Waterpai

Ananth Shyamal and Divya Shyamal

January 2021

Contents

1 Executive Summary

Waterpai is a Raspberry Pi-based smart lawn irrigation system. By utilizing historical, current, and forecast weather data, Waterpai improves irrigation efficiency. The system has been designed to coexist with the existing irrigation controller. The existing external plumbing was not modified due to winterization considerations. Waterpai has added safety features, by introducing a motorized ball valve and flow meter in the internal irrigation plumbing line and a multi-pronged alert system. Waterpai features a web-based user interface, where one can monitor the status of the irrigation and provide feedback.

Waterpai has been in continued development since May 2020, and operational from June 1 to September 30. During that period, we found it to be 11% more cost/water efficient than our existing Hunter Pro-C controller. Even after the end of the season, we have improved the system's reliability and safety, and enhanced it to source on-site data.

The development of Waterpai has been planned in two phases. In Phase I, which we completed this fall, we built a functional irrigation system that can support further irrigation methodology improvements (which will be done in Phase II). Therefore, Phase I was primarily hardware focused, and Phase II will focus mostly on the increased use of on-site sensors and algorithmic improvements (including the use of machine learning).

2 Background

2.1 Irrigation Overview

Our lawn consist of six "zones", each containing some sprinkler heads that cover a certain region. To irrigate a zone, the corresponding solenoid irrigation valve is powered. We only irrigate zones one at a time because to maintain sufficient pressure at the sprinkler heads. The existing Hunter irrigation controller is set to irrigate every three days in the watering season, and on each of these days, irrigates each zone successively beginning at 4 am for a set period of time. Therefore, this system watered inefficiently on two fronts: it could only water on a fixed schedule and the irrigation run times for each zone were not proportional to the zones' areas.

2.2 Motivation

Prior to the operation of Waterpai, our existing irrigation system was not sufficient to keep our lawn looking healthy (dark green grass) throughout the season. During hot and dry periods, the Hunter system did not provide enough water, leading to browning of the grass. Additionally, the system wasted water by watering during periods of rainfall. The Hunter controller also misallocated water, as the water provided to each zone was disproportional to the areas of the zones. These inefficiencies in the Hunter system (exacerbated by the variable Iowa weather) prompted us to design a solution that used evapotranspiration and weather data to schedule irrigation efficiently and keep our lawn green.

2.3 Design Considerations

Below is a list of design considerations we accounted for:

1. There must be a toggle between the Hunter system and our new one. One reason for this is

that when a contractor comes to winterize the system at the end of the season, we can easily switch back to the Hunter system, which they are familiar with.

2. There must be a motorized valve placed on the irrigation line. This would provide an extra layer of safety, so that irrigation to a zone will occur only when both the solenoid for that zone and the valve are open. This valve should be placed before the backflow-preventer, so that it does not affect the winterization routine.

3. The interior board (containing the Raspberry Pi) must be placed in a way so that:

   (a) It doesn't sustain over-heating during the summer, so it cannot be located in the garage.

   (b) It would be able to interface with the motorized ball valve and flow meter in the plumbing line.

   (c) It should be able to interface with a unit in the garage which can trigger the solenoid valves.

3 Implementation

3.1 Evapotranspiration Method

Evapotranspiration is the process by which water is transferred from land to the atmosphere, through evaporation from soil and transpiration from the stomata of plant leaves. Moreover, the process of evapotranspiration, along with precipitation, affects how much water is available to the roots of turf grass. As a result, evapotranspiration - the amount of water lost over a period of time, measured in inches or millimeters - is a key variable for irrigation.

Evapotranspiration depends on several weather variables: mean temperature, solar radiation, relative humidity, and wind speed. One can calculate [6] the amount of evapotranspiration precisely using the Penman-Monteith equation [17], but we opted to use reference evapotranspiration data from the IEM [4] - see Section 3.5. We define the sequence $(ET_{c,t})_{t\in\mathbb{N}}$, where $ET_{c,t}$ is the total crop evapotranspiration in the time frame $[t-1,t)$ and t is the time since initiating Waterpai, in hours (for our purposes, time 0 is the time at which we start the system). Similarly, we define the sequence $(ET_{o,t})_{t\in\mathbb{N}}$, where $ET_{o,t}$ is the total crop evapotranspiration of some reference crop, such as alfalfa, in the time frame $[t-1,t)$. Given $ET_{o,t}$, which is what is given by our weather data, we can calculate $ET_{c,t}$ by the following relation:

$$ET_{c,t} = k_c ET_{o,t},$$

where k_c is the unitless crop coefficient (which doesn't vary with time). For our purposes, the crop is our lawn grass, Kentucky blue grass. The crop coefficient of Kentucky blue grass is $k_c = .8$. Now, we define the sequence $(L_t)_{t\in\mathbb{N}}$, where L_t is the total loss of water in the time frame $[t-1,t)$, with t in hours. To determine the net hourly loss of water from our turf, we have to consider all the incoming and outgoing water from the crop root zone of the soil (see Figure 1 above). Water leaves the crop root zone through evapotranspiration, deep percolation (when water in the soil initially available to the roots of the plant travels below the soil depth of the roots), and surface runoff (which is due to the slope of the lawn). Water enters the crop root zone through precipitation and lawn irrigation. For our purposes, we for simplicity disregard deep percolation and surface runoff when calculating L_t because they are hard to measure as they depend heavily on the slope of the lawn (although we plan to account for runoff - see Section 4). However, as we discuss later, we keep the needed irrigation above zero at all times to reflect deep percolation.

The net loss of water will be the hourly precipitation, P_t (the amount of precipitation fallen in the interval $[t-1, t)$, in inches), subtracted from the hourly evapotranspiration:

$$\begin{aligned} L_t &= ET_{c,t} - P_t \\ &= .8ET_{o,t} - P_t. \end{aligned}$$

Furthermore, define $(N_t)_{t \in \mathbb{N} \cup \{0\}}$ such that N_t is the needed irrigation (in inches) at time t. Then from the equation above, we have the recursive definition

$$\begin{aligned} N_t &= \max(N_{t-1} + L_t, 0) \\ &= \max(N_{t-1} + .8ET_{o,t} - P_t, 0) \quad t \in \mathbb{N}, \quad N_0 = 0 \end{aligned}$$

Note that the needed irrigation can never fall below 0, to account for deep percolation. Once the needed irrigation minus the predicted rainfall in the next three days exceeds some threshold (which we call `replacement`), we check if watering conditions are fair (wind speeds less than 8 mph and the time is between 2 am and 4 pm). If so, we water the lawn with `replacement` inches and subsequently adjust N_t. We choose `replacement` $= .2$, as this was about the highest we could go with no runoff in any of the zones. However, we plan on creating `replacement` values for each individual zone to incorporate site characteristics.

We don't water in the later parts of the day (past 4 pm) because wet grass overnight promotes bacterial and fungal growth, which can be harmful to grass. As a result, most of our irrigations happened in the early morning.

We started running the needed irrigation calculations in early summer, when the lawn was looking in good health (which we verified using a handheld soil moisture meter), and therefore needed irrigation was roughly zero.

3.2 Electronics Overview

All of our electronics are in either our furnace board or garage board. Our flow meter and ball valve (which we added for safety) are in the irrigation line in the furnace, so the circuitry associated with these is on a board in the furnace. The furnace board also contains our Raspberry Pi, because we didn't want it to sustain temperature-related damage in the garage. The garage board (which is housed in a wood box) contains our circuitry to control the solenoids, adjacent to the existing Hunter irrigation control.

The boards in the furnace and garage communicate via the differential I2C protocol [14] through an unused telephone cable that ran from the furnace to the telephone box on the exterior wall of the garage. We drilled a couple holes through the wall, crimped a Cat 5e cable, and extended this telephone cable to be connected to the garage board.

In the garage, we retained the 24 VAC transformer that had previously powered the Hunter system. We built a rectifier using four diodes, and used three capacitors to smooth out the DC voltage. We then fed this to voltage regulator to get an output of 5V, which we used to power the garage circuit. We tapped an 18 VAC transformer to operate the ball valve, and the 3.3V from the Raspberry Pi to power the furnace circuit. In both circuits, we used I/O expanders.

3.2.1 Parts List

We used a Raspberry Pi 4 Model B (4 GB) microcontroller.

For the most part, we worked with ICs and parts that we already had bought from Digikey. Below is a list of ICs (and their part numbers) that we used in our circuitry:

1. I/O Expander: MCP23017
2. H driver: SN754410
3. Decade Counter: CD4026BE
4. Hex Inverter: CD74HC04E
5. NAND Gate: CD74HC00E

Other components/items used:

1. I2C Differential Breakout x2 - A board (using a PCA9615 IC) that functions as a differential I2C (a serial communication protocol) range extender.
2. A wide array of resistors, capacitors, diodes, voltage regulators, LEDs, switches, and relays (including an 8-channel relay board).
3. An array of jumper wires, brackets, Cat 5e and irrigation cable, RJ45 connectors, header pins, IC sockets, and heat sinks.
4. Several breadboards and PCBs.
5. A rain gauge, which is placed in our lawn, to measure on-site precipitation.

The below two images show the two boards part of the Waterpai system: one in our garage and the other in our furnace.

