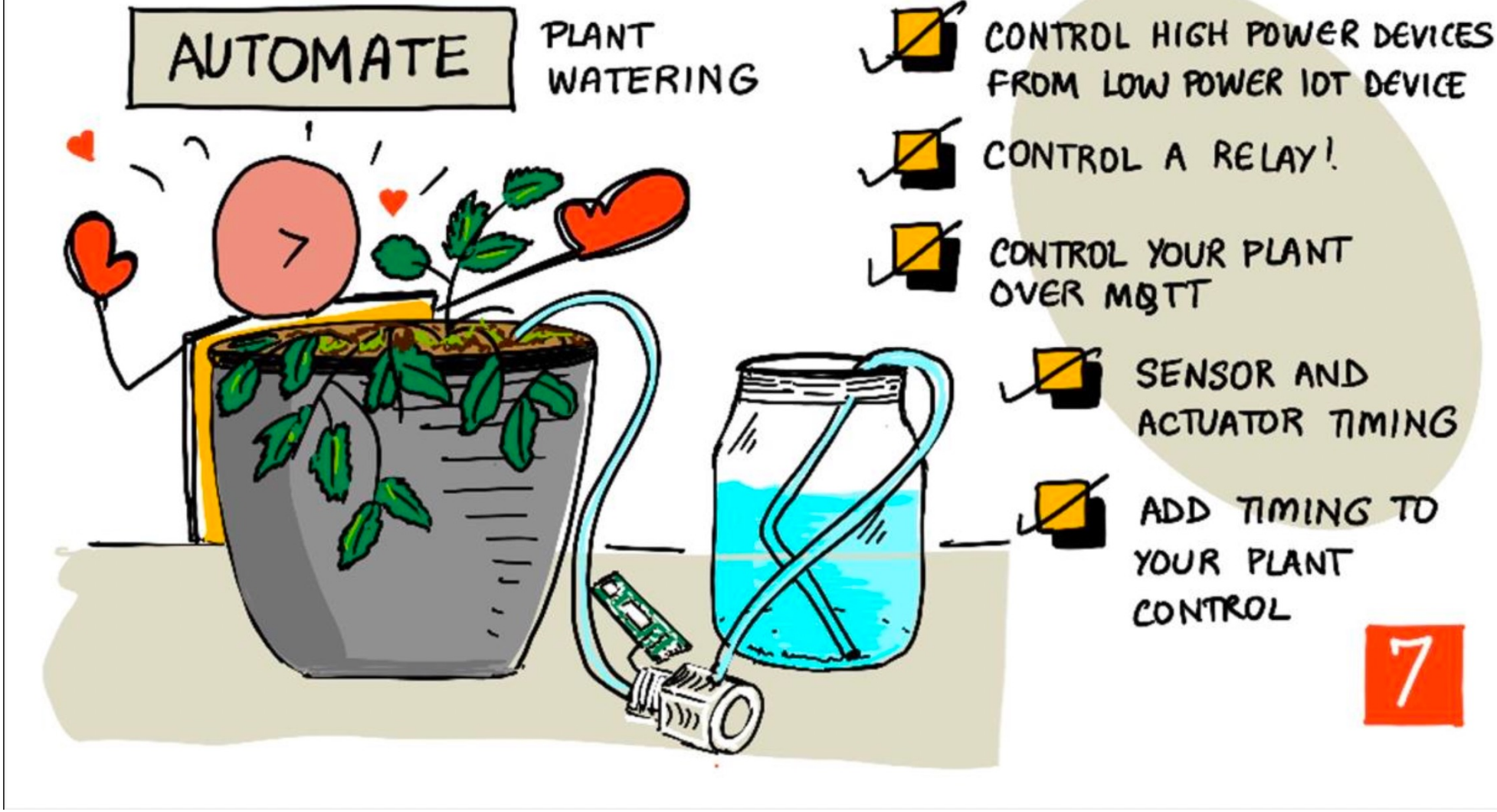
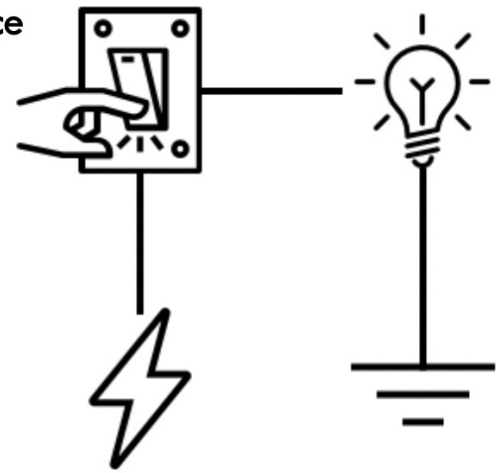
Based on:

AKA.MS/IOT-BEGINNERS


AUTOMATE
PLANT WATERING
CONTROL HIGH POWER DEVICES FROM LOW POWER IOT DEVICE
CONTROL A RELAY!
CONTROL YOUR PLANT OVER MQTT
SENSOR AND ACTUATOR TIMING
ADD TIMING TO YOUR PLANT CONTROL
7

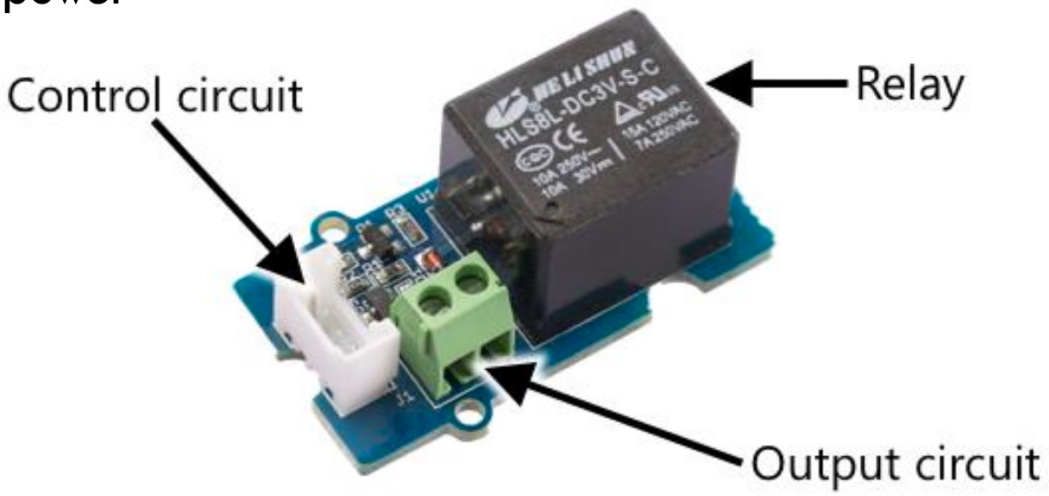
CONTROL HIGH POWER DEVICES FROM LOW POWER DEVICES

- IoT devices use a low voltage
- Pumps use higher voltages
- How can we control a pump from an IoT device
  - Separate power supply to the pump
  - Switch for that power supply controlled by the IoT device



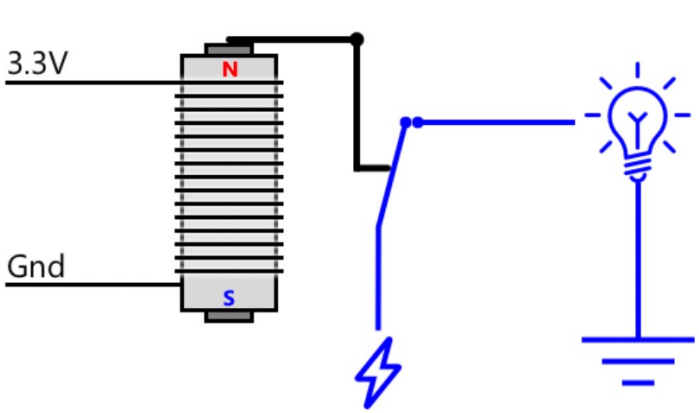
RELAY

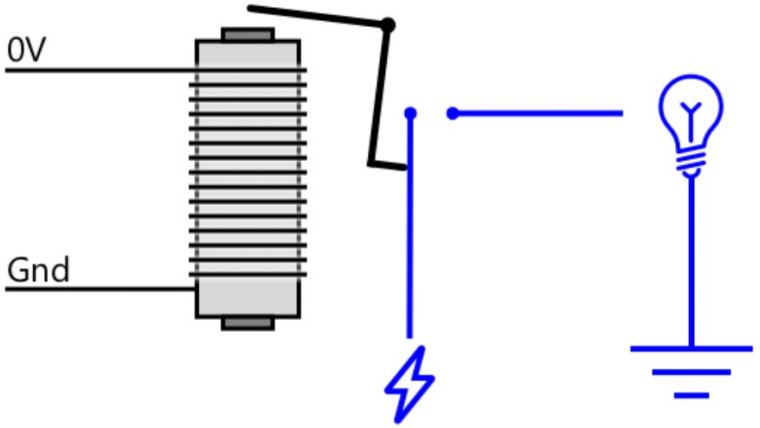
- Relays are electromechanical switches
- Low power to control the switch
- Switch can turn on/off higher power



