Barn Owl

ESE 519 Final Report

Uzval Dontiboyina | Grayson Honan | Nikheel Vishwas Savant

12/20/2016 | Dr. Rahul Mangharam

A - Motivation

Further advances in agricultural automation are necessary to provide food to the world's ever-growing population; working on Barn Owl allowed us to address this important challenge. Our shared interest in automated horticulture and IoT made this quite an exciting project to work on.

Our goal with Barn Owl was to automate a micro-farm using an existing greenhouse outside of the Skirkanich building. Using the closed-loop system we designed, the greenhouse cares for up to four plants automatically. Various sensors (temperature/humidity, lighting, soil moisture) are used to monitor conditions and the environment is controlled accordingly. Watering intervals, fans, and lighting are adjusted by our system to provide proper growing conditions. A front-end interface is used to monitor the greenhouse and also provides manual control if needed. The front-end interface allows users to specify which plants are being grown in order to better tailor the conditions to the needs of a certain plant species. A camera slider system was developed to photograph the plants on demand. These photos can be sent in their original form or can be run through an image processing script to extract additional information.

B - Goal Statement

Baseline Goals:

1. **Functional front-end interface:** Allows users to monitor the greenhouse and provide manual control over lighting, temperature, and watering system. This interface will also allow users to choose the type of plant being grown, which will change the water/lighting/temperature regimen for that particular type of plant.
2. **Sensors:** Four soldered sensor modules used to detect temperature, light, soil moisture, and humidity conditions.
3. **Closed-loop Control:** Water dispersal system, installed fan, heating, and lighting all controlled by automated gardener program (depending the type of plant).

Reach Goals:

1. **NDVI:** Provide NDVI (Normalized Difference Vegetation Index) data to front-end for plant stress detection. We think this will be an interesting challenge because we'll need to interface a camera or infrared sensor of some kind with our system. Based on changes to the greenhouse environment, we will track how the NDVI changes over time. The NDVI readings will form a preliminary model for optimal growing conditions.
2. **Motorized Camera Slider:** The NDVI data will be provided by two cameras. In order to capture NDVI data for all plants being grown, we will create an automated camera slider system to move the camera from one end of the greenhouse to the other. This will be an interesting challenge because no one in the group has mechanical design experience.

Note: we didn't use NDVI image processing, but rather performed alternate image processing. See Part E, section 10 for more details. We also only used a single camera, as a dual camera setup was only needed due to NDVI image processing.

C - System Design

Hardware Design: This project is an iteration on the ESE 350 project from Spring 2016, Isoplantation. Barn Owl uses two main aspects of the Isoplantation team's work: the greenhouse structure built outside the Skirkanich building and some of their purchased components. Using these pre-purchased components saved our budget for new additions and allowed us to immediately start work on the project.

Figure C1. Greenhouse structure lit up at night during a testing session.

To address the sensing aspect of our project, we initially thought we'd use new digital sensors for their accuracy, but we were concerned that running long cables for I^2C or SPI bus would cause the bus capacitance to be over its limit. The four analog sensors used were the LM34, DHT11, a photoresistor, and the SparkFun Soil Moisture sensor. A sensor board was placed in each corner of the greenhouse and a soil moisture sensor was inserted next to each of our four plants.

Soil moisture, light, and temperature sensor outputs were connected to an analog multiplexer, as shown in Part D, System Architecture. The humidity sensors were connected directly to digital pins on a Photon. We chose Photon as our main sensing platform because we knew it would be ideal for rapid prototyping. To make the system more modular and affordable for mass replication, a more powerful and professional-grade microcontroller should be used (the Texas Instruments CC3X00 series is a strong contender, or one might choose the Raspberry PI to control the whole system).

We grew thyme, rosemary, kale, and lettuce. Some images from our camera slider system can be seen in Figure C2 below.

Figure C2. Four images from our camera slider system captured during Demo Day 2's demonstration

The greenhouse lights, fan and heating were controlled via a relay for each. We use the PowerSwitch Tail II relay for this task. Analog pins A1 through A4 were used to control the solenoid valves to provide water for each plant, and pins A5 through A7 controlled the fan, heat, and light relays.

The watering system consists of perforated tubing buried in a circular pattern around each plant. Lighting is clamped to the top of the greenhouse frame, with one clamp light for each plant. The fan is a simple inductor in-duct fan readily found at Home Depot. The heater we purchased is a Patton PUH680-N-U Milk-House utility heater.

The camera slider system used a Raspberry PI and PI camera connected via ribbon cable. The Photon simply isn't powerful enough to handle the resolution of the photographs we wanted to take without doing some low-level buffering and manipulation of the images. Additionally, the many sensors in our greenhouse occupied all the available pins on the Photon, as can be seen in Part D. The construction of the camera slider is further detailed in Part E, section 8.

Software Design:

We tried many different solutions for the front-end of our system, but ultimately decided that a Python TCP server and Python GUI would be the best choice. The general design is as follows: the Photon sends data to the Python server, and the server is the sole writer to a series of data files. These data files are opened and read by the GUI. The GUI is the sole writer to a control.txt file, which is read by the server. This design prevents the server or GUI from writing to the same file. This design choice had the side effect that our GUI would also need to control the scheduling of care tasks (such as watering a plant). Ideally the GUI would be separate from this scheduler, but since we wanted to avoid concurrency issues, we rolled the scheduling in with the GUI.