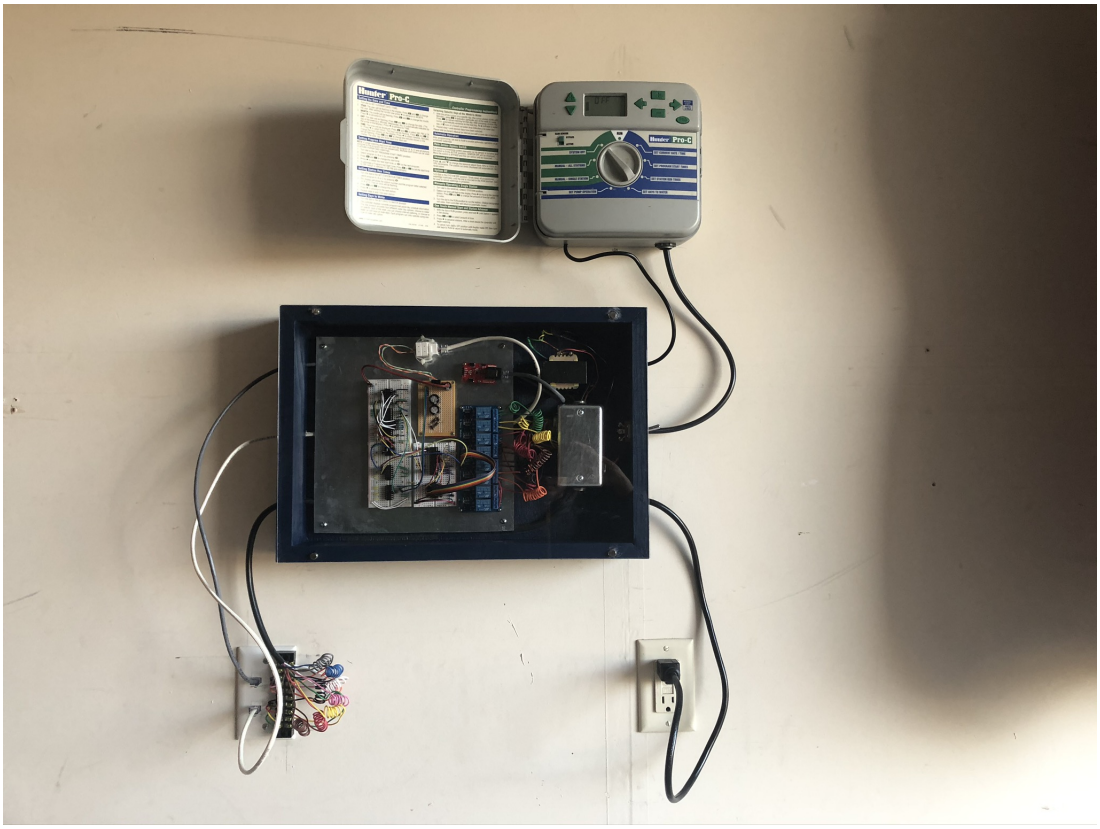


Figure 1: Garage board of Waterpai

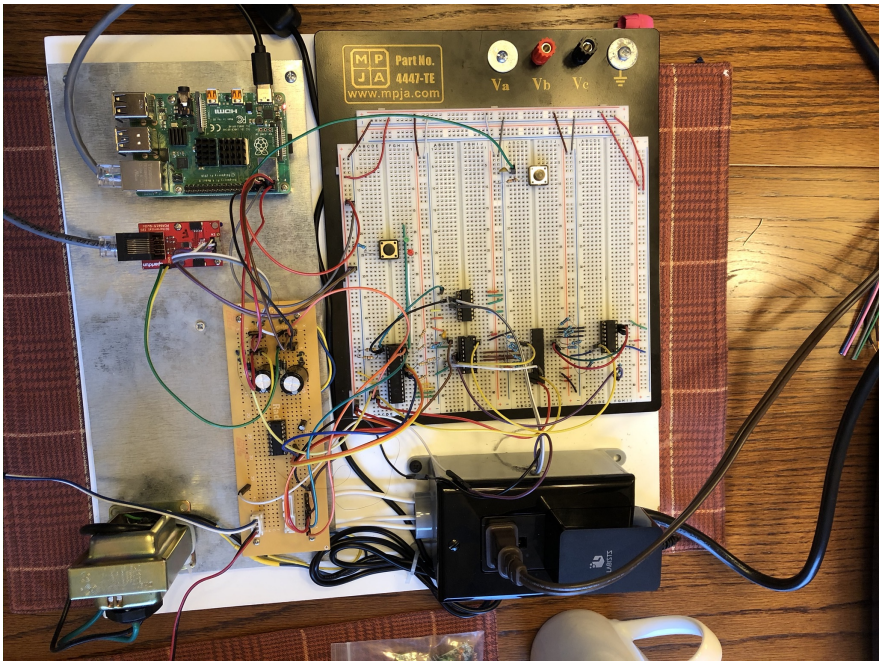


Figure 2: Furnace board of Waterpai

Below are some of the key functionalities of the garage board (more detailed explanations in the Appendix):

1. Relay Board - We used an 8-channel relay board to power on and off the individual solenoids of each irrigation zone.
2. Toggle between two systems - the toggle between the two systems is an SPDT switch embedded in the side of our garage box.
3. The decade counters [10] in the garage board count the amount of rainfall, as indicated by the rain gauge, with each increment representing .022 inches of rainfall.

Below are some of the key functionalities of the furnace board:

1. Flow meter - We had two decade counter ICs (together, they can count from 0 to 99) in the furnace board to count the number of gallons that had flowed. The flow meter outputs a square signal, with a pulse for every gallon that has flowed. We feed this signal into one of the decade counters, and then the output of the decade counters are fed as input to an I/O expander. From these inputs, our Python code deciphers the values of the decade counters to see how many gallons have flowed. We polled the I/O expander once every 5 minutes, considering that zone 2, with the fastest flow rate, used 99 gallons in around 450 seconds. We use the number of gallons to check for abnormalities in the flow rate.
2. Ball Valve - We used an H driver IC to control the closing and opening of the ball valve. One of the key features of the ball valve was that it has simple sensor outputs that indicates if the ball valve is in the fully opened or closed state.
3. We built a rectifier for our AC source here as well, to be fed to the H driver for the operation of the ball valve.