RELAY

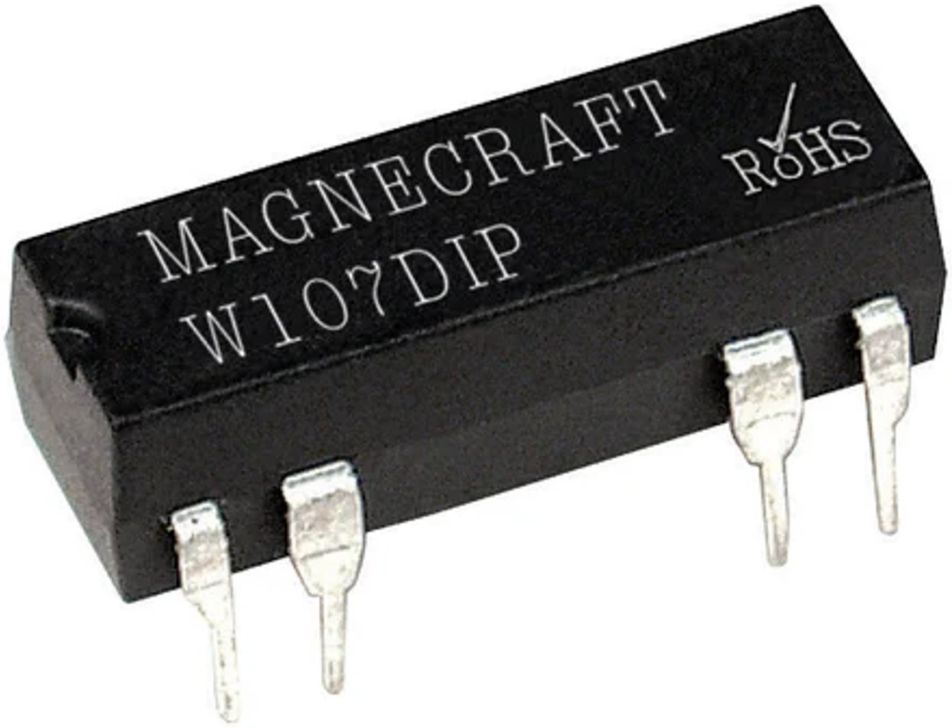
- Power to the relay turns on a magnet, moving a mechanical switch
- Remove the power and the magnet turns off, releasing the switch





RELAY HARDWARE

RELAY HARDWARE



DEMO: CONTROL THE RELAY

Connect the relay

Program the device

DEMO: CONTROL YOUR PLANT OVER MQTT

Send soil moisture data over MQTT

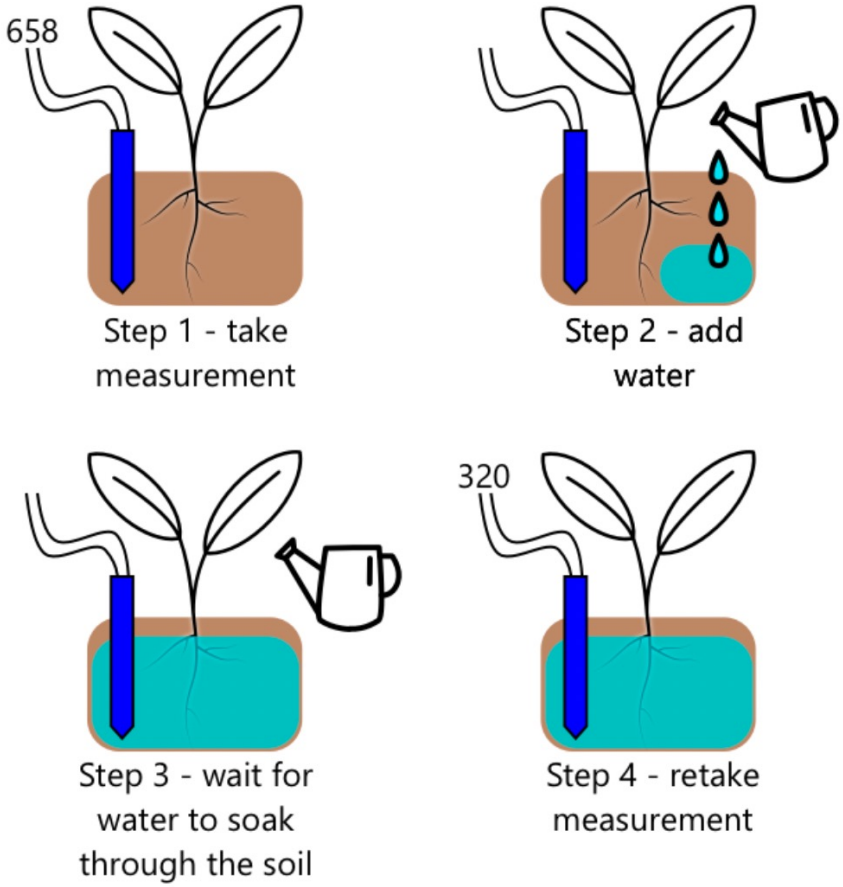
Send commands to control the relay over MQTT

