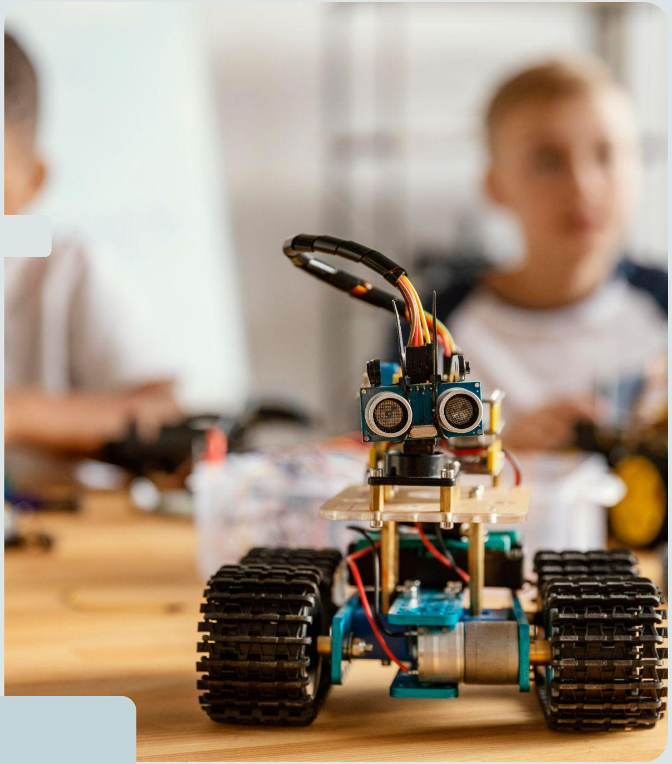
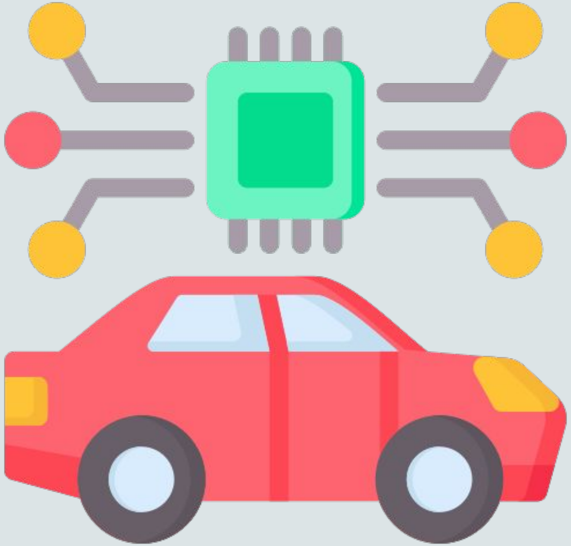
Lab of Iot Presentation

4WD IoT Smart Car



Lab of Iot Project by Luigi Consiglio and Jihad Taoubi

Table of contents



01

Introduction

Now we present an overview of our smart car

Introduction

The **SmartCar 4WD** is an Arduino-based vehicle designed for multiple functionalities, including following paths, avoiding obstacles, and responding to remote controls.

Thanks to our **upgrades**, it now monitors real-time temperature and humidity, making it perfect for exploration and environmental monitoring.

*Powerful and versatile the **ultimate** companion for innovative tech projects!*