There are many different Python GUI options, but we chose to use TKinter, as there was a wide variety of documentation available. In our experience, it seems TKinter was very popular at one time, but is waning in popularity today. This caused some initial difficulty installing the necessary dependencies, but once this was done, we didn't face any significant challenges.

D - System Architecture

With the system design from Part C in mind, we formalized the design into the architecture you'll see in this section. Figure D1 is a high-level view of our system architecture. On the greenhouse side of things, we use the Photon to turn various subsystems on/off depending on the state of the received control variables -- this includes fan, heat, grow lights, and water. The soil moisture, temperature, light, and camera system are used to monitor the environmental state and send feedback to the server. The server receives this new data, displays it live for users to view, and determines what change, if any, is needed in the control variables to bring the environment to the desired conditions.

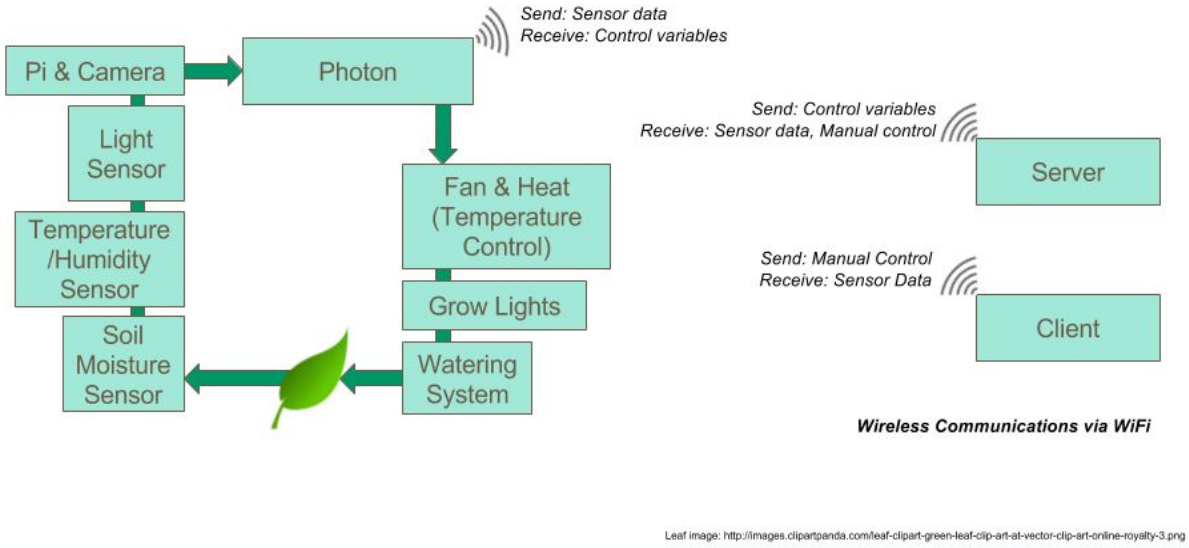


Figure D1. System architecture overview

Figure D2 depicts our so-called "Photon circuit," which is the hub for the sensing aspect of our greenhouse. In Figure D2, the label SM is shorthand for soil moisture sensor; channels C0 through C3 on the MUX are soil moisture sensor analog outputs, which are in turn read by the Photon. LIGHT0 through LIGHT3 are the photoresistor readings, TEMP0 through TEMP3 are the temperature readings, and H0 through H3 are the humidity readings. The nodes labeled FAN, HEAT, and LIGHT are the pins used to control the relays for the fan, heat, and light, respectively. Nodes W0 through W3 control 12V solenoid valves to turn water to each of the four plots either on or off. The general solenoid connection to our system is shown in Figure D3 below.

