Introduction

SCRAP is a modular software platform, specifically designed for the development of autonomous robotic systems. Its distributed and scalable architecture allows it to handle a wide range of complexities, from automating simple household tasks.

System Architecture

SCRAP is structured into three main modules, each with a well-defined role:

- SCRAP.Android: Acts as the user interface, providing an intuitive API for human interaction. The Android app includes speech-to-text (STT) functionality for more natural interaction and a graphical user interface for viewing the robot's status and configuring parameters.
- SCRAP.Net: The computational heart of the platform. It implements:
    - Motion planning: Uses advanced pathfinding algorithms (A*) to generate optimal paths in dynamic and static environments, considering constraints of space, time, and energy.
    - Environment interaction: Handles perception of the environment through sensors (LiDAR, cameras, etc.) and physical interaction with it through actuators (motors, grippers, etc.).
    - Communications: Uses communication protocols (WebSocket, HTTP) to exchange data with other modules and external systems.
    - LLM interface: Integrates large language models to interpret natural language commands and generate complex actions.
- SCRAP.Arduino: The module dedicated to interfacing with the robot's hardware. Manages communication with microcontrollers (typically Arduino) and connected sensors/actuators. Provides a simplified API to control hardware devices and acquire data from sensors.

Command Execution Flow

1. User request: The user sends a command through the Android interface.
2. Command processing: SCRAP.Net receives the command and analyzes it.
3. Planning: The actions necessary to execute the command are planned, taking into account the current state of the robot and the surrounding environment.
4. Execution: Commands are sent to SCRAP.Arduino, which executes them directly on the hardware.
5. Feedback: The robot sends feedback on the execution status to SCRAP.Net, which in turn communicates it to the Android interface.

Environment Mapping and Navigation

SCRAP.Net uses algorithms to build detailed maps of the environment in real time. Maps are used for:

- Localization: Determining the robot's exact position in space.
- Navigation: Planning safe and efficient paths, avoiding static and dynamic obstacles.
- Odometry: Estimating the robot's displacement based on sensor readings.

Hardware Emulator

The emulator integrated into SCRAP.Net allows you to simulate the behavior of the robot in a virtual environment. This feature is essential for developing, debugging, and testing new algorithms and functionalities, without the need for a physical robot.

Technologies Used

- SCRAP.Net: C# and .NET Core for portability and performance.
- SCRAP.Android: Kotlin for Android app development.
- SCRAP.Arduino: C++ for programming Arduino microcontrollers.
- Communications: WebSocket, HTTP, serial.
- Motion planning: A*.

Getting Started with SCRAP.Net

This guide will walk you through the process of setting up SCRAP.Net on your local machine. SCRAP.Net is a project that simulates robot communication with Arduino using HTTP protocol.

Prerequisites

Before you begin, ensure you have the following installed:

- Git
- Visual Studio 2022 (or newer) with the .NET workload installed
- .NET Core 7.0
- eSpeak

Installation Steps

1. Clone the Repository

1. Open a terminal or command prompt.
2. Navigate to the directory where you want to save the project.
3. Run the following command:

```
git clone https://github.com/fra00/SCRAP.Net.git [your_path]
```

Replace `[your_path]` with your desired directory path.

2. Open the Solution

1. Launch Visual Studio 2022 with administrator privileges.
2. Go to "File" -> "Open" -> "Project/Solution".
3. Navigate to the cloned repository and open the file: `[your_path]\SCRAP.Net\Robot\Robot.sln`

3. Configure MainRobot Project

Edit Configuration.cs

1. Open `Configuration.cs` in the MainRobot project.
2. Set the full path to eSpeak:

```
public static string PATH_ESPEAK = "C:\\Program Files (x86)\\eSpeak\\command_line\\";
```

3. Modify eSpeak TTS configuration arguments:

```
public static string ARGUMENTS_ESPEAK = "-v it -p 20 \"[@textToSpeach]\"";
```

Note: `-v it` changes the language to Italian. Remove it for English.

