Lab 3: Real-Time Embedded Controller of a Hot Air Plant using an RTOS

Matthew Salazar (1233223), Chase Keeler (1236835), Waleed Elshowaya (1235545), Cameron Bauman (1236693)

ENGG*4420

Petros Spachos

1.0 Introduction

The objective of this lab was to develop a real-time PID control system for a hot air plant on an STM32 board. We integrated known real-time systems concepts from past labs, such as task scheduling, cross-task communication, and synchronization within uC/OS-|||, but through the STM32F4291-DISC1 board. We utilized the lab 3 resources given in courselink, which provided a base C code with the different tasks, such as buttons, and provided a GUI implementation.

As mentioned, the STM32CubeIDE is used as the development IDE platform. We completed all coding, compiling, and debugging in C/C++ on this IDE. The board's built-in LCD interface, with the aid of the Tera Term, utilizes serial communication to display real-time processes, control information, and the display of control changes by the user.

We developed both automatic and manual control modes for the hot air plant system. The output is based on the temperature feedback set by the user. Through those periodic real-time tasks, semaphores, and a priority level system, we were able to display the appropriate response behavior on Tera Term while ensuring timing constraints and system accuracy were still achieved. Our main requirements we worked to achieve in this lab were to show input & output voltage, demonstrate the push button switch between the 2 modes, change the input setpoint, display live waveform graphics on LCD, and display the running clock to cross-check timing constraints.

The remainder of this report will provide technical background information required to understand the implemented system and main factors behind it, the steps taken in detail to complete the lab and develop the correct design with the required outputs, as well as an overall summary and reflection of the lab.

2.0 Background

Applying real-time system design principles discussed in lecture, this lab required students to extend their understanding of Real-time based task scheduling, synchronization and control of a PID controller for a hot air plant, extending what we already completed in lab 1 and lab 2. Unlike lab 2, where we applied more introductory RTOS concepts using basic tasks and semaphores, lab 3 included a more complex closed-loop control system running on an STM board.

In lab 1, we identified hot air plant dynamics were expressed as a first-order transfer function. In lab 3, the continuous time model had to be converted to a discrete time form for the display STM board so that it could be executed within a periodic real-time task. As shown in the lectures and the lab slides, the z-domain representation is important because it executes at fixed sampling intervals. Incorrect sampling or conversion can lead to unstable or inaccurate control behavior due to the feedback and input. This disagrees with the hard-time constraints we are trying to implement.

uC/OS-||| served as the scheduling framework that managed the system's function into multiple tasks such as out communication, GUI, plant, startup and button. These tasks communicate through message queues in which we develop and expand on in the methodology section. The use of fixed periodic timing is important because it ensures the plant updates, LCD refreshes and communication handling. This is important because they must also not interfere with one another.

We use keyboard-setpoint controls such as 1 and 2 for input changes, A and M for automatic and manual modes. The LCD displays real-time waveform and updates with the keyboard updates after uploading the code using the Tera Term. This also includes implementing event-driven task activation and message passing.

Overall, lab 3 directly builds on the past labs while including new complexed embedded control systems. It emphasizes discrete-time modelling, inter-task communication and synchronization techniques of a real-time feedback control system.

3.0 Methodology

3.1 System Overview

In lab 3, students are responsible for implementing a real-time digital PID controller for the hot air plant model that was previously developed with LabView in an earlier experiment. This iteration of the model utilizes an STM32 embedded development board and the uC/OS-III real time operating system. The system in this lab interacts with an emulated software version of the plant model that simulates temperature dynamics in a real-time feedback loop that performs actuation and monitoring in both manual and automatic modes.

3.2 Hardware & Software Setup

As discussed previously, students developed their real-time systems on the STM32F4 Discovery boards. They utilized onboard LEDs to indicate if the system is in automatic mode or manual mode. Additionally, user input from the keyboard was captured as keystrokes mapped to specific functionality within both automatic and manual mode. The “a” and “m” keystrokes were implemented as user inputs to toggle between automatic and manual modes. The keys “1” through “9” were used to control voltage input in the system for manual mode to incur a step response of output temperature. As well the keys “1” and “2” were implemented to toggle up and down the process variable for input voltage while the system was in automatic mode. The 2.4” TFT LCD was used to display the real-time response graph of the systems output as well as its mode status. An image of the STM32F4 Discovery board can be seen in Figure 1.

Like mentioned previously, this system utilized the uC/OS-III RTOS framework with a sample of code that was provided. STM32CubeIDE was used to develop the application and TeraTerm was configured with the correct baud rate and parameters for serial communication.

3.3 Software Architecture & Task Design

By utilizing the uC/OS-III framework, students were able to develop deterministic task scheduling and modular implementation of functions to meet the lab requirements. The software architecture for this lab consisted of four primary tasks: *Startup*, *Communication, GUI, and Plant*. These were supported by two inter-task message queues for synchronization and preemptive scheduling.

3.3.1 Startup Task

The startup task was responsible for initializing the system once scheduling begins. It configured the board support material by setting parameters for clocks, interrupts, SysTick, etc. It also initializes peripherals such as GPIO, USB, SPI, and I2C, as well as the LCD configuration. After this configuration stage is completed, it creates the application tasks and begins a low priority LED toggling sequence.

3.3.2 Communication Task

The communication is responsible for managing incoming USB CDC messages from the host PC. Incoming data packets from the PC are received and placed into a buffer/queue *“CommQ”* and forwarded to *“PlantInputQ”* so the plant model itself can receive inputs in real-time. This task serves as the primary communication foundation between the host PC interface and the systems control loop. Inputs from the TerraTerm terminal are processed from the PC side of the system and read in by this function before being stored in the buffers described above.

3.3.3 GUI Task

The GUI task was developed to update the LCD with real-time system information consisting of: *Vref (plant input), Vout (plant output), Sampling period, Run-time counter, and a Time series plot of the last N samples.* This task was initialized to execute every 200ms to provide a live visualization of the hot-air plant output signal. The separation of the LCD control into its own task optimized CPU usage by handling the LCD periodically and not having it in the same task as the control algorithm. This task initialized the variables mentioned above as well as the necessary timers it needed. It established its UI interface using board support functions and establishes a plot layout for the input to be graphed. The pixel lengths were mapped to fit the display appropriately on the LCD and different levels of *vref* were mapped to the bottom and top of the y-axis. To plot the continuous oscillating response output of the PID controller in automatic mode, the *BSP_LCD_DrawLine()* function was used to take individual samples after some interval and plot each line. This produces a series of small connected point-to-point lines that appear to be a much larger continuous plot. The output plot on the LCD can be seen in Figure 3.

3.3.4 Plant Task

The Plant Task simulates the behavior of the hot air plant by using a discrete transfer function and PID controller logic. Values for this transfer function were used directly from the PID controller tuned for this system in Laboratory 1. This transfer function was implemented in C by reading in the necessary input parameters and performing this operation in the form of *plant_input* and *plant_output* with the PID constants, error values, and setpoints used from the equation below. This logic can be found in the snippet Figure 2.

$$H(z) = \frac{0.119217}{z - 0.904837}$$

This task waits for messages from the *PlantInputQ* queue which contain user commands such as toggling between automatic or manual mode, changing the set-point, or setting voltage. Once a mode select message is received, the input “a” switches to auto mode, resets the PID state, and listens to set point commands. Similarly, the input “m” switches the system to manual mode and listens for set input voltage inputs. In automatic mode, the set point is adjusted by inputting “1” or ”2” where “1” increases the set point and “2” decreases it (while checking to ensure a negative voltage case does not occur). The *CommQ* queue holds messages from the USB interface while the *PlantInputQ* holds reference voltages from the communication task to the plant task. Once an output is determined it is stored in the *samples* array that will be used for graphing by the GUI task. Overall, the GUI task periodically updates every 200ms and the Plant task runs as fast as it is scheduled but it is logically tied to 200ms sample time. The Communication task is event-driven and message based and the Startup task is a one-time initialization. This structure maintained real-time performance while ensuring stable display updates and replicable simulation behavior.

4.0 Conclusion

This lab further expanded the group to real-time control systems using a GUI and reading keyboard inputs. Through completing this lab, students learned how to design an automatic and manual mode for a hot air plant. Using the previous K_P, $T_{i,}$ and T_d values found in lab 1, K_P, K_i, and K_d values were calculated for tuning parameters for the equation seen in figure 2. By creating a GUI task, our group updated the LCD with real-time system information corresponding to the plant output equation. During the creation of the real-time graph, our group faced struggles when trying to bound the voltage levels to the size of the graph. By finding the row of pixels at the top and bottom of the graph, we were able to assign each voltage level within a range a specific number of pixels, so that the graph was equally spaced. Alongside this, reading in the inputs from the keyboard for increasing and decreasing the setpoint posed a challenge for our group.

The lab mainly taught students a better understanding of how a real-world application can be modelled and simulated to display real-time data. