Interdisciplinary course of

Design and Robotics

XIII° edition, 2025

Project:
Movement and Localization Module

Professors:
Andrea Bonarini, DEIB Department, Politecnico di Milano
Maximiliano Romero, Design department, Politecnico di Milano

Tutors:
Federico Esposito, Samuele Mengarelli, Niccolò Maria Oliva, Rohan Vangal

Group Module: Indoors Movement and Localization

Students:
School of Computer Science and Engineering
Ermelinda Giulivo,
Daniel Mauricio Ruiz Suarez,
Jurij Diego Scandola,

School of Industrial and Information Engineering
Rafael Monllor Ballesteros,

School of Design and Industrial Engineering
Abdul Moiz

INDEX

Abstract

This project presents the design, development, and implementation of the movement module for a mobile social robot intended to manage access to microwaves in a university environment. The goal was to create a compact, autonomous platform capable of omnidirectional navigation, basic obstacle detection, and integration with future functional modules.

The mechanical structure consists of cut wooden platforms joined by vertical columns, housing all electronic and mechanical components. Movement is enabled by three DC motors with omnidirectional wheels, arranged in a triangular configuration, and controlled by motor drivers. A microcontroller governs the system, handling PWM motor control and ultrasonic sensor input, powered by a regulated LiPo battery setup.

The development process focused on early testing of individual components through a modular setup, followed by full system integration on the final chassis. The resulting prototype delivers a robust and adaptable movement platform, capable of supporting higher-level behavior and other modules. It provides a solid foundation for building a functional and socially engaging service robot.

Phase 1: Discover

During the discovery phase, the team organized its structure, established roles for management, conducted research and developed initial proposals for the module.

Team Organization

Our team is composed of four students:

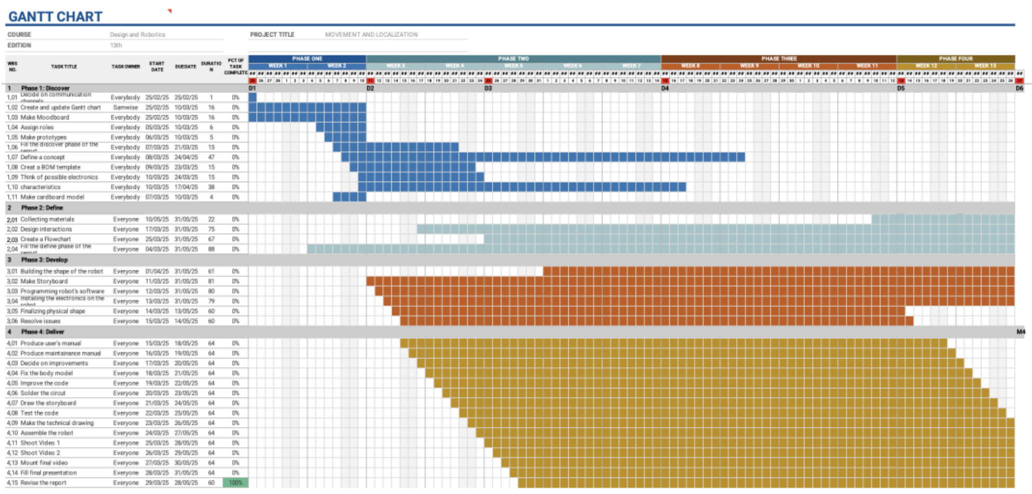
- **Ermelinda Giulivo**
- **Rafael Monllor Ballesteros**
- **Daniel Mauricio Ruiz Suarez**
- **Jurij Diego Scandola**
- **Abdul Moiz**

The team leader and **Backbone member** is Jurij Diego Scandola.
Our responsibilities and roles are:

- **Rapporteur**: Rafael Monllor Ballesteros
  He is responsible to have reports and deliverables ready for the due date and to interface with tutors to provide deliverables.
- **Schedule manager**: Jurij Diego Scandola
  He is responsible to get agreement of the team on the schedule, monitor the (eventually internal) deadlines, plan what needed to match them.
- **Art director**: Daniel Mauricio Ruiz Suarez
  He is responsible for the aesthetical and interaction aspects of the final product.
- **Tech manager**: Ermelinda Giulivo
  She is responsible for the technical aspects of the final product, functionalities and maintenance.
- **Designer**: Abdul Moiz
  He is responsible for the model prototyping and documentation of the Robot

Project Management

Our team plans to solve the tasks according to the following GANTT:



COURSE	Design and Robotics
EDITION	13th

WBS NO.	TASK TITLE	TASK OWNER	START DATE	DUEDATE	DURATION	PCT OF TASK COMPLETE
1	**Phase 1: Discover**					
1,01	Decide on communication channels	Everybody	25/02/25	25/02/25	1	0%
1,02	Create and update Gantt chart	Samwise	25/02/25	10/03/25	16	0%
1,03	Make Moodboard	Everybody	25/02/25	10/03/25	16	0%
1,04	Assign roles	Everybody	05/03/25	10/03/25	6	0%
1,05	Make prototypes	Everybody	06/03/25	10/03/25	5	0%
1,06	Fill the discover phase of the report	Everybody	07/03/25	21/03/25	15	0%
1,07	Define a concept	Everybody	08/03/25	24/04/25	47	0%
1,08	Creat a BOM template	Everybody	09/03/25	23/03/25	15	0%
1,09	Think of possible electronics	Everybody	10/03/25	24/03/25	15	0%
1,10	characteristics	Everybody	10/03/25	17/04/25	38	0%
1,11	Make cardboard model	Everybody	07/03/25	10/03/25	4	0%
2	**Phase 2: Define**					
2,01	Collecting materials	Everyone	10/05/25	31/05/25	22	0%
2,02	Design interactions	Everyone	17/03/25	31/05/25	75	0%
2,03	Create a Flowchart	Everyone	25/03/25	31/05/25	67	0%
2,04	Fill the define phase of the report	Everyone	04/03/25	31/05/25	88	0%
3	**Phase 3: Develop**					
3,01	Building the shape of the robot	Everyone	01/04/25	31/05/25	61	0%
3,02	Make Storyboard	Everyone	11/03/25	31/05/25	81	0%
3,03	Programming robot's software	Everyone	12/03/25	31/05/25	80	0%
3,04	Installing the electronics on the robot	Everyone	13/03/25	31/05/25	79	0%
3,05	Finalizing physical shape	Everyone	14/03/25	13/05/25	60	0%
3,06	Resolve issues	Everyone	15/03/25	14/05/25	60	0%
4	**Phase 4: Deliver**					
4,01	Produce user's manual	Everyone	15/03/25	18/05/25	64	0%
4,02	Produce maintainance manual	Everyone	16/03/25	19/05/25	64	0%
4,03	Decide on improvements	Everyone	17/03/25	20/05/25	64	0%
4,04	Fix the body model	Everyone	18/03/25	21/05/25	64	0%
4,05	Improve the code	Everyone	19/03/25	22/05/25	64	0%
4,06	Solder the circut	Everyone	20/03/25	23/05/25	64	0%
4,07	Draw the storyboard	Everyone	21/03/25	24/05/25	64	0%
4,08	Test the code	Everyone	22/03/25	25/05/25	64	0%
4,09	Make the technical drawing	Everyone	23/03/25	26/05/25	64	0%
4,10	Assemble the robot	Everyone	24/03/25	27/05/25	64	0%
4,11	Shoot Video 1	Everyone	25/03/25	28/05/25	64	0%
4,12	Shoot Video 2	Everyone	26/03/25	29/05/25	64	0%
4,13	Mount final video	Everyone	27/03/25	30/05/25	64	0%
4,14	Fill final presentation	Everyone	28/03/25	31/05/25	64	0%
4,15	Revise the report	Everyone	29/03/25	28/05/25	60	100%

Research

The research phase is a critical foundation for the development of the movement and localization module of the indoor robot. At this stage, the primary objective is to explore and analyse existing technologies, methodologies and systems that enable precise navigation and positioning within indoor environments. Unlike outdoor settings, indoor environments present unique challenges such as signal attenuation, dynamic obstacles and limited access to GPS, making robust and reliable localization and movement a complex task.

Omnidirectional Movement Approach

For the movement and localization module of the indoor robot, an **omnidirectional movement** approach has been selected. This strategy enables the robot to move seamlessly in any direction without the need to rotate its chassis first. Typically implemented using omni-wheels or meconium wheels, omnidirectional systems are particularly advantageous in constrained and dynamic indoor environments. However, this choice also comes with certain trade-offs.

Advantages	**Limitations**
Enhanced Manoeuvrability: Omnidirectional movement allows for instant lateral, diagonal or rotational motion. This is highly beneficial in tight spaces or crowded indoor environments where flexibility is critical.	**Mechanical Complexity:** Omni-wheels and their assemblies are more mechanically intricate than standard wheels, potentially leading to higher manufacturing costs and increased maintenance needs.
Simplified Path Planning: Since the robot can move in any direction at any time, path planning algorithms can be more straightforward compared to traditional differential drive systems, reducing complexity in navigation.	**Lower Traction and Load Capacity:** Due to the design of omni-wheels (which often rely on small rollers), they typically offer less traction and may struggle on uneven surfaces, affecting stability and limiting the robot's carrying capacity.
Improved Positioning Accuracy: Fine adjustments to the robot's position and orientation are easier, which is crucial for tasks that demand high precision, such as docking, object manipulation or alignment tasks.	**Energy Efficiency:** Omnidirectional systems can be less energy-efficient, especially during complex movement patterns, leading to increased power consumption.
Smooth Obstacle Avoidance: The ability to sidestep obstacles without complex turning manoeuvres leads to smoother and often faster responses in dynamic environments.	**Control Challenges:** Maintaining accurate and stable movement requires more sophisticated control algorithms. Wheel slip and small errors in motion can accumulate, potentially affecting localization accuracy if not properly managed.

Defining the Number of Wheels: Three-Wheel vs. Four-Wheel Omnidirectional Configurations

After selecting an omnidirectional movement strategy, the next major design challenge was to determine the optimal number of wheels for the robot. Specifically, we needed to choose between a **three-wheel** and a **four-wheel** omnidirectional configuration. This decision plays a critical role in the robot's stability, manoeuvrability, mechanical complexity and overall performance.

Three-Wheel Omnidirectional Configuration	
Advantages	**Disadvantages**
Simplicity: A three-wheel design is mechanically simpler and often lighter, which can lead to easier construction and reduced cost.	**Reduced Stability:** With only three points of contact, the robot can be less stable, especially when carrying loads or encountering uneven flooring.
Compact Design: It typically requires less space, making it ideal for very small robots or environments where minimal footprint is essential.	**Weight Distribution Challenges:** Balancing the robot's weight evenly across three wheels can be more difficult, possibly leading to inconsistent traction.
Sufficient Mobility: With proper placement (usually forming an equilateral triangle), three wheels can achieve full omnidirectional movement.	**Limited Redundancy:** If one wheel fails or underperforms, the overall system may be more severely affected than in four-wheel configurations.

Four-Wheel Omnidirectional Configuration	
Advantages	**Disadvantages**
Greater Stability: Four points of contact provide better balance and support, especially important for larger robots or those carrying payloads.	**Higher Mechanical Complexity:** More wheels mean a more complicated chassis design, additional motor control requirements and potential maintenance issues.
Improved Traction: Load distribution across four wheels can enhance grip and reduce the risk of wheel slip.	**Larger Footprint:** A four-wheel setup generally results in a bigger base, which could limit manoeuvrability in extremely tight spaces.
Increased Redundancy: The robot can better tolerate slight malfunctions or uneven performance from one wheel without drastic impact.	

Defining the Robot’s Shape: Square, Circular, Polygonal or Triangular Designs

Following the decisions on movement type and wheel configuration, another crucial design consideration was the **shape** of the robot's base. The geometry of the robot directly affects not only its aesthetic but also its **manoeuvrability**, **stability**, **sensor placement** and **ability to navigate tight indoor spaces**. The primary shapes considered were **square**, **circular**, **polygonal** and **triangular** forms, each offering distinct advantages and challenges.

Square Shape	
Advantages	**Disadvantages**
Symmetry: Provides even weight distribution and simplifies sensor and component layout.	**Corner Interference:** Corners can get caught on obstacles or furniture when turning or navigating tight areas.
Ease of Design: Straightforward for mechanical construction and planning wheel placement, especially for four-wheel configurations.	**Reduced Smoothness:** Compared to circular designs, square robots require more careful navigation to avoid snagging.

Circular Shape	
Advantages	**Disadvantages**
Excellent Manoeuvrability: A circular design has no corners, allowing smooth, continuous rotation and movement in all directions without risk of snagging.	**Space Efficiency:** Fitting square or rectangular components inside a circular chassis can lead to wasted internal space.
Ideal for Dynamic Environments: Particularly suited for environments with narrow passages or crowded spaces.	**Construction Complexity:** Slightly more complex to design and fabricate compared to square frames.

Polygonal Shape	
Advantages	**Disadvantages**
Balance Between Round and Angular: A polygonal base provides some benefits of a circular shape (smooth movement) while still offering flat surfaces for easier component mounting.	**Design Complexity:** More complicated to design and assemble compared to pure circular or square robots.
Unique Structural Advantages: Certain polygons can provide better rigidity or accommodate specific sensor layouts.	

Triangular Shape	
Advantages	**Disadvantages**
Compact and Lightweight: Very compact, ideal for small robots using three wheels.	**Limited Stability:** Fewer points of contact with the ground can compromise stability, especially on uneven surfaces.
Simple Three-Wheel Integration: Matches naturally with three-wheel omnidirectional systems.	**Challenging Payload Distribution:** May cause difficulties in evenly distributing components and loads.

Selecting the Sensing Technology: LiDAR, Ultrasonic, Camera and Other Options

An essential step in developing the movement and localization module is selecting the **appropriate sensing technology**. The choice of sensors directly impacts the robot’s ability to perceive its environment, perform accurate localization, avoid obstacles and navigate efficiently indoors. Different types of sensors offer different strengths and weaknesses depending on the operating environment, required precision, cost and computational complexity.

LiDAR (Light Detection and Ranging)	
Advantages	**Disadvantages**
High Precision Mapping: LiDAR provides highly accurate distance measurements and is excellent for creating detailed maps (2D or 3D).	**High Cost:** Quality LiDAR sensors are relatively expensive.
Good Range: Effective at sensing over longer distances compared to cameras or ultrasonic sensors.	**Computational Load:** Processing point clouds requires significant computational resources.
Strong Performance in Low Light: Works reliably regardless of lighting conditions.	