4. Set the Arduino device IP (for emulator, leave as is):

```
public static string HTTP_URL_COMUNICATION = "http://192.168.1.238/?cmdData=";
```

5. Enable hardware simulation and remote Arduino simulation:

```
public static bool FAKE_HW = true;
public static bool FAKE_REMOTE_ARDUINO = true;
```

Edit RobotConfiguration.cs

1. Open `RobotConfiguration.cs` in the MainRobot project.
2. Specify the path to load the map:

```
public static string MAP_FILE_NAME = "img//mappaMuri.png";
```

3. Set the maximum dimensions of the floor plan in cm:

```
public static int WIDHT_MAP = 1000;
public static int HEIGHT_MAP = 1000;
```

4. Set the robot's dimensions in cm:

```
public static int HALF_WIDTH_ROBOT = 15;
public static int HALF_HEIGHT_ROBOT = 15;
```

5. Leave the minimum movement size unchanged:

```
public static int MIN_STEP_FOR_FINDPATH = 10;
```

Creating the Map

The map is a crucial element that you, as the user, need to create. It's used to support the lidar and sensors, and helps speed up pathfinding. Here's how to create and configure the map:

Map Creation Guidelines

1. **Format**: The map should be a PNG image in black and white.
2. **Content**:
   - Black areas represent walls or obstacles.
   - White areas represent the floor or navigable space.
3. **Dimensions**:
   - The image size should be calculated based on the `WIDHT_MAP` and `HEIGHT_MAP` properties in `RobotConfiguration.cs`.
   - In the current configuration, one pixel corresponds to one centimeter.
4. **Creation Tool**: You can use any graphic editor to create this map.

Steps to Create and Configure the Map

1. Create a black and white PNG image representing your environment. Ensure the dimensions match your `WIDHT_MAP` and `HEIGHT_MAP` settings.

2. Save the map file in your project directory, typically in an `img` folder.

3. Update the `MAP_FILE_NAME` in `RobotConfiguration.cs`:

   ```
   public static string MAP_FILE_NAME = "img//yourMapFileName.png";
   ```

4. Ensure the `WIDHT_MAP` and `HEIGHT_MAP` properties in `RobotConfiguration.cs` match your map's dimensions in centimeters:

   ```
   public static int WIDHT_MAP = 1000; // if your map is 1000 pixels wide
   public static int HEIGHT_MAP = 1000; // if your map is 1000 pixels tall
   ```

