Mayank Neupane
Automated plant watering System
Version 1.1
Date released: 2023/06/08

ABSTRACT:

Maintaining indoor plants can become challenging amid busy schedules, often leading to neglect and unhealthy plant growth. To address this issue, an automated watering system leveraging the Arduino Groove Beginner kit, a water pump, an analog moisture sensor, a battery, and requisite wiring has been devised. This system aims to ensure optimal plant moisture levels even in the absence of consistent human intervention.

The setup is straightforward, offering users a user-friendly experience during installation and operation. The water pump, housed within a reservoir matching the plant's height, dispenses water until the soil attains the desired moisture level. Unlike the previous iteration in MATLAB, this upgraded version utilizes Java for enhanced functionality and precision.

In the MATLAB-based design, data analysis and programming were executed, where the pump control relied on an analog moisture sensor's readings. Utilizing an if statement, the MATLAB program triggered the MOSFET switch to activate the pump upon detecting soil dryness below a predetermined threshold. Subsequently, the pump ceased operation once the soil moisture level reached the predefined value.
Although the MATLAB-based system exhibited a success rate of 9 out of 10 trials, it encountered issues with delayed computer-to-board connections and signal transmission. These delays occasionally led to overwatering due to inaccurate control. To rectify this, transitioning to Java was considered imperative.
Java's advanced capabilities enable precise configuration, addressing the system's limitations and ensuring more accurate watering cycles. With Java's robustness, real-time communication and control between the moisture sensor and the pump are expected to significantly enhance accuracy and eliminate overwatering instances. This upgraded system promises improved reliability and effectiveness in maintaining optimal plant health.

DATA COLLECTION:

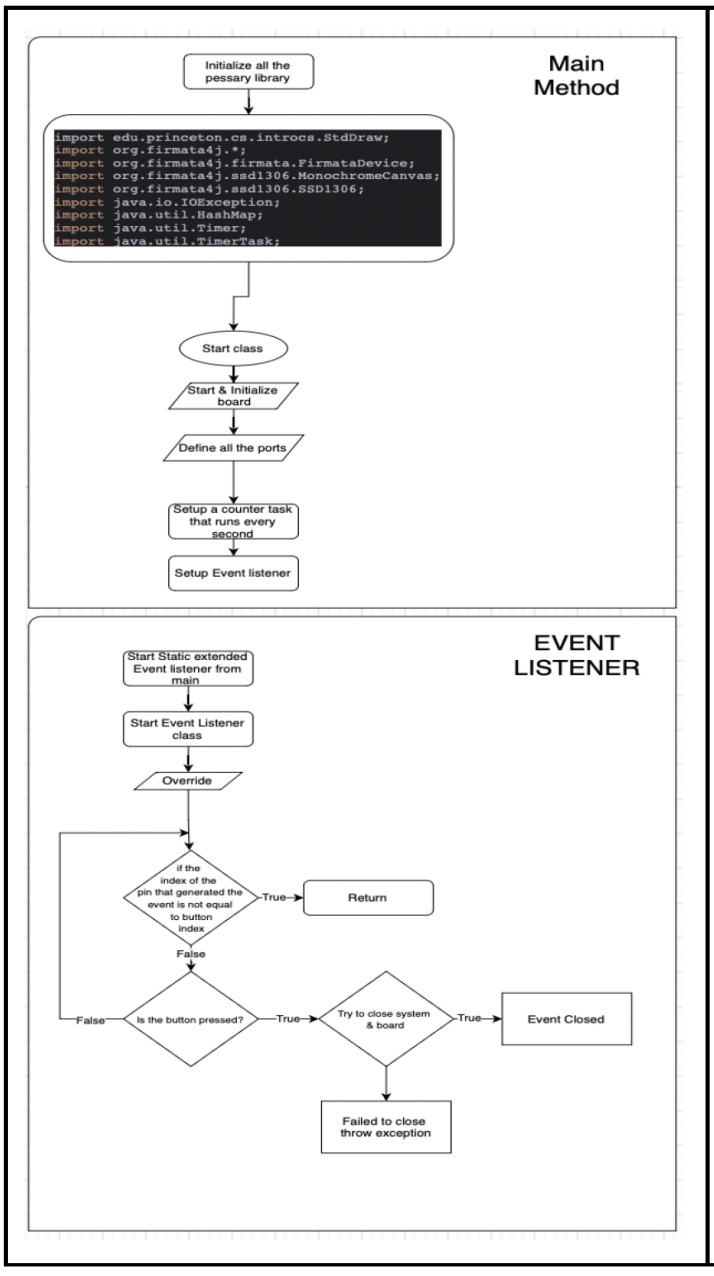
Immersed in dry soil: 660
Immersed in water-saturated soil: 540

TECHNICAL REQUIREMENTS / SPECIFICATIONS:

StdLib and Firmata4j used this project because they provide simple communication with a microcontroller which is an arduino. To achieve this Sid library sends a signal from java to arduino and firmata implements that by communicating with the board. Later Using conditional statements to communicate between the sensors when the program is running. This is achieved with the if loop for moisture and extended event listener to close the program based on the signals sent by the button. For the event listener when the user presses the button signal “0” is sent to the program triggering the event listener class.

Secondly Motor is turned on based on the Moisture sensor value that is being recorded. To produce the live Graph Princeton Library has been used, Sample is run through if loop and added that sample value to the “X value” and the loops is reset to zero when the Sample++ hits 100, Using that values its drawn in the graph to see the visual representation of the data.

Flow Chart included Below.



LIBRARY WITH THERE VERSION USED IN THIS PROJECT:

1. Firmata4j: Allows Arduino to communicate with java, Which runs Firmara library from java.
   Maven: com.github.kurbatov:firmata4j
   Version: 2.3.8
2. Princeton: Allows data collected to show visually as a 2D Graph.
   Maven: com.github.fracpete:princeton-java-examples
   Version: 1.0.2

COMPONENTS LIST:

1. Pump: Supply water to the plant with a 9V battery, What is being controlled by a mosfet board.
2. Moisture Sensor: Providing soil moisture reading to the program to react based on the value provided.
3. Groove board: Microcontrol to implement all the signals being sent.

PROCEDURE & TEST:

1. Data Collection: Initial data gathering established a baseline for program implementation, providing foundational insights for system development.

2. Design Phase: Employed Firmata4j & Sid libraries to structure the code for comprehensibility and ease of understanding across users.

3. Implementation: Merged the design blueprint with collected data, resulting in the creation of an object-oriented program that responds dynamically to incoming signals.

4. Testing: Leveraged prior knowledge of power parameters and sensor readings. Utilized a potentiometer to validate sensor values against documented references. Additionally, employed print statements for specific test cases to verify code execution, employing systematic problem-solving techniques for program optimization.