Smart car functionality

Obstacle avoidance

It detects and avoids obstacles by changing direction automatically

Remote control

The car can be driven remotely using a handheld controller

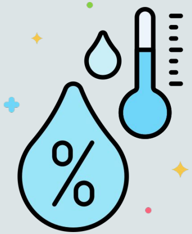
Line tracking

The car follows a black line on a predefined path using light sensors

Smart car functionality

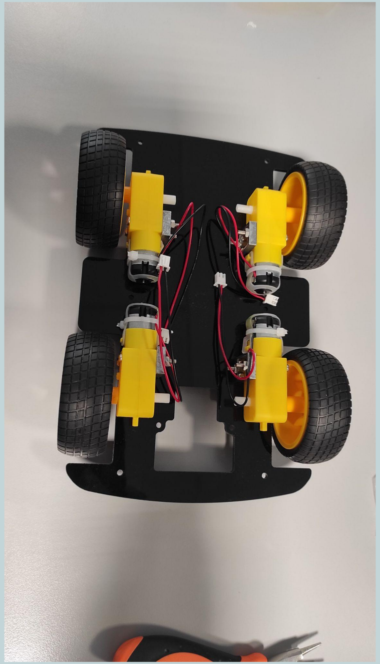
Environmental Monitoring

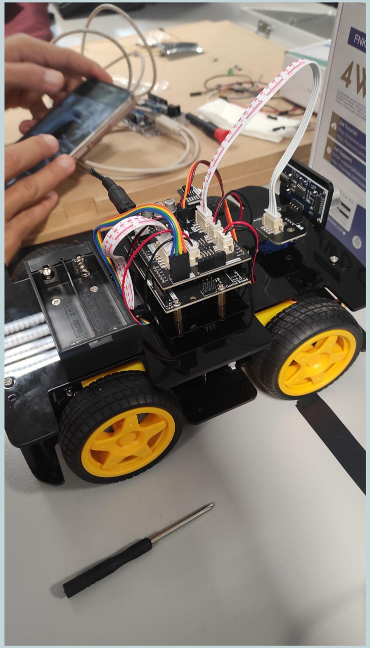
The car can measure and display real-time temperature and humidity levels of its surrounding

