UNIVERSITY OF CANBERRA
INTRODUCTION TO INFORMATION TECHNOLOGY (4478/8936)

Assignment 1: The Solving Problem Process.

PART 1: ON THE SOLVING PROBLEM PROCESS

Step 1: Understand and Define the Problem (Analyse)

Question:

1. What features must the feeder include?

ANSWER:

The feature required:

- Dispense food at scheduled time.
- Monitor food level or consumption.
- System alert if food container empty.

2. What inputs and outputs are needed (e.g., feeding time, sensors)?

ANSWER:

Input:

- Feeding time setting (e.g., 8am,12:30pm, and 6:30pm)
- Food level sensor.
- Bowl weight sensor to control the required pet food consumption and for analyzing if the pet eats the food.

Output:

- DC motor or Servo Motor activation.
- Alerts for food level.

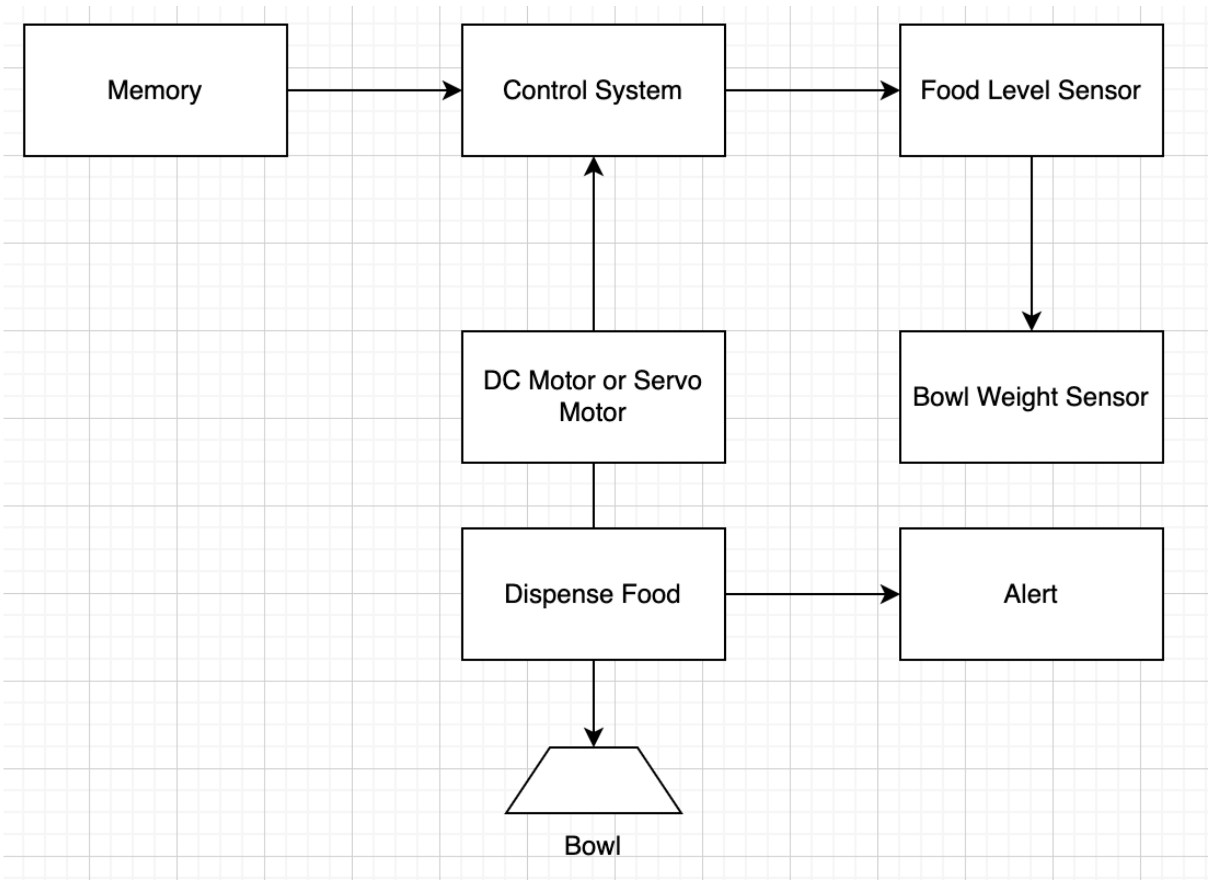
3. What are possible assumption or limitation (e.g., limited memory, one type of pet food)?

ANSWER:

Assumption:

- One Type of pet food at a time.
- Limited memory
- Type of animal like only for cat and dogs.
- System runs continuously

Block Diagram of the System



https://drive.google.com/file/d/1g-dqiU-eB6Q0LEBwrNyUyX5ElDzKAB6B/view?usp=drive_link

Step 2: Organise and Describe the Data

ANSWER:

Input	Type	Sample Values	Constraints
Feeding Time	Time (RTC)	8:00am, 12:30pm, and 6:30pm	Thrice daily
Food Level Sensor	Boolean or Analog	Full, Half, and Empty	Must be greater than 10% to dispense
Bowl Weight Sensor	Analog	0g, 150g, and 300g	Must increase after feeding

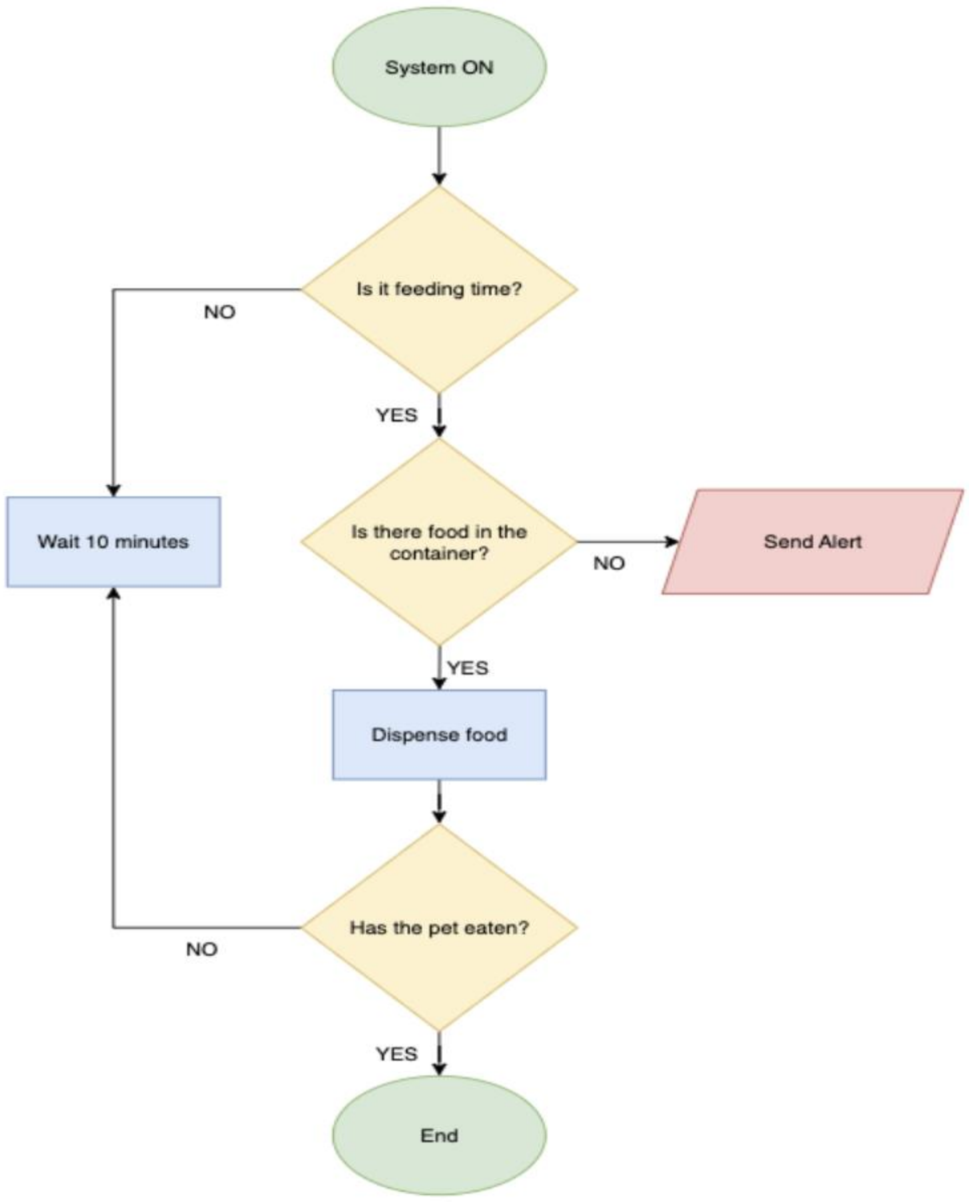
Output	Action	Description
DC Motor or Servo Motor	Rotate	Dispense food
Alert System	Notify Staff	If food is not eaten or food storage is empty

