Helwan University - Faculty of engineering
Communication & Information Program
Graduation Project

Presented by:

Mohammed Khaled Makbool
Nour Eldin Khaled Mohamed

Supervisor:

Assoc. Prof. Fatty M. Salem

Graduation Project Report

Bachelor's Degree in communication & IT Engineering at Helwan University, Cairo, Egypt

July 2022

Contacts

name	email	phone
Mohamed khaled	M.Makbool@outlook.sa	01012214893
NourEldin khaled	nourkhaled2650@gmail.com	01126507835

Acknowledgment

Apart from our efforts, the success of the project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We would like to thank our supervisor, Assoc. Prof. Fatty M. Salem, for bringing the weight of his considerable experience and knowledge to this project. Her high standards have made us better at what we do.

To all our friends and family for helping us survive all the stress from this year and not letting us give up.

We are deeply grateful to all the staff members of the HUCI community, Miss Marwa, and our student advisor Dr. Zaki.

Finally, we wish to acknowledge Helwan University for providing us with a graduation project, without which this work could have never begun.

Abstract

As the world is going into centralizing everything and instead of having thousands of stores and every one of them has its own warehouse we have now a few mega stores with vast warehouses, and when things go big we directly think of automation to save time and effort, and when things go huge, automation becomes a must not be just an option and that is what our project is working on.

SWMS is a prototype of a fully automated warehouse controlled by few human operators and many smart robots which can move smoothly through the entire warehouse and transfer products from gates to shelves and vice-versa

Table of Content

List of Figures

Chapter 1: Introduction

Problem Definition and importance

Warehouse management involve organizing, managing, and maintaining all the processes that occur in a warehouse so that they run as smoothly and efficiently as possible.

According to a report by McKinsey & Company in 2019, about £300 billion (approximately $385 billion) is spent each year, worldwide, on overall warehousing costs. And that amount doesn't include the additional costs of correcting errors and mistakes.

We will cover some of the most common problems faced in warehouse management.

1. Accidental redundancy

Definition:

to create workflows. But if a workflow isn't well organized, you may find that the same operation is accidentally performed more than once. These redundancies increase your labor costs and take up extra time when you have to go back and reverse the mistake. This happens more often in large warehouses than in smaller ones, since there's more space and more inventory to deal with.

Importance:

Redundancy is often noticed in order picking, which is when products are picked from their storage locations in a warehouse to fulfill an order.

In a larger warehouse, multiple people work together to pick products from different parts of the warehouse to fulfill a single order. Since the same order is passed around to multiple people, there's a bigger chance for mistakes, like picking too many of the same products.

2. Messy warehouse layout

Definition:

Over the years, rising storage costs have pushed warehouse managers into making more efficient use of their warehouse space. But a survey conducted by Logistics Management back in 2018 suggests that it doesn't always work—the average warehouse capacity utilized by manufacturers was only around 68%. Not having enough storage because of ineffective use of space is still a common pain point in warehouses.

Importance:

Putting together an optimal warehouse layout can solve this problem. This includes maximizing the use of the floor space and vertical space while leaving enough room for warehouse employees to pass through. It also means looking into ways to use automation and equipment to reduce labor and labor costs, improving the accessibility of products in the warehouse, categorizing inventory in a systematized way, and ensuring that inventory is stored safely.

3. Bad inventory management

Definition:

-Expecting to find a product in a specific location, but realizing that it's actually placed somewhere else.

-Accepting an order on the assumption that you have enough stock to fulfill it, and only later finding out that you don't. Now you have to place a backorder, which significantly extends your order lead time.

-Denying an order after assuming that you don't have enough stock to fulfill it, but then finding out that you do.

-Trying to put away stock that you've received but having trouble finding where to place it.

Importance:

According to a study conducted by Wasp Barcode Technologies, 43% of small businesses either don't track inventory or use a manual method. Another survey by Peoplevox found that 34% of businesses

have delayed shipping because the products mentioned in the order were not actually in stock. Miscalculations can easily happen when inventory checking processes are done manually since this leaves plenty of room for human error. Sometimes they can also happen when using outdated software.

4. Excessive spending on labor

Definition:

There are several types of tasks in a warehouse that labor workers are employed to handle such as general labor like cleaning, forklift operator, material handler, shipping and receiving monitor, shipping specialist, loader, product picking, stock clerk, and overall warehouse manager. With all the technology available today, most warehouses are looking for ways to invest in it.

Importance:

There's a common notion that only automation and equipment are expensive. But what several warehouse managers fail to realize is that manual labor doesn't come cheap either. According to Kane Logistics, labor is one of the biggest expenses some warehouses spend on, ranging from 50-70% of the overall warehousing budget.

Our Solution

Depending on the size of the warehouse there will be many problems with many available solutions and most of the modern solutions depend on automation whether to avoid human error, reduce operational expenses or even reduce the order lead time and that is what our project trying to do through automating every process we call this solution SWMS

Project Idea

SWMS is a system built from 2 separate systems first one we call the slaves which are robotic machines that have pieces of equipment to move from a certain point to a certain destination and to receive instructions and send information to the second system that we call the mastermind which is responsible to manage the slaves and how they can work together to achieve the goal of their existence which is transferring products from shelves to gates where other machines will be responsible to deliver it to the customer.

Objectives

1. building a guide system so slaves can use it to transport.
2. designing small prototypes of the slaves to test the operational process.
3. building a model of the master mind to control the slaves.
4. making a communication system to be used by the mastermind and the slaves.

Chapter 2: Previous Work and Market Survey

Warehouse Management

In the past, warehouse management was very paper-intensive in its coordination of a multitude of activities. This has changed with the introduction of warehouse management system software.

Warehouse management systems (WMS) assist managers in tracking products throughout the entire storage and distribution process. These systems span from simple computer automation systems to high-end, feature-rich management programs that improve order picking, facilitate better dock logistics and monitor inventory management.

History

Warehousing's roots go back to the creation of granaries to store food, which was historically available for purchase during times of famine. As European explorers began to create shipping-trade routes with other nations, warehouses grew in importance for the storage of products and commodities from afar. Ports were the major location for warehouses.

As railroads began to expand travel and transportation, the creation of rail depots for the storage of materials became necessary. In 1891 the American Warehousemen's Association was organized to challenge the railroad companies' control over freight depots. President Theodore Roosevelt significantly strengthened the Interstate Commerce Commission with the passage of the Hepburn Act in 1906. Commercial warehousing began to grow after the government placed more restrictions on railroads.

World War II impacted warehousing in several ways, including the need to increase the size of warehouses and the need for more mechanized methods of storing and retrieving products and materials. As mass-production grew throughout manufacturing, the need for efficient and effective warehousing capabilities grew with it.

Warehouse Functions

Warehousing is a key component of the overall business supply chain. The supply chain consists of the facilities and distribution options for the procurement of materials from the manufacturer to the customer and all points in between. It includes the production of materials into components and finished products and then the distribution to customers.

Warehouse functions include:

- Storing goods to permit the management of product flow or to accommodate longer production runs
- Serving as a mixing point where products from different suppliers are mixed and then distributed to fulfill customer orders
- Serving as a sales branch and customer service location
- Serving as a source of supplies for production
- Serving as a staging area for final packaging or finishing

Warehouse Operations

Warehouses operate in several ways. Public warehousing involves the client paying a standard fee for storing merchandise. Private warehousing is storage and operations

controlled completely by a single manufacturer. Leased warehousing is an option for more stable inventory. Contract warehousing clients pay fees regardless of whether they are using the space or not; this ensures the space is always available for them to use. According to Overview of Warehousing in North America, contract warehousing accounts for more than 60 percent of the U.S. commercial market.

A warehouse stands empty without some form of the product. Delivery of goods and materials takes place either by truck, rail, or boat on a dock or loading area. The goods are received, processed, and then sent into the warehouse for storage.

The storage of goods has been the primary function of warehouses. Once the goods have been received from the manufacturer and/or shipper, they are compactly stored to maximize space within the facility. Products are

placed on pallets, which allow for more consistent stacking and moving within the facility.

Contract and public warehouses receive goods and products from a multitude of manufacturers and shippers. A crucial aspect of warehouse management is inventory control. Inventory control is the ability to locate and track a given product within the warehouse to facilitate quick selection and loading for order fulfillment. It is also the process of maintaining sufficient amounts of product to meet customer demands, while at the same time balancing the expense of keeping products in storage. Perpetual, annual, physical, and cycle counting are all methods of keeping track of inventory.

Order picking is the process of selecting products to fulfill an order. There are several picking methods:

- Discrete or pick-by-order: Specific products are selected on a per-order basis.
- Batch or pick-by-article: Multiples of a product are selected to fulfill multiple orders. The products are sorted in the staging area and combined with other products to fulfill the orders.
- Wave: Products are gathered based on specific routing or shipping criteria.
- Reverse-order: This is used when part of an order is held to be combined with another order.

Reverse-order picking is related to cross-docking, another function of warehouses. Cross-docking is a direct flow of goods from receiving to shipping, with little if any storage. Cross-docking is contingent on the timely delivery of products, accurate management of the loading dock, and effective ordering by the customer.

Warehousing is also involved in the packaging and labeling of a product as it moves through the facility. Proper packaging is necessary for effective storage and to guard against damage. Labeling, or tagging, is an important element of the packaging. Proper labeling improves the ability to identify, track, store, and select the correct product for order fulfillment.

Once the product has been selected or picked, it is brought to a staging area for final processing and shipment. The loading dock is a hub of activity as products are arriving for storage and being staged for distribution. Effective management of this area is crucial for warehouse success. It is here that cross-docking takes place.

The final stage of warehousing is the transportation facet of delivering and shipping goods.

Trends

Since the mid-1990s, warehousing and distribution operations have faced a wide variety of emerging business trends, including the rise of the Internet, a large number of mergers and acquisitions, and an increase in global trade. The warehouse industry found itself recovering from a recession at the start of the twenty-first century, partially brought on by the hype of the dot-com bubble and the excess production created after it burst. It also coped with new methods of distribution, such as just-in-time (JIT) production, where warehousing is unnecessary because products are shipped directly to customers. Additionally, the buying habits of everyone from manufacturers to consumers have dramatically changed, partly in response to improved communications technology and greater global competition. According to a 2004 Warehousing Management survey, competition in warehousing has become extremely tight because businesses seek warehouse firms with extremely thin margins. Companies are succeeding by remaining flexible and investing in the technological advances that are required to improve product tracking and increase efficiency.

Warehousing companies are now striving to become more than simply storage facilities. They are transforming themselves into third-party logistics providers or “3PLs” that provide a wide array of services and functions. In addition to packing and staging pallets, contemporary warehousing facilities offer light manufacturing, call centers, labeling, and other non-storage options. An outcome of the increased 3PL activity is a wave of mergers that are consolidating the industry. Customer demands for one-stop shopping and new technologies are a driving force behind this consolidation. Further development is the rise of fourth-party logistics providers (4PLs), who are essentially asset-less companies that use computer resources to supply 3PL services.

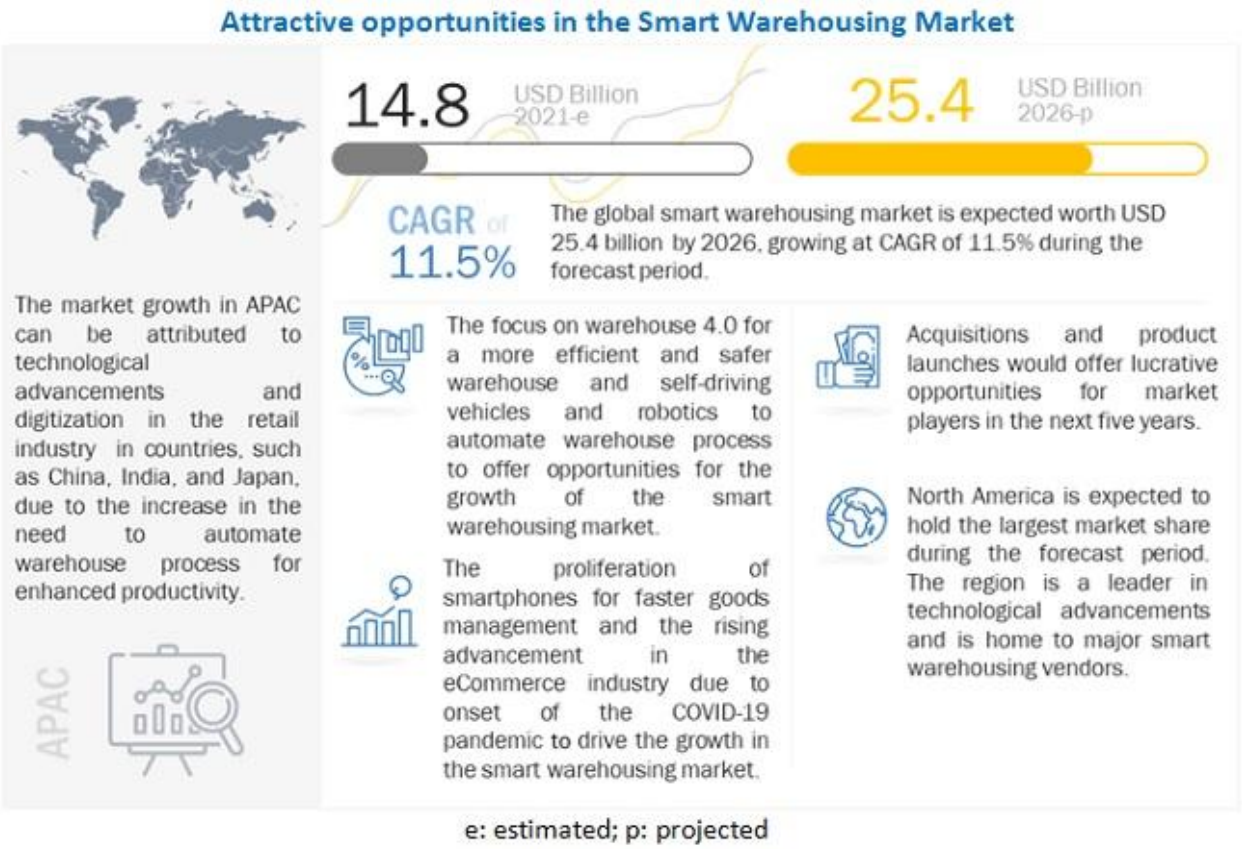
Other trends in warehousing include radio frequency identification (RFID) tags, transportation management systems, pick-to-light technology, and voice-activated receiving and packaging. Voice-activated receiving and packaging allow for warehouse personnel to speak requests into the WMS, thus speeding the entire process. Transportation management systems provide an advanced level of detail on goods prior to their arrival and also provide a

more specific time of delivery. RFID has dramatically improved the ability to effectively manage inventory and track the location of specific goods within the warehouse. Pick-to-light technology improves order picking along warehouse conveyor belts by monitoring and identifying products for specific shipments. Continuous improvements are being made in these technologies. For example, in 2007 a Danish company introduced a passive RFID with privacy features that include encryption, a built-in firewall, and a silent mode. Also in 2007, Hitachi introduced an RFID device thin enough to be embedded in a sheet of paper.

Warehousing is a mature industry seeking methods to maximize profits and striving to add services to compete for customers. The warehousing industry is a key component of the supply chain and will likely remain so as long as there are manufacturers and consumers

Smart Warehouse

The global smart warehousing market size to grow from USD 14.8 billion in 2021 to USD 25.4 billion by 2026, at a Compound Annual Growth Rate (CAGR) of 11.5% during the forecast period. Various factors such as the proliferation of smartphones for faster goods management, the rising advancement in the eCommerce industry due to the onset of the COVID-19 pandemic, the emergence of multi-channel distribution networks, and the dynamic nature and globalization of supply chain networks are expected to drive the growth of smart warehousing market.



due to an increased demand for priority shipping and made-to-order products, which can be attributed to the rise of eCommerce.

Supply Chains need to be prepared to cater to this large and ever-growing consumer demand. This demand can only be catered to if the supply chain processes, or in this case, warehousing processes are advanced enough to match the efficiency required.

An ideal Smart Warehouse has end-to-end ‘smart’ automation, right from a sophisticated Smart Warehouse Management System (WMS) to Automated Truck Loading Systems (ATLS). This saves a lot of time and money and can increase the accuracy of operations throughout the whole lifecycle. While WMS seems like an obvious starting point, many warehouses don’t move beyond a smart WMS.

It is not just limited to increased operational efficiency. Even the decision-making can be handed over to automated systems. For example, whenever a batch of inventory is dispatched into a warehouse, the WMS can

make a decision as to where the pallet should be placed, depending on the category of inventory or any other criteria set by the enterprise.

What are the benefits of Smart Warehousing?

Increased Labor Efficiency:

Tasks such as inventory scans, picking inventory, updating WMS data, etc. are repetitive in nature but are critical enough to be done frequently. These tasks are easy to automate since there is no creative thinking of humans required for them. Artificial Intelligence (AI) powered technologies can be instrumental in automating these tasks. This can give warehouses the flexibility to shift their human resources towards tasks that require creative thinking, thus ensuring optimum management of labor.