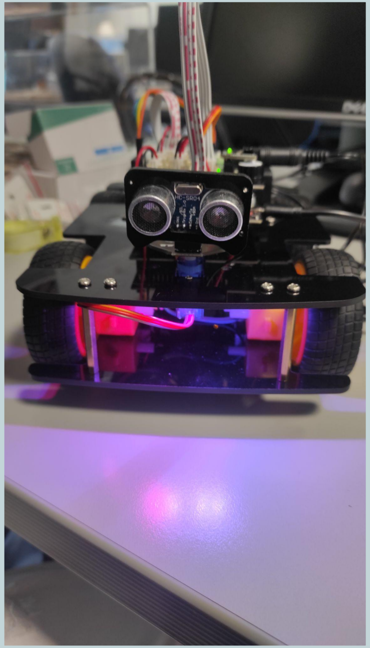


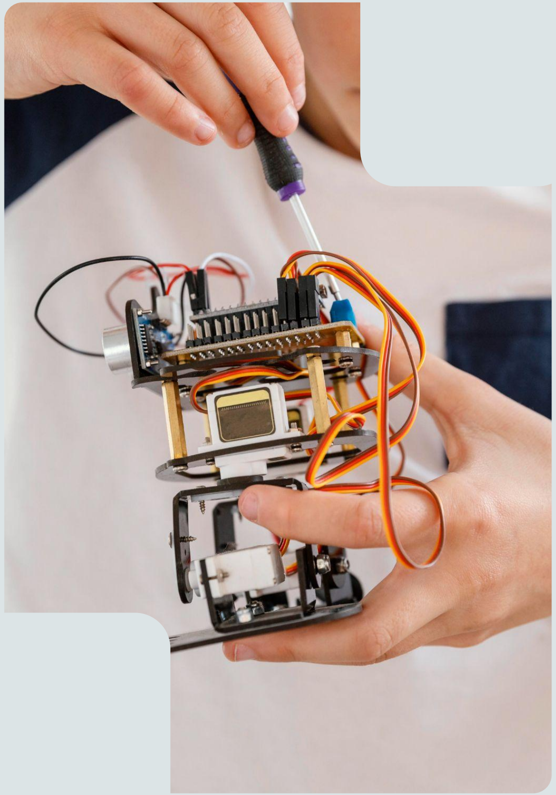
First of all

We assembled the machine





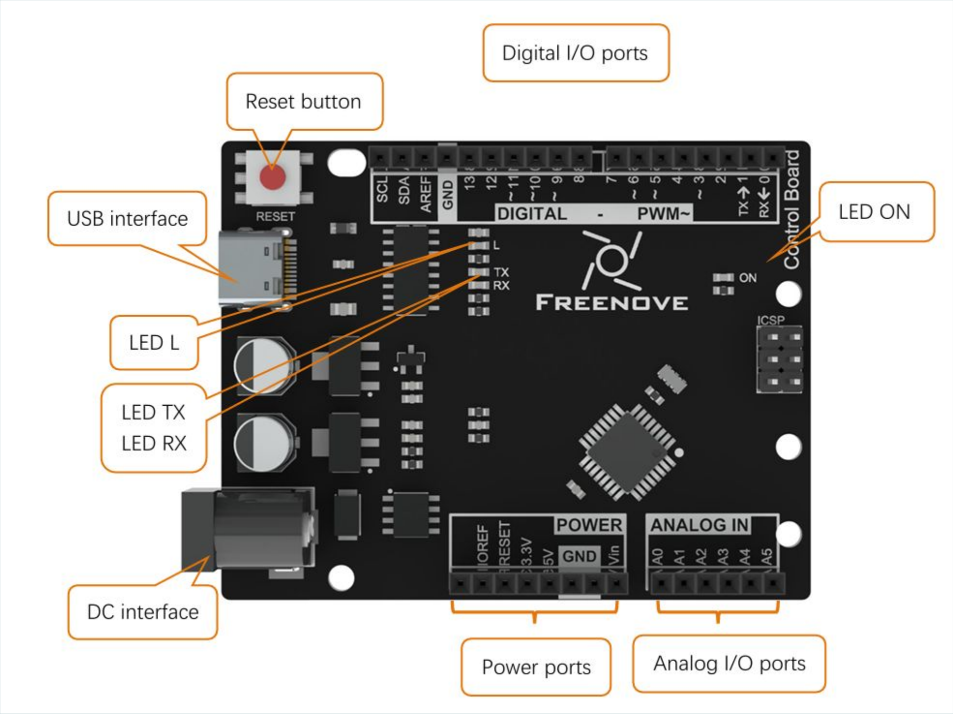




02

Tools

Boards



Control Board

The Arduino board features digital **I/O** pins and analog pins used for reading variable signals.

The **USB** interface allows for power supply and communication with the PC.

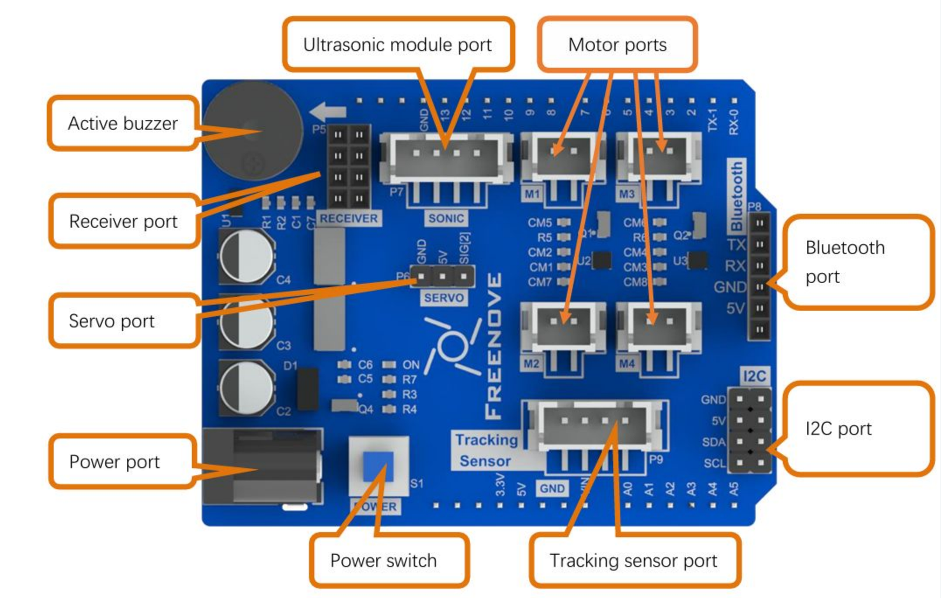
LEDs are included to indicate the status of serial communication, power (ON), and pin D13.

Boards

Extension board

This is an expansion board for robotic projects.

It includes **dedicated ports** for **sensors and modules**: one for the ultrasonic module, four motor ports (M1-M4) to control locomotion, and a connector for line-tracking sensor.



