1. Introduction

In today's busy lifestyle, taking care of plants can be challenging. It's easy to forget to water the plants when you're away from home or have a hectic schedule. To address this issue, I developed a smart plant management system using Raspberry Pi and Telegram. This system allows users to remotely monitor the status of their plants and water them when necessary. By leveraging Telegram's group chat functionality, friends and family can collaboratively manage the plants, making it more efficient and convenient.

This report provides a detailed explanation of the project's main idea, the software and hardware components used, and the code implementation. It also discusses the challenges encountered during the project and the solutions implemented to overcome them. The smart plant management system serves as an example of how IoT technology and Telegram bots can be effectively utilized in real-life applications.

2. Main Idea

The main idea of this project is to manage a plant using a Telegram chatbot. The system includes three primary commands: status, water, and last watered.

- The status command checks the soil moisture level of the plant and provides a real-time update to the user.
- The water command allows users to remotely water the plant.
- The last watered command records the last time the plant was watered.

Additionally, every morning at 9 AM, the system measures the soil moisture level. If the moisture level exceeds the threshold, it records the time in last watered. If the moisture level is below the threshold, it executes the water command to water the plant.

By using these commands, users can easily monitor and manage their plants remotely, ensuring they receive the necessary care even when the users are away from home. The integration of Telegram's group chat functionality also allows friends and family to collaboratively manage the plant, making the process more efficient and convenient.

3. Software

3-1. Libraries and Software Used

This project utilizes several libraries and software tools to manage the plant using a Raspberry Pi and a Telegram chatbot.

The primary programming language used for scripting and automation is **Python**. To control the GPIO pins on the Raspberry Pi, the **RPi.GPIO library** is employed. This library is essential for activating the relay module, which controls the water pump.

For interacting with the Grove sensors and modules, the **Grove.py library** is used. Specifically, it reads data from the soil moisture sensor connected to the Grove Hat on the Raspberry Pi. To install this library, the command pip install grove.py is used.

The **python-telegram-bot library** is crucial for interfacing with the Telegram Bot API. It handles the Telegram bot commands such as status, water, and last watered. This library can be installed using pip install python-telegram-bot.

To schedule periodic tasks, the **schedule library** is used. It is particularly useful for scheduling the daily check of soil moisture every morning at 9 AM. This library can be installed with pip install schedule.

The **logging library** provides a flexible framework for emitting log messages from Python programs. It is used to log various events and errors for debugging and monitoring purposes.

The datetime library supplies classes for manipulating dates and times. It is used to record the time when the plant is watered.

Finally, the **threading library** is used to run scheduled tasks in a separate thread, allowing the main program to continue running smoothly.

These libraries and software components work together to create a system that can monitor and manage the plant remotely through a Telegram chatbot. The combination of these tools ensures that the plant receives the necessary care even when the user is away from home, making the process efficient and convenient.