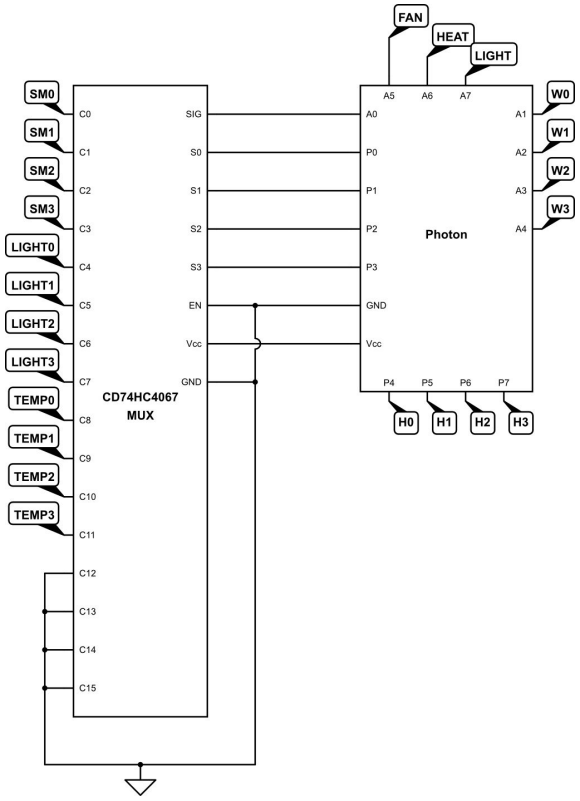


Figure D2. Photon circuit diagram

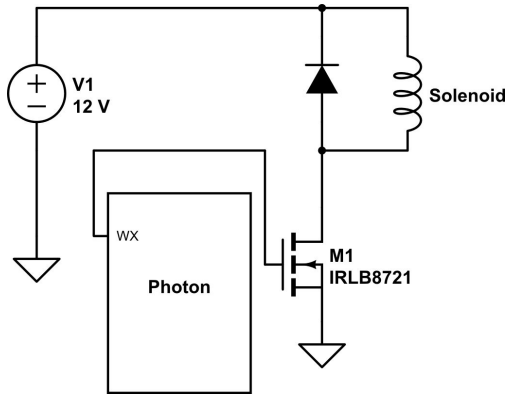


Figure D3. Solenoid connection for each water valve

E - Detailed Development Steps

1. Building the Greenhouse

We did not have a copy of SolidWorks available to create a 3D model (no team members were in Philadelphia more than a few hours after their last final), but we've attempted to provide as much detail as possible.

Figure E1. Wooden frame (Credit for construction photos goes to Isoplantation Team)

a.) The base of the greenhouse is made up of 4x4 posts for the legs (9" long), a sheet of plywood measuring approximately 84" x 44", and 2x12 planks (2 planks cut to 84" and 2 planks cut to 42"). Figure E1 depicts a typical construction of the wooden frame. L-brackets were used to add additional strength to the legs, but the brackets were only added on one side. We'd recommend adding these brackets on at least 2 sides in future builds, as two legs had collapsed within the first year of the installation. We'd also recommend adding 2 legs in the middle of the length of the plywood (labeled "1" and "2" in Figure E1's right-side image) to prevent the plywood from buckling under the weight of wet soil.

b.) An aluminum 80/20 frame was constructed on top of the wooden base. Plastic drop cloth was used as a covering and was secured with custom laser-cut fasteners designed by the Isoplantation team, as no commercially available fasteners could be found. Four pieces of 80/20 approximately 5' tall were mounted upright in each corner of the wooden base. Five 80/20 pieces cut to 84" ran along the top, middle (forming the roof), and approximately 1' from the top of the wooden base (at about knee height). Two pieces of 80/20 cut to 44" ran along the top. See Figure E2 to see the actual greenhouse frame.

Figure E2. Greenhouse frame *(Credit for construction photo goes to Isoplantation Team)*

c.) A control box should be mounted on the side of the greenhouse frame to hold the Photon circuit and additional components, including a power strip and relays. A hole in the bottom of the control box should be cut for wires.

d.) A sheet of acrylic laser cut to the diameter of the fan should be mounted to the side of the greenhouse.

e.) The wooden base should be filled with soil. We recommend 70 pound bags of Miracle Grow soil from Amazon (currently $7.99 with free shipping for Prime customers).

f.) A four-way hose splitter with attached solenoids should be mounted to the wooden base, with rubber perforated tubing running to each plant under the soil.

g.) Four clamp lights should be attached to the middle rail of the greenhouse frame (see Figure E2).

f.) Outdoor heater should be positioned near warm-weather plants with highest priority for heat. See Future Work for ideas about heating.

2. Sensor Boards

Figure E2 depicts the typical sensor board deployed in our greenhouse. We created four identical sensor boards with light, temperature, and humidity-sensing ability and placed them in the four corners of the greenhouse. The sensors used were an LM34, DHT11, and photoresistor.

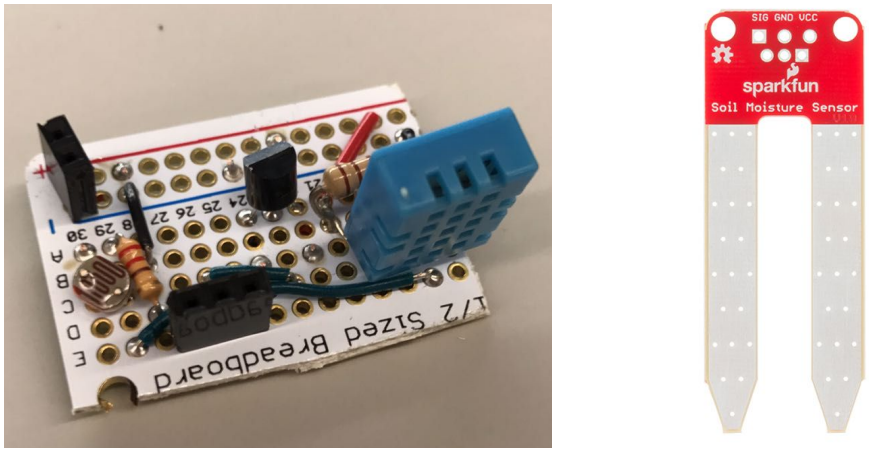


Figure E2. Sensor board and soil moisture sensor. Image source: https://www.sparkfun.com/products/13322

Tin snips or a similar tool can be used to cut larger protoboards to the appropriate size. We chose to add female headers to our sensor board design so that we could easily plug in Vcc, GND, and leads for the three signals coming from the sensors. We'd caution against using female headers to allow the sensors to be plugged in and swapped out, as this leads to unreliable connections. Connections are much more reliable if sensors are directly soldered to the board. Adhesive tape or some type of insulation on the back of the protoboards is the minimum recommended waterproofing. For long-term installations, we'd recommend a conformal coating or plastic enclosures for each sensor board. Note: hot glue is not sufficiently waterproof in most cases.