5. Adjust the robot's dimensions if necessary:

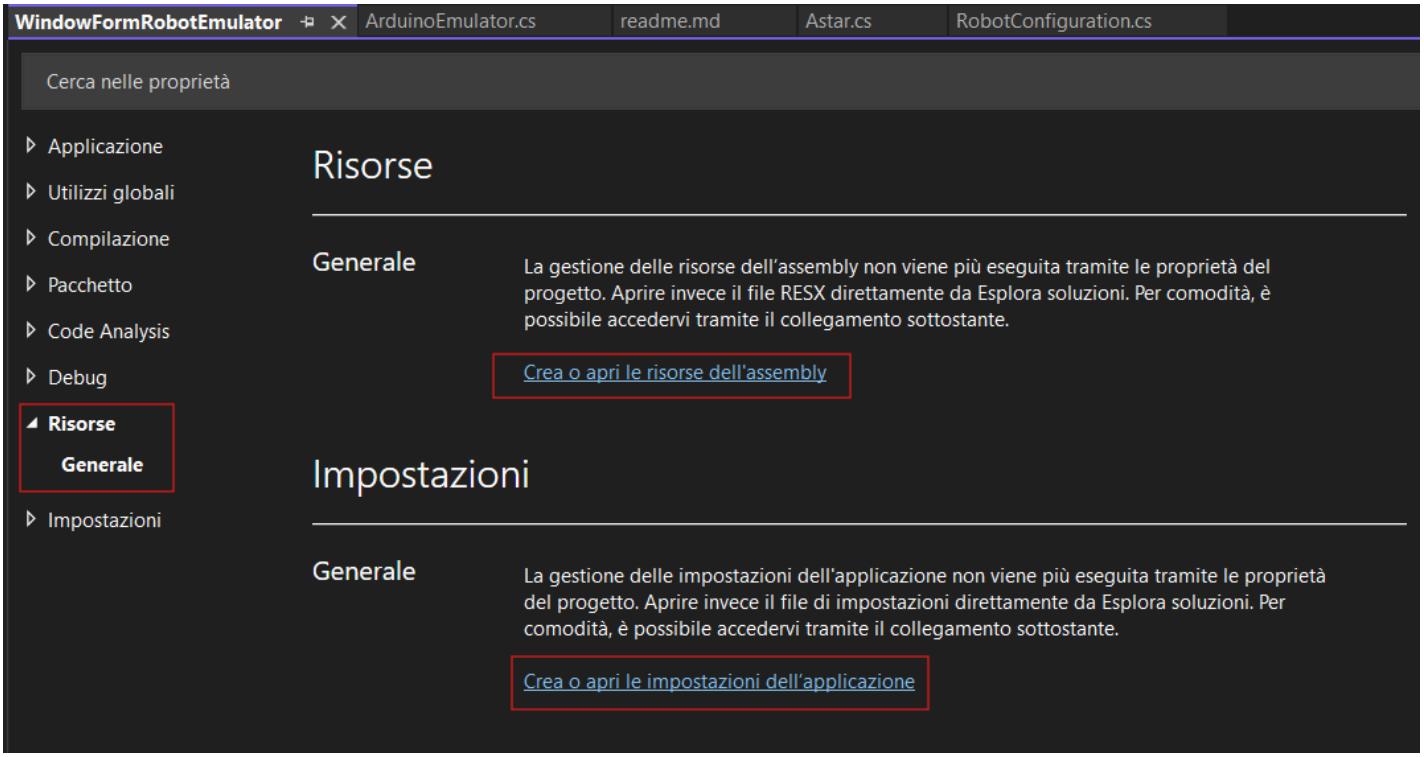
   ```
   public static int HALF_WIDTH_ROBOT = 15;
   public static int HALF_HEIGHT_ROBOT = 15;
   ```

Remember, the accuracy of your map will directly impact the performance of the pathfinding algorithm and the overall simulation.