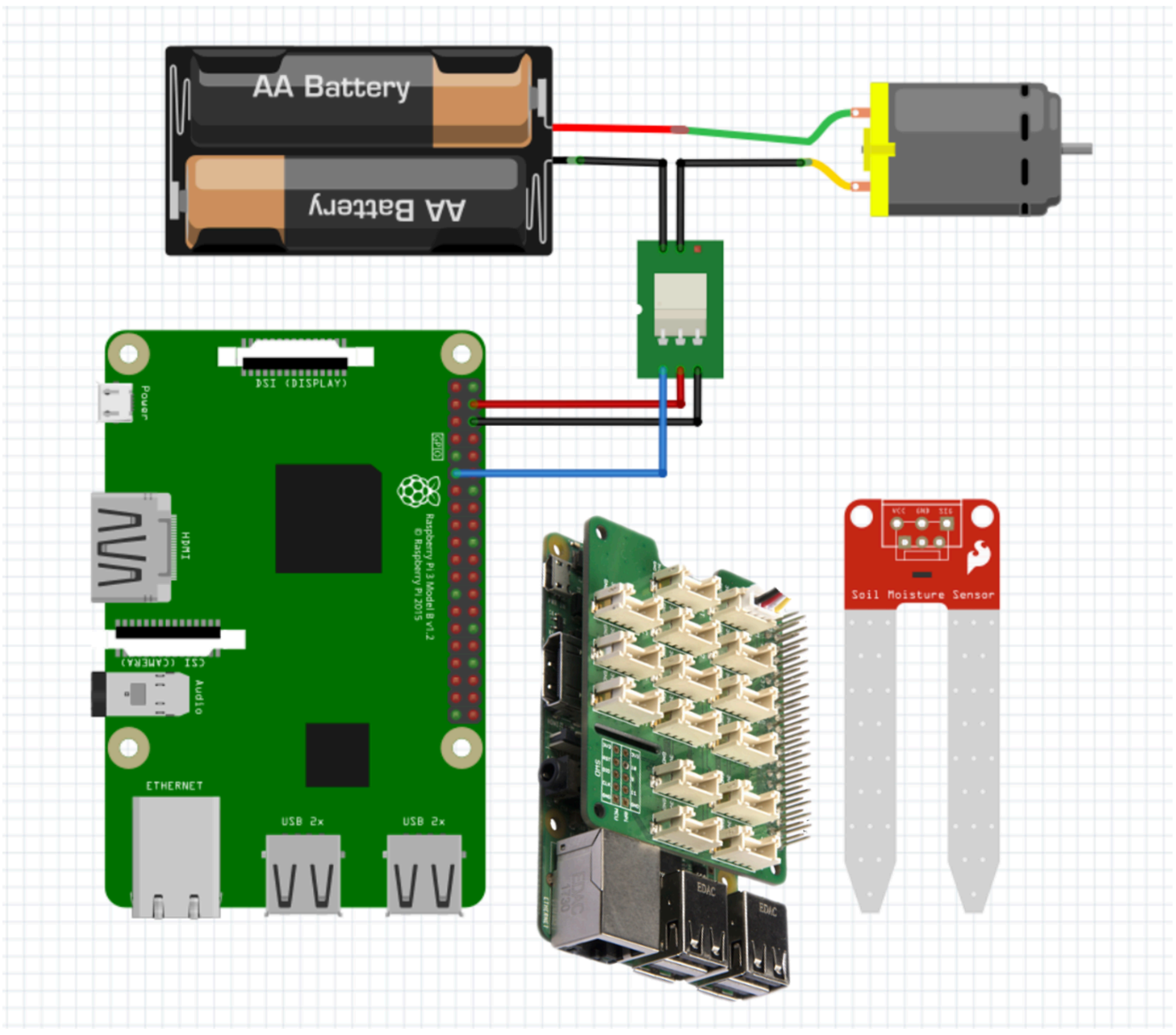
3-2. Challenges Faced

One of the significant challenges encountered during this project was the installation of the Grove.py library. The installation process was difficult and repeatedly failed, which was frustrating and time-consuming. To resolve this issue, I decided to reinstall the Raspberry Pi image provided by Grove. This image comes with the Grove modules pre-installed, which eliminated the installation issues I was facing. Using the pre-configured image from Grove saved a considerable amount of time and ensured that all necessary libraries were correctly set up. This solution allowed me to proceed with the development of the project without further delays.

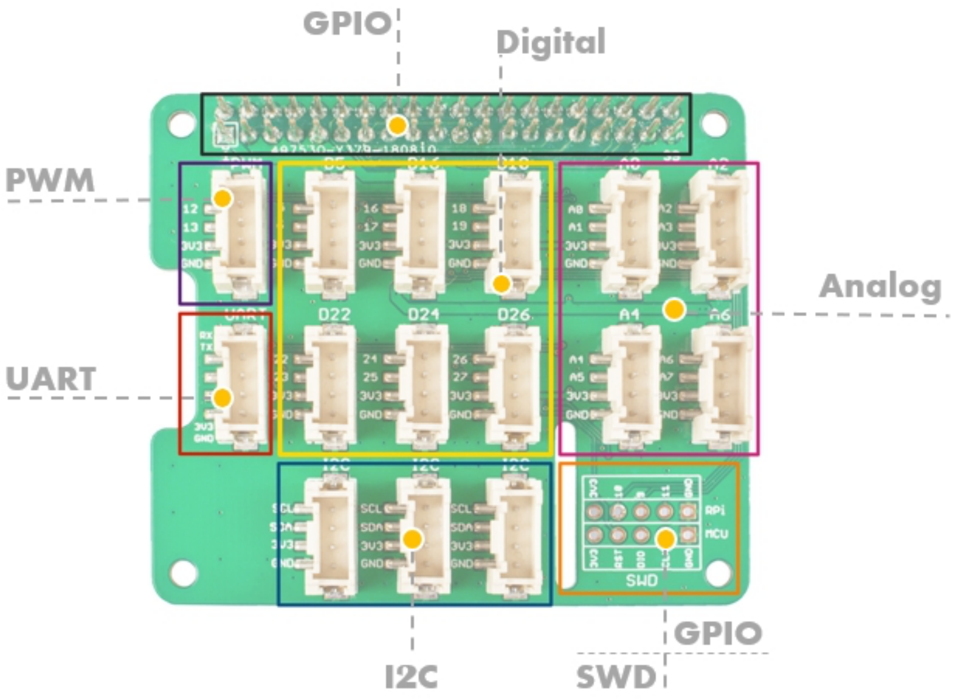
4. Hardware

4-1. Circuit Diagram

The circuit diagram for this project includes a Grove Hat, a soil moisture sensor, a 9V battery, and a water pump.



