MINISTRY OF EDUCATION

GHANA ASSOCIATION OF SCIENCE TEACHERS

ROBOTICS

for Senior High Schools

Year 3



Isaac Nzoley
Dr. Elieel Keelson
Dr. Kwame Oteng Gyasi

MINISTRY OF EDUCATION

GHANA ASSOCIATION OF SCIENCE TEACHERS

ROBOTICS

for Senior High Schools

Year 3

Isaac Nzoley
Dr. Elieel Keelson
Dr. Kwame Oteng Gyasi

Ghana Education Service (GES)



© Ministry of Education 2026

This publication is not for sale. All rights reserved. No part of this publication may be reproduced without prior written permission from the Ministry of Education, Ghana.

Contents

FOREWORD....iv

ROBOTS AND SOCIETY 3....1

Principles of Robotic Systems....2

ROBOT CONTROL PRINCIPLES 3....23

Principles of Robotic Systems....24

SENSORS AND ACTUATORS 3....47

PRINCIPLES OF ROBOTIC SYSTEMS....48

DIGITAL AND ANALOGUE SYSTEM DESIGN 3....66

Digital and Analogue System Design....67

ROBOT PROGRAMMING....78

Programming Robots in Python....79

FOREWORD

Ghana's new Senior High School Curriculum aims to ensure that all learners achieve their potential by equipping them with 21st Century skills, knowledge, character qualities and shared Ghanaian values. This will prepare learners to live a responsible adult life, progress to further studies and enter the world of work. This is the first time that Ghana has developed a Senior High School Curriculum which focuses on national values, attempting to educate a generation of Ghanaian youth who are proud of our country and can contribute effectively to its development.

The Ministry of Education is proud to have overseen the production of these Learner Materials which can be used in class and for self-study and revision. These materials have been developed through a partnership between the Ghana Education Service, teacher unions (Ghana National Association of Teachers- GNAT, National Association of Graduate Teacher -NAGRAT and the Coalition of Concerned Teachers- CCT) and National Subject Associations. These materials are informative and of high quality because they have been written by teachers for teachers with the expert backing of each subject association.

I believe that, if used appropriately, these materials will go a long way to transforming our Senior High Schools and developing Ghana so that we become a proud, prosperous and values-driven nation where our people are our greatest national asset.

Haruna Iddrisu MP

Minister for Education

SECTION

1

ROBOTS AND SOCIETY 3



Principles of Robotic Systems

INTRODUCTION

In previous years, the role of robots in today's world was explored, with a focus on how they affect people, work, and society. Robotic systems, their parts, and their differences from non-robotic systems were also introduced. In this section, knowledge from the previous years is built on. Problems in Ghana and around the world are identified. Ways robots and automated systems can help are explored. Problems in homes, schools, and workplaces are written down and shared. Ideas for how robots can solve them are suggested. Flowcharts and pseudocode are used to show the steps a robot should follow. Simple technical reports are prepared. These reports are read and reviewed by classmates. Helpful feedback is given to improve the work. Clear thinking and teamwork are encouraged throughout.

KEY IDEAS

- Robots improve safety by reducing human exposure to hazardous tasks, increasing efficiency in production processes, and enhancing quality of life by automating mundane (ordinary and repetitive) activities.
- Engaging in feedback on technical reports helps clarify ideas, encourages critical thinking, and improves the ability to present solutions effectively.
- Robots are well-suited for addressing challenges that involve dangerous environments, repetitive tasks that require consistency, and applications where precision is essential.
- Understanding how components like sensors, actuators, and control systems work together is crucial for designing effective robotic systems.
- Recognising the differences between robotic and non-robotic systems clarifies where automation can maximise productivity and efficiency.
- Creating flowcharts and pseudocode helps outline systematic approaches for programming robots and is essential for successful implementation and troubleshooting.

FINDING PROBLEMS ROBOTS CAN SOLVE

Welcome back to the exciting world of robotics! With your two years of robotics experience, you already know how robots work and what they can do. Now, let us explore how to use your skills to solve real-world problems with robots, making life safer, easier, and more inclusive for everyone.

WHAT MAKES A PROBLEM SUITABLE FOR A ROBOT?

You have built robots and programmed them, so you know they are not for every task. Robots shine when problems are too dangerous, repetitive, or precise for humans.

Here are the key characteristics of problems your robots can address, using your skills from Years 1 and 2:

DANGEROUS ENVIRONMENTS

Robots can work in risky places, such as areas affected by galamsey (illegal mining) or fire-damaged markets. For example, a robot you design could inspect unsafe mining sites, keeping people out of harm.



Figure 1.1: Mining in a dangerous environment

REPETITIVE TASKS

Robots love doing the same task repeatedly, like sorting cocoa beans or packaging goods in a market. Using your Year 1 and Year 2 skills, you can program robots to perform these tasks accurately, saving time and reducing errors.

HIGH PRECISION AND ACCURACY

Some tasks need exact movements, like assembling tiny circuits, planting seeds evenly or assisting in surgery. With your experience programming precise movements, you can make robots handle these jobs perfectly every time.

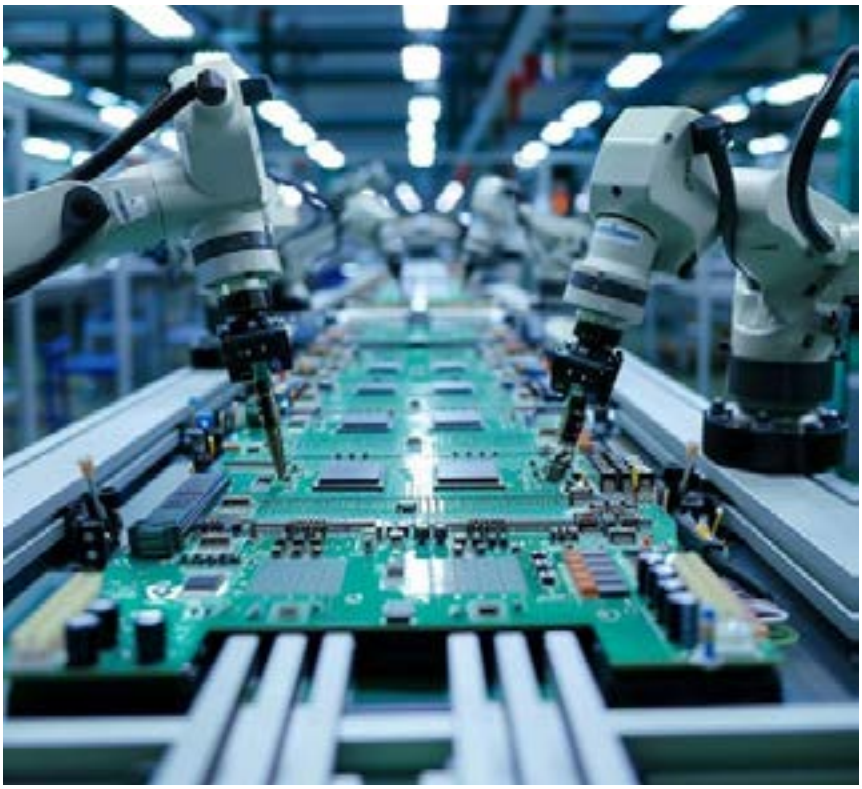


Figure 1.2: Robots assembling tiny electronic circuits. [https://stockcake.com/i/robotic-assembly-line_714539_1059887]

HEAVY LIFTING AND PHYSICAL STRAIN

Lifting heavy items, such as yam in a market, can cause injuries. Robots with strong actuators (like the ones you studied in Year 1) can do it easily, thereby protecting workers.



Figure 1.3: Robot lifting heavy objects [https://www.tinsleycompany.com/equipment-systems/palletizing-machines-robots/]

CONTINUOUS OPERATION

Robots can work without stopping, unlike humans who need rest. Your Year 2 programming skills can help a robot monitor farms or markets non-stop, improving productivity and safety.

INACCESSIBLE LOCATIONS

Robots can go where humans cannot, like narrow water pipes or flooded areas after heavy rains. Your knowledge of mobility (wheeled or legged robots from Year 1) and navigation (Year 2) can help you design robots for these tough spots.

DATA COLLECTION AND MONITORING

Using sensors from Year 1, robots can gather information, such as checking soil quality on farms or monitoring air pollution in cities. Your Year 2 programming can analyse this data to help communities.

Case Study

Think of a challenge in your community, such as cleaning a polluted stream or sorting market goods. How could a robot you design solve it?

HOW ROBOTS SOLVE PROBLEMS: USING YOUR ROBOTICS SKILLS

With two years of robotics under your belt, you know robots have four key abilities: sensing, movement, manipulation, and decision-making. Let us see how these abilities, combined with your skills, can solve problems in Ghana while being inclusive:

SENSING

You have worked with sensors in Year 1 to detect things like light, temperature, or objects. Robots with sensors can monitor pollution in a river or detect smoke in markets, just like assistive devices that help people with visual impairments sense their surroundings.



Figure 1.4: Robot sensors [https://sproboticworks.com/blog/Sensors]

MOVEMENT AND NAVIGATION

From Year 1, you know how to make robots move with wheels or legs and in Year 2, you learned to program navigation. These skills can help you create robots that deliver food supplies to rural areas or navigate crowded markets, helping people in wheelchairs access goods.



Figure 1.5: Robot Navigating through vineyard [https://robotechvision.com/autonomous-navigation-in-the-vineyard-using-a-visual-system/]

MANIPULATION AND INTERACTION

You have built robot arms or grippers and programmed them in previous years to pick up or sort objects. These skills can create robots that assemble products or sort waste, assisting workers.



Figure 1.6: Robotic arm for manipulation [https://www.reichelt.com/de/en/shop/product/robotic_arm-148824]

COMPUTATION AND DECISION MAKING

Programming skills, like flowcharts and feedback loops, let robots analyse data and make decisions. For example, a robot you program could alert farmers about low soil moisture, or help someone with a cognitive disability by automating daily tasks

Case Study

Sketch or describe a robot that uses one of these abilities to solve a problem in your community, such as monitoring crops or delivering supplies.

REAL-WORLD ROBOTIC HEROES

Your robotics skills are already being used worldwide and in Ghana to solve big problems. Here are examples of how robots are making a difference, building on your knowledge:

AGRICULTURE

Robots can plant yams, monitor cocoa crops with cameras, or water fields precisely. In Ghana, these robots help farmers, including those with disabilities who use voice-activated systems, increase yields, and save time.

HEALTHCARE

Robots assist in precise surgeries or deliver medicines in hospitals, making healthcare accessible for all, including patients with mobility challenges.

INDUSTRY AND MANUFACTURING

Robots weld, assemble, or inspect products in factories, improving safety and creating "tech" jobs for people, including those with disabilities, to program or maintain them.

ENVIRONMENT

Robots clean plastic from beaches, monitor air quality, or protect wildlife. In some parts of Africa, drones track poachers to save animals, helping conservation efforts.

SAFETY AND SECURITY

Drones monitor dangerous areas like galamsey sites, and robots help in search and rescue after floods, keeping everyone safe.

LOGISTICS AND TRANSPORTATION

Robots move goods in warehouses or deliver packages to remote areas, improving access for rural communities.

Case Study

In Ghana, drones deliver medical supplies to rural clinics in the Oti Region, ensuring people receive medicines quickly. How could a robot you design improve life in your community?



Figure 1.7: Drone delivering medical supplies. [https://www.pharmaceutical-technology.com/news/zipline-ghana-medical-supplies-drones/]

Activity 1.1 Identifying & Proposing Robotic Solutions for Your Community

You are to undertake this activity together with 2 of your classmates.

Objective: Use your Year 1 and Year 2 robotics skills, such as sensing, movement, manipulation, and decision-making, to propose robot solutions for any of your community's problems.

Materials: Paper, pens, markers, digital devices (optional)

Instructions

Part 1: Find Local Problems (15 minutes)

1. In your groups, list 3–5 problems in your community (e.g., school, market, farm). Focus on the following.
    a. Dangerous tasks (e.g., cleaning polluted rivers).
    b. Repetitive tasks (e.g., sorting cocoa beans).
    c. Precise tasks (e.g., planting seeds).
    d. Heavy lifting (e.g., moving market goods).
    e. Continuous work (e.g., monitoring air quality).
    f. Hard-to-reach places (e.g., inspecting pipes).

Part 2: Propose Robot Solutions (20 minutes)

1. Group Problems: Sort problems into categories (e.g., environment, healthcare, agriculture, safety).
2. Match to Robotics: For each problem, suggest a robot task using your Year 1 and Year 2 skills:
    a. Sensing: e.g., a robot with sensors monitors water quality.
    b. Movement: e.g., a drone delivers medicines.
    c. Manipulation: e.g., a robot arm sorts fish.
    d. Decision-Making: e.g., a robot alerts farmers to low soil moisture.

Part 3: Share Ideas (15 minutes)

1. Deliver a Mini-Presentation: For your chosen problem and robotic solution, give a presentation (1–2 minutes):
    a. Outline the problem and category.
    b. The proposed robot and how it will achieve the task(s).
    c. The benefits to society that using a robot in this context will bring.
2. Presentation methods: Use a poster presentation, a slideshow, or give a verbal explanation.

Part 4: Reflection and class feedback (5 minutes)

Discussion: State clearly how your robot solution(s) will address your community problems. Ask for comments and feedback from the class.

REPORTING ROBOT IDEAS FOR LOCAL CHALLENGES

UNDERSTANDING PROBLEMS IN DEPTH

The following page will guide you to produce a more detailed response to proposing a robotic solution to a local challenge or problem. It also shows how your initial thoughts, ideas and solutions can be presented as a professional report.

To create effective robotic solutions, you must first understand a problem fully. This means examining a challenge in your community, such as in agriculture, healthcare, or waste management, by answering key questions:

1. What is the problem? Describe it clearly. For example, plastic waste in Accra's gutters causes flooding. Where does it happen? Who is affected?
2. Why does it happen? Identify causes, such as a lack of waste collection systems.
3. What is the impact? Explain effects, like health risks or blocked paths for wheelchair users.
4. What solutions exist? Note current methods, such as manual cleaning, and why they fail, like being slow or unsafe.

Use community observations, interviews, or local news to gather information. This analysis helps you see why a robot might be the best solution.

Discussion

Discuss a problem in your school or community, like sweeping your school's large compound or cleaning markets. Write or record your thoughts.

DEFINING THE ROBOT'S ROLE

After analysing the problem, decide what specific task a robot could perform. Use your knowledge (sensors, motors, actuators) and skills (programming, navigation) from Years 1 and 2 to:

1. Identify the Task: Choose a clear action, like a robot collecting waste from gutters or sorting yam by size.
2. Match to Robot Skills: Link the task to robot abilities. For example, waste collection needs a robot with grippers (manipulation) and navigation to work in dirty gutters safely.

3. State the Role: Write a simple statement, like "A robot collects plastic from Accra's gutters autonomously" or "A robot sorts ripe mangoes in a market."

Discussion

Pick a problem, like delivering medicines to rural clinics. Suggest a robot task using your robotics skills.

RECAP: WRITING A STRUCTURED REPORT

Your Year 2 report-writing skills help you communicate ideas clearly. A structured report explains the problem and the robot's role to others, such as teachers or community leaders. A structured report can be written using this simple format:

1. Title Page: Name the problem (e.g., "Cleaning Gutters in Kumasi") and list group members and the date.
2. Table of Contents: List sections with page numbers for easy navigation.
3. Introduction: Briefly describe the problem (e.g., polluted rivers) and the report's purpose (proposing a robot solution).
4. Problem Statement: Explain where the problem occurs (e.g., Kejetia market) and the current situation (e.g., manual sorting is slow). Detail causes (e.g., lack of equipment), impacts (e.g., health risks), and why current solutions fail.
5. Why a robot? Explain why a robot suits the task, linking to your Year 1 and Year 2 skills (e.g., sensors for detecting waste, programming for navigation).
6. Proposed Robotic Role: State the robot's task (e.g., "A robot monitors crop health with sensors").
7. Benefits: Describe improvements, like safer streets or easier access for people with disabilities.
8. Conclusion: Summarise the problem, robot's role, and main benefit.
9. Appendices: Add sketches or data, like a diagram of the robot's gripper.

PRESENTING YOUR IDEAS

Sharing your ideas clearly is key. Follow these tips to effectively present your ideas:

1. Organise Clearly: Follow the report structure (problem, analysis, robot role, benefits).
2. Use Simple Language: Avoid complex terms or explain them clearly.
3. Add Visuals: Use drawings or slides with high-contrast colours for accessibility.
4. Engage Everyone: Express yourself clearly, show enthusiasm, and answer questions.
5. Highlight Impact: Explain how the robot helps all.

Discussion

Form a discussion group which follows the guidance above to present a robot idea. One possible idea could be developing a drone delivering supplies to Kete Krachi in the Oti Region. Talk about your ideas, including what sensors and power sources the drone would have, and what you would include in your report. What are the advantages and disadvantages of using drones to deliver different types of supplies – medicines, food, machine parts, letters?

Activity 1.2 Creation of a Robotic Solution for a Specific Problem

You are to perform this activity using the same groups as used in Activity 1.1.

Objective: Use your robotic skills to propose a robot solution to one of the problems listed in step 1 and follow the guidance above, making sure your proposals and solutions are critically presented and reasoned.

Materials: Paper, pens, markers, digital devices (optional).

Instructions

Step 1: Choose and Analyse a Problem (15 minutes)

1. Select a Problem: In your group, choose one problem in your community (e.g., school, market, farm) related to waste, agriculture, healthcare, or safety. Examples:
    a. Plastic waste in gutters is causing floods.
    b. Slow sorting of crops in markets.
    c. Limited medicine delivery to rural areas.
2. Analyse the Problem: Answer these questions (write or record):
    a. What is the problem? Describe it (e.g., plastic clogs gutters in Accra).
    b. Who is affected? People whose activities are hindered (e.g., floods block wheelchair paths for persons with disabilities).
    c. Why does it happen? List causes (e.g., no waste collection system).
    d. What is the impact? Note effects (e.g., health risks, flooded roads).
    e. Current solutions? Describe what is done now and why it fails (e.g., manual cleaning is slow).

Step 2: Propose a Robot Solution (10 minutes)

1. Define the Robot's Task: Suggest one specific task the robot will do, using your sensors, motors, programming, and navigation skills. Examples:
    a. A robot with grippers collects gutter waste, navigating autonomously.
    b. A drone delivers medicines to rural clinics, avoiding obstacles.

2. Explain why a robot is required for the task: Link the task to robot abilities (e.g., works in dangerous areas, performs repetitive tasks). Example: A robot cleans gutters safely, unlike humans.

Step 3: Create a Mini-Report (10 minutes)

Write a Short Report: This can be submitted in either softcopy or hardcopy, including:

a. Title: Name the problem (e.g., "Cleaning Gutters in Makola").

b. Problem: Summarise the problem, causes, and impact.

c. Proposed Robotic Role: State the robot's task and capability used.

d. Benefits: Explain how it helps everyone in the community.

Step 4: Present and Discuss (10 minutes)

1. Present: Share your report with another group (1 minute), explaining the problem, the robot's role, and the benefits. Use a poster or verbal explanation.
2. Discuss: Share with your group: How does your robot make Ghana inclusive? Share one idea with the class.

REPORTING ROBOT SOLUTIONS TO IDENTIFIED PROBLEMS

PLANNING ROBOT TASKS

To make a robot work well, you need a clear plan of its actions, like a map for a journey. This plan, called an operational sequence, lists the steps the robot takes to complete a task, such as collecting waste or sorting crops. Planning ensures your robot works smoothly without errors, using your sensors, motors, programming, and navigation skills.

TOOLS FOR PLANNING: FLOWCHARTS AND PSEUDOCODE

You have learned to use flowcharts and pseudocode to plan how you will control a robot to perform tasks. These tools help you outline the robot's actions to solve a problem in your community, like sweeping a large compound or delivering medicines.

RECAP: Flowcharts and Pseudocode

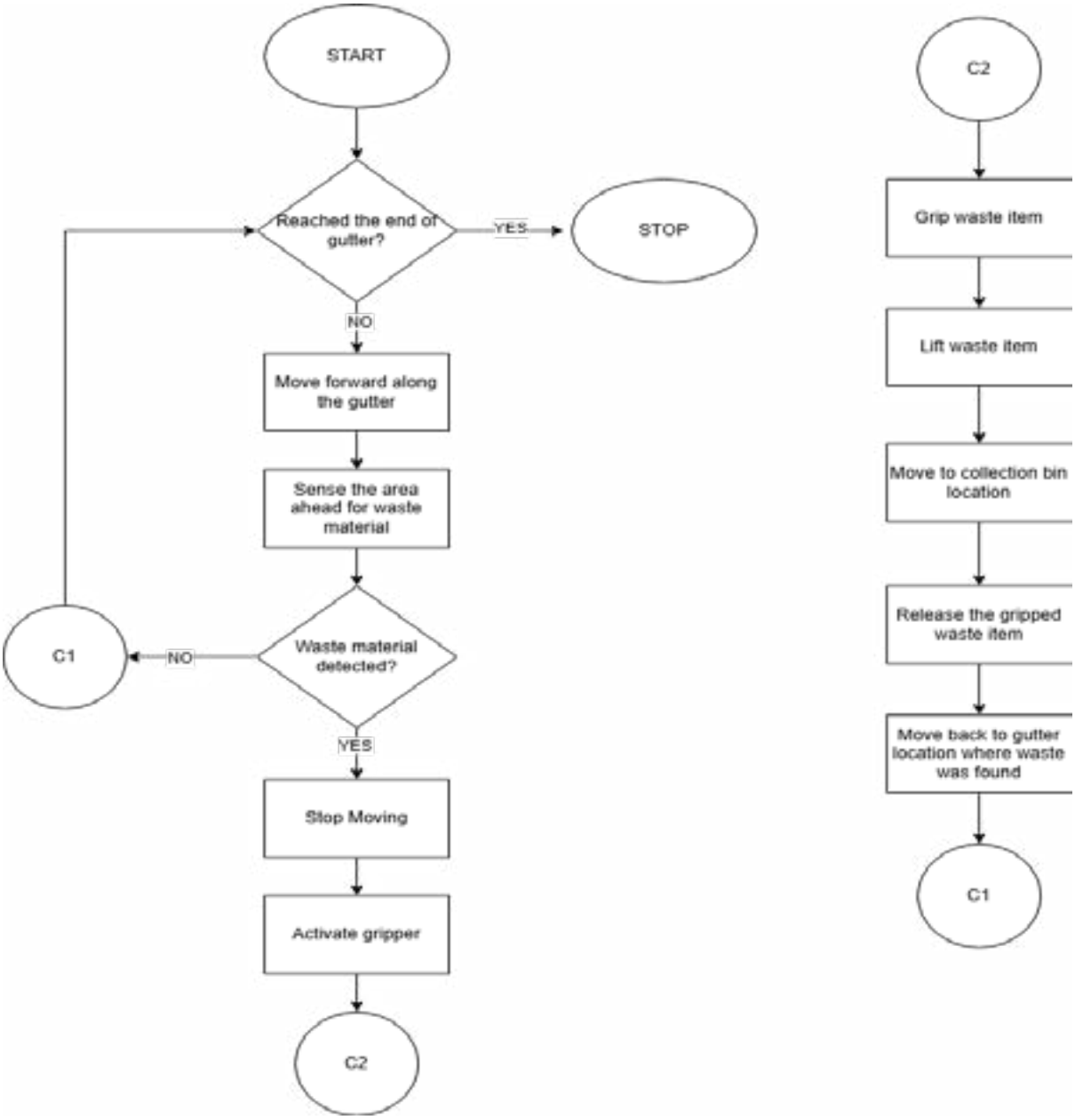
FLOWCHARTS

Flowcharts are visual maps showing a robot's steps, decisions, and loops. They use shapes:

1. Oval: Start or end of the sequence.

2. Rectangle: An action, like “Move Forward” or “Grip Waste”.
3. Diamond: A decision, like “Is Waste Detected?” with Yes/No paths.
4. Parallelogram: Input (e.g., sensor reading) or output (e.g., display message).
5. Arrow: Shows the flow of actions.
6. Circle (Connector): Links parts of a flowchart, especially for large plans. On-page connectors join points on the same page; off-page connectors link to another page.

Example: A flowchart for a robot collecting gutter waste might follow the sequence below. It can be observed that the plan is large, and the flowchart has thus been broken into two sections using the circles (connectors) C1 and C2.



Pseudocode

Pseudocode is a text-based plan using simple words to describe actions, like a recipe. It uses:

1. Sequence: List steps in order, e.g., "Move Forward, Sense Waste".
2. Input/Output: Get data or show results, e.g., "GET sensor_reading" or "DISPLAY message".
3. Decisions: Use IF, THEN, ELSE, e.g., "IF obstacle_detected THEN Turn Left ELSE Move Forward END IF".
4. Loops: Repeat actions, e.g., "REPEAT 5 TIMES Grip Item END REPEAT".

Example Pseudocode (for gutter-cleaning robot):

START

REPEAT UNTIL end of gutter

Move forward

Sense for waste

IF waste detected THEN

Stop

Grip waste

Lift waste

Move to bin

Release waste

Return to gutter

END IF

END REPEAT

STOP

Discussion

Design a simple flowchart or pseudocode for a robot sorting mango in a market.

HOW TO PLAN A ROBOT TASK

1. Define the Task: Know the robot's job, e.g., collecting waste (from your previous lesson).
2. List Steps: Break the task into small actions, like sense, move, and grip.
3. Add Decisions and Loops: Include choices (e.g., "Is path clear?") or repeats (e.g., check multiple areas).

4. Choose a Tool: Use a flowchart for visual planning or pseudocode for text. You can use both!
5. Create the Plan: Draw the flowchart with correct shapes or write clear pseudocode.

Discussion

Plan a robot task for delivering medicines to a rural clinic. Use a flowchart or pseudocode.

INCLUDING PLANS IN A TECHNICAL REPORT

Your flowchart or pseudocode goes in your technical report (from Year 2) to show how the robot works. Add it as a section called "Operational Sequence Outline" after the "Proposed Robotic Role" section. For simple plans, include it directly in the "Proposed Robotic Role" section as a Figure or text. Ensure it:

1. matches the robot's task.
2. uses clear shapes (flowchart) or words (pseudocode).
3. includes connectors for large flowcharts.

Follow this example which will guide you into producing a quality technical report:

Technical Report Cover Page

Title: Robotic Gutter Cleaner

Student Name: Kwame Amanfoo

School: Mikrobot School

Date: 16/7/2025

[START ON A FRESH PAGE]

Table of Contents

1. Introduction
2. Problem Statement
3. Proposed Robotic Role
4. Operational Sequence Outline
5. Potential Impact and Benefits
6. Conclusion
7. Appendices

[START ON A FRESH PAGE]

1. **Introduction**

This section introduces the problem of plastic waste blocking gutters in urban areas and explains the aim of proposing a robotic solution.

