National School of Advanced Sciences and Technologies of Borj Cédria

Advanced Electronics and Nanotechnology

ECOLE IBN AL-HAYTHAM

Technician internship report

Advanced C++ SumoRobot Programming on Embedded Linux

Realised by :

JBELI Walid

Supervisor : BEN KHLIFA Raed

Academic year 2024/2025

Reference: STG TECH24 EAN 15

Contents

List of Figures

List of Tables

Abstract

This internship report details the project I completed during my industrial placement at *Ibh Web Consulting*.This project's goal was to design and create an embedded Linux-based sumo robot with a C programming language. Within this system, there are two primary components that interact: the software algorithms that enable autonomous decision-making, and the hardware platform made up of sensors and actuators.Our autonomous sumo robot is built on the synergy between these two parts, which allows it to recognize opponents and effectively navigate the competition of sumo robots.

Keywords : Sumo robot, competition, embedded Linux, C programming, autonomous operation.

Acknowledgements

I am incredibly grateful and happy to dedicate this page to everyone who helped to make my project a reality. I want to thank each and every one of you from my heartfelt. This initiative has been a huge success because of your help.

My sincere gratitude is extended to **Raed BEN KHLIFA**, my internship supervisor, whose assistance was crucial in finishing this small project. His professionalism and kindness made the office a trusting and productive place to work.

I also warmly thank my brother-in-law, **Ala-Eddine NOUALI**, for his invaluable support throughout the writing of my report. His wise advice and expertise were a great help in refining this document.

My deepest gratitude goes out to my software team, which includes Wassim and Soumaya, and our hardware group, which includes Zeineb, Louay, and Hamza. Their cooperation was essential to the project's success.

Finally, I hope this work meets the expectations and requirements for which it was designed.

Abbreviations List

IT: Information Technology

HP: Hewlett Packard

IR: Infrared

PWM: Pulse Width Modulation

PIC Peripheral Interface Controller

IC: Integrated Circuit

DC: Direct Current

PCB: Printed Circuit Board

RPM: Revolutions Per Minute

RAM: Random Access Memory

UML: Unified Modeling Language

ARM: Advanced RISC Machine

LTS: Long Term Support

GCC: GNU Compiler Collection

IDE: Integrated Development Environments

CCS: Code Composer Studio

TI: Texas Instruments

VsCode: Visual Studio Code

GDB: GNU Debugger

eabi: Embedded Application Binary Interface

JTAG: Joint Test Action Group

USB: Universal Serial Bus

DFU: Device Firmware Upgrade

RTOS: Real-Time Operating Systems

ELF: Excecutable and Linkable Format

LED: Light Emitting Diode

LiPo: Lithium Polymer

MOSFET: Metal-Oxide-Semiconductor Field-Effect Transistor

GPIO: General-Purpose Input/Output

I2C: Inter-Integrated Circuit

ADC: Analog-to-Digital Converter

UART: Universal Asynchronous Receiver-Transmitter

MCU: Microcontroller Unit

IO: Input/Output

PC: Program Counter

SP: Stack Pointer

IVT: Interrupt Vector Table

ISR: Interrupt Service Routine

CPU: Central Processing Unit

RX: Receive

TX: Transmit

NEC: Nippon Electric Company

DMA: Direct Memory Access

ACLK: Auxiliary Clock

SDA: Serial Data Line

SCL: Serial Clock Line

RTC: Real-Time Clocks

EEPROM: Electrically Erasable Programmable Read-Only Memory

VCC: Voltage Common Collector

General introduction

With recent technological advancements, the ongoing digital revolution is significantly influencing almost every aspect of our daily lives.In response, the embedded systems industry has implemented fresh approaches to stay up to date with these developments. This emerging technology is increasingly recognized for its potential to establish innovative and customized interactions between users and machines, even at a global scale.

Internships, meanwhile, play a crucial role in our training. They offer direct exposure, allowing engineers and technicians to choose the best approach for developing and optimizing these systems.

In this context, we chose to complete our internship at *Ibh Web Consulting*, a company specializing in embedded technology services.

During this advanced internship, our host organization assigned us the task of designing a sumo robot programmed in C, operating in an embedded Linux environment.

This report details the different stages of the project's development, divided into three sections:

- **Section 1 : General context of the project.** This section presents an overview of the creation of the Sumo Robot and begins with an introduction to Ibh Web Consulting and its embedded systems expertise. The research will examine models that are currently in use in Tunisia and beyond, with particular emphasis on the Sumo Robot competition. It discusses the current issues and offers the solutions, outlining the robot's purpose and construction.

- **Section 2 : ARM Cross-Compilation in Embedded Linux environment** In this section, we covers the process of ARM cross-compilation within an embedded Linux environment.An overview of Linux and its development environment settings is included.The topic of programming in this setting is then covered, along with the selection of a programming language and the standard organization of embedded projects. After discussing the fundamentals of cross-compilation, toolchain configuration, and executable production, the focus switches to ARM cross-compilation for the MSP430G2553 microcontroller. The section concludes with deployment and testing on the MSP430G2553, including flashing the executable and debugging that follows.

- **Section 3 : Hardware and Software Tools** This section describes the hardware and software tools used in the project.It begins with Hardware Tools, detailing the electronic hardware design and components.It then covers Software Tools, including KiCad for PCB design, PlantUML for UML diagramming, and Box2D for physics simulation.

- **Section 4 : Project Realisation** In this section, we details the implementation of the project.It starts with software architecture, outlining the application and driver layers. Next,it covers programming GPIO, outlining its purpose and setup for the project. The implementation of UART drivers and microcontroller interrupts are then covered with the NEC protocol driver and PWM motor control.Additionally discussed are peripheral drivers, with an emphasis on ADC with DMA and I2C drivers. The part concludes with a description of coding the state machine and simulation to real-world demo, which goes into details into the creation of state machine and how they go from simulation to real-world presentation.

I. General context of the project

This section will start with a brief overview of the development company behind the project. Next, we'll analyze the Sumo robot industry and the robotics sector. After reviewing existing research, we'll assess the current state,present our model as a strong solution to existing challenges, and describe our project methodology.

A. Presentation of the development company

Ibh Web Consulting is a development company founded in 2015 in Kelibia. It is distinct as a company that specializes in software development, custom solutions, C++ robot creation, Linux application design, embedded service management, and energy optimization. The company's main goal is to increase client businesses' visibility while improving their traffic and financial performance.

Figure 1: Logo of Ibh Web Consulting

B. *Ibh Web Consulting* services

Ibh Web Consulting offers a range of comprehensive and customized professional services.It involves digital development, robotic application design, and social media administration. Furthermore, the organization is always accessible to its customers to guarantee follow-up, maintenance, and continuous advancement of its solutions. These services include:

- **Information Technology Development**: It provides excellent IT solutions that are customized to meet the unique requirements of people or companies wishing to incorporate technology into their daily operations in order to improve and simplify them.

- **Programmed robots creation** : The company's web-based agency is committed to helping its customers create and develop highly intelligent programmed robots. Throughout the process, they provide knowledgeable direction to guarantee the development of creative and effective robotic solutions that significantly improve their operations.

- **Embedded Systems Management** : Ibh Web Consulting offers complete embedded service management to its clients. These service ensures secure and dependable operation of the embedded systems and it is available around-the-clock once the required files and configurations are set up.

C. Project presentation

1. Overall context

The project we worked on during our worker internship will be presented at the National School of Advanced Sciences and Technologies of Borj Cedria (ENSTAB) and includes our contribution. The objective is to meet the internship supervisor's requirements by using the theoretical information that was previously learned through the implementation of a project approach.

2. General context of the Sumo Robot competition

Description

A sumo robot is an autonomous robot designed for competitive sumo-style matches.The main objective of a sumo robot is to push or force its opponent out of a circular ring called the "platform". It have a strong, compact construction, and they frequently have ultrasonic or infrared sensors to identify other robots and the edges of the ring. The sumo robot, which is outfitted with motors and actuators, moves tactically to participate in both offensive and defensive moves throughout the bout.We can illustrate the setup of a sumo robot tournament, as depicted in Figure 2.

Figure 2: Sumo Robot Tournament [1]

Competition rules

- **Requirements**: The sumo robot must weigh no more than 500 g and measure 10 by 10 cm. It should be autonomous and equipped with a remote start feature.
- **Warnings** : The robot must not damage the platform or the other robot. Furthermore, using materials like glue that stick to the platform is not allowed.
- **Platform**: Each round must be finished in less than three minutes, and the competition platform has a diameter of 77 cm.

D. Study of existence

1. Tunisian Sumo Robots models

Zumo Robot for Arduino

This Zumo robot, designed by the Ubuy company, is a compact, low-profile tracked platform designed for using an A-Star 32U4 Prime as a compatible device. The robot is outfitted with a pair of 75:1 HP micro metal gear-motors, which provide substantial torque and enable it to attain top speeds of 2 feet per second (60 cm/s). The Zumo has a robust 0.036-inch-thick laser-cut stainless steel sumo blade up front that is ideal for pushing items or engaging in robot combat.The robot is perfect for jobs involving line following and edge avoidance since it has an array of six IR reflecting sensors below this blade that allow it to identify lines and edges on the ground. The high-torque motors of the Zumo are effectively powered by its integrated DRV8835 dual motor drivers and a piezo buzzer that is managed by an Arduino PWM (Pulse Width Modulation) outputs, which reduces the need for processing power when the robot plays sounds, as shown in Figure 3.

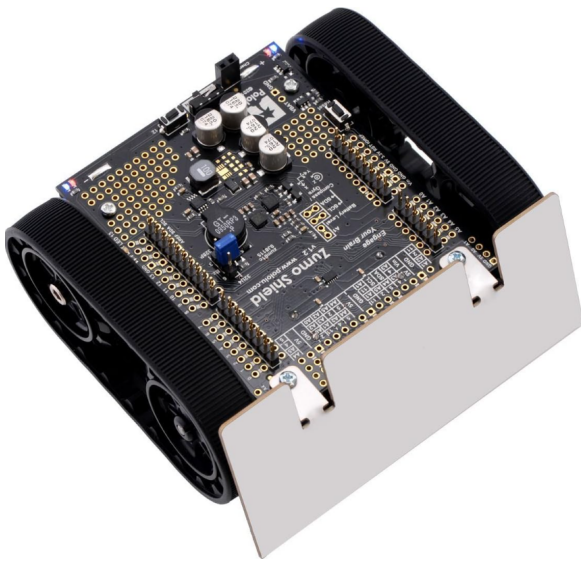

Figure 3: Zumo Robot for Arduino [2]

PIC Super Sumo Robot

The PIC Super Sumo robot,presented by the Ubuy Company, is designed for dynamic control, allowing it to attack or retreat during battle with an opponent. It has a PIC microcon-

troller, which can be reprogrammed, enabling users to reprogram the integrated circuit (IC) to run new applications.

The robot has a control board that is 2.54 x 2.60 inches and a sensor board that measures 2.54 x 1.18 inches. Its power usage is 80 mA, as illustrated in Figure 4.

Figure 4: PIC Super Sumo Robot [3]

2. Sumo Robot Models Abroad

Kit Robot Sumo Midi JSumo

The Kit Robot Sumo Midi JSumo is presented by the RobotShop Canadian company. It is a robot sumo compact designed for weight categories under 1 kg. This fully assembled kit has five unfavorable sensors for best performance, powerful engines with a 25 mm diameter, high-friction wheels, and more. The robot has four motors, seven sensors (two line sensors and five adverse MR45 sensors), an XMotion controller, a LiPo 3S 1000 mAh battery, and a specially made chassis. The electronic wheels (39 mm x 11 mm) are attached directly to the engines, which are rated at 12 V and have the capacity to withstand an overload to 18 V.The set also includes silicone roues for improved traction, as well as spacers,screws, and nuts.The robot model is shown in Figure 5.

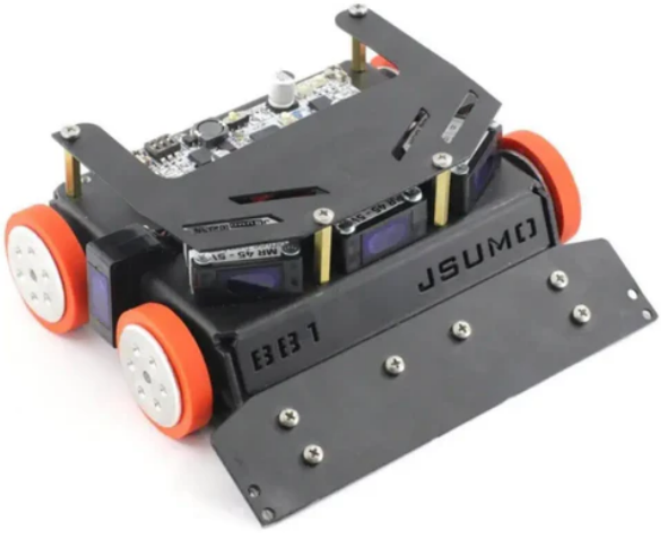

Figure 5: Kit Robot Sumo Midi JSumo [4]

Mini Sumo Robot Kit

This robot kit is exposed by the Robotpark American firm. It is using 9 v batteries.A 16F628A microprocessor and an L293D motor driver are used by the robot. MicroC is used to write the robot's program. The robot weighs 300 grams when assembled. Its measurements are 96 x 100 x 75 mm.It includes the Rocket Veer 4.0 Basic Robotics Card , two QTR1A contrast sensors, three MZ80 object sensors, two Micro DC reduction motors (6V, 400 RPM), two nitro wheels, four plastic clamps, and it consists seven Plexiglas parts, four standoffs, and bolts,as illustrated in Figure 6.

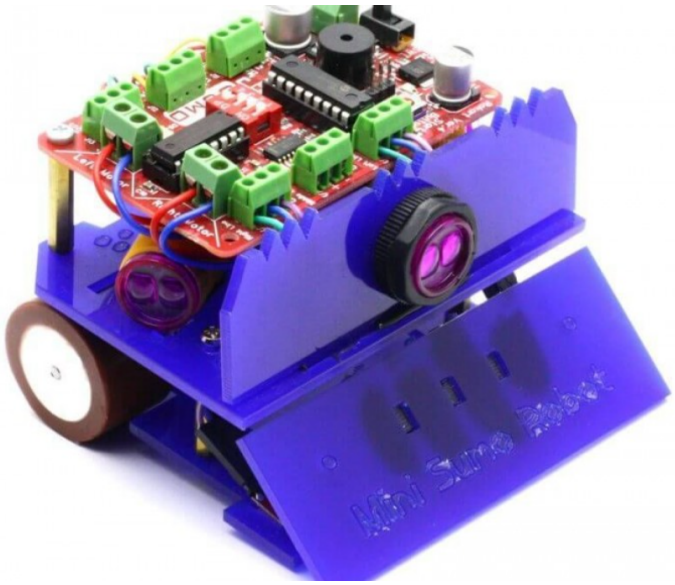

Figure 6: Mini Sumo Robot Kit [5]

E. Problem statement

The Tunisian sumo robots models have some throwbacks such as:

- The zumo robot uses 75:1 HP micro metal gear-motors, which, while powerful, might not offer the higher torque and speed. It employs DRV8835 motor drivers and an Arduino, which can be less efficient and offer less protection for the motors.

- The PIC Super Sumo robot relies on the PIC microcontroller, though re-programmable, might not offer the necessary performance and efficiency.Additionally, its power consumption of 80 mA could impact battery life and overall performance during extended matches.

Also the sumo robots models aboard have some limitations such as:

- the Kit Robot Sumo Midi JSumo have a high power motors which can make the robot overly aggressive, potentially leading to less precise maneuverability during matches.Additionally, the robot's reliance on a high-capacity 3S battery can lead to increased weight and reduced agility.Lastly,The sensors, while numerous, include MR45 sensors that may not offer better resolution and reliability.

- Mini Sumo Robot Kit features DC reduction motors and an L293D motor driver, which offer less efficient speed control and protection. The 16F628A microcontroller also has more limited memory and processing power.

F. Proposed solution

1. Our mission

Our goal is to create a sumo robot that solves the main flaws in current models, hence surpassing their constraints. Our robot has sophisticated brushless DC motors with gearboxes to provide better performance than the Zumo robot, which uses 75:1 HP micro metal gear-motors that might not offer the best torque-to-speed ratio. Compared to the DRV8835 drivers and Arduino setup, we can guarantee improved motor protection and more accurate speed control by utilizing TB6612FNG motor drivers. Our robot uses the MSP430G2553 microprocessor to maximize power consumption and improve overall efficiency, in contrast to the PIC Super Sumo robot, which might have problems with performance and efficiency because of its microcontroller and power consumption. With a 6V power source and lightweight LiPo batteries for increased agility, our design prioritizes balanced performance over the Kit Robot Sumo Midi JSumo, which may be overly aggressive due to its high-power motors and limited maneuverability caused by its large battery. Finally, we incorporate high-resolution VL53LOX infrared sensors to overcome the limitations of competing robots' sensors, guaranteeing higher accuracy and dependability for precise navigation and a competitive edge.

2. Our Sumo Robot model

The sumo robot's design includes a printed circuit board (PCB) to connect small components in a compact and reliable manner.It is powered by two brushless DC motors with gearboxes, offering 400 RPM and a top speed of 2 m/s. Motor speed is regulated by TB6612FNG drivers using

PWM signals. For range and line detection, the robot employs five VL53LOX infrared sensors and four line sensors. An IR receiver captures the competition start signal at 38 kHz. Power is provided by two 3.7V, 750 mAh LiPo batteries, with control managed by an MSP430G2553 microcontroller featuring a 16-bit processor, 16 MHz frequency, 16KB flash memory, and 0.5KB RAM.Our sumo robot is shown on the Figure 7.

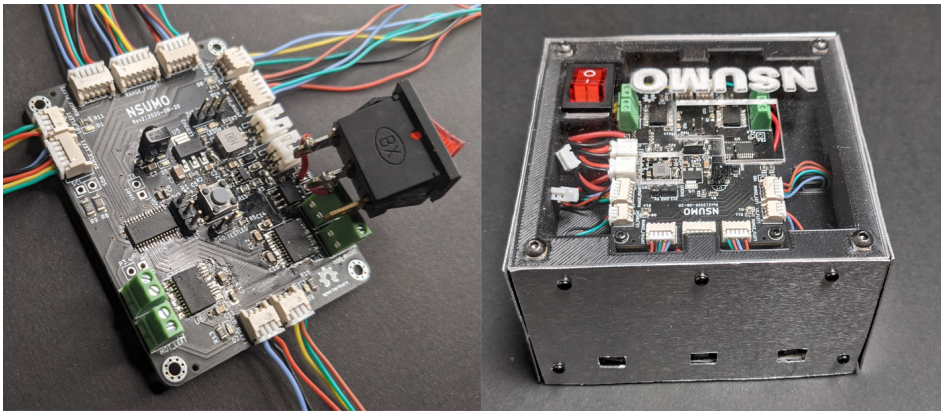

Figure 7: Our Sumo Robot model

G. Project Management Methodology

1. Life cycle using Scrum Methodology

To assure the project's success within the time frame, we used the Scrum methodology, an agile framework that encourages iterative and incremental work organization. The project was divided into two groups: the first concentrated on the sumo robot's hardware, and the second focused on the software.

Each Scrum sprint enabled both groups to work well together, ensuring continuous communication and rapid response to changing project needs. The advantages of this technique are:

- **Continuous and adaptive planning**: With each sprint, we could review and alter priorities based on the project's urgent needs.
- **Integrated testing phases**: Testing was carried out at the end of each sprint to confirm that the developed features fulfilled the required criteria.
- **Clear role definitions**: Each team member had a well-defined role, which allowed for an efficient division of responsibilities.

The Figure 8 describes precisely this project methodology:

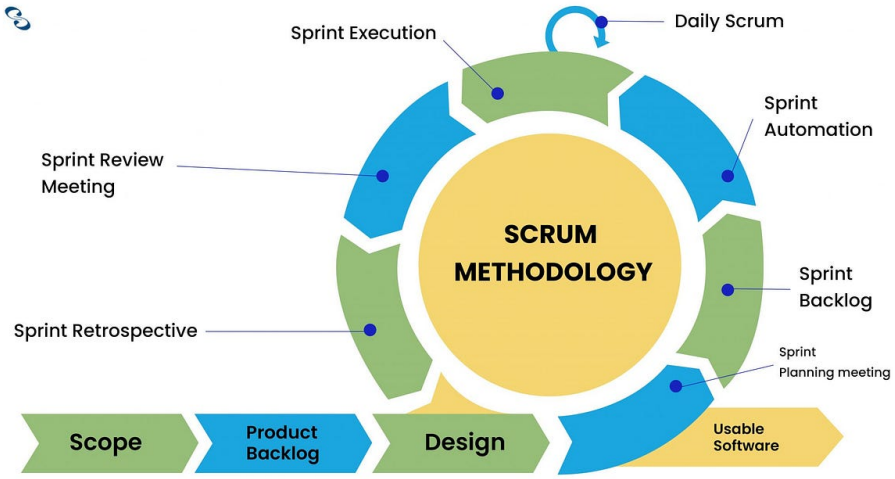

Figure 8: scrum [6]

2. Modeling UML Language

We selected to model our application in UML (Unified Modeling Language) using the PlantUML platform. This powerful modeling language serves as an effective communication tool, making it easier to understand representations and solutions. We may represent object-oriented solutions in an intuitive way using UML. It facilitates the comparison, evaluation, specification, development, and documenting of software artifacts in a complex system. Furthermore, it allows us to map software architecture, create solutions, and communicate business information.

Conclusion

In conclusion, we provide a quick overview of our company, its offerings, the environment in which our project was created, and our goal, which focused on addressing the limitations of existing models and proposing a solution to enhance the performance of a sumo robot. In the next chapter, we will describe our embedded Linux environment and the programming work we undertook, and demonstrate the enhancements we made to cross-compile for the ARM architecture on our MSP430G2553 microcontroller.

II. ARM Cross-Compilation in Embedded Linux environment

This chapter delves into the Embedded Linux Environment, with an emphasis on the application of the MSP430G2553 microcontroller as the ARM-based system. We'll cover the set up the development environment, which programming languages to use, and how to cross-compile, which is necessary for embedded applications.

A. Introduction to the Embedded Linux Environment

1. Overview of the Linux Environment

Open-source and highly adaptable, Linux is known for its efficiency, security, and dependability. Since its inception in 1991, when Linus Torvalds began developing it, Linux has matured into a robust platform that is compatible with a variety of hardware architectures, making it a popular option for a wide range of applications, such as embedded systems, desktops, and servers. Because of its modular design, which enables users to tailor the system to their own requirements, it is the perfect option for developers and engineers in a variety of fields.
Using Linux in embedded devices has a number of benefits:

- **Cost-effective and Open Source**: Because Linux is open-source, there are no license fees, enabling developers to freely access, alter, and share the code. This adaptability is especially useful for embedded systems, where customisation is frequently necessary.

- **Robustness and Stability**: Linux is renowned for its dependability and uninterrupted operation over an extended period of time. This is important because stability and up time are frequently vital in embedded systems.

- **Security**: Linux is a secure option for embedded applications because of its robust security features, which include user permissions, access control, and frequent security upgrades.

- **Flexibility and scalability**: Linux may be expanded to handle complicated systems or shrunk down to run on very little hardware. It is appropriate for a variety of embedded devices, ranging from tiny Internet of Things devices to intricate industrial machinery, due to its scalability.

- **Huge Community and Ecosystem Support**: Linux is fortunate to have a sizable developer community as well as a sizable ecosystem of libraries, tools, and documentation. The embedded system development process is sped up by this support network, which makes development and troubleshooting easier.

- **Long-Term Support**: Long-term support (LTS) versions are available for many Linux distributions. These versions are updated and maintained for lengthy periods of time, giving embedded devices used in the field stability and security.

2. Configuration of the Development Environment

An appropriately configured development environment is necessary for writing Linux software for embedded systems. A collection of tools that have been carefully selected to facilitate the creation, compilation, debugging, and implementation of embedded applications are usually included in this environment. Commonly installed in a development environment based on Linux are the following tools:

Compilers

- **GCC (GNU Compiler Collection)**:It is one of the most popular compilers.Numerous programming languages are supported, such as C and C++, which are frequently utilized in embedded systems. Thanks to GCC's cross-compilation features, such as "arm-none-eabi-gcc", programmers can create executables for other architectures (like ARM) and build code on one (like x86).

- **Clang**:It is a component of the LLVM project that is a GCC substitute that offers quick compilation times, modularity, and thorough error reports. Like GCC, it is appropriate for embedded system development and enables cross-compilation.

Editors and Integrated Development Environments (IDEs)

- **Code Composer Studio (CCS)**:This editor was created especially for Texas Instruments (TI) microcontrollers and embedded processors, including the MSP430 series. It comes with a number of tools for debugging, compiling, and editing code. CCS is an effective tool for creating, testing, and implementing embedded applications in a streamlined and effective manner since it supports a wide range of embedded processors, including ARM-based devices.

- **Visual Studio Code**: VS Code is an extensible lightweight editor that supports a large number of programming languages. Because of its versatility and extensive plugin library, it is frequently utilized in embedded development.

- **Eclipse**: Another well-known IDE, Eclipse allows for a great deal of extensibility and supports a wide range of programming languages with plugins. For embedded development, the GCC toolchain is frequently utilized in tandem with it.

Build Tools

- **Make**: The compilation process is managed by this build automation tool. It specifies how various project components should be connected and compiled using a Makefile. In embedded programming, where projects frequently contain numerous source files and dependencies, make is very crucial.

- **CMake**: A cross-platform build system generator, CMake is used to manage the build process. It creates native build scripts that provide a more adaptable and portable build environment, such as Makefiles or project files for IDEs like Visual Studio.

Debugging Tools

- **GDB (GNU Debugger)**: GDB is the standard debugger for GNU systems.It enables developers to step through the code to find and correct issues, analyze variables, trigger breakpoints, and watch as their programs run. In embedded development, where hardware limitations might make debugging more difficult, GDB is indispensable.

- **Valgrind**:Valgrind is a valuable tool suite that includes memory debugging, memory leak detection, and profiling. Since embedded systems frequently have restricted memory resources, it helps ensure the dependability and efficiency of these applications.

Along with the general-purpose tools mentioned above, embedded development frequently calls for specialized tools made to address the particular difficulties involved in working with embedded systems:

Cross-Compilers

- **arm-none-eabi-gcc**:It is intended for cross-compiling software for ARM-based microcontrollers. Because it doesn't require an underlying operating system, it supports the ARM architecture. For the purpose of creating software for ARM Cortex-M microcontrollers, this toolchain is frequently utilized.

- **MSP430-GCC**:This version of GCC is specifically made for Texas Instruments MSP430 microcontrollers; it enables programmers to write and build code for the MSP430 family of devices, guaranteeing compatibility with its distinct features and architecture.

Flashing tools

- **MSP430-Flasher**: TI MSP430 microcontroller firmware can be loaded using this command-line utility. It enables developers to program, erase, and check code on the microcontroller via a number of programming interfaces, including JTAG.

- **dfu-util**: Used for flashing devices that support the USB Device Firmware Upgrade (DFU) protocol.It is especially helpful for changing the firmware on gadgets that communicate via USB.

Real-Time Operating Systems (RTOS) Tools

- **FreeRTOS**: A popular real-time operating system kernel for embedded devices.Development environments supporting FreeRTOS often include additional tools for configuring and monitoring tasks, interrupts, and system resources in real-time.

- **Zephyr**: An open-source real-time operating system that supports a variety of architectures and it is very scalable. With its configuration, build management, and debugging capabilities, it can be used for a wide range of embedded applications, from basic sensors to intricate Internet of Things systems.

B. Programming in an Embedded Linux Environment

1. Choice of Programming Language

C and C++ are the most widely used programming languages in embedded systems development, and this choice is driven by several key factors:

- **Low-Level Hardware Access**: Direct communication with hardware elements like registers, memory, and peripherals is frequently necessary for embedded systems. C is the perfect language for systems where control and performance are crucial since it allows one to write low-level code that can directly modify hardware.

- **Efficiency and Performance**: Both C and C++ are compiled languages, which means that the processor may immediately execute them as machine code. Because memory and processing power are scarce in embedded devices, this results in extremely efficient code with no overhead

- **Rich Ecosystem**: The C/C++ programming languages have a large ecosystem of frameworks, tools, and libraries made expressly for embedded systems. This shortens the time to market and speeds up development by supporting RTOS , middleware, and device drivers.

2. Typical Structure of an Embedded Project

An embedded project usually adheres to a structured file structure in order to effectively manage the many parts of the system. An embedded project's typical file structure might resemble this:

- **Source Directory (src/)**:This directory contains the main source code files for the project.It involves putting drivers, additional modules, and application logic into practice. Frequently, files are arranged according to their functions; for example, middleware, application code, and device drivers each have their own folder.

- **Header Files (include/ or inc/)**: These files specify the interfaces for the source code, which contain type definitions, macros, and function prototypes. Usually, they are positioned in a different directory so that implementation and interface may be distinguished

easily.

- **Configuration Files (config/)**: Files in this directory determine system configurations, including memory mappings, clock settings, and peripheral configurations. The embedded system can be easily tuned and customized with the help of these files.

- **Build Directory (build/)**: The compiled output files, including executable files, binaries, and object files (*.o), are kept in the build directory. Usually established during the construction process, this directory could be cleared out in between builds.

- **Makefile**:By automating the build process, the Makefile makes sure that when changes are made, only the appropriate files are recompiled.

- **Scripts (scripts/)**: You may find a variety of scripts in this directory that can be used to automate certain build steps, run tests, or flash the microcontroller.

- **Documentation (docs/)**: This directory contains documentation files such as design documents, user manuals, and README files. Maintaining the project and helping other developers comprehend the codebase depend on accurate documentation.

- **External Libraries (lib/)**:The project may rely on external dependencies or third-party libraries found in this directory. These could be other utilities, drivers, or communication stacks.

The Figure 9 outlines a standard framework for an embedded systems project.

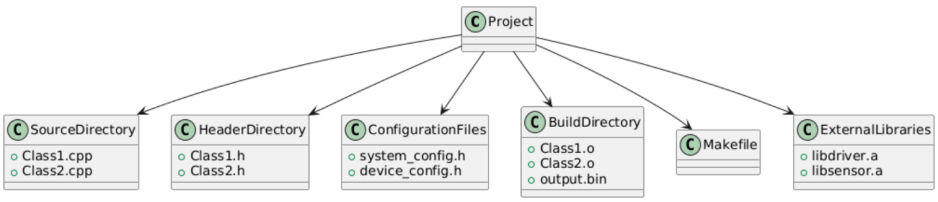

Figure 9: Typical Structure of an Embedded Project

C. ARM Cross-Compilation for MSP430G2553

1. Principle of cross-compilation

Cross-compilation produces executables for non-PC architectures, such as ARM.If a certain architecture is not listed, a software tool must create a toolchain that contains its configuration file.This enables us to create an image for the specified hardware architecture. In most cases, the PC's extensive hardware resources, storage capacity, and powerful processor (x86 architecture) are employed for either:

- Creating a configuration file for the target architecture (for example, ARM for the msp430) and a kernel image that is compatible with it.

- Create an executable for a specific program that is compatible with the target hardware architecture.

The Figure 10 outlines the principle of the cross compilation:

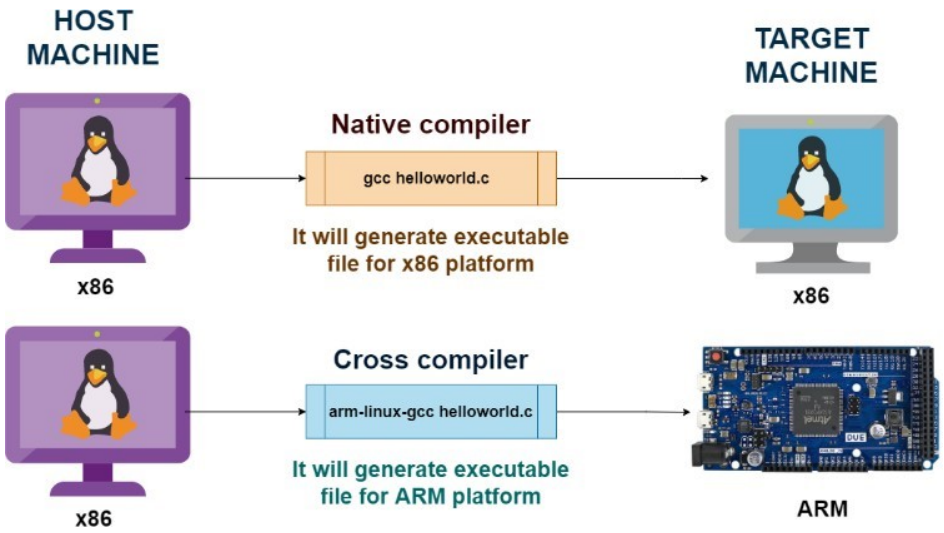

Figure 10: Principle of cross compilation [7]

2. Configuration of the toolchain

The toolchain, for cross-compilation, uses a cross-compiler, assembler, linker, and other related tools to generate executable code for a target architecture different than the one used to construct the code. For example, a cross-compiler could be used to produce code for an ARM processor on a development computer with an x86 architecture. The Figure 11 describes an example of a compilation result using the toolchain:

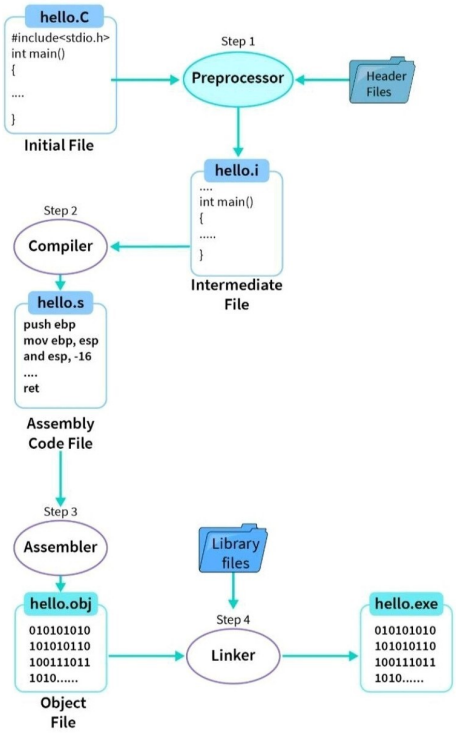

Figure 11: Toolchain [8]

It includes:

- **The preprocessor**: It is the program that acts on the files (libraries) by including them but not on the code. It replaces macros with their contents and deletes comments.

- **The compiler**: It converts the code of each C file into machine instructions (object files). The execution occurs file by file.

- **The linker**: It organizes and links the collected object files before generating the executable file (binary).

Using the command **cpp fichier.c**, you can create an intermediate file (**.i**). The assembler file (**.s**) can then be generated with the command **gcc -S fichier.i**. To compile C and C++ files and obtain object files, we use the commands **gcc -c fichier.c** and **g++ -c fichier.cpp**, respectively. The GNU Compiler Collection (**gcc**) tools are used throughout the procedure. A **.c** file, for example, can be turned into an object file using **gcc -c fichier.c**, and the linker can then construct the executable using **gcc fichier.o -o fichier.** Executing the file is done by typing **./fichier** in the directory where the C and/or C++ files are located. The files **fichier.c**, **fichier.i**, and **fichier.s** are text files, whereas **fichier.o** and **fichier** are binary files in ELF format(Executable and Linkable Format).

Python files can be executed directly using the command **python fichier.py** because Python is an interpreted language that does not require a compilation toolchain. The Python interpreter is usually included by default in most Linux distributions.

