Software requirement specification (SRS) document template

By: Shafeen Chowdhury

Project name: Mechatronic Solution and Documentation

Version: 01

Revision history

Version	Author	Verson description	Date completed

Review history

Approving party	Version approved	Signature	Date

Approval history

Reviewer	Version reviewed	Signature	Date

Table of contents

1 Introduction

Describe the purpose of the document.

1.1	Product scope	List the benefits, objectives, and goals of the product.

The objective of BT-896 (Robot) is to have the ability to solve a maze whilst identifying the benefits of BT-896 is the industrial setting. Overall, the goal of Kettle chili chips is to solve a maze with the harsh environment.

1.2	Product value	Describe how the audience will find value in the product.

The target audience will find value in this product due to usage and opportunities. BT - 896 can be applied to many different aspects which can eventually expand to automating certain everyday task

1.3	Intended audience	Write who the product is intended to serve.

The target market in which we will value this product is keen on engineering, more specifically robotics engineering

1.4	Intended use	Describe how will the intended audience use this product.

The intended use for clank is to solve and identify obstacles in a maze but can be used for different problems. This may include moving blocks identifying different colors.

1.5	General description	Give a summary of the functions the software would perform and the features to be included.

The product will follow a track in a specific maze whilst clearing obstacles on its way but using the wheels for direction and the other parts which allow the product to maneuver over these obstacles such like different colored titles ect.

2 Functional requirements

List the design requirements, graphics requirements, operating system requirements, and constraints of the product.

The robot must be designed with durability, performance to ensure reliable operation, this including assembling all compiled parts are correct, using durable components, ensuring the transmission is strong using a voltmeter, additionally, parts like the battery and lcd must be charged and up to date, as for the robot, the robot should withstand various environmental condition, each element contributes to the robots effectiveness, including its lifespan and risk of failure

3 External interface requirements

3.1	User interface requirements	Describe the logic behind the interactions between the users and the software (screen layouts, style guides, etc).

The robots UI must be simple, responsive and can clearly communicate system status, this includes the LCD to indicate the robot's status with the minimalistic layout and easy to read buttons and responsive feedback reducing user error and improving user experience

3.2	Hardware interface requirements	List the supported devices the software is intended to run on, the network requirements, and the communication protocols to be used.

The robot must support specific hardware interfaces to ensure smooth integration and communication between components. These devices include the raspberry pi and ultrasonic sensors; communication should occur over I2C with net worth from local network.

3.3	Software interface requirements	Include the connections between your product and other software components, including frontend/backend framework, libraries, etc.

The robot's software interface must support reliable development tools when handling data, through Visual studio code using python with libraries like (v.02), and must support connection between software and hardware, the backend must process data while the front end presents status updates

3.4	Communication interface requirements	List any requirements for the communication programs your product will use, like emails or embedded forms.

The robot's communication interface must ensure reliable, secure and user-friendly data exchange between the user and system, including reliable data transmission, effective error reporting and secure communication protocols, effective error reporting ensuring that any problems are consistently sent and received.

4 Non-functional requirements

4.1	Security	Include any privacy and data protection regulations that should be adhered to.

Ensuring the data is safe the robot must follow basic cybersecurity standards, and the use of secure connections ensures updating the software and safe data transfers. These steps ensure the risk of hacking is minimized and protects privacy

4.2	Capacity	Describe the current and future storage needs of your software.

The robot must have enough storage capacity to save and process data, by collecting the information as it navigates it stores it in its memory, allowing the robot to potentially optimize its route

4.3	Compatibility	List the minimum hardware requirements for your software.

The robot should meet the minimum hardware and software requirements to function effectively and properly, this includes the main components like a reliable battery, raspberry pi, and sensors to perform the basics.

4.4	Reliability	Calculate what the critical failure time of your product would be under normal usage.

Under normal usage the critical failure time of the robot system would be around 150 hours of continuous use.

