OPTIMUS

ENGINEERING NOTEBOOK
GHANA ROBOTICS COMPETITION
Engineers League

University of Ghana
October 2025

1. Team Introduction

Team Members:

- Ethan Nartey, Programmer, Level 100
- Daniel Kweku Dade Botchway, Designer, Level 100
- Nelly Amewu, Builder, Level 200

2. Project Goals

2.1. League Challenge

Smart City Builders Challenge: Creating sustainable infrastructure, community services, and clean environments in growing African cities.

2.2. Problem Statement: Our robot must complete missions related to fixing infrastructure, building essential services, and managing urban waste to support sustainable city development.

2.3. Goals:

Complete the three required missions:

- **Fixing a Broken Bridge** (Road Infrastructure)
- **Building Essential Services** (School, Hospital, Workplace)
- **Cleaning the City** (Rubbish Collection)

Points are maximized by designing effective autonomous and manual control strategies.
Practice teamwork, problem-solving, and innovation while connecting the challenge to SDG 9 (Industry, Innovation, and Infrastructure) and SDG 11 (Sustainable Cities and Communities).

3. Brainstorming and Strategy

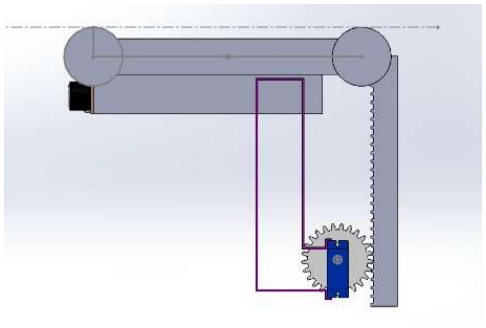
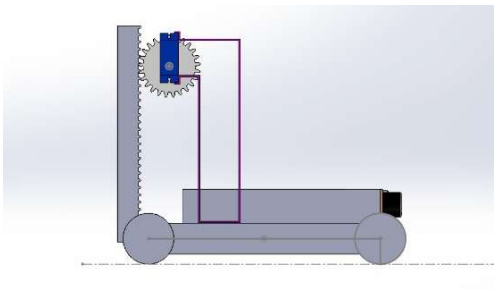
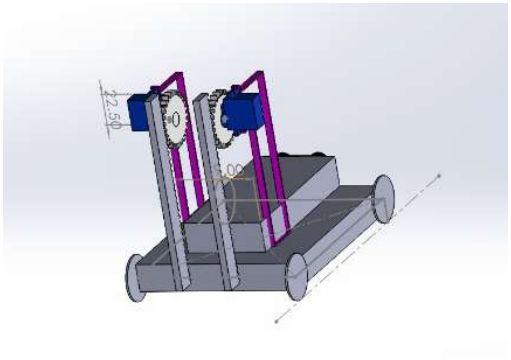
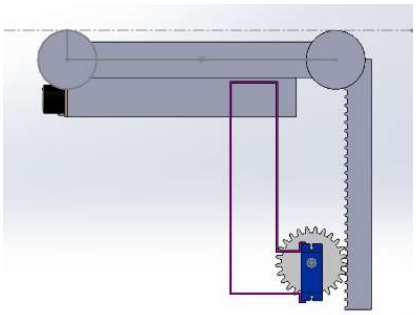
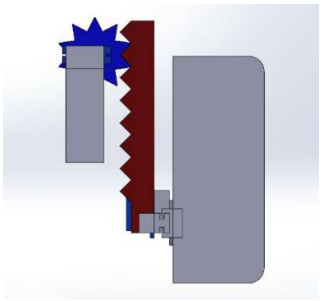
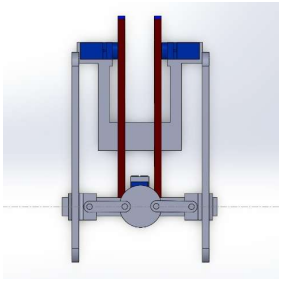
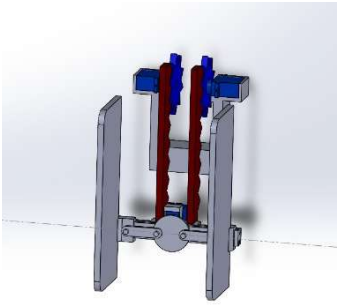
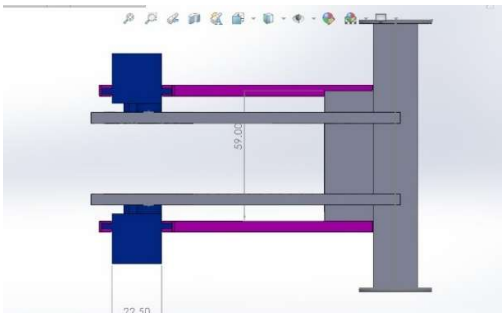
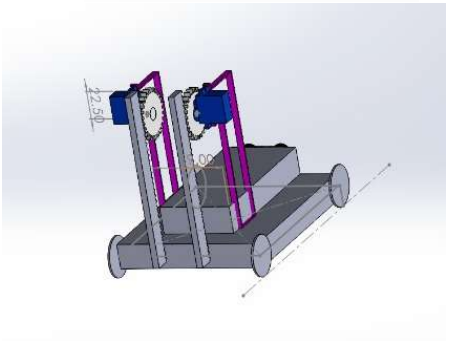
3.1. Initial Ideas:

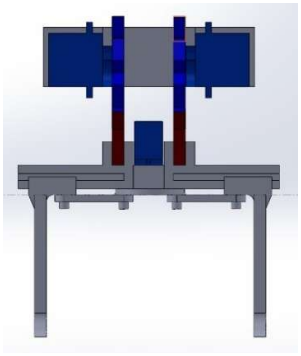
- We considered separating the tasks between each bot so they each specialized in some tasks. This idea was scrapped since teams will be randomized.
- We resolved to start by assembling a base driving robot with Xplore Bot parts from Firefly.io and tried to get the basic movements down before we started discussing attachments to gradually add.
- We explored our options for controlling the bot in manual mode using a Bluetooth module. We considered using the RC Bluetooth Controller app.
- We discussed, experimented with and designed some grab and lift mechanisms for building blocks such as a crane and a grabbing arm.
- We experimented with different turning methods (diagonal vs left-right motor control).
- We sketched and designed some potential 3D-printed interchangeable attachments.
- We explored VSCode, Thonny and block-based coding to see which is best for our work.

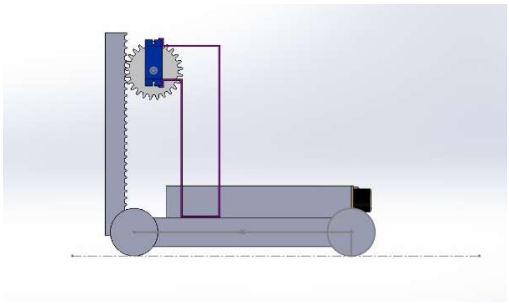
- We tried to analyse the rules and mat to find the simplest and quickest way to perform each task.
- We are considering a distance measurement system to improve autonomous accuracy.

3.2. Sketches

Sketches of base robot, lift mechanism, and attachment ideas.





Strategy Options

- Prioritize bridge repair in autonomous mode. This prevents the -30pts penalty, earns bonus points, and doubles our score for completing the task.
- Use manual mode to stack building blocks for schools/hospitals/workplaces. This is an intricate and specific task granting scalable points with multipliers for taller structures.
- Considering writing code for the bot to build in autonomous mode, but only simple structures.
- Fall-back option for autonomous mode is rubbish collection in case our alliance partner is also doing the bridge in autonomous.
- Assign remaining time to rubbish collection since this has lower points per action but grants steady points, is reliable and easy to repeat, and thus easier to program and may not require complex attachments.

3.3. Pros and Cons

- Bridge repair first: High reward and safety bonus; requires speed and precision.
- Building essential services: High scoring potential with multipliers if we can do it autonomously; more complex, risk of time overruns.
- Rubbish collection: Simple and consistent; low points compared to other tasks.

3.4. Chosen Idea and Why

- The team decided to focus on bridge repair during autonomous since it prevents penalties, secures bonus points, and earns double. We will fall back on rubbish collection in autonomous mode in case our alliance partner handles bridge repair.
- In manual mode, effort will shift to building essential services (for big multipliers) while using leftover time for rubbish collection to secure easy additional points.
- This approach balances high-value tasks (bridge and buildings) with low-risk scoring (rubbish), ensuring steady results.

4. Design Process

For each robot version, we followed the VEX Design Process, which is: Define the problem, Brainstorm, Research, Develop Ideas, Choose the Best Solution, Build a Prototype, Test and Evaluate, Improve, then finally, Share.

Here are our weekly progress and versions as we progressed in our development:

Week 1

- Built the base robot.

- Familiarized with programming interface(Thonny).
- Read the challenge document and developed strategy.
- Started designing extra parts needed for the challenge.
- Took inventory of faulty parts.

Week 2

- Agreed on a game strategy.
- Discussed grab and lift mechanism.
- Figured out pins for front motors
- Tested turning methods. Initially used diagonal wheel control, but improved turning by controlling wheels side-by-side (left vs right). Turning became smoother and sharper.
- Built a prototype grabbing mechanism

Week 3

- Fixed the forward alignment of the bot, making the bot moved straight.
- Modelled a prototype of the lifting mechanism.
- Looked into using blocks code editor for the bot. Could be easier to program the servos and other motors with it.
- Attempted to measure distance with the bot.

Week 4

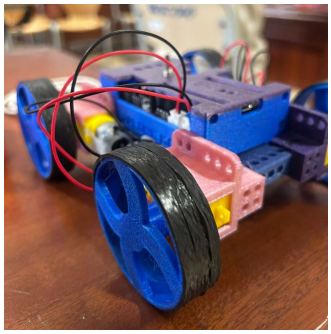
- Changed DC motor in the attachment to a server motor.
- Compiled an engineering notebook.
- Fixed the bot's drift.
- Attached an ultrasonic sensor.

Week 5

- Tested the ultrasonic sensor
- Finalized attachment designs
- Finalized documentation

Version 1 – 23/09/2025

- Sketch/Photo

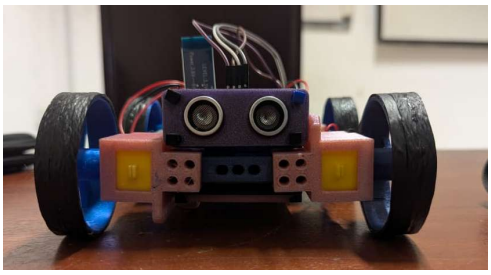


- Testing: Basic movement and turning mechanism
- Code structure:

- Components Used: Raspberry pi pico, Wheels, Motors
- Design Changes: Turning mechanism
- Testing Results: we could connect to the Thonny IDE and run commands, as well as configured the motor pin numbers.

Version 2 – 07/10/2025

- Sketch/Photo



- Testing: Measure distance using the ultrasonic sensor, connect to a mobile remote controller using the Bluetooth module
- Components Used: Ultrasonic sensor, Bluetooth module
- Design Changes: Added an ultrasonic sensor and Bluetooth module
- Testing Results: we were able to work with distances and write code dependent on it thanks to the IR sensor, as well as control our bot using a mobile phone thank to the bluetooth module.

5. Programming

5.1. Code Environment: Visual Studio Code, Xplore IDE

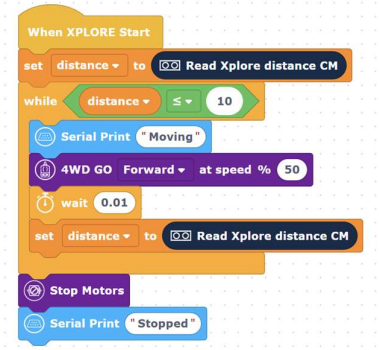
5.2. Sensors/Motors: Initially used diagonal motor control, later switched to left vs right wheel control for smoother turns. We implemented n ultrasonic sensor for discerning distances.

5.3. Code Snippets:

```
from machine import PWM, UART, Pin
from time import sleep
import utime

# Set the relay pins as output pins
# motor 1
ENA = PWM(Pin(10))
m1_forward = Pin(12, Pin.OUT)
m1_back = Pin(11, Pin.OUT)

# motor 2
ENB = PWM(Pin(13))
m2_back = Pin(14, Pin.OUT)
m2_forward = Pin(15, Pin.OUT)

# motor 3
ENC = PWM(Pin(26))
m3_forward = Pin(17, Pin.OUT)
```

```
m3_back = Pin(16, Pin.OUT)

# motor 4
END = PWM(Pin(21))
m4_forward = Pin(19, Pin.OUT)
m4_back = Pin(18, Pin.OUT)

# Bluetooth setup
uart = UART(0, 9600)

# speed of the bot
speed = 65025  # 0 - 65025

ENA.duty_u16(55000)
ENB.duty_u16(52000)
ENC.duty_u16(55000)
END.duty_u16(52000)

def forward():
    m1_forward.on()
    m1_back.off()
    m2_back.off()
    m2_forward.on()
    m3_forward.on()
    m3_back.off()
    m4_forward.on()
    m4_back.off()


def back():
    m1_forward.off()
    m1_back.on()
    m2_back.on()
    m2_forward.off()
    m3_forward.off()
    m3_back.on()
    m4_forward.off()
    m4_back.on()


def right():
    m1_forward.on()
    m1_back.off()
    m2_back.on()
    m2_forward.off()
    m3_forward.on()
    m3_back.off()
    m4_forward.off()
    m4_back.on()
```

```
def left():
    m1_forward.off()
    m1_back.on()
    m2_back.off()
    m2_forward.on()
    m3_forward.off()
    m3_back.on()
    m4_forward.on()
    m4_back.off()


def stop():
    m1_forward.off()
    m1_back.off()
    m2_back.off()
    m2_forward.off()
    m3_forward.off()
    m3_back.off()
    m4_forward.off()
    m4_back.off()


while True:
    if uart.any():
        value = uart.readline()
        print(value)

        if value == b"F":
            forward()
        elif value == b"B":
            back()
        elif value == b"L":
            left()
        elif value == b"R":
            right()
        elif value == b"S":
            stop()
```



6. Challenges and Solutions

- **Programming Issues:**
  - Turns were not behaving as expected for the rear wheels, but we fixed that when we switched to using blocks to code.
  - Xplore IDE was not defining variables in the blocks to python conversion.
- **Mechanical Issues:**
  - Initial turning method (diagonal wheels) caused rough, inefficient turns. We switched to left/right motor configuration, resulting in smoother, sharper turning.
  - Some components that we were given in the Xplore bot box were of low quality (incorrectly printed wheel bridge, busted battery case, lose wheels in the motor, to mention a few) leading to ill-fitted parts and some discrepancies that affect the performance of our bot.
  - Wobbly Wheels.
- **Hybrid Issues:**
  - Wobbly wheels, so the bot couldn't move in a straight line, paired with excessive power in one set of motors. Solved by changing the motor holders and reducing the power in the left set of motors.

7. Teamwork and Roles

7.1. Collaboration Notes:

All members of both teams from the University of Ghana brainstormed strategy together and divided early tasks (base building, programming exploration, mechanism design documentation) effectively, with everyone participating in each part whilst each specializing in their assigned roles

7.2. Individual Learnings:

- Importance of testing multiple approaches (turning logic) and cross-verifying results.
- Refreshing our knowledge of the blocks code editor
- Programming a raspberry pi pico

8. Final Robot Design

- **Description**

  The final version of our robot, Optimus V2, is a compact and stable four-wheeled robot designed for both autonomous and manual control modes. It uses a Raspberry Pi Pico microcontroller as its brain, connected to four DC motors configured in a left-right

wheel control system for smoother and sharper turns. The body is built on the Xplore Bot chassis, with improved motor holders for stability and alignment.

An ultrasonic sensor is mounted at the front of the robot to detect obstacles and measure distances during autonomous runs. This allows the robot to approach or stop precisely when interacting with mission elements like the bridge or rubbish bins. A Bluetooth module is integrated for manual control via a mobile RC Controller app, providing easy manoeuvring during manual game segments.

We utilize interchangeable attachments, such as a lifting arm for building essential services and a scoop-like attachment for rubbish collection. These can be swapped easily depending on the mission requirements.

- **Effectiveness**

  This final robot design effectively fulfils the team's strategy by balancing stability, modularity, and precision:

  - The ultrasonic sensor improves the accuracy of autonomous tasks like bridge repair and rubbish collection.
  - The Bluetooth control system allows responsive and smooth manual driving for stacking and positioning during building missions.
  - The compact structure ensures the bot can navigate tight spaces on the challenge mat without colliding with obstacles.
  - The interchangeable attachment system gives flexibility to adapt to different missions quickly, without needing to rebuild the bot.
  - Adjusted motor speeds and wheel alignment corrections ensure consistent straight movement and precise turning, critical for both scoring reliability and time efficiency.

  Overall, the robot performs consistently, is easy to control, and can complete its core missions efficiently under competition conditions.

- **Trade-offs**

  - Speed vs. Control: To maintain stability and precision, we slightly reduced motor speed. This makes turns smoother but sacrifices some acceleration.
  - Dimension Restrictions: some attachments increased the robot's dimensions slightly, requiring fine-tuning of attachment designs.
  - Complexity vs. Reliability: We opted against adding advanced automation for building tasks in favour of a simpler, more reliable manual stacking process.
  - Material Quality: Some components from the Xplore kit were imperfectly manufactured (e.g., misaligned wheel bridge, loose fits). We compensated through realignment and reinforcement but couldn't achieve perfect symmetry.

  Despite these trade-offs, Optimus V2 achieved a strong balance between precision, durability, and flexibility enabling it to complete multiple challenge missions effectively and consistently.