Assumption
One Type of pet food at a time.
Limited memory
Type of animal like only for cat and dogs.
System runs continuously

Step 3: Plan the Solution (Plan the Algorithm)

ANSWER:

Process Flow Chart of Pet Feeder



https://drive.google.com/file/d/1QPyqi6dN2bQMLNKrvwDERXw24c701snr/view?usp=drive_link

Step 4: Implement the Solution (Word Coding)

ANSWER:

```
# Start System
while True:
  current_time = get_time()   # function to read RTC or system clock

  # Check if current time matches feeding schedule
  if current_time in ["08:00", "12:30", "18:00"]:

    if food_level() > 10:   # check food hopper sensor (% remaining)
      rotate_servo()     # dispense food

      wait(600)        # wait 10 minutes (600 seconds)

      if bowl_weight_unchanged():
        send_alert("Pet did not eat food")

    else:
      send_alert("Food bin empty")

  wait(60)  # repeat check every minute
```

Step 5: Test and Refine the Solution (Debug and Verify)

ANSWER:

Test Cases:

1. Pet eats as expected → Bowl weight increases → No alert
2. Pet does not eat → Bowl weight unchanged → Alert sent
3. Food bin empty → No dispensing → Alert sent

Suggestions:

1. Add retry logic for dispensing
2. Include manual override
3. Log feeding history

Part 3: On AI Agent Integration

ANSWER:

Explore real-world implementation

- Microcontroller: Arduino Uno or ESP32
- Servo Motor or DC Motor: To rotate and dispense food
- RTC Module: Ds3231 for accurate timekeeping
- Weight Sensor: HX711 + load cell under the bowl
- Food Level Sensor: Ultrasonic or IR Sensor
- Alert System: Buzzer, LED, or Wi-Fi module for notifications.

Improve Documentation in README.md file includes:

- Project Overview
- Features
- System Design
- Folder Structure
- Usage instructions
- Acknowledgements.

Short Reflection:

I used Microsoft Copilot to refine the logic of my automated pet feeder system. I asked it to review my Word Code and suggest improvements. It responded with a more modular and descriptive version that included a loop, sensor thresholds, and clearer logic. This helped me understand how to structure my code more efficiently and think about edge cases like sensor failure or missed feedings.

I also asked Copilot to propose alternative solutions. It suggested an event-driven approach, which I hadn't considered. This made me realize that polling every minute might not be the most efficient method, especially for battery-powered systems.

Finally, I explored how this system could be implemented using real-world hardware like Arduino and sensors. Copilot provided a list of components and explained how they could be connected.

Overall, using Copilot helped me improve the quality of my solution, expand my thinking, and better understand how software logic translates to hardware systems. It also made me reflect on the ethical responsibility of ensuring such systems are reliable and safe for animals.