4.5	Scalability	Calculate the highest workloads under which your software will still perform as expected.

The robot must be able to handle large and complex majors with losing performance, by efficiently can navigate 50 x 50 in size and adapt to complex layouts.

4.6 Maintainability	Describe how continuous integration should be used to deploy features and bug fixes quickly.

To ensure the robot works before it deploys and bugs can be fixed efficiently the code will undergo unit testing throughout each section before starting, ensuring that each part of the robot works properly.

4.7 Usability	Describe how easy it should be for end-users to use your software.

The robot should have user friendly UI in which provides clear feedback and can display error messages, have a responsive design which reacts quickly and smooth communication between each part of the robot to reduce the amount of bugs

4.8 Other	List any additional non-functional requirements.

The robot must meet the nonfunctional requirements to ensure safety and a reliable working robot, this includes environmental tolerance consistent reliability, high responsiveness and overall safety

5 Definitions and acronyms

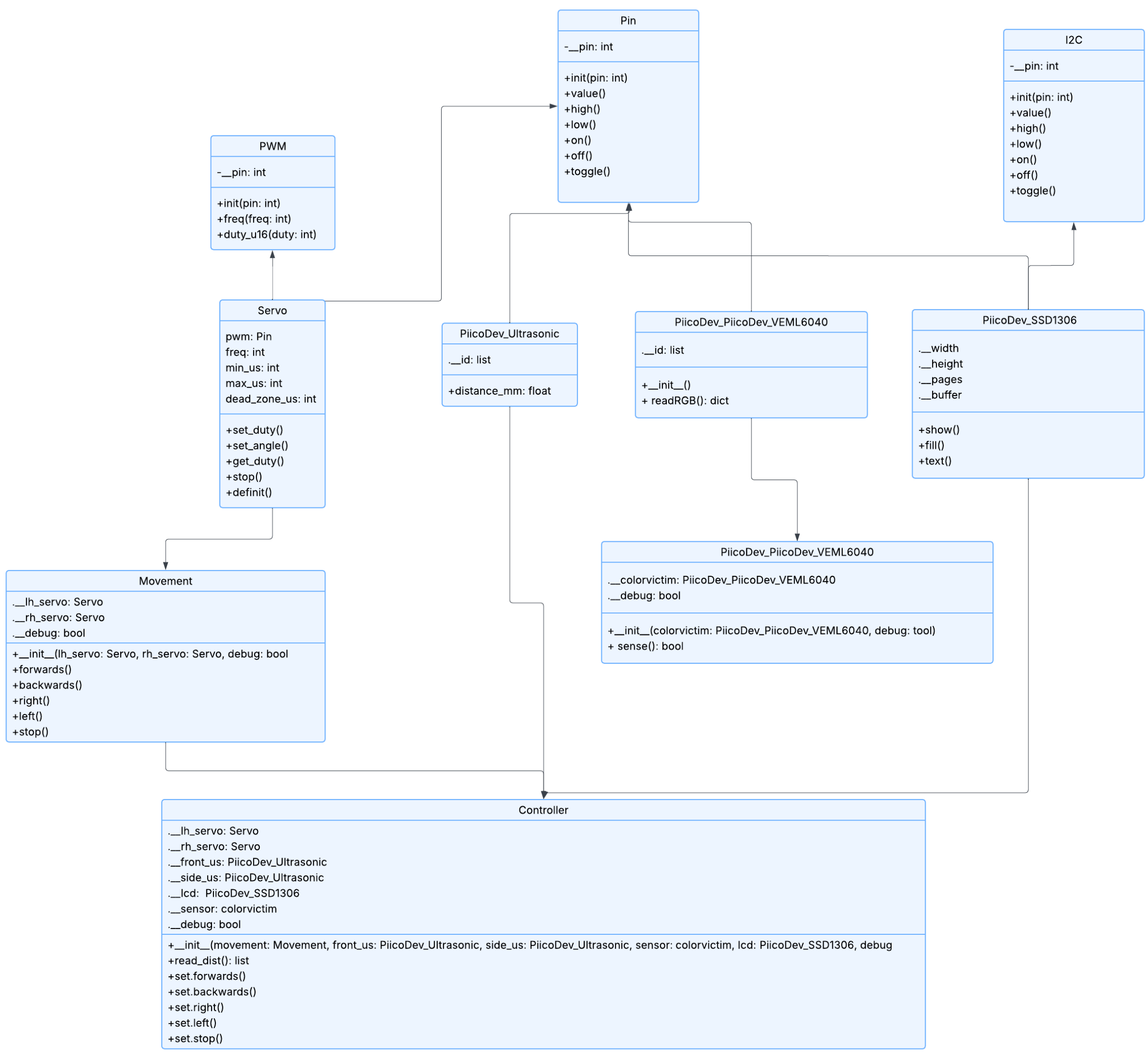
OOP	A programming paradigm that organizes software design around data, or objects, rather than logic and function
I2C	Inter-Integrated Circuit
UI	User-interface
LCD	Liquid-crystal display
Voltmeter	to measure voltage in electrical circuits or electronic devices
Battery	A device that produces electricity