SENSOR AND ACTUATOR TIMING

SOIL MOISTURE TIMING

SOIL MOISTURE TIMING

- Better to add water, wait, then measure.
- Add more if needed
- Too little can be fixed by adding more, too much can't be fixed



ADD TIMING TO OUR SERVER

- Check telemetry
- Check the soil moisture level
- If ok
  - Do nothing
- If too low
  - Send a command to turn the relay on
  - Wait 5 seconds
  - Send a command to turn the relay off
- Wait 20 seconds, then repeat

DEMO: ADD TIMING TO OUR SERVER

```
water_time = 5
wait_time = 20
```

This defines how long to run the relay for (`water_time`), and how long to wait afterwards to check the soil moisture (`wait_time`).

```
def send_relay_command(client, state):
    command = { 'relay_on' : state }
    print("Sending message:", command)
    client.publish(server_command_topic, json.dumps(command))
```

This code defines a function called `send_relay_command` that sends a command over MQTT to control the relay. The telemetry is created as a dictionary, then converted to a JSON string. The value passed into `state` determines if the relay should be on or off.

```
def control_relay(client):
    print("Unsubscribing from telemetry")
    mqtt_client.unsubscribe(client_telemetry_topic)

    send_relay_command(client, True)
    time.sleep(water_time)
    send_relay_command(client, False)

    time.sleep(wait_time)

    print("Subscribing to telemetry")
    mqtt_client.subscribe(client_telemetry_topic)
```

This defines a function to control the relay based off the required timing. It starts by unsubscribing from telemetry so that soil moisture messages are not processed whilst the watering is happening. Next it sends a command to turn the relay on. It then waits for the `water_time` before sending a command to turn the relay off. Finally it waits for the soil moisture levels to stabilize for `wait_time` seconds. It then re-subscribes to telemetry.

```
    if payload['soil_moisture'] > 450:
        threading.Thread(target=control_relay, args=(client,)).start()
```

ASSIGNMENT:

BUILD A MORE EFFICIENT WATERING CYCLE

Vietnamese - German Univeristy

Instructor : Dong Quan Huan

Presented By

Le Duc Thanh Kim

Dao Hong Minh

STEPS FOR REQUIREMENTS

- Measurement and Data
- Watering time calculation
- code server calibration
- Results

Start the mearsuring:
desired moisture: 450
flagged moisture: >650

Readings when start recording :

App.py

```
MQTT connected!
Soil moisture: 658
sending telemetry:  {"soil_moisture": 658}
Message received: {'relay_on': True}
Soil moisture: 656
sending telemetry:  {"soil_moisture": 656}
Message received: {'relay_on': False}
Soil moisture: 653
sending telemetry:  {"soil_moisture": 653}
Soil moisture: 650
sending telemetry:  {"soil_moisture": 650}
Soil moisture: 648
sending telemetry:  {"soil_moisture": 648}
Soil moisture: 645
sending telemetry:  {"soil_moisture": 645}
Soil moisture: 643
sending telemetry:  {"soil_moisture": 643}
Soil moisture: 641
sending telemetry:  {"soil_moisture": 641}
Soil moisture: 638
```

