5. PROGRAM A GREENHOUSE SENSE AND CONTROL SYSTEM

What do an ordinary programmable thermostat in a home and the cruise control system in a vehicle have in common? Both let the user set an ideal value as an input for a program that keeps a system near the desired value. However, the program for cruise control is more complex than a thermostat program because it's very important for vehicle speed to stay very close to the ideal value.

A standard thermostat program is based on *bang-bang* control. When it's cold and the temperature drops to a value a couple of degrees below ideal, *bang!* the heater turns on. The heater runs until the temperature goes a couple of degrees above ideal, then *bang!* the heater turns off. If you completed any of the previous Greenhouse lab investigations, you have programmed a bang-bang control system where one or more sensor-based input determines whether an output should be turned on or off. *Proportional control* is just one part of the programming strategy cruise control uses to keep vehicle speed constant. Proportional control also requires sensor-based inputs, but outputs are *regulated* or set to constantly adjust by increasing or decreasing instead of being turned on or off. Vehicle speed input is closely monitored and the system constantly fine-tunes output settings such as throttle position to get as close to the ideal speed as possible. In this investigation, you will write a 24-hour program that integrates any combination of new, modified, and existing code with at least one use of proportional control to keep a living plant happy and healthy inside a fully automated Greenhouse.

Goals

- Incorporate existing code into a new program.
- Design a program that combines a variety of control structures to autonomously maintain a greenhouse.

Materials and Equipment

- Data collection system
- //control.Node
- Grow Light with included cables and USB power supply
- Greenhouse Sensor
- Greenhouse Sensor Module with cable and stopper
- Soil moisture probe
- USB Pump
- USB Fan
- Power Output Module with cable
- EcoChamber with lid and stoppers
- Assembled watering system*
- Potted plant, ~4" x ~4"
- Shallow dish
- Zip seal sandwich bag
- Ice

*Includes all Greenhouse Accessory Kit components (tubing, connectors, drip irrigation heads, hook-and-loop fasteners, and a #5 one-hole stopper), reservoir filled with tap water for USB pump, and materials to secure drip heads to the plant pot such as several strong rubber bands, zip-ties, and binder clips. Systems that require increased humidity may include a 2" x 2" sponge and bottle caps, however, once the plant is added to the Greenhouse, the setup may be modified as humidity conditions will change.

Safety

Follow these important safety precautions in addition to your regular classroom procedures:

- Keep water away from sensor boxes, electrical plugs, and exposed electronic boards.
- Don't allow exposed electronic boards to contact a metallic or conductive surface.

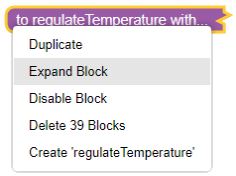
CAUTION:

- *Don't look directly at the LEDs.*
- *Don't touch the LEDs.*

Research

This section will help you understand how to set up and use a proportional control programming approach. Complete the following:

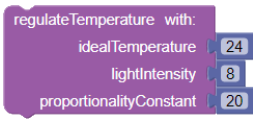
1. Connect the //control.Node to your device, choose any template to create a data display, then open the **Blockly Code Tool** .
2. Open the **Code Library** at the upper-right corner.
3. Change the category from *PASCObot* to *Greenhouse* and open *Regulate Temperature*.
4. Right-click or long-press the *to regulateTemperature with...* function block to open the menu as shown, then choose **Expand Block**.
5. Read the comment bubble to get an idea of this function's purpose.
6. Study the how the code is organized. Look for connections between editable function input values on the first function block (pre-set to 24, 8, and 20) and the code itself in the expanded function block. Predict how this proportional control system works before reading on (*Hint: the item in port B, CH1 is the USB Fan*).



The USB Fan uses bang-bang control to keep the Greenhouse temperature near the *idealTemperature* while the Grow Light is on. If the fan alone can't maintain ideal temperature, *lightIntensity* decreases from the ideal value to a lower intensity until temperature stabilizes near ideal. When temperature is stable the light intensity can gradually increase as long as ideal temperature is met. But what do the *error* and *proportionalityConstant* variables do?