4-2. Details of the Devices



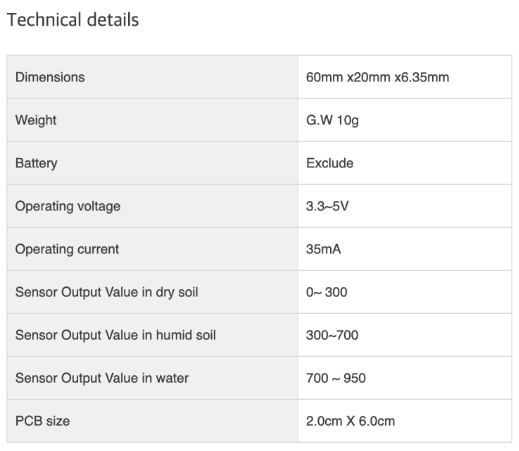
The **Grove Hat** is used because the Raspberry Pi cannot directly read analog sensors. By using the Grove Hat, the Raspberry Pi gains the ability to interface with analog sensors. This significantly simplifies the circuit, as it allows easy connection of various Grove modules through its standardized ports. In this project, the Grove Hat is connected to the Raspberry Pi and provides the necessary analog port for the soil moisture sensor from Grove.

The Grove Hat for Raspberry Pi expands the Raspberry Pi's connectivity options, allowing it to interface with various Grove sensors and modules. It provides multiple analog and digital ports, simplifying the connection process and reducing the need for additional wiring. The Grove Hat enables the Raspberry Pi to read analog sensor data, which it cannot do natively.

The **soil moisture sensor from Grove** is connected to one of the analog ports on the Grove Hat. This sensor measures the moisture level in the soil and provides an analog output that corresponds to the moisture level. The use of the Grove soil moisture sensor, together with the Grove Hat, simplifies the circuit design and makes the connections straightforward.

The soil moisture sensor detects the moisture level in the soil and provides an analog output corresponding to the moisture level. This sensor is easily connected to the analog port on the Grove Hat, simplifying the circuit design.

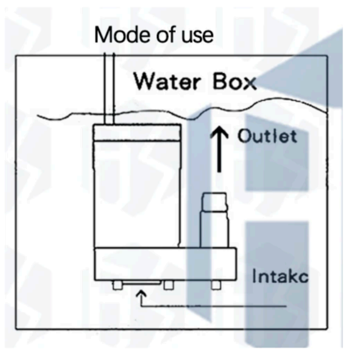
Grove - Soil Moisture Sensor



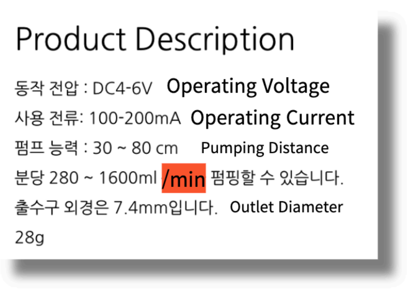
Technical details

Dimensions	60mm x20mm x6.35mm
Weight	G.W 10g
Battery	Exclude
Operating voltage	3.3~5V
Operating current	35mA
Sensor Output Value in dry soil	0~ 300
Sensor Output Value in humid soil	300~700
Sensor Output Value in water	700 ~ 950
PCB size	2.0cm X 6.0cm

The **relay module** acts as a switch to control the water pump. It isolates the Raspberry Pi from the high-power pump circuit, protecting the Pi from potential damage. The relay module is powered by the 9V battery, ensuring reliable operation of the water pump.



Water Pump



Product Description

동작 전압 : DC4-6V Operating Voltage
사용 전류: 100-200mA Operating Current
펌프 능력 : 30 ~ 80 cm Pumping Distance
분당 280 ~ 1600ml /min 펌핑할 수 있습니다.
출수구 외경은 7.4mm입니다. Outlet Diameter
28g

The **water pump** is a small electric pump powered by the 9V battery. It delivers water to the plant when activated by the relay module, ensuring the plant is watered as needed. The water pump is a small electric pump powered by the 9V battery. It delivers water to the plant when activated by the relay module, ensuring the plant is watered as needed.

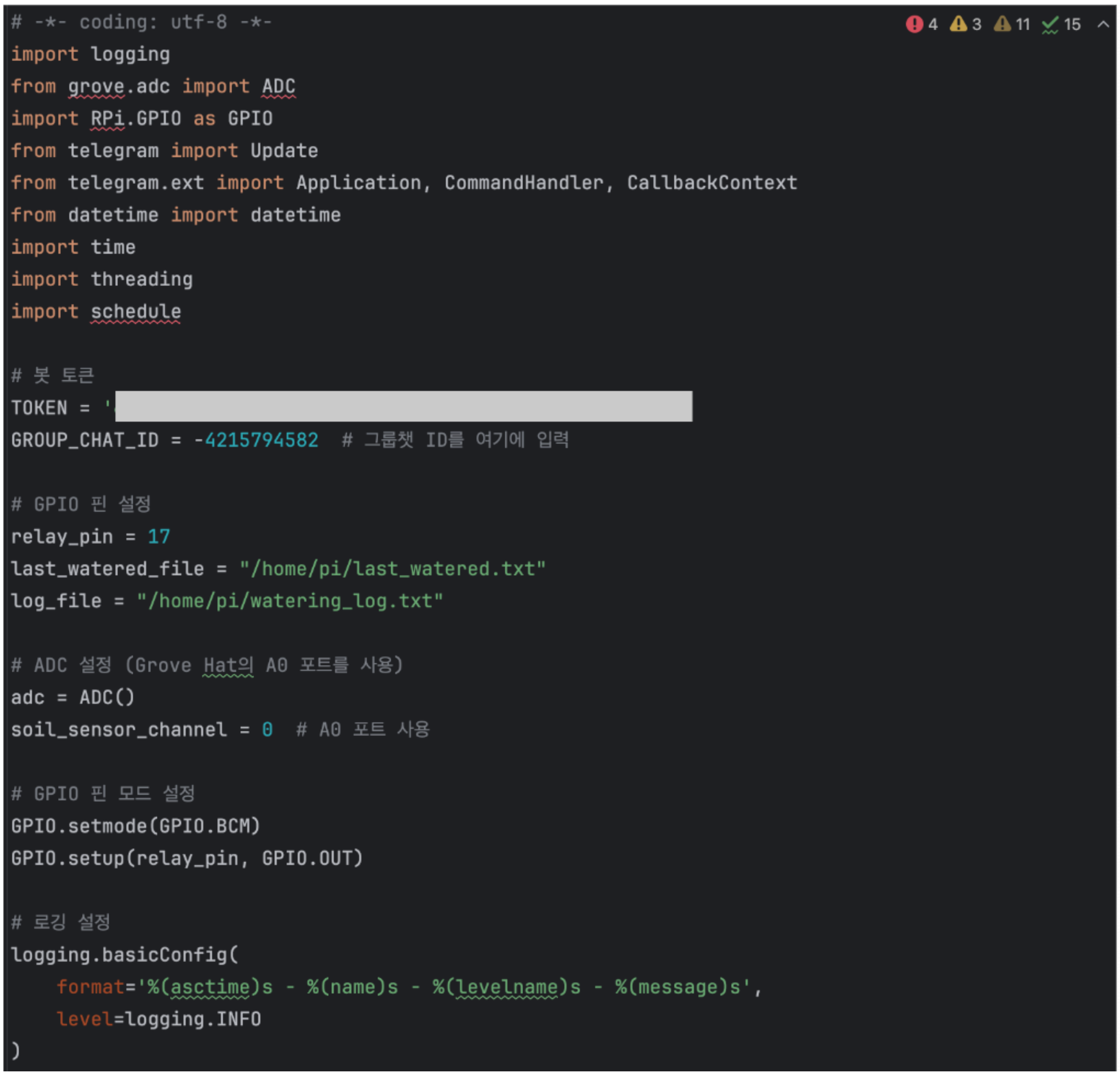
A **9V battery** is used to power the relay module and the water pump. This ensures that the pump receives sufficient power to operate efficiently. The water pump is connected to the relay module and is controlled by the Raspberry Pi via the relay module to water the plant as needed.

4-3. Challenges Faced

One of the significant challenges encountered during the hardware setup was figuring out how to use the water pump. I was unsure where to connect the positive and negative terminals of the pump and how to integrate it with the Raspberry Pi. The most challenging part was combining the software and hardware. While both the code and hardware worked well individually, the hardware would stop working whenever I combined them.

However, by carefully configuring the GPIO pins and searching online for instructions on connecting the relay module, battery, and pump, I was able to get everything working together. Following the instructions I found through extensive googling, I was able to properly connect the relay module, battery, and pump, which resolved the issues I was facing.

