Final Project Report

— EECS 113 (2020 Spring) —

"IoT-Driven Irrigation System"

SAMUEL ANDREW BONDOC | 35261931

12 July 2020

Project Description

This project simulates an irrigation system that optimizes watering on an hourly basis by taking local sensor data measurements and comparing them online weather station data for the region.

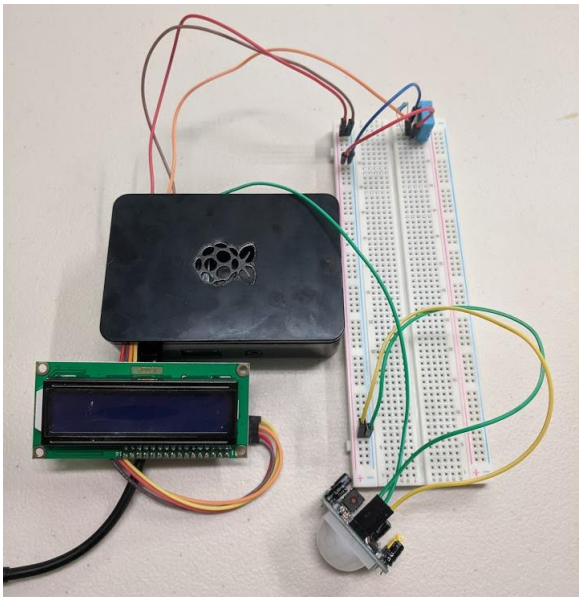

Figure 1. Irrigation system controller, including an LCD, a PIR sensor, and a temperature/humidity sensor controlled by a Raspberry Pi 3 B+.

Every minute, the system takes humidity and temperature measurements from the attached sensor, seen in the top right of Figure 1. The top line of the LCD constantly cycles through and displays these values, as well as their hourly averages.

Every hour, the system downloads humidity and temperature data from the local CIMIS station. A timestamped scrolling message showing collected CIMIS data, calculated ETo, and hourly water requirement is displayed on the second line of the LCD display.

After taking the hourly CIMIS measurements and calculating water requirements, the irrigation system starts watering the crops. The duration for which the water should be flowing is calculated using the given flow rate of the system and the volume to be distributed.

If the PIR sensor (bottom right of Figure 1) detects motion during irrigation, the system will stop water from flowing until the PIR no longer detects motion.

Schematic

The design utilizes a Raspberry Pi 3 B+ as the microcontroller unit for a DHT-11 humidity and temperature sensor, a 16x2 I2C LCD, and an HW-416 PIR sensor. The schematic for the design is presented in Figure 2.