2. **Problem Statement**

The problem occurs in neighbourhood gutters in Accra, where waste builds up and blocks water flow, causing flooding. Manual cleaning is unsafe and slow. This problem is dangerous and repetitive, making it suitable for robots. Robots can move along gutters, detect waste using sensors, and collect waste without harm.

3. **Proposed Robotic Role**

The robot will move along a gutter, detect plastic waste using a camera or sensor, collect the waste with a gripper, and drop it into a bin nearby.

4. **Operational Sequence Outline**

The step-by-step process below shows how the robot will perform its task: Pseudocode:

START

REPEAT UNTIL gutter is fully cleaned

Move forward along the gutter

CHECK for plastic waste using a sensor

IF plastic waste is detected THEN

Stop movement

Activate gripper

Pick up waste

Move to waste bin

Drop waste

Return to gutter

END IF

END REPEAT

STOP

***Note**

*For a flowchart version, replace the pseudocode with a diagram using standard symbols and connectors. **

5. **Potential Impact and Benefits**

The robot will reduce manual labour, increase safety, and improve sanitation in the area.

6. **Conclusion**

This report has shown that a robot can help solve the problem of blocked gutters. The operational plan makes the robot's actions clear and practical.

7. **Appendices**
    a. Sketches of the robot
    b. Sensor specifications
    c. Photo of the local problem area

CRITIQUING TECHNICAL REPORTS

Reviewing reports helps improve your ideas and communication. A critique checks if a report is clear, accurate, and useful, whether you review your own work or a peer's. This ensures your robotic solution is strong and inclusive.

STEPS TO CRITIQUE A REPORT

1. **Understand the Goal**

Know the report's purpose (e.g., propose a robot for a Ghanaian problem).

2. **Read Fully**

Read the report once to grasp its ideas and flow.

3. **Check Key Parts**
    a. Is the problem clear and well-analysed (causes, impacts)?
    b. Does it explain why a robot is suitable, using Year 1 and Year 2 skills?
    c. Is the robot's role specific (e.g., "collect plastic from gutters")?
    d. Is the flowchart or pseudocode accurate and easy to follow?
    e. Does the solution help everyone, including people with disabilities?
    f. Is the report clear, organised, and professional?
4. **Note Strengths**

Highlight what is done well, like a clear flowchart.

5. **Find Improvements**

Point out unclear or missing parts, e.g., "The problem's impact on wheelchair users is not explained."

6. **Give Feedback**

Suggest specific fixes, like "Add a decision point to the flowchart for obstacle detection."

Discussion

Design and swap reports with a classmate. Check if their flowchart or pseudocode matches the robot's task and suggest one improvement.

GIVING HELPFUL FEEDBACK

1. Focus on the Report: Comment on the content, not the person.
2. Be Specific: Say, "The pseudocode on page 2 misses a loop for repeating tasks," not "This is confusing."

3. Offer Suggestions: e.g., "Add a connector to link flowchart pages for clarity."
4. Be Balanced: Note strengths (e.g., "Great problem description") and improvements.
5. Use Kind Words: Be respectful and clear.
6. Explain Why: Link feedback to criteria, e.g., "A clearer flowchart helps readers understand the robot's actions."

Activity 1.3 Planning and Reviewing Robot Tasks in Ghana

You are to perform this activity using the same groups as used in Activities 1.1 and 1.2

Objective: Use your Year 1 and Year 2 robotics skills to plan a robot's task sequence for a Ghanaian problem, represent it with a flowchart or pseudocode, integrate it into a technical report, and review a peer's report.

Materials: Paper, pens, chart paper, markers, digital devices (optional).

Instructions

Step 1: Break Down the Robot's Task (10 minutes)

1. Refer to Your Report from Activity 1.2: Look at the robotic role you defined in your technical report from Activity 1.2 (e.g., a robot cleaning gutters in Accra or sorting crops in a market).
2. List Steps: Break the role into 5–7 simple, ordered steps using your Year 1 (sensors, motors) and Year 2 (programming, navigation) skills. Example for gutter-cleaning robot:
    a. Move along gutter.
    b. Sense for waste.
    c. Stop if waste is detected.
    d. Grab and lift waste.
    e. Move to bin.
    f. Release waste.
    g. Return to gutter.

Step 2: Create a Flowchart or Pseudocode (15 minutes)

1. Choose a Tool: Decide to use a flowchart (visual) or pseudocode (text) to show your steps.
2. Make the Representation: Draw the flowchart or write pseudocode for your steps. Example pseudocode for gutter-cleaning:

Step 3: Update Your Technical Report from Activity 1.2 (10 minutes)

1. Add a Section: Create a new section in your report called "Operational Sequence Outline" after the "Proposed Robotic Role" section.
2. Include Steps and Representation: Add your step list and flowchart or pseudocode. Ensure it matches the robot's role.
3. Reference It: In the "Proposed Robotic Role" section, note that details are in the new section.
4. Update Table of Contents: Add the new section with its page number.

Step 4: Peer Review and Feedback (10 minutes)

1. Swap Reports: Exchange your updated report with another group.
2. Review the Report: Check for:
    a. Clear problem description.
    b. Suitable robot role linked to Year 1 and Year 2 skills.
    c. Accurate, readable flowchart or pseudocode.
    d. Organised structure and clear language.
3. Give Feedback: Write or say:
    a. One strength (e.g., "Clear flowchart shows all steps").
    b. One improvement (e.g., "Add a loop to repeat waste collection").

EXTENDED READING

Resource	QR Code to Accessing Resource
1. Tutorial on using connectors in Flowcharts (Video Resource).	
2. Technical report writing format sample: A comprehensive guide (Web Publication)	

3. How to Review and Provide Constructive Feedback as a Peer Reviewer (Paper Publication)	
4. Sample Spike Prime Projects (Video Resource)	
5. 10 Incredible LEGO SPIKE Prime robots for Medical Applications (Video Resource)	

REVIEW QUESTIONS

These review questions require you to use knowledge from years 1 and 2

1. A robot collects waste from a market. The pseudocode for the market waste collection robot includes the following simplified logic:

   START

   Scan area for plastic waste

   Move to waste location

   Collect waste

   Continue scanning

   END

   Rewrite the pseudocode and insert an appropriate step to improve inclusivity for visually impaired users and justify your choice.

2. A robot designed to monitor air quality in Accra must assess pollution levels and send an alert to local authorities if levels are high. If pollution exceeds a predefined threshold, it sends an alert; otherwise, it continues monitoring.

   Explain which flowchart symbols should be used to represent this logic effectively and justify your selection based on their function.

3. Evaluate the advantages of using pseudocode over a flowchart to outline a robot's task sequence for delivering medical supplies to rural Ghanaian clinics.

4. Explain why using standard flowchart symbols is essential when creating an operational sequence for a robot designed to solve a Ghanaian problem, such as cleaning plastic waste from a market.

5. Study the pseudocode and robot description below.

   The robot uses a camera sensor to detect fish size and a gripper arm to sort fish into bins (small, medium, large). It handles errors by placing unidentified fish in a separate bin. It repeats this process while the conveyor belt is moving.

   Pseudocode:

   START

   WHILE conveyor belt is moving

   CHECK if fish is on belt

   IF no fish

   continue

   ELSE

   IDENTIFY size of fish

```
    IF fish is large
    place fish in medium bin
    IF fish is medium
    place fish in medium bin
    IF fish is small
    place fish in small bin
    IF size not identified
    skip fish
END WHILE
STOP
```

Answer the following questions

a. Which parts of the pseudocode are logically correct?

b. Identify one error in the pseudocode and explain the consequences if left unchanged.

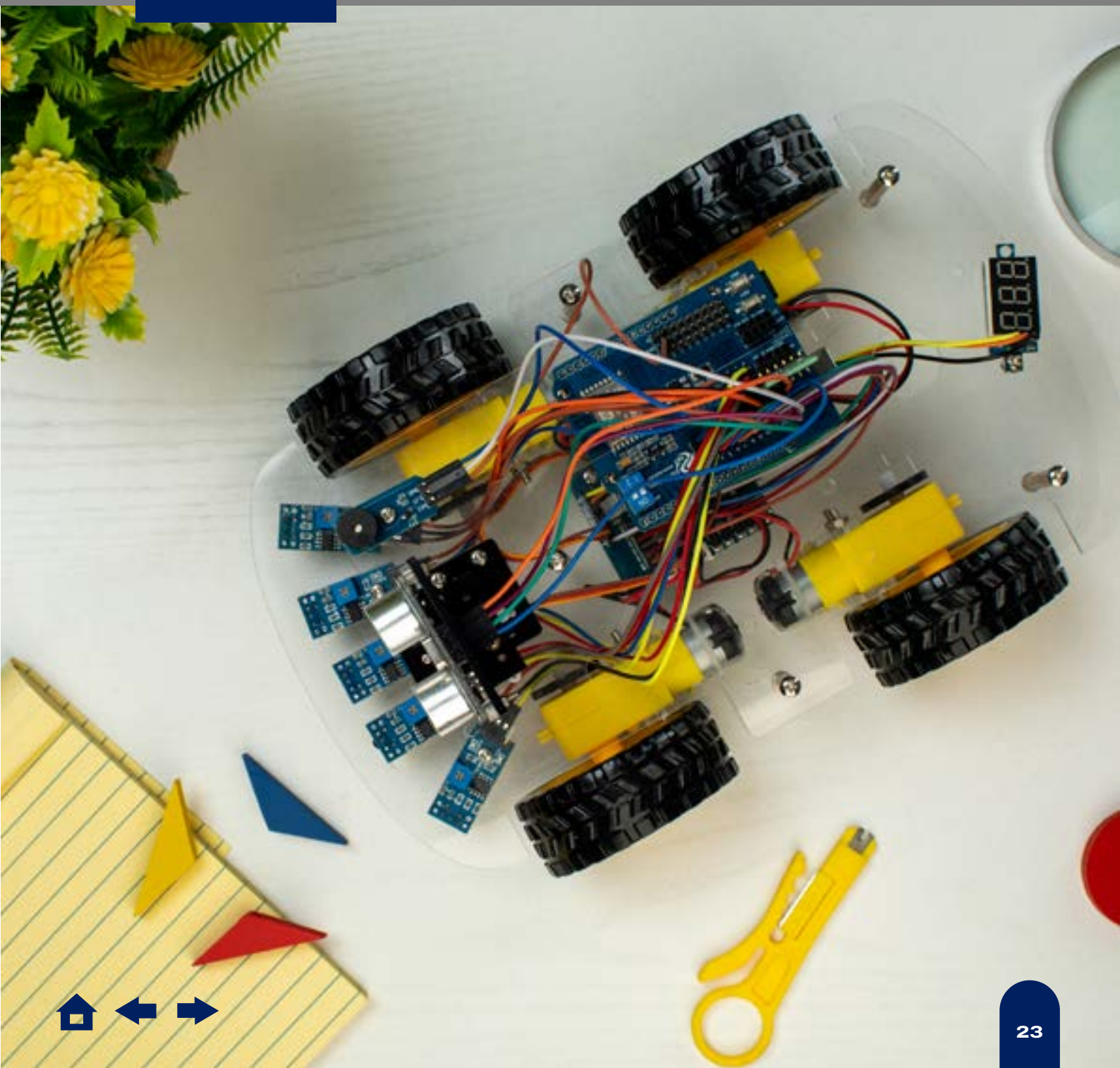
c. Describe how a feedback loop could improve the robot's error handling and sorting accuracy.

d. Write a short report summarising the robot's task, strengths and weaknesses in the pseudocode, and your recommendations for improvement. Justify your proposed changes.

SECTION

2

ROBOT CONTROL PRINCIPLES 3



Principles of Robotic Systems

INTRODUCTION

Welcome to your next big mission. You have already built robots, learned about their inner workings, and even acted as a detective to figure out their hidden logic. Now, you will become a robot doctor, an expert at making sure your creations are not only clever but also dependable. This is where you learn to find problems before they become big issues and make sure your robots work perfectly, every single time. Get ready to put your skills to the ultimate test and build robots that are truly ready for the real world!

KEY IDEAS

- Autonomous systems allow robots to operate independently, using feedback control to make real-time adjustments based on their environment.
- Sensors function as the robot's eyes and ears, gathering information about its surroundings, while the control logic processes this data to determine actions.
- A feedback loop consisting of sensing, comparing, and acting enables robots to adapt and optimise their performance without human intervention.
- Designing robots with feedback mechanisms can significantly enhance efficiency across various industries, such as agriculture, food processing, and transportation in Ghana.
- A structured approach to problem-solving involves analysing inefficiencies, defining controlled variables, selecting appropriate sensors and actuators, and developing clear logic rules for the robot's operations.
- Testing and troubleshooting robots ensure reliability and performance, enabling continuous improvements through iterative cycles of diagnosis, refinement, and optimisation.

DESIGNING AUTONOMOUS FEEDBACK SYSTEM PROTOTYPES

Welcome, future robot builders! You have already explored the exciting world of robots and how they interact with us. Now, get ready to dive even deeper and unlock the secret to making robots super smart and independent!

THE MAGIC OF AUTONOMOUS SYSTEMS AND FEEDBACK: ROBOTS THAT THINK FOR THEMSELVES!

Imagine a robot that can work all by itself, without you constantly telling it what to do. Sounds amazing, right? That is what an autonomous system is all about! Think of it like a plant that knows when it needs water, or a car that can drive itself. These systems can make their own decisions and adjust what they are doing based on what is happening around them.

So, how do they do it? The secret ingredient is **feedback control!** Remember how in previous years you learned about inputs and outputs? Well, feedback is like a robot's own personal detective. It is constantly checking how well it is doing and then uses that information to make adjustments.

HOW IT WORKS

SENSING (THE ROBOT'S EYES AND EARS)

The robot uses sensors (like the ones you explored in Year 1) to gather information about its surroundings and its own actions. Is the temperature too high? Is the object in the right place? These sensors are always on the lookout!

COMPARING (THE ROBOT'S BRAIN)

The robot then takes the information from its sensors and compares it to what it should be doing (we call this the "setpoint"). It is like checking if your actual walking speed matches the speed you want to walk at.

ACTING (THE ROBOT'S MUSCLES)

If there is a difference between what is happening and what should be happening, the robot's "brain" sends a message to its actuators (like the motors and grippers you have worked with!). These actuators then make the necessary adjustments to get things back on track.

This continuous cycle of sensing, comparing, and acting is the feedback loop! It is what allows robots to adapt to changes and keep working smoothly without needing you to guide them every step of the way. It is like magic, but it is really clever engineering!

Imagine this!

You design a robot that sorts tomatoes by their ripeness. It uses a colour sensor to "see" the tomato's colour (that is the feedback!). If the sensor tells the robot the tomato is green (unripe), it sends a signal to a little flap (an actuator) to push it into the "unripe" bin. If it is bright red, it goes into the "ripe" bin! The robot keeps checking and adjusting, making sure every tomato ends up in the right place. That is the power of feedback control!

SUPERCHARGING GHANA: MAKING THINGS BETTER, FASTER, AND SMARTER!

Now that you are becoming a feedback pro, let us talk about how these amazing autonomous systems can make a huge difference right here in Ghana! When we talk about efficiency in local industries, we mean doing tasks with less waste, less time, fewer materials, less energy, and less effort.

Think about it:

1. **Farming**

Imagine how much time and effort go into manually planting seeds or sorting harvested crops like cocoa beans or yams. What if a robot could do it precisely and quickly, reducing crop loss and saving precious time? This could mean more food for everyone and more income for farmers!

2. **Food Processing**

Think about the steps involved in making delicious local foods. Are there parts that are slow or where things could go wrong? Robots with feedback could help mix ingredients perfectly, control temperatures precisely, or even package items faster and more consistently.

3. **Textile Production**

Cutting fabrics manually can lead to waste. What if a robot could cut patterns with perfect precision, saving valuable material and making clothes more affordable?

4. **Craft Making**

Many beautiful Ghanaian crafts involve repetitive steps. Robots could help with tasks like sanding or polishing, ensuring consistent quality, and freeing up artisans to focus on the creative parts!

5. **Transportation and Logistics**

Ever seen packages being sorted at a busy market? It can be a slow process! Robots could help sort items, load and unload goods, or even navigate through crowded areas to deliver packages faster and more efficiently.

By designing robots with feedback, you are not just building cool machines; you are building solutions that can help people, boost businesses, and improve daily life in your community!

DESIGNING A FEEDBACK SYSTEM FOR EFFICIENCY

So, how do you go from a problem to a super-efficient robot solution? It is like being a detective and an inventor all rolled into one!

1. Problem Analysis (Be a Problem Detective): First, really understand the problem you want to solve. What makes it slow, wasteful, or difficult? Talk to people, observe what is happening, and dig deep!

2. Identify the "Controlled Variable": What is the one thing your robot needs to keep an eye on and control to make things better? Is it the position of an object, the temperature of something, or the colour of a fruit?
3. Choose Your Robot's Senses: Based on what you need to control, which sensor will give your robot the best information? Think back to your Year 1 lessons on light sensors, temperature sensors, or even touch sensors!
4. Give Your Robot Muscles (Actuators): What kind of actuator will your robot use to make changes? Will it be a motor to move something, a gripper to pick something up, or a pump to move liquids? You are the boss!
5. Set the Target (Setpoint): What is the ideal state or value you want your robot to achieve? For example, if it is sorting tomatoes, your setpoint would be the specific colour range for "ripe."
6. Outline the Brain's Rules (Control Logic): This is where you tell your robot how to use the information from its sensors to control its actuators. It is like writing a simple "if-then-else" rule: "IF the tomato is red THEN put it in the ripe bin ELSE put it in the unripe bin."

Now, it is time to get busy with some exciting activities that will help you put your new knowledge into action. These tasks will challenge you to think like a robotic engineer and solve real-world problems!

Activity 2.1: The Efficiency Detectives!

Before starting this activity, organise yourself into a group of 3 to 4 people.

Objective: To identify and deeply analyse an inefficiency in a local Ghanaian industry or environment that a robot could help solve.

Duration: 30-40 minutes

Materials: Large paper/chart paper, markers, pens, digital devices (optional for research)

Instructions

1. Brainstorming Local Challenges (10 minutes): As a group, think of different tasks performed in local industries or even within your school or home that seem inefficient. Do not be afraid to think big or small! Examples:
    a. Slow sorting of produce at the market (like mangoes, yams, or even plastic bottles for recycling).
    b. Manual mixing of ingredients in a small bakery.
    c. Inefficient watering of plants in a school garden.
    d. Time-consuming cleaning of the school compound.
    e. Manual checking of product quality in a small craft workshop.

2. Choose Your Mission (5 minutes): From your brainstormed list, choose one specific task that you think a robot with feedback control could significantly improve.
3. Become Efficiency Experts (15 minutes): Now, let us become efficiency detectives! For your chosen task, discuss and answer the following questions in detail. Write your answers clearly on your chart paper:
    a. What is the current process? Describe exactly how this task is done right now, step-by-step.
    b. Where are the bottlenecks? What are the "slow spots" or points where things get stuck or delayed?
    c. What kind of waste is happening? Is it wasting:
        i. Time? How much time does it take? Could it be faster?
        ii. Materials? Is anything being damaged, spilt, or used unnecessarily?
        iii. Energy? Are people getting tired quickly? Is there a lot of manual effort?
        iv. Effort? Are there repetitive or boring tasks that people have to do over and over?
    d. Why is this inefficiency a problem? What are the negative consequences for people, the environment, or the business? (For example, slow sorting leads to spoiled tomatoes, or manual cleaning is tiring and inconsistent).
    e. Can you quantify the inefficiency? Even an estimate is good! For example:
        - "It takes X minutes to sort one bag of yams."
        - "About Y% of materials are wasted during cutting."
        - "It takes Z people to do this task."
2. Share Your Findings (5 minutes per group): Each group will briefly present their chosen problem and their detailed analysis of the inefficiencies to the class. Be ready to explain why your chosen problem is a great candidate for a robot solution!

Challenge yourself

Try to interview someone who performs this task to get real-world insights!

Activity 2.2 Designing Your Robot Hero!

Before starting this activity, organise yourself into a group of 3 to 4 people.

Objective: To design a conceptual feedback-driven robotic system to solve the inefficiency identified in Activity 2.1.

Duration: 45-50 minutes

Materials: Paper, pens, coloured pencils/markers (for sketching), digital devices (optional for research/diagrams)

Instructions

1. Recap Your Problem (5 minutes): Start by quickly reviewing the specific inefficiency problem your group chose in Activity 2.1. Keep it in mind as you design!
2. Define Your Robot's Mission (5 minutes): What exactly will your robot do? Be super specific! Instead of "clean the school," maybe it is "collect fallen leaves from the football field."
3. Identify the Controlled Variable (10 minutes): What one thing about the task needs to be constantly monitored and controlled by your robot to make it efficient?

   Examples

   a. For sorting tomatoes: The colour of the tomato.
   b. For precise watering: The soil moisture level.
   c. For consistent mixing: The thickness of the mixture.
   d. For collecting leaves: The presence of leaves on the ground.

   Discuss and choose the most critical variable.
4. Select Your Robot's Senses (Sensors!) (10 minutes): Based on your controlled variable, what kind of sensor would your robot need to "see" or "feel" that variable? Think back to what you learned in Year 1!

   Examples:

   a. For colour: A colour sensor.
   b. For soil moisture: A moisture sensor.
   c. For thickness: A viscosity sensor or a weight sensor.
   d. For leaves: An object detection sensor (like ultrasonic or infrared).

   Explain why you chose that specific sensor.
5. Give Your Robot Muscles (Actuators!) (10 minutes): Now, how will your robot act on the information it gets from the sensor to make things efficient? What kind of actuator will it use? Again, think back to Year 1!

   Examples:

   a. For sorting: A servo motor controlling a flap, or a pneumatic arm.
   b. For watering: A pump to release water.
   c. For mixing: A motor to spin a stirrer.
   d. For leaves: A motor to power a vacuum or a gripper to pick them up.

   Explain why you chose that specific actuator.

6. Outline the Brain's Feedback Logic (The Smart Part!) (10 minutes): This is the most exciting part! How will your robot use the sensor's input to control the actuator's output to achieve efficiency? Write down a simple rule.
   a. Example (for leaf collection): "IF a leaf is detected THEN activate vacuum motor ELSE move forward."
   b. Example (for watering): "IF soil moisture is below setpoint THEN activate water pump ELSE turn off pump."
7. Sketch Your Robot Hero! (Optional, but Fun!): On a separate piece of paper, draw a simple sketch of your robot. Label its parts: the sensor, the actuator, and where the "brain" (control unit) would be. Show how the feedback loop would work in your drawing!

Think Inclusively

As you design, how can your robot help everyone, including people with disabilities? Could it have voice commands, large buttons, or visual indicators?

Activity 2.3 The Robot's Playbook: Flowcharts and Pseudocode!

Objective: To outline the step-by-step operational sequence of your designed robot using either a flowchart or pseudocode, and to justify how it improves efficiency.

Duration: 40-45 minutes

Group Size: 3-4 learners (each learner contributes to the flowchart/pseudocode and justification)

Materials: Large paper/chart paper, markers, pens, ruler (for flowcharts), digital devices (optional for typing pseudocode)

Instructions

1. Choose Your Planning Tool (5 minutes): As a group, decide if you want to use a flowchart (visual map with shapes) or pseudocode (text-based steps like a recipe) to plan your robot's actions. Remember, you learned about both in Year 2!
2. Outline the Robot's Steps (20 minutes): Now, working together, create a detailed step-by-step plan for your robot using your chosen tool.

   For a Flowchart, remember the following:
   a. Use the correct shapes: Ovals for Start/End, Rectangles for Actions, Diamonds for Decisions, Parallelograms for Input/Output, and Arrows for flow.
   b. Make sure every step is clear and logical.
   c. Show the feedback loop clearly (e.g., "Sense X" leading to a "Decision" that then triggers an "Action").

For Pseudocode, remember the following:

a. Use simple, clear language.

b. Use commands like START, END, READ, IF...THEN...ELSE, REPEAT, MOVE, ACTIVATE.

c. Show the sequence of actions and how decisions are made based on sensor input.

Example (for the tomato sorter)

START

WAIT FOR TOMATO TO REACH SENSOR

READ COLOUR_VALUE FROM COLOUR_SENSOR

IF COLOUR_VALUE is RIPE:

ACTIVATE_ACTUATOR_FOR_RIPE_BIN

ELSE IF COLOUR_VALUE is SEMI_RIPE:

ACTIVATE_ACTUATOR_FOR_SEMI_RIPE_BIN

ELSE:

ACTIVATE_ACTUATOR_FOR_OTHER_BIN

WAIT FOR ACTUATOR ACTION COMPLETION

GO TO START

END

3. Write Your Mini-Report (15 minutes): Now, it is time to document your brilliant design! On a new piece of paper, create a short report with these sections.

a. Title: Give your project a catchy title (e.g., “Automated Mango Sorter for Kumasi Market”).

b. Problem Description: Briefly describe the inefficiency problem you identified in Activity 2.1.

c. Proposed Robot System: Briefly describe your robot from Activity 2.2: what it does, what sensor it uses, and what actuator it uses.

d. Feedback Mechanism Explained: Clearly explain the feedback loop in your robot. What is the controlled variable? How does the sensor measure it? How does the robot’s “brain” use that information to tell the actuator what to do?

e. Operational Sequence Outline: Include your flowchart or pseudocode here! Make sure it is neat and easy to understand.

f. Justification for Efficiency Improvement: This is crucial! Explain how your robot’s design, especially its use of feedback, makes the task much more efficient than the current method. Be specific! (e.g., “The robot

sorts tomatoes 5 times faster than humans, reducing spoilage by 20%"). Also, do not forget to mention how it benefits everyone.

4. Peer Review & Refine (5 minutes): Exchange your report with another group. As a reviewer, check:
   a. Is the problem clear?
   b. Is the robot's role well-defined?
   c. Is the flowchart/pseudocode accurate and easy to follow?
   d. Does the report clearly explain how the feedback loop improves efficiency?
   e. Does it consider inclusivity?
   f. Provide polite and helpful suggestions for improvement!

Bonus Challenge

Can you think of a real-world constraint that might affect your robot's design (e.g., limited power, dusty environment)? How would you adjust your plan?

Activity 2.4 The Robot Pitch!

Objective: To present your team's robot design and its efficiency improvements to the class clearly and engagingly.

Duration: 3-5 minutes per group presentation, plus 2 minutes for Q&A.

Group Size: 3-4 learners

Materials: Your mini report from Activity 2.3, chart paper/markers (if you drew a flowchart/sketch), or a digital presentation (if available).