A female header should be soldered to the soil moisture sensor depicted in Figure E2. We recommend a 3-pin screw terminal type connector.

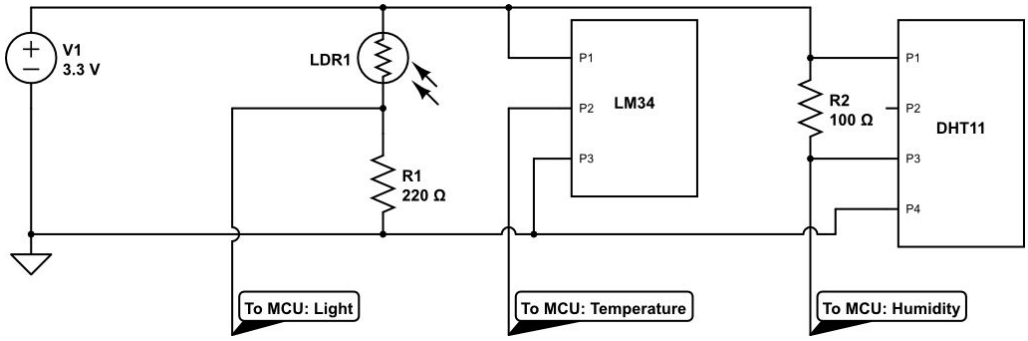


Figure E3. Circuit diagram for sensor board shown in Figure E2.

Figure E3 shows the connections needed for our sensor board. Additional 3.3V sensors can easily be accommodated in our design.

3. Microcontroller Programming - Sensors

The following section pertains to the file Photon.ino, which can be found in the "/code/Photon" directory. This code was written for a Photon, which uses an Arduino-like language. This code is easily portable to other platforms that use an Arduino-like language. This file should be flashed to the Photon and the Photon should be connected to a hotspot of some kind.

To begin using this file, change the server IP address at the top of the file (Line 51). In the setup function, pin modes are assigned for the MUX signal pins (s0_pin - s3-pin), the fan, heat, light, and water controls. We also initialize all the DHT11 sensors using an Adafruit DHT library (included in the Libraries folder) and attempt a connection to our server. This initial connection will either print "Connected!" or "Connection failed." over the serial for debugging purposes only.

In the main program loop, we select the appropriate channel on the analog mux, take readings, save the values, and process the data if needed (we use a moving average for each soil moisture sensor, and take an average of the four humidity, light, and temperature values). After the data has been acquired, we concatenate a string to be sent to our server, containing all the most recent sensor data. Once again, debug information is printed if this data can't be sent to the server. The server responds with the current state of the control variables (water valve controls, heat, fan, and lights). In lines 256 through 288, we acquire this string, process it extracting the current state of our control variables, and adjust the system state accordingly.

4. Sensor Server

The following section pertains to the file server.py, which can be found in the "/code/GUI and Server" directory. To run the server, simply call "python server.py."

The sensor server handles TCP requests from the Photon code mentioned in section 3. When the file is initially run, the script checks internet connectivity and determines the host IP address. If data is currently held in the data CSV files, this is cleared to create a fresh log. This step can be skipped (commented out) if that functionality isn't desired.

When the server receives data from the Photon, it is segmented into the appropriate data CSV file and appended. The received data is also written to the console. The Photon can send the temperature, humidity, light, and soil moisture data in any order, and the text parsing in the server.py file will correctly segment the data.

After writing the latest sensor data to CSV files, the server reads the control.txt file and sends the contained control variables back to the Photon. As mentioned in section 3, this string is subsequently processed by the Photon to extract the current state of each control variable.

5. GUI and Scheduling

The following section pertains to the file GUI.py, which can be found in the "/code/GUI and Server" directory. To run the server, simply call "python GUI.py." Platform-specific installation instructions for TKinter can be found at http://tkinter.unpythonic.net/wiki/How_to_install_Tkinter.

The GUI.py file contains a class for the two main frames and a class called GreenhouseApp for initialization. The GreenhouseApp class creates the frames StartPage and PageOne, and packs them into a high-level container. The StartPage class contains functions to turn the water, light, fan, and heat from manual to automatic mode, and vice versa. This class also creates TK buttons, drop-down menus, and a canvas for the temperature, humidity, and soil moisture graphs. The switchPlot functions are called when a drop-down menu selection is made, and these functions update the environmental values based on the selected plant's CSV file.

The PageOne class displays the images plot1.jpg through plot4.jpg, and when the refresh button is pushed, updates the displayed images to the latest copy of the files.

The next major section of code is the animate function, which updates the graphs on the StartPage every 1000 ms. The data CSV files written by server.py are opened by the animate function and the latest data point is appended to the individual graphs' X and Y data point lists. These X and Y coordinate lists are used to plot the correct data for temperature, humidity, and the soil moisture graphs.