Ultrasonic Sensors	
Advantages	**Disadvantages**
Low Cost: Affordable and easy to integrate.	**Limited Precision:** Not suitable for detailed mapping or fine localization.
Simplicity: Straightforward for basic obstacle detection at short ranges.	**Susceptibility to Noise:** Environmental factors like soft surfaces can absorb sound waves, reducing effectiveness.
Lightweight: Small and energy efficient.	

Cameras (RGB or Depth Cameras)	
Advantages	**Disadvantages**
Rich Environmental Information: Cameras can provide detailed images for advanced perception tasks like object recognition and visual SLAM.	**Lighting Dependence:** Performance can drop significantly in poor lighting conditions.
Cost-Effective: Compared to LiDAR, cameras are generally cheaper.	**Complex Processing:** Vision-based algorithms are computationally intensive and require robust processing pipelines.
Versatility: Depth cameras (like Intel RealSense) combine some of the benefits of LiDAR and standard imaging.	

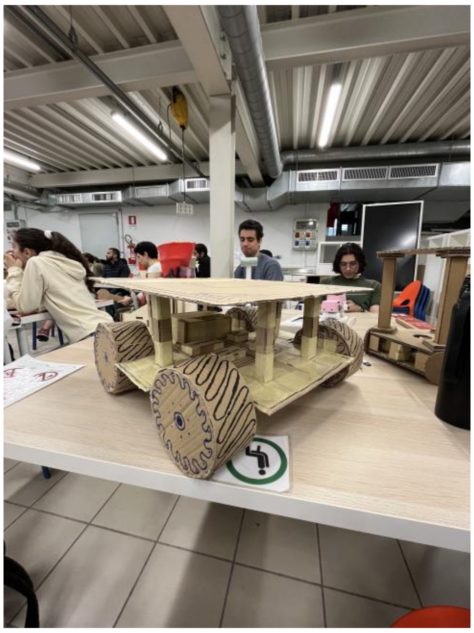
Other Options

- **IMU (Inertial Measurement Unit):**
    - **Provides orientation and movement data** to complement other localization methods.
    - **Sensitive to Drift:** Long-term accuracy can degrade without correction from external sensors.
- **Infrared Sensors:**
    - **Good for simple proximity detection** but limited in range and precision.
- **Magnetic Sensors:**
    - **Useful for specific localization systems** using magnetic markers but not practical for general indoor navigation.

Group cardboard prototype proposal

When tasked with creating a cardboard prototype of the module, the group presented four different design proposals.

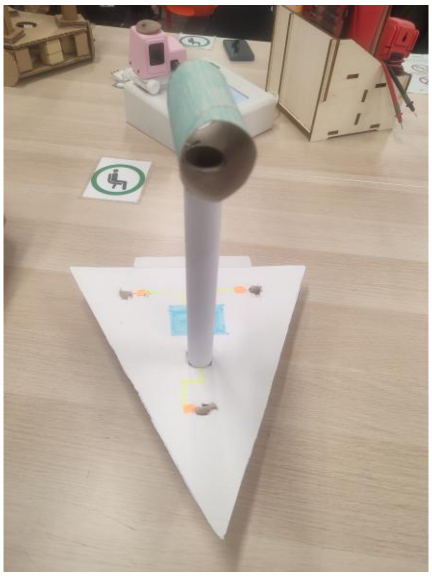
Omnidirectional robot with four wheels positioned at the vertices of a square
Ermelinda Giulivo



Omnidirectional robot with four wheels positioned at the vertices of a square	
Pros	**Cons**
Excellent Stability: The square layout provides a wide and balanced support base, enhancing stability, especially when carrying payloads or navigating uneven indoor floors.	**Larger Turning Radius Compared to Circular Robots:** Although it can strafe and rotate, the square footprint is bulkier when fitting into very tight or irregularly shaped spaces.
Full Omnidirectional Mobility: With wheels positioned symmetrically, the robot can move smoothly in any direction —	**Wheel Synchronization Complexity:** Precise control of all four wheels is essential. Small discrepancies in motor

forward, backward, sideways or diagonally — without needing to rotate first.	performance can cause drift or errors in movement if not properly calibrated.
Simple Mechanical Symmetry: The square design makes mechanical construction easier and ensures even distribution of forces, reducing stress on the frame.	**Higher Energy Consumption:** Coordinated omnidirectional movement (especially diagonal motion) can be less energy-efficient compared to simpler drive systems.
Predictable and Balanced Control: Because of the symmetry, the control algorithms (e.g., for velocity distribution across wheels) can be simpler and more consistent.	**Cost and Mechanical Complexity:** Four omni-wheels and corresponding motors, encoders and controllers increase system cost and complexity compared to simpler two-wheel or three-wheel robots.
Good for Sensor Placement: The square chassis offers clear and logical positions for sensors, allowing for easy 360-degree coverage with minimal blind spots.	**Vulnerability to Wheel Slippage:** Omni-wheels rely on small rollers, which can slip on smooth or dusty indoor surfaces, potentially affecting localization accuracy.

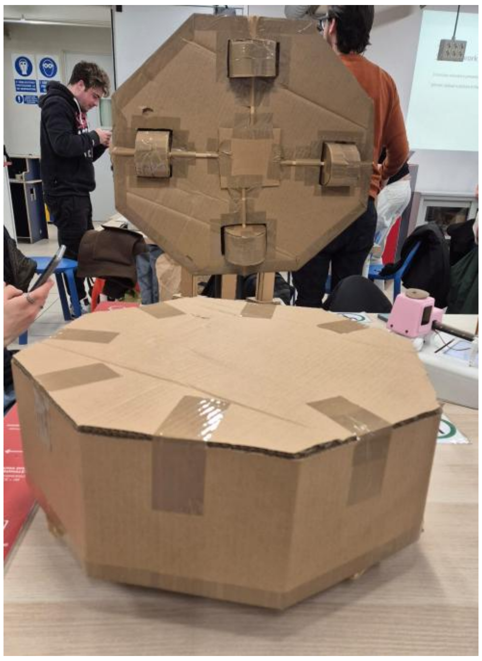
Triangular-shaped, three-wheel omnidirectional robot
Rafael Monllor Ballesteros



Triangular-shaped, three-wheel omnidirectional robot	
Pros	**Cons**
Compact and Lightweight Design: The triangular layout results in a very compact structure, making the robot ideal for navigating narrow spaces or operating in environments where size and weight are critical.	**Reduced Stability:** With only three points of ground contact, the robot is more prone to tipping, especially if it carries uneven loads or accelerates/decelerates quickly.
Mechanical Simplicity: With only three wheels and motors, the mechanical structure and wiring are simpler compared to four-wheel designs, reducing build time and potential failure points.	**Challenging Weight Distribution:** Careful design is required to ensure the centre of mass is perfectly balanced within the triangle; otherwise, one wheel might

	bear too much load, leading to uneven wear or movement issues.
Cost Efficiency: Fewer wheels, motors and controllers result in a lower overall cost, making it a more economical option for many projects.	**Limited Load Capacity:** Because of the lighter structure and fewer contact points, three-wheel designs are generally not suitable for heavy-duty applications.
Full Omnidirectional Mobility: Despite having only three wheels, the robot still achieves full 360-degree movement capabilities when wheels are properly placed at 120° intervals.	**Control Complexity for Stability:** Dynamic movements (especially rotations and fast translations) can introduce more vibration or wobble compared to a four-wheel robot.
Efficient for Small Robots: Ideal for light-duty applications where high payload capacity is not necessary.	**Less Redundancy:** If one motor or wheel fails, the robot's ability to move properly is much more compromised than with four-wheel designs.

Polygonal-shaped omnidirectional robot with four wheels arranged in a cross configuration
Daniel Mauricio Ruiz Suarez

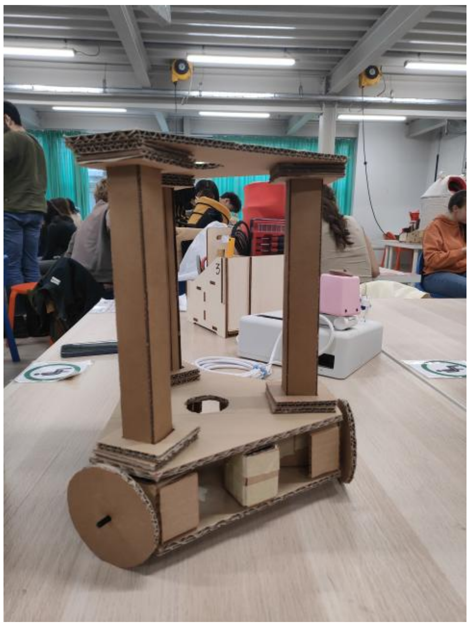


Polygonal-shaped omnidirectional robot with four wheels arranged in a cross configuration	
Pros	**Cons**
Excellent Centralized Stability: Placing the wheels in a cross (or X) pattern centred around the robot's body provides good balance and helps maintain stability during omnidirectional movement.	**Mechanical Complexity:** Designing a strong and lightweight polygonal chassis that properly supports a cross-wheel configuration can be challenging compared to square or triangular designs.

Efficient 360° Movement: The cross configuration offers excellent manoeuvrability, allowing smooth lateral, diagonal and rotational movement without complicated reorientation.	**Vulnerability to Impacts:** With wheels projecting further out from the centre in the cross arrangement, they may be more exposed to potential collisions or impacts with obstacles.
Optimized Use of Space: The polygonal chassis (such as hexagonal or octagonal) better fits the cross pattern while allowing more efficient internal layout of components compared to a purely circular design.	**Load Distribution Sensitivity:** Uneven weight distribution can negatively affect movement performance, as each wheel may bear different loads during turns or fast translations.
Simplified Path Planning: Because the wheels are symmetrically arranged, control algorithms for movement and localization become more predictable and manageable.	**Potential for Increased Size:** Depending on the polygon shape and wheel placement, the overall footprint of the robot could become larger than necessary for tight indoor environments.
Better Obstacle Navigation: The extended wheel positions can help in negotiating tight spaces or approaching obstacles at different angles more smoothly.	**Higher Control Precision Needed:** Maintaining synchronized wheel movement across a cross configuration demands precise motor control to avoid unwanted drift or vibration, especially at higher speeds.

Circular-shaped, three-wheel omnidirectional robot
Jurij Diego Scandola

Note: The circular base was not included in the prototype due to insufficient cardboard material.



Circular-shaped, three-wheel omnidirectional robot	
Pros	**Cons**

Excellent Manoeuvrability: The circular design eliminates corners, allowing smooth and uninterrupted movement and rotation in any direction — ideal for navigating tight, cluttered indoor environments.	**Reduced Structural Simplicity:** Building a strong, circular chassis (especially with flat materials like cardboard or sheet metal) can be mechanically more complex compared to square or polygonal shapes.
Compact and Symmetrical: The circular form naturally distributes mass and components around the centre, enhancing balance and improving dynamic stability during motion.	**Lower Load Capacity:** Three-wheel setups naturally provide less stability and support for heavy loads compared to four-wheel configurations and the circular frame can limit internal mounting options for large or heavy components.
Efficient for Omnidirectional Control: The 120° placement of the three wheels around the circle simplifies omnidirectional movement algorithms and ensures consistent movement performance.	**Challenging Internal Layout:** Fitting rectangular or square components (like batteries, boards, and sensors) inside a circular space can be inefficient and may lead to wasted internal volume.
Minimal Risk of Snagging: Without edges or corners, the robot can more easily avoid getting caught on obstacles, furniture, or tight doorways.	**Sensitivity to Weight Imbalance:** Proper balancing is critical; even slight asymmetry in weight distribution can significantly impact the robot's movement precision and stability.
Aesthetically Appealing: The circular shape often results in a cleaner, more modern appearance, which can be a factor in user-facing or commercial applications.	**Less Redundancy:** With only three wheels, if one wheel or motor fails, the robot's mobility is seriously compromised compared to a four-wheel design.

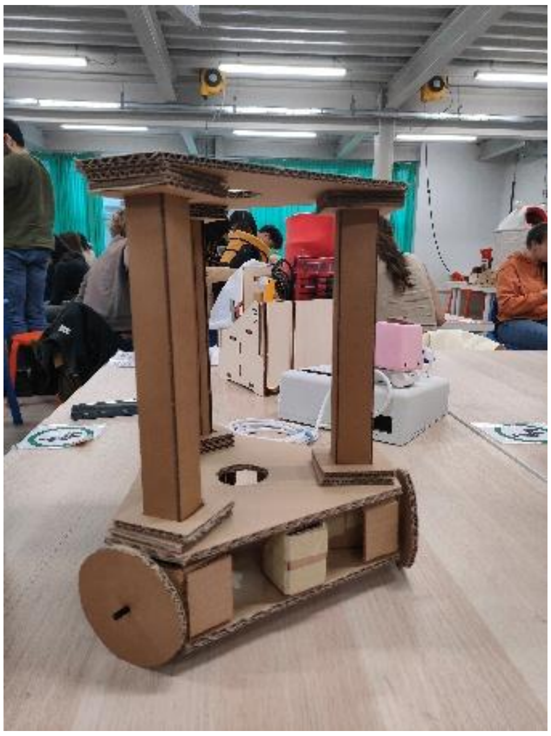
Phase 2: Define

After conducting broad research on existing social robots, user needs, and contextual use within shared spaces like university buildings, we moved on to define more specific goals for our own robot. Our challenge was to design a friendly, autonomous assistant capable of organizing turns for a shared microwave, while maintaining a mobile and socially engaging presence.

In this phase, we started shaping our concept through rough prototypes to test:

- **Basic movement mechanisms** using omnidirectional wheels.
- **Aesthetic identity**, considering a space between two separate bases for hiding electronics.
- **Key functionalities** for getting proper movement of the robot.

The cardboard prototype that best adjusted to these ideas can be seen in the following picture.



These early tests helped us set a clearer direction for both technical and conceptual development, bridging the gap between ideas and implementation.

