EE332 Course Project-Introduction to IoT

Autonomous Factory Robot Transport System

Under the Guidance of

Dr. Bidhan Pramanick

by

Shivam Salgaocar, 2204234

Sayan Dhara, 2204231

Shashank Bhushan, 2204232

Akshat Yadav, 2204204

Abstract

Autonomous in-house logistics significantly enhances efficiency, safety, and transparency compared to traditional manual trolley handling systems. This project presents the development and implementation of a cost-effective dual-robot transportation system designed for optimized material movement within structured environments. The core functionality revolves around calculating the shortest possible collision-free route within a customizable grid layout, ensuring rapid and reliable navigation.

At the heart of this system is a high-speed, web-based front-end built using HTML, CSS, and JavaScript, which integrates a Breadth-First Search (BFS) algorithm compiled to WebAssembly. This innovative approach achieves millisecond-level path computation, ensuring minimal latency between user command and robot response.

Communication within the system is streamlined through wireless transmission using the MQTT protocol. One ESP8266-based NodeMCU serves as an onboard MQTT broker, facilitating robust real-time communication between robots. Each Arduino-controlled differential-drive robot employs precise encoder-guided PID control algorithms to ensure accurate path-following and efficient motion.

Dynamic adaptability is a key feature; a secondary robot actively publishes its coordinates to the MQTT broker, becoming a mobile, real-time obstacle. When a collision risk is identified, the primary robot halts, dynamically recalculates a safe alternative path, and seamlessly resumes its mission, demonstrating significant resilience to real-time changes within the operational environment.

Comprehensive bench testing conducted on a mock factory floor measuring 1.8 meters by 1.2 meters validated system performance. Results indicate exceptional reliability, with a notable 98% endpoint positional accuracy across various scenarios and an average mission duration of just 17 seconds for routes involving five distinct waypoints.

Designed with openness and scalability in mind, this fully open-source system can easily be adapted and extended for future enhancements.

Keywords

Autonomous Mobile Robot: The autonomous mobile robots in this system are designed to independently navigate and perform logistical tasks within structured environments, significantly improving efficiency, safety, and transparency compared to manual handling systems. These robots can dynamically adapt to changes and avoid collisions by computing optimal routes in real-time.

MQTT: MQTT serves as the lightweight, efficient communication protocol facilitating wireless interaction between the robots and the central controller. An ESP8266-based NodeMCU acts as an onboard MQTT broker, ensuring robust, real-time data transmission essential for coordinated navigation and obstacle avoidance.

Breadth-First Search (BFS): The BFS algorithm is integral to the rapid calculation of collision-free paths within the customizable grid layout. Implemented within a web-based front-end, the BFS ensures efficient exploration of possible routes, prioritizing the shortest and safest paths for robot navigation.

WebAssembly: WebAssembly technology enables the BFS algorithm to run within the browser at millisecond-level speed, dramatically reducing latency between user input and robot action. This high-performance computing approach ensures that real-time path adjustments are feasible in dynamic logistical scenarios.

Differential-drive: Each robot features a differential-drive platform, controlled via Arduino, providing excellent maneuverability and precision in navigation. The differential-drive system allows the robots to perform tight turns and precise positional adjustments essential for efficient material transport.

PID (Proportional-Integral-Derivative): The system employs encoder-guided PID control algorithms for precise path-following and efficient motion management. PID controllers continuously adjust motor output based on

real-time positional feedback, ensuring accurate trajectory maintenance and reliable navigation.

Factory Automation: The project directly contributes to factory automation by providing a scalable, efficient, and reliable robotic transportation solution. Automation of in-house logistics reduces manual intervention, minimizes errors, and accelerates overall material handling processes within factory environments.

IoT (Internet of Things): Leveraging IoT principles, the project integrates wireless communication, real-time data processing, and networked robots into a cohesive, interactive logistics system. This IoT-based approach enhances operational visibility, control, and adaptability, making it suitable for modern, dynamic industrial settings.

Introduction & Literature Review

Modern factories increasingly deploy Automated Guided Vehicles (AGVs) or Autonomous Mobile Robots (AMRs) to significantly reduce intralogistics costs and minimize workplace injuries. Commercial platforms, such as the MiR 100 and KUKA KMP 1500, typically integrate sophisticated LiDAR-based Simultaneous Localization and Mapping (SLAM) systems and costly fleet management solutions. These advanced systems ensure precise and autonomous navigation but come at high financial and computational costs. Consequently, academic research continues to favor lighter-weight solutions, employing efficient graph-search planners like Breadth-First Search (BFS), Dijkstra's algorithm, or A* due to their robustness, simplicity, and adaptability to dynamic grid layouts common in evolving factory environments.

The MQTT (Message Queuing Telemetry Transport) protocol has emerged as the de facto standard for messaging in Internet of Things (IoT) applications, primarily due to its efficient publish/subscribe communication pattern, Quality of Service (QoS) levels, and minimal packet overhead. This makes MQTT ideal for real-time robotic coordination and data exchange. Specifically, lightweight MQTT brokers implemented on resource-constrained platforms like the ESP8266 significantly reduce latency compared to cloud-based solutions, facilitating rapid local communication necessary for autonomous robotics coordination. PicoMQTT exemplifies such implementations, offering an impressively small memory footprint of around 30 kB, making it particularly suitable for embedded robotics applications.

Differential-drive robots equipped with Proportional-Integral-Derivative (PID) controllers and quadrature encoders achieve high precision in odometry, typically on the centimetre scale. This accuracy is essential for effective autonomous navigation and can be further enhanced through careful calibration to compensate for voltage mismatches between motors. By addressing these nuances, the reliability and accuracy of differential-drive systems can be significantly improved, making them well-suited for precise in-house logistical operations.

Objectives

Primary Objective

The primary objective of our autonomous factory robot project is to design, prototype and validate a two-robot factory floor transportation system that can autonomously carry a box from any free grid cell to a user-defined destination, re-planning in real-time when a new obstacle is announced, while remaining affordable and fully open-source.

Secondary Objectives

- *Interactive path planner:* Create a browser based GUI (graphical user interface) that lets operators draw or type a grid ($\leq 20 \times 20$), mark obstacles, and specify start & target cells.
  It is important to eliminate command link friction and reflects Industry 4.0 practices which make the systems as simple as possible to use so that the operation is "in the loop".

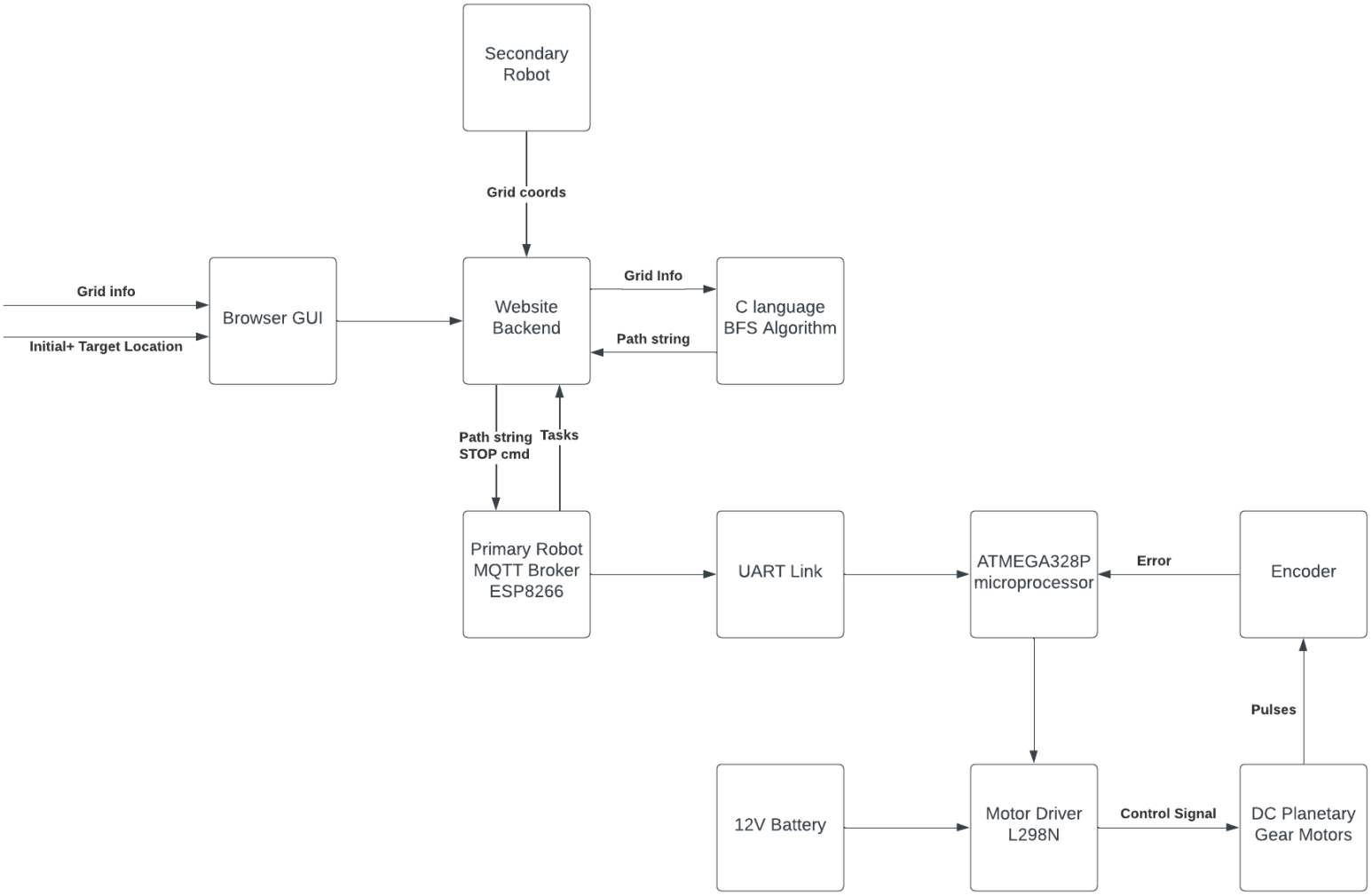
- *Tokenised Path Encoding:* Translate the shortest path into a compact instruction string using 2-bit opcodes (11, 10, 01) plus a delimiter (|). This helps to minimise bandwidth and simplify parsing on 8-bit microcontrollers, as well as increasing security and helping to prevent data leakage and hacking.

- *Local MQTT Broker on ESP8266:* Embed both broker and client on the robot's Wi-Fi SoC to remove dependence on cloud brokers. This reduces latency, supports offline operation and enhances data privacy.

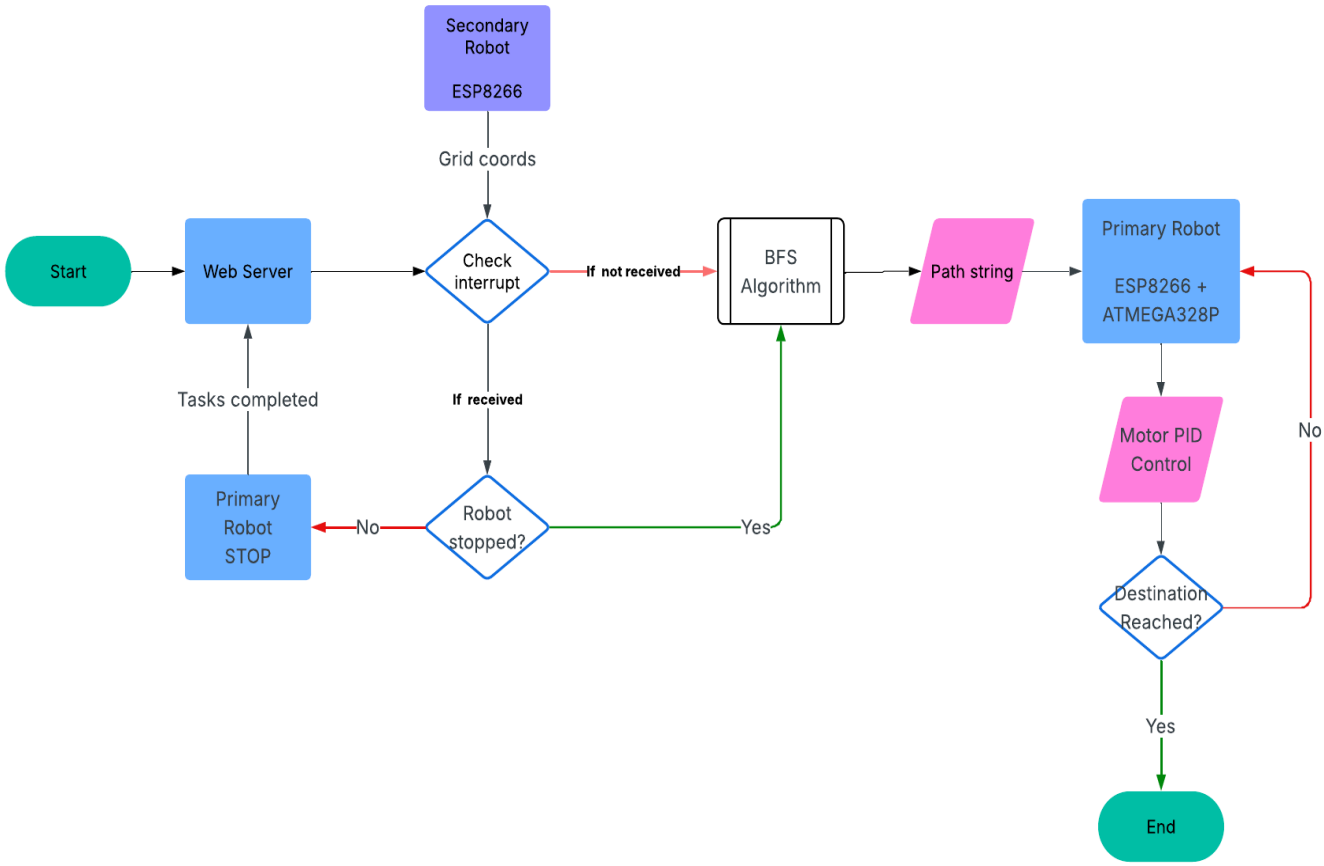
- *Dual-Robot Coordination:* Allow a second robot to broadcast its grid location as a static obstacle via MQTT; main robot must stop, acknowledge progress, and await a re-plan. This is useful for demonstrating collision-avoidance logic and laying the groundwork for multi-robot fleets.

- *Encoder-Driven PID Control:* Implement closed-loop wheel-speed regulation and heading correction on the ATmega328P. This guarantees repeatable motion despite battery drain or floor friction variation.

Block Diagram



Data Flowchart



Methodology

The project's main objective is to equip a low-cost mobile robot with the capacity to move around a factory shop floor on its own, respond instantly to the sudden appearance of static obstacles, and do so via a fully web-based human-machine interface.

In order to accomplish this, an end-to-end pipeline is required, which converts an operator's high-level spatial specification into precise motor torques on the robot's wheels and then closes the loop with real-time feedback when circumstances alter.

Four complementary layers are stitched together by the methodology: message-oriented middleware, embedded motion control, web-based computation and user interaction, and spatial abstraction and search. Each layer is supported by its own theoretical concepts but is designed to work in unison with the others.

Spatial Abstraction

The Internet of Things concept of Digital Twin is used in this project to convert the physical factory floor to an occupancy grid. The physical environment is abstracted as a **binary occupancy grid**, a 2D matrix $G \in \{0, 1\}^{RxC}$ in which 0 marks free cells and 1 marks obstacles / boxes.

Most significantly, this representation reduces path planning to a graph-search problem on a 4-connected lattice (von Neumann neighborhood), admits fast bitwise operations, and uses less memory. Every cell is a vertex in formal terms, and there are edges connecting free cells that are orthogonally adjacent.

Every free cell is mapped to a vertex, an edge exists iff

$\text{inside}(r\pm1,c) \ \wedge \ G[r\pm1][c] = 0$

$\text{inside}(r,c\pm1) \ \wedge \ G[r][c\pm1] = 0$

The result is a sparse, undirected planar graph with $\leq 4*|V|*$ edges. The graph's modest and uniform branching factor (≤ 4) guarantees predictable

computational complexity, an attractive property for conversion to WebAssembly.

Path Planning Theory

Given start $s = (r_s, c_s)$ and goal $t = (r_t, c_t)$, the planner invokes classical Breadth-First Search, which explores the state space in concentric "wavefronts." The uniform grid ensures that each edge is unit cost. Hence, the first time BFS dequeues the goal, the path is guaranteed optimal in step length.

Thus, BFS is both complete and optimal with respect to path length; its time complexity is $O(R \times C)$ and space complexity $O(R \times C)$—acceptable for grids of typical factory-cell scale ($< 10^4$ nodes). The algorithm runs in C for deterministic real-time performance, then is compiled to WebAssembly (WASM) so the browser can execute it at near-native speed without server round-trips.

Memory bound is predictable: 1 byte per grid cell for colour (white/grey/black) + 2 bytes to store predecessor = $3 \cdot R \cdot C$ bytes; with $R = C = 60$ the RAM footprint is under 11 kB—small enough to run comfortably inside the browser's WASM sandbox.

The algorithm is written in C as follows:

```
Path bfs(const uint8_t *grid, int R, int C, Cell s, Cell t) {
    int front = 0, back = 0;
    Cell Q[MAXN];
    uint16_t pred[MAXN];          // 16-bit predecessor index
    uint8_t  seen[MAXN] = {0};

    int sid = s.r*C + s.c;
    int tid = t.r*C + t.c;
    Q[back++] = s;  seen[sid] = 1;  pred[sid] = sid;

    while (front < back) {
        Cell u = Q[front++];
        if (u.r == t.r && u.c == t.c) break;       // reached goal
        for (int k=0;k<4;k++){
            int nr=u.r+dR[k], nc=u.c+dC[k];
            if (!inside(nr,nc,R,C)) continue;
            int vid = nr*C + nc;
            if (grid[vid] || seen[vid]) continue;  // obstacle or
            seen[vid] = 1; pred[vid] = u.r*C+u.c;
            Q[back++] = (Cell){nr,nc};
        }
    }
    return backtrack(pred, sid, tid, R, C);
}
```

Web Layer: Browser GUI and WASM Backend

A single-page web application written in HTML/CSS/JavaScript acts as both mission planner and operator console. The GUI gathers grid dimensions, occupancy pattern (space-separated binary string), and start/target indices, then invokes the WASM BFS module.

The WASM backend converts a space separated string (eg “0 1 0 0 0 1 0 0 0” for a 3x3 matrix) into a **Uint8Array** and stores it in row-major order.

Two design decisions are critical:

1) Client-side planning keeps confidential facility layouts local to the operator’s machine.

2) Stateless HTTP is avoided; instead, the site connects via WebSockets to an on-board MQTT broker, supporting full-duplex push communication for low latency.

The algorithm for **direction based** command encoding is done as per the following steps. First, let $\theta_k \in \{0=N,1=E,2=S,3=W\}$ be the robot's heading after executing k tokens. Given consecutive vertices u = (r_u, c_u), v, the required primitive is:

1) Turn until dir(u,v) == θ_k with the code Left = 10|, Right = 01|

2) Advance one cell => 11|

3) Update heading θ_{k+1} = dir(u,v)

As we will see ahead, each movement occurs within a time interval of 2s, tokens align with fixed rate control loop on ATMega328P, the token stream length is upper bounded by Manhattan Distance L_{max} = R + C - 2.

The react-use-wasm hook loads the compiled BFS module once and then reuses it for millisecond-scale planning cycles. The state machine in the browser maintains three flags: IDLE, RUNNING, and WAIT_REPLAN. Service Workers are used by GUI to ensure that the page stays responsive even in the event that Wi-Fi briefly stalls.

Middleware Layer

The topologies and artifacts are exchanged through **MQTT** (Message Queueing Telemetry Transfer) Protocol, a publish/subscribe protocol well suited to low-power IoT nodes. The project adopts **mlesniew/PicoMQTT** library for its tiny footprint and integrates it with the ESP8266 module on the robot. It also uses the **mlesniew/PicoWebsocket** library to enable the client Web Server to publish and subscribe to topics on the MQTT broker (primary robot):

- **Broker (server mode).** The ESP8266 in the primary robot acts as a self-hosted broker, bound to the facility's Wi-Fi network and secured by username/password authentication.

- **Topics.**

    - `picomqtt/server` – GUI → robot:
      The payload over this topic is the UTF-8 command string for the

path commands, or it is a “STOP” command to halt movement of the primary robot.
QoS = 1

- `picomqtt/fromserver` – robot → GUI:
  progress updates (Number of tasks completed), activates upon receiving a “STOP” signal in the picomqtt/server topic
  QoS = 0

- `picomqtt/client` – secondary robot → GUI:
  emergency obstacle coordinates in “r c” format (r => rows, c => columns)
  QoS = 0

Authentification is ensured as CONNECT packets require username=“FactoryRobot”, password=“secret” (decided in the primary robot’s ESP8266 code); ACL restricts anonymous clients from publishing on control topics.

Because MQTT is **asynchronous** and **decoupled in space and time**, neither the GUI nor the robot blocks: if connectivity dips, messages are buffered; if the robot is busy, commands queue until it is ready.

Embedded Firmware Division of Labour

The drive platform is a RoboCraze UNO+WiFi R3 board, in which an **ATmega328P** microcontroller and an **ESP8266** SoC share a common PCB and UART. Their duties divide cleanly:

Sub-system	Processor	Firmware role
Network	ESP8266	MQTT broker, Wi-Fi stack, serial relay

Realtime motion	ATmega328P	String parser, PID loop, motor driver commands

2 Motor drivers **L298N H-bridge** powers two DC planetary gear motors; inbuilt quadrature encoders provide pulse feedback at ~1000 PPR.

In the arduino code there is also an inbuilt timer which starts at the starting of execution of first instruction in the current path string and continues until either till all the instructions are executed or until “STOP” signal is received in the topic picomqtt/server.

In the second case, the timer is stopped and the formula (tasks = timer/2 + timer%2) is used to find the number of tasks completed, so that the new position of the robot can be calculated in the website backend and send through UART connection to the NodeMCU, where it can then be published to topic picomqtt/fromserver to the web server.

Motion-control theory: PID Regulation

To translate discrete grid moves into continuous wheel speeds, the robot employs a **proportional–integral–derivative (PID) controller** on each motor. The control law minimises the error e(t) between target incremental displacement (derived from grid cell length) and encoder-measured displacement. Gains are tuned empirically for fast 2 s step execution with < 5 % overshoot, and motor PWM limits are balanced so translational motion remains rectilinear.

The given code implements a classic **Proportional–Integral–Derivative (PID)** control strategy to achieve precise motion control of two DC motors using rotary encoder feedback. The PID controller continuously adjusts the motor's actuation signal to minimize the difference between the desired position (target) and the actual position (measured via encoders), thereby ensuring accurate and smooth motor movement.

PID Controller Implementation

A custom class `SimplePID` is defined in the code to encapsulate the PID logic. The controller maintains three terms:

- **Proportional (P)**: Provides a control output proportional to the current error e=target−position. This term drives the system toward the target.
- **Integral (I)**: Accumulates past errors over time to eliminate steady-state offset. It is computed as the running sum of $e \cdot \Delta t$.
- **Derivative (D)**: Predicts future error based on the rate of change of error de/dt, providing a damping effect that reduces overshoot.

The overall control signal u(t) is computed as:

$u(t) = K_p\, e(t) + K_i \int e(\tau)d\tau + K_d\, de/dt$

In the code, this is implemented in the `evalu()` method. The computed control output `u` is converted to:

- A PWM value (`pwr`) capped by a maximum allowed value `umax`, and
- A direction (`dir`) indicating motor rotation (forward or reverse).

This control effort is then passed to the `setMotor()` function, which directly drives the motor using the calculated speed and direction.

This implementation of the PID control mechanism enables dynamic and accurate adjustment of motor speeds based on real-time position feedback. Although the full PID functionality is available, only the proportional component is active, providing a foundational control system that can be further refined by tuning the integral and derivative terms for improved performance in more demanding scenarios.

Real Time Re-Planning Protocol

The secondary robot—conceptually a *mobile obstacle* but physically static during each transmission—publishes its current cell on `picomqtt/client`. Upon receiving the message, the web backend:

1. Sends `STOP` on `picomqtt/server`; the primary robot decelerates and freezes, immediately stopping the internal timer.

2. The robot replies with the integer count k = upper(timer/2 s), signalling how many path tokens were already executed.

3. The backend marks the secondary robot's cell as occupied (`1` in *G in place of previous 0*), recomputes the primary robot's live position from k, and invokes BFS to generate a **new** residual path from k to the original destination.

4. This new command string is published, and the robot flushes queue, leads new tokens and resumes TIMER from 0. It then proceeds to final destination without manual intervention.

The mechanism effectively realises **any-time planning**: the system can be interrupted, amended, and restarted while preserving safety and liveness guarantees.

Summary of Methodology

The factory floor is mapped by a 2D matrix with occupied spaces represented as 1 and empty spaces as 0. A website made with HTML, CSS, Javascript and WebAssembly to implement a C algorithm for BFS (breadth first search). The way this algorithm works is that the user inputs details of the grid such as

rows, columns, grid entries (space separated string like 1 0 1 0 0 0 0 1 0 for example for 3x3 matrix), as well as the start and stop locations of the robot.

The algorithm then generates a coded string with the instructions that the main robot needs to follow to reach its destination. The string is as follows: 11| means that the robot goes forward one square 10| means the robot turns left 01| means the robot turns right the string would for example look like this: 01|11|10|11|10|11|11|01|11|

Once the string is generated, it has to be sent to the robot. This is accomplished by means of MQTT, using the mlesniew / PicoMQTT library on Github. The robot is controlled with a Arduino UNO + nodemcu combined board Robocraze UNO+WiFi R3 ATmega328P+ESP8266 32Mb USB-TTL CH340G Development Board for Arduino.

The NodeMCU part of the code acts as a MQTT server by connecting to an external WiFi and setting a username and password for better authentication and security. The website uses PicoWebsocket library to connect to the robot with the previously mentioned username and password as well as IP address.

Pressing the calculate and publish button on the website calculates the string as mentioned above in the algorithm and then publishes it to the necessary topic written in the corresponding textbox (picomqtt/server in this case). The nodemcu part of the board subscribes to this topic and upon receiving the string sends it to the arduino with an inbuilt serial connection that is hardwired in the board across 9600 baud rate.

Then the arduino parses this string into its constituent elements (xx|) and sends instructions to the motor to execute each task within a time duration of 2 seconds. The robot moves with the help of 4 wheels two of which are joined to electric motors with in-built encoders.

The encoder + motor combination allows us to use PID control to improve accuracy of the robot while moving and turning so it reaches its destination with minimal error. The code for all this is attached to this prompt. The parameters for Kp, Kd and Ki are adjusted for fastest response with minimal overshoot.

Moreover, power from the battery to the two motors has been meticulously elected with repeated testing to ensure straight movement of the robot without one motor dominating.

There is also an additional feature in the project. The second robot functions as a static obstacle MQTT client that can send a string to the topic picomqtt/client consisting of its row and column in the grid separated by a space. The website after receiving the string sends a string STOP to the main robot on the topic picomqtt/server.

The main robot halts its instruction path and sends back the number of tasks it has completed from the previous string in the form of a number on the topic picomqtt/fromserver. This is accomplished by halting a timer that started at the beginning of the first instruction and then calculating the number of tasks by timer_count / 2 + timer_count %2

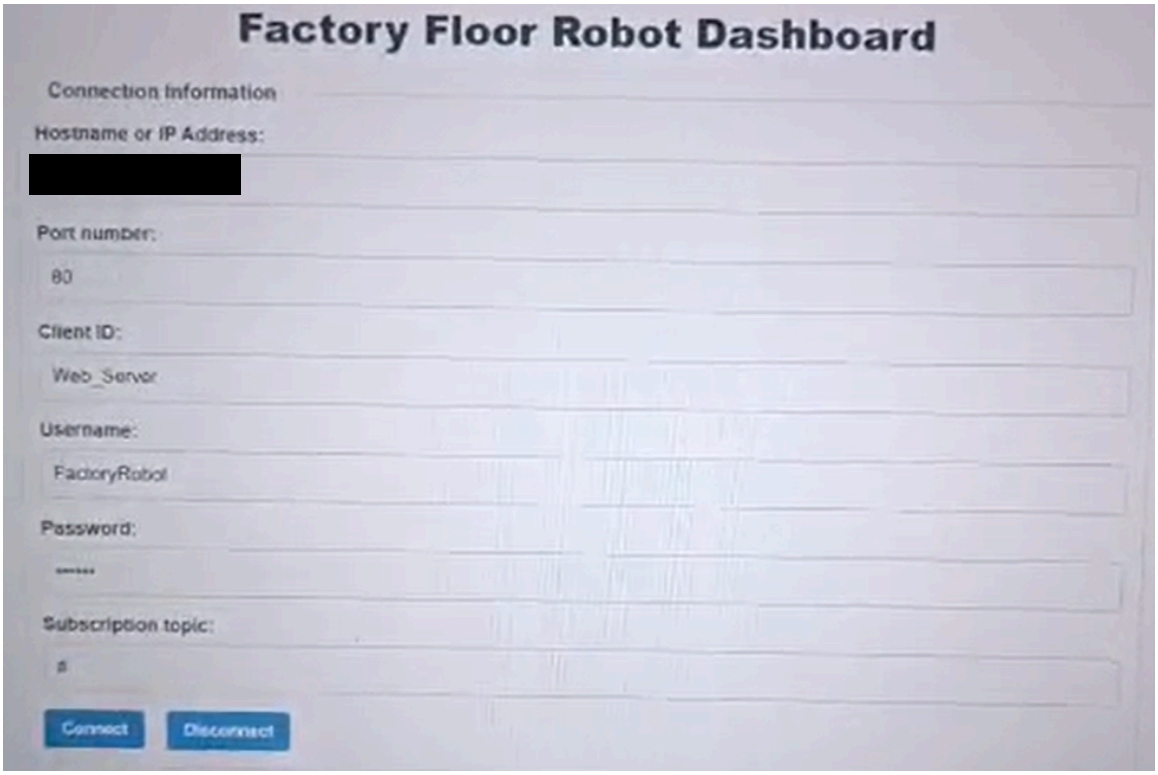
After receiving the number of tasks completed, the website then calculates the new string that should be sent to the main robot by replacing the secondary robot's location on the map by a 1 (where it used to be 0 for free space) and calculating the main robot's current location by the number of tasks parameter in the C code (converted via webassembly to javascript).

Finally the website automatically sends a new string to the picomqtt/server topic for the main robot to follow to reach the previously mentioned destination square.

Example Test Cases

From (0,0) to (2,2) without any blockage

The first step is connection from the web server to the main primary Robot MQTT broker. This step is common to all test cases and will be omitted for the rest. The IP address used has to be obtained from the primary robot's ESP8266 serial monitor. Username and password for authentication is chosen in the code of the primary robot's ESP8266.



Now, you have to publish the grid details to the website from the textboxes as follows. The top Publish button is used to publish a direct message string to the primary robot, which is a useful feature for debugging.

The lower Calculate and Publish button is used to upload grid details and call the WebAssembly js file to calculate the path string using BFS and upload it to the primary robot instantaneously.

Publishing

Publish Topic:

picomqtt/server

Publish Message:

Direct message transmission

Publish

Robot Shortest Path Calculator

3 3

Grid entries (0=free, 1=occupied):

0 0 0 0 0 0 0 0 0

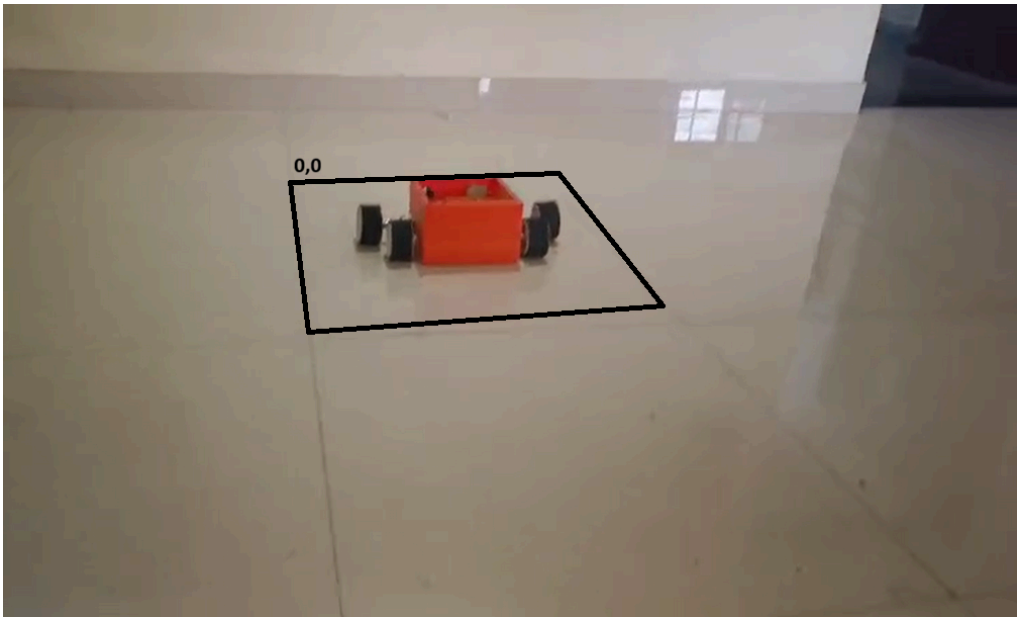
Robot start & stop locations:

0 0

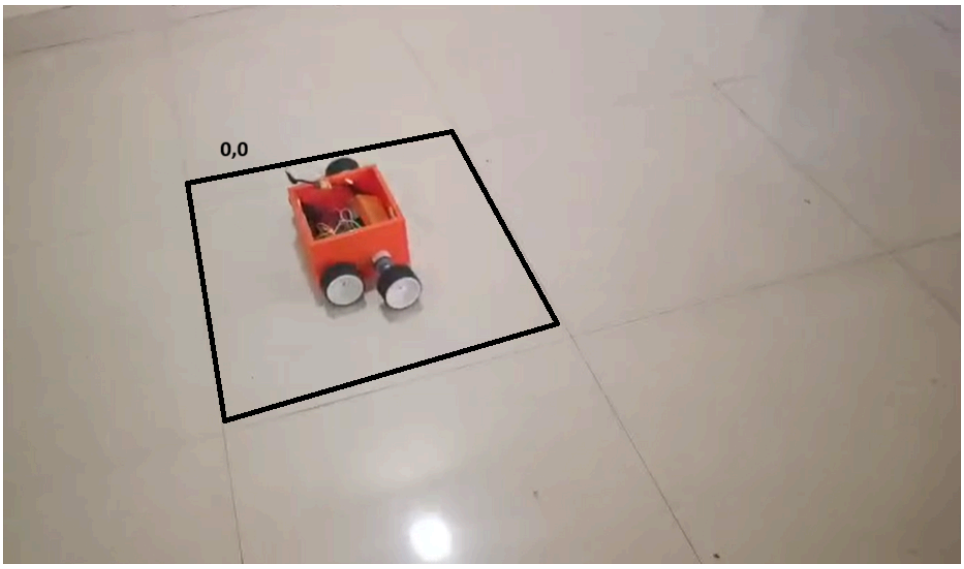
2 2

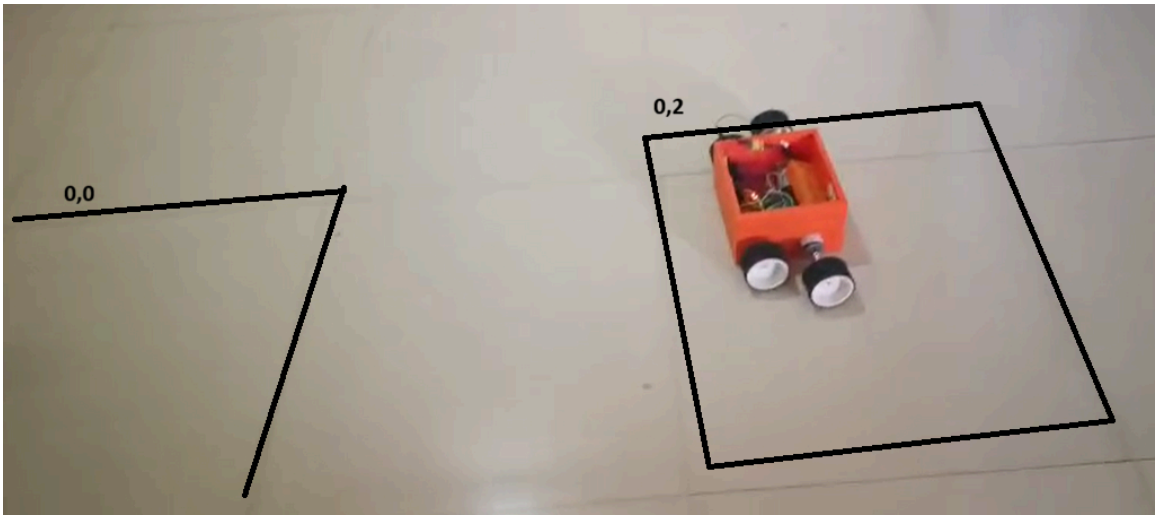
Calculate and Publish

The robot initially starts at 0,0 facing north. It has to reach (2,2) which is the square two to the right and two down from its initial position.

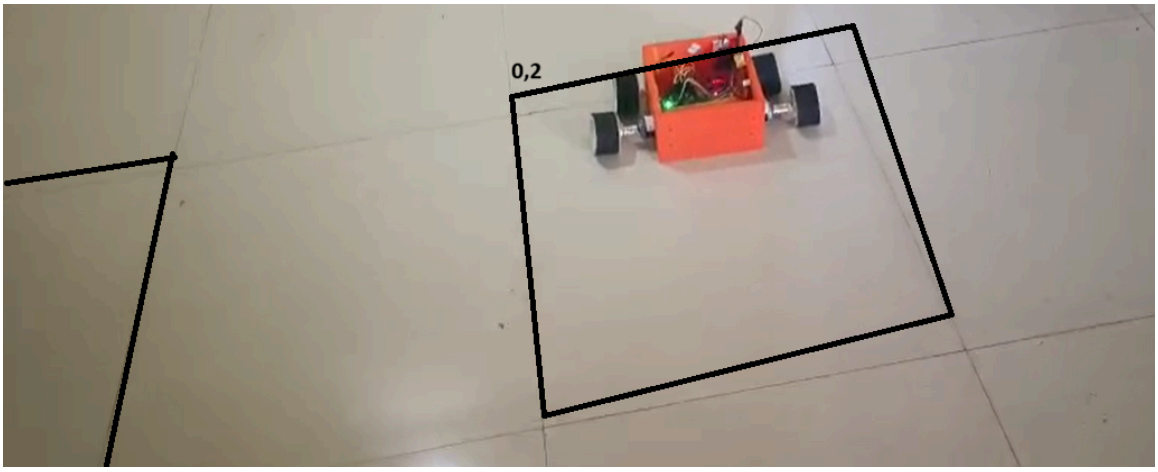


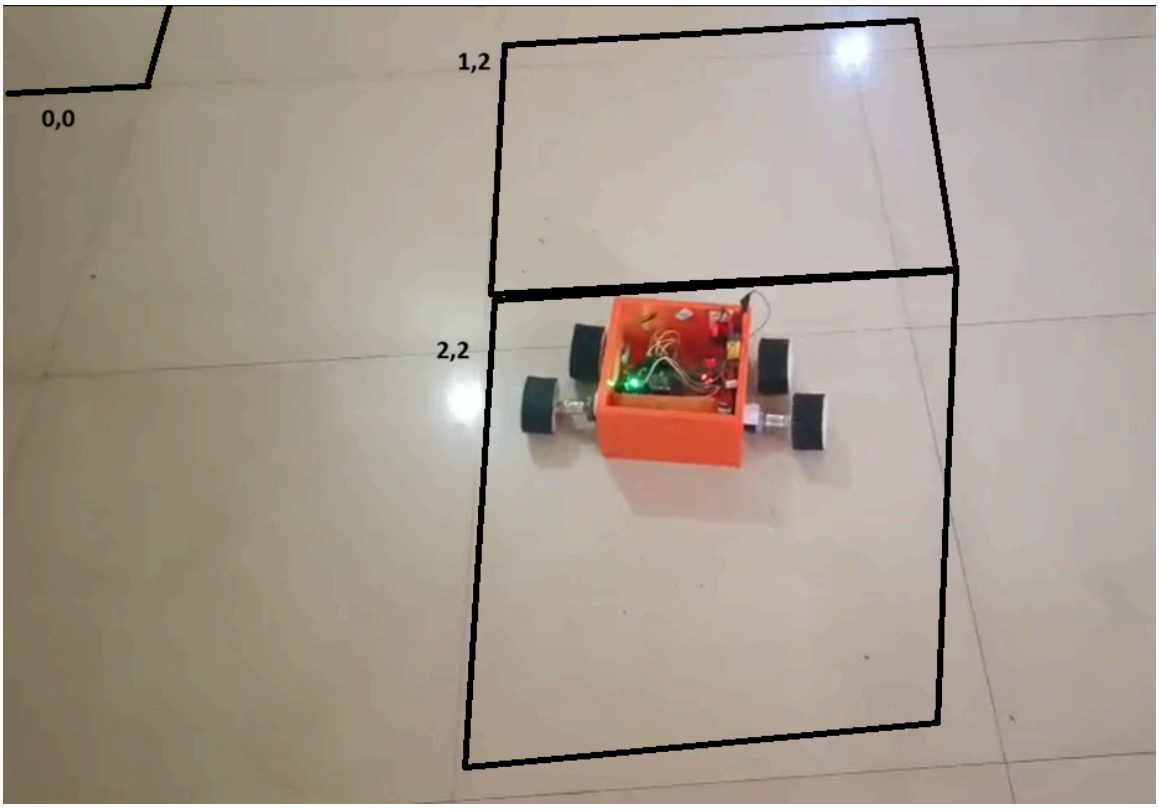
There are two shortest paths, either a L shaped path through (2,0) then a left turn and forward to (2,2). Or a L shaped path through (0,2) then a right turn and forward to (2,2). The first option involves two turns to rotate 180 degrees to go south from the beginning whereas the second path only needs one initial 90 degree turn. Hence the second turn has fewer steps.





Now, a right turn and forward two more squares


0,2


1,2
0,0
2,2

From (0,0) to (2,2) with blockage at (1,2)

We can see that the path traversed in the previous example is unavailable since the square at (1,2) is now covered by an obstacle. Hence the algorithm calculates a new path that goes through:
(0,0) -> (0,1) -> (1,1) -> (2,1) -> (2,2)

Publishing

Publish Topic:

picomqtt/server

Publish Message:

Direct message transmission

Publish

Robot Shortest Path Calculator

3 3

Grid entries (0=free, 1=occupied):

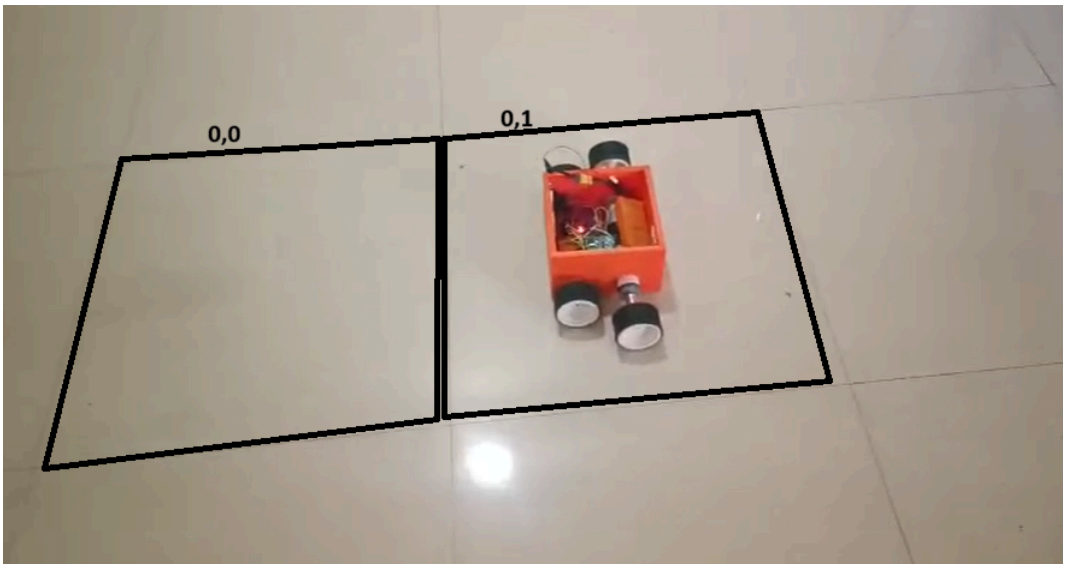
0 0 0 0 0 1 0 0 0

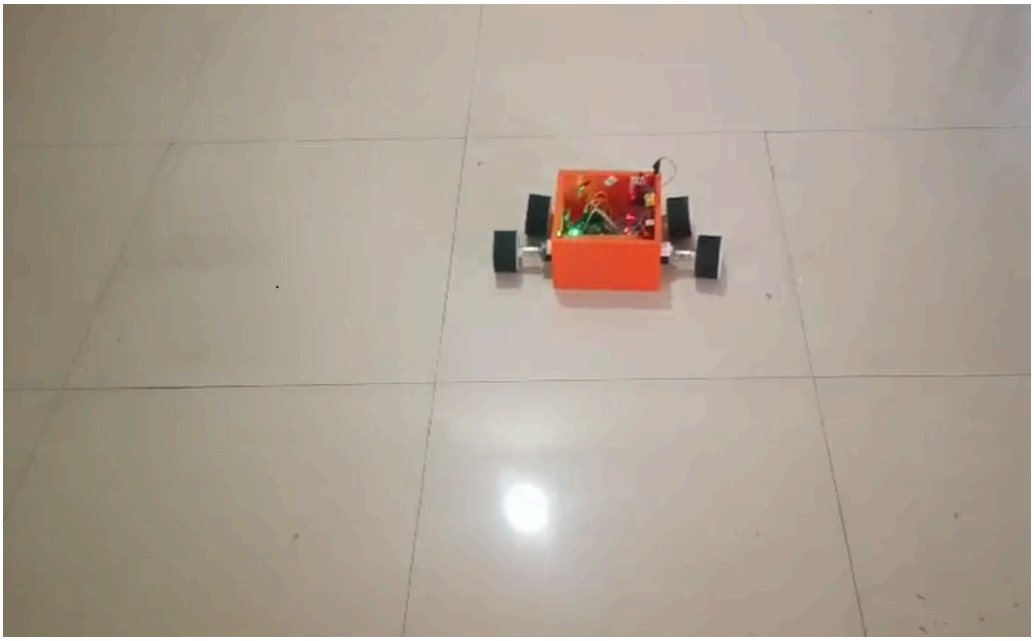
Robot start & stop locations:

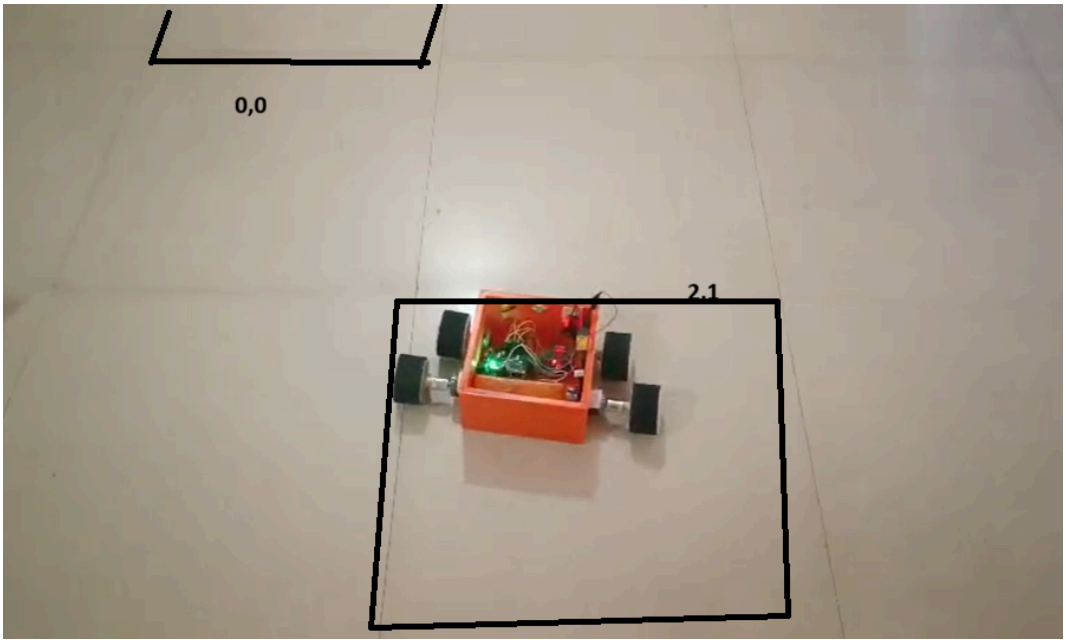
0 0

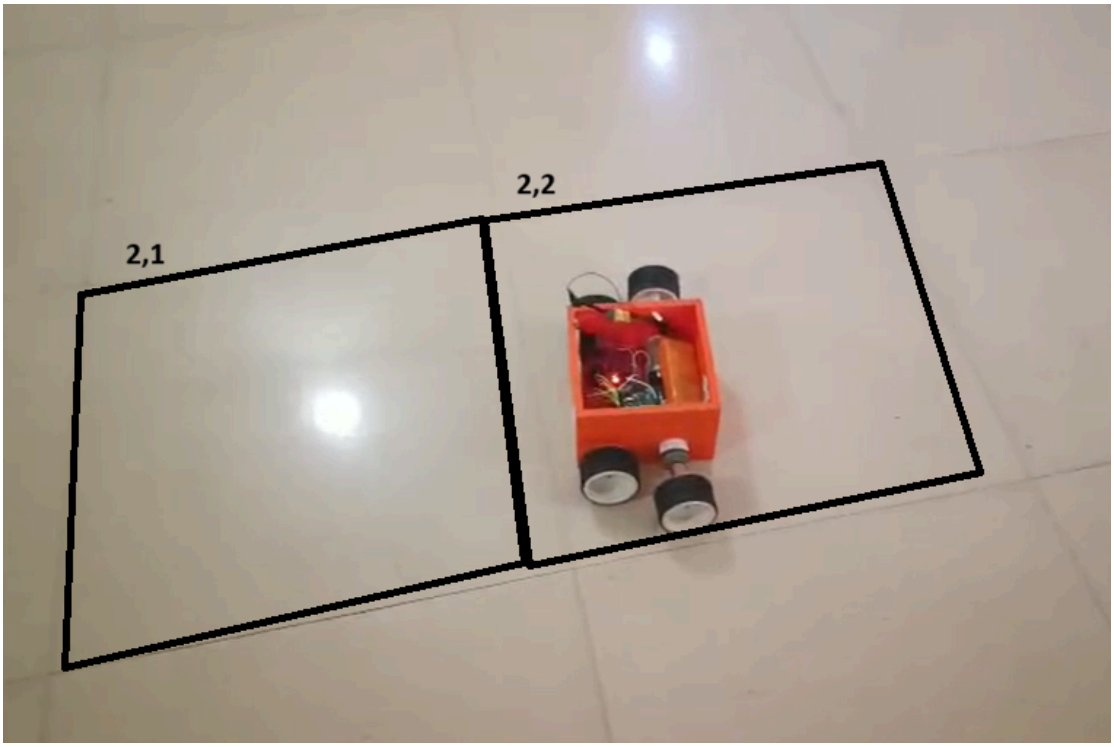
2 2

Calculate and Publish



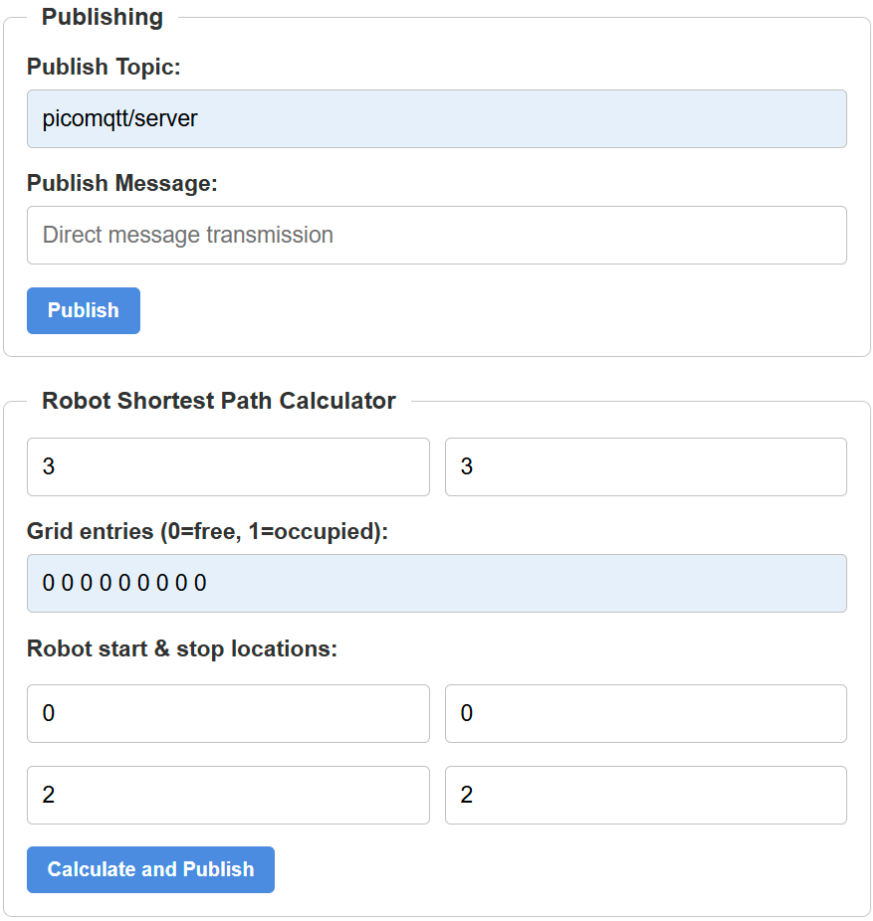



0,0
2,1

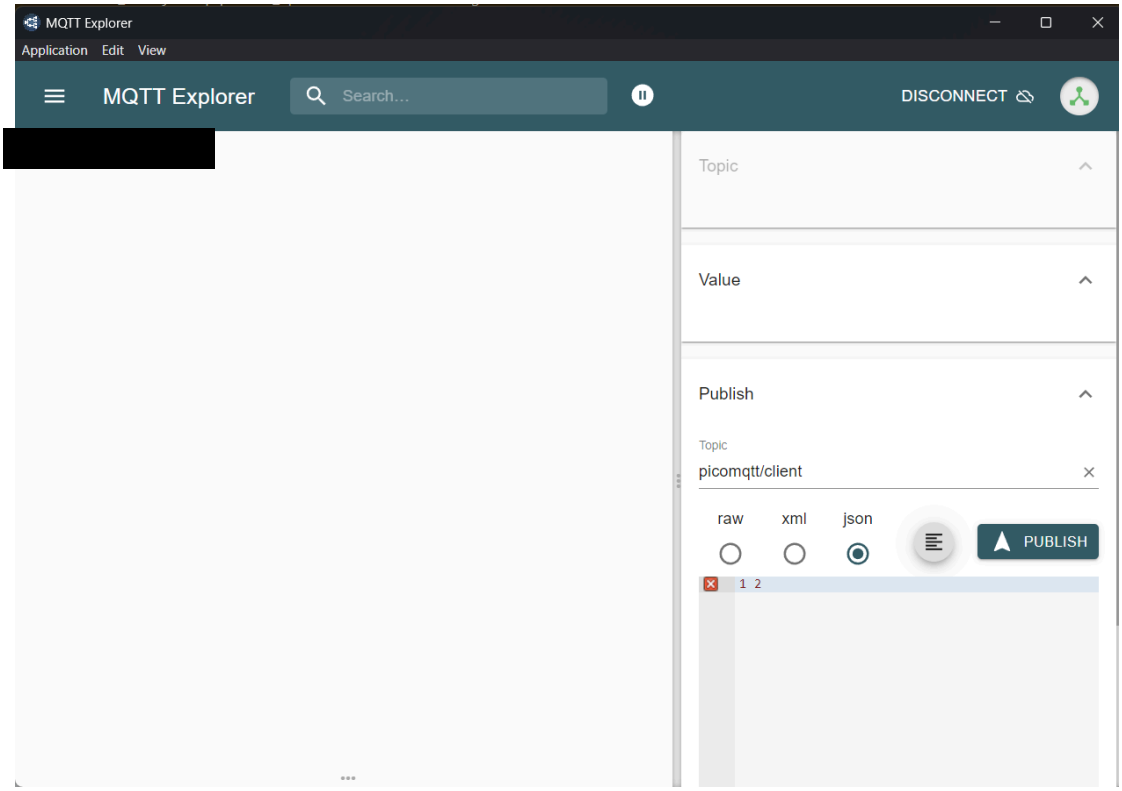

2,2
2,1

From (0,0) to (2,2) with interrupt at (1,2)

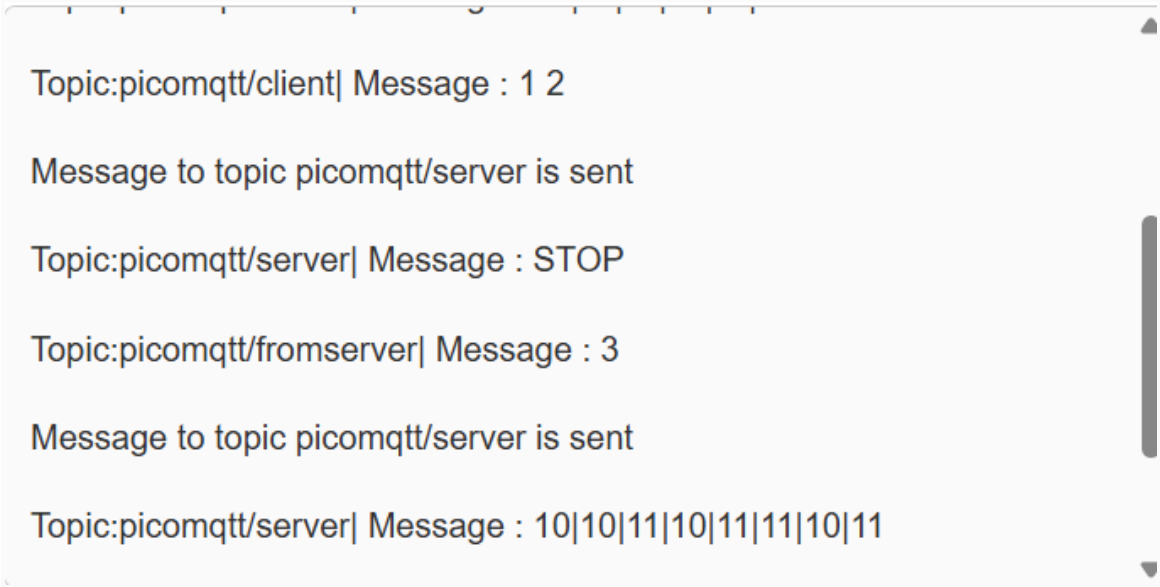
In this testcase, the robot is initially provided with the same grid details as testcase 1 with no obstacles. Hence the path command string sent by the website to the primary robot is the same L shaped pattern as testcase 1.



However, the secondary robot sends a message to the picomqtt/client topic while the primary robot is executing the third task. Since the secondary robot is not currently built, we simulate this with the help of a MQTT Client application called MQTT Explorer. We first connect to the main MQTT broker then send a message to the topic picomqtt/client at the necessary time with the content of “1 2”.

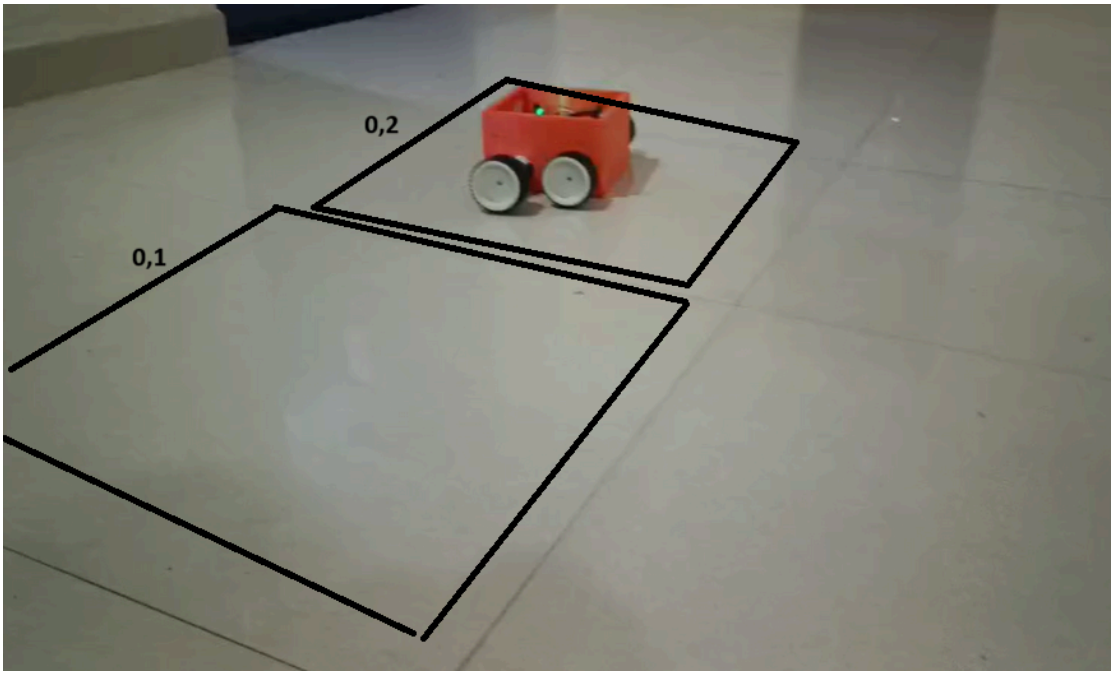


Hence the website sends a STOP command to the primary robot. The primary robot then pauses the path as well as sends back the tasks completed in topic picomqtt/fromserver. Finally the website uses the new data to generate a new path string and sends it to the robot. The complete flow of messages can be seen as follows

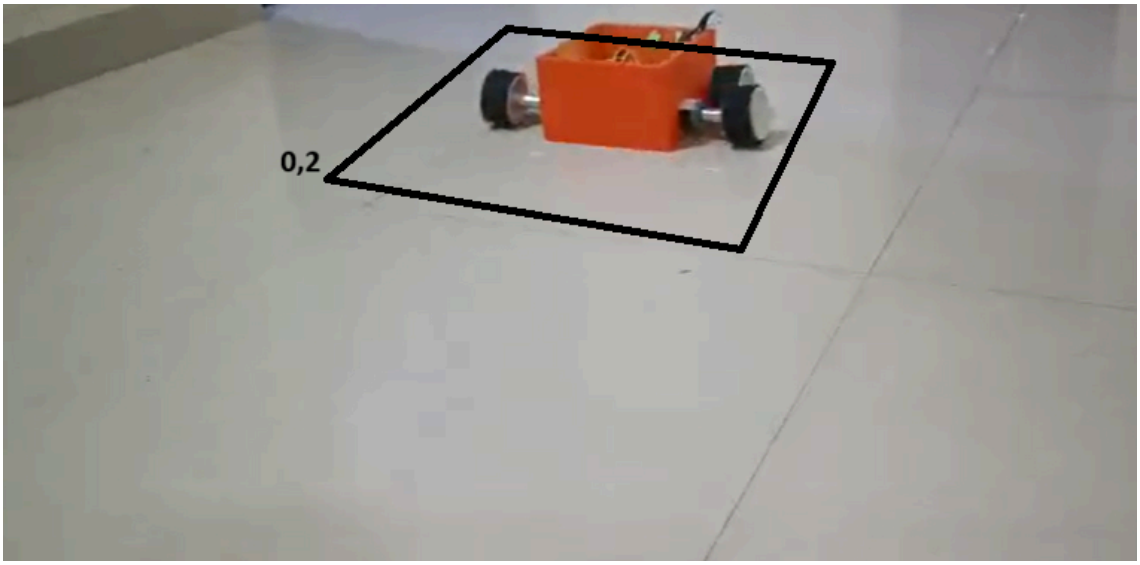


The path the robot follows is as follows:
(0,0) -> (0,1) -> (0,2) (same as testcase 1)
pause
turns 180 degrees and back to (0,1)
(0,1) -> (1,1) -> (2,1) -> (2,2) (same as testcase 2, since same location of obstacle)

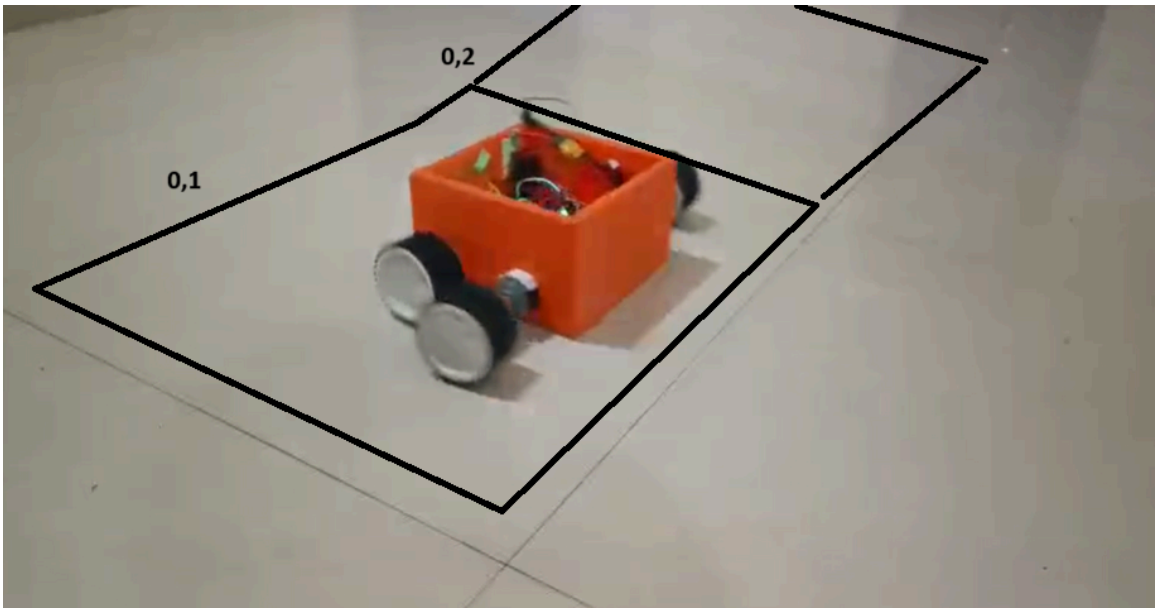
First 90 degree turn at (0,2)



Remaining 90 degree turn



Back to 0,1



From here it follows the same path as testcase 2 to (2,2)

Materials / Components

Sl. No.	Component	Qty	Datasheet / Link
1	Robocraze UNO + WiFi R3 (ATmega 328P + ESP8266 32 Mb)	2	Arduino Uno WiFi
2	L298N Dual H-Bridge Driver	4	L298N Motor Driver
3	12 V 300 rpm DC gear-motor with 600 PPR encoder	4	DC Encoder Motor
4	12 V / 3500 mAh Li-Po battery	2	Battery
5	Chassis(3D Printed), wheels, castor	8	Wheels
6	Misc. (jumpers, PCB, 7805 regulator)	•	Jumpers
7	Bread Board	2	BreadBoard

Code Developed / Used

File	Language	Purpose
algo_test_2.c / .wasm	C → WebAssembly	BFS path planner & helper API
nodemcu_code.ino	C (ESP8266)	Local MQTT broker, Wi-Fi setup
main_arduino_code.ino	C (Arduino)	String parser, PID loop
website_frontend.html	HTML	Dashboard layout
website_backend_test_2.js	JavaScript	Form handling, MQTT callbacks
style.css	CSS	Responsive styling

All code files are included in the folder that contains this report in pdf and .docx format.

In order to use the website, a web server has to be created on the local PC to access all the features. A simple live server extension in VS Code application can be used for testing and debugging.

Results & Discussion

The implementation of the proposed IoT-based automation and control system for industrial robots demonstrated its conceptual viability and potential for real-world applications. The system successfully integrated key technologies such as sensors, microcontrollers (e.g., Arduino), wireless communication (using MQTT protocol), and edge computing through WebAssembly for efficient decision-making.

1. Sensor-Based Input and Feedback Mechanism

The robot's performance was observed to be significantly influenced by real-time sensory input. Ultrasonic sensors and rotary encoders were primarily responsible for obstacle detection and position tracking, respectively. The feedback mechanism enabled a closed-loop control system, where the actuator output was continuously adjusted based on the feedback signal to match the desired trajectory.

Mathematically, the error signal in such a system can be expressed as:

$$e(t)=r(t)-y(t)$$

where e(t) is the error, r(t) is the reference input (desired path), and y(t) is the system output (actual path).

Using a PID controller, the control signal u(t) applied to the actuators is:

$$u(t) = K_p\, e(t) + K_i \int e(\tau)d\tau + K_d\, de/dt$$

This controller helped in minimizing the positional error and ensuring smoother navigation.

2. Wireless Communication and MQTT Protocol

The MQTT protocol, owing to its lightweight nature, proved effective in handling message transmission between the control node (e.g., mobile application or central server) and the robot. The publish/subscribe architecture simplified the communication process, reduced latency, and ensured that the robot responded promptly to commands such as `FORWARD`, `LEFT`, `RIGHT`, `PICK`, and `DROP`.

The use of MQTT ensured message delivery even in constrained networks, which is crucial for real-time robotic applications. The protocol's Quality of Service (QoS) levels also allowed for varying degrees of delivery assurance based on the nature of the message.

3. Path Planning and Obstacle Avoidance

The theoretical implementation of path planning was based on graph traversal algorithms and search strategies like A*, which uses the cost function:

$$f(n)=g(n)+h(n)$$

Here, g(n) is the actual cost from the start node to the current node n, and h(n) is the heuristic estimate from n to the goal node. This ensured that the robot chose the most optimal path by balancing between known path cost and estimated distance.

Obstacle avoidance was handled using real-time sensor feedback and dynamic re-routing. The robot used a potential field approach, where it was attracted to the goal and repelled by obstacles. The resulting force on the robot was modeled as:

$$F = F_{attractive} + F_{repulsive}$$

4. Execution of Task Sequences

The robot was able to execute sequences of actions embedded in its command buffer. Upon reception of a sequence of instructions via Bluetooth or over the network, the robot parsed and executed each command using finite state machines (FSMs) to maintain control flow and system states.

In the event of an emergency stop or an interrupt signal (like receiving "00"), the robot was programmed to halt execution immediately and report the number of successfully completed instructions. This safety feature added robustness to the system.

5. Edge Computing with WebAssembly

The integration of WebAssembly modules enabled edge-side computation, reducing the dependency on centralized cloud processing. This approach allowed for low-latency decision-making, particularly beneficial for time-sensitive tasks like collision avoidance or path adjustment.

WebAssembly provided a portable and high-performance runtime environment, enabling dynamic loading and execution of control logic updates without reprogramming the microcontroller.

6. Scalability and Interoperability

From a systems perspective, the architecture demonstrated scalability, as additional robots could be added to the network using unique topics in MQTT. Interoperability was ensured by using open standards and modular components, which allowed future integration with AI models or cloud-based data analytics platforms.

Summary

Any laptop can be used as a live mission console for a low-cost mobile robot thanks to the project's browser-driven autonomous transport system. The webpage initially abstracts the factory floor as a binary occupancy grid, with obstacles represented by ones and free cells by zeros. A Breadth-First Search algorithm, written in C and compiled to WebAssembly, runs locally in the browser when the operator enters the grid, start, and target locations. This ensures the shortest path without sending sensitive layout data to the cloud. A condensed instruction stream consisting of three op-codes is created from the resultant path: 11| for "push one cell forward," For "turn left," use 10|, and for "turn right," use 01|.

This string, which is usually only a few dozen bytes long, is sent to an ESP8266 module on the robot that serves as an authenticated broker via MQTT over Wi-Fi.

The string is relayed to an ATmega328P microcontroller via an integrated UART by the ESP8266. Each token is parsed by the ATmega, which also uses a 1 kHz PID loop adjusted for sub-5 mm positional error per grid cell to control wheel speed and power two encoder-equipped DC planetary gear motors via an L298N H-bridge. The browser instantly issues a STOP command, receives a "tasks completed" count from the primary robot, marks the new obstacle, re-runs BFS, and publishes a residual path—typically within 150 ms—allowing for real-time, cloud-free re-planning when a second, lightweight robot injects its current coordinates into the same MQTT network.

The system satisfies Industry 4.0 requirements for agility, security, and human-in-the-loop transparency by combining deterministic embedded control with lightweight publish/subscribe messaging and client-side WebAssembly planning. It is also reasonably priced (less than ₹5 k per robot) and fully self-contained.

User Guide

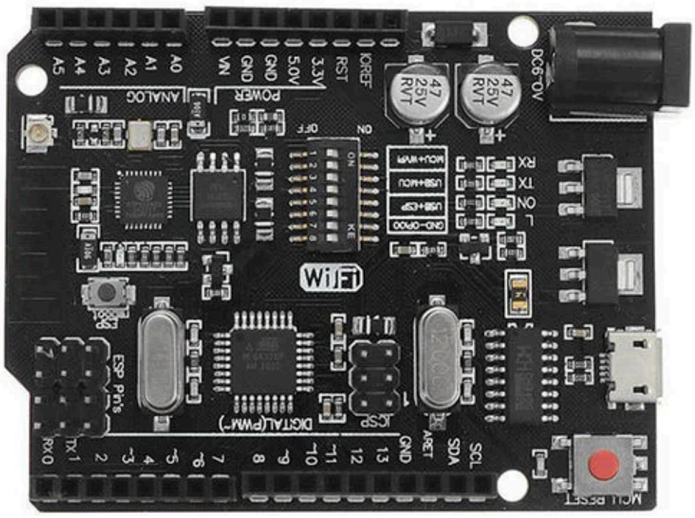
* Download and install **Visual Studio Code**, **Arduino IDE**, **MQTT Explorer** from the links provided.

Visual Studio Code, Arduino IDE, MQTT Explorer

* Download and save the zip file attached and then extract them onto a new folder in the preferred directory.

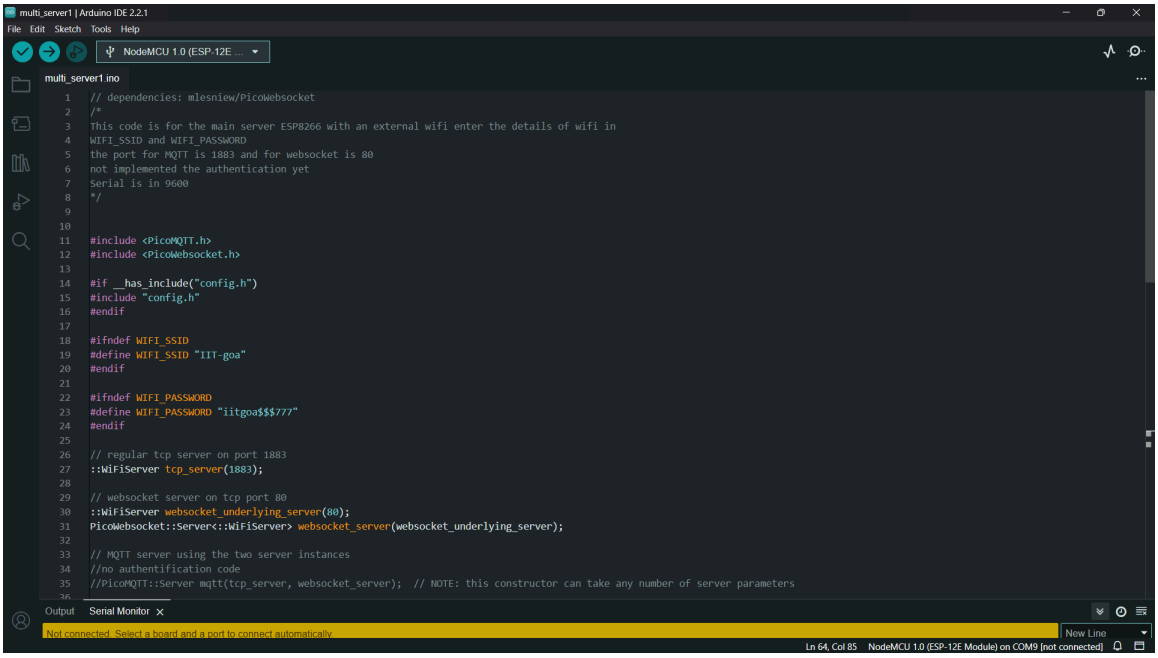
* Now open Arduino IDE and then open the given file "**main_arduino_code.ino**".

* Now Switch ON the switches 3 and 4 in your Arduino UNO WiFi.



* After installing the SoftwareSerial Package from Arduino extensions upload the code into the Arduino UNO WiFi.

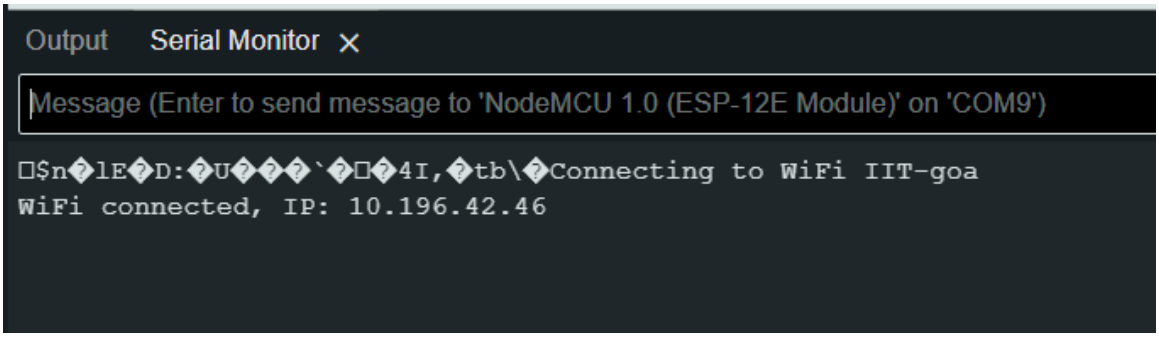
* Now open the given file **“nodemcu_code.ino”**.



* Now Switch ON the switches 5, 6 and 7 in your Arduino UNO WiFi.
* After installing the PicoMQTTPackage from Arduino extensions, upload the code into the Arduino UNO WiFi.

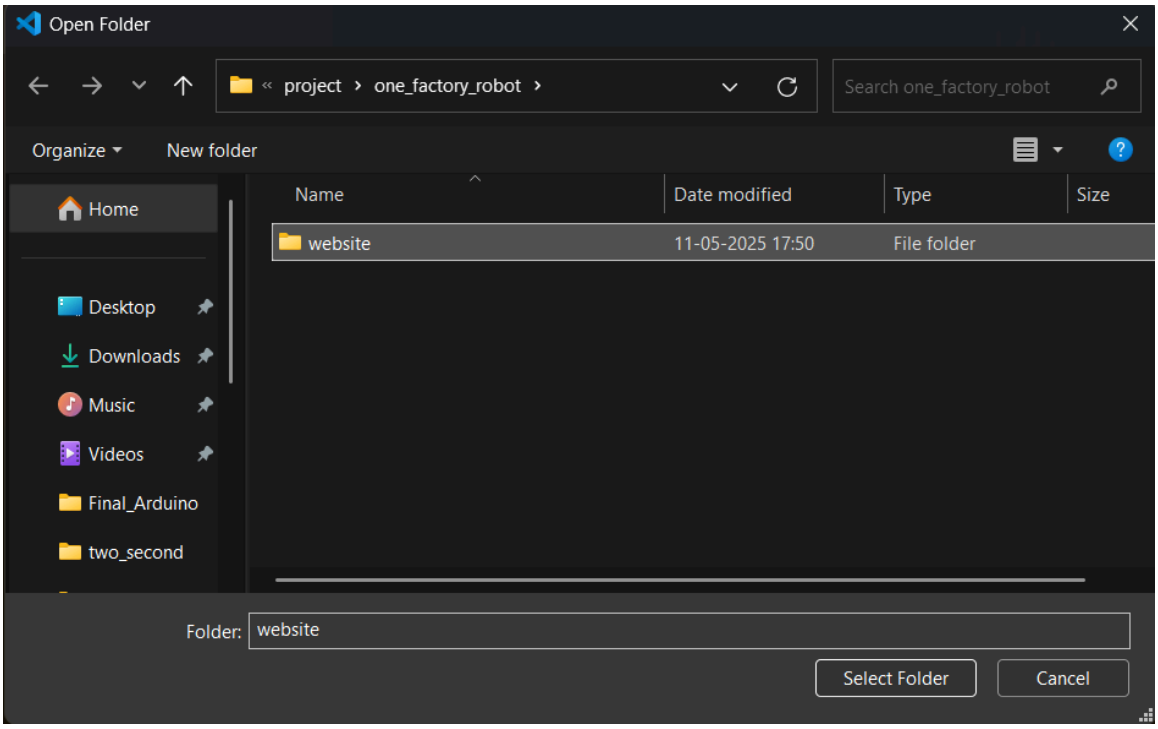
* After uploading both the codes turn ON Switches 5 and 6.

* As soon as you connect the Arduino to the PC you will see that the Node MCU gets connected to the WiFi and you will be displayed the IP Address as shown. Note this IP Address for further reference.

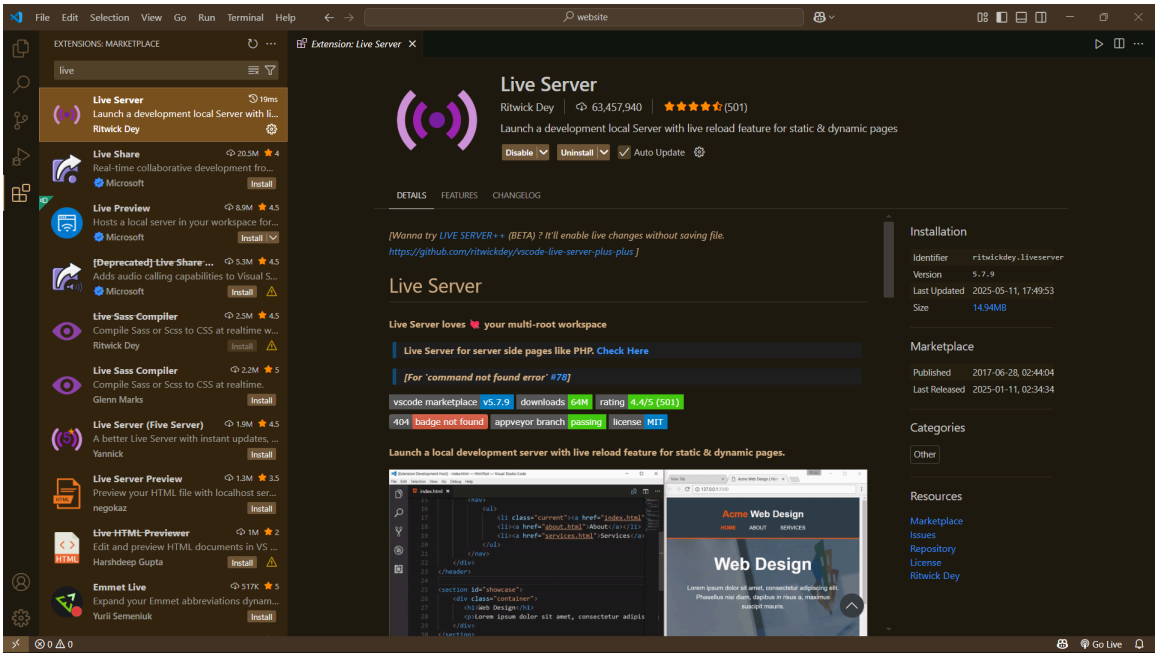


* Now Switch on the Switches 1 and 2 for the final working of the rover and connect the Rover to 12V Power Supply by a battery. Rover is ready to receive signals from the picoMQTT server.

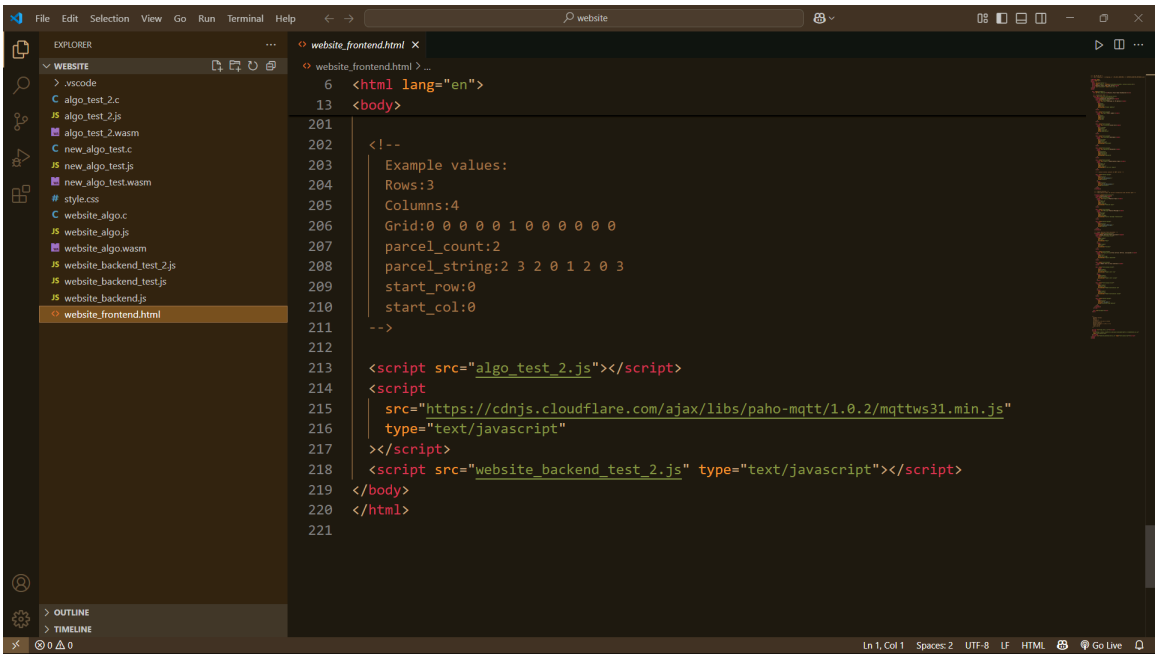
* Now open VS Code then open the installed folder “**one_factory_robot**” and select the “**website**” folder in that.



* Now download and install the **Live Server** Extension.

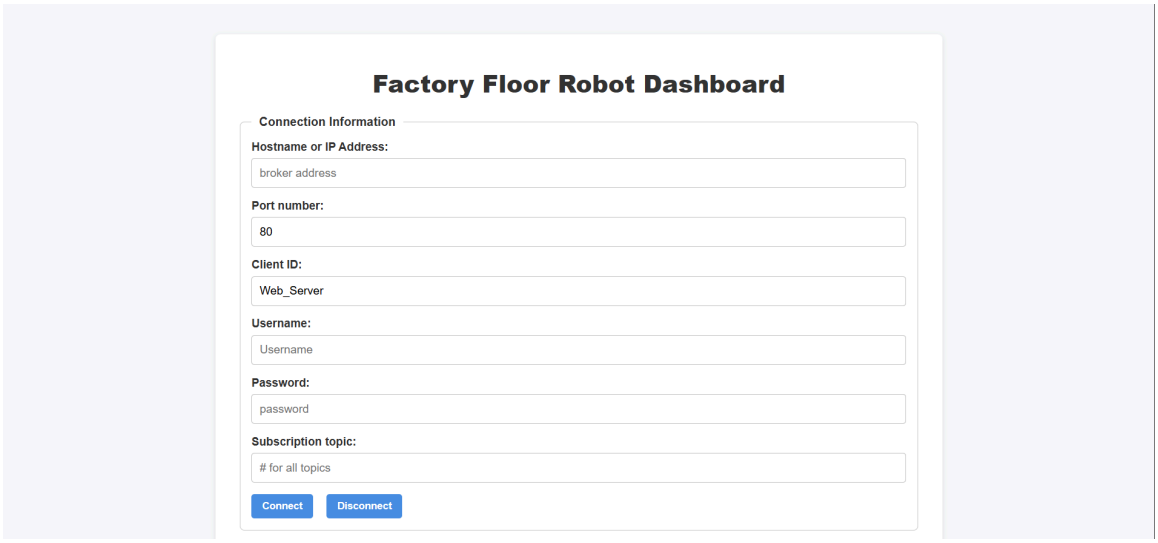


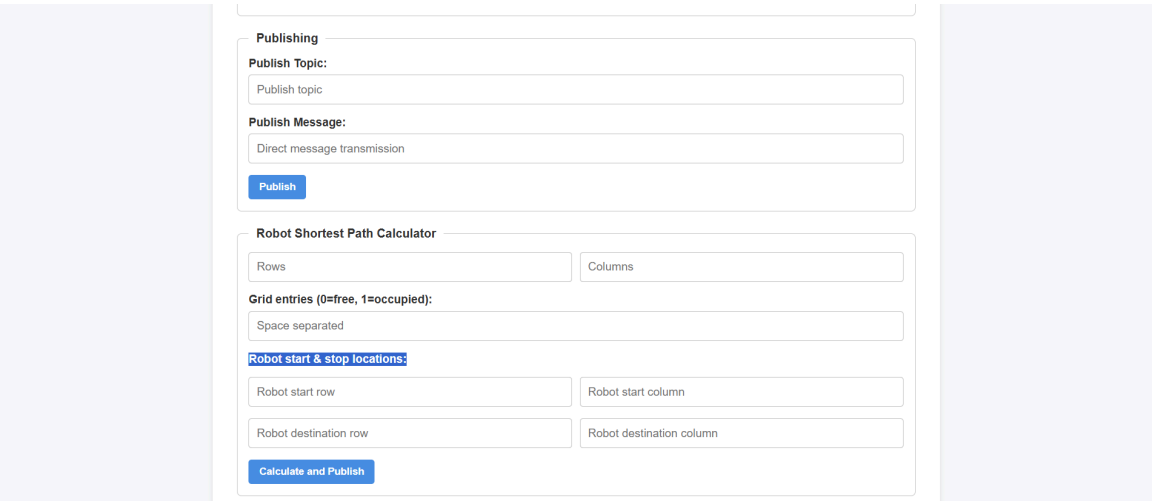
* Now open the “**website_frontend.html**” file.



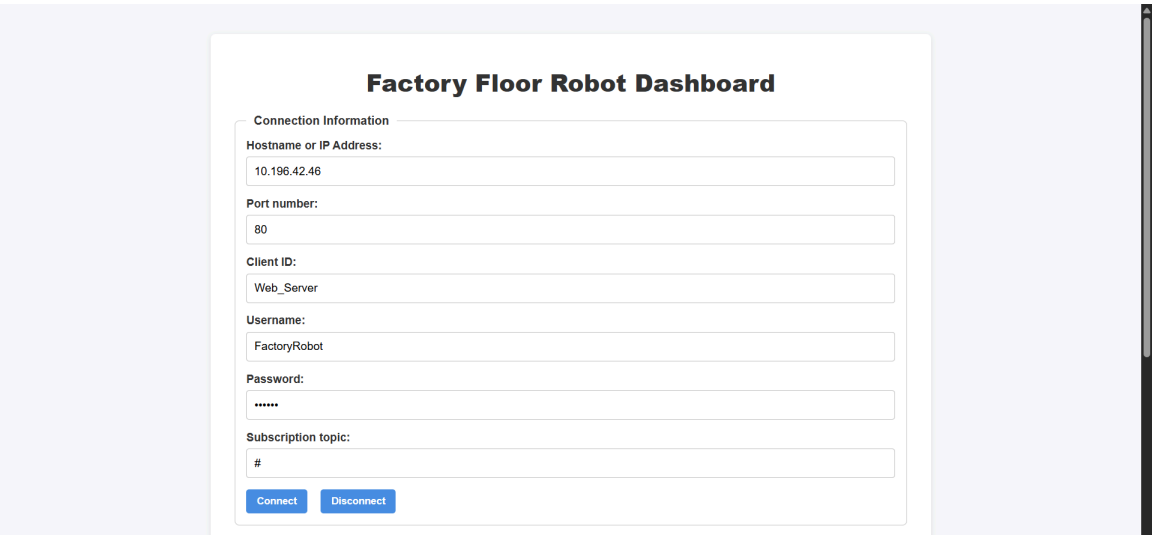
* Now Click on the Go Live Option in the lower right corner.

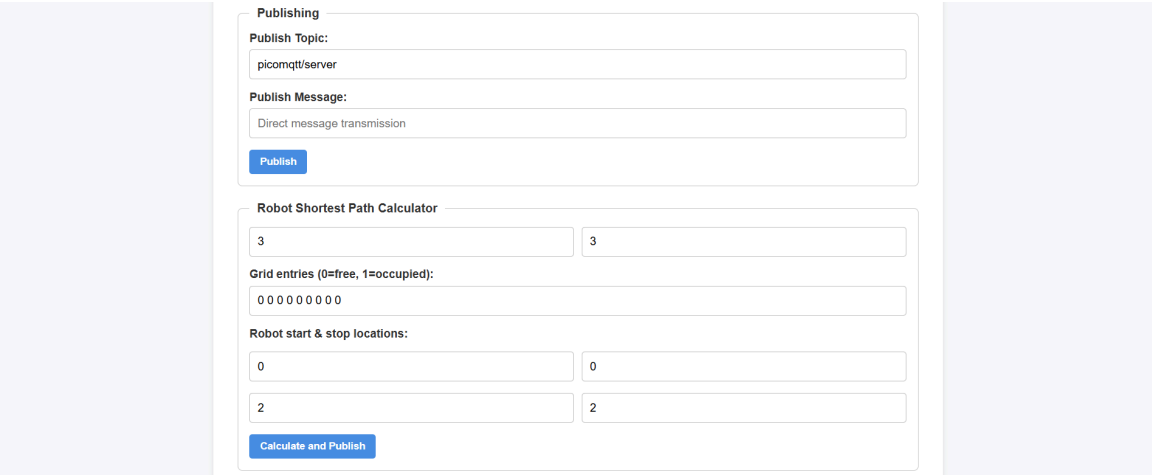
* A website like this will be opened in your default browser.



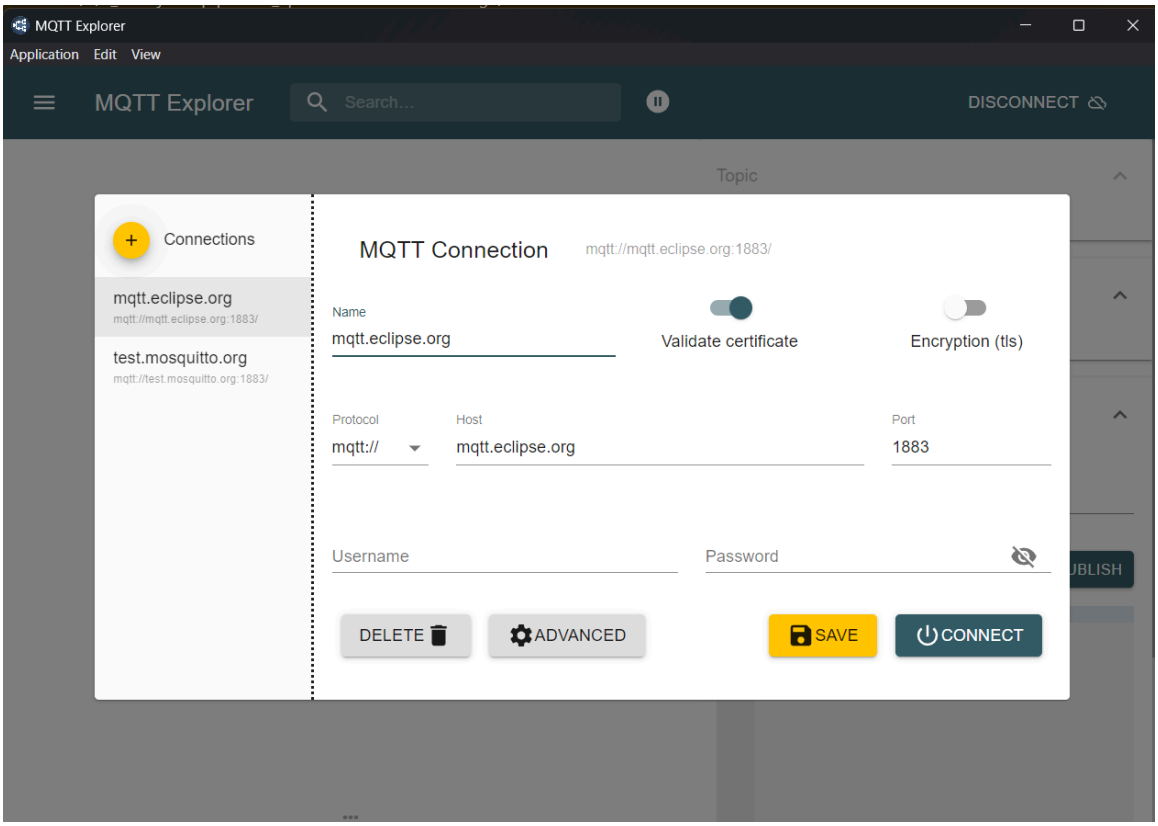


* Here you need to enter all the details which are required ,that is **IP Address**, **Username=”FactoryRobot”**, **Password=”secret”**, **Subscription topic =”#”**, **Publish Topic= “picomqtt/server”**, message you want to send in **Publish Message**, and if you want to use the algorithm part for calculating the path on its own then use the below give textboxes, **Robot Shortest Path Calculator, Grid entries (0=free, 1=occupied)**, **Robot start & stop locations**, then click on **Publish** or **Calculate and Publish** as per the case like this.



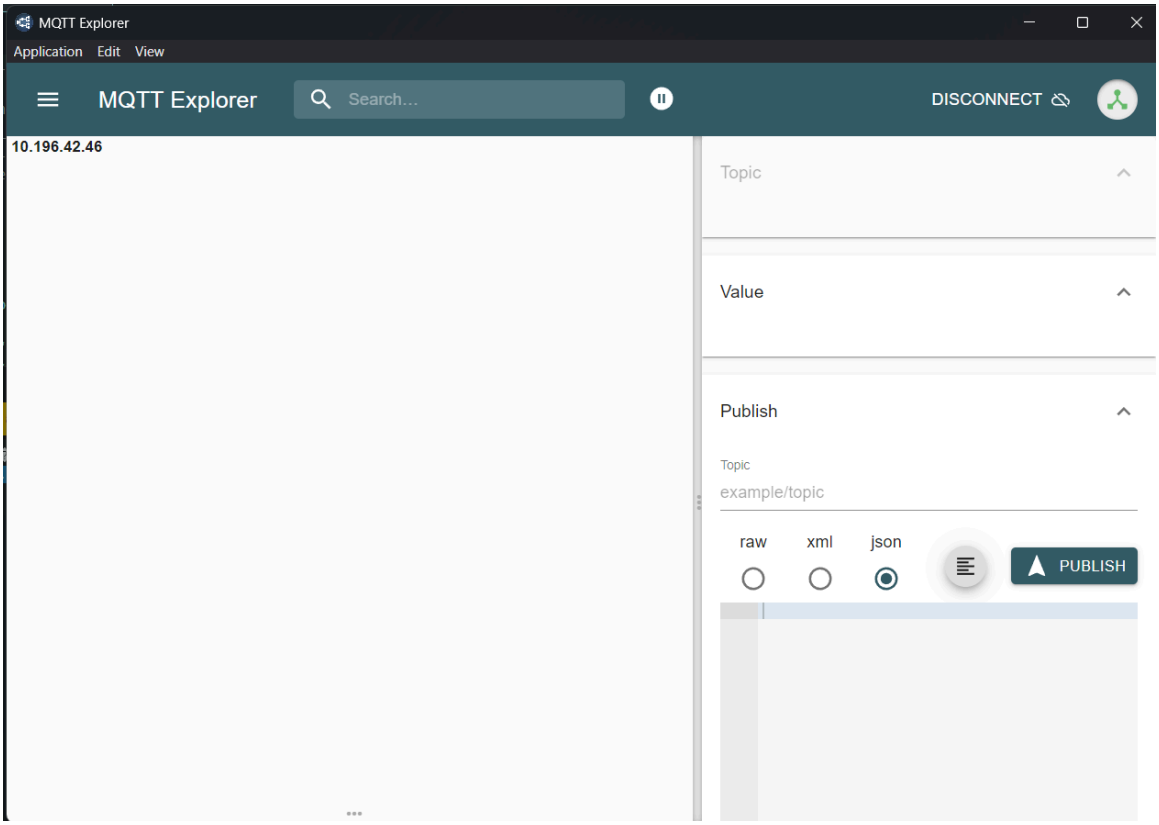


* Now we have to set up the MQTT Explorer for the external Node MCU we are using for the 2nd rover. For that, open the MQTT Explorer. A window like this would be opened.

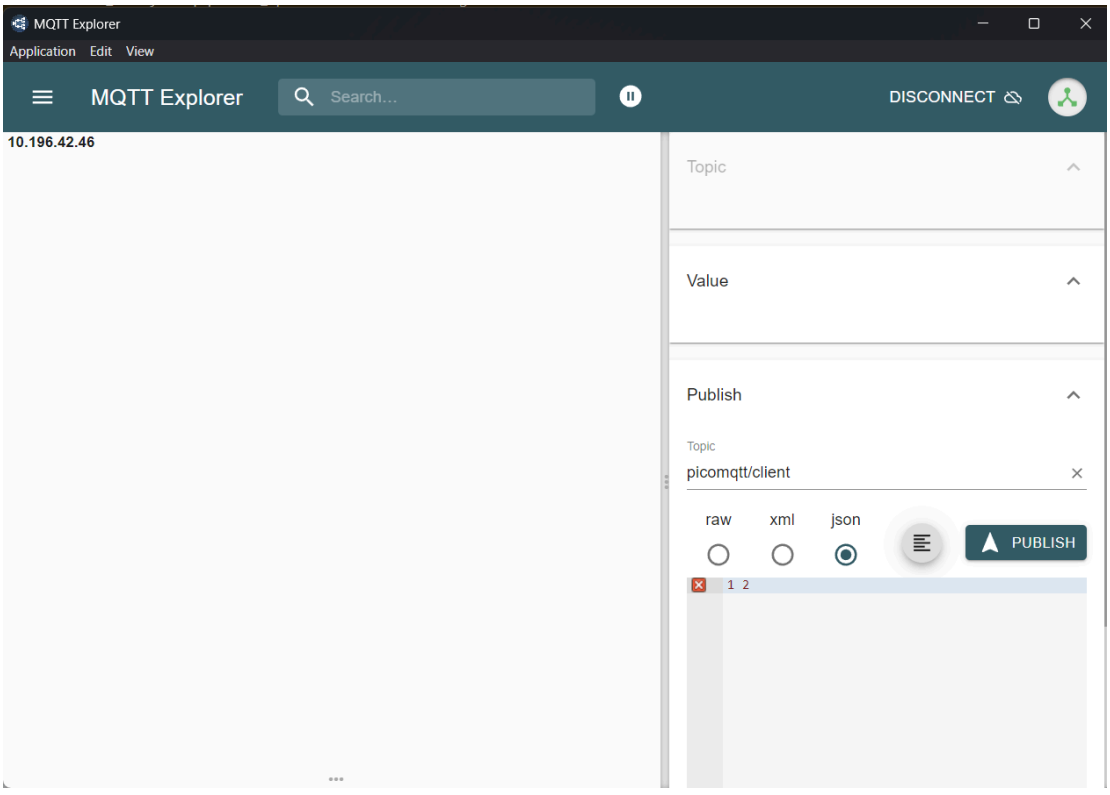


* Now enter the Name =”iitgoa”, Host= IP Address, Username=”FactoryRobot”, Password=”secret”. Then Click on Connect if the rover is already connected to the power supply and WiFi.