At the end of its run, the animate function also calls careGreenhouse and carePlots. The careGreenhouse function simply checks if light, heat, or humidity need to be turned on due to a manual flag being set or an environmental condition being met -- if so, the relevant part of the control.txt file is updated. Light conditions are checked every 30 minutes to see if light should turn on or remain in the off state, or vice versa. Temperature and humidity are hysteresis-controlled, e.g. if the greenhouse is considered too cold, the heat will turn on and remain on past the target temperature, once the high set-point is met, the heat will turn off, and will remain off until the heat drops below the low set-point. Humidity works in a similar fashion. The carePlots function handles watering each of the plots in much the same way, e.g. if a plot's soil moisture is detected to be below the dry-point for that particular plant type, the water runs for 10 minutes. As the soil dries over the next ~24 hours, it falls below its dry-point again, and the watering system is triggered to turn on.

This scheduling system closes the loop by sensing the environmental conditions and adjusting appropriately. If, for example, a plant doesn't need to be watered, then the system detects that, and the water is never turned on. This scheduling system doesn't run a static schedule for every single day, but rather it adapts to the current state of the

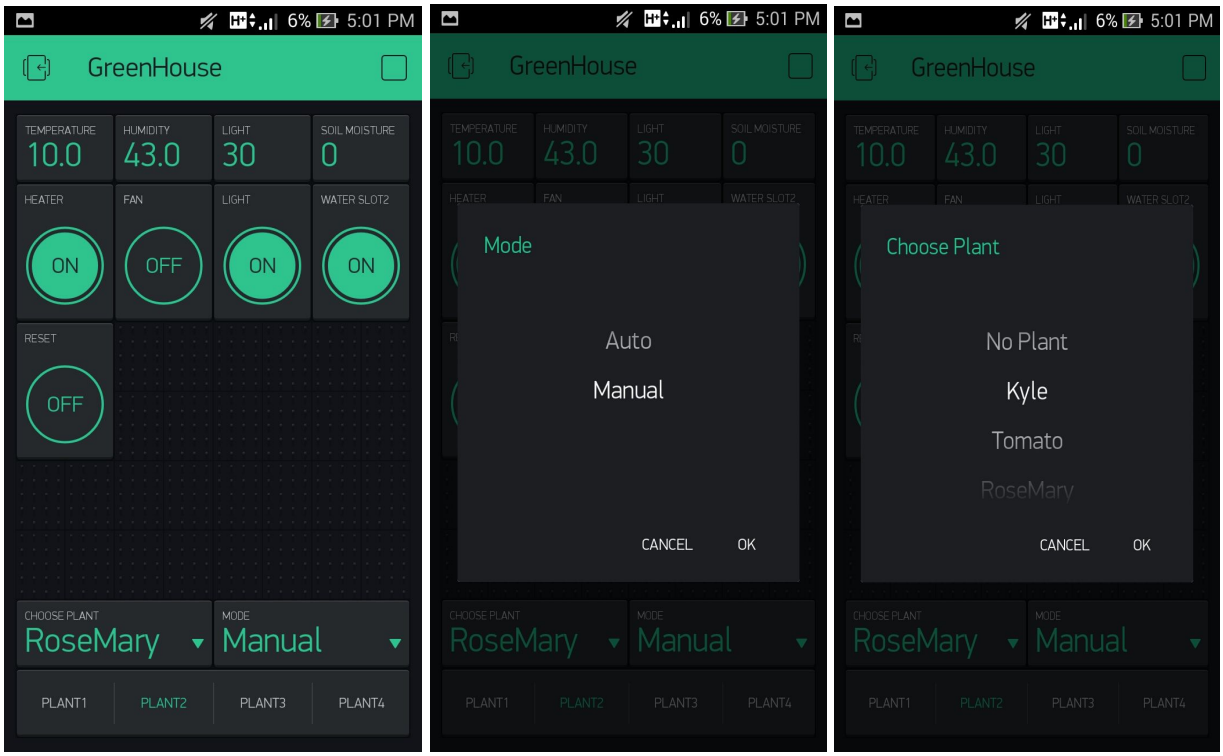
greenhouse. The logic used is adaptable, such that if a static schedule is desired (such as "water the plants every day at 7AM"), the changes needed to the code are minimal.

6. Mobile Application

The following section pertains to the file mobileapplication.ino, which can be found in the "/code/Photon" directory.

In order to have a great user experience and to promote ease of accessibility, we developed an Android application named GreenHouse that is built on the Blynk platform. The GreenHouse application can control hardware remotely, it can display sensor data, it can store data, visualize it and also can be used for scheduling the control flow. The GreenHouse application leverages the cloud capacity of the Blynk server and the Internet connectivity of the Photon to provide easy interaction between the greenhouse environment and the application. The GreenHouse application is run on the open-source Blynk server which is responsible for all the communications between the smartphone and the hardware. This Application has a maximum handling capacity of 100 users.