Instead of increasing the amount of personnel, warehouses can utilize those salaries to hire/retain staff for more important tasks that require human intelligence.

Reduced Costs:

Labor costs are a huge cause of burning pockets of supply chains, with them making up two-thirds of the operating costs (source). These costs reach astronomical levels for bigger and more dynamic warehouses and can have a compounding effect on warehouses that are experiencing growth.

During a slowdown, the labor salaries can be tremendously saved since warehouses don’t need to pay for unproductive hours, and instead, have technologies do the work as and when required.

The costs of managing labor aren’t just about wages. Warehouses can end up spending on fighting lawsuits from workers or unions for a variety of reasons, thus ending up draining precious resources.

While automation may seem like a lot of upfront investment, a deep dive into the ROI or payback periods they provide paints a different picture. For example, autonomous inventory drones generally deliver a payback period of just less than a year for warehouses in developed economies such as the USA, Canada, UK, Germany, etc.

Time Efficiency:

Whether it is in the Supply Chain industry or otherwise, a commonly acknowledged fact about automation is that it is known to save a lot of precious time required to do certain tasks.

Warehousing is no different. In fact, the time-saving factor is augmented when one considers the fact that there are not just one, but several repetitive tasks involved in warehousing operations. Right from unloading goods off the truck to scanning the inventory, almost all the tasks are repetitive and mundane that don't require human intelligence.

Such tasks can and should be automated. Smarter solutions can carry out the same tasks more quickly and with added benefits. An ATLS can load multiple heavy goods inside a truck at a quick rate, as compared to a human loading one or two items at a time.

Saving a lot of man-hours has a compounding effect. The time saved can be put to use for productive tasks, which can have a positive impact on overall customer satisfaction.

Data-driven Decision Making:

Data is the new oil only if it is managed correctly and put to optimum use. Else, it can well turn into a chaotic mess with no clear output.

In a smart warehouse, the end-to-end collection and segregation of data are done effectively. This means that relevant

data is collected from the relevant source, and has a clear actionable intent. Let's check three examples below:

-Maersk uses a Remote Container Management (RCM) system to check the status of temperature, location, and power supply for their containers.

-UPS found that trucks turning left was costing them a lot of money, so now their drivers have minimized or eliminated taking left turns. Left turns now constitute only 10% of their routes (yes, you read that right!)

-DB Schenker has deployed a Decision Support Tool to simulate daily scheduling and optimize overall operations.

COVID-19 Impact

The COVID-19 pandemic disrupted global supply chains and has outstripped and impacted both supply and demand at a higher pace. But the impact of COVID on the warehouse is going to have long and lasting effects. As the virus spreads throughout the world, there is an outbreak or transportation delay in one part of the world that would have a devastating impact across the globe, causing shutdowns due to warehouse closures or missing or delayed supplies. At the warehouse, some organizations were left with excess inventory they could not ship sitting in inventory and a few other firms were at a standstill as they waited to receive inventory at their depleted warehouse. Some industries witnessed unprecedented demand while other industries saw the demand fall, leading to a negative impact on the overall smart warehousing market. Post COVID-19.

Challenges Towards Embracing Smart Warehouses

Like every other technology, smart warehouses do come with their own set of challenges. The challenges are further augmented when supply chains look to scale their warehousing operations across locations. A few challenges that we have observed are -

High costs:

Despite the technologies of smart warehouses proving to deliver good ROIs, the higher implementation cost, in the beginning, turns away many supply chain managers. Due to the sheer scale and sophistication of these technologies, one cannot deny that they do demand high upfront investment.

Operational changes:

While some parts of a smart warehouse, such as an advanced WMS or inventory drones, do not require warehouses to change their configuration or modus operandi, many technologies do require operational changes in the warehouse. Thus the transition from a traditional warehouse to a smart warehouse can be time-consuming, especially if the warehouse managers aren’t keen on changing their present operations or layout.

Downtimes:

This can easily qualify as one of the biggest challenges since this can result in a significant loss of time and money if not solved in a timely manner. If any component of the smart warehouse experiences downtime, it can affect the workflow of the operations. This could further be a problem if all the technologies inside a warehouse are integrated with each other, thus resulting in a snowball effect.

Key Market Players

The major players in the smart warehousing market include Oracle (US), SAP (Germany), Manhattan Associates (US), PSI Logistics (Germany), PTC (US), Tecsys (Canada), Reply (Italy), IBM (US), Infor (US), Korber (Germany), Generic (France), Microlistics (Australia), Blue Yonder (US), Vinculum (India), Epicor (US), Softeon (US), 3PL Central (US), Synergy Logistics (US), BlueJay Solutions (US), Mantis (US), WareIQ (India), Foysonis (US), Logiwa (US), Increff (India), Locus Robotics (US), ShipHero (US), Orderhive US), EasyEcom (India), Unicommerce (India), and IAM Robotics (US). Partnerships, new product developments, mergers and acquisitions, product enhancements, and acquisitions are the key strategies implemented by major vendors in the smart warehousing market.

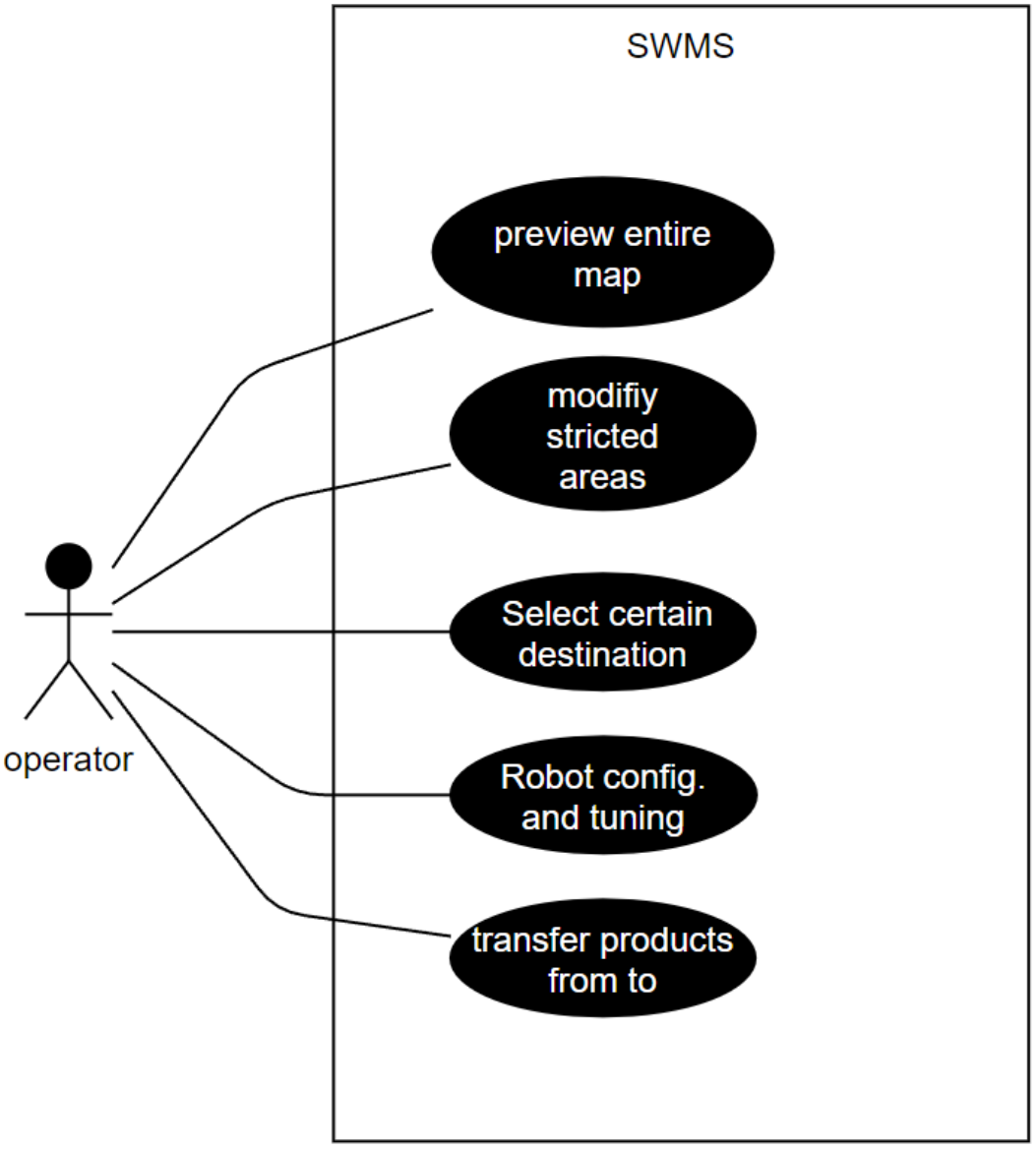
Chapter 3: System Component

Overview

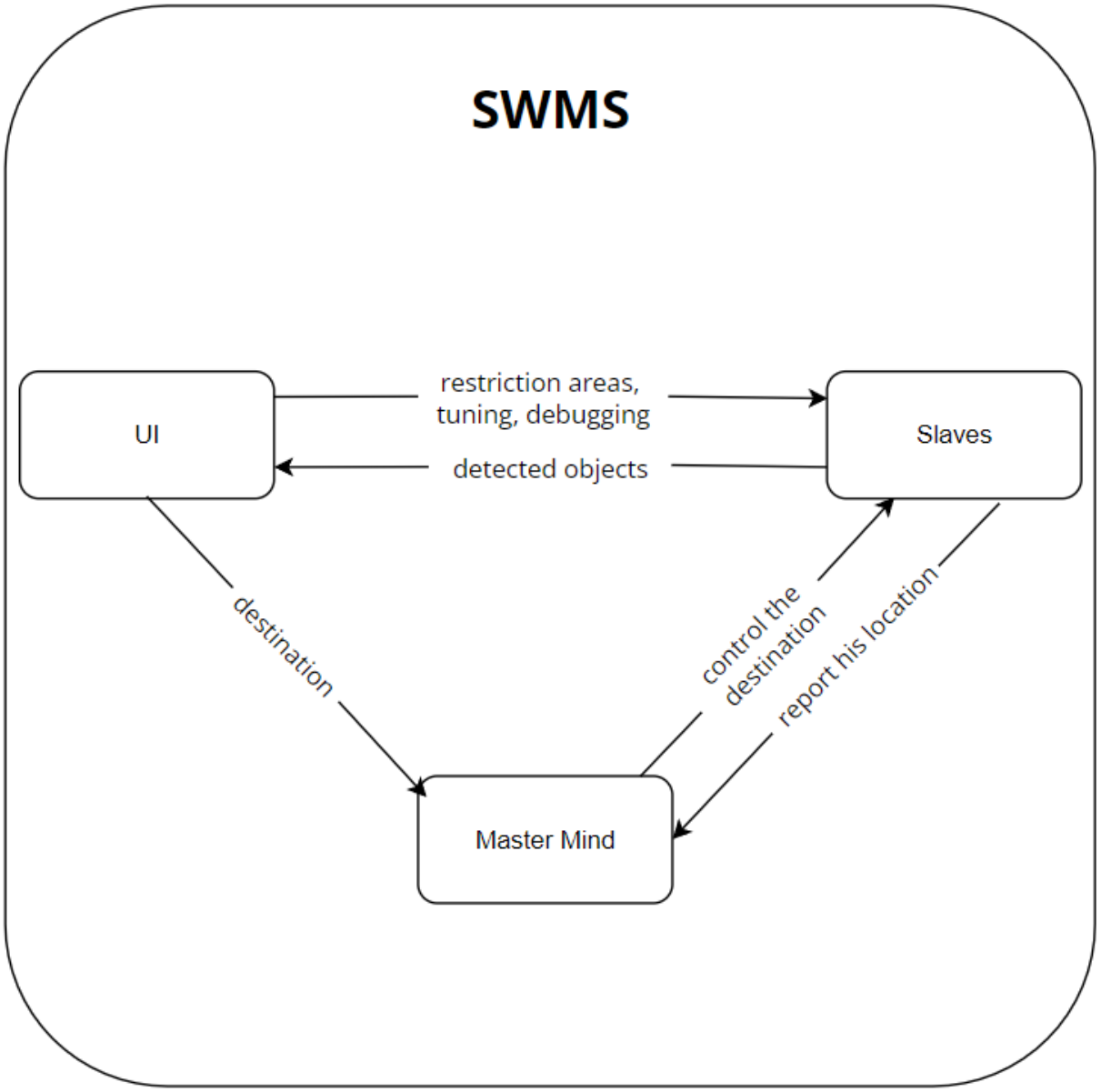
Designing and implementing a system to manage warehouses via Robots smartly and efficiently controlled and managed by IoT(Internet of Things) protocols. The system will be connected to the internet and synchronized with the store (or the company) website to take the orders from customers and detect the location of the package to send the nearest robot to the package to collect it and prepare it to delivery. Robots will have sensors that can detect the warehouse state, temperature and …more. Robots can also find obstacles and walls and exchange this knowledge over the system.We will be able to develop the system and add robots easily without modifying anything in the core system thanks to the shared knowledge throw the IoT system.

System Block Diagram

Process Use Case



Internal Block Diagram



Communication System

In the communication part we used two protocols: Bluetooth and MQTT using WiFi.

1. Bluetooth

a small rang communication protocol we are using to make direct communication to the robots and configure it and make any maintenance.

2. MQTT over WiFi

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

The MQTT IoT communication protocol must have one broker -to manage the received and sent messages- and multi nodes work as clients.

In our solution we treat the slave robots as clients and run a brocker on any computer on the network -we also trayed to run the brocker in ESP8266 small microcontroller SoC and it works- and the UI can join the network as client also from any computer to control the system.

Slaves Robots

The heart of the project it's a multi robot system connected to the network and interacting with it. The main purpose of these robots is taking the shortest path to the destination point to carry the package and deliver it to the warhouse gate. Eache robot has an independent computing unit, sensors, and actuators.

1. Microcontroller

We decided to choose the NodeMCU-32 development board as the main controlling unit of the robot because of its high computing power and low energy consumption and it has widely useful built in peripherals and also it contains a builtin WiFi and Bluetooth and it's well known in the IoT field.

Hybrid Wi-Fi & Bluetooth Chip

ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. ESP32 can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

High Level of Integration

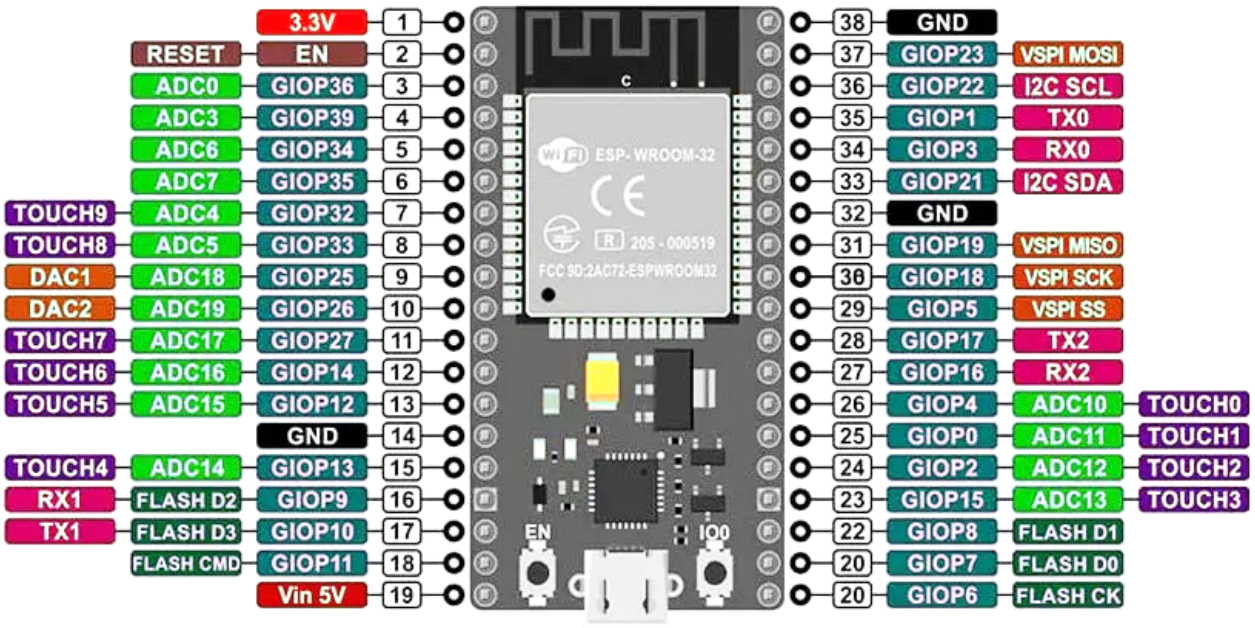
ESP32 is highly-integrated with in-built antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules. ESP32 adds priceless functionality and versatility to your applications with minimal Printed Circuit Board (PCB) requirements.