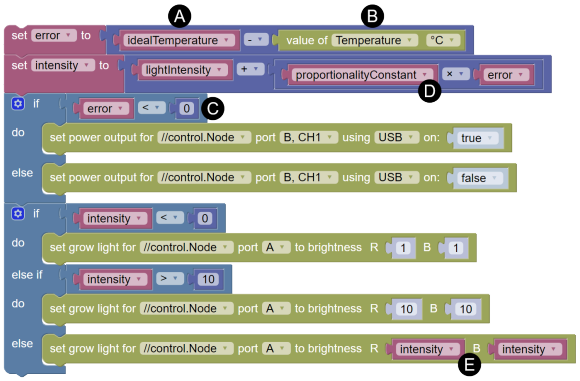
First, let's look at the 3 editable function inputs:

- *idealTemperature* - The target temperature is set to 24 °C.
- *lightIntensity* - The Grow Light red and blue intensities are initially set to level 8.
- *proportionalityConstant* - This has a value of 20... what exactly does that mean?



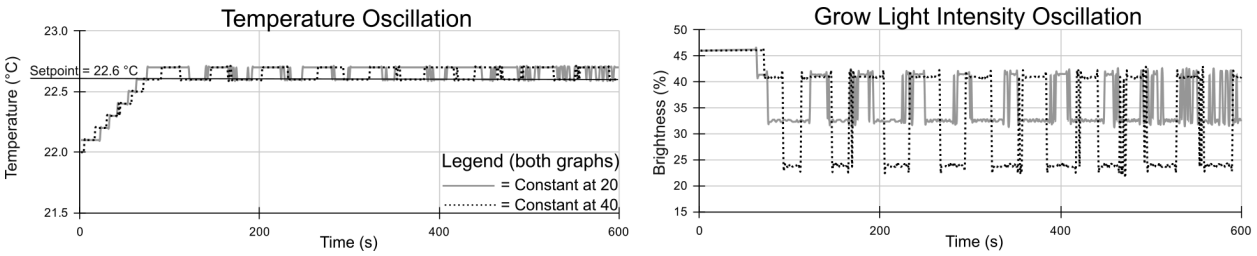
As seen in the code, proportional control programs need 3 things: an ideal value for a variable called the *setpoint* (A); a sensor measurement input called a *process variable* which checks real-time measurements (B) against the setpoint; and *error*, which is the difference between the setpoint and the actual temperature, (A) - (B).

The farther the actual temperature gets from the setpoint, the bigger error gets. If there is no difference between actual temperature and setpoint, error is zero and the system is stable. If the measured temperature is higher than setpoint, error is negative or below zero (C) and the fan will turn on; otherwise it will be off.



The USB Fan can't be set to different power levels so you can only use bang-bang control. The Grow Light has adjustable intensity so it can use proportional control. Grow Light output is *proportional* to error, which means the ratio between Grow Light intensity and error is a fixed value equal to the *proportionality constant*. The constant multiplies error (D) and adds it to the initial Grow Light intensity specified in the function input to calculate the *intensity* variable. The *intensity* variable is then used to update the Grow Light to a new intensity (E) as needed.

Error size and its positive or negative status make a difference. Negative error sets *intensity* lower than initial intensity, while positive error (where the actual temperature is lower than setpoint) sets *intensity* higher than initial intensity. Higher proportionality constants make greater changes to light intensity when the Greenhouse temperature moves away from ideal.



The graphs above show temperature data (left) and brightness data (right) for two 10-minute runs with the *regulateTemperature* function in a *repeat while true* loop. The setpoint was 22.6 °C for both runs. The solid line shows results when the proportionality constant was set to 20, and the dotted line shows results for the constant set to 40. On the left, both constants allow temperature to *oscillate* (repeatedly go above and below) around the setpoint. Did you notice that when the constant is set to 40, temperature is more stable between oscillations? Now look at the difference in Grow Light intensity oscillations on the right. The brightness drops *a lot more* when the constant is set higher - so less intense light and less heat is added to the chamber, that's why temperature is more stable with a higher proportionality constant.

Prototype

Part 1: Setup

Greenhouse Setup

1. Turn off the //control.Node and close SPARKvue.
2. Water your plant if needed.
3. Set the Greenhouse Sensor Module ❶, moisture probe ❷, and Grow Light ❸ in the Greenhouse lid as shown in Figure 1.

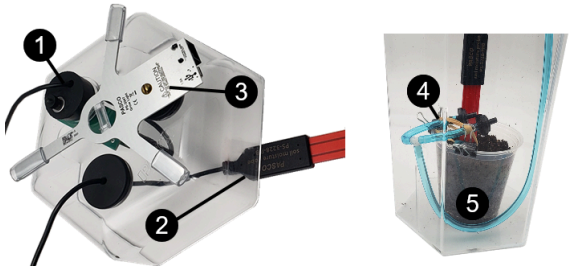


Figure 1. Greenhouse setup

4. Gently pull about 1 foot of extra soil moisture probe cable through the lid.
5. Use binder clips and rubber bands ❹ to secure tubing and drip heads to the plant pot as shown in Figure 1 (your pot should include a plant).
6. Push the moisture probe into the pot as far as it can go near a drip head as shown. Compact the soil around the probe.
7. Set the plant inside the Greenhouse with drip heads pointed slightly upward and with tubing set in a hook shape as shown ❺ to avoid air bubbles and excess drainage.
8. Optional: If standing water is part of your humidity control strategy, arrange bottle caps, drip heads, and standing water as needed. If a wet sponge is part of your strategy, devise a system for keeping the sponge at the proper height and distance from the fan once the fan is installed.

