, delay, counter):
   while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName,
time.ctime(time.time()))
        counter -= 1


threadLock = threading.Lock()
threads = []


# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)


# Start new Threads
thread1.start()
thread2.start()


# Add threads to thread list
threads.append(thread1)
threads.append(thread2)
```

```
# Wait for all threads to complete
for t in threads:
   t.join()
print "Exiting Main Thread"
```

When the above code is executed, it produces the following result –

```
Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread
```

2.3. Multi-Robot System Description

At this phase of the system we have more than one robot working together, actually the system has three forklifts to implement the required tasks working simultaneously. The multi-robot phase mainly considers the interaction among the system's three main executers, which are the forklifts, and the production line that holds all the objects the system is implemented upon moving and carrying.

2.3.1. Brain Prospects

At the beginning the robots are also placed in specific locations to avoid traffic jams in the warehouse.

The brain always checks the database if there is any product on the production line need to be served and also always checks the robots state if it is idle or busy. Concerning this matter we have the three following cases that can be probable to the brain at this phase.

I. **If there are no products on the production lines.**
When the brain accordingly gives an order to the robots to remain idle until a further announcement unless they were hired in another task.

II. **If the brain found products on the production lines, but the robots is found to be busy doing a task.**
When the brain accordingly waits any robot to be idle or to finish its task, then hire it.

III. **If there are products on the production lines and some robots are found to be idle.**
When the brain accordingly goes through its main function which we mainly talk about in the previous single robot section.

2.3.2. Brain Steps of Implementation:

The main steps that the brain follows to take a decision upon each

of the probable prospects that can be found in the system, are illustrated as follow;

1- Checks for the robot statue (idle or busy), and take idle ones.
2- The brain specifies the product line where there are products on need to be stored, this product line is specified by a point in the map we work on, and so we can consider this point as an end point to the robot to reach.
3- Runs Dijkstra algorithm on all idle robots to find the shortest path from each robot initial point to the point of the product.
4- Chooses the one with the shortest path, and hires it to do the task.
5- Changes the state of the chosen robot to “busy”.
6- When the robot reaches the specified production line, the brain gives an order to the robot to upload the product or the package from the production line.
7- According to the product or the package, the brain knows the corresponding point for storing this package from its database.
8- Then brain checks the flags of storing location, and chooses one of storing location according to its priority.
9- Also it changes the flag of the chosen storing location to be “0”, indicating that this place is to be filled with a package.
10- Now the brain assigns the robot with a task to carry the product to the appropriate storing location which is done by running Dijkstra algorithm for the second time to find the shortest path to the specified storing location.

11- At the end of the second path, or at the place of storage, the brain gives an order to the robot to download the package it carries in the specified point.

12- After the robot finishes its task, the brain returns the state of robot to “idle”.

13- The robot returns back to its specific location, also by applying Dijkstra's algorithm to find the shortest path, unless the brain hires it to do another task.

14- Finally, the process is always repeated during the warehouse work hours.

2.4. Multi-Robot Phase Simulation

The difference in this phase is that we want to simulate different robots working together in order to complete the storage process of different packages from different production lines.