Strategy

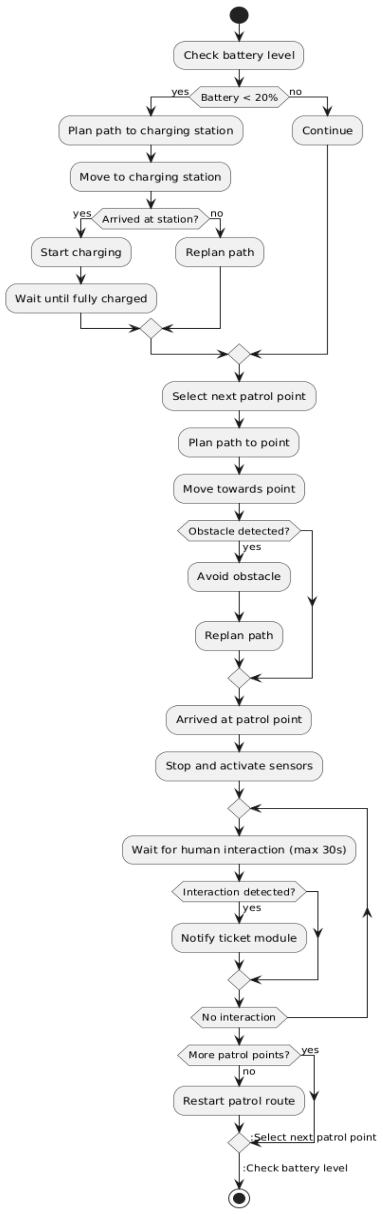
Our strategy during this phase focused on transforming abstract ideas into concrete design and implementation plans for the movement module of the robot. We followed a structured approach to ensure consistency across form, function, and technology.

1. **Defining functionalities**
   We began by clearly outlining the movement module's responsibilities: patrolling the space, pausing at predefined locations to allow human interaction, and navigating autonomously to the charging station when battery levels are low. These core behaviors served as a foundation for every subsequent design decision.
2. **Considering the physical form**
   We aimed to create a friendly and humorous appearance aligned with the social nature of the robot. The form of a cooking pot was chosen for its simplicity, roundness, and instantly recognizable shape, which also fits thematically with the microwave queue scenario.
3. **Designing the 3D model of the movement module**
   Using 3D modeling tools, we created a rough prototype of the movement module's structure. This model focused on the chassis and physical arrangement of elements such as omnidirectional wheels, base support, and electronic housing. It helped us evaluate size constraints, stability, and potential mounting points for components.

4. **Selecting electronic components**
   We identified the necessary electronic components, including three omnidirectional wheels, motor drivers, a microcontroller (ESP32), sensors for obstacle detection and battery monitoring, and components for the ticket dispensing system. We also defined the logical wiring and signal flow between them.
5. **Beginning software development**
   In parallel, we started building the logic behind the movement module in code. This included route planning, obstacle avoidance, pausing logic, and low battery detection routines.

Functionalities

The movement module was designed to enable the robot to perform three key behaviors: patrol autonomously through a mapped environment, stop periodically to allow interaction with users, and navigate to a charging station when the battery level is low. These main functionalities can be seen in the following flow chart:



Electronics

The design of the electronics during the development phase was guided by the functional needs of the movement module. At this stage, we focused on defining a

system that would enable **precise motor control**, allow for **basic sensing capabilities**, and remain **modular and scalable** for future upgrades.
Given that the robot needed to move holonomicaly using **three omnidirectional wheels**, it was essential to control **three independent DC motors** with precision. This required a microcontroller with multiple PWM-capable digital outputs and a structure that allowed us to drive each motor bidirectionally. We selected the **ATmega328P** microcontroller, due to its compatibility with the Arduino platform, which offered a familiar programming environment, low-level control, and access to a wide library of tested code.

To drive the motors, we needed reliable motor drivers that could handle the current demand and support direction and speed control. We opted for **DRV8871 single-channel H-bridge drivers**, one for each motor. These components provided sufficient current capacity, protection features, and a straightforward interface with the **ATmega328P**, using two digital pins per driver.

In addition to movement, the robot needed a basic level of **obstacle detection** to eventually navigate or stop when required. For this, we planned the integration of **ultrasonic sensors (HC-SR04)** around the perimeter. These sensors are simple, affordable, and widely used in robotics, but they require careful timing to avoid interference between readings. The ATmega328P offered enough GPIOs to connect to six sensors, with logic implemented to trigger them sequentially.

All components would be powered by a **LiPo battery**, but since the system needed a stable 5V supply for both logic-level components and sensor operation, we included a **step-down voltage regulator (LM2596)** in the plan. This ensured consistent voltage despite the battery's natural variation underload.

From a wiring and assembly perspective, we anticipated the need to route motor cables and sensor wires efficiently between the top and bottom of the chassis. This led us to consider a vertical layout, with the **motor drivers placed near the motors** (on the bottom of the base), and the **microcontroller and logic-level components mounted on top**, minimizing wire length and simplifying troubleshooting.

In short, the electronics defined in this phase were selected not just for their individual performance, but for their ability to **integrate seamlessly** within a compact, layered robot structure, and support **clear, testable behaviors** during the next implementation stages.

Coding

The code developed so far focuses on the core functionalities of the movement module: patrolling the environment, stopping periodically for interaction, and returning to a charging station when battery levels are low. The code is being developed in C++, using the Arduino framework, since it offers:

- High compatibility with our chosen microcontroller (ATmega328P).
- Access to well-documented libraries for motor control, timers, and sensors.
- A simple structure that facilitates rapid prototyping and debugging.

The microcontroller is programmed directly via the Arduino IDE, allowing us to flash and test code iterations easily. The robot can switch between various behaviors such as:

- Patrolling predefined points.
- Stopping and waiting for user interaction.
- Navigating to the charging station when the battery is low.

In this phase, we focused on:

- Defining key states and transitions.
- Mapping out which pins control each motor and sensor.
- Testing the motors and sensors.

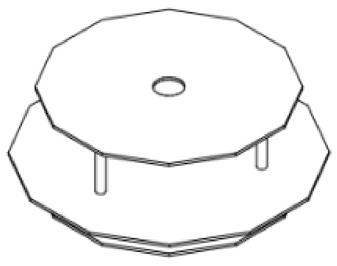
The code used for the testing of the HC-SR04 ultrasonic sensors, and the motors can be seen in the Appendix.

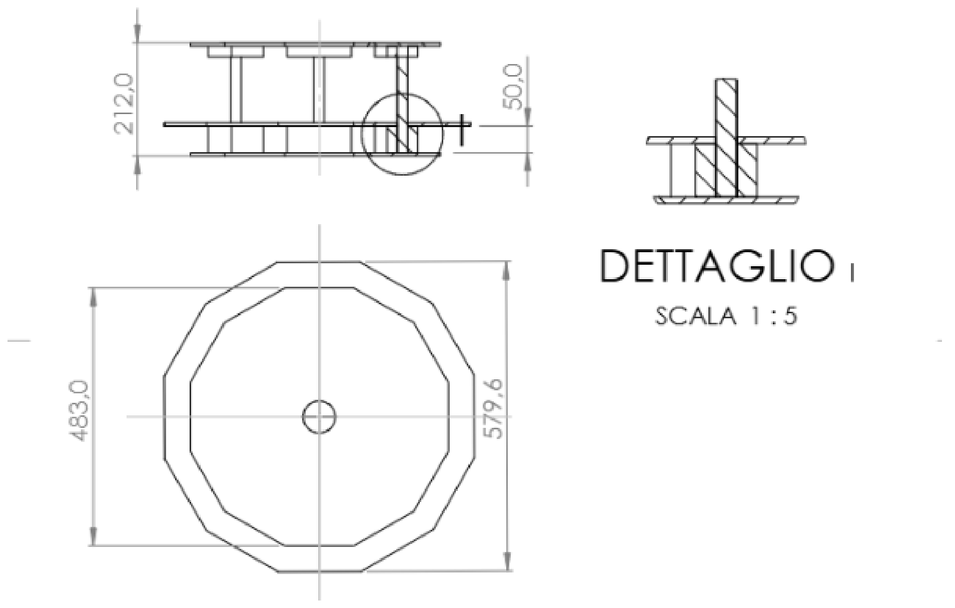
Structure

The physical structure of the movement module was designed to be simple, stable, and compatible with the circular “cooking pot” appearance of the final robot. Its geometry allows for efficient assembly and proper distribution of electronic components, motors, and wheels.
The chassis consists of two main dodecagonal plates:

- A **lower base plate** that holds the motors, wheels, the battery, and electronics.
- An **upper plate** connected by four vertical rods, providing structural support and space for mounting future upper modules (such as the ticket dispenser).

Between the two plates, there's ample vertical space to safely house the internal wiring and maintain separation between moving and static parts. The structural elements were dimensioned using CAD software and detailed in the following technical drawings.

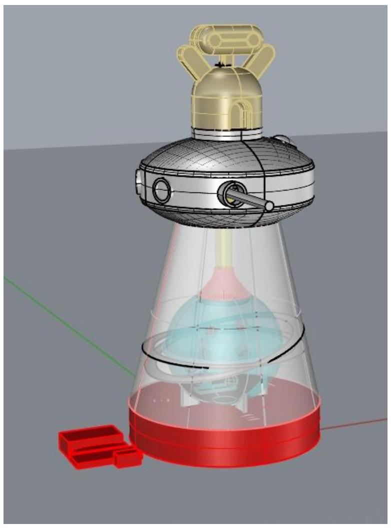




The structure is primarily symmetrical, promoting balance and helping the robot move stably in any direction. This design prioritizes modularity, allowing individual parts to be adjusted, replaced, or expanded without affecting the rest of the system.

Shape

The shape of the robot was designed to convey friendliness, uniqueness, and a sense of purpose within its social environment. Taking inspiration from kitchen elements, the robot adopts a stylized form reminiscent of a cooking pot, reinforcing its role as a ticket dispenser for a microwave queue.
The overall geometry is conical, with a wide dodecagonal base that houses the movement module and narrows as it rises, creating a silhouette that is both functional and expressive. The lower section (in red) is where the wheels and internal electronics are stored. The transparency of the conical section allows for visual access to internal components and lighting, adding both a technological and playful aesthetic.
At the top of the robot, the head is formed by a metallic-looking circular cap with a symmetrical design, integrating visual sensors, lights, or interface points. A pair of stylized “ears” or “handles” give it a characterful appearance, reinforcing its role as a social entity that invites interaction. The shape also leaves space at the front for a clearly visible ticket dispensing area.



Concept

The concept behind our robot is to create a social and functional assistant that organizes the use of a shared microwave in a university environment.
By dispensing turn tickets and patrolling the space autonomously, the robot aims to bring order to an otherwise chaotic or informal process, while adding an element of humor and engagement through its unique design.
Inspired by everyday kitchen objects, its cooking pot-like appearance makes the robot immediately relatable and non-intimidating, encouraging students to approach it easily. Its ability to move, pause, and react transforms a mundane waiting experience into a playful and organized interaction, promoting better coexistence in shared spaces.

The combination of functionality and personality defines the essence of this project:
a robot that not only helps manage tasks but also becomes a memorable part of its environment.

Phase 3: Develop

In this phase we describe the development process, departing from the first prototype to the final improvements regarding:

Strategy

Our development strategy followed a bottom-up approach, starting with the physical structure and gradually integrating electronics and software. This allowed us to test each stage independently and ensure that all subsystems could function reliably before full integration.

The first step was to assemble the **structure** of the movement module in Bovisa's "*Prototipi* lab." Using wood as the base material, we cut and mounted the two dodecagonal plates, positioning the vertical supports and verifying spacing and alignment based on our technical drawings. This gave us a solid physical base to work with and validated the dimensions from our CAD model.

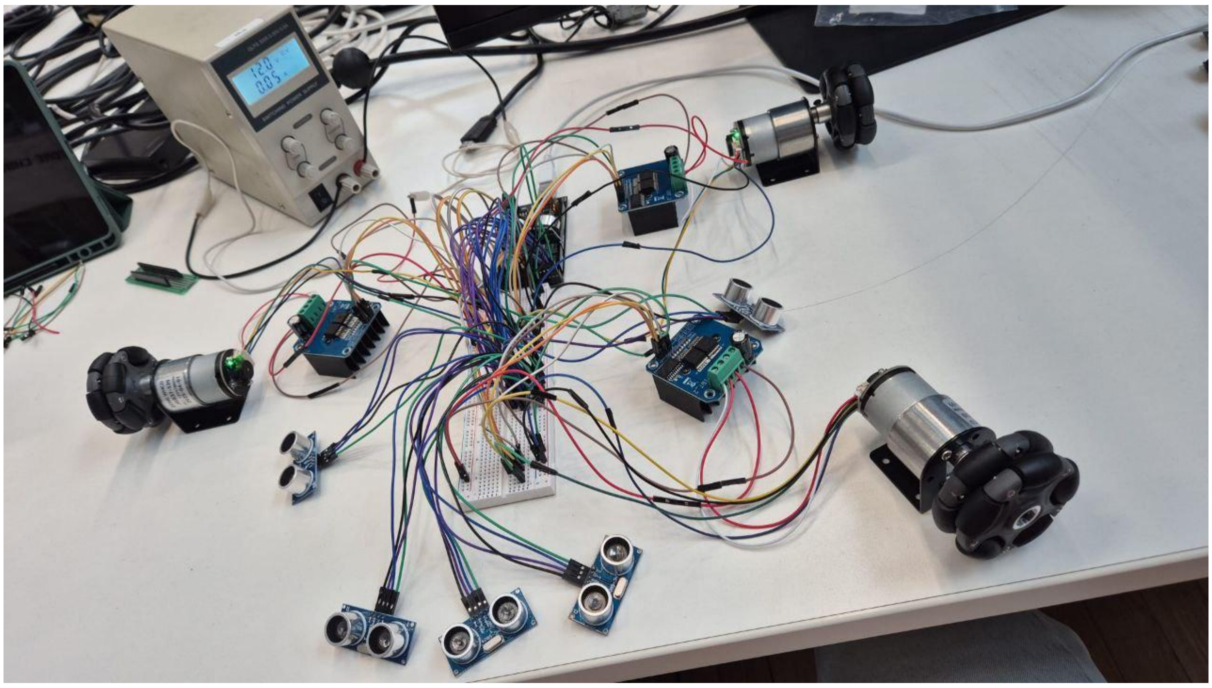
Once the chassis was complete, we proceeded to **mount the electronic components** onto the structure. The three motors were installed and connected to their respective motor drivers, which were in turn wired to the microcontroller and power supply. The placement of components was adapted slightly during this stage to fit the real-world constraints of the wooden base, which differed slightly from the original digital model.

Finally, with the hardware in place, we began **programming the microcontroller**. We uploaded and tested the first version of the code, focusing on motor control and basic behavioral states for future expansion. This phase allowed us to validate our approach and confirm that the robot could move and behave according to the logic defined.

Electronics

Before assembling the electronics on the wooden structure, we first created a **fully functional bench test setup** to validate our components and connections. This early prototype allowed us to test the behavior of **DC motors with omnidirectional wheels** and **ultrasonic sensors** in a controlled environment.

We mounted the three motors on metal brackets and connected them to their respective **DRV8871 motor drivers**. These, in turn, were wired to a breadboard where we centralized all the connections, including the **ATmega328P microcontroller**, **six HC-SR04 ultrasonic sensors**, and **power regulation circuits**. We used a laboratory power supply to ensure voltage stability during the tests.