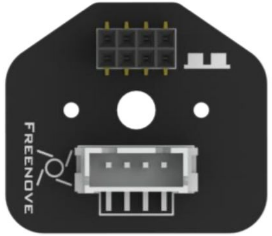
Sensors used

Line-tracking infrared sensor

Line -tracking sensor

Sensor that emit and detect infrared light to identify black and white areas

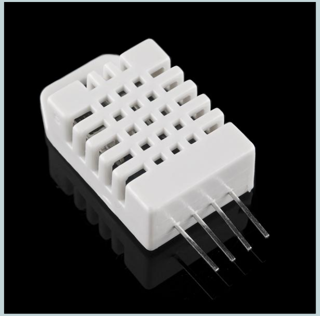
Ultrasonic module connector



Ultrasonic module

Sensor that measure distance by emitting ultrasonic waves and detecting the reflection

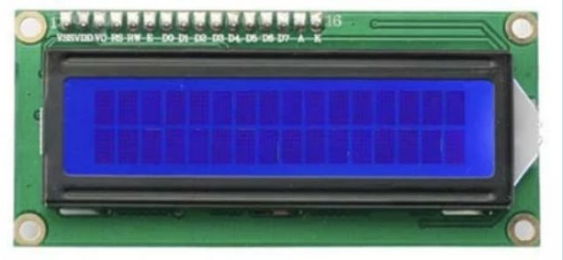
Other sensor



Temperature and Humidity sensor

A sensor that measures ambient temperature and humidity levels

Other tools

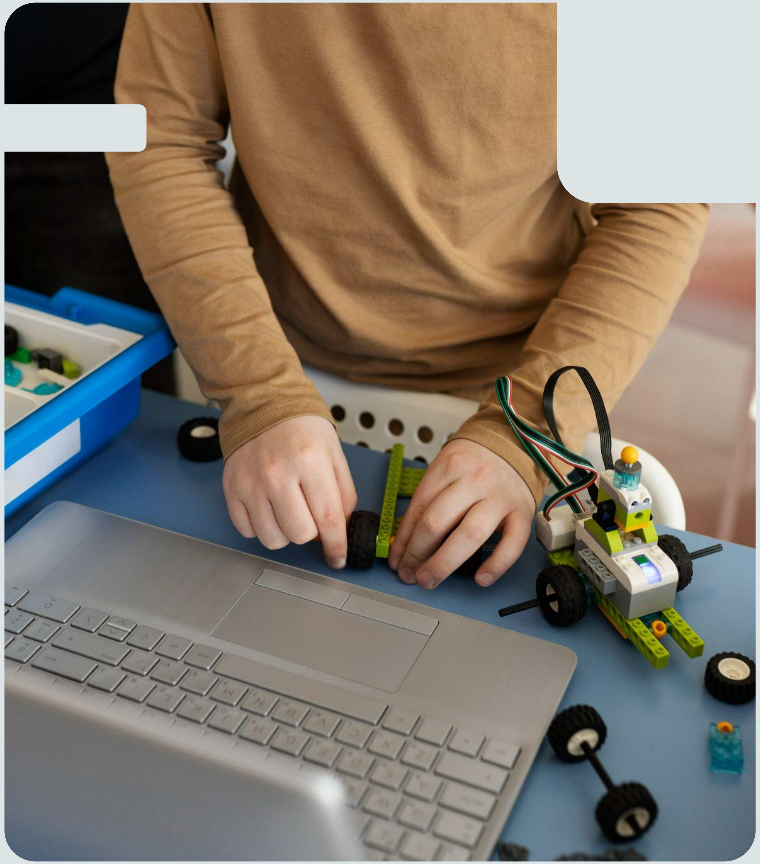


LCD Display Module

A small screen used to display data, such as temperature and humidity

Remote Control and IR Receiver

A remote with an infrared receiver to send and receive commands



03

Code

Tracking line code

```
void loop() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = getTrackingSensorVal(); //get sensor value

  switch (trackingSensorVal)
  {
    case 0:   //000
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED); //car move forward
      break;
    case 7:   //111
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED); //car stop
      break;
    case 1:   //001
      motorRun(TK_TURN_SPEED_LV4, TK_TURN_SPEED_LV1); //car turn
      break;
    case 3:   //011
      motorRun(TK_TURN_SPEED_LV3, TK_TURN_SPEED_LV2); //car turn right
      break;
    case 2:   //010
    case 5:   //101
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);  //car move forward
      break;
```