Ultra-Low Power Consumption

Engineered for mobile devices, wearable electronics and IoT applications, ESP32 achieves ultra-low power consumption with a combination of several types of proprietary software. ESP32 also includes state-of-the-art features, such as fine-grained clock gating, various power modes and dynamic power scaling.

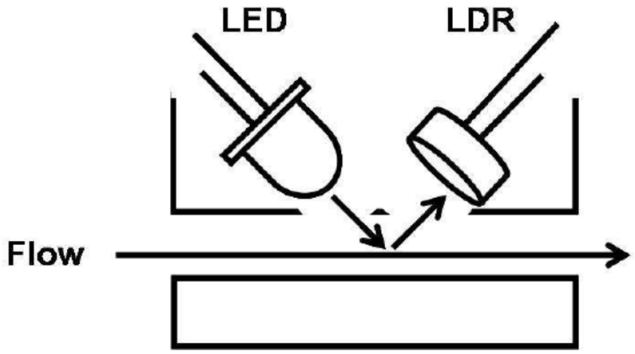
Robust Design

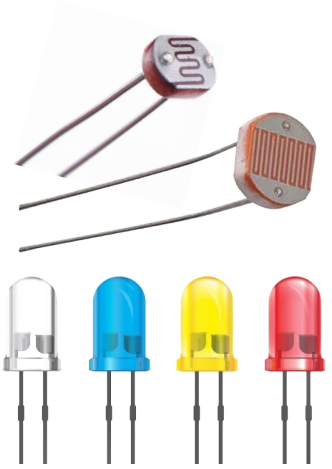
ESP32 is capable of functioning reliably in industrial environments, with an operating temperature ranging from –40°C to +125°C. Powered by advanced calibration circuitries, ESP32 can dynamically remove external circuit imperfections and adapt to changes in external conditions.



2. Sensors

The robots use the LDR (Light Dependent Resistor: Photoresistor) as main sensors for line follower and wall/object detection. And the robot has LEDs that act as light transmitters for the sensors case of low light conditions.





We chose the LDR because of its cheap price and widely available in the market. The LDR has a high latency factor and it's some kind of noisy components but we can handle it with some signal processing and some software algorithms for filtering.

3. Motors

The main actuator in the robots is an N20 geared motor and wheel. We chose it because of its small size and high torque and it will be very suitable for this prototype.

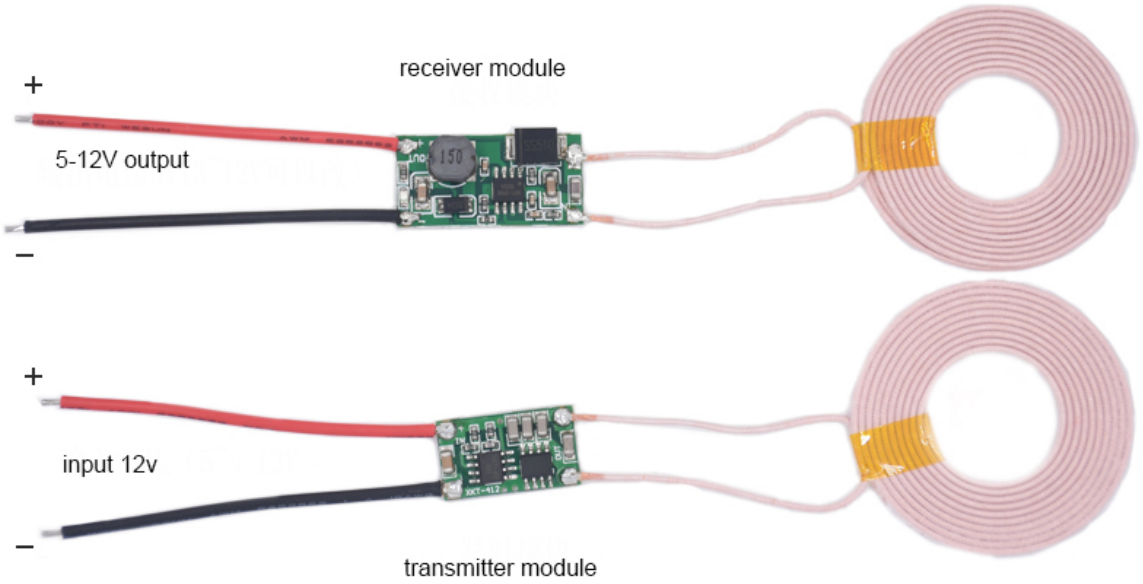
And it comes with all necessary parts to implement it to the robot. And it comes with a big variety of speeds and torques and different operating voltages.

4. Batteries

We go with rechargeable Panasonic NCR18650B Li-ion Batteries to benefit from high power with reasonable price and it comes with a huge capacity (3200mAh).

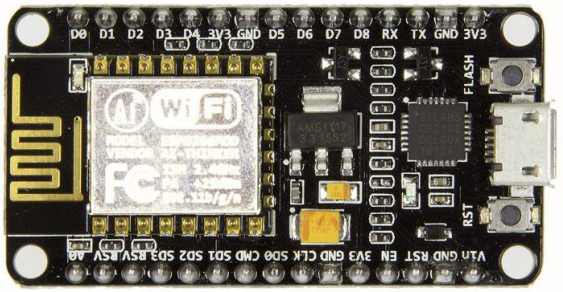
5. Wireless Charger

Wireless charging is a good feature that we can add to the robots to make it easier to charge and don't need any human intervention. With it the robots can automatically go to the charging station and start charging.



Mastermind

The mind of the system takes any order and decides which robot is near to the pickup point and sends it. Any computer or micro controller can act as a mastermind and we chose to make it small and power saver so we chose a NodeMCU with ESP8266 SoC.



ESP2866

High Durability

ESP8266EX is capable of functioning consistently in industrial environments, due to its wide operating temperature range. With highly-integrated on-chip features and minimal external discrete component count, the chip offers reliability, compactness and robustness.

Compactness

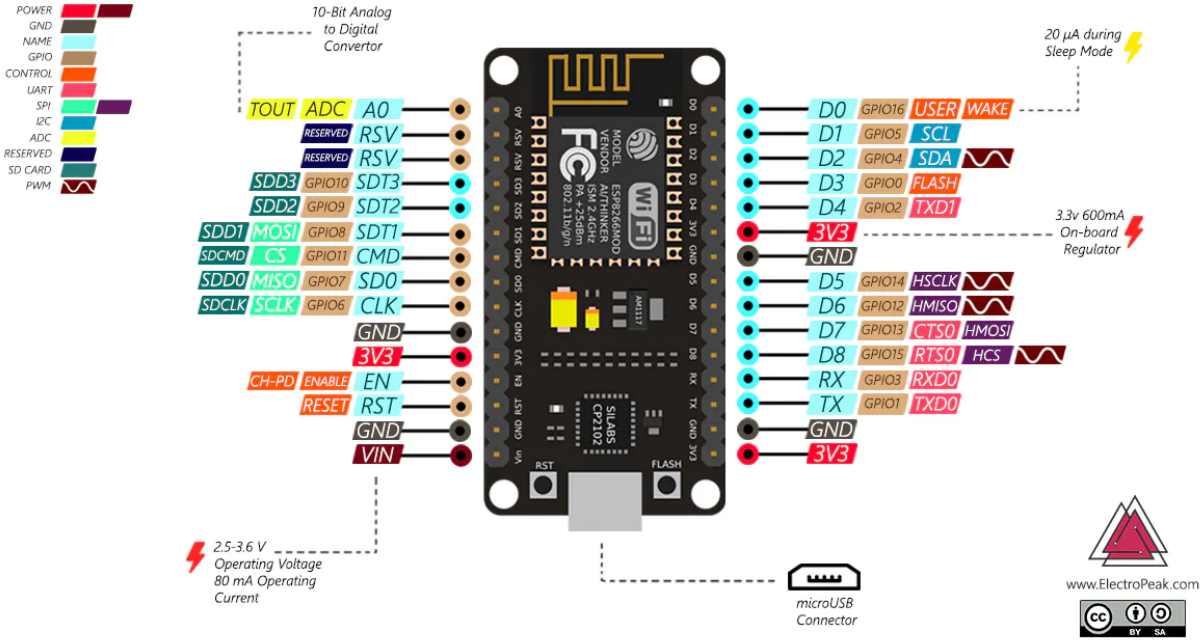
ESP8266EX is integrated with a 32-bit Tensilica processor, standard digital peripheral interfaces, antenna switches, RF balun, power amplifier, low noise receive amplifier, filters and power management modules. All of them are included in one small package, our ESP8266EX.

Power-Saving Architecture

Engineered for mobile devices, wearable electronics and IoT applications, ESP8266EX achieves low power consumption with a combination of several proprietary technologies. The power-saving architecture features three modes of operation: active mode, sleep mode and deep sleep mode. This allows battery-powered designs to run longer.

32-bit Tensilica Processor

The ESP8266EX microcontroller integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow about 80% of the processing power to be available for user application programming and development.



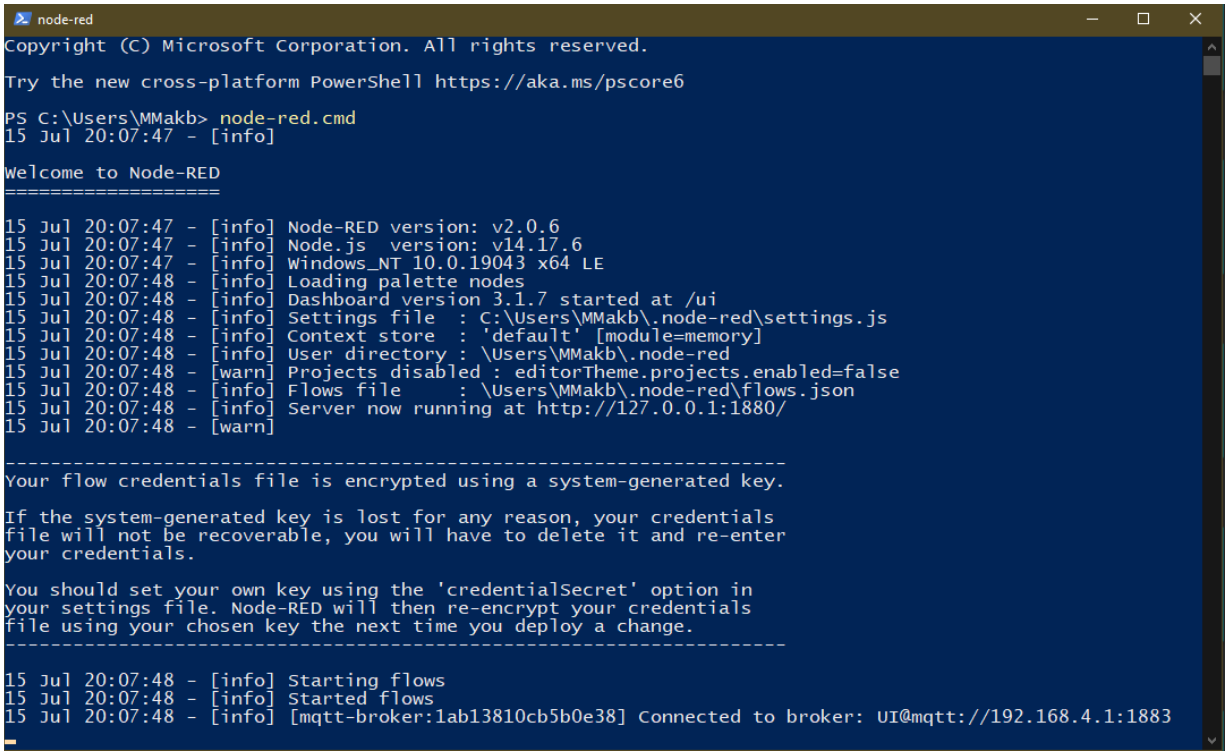
User Interface

For the user interface we chose the Node-Red platform to design it because it's easy to use and fast to implement and widely known in the IoT field.

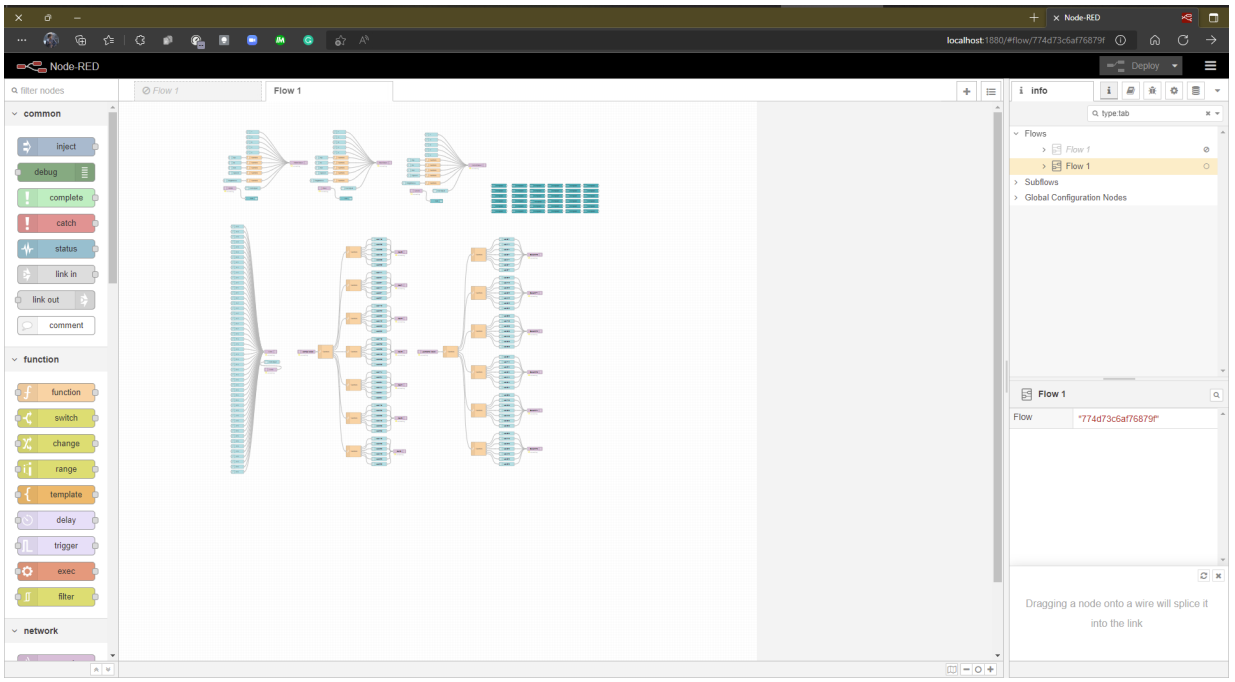
Node-Red

Node-RED is a programming tool for wiring together hardware devices, APIs, and online services in new and interesting ways.

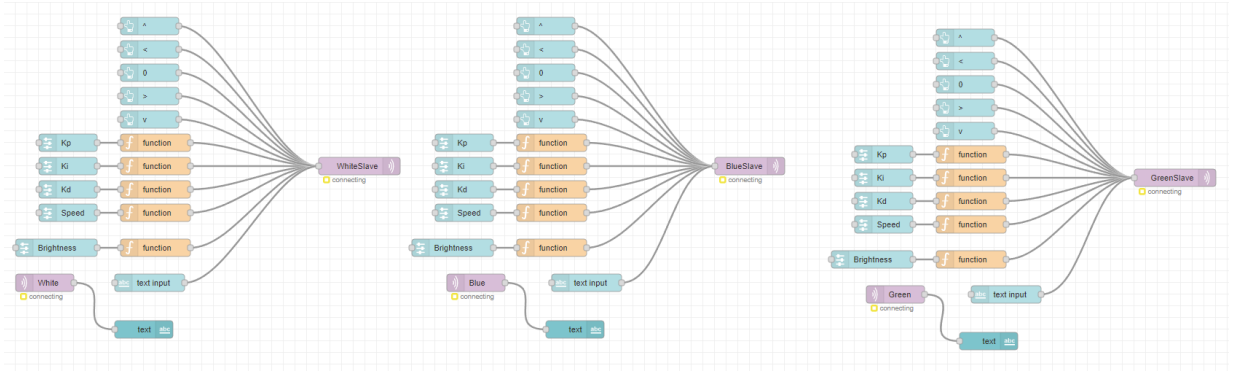
It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single click.



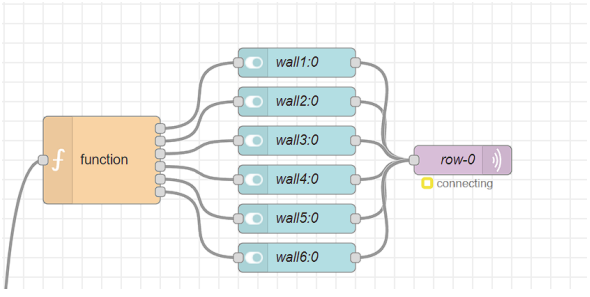
Node-Red interface

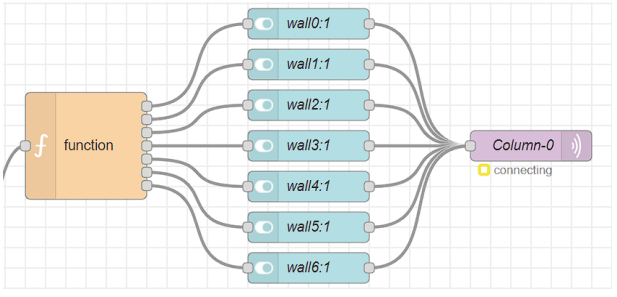


Controlling panel Nodes

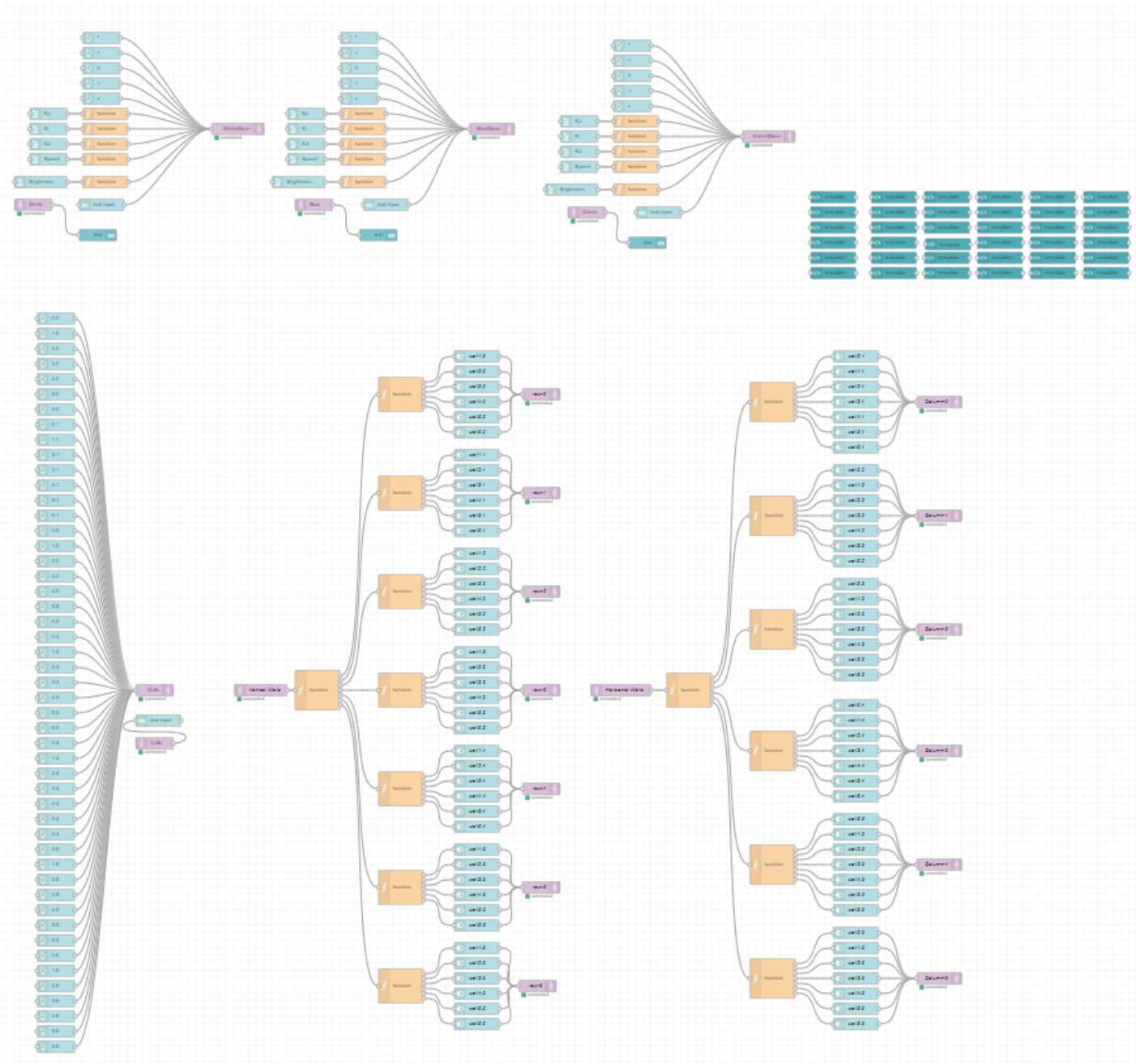


Wall/Object handling Nodes

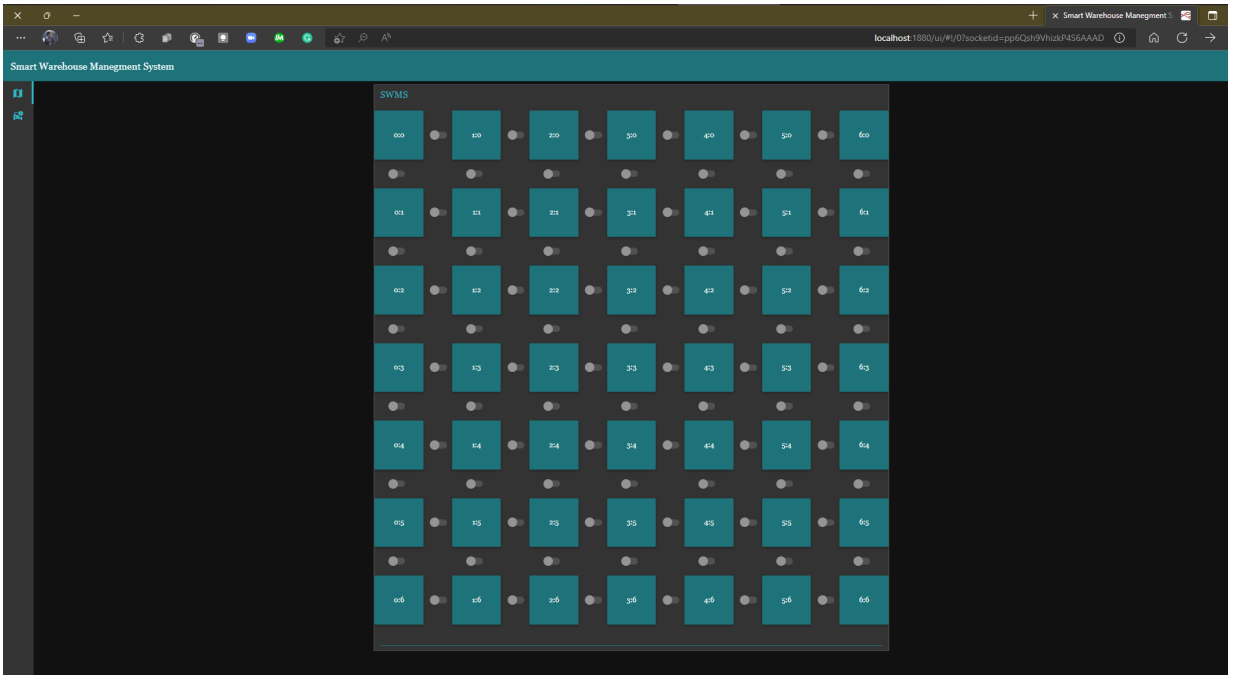




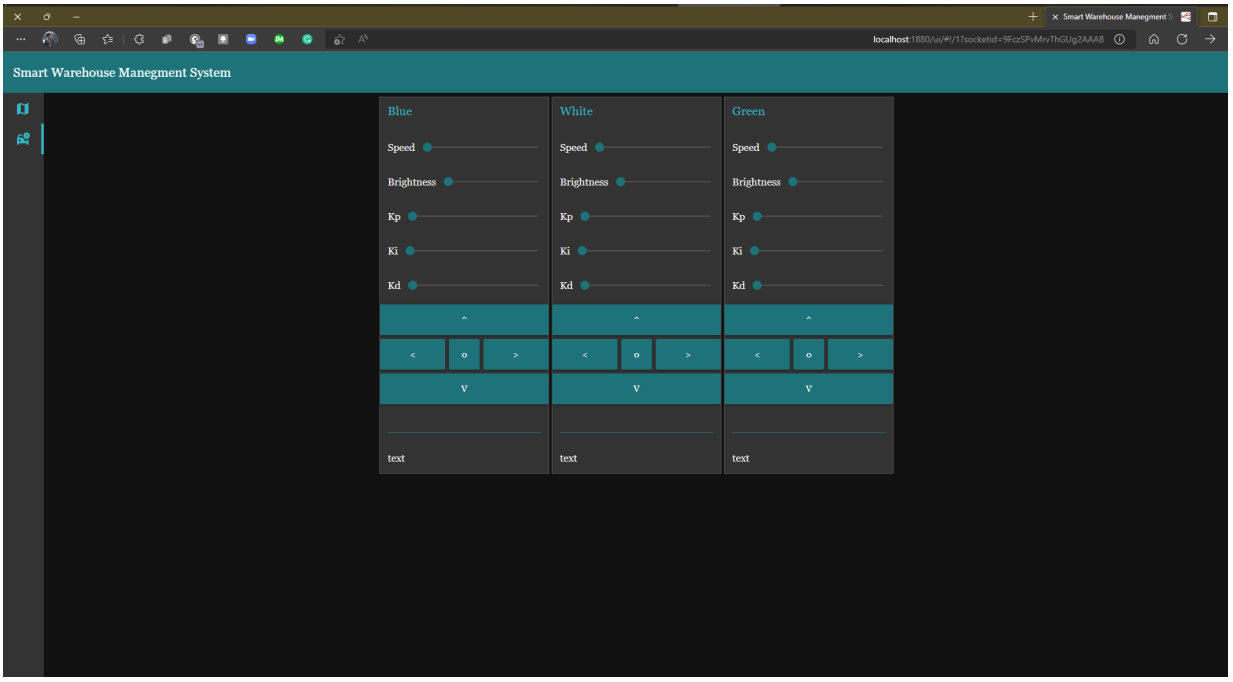
Overall design



The UI of the warehouse map:



The UI of robot controlling panel:

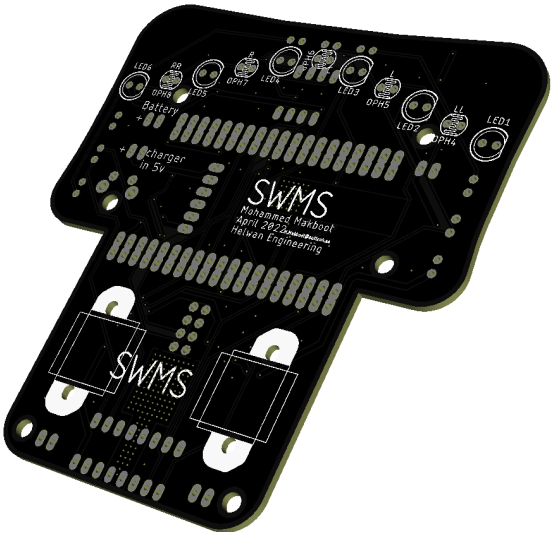


Chapter 4: System Integration

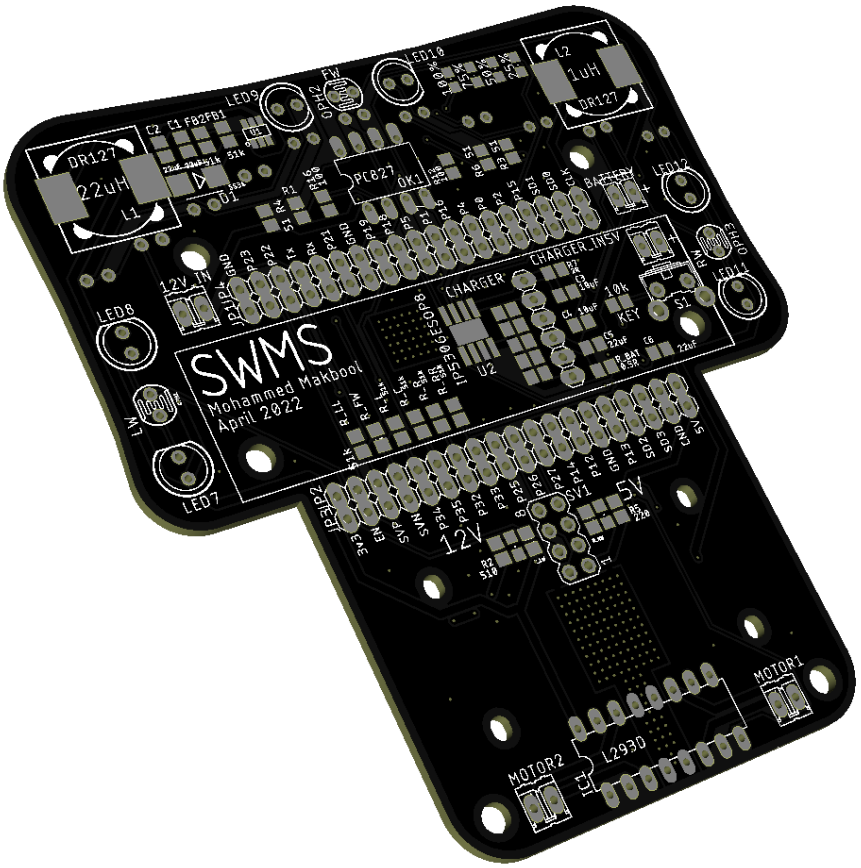
Slaves robots

Hardware

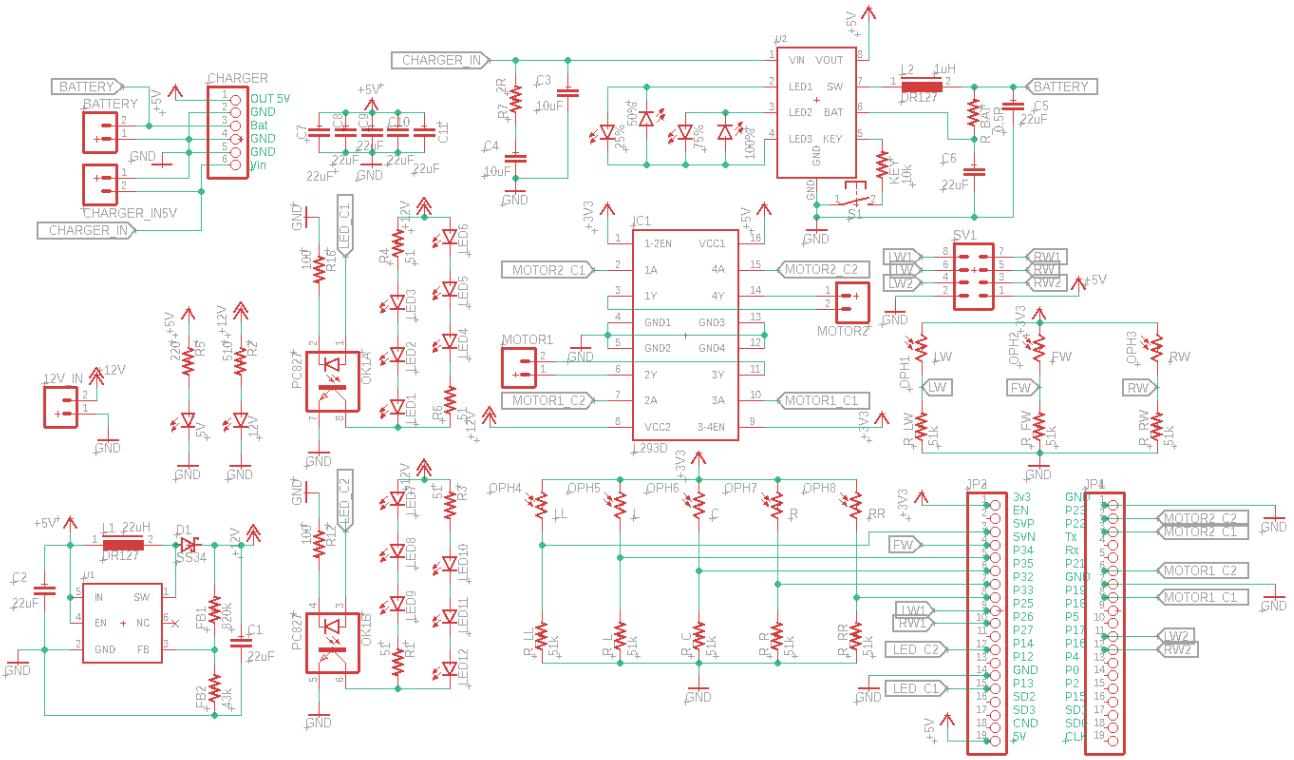
To put everything together we decided to design a printed circuit board (PCB) to integrate every part of the robot in one PCB to reduce the hardware and wiring problems.



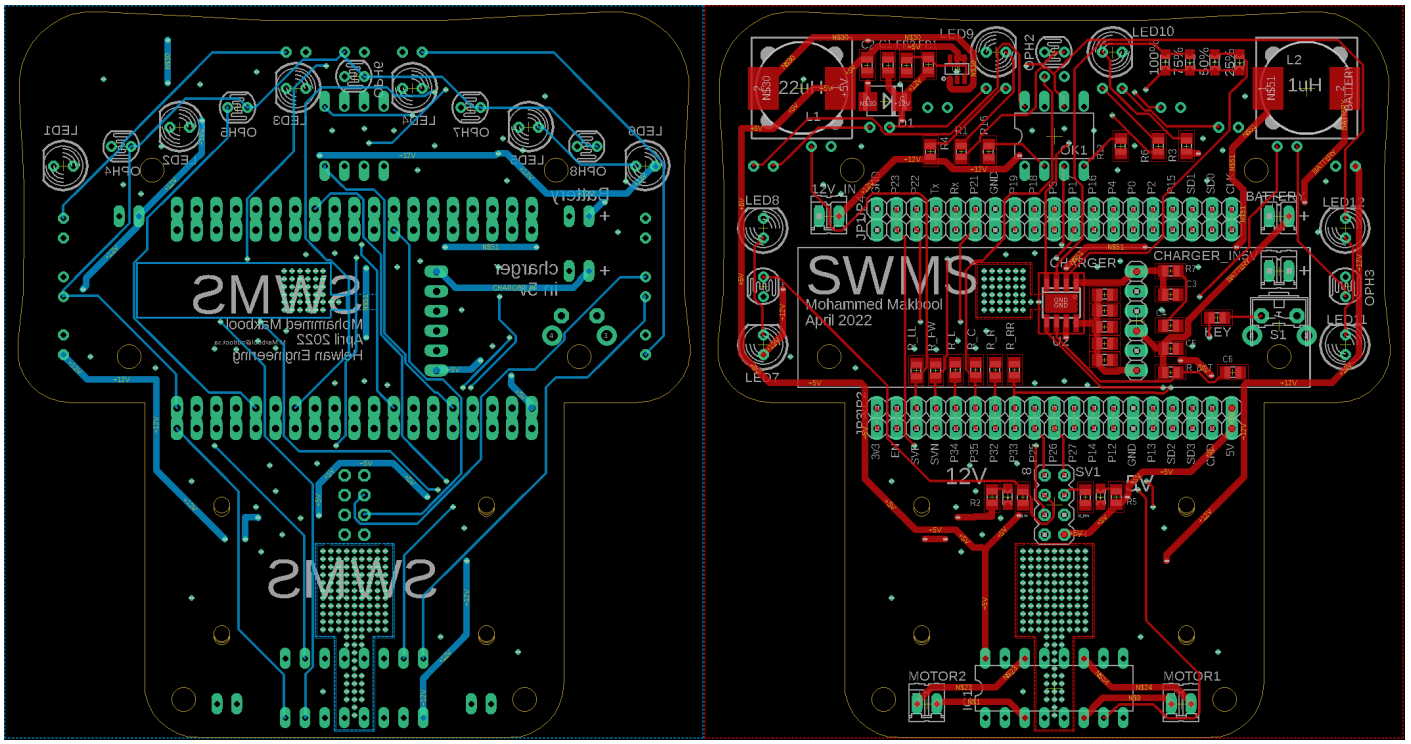
PCB Design



PCB Schematic

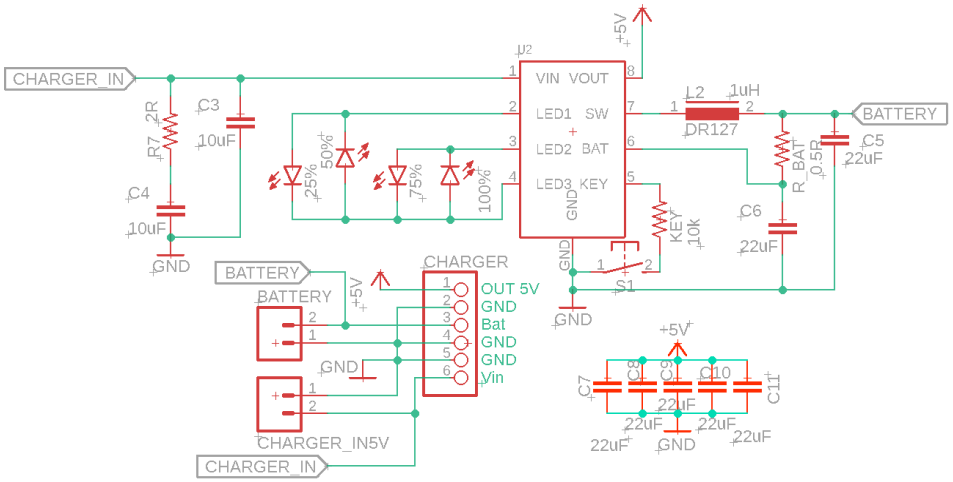


PCB Layout

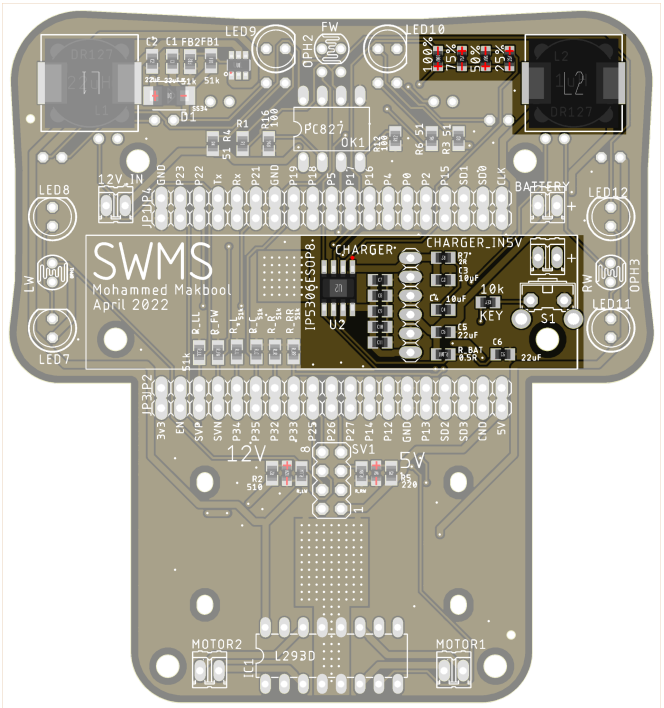


Battery Charger and Management System (BMS)

We need a system to take care of battery charging and discharging to protect the battery and avoid overcharge and over-discharge that can harm the battery so we implement IP5306 Fully-Integrated Power Bank System-On-Chip with 2.1A charger, 2.4A discharger.

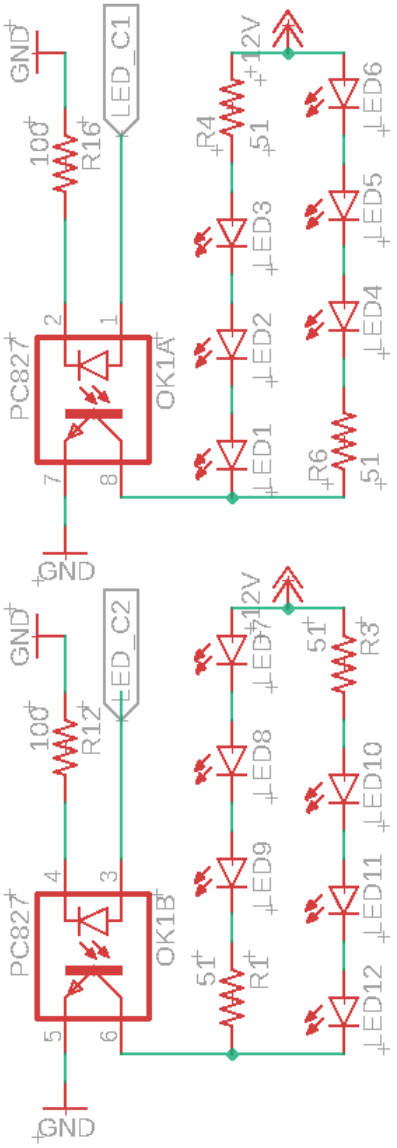


IP5306 is a fully-integrated multi-function power management SoC. It integrates a boost converter, a Li battery charger management system and a battery state of charge indicate controller. It provides a turn-key solution for power bank and portable charger applications. IP5306's high integration and rich features make the minimized component number in application. It can effectively downsize the application and lower the BOM cost. IP5306 only needs a single inductor to realize step-down and step-up which provides 2.4A output current. It can switch to standby mode at light load automatically IP5306's synchronous switching charger provides 2A charging current. Its efficiency is up to 91%. It regulates the charging current by IC temperature and input voltage. IP5306 integrates voltage based fuel gauge indication of 1/2/3/4 LEDs.



LED Controller

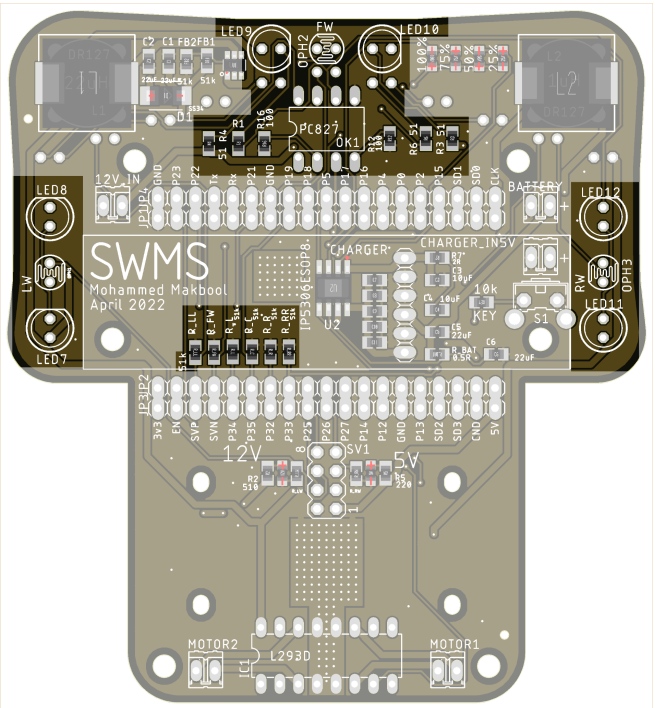
Because of large amount of LEDs the current of them will make an overload that microcontroller can't handle so we need some kind of hardware to solve this problem and a suitable one is optocouplers and we choose Sharp PC827 dual channel optocoupler to control all LEDs of the robot via 2 channels. One for the line follower and another one for wall detection.



■ Absolute Maximum Ratings (T_a=25°C)

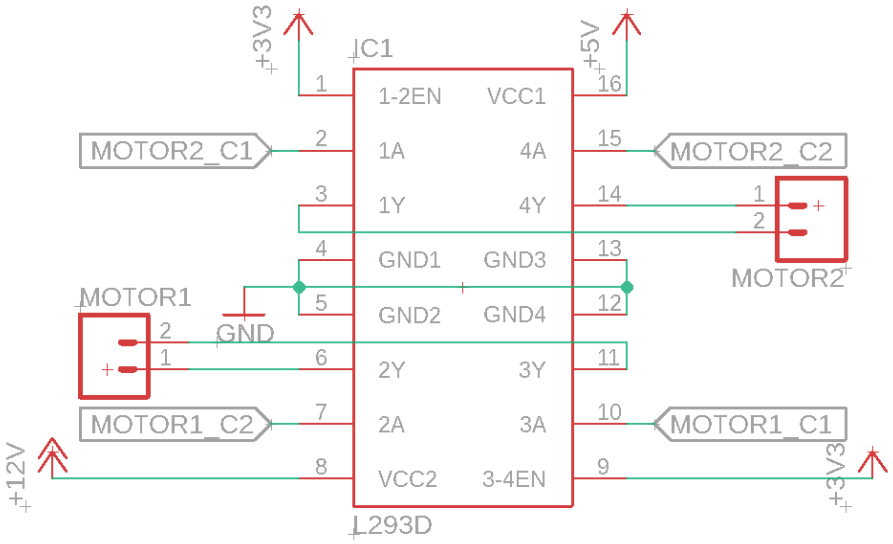
	Parameter	Symbol	Rating	Unit
Input	Forward current	I_F	50	mA
	*1 Peak forward current	I_{FM}	1	A
	Reverse voltage	V_R	6	V
	Power dissipation	P	70	mW
Output	Collector-emitter voltage	V_{CEO}	35	V
	Emitter-collector voltage	V_{ECO}	6	V
	Collector current	I_C	50	mA
	Collector power dissipation	P_C	150	mW
Total power dissipation		P_{tot}	200	mW
*2 Isolation voltage		$V_{iso\ (rms)}$	5	kV
Operating temperature		T_{opr}	−30 to +100	°C
Storage temperature		T_{stg}	−55 to +125	°C
*3 Soldering temperature		T_{sol}	260	°C

*1 Pulse width≤100μs, Duty ratio:0.001
*2 40 to 60%RH, AC for 1 minute
*3 For 10s

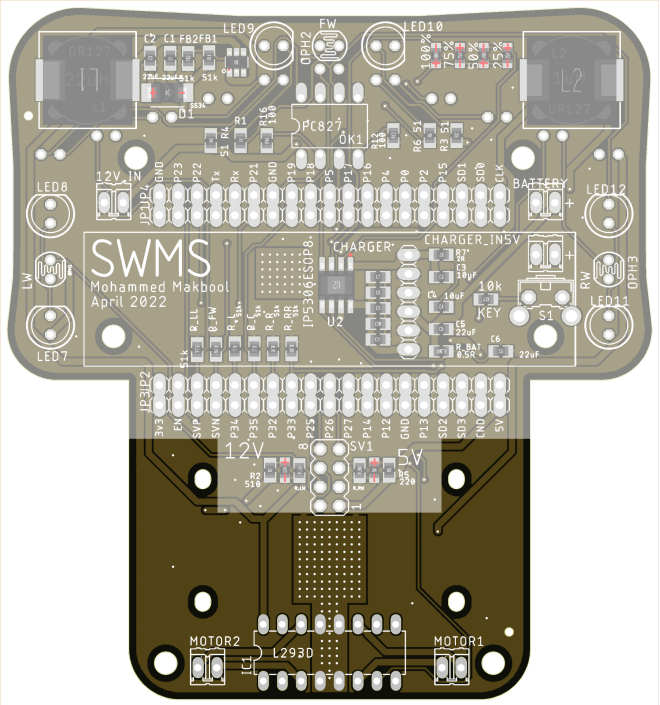


Motor Driver

If the microcontroller can't handle the small current of LEDs, it can't handle the huge current of the motors, so we need a motor driver to take this job. And Texas Instruments L293D motor driver IC is the suitable one.



The L293D device are quadruple high current half-H drivers. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. The device is designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.



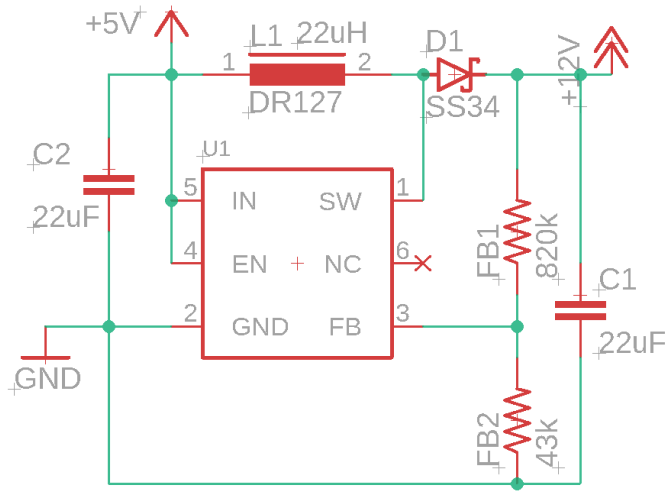
Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. The L293D is characterized for operation from 0°C to 70°C.

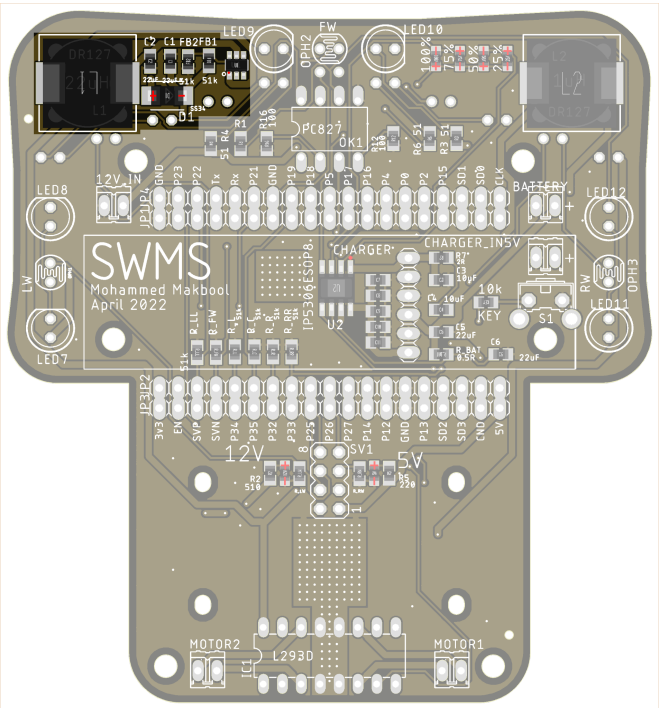
Boost Converter

As we know we can't run a motor with 3 volts directly from the battery so we need to boost up the voltage to 9V-12V range so that our motor can work fine. One of the most popular boost converter ICs is MT3608.

The MT3608 is a constant frequency, 6-pin SOT23 current mode step-up converter intended for small, low power applications.

The MT3608 switches at 1.2MHz and allows the use of tiny, low cost capacitors and inductors 2mm or less in height. Internal soft-start results in small inrush current and extends battery life. The MT3608 features automatic shifting to pulse frequency modulation mode at light loads. The MT3608 includes under-voltage lockout, current limiting, and thermal overload protection to prevent damage in the event of an output overload. The MT3608 is available in a small 6-pin SOT-23 package.

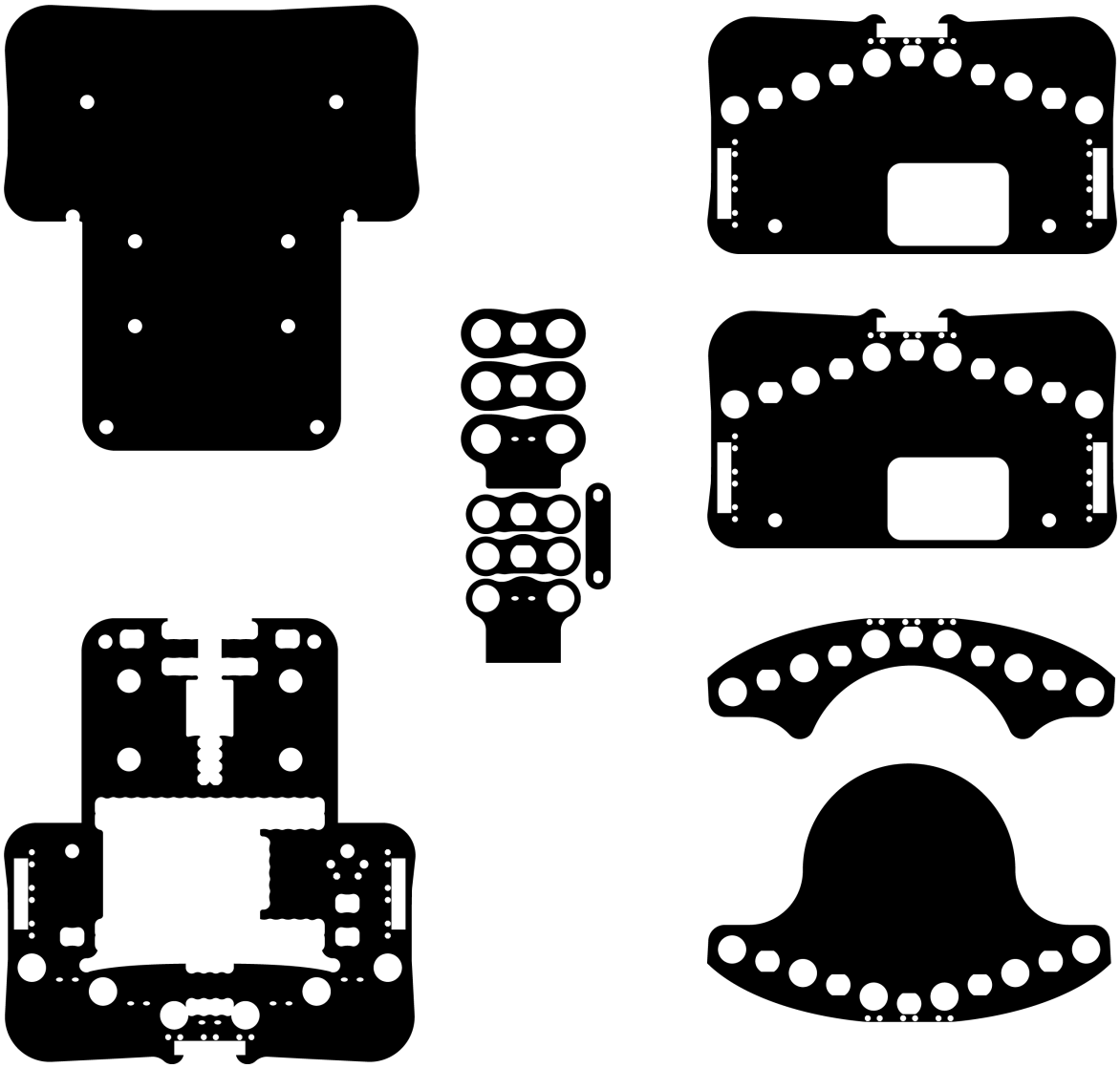




- Integrated 80mΩ Power MOSFET
- 2V to 24V Input Voltage
- 1.2MHz Fixed Switching Frequency
- Internal 4A Switch Current Limit
- Adjustable Output Voltage
- Internal Compensation
- Up to 28V Output Voltage
- up to 97% Efficiency

Laser Cutting

The sensors need to be isolated from the environment external lighting and the light of internal LEDs need to be guided. So we designed some parts as housing for the robot and sensors and LEDs. And cut it with a Co2 laser cutter on MDF wood.



Software

Main code file

```
/*
* June 2022
* SWMS Master Code as Client.
*
* The slave robots code that responsable of everything on the
robot,
*from sensors and motors to wifi, Bluetooth and connections
*/

// define to activate Blutooth Serial,
// if commented the USB Serial will be activated
#define BT

#define thisRobot WhiteRobot
#define GreenRobot 0
#define BlueRobot 1
#define WhiteRobot 2

// include wifi lib
#include <WiFi.h>
// unclude MQTT lib
#include <PubSubClient.h>
// include robot class
#include "Robot.h"

// wifi and MQTT config
const char *ssid = "SWMS";
const char *password = "12345678";
const char *mqtt_server = "192.168.4.1";
const char *publish = "White";
const char *subscribe = "WhiteSlave";
const char *Location = "White/Location";
```

```
// choose between Bluetooth and USB
#ifdef BT
#define com SerialBT
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;
#else
#define com Serial
#endif

WiFiClient espClient;
PubSubClient client(espClient);

// creat robot object
Robot robot;

// creat buffer to store trsansmited messges
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

// debuging header that contain all debuging fun.
#include "Debug.h"
// header file that contain all MQTT and Wifi handeling fun.
#include "mqtt.h"
```

```
void setup()
{
// start Serial for debuging
#ifdef BT
  SerialBT.begin(publish);
#else
  Serial.begin(115200);
#endif

  // attach robot component "Sensors and motors"
  robot.attach();
  // set wifi config and start wifi connection
  setup_wifi();
  // set MQTT config
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  // connect to the broker
  if (!client.connected())
    reconnect();
  // start calibration of sensors for 5 sec
  robot.calibrate(5000, false);
  // set the robot start location and direction
  Cells::targetColumn = robot.mouseColumn = 6;
  Cells::targetRow = robot.mouseRow = 6;
  robot.mouseHeading = 0;

  // publish robot location
 snprintf(msg, MSG_BUFFER_SIZE, "%d%d%d", robot.mouseColumn,
robot.mouseRow, robot.mouseHeading);
  client.publish(Location, msg);

  // set map borders
  Cells::Borders();
  // solve the map to find pathes
  Cells::solve();
  // set LED toggling off
  Robot::toggle = false;
}
```

```
void loop()
{
  // if connection lost try to reconnect
  if (!client.loop())
    reconnect();
  // check if there any debuging meesges
  serial_debug();

  // don't move untill you recive destination from master and permation to
go
  if (Cells::values[robot.mouseRow][robot.mouseColumn] != 0 && you_can_move)
  {
    // check the direction of the next cell and turn to it
    robot.Turn();
    // if you see wall, you need to check many times to be sure,
    // if it's for sure add wall on your location and publish it to the
network
    if (robot.Wall())
      if (robot.Wall_check())
        addWall(robot.mouseHeading);
      else
        robot.Move_Cell();
    else
      // if there is no wall move one cell forword
      robot.Move_Cell();

    // publish your new location and diraction
    snprintf(msg, MSG_BUFFER_SIZE, "%d%d%d", robot.mouseColumn,
robot.mouseRow, robot.mouseHeading);
    client.publish(Location, msg);
  }
  else
    you_can_move = false;

  // show robot location if needed
  if (flag)
    com.printf("\nRobot Column is  %d, Robot Row is  %d, Robot Head is %d ",
robot.mouseColumn, robot.mouseRow, robot.mouseHeading);
  flag = false;
}
```

Robot.h

```
#ifndef Robot_h
#define Robot_h

#include "Arduino.h"
#include "Sensors.h"
#include "Motors.h"
#include "Cell.h"

class Robot : public Sensors, public Motors, public Cells
{
public:
    inline static bool toggle;
    int Position;
    bool first_turn = true;
    void Move_Cell(byte N);
    void Move_Cell();
    void Turn();
    void right_line();
    void left_line();
    void calibrate(unsigned long calTime, bool mode);
    void attach();
};

#endif
```

Robot.cpp

```
#include "Robot.h"

void Robot::Move_Cell(byte N)
{
    for (int8_t i = 0; i < N; i++)
        Move_Cell();
}

void Robot::Move_Cell()
{
    do
    {
        forword(Speed);
    } while (get_robot_position(toggle) == 5);
    delay(3 * (300 - Speed));

    do
    {
        Position = get_robot_position(toggle);
        switch (Position)
        {
        case 6:
            break;
        case 5:
            if (get_robot_position(toggle) == 5)
                stop();
            break;
        default:
            follow_line(Speed, Position);
        }
    } while (get_robot_position(toggle) != 5); //&&
!Wall_limit());

    mouseRow += neiCells[mouseHeading][0];
    mouseColumn += neiCells[mouseHeading][1];

    stop();
    first_turn = true;
}
```

```
void Robot::Turn()
{
    int best = findBastawese();
    while (mouseHeading != best)
        switch (mouseHeading - best)
        {
        case -2:
        case -1:
        case 3:
            right_line();
            break;
        case 1:
        case 2:
        case -3:
            left_line();
            break;
        }
}
void Robot::right_line()
{
    do
    {
        forword(Speed);
    } while (get_robot_position(toggle) == 5);

    if (first_turn)
        delay(3 * (300 - Speed));

    do
    {
        right(Speed);
    } while (get_robot_position(toggle) != 6);
    delay(3 * (300 - Speed));

    while (get_robot_position(toggle) == 6)
        ;
    delay(3 * (300 - Speed) + 30);
    stop();

    mouseHeading++;
    mouseHeading %= 4;
    first_turn = false;
}
```

```
void Robot::left_line()
{
    do
    {
        forword(Speed);
    } while (get_robot_position(toggle) == 5);

    if (first_turn)
        delay(3 * (300 - Speed));

    do
    {
        left(Speed);
    } while (get_robot_position(toggle) != 6);

    delay(3 * (300 - Speed));
    while (get_robot_position(toggle) == 6)
        ;
    delay(3 * (300 - Speed) + 30);
    stop();

    mouseHeading += 3;
    mouseHeading %= 4;
    first_turn = false;
}

void Robot::attach()
{
    attachSensors();
    attachMotor();
}
```

```
void Robot::calibrate(unsigned long calTime, bool mode)
{
    int16_t Cal[SAZ];
    Wall_Threshold = 0;
    for (int i = 0; i < SAZ; i++)
    {
        Maximum[i] = 0;
        Minimum[i] = 10000;
        Threshold[i] = 0;
    }
    for (int i = 0; i < numReading; i++)
    {
        get_sensor_smoothed_value(Cal, mode);
        get_wall_smoothed_value();
    }
    unsigned long now = millis();
    while (millis() - now < calTime)
    {
        if (millis() - now < calTime / 4)
            right(Speed);
        else if (millis() - now < 3 * calTime / 4)
            left(Speed);
        else
            right(Speed);

        if (get_wall_smoothed_value() > Wall_Threshold)
            Wall_Threshold = get_wall_smoothed_value();
        get_sensor_smoothed_value(Cal, mode);
        for (int i = 0; i < SAZ; i++)
        {
            if (Maximum[i] < Cal[i])
                Maximum[i] = Cal[i];
            if (Minimum[i] > Cal[i])
                Minimum[i] = Cal[i];
        }
    }
    stop();
    for (int i = 0; i < SAZ; i++)
        Threshold[i] = (Minimum[i] + Maximum[i]) / 2;
    Wall_Threshold /= 10;
    Wall_Threshold *= 8;
}
```

Sensores.h

```
#ifndef Sensors_h
#define Sensors_h

#include "Arduino.h"

#define numReading 7

#define SAZ 5 // Sensor array size

class Sensors
{
public:
    int16_t Sensed[SAZ];
    int Max_Bright = 10;

    // variables for smoothing
    uint8_t iindex = 0;
    int Total[SAZ] = {0};
    int16_t Readings[SAZ][numReading];
    int16_t Maximum[SAZ], Minimum[SAZ], Threshold[SAZ];

    uint8_t windex = 0;
    int Total_Wall = 0;
    int16_t Wall_Readings[numReading];
    int16_t Wall_Threshold;

    double correlation(int16_t *X, int16_t *Y);
    void get_sensor_raw_value(int16_t *sensor_value, bool
Toggle);
    void get_sensor_smoothed_value(int16_t *sensor_value, bool
mode);
    uint16_t get_wall_raw_value(bool Toggle);
    int get_wall_smoothed_value();
    bool Wall();
    bool Wall_check();
    bool Wall_limit();
    int8_t get_robot_position(bool mode);
    void attachSensors();
};
#endif
```

Sensors.cpp

```
#include "Sensors.h"

#define LED_C1 13
#define LED_C2 14

#define LL 36
#define L 34
#define C 35
#define R 32
#define RR 33

#define FW 39

double Sensors::correlation(int16_t *X, int16_t *Y)
{
  double MeanX, MeanY, sX, sY, sXY;
  MeanX = MeanY = sX = sY = sXY = 0;

  for (int i = 0; i < SAZ; i++)
  {
    MeanX += X[i];
    MeanY += Y[i];
  }
  MeanX /= SAZ;
  MeanY /= SAZ;

  for (int i = 0; i < SAZ; i++)
  {
    sX += (X[i] - MeanX) * (X[i] - MeanX);
    sY += (Y[i] - MeanY) * (Y[i] - MeanY);
    sXY += (X[i] - MeanX) * (Y[i] - MeanY);
  }
  return (sXY / sqrt(sX * sY));
}
```

```
void Sensors::get_sensor_raw_value(int16_t *sensor_value, bool
Toggle)
{

  // int B = map(analogRead(FW), 0, 1023, Max_Bright, 1);
  if (Toggle)
  {
    int16_t LED_off_value[SAZ];
    analogWrite(LED_C1, 0);
    delay(13);

    LED_off_value[0] = analogRead(LL);
    LED_off_value[1] = analogRead(L);
    LED_off_value[2] = analogRead(C);
    LED_off_value[3] = analogRead(R);
    LED_off_value[4] = analogRead(RR);

    analogWrite(LED_C1, Max_Bright);
    delay(13);

    sensor_value[0] = analogRead(LL) - LED_off_value[0];
    sensor_value[1] = analogRead(L) - LED_off_value[1];
    sensor_value[2] = analogRead(C) - LED_off_value[2];
    sensor_value[3] = analogRead(R) - LED_off_value[3];
    sensor_value[4] = analogRead(RR) - LED_off_value[4];

    analogWrite(LED_C1, 0);
  }
  else
  {
    analogWrite(LED_C1, Max_Bright);

    sensor_value[0] = analogRead(LL);
    sensor_value[1] = analogRead(L);
    sensor_value[2] = analogRead(C);
    sensor_value[3] = analogRead(R);
    sensor_value[4] = analogRead(RR);
  }
}
```

```
void Sensors::get_sensor_smoothed_value(int16_t *sensor_value,
bool mode)
{
  get_sensor_raw_value(sensor_value, mode);
  for (int i = 0; i < SAZ; i++)
  {
    Total[i] -= Readings[i][iindex];
    Readings[i][iindex] = sensor_value[i];
    Total[i] += Readings[i][iindex];
    sensor_value[i] = Total[i] / numReading;
  }
  iindex++;
  if (iindex >= numReading)
    iindex = 0;
}

uint16_t Sensors::get_wall_raw_value(bool Toggle)
{
  if (Toggle)
  {
    int16_t LED_off_value;
    analogWrite(LED_C2, 0);
    delay(7);
    LED_off_value = analogRead(FW);
    analogWrite(LED_C2, 10);
    delay(13);
    return analogRead(FW) - LED_off_value;
    analogWrite(LED_C2, 0);
  }
  else
  {
    analogWrite(LED_C2, 1);
    return analogRead(FW);
  }
}
```

```
int Sensors::get_wall_smoothed_value()
{

  Total_Wall -= Wall_Readings[windex];
  Wall_Readings[windex] = get_wall_raw_value(true);
  Total_Wall += Wall_Readings[windex];

  iindex++;
  if (iindex >= numReading)
    iindex = 0;

  return Total_Wall / numReading;
}

bool Sensors::Wall()
{
  if (get_wall_raw_value(true) > Wall_Threshold)
    return true;
  else
    return false;
}

bool Sensors::Wall_check()
{
  for (int i = 0; i < numReading; i++)
    get_wall_smoothed_value();
  if (get_wall_smoothed_value() > Wall_Threshold)
    return true;
  else
    return false;
}

bool Sensors::Wall_limit()
{
  if (get_wall_raw_value(true) > (2 * Wall_Threshold))
    return true;
  else
    return false;
}
```

```
int8_t Sensors::get_robot_position(bool mode)
{
  int16_t Sens[SAZ], T[SAZ];
  int8_t Po = 0;
  double r[SAZ * 2 + 2];
  get_sensor_raw_value(Sens, mode);
  for (int i = 0; i < SAZ; i++)
    T[i] = Maximum[i];
  for (int i = 0; i < SAZ; i++)
  {
    T[i] = Minimum[i];
    r[2 * i] = correlation(Sens, T);
    T[i] = Maximum[i];
  }

  for (int i = 0; i < SAZ - 1; i++)
  {
    T[i] = Minimum[i];
    T[i + 1] = Minimum[i + 1];
    r[(2 * i) + 1] = correlation(Sens, T);
    T[i] = Maximum[i];
    T[i + 1] = Maximum[i + 1];
  }

  T[1] = Minimum[1];
  T[2] = Minimum[2];
  T[3] = Minimum[3];
  r[SAZ * 2 + 1] = correlation(Sens, T);
  r[SAZ * 2 - 1] = correlation(Sens, Minimum);
  r[SAZ * 2] = correlation(Sens, Maximum);

  for (int i = 1; i < (SAZ * 2 + 2); i++)
    if (r[i] > r[Po])
      Po = i;
  if (Po > 8)
    if (Sens[1] < Threshold[1] && Sens[2] < Threshold[2] &&
Sens[3] < Threshold[3] && (Sens[0] < Threshold[0] || Sens[4] <
Threshold[4]))
      Po = 9;
    else
      Po = 10;
  return Po - SAZ + 1;
}
```

```
void Sensors::attachSensors()
{
  analogReadResolution(10);
  pinMode(LED_C2, OUTPUT);
  pinMode(LED_C1, OUTPUT);
  analogWrite(LED_C2, 0);
  analogWrite(LED_C1, 0);
  pinMode(LL, INPUT);
  pinMode(L, INPUT);
  pinMode(C, INPUT);
  pinMode(R, INPUT);
  pinMode(RR, INPUT);
}
```

Motors.h

```
#ifndef Motors_h
#define Motors_h

#include "Arduino.h"
#include "PID.h"

class Motors : public PID
{
public:
    int Speed = 170;
    int soft_start;
    void forword(int S);
    void left(int S);
    void right(int S);
    void backword(int S);
    void stop();

    void follow_line(int S, int Position);
    void right_line();
    void left_line();

    void attachMotor();

    void Set_K(int kn, double value);
    double Get_K(int kn);
    void Update_K();
};
#endif
```

Motors.cpp

```
#include "Motors.h"

#define MOTOR1_C1 19
#define MOTOR1_C2 21
#define MOTOR2_C1 22
#define MOTOR2_C2 23

void Motors::forword(int S)
{
  analogWrite(MOTOR2_C1, S);
  analogWrite(MOTOR2_C2, 0);
  analogWrite(MOTOR1_C1, S);
  analogWrite(MOTOR1_C2, 0);
}
void Motors::left(int S)
{
  analogWrite(MOTOR2_C2, S);
  analogWrite(MOTOR2_C1, 0);
  analogWrite(MOTOR1_C1, S);
  analogWrite(MOTOR1_C2, 0);
}
void Motors::right(int S)
{
  analogWrite(MOTOR2_C1, S);
  analogWrite(MOTOR2_C2, 0);
  analogWrite(MOTOR1_C2, S);
  analogWrite(MOTOR1_C1, 0);
}
void Motors::backword(int S)
{
  analogWrite(MOTOR2_C2, S);
  analogWrite(MOTOR2_C1, 0);
  analogWrite(MOTOR1_C2, S);
  analogWrite(MOTOR1_C1, 0);
}
```

```
void Motors::stop()
{
  analogWrite(MOTOR2_C2, 0);
  analogWrite(MOTOR2_C1, 0);
  analogWrite(MOTOR1_C2, 0);
  analogWrite(MOTOR1_C1, 0);
}
void Motors::follow_line(int S, int Position)
{
  int E = compute(0, Position, S * (-1), S);
  analogWrite(MOTOR2_C1, constrain(S - E, 0, S));
  analogWrite(MOTOR2_C2, 0);
  analogWrite(MOTOR1_C1, constrain(S + E, 0, S));
  analogWrite(MOTOR1_C2, 0);
}

void Motors::attachMotor()
{
  pinMode(MOTOR2_C1, OUTPUT);
  pinMode(MOTOR2_C2, OUTPUT);
  pinMode(MOTOR1_C1, OUTPUT);
  pinMode(MOTOR1_C2, OUTPUT);
  reset();
  Update_K();

  // K[0] = 40;
  // K[1] = 0.35;
  // K[2] = 0.3;
}

void Motors::Set_K(int kn, double value)
{
  K[kn] = value;
}
```

```
double Motors::Get_K(int kn)
{
  return K[kn];
}

void Motors::Update_K()
{
  K[0] = (double)(Speed / 4);
  K[1] = K[2] = K[0] / (double)300;
}
```

PID.h

```
#ifndef PID_h
#define PID_h

#include "Arduino.h"

class PID
{
  public:
    uint32_t lastTime;
    volatile double output;
    double lastErr;
    double errSum;
    double K[3];

    PID();
    double compute(long setPoint, long sensor, double minOut, double
maxOut);
    void reset();
};

#endif
```

PID.cpp

```
#include "PID.h"
PID::PID()
{
  K[0] = 0.07;
  K[1] = 0.03;
  K[2] = 0.03;
}

double PID::compute(long setPoint, long sensor, double minOut, double
maxOut)
{
  // Calculate time difference since last time executed
  uint32_t now = millis();
  double timeChange = (double)(now - lastTime);

  // Calculate error (P, I and D)
  double error = setPoint - sensor;
  errSum += error * timeChange;
  errSum = constrain(errSum, minOut, maxOut);
  double dErr = (error - lastErr) / timeChange;

  // Calculate the new output by adding all three elements together
  double newOutput = (K[0] * error + K[1] * errSum + K[2] * dErr);

  // If limit is specifyed, limit the output
  output = constrain(newOutput, minOut, maxOut);

  lastErr = error;
  lastTime = now;
  return (output);
}

void PID::reset()
{
  errSum = 0;
  lastErr = 0;
  lastTime = 0;
}
```

Cell.h

```
#ifndef Cells_h
#define Cells_h

#include "Arduino.h"

#define ROWS 7
#define COLUMNS 7

const int8_t neiCells[4][2] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
const int8_t neiWalls[4][2] = {{0, 0}, {0, 1}, {1, 0}, {0, 0}};

class Cells
{

private:
  // vertical walls array
  inline static boolean verticalWalls[ROWS][COLUMNS + 1];

  // horizontal walls array
  inline static boolean horizontalWalls[ROWS + 1][COLUMNS];

public:
  // value array
  inline static byte values[ROWS][COLUMNS];

  byte mouseRow;
  byte mouseColumn;
  byte mouseHeading;

  inline static byte targetRow;
  inline static byte targetColumn;
```

```
// Constructor method (called when the maze is created)
 static void Borders()
 {
   // initialize verticalWalls (add exterior walls)
   for (byte i = 0; i < ROWS; i++)
   {
     for (byte j = 0; j < COLUMNS + 1; j++)
     {
       if (j == 0 || j == COLUMNS)
       {
         verticalWalls[i][j] = true;
       }
     }
   }

   // initialize horizontalWalls (add exterior walls)
   for (byte i = 0; i < ROWS + 1; i++)
   {
     for (byte j = 0; j < COLUMNS; j++)
     {
       if (i == 0 || i == ROWS)
       {
         horizontalWalls[i][j] = true;
       }
     }
   }
 }
```

```
static void addWall_V(int wallColumn, int wallRow, bool state)
{
  verticalWalls[wallRow][wallColumn] = state;
  Cells::solve();
}
```

```
static void addWall_H(int wallColumn, int wallRow, bool state)
{
  horizontalWalls[wallRow][wallColumn] = state;
  Cells::solve();
}
```

```
static void solve()
{

  for (int8_t i = 0; i < ROWS; i++)
  {
    for (int8_t j = 0; j < COLUMNS; j++)
    {
      values[i][j] = 255;
    }
  }
  values[targetRow][targetColumn] = 0;

  bool cont = true;
```

```
while (cont)
    {
      cont = false;
      for (int8_t i = 0; i < ROWS; i++)
      {
        for (int8_t j = 0; j < COLUMNS; j++)
        {
          if (values[i][j] < 255)
            for (int8_t k = 0; k < 4; k++)
            {

              int8_t neiCellRow = i + neiCells[k][0];
              int8_t neiCellColomn = j + neiCells[k][1];

              int8_t neiWallRow = i + neiWalls[k][0];
              int8_t neiWallColomn = j + neiWalls[k][1];

              bool wall = false;

              if (k == 0 || k == 2)
                wall = horizontalWalls[neiWallRow][neiWallColomn];
              else
                wall = verticalWalls[neiWallRow][neiWallColomn];
              if (values[neiCellRow][neiCellColomn] == 255 && !wall)
              {
                values[neiCellRow][neiCellColomn] = values[i][j] + 1;
                cont = true;
              }
            }
        }
      }
    }
  }

  int8_t findBastawese();
};
#endif
```

Cell.cpp

```
#include "Cell.h"

#define NORTH 0
#define EAST 1
#define SOUTH 2
#define WEST 3

// function that check all four neighbor cells to find best way
the //robot should go and return best cell direction
int8_t Cells::findBastawese()
{
  byte bestValue = 255;
  int8_t bestDir = 0;

  for (int8_t k = 0; k < 4; k++)
  {
    int8_t neiCellRow = mouseRow + neiCells[k][0];
    int8_t neiCellColomn = mouseColumn + neiCells[k][1];

    int8_t neiWallRow = mouseRow + neiWalls[k][0];
    int8_t neiWallColomn = mouseColumn + neiWalls[k][1];

    bool wall = false;
    if (k == 0 || k == 2)
      wall = horizontalWalls[neiWallRow][neiWallColomn];
    else
      wall = verticalWalls[neiWallRow][neiWallColomn];
    if (values[neiCellRow][neiCellColomn] < bestValue && !wall)
    {
      bestValue = values[neiCellRow][neiCellColomn];
      bestDir = k;
    }
    else if (values[neiCellRow][neiCellColomn] == bestValue &&
!wall && mouseHeading == k)
      bestDir = k;
  }
  return bestDir;
}
```

Depug.h

```
#include "Arduino.h"

bool flag = 0, you_can_move = 0;

void serial_debug()
{
  if (com.available())
  {
    char Received = com.read();
    int motor = 0;
    int Kn;

    switch (Received)
    {
    case 'c':
      com.println("\nPlease Enter Calibration Time.");
      while (!com.available())
        ;
      com.println("\nCalibration on progres .....");
      robot.calibrate(com.parseInt(), Robot::toggle);
      break;

    case 'P':
      com.printf("\nRobot Position is  %d ",
robot.get_robot_position(Robot::toggle));
      break;

    case 'h':
      robot.stop();
      break;
    case 'f':
      robot.forword(robot.Speed);
      break;
    case 'b':
      robot.backword(robot.Speed);
      break;
```

```
case 'l':
  robot.left(robot.Speed);
  break;
case 'r':
  robot.right(robot.Speed);
  break;
case 'B':
  robot.Max_Bright = com.parseInt();
  break;
case 'S':
  robot.Speed = com.parseInt();
  robot.Update_K();
  break;

case 'R':
  robot.get_sensor_raw_value(robot.Sensed, Robot::toggle);
  for (int i = 0; i < SAZ; i++)
    com.printf("\nRAW Sensor %d = %d\t", i, robot.Sensed[i]);
  break;

case 's':
  robot.get_sensor_smoothed_value(robot.Sensed, Robot::toggle);
  for (int i = 0; i < SAZ; i++)
    com.printf("\nSmoothed Sensor %d = %d\t", i, robot.Sensed[i]);
  break;

case 'K':
  Kn = com.parseInt();
  com.printf("\nPlease Enter K %d .", Kn);
  com.read();
  while (!com.available())
    ;
  robot.Set_K(Kn, com.parseFloat());
  com.printf("\nK %d Offset = %f", Kn, robot.Get_K(Kn));
  break;

case 'o':
  flag = !flag;
  break;
```

```
    case 'T':
      Robot::toggle = !Robot::toggle;
      com.printf("\nRobot::toggle is %s", Robot::toggle ? "true" :
"false");
      break;

    default:
      com.printf("\n%c\n", Received);
    }
  }
}
```

mqtt.h

```
#include "Arduino.h"

int parseInt(byte *c)
{
  boolean isNegative = false;
  int value = 0;
  if (c[0] < 0)
    return 0; // zero returned if timeout
  do
  {
    if (c[0] == '-')
      isNegative = true;
    else if (c[0] >= '0' && c[0] <= '9')
      // is c a digit?
      value = value * 10 + c[0] - '0';
    *c++;
  } while ((c[0] >= '0' && c[0] <= '9'));

  if (isNegative)
    value = -value;
  return value;
}
```

```
float parseFloat(byte *c)
{
  boolean isNegative = false;
  boolean isFraction = false;
  long value = 0;
  float fraction = 1.0;
  // ignore non numeric leading characters
  if (c[0] < 0)
    return 0; // zero returned if timeout

  do
  {
    if (c[0] == '-')
      isNegative = true;
    else if (c[0] == '.')
      isFraction = true;
    else if (c[0] >= '0' && c[0] <= '9')
    { // is c a digit?
      value = value * 10 + c[0] - '0';
      if (isFraction)
        fraction *= 0.1f;
    }
    *c++;
  } while ((c[0] >= '0' && c[0] <= '9') || c[0] == '.');

  if (isNegative)
    value = -value;
  if (isFraction)
    return value * fraction;
  else
    return value;
}
```

```
void callback(char *topic, byte *payload, unsigned int length)
{
  snprintf(msg, MSG_BUFFER_SIZE, "%d/Go", thisRobot);
  String go = msg;
  snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/H", (thisRobot + 1) % 3);
  String WallsH = msg;
  snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/H", (thisRobot + 2) % 3);
  String WallsH1 = msg;
  snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/V", (thisRobot + 1) % 3);
  String WallsV = msg;
  snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/V", (thisRobot + 2) % 3);
  String WallsV1 = msg;

  com.print("Message arrived [");
  com.print(topic);
  com.print("] ");
  String t = topic;

  for (int i = 0; i < length; i++)
  {
    com.print((char)payload[i]);
  }
  com.println();

  int motor = 0;
  int Kn;

  if (t == subscribe)
    switch (payload[0])
    {
    case 'c':
      Kn = parseInt(payload + 1);
      snprintf(msg, MSG_BUFFER_SIZE, "\nCalibration on progres
.....");
      client.publish(publish, msg);
      robot.calibrate(Kn, Robot::toggle);
      break;
```

```
    case 'P':
      snprintf(msg, MSG_BUFFER_SIZE, "\nRobot Position is  %d ",
robot.get_robot_position(Robot::toggle));
      client.publish(publish, msg);
      break;
    case 'h':
      robot.stop();
      break;
    case 'f':
      Kn = parseInt(payload + 1);
      if (Kn == 0)
        Kn++;
      robot.Move_Cell(Kn);
      break;
    case 'b':
      Kn = parseInt(payload + 1);
      if (Kn == 0)
        Kn++;
      robot.right_line();
      robot.right_line();
      robot.Move_Cell(Kn);
      break;
    case 'l':
      robot.left_line();
      break;
    case 'r':
      robot.right_line();
      break;
    case 'B':
      robot.Max_Bright = parseInt(payload + 1);
      break;
    case 'S':
      robot.Speed = parseInt(payload + 1);
      robot.Update_K();
      break;
```

```
    case 'R':
      robot.get_sensor_raw_value(robot.Sensed, Robot::toggle);
      for (int i = 0; i < SAZ; i++)
      {
        snprintf(msg, MSG_BUFFER_SIZE, "\nRAW Sensor %d = %d\t", i,
robot.Sensed[i]);
        client.publish(publish, msg);
      }
      break;

    case 's':
      robot.get_sensor_smoothed_value(robot.Sensed, Robot::toggle);
      for (int i = 0; i < SAZ; i++)
      {
        snprintf(msg, MSG_BUFFER_SIZE, "\nSmoothed Sensor %d = %d\t",
i, robot.Sensed[i]);
        client.publish(publish, msg);
      }
      break;

    case 'K':
      Kn = parseInt(payload + 1);
      robot.Set_K(Kn, parseFloat(payload + 2));
      snprintf(msg, MSG_BUFFER_SIZE, "\nK %d Offset = %f", Kn,
robot.Get_K(Kn));
      client.publish(publish, msg);
      break;

    case 'o':
      flag = !flag;
      break;

    case 'T':
      Robot::toggle = !Robot::toggle;
      snprintf(msg, MSG_BUFFER_SIZE, "\nToggle is %s", Robot::toggle ?
"true" : "false");
      client.publish(publish, msg);
      break;
    }
```

```
  else if (t == "Walls/V" || t == WallsV || t == WallsV1)
  {
    Cells::addWall_V((int)(payload[0] - '0'), (int)(payload[2] - '0'),
(bool)(payload[4] - '0'));
    com.printf("\nVertival wall %d:%d  %s", (int)(payload[0] - '0'),
(int)(payload[2] - '0'), (bool)(payload[4] - '0') ? "true" : "false");
  }


  else if (t == "Walls/H" || t == WallsH || t == WallsH1)
  {
    Cells::addWall_H((int)(payload[0] - '0'), (int)(payload[2] - '0'),
(bool)(payload[4] - '0'));
    com.printf("\nHorizontal wall %d:%d  %s", (int)(payload[0] - '0'),
(int)(payload[2] - '0'), (bool)(payload[4] - '0') ? "true" : "false");
  }


  else if (t == go)
  {
    Cells::targetColumn = (byte)(payload[0] - '0');
    Cells::targetRow = (byte)(payload[2] - '0');
    Cells::solve();
    you_can_move = true;
    com.printf("\nTarget Cell %d:%d  ", Cells::targetColumn,
Cells::targetRow);
  }
}
```

```
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  com.println();
  com.print("Connecting to ");
  com.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    com.print(".");
  }

  randomSeed(micros());

  com.println("");
  com.println("WiFi connected");
  com.println("IP address: ");
  com.println(WiFi.localIP());
}
```

```
void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected())
  {
    com.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = publish;
    // Attempt to connect
    if (client.connect(clientId.c_str()))
    {
      com.println("connected");
      // Once connected, publish an announcement...
      client.publish(publish, "Hi, I'm in.");
      // ... and resubscribe
      client.subscribe(subscribe);
      client.subscribe("Walls/V");
      client.subscribe("Walls/H");
      snprintf(msg, MSG_BUFFER_SIZE, "%d/Go", thisRobot);
      client.subscribe(msg);
      snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/H", (thisRobot + 1) %
3);
      client.subscribe(msg);
      snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/H", (thisRobot + 2) %
3);
      client.subscribe(msg);
      snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/V", (thisRobot + 1) %
3);
      client.subscribe(msg);
      snprintf(msg, MSG_BUFFER_SIZE, "%d/Walls/V", (thisRobot + 2) %
3);
      client.subscribe(msg);
    }
    else
    {
      com.print("failed, rc=");
      com.print(client.state());
      com.println(" try again in 3 seconds");
      // Wait 3 seconds before retrying
      delay(3000);
    }
  }
```

```
}
```

```
void addWall(int8_t wallDirc)
{
  switch (wallDirc)
  {
  case 0:
    Cells::addWall_H(robot.mouseColumn, robot.mouseRow, true);
    snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1", robot.mouseColumn,
robot.mouseRow);
    client.publish("Walls/H", msg);
    break;
  case 1:
    Cells::addWall_V(robot.mouseColumn + 1, robot.mouseRow, true);
    snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1", robot.mouseColumn + 1,
robot.mouseRow);
    client.publish("Walls/V", msg);
    break;
  case 2:
    Cells::addWall_H(robot.mouseColumn, robot.mouseRow + 1, true);
    snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1", robot.mouseColumn,
robot.mouseRow + 1);
    client.publish("Walls/H", msg);
    break;
  case 3:
    Cells::addWall_V(robot.mouseColumn, robot.mouseRow, true);
    snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1", robot.mouseColumn,
robot.mouseRow);
    client.publish("Walls/V", msg);
    break;
  }
}
```