The system simulation can be clearly illustrated in the following steps;

1. Firstly, each robot gets its path from its initial point to one of the production lines(specified by the system)to carry the package from there. As shown in fig. 6.21, each robot starts to move according to the path it has taken to the specified production line

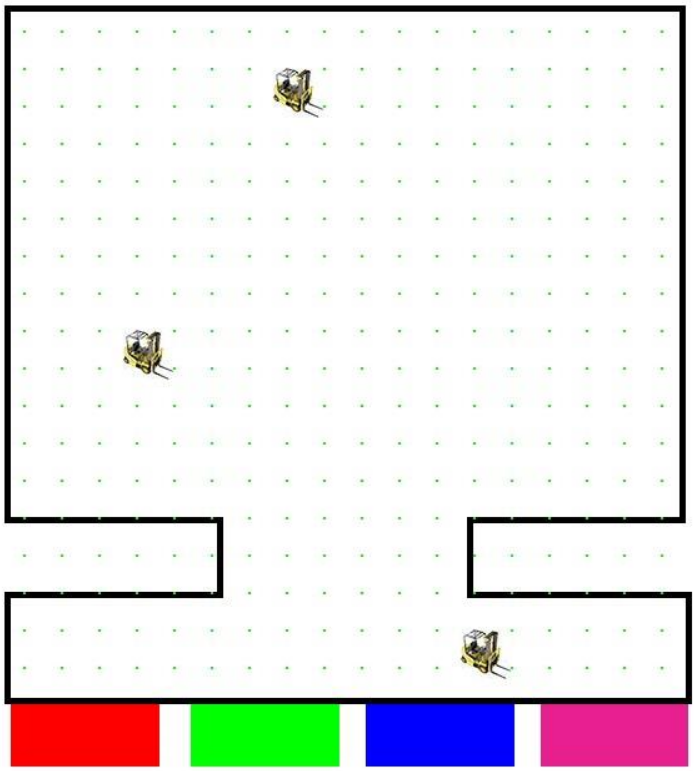


Figure 6.21.

2. After each robot has arrived to the specified production line, it uploads the corresponding package. Then, it gets the second path to the corresponding racks to store this package. As shown in figure 6.22 , each robot carries the package through the path to the racks.

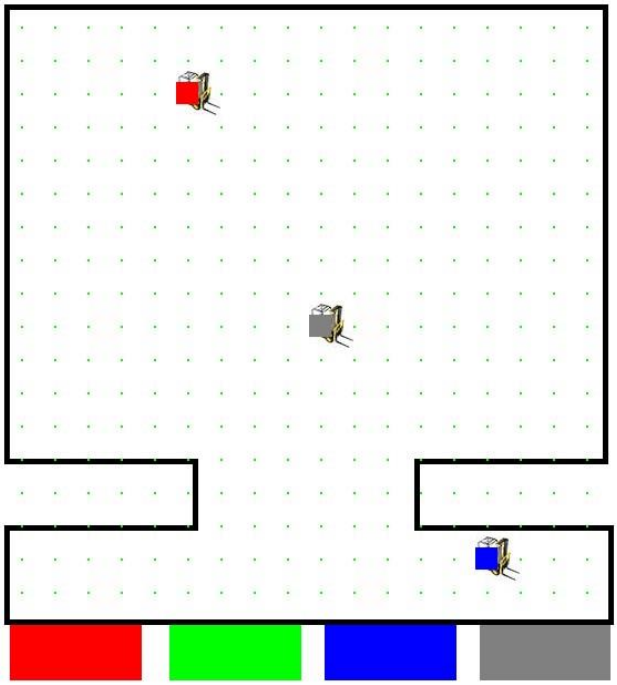


Figure 6.22.

3. Finally, each robot stores the package it carries in the racks corresponding to the type of the package. Then the system repeats until finishing the whole packages on the production lines. As shown below the robots has stored their packages and then begins to repeat the process. See fig. 6.23.

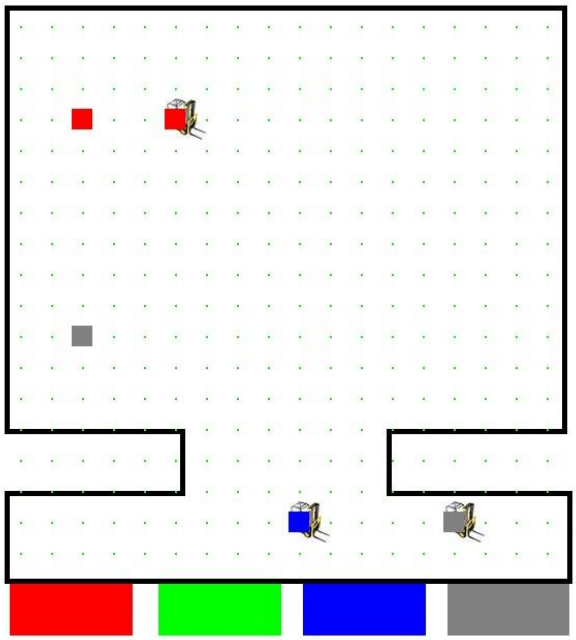


Figure 6.23.

2.5. Forklifts Collision

The main function works on three different threads for each robot (as we are working on three robots) that work simultaneously, each thread gives the specific path for each one of the three robots so each robot walks through its specific path, which can lead into collision between robots through their paths either to the production lines or to the storage locations.

This problem is solved through the hardware system by using an ultrasonic sensor on each robot, which stops the robot suddenly if it detects any obstacle or any other robot approaching it through its path.

Ultrasonic sensor detects the distance of the closest object in front of it. Working by sending out a burst of ultrasound waves and listening for the reflected echo when it bounces off an object.

It pings the obstacles with ultrasound waves.

Previously, Ultrasonic sensor has been clearly explained in chapter 2 as one of the system components.

Chapter 7

"Future Work"

1- **Managing the forklift robots using only Raspberry Pi boards without using Arduino boards to reduce the cost.**

We used Arduino board in motor control and controlling the crane of the forklift, after that we had to use the Raspberry Pi board for the barcode scanner interface and for the communication between other forklifts using the Wi-Fi module on the Raspberry Pi. So we can control all of this with Raspberry Pi only without using Arduino kits.

2- **Increasing the number of forklifts in the production line.**

Using two or three forklifts in a huge production line with a large amount of products wouldn't be efficient, so we need to increase the number of forklifts.

The system's idea is clear and efficient, so it can be generalized and doubled to be used in a more big system with more forklifts to bigger covered area.

3- **Using cameras with image recognition algorithms to be able to detect the required cargo.**

Instead of using barcode scanners to identify the boxes, we could use a real time processing camera with image recognition algorithms to identify the boxes.

4- **Using ZigBee protocol in communication.**

Changing the communication protocol used form Wi-Fi to ZigBee as it is simpler and less expensive, and testing the effect of this on the reliability of communication.

5- Real time monitoring system.

Implementing a real time monitoring system with a graphical user interface to reveal the exact location and current activity of every forklift.

6- Using Inertial Measurement Unit (Razor IMU) for forklift localization instead of line tracking algorithm.

Instead of using the old IMU module which has a great problem in accumulating the deviation error ending up with a different route for the forklift, and instead of using the line tracking algorithm which means that drawing the lines on the floor of the warehouse in mandatory, we can use the new Razor IMU module for the forklift localization which solves the error accumulation problem and contains more accurate gyroscopes and accelerometers.

7- Swarming algorithm.

Swarming behavior is a collective behavior inspired by insects such as ants and bees. Swarm robotics is a new approach to the coordination of multi-robot systems which consist of large numbers of simple physical robots. Each robot is quite simple, but

the emergent behavior of the swarm is more complex. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment.