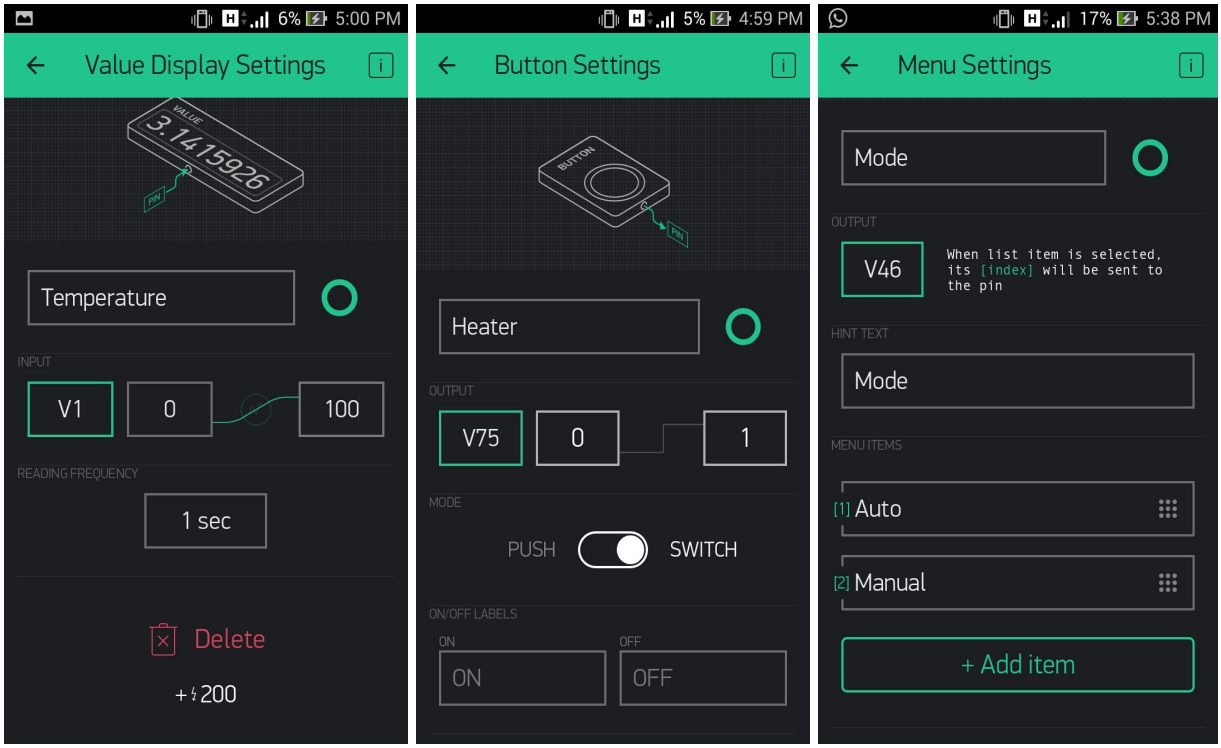
The front end of the application consists of the buttons, switches, sensor meters, and graphs. This dashboard can continuously add/modify the data representation (Graphs, meters, digital, analog outputs). The GreenHouse application GUI is shown below:



The GUI displays temperature in Celsius scale, Humidity in %, Light in % and Soil Moisture binary (0 means water needed). There are different tabs for different plants. Users can choose which type of plant they want to monitor, and there are two modes: auto mode will be a preconfigured optimal conditions for the plant and manual mode allows users to control the variables based on their knowledge. A reset button is provided, which once pressed, will clear the session variables for that tab, that particular plant slot will be changed to No Plant and the Mode will be changed to Auto.

Middleware:

In order to display the data onto the application GUI, it was required for us to use Virtual Pins. We designed Virtual Pins to send any data from your microcontroller to the GreenHouse app and back. In order to change the color of the buttons when pressed and to synchronize the common resources such as light, fan, and heat, we use the Blynk wrapper classes and sync functions for the middleware. To restore the session variables after power loss at the Greenhouse we use Blynk cloud's storage.



Backend: This GreenHouse application continuously communicates with the Photon microcontroller at the rate of 1 complete system update per 2 seconds. The back end part, which includes the Photon code, has all the main logic. It collects the sensor data, updates them periodically and virtually writes them onto the corresponding virtual pins, which show the final outputs to the user. As soon as a request is received from the user through the GreenHouse application, a High priority interrupt is generated and the user request is processed first and then the Photon goes back to its normal work. We were able to

achieve the high synchronization speeds not only over the different tabs in the dashboard, but also across the application-hardware channel.

We also solved the problem of having session variables lost when there is a power loss or loss of connectivity. The Photon when it faces any of the issues would restart the program and lose the session variables. This is where we use the cloud capability of Blynk server and continuously save the session variable to the Blynk server. So that, until and unless a hard reset is done or reset button from the app is pressed, the session variables are preserved.

Instead of recreating this application, this application can be downloaded into any Android phones and can be modified accordingly to the next team's requirement.
To start using it:
1. Download Blynk App: http://j.mp/blynk_Android or http://j.mp/blynk_iOS
2. Touch the QR icon and point the camera to the code below or open the link below - http://tinyurl.com/z97dhgn

7. System Integration

In this section, we'll be combining the sensor boards from section 3, the TCP server from section 4, and the interfaces from sections 5 and 6 to test the cohesive system.