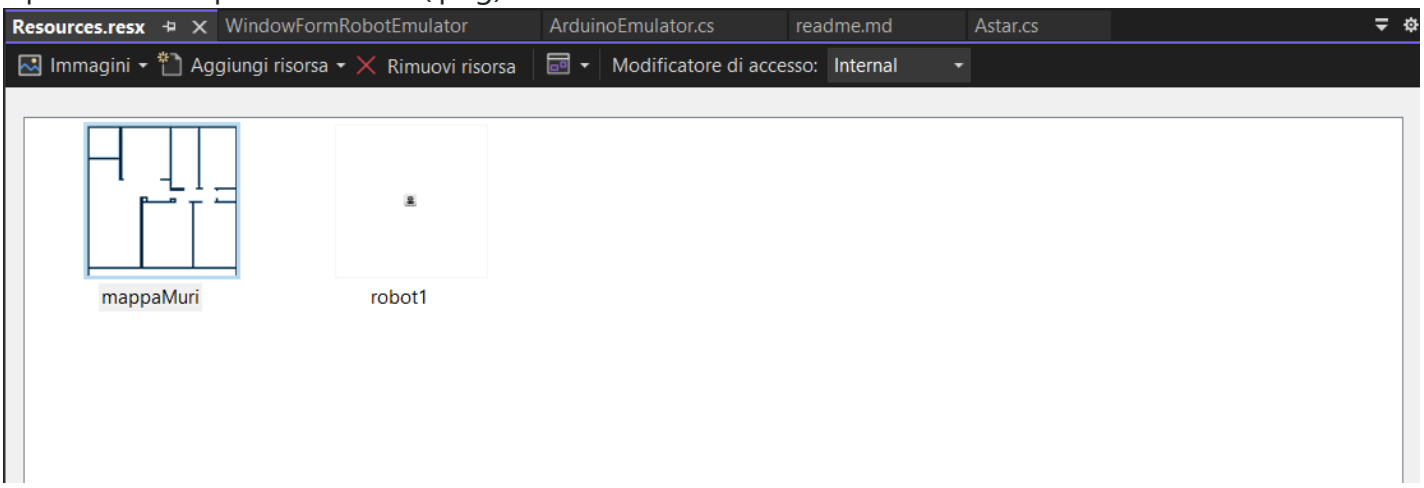
Configure the Emulator

1. Right-click on the WindowFormRobotEmulator project and select "Properties".

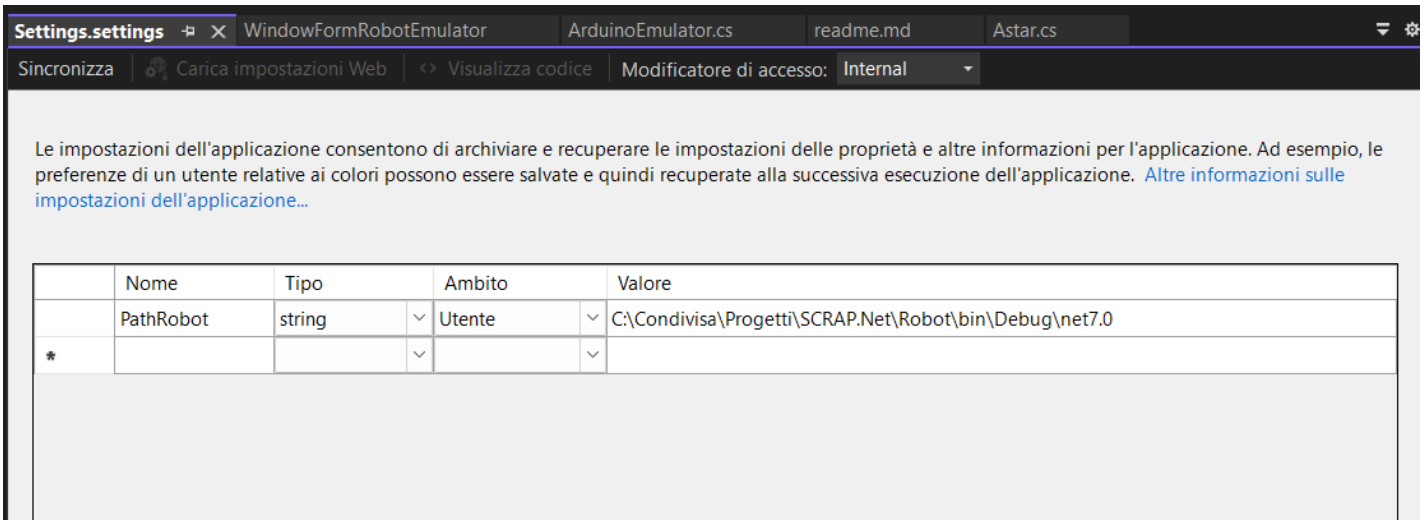
2. Go to the "Resources" section.



3. Update the map file resource (.png).



4. Modify the path where SCRAP.NET is executed (should be the bin folder of the MainRobot project).



Run the Projects

1. Right-click on WindowFormRobotEmulator -> Debug -> Start New Instance
2. Right-click on MainRobot -> Debug -> Start New Instance

Verification

If everything is set up correctly:

1. A shell should open, executing a series of commands.
2. The emulator should launch.

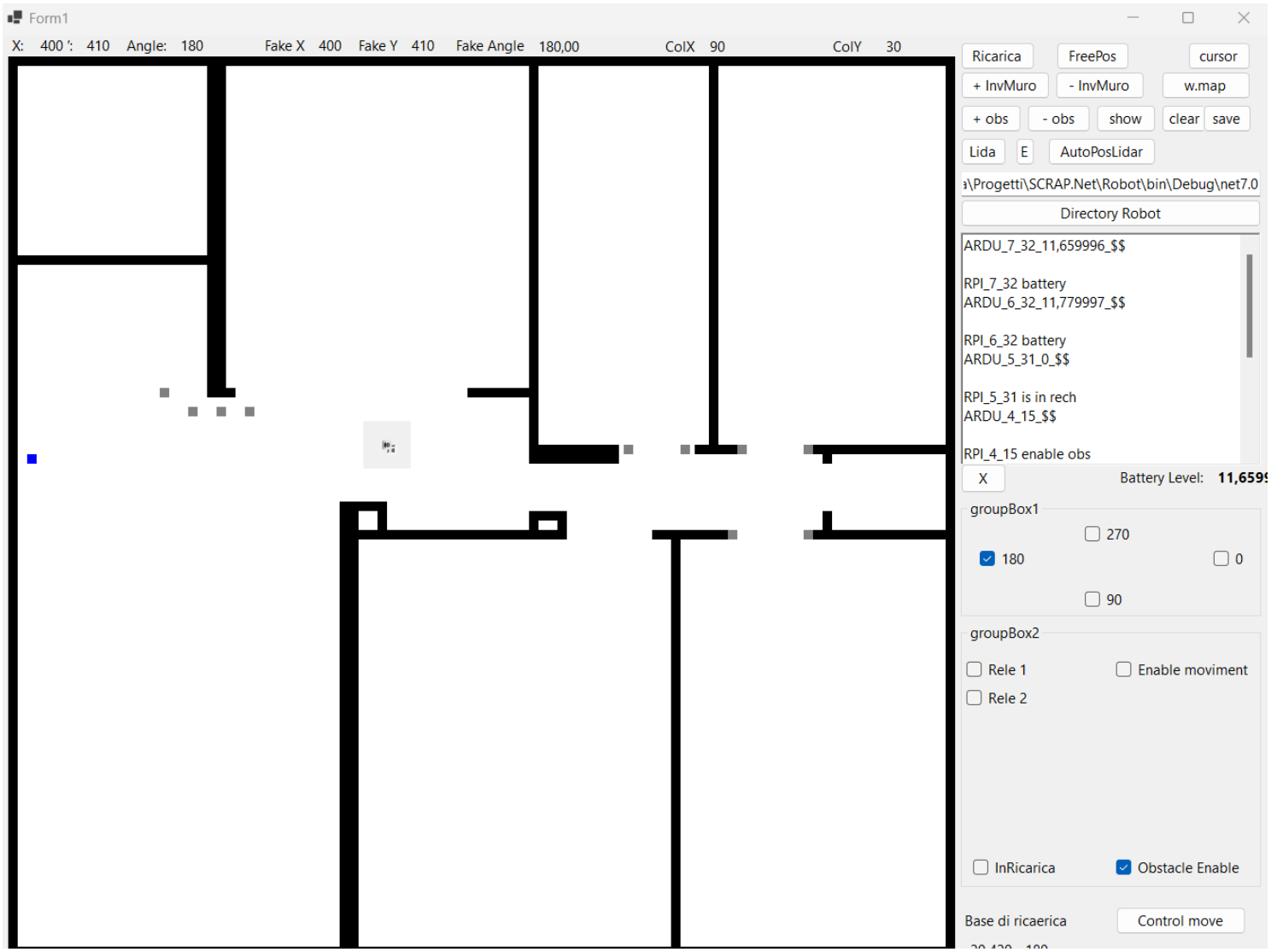
You should see a console window with various commands being executed, and the emulator interface should appear.

```
Command runned in  (1004)
************************************************
send command for EnableObstacleFind
HttpComunication RunCommand exec RPI_4_15 EnableObstacleFind
File responseCommand.json was modified
HttpComunication RunCommand result ARDU_4_15_$$

Command runned in  (987)
************************************************
send command for get source of alimentation
HttpComunication RunCommand exec RPI_5_31 get source of alimentation
File responseCommand.json was modified
HttpComunication RunCommand result ARDU_5_31_0_$$

Command runned in  (985)
Listening...
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Condivisa\Progetti\SCRAP.Net\Robot\bin\Debug\net7.0
************************************************
send command for GetLevelOfAlimentation
HttpComunication RunCommand exec RPI_6_32 GetLevelOfAlimentation
File responseCommand.json was modified
HttpComunication RunCommand result ARDU_6_32_11,779997_$$

Command runned in  (1126)
```



SCRAP.Net Emulator Guide

Overview

The SCRAP.Net emulator is an experimental tool designed to simulate the hardware and asynchronous communication of the robot. It provides a way to test and develop SCRAP.Net functionalities without physical hardware.

Important Notes

- The emulator is experimental and contains test features.
- Not all functionalities are guaranteed to work perfectly.

Configuration

To use the emulator, you need to modify the `Configuration.cs` file in the MainRobot project:

```
public static bool FAKE_HW = true;
public static bool FAKE_REMOTE_ARDUINO = true;`
```

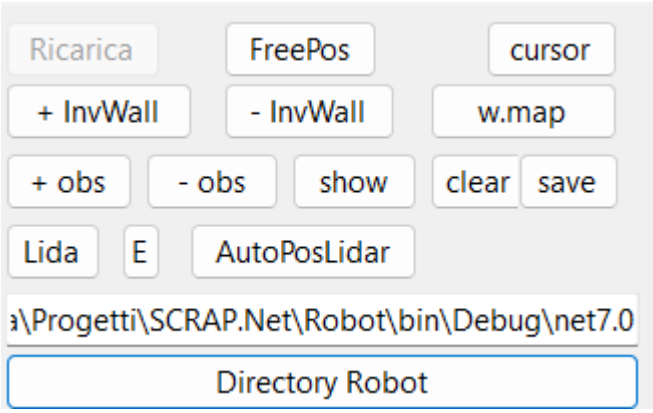
These settings enable the use of fake communication classes for the emulator.

How It Works

1. When a command is sent, the fake communication class writes it to a file (`command.json`).
2. The emulator detects the file change, parses, and executes the fake command.
3. Once completed, the emulator writes the response to `responseCommand.json`.
4. The system reads the response and proceeds with the next command.

Note: In debug mode, the main folder is `.\SCRAP.Net\Robot\bin\Debug\net7.0`

Emulator Interface



Control Buttons

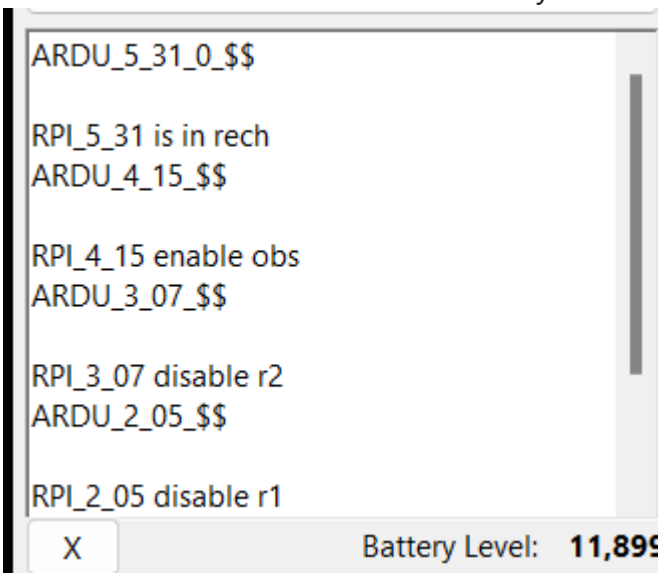
- **FreePos**: Position the robot within the map
- **Cursor**: When clicked, simulates movement to the clicked point on the map
- **+InvWall / -InvWall**: Add/remove "invisible wall" points as obstacles
- **+obs / -obs**: Add/remove temporary obstacles (for testing)
- **show**: Display obstacles encountered by the robot
- **clear**: Remove all temporary obstacles
- **save**: Save obstacles added to the map via clicks