5. Codes



```python
# -*- coding: utf-8 -*-
import logging
from grove.adc import ADC
import RPi.GPIO as GPIO
from telegram import Update
from telegram.ext import Application, CommandHandler, CallbackContext
from datetime import datetime
import time
import threading
import schedule

# 봇 토큰
TOKEN = '[redacted]
GROUP_CHAT_ID = -4215794582  # 그룹챗 ID를 여기에 입력

# GPIO 핀 설정
relay_pin = 17
last_watered_file = "/home/pi/last_watered.txt"
log_file = "/home/pi/watering_log.txt"

# ADC 설정 (Grove Hat의 A0 포트를 사용)
adc = ADC()
soil_sensor_channel = 0  # A0 포트 사용

# GPIO 핀 모드 설정
GPIO.setmode(GPIO.BCM)
GPIO.setup(relay_pin, GPIO.OUT)

# 로깅 설정
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
```

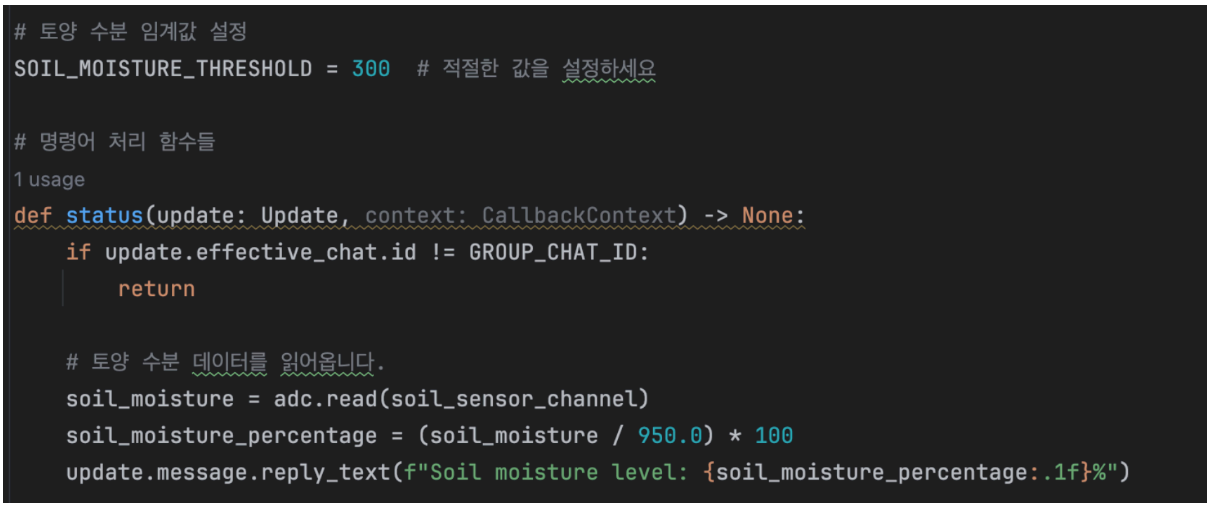
importing the necessary libraries and modules for the project. These include libraries for logging, handling GPIO pins, interacting with the ADC for reading sensor values, and managing the Telegram bot.

The first part of the code initializes the Telegram bot by loading the bot token and the group chat ID. The TOKEN variable holds the bot token, which is used to authenticate the bot with the Telegram API. The GROUP_CHAT_ID variable contains the ID of the Telegram group chat where the bot will operate.

Next, the code sets up the GPIO pins and file paths for logging. The relay_pin variable is set to 17, which specifies the GPIO pin connected to the relay module. The last_watered_file and log_file variables define the paths to the files where the last watering time and watering logs will be stored, respectively.

The ADC (Analog-to-Digital Converter) is then configured. The adc variable is initialized using the ADC() class from the Grove library, and soil_sensor_channel is set to 0, indicating that the soil moisture sensor is connected to the A0 port on the Grove Hat.

The code proceeds to set up the GPIO pin mode and the relay pin. The GPIO.setmode(GPIO.BCM) function configures the GPIO to use the BCM (Broadcom) pin numbering. The GPIO.setup(relay_pin, GPIO.OUT) function sets the relay pin as an output pin, allowing the Raspberry Pi to control the relay module.



```
# 토양 수분 임계값 설정
SOIL_MOISTURE_THRESHOLD = 300  # 적절한 값을 설정하세요

# 명령어 처리 함수들
1 usage
def status(update: Update, context: CallbackContext) -> None:
    if update.effective_chat.id != GROUP_CHAT_ID:
        return

    # 토양 수분 데이터를 읽어옵니다.
    soil_moisture = adc.read(soil_sensor_channel)
    soil_moisture_percentage = (soil_moisture / 950.0) * 100
    update.message.reply_text(f"Soil moisture level: {soil_moisture_percentage:.1f}%")
```

The provided code snippet begins by setting up the threshold for soil moisture. The variable SOIL_MOISTURE_THRESHOLD is set to 300, which is the value used to determine whether the soil moisture level is sufficient.

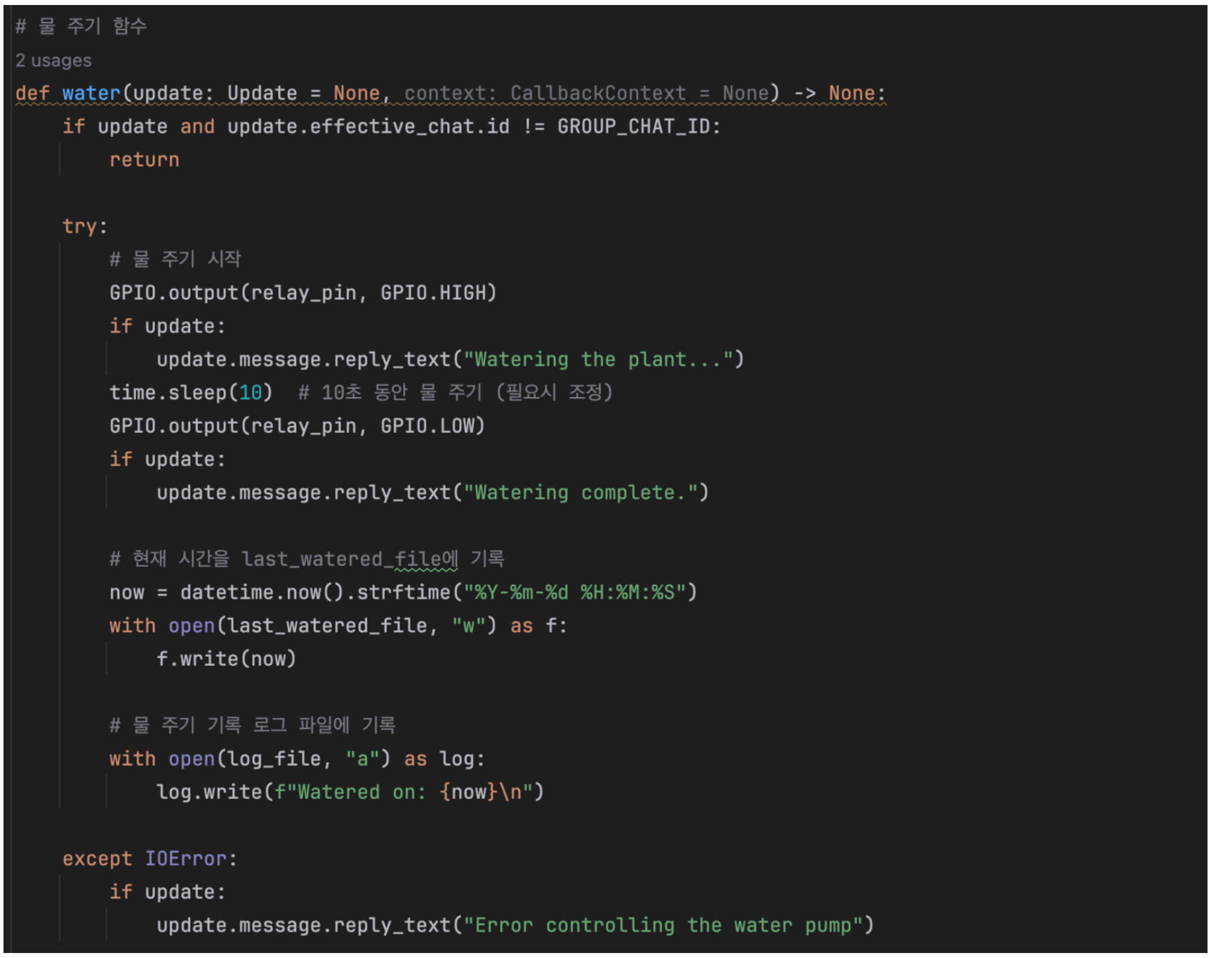
The status function is responsible for handling the status command from the Telegram bot. This function checks the soil moisture level and reports it back to the user. Here's a detailed explanation of the function:

The function **status** takes two parameters, update and context, both related to the Telegram bot's operation. It first checks if the message is from the correct group chat by comparing update.effective_chat.id with GROUP_CHAT_ID. If the message is from a different chat, the function returns without doing anything.

If the message is from the correct group chat, the function proceeds to read the soil moisture data from the sensor. It uses the adc.read(soil_sensor_channel) method to get the raw value from the soil moisture sensor. This raw value can range from 0 to 950.

To make the data more user-friendly, the raw soil moisture value is converted to a percentage. This conversion is done by dividing the raw value by 950.0 and then multiplying by 100. The result is stored in soil_moisture_percentage, which represents the soil moisture level as a percentage.