One of the more tedious tasks for this project is getting all the sensor boards connected in the greenhouse. We recommend using Command Hook velcro strips to attach the Photon circuit board to a waterproof enclosure with a small opening in the bottom for wires. At this point, the Photon circuit board should resemble Figure D2 with four duplications of Figure D3 for the four water valves. This allows the sensors to be read, and the environmental conditions to be controlled. Detkin Lab usually stocks 16-wide ribbon style wire, which can be cut and split into the right length and width (see Figure E4).

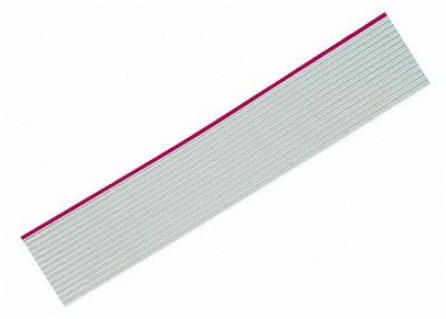


Figure E4. Ribbon cable. Image source:
http://www.jlmaudio.com/shop/images/source/10_wire_ribbon_cable_1_.jpg

This ribbon wire should have male headers on both ends and should be either 3-wide or 2-wide. In our design, we needed eight total wires that were 3-wide to attach our four sensor board outputs to our Photon circuit and to attach our four soil moisture sensors to our Photon circuit. For the 2-wide wires, use these to attach Vcc and GND to the four sensor boards.

Once the Photon circuit and sensors are installed, the water valve controls should be connected to a 12V source (as seen in Figure D3) capable of providing ~5 amps of current. The solenoids used can draw a significant amount of current, and this 12V source will also be used to power the Raspberry PI in future steps.

Run the GUI and server scripts in two separate Terminal windows (Using the command python scriptNameHere.py). If the Photon is successfully sending data to the server, you should see the sensor values being printed to the console. These values should also be reflected in the live graphs of the GUI. The fan, heat, light, and water valves can all be controlled from the GUI at this point as well.

8. Camera Slider Assembly

For the camera slider, we reused several components from the Lab 6 balance bot to save money and time. The three motors used in the camera slider were powered by two A4988 motor drivers (from the balance bot) and a SparkFun EasyDriver that we purchased. The two motors used to move the camera along the slider came from the balance bot itself, and a third was purchased from SparkFun (ROB-09238) in order to control the rotation of the camera. 1” PVC pipe and T-connectors formed the length of the slider.

We created an I-shape with the height of the I being the 84” length of the greenhouse, and the T-connectors at the ends. The spare 3’ of PVC piece from the initial 10’ pole was cut into four equal sections to form the top and bottom bars of the “I” (fitting into the T-connectors). We used rope running through the top and bottom bars of the “I” to hang the slider from the greenhouse frame.

Wheels and accompanying motors from the balance bot were mounted to 1-gang low-voltage brackets from Home Depot and attached to either end of the PVC pipe (see Figure E5 and video link below).

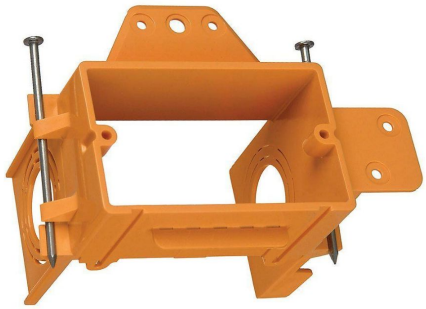


Figure E5. 1-Gang low-voltage bracket from Home Depot. Image source: http://www.homedepot.com/p/1-Gang-Low-Voltage-Bracket-SC100A/ 100157326

The slider portion for the camera itself was also built from a 1-gang low-voltage bracket, with the Raspberry PI and PI camera on-board. To see the entire system in action, please view our camera slider video: https://www.youtube.com/watch?v=YH9azuOEQMQ.

To provide power to the Raspberry PI, we recommend using the 12V DC source used for the water valves and a 5V linear regulator such as the LM7805. This will provide a clean 5V signal to the PI.

9. Raspberry PI Programming Part I - Camera Slider Controls

The following section pertains to the files found in the "/code/Raspberry PI" directory.

To move the camera along the slider manually (for testing purposes), use the sliderMotor.py script. To rotate the camera to a different position manually (for testing purposes), use the cameraMotor.py script. For both of these testing scripts, you must specify a direction and a number of steps, e.g. "python cameraMotor.py left 300".

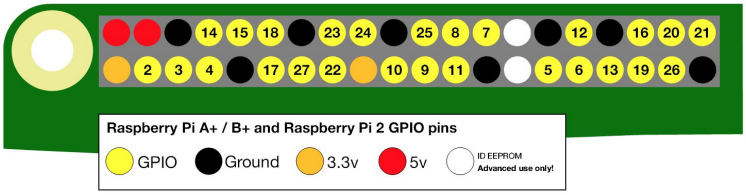


Figure E6. Raspberry PI 3 GPIO pin layout relevant to the code in this section. Source: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/

Running the script halt.py will stop all motors if needed for testing. The move.py script takes pictures of all four plots, rotates the camera as needed, and moves the camera along the slider as needed.

10. Raspberry PI Programming Part II - Image Processing

The following section pertains to the detect_plants.py script found in the "/code/Raspberry PI" directory.