Instructions

1. Prepare Your Pitch (10 minutes): Review your mini report. Decide who will explain each part (problem, robot design, feedback, operational sequence, benefits). Practice presenting it clearly and confidently!
2. Engage Your Audience: Speak loudly and clearly. Look at your classmates and teacher. Show enthusiasm for your robot idea!
3. Highlight the "Why": Focus on why your robot is needed and how it will truly make things better in Ghana, emphasising the efficiency gains and inclusivity features.
4. Be Ready for Questions: After your presentation, other groups and your teacher might ask questions. Be prepared to explain your choices and justify your design!

Presentation Tips

   a. Visuals are Key! Use your flowchart or sketch to help explain your robot's actions.

b. Tell a Story: Start with the problem and end with the amazing solution your robot provides.

c. Teamwork: Make sure everyone in the group participates in the presentation.

Individual Challenge: "My Robot Idea!"

Objective: To individually brainstorm and sketch a robot that uses feedback to solve a personal or household inefficiency.

Duration: 20-25 minutes

Materials: Paper, pens, coloured pencils.

Instructions

1. Think Local, Think Small: Forget big factories! Think about your home, your school desk, or even a simple chore you do every day. Is there something that is repetitive, takes too long, or is a bit wasteful?

   Examples:

   a. Forgetting to close the fridge door.

   b. Watering a plant too much or too little.

   c. Sorting your stationery on your desk.

   d. Making sure your room is tidy.

2. My Inefficiency: Clearly state the inefficiency you want to tackle.
3. My Robot's Mission: What specific task will your robot do?
4. Feedback Hero:

   a. What is the controlled variable for your robot? (e.g., fridge door status, soil wetness, pen location).

   b. What sensor would your robot use to detect this? (e.g., a magnet sensor, a moisture sensor, a light sensor).

   c. What actuator would your robot use to act on it? (e.g., a small motor to pull the door, a pump, a robotic arm).

   d. What is the simple feedback rule for your robot? (e.g., "IF fridge door is open THEN activate motor to close it").

5. Sketch it Out! Draw your robot! Make it fun and functional. Label its sensor, actuator, and any other important parts. Show how the feedback loop works in your drawing.
6. Reflect: How would your robot make your life (or someone else's) more efficient and easier?

REVERSE ENGINEERING AUTONOMOUS ROBOT CONTROLS

Hello future roboticists and innovators! Get ready to take on the role of a robotics detective! You have already built robots and understand how their parts work. Now, you will learn how to look at an existing robot and figure out its secrets; how it thinks and makes decisions all by itself. This is called Reverse Engineering, and it is one of the most exciting parts of robotics!

REVERSE ENGINEERING IN ROBOTICS

Imagine you have a fantastic gadget, and you want to know exactly how it works. You do not have the user manual. Reverse engineering is like carefully observing and analysing that gadget to understand its design, its parts, and its logic. In robotics, this is not just about taking a robot apart. It is about figuring out the brain behind the brawn; the set of rules and decisions that lets the robot act on its own.

WHY DO WE DO THIS?

1. To Become Smarter: You will get a deep understanding of how engineers use ideas like feedback loops in real life.
2. To Learn from the Best: You will discover clever ways other engineers have solved problems. It is like peeking into their notebook!
3. To Find Problems: If a robot is not working right, you can reverse-engineer it to figure out what is wrong.
4. To Get Inspired: The lessons you learn can spark amazing new ideas for your own robots to solve local challenges.

CHARACTERISTICS OF A FUNCTIONAL AUTONOMOUS ROBOTIC SYSTEM

An autonomous robot is like a living thing, with its own senses, brain, and muscles. When you reverse engineer, you are looking for these key characteristics:

1. Sensing (The Robot's "Senses"): A robot uses sensors to gather information from its surroundings. It could be a light sensor to see if it is day or night, or a sound sensor to detect noise.
2. Processing (The Robot's "Brain"): This is where the robot's brain, or processor, takes all the information from its sensors and thinks about what to do. It runs the control logic.
3. Actuation (The Robot's "Muscles"): After the brain makes a decision, it sends a signal to an actuator (like a motor) to move a part of the robot and perform an action.
4. Adaptability/Responsiveness: This is a robot's ability to adjust what it is doing based on what its sensors are telling it. It is what makes a robot autonomous. It is the feedback loop in action!

5. Goal-Oriented Behaviour: The robot operates to achieve a defined objective without constant human instruction.

THE REVERSE ENGINEERING PROCESS FOR CONTROLS: THE ROBOT DETECTIVE'S GUIDE

This is your step-by-step guide to becoming a robotics detective. You will go from observing a robot's movements to uncovering its hidden logic!

1. **Step 1:** Become a Master Observer

First, just watch the robot. What is it doing? What is its goal? How does it behave? Pay close attention to what causes it to start or stop an action. For example, if you see an automated hand-washing station at a market, ask yourself:

a. What makes the soap come out? Is it a person's hands?

b. How does it know when to stop dispensing?

c. What if someone just stands there without putting their hands under the sensor?

2. **Step 2:** Find the Parts

Now, you will look for the main components. Based on your observations, you can guess which parts are the sensors and which are the actuators. Remember your lessons from Year 1 on robot subsystems! You may not be able to open the robot, but you can identify the parts based on what they do.

3. **Step 3:** Follow the Data Path

Imagine you can see the flow of information like a river. How does the information from a sensor get to the brain? And how does the brain's decision travel to the actuator? You will trace the feedback loop by asking: "What information is the sensor sending that directly changes what the actuator is doing to reach the robot's goal?"

4. **Step 4:** Crack the Code

This is the most challenging and exciting part! Based on everything you have observed and the parts you have identified, you will try to figure out the robot's "rules." What are the "if-then-else" statements that make it work? For example, "IF a hand is detected, THEN dispense soap." You are now thinking like a robot programmer!

5. **Step 5:** Rebuild the Logic

Finally, you will create your own maps and plans to show the world what you have discovered. You can use flowcharts and pseudocode (remember those from Year 1 and 2?) to explain the robot's logic. This is your chance to show off your conceptual "rebuild" of the system!

Activity 2.5 The Robot Safari!

Before starting this activity, organise yourself into a group of 3 to 4 people.

Your Mission: Go on a hunt to find a functional autonomous robot in a video or a physical space. If you do not find one out there, you can select any of the following videos.

Link and Description	QR Code
Navigate maze with distance sensor: turn right, left, behind (Video Resource)	
WRO 2025 Junior full missions - spike prime (Video Resource)	
WRO 2025 Elementary (SPIKE Prime) (Video Resource)	

Your Toolkit: A notepad, a pen, and your sharp observational skills.

The Task

1. Watch the Robot: Observe the robot's every move. What is its overall job?
2. Input/Output: Write down everything the robot senses (inputs) and everything it does (outputs).
3. Find the Parts: Identify the likely sensors and actuators. What are they, and what do you think they do?

Activity 2.6 The Feedback Blueprint!

Complete this activity in the same group as Activity 2.5.

Your Mission: Based on your observations from Activity 2.5, figure out the robot's main feedback loop.

Your Toolkit: Paper and a pen.

The Task

1. The Goal: What is the robot trying to achieve or maintain? That is its desired state.
2. The Information: What is the sensor measuring? That is its sensed state.
3. The Action: What does the actuator do?
4. Draw a simple block diagram to show this entire feedback loop. Label all the parts clearly: the sensed state, the desired state, the sensor, the processor, and the actuator.

Activity 2.7 Cracking the Code with Flowcharts!

Before starting this activity, organise yourself into a group of 3 to 4 people.

Your Mission: Translate your observations into a set of rules that the robot's brain follows.

Your Toolkit: Your notebook and your knowledge of flowcharts or pseudocode.

The Task

1. Infer the Logic: Think about the "if-then-else" statements that make your robot work.
2. Create a flowchart or write out the pseudocode to show the robot's decision-making process from start to finish. Make sure it is clear and easy to follow.

Activity 2.8 The Reverse Engineering Report!

Before starting this activity, organise yourself into a group of 3 to 5 people.

Your Mission: Compile all your findings into a comprehensive report.

Your Toolkit: Your notes, diagrams, and flowcharts.

The Task

1. Create a report which has clear sections: a description of the robot, the parts you identified, your feedback loop diagram, a description of the loop's operation, and your flowchart or pseudocode.
2. Add a Conclusion: End your report by explaining what you learned about autonomous systems from this whole experience.

ROBOT TROUBLESHOOTING: DIAGNOSIS TO IMPROVEMENT

Welcome to your next big mission! So far, you have built robots, learned about their inner workings, and even acted as a detective to figure out their hidden logic. Now, you will become a robot doctor, an expert at making sure your creations are not only clever but also reliable. This is where you learn to find problems before they become big issues and make sure your robots work perfectly, every single time. Get ready to put your skills to the ultimate test and build robots that are truly ready for the real world!

TESTING OPERATIONAL ROBOTS AND IDENTIFYING FLAWS

Before any robot can be trusted to do an important job, like sorting cocoa beans or helping farmers, we must check it carefully. This process is called testing, and it is how we make sure our robots are reliable, safe, and ready for anything!

WHY IS TESTING SO IMPORTANT?

1. Reliability: This makes sure your robot does its job consistently, without any unexpected failures. For example, a robot that collects rubbish must not suddenly stop working halfway through its route.
2. Performance: This checks if your robot is doing its job well and quickly. You would want to know if a drone can deliver goods fast enough or if a sorting machine is accurately separating items.
3. Safety: This is one of the most important checks. It identifies any behaviour that could be dangerous to people or their surroundings. For example, we must test a self-driving car to ensure it sees and avoids people, animals, and other cars.
4. Validation: This is your final check to make sure the robot you built actually solves the problem you set out to solve. It proves that all your hard work truly paid off.
5. Optimisation: Testing provides valuable information that helps you make your robot even better. You might find a way to make it use less power, move more smoothly, or be even more robust in difficult situations.

YOUR GAME PLAN: DESIGNING A COMPREHENSIVE TEST PLAN

A good test plan is like a football coach's playbook for an important match. It is not just a simple checklist; it is a strategic document that ensures you check every important part of your robot's performance.

A great test plan includes

1. Test Objectives: These are the goals for your test. What exactly are you trying to find out? For example, your objective could be to "check if the robot can follow a line without veering off" or "see if the robot can correctly sort red and green objects."

2. Test Environment: This describes the conditions your robot will be tested in. Will it be on a smooth floor or a bumpy one? What about the lighting? Is it dark or bright? This helps you create a fair and repeatable test.
3. Test Cases/Steps: These are the specific, step-by-step instructions for what the robot must do. For a line-follower, a test case might be "start the robot on a straight line," followed by "observe it as it approaches a corner," and so on.
4. Expected Outcomes: For each step in your test plan, you must write down what the robot should do if it is working correctly. For example, the expected outcome for the line-follower could be "the robot follows the line smoothly and turns accurately at the corner."
5. Observation and Documentation: This is about how you will keep track of what happens during your test. You will need a way to measure your robot's performance using numbers and a way to write down any flaws you find. When you are measuring with numbers, think of this as giving your robot a report card. You can measure things like the time it takes to finish a task, the percentage of items it sorts correctly, or the number of times it bumps into something. This gives you clear proof of how well your robot is performing. You will also need a special format to write down any unexpected behaviour. This helps you keep your notes organised and makes it easy to find and fix problems later.

EFFECTIVE METHODS OF OBSERVATION AND DOCUMENTATION

Once you have your test plan, you need to use your detective skills to find any flaws, any time the robot does something unexpected. Here is how you will observe and document a robot's behaviour.

1. Visual Inspection

Just like a doctor looks at a patient, you will look at your robot as it moves. Is its movement smooth or jerky? Do any parts look loose or out of place? Does it seem to be struggling? This is your most direct way of spotting problems.

2. Sensor Data Monitoring

A robot's sensors are its eyes and ears. Sometimes, a robot behaves strangely because a sensor is giving it bad information. By checking the data coming from your sensors, you can see if they are working properly. This will feel a lot like what you learned about how sensors work in Year 1!

3. Actuator Response Monitoring

This is about watching your robot's muscles in action. Are the motors, grippers, or pumps doing what the robot is telling them to do? You can watch their movements, listen for strange sounds, or even gently feel for unusual vibrations to see if they are responding correctly.

4. Timing

Is your robot taking too long to complete a task? By timing different parts of its operation, you can find bottlenecks and discover where things are slowing down. This is crucial for making your robot super-efficient.

5. **Error Counting**

You can get a lot of information by counting how many times a specific mistake happens. For example, if a sorting robot misplaces an item, you will count how many times it happens in ten attempts. This gives you solid evidence of a problem and helps you track if your fixes are working.

Tracing Shortcomings/Flaws to Root Causes

After you have documented the flaws, the next step is to figure out why they are happening. Just like a doctor diagnoses a disease from a patient's symptoms, you will diagnose the root cause of your robot's problems.

There are three main culprits for most robot problems:

1. Algorithm Flaw: This is a problem with the strategy or the logic you designed. Imagine you are giving your robot a recipe to bake a cake, but the recipe itself is wrong. Even if the robot follows the instructions perfectly, the cake will not turn out right.
2. Design Flaw: This is a problem with the robot's physical build. Maybe a component is too heavy, a sensor is placed in the wrong position, or a part is too weak. It is like having the right recipe but using a broken mixer to try to bake the cake.
3. Coding Error: This is a simple mistake in your code. You might have a typo, an incorrect number, or a wrong command. It is like having the correct recipe, but you wrote down the wrong amount of sugar.

THE IMPROVEMENT CYCLE: ITERATIVE FAULT CORRECTION AND PERFORMANCE IMPROVEMENT

Building a great robot is a journey of continuous improvement, not a one-time thing. The test-fix-retest cycle is your roadmap to success. You will keep going through these steps until your robot is the best it can possibly be.

1. Test: Run your test plan and find the flaws.
2. Identify: Pinpoint a specific flaw you want to fix.
3. Trace: Figure out if the problem is a coding error, a design flaw, or an algorithm flaw.
4. Fix: Make one small change to solve the problem.
5. Re-Test: Run your test plan again to see if your fix worked. Check that your fix did not break anything else!
6. Document: Write down what you changed and what happened. This is your professional journal!
7. Repeat: Then you repeat the cycle until your robot performs exactly as you expect. This is how you will turn a good idea into an amazing, reliable, and efficient robotic solution!

YOUR ROBOT'S JOURNAL: DOCUMENTATION OF ITERATIONS FOR STRUCTURED REPORTING

Keeping a detailed journal of your work is one of the most important habits of a professional engineer! This is where you write down every change you make to your robot and what happened as a result. Think of it as a detailed detective log.

For each change you make, you should write down:

1. The Flaw You Were Fixing: Be specific! What was the exact problem you were trying to solve?
2. The Change You Made: Describe exactly what you did. Did you change a number in the code? Did you add a new piece to the robot? The more specific you are, the better.
3. The Results: What happened after you made the change and ran your test again? Did the flaw go away? Did the robot get better? Did you find a new problem?

Keeping this journal helps you remember what you have tried and what worked, so you can learn from your mistakes and get your robot working perfectly.

CRUCIAL CONSIDERATIONS DURING ITERATION

The following are critical considerations to consider when carrying out iterative tests:

1. The One Change Rule

When you are fixing a problem, it is important to make only one change at a time. Imagine you are trying to find out why a meal tastes bad. If you change the amount of salt, pepper, and sugar all at once, you will never know which ingredient was the problem! By making just one change, like adjusting the salt, you can clearly see if your fix worked or if it made things worse. This helps you understand exactly what is causing the issue.

2. The Regression Test

This is like a quality check for your fixes. After you fix one problem, you must make sure you have not accidentally broken something else that was working before. For example, if you fix your robot's steering, you should re-test not just the turns, but also its ability to move straight. This ensures your robot remains reliable and that your fixes do not create new, unexpected issues.

3. Error Correction vs. Performance Improvement

The testing and fixing cycle is not just for correcting mistakes; it is also for making your robot even better! Once all the major errors are gone, you can use the same process to optimise its performance. You might test to see if you can make your robot move faster, sort items more accurately, or use less power. This is how you take a functional robot and turn it into a fantastic one!

These activities are designed to get you thinking like a professional robotics engineer. You will use your newfound knowledge to diagnose, fix, and improve a robot, all while documenting your journey.

Activity 2.9 The Robot Check-Up!

Before starting this activity, organise yourself into a group of 3 to 5 people.

Your Mission: Work as a team of robot doctors to find the "symptoms" of a sick robot.

Your Toolkit: A robot or simulation with a known, repeatable flaw, a notepad, and your brain.

The Task

1. Meet Your Patient: Your teacher will provide a robot with a task. Observe it to understand its purpose.
2. Create Your Test Plan: As a group, create a simple test plan for the robot. Remember to include test objectives, the environment, specific test cases, and the expected outcome for each.
3. Run the Diagnostics: Execute your test plan step-by-step. Use all your "detective senses": visual inspection, listening for sounds, and so on, to observe the robot's behaviour.
4. Document the Symptoms: For every time the robot does something it is not supposed to, write it down as a flaw. For each flaw, describe exactly what happened and when.

Activity 2.10 The Robot's Root Cause! (Group Activity)

Your Mission: You have the symptoms, now it is time to diagnose the root cause of the problem.

Your Toolkit: Your notes from the "Robot Check-Up," a copy of the robot's code (or a simple flowchart of its logic), and a physical robot if available.

The Task

1. Review the Evidence: Look at the list of flaws your team documented.
2. Form a Hypothesis: For each flaw, discuss as a group and decide if it is most likely an algorithm flaw, a design flaw, or a coding error.
3. Justify Your Diagnosis: Explain why you chose that category. For example, "We think this is a coding error because the motor command uses the wrong number, causing the robot to turn too far."
4. Present Your Findings: Share one of your flaw analyses with the class. Explain the symptom and your diagnosis, using your justification to prove your point!

Activity 2.11 The Iteration Challenge! (Individual or Group Activity)

Your Mission: Fix the robot! This is your chance to use the test-fix-retest cycle.

Your Toolkit: The robot or simulation, your documented flaws, and the ability to update its code or design.

The Task

1. Choose a Problem: Pick one of the critical flaws your group diagnosed.
2. Make One Change: Based on your diagnosis, make just one change to the robot's code or physical design.
3. Re-Test: Rerun the specific test case that caused the flaw. Did your fix work? Did it accidentally create a new problem?
4. Document Your Progress: Write down what you changed, what you observed, and what you will do next. This is your change log, and it is one of the most important parts of a real engineer's work!
5. Reflect: If the fix did not work, or if it made things worse, reflect on why. What new plan will you try?

Activity 2.12 The Troubleshooting Pitch!

Before starting this activity, organise yourself into a group of 3 to 4 people.

Your Mission: Present your team's troubleshooting journey to the class.

Your Toolkit: Your notes from all the previous activities, a chart paper, or a digital presentation.

The Task

1. Prepare Your Pitch: As a group, decide who will explain each part of your project:
    a. The robot and its original problem.
    b. The flaws you found.
    c. Your diagnosis (root cause) for one of the flaws.
    d. The change you made and what happened during your re-test.
    e. A reflection on what you learned.
2. Engage Your Audience: Use your notes and visuals to tell the story of your troubleshooting journey.
3. Show Your Results: Use your documentation to show how your fix did (or did not) improve the robot's performance.
4. Q&A: Be ready to answer questions from your classmates and teacher about your choices and your process!

EXTENDED READING

Resource	QR Code to Access Resource
LEGO Robotics Spike Prime Quality Check Bot (Video Resource).	
Fruit Sorting Robot (Video Resource).	
How Do I Troubleshoot a Robot at Competition? (Video Resource).	
Appyling the Troubleshooting Process (Video Resource)	

REVIEW QUESTIONS

1. You have been provided with a simple, operational line-following robot. Its task is to navigate a black line on a white surface, complete a 90-degree turn to the left, and then follow the line to a designated finish area. Your mission is to act as a robotics engineer, systematically test this robot, identify its flaws, implement a fix for one critical problem, and document the entire process.

   Submit a concise technical report (maximum 2 A4 pages) detailing your work. Your report should follow the structure below.

   a. Test Plan Development (4 marks)

      Create a detailed test plan for the robot's task. It must include:

      Clearly stated test objectives.

      A description of the test environment.

      Specific, step-by-step test cases for the entire path.

      The expected outcome for each test case.

      The methods you will use to observe and document performance.

   b. Flaw Identification and Root Cause Analysis (4 marks)

      Execute your test plan. Document any observed flaws, providing a precise description of the behaviour and when it occurred. For each flaw, categorise it as an algorithm flaw, a design flaw, or a coding error, and provide a clear justification for your choice.

   c. Correction and Iteration (4 marks)

      Select one critical flaw from your analysis. Implement a targeted fix for it. Re-test the robot to see if the flaw is resolved and if any new issues have appeared. Document the change, the re-test results, and reflect on the outcome.

   d. Technical Report (3 marks)

      Organise your findings into a clear, well-structured report. It should summarise your test plan, list the identified flaws with their root cause analysis, detail the correction and re-test results, and conclude with what you learned from the process.

2. Consider a functional autonomous robotic system designed to water plants in a school garden. The system uses a sensor to detect if the soil is dry and a pump to release water. It operates independently, ensuring the plants receive the right amount of water without human intervention.

   a. Identify the primary sensor and actuator in this system. Justify your choices. (2 marks)

b. Describe the core feedback loop of this system, indicating the sensed state, desired state, and the general action taken by the actuator. (3 marks)

c. Using pseudocode or a flowchart, outline the main control logic you would infer for this autonomous sprinkler system. (3 marks)

3. Consider a local community initiative to improve waste management. Currently, the sorting of plastic bottles from glass bottles and paper is done manually, which is slow and prone to errors.

Propose a conceptual feedback-driven autonomous system to efficiently sort plastic bottles at a collection point.

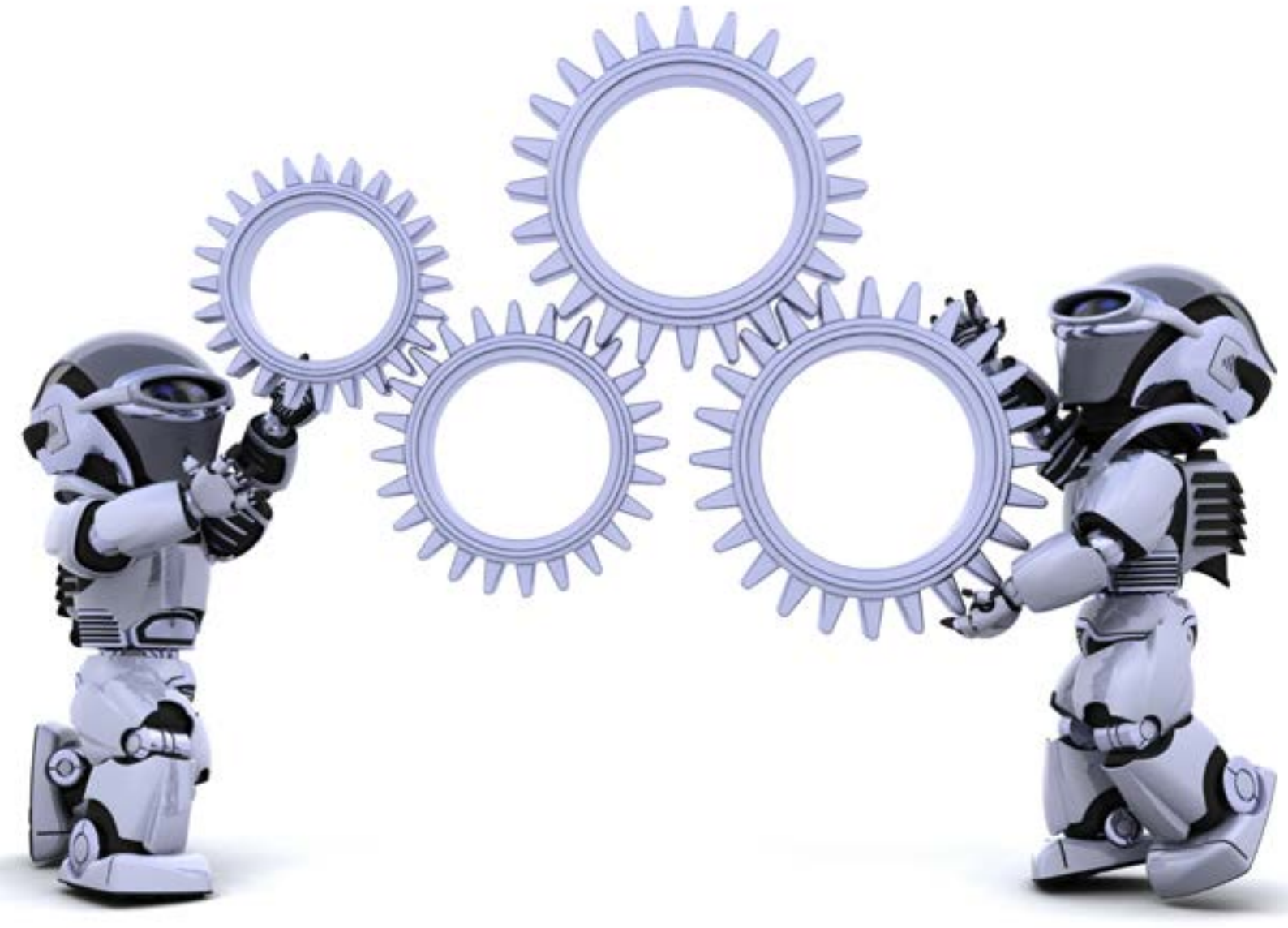
a. Identify ONE specific variable that needs to be controlled for efficient sorting. Justify your choice. (2 marks)

b. For the variable identified in (a), propose the type of sensor required to measure it. Explain why this sensor is suitable. (2 marks)

c. Briefly describe the feedback control logic that would enable the system to autonomously sort bottles based on sensor input. (3 marks)

SECTION

3 SENSORS AND ACTUATORS 3



PRINCIPLES OF ROBOTIC SYSTEMS

INTRODUCTION

In your first and second years, you learned about basic sensors, actuators, and how they work together to control simple robotic systems. Now, you will explore advanced industrial actuators and how they are used in real or simulated automation systems. You will go beyond identifying parts to exploring how systems function by applying STEM—Science, Technology, Engineering, and Mathematics—as an integrated framework for solving real-world problems. This means investigating scientific principles, using technological tools, designing engineering solutions, and applying mathematical reasoning to understand and automate industrial tasks. A key skill you will develop is the ability to select and justify your choice of actuator, which is also called a final control element, for specific jobs. You will also practice analysing systems and recommending automation solutions that suit Ghana's industries. By the end of this section, you should be able to confidently observe a process, explain how automation can help, and choose the best actuator to get the job done.