Other Features

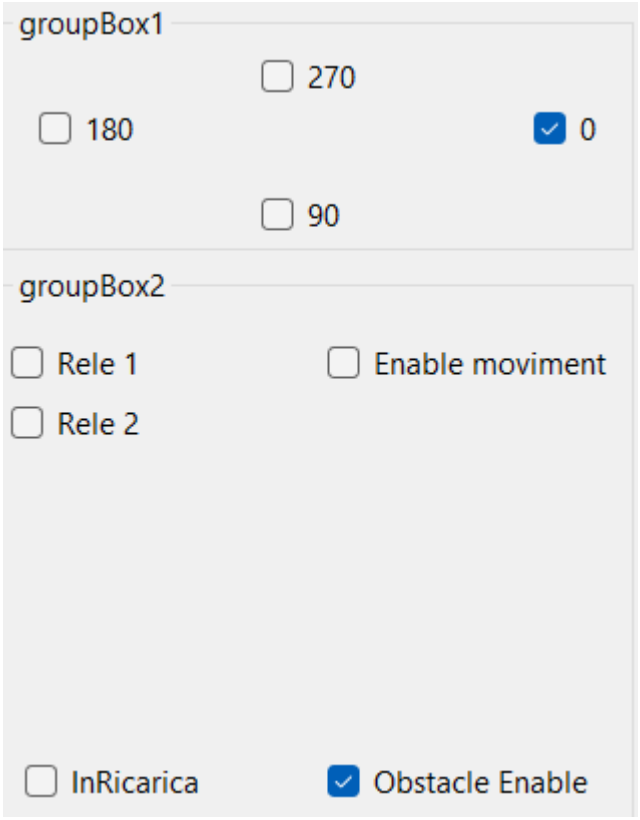
- **w.map**: Visualize the weight of detected obstacles in grayscale (best with lidar)

- **Lidar and AutoPosLidar**: Experimental Lidar functionality
- **Robot Directory button**: Modify the directory where SCRAP.Net runs

- **Command Textbox**: Shows the history of commands received from SCRAP.NET



- **Robot Position and Status**: Displays current robot information



- **"Control Move" Button**: Opens a form for manually moving the robot

Experimental Features

Some buttons and features (like Lidar) are experimental and not final. Use these with caution and expect potential changes or issues.

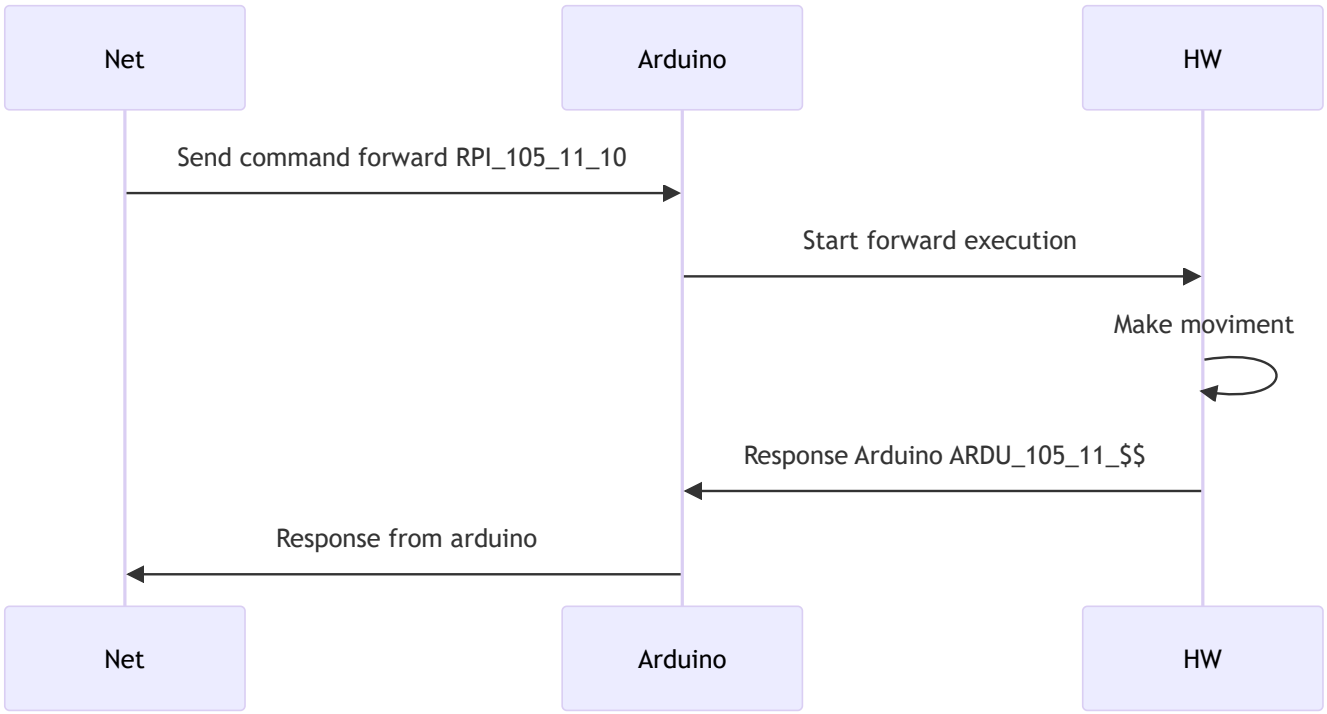
SCRAP.Net and Arduino Communication Guide