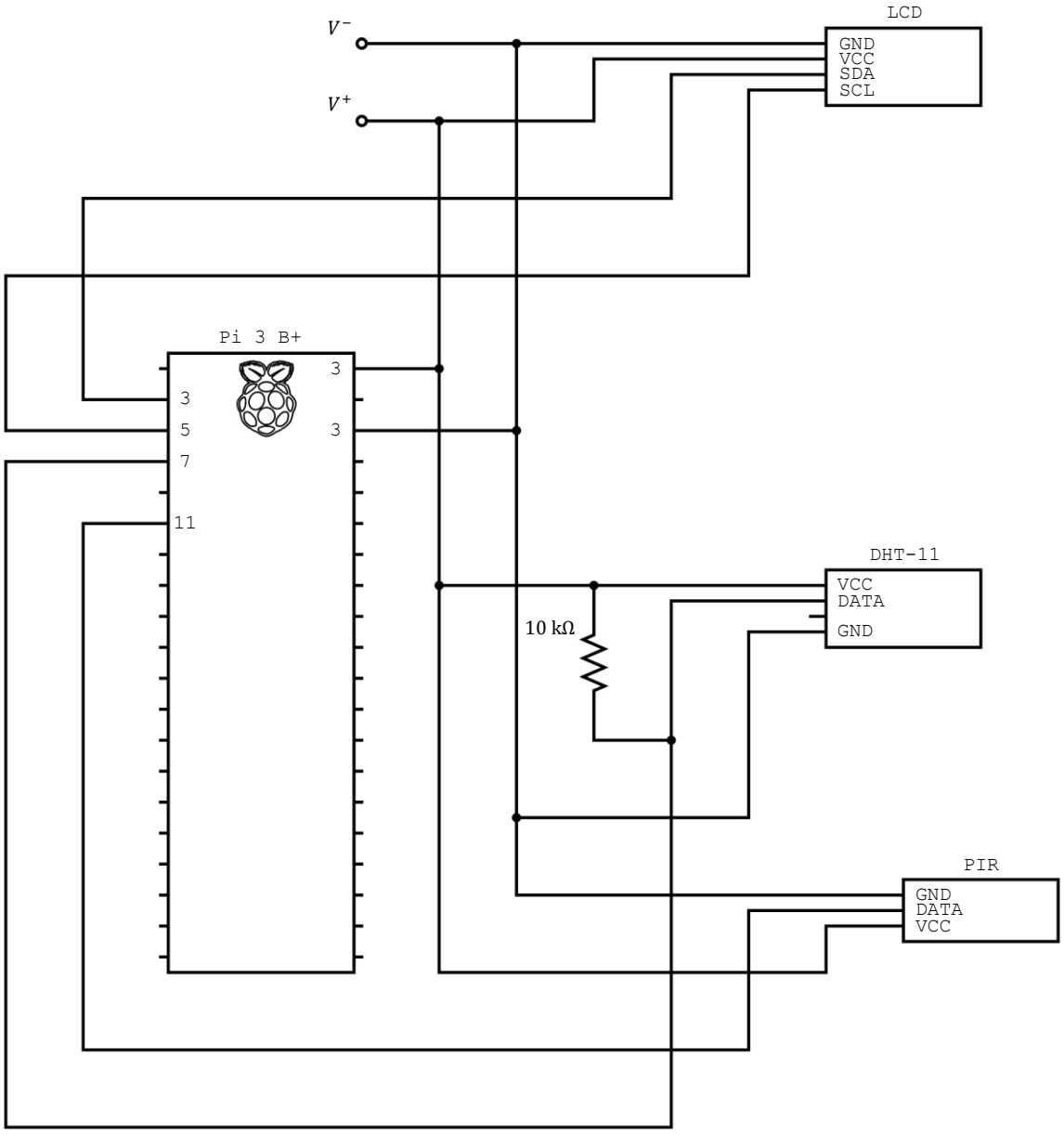

Figure 2. *Irrigation control schematic. Microcontroller power source not shown. Only used ports on the microcontroller are labelled. Empty pin on DHT-11 does not transmit data nor require input.*

The nodes V^+ and V^- in the schematic denote connections to a 5 V output pin and a ground pin, respectively, on the microcontroller unit. These represent the vertical nodes on the breadboard. Note that all `VCC` pins are connected to V^+ and all `GND` pins are connected to V^-.

A parallel resistor is required between the `VCC` and `DATA` pins of the DHT-11 in order to regulate the voltage passed through the `DATA` port.

Environmental Considerations

Given the complexity of determining the optimal amount of water to distribute, simplifications had to be made based on the kinds of data that can be acquired by the system.

Evapotranspiration Formula

Evapotranspiration (ETo) is determined by several factors that cannot all be measured by this irrigation system. From the Penman-Monteith equation for ETo ET, we can gather the following relationships:

$$ET \propto T$$
$$ET \propto \frac{1}{H}$$

Where ET is proportional to temperature T and inversely proportional to humidity H. This follows the idea that higher temperatures increase the rate of evaporation, and higher humidity lowers diffusive tendencies of water molecules to the air.

Using the above proportions, the relationship between ETo, humidity, and evaporation can be described in terms of an arbitrary environmental constant k:

$$ET = k \cdot \frac{T}{H}$$

In order to find this environmental constant, values taken from CIMIS databases for ETo, temperature, and humidity were plugged into the above equation rearranged to solve for k.