KEY IDEAS

- Advanced actuators (valves, hydraulic cylinders, contactors, relays) act as final control elements, converting energy into physical actions for industrial tasks like fluid control or electrical switching in Ghanaian industries.
- Actuators transform energy (e.g., electrical to magnetic in relays, hydraulic pressure to force in cylinders) to perform tasks. Understanding how this is done is essential for understanding their roles in automation systems.
- Select actuators based on operating conditions (e.g., dust, temperature) and safety needs, such as fail-safe valves for water treatment or contactors for safe motor switching.
- Match actuators to industrial tasks by evaluating motion type (linear, rotary, flow) and force/torque needs. For example, hydraulic cylinders may be suited to high-force linear motion in mining operations due to their pressure amplification capabilities, while precision-controlled valves are essential in cocoa processing to regulate flow rates and maintain consistent thermal conditions during fermentation or drying.
- Justify actuator choices by linking their operating principles to task requirements, incorporating factors like maintenance expertise or power availability based —taking into account conditions in Ghana such as variable grid reliability, availability of skilled technicians.
- Identify actuator limitations (e.g., cost, environmental vulnerabilities) and propose solutions, such as robust designs for dusty environments in Ghana's industries.

ANALYSING INDUSTRIAL ACTUATOR PRINCIPLES

In this section, you will learn about advanced actuators used in industrial automation. These go beyond the basic motors and servos you have used in smaller robots.

You will explore how important components like valves, hydraulic systems, contactors, and relays work. These powerful devices are known as final control elements. They help machines perform tasks like opening, closing, lifting, or switching power in large systems.

By understanding how these components operate, you will gain insight into how automated industrial systems function. This knowledge is important when analysing how machines work and suggesting the right parts to improve or automate a process.

By the end of this section, you will be better prepared to explain how advanced actuators work and how they are used in real-life industries.

UNDERSTANDING ACTUATORS AS FINAL CONTROL ELEMENTS

In robotics, you already know how to use small motors and servos to move parts. But in big factories and machines, we need much stronger tools called advanced actuators. These are like the muscles of industrial machines. They help control things like water, gas, heavy parts, or powerful machines. They are called final control elements because they are the last part that makes the machine do something after receiving a signal.

Consider four types of advanced actuators

1. Valves
2. Hydraulic cylinders
3. Contactors
4. Relays

VALVES – LIKE TAPS FOR FLUIDS

Imagine you are building a robot that can pick up objects, like a robotic arm. For the arm to move, you send a signal through codes that tell the arm exactly what to do, like “grab” or “lift.” This signal controls the robot’s actions precisely.

Now, think of a valve in a factory. A valve is like a gate that controls the flow of liquids or gases, such as water or air, in big machines or pipes. Just like your robot arm, a valve gets a signal from a switch or a computer that tells it to open, close, or adjust how much liquid flows. This helps manage the flow in a controlled way, like how you control your robot’s movements.

In robotics, you use signals to make small, precise actions, like moving a motor. In industrial systems, valves do something similar but on a bigger scale, controlling large

amounts of fluid to keep machines running smoothly. So, valves are like the "control logic" of robotics but applied to managing fluids in factories.

In Ghana, valves are typically used in the Cocoa processing industries, water treatment plants, and hydro-electrical or fossil fuel powered power stations.



Figure. 3.1: Manually operated valves in an industrial setting (wheel turning control) ([https://www.powermotiontech.com/technologies/other components/article/21254177/10-things-to-know-about-operating-valves-in-industrial-settings]

The operation of valves generally depends on the type of valve. For instance, some valves open or close when electricity or pressure is applied. An example of such valve is a solenoid valve which is a valve that uses a magnet to move a part that allows fluid to pass.

Resource Link	QR Code
https://www.youtube.com/watch?v=hVVIkQQbSHs How Solenoid valves work [Video resource]	

Types of Valves

There are many types of valves with different functionalities and structures. Some of the commonly used ones are:

1. Gate Valve – Opens or shuts flow completely
2. Globe Valve – Adjusts the flow gently
3. Ball Valve – A ball controls flow
4. Butterfly Valve – Spins to control thick liquids
5. Solenoid Valve – Works fast using electromagnetism



Figure 3.2: Types of valves [https://chemicalengineeringworld.com/different-types-of-valves/]

Resource Link	QR Code
https://chemicalengineeringworld.com/different-types-of-valves/ Types of valves [Reading resource]	

HYDRAULIC CYLINDERS – SUPER STRONG PUSHERS AND LIFTERS

Robotic arms move because motors or actuators use energy (like electricity) to create precise movements, like picking up a ball. The arm needs controlled energy to move smoothly and do its job without jerking or breaking.

Now, a hydraulic cylinder which is also called a "hydraulic jack" or "hydraulic ram" is like a super strong version of that robotic arm, used in big machines like bulldozers or cranes. It is a tube with a piston inside that moves when liquid (usually oil) is pumped in. The liquid's pressure pushes the piston, creating a lot of force to lift heavy things or push hard, like in construction or factories.

Here is the connection

in your robotic arm, you control energy (electricity) to make small, precise movements. In a hydraulic cylinder, energy is transferred through liquid pressure to create powerful movements for big tasks. Both use controlled energy transfer, but hydraulic cylinders scale it up for industrial jobs needing massive force, like lifting a car, while your robot arm might just lift a cup. It is based on Pascal's Law which states that Force = Pressure * Area.

In Ghana, Hydraulic cylinders are mostly used in tractors on farms, cranes on construction sites and mining equipment.



Figure 3.3: Hydraulic Cylinder in a tractor. [https://seahydrosys.com/applications/]

Resource Link	QR Code
https://www.youtube.com/watch?v=52IMMQSB9Hs How does a hydraulic cylinder work? [video resource]	

Types of Hydraulic Cylinders

There are many different types of hydraulic cylinders with different characteristics that depend on their application. Some of the common types are:

1. Single-Acting – Pushes with fluid, comes back with a spring
2. Double-Acting – Pushes and pulls with fluid
3. Telescopic – Extends in stages like a telescope
4. Tie-Rod – Very strong and used in big machines

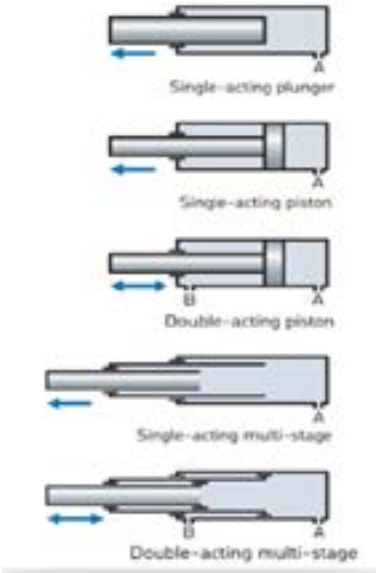


Figure 3.4: Types of Hydraulic cylinders. [https://winnelliehydraulics.com.au/how-stuff-works/hydraulic-cylinders/#hpdensity-category]

Resource	QR Code
Types of hydraulic cylinders [Reading resource] https://blog.enerpac.com/types-of-hydraulic-cylinders/	

CONTACTORS – BIG SWITCHES FOR POWERFUL MACHINES

Imagine you are working on a robot that has a powerful motor to spin wheels or lift something heavy. To control that motor, you do not want to use a tiny switch that could burn out or be unsafe. Instead, you use a special device like a switch that can handle the motor's power while being controlled by a small, safe signal, like pressing a button or sending a command from your robot's code.

A contactor is like that special device, but for big industrial machines, like motors in a factory or huge lights in a stadium. It is a heavy-duty switch that can handle a lot of electricity (high power) to turn on or off things like conveyor belts, pumps, or massive fans. You control this big switch with a small, low-voltage signal, like flipping a switch or sending a signal from a control panel. This keeps things safe because you are not directly touching the high-power circuit.

In robotics, you use small signals to control motors or lights in your project. Contactors do the same thing in industries, but they are built to handle much bigger electrical loads, like in manufacturing plants or water treatment systems.

In Ghana, contactors are mostly used in water pumps to move water from underground reservoirs to overhead tanks like "Polytanks". They are also used in big industries to start and stop heavy machines like grinders, conveyor belts, and mixers safely without damaging the electrical systems.



Figure 3.5: Contactors. [https://cselectric.co.in/blog/explore-the-different-types-of-contactors-and-how-they-work/]

Resource Link	QR Code
https://www.youtube.com/watch?v=FCsgUXPRew8 What is a contactor? [Video Resource]	

Types of Contactors

Some commonly used contactors and their use cases are listed below.

1. Electromagnetic Contactors: These are the most common type. They work using a magnetic coil. When electricity flows through the coil, it pulls contacts together to switch the power on. They are used in water pumps, streetlights, and machines in factories.
2. Manual Contactors (Manual Starters): These contactors are turned on and off by hand. They do not use electricity to switch. Instead, a person has to press a button or move a lever. They are found in old school water pump panels or small workshop machines.
3. Solid-State Contactors: These do not have moving parts. They use electronic components instead of magnetic coils. They are faster, quieter, and last longer, but are more expensive. They are used in modern and sensitive machines like medical or food-processing equipment.
4. Definite Purpose Contactors: These are made for specific tasks like air conditioners, refrigerators, or heating systems. They are small, low-cost, and easy to replace. They are common in homes and small businesses like cold stores.
5. Reversing Contactors: These are two contactors used together to change the direction of a motor. They are useful in robotic arms, electric gates, or automatic grinding machines.

RELAYS – SMALL ELECTRIC SWITCHES

Picture your robot project: maybe it has a sensor that detects an object, and you want that sensor to turn on a light or start a motor. You do not want the sensor's tiny signal to directly control the motor because it is not strong enough and could get damaged. Instead, you use a relay, a device that acts like a middleman. A relay takes a small signal from your sensor (or your robot's code) and uses it to safely control something bigger, like a motor or light.

Here is how it works: a relay has an electromagnetic coil, which is like a tiny magnet that turns on when a small voltage (the control signal) flows through it. When the coil gets energised, it creates a magnetic field that pulls a switch (called an armature) to open or close electrical contacts. This switch can then control a bigger circuit, like turning on a motor or light, without the small signal touching the big power directly.

Now, compare this to contactors. Both relays and contactors work in a similar way by using a small signal to control bigger power. However, contactors are like the "heavyweight champions" built for really big jobs, like controlling huge motors in factories. Relays, on the other hand, are used for smaller tasks, like in your robot or in home appliances, where the power is not as massive.

So, in your robotics projects, relays are like the control switches you use to safely manage motors or lights with small signals. In Ghana, they are used in automated irrigation, building controls (air conditioners, lights) and simple factory systems.



Figure 3.6: A typical relay without its outer casing [https://www.explainthatstuff.com/howrelayswork.html]

Resource Link	QR Code
How Relays work.	

Types of Relays

Below are some common types of relays and their applications.

1. Electromechanical Relay (EMR): It uses a coil and mechanical contacts for general-purpose switching, industrial control panels, and automation systems. It is simple, reliable, and cost-effective.
2. Solid-State Relay (SSR): It uses semiconductor components (no moving parts) to provide high-speed switching and silent operations and is mostly used in environments requiring durability. It has a long lifespan, generates fast response, and is vibration resistance.
3. Thermal Overload Relay: Senses heat generated by excessive current to provide motor protection against overloads. It prevents motor damage by disconnecting the circuit during overheating.

4. Time Delay Relay: It introduces a delay before switching on or off and is used in Sequential control systems, automation, and lighting control. It enables timed operations and process coordination.



Figure 3.7: Common types of relays. [https://www.quisure.com/blog/faq/what-are-the-common-types-of-relays]

Resource Link	QR Code
What are the common types of relays [reading resource]	

SUMMARY TABLE TO COMPARE ACTUATORS

Feature	Valves	Hydraulic Actuators	Contactors	Relays
Energy Type	Electric / fluid	Hydraulic fluid	Electricity	Electricity
Control Signal	Electric / pressure	Hydraulic pressure	Low voltage	Low voltage
Output Action	Control fluid flow	Push or spin parts	Switch big circuits	Switch small circuits
Application Environment	Factories	Big machines	Electrical systems	Control systems
Power Handling	Medium–High	Very High	High	Low–Medium
Speed (Response Time)	Fast (solenoids)	Medium	Medium–Fast	Fast

Activity 3.1 Choosing the Right Actuator

In this activity, you will explore how valves, hydraulic cylinders, contactors, and relays work by building simple models, testing simulations, and applying them to real-world scenarios in your assigned group.

Resources: Simple materials (e.g., straws, syringes, cardboard, toothpick, thread, switches, wires, small LED/buzzer), actuator diagrams/videos, paper/whiteboard, digital drawing tools, or simulation software (if available)

Instructions

1. Getting Started

   Actuators make things move in automation, like motors in your robots! In your group, watch the video in the resource link below or any DIY video on how to build actuators. Focus on four actuators: valves, hydraulic cylinders, contactors, and relays.

Resource Link	QR Code
How to make remote control Hydraulic Crane from Cardboard.	

2. Building and Analysis of Actuators
   a. In your group, pick one actuator and build a simple model:
      i. Valve: Use a straw and tape to mimic a gate controlling water flow.
      ii. Hydraulic Cylinder: Use a syringe filled with water to push a piston (another syringe).
      iii. Contactor/Relay: Connect a switch to a low-voltage circuit with an LED/buzzer to simulate controlling a larger load.
   b. Test your model (e.g., push water through the syringe, toggle the switch). Answer the following questions:
      i. Energy source: What powers it (e.g., water pressure, electricity)?
      ii. Control signal: What triggers it (e.g., pushing syringe, flipping switch)?
      iii. Internal mechanism: How does it work (e.g., piston moves, switch closes circuit)?
      iv. Function: What does it do in a system?
      v. Applications: Where is it used in Ghana (e.g., relays in small pumps, valves in water treatment)?

c. Sketch a labelled diagram of your model's internal parts (e.g., syringe piston for hydraulic cylinder).

Use the template below if needed.

Actuator Name:

Feature	Answer
Energy Source	
Internal Mechanism	
Control Signal	
Function	
Applications	

3. Solve a Real-World Scenario

Scenario

A small factory in Bawku wants to automate the movement of heavy drums in its storage unit. The factory needs an actuator that can lift and lower these drums safely and repeatedly.

Choose one actuator from the following list that you think is most suitable: Hydraulic cylinder, Relay, Contactor, Valve.

Answer the following in your exercise book or notebook:

a. What energy source does your chosen actuator use?

b. How does it receive and respond to a control signal?

c. Why is it the best fit for lifting heavy drums?

d. Where else in Ghana have you seen or heard of this actuator being used?

e. Draw and label a simple diagram showing how the actuator works in the scenario.

4. Share Your Findings

Show your model or diagram to the class and explain one key finding (e.g. how your relay model switches power).

Discussion: How do these actuators relate to your robot projects?

5. Compare Two Actuators:

Compare your group's actuator with that of another group. Compare:

a. Energy type: Electricity, hydraulic fluid, etc.

b. Output action: Switching, pushing, etc.

c. Power handling: Small loads (relays) or large loads (hydraulic cylinders)?

d. Response time: Fast (relays) or slower (hydraulic cylinders)?

Feature	Your Group's Actuator	Different group's actuator
Energy Type		
Output Action		
Power Handling		
Response Time		

SELECTING FINAL CONTROL ACTUATORS

Here, you will explore how industrial actuators are used to solve real-world automation problems. You will look at different control tasks like moving heavy objects, managing fluid flow with precision, or safely operating high-power electrical systems. You will be able to confidently recommend actuators for different industrial challenges and support your decisions with clear reasoning.

Recap: What Are Actuators?

Actuators are like the muscles of machines. They take a signal like a button press or a computer command and turn it into action, such as lifting a heavy load, turning a wheel, or controlling water flow. Think of them as the doers or actors that bring robots and industrial systems to life! You will focus on four key actuators: valves, contactors, relays, and hydraulic actuators.

Valves: Control the flow of liquids or gases, like a tap managing water in a treatment plant.

Contactors: Heavy-duty switches that safely turn on or off big machines, like factory motors.

Relays: Small switches for low-power signals, like controlling a light in your robot.

Hydraulic Actuators: Use liquid pressure to lift or push heavy things, like a car jack.

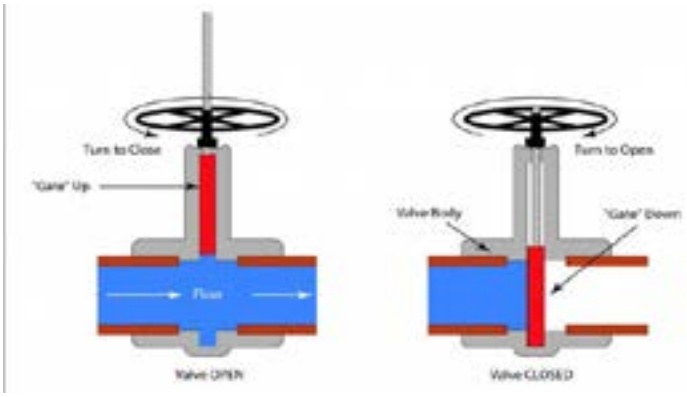


Figure 3.8: Diagram of a manually operated valve opening to control water flow in a pipe.

Think About It

In your notebook, write one sentence about what each actuator does. Example: "A contactor switches on a big motor safely."

HOW TO CHOOSE THE RIGHT ACTUATOR

Choosing an actuator is like picking the best tool for a job. You need to think about what the task requires.

Here is a simple checklist to guide you, making sure everyone can find the perfect actuator:

1. What movement is needed? Pushing or pulling (linear, like lifting a bag), turning (rotary, like a conveyor), controlling flow (like water), or switching power (like a motor)?
2. How much strength? Big force for heavy loads (e.g., 3 tonnes) or small signals (e.g., a light)? Hydraulic actuators are super strong; relays handle tiny signals.
3. How fast? Quick actions (e.g., packing items) or slow and powerful (e.g., lifting)? Contactors and relays are fast; hydraulic actuators are slower.
4. How precise? Need exact control (e.g., measuring water flow)? Valves can be precise for fluids.
5. Where is it used? Hot, dusty factories? Wet water plants? Choose actuators that will not break in tough conditions.
6. What power is available? Electricity, compressed air, or hydraulic fluid? Pick what the site has.
7. Is it safe? Must it avoid shocks or stop safely if power fails? Contactors and valves often have safety features.
8. Cost and upkeep? Can the company afford it? Is it easy to fix using local materials? Relays are cheap and simple to maintain.

Try it Out

List two tasks (e.g., lifting a box, turning on a pump) and use the checklist to guess which actuator might work best for each.

ACTUATOR STRENGTHS AND WEAKNESSES

Each actuator has its own superpowers and limits. Check out this table to see what they do best and where they need help:

Actuator	How It Works	Best For	Limitations
Hydraulic Actuator	Fluid pressure moves piston	Lifting heavy loads	Slow, may leak
Pneumatic Actuator	Compressed air moves piston/rotor	Fast, repetitive, moderate force tasks	Noisy, limited precision

Electric Actuator (DC/ AC Motor)	Electrical energy drives motion	Controlled rotation, conveyors	May overheat with prolonged use
Servo Motor	Motor with feedback system	Precision control robotics	Higher cost
Solenoid	Electromagnetic motion	On/off pushing or pulling	Limited to short stroke and force
Valve	Controls fluid flow	Water or gas control	Specific to flow type
Contactor	Switches high power	Big motors	Not for small signals
Relay	Switches low power	Signal control	Not for heavy loads

ACTUATORS IN GHANA: REAL-LIFE EXAMPLES

Consider actuators in action! Imagine you are an engineer at a metal workshop in Accra. The task is to lift steel beams weighing over 3 tonnes. Here are the considerations to help you choose:

1. Choice: Hydraulic actuator (cylinder)
2. How It Works: Pressurised liquid pushes a piston to create strong lifting force.
3. Why It Fits: It is powerful enough for 3 tonnes, safe with proper setup, and common in Ghana's metal workshops and vehicle garages. Local mechanics can fix it if it leaks.
4. In Ghana: Hydraulic actuators are used in car repair shops to lift vehicles or in construction to move heavy materials.



Figure 3.9: A hydraulic actuator for lifting goods [https://www.honeywellhydraulics.com/hydraulic-cylinder-for-goods-lift/]

Now, picture a water treatment plant in Kumasi that needs to control water flow and switch on a large pump. A valve controls the water, and a contactor handles the pump's power. Why? Valves precisely manage flow, and contactors safely switch high power, both common in Ghana's water systems.

Activity 3.2 Pick and Justify Your Actuator

Scenario

A Ghanaian factory wants to lift fifty 80 kg bags of cocoa onto a conveyor belt. The area is hot and dusty, with no compressed air, but electric power is available. The system must be safe, reliable, and require little maintenance.

Your Task

Complete the steps below. You may write, speak, or draw your answers. Learners with visual, hearing, or writing difficulties may use assistive tools or alternative formats with support from the teacher.

1. Choose an Actuator and explain how it works. Pick one:
    a. Hydraulic actuator
    b. Valve
    c. Contactor
    d. Relay
2. Use the checklist below to support your explanation:
    a. Motion type
    b. Force needed
    c. Environment
    d. Power availability
    e. Safety and maintenance
3. Give a Local Example

    Where is this actuator used in Ghana? E.g., a cocoa factory, water plant, or workshop.
4. Show How It Works: Draw or describe how your actuator performs the task.
5. Present your work to the class. Use your own preferred method (e.g., speaking, showing a diagram, reading aloud, etc.).
6. After each presentation:
    a. Ask a question or share one insight
    b. Compare different actuator choices

    The teacher will summarise the key learning points and write them on the board or display them visually for everyone.

Reflection Prompt

Respond to one:

1. One thing I learned from a classmate is...
2. I would improve my actuator choice by...
3. Now I understand that actuators must...

EXTENDED READING

RESOURCE LINK	QR CODE
Solenoid Valve Explained | Types and Application (Video)	
Introduction to Hydraulic Cylinders: Cylinders, Inc. (2024, August 14). *What is a hydraulic cylinder? How it works, types, uses*. Cylinders, Inc.(Web Publication)	
Understanding Electrical Contactors: RS Components. (n.d.). A complete guide to contactors. RS Components. (Web Publication)	
Relay vs Contactor Comparison: Wevolver. (2023, September 19). *Contactor vs Relay: Understanding the differences and applications*. Wevolver. (Web Publication)	
Overview of Industrial Actuator Types: Tameson.com. (2023, February 10). *What is an actuator - Types and applications*. Tameson.com. (Web Publication)	

REVIEW QUESTIONS

1. What type of valve uses a ball to control the flow of liquids or gases?
2. According to Pascal's Law, what is the formula used to calculate force in a hydraulic system?
3. What type of contactor is commonly used in water pumps, streetlights, and factory machines?
4. Name the advantage of a solid-state relay over an electromechanical relay.
5. How is the function of a valve in an industrial system similar to the movement control of a robotic arm?
6. How does the energy transfer in a hydraulic cylinder differ from that in a robotic arm, and why is this difference important in industrial applications?
7. Why are electromagnetic contactors safer to use than manual contactors in high-power systems like water pumps or factory machines?
8. Why are contactors preferable to relays for use in industrial machinery?
9. Industrial systems often require the safe and reliable switching of high-power motors.

   Explain the role of relays, valves, contactors, and solenoids in controlling electrical or mechanical systems. Based on your analysis, recommend the most suitable actuator for switching a high-power industrial motor ON and OFF, and justify your choice using technical reasoning and real-world examples.
10. You are tasked with selecting the most appropriate actuator to switch a high-voltage industrial pump ON and OFF. Four actuators relay, valve, contactor, and hydraulic cylinder are available.

    Analyse the function, strengths, and limitations of each actuator in this context. Then, rank them from most to least suitable for safely and effectively switching ON/OFF a high-voltage industrial pump. Justify your ranking with technical reasoning and practical examples.
11. Why would a hydraulic cylinder be unsuitable for controlling the direction of a fan in a clean room laboratory?
12. A water pumping system in Tamale needs to control flow and start a large motor.

    Recommend a combination of two actuators and explain why each one is suitable, referring to their operating principles.
13. A factory in Kumasi has a dusty environment and needs to automatically switch on a motor for a conveyor belt every morning. Which actuator would you recommend between a relay and a contactor? Justify your answer.

REVIEW QUESTIONS

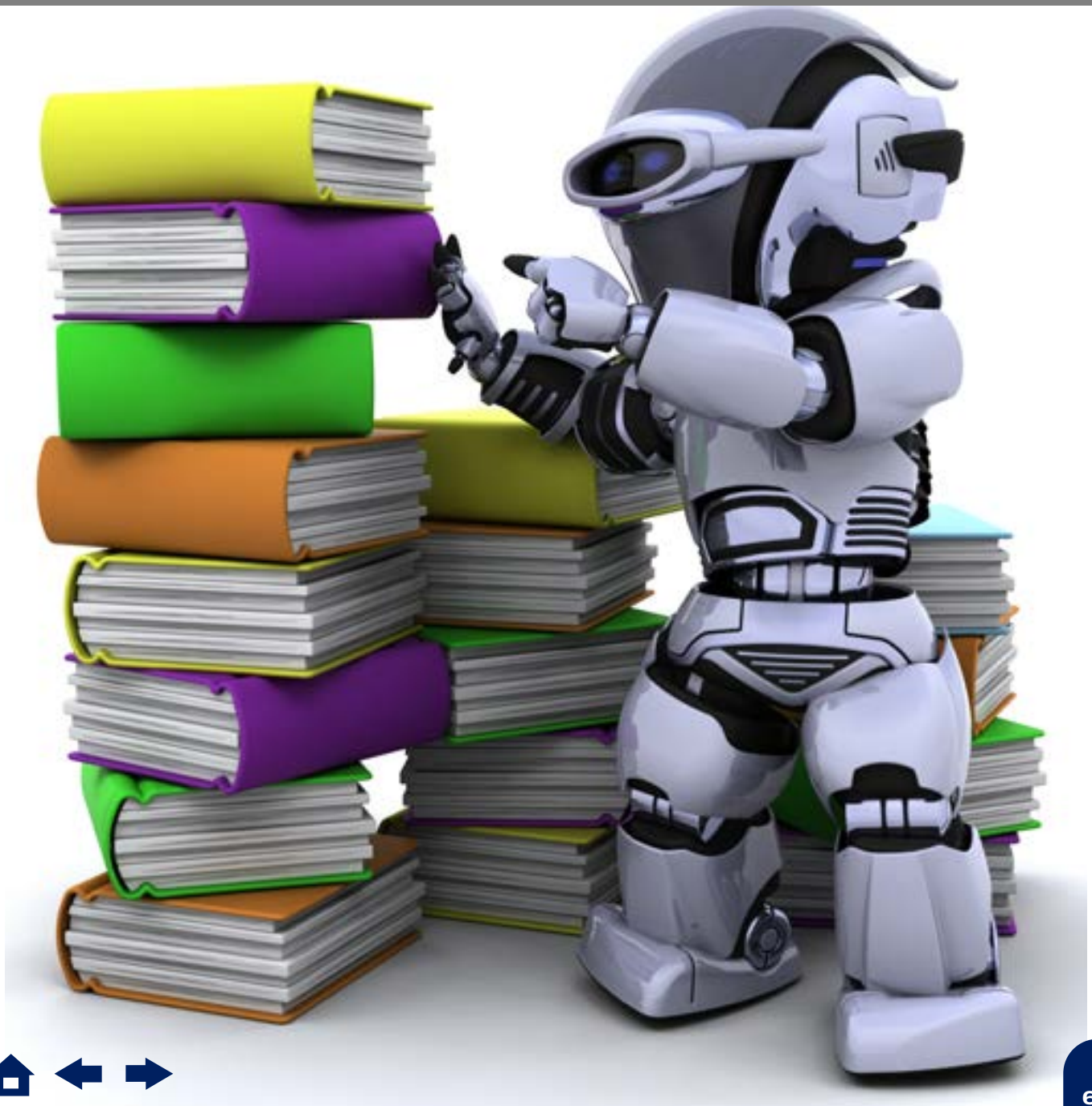
14. Match each actuator type below with the most suitable industrial task in Ghana and justify your match with one reason.