Finally, the function sends a message back to the user with the soil moisture level, formatted to one decimal place. The message is sent using update.message.reply_text, which displays the soil moisture level in a clear and understandable format.



```python
# 물 주기 함수
2 usages
def water(update: Update = None, context: CallbackContext = None) -> None:
    if update and update.effective_chat.id != GROUP_CHAT_ID:
        return

    try:
        # 물 주기 시작
        GPIO.output(relay_pin, GPIO.HIGH)
        if update:
            update.message.reply_text("Watering the plant...")
        time.sleep(10)  # 10초 동안 물 주기 (필요시 조정)
        GPIO.output(relay_pin, GPIO.LOW)
        if update:
            update.message.reply_text("Watering complete.")

        # 현재 시간을 last_watered_file에 기록
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        with open(last_watered_file, "w") as f:
            f.write(now)

        # 물 주기 기록 로그 파일에 기록
        with open(log_file, "a") as log:
            log.write(f"Watered on: {now}\n")

    except IOError:
        if update:
            update.message.reply_text("Error controlling the water pump")
```

The function **water** takes two optional parameters, update and context, both related to the Telegram bot's operation. It first checks if the message is from the correct group chat by comparing update.effective_chat.id with GROUP_CHAT_ID. If the message is from a different chat, the function returns without doing anything.

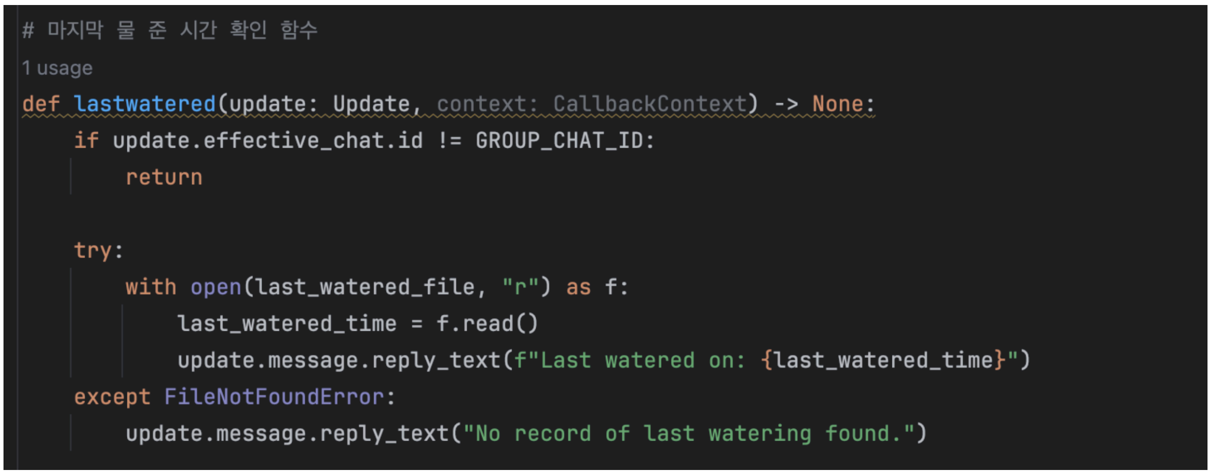
The function then enters a try block to handle the watering process. It starts by activating the relay module to turn on the water pump using GPIO.output(relay_pin, GPIO.HIGH). If the function was called through a Telegram bot command, it sends a message to the user indicating that watering is in progress.

The function waits for 10 seconds (time.sleep(10)) to allow sufficient watering time, which can be adjusted if necessary. After 10 seconds, it deactivates the relay module to turn off the water pump using GPIO.output(relay_pin, GPIO.LOW). If the function was called through a Telegram bot command, it sends another message to the user indicating that watering is complete.

Next, the function records the current time as the last watered time. It retrieves the current time using datetime.now().strftime("%Y-%m-%d %H:%M:%S") and writes this time to the last_watered_file.

Additionally, the function appends a log entry with the watering time to the log_file. This log keeps a record of all watering events for future reference.

If an IOError occurs during the file operations, the function catches the exception and, if the function was called through a Telegram bot command, it sends an error message to the user indicating that there was an issue controlling the water pump.