9. Place the lid on the Greenhouse and secure the water system tubing stopper and USB Fan in the side holes.
10. Gently pull excess moisture probe cable back through the lid and align the Grow Light with the light sensor.
11. Build a cooling system as shown in Figure 2. Add ice cubes to a shallow dish until it is nearly full (A).

Figure 2. Cooling system placed beneath lid

12. Set the dish inside the zip seal bag, remove excess air from the bag, and seal it.
13. Use the cooling system when you need temperature to rapidly decrease during testing. To use, set the chamber lid on the table with the cooling system positioned directly beneath the stopper where the Sensor Module temperature sensor sits (B, arrow). Do not allow the cooling system to contact any part of the Sensor Module.

Hardware Setup

1. Connect the moisture probe (1) and light, humidity, and temperature sensor cables (2) to the Greenhouse Sensor as shown in Figure 3. Plug the sensor into the //control.Node (3).

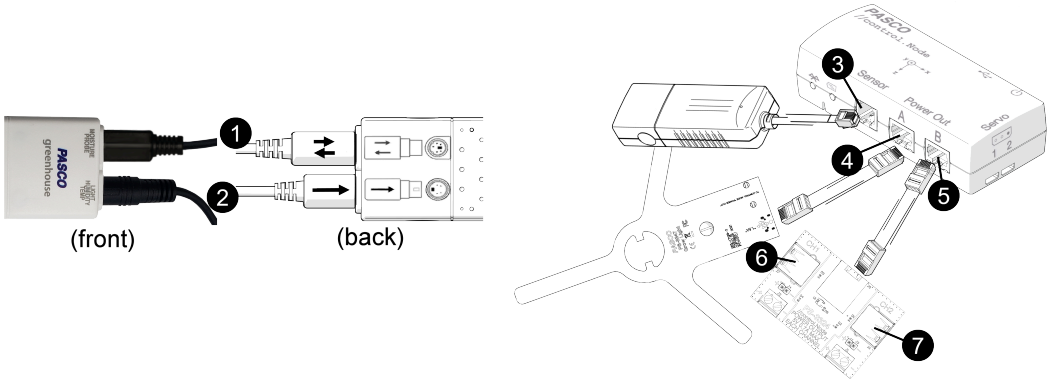


Figure 3. Hardware setup

2. Plug the Grow Light into //control.Node port A (4). Connect the light to its USB power supply.
3. Plug the Power Output module into //control.Node port B (5).
4. Plug the USB Fan into Channel 1 (6), and plug the USB Pump into Channel 2 (7).

Part 2: Upload Code to the ..//control.Node

The //control.Node has a feature that lets you store your Blockly code right on the device itself - so you don't even need to be connected to SPARKvue to run your program! You can see how this is handy for a Greenhouse that needs to run on its own for days, weeks, or months at a time.

1. Open SPARKvue and connect the //control.Node to your device.
2. Select ☑ **Temperature** and any other Greenhouse Sensor measurements you wish.

3. Disable the //control.Node On-board Sensor measurements by changing the slider from on to off.
4. Select any data display template you like. When the data display opens, start collecting data to record the current temperature, then stop collecting data.
5. Use the icon to open the **Code Tool**.
6. Go to the **Code Library** and insert the *Regulate Temperature* Greenhouse function.
7. Change the *idealTemperature* variable from the default value of 24 °C to 0.2 °C higher than the temperature you just recorded.
8. Place the function block inside a *repeat while true* loop. Test the code for a few seconds to make sure it works. Revise if necessary.
9. Use the **Upload code** icon at the top-right to send the code to the //control.Node. You will hear a series of beeps that indicate code upload is successful.
10. Use the green **Start Code Execution** button at the top-right to start running the program stored on the //control.Node. When uploaded code is running, you will see a flashing blue light on the //control.Node.
11. Close SPARKvue. Notice how the outputs continue to work in the Greenhouse and the blue light continues to flash.
12. Open SPARKvue and connect the //control.Node, then repeat steps 2 and 3.