Micro Python OOP Pi Pico Mini Project

Research and planning

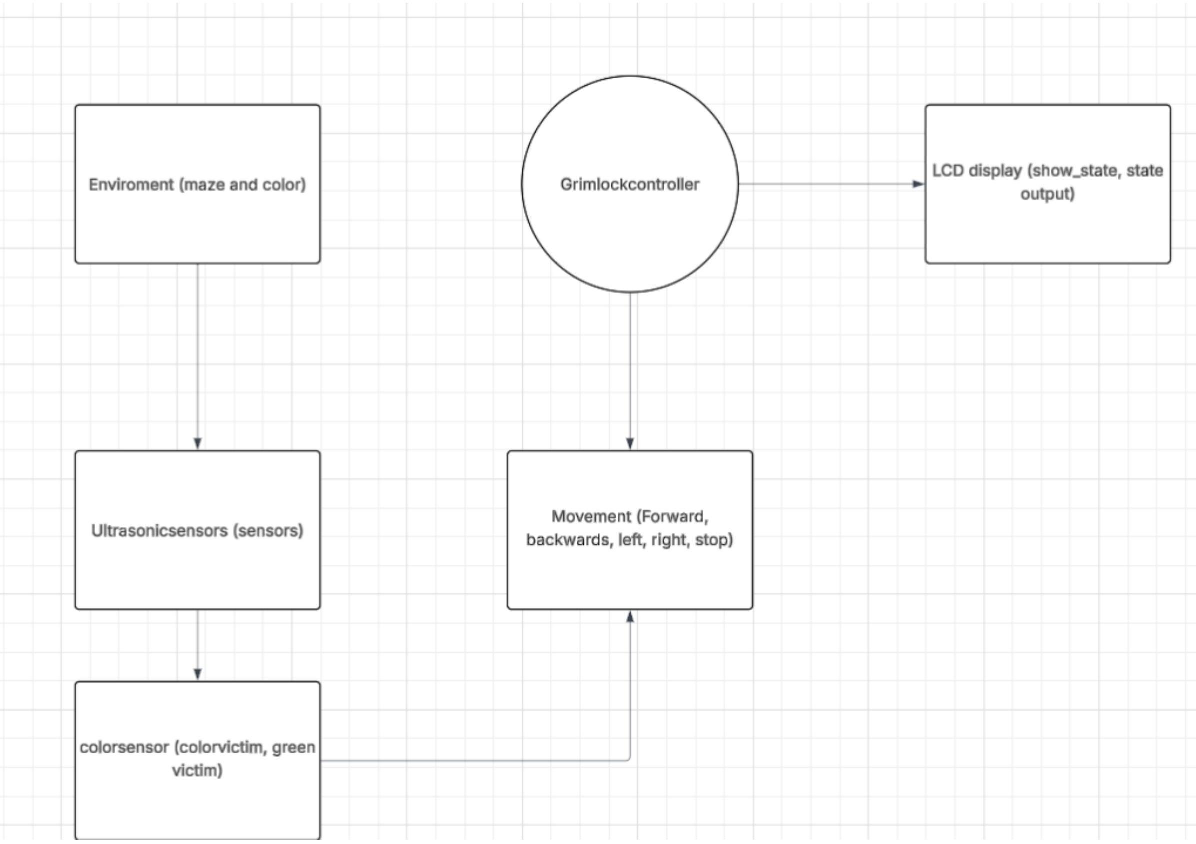
This consists of the diagrams, including UML class diagram, DFD, flowchart, wiring diagram