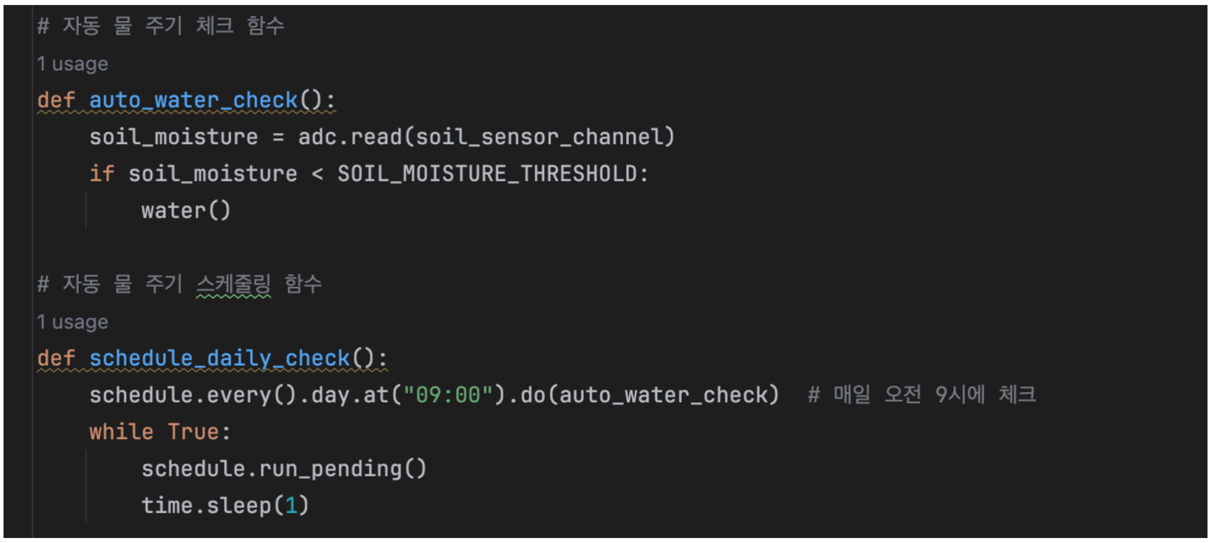
```
# 마지막 물 준 시간 확인 함수
1 usage
def lastwatered(update: Update, context: CallbackContext) -> None:
    if update.effective_chat.id != GROUP_CHAT_ID:
        return

    try:
        with open(last_watered_file, "r") as f:
            last_watered_time = f.read()
            update.message.reply_text(f"Last watered on: {last_watered_time}")
    except FileNotFoundError:
        update.message.reply_text("No record of last watering found.")
```

The function lastwatered takes two parameters, update and context, both related to the Telegram bot's operation. It first checks if the message is from the correct group chat by comparing update.effective_chat.id with GROUP_CHAT_ID. If the message is from a different chat, the function returns without doing anything.

The function then enters a try block to handle reading the last watered time from the file. It opens the last_watered_file in read mode ("r") and reads the content of the file, which contains the timestamp of the last watering. This timestamp is stored in the variable last_watered_time.

If the file is successfully read, the function sends a message back to the user with the last watered time using update.message.reply_text. The message is formatted to display the last watered time clearly.

If the file is not found (FileNotFoundError), the function catches the exception and sends a message to the user indicating that there is no record of the last watering. This ensures that the user is informed even if the watering history is not available.



```
# 자동 물 주기 체크 함수
1 usage
def auto_water_check():
    soil_moisture = adc.read(soil_sensor_channel)
    if soil_moisture < SOIL_MOISTURE_THRESHOLD:
        water()

# 자동 물 주기 스케줄링 함수
1 usage
def schedule_daily_check():
    schedule.every().day.at("09:00").do(auto_water_check)  # 매일 오전 9시에 체크
    while True:
        schedule.run_pending()
        time.sleep(1)
```

The **auto_water_check** function is designed to read the soil moisture level and determine if the plant needs to be watered. It begins by using the ADC (Analog-to-Digital Converter) to read the current soil moisture level from the sensor. The function then compares this moisture level with a predefined threshold, SOIL_MOISTURE_THRESHOLD. If the soil moisture level is below this threshold, indicating that the soil is too dry, the function calls the water function to activate the water pump and water the plant. This function ensures that the plant is automatically watered whenever the soil moisture level falls below the specified threshold, maintaining optimal soil moisture levels for plant health.

The **schedule_daily_check** function sets up a daily schedule to automatically check the soil moisture level and water the plant if necessary. It schedules the auto_water_check function to run every day at 9:00 AM using the schedule library. Once scheduled, the function enters an infinite loop where it continuously checks for any scheduled tasks that are due to run. The loop runs all the tasks that are scheduled to be executed and pauses for 1 second between each check to avoid overloading the CPU. This function ensures that the auto_water_check function is executed every day at the specified time, automating the process of checking the soil moisture level and watering the plant as needed.

6. Conclusion

In conclusion, this project successfully demonstrates how to use a Raspberry Pi and a Telegram bot to create an efficient and automated plant watering system. By leveraging the capabilities of the Grove Hat, soil moisture sensor, relay module, and water pump, we can remotely monitor and manage plant care. The integration of software and hardware through Python programming and careful configuration ensures that the plant is watered when necessary, maintaining optimal soil moisture levels. This system not only simplifies plant care but also enables collaborative management through Telegram, making it an effective solution for busy individuals and families.

Thank you.