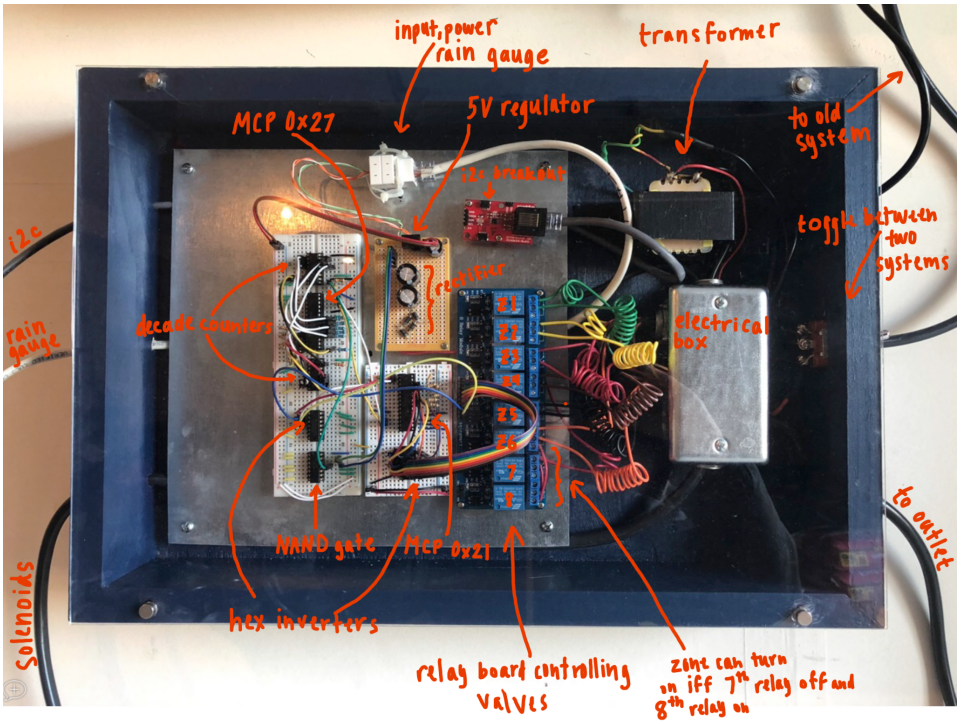


Figure 3: Annotated garage board

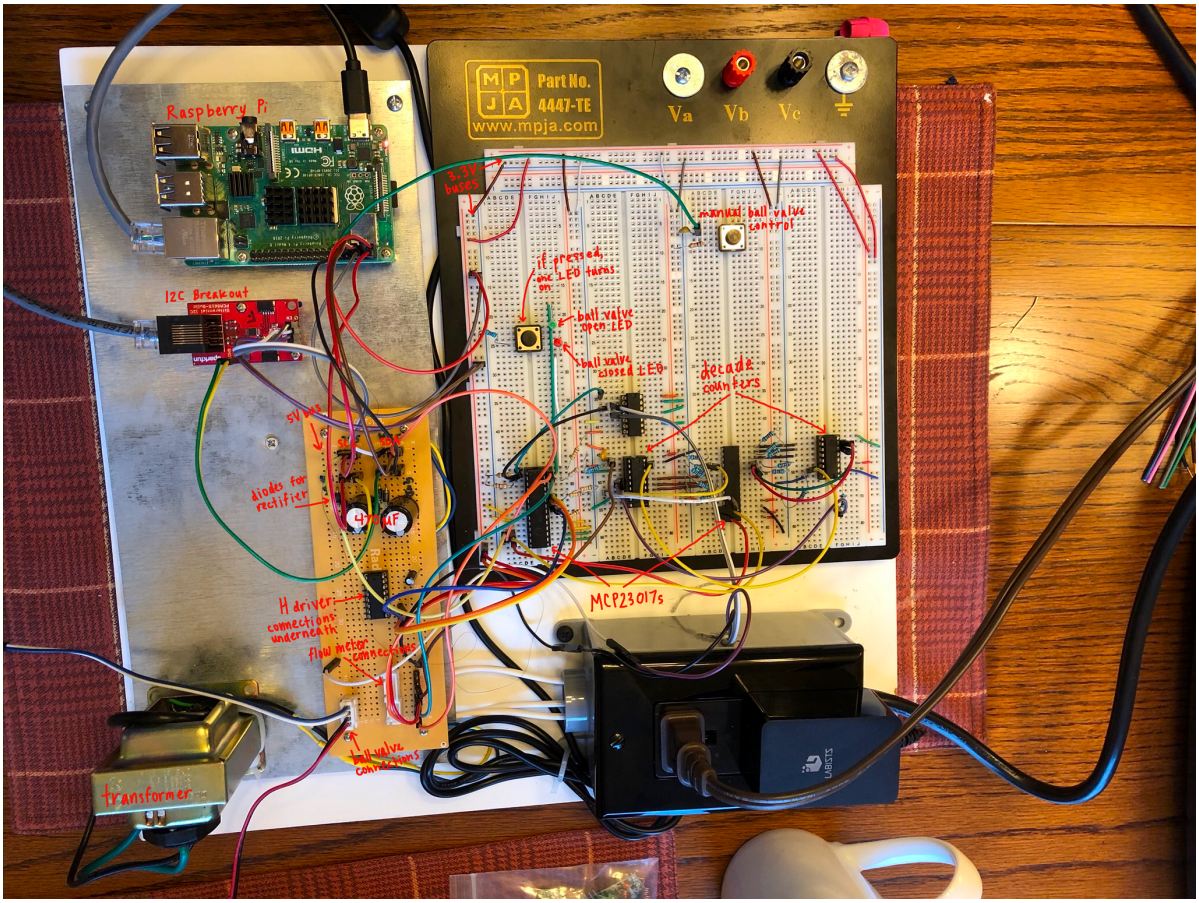


Figure 4: Annotated furnace board

3.2.2 Circuit Diagram

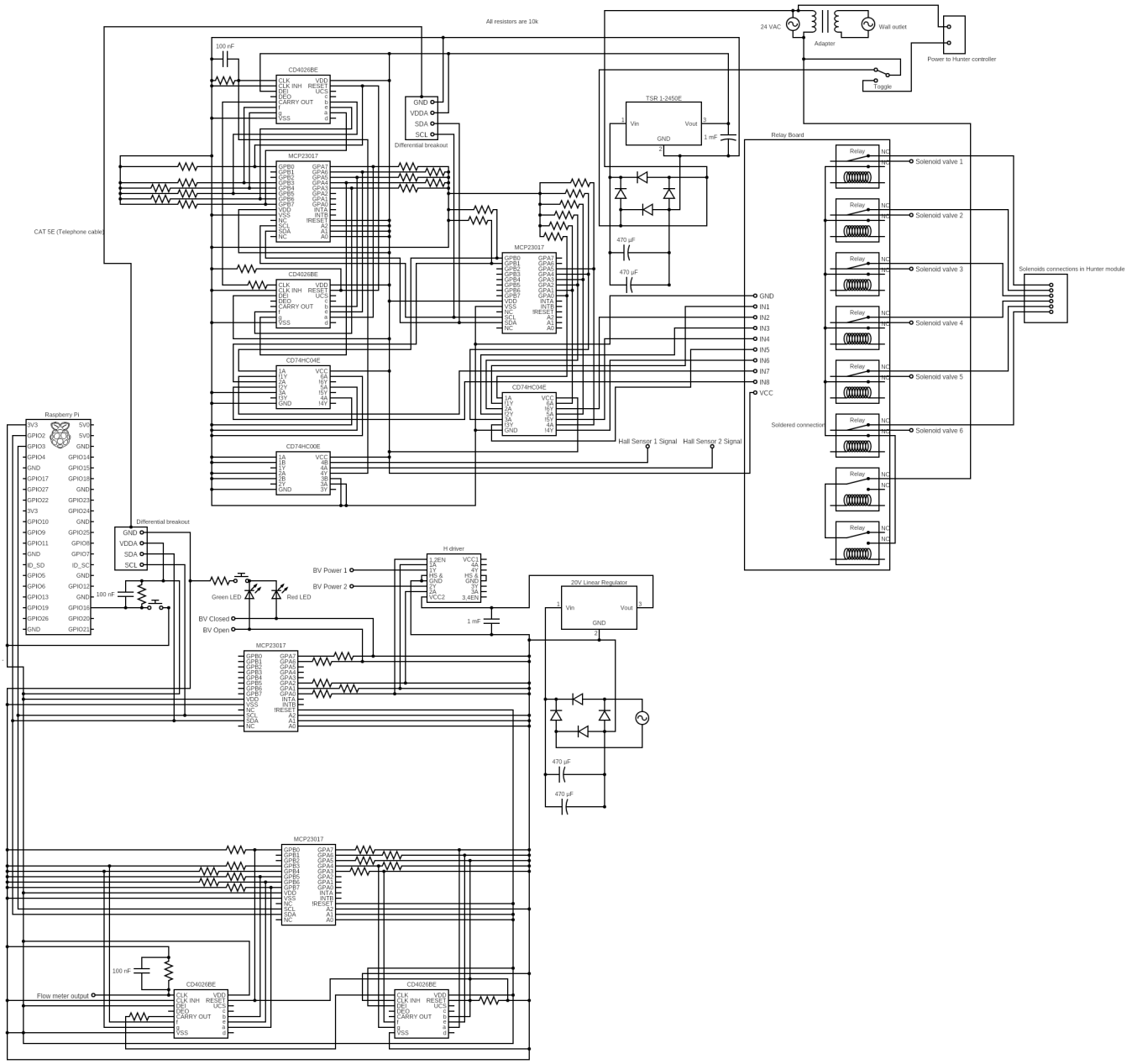


Figure 5: Circuitry for Waterpai

3.3 Housing the Electronics

We built a wooden box for our garage board using a 11" × 16.5" plywood board as the base and several planks of pine wood for the sides. We used a miter saw (angling the blade 45° to give angled cuts) to cut the sides for our box. We attached them to the base with wood screws screws, and joined the sides together using wood glue (filling the gaps with wood putty). After sanding the sides of the box, we stained it with a dark blue wood stain. We then cut an acrylic sheet for the lid of

the box, which would hold on to the box with magnets. We first placed four screws into the border of the box, so that the heads of the screws were well below the surface. We then placed four disc magnets on these screw heads, so that the top surface of the magnets were below the surface of the wood. Then we glued four more magnets onto the acrylic, so that when the lid was placed on the box, the magnets were below the surface of the wood and the acrylic was flush to the surface. To keep the magnets fastened to the acrylic, we added another four magnets on the opposite side of the acrylic, directly over the existing ones. The finished box is shown below.

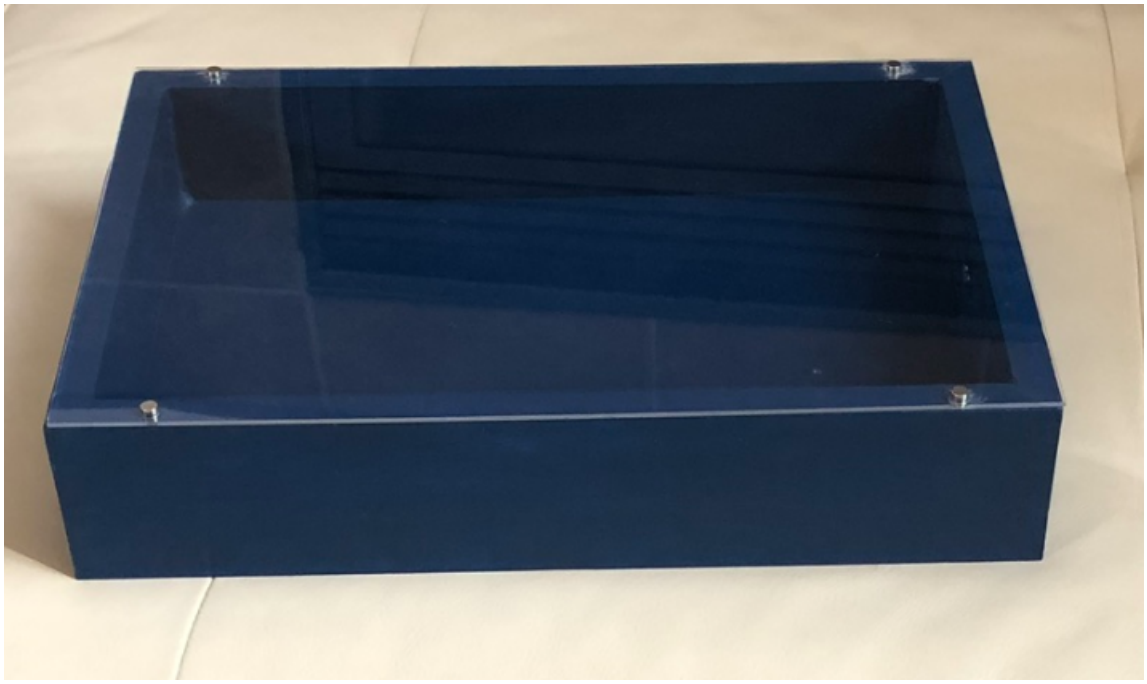


Figure 6: Finished garage box

Since the furnace board is indoors, we did not build an enclosure for it. Rather, we nailed two pieces of old trim onto the sides of the board so that we could put an anti-static sheet over the board.

3.4 Plumbing

In order to install the flow meter and the ball valve, we needed to do some plumbing. Our dad helped us with this part o ensure there would not be any leaks, as any could be disastrous. In choosing the best design for our plumbing, we had the following design considerations:

1. At the end of the season, removing any residual water in the pipes should be easy.
2. The dial on the flow meter should be in a position such that it is easy to read.
3. The ball valve and flow meter be ideally close enough so that placing the wires from each one into a dual row terminal strip connector was feasible. On the other row of the strip, we had a CAT 6 cable (stripped into its conductors and placed in the connector) and a power cable, which were ran along the ceiling to be connected to the furnace board.
4. Per the manual of the flow meter, the incoming pipe had to have length at least 10 times the diameter of the pipe (so 10 inches) and the outgoing pipe had to have length at least 5 times the diameter (so 5 inches).