UML Class Diagram

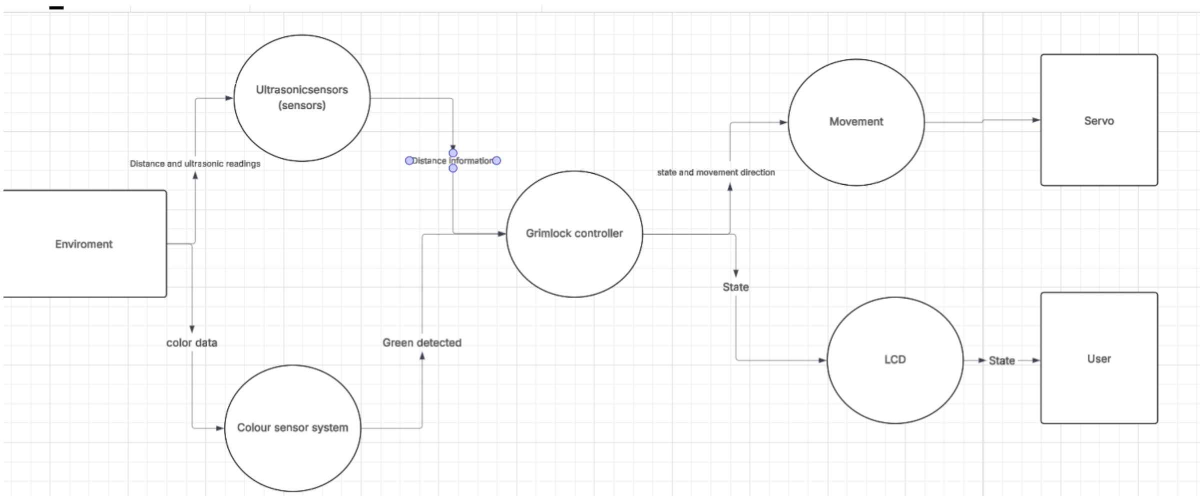


Data Flow Diagrams

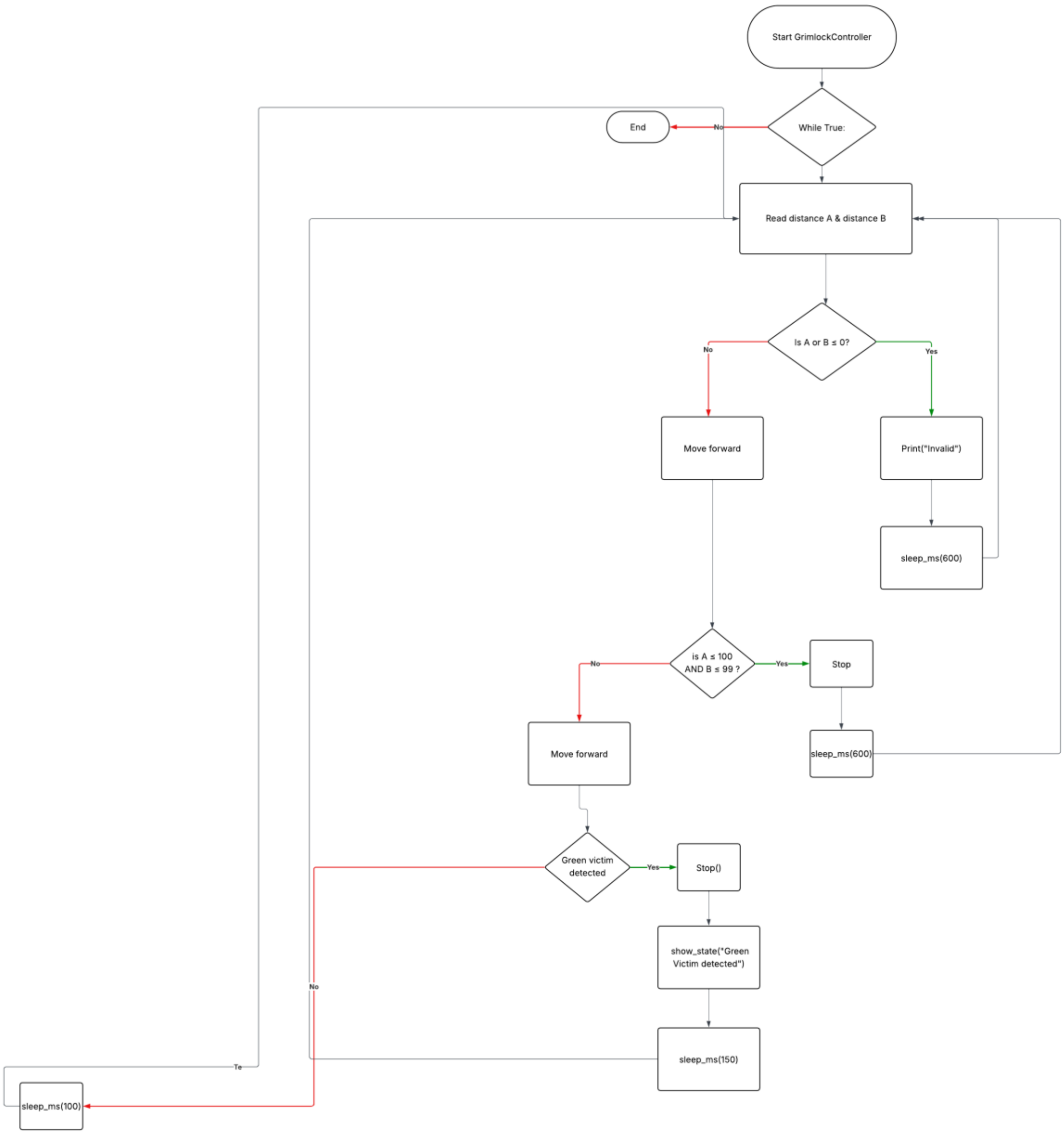
Data flow diagram 0:



Data flow diagram 1:

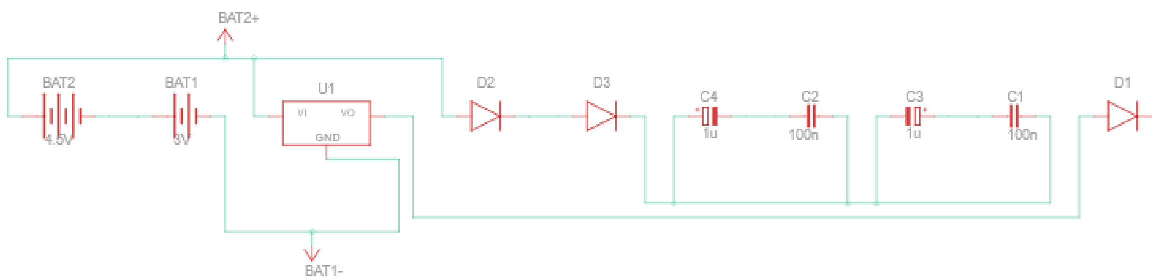


Flowchart



Start GrimlockController
While True:
No
End
Read distance A & distance B
Is A or B ≤ 0?
No
Yes
Move forward
Print("Invalid")
sleep_ms(600)
is A ≤ 100
AND B ≤ 99 ?
No
Yes
Stop
sleep_ms(600)
Move forward
Green victim
detected
Yes
Stop()
show_state("Green
Victim detected")
sleep_ms(150)
No
Te
sleep_ms(100)