Actuator	**Industrial Task**	**Reason** (1 sentence)
Valve	A. Switching on a classroom light	
Relay	B. Starting a heavy grinder	
Contactor	C. Lifting metal beams in a garage	
Hydraulic Cylinder	D. Controlling water in a treatment plant	

SECTION

4

DIGITAL AND ANALOGUE SYSTEM DESIGN 3



Digital and Analogue System Design

INTRODUCTION

Hello, future robot detectives! You have already learned how to look at a working robot and figure out its secrets, like what its purpose is and how its parts move. That is an amazing skill! Now, get ready to dive even deeper and discover the inner workings of a robot's brain. This is where you will learn to speak its secret language and understand how it makes lightning-fast decisions. By the end of this journey, you will not just be building robots; you will be able to look at any unknown circuit and figure out exactly what it does, how to make it simpler, and how to build it yourself. This is the ultimate power for a robotics engineer!

KEY IDEAS

- Boolean Expressions serve as a mathematical representation of a circuit's logic, translating its behaviour into a formal language for analysis and design.
- Circuit Implementation involves constructing a physical or simulated circuit based on a logic gate diagram, translating theoretical designs into functioning systems.
- Logic Simplification focuses on optimising Boolean expressions to minimise factors like cost, power consumption, and physical size, utilising techniques such as Karnaugh Maps.
- Real-World Applications highlight the relevance of digital circuit knowledge in various fields, enabling solutions to everyday engineering and computational problems.
- Reverse-Engineering provides a structured method for dissecting existing circuits to uncover their components and control logic, allowing for better understanding and potential improvements.
- Systematic Observation entails testing all potential input combinations of a circuit to document and analyse its output behaviour, forming a comprehensive basis for further analysis.

THE ART OF REVERSE-ENGINEERING DIGITAL CIRCUITS

Remember how you became a detective to figure out a robot's behaviour? We are going to use those same amazing skills to crack the code of combinational circuits. Think of a combinational circuit as the simplest kind of robot brain; it has no memory and always gives the same output for the same input. It is like a super-fast, simple decision-maker.

YOUR CIRCUIT'S SECRET CODEBOOK: THE TRUTH TABLE

When you were a robot detective, you created a secret codebook for a robot's behaviour just by watching what it did. A truth table is exactly that, but for a digital circuit! It is the complete blueprint that shows every possible input combination and the exact output the circuit produces for each one. Think of it as a set of rules. For example, if you give a circuit one set of inputs (its 0s and 1s), the truth table tells you exactly what the output should be. You will systematically test every single combination.

How it works

1. You will apply every binary combination (using 0s and 1s) to the circuit's inputs and meticulously record what happens. A 0 means low voltage, which is a signal to a computer to turn off, and a 1 means high voltage, a signal to turn on. An easy way to see this is with an LED; if the light is off, that is a 0, and if the light is on, that is a 1.
2. You will repeat this for every single input combination until your truth table is complete.

HOW TO CONSTRUCT A TRUTH TABLE FROM A PHYSICAL "BLACK-BOX" CIRCUIT

Now it is time to put your detective skills to work! We are going to build a truth table from a physical circuit, just like a real engineer. Imagine you are at a local company in Ghana that prepares fruits for a delicious orange drink. They have a machine that sorts oranges, and you have a mysterious circuit from it. Your mission is to figure out how this circuit decides which oranges are good enough for the drink.

This is what your circuit's "senses" tell it:

1. Sensor A: Detects ripeness - 1 if the orange is ripe, 0 if it is not.
2. Sensor B: Detects colour - 1 if the orange is yellow, 0 if it is green.
3. Sensor C: Detects sweetness - 1 if the orange is sweet, 0 if it is sour.

The single output, Z, lights up an LED to show if the orange is selected for the drink.

Here is how you will solve this mystery

1. Label your clues: Make sure you clearly label your inputs as A, B, and C, and your output as Z. This keeps everything neat.
2. Prepare your notepad/document: Draw a blank truth table on a piece of paper. It should have columns for your three inputs and one for the output. Remember, with three inputs, there are eight (23) different combinations you need to check! This is shown in Table 4.1.

Table 4.1: Blank Truth Table with 3 inputs (A, B, C) and 1 output (Z)

A	B	C	Z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

3. Set the switches: For each row on your truth table, set the physical switches or connections on your "black box" circuit to the corresponding 0s and 1s. This is you telling the circuit what to "see."

   Safety First: Before you connect anything, always make sure the power is disconnected. This is a crucial rule for all robotics engineers. Once your switches are set, you can connect the power supply.

4. Observe and record: Look at the LED on the output. If it lights up, the output is a 1. If it stays off, the output is a 0. Write this down in the Z column of your table.

5. Repeat: Now, disconnect the power, change your switches to the next combination, and repeat the process. Continue until you have tested all eight combinations and your entire truth table is complete.

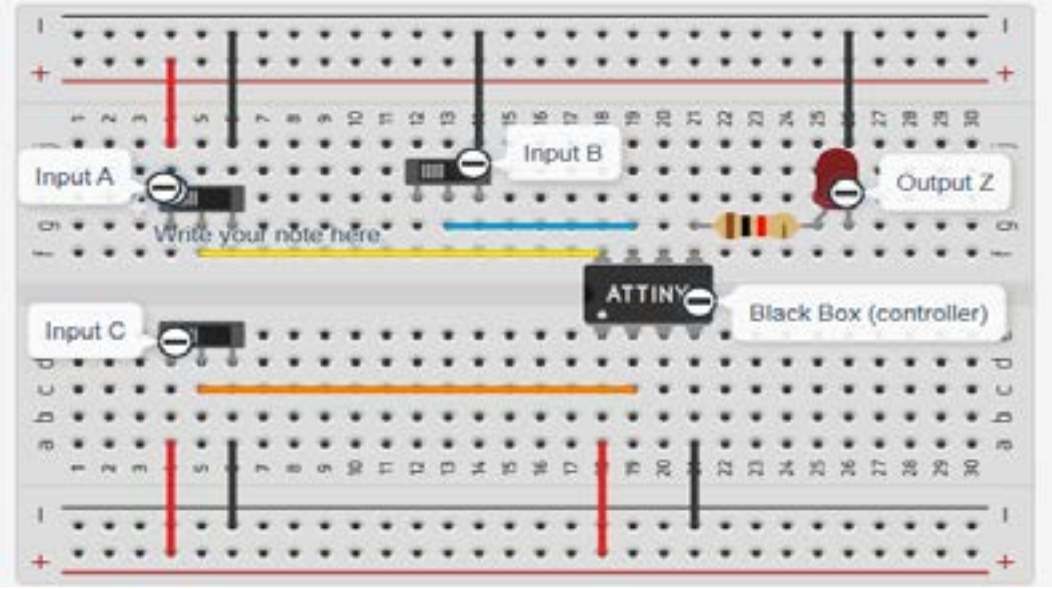


Figure 4.1: A simplified "black box" circuit with inputs controlled by switches and an output indicated by an LED.

BOOLEAN EXPRESSIONS: FORMALISING THE INFERRED CONTROL LOGIC

Once you have your truth table, you have the data you need to translate the circuit's behaviour into a mathematical language. This is called Boolean algebra, and it is the formal way to describe how a digital circuit makes decisions. The easiest way to do this is using the sum-of-products (SOP) form, which you have already been introduced to in your second year. It is a straightforward process:

1. Focus on the 1s. Look at your truth table and find every row where the output is a 1. These are the conditions where the circuit turns "on."
2. Create product terms. For each of those 1s, write a "product term." A product term is a way of showing inputs combined with an AND relationship. If an input is a 0 in that row, you will use its opposite form (for example, A' or). If it is a 1, you will use the input just as it is (A).
3. Combine with an OR. Finally, you join all your product terms together using an OR relationship.

For example, for our orange sorting machine, imagine if the truth table looks like what is presented in ***Table 4.2.***

Table 4.2: Truth Table of the Orange Fruit Sorting Machine

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The rows where 'Z' is '1' are:

$$A = 0, B = 0, C = 1 \rightarrow A'B'C \text{ or } \overline{A}\,\overline{B}$$
$$A = 0, B = 1, C = 0 \rightarrow A'BC' \text{or } \overline{A}B\overline{C}$$
$$A = 1, B = 0, C = 1 \rightarrow AB'C \text{ or } A\overline{B}C$$
$$A = 1, B = 1, C = 0 \rightarrow ABC' \text{ or } AB\overline{C}$$

The un-simplified Boolean expression would be: Z = A'B'C + A'BC' + AB'C + ABC'.

This expression is a perfect mathematical description of your circuit's behaviour!

MAKING IT SIMPLE: LOGIC SIMPLIFICATION

When you first write a Boolean expression from a truth table, it can be long and complicated. It is like telling a long story when a few simple words will do. Simplifying your expression means using fewer terms and variables, which has amazing benefits:

1. Saves money: Fewer terms mean you need fewer logic gates, which lowers the cost of building the circuit.
2. Saves power: A simpler circuit uses less electricity, which is great for battery-powered robots.
3. Smaller size: Fewer gates mean a smaller, more compact circuit.
4. Faster decisions: A simpler circuit can often make decisions more quickly.

You will use a tool you were introduced to in Year 2: the Karnaugh Map (K-Map). It is a smart grid that helps you visually find and group common terms in your expression so you can make it much simpler. Once you simplify your expression, you have a new, more efficient blueprint to build from.

REMINDER ON HOW TO USE A K-MAP: THE LOGIC PUZZLE

Think of the Karnaugh Map, or K-Map, as a special kind of puzzle grid that helps you find the simplest way to describe a circuit's logic. The map arranges the 0s and 1s from your truth table in a clever way. It places them so that any two adjacent squares are different by only one variable. This is the secret trick that makes simplifying so easy!

Here are the rules for solving the puzzle by grouping the 1s

1. Rule 1: Powers of Two. Your groups must contain 1s in numbers that are a power of two. This means your groups can be made of 1, 2, 4, 8, or more 1s.
2. Rule 2: Rectangles Only. Your groups must be in a perfect rectangular or square shape.
3. Rule 3: Wrap Around. You can wrap your groups around the edges of the map, like a video game screen. The top and bottom edges are next to each other, and so are the left and right edges.
4. Rule 4: Be a Minimalist. Always make the largest possible groups first. You want to use the fewest groups possible to include all the 1s.
5. Rule 5: Get All the 1s. Make sure every single 1 on your map is included in at least one group. You can reuse a 1 in more than one group if it helps you form a larger group.

For each group you create, you will write a simplified term. You look at the variables that stay the same across all the squares in that group. Any variable that changes (from 0 to 1 or 1 to 0) gets eliminated. The variables that stay constant form your new, simpler term.

Finally, you will combine all your simplified terms together using an OR relationship to get your final, perfectly simplified Boolean expression. So, for the orange sorting machine example, the simplified Boolean expression would be generated using the K-Map as shown in ***Figure 4.2.***

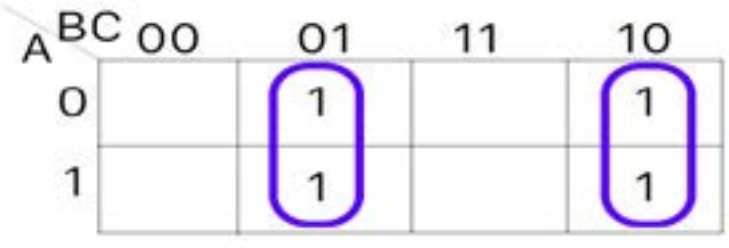


Figure 4.2: K-Map of the Orange Fruit Sorting Machine Black-box

From the K-Map, the simplified Boolean expression is Z = B'C + BC'

This simplified expression is a very common one and is known as the **Exclusive** OR or **XOR** function. It means the output is a 1 when either input B or input C is a 1, but not both, as such it can also be expressed as $Z = B \oplus C$

BRINGING THE LOGIC TO LIFE: BUILDING YOUR CIRCUIT

The final and most exciting step is to turn your simplified logic into a real, working circuit! This is done by following these steps:

1. **The Blueprint**

The Logic Gate Diagram: The first thing you will do is create a logic gate diagram. This is your blueprint. Using the simplified Boolean expression you created, you will draw a picture using special symbols for AND, OR, and NOT gates. For our orange sorting machine, the simplified expression was Z = B'C + BC'. This logic is special because it is also known as an Exclusive OR or XOR function. Your blueprint for this would look like this:

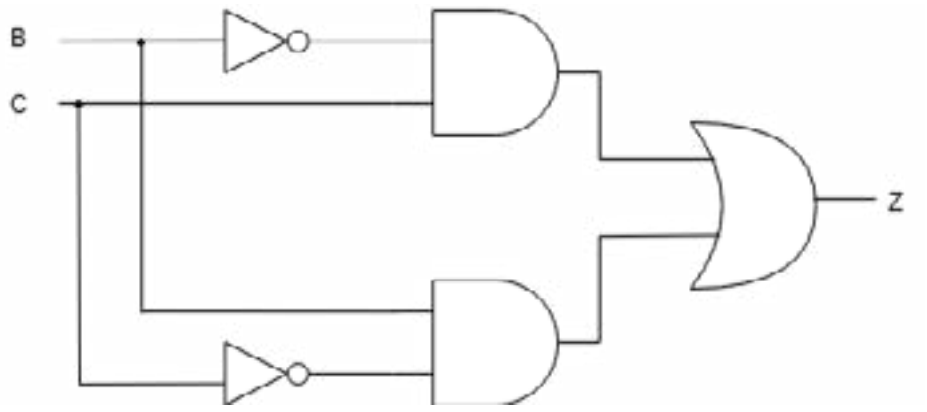


Figure 4.3: Logic Gate Diagram for **c**

Remember that this is the same as $Z = B \oplus C$ shown in Figure 4.4.

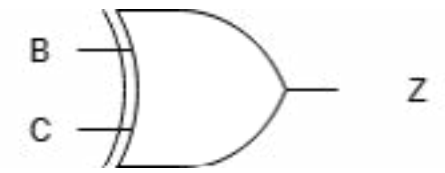


Figure 4.4: Logic Gate Diagram for $Z = B \oplus C$

2. **Bringing the Blueprint to Life**

Now it is time to build your circuit. You have two great options:

Option 1: Build a Physical Prototype

This is a wonderful way to get hands-on experience. You will use a breadboard (a board with holes to plug components into without soldering) and small chips called Integrated Circuits (ICs) that contain the logic gates you need. You will carefully connect the wires, switches, and LEDs according to your logic gate diagram. Just remember to be careful and pay attention to which pins on the chips are for power, ground, and inputs/outputs.

Option 2: Use Software Simulation

A digital circuit simulator is a fantastic tool that lets you build and test your circuit on a computer screen. This is a safe and efficient way to make sure your design works perfectly before you even touch any physical components.

3. **The Final Test**

Once your circuit is built (either physically or in a simulation), you must do the most important step: test and verify it! Go back to your original truth table and test every single input combination on your new circuit. Does the actual output match what you wrote down in your table? If it does not, do not worry! This is a normal part of the process. You are now in troubleshooting mode. Go back and check your work, piece by piece. Is your wiring correct? Did you make a mistake in your K-map? Did you write your Boolean expression correctly? This process of finding and fixing mistakes is a key skill for any engineer.

PUTTING YOUR SKILLS TO WORK: REAL-WORLD ROBOT APPLICATIONS

The skills you have just learned are not just for the classroom; they are used every day to build incredible things! By mastering how to reverse-engineer and simplify logic, you are ready to create your own smart systems that solve real problems.

1. Smart Security: Imagine designing a security light for a compound that only turns on when a motion sensor is triggered, and it is after sunset. Or think about a school gate that only opens when a valid ID card is scanned, and the person stands on a pressure pad.
2. Automated Farming: The knowledge you have gained can help farmers. You could create a circuit for an irrigation system that activates a water pump only when a soil moisture sensor detects that the ground is dry and the sun has set (to prevent water from evaporating too quickly).
3. Textile Factory Safety: You could design a safety system for a sewing machine that makes sure the machine's motor can only turn on when a safety cover is in place and the emergency stop button is not pressed. This is called a safety interlock.

4. Traffic Management: Even a simple traffic light at a busy intersection uses combinational logic! It can use sensors to detect if a vehicle is waiting and change the light when the timer for the other direction runs out.

These examples are just the beginning. You could also design

1. A Smart Vending Machine: A circuit that checks if you have inserted the correct amount of money and pressed the button for a specific drink before releasing it.
2. An Automated Room Fan: A system that turns on a fan only when the room temperature is above a certain point and someone is inside the room (detected by a motion sensor).

The ability to translate a problem into logic and simplify it is a powerful skill. What other problems can you think of that a simple circuit could solve?

You are now equipped with an incredible set of skills. Here are some activities to help you master them!

Activity 4.1 The Logic Puzzle Challenge (Individual)

Your Mission: Your teacher will give you a list of different truth tables. Your mission is to derive the simplified Boolean expression for each one. This is a great way to practice your K-Map skills.

The Challenge

1. For each truth table, create a K-Map.
2. Clearly show your groupings on the map.
3. Write down the final simplified Boolean expression.
4. Can you describe in words the fundamental logical function of each circuit?

Activity 4.2 The Real-World Application Challenge

Before starting this activity, organise yourself into a group of 3 to 5 people.

Your Mission: Your group will choose a real-world problem from your community that can be solved with a simple circuit. Think about things like an automated security light or a machine that sorts different items.

The Challenge

1. Define the Logic: Identify the inputs (sensors) and the outputs (actuators). Create a truth table for your system.
2. Formalise and Simplify: Derive the Boolean expression for your system and then use a K-Map to simplify it.

3. Build and Test: Draw the final logic gate diagram for your simplified circuit. Then, either build it using simulation software or a breadboard.
4. Present Your Solution: Present your project to the class. Explain the problem, show your truth table, explain your simplified logic, and demonstrate that your final circuit works as intended.

EXTENDED READING

Resource	QR Code to Access Resource
1. Logic Gates and Truth Tables - Step-by-Step Lesson from the Classroom (Video Resource).	
2. Boolean Algebra and Logic Gates (Web Publication)	
3. Karnaugh Maps and Their Use in Boolean Algebra Simplification (Web Publication)	
4. DigiSim - Digital Logic Simulator (Online Simulator)	

REVIEW QUESTIONS

The “Safe Cargo” Alert System

A local logistics company in Ghana wants to install a simple automated warning system in its delivery trucks to ensure the safe transportation of goods, particularly on rough roads or during bad weather. The system needs to warn the driver with an alarm when certain conditions are met, protecting both the cargo and the vehicle.

The inputs are as follows:

S (Speed Sensor): A 1 if the truck is moving at a high speed, a 0 if it is at a low speed or stopped.

L (Load Sensor): A 1 if the truck is carrying an overloaded cargo, a 0 if the load is within safe limits.

H (Hill Sensor): A 1 if the truck is on an uphill incline, a 0 if it is on a flat road or going downhill.

W (Weather Sensor): A 1 if the weather is rainy, a 0 if it is clear.

The output is as follows:

A (Warning Alarm): A 1 if the alarm should be **ON**, a 0 if it should be **OFF**.

Rules for the Warning Alarm (A):

1. The alarm (A) must be **ON** if the truck is at high speed (S=1) **AND** it is raining (W=1).
2. OR, the alarm (A) must be **ON** if the truck is overloaded (L=1) **AND** it is on an uphill incline (H=1).
3. Critical Condition: The alarm (A) must **always** be **OFF** if the truck is stopped or at low speed (S=0), regardless of any other input conditions. This rule overrides all others.

Questions

1. Truth Table (10 marks) Create a complete truth table for the Warning Alarm (A) output, including all possible input combinations for S, L, H, and W. Populate the A column based on the specified rules. Explicitly note how the critical override condition affects the output.
2. Boolean Derivation & Simplification (15 marks) Derive the un-simplified Sum-of-Products (SOP) Boolean expression for the Warning Alarm (A) output from your truth table. Then, draw a Karnaugh Map for the A output and use it to simplify the Boolean expression to its most minimal form. Clearly show all groupings on the K-map.

REVIEW QUESTIONS

3. Logic Diagram (5 marks) Draw the logic gate diagram for the simplified circuit using standard logic gate symbols.
4. Implementation and Testing (15 marks) Implement your simplified circuit using either software simulation or a physical breadboard prototype. Provide a clear screenshot of your functional circuit in the simulator or a photograph of your physical prototype. Develop a test plan by listing at least five critical input combinations that rigorously test all aspects of the rules, including the override condition. Execute your test plan and document the observed outputs for each test case.

SECTION

5

ROBOT PROGRAMMING



Programming Robots in Python

INTRODUCTION

In the past two years, you have built simple robots and used block-based programming to control how they moved and responded. You connected sensors, used motors, and gave basic commands. The robots you built in Year 2 can now be taken to the next level by learning how to control them with code you write. Instead of just switching things on and off by hand, you will use a text-based programming language to tell your robot exactly what to do—like how fast to move, when to stop, or how to respond to changes around it. You will not need to build a whole new robot from scratch; you could upgrade the ones you already made by adding a small computer (like a micro: bit or Raspberry Pi Pico) and writing instructions for it. This means you will be learning how to think like a programmer while still using the creations you have already built. It is a new kind of challenge—and a big step toward building smarter, more responsive robots.

This section continues your journey in robotics by showing you how to write instructions using Python, a text-based programming language used around the world. It will introduce you to writing clear commands using words and symbols instead of dragging blocks. You will learn how to control motors, respond to sensors, and create actions that are more advanced. The skills gained in this section will help you build smarter robots that can think and act with more detail.

KEY IDEAS

- Conditional Logic and Control Flow are essential for programming decisions, enabling robots to respond intelligently to various conditions using constructs like if, if-else, and loops (while, for), which guide the sequence of execution based on real-time inputs.
- Data Types and Variables provide the foundational knowledge necessary for handling different types of data (integers, floats, Booleans, strings) effectively within Python, essential for storing and manipulating information in robot control systems.
- Function Design and Modular Programming cultivate organised coding practices by emphasising the creation of reusable functions, which streamline code development and facilitate scalability in robotic applications.
- List Structures and Data Management introduces students to storing and handling sequences of data, such as sensor readings and robot actions, allowing for more complex data manipulation and organisation in robotics projects.
- Text-Based Programming with Python equips learners with important skills as they transition from block-based to text-based coding, enhancing precision and flexibility, and exposing them to industry-standard programming tools for modern robotics.

- User Input and Interactive Control emphasise the importance of real-time interaction in programming, enabling learners to design systems that respond dynamically to user input and sensor data, enhancing the functionality and user experience of robotic systems.

UNDERSTANDING PROGRAMMING IN ROBOTICS

In this part of the section, you will move from block-based programming (which uses drag-and-drop blocks) used in Year One to text-based programming using Python. This means that instead of putting together coloured blocks as you did with the LEGO SPIKE™ app, you will now write your own instructions in words and symbols.

Python is one of the most popular programming languages in the world. It is used to control many real robots, machines, and computer systems. In this section, you will learn how to do the following.

1. Write simple Python code
2. Use variables to store values like motor speed or sensor readings
3. Understand data types like numbers, words, and true/false values
4. Perform operations like adding, subtracting, or comparing values
5. Create and use lists to store a group of items

This part of the section will also help you use what you already know, like how to follow steps in order (sequencing), store information (variables), and make decisions (logic). But now, you'll do it by typing commands instead of using blocks.

The goal is to help you turn algorithms and flowcharts (which you learned to draw before) into real Python code that can control a robot. This helps you understand how clear instructions can make robots do complex tasks.

MOVING FROM BLOCK CODING TO PYTHON

In Year One, you started robotics using a visual tool called the LEGO SPIKE™ Prime App. It allowed you to build programs by dragging and snapping blocks together. Each block represented a command like moving a motor or waiting for a few seconds. This helped you focus on how your program works without worrying about spelling or typing.



Figure 5.1: Coding blocks in Block programming

Now that you have gained experience, it is time to move on to text-based programming using a language called Python. With Python, instead of dragging blocks, you type real code using letters, numbers, and symbols. This kind of programming gives you more power, control, and flexibility to build more advanced robot behaviours.

At first, typing code may feel different, but it helps you learn skills that are used in real-world robotics and engineering.

COMPARING BLOCK CODE AND PYTHON CODE

Block-based programming uses pre-shaped visual pieces that fit together like a puzzle. For example, you can easily connect a "move motor" block to a "wait" block. This ensures your commands are in the right order.

In Python, you have to type each instruction as a line of code. The robot follows these lines one after the other, and the way you space (indent) your code matters a lot. Python does not use visual blocks, so it needs you to write your instructions clearly and carefully.

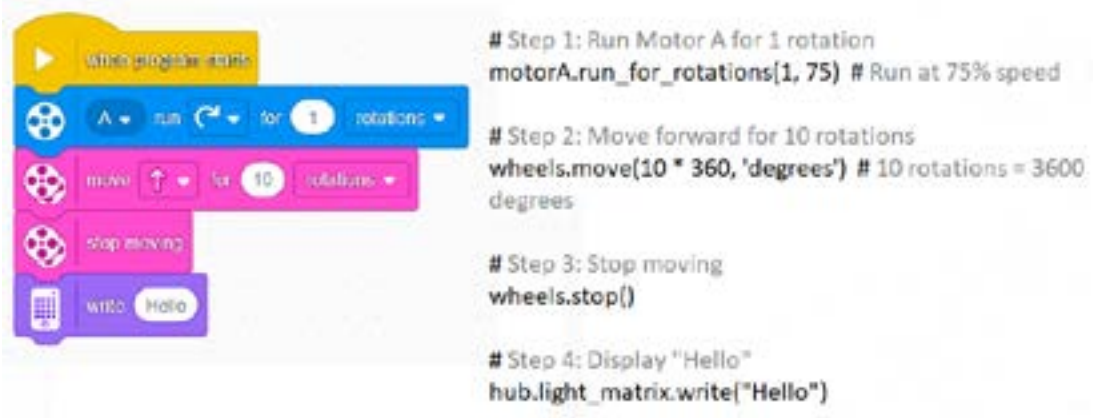


Figure 5.2: Coding Sequence in a Block-Based environment vs a text-based environment

Let us say you want your robot to move a motor, wait for a few seconds, stop and show a message. In block-based programming (like in the LEGO SPIKE™ app), you would simply drag and stack blocks, one for each action. This is demonstrated in the block in the ***Figure 5.2.***

In Python (text-based coding), you will write three lines of code. Each line gives one instruction to the robot as shown in ***Figure 5.2.*** This may look different, but it is doing the same thing, just written in a new way.