Our current irrigation pipe ran on high on the wall, close to the ceiling. Simply cutting that pipe and inserting the ball valve and flow meter along this ceiling pipe would violate the first two design considerations. We bought several feet of 1"-diameter copper pipe, along with other plumbing tools and supplies. We implementing the design shown in the figure below:

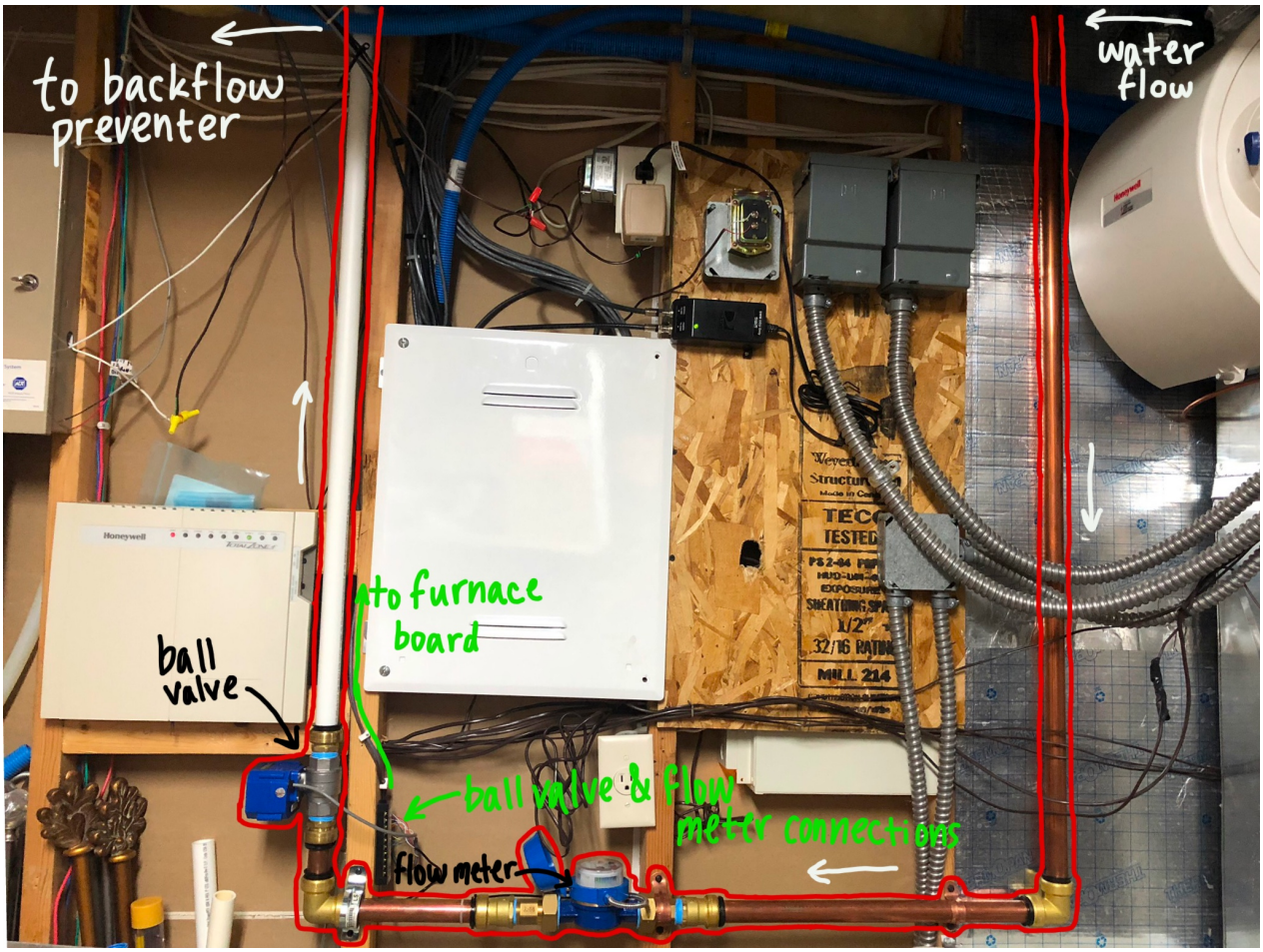


Figure 7: Plumbing

This configuration accomplished all of the above goals. Removing any residual water would be done by removing the pipe from the vertical portion of the L-joint in the bottom-left. Then, turning the bottom pipe 180 degrees would allow any excess water to flow out, and opening the ball valve would release water above the ball valve. As shown in the picture, the dual row terminal strip attached to the side of the stud contains all 8 wires from the ball valve and flow meter.

3.5 IEM API

We gather the weather data needed to implement the evapotranspiration method from Iowa State University's Iowa Environmental Mesonet (IEM) over the internet (via HTTP) using its API. The weather data is made available in the JSON format. Using Python's `json` library, we are able to extract hourly temperature, humidity, solar radiation, precipitation, wind speed, and evapotranspiration data from the IEM. Currently, we only use the hourly evapotranspiration and precipitation data.

3.6 WeatherData MariaDB Database

We store the hourly weather data from the IEM in a MariaDB (a fork of the MySql database management system) database called WeatherData. We initially stored all our data in a `csv` file, but transitioned to storing it in a SQL database.

Below is a list of the tables that are in the WeatherData SQL database.

1. `ETData` - Stores all the weather data extracted from the IEM. The table has eight fields: the weather station (CIRI4), time the data was recorded (to the second), the temperature, relative humidity, solar radiation, hourly precipitation, wind speed, and hourly evapotranspiration data recorded at the hour represented by the time field. The time field increments hourly from the start of the irrigation season (12 am on June 1), as it does for the other tables.
2. `IrrigationData` - Stores the cumulative, hourly needed irrigation (which takes into account our own irrigation) and the associated time field representing the corresponding hour.
3. `WateringHistory` - Stores the start time of each irrigation. According to this table, Waterpai watered our lawn a total of 37 times this season (starting June 1).

We update all these tables using `ISUWeatherData.py`. Storing these various data sets in SQL tables made it easier to add new entries, update entries, and access entries - especially for their use in the dashboard.

3.7 Main Steps of the Software Implementation

Our code is split into two Python programs: `ISUWeatherData.py` and `WaterLawn.py`. All our code is in the following Github repository: https://github.com/waterpaisystem/Documentation Below is a short summary of what each Python code does.

- `ISUWeatherData.py` - extracts the relevant weather data from the the IEM API and from the SQL database, carries out the evapotranspiration method (utilizing the WeatherBit API for forecast weather), initiates `WaterLawn.py` if it deems the lawn needs irrigation, and updates the tables in the SQL database.
- `WaterLawn.py` - irrigates each zone by controlling the solenoids (and opening/closing the motorized ball valve) and keeps track of the gallon consumption during irrigation for leak detection.

Note that `ISUWeatherData.py` is run every three hours (as we retrieve hourly weather data from the IEM and the run time of irrigation is between two and three hours) using a Crontab. The following steps illustrate our software implementation of the evapotranspiration method and irrigation. The first four steps are executed in `ISUWeatherData.py`, and the rest in `WaterLawn.py`.

1. We retrieve the hourly weather data (temperature, humidity, solar radiation, precipitation, wind speed, and evapotranspiration) from the IEM API that is not already in the `ETData` table, and add this new data to the table.
2. Using this new weather data and the most recently recorded needed irrigation value (from the last row of the `IrrigationData` table), we calculate the current needed irrigation and add this value to the `IrrigationData` table.

3. Using the WeatherBit's hourly/daily forecast APIs, we determined the future 3-day precipitation and the current wind speed. Using these values, we determined if the lawn needed to be irrigated using the following three conditions:

   - The current needed irrigation minus the predicted rainfall in the next three days is above a certain threshold (which we call `replacement` and set to .2).
   - The current wind speed is less than 8 mph. This constraint was imposed because heavy winds disrupt sprinkler irrigation and lead to an uneven distribution of water.
   - The time of day is between 2 am and 4 pm. This constraint was imposed because irrigation in the late afternoon/night can promote harmful fungal growth on the grass.

4. If it is determined that the lawn needs to be irrigated, we first update the most recent value of needed irrigation by subtracting `replacement` from it (this is not a new entry in the table). We then insert the current time in the `WateringHistory` table, and run `WaterLawn.py` to initiate the irrigation.

5. In `WaterLawn.py`, the zones are watered in the following order:

Zone	Runtime (seconds)
2	695
3	866
1	348
4	1450
5	2224
6	978
2	695
3	866

Figure 8: The schedule WaterLawn.py that follows in one irrigation cycle.

   The total irrigation run times for each zone is calculated using the zone flow rates and areas. Note that zones 2 and 3 are irrigated in two periods as these zones are most prone to run off due to their sloped terrain.

6. The following steps are executed during each time a zone is irrigated with a specified run time.

   - The ball valve is opened and the solenoid is opened soon after.
   - If no water flows in the first 20 seconds of the zone, then the schedule is modified so that it no longer contains zones that have already been watered, and `WaterLawn.py` is restarted. The program cannot be restarted more than 10 times (to prevent an infinite loop); it will be quit after the 11th restart attempt.
   - If the flow rate is abnormally low or high after the first 20 seconds, the program is quit and an email alert is sent.
   - After the zone has been irrigated for the specified amount of time, the solenoid is closed and soon after, so is the ball valve.

7. Once all zones have been properly irrigation, the schedule is restored for the next irrigation cycle.

3.8 Web Server Dashboard

We used a Apache web server to display some useful information from our database like the status of irrigation and a couple of graphs that display weather data trends. Our PHP code is contained in `index.php` and `userfeedback.php` - see our Github repository: https://github.com/

waterpaisystem/Documentation

The dashboard allows Waterpai to take in user input. Currently, it asks the user how the lawn is looking, and based on the response, adjusts the needed irrigation field in the last row of the `IrrigationData` table appropriately. For example, if the user indicates that the lawn is "OK (a few, small brown spots)", then the needed irrigation field is increased by .2, prompting an additional watering. We plan on adding more user input, such as zone max watering times before runoff occurs and evapotranspiration scale factors for each zone to somewhat take into account shade and varying terrain. Below are some pictures of the dashboard.

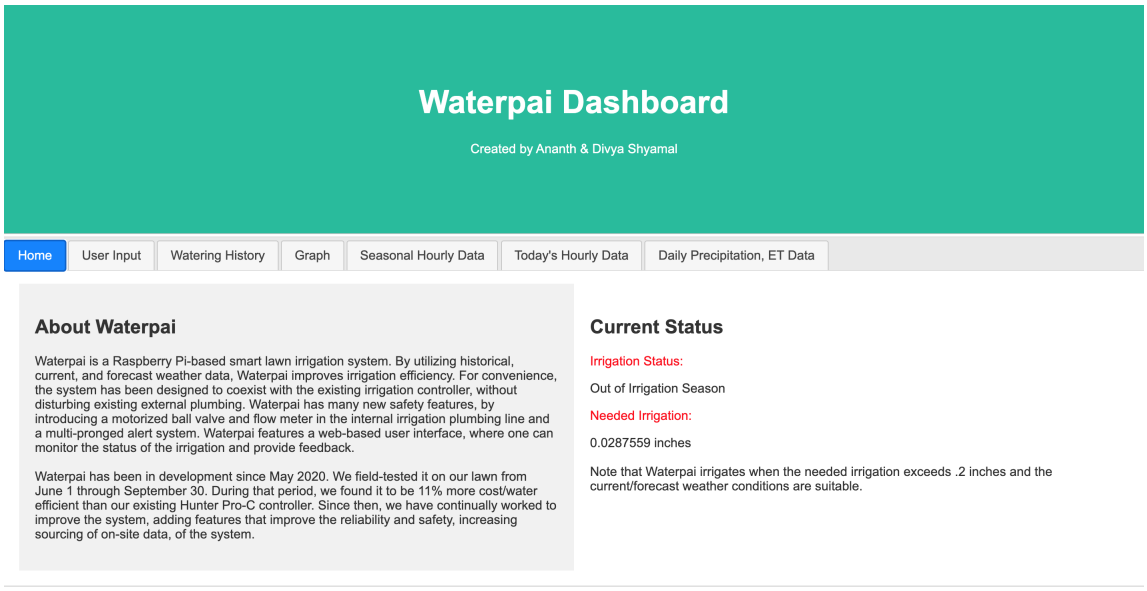


Figure 9: The home page of our dashboard, which contains a brief overview of Waterpai and the current status of irrigation.

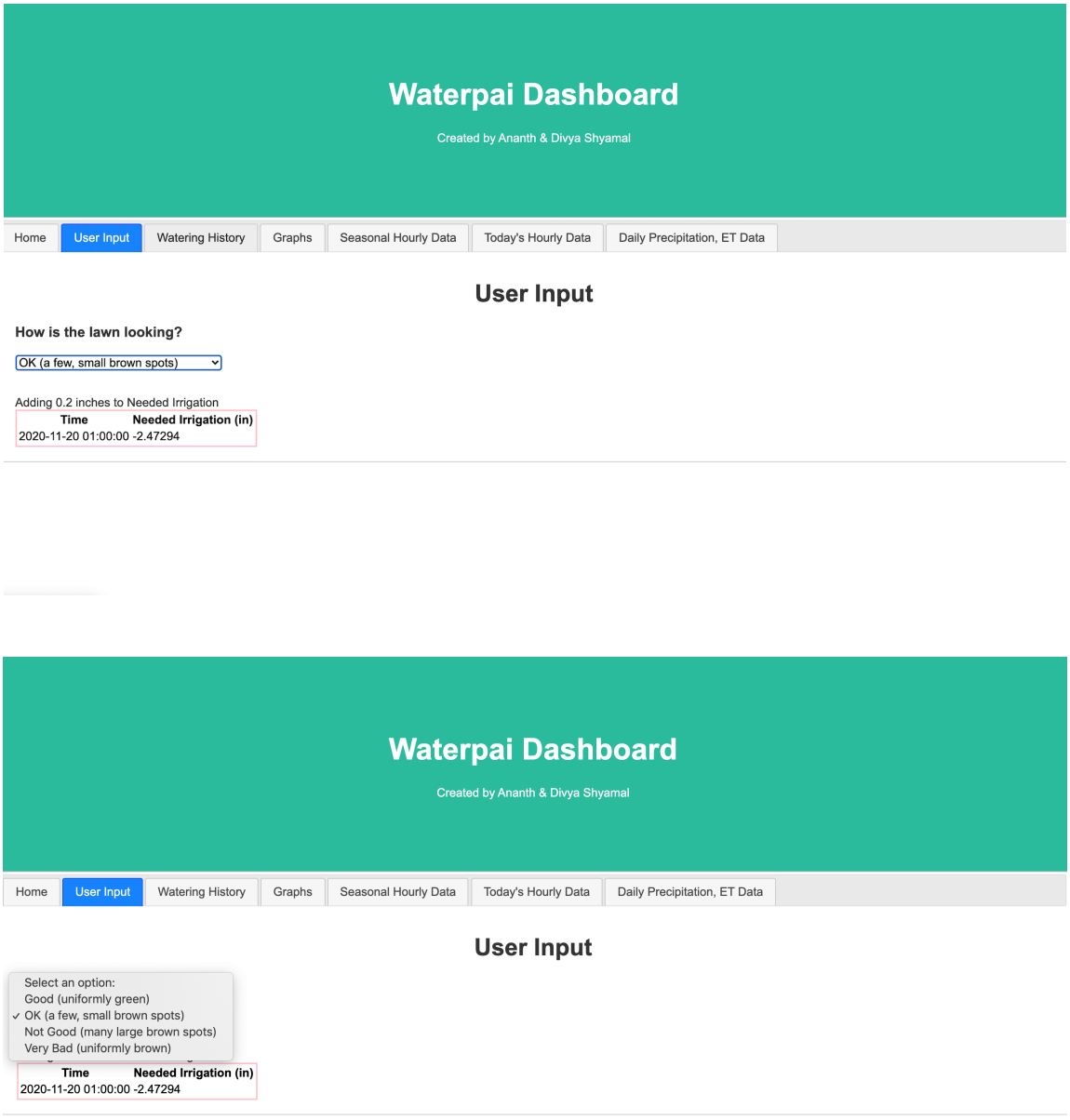
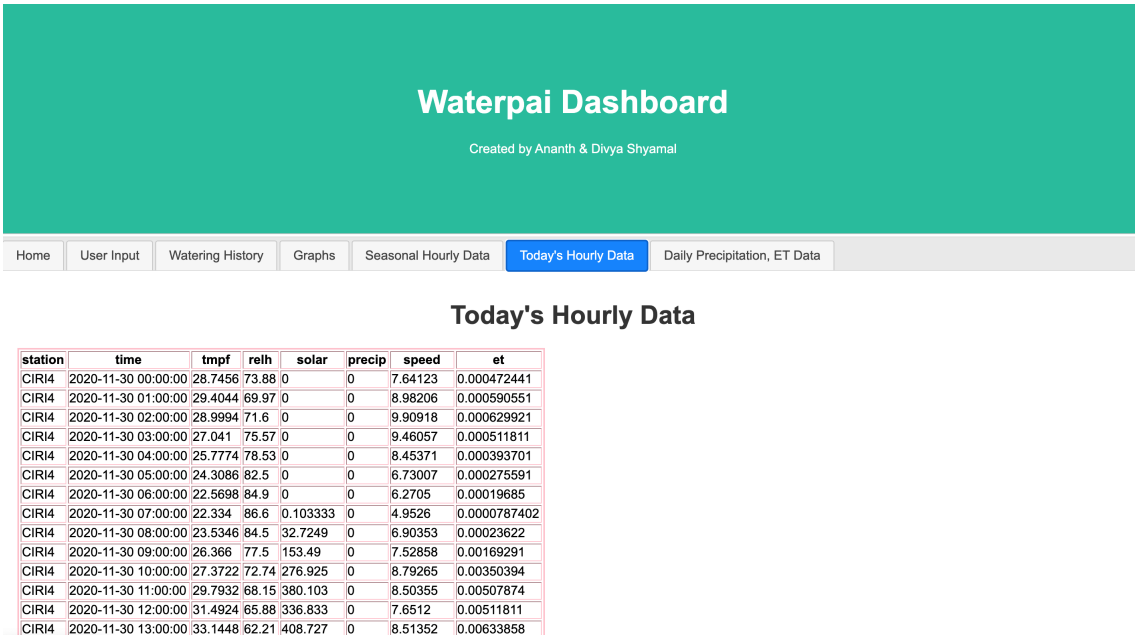
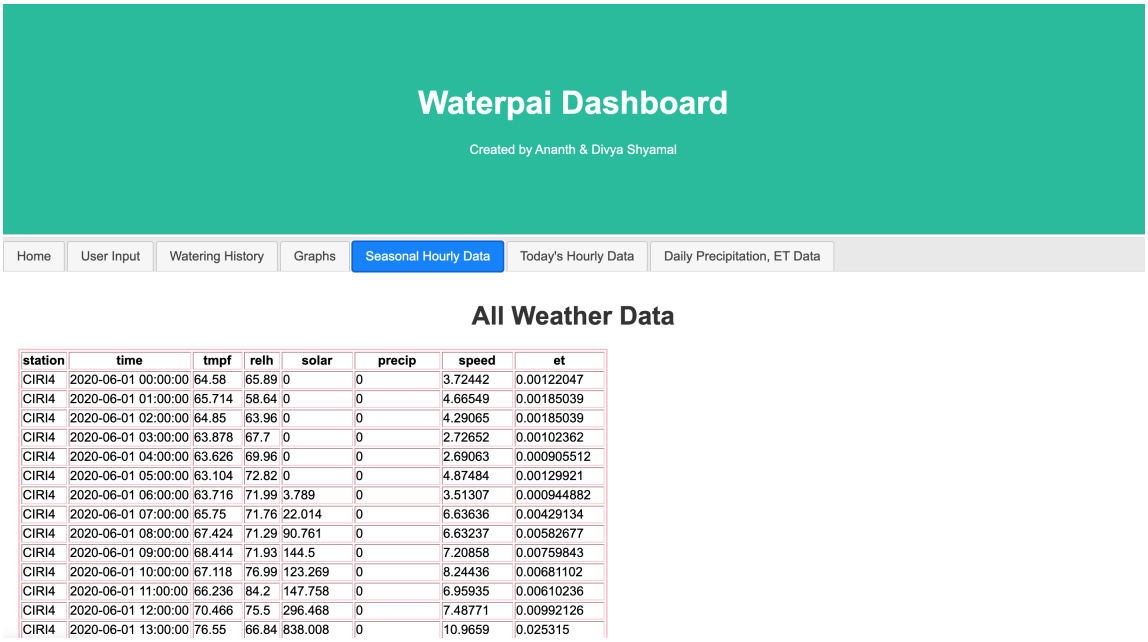


Figure 10: User input page on our dashboard where one can change needed irrigation by indicating how the lawn looks.