> ***NOTE:*** *Uploaded code can only return sensor measurements; text and numeric outputs cannot be retrieved from uploaded code. Text and numeric outputs can only be retrieved in a data display when code execution is initiated with the* ***Start*** *button in SPARKvue.*

13. Start collecting data; allow a few seconds of data collection, then stop collecting data. Notice how the uploaded program continues to run whether data collection is running or not.
14. To stop the program from running on the //control.Node, either open the **Code Tool** and choose the **Stop Code Execution** icon at the top-right, or turn off the //control.Node.

Part 3: Review Existing Code

1. Go to the main menu in SPARKvue and choose **Open...**. Navigate to any previously saved work from any previous Greenhouse investigation that contains working code and open the file. Alternatively, you can reference code screenshots or sketches made from previous investigations.
2. Review your code. Decide where changes must be made for the program to work within a 24-hour cycle; make notes as you work.
3. Repeat steps 1-2 for all existing Greenhouse coding investigations. Use Code Library functions if desired.
4. Rebuild or open and run any previously-created code that has text outputs. While data collection is running, go to the data display and build a **New Page**.
5. Choose the full-page layout and add a **Table Display**.
6. Choose **Select Measurement** to add at least one Greenhouse Sensor-based measurement and at least one text output from the code. To add a text output, choose **Select Measurement** then switch from the **Sensors** tab to the **User-entered** tab in the menu that opens to the right.
7. Open the **Table Tools** menu below the table to add or remove table columns.
8. Observe data as it is added to the table. Use this type of data observation when you're troubleshooting code as it sometimes helps reveal new information.

Part 4: Plan Your Code

On a separate paper, sketch a flow chart that shows how you plan to modify and combine existing code to create a single program that:

1. runs on a 24-hour cycle with the daily start time defined by you;

2. determines a logical order of input and output events using appropriate loops and time blocks;
3. uses functions to keep program events separate and organized;
4. provides regular sensor input (temperature, humidity, brightness, soil moisture) to monitor conditions at an appropriate time interval for each measurement;
5. uses measurement inputs to trigger USB Pump activity, USB Fan action, and Grow Light intensities that maintain the preferred temperature, humidity, brightness, and soil moisture conditions for your plant throughout the day (remember to set the ideal temperature some amount above the current room temperature);
6. incorporates visual alerts (text output in a Digits display) and audio alerts (through the //control.Node speaker); and,
7. incorporates bang-bang control and proportional control.

The flow chart should include the approximate time of day each event occurs as well as ideas of the types of blocks you will use. The flow chart does not need to show all lines of code; its purpose is to help you plan and think about the things that need to happen in a logical order with the correct kinds of variables, loops, sleep delays, etc. needed for a successful program. For example, a flow chart step might say "sleep for an hour then use a *for each item* loop with a list to gradually turn on the light" instead of listing the many blocks that would be required to complete the task.

Test

Start a new experiment in SPARKvue and begin converting your flow chart to code. Remember to test each code feature as you work so troubleshooting is easier, and save your work as you go. For reference, there are 3,600 seconds per hour and 86,400 seconds per 24-hour day. Write code with seconds in mind, but when testing, change the time units from seconds to milliseconds to condense 1 hour into 3.6 seconds. This way, 24 hours is condensed into only about 1.5 minutes. Your program must meet all of the criteria outlined in step 4 of the previous section. If you get stuck during troubleshooting, try creating a table to see code results directly. Screenshot and print or sketch your code on a separate paper when finished, then answer the questions that follow. Save your work in SPARKvue for future reference. If you plan to let the greenhouse run autonomously on a long-term basis, keep the //control.Node plugged in to USB power, upload code, and execute code on the //control.Node before you exit SPARKvue.

1. Summarize the events that occur in your 24-hour program, and explain how your program was written to ensure the correct timing over a 24-hour period.

2. What were the top 2 challenges you faced when combining multiple input and output events into a single program?

3. What advice would you give your peers on how to be successful when combining multiple input and output events into a single program?

4. Explain your approach to using proportional control in your 24-hour program. Describe at least one advantage this type of control has over bang-bang control.

Improve

- Research the feeding or fertilization recommendations for your plant; add text and audio reminders to keep up with the recommended schedule.
- Instead of beeps, code a few notes from recognizable songs. For example, recreate a few notes similar to the chorus from Rhianna's *Umbrella* or Gene Kelly's *Singin' In The Rain* while the USB Pump is watering plants.
- Use a distinctive, attention-getting color changing or flashing Grow Light pattern to convert any text-based alert into a light-based alert.