WHY MOVE TO TEXT-BASED PROGRAMMING?

As your robotics projects get more advanced, blocks start to limit you. Python lets you do much more.

Here is why switching is important.

1. You Can Build Bigger Programs Easily (Scalability): With Python, you will learn how to reuse code using functions and modules (We will look at these terms in

detail later in this section). This helps when building complex robots without rewriting everything.

2. Better Control of the Robot: Python gives you more precise control over sensors and movement. You can also customise robot behaviour exactly how you want it.
3. It is What Professionals Use: Python is used in real robotics, like drones, medical robots, and factory automation. Learning it prepares you for careers in engineering and tech.
4. Easier to Find and Fix Errors (Debugging): Python has tools to check your code step by step, find mistakes, and test changes. You can see what is wrong faster than with blocks.

TOOLS THAT HELP YOU TRANSITION FROM BLOCKS TO TEXT

Switching from drag-and-drop blocks to typing Python code can be challenging at first. But there are tools designed to make this change easier. Here are some free tools you can explore:

1. **EduBlock**

EduBlock lets you learn Python by using blocks first. As you drag and drop blocks to build a program, EduBlock shows you the real Python code being generated. This helps you see the connection between block commands and text-based code in real time. It's a great way to build confidence before writing full Python programs on your own.

2. **Tinker Coders**

Tinker Coders is an online platform that helps students learn Python and robotics in a fun and interactive way. It uses step-by-step lessons and block-to-text activities that guide you from beginner to more advanced skills. You can even simulate robots and test your code in a virtual environment—great for practicing when you don't have a physical robot.

3. **Blockly**

Blockly is a visual tool developed by Google that works like LEGO SPIKE blocks. It allows you to create programs by dragging blocks, and then it translates those blocks into Python, JavaScript, or other code. This helps you get used to reading and understanding text code while still using the blocks you're familiar with.

GETTING STARTED WITH PYTHON FOR ROBOTICS SIMULATION

Python is a popular programming language that is easy to read and write, which makes it perfect for beginners. Even though it is simple to learn, Python is also a powerful tool used by professionals to build websites, create software, study data, and even train robots using artificial intelligence.

To write and run Python programs, you must first check if Python is installed on your computer.

Activity 5.1 Checking and Installing Python on your Computer

In this task, you need to check whether your computer has Python installed on it and install it if it is not.

Tasks

1. Open the Command Line
   a. On Windows: Press Windows + R, type cmd, and press Enter.
   b. On macOS: Open Terminal from Applications > Utilities.
   c. On Linux: Open your terminal (shortcut may be Ctrl + Alt + T).
2. Type this in the terminal or command prompt and press Enter:

   python –version or python3 --version
3. Observe the result
   a. If you see something like Python 3.10.4 or any other higher or lower version, then Python is installed
   b. If you see a message like 'python' is not recognised as an internal or external command..., then Python is not installed.
4. Record your result in the checklist

Task	Done (✓)	Comment
Opened the terminal		
Entered the python version command		
Observed and recorded result		

Note

Comment for observed and recorded result can either be installed or not installed.

5. Prepare to Install Python (If Python is not installed)

   Follow the link below and watch how to download and install Python.

Link and Description	QR Code
Python Installation : Python Setup and IDE installation	

Note

After installation, follow steps 1, 2 and 3 once again to confirm your Python installation.

To write and run Python programs, we use special tools called IDEs (Integrated Development Environments). These tools help you type, test, and run your code easily.

Some popular Python IDEs include:

a. IDLE – comes with Python when you install it

b. Thonny – simple and great for beginners

c. PyCharm – more advanced, for bigger projects

d. Geany and VS Code – also used by many programmers

If you are using LEGO Education SPIKE, it already has a Python editor built-in!

WHAT IS INSIDE THE SPIKE PRIME HUB?

The SPIKE Prime Hub you have used in class since Year one is like a small computer. It is called a microcontroller. It has limited memory and processing power, which means it cannot run the full version of Python. Instead, it uses a smaller version called Micro Python. Micro Python works just like Python but is designed to run on small devices like the Hub.

For now, you will learn how to use standard Python on a computer. Once you are comfortable with it, moving to Micro Python (for controlling the SPIKE Hub) will be easy and fun.

UNDERSTANDING HOW PYTHON IS WRITTEN (SYNTAX)

To write Python programs, you must first understand syntax. This refers to the rules that guide how Python code should be written.

HOW PYTHON CODE WORKS

1. Each line of Python code usually ends with a line break (you press the Enter key).
2. Indentation is very important in Python. Indentation in Python means adding spaces before a line of code to show which lines belong together. If your instructions belong to the same group (or block), they must all begin with the same amount of space. If the indentation is not correct, the program will give a syntax error.
3. Python files are saved with a .py extension (for example: robot_program.py).

When you run a .py file, your editor sends it to something called the Python interpreter. The interpreter reads the code line by line and performs the instructions written.

Example: Print("Hello world")

This line tells Python to display the word "Hello" on the screen. The word print is a function, and functions are used in Python to perform specific tasks. You will learn more about them later.

MAKING CODE EASIER TO READ – SYNTAX HIGHLIGHTING

Most modern editors or IDEs colour your code using a feature called syntax highlighting. This helps you see the different parts of your program more clearly. Here are some examples but remember that between different IDEs there is a lot of variation:

Element	Appearance in editor	What it means
print or input	Usually, blue	A keyword or function
"text in quotes"	Usually, orange	A string (text to be displayed or stored)
# this is a comment	Grey or green	A comment (notes for the programmer)
()	Black or blue	Used when calling (using) a function

BUILDING PYTHON PROGRAMS

Everything above forms the foundation of writing Python code. Now that you understand these basic rules, you are ready to begin creating real Python programs to control robots and solve problems.

UNDERSTANDING VARIABLES AND HOW DATA IS STORED

In programming, a variable is a name you give to a storage space in the computer's memory. It is similar to variables in mathematics, where a letter like x or y is used to represent a value.

In your earlier experience with LEGO SPIKE™ Prime block programming, you created variables to store values that you wanted to use later in your program. For example, you might have stored the speed of a motor or the number of seconds to wait.



Figure 5.3: Creating variables in block programming

In Python, variables work similarly. You create a variable by choosing a name and then using the equal sign (=) to give it a value. This is called assignment.

speed = 50

message = "Hello, robot"

In the first line:

1. speed is the variable name
2. 50 is the value stored in the variable

In the second line:

3. message is the variable
4. "Hello, robot" is a piece of text (a string) stored in it

Below is an example of how variables are created and used in block and Python code.



Figure 5.4: Variable Usage in block and python programming

In ***Figure 5.4***, the text "Hello world!" is saved in a variable named message in both block and python. It is then written onto the screen of the hub. In python, the variable message is passed into the print() function, which tells the computer to display the value stored in the variable on the screen.

IMPORTANT RULES FOR NAMING VARIABLES IN PYTHON

When working with variables in Python, there are some rules and good practices you must follow. These help you avoid errors and write code that is easy to understand.

1. Use only letters, numbers, and underscores: A variable name can start with a letter or an underscore (_), but not with a number.

   ✅ Correct: message_1

   ❌ Incorrect: 1_message

2. Do not use spaces in variable names

   ✅ Correct: greeting_message

   ❌ Incorrect: greeting message (This will cause an error.)

3. Do not use Python's reserved words: These are words Python already uses for special commands.

   ❌ Avoid names like: print, is, class, return, pass, None

4. Use short but meaningful names: Choose names that clearly describe what the variable holds.

   ✅ Good: student_name, name_length

   ❌ Not helpful: s_n, x, data123

5. Be careful with confusing letters: Letters like lowercase l, uppercase I, and uppercase O can look like the numbers 1 and 0. Avoid using them in a way that may cause confusion.

Why These Rules Matter

Following these rules:

1. Prevent errors in your code
2. Make your program easier to read and fix
3. Help other people (like your teacher or teammates) understand your work

Activity 5.2 Identifying Your Robot Using Variables

You are going to introduce your robot by creating a digital ID card.

Tasks

1. Open your LEGO SPIKE™ Prime (Or any other block programming app you have installed).
2. From what you learnt in year one, create two variables using the variable block and name them and .
3. Assign your robot a unique name like “AccraBot-Alpha” or make up your own.
4. For the purpose, assign a unique purpose. For example: “Cleaning gutters in cities”.
5. Use the Display block to show both values on the hub or console.
6. Replicate this in your Python IDE (IDLE, Pycharm, VS Code)
7. Repeat the same process in Python, take note of the similarities and differences in using blocks and Python.

Discussion

1. What was easier in Blocks? What was more powerful in Python?
2. Share with the class one thing you learned about using Python that you could not do easily in Blocks.

UNDERSTANDING BASIC DATA TYPES IN PYTHON

In Python, every variable stores a certain type of data. The data type tells Python what kind of value is being stored and how it should be used.

Here are the four basic data types you will use most often in robotics simulation:

1. **Integers (int):** These are whole numbers — they do not have decimal points. Examples: 0, 10, -5

Used for:

a. counting things (like number of rotations)
b. setting motor speed levels
c. anything that uses whole or full numbers

2. **Floating-Point Numbers (float):** These are numbers with decimal points. Examples: 3.14, -0.5, 100.0

   Used for:

   a. measuring distance
   b. time duration (e.g., wait for 1.5 seconds)
   c. sensor readings that need more precision

3. **Booleans (bool):** These can only be True or False.

   Used for:

   a. checking if a button is pressed or not
   b. making decisions (e.g., "If the object is detected, stop")
   c. conditions that lead to different actions

4. **Strings (str):** These are pieces of text, written inside quotation marks. Examples: "Hello World", "Motor A"

   Used for:

   a. displaying text on the screen
   b. naming sensors or motors
   c. showing messages or results

Learning how to choose the right data type helps your robot store values correctly and make better decisions during simulations or tasks.

Activity 5.3 Recording Operation Details

Tasks

1. Create a variable called . Store the number of minutes your robot has been working
2. Create a variable called and store the robot's battery level.
3. Create a variable called and set it to either True or False to show if the robot is working properly.
4. Discuss with your partner the appropriate data type for the created variables and why you chose it.
5. Print all this information clearly.

Tip: Use labels so your printout is easy to read.

OPERATIONS IN PYTHON

In Python, you can do maths operations by using common symbols called operators. In year one, you used some common operators in block programs. Figure. 5.5 shows some operator blocks used in block programming, as well as how they are used and the corresponding Python code.



Figure 5.5: Operators in Block Programming and Python

Here are the most used Python math operators:

Operator	What it Does	Example	Meaning
+	Add	5 + 3	Adds two numbers
-	Subtract	10 – 2	Subtracts one number from another
*	Multiply	4 * 2	Multiplies the numbers
/	Divide	10 / 2	Divides and gives a decimal result
//	Integer Divide	10 // 3	Gives only the whole number part
%	Modulo	10 % 3	Gives the remainder
**	Exponent (Power)	2 ** 3	Means 2 to the power of 3 or $2^3(2 \times 2 \times 2)$

Figure 5.6: shows how various data types are used with the various operations. The "//" operator, as shown, is used to integer divide. It will remove all decimals.

```
>> print(5+10)
15
>> print(10/3)
3.333333333333335
>> print(10//3)
3
>> print("ab"+"cd")
abcd
>> print(7+"ab")
TypeError: unsupported operand
type(s) for +: 'int' and 'str'
>> print(7,"ab")
7 ab
```

Figure 5.6: Sample Python operations

1. You can add or multiply numbers, floats, and strings, but you cannot mix different data types in the same operation, as shown when 7 was added to the string "ab" in ***Figure 5.6.***
2. When you divide 10 by 3 in Python, you get a result like 3.333333335. That's because computers use a system called "floating-point approximation" to handle decimals. Unlike humans, computers can't represent recurring decimals (like 3.333...) exactly, so they round the number to the closest value they can store. This rounding sometimes leads to tiny differences, like the unexpected 5 at the end. It doesn't mean the number is wrong—it's just how computers manage numbers that go on forever.
3. If you use the // operator in the same sum, 10 // 3, Python gives 3, which is the whole number part only, with the decimal removed.

In mathematics (algebra), expressions like: are read as "x is equal to 3 plus 2". In Python, the same rule applies, but remember:

1. Python always evaluates the right-hand side first
2. Then it stores the result in the variable on the left-hand side

In the example , Python first does (which gives 12), then saves that result into the variable .

Activity 5.4 Calculating Robot Performance

Your robot uses 0.5% of power every minute.

Task

1. Calculate how much power has been used during the mission. Call it
2. Calculate how many more minutes the robot can run before it runs out of power. Call it .
3. Print these two results with good labels.

Hint

*power_used=mission_duration*0.5*

time_left = current_power_level/0.5

UNDERSTANDING LISTS IN PYTHON

A list in Python is a way of storing multiple values together in one place.

You can think of it like a basket that holds many items, such as numbers, words, or a mix of both.

KEY FEATURES OF LISTS

1. Ordered: The items in a list keep their position (first, second, third…).
2. Changeable (Mutable): You can add, remove, or change items in a list.
3. Flexible: A list can store different types of data – numbers, words, or even both in one list.

HOW LISTS ARE USEFUL IN ROBOTICS

MOVEMENT LOGS

You can use a list to record robot movements as they happen. Previously you did this in block coding by using the list block as shown in Figure 5.7. But in python, we do this by putting the items in a square bracket [] and each individual item separated by a comma (,) as shown in the ***Figure 5.7.***



Figure 5.7: Creating lists in Block and python code

Lists can also be used to store items like sensor input history or robot state information.

Example

light_readings = [50,52,48,55,60]

robot_state = [10,20,"moving forward",75.5

COMMON THINGS YOU CAN DO WITH LISTS

Action	Function Used	Example
Add an item to the end	append()	my_list.append("stop")
Insert item at a position	insert()	my_list.insert(1, "turn")
Remove an item by value	remove()	my_list.remove("left")
Remove an item by position	pop()	my_list.pop(0)
Access an item by index	Use square brackets	my_list[2] gets the third item

For more on lists and how they are used in python programming follow the link below

Link	QR CODE
How to Use Lists in Python	
Python Tutorial for Beginners 4: Lists, Tuples, and Sets	

Activity 5.5 Logging Robot Actions

Tasks

1. Create a list called .
2. Inside the list, store five actions your robot has performed today. Examples:
   a. "Scanned crops"
   b. "Uploaded data"
   c. "Charged battery"
   d. "Avoided obstacle"
   e. "Sent report to HQ"
3. Print all the activities.
4. Print only the first and the last activity using indexing.

Activity 5.6 Creating the Status Report

It is time to combine everything you have learnt so far into a single robot status report.

Task: Create and print a report following these points:

Print:

1. A title:

   === ROBOT SYSTEM DAILY STATUS REPORT ===

2. The robot ID and purpose
3. Mission duration and power level
4. If the robot is operational
5. Power used and time left
6. A full list of activities
7. The first and last activities

Challenge for fast finishers: Try using f-strings like this for cleaner formatting:

print(f"Robot ID: {robot_id}")

Activity 5.7 Debug and Review

Tasks

1. Run your program and check for errors.
2. If you get an error:
   a. Read the message
   b. Check if it is a , , , or
3. Research and find out what these errors are and what causes them. Fix the problem and re-run.
4. Once your program works, help a classmate debug theirs.
5. Reflect on the completed activity

Think: What was the easiest part of today's activity? What was most difficult?

Pair: Talk to a classmate and compare how you designed your robot's log or calculated time left.

Share: Share your working code or best idea with the class.

MAKING DECISIONS AND REPEATING ACTIONS IN PYTHON

In this part of the section, you will learn how to make your robot think and decide what to do based on different conditions. You will also learn how to make your robot repeat actions when needed. This part of the section introduces two powerful tools in Python programming:

1. **Conditional statements** like , *if, if-else and if-elif-else*
2. **Loops** *like while and for*

These are used to:

1. Tell the robot to do something only if a certain condition is true (e.g., "If the battery is low, return to base")
2. Make the robot repeat actions a certain number of times or until something changes (e.g., "Keep moving until the obstacle is gone")

You will also learn about two important commands:

1. *break* – to stop a loop early
2. *continue* – to skip a step and go to the next one

Recap: What You Already Know

Back in Year 1, you used blocks to:

1. Create programs that made decisions
2. Draw loop and logic diagrams
3. Work with simple conditions like "If the sensor detects light, turn right"

Now, you will take that knowledge further by writing the same logic in Python. You will also connect it with what you learned in Boolean Algebra (Year 2, Week 9) to make your robot respond to the environment in more advanced ways.

This new skill will help you create smarter robots that can make decisions, repeat tasks, and respond to sensor inputs, all using Python code.

Activity 5.8 Recap – Control Flow Basics

Before starting this activity, organise yourself into a group of 2 to 3 people.

This task is to link what you already know about control flow concepts from block codes.

Task

Match each term to its purpose in Block programming by completing the following table:

Control Flow Concept	Purpose	Example
if		
else		
else if		
for		
while		
break		
continue		

Hint

For each term, give a real-world robot example.

Example

if – If the robot sees an obstacle, it turns left.

Support: Use your flowcharts from Year One or ask a friend for help. Your teacher may display a sample chart.

CONDITIONAL STATEMENTS

WHAT ARE CONDITIONAL STATEMENTS?

Conditional statements help your program make decisions. They tell your robot what to do when a certain condition is true — just like saying, "If it is raining, stay inside."

In Year One, you used "If blocks" in block programming to do this. Now, in Python, you will learn how to write the same logic using text-based code.

1. **Simple if Statements**

Use a simple *if* when you only want to do something if one condition is true. In block programming, you used the If block to check one condition (e.g., "If sensor = dark"). Same thing can be done as shown in the ***Figure 5.8.***



Figure 5.8: Simple if statements

In this example, if the battery level is less than 20, the robot will print a message. If not, it will do nothing.

2. ***if-else*** Statements

Use when there are two possible outcomes — one if the condition is true, and another if it is false. In block programming, this is like the *If-Else* block, where the robot does something or something else. It functions the same way in Python but is written differently, as shown in ***Figure 5.9.***

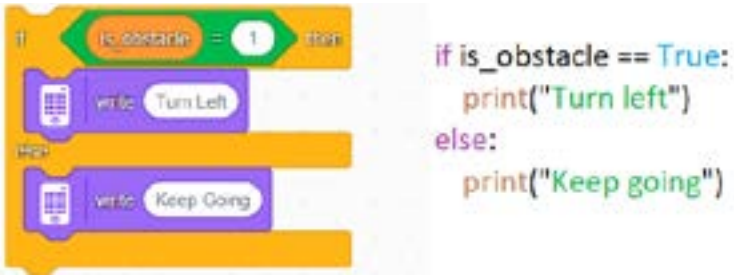


Figure 5.9: if-else Statements

If there is an obstacle, the robot turns. Otherwise, it keeps moving.

3. ***if-elif-else*** Chains

Python has something more advanced called (In other languages it is referred to as if-else if-else). This lets you check more than two conditions one after the other. This does not exist exactly in block programming, but it works like several if-else blocks placed in a row.

```
traffic_light = "yellow"
if traffic_light == "green":
        print("Go")
elif traffic_light == "yellow":
        print("Slow down")
else:
        print("Stop")
```

Here, the robot checks the colour of the traffic light and responds accordingly.

NESTED DECISION CONDITIONS

Sometimes, a program needs to check more than one condition before deciding what to do. In such cases, you can put an *if, elif, or else* or block inside another. This is called a nested decision condition.

This is helpful in robotics. For example, a robot might first check if the battery is enough and then decide whether to move forward or stop based on whether there is an obstacle.

Example: Battery and Obstacle Check

```
battery_level = 60
obstacle_detected = True
if battery_level > 50:
  print("Battery level sufficient.")
  if obstacle_detected:
  print("Stop and reroute.")
else:
  print("Continue moving forward.")
else:
  print("Battery too low! Return to charging station.")
```

What is happening

The robot first checks the battery level.

1. If the battery is more than 50%, it checks if there is an obstacle.
    a. If there is an obstacle → Stop and reroute.
    b. If no obstacle → Move forward.
2. If the battery is low → Return to the charging station.

Tips for Using Nested Conditions

1. Use proper indentation. This helps show which blocks of code belong together.
2. Do not go too deep. If there are too many if statements inside each other, the code becomes hard to read. Try to break the code into functions instead.
3. Use or to combine conditions when possible. This can make the code shorter and clearer.

WHY ARE CONDITIONAL STATEMENTS IMPORTANT IN ROBOTICS?

Conditional statements are useful when your robot needs to:

1. Decide when to stop, turn, or move
2. React to sensor inputs (e.g., light, sound, distance)
3. Change behaviour based on environment or data
4. They help your robot become smarter and more responsive.

For more on if-else statements and how they are used in python, follow the link below:

Link	QR Code
If Else Statements in Python	
Conditionals and Booleans - If, Else, and Elif Statements	

Activity 5.9 Temperature-Based Movement

Tasks

1. Ask the user to enter a temperature (like 20°C).
2. Use if, elif, and else to check the temperature and display what the robot should do:
   a. Above 25 → Move to cooler area
   b. 18 to 25 → Stay in place
   c. Below 18 → Seek warmer area

Challenge Tasks

1. Can you add a 4th category for extreme heat (above 35°C)?
2. Try using and to accept values from the user.

UNDERSTANDING LOOPS IN PYTHON

In programming, loops are used to repeat a block of code. This is helpful when a robot needs to keep doing something — like checking a sensor, moving forward several times, or waiting for an event.

In Year One, you used loops in block programming such as Repeat Until, Wait Until or Forever.



Figure 5.10: Looping constructs in Block programming

Now, you will learn how to do the same things in Python using two main types of loops:

1. *while loops*
2. *for loops*

WHILE LOOPS – REPEAT WHILE A CONDITION IS TRUE

Use a **while** loop when you want your robot to keep repeating an action until something changes — for example, until a sensor detects something.

```
count = 1
while count <= 5:
  print(count)
  count += 1
```

What is Happening?

1. The program starts with count = 1.
2. It prints the number and then adds 1.
3. This continues until count is more than 5.
4. The loop then stops.

while loops are useful when you do not know how many times to repeat — like checking a sensor continuously until a value is detected.

FOR LOOPS – REPEAT FOR A SET NUMBER OF TIMES

Use a **for** loop when you already know how many times you want to repeat something.

```
for number in range(1, 6):
    print(number)
```

What is Happening?

1. This loop runs five times, from 1 to 5
2. The range(1, 6) creates the list: [1, 2, 3, 4, 5].
3. Each number is printed one at a time.

for loops are perfect for repeating a task a known number of times, such as moving forward 5 steps.

LOOP CONTROL STATEMENTS

Sometimes, you may want to change what happens inside a loop. Python gives you special tools for this:

1. **break – Stop the Loop Immediately**

Used when a condition is met and you want to exit the loop early.

```
while True:
  command = input("Enter command: ")
    if command == "stop":
      break
```

The loop continues forever until the user types ".

2. **continue – Skip to the Next Loop Step**

Used when you want to skip the rest of the current loop and move to the next round.

```
for num in range(10):
    if num % 2 == 0:
     continue
       print(num)
```

This will only print odd numbers, because even numbers are skipped.

AVOIDING INFINITE LOOPS

An infinite loop happens when the condition is never false, so the loop runs forever.

❌ **Example of a Problem**

```
counter = 0
    while counter < 5:
     print("Counting:", counter)
        # counter is never increased, so the loop never ends
```

Always make sure your loop has a way to stop, especially when working with robots.

NESTED LOOPS

A nested loop means placing one loop inside another. This is useful when a robot needs to repeat actions inside another repeating task — like scanning every square in a grid or testing different motor speeds.

BASIC STRUCTURE (WHAT IT LOOKS LIKE)

```
for outer in range(3):
    print("Outer loop:", outer)
     for inner in range(2):
        print(" Inner loop:", inner)
```

The outer loop runs first, and inside it, the inner loop runs completely before the next outer loop begins.

Use Cases in Robotics

Scenario	What Nested Loops Do
Scanning a 2D grid	Outer loop = rows, inner loop = columns
Testing multiple sensor-motor combinations	Outer = sensors, inner = motor speeds
Line-following with pattern detection	Outer = lines, inner = pattern on each line

Example 1: 3×3 Grid Scan

```
for row in range(3):
    for col in range(3):
        print(f"Scanning cell at Row {row}, Column {col}")
```

Output:

Scanning cell at Row 0, Column 0

Scanning cell at Row 0, Column 1

...

Scanning cell at Row 2, Column 2

Example 2: Sensor Threshold + Motor Speed Test

```
sensor_thresholds = [10, 20, 30]
motor_speeds = [50, 100]
  for threshold in sensor_thresholds:
    for speed in motor_speeds:
        print(f"Testing with sensor {threshold} and motor speed {speed}")
```

This lets the robot test every combination of sensor and motor setting.

Example 3: Nested While Loop – Wait Until Path is Clear

```
obstacles = [[True, False], [False, False]] # 2x2 grid
row = 0
  while row < len(obstacles):
      col = 0
        while col < len(obstacles[row]):
          if obstacles[row][col]:
            print(f"Obstacle at ({row},{col})! Waiting...")
        else:
       print(f"Path clear at ({row},{col}). Proceed.")
      col += 1
row += 1
```

This program simulates a robot checking every square and waiting if it finds an obstacle.

Tips When Using Nested Loops

1. Keep loops short and simple (no more than 2–3 layers deep).
2. Use clear variable names (like row, col, speed, etc.)
3. Use functions if things get too long or messy.
4. If you use or , remember they only affect the current loop, not both loops.

For more on loops and how they are used in python, follow the link below:

Link	QR Code
For Loops in Python | Python for Beginners	
While Loops in Python | Python for Beginners	

WHY CONDITIONAL STATEMENTS AND LOOPS MATTER IN ROBOTICS

In robotics, the way a robot makes decisions and repeats actions is called control flow. This is what helps the robot behave on its own without being told what to do every second.

Here is how you will use what you are learning:

1. Use if statements to help the robot make decisions based on what its sensors detect.

For example:

"If the robot sees an obstacle, stop"

"If the line is on the left, turn left"

2. Use loops when the robot needs to do something over and over again, like:
   a. Checking sensors continuously
   b. Following a line on the ground
   c. Repeating movements like moving forward five times

You have also learnt how to

1. stop loops at the right time using
2. skip a step using

These skills will help you write robot programs that are smart, clear, and efficient, and will also make it easier to find and fix errors (debugging) when something goes wrong.

Activity 5.10 Repetitive Scanning Routine

Tasks

1. Use a **for** loop to repeat the scan five times.
2. For each scan, print:
   "Performing scan operation... [Scan number]"
3. Use to skip detailed analysis if the scan number is even.

Hint: Use this inside the loop:

```
if scan_number % 2 == 0:
continue
```

Challenge: Add a short between scans to make it more realistic (you will need to import time).

Activity 5.11 Emergency Stop Feature

Tasks

1. Create a loop that keeps asking:
   "Enter command:"
2. If the user types "stop", print:
   "Emergency stop initiated. Robot shutting down."
   And break the loop.
3. For any other command, print:
   "Command received: [command]. Robot continues operation."

Support: Check that your loop does not run forever by using a break at the right time.

Challenge: Count how many commands the robot receives before stopping.

Bonus Challenge (Optional)

Add a final message that says:

```
print("Robot routine completed. Awaiting next mission.")
```

Or try saving your log of commands or scans to a text file using:

```
with open("robot_log.txt", "w") as file:
file.write("Log goes here...")
```

Activity 5.12 Writing a Code for a Scenario – Test and Troubleshoot

Scenario

You are programming a robot to monitor environmental conditions in a lab. The robot must perform a series of temperature scans and respond appropriately based on the readings. However, to conserve energy, it should skip certain scans and be able to shut down instantly if an emergency is detected.

Your Challenge

Write a program that meets the following requirements:

1. Temperature Message
2. After each scan, the robot should display a message based on the temperature reading (e.g., "Temperature is safe" or "Warning: High temperature").
3. Scan Loop
4. The robot must perform 5 scans in total.
5. It should skip scans with even numbers (i.e., scan 2 and scan 4 should be skipped).

6. Use a loop and conditional logic to achieve this.
7. Emergency Stop
8. Include a condition that simulates an emergency stop (e.g., a button press or a specific sensor value).
9. If triggered, the robot should immediately exit the loop and stop scanning.
10. Loop Control
11. Ensure that all loops stop when expected, whether due to completing the scan sequence or an emergency stop.

Tasks

1. Use this checklist to test your full program. Copy the table on to paper or in your notebook, adding the detail in the second and third columns.

Feature	Did it work?	Notes / Comments
Temperature message appears correctly		
5 scans run, even scans are skipped		
Emergency stop ends the loop		
All loops stop when expected		

2. Ask a partner to test your code too. Help each other fix bugs.
3. Show your teacher or group how your robot program works. You can:
    a. Run the code live and talk through it
    b. Record a video of the program with your voice explaining
    c. Write a short paragraph explaining how your code makes decisions and repeats tasks

Tip: Be sure to mention

1. What happens when different temperatures are entered
2. What the robot does in each scan
3. How the emergency stop works.

Reflection: Answer one of these in your discussion group:

1. Which part of the robot logic challenge was easiest for you? Why?
2. Which part challenged you the most? How did you overcome it?
3. How do you think real robots use similar logic?

FUNCTIONS AND THEIR ROLE IN ROBOTICS PROGRAMMING

In this part of Section 5, you will explore a new and powerful idea in programming called functions. A function is a small, self-contained piece of code that does a specific task. You can give it input (called a parameter), and it can send back output (called a return value).

Using functions helps us:

1. Organise code better
2. Avoid repeating the same instructions
3. Solve big problems by breaking them into smaller parts

This approach is known as modular programming, and it is widely used in robotics to make robot programs easier to build, read, fix, and reuse.

WHY FUNCTIONS MATTER IN ROBOTICS

In robotics, we often need to repeat actions—like checking a sensor, moving forward, or analysing data. Instead of writing the same code many times, we can create a function for that action and use it whenever we need it.

HOW TO CREATE AND USE A FUNCTION

We use the keyword to define a function. Then we can call it by name whenever we want to use it. Example:

```
def greet():
  print("Hello, Robot")
# Calling the function
  greet()
```

In robotics, you can create similar functions to control motors, check sensors, or start and stop movement.

Activity 5.13 Create Your Own Functions to Define Robot Actions

In this activity, you will learn how to define and use basic functions in programming and link them to parameters. In programming, a parameter is a variable listed in a function's definition that acts as a placeholder for the value passed into the function when it is called.

Look at this example in Python:

```
def greet(name):
```

print("Hello, " + name)

In this example name is the parameter that allows the function to customise its output based on the input it receives.

Task

Simulate a robot performing three actions using Python functions:

1. Moving forward (distance) (should print out distance in steps the robot moves forward)
2. Turning (angle) (should print out the angle in degrees which the robot is turning)
3. Lifting its arm (height) (should print out the height in centimetres to which the arm is lifted)

Each action will be represented by a function that prints out what the robot is doing.

FUNCTIONS WITH PARAMETERS

You can give information to a function using parameters. These are like input values that help the function decide how to behave. For the first example above the parameter is **steps.**

```
def move_forward (distance):

print(f"The robot is moving forward {distance} steps.")

# Calling the function

move_forward(5)
```

In this example, the number of steps (5) is passed into the function. The robot could use this number to decide how far to move.

Here are some examples of how this is implemented:

You can create a function that accepts multiple parameters and uses them to perform a function.

```
def turn_robot(direction, degrees):

print("Turning", direction, "by", degrees, "degrees")

# Calling the function

turn_robot("left", 90)

turn_robot("right", 45)
```

This function, , takes two parameters, direction (e.g., "left" or "right") and degrees (e.g., how far the robot should turn). By changing the input values, we control the direction and angle the robot turns. It shows how a single function can be reused with different values to control different behaviours.

FUNCTIONS THAT RETURN VALUES

Sometimes, a function needs to send back a result, like a sensor reading or a number.

```
def get_speed():
return 10
speed = get_speed()
print("Current speed:", speed)
```

This lets the robot make decisions based on the value it receives.

Below is another application of functions with multiple parameters and return value

```
def calculate_remaining_time(battery_percent, usage_rate_per_minute):
remaining_minutes = battery_percent / usage_rate_per_minute
return remaining_minutes
# Example use
battery = 60.0 # 60% battery left
usage_rate = 2.0 # robot uses 2% per minute
time_left = calculate_remaining_time(battery, usage_rate)
print("Estimated time left:", time_left, "minutes")
```

This function takes the robot's battery level and usage rate, then calculates how many minutes the robot can keep running. The function does a calculation and returns the result.

The result is stored in a new variable, , which we can then use or print. This kind of function helps robots plan based on changing conditions.

Activity 5.14 Return a Value – Simulate a Sensor Goal:

In this activity, you will learn how to make functions give back information (return values).

Tasks

1. Write a function get_sensor_reading() that returns a random number between 0 and 100 (use import random and random.randint()).

2. Store the returned value in a variable and print a message like: "Light sensor reads: 85"

Challenge Extension: Create a function calculate_remaining_time(battery, usage_rate) that returns how long the robot can keep running.

Print a complete sentence using the result.

Hint: To calculate the remaining time, divide the battery level by the usage rate. You can format the result to two decimal places using:

print(f"Remaining run time: {time:.2f} hours.")

USING DICTIONARIES TO CONTROL ROBOT COMMANDS

A dictionary in Python is like a real dictionary: it connects one thing to another. In robotics, you can link words like "left" or "right" to robot actions.

```
def turn_left():
    print("Turning left")
def turn_right():
    print("Turning right")
commands = {
    "left": turn_left,
    "right": turn_right
}
command = input("Enter command: ")
if command in commands:
    commands[command]()
else:
    print("Invalid command")
```

This creates a flexible command system where typing a word makes the robot do something.

Here is another example of application of dictionaries in advanced situations:

```
def start_cleaning():
    print("Cleaning started... Vacuum motors engaged.")
```

```
def return_to_base():
    print("Returning to charging station.")
def check_status():
    print("Battery level: 75%. Cleaning in progress.")
# Dictionary mapping commands to actions
robot_actions = {
    "start": start_cleaning,
    "home": return_to_base,
    "status": check_status
}
# Simulate user input
user_command = input("Enter command (start/home/status): ").lower()
if user_command in robot_actions:
    robot_actions[user_command]() # Call the matched function
else:
    print("Unknown command. Please try again.")
```

This dictionary connects text commands (like "start") to actual robot functions:

Each command is a key, and the value is a function. If the user types "start", the program calls . If the input is not recognised, it prints an error message.

This is how real robot control panels often work—responding to commands using mapped functions.

Activity 5.15 Command Your Robot with a Dictionary

In this activity, you will create dictionaries to control robot actions based on user commands.

Tasks

1. Write at least 3 robot action functions: start(), stop(), and status().
2. Create a dictionary that links each command to a function.
3. Ask the user to type a command (like "start") and use the dictionary to run the correct function.

Challenge Extension

1. Add a "reset" command that prints "Resetting all systems...".
2. Make sure your program handles unknown commands with an error message.

GETTING INPUT FROM THE USER

Python allows you to ask the user for input using the input() function. This is useful in robotics when setting modes, giving commands, or testing actions.

```
name = input("Enter your robot's name: ")
print("Hello", name)
```

This is like giving your robot a name or setting up instructions before starting.

Here is another example of how to accept user input and use the values accepted. In this situation, the input function is encapsulated in an int method which converts any number entered into an integer. Any number entered by the user is first converted into an integer before it is passed to the speed variable.

```
speed = int(input("Enter speed (1 to 10): "))
direction = input("Enter direction (forward/backward): ")
if 1 <= speed <= 10 and direction in ["forward", "backward"]:
    print("Moving", direction, "at speed level", speed)
else:
    print("Invalid input. Please enter correct speed and direction.")
```

This program:

1. asks the user to enter a speed (as a number) and a direction.
2. checks if the values are valid using an statement. If the inputs are OK, it simulates robot movement. Otherwise, it shows an error.

This is useful in robotics where we control movement settings manually, for example when testing or configuring robot behaviour.

CREATING MENU-BASED ROBOT PROGRAMS

You can use functions and loops to create a menu that lets users pick what they want the robot to do.

```
def start():
    print("Robot started")
def stop():
    print("Robot stopped")
def show_menu():
    print("1. Start\n2. Stop\n3. Exit")
while True:
    show_menu()
    choice = input("Select an option: ")
    if choice == "1":
    start()
    elif choice == "2":
    stop()
    elif choice == "3":
    break
    else:
     print("Invalid choice")
```

This type of program is useful when you want to let the user choose what happens next—like starting the robot, stopping it, or quitting the program.

For more on functions and how they are used in Python, follow the link

Link	QR Code
Functions	

Activity 5.16 Build a Robot Menu

In this activity, you will learn how to build a menu system using while loops and if statements.

Tasks

1. Create a function called show_menu() that prints:
   a. Start Patrol
   b. Recharge
   c. Shutdown
2. Use a loop to keep showing the menu until the user types 3.
3. Run the correct function depending on the user's choice.

Challenge Extension: Add more menu options, like:

"d. Check Temperature"

"e. Calibrate Sensors"

UNDERSTANDING PYTHON ERRORS AND ERROR HANDLING

When writing Python programs, it is normal to make mistakes. These mistakes are called errors, and they can stop the program from working properly. Learning how to identify and fix these errors is an important skill for every programmer.

COMMON TYPES OF ERRORS IN PYTHON

1. **Syntax Errors**

These occur when the rules of Python are not followed. For example, forgetting to add a colon or using the wrong punctuation.

```
if number == 10 # Missing colon
    print("Number is 10")
```

Fix: Add the missing colon at the end of the if line.

2. **Name Errors**

These occur when you try to use a variable or function that has not been defined.

```
print(result) # result has not been defined yet
```

3. **Type Errors**

These occur when you try to combine different types of data that do not match.

```
print("Age: " + 15) # Cannot combine a string with a number
```

Fix: Convert the number to a string using str(15).

4. **Value Errors**

These occur when the data type is correct, but the value is not suitable.

```
int("hello") # You cannot change a word into a number
```

5. **Zero Division Error**

These occur when you try to divide a number by zero.

```
print(10 / 0) # Division by zero is not allowed
```

HANDLING ERRORS USING TRY AND EXCEPT

Instead of letting the program crash when there is a problem, you can handle the error using a try and except block. This allows your program to keep running and show helpful messages.

Example:

```
try:
  number = int(input("Enter a number: "))
  result = 10 / number
    print("Result is:", result)
  except ZeroDivisionError:
    print("You cannot divide by zero.")
  except ValueError:
    print("Please enter a valid number.")
```

This program checks for two kinds of errors. If the user enters 0, it shows a helpful message. If the user enters text instead of a number, it also shows a clear message.

Tips for Error Handling

1. Test your code using different types of inputs.
2. Use print() statements to check where something might be going wrong.
3. Read the error message — it tells you the type of error and where it happened.
4. Use and blocks to catch problems and help the program run smoothly.

To learn more about Python errors and error handling, explore these beginner-friendly video tutorials:

Link	QR Code
Python Tutorial: Using Try/Except Blocks for Error Handling	
Python Exception Handling Tutorial for Beginners	

Activity 5.17 Spot, Catch, and Fix! – Handling Python Errors

Write a Python program that simulates asking a user for a number and performs a simple calculation. Your job is to catch any errors that may happen and print helpful messages instead of crashing the program.

Tasks:

1. Start a new Python file.
2. Ask the user to enter a number with this code:

   number = input("Enter a number: ")
3. Try to divide 100 by the number and print the result.
4. Use a and block to catch errors like:
   i. The user entering text instead of a number
   ii. The user entering 0 (which causes a division by zero error)

Challenge Extension: Input + Sensor Simulation

1. Use your get_sensor_reading() function from Activity 5.16.
2. Add a try block that:
   i. Gets a number from the user.
   ii. Adds it to the sensor reading.
   iii. Prints the result.
3. Use to catch invalid inputs (e.g., if the user types "ten" instead of 10).

Discuss your program with the class and explain

1. What would happen if you did not use and ?
2. What kind of error messages did you see before adding error handling?

3. Why is it important to plan for errors in real robot programs?

Bonus Debug Tip

Add a print() inside the except block that explains what went wrong, like this:

except ValueError as e:

print("There was a problem:", e)

This helps you understand the error better when testing your code!

WHY GOOD CODE STRUCTURE MATTERS IN ROBOTICS

When you organise your code using functions, dictionaries, and menus with user input, your robot becomes easier to control and manage.

Here is what structured code helps you do:

1. **Break Your Program into Clear Parts**

Each part of the robot's task (e.g., start, move, stop) is placed in its own function. This makes it easier to understand what the code does.

2. **Make Changes Easily**

Need to change how the robot turns? Just update one function — no need to change many parts of the code.

3. **Find and Fix Problems Faster**

When errors happen, it's easier to find the cause if your code is neatly separated into sections.

4. **Reuse Code for Other Tasks**

You can use the same function again in a new program — like copying and pasting tools that already work!

In Real Life, robots in factories or smart devices use these same ideas to stay reliable, easy to upgrade, and ready for complex tasks. Learning these skills now will prepare you for real-world robot programming challenges in the future!

Activity 5.18 Prime Scanner Bot – Full Project Integration Challenge

Objective: Design and build a complete Python-based robot control simulation that incorporates structured programming techniques and decision-making logic. This challenge will test your ability to combine functions, user input, conditionals,

loops, and program structure — using the engineering process you have practised in earlier years.

Your Challenge Task

You are to design, plan, code, and test a simulated robot that can check whether a number is a prime number and respond accordingly.

Step 1: Planning the Program

1. Write a Pseudocode Algorithm that outlines the steps your robot will take:
    a. Prompt the user for a number.
    b. Check if the number is prime.
    c. Respond based on the result.
    d. Repeat until the user decides to exit.
2. Draw a Flowchart showing the logic and decision paths: Include loops, condition checks, and function calls.

Step 2: Translate your algorithm and flowchart into working Python code. Your program must include:

1. A Menu System using a loop and input():

    Options:

    a. Check if number is prime
    b. Get sensor reading (simulated)
    c. Robot status
    d. Exit
2. Function Definitions:
    a. is_prime(n) – Returns True or False.
    b. get_sensor_reading() – Simulates returning a sensor value (e.g., temperature).
    c. status() – Prints the current robot status (e.g., "Battery at 75%, no errors detected").
3. A Dictionary to Map Menu Options (Optional Extension): Use a dictionary to map numeric choices to corresponding functions.
4. Conditional Logic:
    a. Within your is_prime function, use a loop and conditionals to determine if the number is prime.
    b. Provide meaningful feedback based on the result:
        i. "Prime number detected. Send to Analysis Unit."
        ii. "Not prime. Assign to Regular Operations."
5. Error Handling:

a. Gracefully handle invalid inputs (non-numeric or out-of-range values).
b. Show helpful messages when users select an invalid option.

Step 3: Testing and Troubleshooting

1. Run your program with different inputs.
2. Observe the output and debug any issues (e.g., logic errors, input errors, menu not looping).
3. Refactor your code for clarity and reusability.

Note

What You Should Submit for Formal Assessment

1. Your final **Python script (.py or .txt)**.
2. A **brief write-up or scanned image** of your:
   a. Pseudocode
   b. Flowchart
3. A short reflection (3–5 sentences) explaining:
   a. What worked well
   b. What was most challenging
   c. How you solved any problems you encountered

Extra Challenge (Optional)

Enhance your robot's logic:

1. Add a timer delay after each action.
2. Add a function prime_summary(start, end) that prints all primes between two numbers.
3. Create robot "modes" (e.g., PrimeCheckMode, SensorCheckMode) and switch between them.

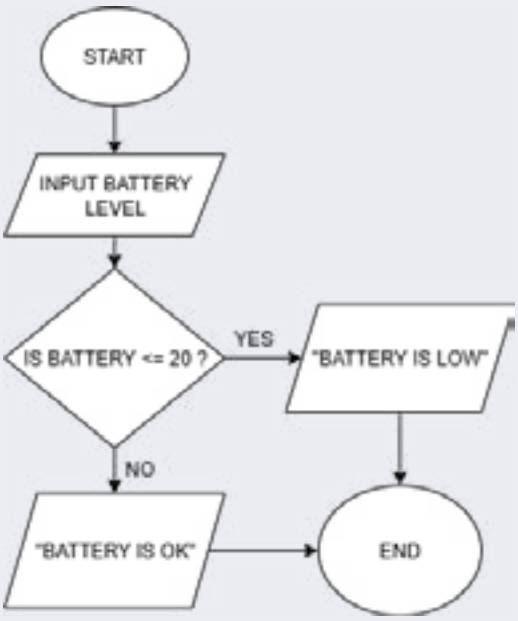
EXTENDED READING

Link	QR Code
Easy Python Coding for Robots: Getting Started	

Easy Python Coding for Robots: Getting Started	
Python Tutorial at w3schools	
Python 3 Beginner Course at CodeAcademy	

REVIEW QUESTIONS

1. Create a dictionary that maps the commands “start”, “stop”, and “report” to corresponding functions for a robot. Write a Python program that prompts the user for a command and uses the dictionary to run the correct function.
2. Explain the role of return values in robotics programming. Give an example where returning a value is more useful than just printing a message.
3. Describe the difference between syntax errors and runtime errors in Python and give one example of each from your robotics projects.
4. Convert the following flowchart into Python code:



5. Challenge Task: Write a complete Python program that simulates a robot with the following capabilities:
   a. Ask the user for a command using input()
   b. Use a dictionary to map the command to a function (e.g., move(), turn(), shutdown())
   c. Each function must include a parameter (e.g., direction or distance)
   d. Validate user input and handle errors (e.g., invalid command or type error)
6. Design a menu-based program that lets the user:
   a. Select between robot modes (e.g., Patrol, Standby, Recharge)
   b. Use a while loop to keep the menu running until “Exit” is chosen
   c. Implement each menu item as a separate function
   d. Handle possible input errors using try-except

REVIEW QUESTIONS

7. Debugging Exercise: The code below is intended to move the robot and calculate remaining battery life, but it contains errors. Identify and fix all errors:

```
def move(distance):
  print("Moving", distance, "units")
def get_battery_use():
  usage = input("Enter usage rate: ")
  return 100 / usage
move(10)
  print("Remaining time:", get_battery_use)
```

8. Scenario-Based Problem: Imagine you are programming a robot to monitor room temperature and adjust actions:

   If temperature > 30: cool down

   If temperature < 18: warm up

   Otherwise: maintain state

   Write a function adjust_temperature(temp) to handle this and include exception handling in case a non-numeric input is provided.

9. Algorithm Translation Task: You are given this pseudocode:

```
START
INPUT: robot name
DISPLAY: Welcome robot_name
IF robot_name is "Zeta"
 DISPLAY: Initiating advanced mode
ELSE
 DISPLAY: Basic mode active
END
```

   Write a Python function that implements this algorithm. Add input validation to handle empty inputs.

10. Reflection Question: Why is modular programming (using functions) important in robotics? How does it improve code maintenance, error detection, and scaling for larger robot projects?

ANSWERS TO REVIEW QUESTIONS

Section 1

1. The pseudocode should look like this, no other order.

START

Scan area for plastic waste

Emit auditory and tactile signals to indicate presence

Move to waste location

Collect waste

Continue scanning

END

This step improves inclusivity by notifying visually impaired individuals when the robot is operating nearby, reducing risk, and enhancing accessibility.

2. A Decision symbol should be used to check whether pollution levels exceed the predefined threshold.

If the condition is true, the flow proceeds to an Input/Output symbol to send an alert.

If false, the robot continues monitoring using a feedback loop—represented by a Connector or another Decision symbol returning to the pollution check step.

This loop allows the robot to maintain ongoing surveillance when pollution is low, reducing unnecessary alerts and improving energy efficiency. Over time, data from repeated monitoring could also inform threshold adjustments, enhancing the system's adaptability to environmental changes.

3. Sample response

Using pseudocode is often better than a flowchart when outlining a robot's task for delivering medical supplies because:

a. It is faster to write and edit, especially when tasks are complex.

b. It allows for more detailed instructions, like how the robot handles errors or restarts.

c. It is easier to translate into actual programming code later on.

Flowcharts are good for visualizing logic, but pseudocode provides better step-by-step clarity, especially in real-life conditions like remote delivery routes or low-connectivity areas in rural Ghana.

4. Sample response

Using standard flowchart symbols is important because:

a. It helps others understand your design easily, especially in teamwork.

b. It avoids confusion — everyone recognizes what a decision, process, or input/output means.

c. In Ghanaian contexts like plastic waste cleaning, standard symbols help when different teams (engineers, students, officials) review or test the robot.

d. It also ensures that the logic flows properly, which is necessary for safety and efficiency in public areas like markets.

5. Sample response

a.

- The use of a WHILE loop to maintain operation while the conveyor belt is moving reflects continuous processing—an appropriate control structure.
- The initial check for fish presence avoids unnecessary classification attempts when the belt is empty.
- The general attempt to branch logic based on fish size (even if flawed) shows appropriate conditional design.

b. The line IF fish is large → place fish in medium bin is incorrect.

Consequences

- This misclassification would corrupt data logs on catch sizes, affecting inventory, pricing, and sustainability records.
- It may overwhelm the medium bin, cause overflow errors, or confuse workers handling sorted fish.
- Over time, these misclassifications could damage trust in automated sorting and increase manual override frequency.

c.

- A sensor-based feedback loop could monitor bin contents and compare real-time classification against stored fish dimension parameters.
- If patterns of misclassification emerge (e.g. large fish frequently placed in medium bin), the system could:
  - Flag the issue,
  - Retrain its classification threshold,
  - Or alert operators for intervention.

- This adaptive behaviour improves reliability and supports machine learning integration, enabling continuous improvement from field performance.

d. Short Report: Critique and Improvement Plan

Summary of Task: The robot sorts fish on a conveyor belt by size, using a camera sensor and gripper arm, while handling unidentifiable cases and ensuring continuous operation.

Strengths

- Logical control flow with WHILE loop and conditionals.
- Basic error handling exists for unknown fish.
- Design anticipates real-world variability in inputs.

Weaknesses

- Misclassification of large fish into medium bin undermines sorting accuracy.
- No feedback mechanism to detect or correct performance issues.
- Lack of data logging or validation against stored classification standards.

Recommendations

- Correct logical errors: Replace IF fish is large → place in medium bin with the correct bin assignment.
- Add feedback loop:
  - Include real-time size validation and bin capacity monitoring.
  - Example pseudocode snippet:

```
IF size detected THEN
    Compare to stored size parameters
    Confirm bin assignment
    IF mismatch detected THEN
    Log error AND reclassify
END IF
```

- Introduce data logging: Track frequency of unidentified fish and sorting outcomes to improve future calibration.

Justification

These changes promote adaptive intelligence, improve sorting accuracy, and support ethical automation in a market setting. They elevate the

robot's role from static sorter to dynamic decision-maker, aligning its logic with principles of inclusive and sustainable engineering.

Section 2

1. The following is an example of an expected answer:

Technical Report: Troubleshooting a Line-Following Robot

a. Test Plan Development

Objectives: To verify the robot's ability to follow a straight line and accurately navigate a 90-degree left turn.

Test Environment: A classroom floor with good overhead lighting. The surface is white, and the line is a 2 cm thick black electrical tape. There are no obstacles.

Test Cases & Expected Outcomes:

Case 1 (Start): Place the robot on the starting line, facing forward. Expected Outcome: Robot moves forward smoothly, following the line.

Case 2 (Straight Line): Observe the robot as it travels down the first straight section. Expected Outcome: Robot follows the line without drifting more than 1 cm to either side.

Case 3 (90-Degree Turn): Observe the robot as it reaches and attempts the left turn. Expected Outcome: The robot should turn cleanly and continue following the line on the new path.

Case 4 (Final Stretch): Observe the robot on the final straight section to the finish. Expected Outcome: The robot should continue following the line and stop at the end.

Observation Methods: Visual inspection to check for smooth movement and physical drift. Timing the turn to see if it is quick and responsive.

b. Flaw Identification and Root Cause Analysis

Flaw 1: The robot drifts noticeably from side to side while following the straight line.

Root Cause Hypothesis: Algorithm Flaw.

The robot's control logic is likely a simple on/off or "bang-bang" controller. This means it either turns hard left or hard right, without smoothly adjusting its motors. The robot is constantly over-correcting, causing the back-and-forth drift.

Justification

The robot is not following the line precisely, but it is not completely losing the line either. The sensor is working, and the motors are responding. This points to the logic that translates sensor input into motor output being too basic and not nuanced enough for smooth control.

Flaw 2: When the robot encounters the 90-degree turn, it often overshoots the corner and loses the line before eventually finding it again.

Root Cause Hypothesis: Coding Error.

The command to turn left is likely a fixed duration. For example, "turn left for 500 milliseconds."