3. Generating the Executable for the MSP430G2553

In our project, we built the main source code and generated the executable file for the MSP430G2553 microcontroller using the following steps:

- **Toolchain setup**:We used the GCC toolchain from the msp430-GCC-OPENSOURCE package to compile and link code for the MSP430G2553 microcontroller.

- **Environment Configuration**:We add the path to the **msp430-gcc** compiler to our environment, making it accessible from our project directory.

- **Compilation Command**:We compiled the main.c source file using the command:

  'msp430-elf-gcc -mmcu=msp430g2553 -I /home/walid/Desktop/msp430-gcc/lib/gcc/msp430-elf/9.3.1/plugin/include/config/msp430 -L /home/walid/Desktop/msp430-gcc/lib/gcc/msp430-elf/9.3.1/plugin/include/config /msp430 -Og -g -Wall main.c -o blink'

  This command defines:

  - **Target Specification**:The (**-mmcu=msp430g2553**) flag specifies the target microcontroller.
  - **Header and Library Paths**: The (**-I**) and (**-L**) options include the proper header files and link to the required libraries.
  - **Compilation Enhancements**:We utilized the **-Og** flag for optimal debugging, **-g** to add debugging information, and **-Wall** to enable all compiler warnings. includes the appropriate header files , and links the required libraries.

  To ease the build process, particularly when additional files are added, a Makefile can be constructed to automate the compilation and linking procedures, making project execution easier.

D. Deployment and Testing on the MSP430G2553 Microcontroller

In our project, the deployment and testing phases are critical, especially since we use the MSP430 LaunchPad as an evaluation microcontroller before integrating and testing the code on our sumo robot. This ensures that the code is properly tested and debugged on a controlled platform, reducing errors during final implementation.

1. Flashing the Executable

To flash the MSP430G2553, we used the MSP430 LaunchPad, which is a development kit for the MSP430 microcontroller family. The LaunchPad's onboard USB-based interface simplifies programming and testing by enabling for smooth code uploading and real-time debugging. It enables in-system programming and includes built-in peripherals

such as LEDs and buttons, allowing for rapid code testing and debugging without the use of extra hardware. This makes the LaunchPad an effective tool for creating and improving embedded programs.

2. Testing and Debugging

Once the executable has been flashed onto the MSP430G2553, we use the LaunchPad to perform extensive testing and troubleshooting. The LaunchPad includes crucial debugging features like real-time code examination, breakpoint setup, and variable monitoring using CCS.

This enables us to step through the code, discover and resolve bugs, and test interactions with the microcontroller's built-in peripherals. By recreating conditions comparable to those seen in the sumo robot, we can test the code's performance and confirm that it operates as predicted.

This iterative process of testing and modifying code on the LaunchPad helps to reduce issues before moving to the sumo robot. Once the code is stable and works well on the LaunchPad, it is transmitted to the sumo robot's MSP430G2553 microcontroller. This thorough testing lowers the likelihood of experiencing problems in the final, more complicated robot environment.

Conclusion

We've covered the fundamentals of working with Embedded Linux and cross-compiling for the MSP430G2553. In the following chapter, we'll look at the hardware and software tools used in our project, expanding on this foundational knowledge.

III. Hardware and Software Tools

This chapter presents a comprehensive overview of the hardware and software components utilized in this project. We will detail the equipment selected for the sumo robot and delve into the software tools chosen, including KiCad and Box2D.

A. Hardware Tools

1. Detailed Electronic Hardware Design

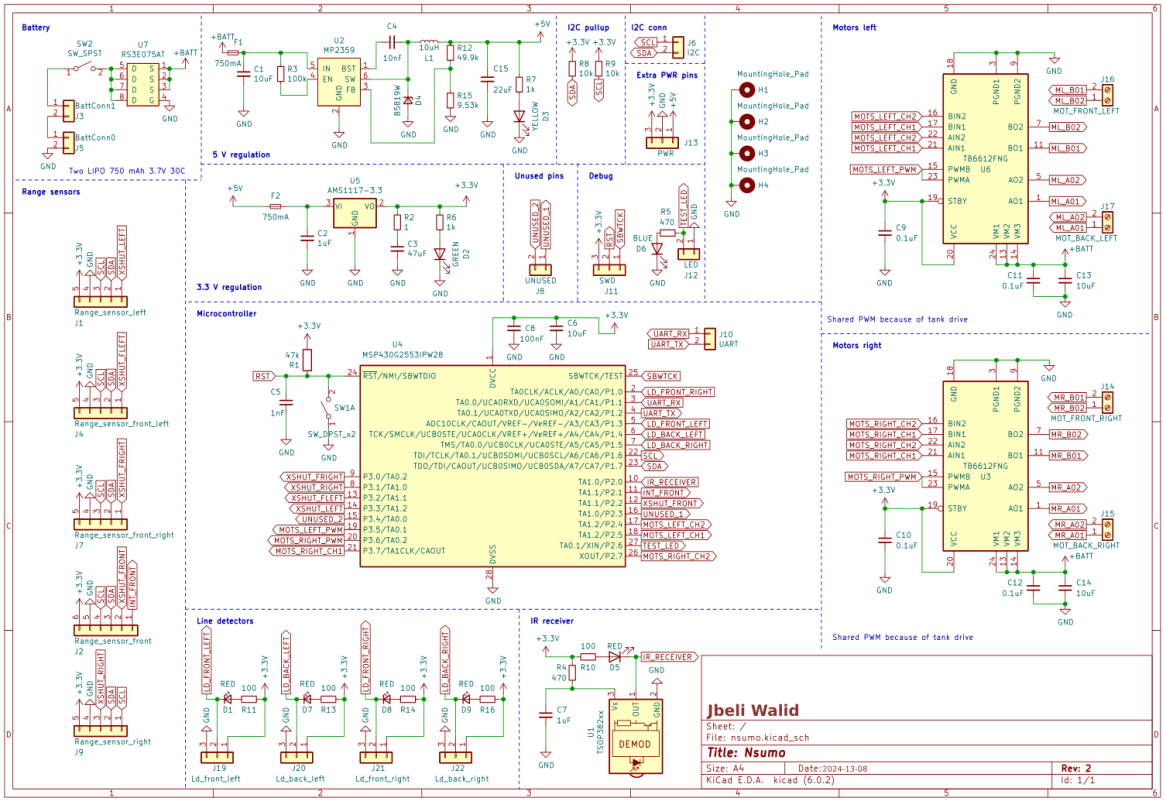

Figure 12: Electronic Hardware Design

The Figure 12 outlines our project's electronic hardware design, which includes various important components to ensure optimal performance and protection. The system is powered by two LiPo batteries, which provide a total voltage of 7.4 V. To protect the circuit from damage caused by faulty battery connections, a MOSFET is used for reverse polarity protection. The majority of the system's components, including the range sensor, line sensors, microcontroller, and IR receiver, require a 3.3 V supply. However, the motors work differently because their voltage is governed by motor drivers, which adjust the power to control the motor speed.To convert the 7.4 V from the batteries to the required 3.3 V for the components, the design contains two voltage regulators: a switching regulator that reduces 7.4 V to 5 V, followed by a linear regulator that reduces this to 3.3 V. This careful design assures consistent power delivery and protects the circuit from damage.

2. Detailed Hardware Components

PCB

The PCB is our fundamental component of electronic design, which is a vital component because it provides a physical foundation for mounting and connecting electronic components. It converts circuit schematics into a physical layout by using copper traces to link components. The PCB maintains steady operation by controlling power distribution, signal integrity, and thermal management. Essentially, it acts as the foundation for electronic gadgets, transforming creative ideals into usable and dependable hardware, as shown in Figure 13 .

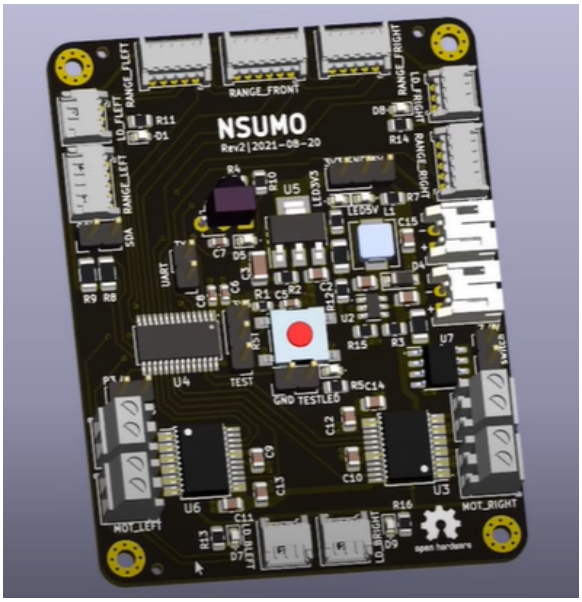

Figure 13: PCB Design

MSP430G2553 Microcontroller

The MSP430G2553 microcontroller is the core of the PCB, acting as the central unit that coordinates all of the robot's functions. It is responsible for analyzing input from numerous sensors, running complicated control algorithms, and regulating motor commands. This 16-bit microcontroller has a clock speed of 16 MHz, ensuring a mix of performance and efficiency. It has 16KB of flash memory to store firmware and control code, as well as 0.5KB of RAM to handle and operate real-time data. With 28 pins, it provides a variety of input/output choices for interacting with other components on the board. Its low-power architecture makes it ideal for battery-powered devices, resulting in longer operational life and more efficient energy use in the robot's compact size,as illustrated in Figure 14.

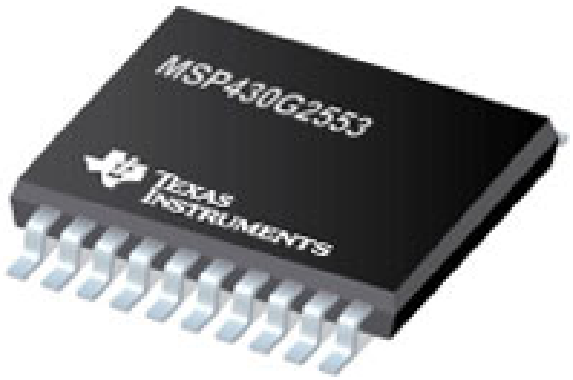

Figure 14: MSP430G2553 Microcontroller

DC Brushless Motors with Gearboxes

The robot is outfitted with two high-torque brushless DC motors with gearboxes, which provide significant torque and accurate speed control required for effective maneuvering and obstacle management in the competition. These motors are rated at 400 RPM and run at 6V, delivering speeds of up to 2 m/s, allowing for quick and nimble movement. The integrated gearboxes increase torque output, making the motors ideal for the rigorous performance requirements of competitive sumo robots. This combination of high torque and speed control enables the robot to handle difficult settings and outperform in competitive scenarios,as displayed is Figure 15.

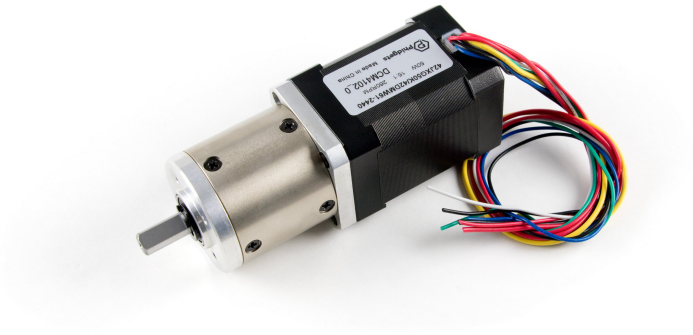

Figure 15: DC Brushless Motors with Gearboxes

TB6612FNG Motor Drivers

The PCB has two TB6612FNG motor drivers, which are required for precise control of the robot's brushless DC motors. These drivers control both the direction and speed of the motors, allowing for precise and responsive movement that is required for effective navigation and maneuvering in the competition . Each TB6612FNG driver can sustain up to 1.2A of continuous current and 3.2A of peak current per channel, providing enough power to meet the motors' high torque requirements. This large current capacity ensures that the motors function consistently and efficiently even under severe situations, enabling for smooth operation and quick adaptation to the robot's surroundings,as presented is Figure 16.