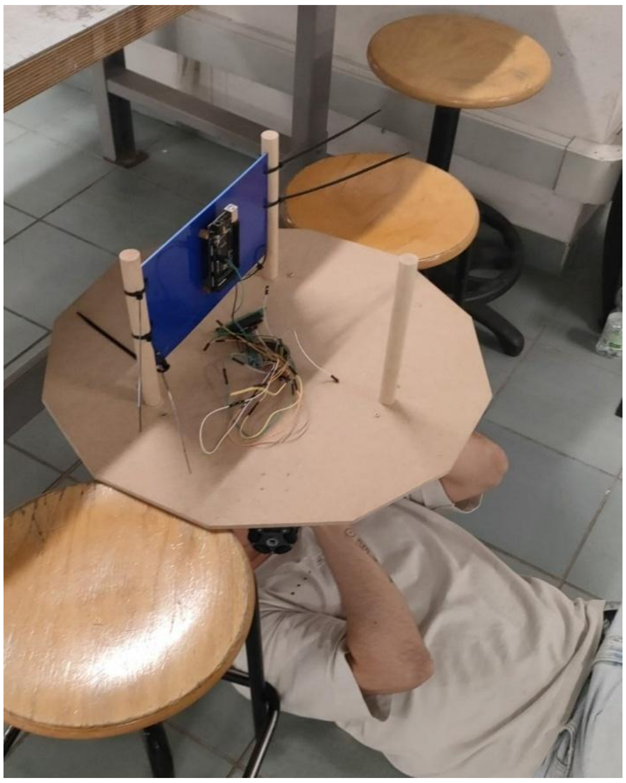
This preliminary setup allowed us to:

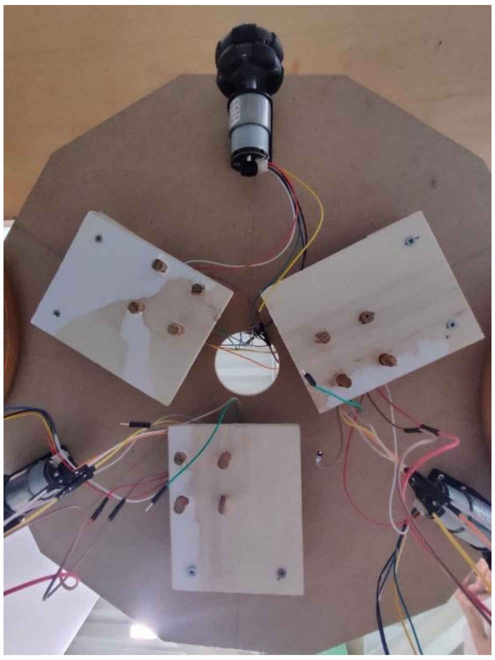
- Confirm that each motor could be controlled independently via PWM signals
- Observe how the omnidirectional wheels behaved in response to speed and direction changes
- Verify the correct operation and reading of the ultrasonic sensors
- Refine the wiring plan before final installation

Once the behavior of the components was confirmed, we moved on to physically **assembling the electronics onto the wooden chassis**.

Once the behaviour of the components was confirmed, we moved on to physically **assembling the electronics onto the wooden chassis**. The three motors were mounted onto the underside of the lower wooden base. Their positioning followed the triangular configuration defined in our design, ensuring balanced and effective omnidirectional movement.

Once the motors were secured, we constructed **custom wooden supports** on the upper side of the base to hold the **motor drivers** (DRV8871). These supports allowed us to fix the drivers in place, providing both accessibility and protection for the connections. With the mechanical and structural elements ready, we proceeded to **wire the motors to the drivers**, carefully routing the motor cables through openings in the base. This ensured that the wiring remained organized and out of the way of moving parts.

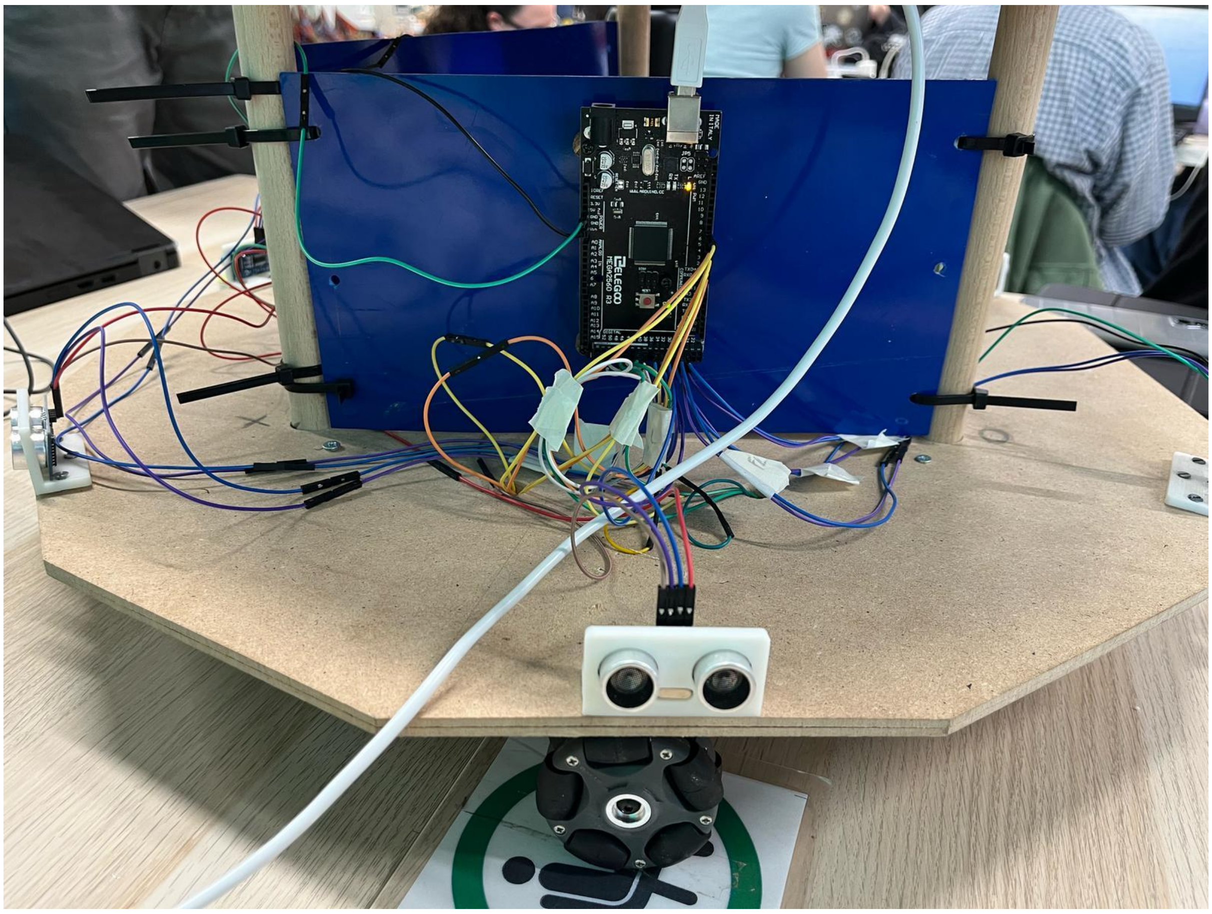




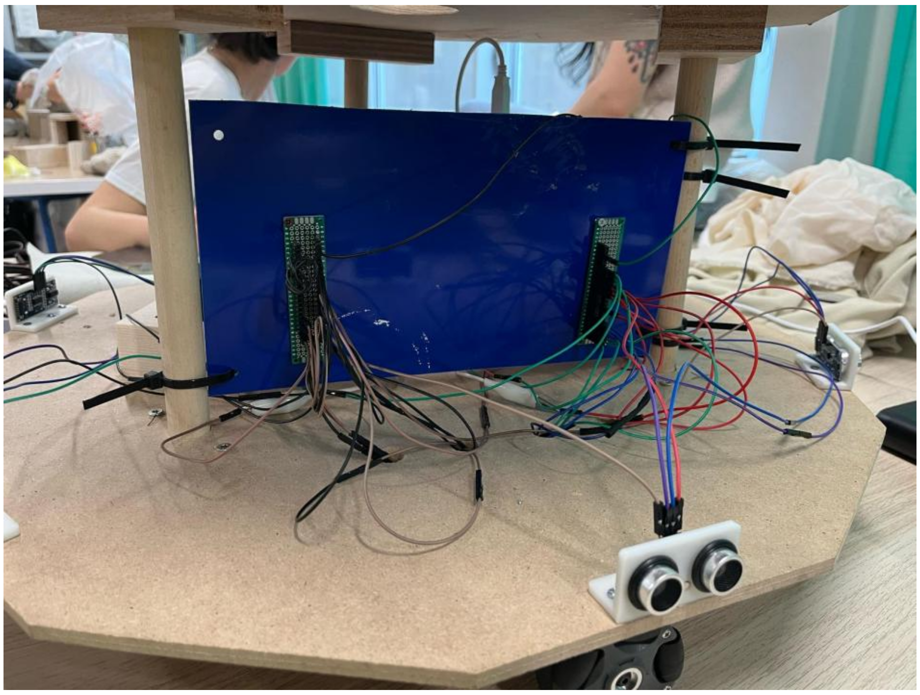
The cables were then guided to the top layer of the chassis, where we planned to place the **microcontroller, power distribution components**, and **sensor modules**. The next step was to connect the **motor drivers to the microcontroller**, completing the control path between the motors and the logic unit. To organize this section of the circuit, we mounted the microcontroller onto a **vertical blue plastic panel** attached to the structure using zip ties. This panel allowed us to keep the wiring centralized, accessible, and separated from the moving parts below.

Once the core control system was in place, we installed the **ultrasonic sensors** around the robot's perimeter. These were mounted on **alternating faces of the dodecagonal wooden base**, using custom white brackets to maintain stable orientation and optimal detection angles. The sensors were positioned to provide wide environmental coverage and were each connected to the microcontroller via dedicated digital pins.

This configuration helped minimize interference between sensors and kept the wiring tidy. Additionally, the vertical layout of the electronics, combined with the wide base, made it easy to manage power distribution, signal integrity, and future maintenance.

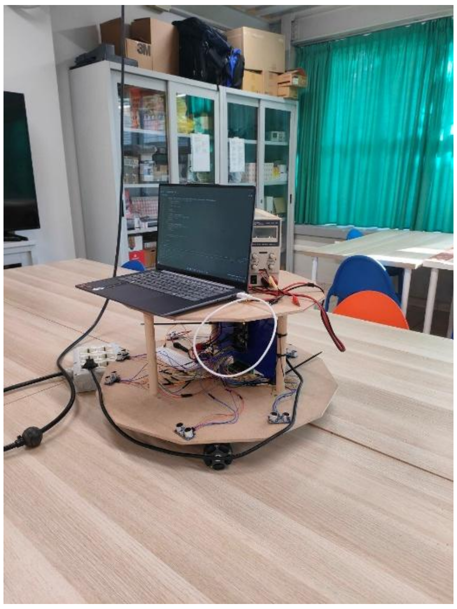


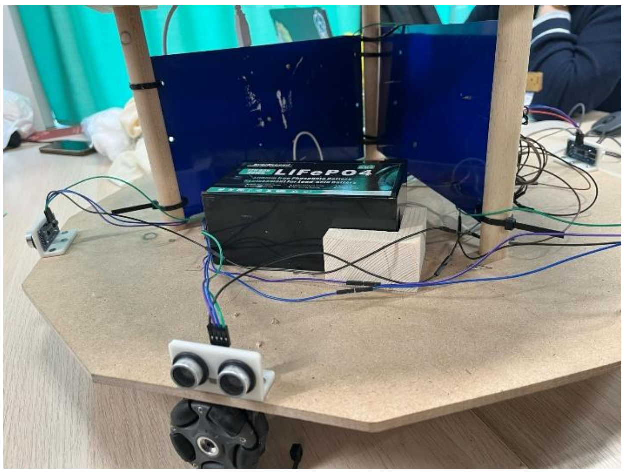
To manage power and grounding efficiently, we used **two small green protoboards** mounted on another blue panel. The **right-hand board** was used to **distribute power (VCC)** to all critical components, including the ultrasonic sensors, motor drivers, and microcontroller. The **left-hand board** was dedicated to **ground (GND) connections**, ensuring a common reference for the entire circuit. This setup allowed us to organize the wiring more cleanly, reduce loose connections, and centralize the power layout in a way that is both stable and maintainable.



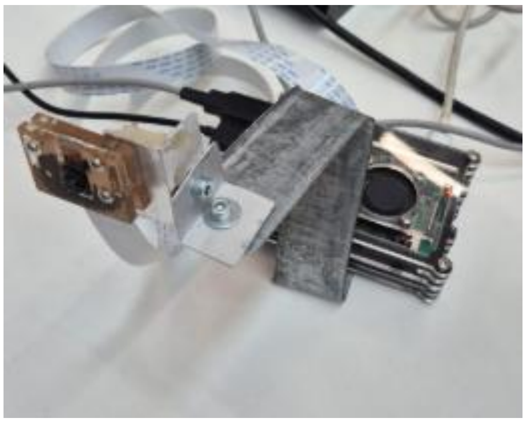
To verify that the electronics were functioning correctly after assembly, we first powered the system using a **laboratory power supply**. This allowed us to set a precise voltage and current limit, ensuring a safe environment for initial testing and protecting the components from potential surges or wiring errors.

Once the wiring was confirmed and all components—motors, drivers, sensors, and microcontroller—responded as expected, we replaced the power supply with the robot's actual **LiFePO4 battery**. With the battery installed and secured to the base, we repeated the tests to confirm that the entire system could run autonomously on its intended power source.
This step-by-step approach helped us detect and correct small connection issues early and ensured a smooth transition to portable, untethered operation.





Additionally, during the development process, we decided to incorporate a **camera module** into the system to enable docking with the charging station using **QR code detection**. This addition allowed the robot to recognize visual markers and orient itself toward the charging point when needed. Since this feature was not planned from the beginning, it was not included in the initial design phase. The decision to implement it emerged organically as the project evolved and we identified the need for more reliable and autonomous behaviour during low-battery scenarios.



Coding

During the development phase, we focused on implementing and validating the full control system for the robot. The process began with an **alpha testing program** (available in the annex), designed to test each subsystem independently—motors, sensors, encoders, and AprilTag detection—before integrating them into a unified control architecture.