Irrigation System

The rate at which water is distributed depends on the actual piping and sprinkler configurations of the irrigation system. Sprinklers are designed to output water at a certain rate over a certain area. Given those two parameters, the amount of sprinkler heads required to cover an entire area would therefore be:

$$sprinklers\ required = \left\lceil \frac{total\ crop\ area}{individual\ sprinkler\ coverage} \right\rceil$$

Based on this, the total rate at which water is delivered to the system would be:

$$total\ rate = sprinklers\ required \cdot individual\ sprinkler\ rate$$

For the purposes of this system, standard values of ~ 3.0 gal/min and ~ 115 ft^2 were assumed for the individual sprinkler head rate and coverage, respectively.

Code Overview

All the system-specific code is contained in a single Python file that imports publicly available modules to handle the I/O of the hardware components.

Modules

`sys` – Grants access to system files. Used to modify the path for custom module imports.
`threading` – Enables parallel processes to be called and run as objects. Used to asynchronously run the LCD.
`urllib` – Enables internet data retrieval. Used to download CIMIS data.
`os` – Grants access to directory files. Used to delete temporary CIMIS data download.
`Adafruit_DHT` (https://github.com/adafruit/Adafruit_Python_DHT) – Simple interface for DHT-11 sensor. Used to gather humidity and temperature data.
`lcddriver` (https://github.com/the-raspberry-pi-guy/lcd) – Simple interface for the LCD. Used to display readings on the LCD.
`time` – Includes time-based functions. Used to implement waits.
`datetime` – Includes global clock functions. Used for timestamps.
`RPi.GPIO` – Grants access to Pi ports. Used for direct access to PIR sensor.

Retrieving Sensor Data

In order to get data from the DHT-11 sensor, the `Adafruit_DHT` module is used to obtain the humidity and temperature as a tuple. The module checks the port for serialized data and reads it to the program. In the case of this particular system, the voltage has to be stepped down with a resistor first, and the data passed to port 7 on the Pi. The values obtained are then stored in global variables for asynchronous access.

A global list of data points is maintained throughout the program, where data tuples are appended as time progresses. To find the average, all the data is summed up and divided by the number of items in the list. Once the data count reaches 60, old data starts getting removed to keep the hourly average up-to-date.

Retrieving CIMIS Data

The Pi must have internet connection to access the CIMIS database. Tabulated reports may be downloaded from the CIMIS FTP server using the `urllib` module. Once the file is downloaded, its contents are written to a string and the file is deleted.

Because the files in the database are in .csv format, the string may be parsed using a comma and newlines as delimiters. From there, the program finds the last line that has non-empty data in it and searches the preset columns for humidity and temperature data. These are returned as a tuple and stored in global variables for asynchronous access.

Calculations

Once the sensor and CIMIS data are obtained, the program simply plugs the constants and obtained values into the given formulas. A linear derating procedure is used to adjust the ETo of as calculated by sensor data and the ETo as calculated by the CIMIS data.

PIR Sensor Detection

The PIR sensor is simpler than the temperature and humidity sensor because it sends out binary data representing whether or not movement is detected. As such, the `RPi.GPIO` module was used directly to access the data sent to the port connected to the data pin of the PIR. The data pin is connected to port 11 on the Pi and a threaded loop constantly checks input from the port, updating the corresponding global variable in the code for motion detection.

LCD Management

Problems arise with the LCD due to the fact that all other functions are handled in parallel threads using the `threading` module, whereas the LCD can only read serialized data from a single port. Therefore, only one function can send an LCD message at a time or else the LCD will not display correctly.

To compensate for this, LCD messages are pre-processed and sent in batches to the LCD. Essentially, the LCD function runs on a clock that reads the necessary information and sends it to the LCD all at once. Instead of external functions sending messages to the LCD, the functions change flags that the LCD reads to determine what to display at the current cycle.

The scrolling messages are handled by dividing the message into segments to be displayed at changing positions on the LCD during each clock cycle.