Wiring diagram:



Material components list

- 2x Servo Wheels
- Arduino Uno
- HC-SR04 Ultrasonic Sensor
- 5 Channel Digital Line Finder or 1 or more Digital/Analog Line Finder
- Dupont Cables
- 2x 3.7v 18650 Polymer Lithium Ion Battery
- 2x LiPo holder
- XL4015 DC to DC Converter Module (alternative to use Diodes in series)
- 5x20 PCB Fuse holder
- 5x20 fuse
- 2x 100µF 25v capacitors
- Wood

- 4x male to male wires
- 4x male to female wires

Power supply calculations

Battery pack (2x 3.7V):
7.4V

2 capacitors (step down of ~0.5V each)
7.4V -> 6.4V

De amplifier (down to 5V)
6.4V -> 5V

Servos
5V received

Testing and Evaluation

Unit Test Figure 1

```python
from machine import Pin, PWM
from movement import Move
from servo import Servo
from time import sleep

lh_servo_pwm = PWM(Pin(16))
rh_servo_pwm = PWM(Pin(15))

# Use the same style as in movement.py unless you
left = Servo(pwm=lh_servo_pwm)
right = Servo(pwm=rh_servo_pwm)

movement = Move(left, right, True)

print("Forwards")
movement.forward()
sleep(1)
print("backward")
movement.backward()
sleep(1)
print("left")
movement.left()
sleep(1)
print("right")
movement.right()
sleep(1)
print("stop")
movement.stop()
sleep(1)
```

Unit Test Figure 2

```python
from colorsensor import colorvictim
from time import sleep_ms
from machine import Pin, PWM
from servo import Servo
from movement import Move

lh_servo_pwm = PWM(Pin(16))
rh_servo_pwm = PWM(Pin(15))
lh_servo = Servo(pwm=lh_servo_pwm)
rh_servo = Servo(pwm=rh_servo_pwm)
movement = Move(lh_servo, rh_servo, debug=True)

sensor = colorvictim("green", debug=True)

while True:
    if sensor.greenvictim():
        print("Green detected")
        movement.stop()
    else:
        print("Not the victim")
        movement.backward
    sleep_ms(1000)
```