In this first version, we structured the tests as **modular routines**, enabling selective testing of each component. This helped verify that the motors responded to PWM signals correctly, the ultrasonic sensors provided consistent readings, and the encoder data could be read and interpreted reliably. These tests were crucial to ensure that hardware and wiring were functioning as expected.

Once individual components were validated, we moved on to the **final implementation** of the control system. The core of the system is organized as a **state-driven architecture**, with multiple operation modes:

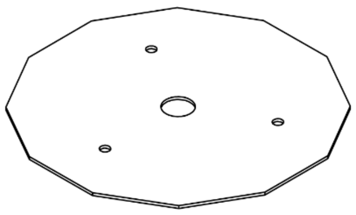
- **Manual mode**, which receives velocity commands over serial communication for direct control.
- **Autonomous mode**, in which the robot patrols predefined paths while avoiding obstacles.
- **Tag following mode**, where it adjusts its motion based on the position of a detected AprilTag.
- **Charging mode**, which guides the robot toward a docking station when low battery is detected.

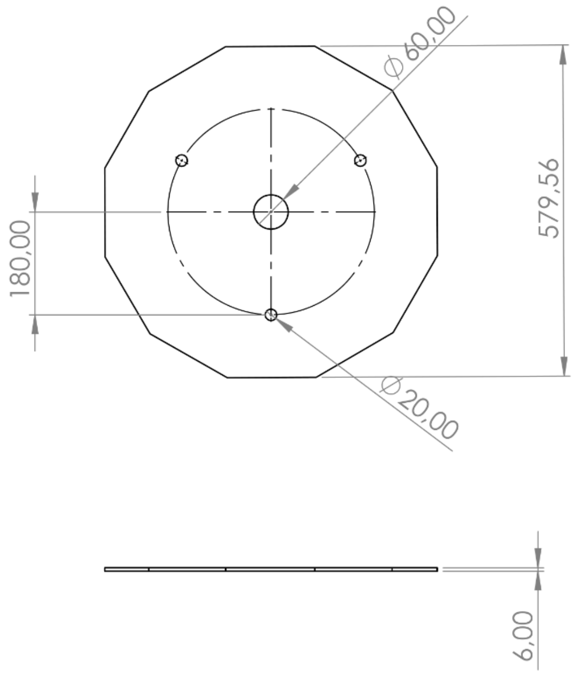
Each mode encapsulates its own logic and can be triggered via commands, making the system modular and easy to expand. Additionally, a **20Hz control loop** manages sensor updates, decision-making, and movement commands, maintaining responsive and consistent behavior.
This structure has proven reliable in testing, with smooth transitions between modes and stable behavior in autonomous operation. Full implementation details and source code are available in the **annex** to this report.

Structure

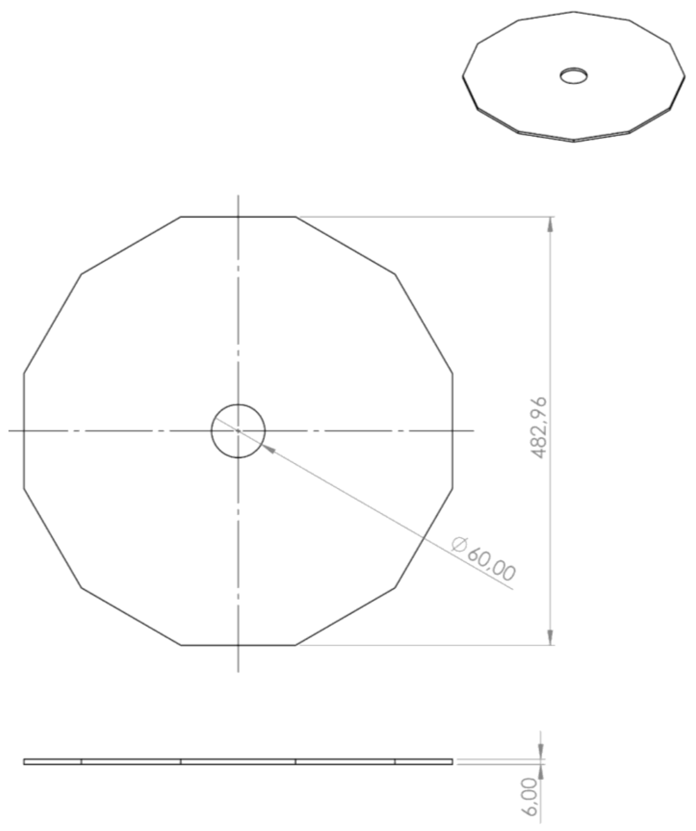
During this development phase, we focused on building each physical component of the robot's structure based on the technical drawings and models defined in the previous stage. The entire chassis was built of wood using standard tools available in Bovisa's *Prototipi* lab. This choice of material allowed for fast prototyping, easy modifications, and rigid support for the movement module.

1. **Base panel.**
   This large dodecagonal panel serves as the foundation for the robot. It supports the motors, the battery, and the lower part of the vertical columns. We cut it from an MDF panel (220 x 152 cm, 3mm) with precision to match the dimensions in the technical drawing, ensuring symmetry and enough surface area for wheel spacing and internal wiring. It serves to mount the electronics and vertical panels and helps complete the structural frame.

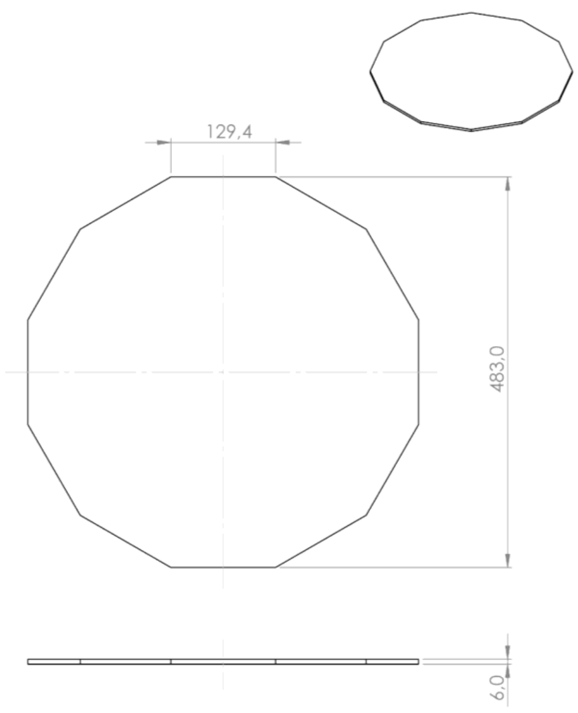




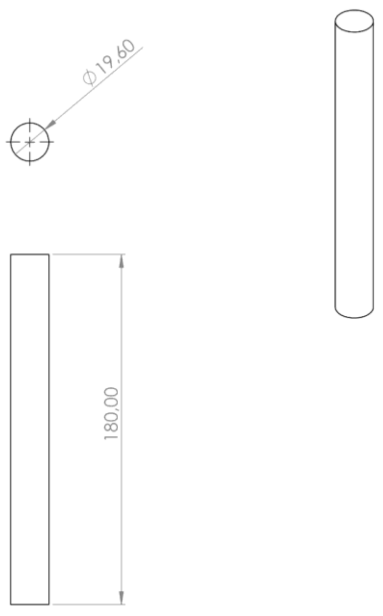
2. **Upper platform.**
   This smaller dodecagonal plate forms the top of the movement module. We constructed it to match the base's outer shape but at a smaller scale. It serves to mount the other modules and helps complete the structural frame.



3. **Lower platform.**
   This element was originally designed as a lower platform, located below the main base. Its purpose was to hold some of the electronic components, freeing up space on the main base. However, during the assembly and testing phase, we encountered a practical issue: this platform reduced the vertical clearance between the wheels and the ground, interfering with proper contact and compromising movement. In sight of this we decided to remove this platform and relocate its electronic components in the bottom face of the base panel.

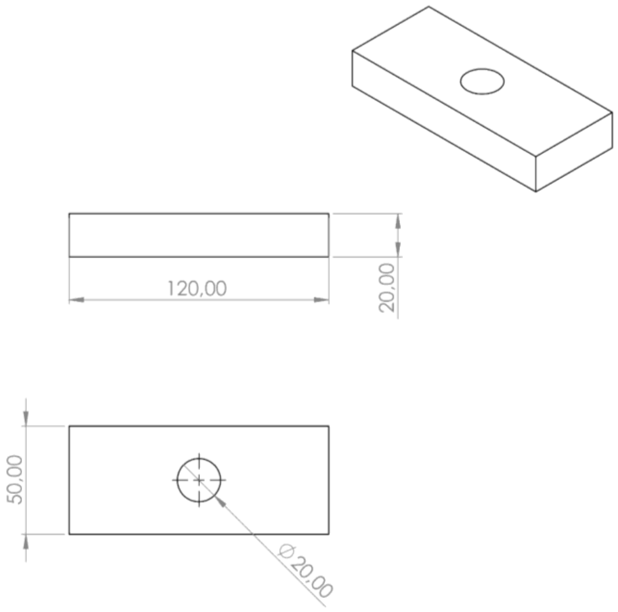


4. **Columns**
   Three vertical wooden rods were cut and sanded to precise height to connect the base and the top plate. They ensure the structure is rigid and that both plates remain parallel.

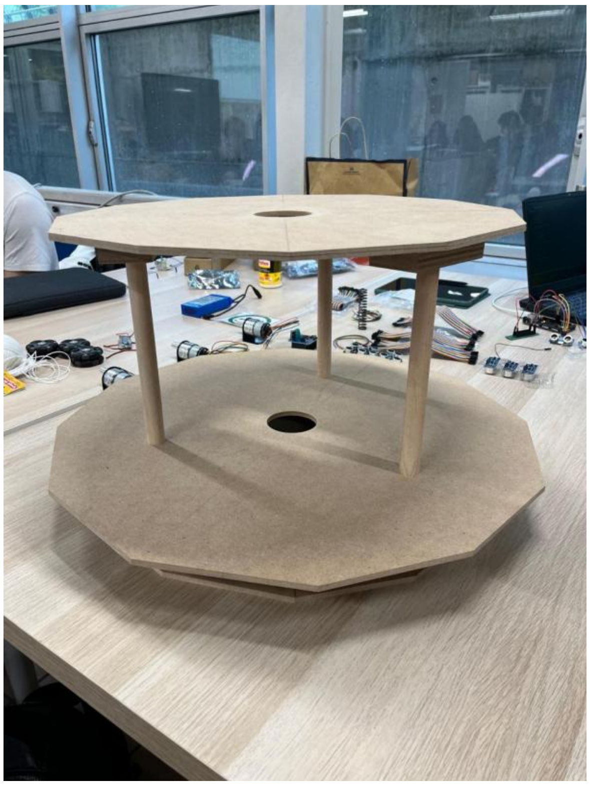


5. **Columns supports**
   These wooden components serve as the **connection points between structural levels**. Three of them are placed between **the lower platform and the base panel**, acting as a solid foundation and providing vertical spacing. The other three are positioned **between the top ends of the columns and the upper platform**, ensuring a stable connection. There are in total six units of

these types of pieces, which help to distribute weight and keep the vertical supports firmly aligned, contributing significantly to the overall rigidity and symmetry of the structure.

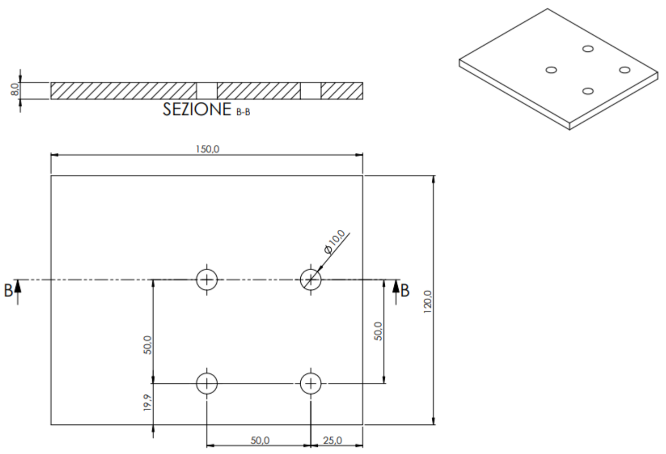


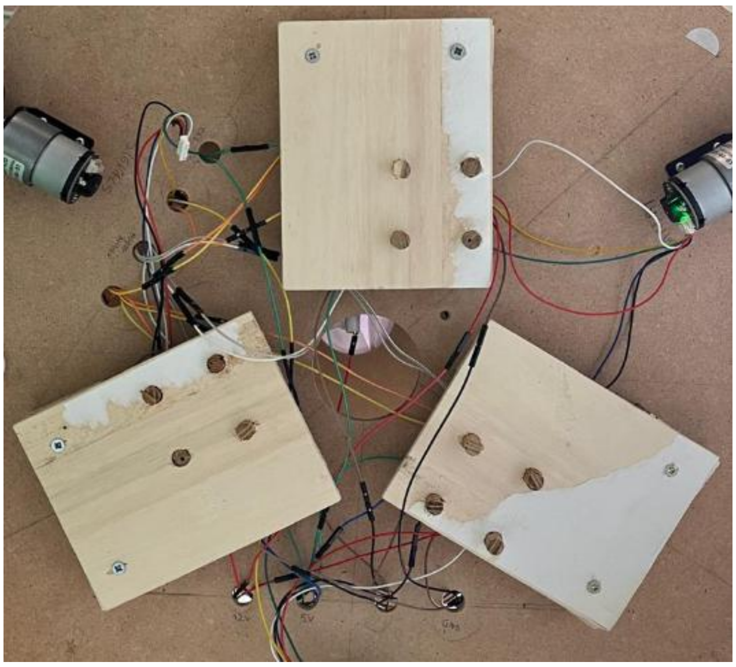
Each part was cut, assembled, and adjusted manually, allowing us to iteratively test fit and stability during the process. The built structure was robust enough to support the entire electronic system and the external aesthetic modules, while remaining accessible for maintenance and future upgrades.



However, during testing, we found that this additional level **reduced the clearance between the omnidirectional wheels and the ground**, which significantly affected the robot's ability to move.

To resolve this, we decided to **remove the lower platform entirely**. As an alternative, we reorganized the layout of the components. The **electronic modules that were originally intended to be mounted on the removed platform were relocated** to the **base panel**. In the bottom face of the base panels, three additional pieces like the ones that can be seen in the image were assembled to have more space for relocating the electronic modules. This change improved ground contact for the wheels and simplified the structural layout, while still maintaining proper distribution of the components across the chassis.