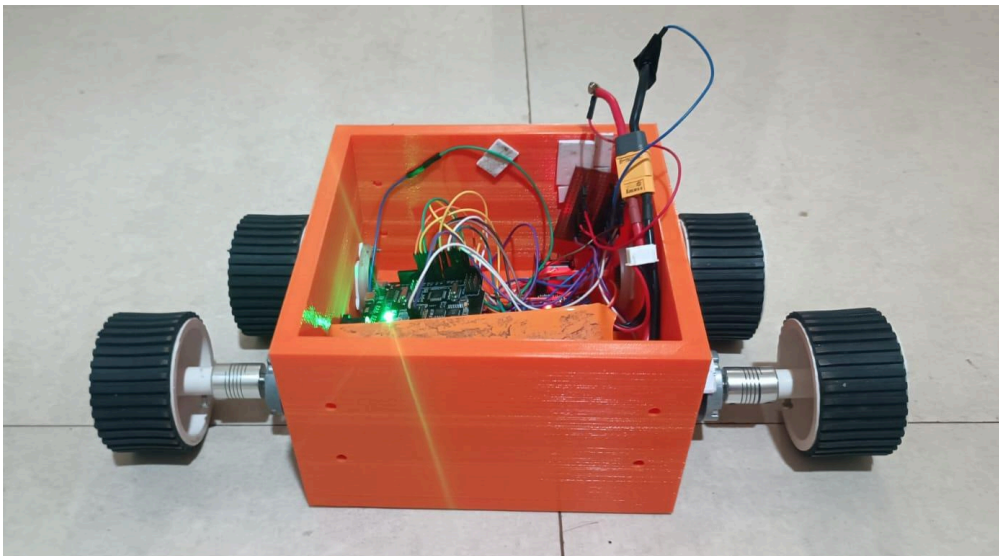
* A new window will open like this.



* Enter the **Topic =”picomqtt/client”** and enter the message to publish for example **“1 2”** then click on **PUBLISH** whenever necessary**.**

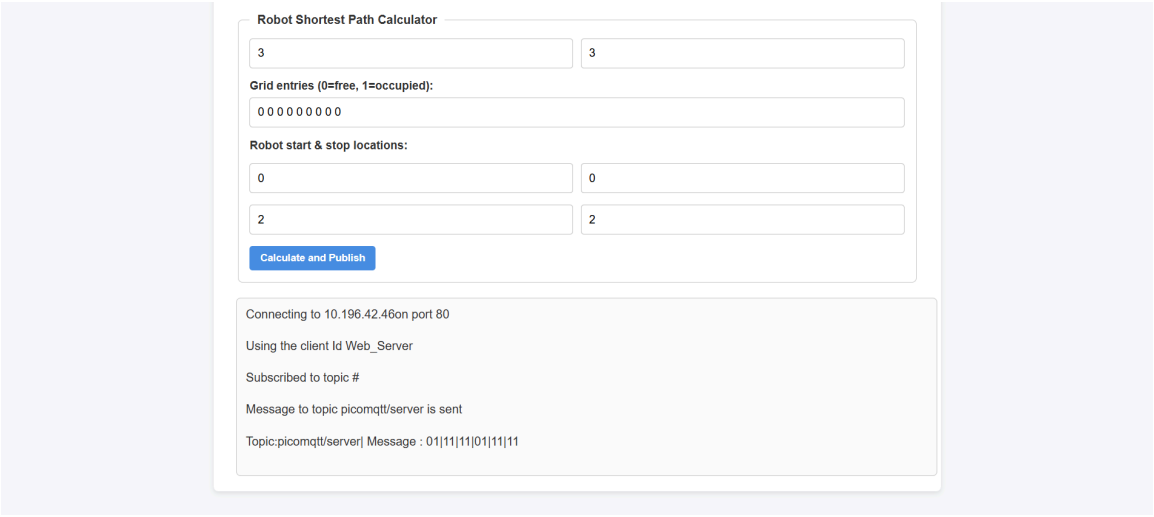


* Now the setup of the system has been completed. Make sure that the physical connections are completed and the rover is connected to the power supply.

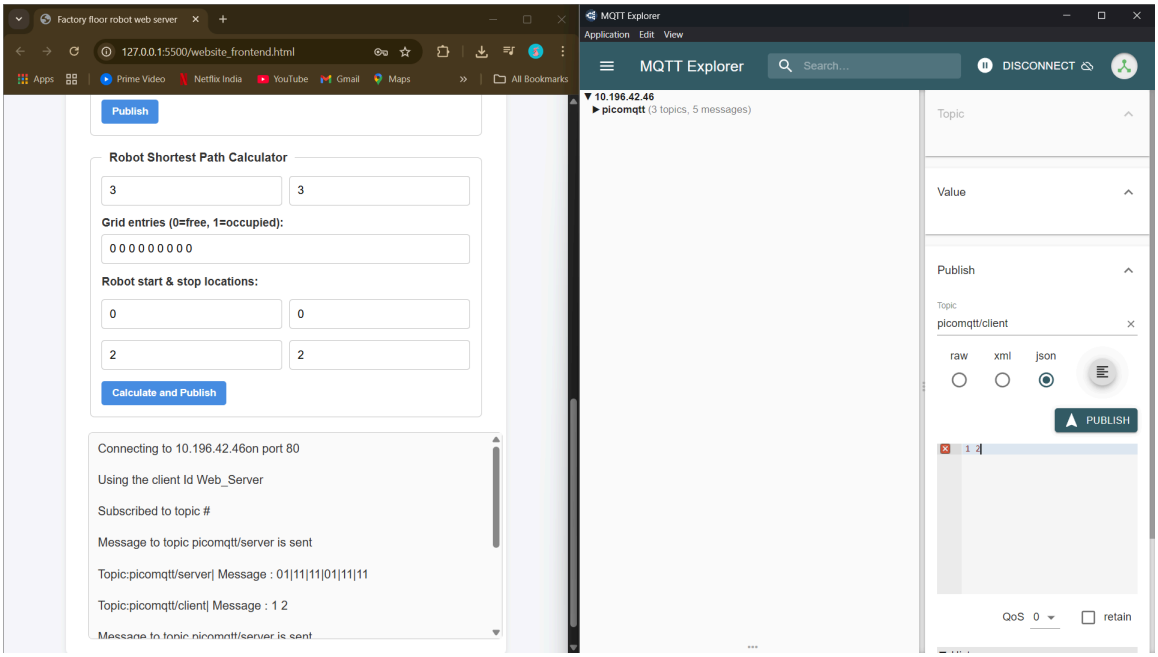


* Now go on the website again and after entering all the instructions click on **Connect** and then after **Calculate and Publish**. Now the rover will start moving and go to the destination as instructed.

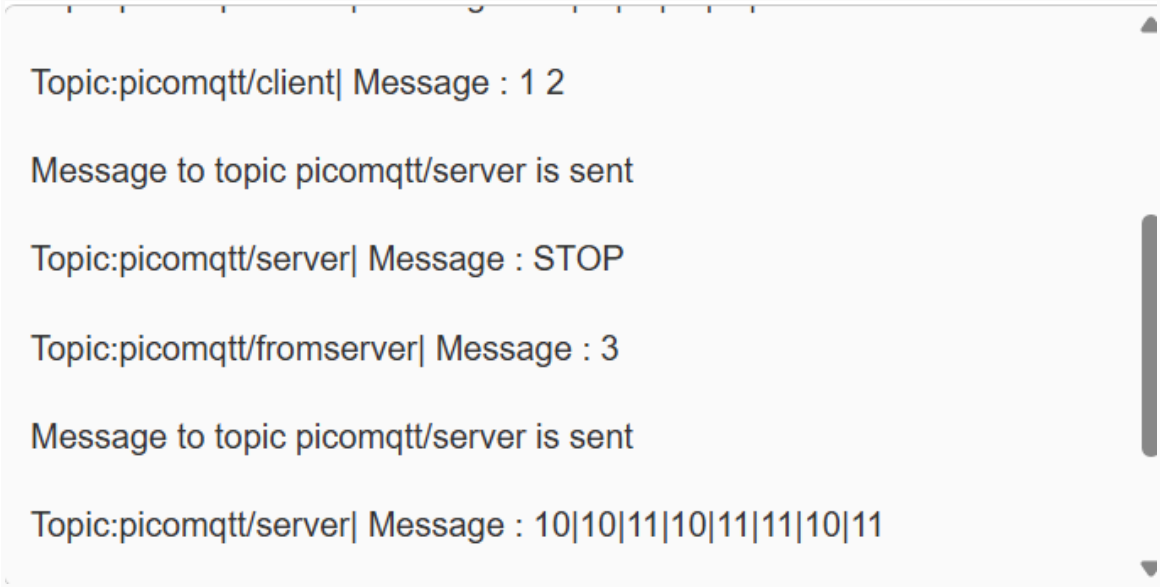
* These instructions can be seen when the rover is activated depending on the details entered.



* Now to take the 2nd Rover into consideration open the MQTT Explorer side by side and enter the necessary details. Connect it and run the rover for any set of instructions and then in between publish the coordinates of the 2nd rover that needs to be ignored while calculating the path.



* We can observe the commands sent from the server to the rover **“01|11|11|01|11|11”,** then message is received from Node MCU to server “1 2” which are the coordinates of the 2nd Rover. Then the server sends **“STOP”** to the Arduino then the Arduino reverts back with the number of commands completed till that is **“3”**. Then the algorithm computes the current location of the rover and then gives the further set of instructions to the rover to go to the specified path **“10|10|11|10|11|11|10|11”.**



* Now the Rover goes to the specified location and the whole system works in this manner.

Future Work

1. Dynamic Obstacle Tracking and Evasion

In the current design, the secondary robot acts as a static obstacle only during re-planning events. To support full-fledged dynamic obstacle avoidance, future versions of the robot system can integrate:

- **Ultrasonic, LiDAR, or Depth Cameras**: These sensors will allow the robot to detect and track humans, other robots, or moving carts in real-time.
- **Kalman Filters or Bayesian Tracking**: For predicting obstacle trajectories and deciding evasive maneuvers based on motion models.
- **Interrupt-Driven Safety Halts**: The robot could be equipped to immediately stop if an object enters a safety radius, then resume after re-validation.

This would allow the robots to operate in more fluid environments such as busy factory floors or shared warehouse lanes.

---

2. Robotic Gripping and Load Handling

While the current robot can follow waypoints and simulate transport, it lacks mechanical load handling. The addition of:

- **Servo-Driven Gripper Arms** or **Electromagnetic Lifters**,
- **Lift Actuators** to raise or lower payloads,
- **Load Balancing Logic** to avoid tipping under weight shifts,

...would transform the prototype from a transport-only bot to a complete pick-and-place system. Integration with conveyor endpoints could also support hybrid assembly lines.

---

3. Multi-Robot Fleet Coordination

The current implementation supports two robots with basic MQTT coordination. Scaling this to a larger fleet requires:

- **Centralized Task Allocation Algorithms**: Like auction-based or contract-net protocols to assign jobs to robots based on proximity, energy, or availability.

- **Decentralized Consensus Algorithms**: For peer-to-peer decision-making in the absence of a centralized scheduler.

- **Collision-Free Multi-Agent Path Planning**: Algorithms such as Cooperative A*, Conflict-Based Search (CBS), or Priority Graphs can ensure safe navigation in shared spaces.

Fleet orchestration software can also be integrated with a Factory Execution System (FES) for real-time order fulfillment.

---

4. Energy-Aware and Autonomous Charging

To ensure round-the-clock operation, the system could be made energy-conscious by:

- **Monitoring Battery State-of-Charge (SoC)** and estimating task feasibility before assignment.

- **Wireless Charging Pads or Docking Stations**: Robots can autonomously return to charge without human intervention.

- **Energy-Aware Scheduling**: Prioritizing short or nearby tasks when battery is low, or balancing task load among the fleet to prevent downtime.

Acknowledgements

We would like to express our sincere gratitude to **Dr. Bidhan Pramanick**, our project guide, for his unwavering support, guidance, and encouragement throughout the duration of this course project. His insights into embedded systems, wireless protocols, and control theory were instrumental in shaping our understanding and execution of the Autonomous Factory Robot Transport System.

We are also thankful to the **Department of Electrical Engineering, IIT Goa**, for providing the necessary infrastructure and resources to carry out our work efficiently. The learning environment and collaborative spirit within the department played a crucial role in our team's progress.

A special thanks to our peers and lab assistants who offered valuable feedback and assistance during testing and debugging sessions. Their contributions helped us identify and resolve practical challenges in real-time motion control and MQTT-based communication.

Finally, we would like to acknowledge the efforts and dedication of each team member — **Shivam Salgaocar**, **Sayan Dhara**, **Shashank Bhushan**, and **Akshat Yadav** — whose consistent collaboration and technical competence made this project a successful and enriching experience.

References

1. LaValle, S. M. *Planning Algorithms*. Cambridge University Press, 2006.

2. MQTT Standard v3.1.1. OASIS, 2014.

3. Phelps, T. *WebAssembly: The Definitive Guide*. O'Reilly Media, 2020.

4. Hart, P. E. et al. "A Formal Basis for Heuristic Pathfinding." *IEEE TSSC*, 1968.

5. M. Silva, "Motor Controller with PID for Differential-Drive Robot," GitHub, 2023.

6. Siciliano, B. et al. *Robotics: Modelling, Planning and Control*. Springer, 2010.

7. S. Lee, "Overview of Autonomous Mobile Robot Path Planning Algorithms," *IJRR*, 43(9), 2024.

8. M. Leśniewski, "PicoMQTT – A Lightweight MQTT Broker/Client for ESP8266/ESP32," GitHub repository, v1.3.0, 2025.

9. Intel. "Edge Control with WebAssembly." Intel Whitepaper, 2021.

10. Arduino Documentation. https://www.arduino.cc/