MasterMind

Hardware

For now the mastermind hardware is very simple as we just need a simple microcontroller to do two simple tasks:

1st is solving the map to know which Slave is the nearest to the target, and to find the shortest path from this slave location to the destination.

2nd is to communicate with the system component using wifi protocol

both missions can be handled easily by our beloved **ESP 8266**.

Software

```
/*
 *  july 2022.
 *  SWMS Master Code as Client.
 *
 * Connecting to WiFi then try to connect to the MQTT broker to
receive
 * destination Messages and redirect them to the nearest robot.
 */

// include WiFi lib
#include <ESP8266WiFi.h>
// include MQTT lib
#include <PubSubClient.h>
// include cells class
#include "Cell.h"

#define GreenRobot 0
#define BlueRobot 1
#define WhiteRobot 2

volatile int Robot = 3;
volatile int Minimum = 3;

// creat an object for each robot
Cells cells[3];
```

```
// to store the distance of each robot from the target
byte RobotsValues[4];
bool active[3] = {true, true, true};

// WiFi config
const char *ssid = "SWMS";
const char *password = "12345678";
const char *mqtt_server = "192.168.4.1";
bool WiFiAP = false; // Do yo want the ESP as AP?
WiFiClient espClient;
PubSubClient client(espClient);

// Buffers to handel transmitted messeges
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

#define TOP_BUFFER_SIZE (50)
char top[MSG_BUFFER_SIZE];
```

```
      // callback fun tro handel recived messges
      void onData(char *topich, byte *data, uint32_t length)
      {
        String topic = topich;
        digitalWrite(LED_BUILTIN, LOW);
        char payload[length + 1];

        os_memcpy(payload, data, length);
        payload[length] = '\0';

        Serial.println("\nreceived topic '" + (String)topic + "' with data
'" + (String)payload + "'");
        // printClients();


       if (topic == "Walls/V")
        {
          Cells::addWall_V((int)(payload[0] - '0'), (int)(payload[2] -
'0'), (bool)(payload[4] - '0'));

      Serial.printf("\nVertival wall %d:%d  %s", (int)(payload[0] - '0'),
(int)(payload[2] - '0'), (bool)(payload[4] - '0') ? "true" : "false");
        }

        else if (topic == "Walls/H")
        {
          Cells::addWall_H((int)(payload[0] - '0'), (int)(payload[2] -
'0'), (bool)(payload[4] - '0'));
          Serial.printf("\nHorizontal wall %d:%d  %s", (int)(payload[0] -
'0'), (int)(payload[2] - '0'), (bool)(payload[4] - '0') ? "true" :
"false");
        }
```

```
      else if (topic == "Cells")
        {
          Cells::targetColumn = (byte)(payload[0] - '0');
          Cells::targetRow = (byte)(payload[2] - '0');
          Cells::Borders();
          Cells::solve();

          for (int i = 0; i < 4; i++)
            RobotsValues[i] = 255;

          if (active[GreenRobot])
            RobotsValues[GreenRobot] =
Cells::values[cells[GreenRobot].mouseRow][cells[GreenRobot].mouseColumn];

          if (active[BlueRobot])
            RobotsValues[BlueRobot] =
Cells::values[cells[BlueRobot].mouseRow][cells[BlueRobot].mouseColumn];

          if (active[WhiteRobot])
            RobotsValues[WhiteRobot] =
Cells::values[cells[WhiteRobot].mouseRow][cells[WhiteRobot].mouseColumn];

          for (int i = 0; i < 3; i++)
            if (RobotsValues[i] < RobotsValues[Minimum])
              Minimum = i;

          snprintf(msg, MSG_BUFFER_SIZE, "%d:%d", Cells::targetColumn,
Cells::targetRow);
          snprintf(top, TOP_BUFFER_SIZE, "%d/Go", Minimum);
          client.publish(top, msg);

          Serial.printf("\nTarget Cell %d:%d . \n Robot:%d ",
Cells::targetColumn, Cells::targetRow, Minimum);
        }
```

```
        else if (topic == "Green/Location")
          Robot = GreenRobot;

        else if (topic == "Blue/Location")
          Robot = BlueRobot;

        else if (topic == "White/Location")
          Robot = WhiteRobot;

        if (Robot < 3)
        {
          if
(!Cells::horizontalWalls[cells[Robot].mouseRow][cells[Robot].mouseColumn])
          {
            snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:0",
cells[Robot].mouseColumn, cells[Robot].mouseRow);
            snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/H", Robot);
            client.publish(top, msg);
          }

          if
(!Cells::verticalWalls[cells[Robot].mouseRow][cells[Robot].mouseColumn +
1])
          {
            snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:0",
cells[Robot].mouseColumn + 1, cells[Robot].mouseRow);
            snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/V", Robot);
            client.publish(top, msg);
          }
          if (!Cells::horizontalWalls[cells[Robot].mouseRow +
1][cells[Robot].mouseColumn])
          {
            snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:0",
cells[Robot].mouseColumn, cells[Robot].mouseRow + 1);
            snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/H", Robot);
            client.publish(top, msg);
          }
```

```
          if
(!Cells::verticalWalls[cells[Robot].mouseRow][cells[Robot].mouseColumn])
          {
            snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:0",
cells[Robot].mouseColumn, cells[Robot].mouseRow);
            snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/V", Robot);
            client.publish(top, msg);
          }

          cells[Robot].mouseColumn = (byte)(payload[0] - '0');
          cells[Robot].mouseRow = (byte)(payload[1] - '0');
          cells[Robot].mouseHeading = (byte)(payload[2] - '0');

          snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1",
cells[Robot].mouseColumn, cells[Robot].mouseRow);
          snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/H", Robot);
          client.publish(top, msg);

          snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1",
cells[Robot].mouseColumn + 1, cells[Robot].mouseRow);
          snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/V", Robot);
          client.publish(top, msg);

          snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1",
cells[Robot].mouseColumn, cells[Robot].mouseRow + 1);
          snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/H", Robot);
          client.publish(top, msg);

          snprintf(msg, MSG_BUFFER_SIZE, "%d:%d:1",
cells[Robot].mouseColumn, cells[Robot].mouseRow);
          snprintf(top, TOP_BUFFER_SIZE, "%d/Walls/V", Robot);
          client.publish(top, msg);
        }
        Robot = 3;
        Minimum = 3;
        digitalWrite(LED_BUILTIN, HIGH);
      }
```

```
// wifi start as client
void startWiFiClient()
{
  Serial.println("Connecting to " + (String)ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.println("WiFi connected");
  Serial.println("IP address: " + WiFi.localIP().toString());
}


// wifi start as access point
void startWiFiAP()
{
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
  Serial.println("AP started");
  Serial.println("IP address: " + WiFi.softAPIP().toString());
}
```

```
void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected())
  {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "Master";
    // Attempt to connect
    if (client.connect(clientId.c_str()))
    {
      Serial.println("connected");
      // Once connected, resubscribe to these topics
      client.subscribe("Cells");
      client.subscribe("0/Go");
      client.subscribe("1/Go");
      client.subscribe("2/Go");
      client.subscribe("Walls/V");
      client.subscribe("Walls/H");
      client.subscribe("0/Walls/V");
      client.subscribe("0/Walls/H");
      client.subscribe("1/Walls/V");
      client.subscribe("1/Walls/H");
      client.subscribe("2/Walls/V");
      client.subscribe("2/Walls/H");
      client.subscribe("Green/Location");
      client.subscribe("Blue/Location");
      client.subscribe("White/Location");
    }
    else
    {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 3 seconds");
      // Wait 3 seconds before retrying
      delay(3000);
    }
  }
}
```

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  // set borders of the map
  Cells::Borders();
  // solve the map and find all pathes
  Cells::solve();
  // set green robot start location
  cells[GreenRobot].mouseColumn = 6;
  cells[GreenRobot].mouseRow = 0;
  cells[GreenRobot].mouseHeading = 2;
  // set blue robot start location
  cells[BlueRobot].mouseColumn = 0;
  cells[BlueRobot].mouseRow = 6;
  cells[BlueRobot].mouseHeading = 0;
  // set white robot start location
  cells[WhiteRobot].mouseColumn = 6;
  cells[WhiteRobot].mouseRow = 6;
  cells[WhiteRobot].mouseHeading = 0;
  //start Serial monitor
  Serial.begin(115200);
  Serial.println("\n");
  // Start WiFi
  if (WiFiAP)
    startWiFiAP();
  else
    startWiFiClient();
  // Start the broker
  client.setServer(mqtt_server, 1883);
  client.setCallback(onData);
  Serial.println("Starting MQTT broker");
  if (!client.connected())
    reconnect();
}
void loop()
{
  if (!client.loop())
    reconnect();
}
```

Chapter 5: Challenges

Big Accomplishments

It is usually very easy to break a single stick but when things go big and we find out that a hundred sticks await for us to be broken, this is the true challenge that we had.
Making a single piece of code to do a single function is what we usually used to do but here we are now after breaking hundreds of sticks, looking behind to see all that we have been through to get here, let's take a fast overview on the biggest challenges that we really consider beating them truly big accomplishments.

Optimum Microcontrollers

One of the main challenges is choosing the right microcontroller as we have many required peripherals as a minimum requirement for our slaves to be built, let’s mention some:

1-Analog signal read

Every slave should have 8 sensors to be able to distinguish between white and black color, follow a black line in a white area, and detect walls.

2-Built-in Wifi

Having a wifi built-in on the microcontroller is not a must as we can have it on a separate circuit on the Pcb but for minimization having as many peripherals as possible on the smallest possible area is a must so we can make small and lightweight slaves.

the esp32 was the answer for many and the solution for many required peripherals which saved us so much effort and provided us with perfect performance on a small chip.

Lighting Noise

The main guild for the slave to move is to detect signals from his environment to know where exactly is the black line to keep following and where is the white area to avoid, which required a pure signal from the sensor every short period to redirect the slave, but the pure signal is impossible in an environment with many lighting sources. So we needed to filter the sensor signal before

taking a decision on its value, so we wrote an algorithm to read a Raw signal from the environment then open the robot light and read another signal, and subtract the amplitude of both signals to get a pure value that can be compared to a threshold later to know exactly what color the sensor is reading.

Power Management

If we were asked what is the most challenging part of the entire project the answer will be instantly power management as providing the circuit with enough power for a long period of time to keep the slave up and running is a must while

there are problems that can’t be predicted exactly and calculations are not enough at the same time, because of the tolerance percentage of every component, and there is a risk to burn components while testing.

Chapter 6: Conclusion and Future Work

‘Our imagination is the only limit to what we can hope to have in the future.’
Charles F. Kettering

Conclusions

One of the main goals of our work is to express this problem with constraints and to propose methods to solve it based on what we have studied over the last five years in our college.

In this study, we addressed the problem of warehouse management systems and had an overview of possible solutions to enhance them in every aspect aiming for an error-free system, maximizing the utilized capacity, decreasing the expenses, and increasing the speed of the internal processes.

And to do that we had to lean on technology and try to automate every operation as possible, as most of these goals can be achieved using technology.

In the beginning, we searched for the best solution to begin working on, then looked for the knowledge requirements to follow and start learning them including:

1-the software technologies like NodeRed and C programming language to program our system.

2-hardware market analysis to know what is valid to do what we need from the available stock in our market.

3-designing tools to make a virtual implementation before getting our feet wet.

And lastly combining all this knowledge to make a fully tested and integrated system consisting of 3 partial systems that we had previously mentioned to make a user-friendly system that can efficiently manage an entire warehouse.

Future Work

This paragraph can go very long as what we really wanted to do was far more than what we have reached because of the limitation of resources and time but we still have many improvements to our system will mention a few here:

1-replacing the mastermind so instead of being just a central unit that can make simple computation and limited communication to be a set of drones loaded with cameras that have computer vision to monitor the entire warehouse and recognize the status and the location of every slave and report this information to a central unit that is responsible to analyze the data using machine learning algorithms to find the best scenario to achieve the goal.

2-building products database to get perfect management of the stocks and giving early warning if any product nears the out.

3-redesigning the slaves to give them a higher speed and equip them with lifters to be able to pull the product and hold it independently without the need for labor to place it on the slave's head.

Reference

1. https://www.zoho.com/inventory/guides/common-problems-in-warehouse-management-and-their-solutions.html
2. https://www.encyclopedia.com/social-sciences-and-law/economics-business-and-labor/businesses-and-occupations/warehouse
3. https://www.marketsandmarkets.com/Market-Reports/smart-warehousing-market-199732421.html
4. Bluetooth®: https://www.bluetooth.com/
5. Wi-Fi Alliance: https://www.wi-fi.org/
6. MQTT - The Standard for IoT Messaging: https://mqtt.org/
7. OASIS: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt
8. ESP32: https://www.espressif.com/en/products/socs/esp32
9. LDR: https://www.electronics-notes.com/articles/electronic_components/resistors/light-dependent-resistor-ldr.php
10. Lithium-ion battery: https://en.wikipedia.org/wiki/Lithium-ion_battery
11. NCR18650B: https://www.orbtronic.com/content/NCR18650B-Datasheet-Panasonic-Specifications.pdf
12. ESP8266 MCU: https://www.espressif.com/en/products/socs/esp8266
13. Node-RED: https://nodered.org/