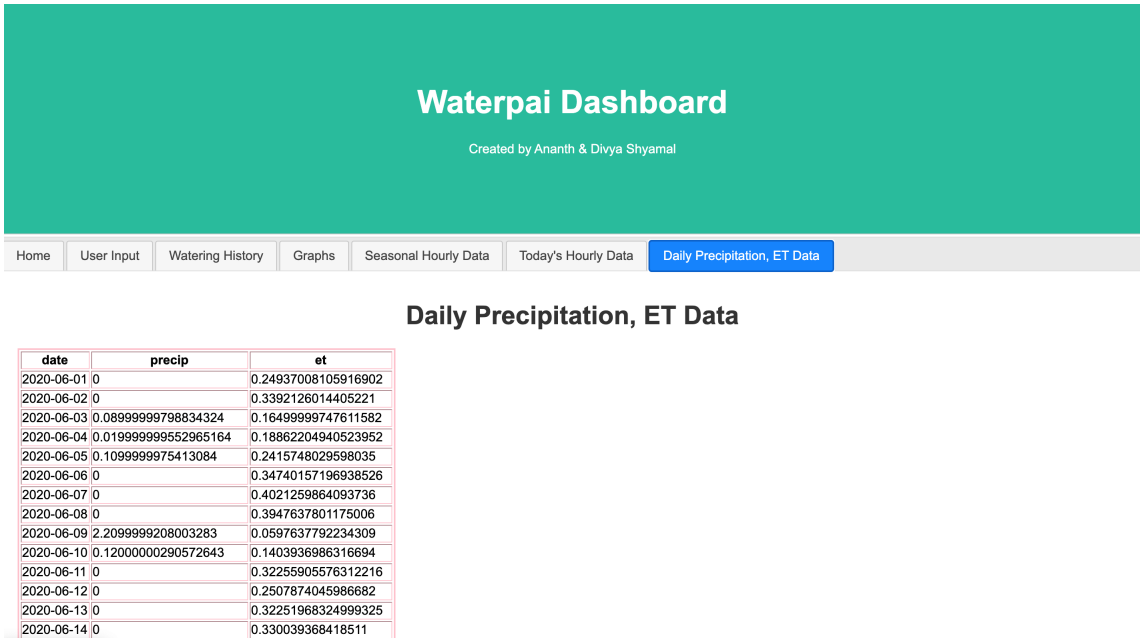
station	time	tmpf	relh	solar	precip	speed	et
CIRI4	2020-11-30 00:00:00	28.7456	73.88	0	0	7.64123	0.000472441
CIRI4	2020-11-30 01:00:00	29.4044	69.97	0	0	8.98206	0.000590551
CIRI4	2020-11-30 02:00:00	28.9994	71.6	0	0	9.90918	0.000629921
CIRI4	2020-11-30 03:00:00	27.041	75.57	0	0	9.46057	0.000511811
CIRI4	2020-11-30 04:00:00	25.7774	78.53	0	0	8.45371	0.000393701
CIRI4	2020-11-30 05:00:00	24.3086	82.5	0	0	6.73007	0.000275591
CIRI4	2020-11-30 06:00:00	22.5698	84.9	0	0	6.2705	0.00019685
CIRI4	2020-11-30 07:00:00	22.334	86.6	0.103333	0	4.9526	0.0000787402
CIRI4	2020-11-30 08:00:00	23.5346	84.5	32.7249	0	6.90353	0.00023622
CIRI4	2020-11-30 09:00:00	26.366	77.5	153.49	0	7.52858	0.00169291
CIRI4	2020-11-30 10:00:00	27.3722	72.74	276.925	0	8.79265	0.00350394
CIRI4	2020-11-30 11:00:00	29.7932	68.15	380.103	0	8.50355	0.00507874
CIRI4	2020-11-30 12:00:00	31.4924	65.88	336.833	0	7.6512	0.00511811
CIRI4	2020-11-30 13:00:00	33.1448	62.21	408.727	0	8.51352	0.00633858

Figure 11: Tables on our dashboard showing today's hourly weather data.



station	time	tmpf	relh	solar	precip	speed	et
CIRI4	2020-06-01 00:00:00	64.58	65.89	0	0	3.72442	0.00122047
CIRI4	2020-06-01 01:00:00	65.714	58.64	0	0	4.66549	0.00185039
CIRI4	2020-06-01 02:00:00	64.85	63.96	0	0	4.29065	0.00185039
CIRI4	2020-06-01 03:00:00	63.878	67.7	0	0	2.72652	0.00102362
CIRI4	2020-06-01 04:00:00	63.626	69.96	0	0	2.69063	0.000905512
CIRI4	2020-06-01 05:00:00	63.104	72.82	0	0	4.87484	0.00129921
CIRI4	2020-06-01 06:00:00	63.716	71.99	3.789	0	3.51307	0.000944882
CIRI4	2020-06-01 07:00:00	65.75	71.76	22.014	0	6.63636	0.00429134
CIRI4	2020-06-01 08:00:00	67.424	71.29	90.761	0	6.63237	0.00582677
CIRI4	2020-06-01 09:00:00	68.414	71.93	144.5	0	7.20858	0.00759843
CIRI4	2020-06-01 10:00:00	67.118	76.99	123.269	0	8.24436	0.00681102
CIRI4	2020-06-01 11:00:00	66.236	84.2	147.758	0	6.95935	0.00610236
CIRI4	2020-06-01 12:00:00	70.466	75.5	296.468	0	7.48771	0.00992126
CIRI4	2020-06-01 13:00:00	76.55	66.84	838.008	0	10.9659	0.025315

Figure 12: Tables on our dashboard showing hourly weather data over the season.



date	precip	et
2020-06-01	0	0.24937008105916902
2020-06-02	0	0.3392126014405221
2020-06-03	0.08999999798834324	0.16499999747611582
2020-06-04	0.019999999552965164	0.18862204940523952
2020-06-05	0.1099999975413084	0.2415748029598035
2020-06-06	0	0.34740157196938526
2020-06-07	0	0.4021259864093736
2020-06-08	0	0.3947637801175006
2020-06-09	2.2099999208003283	0.0597637792234309
2020-06-10	0.12000000290572643	0.1403936986316694
2020-06-11	0	0.32255905576312216
2020-06-12	0	0.2507874045986682
2020-06-13	0	0.32251968324999325
2020-06-14	0	0.330039368418511

Figure 13: Tables on our dashboard showing daily precipitation and evapotranspiration over the season.

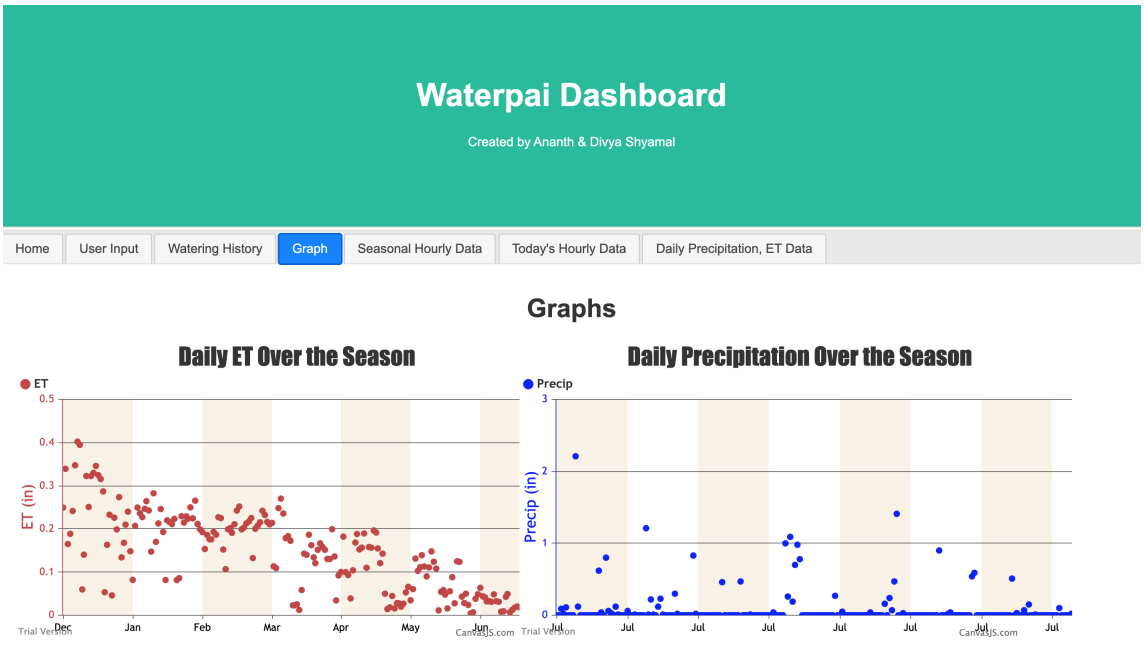


Figure 14: Graphs on our dashboard showing how daily precipitation/evapotranspiration change over the season.

3.9 Safety Measures

3.9.1 Leak Detection

Using the decade counters to count the number of gallons that have flowed, we can detect irregularities in flow rate. A flow rate that is significantly higher than usual is indicative of a leak. If this is detected, the irrigation will cease and an email is sent to us, alerting us of the problem. A flow rate that is significantly lower than usual would also cause the program to cease, and we would be alerted by email.

3.9.2 Ball Valve

We added a motorized ball valve to the plumbing line in the furnace as a safety measure against accidental irrigation, preventing irrigation if, for some reason, the relays were accidentally triggered.

3.9.3 Relay Board

In addition to the ball valve as a safety measure, we incorporated the 7th and 8th relays on the relay board to decrease the chance of an unintended irrigation cycle. We wired the board such that the other 6 relays, which are connected to the solenoids, can only be turned on if and only if the 7th relay is off and the 8th relay is on.