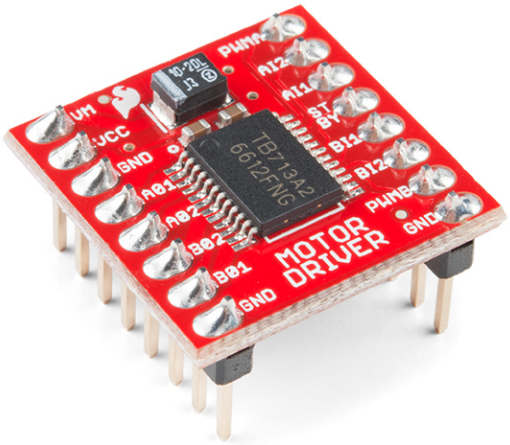

Figure 16: TB6612FNG Motor Drivers

VL53LOX Infrared Range Sensors

The robot features five VL53LOX infrared sensors, which are integral for distance measurement and obstacle detection. These sensors assist in navigating the robot and avoiding collisions by measuring the time it takes for emitted infrared light to bounce back after hitting an object. With a high precision, the VL53LOX sensors provide reliable detection of obstacles and precise distance measurements even in close proximity. This capability ensures that the robot can effectively sense and respond to its environment, enabling smooth and accurate navigation through the competition arena,as illustrated in Figure 17.

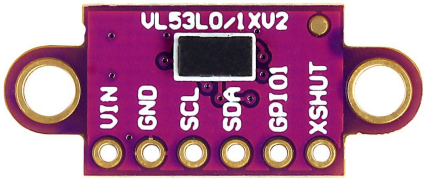

Figure 17: VL53LOX Infrared Range Sensors

The sensor characteristics are given in Table 1 :

Table 1: VL53LOX Infrared Range Sensor characteristics.

Operating Voltage	2.6V to 5.5V
Output Type	Digital (I^2C interface)
Optimal Sensing Distance	30 mm to 2 m
Maximum Sensing Distance	Up to 2 m
Accuracy	±3 mm of the reading

QRE1113 Line Sensors

The robot employs four QRE1113 line sensors, which are required for line following and edge recognition, ensuring that the robot maintains within designated path bounds or reliably recognizes edges, which is vital in competitive situations. These sensors use a reflecting optical system in which an infrared LED emits light, which is reflected off surfaces and sensed by a phototransistor. This design enables the sensors to detect lines or limits by measuring the intensity of reflected light, making them extremely dependable for discriminating between different surfaces. The precision and reactivity of these sensors are critical to the robot's ability to navigate complicated terrain, stay on course, and prevent accidental deviations during competition,as indicated in Figure 18.

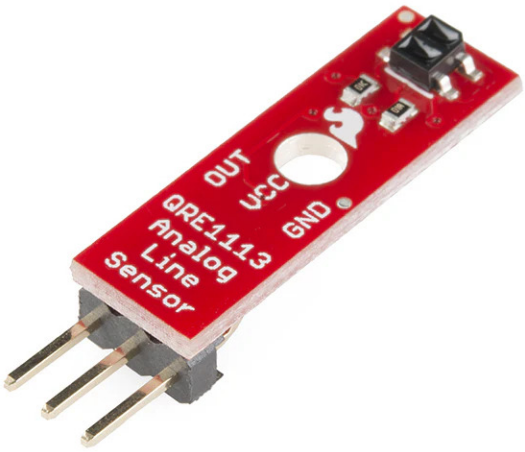

Figure 18: QRE1113 Line Sensors

The characteristics of this sensor are given in Table 2 :

Table 2: QRE1113 Line Sensor characteristics.

Operating Voltage	2.2V to 5.0V
Output Type	Analog
Optimal Sensing Distance	3 mm
Maximum Sensing Distance	10 mm
Accuracy	±1-2 mm of the reading

TSOP38238 IR Receiver

The robot is equipped with an infrared receiver, which is critical in capturing start signals transmitted at 38 kHz. This capability enables the robot to respond to remote control commands or activate at the start of a competition, which is critical for synchronized operation and exact timing. The IR receiver is intended to detect infrared signals at this exact frequency, ensuring that start orders are received accurately and consistently. This dependability allows the robot to respond quickly and properly to activation signals, which is critical for good performance in competitive circumstances,as shown in Figure 19.

Figure 19: TSOP38238 IR Receiver

B. Software Tools

In the development of our sumo robot project,we utilized several advanced software tools to create a comprehensive and robust design such as:

1. KiCad

KiCad was the major software used to construct the robot's comprehensive electronic hardware design. This robust open-source program enabled us to design and simulate the PCB , assuring accurate placement of components such as the MSP430G2553 microcontroller, motor drivers, sensors, and power management circuits. Using KiCad's comprehensive libraries and simulation capabilities, we were able to meticulously arrange the electrical connections and optimize the layout for signal integrity and thermal management, both of which are crucial in high-performance robotics.

2. PlantUML

To explain and record the robot's capabilities, we used PlantUML, a powerful tool for constructing UML diagrams. PlantUML was useful for visualizing the architecture and behavior of the robot's control systems using class diagrams. These diagrams presented a clear and structured representation of the relationships between various software components, such as sensor data processing, motor control, and decision-making algorithms.

Using PlantUML, we were able to effectively describe the functional flow and interactions within the robot, making the development process more efficient and collaborative.

3. Box2D

To simulate the robot's physical interactions and movements, we used Box2D, a 2D physics engine. Box2D was critical in testing and refining the robot's performance in a virtual environment prior to deployment. This simulation tool enabled us to study the robot's dynamics, such as collision detection, force application, and friction management, all of which are critical in a sumo robot tournament where physical interaction with the opponent and surroundings is important. By modeling many scenarios in Box2D, we were able to optimize the robot's design for balance, speed, and maneuverability, guaranteeing that it performs well in real-world competition.

Conclusion

In this chapter, we analyzed the essential hardware and software components required for implementing this project. In the next chapter, we will focus on the project's realization and the simulation with Box2d.

IV. Project Realisation

This chapter explores the practical implementation of the sumo robot project, focusing on key aspects such as software architecture, GPIO programming, handling multiple hardware versions, and motor control using PWM. It also covers the integration of peripheral drivers and the transition from simulation to real-world testing.

A. Software Architecture

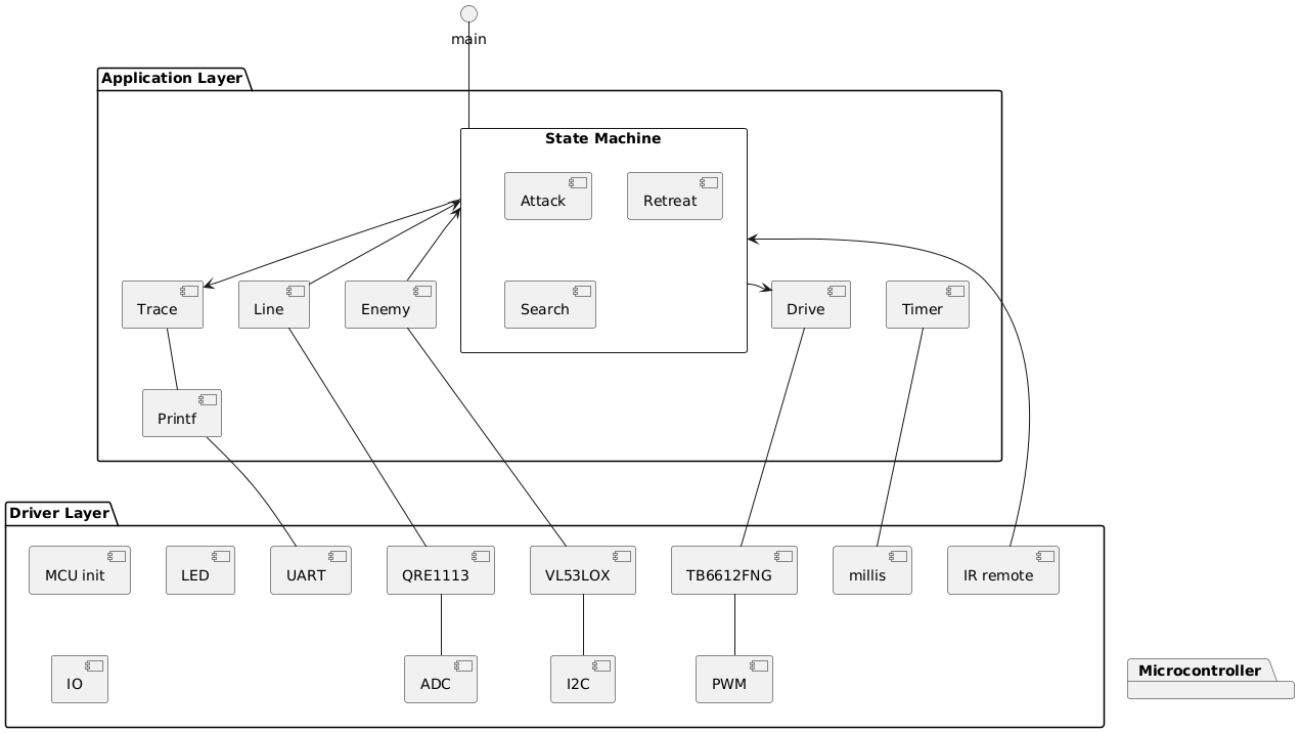

Figure 20: Software Architecture

This figure 20 illustrates the software architecture of the sumo robot, showcasing the key modules and their interactions within the system. The architecture is divided into two main categories: the application layer and the driver layer.

1. Application layer

In the application layer, the central component is the **State Machine**, which governs the robot's behavior.It generates commands to operate the motors after making decisions based on data from several sensors. The **Enemy** module, for instance, interprets the array of distance values into a simplified representation, such as determining whether an enemy is close, nearby, or at a precise place, by processing data from the **VL53L0X** sensor via the **I2C** interface. Similarly, the **Line** module uses the **ADC** module to collect data from the **QRE1113** sensor, calculating the line's position:left, right, or somewhere else and sending it to the **State Machine**.

Based on these inputs, the **State Machine** determines the sumo robot's movement by sending commands to the **Drive** module, and it communicates with the driver layer to

carry out these operations.This allows it to determine the movement of the sumo robot. There are several states in which it functions, including **Retreat**, **Attack**, and **Search**. The **Trace** and **Printf** modules are additional application layer modules that enable the **State Machine** to emit messages to a host computer via **UART** so that it is able to control the duration of its movements and coordinate actions, such as turning left for a second for example, by using the **Timer** module to measure elapsed time.

2. Driver layer

The driver layer carries out the coordinated motions coming from the application and it includes several key components such as:

- **MCU init module**:handles the microcontroller's initialization, configuring the clock rate and setting up IO pins.
- **IR remote module**:uses the timer peripheral to interpret signals from the IR receiver.
- **PWM module**:generates PWM signals to control the motor driver, setting the direction of movement.
- **LED module**:manages the code for toggling the test LED on the PCB.
- **millis module**:provides timer functionality.
- **IO module**:contains functions for configuring individual pins.

B. Programming GPIO

In this section,we will examine the crucial function of GPIO in our sumo robot, along with important implementation specifics and configuration.

1. GPIO's Role

In embedded systems, GPIO (General-Purpose Input/Output) pins are adaptable parts that are necessary for interacting with a range of external devices. The pins can be set up to read or drive digital signals depending on whether they are in the input or output mode. While in output mode, GPIO pins can drive LEDs or relays, they can also read signals from digital sensors, switches, or buttons in input mode. It include interrupt support to identify events like button pushes and programmable pull-up/pull-down resistors to keep input pins in a default state. Additionally, they have drive strength capabilities, which establish what kinds of devices the pin can power directly.

2. Programming and Configuring GPIOs in the Project

According to the pin assignments listed in our schematic, we control the IO pins on the robot and the development board for our project. We create an abstraction in C code to make writing to registers easier and to configure and initialize all of the pins from one central location, which helps to speed this procedure. This abstraction comes with features to determine array sizes and maximize code performance, like a **defines.h** file and timer control. It creates arrays for several GPIO-related registers like input, selection,

interrupt, direction, and output ,across multiple ports in order to setup and configure GPIO pins on an MSP430 microcontroller.