Shape:

During the development phase, we began translating the conceptual design of the robot into a tangible and expressive physical form. The original inspiration was a cooking pot, round, friendly, and slightly whimsical; symbolizing the robot's function related to managing microwave access. However, as construction progressed, we adapted this concept to fit the realities of available tools, materials, and structural constraints in the lab.

As previously mentioned, the robot’s physical skeleton was built using **two dodecagonal wooden platforms;** a base and a top, offering a structure that approximates a circular form while much easier to manufacture using flat wood sheets. This polygonal configuration provided excellent stability, allowed consistent mounting points, and gave us an ideal layout for placing the ultrasonic sensors around the perimeter, on alternating sides.

Once the structural base was completed and tested, we began the **aesthetic transformation** of the robot. In the first stage, the robot was covered with a **white fabric draped over the wooden base**, giving it a soft and continuous outer surface. At this point, we also added the **first vertical supports** that would later hold the turn-dispenser module, made of transparent tubing and filled with balls to simulate a ticket lottery system.

In the second stage, we moved beyond the clean fabric and began attaching **cotton padding** all around the robot. This gave the robot a fluffy, cloud-like appearance, softening its visual presence and introducing a playful, imaginative look that would appeal to users. This cotton layer also served to subtly obscure the structural hardware beneath, enhancing the illusion of a light, floating device.

Finally, we embedded **LED lights inside the cotton structure**, creating a visual effect reminiscent of a **storm cloud**. When lit from within, the cotton shimmered and pulsed, adding depth and dynamic texture to the robot's presence. This stage brought the design much closer to an animated, expressive character rather than just a mobile object.

Throughout this process, we continuously adjusted the shape based on technical needs and visual testing. The final form successfully merges functionality and storytelling, transforming the movement module from a structural base into a **key part of a characterful robot**.

Phase 4: Deliver

Final Robot description

Strategy

The main objective of this final phase was to **prepare the robot for presentation and demonstration**, ensuring that the system was functional, understandable, and representative of the work carried out throughout the project.

We began by **defining a clear list of deliverables**, which included:

- A working prototype of the movement module with integrated electronics and basic behaviors
- A clean and secure physical assembly, including internal wiring, battery placement, and sensor positioning
- A stable version of the control code capable of switching between manual and autonomous modes
- Visual and written documentation to support the explanation of the design and development process
-

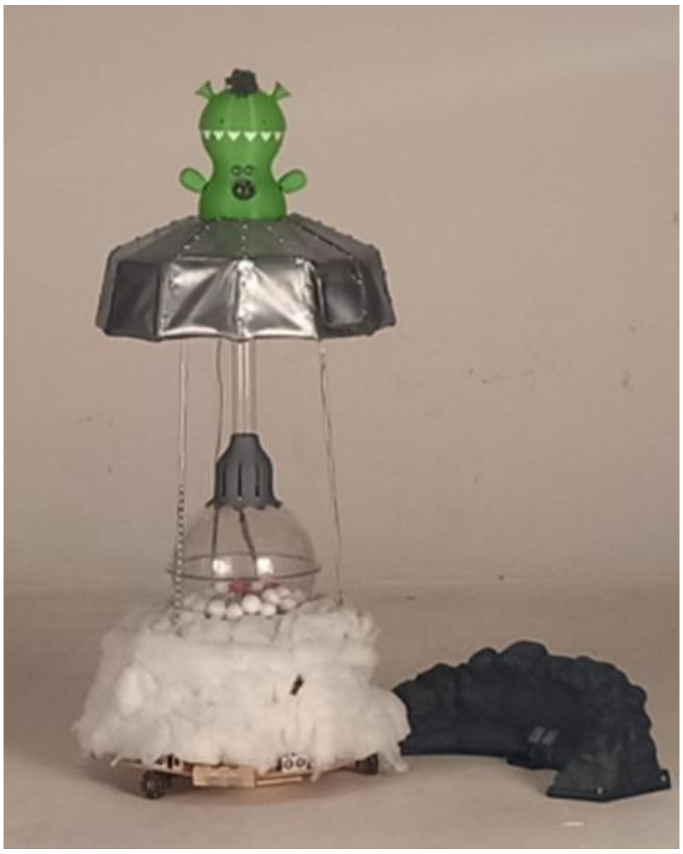
To organize the work, we split tasks across the team, focusing on:

- Final code polishing and debugging
- Mechanical adjustments and reinforcement of the structure
- Aesthetic finishing touches to the robot's appearance
- Preparation of media and presentation material (images, diagrams, video, report)

This strategic approach allowed us to arrive at a functional and communicative prototype, aligned with the original goals and suitable for public demonstration and evaluation.

Shape

In the final delivery stage, the robot evolved from a technical prototype into a **fully integrated, character-driven system**. The movement module (previously shaped into a cloud-like base) was transformed into the foundation for a visually expressive and narratively rich design.



The bottom of the robot remained covered in **soft white cotton**, representing a storm cloud. This not only gave the robot a magical and playful appearance, but also **concealed mechanical components** such as wheels, motors, and sensors. Embedded **LEDs inside the cotton layer** enhanced the illusion by simulating internal lightning effects, adding a sense of animation and depth.
Rising from this cloud base, a **transparent vertical column filled with white balls** was installed to symbolize the microwave ticketing mechanism. This interactive visual metaphor helped communicate the robot’s core function in a light-hearted way. The structure was topped with a **dome shaped like a UFO**, reinforcing the fictional theme of an alien-operated machine that has landed in a university setting to distribute turn tickets.

At the peak, a **green alien figure** served as the face of the robot, further amplifying its friendly and approachable character. This figure helped attract attention and invited interaction, especially in the social environment of a shared student space.

The integration between the modules was carefully planned, both structurally and visually. The cloud base, the transparent ticket chamber, and the dome were aligned with the underlying mechanical layout, while also building a **consistent narrative language**. Every part contributed to the overall identity of the robot—not just as a machine, but as a **social object** with personality and purpose.

For full reference, **all technical drawings and 3D models are included in the documentation package.** These materials illustrate how the shape was designed, tested, and assembled, and serve as a foundation for future iterations or replication.

Mechanics

The final mechanical structure of the robot is composed of **two dodecagonal wooden platforms**: a **base platform** and an **upper platform**, connected by four vertical wooden columns. This simplified configuration was adopted after removing the initially planned intermediate layer, which had interfered with wheel clearance.

The **base platform** is the core of the system and holds **all the critical mechanical and electronic components**:

- On the **underside**, we mounted the **three DC motors**, each connected to a **58 mm omnidirectional wheel** in a triangular layout to enable holonomic movement. The **motor drivers** are also fixed on this side, close to the motors, to reduce wiring length and improve accessibility.

- On the **upper side** of the same platform, we installed the **microcontroller**, **sensor wiring**, **power distribution boards**, and **battery**. A vertical plastic panel helps organize the electronics, and the ultrasonic sensors are positioned around the base perimeter on alternating sides.

The **upper platform** is structurally intended to support **other modules**, such as ticket dispensers or user interface components. It is fixed to the columns and aligned with the base to maintain symmetry and provide vertical expansion space.

All mechanical parts were fabricated using MDF panels and wooden supports, based on our technical drawings. The final structure provides a **stable, modular, and compact skeleton** for the movement module, optimized for both functionality and future adaptability.

Electronics

The electronics of the movement module were designed to support omnidirectional movement and basic obstacle detection. Our approach was to keep the system modular, efficient, and easy to assemble, using well-documented components compatible with Arduino-based development.

Main components and setup

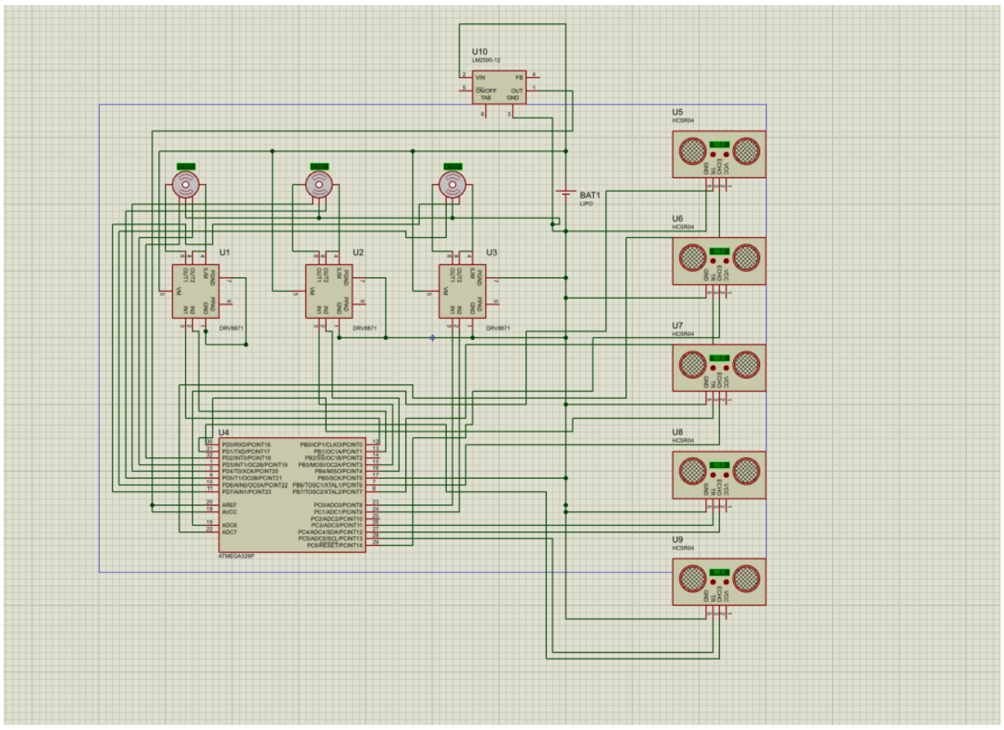
- **Microcontroller – ATmega328P**: Acts as the core of the movement module. It handles all the logic related to motor control, sensor reading, and decision-making. We used digital pins to interface with motor drivers and ultrasonic sensors.

- **DC Motors – JGB37-520 with encoders**: Three brushed DC motors are responsible for omnidirectional movement. Encoders are included for possible future use with closed-loop speed or distance control.
- **Motor Drivers – DRV8871**: Each motor is controlled through a dedicated DRV8871 H-bridge motor driver. These drivers are compact, reliable, and support the current required by the motors. Each driver is connected to the microcontroller via two control pins for speed and direction.
- **Omnidirectional Wheels – 58mm Nylon**: The wheels are attached to the motor shafts using 6mm aluminum hubs. These allow smooth and accurate multidirectional motion, essential for a patrol robot that must reposition itself easily.
- **Power Supply – LiPo battery with LM2596 regulator**: The circuit is powered by a LiPo battery. To provide consistent voltage to logic-level components and motor drivers, an LM2596 step-down regulator is used. It ensures the ATmega328P, and sensors receive stable 5V output.
- **Obstacle Detection – HC-SR04 ultrasonic sensors**: Five ultrasonic sensors are distributed around the body of the robot to allow for basic obstacle detection. Each sensor is connected to a pair of digital pins on the ATmega328P, and they are intended to be triggered one at a time to avoid signal interference.
- **Wiring – Dupont cables and prototyping layout**: For prototyping, we used male-to-male and female-to-male Dupont jumper cables, enabling flexible reconfiguration and easy debugging during testing.
- **Camera taken from the AirLab**: A compact camera module was added during the development phase to enable visual detection of the charging station via a QR code. This component provides the robot with basic computer vision capabilities, expanding its autonomy beyond obstacle-based navigation.

System relationship overview

- The **ATmega328P** receives power from the regulated 5V supply.
- It sends PWM and direction signals to the **DRV8871 motor drivers**, which in turn control each of the **three motors**.
- The same microcontroller reads **ultrasonic distance values** from the HC-SR04 sensors.
- Based on sensor input and predefined logic, the microcontroller controls movement and reacts to obstacles through the actuation of the motors.

This configuration ensures the robot is capable of autonomous and reactive movement, with a clean base that allows for future integration of more advanced sensing and interaction features. A complete scheme of the electronics of the module can be seen below:



Informatics

The final implementation of the movement module software was developed in C++ for an Arduino-compatible microcontroller. It is based on a state-driven architecture with multiple operation modes: manual, autonomous, tag-following, and charging. The system runs a control loop at 20 Hz, handling sensor updates, motor control, and decision-making in real time.

Development began with a modular alpha testing script that allowed us to validate motors, ultrasonic sensors, encoders, and AprilTag detection independently. Once each subsystem was confirmed to work reliably, we moved to the unified control system, with clear routines for input processing and motion execution.

We used only core Arduino libraries to maintain simplicity and portability. The code is structured, commented on, and ready for further development. The test scripts and the final implementation are available in the appendix and can also be downloaded at the following link: https://github.com/Daru-Mau/ILMModule.git.

Conclusion

Throughout this project, we gained valuable hands-on experience in building a functional robotic system from the ground up. One of the most important lessons was the need to balance ambition with time management. Although we had a clear concept early on, the time constraints forced us to adjust and prioritize essential features. This highlighted the importance of starting integration early and making incremental progress rather than deferring complex tasks until the final stages.

We also learned the value of clear and constant communication within the team. Coordinating tasks, sharing updates, and adapting plans based on each member’s progress made the difference between individual work and collaborative engineering. Dividing the workload logically (mechanics, electronics, code) helped us move forward efficiently, but it was equally important to stay synchronized.

Movement control also presented technical challenges. While omnidirectional movement offered great flexibility, it also introduced some complexities, particularly during initial testing and mechanical adjustments. We had to iterate several times to achieve reliable, smooth motion.

Finally, working together as a coordinated team proved to be one of the most critical factors for progress. Regular updates, shared responsibilities, and openness to feedback allowed us to align our efforts and avoid bottlenecks, especially during the most intense phases of assembly and testing. If we were to start over, we would put even more emphasis on planning for the unexpected, testing early, and working incrementally with deliverable checkpoints.