7.1. Forklifts

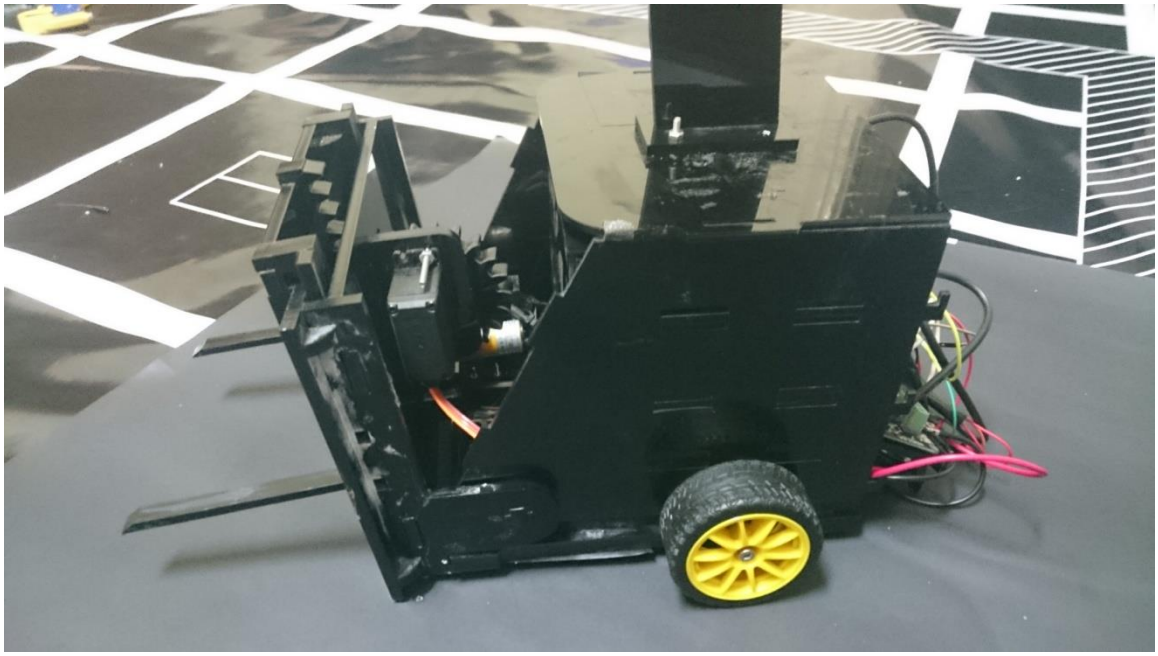


Figure 7.1.

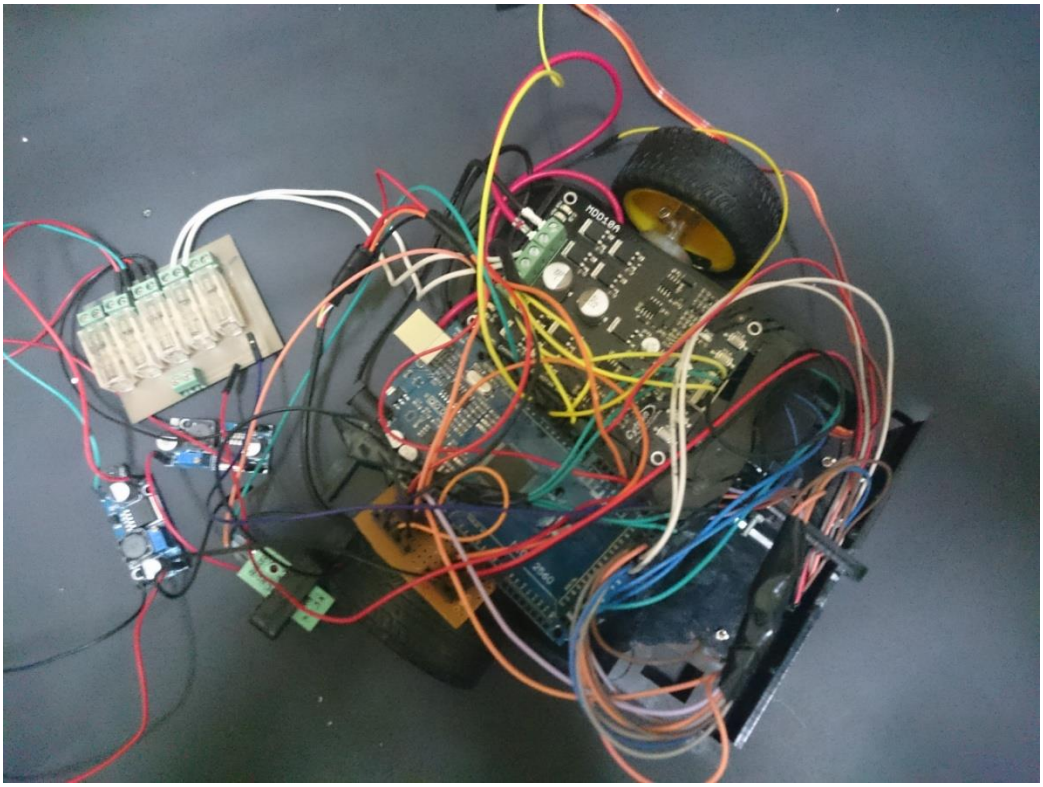


Figure 7.2.