Justification

If the turn is a fixed-time command, the robot will always turn for the same amount of time, regardless of its speed or how far it needs to turn. A faster robot would overshoot more than a slower one. A more robust algorithm would use sensor input to determine when the turn is complete, but this behaviour suggests a simple, hard-coded command.

c. Correction and Iteration

- Selected Flaw for Correction: Flaw 1 (Drifting on the straight line).

  Change Implemented: I implemented a simple Proportional (P) controller to replace the simple on/off logic. The code now calculates the error (how far the sensor is from the line's centre) and adjusts the motor speeds proportionally. A small error leads to a small speed adjustment, and a large error leads to a larger adjustment.

  Re-Test Results: The robot now follows the straight line with significantly less drifting. The back-and-forth motion is almost entirely eliminated, and it follows the line much more smoothly. I observed no new issues. The robot still overshoots the 90-degree turn, but the straight-line performance is much improved.

- Reflection: The fix was successful in addressing the drifting problem. This confirms that the original control logic was the source of the flaw. To fix the overshoot problem at the turn, I would need to implement a more advanced control strategy for turns, such as a different algorithm or using another sensor to better detect the corner's end.

Note

This is just one example of a high-quality response. Other candidates may choose a different robot, task, or flaws to address, but the quality of the answer would be judged by its adherence to the rubric's requirements for detail, accuracy, and justification.

2. The following is an example of an expected answer:

a. Sensor: Soil moisture sensor.

Justification: This sensor is the "sense" of the system. It is specifically designed to measure the wetness of the soil, which is the crucial variable that triggers the system's autonomous action.

Actuator: Water pump.

Justification: This actuator is the "muscle" of the system. Its job is to physically perform the action of releasing water, converting the electrical signal from the controller into mechanical work to move water to the plants.

b. Sensor: Soil moisture sensor.

Sensed State: The current moisture level of the soil.

Desired State: The moisture level of the soil is above a predetermined threshold, indicating the soil is wet enough.

General Action: The system constantly senses the soil's moisture level. If the sensed moisture level is below the desired threshold, the actuator (the water pump) turns on to release water. Once the moisture level rises above the threshold, the pump turns off. This continuous cycle maintains the desired state.

c. Pseudocode outlining the main control logic

```
START
LOOP FOREVER:
READ soil_moisture_level from sensor
IF soil_moisture_level < dry_threshold THEN
ACTIVATE water_pump
ELSE
DEACTIVATE water_pump
END IF
END LOOP
END
```

3. The following is an example of an expected answer:

a. Variable: The material type of the bottle (e.g., plastic versus glass).

Justification: Accurately identifying the material of the bottle is the most critical step for sorting. Without this information, the system cannot make

a decision on where to place the item, which makes the entire process inefficient and leads to waste being incorrectly categorised.

b. Sensor Type: An ultrasonic sensor.

Explanation: An ultrasonic sensor can be used to measure the distance to the bottle as it passes. Since plastic and glass bottles have different densities and shapes, the sensor's readings would vary, providing a way to differentiate between the materials without needing to touch them. A different sensor, like a colour sensor, could also work, but an ultrasonic sensor provides a direct measure of the object's physical characteristics.

c. The system would operate in a continuous loop. The ultrasonic sensor would first measure the distance to the bottle to determine its material. This sensed distance would be compared to a predefined setpoint (e.g., a specific distance range for plastic). If the measurement is within this setpoint, the controller would send a signal to an actuator (e.g., a motor controlling a diverting flap or arm). This actuator would then push the plastic bottle into the correct collection bin. This process would then repeat for the next bottle, allowing the system to sort without human guidance autonomously and continuously.

Section 3

1. Ball valve – It uses a rotating ball with a hole through it to start or stop flow.
2. Force = Pressure × Area
3. Electromagnetic contactors are commonly used in water pumps, streetlights, and machines in factories.
4. One from: high speed switching, silent operation, lasts longer, vibration resistance.
5. Both systems use signals to control actions. In robotics, signals tell the arm to move in specific ways like "grab" or "lift." In industrial systems, signals tell valves to open, close, or adjust flow. This allows precise control—whether it is moving a motor or regulating fluid—based on input commands.
6. In a robotic arm, energy is transferred through electricity to create small, precise movements. In a hydraulic cylinder, energy is transferred through pressurised liquid, which creates powerful movements. This difference is important because hydraulic systems are better suited for heavy-duty tasks like lifting cars or operating construction equipment, while robotic arms are ideal for delicate or precise actions.
7. Electromagnetic contactors are safer because they allow high-power machines to be controlled using a small, low-voltage signal. This means the operator does not directly interact with the high-power circuit, reducing the risk of

electrical shock or damage. In contrast, manual contactors require a person to physically switch the device on or off, which can be unsafe in high-power systems.

8. Contactors are preferable to relays in industrial machinery because they are designed to handle much higher electrical loads. Industrial machines like grinders, conveyor belts, and pumps require large amounts of power, which relays are not built to manage safely or reliably. Contactors use strong electromagnetic coils and heavy-duty contacts that can switch high-power circuits without overheating or wearing out quickly. This makes them safer and more durable for controlling heavy machinery, whereas relays are better suited for smaller tasks with lower power requirements, such as in robotics or home appliances.

9. Sample answer

Actuators are devices used to control systems by converting a signal into mechanical or electrical movement.

- Relay: A relay is an electrically operated switch typically used for low to moderate power applications. It is suitable for controlling circuits but is not ideal for high-power industrial motors due to its limited current-handling capacity.
- Valve: Valves control the flow of liquids or gases in hydraulic or pneumatic systems. While essential in fluid power systems, they are not used for switching electrical motors.
- Contactor: A contactor is a heavy-duty switch designed to handle large currents and high voltages. It is specifically built to switch high-power electrical devices like industrial motors ON and OFF safely.
- Solenoid: A solenoid converts electrical energy into linear motion and is often used to activate mechanical systems. Though useful in automation, it lacks the current-handling capability needed for directly switching high-power motors.

Recommendation and Justification: The contactor is the most suitable actuator for switching a high-power industrial motor ON and OFF. It is designed for high-load applications, offers safety mechanisms, and can be remotely controlled, making it the preferred choice in factories, industrial plants, and large machinery systems.

Real-world example: In a manufacturing plant, large conveyor belt motors are often controlled using contactors. These allow operators to manage the motors safely from control panels, ensuring both safety and efficiency during operations.

10. Sample answer

High-voltage industrial pumps require actuators that can handle high electrical loads safely, reliably, and efficiently. Each actuator listed serves different functions, and their suitability varies based on application.

- Contactor: This is the most suitable option. It is specifically designed for switching high-power electrical devices. Contactors can handle large currents and voltages, and are equipped with arc suppression features, making them ideal for controlling high-voltage pumps.
- Relay: While relays perform a similar switching function, they are generally rated for lower current and voltage levels. They may be used in control circuits but are not reliable for directly switching high-power industrial loads.
- Valve: A valve is used to control the flow of fluids or gases and does not interact with electrical circuits. It is irrelevant in this context, though useful in fluid control systems.
- Hydraulic Cylinder: This actuator produces linear motion using pressurised fluid. It is designed for mechanical movement and not for electrical switching, making it the least relevant in this scenario.

Recommended Ranking (Most to Least Suitable):

Contactor > Relay > Valve > Hydraulic Cylinder

Justification with Example:

In water treatment plants, industrial pumps are commonly turned ON and OFF using contactors due to their ability to safely manage high currents. Using a relay in such a case might lead to overheating or failure, while valves and hydraulic cylinders would not perform the electrical switching required.

11. Sample answer

Hydraulic cylinders are not precise, may leak, and are unsuitable for clean environments. A servo motor would offer better accuracy and cleanliness.

12. Sample answer

Recommended actuators: Valve (for flow control) and Contactor (for motor switching). Valves control fluid movement using mechanical or electrical signals. Contactors handle large electric currents safely and are commonly used in pumping stations.

13. Sample answer

A contactor will be recommended because it is better suited to high-power switching in dusty industrial settings. It is more durable, safer, and easier to maintain compared to a relay.

14. Sample answer

Actuator	Industrial Task	Reason (1 sentence)
Valve	D. Controlling water in a treatment plant	Regulates fluid flow accurately
Relay	A. Switching on a classroom light	Suitable for low-power switching
Contactor	B. Starting a heavy grinder	Handles high current loads
Hydraulic Cylinder	C. Lifting metal beams in a garage	Provides strong linear motion for lifting

Section 4

The following is an example of a candidate's answer:

1. Truth Table

Comment: The first step is to systematically list all 16 (24) input combinations. The 'A' column is then populated based on the given rules. The most important part is applying the override rule: 'A' is always 0 when S=0. This means the first 8 rows of the truth table will all have a 0 for the output, regardless of the values of L, H, and W. The other rules are only applied when S=1 (the last 8 rows).

S	L	H	W	A	Comments
0	0	0	0	0	Override rule: S=0, so A=0.
0	0	0	1	0	Override rule: S=0, so A=0.
0	0	1	0	0	Override rule: S=0, so A=0.
0	0	1	1	0	Override rule: S=0, so A=0.
0	1	0	0	0	Override rule: S=0, so A=0.
0	1	0	1	0	Override rule: S=0, so A=0.
0	1	1	0	0	Override rule: S=0, so A=0.
0	1	1	1	0	Override rule: S=0, so A=0.
1	0	0	0	0	
1	0	0	1	1	Rule 1: S=1 AND W=1.

1	0	1	0	0	
1	0	1	1	1	Rule 1: S=1 AND W=1.
1	1	0	0	0	
1	1	0	1	1	Rule 1: S=1 AND W=1.
1	1	1	0	1	Rule 2: L=1 AND H=1.
1	1	1	1	1	Rules 1 and 2 are both true.

2. Boolean Derivation & Simplification

Comment: The un-simplified expression is derived directly from the rows where A is 1 (Highlighted rows). There are 5 such rows. A learner's answer should show each product term clearly.

Un-simplified SOP Expression

OR

Note

The terms can be written in a different order, which is also correct, as long as the terms are the same.

Karnaugh Map for A

Comment: The K-map must be a 4-variable map, and the 1s must be placed in the correct cells according to the truth table. The grouping is critical. The two groups are of size 4 and 1. The most optimal solution must use the largest groups possible.

Simplified Boolean Expression

- Group 1 (Red rectangle): The 1s at 1001, 1011, 1101, and 1111 form a group of four. In this group, S is always 1 and W is always 1. L and H both change and are therefore eliminated. This gives us the term SW.
- Group 2 (Blue rectangle): The 1 at 1110 must be covered. It has no neighbouring ungrouped 1 as such it stands alone. This gives us the term SLHW'.

Therefore, the simplified expression is: A = SW + SLHW' or

SL \ HW	00	01	11	10
00				
01				
11		1	1	1
10		1	1	

Figure A: K-map for 'A'

3. Logic Gate Diagram

Comment: The diagram must accurately represent the simplified expression A = SW + SLHW'. This expression requires two AND gates, one NOT gate and one OR gate. The inputs and outputs should be clearly labelled.

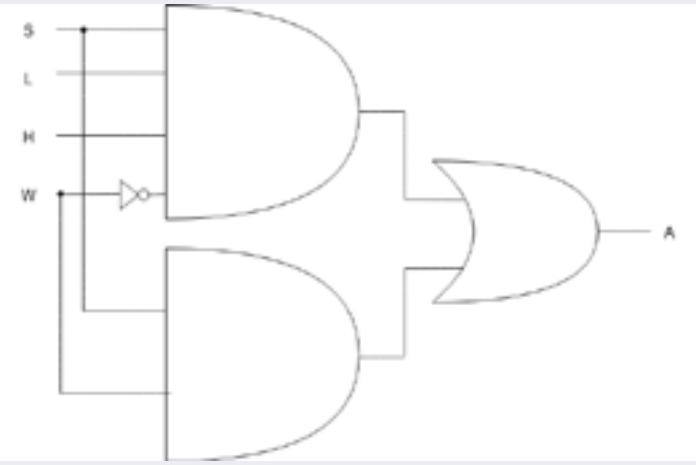


Figure B: Logic Gate Diagram of Simplified Expression of 'A'

4. Implementation and Testing

Comment: The screenshot or photograph should clearly show the circuit built in a simulation software or on a breadboard. The test plan must cover the core logic of the rules, especially the override. The documentation should be clear and show that the circuit works as expected. A learner who has issues should document their troubleshooting steps, for example, "Initial test for S=1, L=1, H=1, W=0 gave A=0, but the test plan expected A=1. I checked my wiring and found the wire from the H input was disconnected from the AND gate. Reconnecting it solved the issue."

Implementation

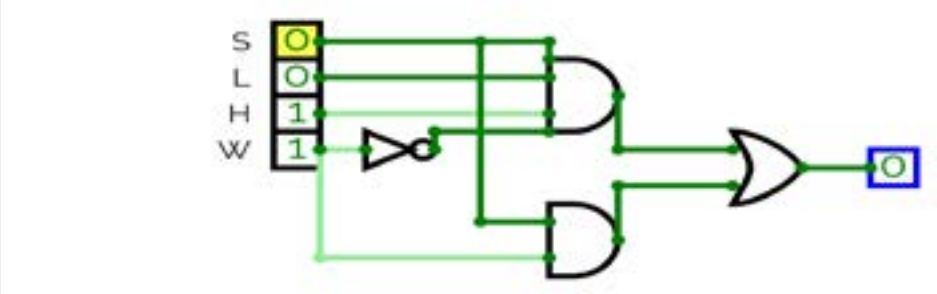


Figure C: Simulation of Simplified Expression of 'A' (Simulated using one CircuitVerse - Online Digital Logic Circuit Simulator)

Test Plan

- Expected Behaviour: The circuit should correctly implement the simplified Boolean expression A = SW + SLHW'.
- Test Cases:

Test Case	Inputs (S, L, H, W)	Expected Output (A)	Observed Output (A)
1	0000 (Override)	0	0
2	0111 (Override)	0	0
3	1001 (Rule 1)	1	1
4	1110 (Rule 2)	1	1
5	1111 (Both rules)	1	1

Conclusion: The circuit performed as expected. All test cases matched the expected output, confirming that the implementation is correct and the logic is sound.

Section 5

1. Python program that prompts the user for a command and uses the dictionary to run the correct function.

```
def start():
print("Robot is starting...")
def stop():
print("Robot has stopped.")
def report():
```

```
print("Robot status: Battery at 80%, temperature normal.")
commands = {
"start": start,
"stop": stop,
"report": report
}
user_input = input("Enter a command (start/stop/report): ").lower()
if user_input in commands:
commands[user_input]()
else:
print("Invalid command.")
```

2. Return values allow a function to send back data that can be used later. For example, a function that reads a temperature sensor should return the temperature so the robot can decide what to do. Example:

```
def read_temperature():
return 28
temp = read_temperature()
if temp > 25:
print("Activating cooling system.")
```

In this case, the returned value helps the robot make a decision.

3. Difference between syntax errors and runtime errors in Python and an example each

Syntax Error	Runtime Error
Mistake in the structure of the code, detected before the program runs. ***Example***: print("Hello" → missing closing parenthesis	Runtime error: Occurs when the program runs but encounters a problem. ***Example***: result = 10 / 0 → causes a ZeroDivisionError

4. Pseudocode Summary:

Start →Input battery level →If battery < =20 →print "Low battery!" →else print "Battery OK" →End

```
battery = int(input("Enter battery level: "))
if battery <=20:
```

```
print("Low battery!")
else:
print("Battery OK")
```

5. Complete robot simulation using input, dictionary, and error handling:

```
def move_forward(distance):
print(f"Moving forward {distance} units")
def turn(direction):
print(f"Turning {direction}")
def shutdown():
print("Shutting down...")
commands = {
"move": move_forward,
"turn": turn,
"shutdown": shutdown
}
try:
cmd = input("Enter command (move/turn/shutdown): ").lower()
if cmd in ["move", "turn"]:
value = input("Enter value (distance or direction): ")
commands[cmd](value)
elif cmd == "shutdown":
commands[cmd]()
else:
print("Invalid command.")
except Exception as e:
print("Error occurred:", e)
```

6. Menu-based program using functions and loop:

```
def patrol():
print("Robot is patrolling...")
def recharge():
print("Robot is recharging...")
```

```
def show_menu():
print("\n1. Patrol Mode\n2. Recharge\n3. Exit")
while True:
show_menu()
choice = input("Choose an option: ")
if choice == "1":
patrol()
elif choice == "2":
recharge()
elif choice == "3":
print("Exiting program.")
break
else:
print("Invalid choice. Try again.")
```

7. Debug the faulty code:

Original (Faulty):

```
def move(distance):
print("Moving", distance, "units")
def get_battery_use():
usage = input("Enter usage rate: ")
return 100 / usage
move(10)
print("Remaining time:", get_battery_use)
```

Corrected:

```
def move(distance):
print("Moving", distance, "units")
def get_battery_use():
usage = float(input("Enter usage rate: "))
```

```
return 100 / usage
move(10)
print("Remaining time:", get_battery_use())
```

8. Temperature adjustment with error handling:

```
def adjust_temperature(temp):
if temp > 30:
print("Cooling down...")
elif temp < 18:
print("Warming up...")
else:
print("Maintaining temperature.")
try:
temperature = float(input("Enter room temperature: "))
adjust_temperature(temperature)
except ValueError:
print("Please enter a valid number.")
```

9. Translate pseudocode to Python with input validation:

```
def greet_robot():
name = input("Enter your robot's name: ")
if name.strip() == "":
print("Name cannot be empty.")
return
print("Welcome", name)
if name.lower() == "zeta":
print("Initiating advanced mode.")
else:
print("Basic mode active.")
greet_robot()
```

10. Modular programming using functions helps break down complex robot tasks into smaller parts. This improves readability, reduces repetition, and makes it easier to test or fix specific parts of a robot's code. For example, separate functions for moving, scanning, or checking battery allow you to reuse code and update only one section if the robot's behaviour changes. It also allows teams to work on different parts at the same time in larger projects.

REFERENCES

1. Bolton, W. (2021). Mechatronics: Electronic control systems in mechanical and electrical engineering (7th ed.). Pearson Education.
2. Craig, J. J. (2018). Introduction to robotics: Mechanics and control (4th ed.). Pearson.
3. Crane, D. (2022, December 15). How to program a robot with Python. ActiveState Blog. Retrieved from https://www.activestate.com/blog/how-to-program-a-robot-with-python/
4. Cylinders, Inc. (2024, August 14). What is a hydraulic cylinder? How it works, types, uses. Enerpac Blog. Retrieved from https://blog.enerpac.com/types-of-hydraulic-cylinders/
5. Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2014). Feedback control of dynamic systems (7th ed.). Pearson.
6. Gray, D. (2022). Python exception handling tutorial for beginners [Video]. YouTube. https://www.youtube.com/watch?v=PHzm_Iox1mE
7. Hughes, A., & Drury, B. (2019). Electric motors and drives: Fundamentals, types, and applications (5th ed.). Newnes.
8. Kev's Robots. (2024). Error handling and exceptions in Python. Kev's Robots Learning Platform. https://www.kevsrobots.com/learn/python/06_errors.html
9. Kev's Robots. (2025). Python basics for robotics. Kev's Robots Learning Platform. https://www.kevsrobots.com/learn/pca9685/03_python_basics_for_robotics.html
10. Library & Information Science Academy (LIS Academy). (2024, April 2). Comprehensive guide to writing technical reports: Types, functions, and structure.
11. Maryland MEP. (n.d.). Robot troubleshooting basics. Educate MD MEP. Retrieved August 8, 2025, from https://www.educate.mdmep.org/courses/robot-troubleshooting
12. Mikell, P. G. (2014). Automation, production systems, and computer-integrated manufacturing (4th ed.). Pearson Education.
13. Niku, S. B. (2011). Introduction to robotics: Analysis, control, applications (2nd ed.). Wiley.
14. Pen2Paper. (2025). Technical report writing best practices for professionals in 2025.
15. Petruzella, F. D. (2016). Industrial electronics (5th ed.). McGraw-Hill Education.
16. PrimeLessons. (n.d.). Python lessons for LEGO® Education SPIKE™ Prime. PrimeLessons.org. https://primelessons.org/en/PyLessons.html
17. Shelly, G. B., Cashman, T. J., & Vermaat, M. E. (2010). Discovering computers: Fundamentals (4th ed.). Cengage Learning.
18. Tech With Tim. (2023). Python error handling – try/except tutorial [Video]. YouTube. https://www.youtube.com/watch?v=NIWwJbo-9_8
19. The Coding Fun. (2021, January 9). LEGO Spike Prime: Translate programming blocks into Python code. TheCodingFun.com. https://thecodingfun.com/2021/01/09/lego-spike-prime-translate-programming-blocks-into-python-code/
20. Uchiyama, T. (2022). Control LEGO SPIKE Prime with Python (Beginner's guide) [Video]. YouTube. https://www.youtube.com/watch?v=wZc1ix00cZY
21. van den Berg, I., Admiraal, W., & Pilot, A. (2006). Designing student peer assessment in higher education: Analysis of written and oral peer feedback. Teaching in Higher Education, 11(2), 135–147. https://doi.org/10.1080/13562510600583925
22. Watt, L. (2021). How to program LEGO SPIKE Prime with Python (Step-by-step demo) [Video]. YouTube. https://www.youtube.com/watch?v=lmOWPjEhGPw

Resource	QR Code to Access Resource
Electronics Tutorials. (n.d.). *Boolean Algebra and Logic Gates*. Retrieved from https://www.electronics-tutorials.ws/boolean/boolean_algebra.html	
GeeksforGeeks. (n.d.). *Introduction of K-Map (Karnaugh Map)*. Retrieved from https://www.geeksforgeeks.org/digital-logic/introduction-of-k-map-karnaugh-map/	
Khan Academy. (n.d.). *Logic Gates*. Retrieved from https://www.khanacademy.org/computing/computer-science/logic-gates	
Tutorialspoint. (n.d.). *Digital Electronics - Boolean Algebra*. Retrieved from https://www.tutorialspoint.com/digital-electronics/digital-electronics-boolean-algebra.htm	
CircuitVerse. (n.d.). *Digital Logic Circuit Simulator*. Retrieved from https://circuitverse.org/	
The Classroom. (n.d.). *Logic Gates and Truth Tables - Step-by-Step Lesson from the Classroom*. [Video]. YouTube. https://www.youtube.com/watch?v=gnyDqpRyoqI&t=818s	

GLOSSARY

- **Actuator:** The component of a robot that enables movement, converting energy into physical action, similar to muscles. Examples include motors and hydraulic cylinders.
- **Algorithm Flaw:** An error in the robot's logic that can lead to incorrect actions despite the code being accurate. For instance, a robot programmed to turn right may need to turn left instead.
- **Argument:** A value passed into a function that customizes its behaviour during execution.
- **Autonomous System:** A robot capable of independent operation and decision-making, utilizing sensors and programming.
- **Boolean Algebra:** A mathematical system to simplify and formalize the logic of digital circuits.
- **Break Statement:** A command that immediately stops a loop, skipping any remaining iterations.
- **Coding Error:** A mistake or typo in the code that can confuse the robot or cause malfunction, akin to a spelling error in a recipe.
- **Combinational Circuit:** A digital circuit whose output depends solely on its current inputs, without memory.
- **Command Dictionary:** A Python dictionary that associates input commands with specific robot functions.
- **Conditional Statement:** A structure that executes code only when a specified condition is true.
- **Contactor:** A heavy-duty electrical switch for controlling high-power loads.
- **Continue Statement:** A command that skips the current iteration of a loop and continues with the next.
- **Control Logic:** The set of instructions a robot follows to make decisions, encapsulating conditional operations.
- **Controlled Variable:** The parameter a robot aims to maintain at a specific level, such as moisture in a soil sensor.
- **Decision-Making:** The process by which a robot selects actions based on available data or conditions.
- **Design Flaw:** A physical or mechanical issue in a robot's structure that may affect its performance, like being too top-heavy.
- **Electromagnetism:** The interaction between electric currents and magnetic fields.
- **Error Handling:** Strategies for managing coding errors without causing program crashes.
- **Feedback Loop:** A process in which a robot checks its actions against desired outcomes and adjusts accordingly.
- **Final Control Element:** The last device in a control system that performs the required action.
- **Flowchart:** A visual representation of the steps a program or robot follows, aiding in mapping control logic.
- **Function:** A reusable code block designed for specific tasks.

- **Gripper:** A tool on a robot used to hold or manipulate objects, similar to a mechanical hand.
- **Hydraulic Cylinder:** A component using pressurized liquid to produce movement.
- **Industrial Automation:** The use of machinery to perform tasks with minimal human intervention.
- **Input Function:** Captures user input from the keyboard during program execution.
- **Integrated Circuit (IC):** A compact electronic chip containing logic gates and other components.
- **Karnaugh Map (K-Map):** A visual technique for simplifying Boolean expressions by grouping adjacent terms.
- **Logic Gate Diagram:** A schematic that outlines circuit operations using symbols for logic gates (AND, OR, NOT).
- **Loop:** A control structure that repeats a block of code multiple times.
- **Manipulation:** A robot's ability to physically interact with or modify objects.
- **Menu-Based Program:** A user interface that presents options for controlling program flow.
- **Modular Programming:** The practice of dividing a program into manageable, reusable functions.
- **Navigation:** The method a robot uses to move safely and accurately within an environment.
- **Operational Sequence:** A step-by-step outline of how a robot completes a task.
- **Parameter:** A placeholder variable in functions that accepts input values.
- **Pascal's Law:** The principle stating that pressure applied to a fluid is transmitted equally in all directions.
- **Pseudocode:** A simplified way of expressing a program's logic in plain language, which assists in planning before actual coding.
- **Relay:** A small switch activated by electricity, commonly used in low-power circuits.
- **Return Statement:** Sends a value back from a function to the part of the program that called it.
- **Reverse Engineering:** Analysing an existing system to understand its construction and functioning, enhancing knowledge of component interactions.
- **Robotic System:** A complete robot comprising sensors, actuators, and control elements.
- **Sensor:** A device that gathers information from the environment, acting like the robot's senses.
- **Setpoint:** The target value a robot strives to achieve, such as maintaining a specific temperature.
- **Solenoid Valve:** A valve controlled by an electric coil and magnetic field.
- **Sum-of-Products (SOP):** A method for creating Boolean expressions by combining product terms from a truth table.
- **Technical Report:** A formal document outlining a robot problem and proposed solutions.
- **Troubleshooting:** The systematic process of identifying and fixing issues in a robot through testing and root cause analysis.

- **Truth Table:** A comprehensive table showing all possible input combinations for a circuit and their resulting outputs.
- **Valve:** A device that regulates the flow of liquids or gases.
- **While Loop:** A control structure that repeats code while a specified condition remains true.

Acknowledgements













List of Contributors

Name	Institution
Isaac Nzoley	Wesley Girls High School
Dr. Elieel Keelson	Kwame Nkrumah University of Science and Technology
Dr. Kwame Oteng Gyasi	Kwame Nkrumah University of Science and Technology