Appendix

A.Code

A.1. HC-SR04 testing:

```
const int trigPins[6] = {2, 4, 6, 8, 10, 12}; // Trigger pins for 6 sensors

const int echoPins[6] = {3, 5, 7, 9, 11, 13}; // Echo pins for 6 sensors

long distances[6]; // Array to store distance readings

void setup() {

  Serial.begin(9600);

  for (int i = 0; i < 6; i++) {

    pinMode(trigPins[i], OUTPUT);

    pinMode(echoPins[i], INPUT);

  }

}

void loop() {

  for (int i = 0; i < 6; i++) distances[i] = readDistance(trigPins[i], echoPins[i]);

  for (int i = 0; i < 6; i++) {

    Serial.print("Sensor");

    Serial.print(i + 1);

    Serial.print(":");

    Serial.print(distances[i]);

    Serial.println("cm");

  }

  Serial.println("------------------------");

  delay(500);

}

long readDistance(int trigPin, int echoPin) {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 30000);

  long distance = duration * 0.034 / 2;

  if (distance == 0 || distance > 400) return -1;

  return distance;

}
```

A.2. Engines and HC-SR04 Integration

```
// === HolonomicDrive Class ===

class HolonomicDrive {

public:

  HolonomicDrive(int rF, int lF, int enF, int rL, int lL,

  int enL, int rR, int lR, int enR) {

    // Assign motor pins

    RPWM_RIGHT = rF;

    LPWM_RIGHT = lF;

    EN_RIGHT = enF;

    RPWM_LEFT = rL;

    LPWM_LEFT = lL;

    EN_LEFT = enL;

    RPWM_BACK = rR;

    LPWM_BACK = lR;

    EN_BACK = enR;


    // Set motor pins as outputs

    int pins[] = { RPWM_RIGHT, LPWM_RIGHT, EN_RIGHT,

                   RPWM_LEFT, LPWM_LEFT, EN_LEFT,

                   RPWM_BACK, LPWM_BACK, EN_BACK };


    for (int i = 0; i < 9; i++) {

      pinMode(pins[i], OUTPUT);
    }

    enableMotors();
  }

  void enableMotors() {
    digitalWrite(EN_RIGHT, HIGH);
    digitalWrite(EN_LEFT, HIGH);
    digitalWrite(EN_BACK, HIGH);
  }

  void moveForward(int speed) {
    moveMotor(RPWM_RIGHT, LPWM_RIGHT, speed);
    moveMotor(RPWM_LEFT, LPWM_LEFT, speed);
    moveMotor(RPWM_BACK, LPWM_BACK, speed);
  }

  void moveBackward(int speed) {
    moveMotorBackward(RPWM_RIGHT, LPWM_RIGHT, speed);
    moveMotorBackward(RPWM_LEFT, LPWM_LEFT, speed);
    moveMotorBackward(RPWM_BACK, LPWM_BACK, speed);
  }

  void slideLeft(int speed) {
    moveMotor(RPWM_BACK, LPWM_BACK, speed);
    moveMotorBackward(RPWM_LEFT, LPWM_LEFT, speed);
    stopMotor(RPWM_RIGHT, LPWM_RIGHT);
  }

  void slideRight(int speed) {
    moveMotorBackward(RPWM_BACK, LPWM_BACK, speed);

    moveMotor(RPWM_LEFT, LPWM_LEFT, speed);

    stopMotor(RPWM_RIGHT, LPWM_RIGHT);

  }


  void rotateLeft(int speed) {

    moveMotorBackward(RPWM_RIGHT, LPWM_RIGHT, speed);

    moveMotor(RPWM_LEFT, LPWM_LEFT, speed);

    moveMotorBackward(RPWM_BACK, LPWM_BACK, speed);

  }


  void rotateRight(int speed) {

    moveMotor(RPWM_RIGHT, LPWM_RIGHT, speed);

    moveMotorBackward(RPWM_LEFT, LPWM_LEFT, speed);

    moveMotor(RPWM_BACK, LPWM_BACK, speed);

  }


  void stopAll() {

    stopMotor(RPWM_RIGHT, LPWM_RIGHT);

    stopMotor(RPWM_LEFT, LPWM_LEFT);

    stopMotor(RPWM_BACK, LPWM_BACK);

  }


private:

  int RPWM_RIGHT, LPWM_RIGHT, EN_RIGHT;

  int RPWM_LEFT, LPWM_LEFT, EN_LEFT;

  int RPWM_BACK, LPWM_BACK, EN_BACK;


  void moveMotor(int rpwm, int lpwm, int speed) {

    analogWrite(rpwm, speed);
```

```
        analogWrite(lpwm, 0);
    }

    void moveMotorBackward(int rpwm, int lpwm, int speed) {
        analogWrite(rpwm, 0);
        analogWrite(lpwm, speed);
    }

    void stopMotor(int rpwm, int lpwm) {
        analogWrite(rpwm, 0);
        analogWrite(lpwm, 0);
    }
};

// === Pin Definitions ===

// Ultrasonic Sensors
const int TP_FRONT = 22, EP_FRONT = 23;
const int TP_FRONT_LEFT = 24, EP_FRONT_LEFT = 25;
const int TP_FRONT_RIGHT = 26, EP_FRONT_RIGHT = 27;
const int TP_LEFT = 28, EP_LEFT = 29;
const int TP_RIGHT = 30, EP_RIGHT = 31;
const int TP_BACK = 32, EP_BACK = 33;

// Motor Driver Pins
const int RPWM_RIGHT = 37, LPWM_RIGHT = 36, REN_RIGHT = 39, LEN_RIGHT = 38;
const int RPWM_LEFT = 43, LPWM_LEFT = 42, REN_LEFT = 45, LEN_LEFT = 44;
const int RPWM_BACK = 49, LPWM_BACK = 48, REN_BACK = 51, LEN_BACK = 50;

// Encoder Pins
```

```
const int ENCODER_FRONT_A = 33;

const int ENCODER_FRONT_B = 32;


// === Globals ===

float distFront, distFrontLeft, distFrontRight, distLeft, distRight, distBack;

const float OBSTACLE_DISTANCE = 10.0;  // cm


// === Setup ===


void setup() {

  Serial.begin(9600);


  int trigPins[] = { TP_FRONT, TP_FRONT_LEFT, TP_FRONT_RIGHT,

  TP_LEFT, TP_RIGHT, TP_BACK };

  int echoPins[] = { EP_FRONT, EP_FRONT_LEFT, EP_FRONT_RIGHT,

  EP_LEFT, EP_RIGHT, EP_BACK };


  for (int i = 0; i < 6; i++) {

    pinMode(trigPins[i], OUTPUT);

    pinMode(echoPins[i], INPUT);

  }


  int motorPins[] = {

    RPWM_RIGHT, LPWM_RIGHT, REN_RIGHT, LEN_RIGHT,

    RPWM_LEFT, LPWM_LEFT, REN_LEFT, LEN_LEFT,

    RPWM_BACK, LPWM_BACK, REN_BACK, LEN_BACK

  };


  for (int i = 0; i < 12; i++) pinMode(motorPins[i], OUTPUT);
```

```
    digitalWrite(REN_RIGHT, HIGH);
    digitalWrite(LEN_RIGHT, HIGH);
    digitalWrite(REN_LEFT, HIGH);
    digitalWrite(LEN_LEFT, HIGH);
    digitalWrite(REN_BACK, HIGH);
    digitalWrite(LEN_BACK, HIGH);

    pinMode(ENCODER_FRONT_A, INPUT);
    pinMode(ENCODER_FRONT_B, INPUT);

    Serial.println("ROBOT READY");
}

// === Distance Sensing ===

float readDistance(int trigPin, int echoPin) {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH, 30000);
    return (duration <= 0) ? 999.0 : duration * 0.034 / 2;
}

void updateDistances() {
    distFront = readDistance(TP_FRONT, EP_FRONT);
    distFrontLeft = readDistance(TP_FRONT_LEFT, EP_FRONT_LEFT);
    distFrontRight = readDistance(TP_FRONT_RIGHT, EP_FRONT_RIGHT);
```

```
  distLeft = readDistance(TP_LEFT, EP_LEFT);

  distRight = readDistance(TP_RIGHT, EP_RIGHT);

  distBack = readDistance(TP_BACK, EP_BACK);

}


// === Motor Logic ===


void setMotor(int rpwm, int lpwm, float motorSpeed) {

  motorSpeed = constrain(motorSpeed, -255, 255);


  if (motorSpeed > 0) {

    analogWrite(rpwm, motorSpeed);

    analogWrite(lpwm, 0);

  } else {

    analogWrite(rpwm, 0);

    analogWrite(lpwm, -motorSpeed);

  }

}


// === Triskar Movement Logic ===


void moveRobot(float vx, float vy, float omega, int speed = 180) {

  float v_front = -vx + omega;

  float v_left = 0.5 * vx - 0.866 * vy + omega;

  float v_right = 0.5 * vx + 0.866 * vy + omega;


  float maxVal = max(max(abs(v_front), abs(v_left)), abs(v_right));


  if (maxVal > 1.0) {

    v_front /= maxVal;
```

```
        v_left /= maxVal;

        v_right /= maxVal;

    }


    setMotor(RPWM_RIGHT, LPWM_RIGHT, v_front * speed);

    setMotor(RPWM_LEFT, LPWM_LEFT, v_left * speed);

    setMotor(RPWM_BACK, LPWM_BACK, v_right * speed);

}


void stopAllMotors() {

    analogWrite(RPWM_RIGHT, 0);

    analogWrite(LPWM_RIGHT, 0);

    analogWrite(RPWM_LEFT, 0);

    analogWrite(LPWM_LEFT, 0);

    analogWrite(RPWM_BACK, 0);

    analogWrite(LPWM_BACK, 0);

}


// === Main Loop ===


void loop() {

    updateDistances();


    Serial.print("F: "); Serial.print(distFront);

    Serial.print(" FL: "); Serial.print(distFrontLeft);

    Serial.print(" FR: "); Serial.print(distFrontRight);

    Serial.print(" L: "); Serial.print(distLeft);

    Serial.print(" R: "); Serial.print(distRight);

    Serial.print(" B: "); Serial.println(distBack);
```

```
  if (distFront < OBSTACLE_DISTANCE || distFrontLeft < OBSTACLE_DISTANCE ||
  distFrontRight < OBSTACLE_DISTANCE) {
    Serial.println("Obstacle in front - rotating to avoid");
    if (distLeft > distRight) {
      moveRobot(0.0, 0.0, -1.0);
    } else {
      moveRobot(0.0, 0.0, 1.0);
    }
    delay(500);
    stopAllMotors();
  }
  else if (distLeft < OBSTACLE_DISTANCE) {
    Serial.println("Obstacle on left - sliding right");
    moveRobot(1.0, 0.0, 0.0);
    delay(400);
    stopAllMotors();
  }
  else if (distRight < OBSTACLE_DISTANCE) {
    Serial.println("Obstacle on right - sliding left");
    moveRobot(-1.0, 0.0, 0.0);
    delay(400);
    stopAllMotors();
  }
  else {
    Serial.println("Path is clear - moving forward");
    moveRobot(-1.0, 1.0, 0.0);
  }

  delay(200);
}
```

A.3. Alpha Script

```
class AlphaTest {

public:

  // Test modes

  enum TestMode {

    MOTOR_TEST,

    SENSOR_TEST,

    ENCODER_TEST,

    IMU_TEST,

    TAG_TEST

  };


  void runTest(TestMode mode) {

    switch(mode) {

      case MOTOR_TEST:

        testMotorSequence();

        break;

      case SENSOR_TEST:

        testSensors();

        break;

      case ENCODER_TEST:

        testEncoders();

        break;

      case IMU_TEST:

        testIMU();

        break;

      case TAG_TEST:

        testAprilTag();
```

```
            break;
        }
    }

private:
    void testMotorSequence() {
        // Test each motor individually
        Serial.println("Testing Front Motor");
        testSingleMotor(RPWM_RIGHT, LPWM_RIGHT);

        Serial.println("Testing Left Motor");
        testSingleMotor(RPWM_LEFT, LPWM_LEFT);

        Serial.println("Testing Back Motor");
        testSingleMotor(RPWM_BACK, LPWM_BACK);

        // Test combined movements
        Serial.println("Testing Forward Movement");
        moveRobot(-1.0, 0.0, 0.0, 150);
        delay(2000);
        stopAllMotors();

        Serial.println("Testing Rotation");
        moveRobot(0.0, 0.0, 1.0, 150);
        delay(2000);
        stopAllMotors();
    }

    void testSingleMotor(int rpwm, int lpwm) {
        // Forward
```

```
  analogWrite(rpwm, 150);

  analogWrite(lpwm, 0);

  delay(2000);

  // Stop

  analogWrite(rpwm, 0);

  analogWrite(lpwm, 0);

  delay(1000);

  // Backward

  analogWrite(rpwm, 0);

  analogWrite(lpwm, 150);

  delay(2000);

  // Stop

  analogWrite(rpwm, 0);

  analogWrite(lpwm, 0);

  delay(1000);

}


void testSensors() {

  int maxTests = 50;

  for(int i = 0; i < maxTests; i++) {

    updateDistances();

    printSensorData();

    delay(200);

  }

}


void testEncoders() {

  long startLeft = leftEncoderCount;

  long startRight = rightEncoderCount;
```

```
    // Move forward briefly
    moveRobot(-1.0, 0.0, 0.0, 150);
    delay(1000);
    stopAllMotors();

    // Check encoder counts
    Serial.print("Left Encoder Delta: ");
    Serial.println(leftEncoderCount - startLeft);
    Serial.print("Right Encoder Delta: ");
    Serial.println(rightEncoderCount - startRight);
}

void testIMU() {
    float startTheta = theta;

    // Rotate 90 degrees
    moveRobot(0.0, 0.0, 1.0, 150);
    delay(1000);
    stopAllMotors();

    Serial.print("Rotation (degrees): ");
    Serial.println((theta - startTheta) * 180/PI);
}

void testAprilTag() {
    int testDuration = 30; // seconds
    unsigned long startTime = millis();

    while(millis() - startTime < testDuration * 1000) {
        if(tagDetected) {
            Serial.print("Tag detected - ID: ");
            Serial.print(tagID);
            Serial.print(" X: ");
            Serial.print(tagX);
            Serial.print(" Z: ");
            Serial.println(tagZ);
        }
        checkSerialForTag();
        delay(100);
    }
  }
};
```