Read data from the tracking sensors and decide how to control the motors, allowing the car to follow a line.
The logic is based on the combined value of the 3 sensors (L,C,R), which indicate the position of the line relative to the car.

```
void motorRun(int speedl, int speedr) {
  int dirL = 0, dirR = 0;
  //I due blocchi servono a controllare
  if (speedl > 0) {
    dirL = 0 ^ MOTOR_DIRECTION;
  } else {
    dirL = 1 ^ MOTOR_DIRECTION;
    speedl = -speedl;
  }

  if (speedr > 0) {
    dirR = 1 ^ MOTOR_DIRECTION;
  } else {
    dirR = 0 ^ MOTOR_DIRECTION;
    speedr = -speedr;
  }
```

Control the car's motors, specifying the direction and speed for each motor (L & R).
Values can be positive or negative: Positive for forward rotation; Negative for reverse rotation.

Obstacle Avoidance Logic

```
void updateAutomaticObstacleAvoidance() {
  int distance[3], tempDistance[3][5], sumDisntance;
  static u8 leftToRight = 0, servoAngle = 0, lastServoAngle = 0;  //
  const u8 scanAngle[2][3] = { {150, 90, 30}, {30, 90, 150} };

  for (int i = 0; i < 3; i++)
  {
    servoAngle = scanAngle[leftToRight][i];
    servo.write(servoAngle);
    if (lastServoAngle != servoAngle) {
      delay(130);
    }
    lastServoAngle = servoAngle;
    for (int j = 0; j < 5; j++) {
      tempDistance[i][j] = getSonar();
      delayMicroseconds(2 * SONIC_TIMEOUT);
      sumDisntance += tempDistance[i][j];
    }
    if (leftToRight == 0) {
      distance[i] = sumDisntance / 5;
    }
    else {
      distance[2 - i] = sumDisntance / 5;
```

This is the main function controlling the obstacle avoidance system. It makes the car scan the area in three directions: left, center, and right.

The `scanAngle` array tells the servo where to point (150° for left, 90° for center, 30° for right).

Based on the distance readings, the car decides to turn left, right, or move forward

To improve accuracy, the system takes five measurements at each angle and calculates the average.

Obstacle Avoidance Logic

This function handles the ultrasonic sensor, which is the 'eyes' of the system.

The sensor sends a sound pulse, then measures how long it takes for the echo to return. This time is used to calculate the distance.

If no echo is detected, the function assigns a maximum distance to indicate no obstacle is nearby.

Ensures reliable obstacle detection, which is essential for the car to navigate effectively

Ultrasonic Sensor Function

```
float getSonar() {
  unsigned long pingTime;
  float distance;
  digitalWrite(PIN_SONIC_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_SONIC_TRIG, LOW);
  pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT);
  if (pingTime != 0)
    distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000;
  else
    distance = MAX_DISTANCE;
  return distance; // return the distance value
}
```

MQTT code

1) Reads data (temperature and humidity).

2) Publishes the data to the MQTT broker on specific topics (temp, umid).

3) The MQTT broker (broker.emqx.io): Receives messages from the device. Forwards the messages to the Node-RED client subscribed to those topics.

```
const char broker[] = "broker.emqx.io";
int port = 1883;
const char topic1[] = "temp";
const char topic2[] = "umid";
const char* ssid = "WiFi-LabIoT";
const char* password = "s1jzsjkw5b";
```

Output

```
Connected to WiFi!
Attempting to connect to the MQTT broker on localhost: broker.emqx.io
Connected to MQTT broker on localhost
```

```
Publishing temperature: 22
Publishing humidity: 58
Publishing temperature: 23
Publishing humidity: 74
Publishing temperature: 23
Publishing humidity: 81
Publishing temperature: 24
Publishing humidity: 65
```