The goal of our image processing was to detect and mark green plants in a (non-green) soil area image using Python OpenCV.

A. Capture the images:

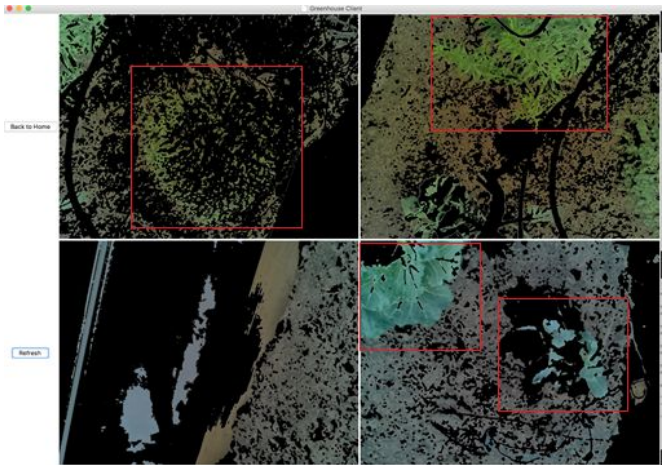
The PI camera was used to capture the images at four plots:

B. Run the script:

We ran the script detect_plants.py in Python for image processing.

- Median filter to remove image artifacts
- Image mask to detect green hues

C. After image processing:



11. Raspberry PI Programming Part III - Sending Images

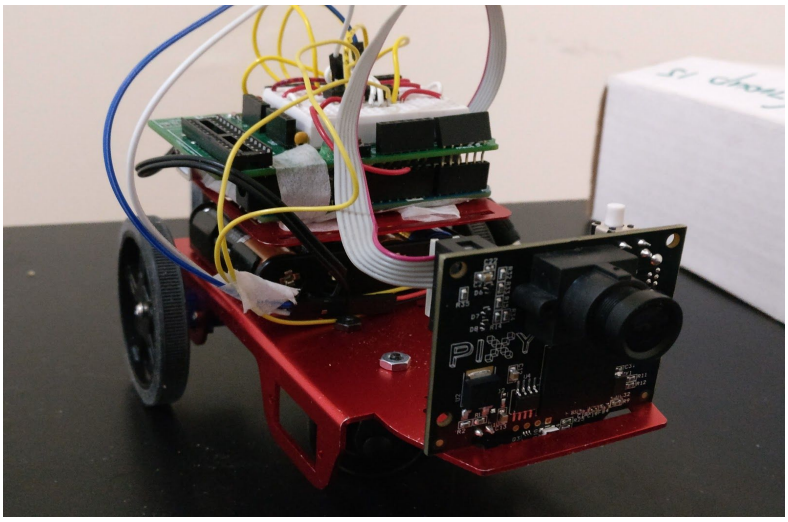
The following section pertains to the send.py, camera.sh, and process.sh scripts found in the "/code/Raspberry PI" directory and the stackSERVER.py script found in the "/code/GUI and Server" directory.

The send.py script connects to an accompanying image TCP server script running on the host PC (stackSERVER.py). The send.py script will send each of the four images captured to the host PC server.

The camera.sh shell script will run move.py and send.py. Running process.sh will run move.py, and then detect_plants.py, and then send.py.

F - Additional Work or Making it Cooler

Since the camera sliding system couldn't access every part of the greenhouse, We thought that having a robot which can travel throughout the greenhouse would give lot of advantages. Our plan was to use a robot to find pests and weeds in the greenhouse and remove them with the help of this robot. The robot was built using Arduino Microcontroller, 2 DC motors, Pixy Camera and was powered by 4 AA batteries. Provided that the robot has wheels which can handle the greenhouse terrain, it can be used to spray nutrients/pesticides to individual plants. We have trained our robot to scan the area randomly after certain intervals of time and look for an Android plush toy - its target. If it finds the Android plush toy, the robot will drive towards it. We developed this aspect of the project as an extra and didn't readily have access to actuators with which we could have demonstrated the spray mechanism. For now, this proof of concept just made the robot to go near the plush toy, maintaining a distance of 1 foot.



You can view a demonstration of the progress we made on this extra feature at the video link below:

https://drive.google.com/open?id=0By_xES1y3X9YdHpKSzlNQ1dZMEU

G - Future Work

Some improvements we would like to make to our system include making the camera slider more reliable, because currently it suffers from slippage issues. We recommend laser cutting a rack and pinion style track and attaching this track to the PVC pipe, rather than using a pulley system.

We think a smart robot to chase away pests would be an interesting addition (see Part F for our initial progress on this).

The user interface was made to provide manual control and to provide an easy way to select an automation regimen, but we think the GUI could be expanded to allow more flexibility for static schedules. Users may want the greenhouse to execute a pre-defined schedule that doesn't change every day, or they may want a compromise between a fully-dynamic system and a static schedule. Adding the flexibility to set timers for how long manual mode is turned on for would be a good first step towards this goal.

Heating the greenhouse can be done with multiple heating units, or a single heating unit and a system of ductwork to transfer heat evenly throughout the greenhouse. Cold/warm-weather plants can be separated by acrylic or plexiglas sheets for additional flexibility.