In addition, the abstraction contains interrupt handling methods and establishes the basic configuration structure for the GPIO pins, including their direction, output state, resistor status, and selection. This strategy emphasizes how important GPIOs are to maintaining efficient communication with project's external constituents.

The Figure 21 presents the IO pins schematic of our microcontroller .

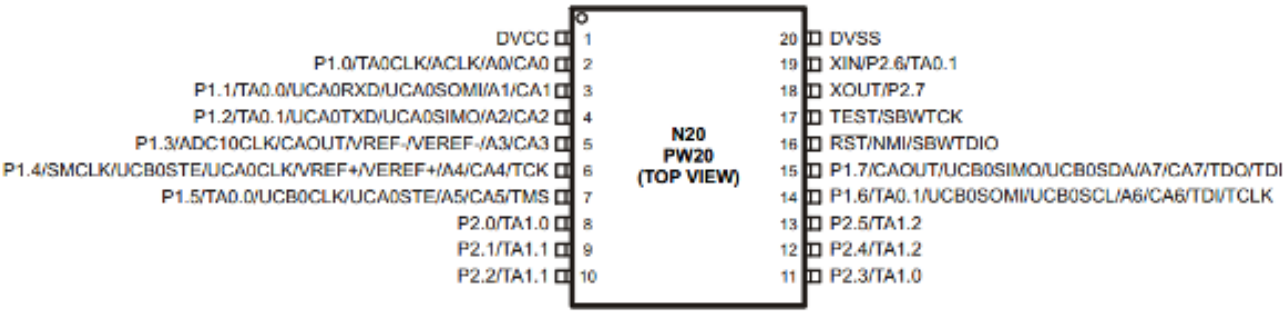

Figure 21: IO pins [9]

C. Interrupts and UART Driver

1. Microcontroller Interrupts

Interrupts are mechanisms that allow a microcontroller to efficiently manage events by temporarily halting its current tasks to prioritize and address more urgent ones. This approach allows the CPU to react to specific conditions, such as a button press, without the need for continuous polling, which can be inefficient and consume more power.

Interrupt process and its advantages

The interrupt controller is an essential component in interrupt management as it keeps an eye out for events on multiple peripherals, including GPIO ports, I2C, UART, and ADC. A register is updated when the robot presses a button, which modifies the voltage on the GPIO pins. When the interrupt controller notices a change, it alerts the CPU.

The CPU saves its current state, which includes the Program Counter (PC), Stack Pointer (SP), and status registers, to the stack upon receiving the interrupt signal. Next, it accesses the Interrupt Vector Table (IVT), which is kept in program memory, to obtain the Interrupt Service Routine (ISR) address. The CPU uses this address to load programs into the PC and runs the code-defined ISR.

Following the completion of the ISR, the CPU restarts regular program execution by restoring the saved state from the stack. Compared to polling, which entails continuously checking a condition and may result in higher CPU cycle and lower power efficiency, this interrupt-driven method is more effective. The system minimizes unnecessary CPU activity while staying responsive to high-priority events by effectively man-

aging interruptions.

Interrupts add complexity to the system even though they provide benefits like less CPU cycle waste and increased power efficiency. To guarantee correct system operation, great care must be taken when designing and maintaining ISRs and state restoration. Interrupts, in general, improve the microcontroller's effectiveness and responsiveness by allowing for quick responses to events and reducing the time delay.

The figure 22 demonstrates the interrupt process cycle during the program's execution.

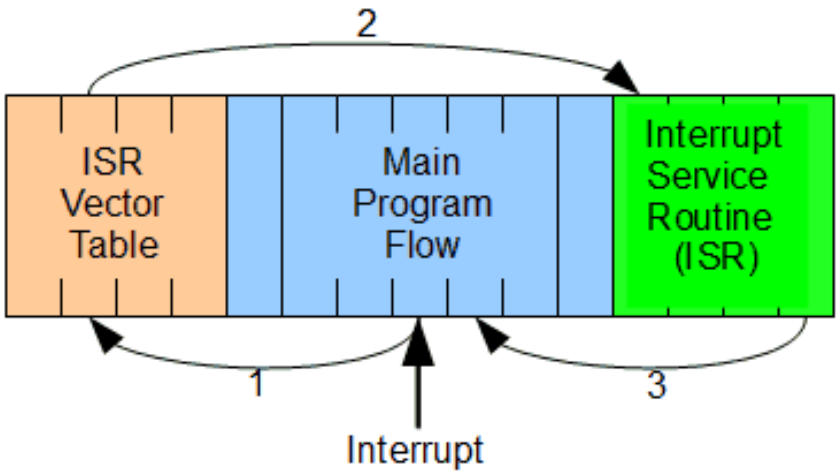

Figure 22: Interrupt process cycle [10]

2. UART driver implementation

In embedded systems, UART (Universal Asynchronous Receiver/Transmitter) is essential for serial communication, converting parallel data from a microcontroller into serial data for transmission, and contrarily for reception.When interacting with other microcontrollers, computers, sensors, or other external devices, this is quite helpful. RX (Receive) and TX (Transmit) are the two pins on the microcontroller that commonly indicate UART. We use a USB-to-UART converter chip to connect the microcontroller of the Sumo robot to the desktop computer so that communication between the two can be facilitated. The data stream from UART to USB is translated by this chip. We develop a UART driver for our Sumo Robot in two stages: first, we use polling, and then we improve it with ring buffers and interrupts for more effective data handling.

Polling-Based UART Driver

The CPU actively monitors the UART status register in a polling-based system to determine whether data is prepared for transmission or reception. Although simple, this approach can be inefficient because the CPU is always busy watching for UART activity.

- **Polling Transmission**: When the transmitter is ready, data is loaded into the UART data register. Until the communication is finished, the CPU polls.

- **Polling Reception**: Prior to reading the data register, the CPU polls the UART receiver until data is available.

Interrupt-Driven UART with Ring Buffer

When the data is ready, the UART initiates an interrupt. As a result, the CPU can work on other projects while it waits for data. Incoming data is momentarily stored in a ring buffer until the CPU is prepared to process it. Because it reuses memory in a cyclical form, a ring buffer, sometimes called a circular buffer, facilitates efficient data management.

- **Interrupt on Receive**: A character is added to the ring buffer and an interrupt is triggered upon data reception.

- **Polling Transmit**: When the UART is ready, data is transferred from the ring buffer.

The figure 23 represents the serial communication with UART driver.

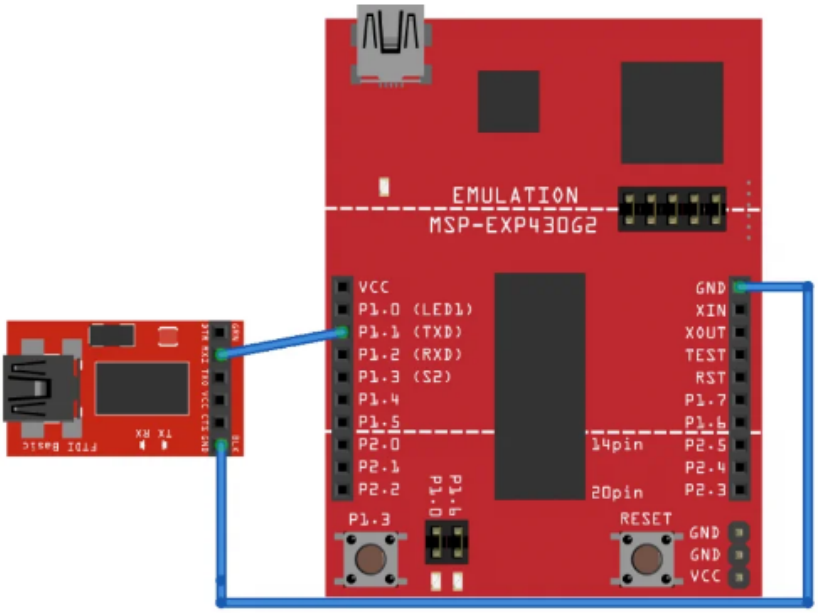

Figure 23: UART communication [11]

To output formatted strings on the terminal, we implement a **printf** function suited for microcontroller environments by using the UART driver. The standard **printf** can be too heavy for embedded systems due to its memory usage, so we use a lightweight external implementation.

Printf Implementation

Through the UART connection, we will use a desktop terminal interface to communicate with the microcontroller. Serial communication is accomplished with the **picocom** tool:**picocom -b 115200 /dev/ttyUSB0**.
This command opens the serial communication at a baud rate of 115200 with the microcontroller.

To make debugging easier, we prefix every log with the filename and line number by wrapping the **printf** function in a TRACE macro. This is particularly helpful for spotting problems in the development stage. We add trace logging to our assert functionality to enhance debugging. When an assertion fails, the PC register,which contains the MCU's current execution point,is logged along with the failure location.
Later, we use a tool like **addr2line** to retrieve the file and line number. The Figure 24 shows an example of a formatted string on the terminal related to remote control of the IR receiver buttons.

Figure 24: Printf implementation

D. NEC Protocol Driver and Motor Control with PWM

1. NEC Protocol Driver

To interpret infrared signals sent between a robot and an IR remote, we have created a simple driver. The NEC protocol, which is commonly used for communication between gadgets like televisions and remote controls, is followed by this signal. An infrared signal is sent by the remote control when a button is pressed. This signal is detected by the IR receiver, which is attached to pin 2.0 on the PCB. It then triggers a pin on the microcontroller's GPIO, turning the signal into a digital signal. Software then decodes this digital signal into a 32-bit message. By leveraging GPIO interrupts to detect signal edges and using the microcontroller's timer peripheral to measure the timing between the remote and the receiver, we can accurately interpret the transmitted data.

2. Motor control with PWM

PWM overview

PWM is a technique widely used to control the amount of power delivered to devices like motors by adjusting the duty cycle of a digital signal.In motor control , it enables us to alter the power supplied while preserving a steady supply voltage, hence regulating motor speed. We can efficiently manage the speed of the motor or even the brightness of the LEDs by varying the length of time the signal is high. The key PWM Concepts includes:

- **Duty Cycle**: The proportion of time that a signal is active in a certain period. A signal with a 50 per cent duty cycle is high 50 per cent of the time.
- **Frequency**:The PWM signal's frequency indicates how often it repeats. Smoother control is the outcome of higher frequencies.

- **Channels**:it may create numerous PWM signals on the MSP430G2553 microcontroller by using timers such as **TimerA**.

Motor Control Implementation

In our robot, The 4 motors are managed by two TB6612FNG motor drivers, each responsible for controlling the motors on either side of the robot:one driver controls the motors on the left, while the other manages the motors on the right. Four essential pins are attached to each motor driver in order to provide power and control functions:

- **Battery power supply**: The voltage and current required to run the motors are supplied by this pin. For the motors to operate at their best, a steady power supply is provided by the battery.
- **Two pins for motor direction control**: These pins are crucial for figuring out which way the motors rotate. The logic levels on these pins can be adjusted to cause the motors to revolve forward or backward, giving the robot variable movement.
- **One pin to control the speed via PWM**:This pin is connected to a PWM signal, which controls the speed of the motor by adjusting its duty cycle.

The wiring of the motor and motor driver prototype is shown on the Figure 25 .