Class Figure 3

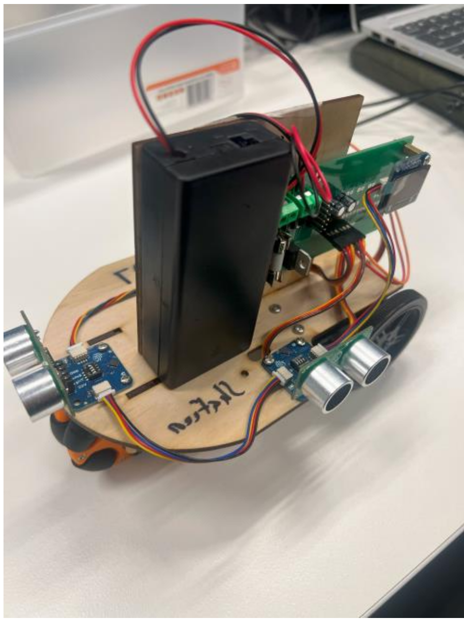
```python
from machine import Pin, PWM
from servo import Servo
from time import sleep_ms
from PiicoDev_Ultrasonic import PiicoDev_Ultrasoni
from PiicoDev_VEML6040 import PiicoDev_VEML6040
from colorsensor import colorvictim
from movement import Move
from lcd import show_state

class GrimlockController:
    def __init__(self):
        # Setup
        lh_servo_pwm = PWM(Pin(16))
        rh_servo_pwm = PWM(Pin(15))
        self.lh_servo = Servo(pwm=lh_servo_pwm)
        self.rh_servo = Servo(pwm=rh_servo_pwm)
        self.range_a = PiicoDev_Ultrasonic(id=[1, 
        self.range_b = PiicoDev_Ultrasonic(id=[0, 
        self.movement = Move(self.lh_servo, self.r
        self.sensor = colorvictim(debug=True)

# Main loop
    def run(self):
        while True:
            distancea = self.range_a.distance_mm
            distanceb = self.range_b.distance_mm
            if distancea <= 0 or distanceb <= 0:
                print("Invalid reading from ultras
                sleep_ms(100)
                continue  # Skip this loop
            self.movement.backward()
            show_state("forward")
            print("Distance A (Left):", distancea,
            if distancea <= 100 and distanceb <= 9
                self.movement.stop()
                show_state("stop")
                sleep_ms(600)
                if distancea > distanceb:
                    self.movement.left()
                    show_state("left")
                elif distancea < distanceb:
                    self.movement.right()
                    show_state("right")
                else:
                    self.movement.left()
                    show_state("left")
                sleep_ms(600)
            elif distancea <= 100 and distanceb >=
                self.movement.stop()
                show_state("stop")
                sleep_ms(600)
                self.movement.right()
                show_state("right")
                sleep_ms(600)
            elif distancea >= 101 and distanceb <=
                self.movement.stop()
                show_state("stop")
                sleep_ms(600)
                self.movement.left()
                show_state("left")
                sleep_ms(600)
            if self.sensor.greenvictim():
                show_state("Green victim detected!
                sleep_ms(450)
                self.movement.stop()
                show_state("stop")
                sleep_ms(450)
                continue  # skip the rest of the l
            sleep_ms(100)

controller = GrimlockController()
controller.run()
```

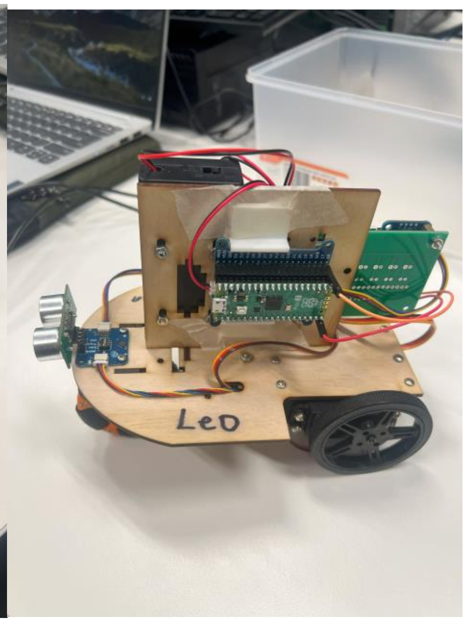
Class Figure 4

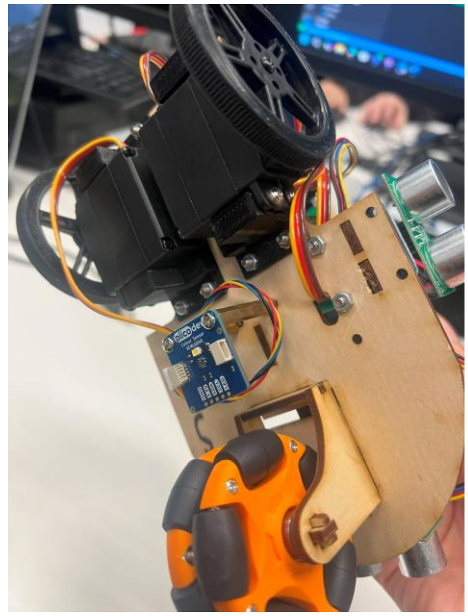
Class Figure 5

```python
from machine import Pin, PWM
from servo import Servo
from time import sleep_ms


lh_servo_pwm = PWM(Pin(16))
rh_servo_pwm = PWM(Pin(15))


freq = 50
min_us = 500
max_us = 2500
dead_zone_us = 1500


# Only create the servo objects you need
lh_servo = Servo(pwm=lh_servo_pwm)
rh_servo = Servo(pwm=rh_servo_pwm)


class Move:
    def __init__(self, lh_servo, rh_servo
        self.__lh_servo = lh_servo
        self.__rh_servo = rh_servo
        self.__debug = debug

    def forward(self):
        if self.__debug:
            print("forward")
        self.__lh_servo.set_duty(2500)
        self.__rh_servo.set_duty(500)

    def backward(self):
        if self.__debug:
            print("backward")
        self.__lh_servo.set_duty(500)
        self.__rh_servo.set_duty(2500)

    def left(self):
        if self.__debug:
            print("left (45 degrees)")
        self.__lh_servo.set_duty(500)
        self.__rh_servo.set_duty(500)
        sleep_ms(500)
        self.stop()

    def right(self):
        if self.__debug:
            print("right (45 degrees)")

        self.__lh_servo.set_duty(2500)
        self.__rh_servo.set_duty(2500)
        sleep_ms(500)
        self.stop()

    def stop(self):
        if self.__debug:
            print("stop")
        self.__lh_servo.stop()
        self.__rh_servo.stop()
```

```python
from PiicoDev_VEML6040 import PiicoDev_VEML6040
from PiicoDev_Unified import sleep_ms

colourSensor = PiicoDev_VEML6040()

class colorvictim:
    def __init__(self, debug=False):
        self.__debug = debug

    def greenvictim(self, threshold=150):
        data = colourSensor.readRGB()
        red = data['red'] # extract the RGB information from data
        grn = data['green']
        blu = data['blue']
        if self.__debug:
            print(str(blu) + " Blue  " + str(grn) + " Green  " +
        if grn > red + threshold and grn > blu + threshold: # thr
            return True
        else:
            return False
```







Justification of the object-oriented programming technique
How has polymorphism, encapsulation, inheritance, and abstraction been applied to the figures above

Figure 1
Encapsulation
- Servo object hides all PWM pin setup
- Move hides the sequence of servo action needed to go forward

Inheritance
- Servo inherited from the servo reusing the same setup logic

Polymorphism
- Move is designed to work with any motor like object

Abstraction
- Calls like movement.right() covers the servo angle timing

Figure 2
Encapsulation
- Colorvicitm, servo, move hider their own internal workings

Inheritance
- Servo inherited from the Servo class reusing the same setup logic

Polymorphism
- Move is designed to work with any motor like object

Abstraction
- High level calls like movement.forward() mask the complex servo and timing code

Figure 3

Encapsulation

- Servo, Move, PiicoDev_Ultrasonic, and colorvictim each hide complex hardware interactions

Inheritance

- Servo inherited from the Servo class reusing the same setup logic

Polymorphism

- Move is designed to work with any motor like object

Abstraction

- High level calls like movement.forward() and show_state mask the complex servo and timing code

Figure 4

Encapsulation

- Move bundles all movement behavior in one class

Inheritance

- Servo inherits behavior from the servo class

Polymorphism

- Move accepts any servo-like objects that implement. set_duty() and .stop().

Abstraction

- Hides the high level code e.g .forward()

Figure 5

Encapsulation

- Wraps all the logic for reading the color sensor and determining the green victim

Inheritance

- PiicoDev_VEML6040 handles low-level I2C communication and RGB data reading.

Polymorphism

- Any sensor class with a .readRGB() method returning the same data format could be plugged in

Abstraction

- -.greenvictim() is a simple, high-level action name that hides the details of threshold