A.4. Final Implementation

```
class RobotController {
public:
  enum OperationMode {
    MANUAL,
    AUTONOMOUS,
    TAG_FOLLOWING,
    CHARGING
  };

  RobotController() : currentMode(MANUAL) {
    setupHardware();
    calibrateSensors();
  }

  void run() {
    while(true) {
          updateSensors();

          processCommands();

          updateState();

          controlLoop();

          reportStatus();

          delay(50); // 20Hz update rate

    }

  }


private:

  OperationMode currentMode;

  bool emergencyStopped = false;

  unsigned long lastUpdate = 0;


  void setupHardware() {

    // Initialize all pins

    setupMotors();

    setupSensors();

    setupCommunication();


    Serial.println("Hardware initialization complete");

  }


  void calibrateSensors() {

    // Zero encoders

    leftEncoderCount = 0;

    rightEncoderCount = 0;


    // Calibrate IMU

    imu.calibrateGyro();
```

```
  // Reset position
  x_pos = 0.0;
  y_pos = 0.0;
  theta = 0.0;
}

void updateSensors() {
  updateDistances();   // Ultrasonic sensors
  updatePosition();    // Encoders
  updateIMUData();    // IMU
  checkSerialForTag(); // AprilTag
}

void processCommands() {
  if(Serial.available()) {
    char cmd = Serial.read();
    switch(cmd) {
      case 'M': currentMode = MANUAL; break;
      case 'A': currentMode = AUTONOMOUS; break;
      case 'T': currentMode = TAG_FOLLOWING; break;
      case 'C': currentMode = CHARGING; break;
      case 'E': emergencyStop(); break;
      case 'R': resetEmergencyStop(); break;
    }
  }
}

void updateState() {
  unsigned long now = millis();
  float dt = (now - lastUpdate) / 1000.0;

  lastUpdate = now;



  // Update localization with sensor fusion

  updateLocalization(dt);



  // Check for obstacles

  checkObstacles();

}



void controlLoop() {

  if(emergencyStopped) {

    stopAllMotors();

    return;

  }



  switch(currentMode) {

    case MANUAL:

      handleManualControl();

      break;



    case AUTONOMOUS:

      handleAutonomousMode();

      break;



    case TAG_FOLLOWING:

      handleTagFollowing();

      break;



    case CHARGING:
            handleChargingMode();
            break;
    }
}

void handleManualControl() {
    if(Serial.available() >= 3) {
        float vx = (Serial.read() - 128) / 128.0;
        float vy = (Serial.read() - 128) / 128.0;
        float omega = (Serial.read() - 128) / 128.0;
        moveRobot(vx, vy, omega);
    }
}

void handleAutonomousMode() {
    // Simple obstacle avoidance
    if(obstacleDetected()) {
        avoidObstacle();
    } else {
        // Continue on path or exploration
        moveRobot(-1.0, 0.0, 0.0);
    }
}

void handleTagFollowing() {
    if(tagDetected) {
        // Proportional control to follow tag
        float angularCorrection = tagX * 2.0;
        moveRobot(-0.8, 0.0, angularCorrection);
    } else {
            // Search pattern when tag lost
            moveRobot(0.0, 0.0, 0.5);
        }
    }

    void handleChargingMode() {
        if(atChargingStation()) {
            stopAllMotors();
            // Enable charging circuit
        } else if(tagDetected && tagID == CHARGING_TAG_ID) {
            approachChargingStation();
        } else {
            searchForChargingStation();
        }
    }

    void reportStatus() {
        // Send status over Serial every 500ms
        static unsigned long lastReport = 0;
        if(millis() - lastReport > 500) {
            sendStatusReport();
            lastReport = millis();
        }
    }
};

// Main program
RobotController robot;

void setup() {
    Serial.begin(115200);

    Wire.begin();

    robot = RobotController();

}


void loop() {

    robot.run();

}
```

B. Components List

- 3 × JGB37-520 12V DC Motors with encoders
- 3 × 58mm Omnidirectional wheels
- 3 × DRV8871 Motor drivers
- 1 × ATmega328P Microcontroller board (Airlab)
- 6 × HC-SR04 Ultrasonic distance sensors
- 1 × 11.1V 2200mAh LiPo battery
- 1 × Camera module (Airlab)
- Various wooden panels, supports, and fasteners
- LED strips for illumination (Airlab)
- Cotton padding and white fabric for aesthetic covering
- Transparent tube for ticket dispensing mechanism
- Decorative elements for the robot character

USER MANUAL

INDEX

1. What is this product?

This movement module is a sub-component of a larger social robot designed to manage a microwave queue in a shared student environment. Its primary function is to provide mobility to the robot, allowing it to patrol autonomously, avoid obstacles, and move toward charging station as needed. The module includes omnidirectional wheels, DC motors, motor drivers, a microcontroller, sensors, and a battery-powered system.

2. Product Overview

The main components of the movement module are the following:

- A. Upper wooden platform
- B. Base wooden platform
- C. 3x Omnidirectional wheels
- D. 3x DC motors with drivers (mounted underneath)
- E. ATmega328P microcontroller
- F. HC-SR04 ultrasonic sensors
- G. LiPo Battery and LM2596 voltage regulator
- H. Power distribution boards

3. How to Insert the Battery

1. Lift the fabric cover to access the batteries.
2. Carefully connect the LiPo battery to the main power connector.
3. Secure the battery in place to prevent movement during operation.

4. How to Switch ON / OFF

• To switch ON: Plug in the LiPo battery connector. The system powers up automatically.
• To switch OFF: Unplug the battery. Ensure all LEDs and movement are off before handling.

5. User Feedback

The movement module provides the following feedback to the user:
- Motion response: The robot begins patrolling or adjusting based on sensor input.
- Serial messages: Debug or status messages can be read via USB if connected to a PC.

MAINTAINANCE MANUAL

INDEX

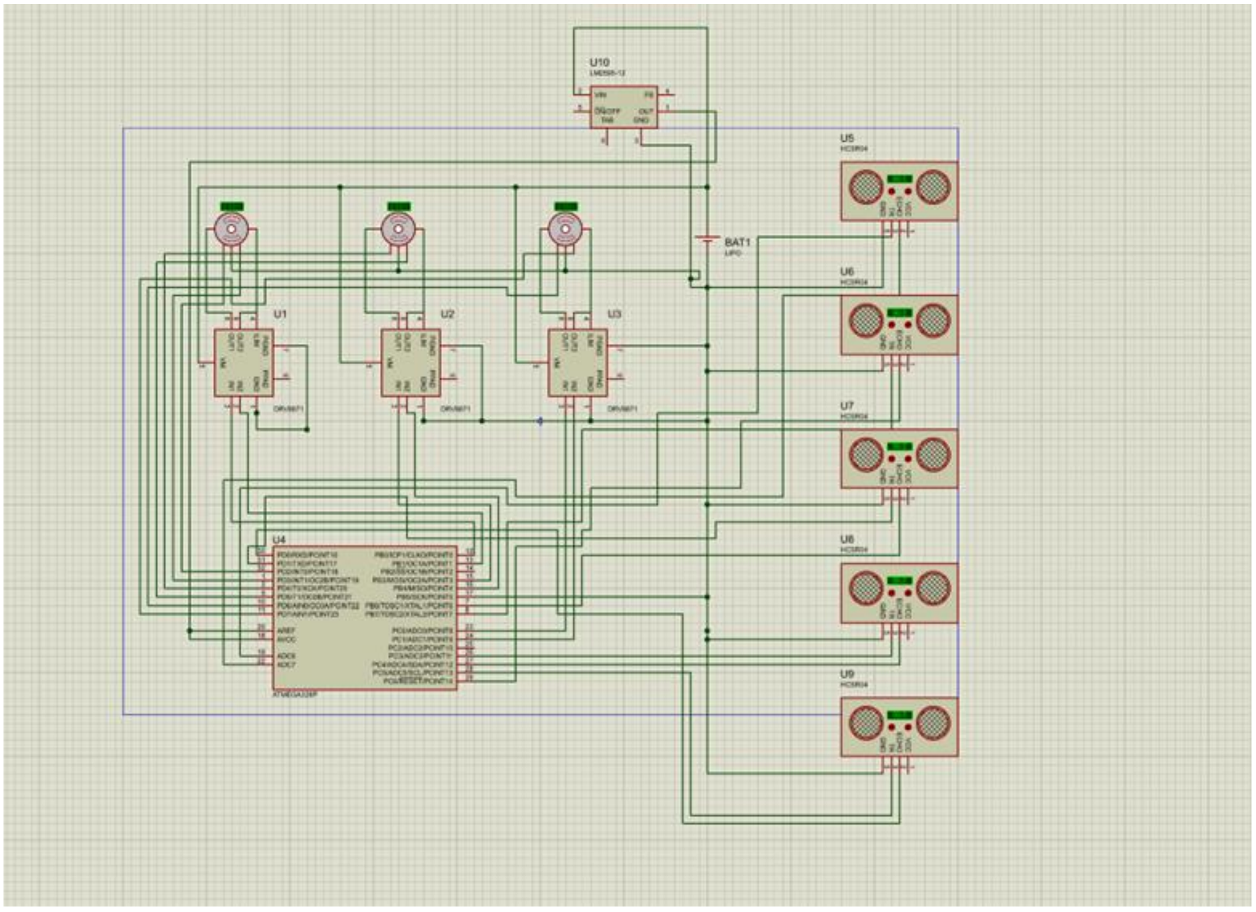
1. Safety Guidelines

To ensure safe handling, operation, and maintenance of the movement module, follow these guidelines:

- Always power off and disconnect the battery before performing any electrical or mechanical maintenance.
- Avoid touching motor shafts or wheels while the robot is powered, as sudden movements may cause injury.
- Use insulated tools when working near live circuits.
- Keep liquids away from the robot to prevent short circuits or corrosion.
- Dispose of damaged batteries according to local electronic waste regulations.
- Do not operate the robot on uneven surfaces or near stairways to prevent tipping or damage.
- Ensure the work area is clean, well-lit, and free from unnecessary hazards.

2. Circuit Overview

The following schematic illustrates the electrical connections for the movement module. It shows how the microcontroller (ATmega328P) interfaces with motor drivers (DRV8871), DC motors, ultrasonic sensors (HC-SR04), and the power supply module (LM2596). This diagram serves as a reference for debugging and maintenance of the wiring and logic.



3. Tools Required

For performing maintenance on the movement module, the following tools are recommended:

- Phillips screwdriver (medium and small size)
- Multimeter for voltage and continuity testing
- Soldering iron (for repairing broken joints)
- Hot glue gun (for reattaching loose components)
- Small pliers and wire cutter
- USB-to-Serial cable (for reprogramming the microcontroller)
- Cleaning cloth and isopropyl alcohol (for sensor maintenance)

4. Routine Maintenance

To ensure long-term performance and reliability, follow the recommended routine maintenance schedule below:

Task	Frequency	Notes
Check motor and driver connections	Weekly	Ensure no wires are loose or frayed
Clean ultrasonic sensor surfaces	Weekly	Use a dry cloth or alcohol if needed
Inspect wheels and motor mounts	Biweekly	Tighten any loose screws
Charge or replace battery	As needed	Avoid full discharge
Run alpha test script	Monthly	Verify motor and sensor functionality
Check structural stability	Monthly	Ensure platforms and columns are secure

5. Power System

The robot's power system consists of a LiPo battery connected to an LM2596 voltage regulator. The regulator ensures a stable 5V output for powering the microcontroller and sensors. Maintenance tasks include:

- Regularly inspect battery terminals for corrosion or loose contacts.

- Ensure the voltage regulator heatsink is not obstructed.

- Monitor battery charge cycles and avoid deep discharges.

- Replace the battery if swelling or overheating occurs.

6. Motors and Drivers

The movement module uses three JGB37-520 DC motors paired with DRV8871 H-bridge motor drivers. The motors are mounted beneath the base platform, with the drivers installed above them. To maintain optimal performance:

- Inspect motor shafts and ensure smooth wheel rotation.

- Tighten all mounting screws securing motors and drivers.

- Check for burnt smell or overheating in drivers, indicating overcurrent issues.

- Clean dust from motor vents and ensure proper ventilation.

7. Sensors and camera

The robot uses HC-SR04 ultrasonic sensors for obstacle detection. There are five sensors distributed around the base. To ensure reliable measurements:

- Keep sensor surfaces clean and free from dust or smudges.

- Test each sensor regularly using diagnostic scripts.

- Verify that the sensors are properly mounted and oriented.

- Check for stable electrical connections to the microcontroller.

A camera module is installed on the movement module to enable AprilTag recognition, particularly for docking with the charging station. To maintain reliable visual tracking:

- Clean the camera lens periodically with a microfiber cloth.

- Ensure the camera is firmly mounted and correctly aligned.

- Verify camera communication with the microcontroller during testing.

- Inspect the QR markers and replace if damaged or worn out.

8. Structure and Exterior

The robot's body is made of laser-cut wooden pieces forming two main platforms connected by vertical columns. During regular maintenance:

- Inspect wooden panels for cracks, bending, or wear.

- Tighten screws and connections between structural components.

- Check the stability of the vertical supports and platforms.

- Ensure fabric covers or aesthetic elements are clean and not obstructing components.

9. Software and Testing Tools

The system includes test routines for each subsystem and a full integrated control program. For software-related maintenance:

- Use alpha test scripts to validate motors, sensors, and camera functionality.

- Monitor serial output for status messages or error reports.

- Re-upload firmware in case of persistent malfunction or code updates.

- Document and track software changes to maintain consistency across updates.

10. Error Diagnosis and Correction

To ensure the long-term reliability and maintainability of the robot, a structured troubleshooting methodology was established. This process helps identify and correct system malfunctions related to movement, obstacle detection, and docking behavior.

1. Identify the Faulty Function:

Begin by determining which core function is failing:

- **Movement** (motors not responding, erratic motion)
- **Obstacle Avoidance** (ultrasonic sensors unresponsive)
- **Charging Station Detection** (robot fails to locate or approach the station)

2. Analyze Potential Causes:

Each issue can stem from several potential sources:

- **Movement:** Faulty motor, driver, microcontroller output, electrical connection, or control logic
- **Obstacle Avoidance:** Ultrasonic sensor failure, logic error, or wiring issue
- **Charging Station Detection:** Camera malfunction, incorrect software handling, or communication failure

3. Component-Specific Checks:

Use the following tests to isolate the failing component:

- **Motor:** Connect it to a known working controller and driver; verify bidirectional rotation.

- **Motor Driver:** Bridge a known controller and motor; test for correct actuation.
- **Microcontroller:** Upload a test sketch to verify PWM or digital output using a voltmeter.
- **Electrical Connections:** Use a multimeter to ensure continuity and voltage delivery across all power and signal lines.
- **Ultrasonic Sensors:** Attach to a known working microcontroller and verify accurate distance readings.
- **Camera:** Connect to a computer or processing unit and ensure live data stream and correct detection behavior.
- **Code:** If all hardware tests pass, the issue likely lies in the software logic. Re-examine the handling of relevant components.

4. Corrective Action:

Once the faulty component is identified:

- Replace any defective hardware.
- Repair or rewire faulty connections.
- Revise and upload corrected firmware if a software error is found.

11. Mantainance and Log Template

Date	**Performed By**	**Task Description**	**Components Affected**	**Issues Found**	**Action Taken**	**Next Check**
YYYY-MM-DD	Name	Example: Sensor cleaning	HC-SR04 Ultrasonic	None	Cleaned all sensors	YYYY-MM-DD