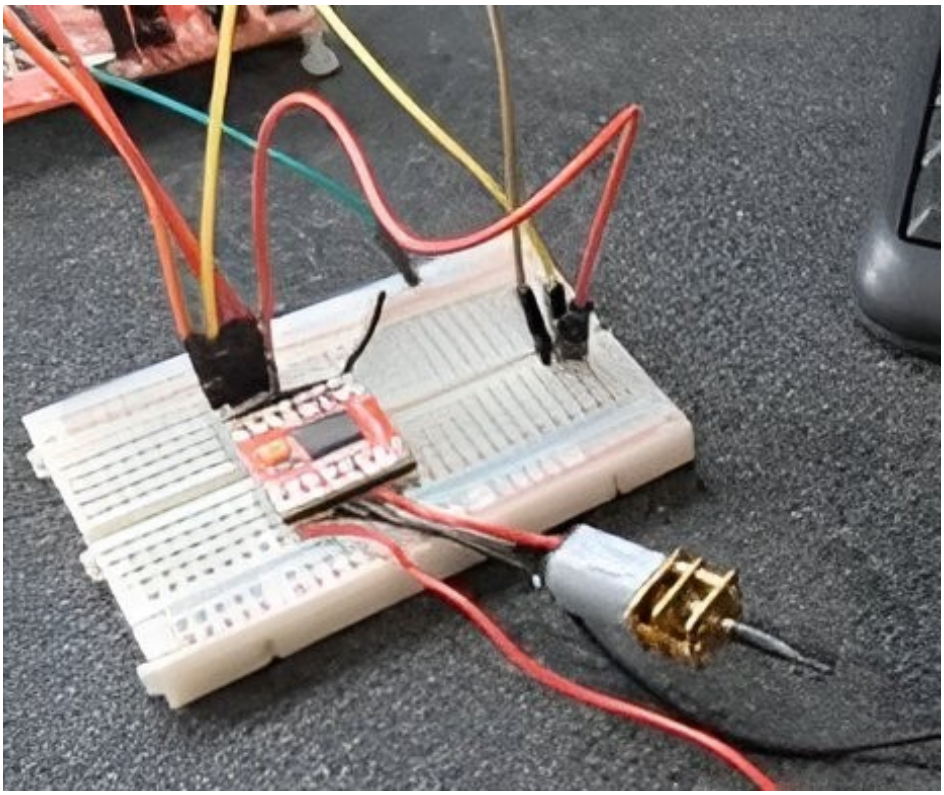

Figure 25: Motor control with PWM

E. Peripheral Drivers

1. ADC driver with DMA

Overview of ADCs

ADC is a crucial component in embedded systems, responsible for converting continuous analog signals, like voltage, into a digital format that the microcontroller can process.Applications utilizing sensors, such as potentiometers, light sensors, and temperature sensors, are common uses for ADCs. When setting up an ADC, we specify variables like:

- **Resolution**: Establishes the level of conversion precision, such as 8-bit, 10-bit, etc.
- **Reference Voltage**:Defines the maximum voltage that the ADC can interpret.
- **Sampling Rate**:Determines how frequently the analog signal is sampled by the ADC.

ADC Driver Implementation with DMA

We create an ADC driver for our robot to manage the sampling of values from every QRE1113 line sensor. It recognizes the white line denoting the edge of the platform. The line sensor measures the light that is reflected back after emitting a signal. Different voltage levels are produced by the reflected light based on whether the sensor is above the white border or the black platform.An analog voltage signal is provided by each sensor's output pin, which is attached to our microcontroller. These analog voltages must then be converted into digital values via the ADC. We can ascertain the robot's location in relation to the platform's edge with the use of these digital values.

The Figure 26 presents the line sensor circuit.

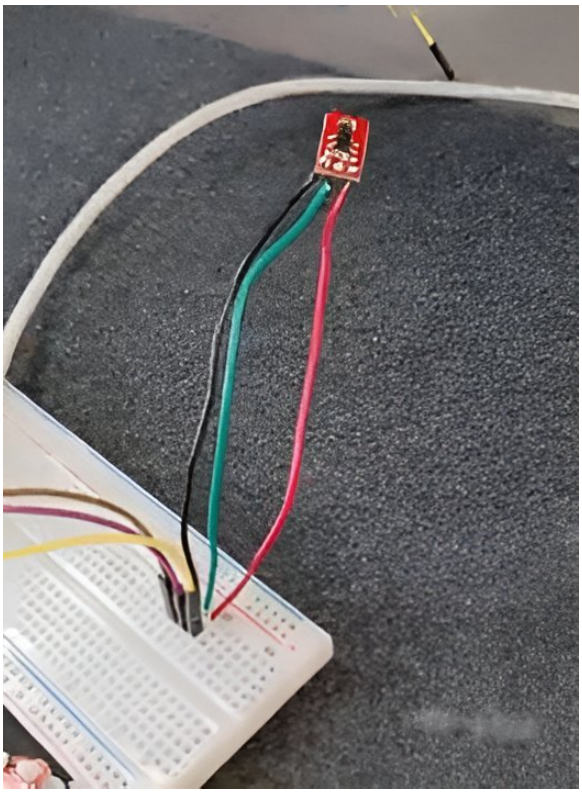

Figure 26: Line sensor circuit

To manage this setup efficiently, we create the three system layers that are already shown in our software architecture:

- **ADC Layer Implementation**:Our primary concerns in the first layer are setting up the ADC to read sensor values and utilizing DMA to lighten CPU burden. The data transfer between the ADC and memory is handled via DMA, which eliminates the need for the microcontroller to continuously monitor and process each reading. As a result, we can collect data from each of the four sensors without taxing the CPU.

  DMA operates by copying the ADC-sampled values directly into memory. An interrupt tells the CPU that fresh values are available when the transfer is finished. We can always process the most recent sensor values because the ADC driver is set up to sample the sensors continually.

  To further optimize the process, we configure the ADC driver to use the ACLK (Auxiliary Clock), which operates at a slower frequency, reducing the load on the microcontroller while maintaining sufficient accuracy for our application.

- **QRE1113 Layer Implementation**:The second layer is then constructed.It processes the ADC's raw digital results and interprets them as sensor readings.

- **Application Layer**:In the final layer, we translate the sensor readings into positional information, allowing the robot to understand whether it is aligned with the

white line.

The Figure 27 shows the line sensor results on the terminal.

Figure 27: Line sensor results

In this context, the **Line 4** indicates that the line sensor predicts the black color of the platform and the **Line 1** indicates that it predicts the white color of the platform's edge

2. I2C driver

Overview of I2C

I2C (Inter-Integrated Circuit) is a popular serial communication protocol that uses a straightforward two-wire bus to link low-speed peripherals to microcontrollers. This protocol works well, particularly for systems that have several devices that they must connect with. I2C includes wires such as:

- **SDA (Serial Data Line)**: Carries the data between devices.
- **SCL (Serial Clock Line)**: Provides the clock signal to synchronize data transmission.

Sensors, EEPROMs, RTCs (Real-Time Clocks), and other low-speed peripherals are frequently connected via I2C. Perfect for our use case requiring range sensors, it allows connection with devices that have low to moderate data transfer requirements. I2C is ideal for our project because it simplifies wiring by enabling effective communication between the microcontroller and five VL53L0X sensors:3 in front, 2 on the sides with just two wires:SDA and SCL.

I2C Driver Implementation

We establish the three system layers that are already visible in our software architecture in order to effectively handle this setup:

- **I2C Driver Implementation**:To interact with the VL53L0X sensors,we manage the I2C connection between the microcontroller and the range sensors.The I2C was configured in polling mode without interrupts to avoid the complexity of interrupt-driven communication and to maintain simplicity.six pins VL53L0X sensor were

used in the test circuit; its pins were labeled GND for ground, VCC for power, and SDA and SCL for I2C communication with the logic analyzer and microcontroller. The interrupt pin was not used in the configuration because of the polling technique.Instead, the XSHUT pin was used to place the sensor in standby mode. A logic analyzer was used to confirm the communication between the microcontroller and the I2C bus. Additionally, two 4.7 kOhm resistors were used as pull-ups on the SDA and SCL lines.

We can present this wiring prototype in Figure 28.

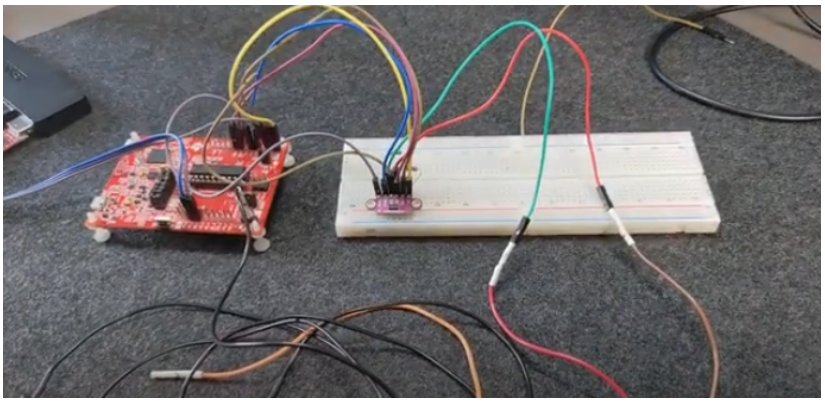

Figure 28: Range sensor wiring prototype

- **VL53L0X Layer Implementation**:We control communication with the VL53L0X range sensors and retrieve millimeter-based distance data. These sensors are essential for determining how close the enemy is,with each sensor being polled to obtain precise distance readings in millimeters.

- **Enemy Detection Application layer**:In this final layer,we processed the distance information to estimate the enemy's position and emit a meaningful message.The output is a textual message that indicates the enemy's location in relation to the robot, such as "Enemy in front." The input is a distance data that is transformed into a logical position (example: far,middle,close). The system is more flexible, easier to debug, and more effective at processing sensor data thanks to this layered design, which guarantees a clear separation between hardware communication (I2C), sensor handling (VL53L0X), and application logic (enemy recognition).

The Figure 29 presents an example of textual messages that indicate the enemy's positions on the terminal.

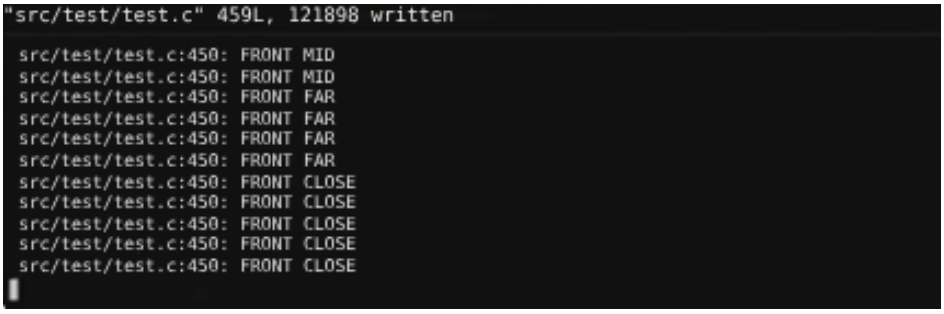

Figure 29: Range sensor results

F. Coding the State Machine and Simulation to Real-world Demo

1. State machine

A state machine is a model used to represent the behavior of a system that can exist in different states. It enables the management of intricate action sequences and the switching between states in response to certain circumstances or occurrences. State machines are very helpful in robotics because they allow robots to carry out independent tasks, respond to their surroundings, and alter their behavior dynamically while keeping an orderly, systematic sequence of operations.

For instance, our sumo robot has to wait for the signal to start, look for the enemy, attack, and retreat from the edge, among other responsibilities. The details of our state machine functionality are shown in Figure 30 .