Overview

SCRAP.Net is the main program that interfaces with the hardware. It breaks down user requests (e.g., "go to the kitchen") into a series of commands executed by Arduino. Arduino receives these commands, interfaces with the hardware, and once the command is completed, it responds to SCRAP.Net. This process continues until the initial task is completed.

Communication Flow

Here's an example of how a command to move forward 10 cm is sent and processed:



Message Format

The data transmission format is structured as follows:

```
[sender]_[id]_[IdCommand]_[param1]_[param2]_[param3]_[param4]_[end]
```

Example:

```
RPI_1_11_10_&&
```

The data transmission format was selected to provide compatibility with various protocols, including serial and HTTP.

Components:

- **[sender]**: Message sender (RPI for SCRAP.Net, ARDU for Arduino)
- **[id]**: Auto-incremental command identifier
- **[IdCommand]**: ID of the command to send
- **[param1...4]**: Optional parameters (arrays are split based on SERIAL_ARRAY_SEPARETOR)
- **[end]**: End of message string

Configuration

These parameters can be configured in the `Configuration.cs` file:

```
public static string SERIAL_SEPARETOR = "_";
public static string SERIAL_ARRAY_SEPARETOR = ";";
public static string SERIAL_START_MESSAGE_RPI = "RPI";
public static string SERIAL_START_MESSAGE_ARDU = "ARDU";
public static string SERIAL_END_MESSAGE = "$$\r\n";`
```

`

IMPORTANT: The same configurations must be implemented in SCRAP.Arduino and must match, otherwise communication will fail.

Command Implementation

In SCRAP.Net, all Arduino commands are implemented and declared in the `CommandComunication` class of the `MainRobot.Robot.Comunication` namespace.

List Command

Method Name	**Parameter 1**
DisableRele1	05
EnableRele1	06
DisableRele2	07
EnableRele2	08
EnableFakeMoviment	09
Stop	10
Forward	11{Configuration.SERIAL_SEPARETOR}{distance}

Method Name	Parameter 1
StartLeftMotor	12{Configuration.SERIAL_SEPARETOR}{angle}
StartRightMotor	13{Configuration.SERIAL_SEPARETOR}{angle}
Backward	16{Configuration.SERIAL_SEPARETOR}{distance}
EnableMoviment	21
DisableMoviment	22
GetLevelOfAlimentation	32
ReadLidar	33
DisableObstacleFind	14
EnableObstacleFind	15
IsInRecharge	17
OutInRecharge	18
GetDistanceFrontSensor	30
GetSourceOfAlimentation	31
MoveServo	40{Configuration.SERIAL_SEPARETOR}{numServo} {Configuration.SERIAL_SEPARETOR}{angle}

1. Always ensure that the configuration in SCRAP.Net and SCRAP.Arduino match.
2. Use meaningful and unique command IDs to avoid confusion.
3.