Server.py

```
MQTT connected!
Message received: {'soil_moisture': 658}
Unsubscribing from telemetry (stop measuring when watering)
average moisture reading decrease:  None
Watering the plant for 1 seconds.
Sending message: {'relay_on': True}
Sending message: {'relay_on': False}
Wait time for moisture stabilization in the initial phase
Subscribing to telemetry
Message received: {'soil_moisture': 630}
Unsubscribing from telemetry (stop measuring when watering)
average moisture reading decrease:  None
Watering the plant for 1 seconds.
Sending message: {'relay_on': True}
Sending message: {'relay_on': False}
Wait time for moisture stabilization in the initial phase
Subscribing to telemetry
Message received: {'soil_moisture': 601}
Unsubscribing from telemetry (stop measuring when watering)
average moisture reading decrease:  None
```

Watering time adjustment method

$$\text{watering time} = \frac{(\text{soil moisture} - \text{desired moisture})}{\text{average decrease}}$$

Data collection method:

Turn on relay: 1 second

Turn off relay: 20 second (rest time)

repeat 6 times

Relevant code snippet

```python
num_iterations = 6
moisture_readings = []

def calculate_average_moisture_decrease():
    if len(moisture_readings) == num_iterations:
        differences = [moisture_readings[i-1] - moisture_readings[i] for i in range(1, num_iterations)]
        average_decrease = statistics.mean(differences)
        return average_decrease
    else:
        return None
```

```python
def control_relay(client, initial):
    print("Unsubscribing from telemetry (stop measuring when watering)")
    mqtt_client.unsubscribe(client_telemetry_topic)

    if initial:
        water = test_time
    else:
        water = watering_time  # Calculated watering time based on average decrease

    print(f"Watering the plant for {water} seconds.")
    send_relay_command(client, True)
    time.sleep(water)
    send_relay_command(client, False)

    print("Wait time for moisture stabilization in the initial phase")
    time.sleep(wait_time)

    print("Subscribing to telemetry")
    mqtt_client.subscribe(client_telemetry_topic)
```

```python
test_time = 1
watering_time=0
wait_time = 20
initial = False
```

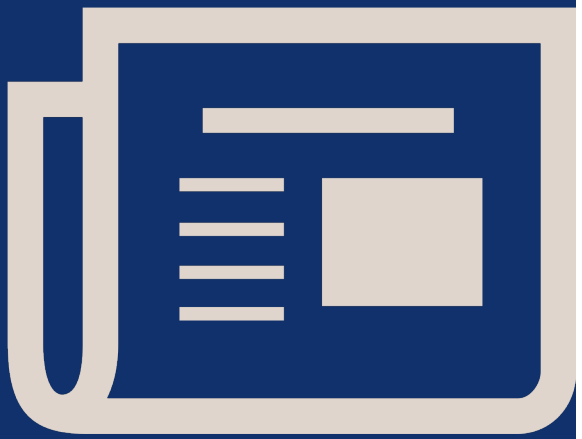
Relevant code snippet

```
def handle_telemetry(client, userdata, message):
    payload = json.loads(message.payload.decode())
    print("Message received:", payload)

    global watering_time, initial, average_decrease

    if len(moisture_readings) < num_iterations:
        initial = True

        threading.Thread(target=control_relay, args=(client,initial)).start()
        moisture_readings.append(payload['soil_moisture'])
        average_decrease = calculate_average_moisture_decrease()
        print("average moisture reading decrease: ",average_decrease)

    elif payload['soil_moisture'] > 650:
        initial = False

        desired_moisture = 450
        watering_time = int((payload['soil_moisture'] - desired_moisture) / average_decrease)
        print("Updated watering time:", watering_time)
        threading.Thread(target=control_relay, args=(client,initial)).start()

mqtt_client.subscribe(client_telemetry_topic)
mqtt_client.on_message = handle_telemetry
```

successfully
calculated average:

```
average moisture reading decrease:  29.2
```

Updated watering time
accordingly:
(653-450)/29.2 ~ 6 seconds



```
Message received: {'soil_moisture': 650}
Message received: {'soil_moisture': 653}
Updated watering time: 6
Unsubscribing from telemetry (stop measuring when watering)
Watering the plant for 6 seconds.
Sending message: {'relay_on': True}
Sending message: {'relay_on': False}
Wait time for moisture stabilization in the initial phase
Subscribing to telemetry
Message received: {'soil_moisture': 451}
Message received: {'soil_moisture': 450}
Message received: {'soil_moisture': 449}
Message received: {'soil_moisture': 450}
```


THANK
YOU