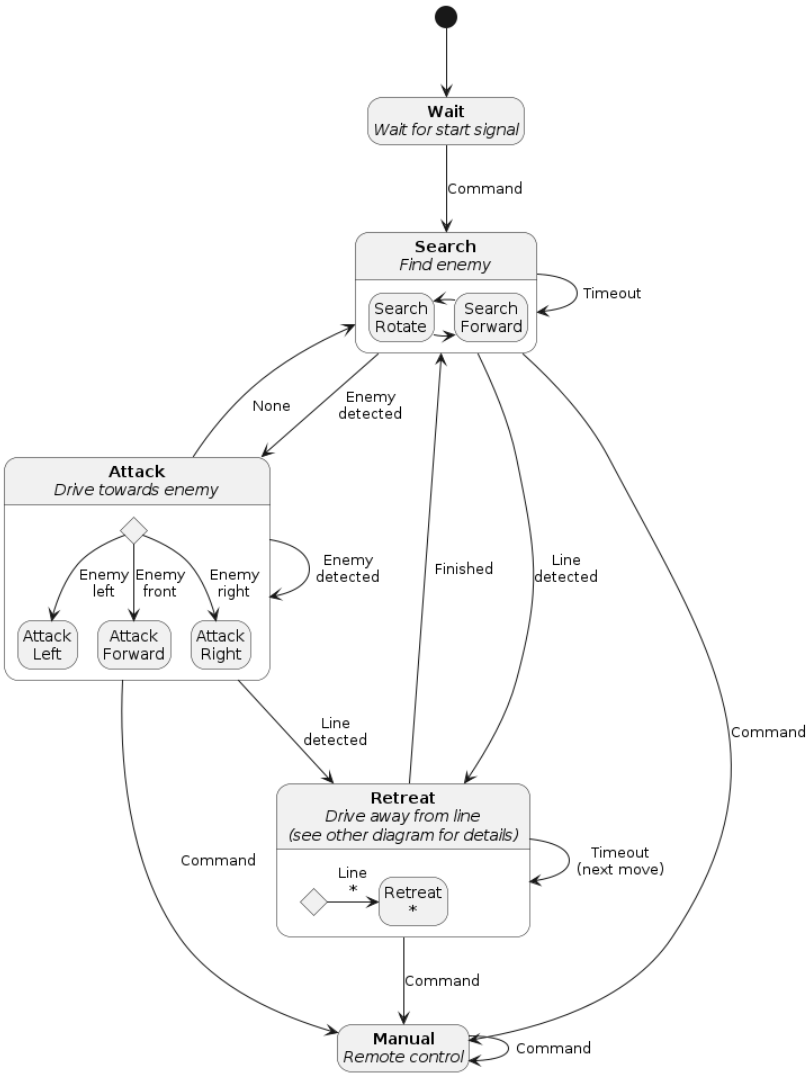

Figure 30: State machine

The diagram illustrates the behavior of our autonomous robot.It begins by waiting for a start signal. Once activated, it enters a search phase, rotating and searching for an enemy. If an enemy is detected, the robot transitions to the attack phase, driving towards the enemy and attacking from the appropriate direction. If no enemy is found

or a timeout occurs, the robot retreats. The retreat phase involves driving away from a detected line, with the specific actions detailed in next diagram .If the robot encounters a line during retreat or the timeout expires, it returns to the search phase. The diagram also includes a manual control option, allowing for human intervention. Overall, this diagram outlines the basic decision making process for the robot's navigation and combat operations.

Also,we present the retreat state that is explicated in Figure 31 .

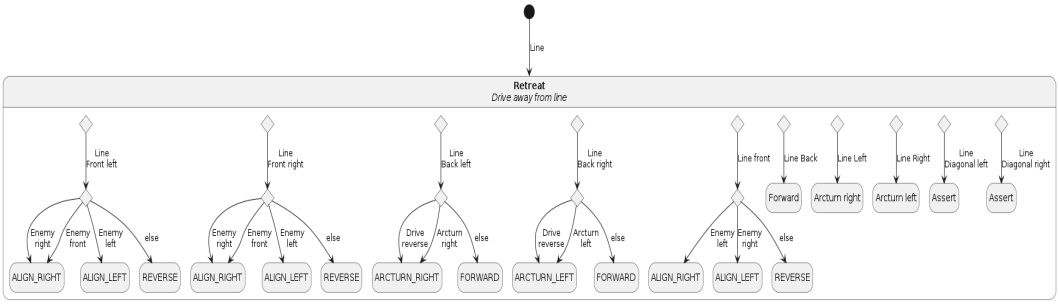

Figure 31: Retreat state

The provided state diagram outlines a retreat strategy for a military unit. The diagram starts with an initial state, "Retreat", which triggers a series of actions to ensure safe withdrawal. The unit is expected to drive away from the enemy while maintaining formation. The "Line" state represents the main formation, and the "Front right," "Front left," "Back left," and "Back right" states indicate specific positions within the line. Various maneuvers, such as "Hack left," "Diagonal left," and "Diagonal right," are available to adjust the formation as needed. The diagram also includes states for "Forward" movement, "Arcturn" turns (right and left), and "Assert" actions, which could be used for defensive or offensive purposes. Additionally, there are states for "Enemy" encounters and "Framy" actions, which might involve specific tactics or strategies. The diagram concludes with the "Align" states, ensuring the unit is properly positioned for further actions. Overall, this state diagram provides a structured approach to retreating from enemy forces, maintaining formation, and executing necessary maneuvers.

2. Simulation to Real-world Demo

Our project's objective is to keep up a basic interface that programmers may use to test and troubleshoot their robot code in a virtual setting before moving on to hardware. The simulation begins by running the state machine in Bots2D, where we verify that our robot interacts with the environment as expected by detecting white lines and engaging with the opposing robot.After the simulation runs successfully, we upload the code to the microcontroller of the sumo robot, which then operates in a real-world setting with the identical behaviors.
The simulator has the following features:

- **Physics Engine**: The simulator simulates object interactions using a strong physics engine to imitate real-world physics.
- **Graphics**: A basic 2D graphical interface provides visual feedback during simulation testing.

- **Input/Output**: The simulator uses mouse and keyboard inputs, while in the real project, sensor data is used as input, and physical motors and actuators produce outputs.
- **Software Architecture**: The entire system is designed to isolate target code, allowing for a seamless transition between simulation and hardware without needing extensive modifications to the logic or structure.

We created a test application with multiple packages and ran a simulation on the platform with a single robot that was manually operated through a binary execution,as shown in Figure 32.

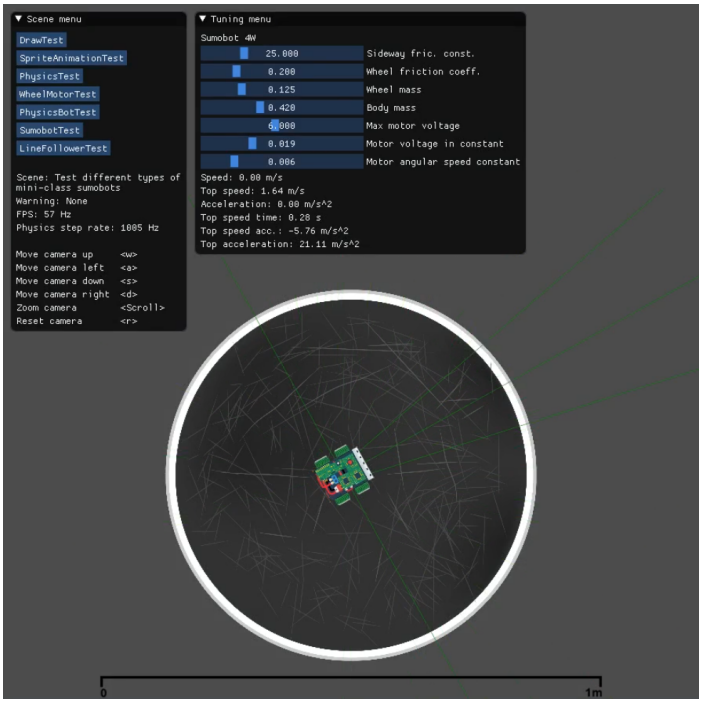

Figure 32: Manual robot

Next, we included our robot's functionality that was programmed into the state machine by integrating our project's simulation into the Box2D environment. After the binary was constructed and runned, the simulation demonstrated that our robot,which was under the control of the state machine,could recognize a white line and make an effort to push the other manual robot off the platform,as presented in Figure 33.

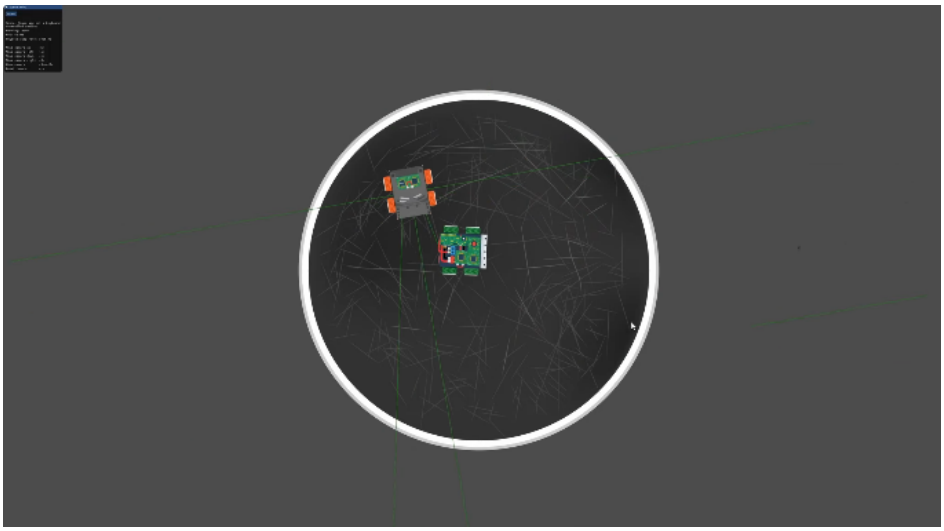

Figure 33: Simulation

In this simulation, the desktop takes on the role of a microcontroller emulator, allowing developers to mimic the behavior of an actual microcontroller without the need for physical hardware.It provides a controlled setting to experiment with different scenarios and edge cases that might be harder to replicate in a physical setup.

Conclusion

In this chapter, the realization of the sumo robot was achieved through the implementation of crucial components like GPIOs, motor control, and peripheral drivers. The challenges of handling multiple hardware versions were addressed, and the robot's behavior was successfully tested in both simulated and real-world environments. This marks a significant step toward the final deployment of the project.

V. Conclusions and perspectives

A. Conclusions

At the conclusion of this report, it will be clear that the sumo robot development project in an embedded Linux environment gave us a great chance to put our theoretical knowledge to use in a real-world situation.

We were given the task of finishing an extensive project, which included requirements analysis, implementation, and testing. Precise attention was paid to every stage, including programming interfaces, communication protocols, and hardware design. It was essential for us to demonstrate our methodology by using embedded Linux-specific development tools and procedures.

Our abilities in teamwork, project management, and the usage of different programming languages and tools have all improved as a result of this experience. We're happy to report that we achieved our goals and added extra features like: :

- Incorporation of efficient communication methods.
- Sophisticated I/O and peripheral management.
- Optimization of robot performance in real-world conditions.
- Creation of comprehensive and accessible technical documentation.

As a result, we have advanced our knowledge of embedded system and helped to create a competitive sumo robot that can perform well in contests.

B. Perspectives

However, there are still opportunities for improvement to optimize the user experience and the efficiency of the sumo robot project by:

- Incorporating additional features into the control and monitoring system of the robot.
- Enhancing the security of the embedded software and communication protocols.
- Integrating advanced sensors and modules to improve the robot's performance and capabilities.
- Refining the robot's algorithms and control mechanisms to enhance its competitive edge in sumo competitions.

References

[1] https://www1.villanova.edu/villanova/engineering/newsevents/newsarchives/2019 /studentsAlumni /sumobot-competition.html

[2] https://www.ubuy.tn/en/product/3290AAK-zumo-robot-for-arduino-v1-2-assembled-with-75-1-hp-motors-by-pololu

[3] https://www.ubuy.tn/en/product/14KZ3780-pic-super-sumo-robot-re-program-electronic-fa1113

[4] https://eu.robotshop.com/fr/products/kit-robot-sumo-midi-jsumo-bb1-assemble

[5] https://www.robotpark.com/Mini-Sumo-Robot-Kit

[6] https://blog.stackademic.com/excelling-in-software-development-with-scrum-methodology-part-2-e2d0b29437ce

[7] https://www.techjedi.in/2022/07/11/cross-compilation-with-example

[8] https://www.scaler.com/topics/c/compilation-process-in-c

[9] https://www.ti.com/product/MSP430G2553?keyMatch=msp430g2553tisearch=universal$_search$

[10] https://embedds.com/basic-understanding-of-microcontroller-interrupts

[11] https://microcontrollerslab.com/uart-serial-communication-with-msp430-microcontroller