3.9.4 Alert System

Using the `yagmail` Python library, we sent emails to ourselves on various occasions, including if the irrigation system had been programatically turned on, if the flow rate was unusually high or low, or if there was an error in populating the database (likely due to possible duplicate entries in the data extracted from the IEM).

4 Waterpai vs the Existing System

4.1 Runtimes and Gallon Consumption

As mentioned in Section 2, the existing Hunter controller could only irrigate based on a fixed schedule and the run times for each zone in an irrigation cycle were not proportional to their areas. Waterpai improves on both of these fronts by utilizing weather data (making the irrigation schedule irregular) and a flow meter (so that its zone run times were proportional to the zones' areas). See the table below:

Zone	Hunter Runtime (min)	Waterpai's Runtime (min:sec)
1	15	5:48
2	30	23:10
3	20	28:50
4	30	24:10
5	30	37:04
6	25	16:18

Figure 15: The run times for each of the six zones in our irrigation system in each system.

To calculate the amount of water (in gallons) applied to each zone during irrigation, we needed the areas and flow rates (in gallons per second) of each zone. We determined the zone areas by using our county assessor's mapping tool and the flow rates by installing a flow meter. These values are shown in the table below.

Zone	Area (sq ft)	Flow rate (gal/sec)
1	551	.197
2	2447	.219
3	1737	.125
4	2496	.215
5	3787	.212
6	987	.122

Figure 16: The area and flow rate for each of the six zones in our irrigation system.

These values enabled us to calculate the gallon consumption of both systems, shown below.

Zone	Gallons used by Waterpai	Gallons used by Hunter
1	2742	7092
2	12176	15768
3	8660	6000
4	12470	15480
5	18860	15264
6	4773	7324

Figure 17: The number of gallons used on each zone during the season.

4.2 Cost Comparison Calculations

One reason we created Waterpai was to lower the cost of irrigation. The existing Hunter irrigation system was programmed to water every 3 days, with the run times for each zone shown in the figure above. We irrigated from June 1 to September 30, 2020, so the existing system would've irrigated about 40 times. Using the flow rates and Hunter run times for each zone (see the figure above), the total number of gallons of water used by the existing system is given by

$$40 \cdot 60\,(15 \cdot .197 + 30 \cdot .219 + 20 \cdot .125 + 30 \cdot .215 + 30 \cdot .212 + 25 \cdot .122) \approx 66924 \text{ gallons.}$$

From the `WateringHistory` SQL table in the `WeatherData` MariaDB Database (see section 7.2), Waterpai irrigated 37 times over the season. Now, using the flow rates and Waterpai's run times for each zone, the total number of gallons of water used by Waterpai is given by

$$40\,(348 \cdot .197 + 1390 \cdot .219 + 1732 \cdot .125 + 1450 \cdot .215 + 2224 \cdot .212 + 978 \cdot .122) \approx 59681 \text{ gallons.}$$

Thus, Waterpai is approximately 11% more efficient than the existing Hunter system.

Our city (Coralville, Iowa) charges \$1.80 for every 100 cubic feet of water consumed and \$4.40 for every 100 cubic feet of water in the sewage. Thus, our irrigation costed \$6.20 for every 100 cubic feet of water used. Using the conversion 1 cubic foot $= 7.48052$ gallons, the total water costs of the existing system is \$554.68 as compared to \$494.91 for Waterpai - a savings of \$59.77 per year.

5 Further Improvements

There are a number of improvements we plan to make to Waterpai this spring and summer. The main focus will be on improving our watering algorithm using site characteristics and machine learning. This is Phase II of developing Waterpai.

5.1 Site Characteristics

There are many ways our irrigation watering methodology, which was based on the simple evapotranspiration calculation, can be refined and improved. Our evapotranspiration method assumes the following characteristics of the lawn, which can lead to sub-optimal irrigation.

1. The lawn is flat and is uniformly of the same soil type.
2. There are no shadows on the lawn (e.g. from trees and houses).
3. There are no tree roots under the lawn.
4. The lawn is covered evenly by the sprinklers.

Essentially, the evapotranspiration method assumes the terrain to be uniform, which clearly isn't a characteristic of any lawn. To remedy this, we plan on having specific `replacement` values (see Section 3.7) for each zone, taking into account the slope of the lawn. More sloped areas should have shorter, but more frequent waterings than flatter areas in order to minimize runoff. Additionally, through our own observations, we noticed how areas of the lawn near trees required different amounts of water than other parts of the lawn. Specifically, areas underneath trees generally had a lower needed irrigation, while areas which were outside the shade provided by trees but contained tree roots had a higher needed irrigation.

We plan on using soil moisture sensors to help gauge how soil moisture is affected by the lawn terrain. This will also help us take into account deep percolation, a factor we omitted in the evapotranspiration method. To accommodate this, we plan on using the soil moisture method - a variant of the evapotranspiration method we are currently using that determines when to water based on the soil moisture content in the soil. For more details, look at the Appendix.

We also plan to take into account runoff by making the max watering times for each zone before runoff a user input to the dashboard.

5.2 Suitable Watering Conditions

As of right now, Waterpai only irrigates if the cumulative forecast 3-day precipitation (from the WeatherBit API) is less enough so that the 3-day forecast needed irrigation is greater than the threshold of .2 inches. We plan on refining this future precipitation constraint by simulating our algorithm on historical data using various such precipitation constraints.

5.3 Counting by Gallons Instead of Minutes

Currently, given the number of gallons that the lawn needs, we are running irrigation cycles for the corresponding amount of time (calculated with the flow rate of each zone). Although this has proved to be sufficiently accurate (we have encountered at most 2% error), we can directly count the number of gallons that have flowed. We plan to implement this in the following way: if we need x

gallons, then we start the value on the counter chips at $100 - (x \mod 100)$ (which we will do with the help of a multiplexer and a GPIO pin from the Raspberry Pi). We feed the carry out of the decade counter corresponding to the ten's digit to a GPIO pin on the Raspberry Pi. On the $\lceil \frac{x}{100} \rceil$th callback to the GPIO, the watering for that zone should stop. See the example in section A.3 of the Appendix.

5.4 Adding Buzzer to Alert System

We plan on adding an electronic buzzer to our furnace board circuitry, which we will utilize in conjunction with the email alert system.

6 Appendix

A Electronics

A.1 Tools used

Tools used:

1. An array of drills, screwdrivers, hammers, saws, wrenches, pliers, levels, measuring tapes/rulers, a multimeter, and stud finders.
2. Soldering tools: various lead solders, flux, solder braid, solder suction gun, and soldering heat sink.
3. Plumbing tools: plumbing wrenches, thread seal tape, deburring tool, and copper/PEX pipe cutters.
4. Woodworking tools: miter saw, RotoZip, wood glue, sandpaper, clamps, wood filler, wood semi-transparent stain

A.1.1 Oscilloscope

Our two-channel, digital oscilloscope also came in handy throughout the project when debugging. For instance, when the decade counters didn't seem to counting, we used an oscilloscope to view the voltage it was outputting. In addition to debugging, we used the oscilloscope to read the signal from the flow meter, and by using the cursor function, we determined the flow rate for each zone. Finally, we used the oscilloscope to view the differential I2C signals and help us understand the formats of the signals.

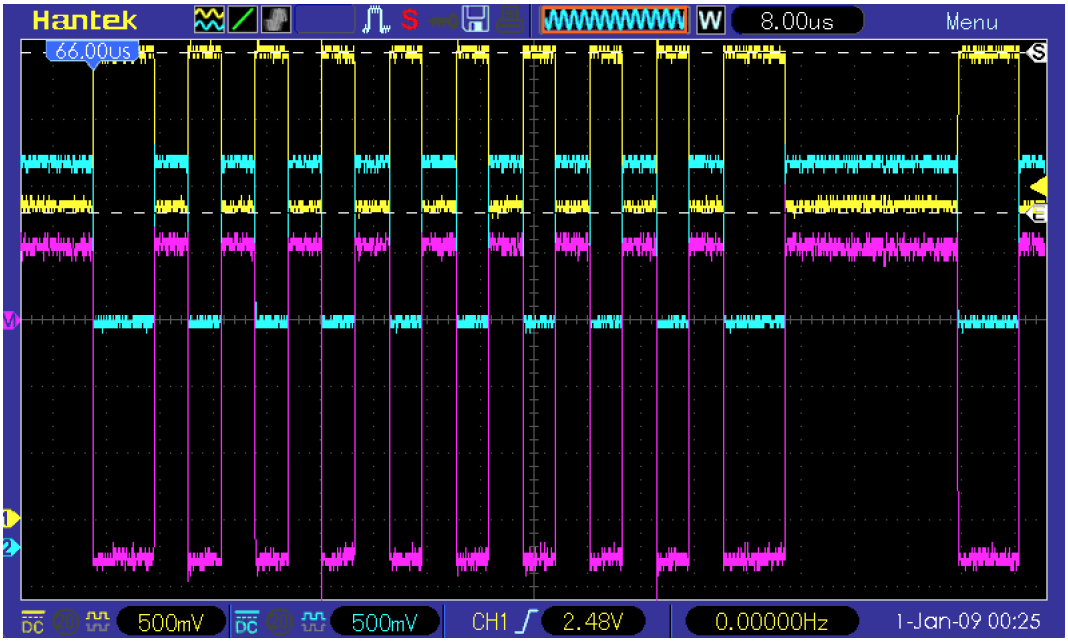


Figure 18: Differential SCL Signal

A.1.2 Soldering iron station

We figured that some parts of our circuit, especially the high-voltage parts, were much better soldered to a PCB than at risk of being pulled from a breadboard. Specifically, we soldered the voltage rectifier circuit in both boards and many of the outgoing connections (header pins for SDA/SCL lines and 5V/ground, as well as connections to the ball valve and flow meter). We also desoldered parts from old components that would be useful for our project (like a 3-state switch and a USB head).