Figure 7.3.

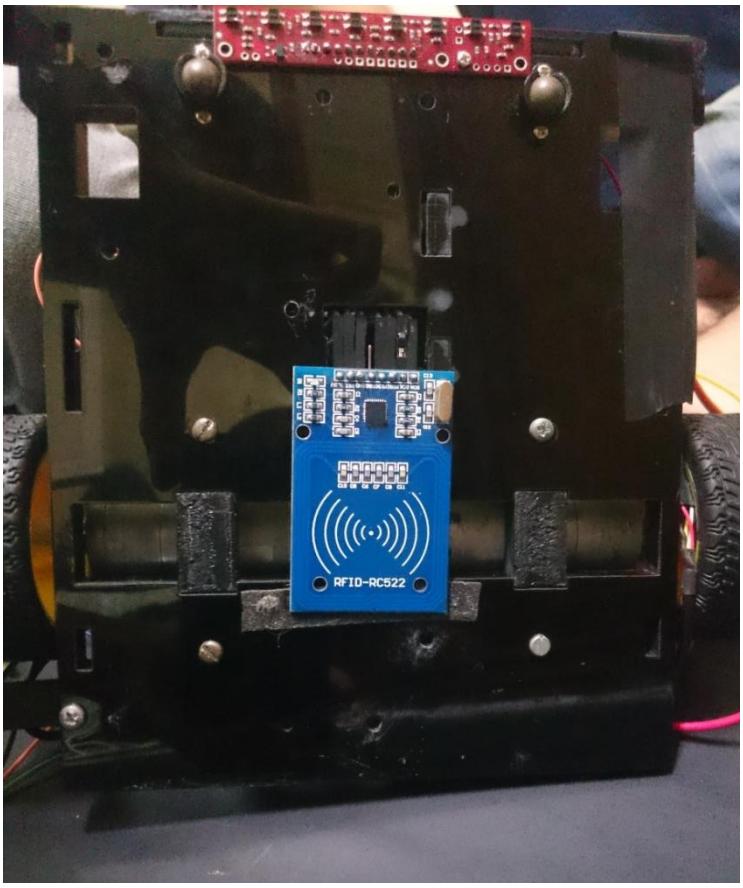
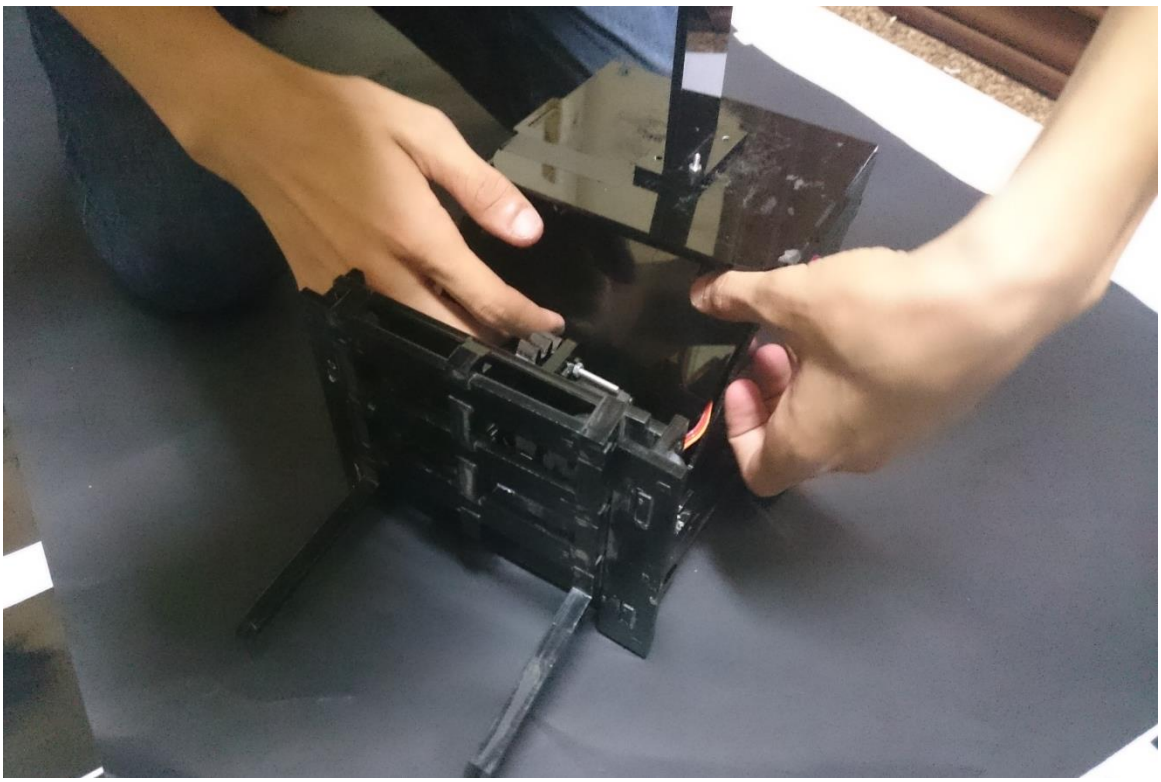


Figure 7.4. and Figure 7.5.



References

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Pressand McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the Association for Computing Machinery **34** (3): 596–615. doi:10.1145/28869.28874.
- Zhan, F. Benjamin; Noon, Charles E. (February 1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks". Transportation Science **32** (1): 65–73.doi:10.1287/trsc.32.1.65.
- Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, Jr., S. R.; Petry, R. M.; Seitz, R. N. (1957). Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems. Cleveland, Ohio: Case Institute of Technology.
- Knuth, D.E. (1977). "A Generalization of Dijkstra's Algorithm". Information Processing Letters **6** (1): 1–5. doi:10.1016/0020-0190(77)90002-3.
- Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). "Faster Algorithms for the Shortest Path

Problem". Journal of Association for Computing Machinery (ACM) **37** (2): 213–223. doi:10.1145/77600.77615.

- Raman, Rajeev (1997). "Recent results on the single-source shortest paths problem". SIGACT News **28** (2): 81–87. doi:10.1145/261342.261352.
- Thorup, Mikkel (2000). "On RAM priority Queues". SIAM Journal on Computing **30** (1): 86–109. doi:10.1137/S0097539795288246.
- Thorup, Mikkel (1999). "Undirected single-source shortest paths with positive integer weights in linear time". journal of the ACM **46** (3): 362–394.doi:10.1145/316542.316548.
- http://www.pygame.org/
- http://www.tutorialspoint.com/python/python_database_access.htm
- http://slav0nic.org.ua/static/books/python/Python%20-%20Database%20Application%20Programming%20Interface.pdf
- https://www.mysql.com/why-mysql/topreasons.html
- SUN microsystems: Multithreaded Programming Guide
- http://www.tutorialspoint.com/python/python_multithreading.htm
- https://www.seeedstudio.com/item_detail.html?p_id=2521
- http://www.link-labs.com/category/m2m-iot-applications/