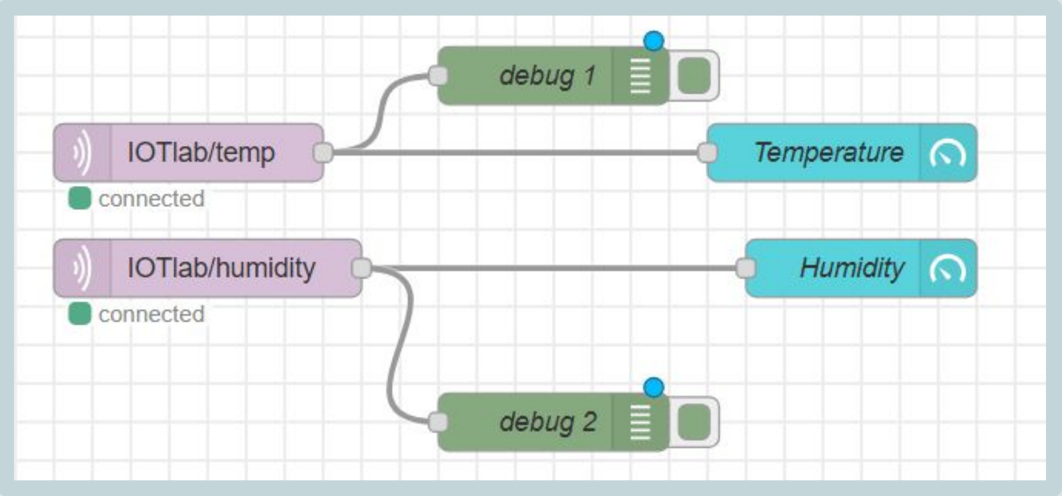
04
Web Application

Web Application

To open Node Red : node-red command in cmd
And then go to: http://127.0.0.1:1880

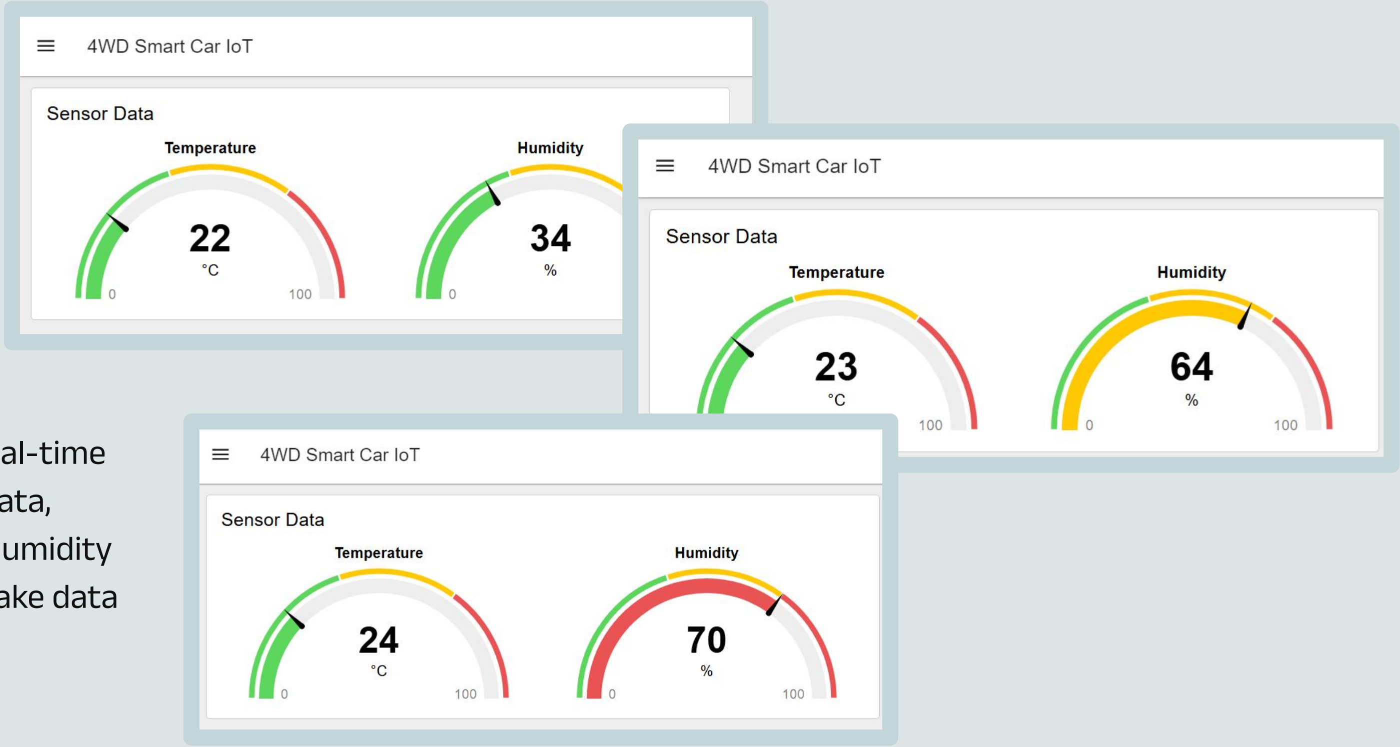
Flow

To build our web-app, we used Node Red tool that help us to easily connect devices, process data, and create dashboards using a simple interface.



Web Application

Design

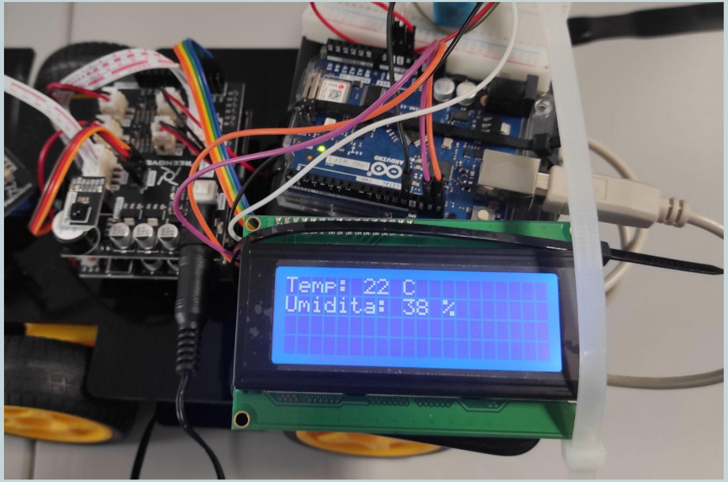


The dashboard provides a real-time visualization of the sensor data, including temperature and humidity readings. It is designed to make data monitoring easy .

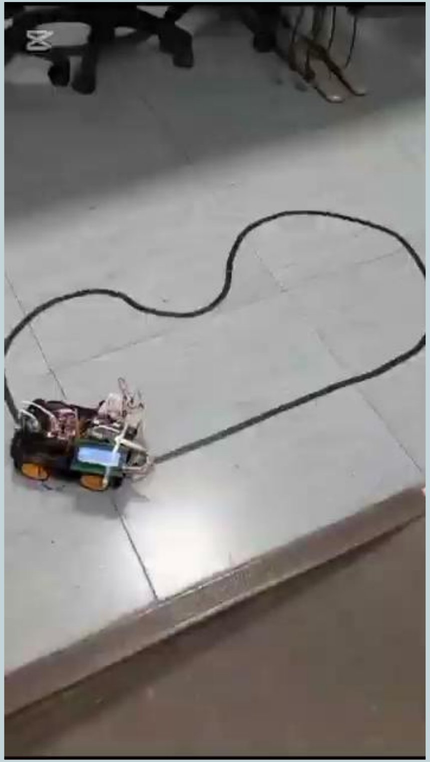
05

The lab results

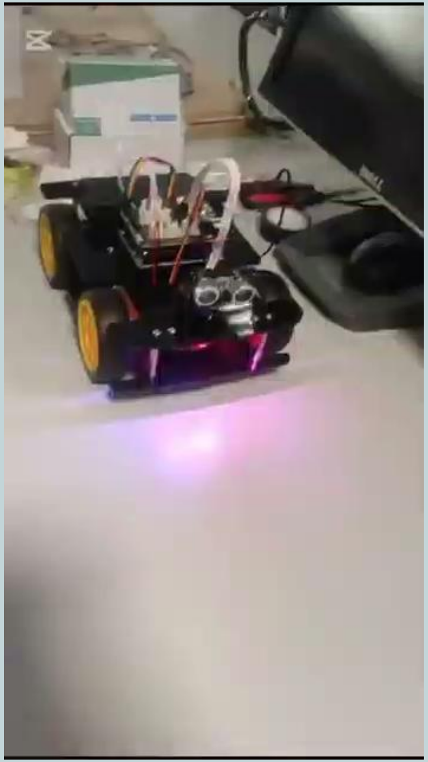
Get temperature and humidity



Follow the path



Remote control



Obstacles avoidance

Thanks!

Luigi Consiglio (0522501894)
Jihad Taoubi (0522501938)