A.2 Decade Counter

We used two pairs of decade counters, with each pair being able to count up to 99, to count the number of gallons that had flowed through the flow meter in the furnace and to count the pulses from the rain gauge. One decade counter indicated the ten's digit of the count, and the other indicated the one's digit of the count. We determined that feeding the signal from the flow meter directly as input to an MCP was quite inefficient. The output from the flow meter is a square wave with each pulse having width approximately .5 seconds. This means that we would have to poll the MCP every .5 seconds to ensure that we hadn't missed a pulse. Furthermore, if the pulse width is not exactly .5 seconds, we run the risk of either double-counting or missing that pulse. So using an MCP to count the number of gallons is inefficient and potentially inaccurate.

On the other hand, if each gallon takes 4.55 seconds to flow (which happens to be the fastest flow rate of any of our zones), then we would only have to poll the decade counters about every $99 * 4.55 = 450.45$ seconds, which is a vast improvement from polling every .5 seconds.

A decade counter is designed to display the count on a seven-segment display. The following table indicates how each digit is represented on a seven-segment display (where the segments are labelled with the letters a through g):

	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Figure 19: Representation of each digit on a seven-segment display.

To determine which digit the counter is at, we could have all seven segments as an input to an MCP chip and from there determine the digit we are at using the table above. However, we realized that we didn't need to use seven inputs on the MCP to determine which digit we were at: it could be done with the 5 segments a, b, e, f, and g (see table above).

A.3 Gallon Counter Example

If we want to irrigate a zone with 240 gallons, we first set the value on the decade counters to 60 (which would be done by first resetting the counters and then feeding it signal with 60 pulses, which would come form a GPIO pin). Then, we start the irrigation. On the $\lceil \frac{240}{100} \rceil = 3$rd carryout of the first decade counter (so when the count is at 300 gallons), the irrigation will stop, after 240 gallons - as desired.

A.4 Rain Gauge

After reviewing the precipitation data from the IEM, we realized it was reasonably inaccurate - this was somewhat expected since rain is highly localized and the nearest IEM weather station was roughly 25 miles away from our house. Consequently, we decided to purchase a rain gauge to put in our backyard, so our precipitation data would be as relevant and accurate as possible. We have yet to write code to incorporate this data into our system, but we have installed all the appropriate hardware.

The rain gauge we purchased transmits via 433Mhz data to an LCD monitor and does not provide us a signal output. We were able to work around this. Our specific rain gauge is a double-spooned tipping bucket, as opposed to being single-spooned. The spoon tipping to either side indicates .022 inches of rainfall. We measured this experimentally by putting different amounts of water through the device and counting how many clicks there were, with each click corresponding to the tipping of a spoon. We mounted two magnetic sensors on the gauge: one under each spoon. The sensors were designed so that when the magnetic flux density around it was sufficiently large, their signal pins were pulled down to LOW. Then, we added a magnet to the bottom of each spoon (with the use of epoxy glue). So when the double-spoon was tipped to the left (so that the left spoon touches the bottom and the right spoon at its highest point), the sensor on the left would output LOW, while the sensor on the right would output HIGH. When the double-spoon is in the process of tipping from one side to the other (which is very brief), both sensors would output HIGH. To summarize, we have the following table:

Left sensor	Right sensor	State
LOW	LOW	Not possible
LOW	HIGH	Tipped to the left
HIGH	LOW	Tipped to the right
HIGH	HIGH	In transition

Figure 20: Outputs of the Hall-effect sensors for specific states of the double-spoon system.

Note that we could not just place both outputs as the clock input to our decade counter, as this would create a short between the two sensors when the bucket is tipped. We decided to use a gate that would take in the two sensor outputs and return a single output to the decade counter. We used a NAND gate (an XOR gate would work too), which would return $HIGH$ only when the bucket was tipped, telling us that .022 inches had fallen.

Left sensor	Right sensor	Output
LOW	LOW	HIGH (but will never occur)
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW

Figure 21: Truth table for a NAND gate.

Therefore, the NAND gate will output HIGH whenever the bucket is tipped, and the the number of LOW-to-HIGH transitions from the output of the NAND gate will match with the number of .022-inch increments.

A.5 Ball Valve

Prior to irrigating each zone, we opened the ball valve, and at the end of each zone watering, we closed it. Besides being a safety measure, the ball valve reduced the water surge that happened at the beginning of starting each zone - we waited a couple seconds after we opened the ball valve to open the soleniod valve, and the water surge was largely mitigated. This led to a constant flow rate, instead of a large dip in flow rate at the beginning.

Our ball valve has 5 wires: two for the input voltage (depending on which input receives the positive voltage, the valve will open or close) and three that would signal whether the ball valve was open or close. In addition, we have three wires from the flow meter: two that gave the output (so we only needed one of them), and a ground wire. We used a CAT 5E cable to connect the three input wires from the ball valve and the wires from the flow meter to the circuit (we ran this wire along the ceiling), and then we used a thicker cable for the power to the ball valve.

A.6 H driver

We used an H driver to operate the ball valve. The following table shows the inputs to the H driver (which we output from the I/O expander) to open or close the ball valve.

1,2 Enable	1A	2A	Ball Valve action
HIGH	HIGH	LOW	Opening
HIGH	LOW	HIGH	Closing

Table 1: H driver operation

A.7 Callback for manual control of the ball valve

We integrated two push-button switches on the furnace board to see the state of the ball valve through red and green LEDs (indicating whether it is open or closed) and to also change the state of the ball valve (open if it is closed and vice versa). We enable this by connecting a switch to a GPIO pin of the Raspberry Pi and using a callback function. In addition, we disable the callback function when the system is irrigating.

B Soil Moisture Method

This summer, we plan on planting soil moisture sensors around our lawn to better take into account the terrain. Using these soil moisture sensors, we will be able to determine when to water using the soil moisture method.

Soil moisture is measured through two variables: soil water potential (which describes the movement of water to regions of higher potential) and the volumetric soil water content . Soil moisture sensors measure water content by using capacitance, taking advantage of the polarity of water. Next, we define a few terms:

- Field Capacity (θ_{FC}) - The water content in the soil a couple days after saturation.
- Permanent Wilting Point (θ_{PWP}) - The water content in the soil at which the plant wilts and cannot recover.
- Maximum Allowable Depletion (MAD) - the maximum percent of the water content in the root zone that the plant can extract. $MAD \approx 50\%$ for lawn grass.

So the amount of water that can be removed from the soil before the crop becomes stressed (the allowable depletion) is given by $.5 \cdot (\theta_{FC} - \theta_{PWP})$. Therefore,

$$\theta_{LL} = \theta_{FC} - .5 \cdot (\theta_{FC} - \theta_{PWP}),$$

where θ_{LL} denotes the lowest we want the soil water content to get before irrigating. Ideally, we irrigate when the soil moisture content goes below θ_{LL}, and we want to irrigate so that the soil water content remains below the field capacity - otherwise, we would be wasting water and contributing to surface runoff.

Now since the root depth of lawn grass is typically around 12 inches [1], we irrigate the lawn with

$$12\,(\theta_{FC} - \theta_{LL}) = 6\,(\theta_{FC} - \theta_{PWP})$$

inches of water.

C Roles

The two of us worked together on this project, with Divya a bit more hardware-oriented and Ananth a bit more software-oriented. We were mentored by our dad throughout this project, especially when plumbing and purchasing electronic components.

References

[1] A. A. Andales, J. L. Chávez, T. A. B. (2019). Irrigation Scheduling: The Water Balance Approach. https://extension.colostate.edu/docs/pubs/crops/04707.pdf".

[2] A. Murthy, C. Green, R. Stoleru, S. Bhunia, C. Swanson, T. Chaspari (2019). Machine Learning-based Irrigation Control Optimization.

[3] Digikey (2011). I2C Bus Fundamentals. https://www.digikey.com/en/ptm/n/nxp-semiconductors/i2c-bus-fundamentals/tutorial.

[4] IAState-Mesonet (2020). IEM API. https://mesonet.agron.iastate.edu/api/.

[5] K. Hans, A. Jayakumar, D. G. (2016). A Review of Intelligent Practices for Irrigation Prediction.

[6] Lincoln Zotarelli, Michael D. Dukes, Consuelo C. Romero, Kati W. Migliaccio, and Kelly T. Morgan (2018). Step by Step Calculation of the Penman-Monteith Evapotranspiration. https://edis.ifas.ufl.edu/ae459.

[7] Lipford, D. (2020). How to Identify the Cause of Brown Spots in your Lawn. https://todayshomeowner.com/how-to-identify-the-cause-of-brown-spots-in-your-lawn/.

[8] Meter Group (2017). The researcher's complete guide to soil moisture. https://www.metergroup.com/environment/articles/the-researchers-complete-guide-to-soil-moisture/?lang=0&access=true.

[9] O'Geen, A. T. (2013). Soil Water Dynamics. https://www.nature.com/scitable/knowledge/library/soil-water-dynamics-103089121/.

[10] Platt, C. (2009). *Make: Electronics*. Make Community, LLC.

[11] RasPi.TV (2013). How to use interrupts with Python on the Raspberry Pi and RPi.GPIO. HowtouseinterruptswithPythonontheRaspberryPiandRPi.GPIO.

[12] Sharma, V. (2019). Evapotranspiration-based irrigation scheduling or water-balance method. https://extension.umn.edu/irrigation/evapotranspiration-based-irrigation-scheduling-or-water-balance-method.

[13] soilsensor.com (2020). Soil Moisture and Irrigation. https://soilsensor.com/soil/soil-moisture-and-irrigation/.

[14] Sparkfun (2020). Differential I2C Breakout. https://www.sparkfun.com/products/14589.

[15] Vasquez, J. (2017). Staking The Leap Off Board: An Introduction to I2C Over Long Wires. https://hackaday.com/2017/02/08/taking-the-leap-off-board-an-introduction-to-i2c-over-long-wires/.

[16] Weed Man (2020). Leaf Disease. https://weedman.com/resources/lawn-care-library/leaf-disease#:~:text=If%20your%20lawn%20is%20being,and%20along%2C%20the%20leaf%20blade.

[17] Wikipedia (2020). Penman–Monteith equation. https://en.wikipedia.org/wiki/Penman%E2%80%93Monteith_equation.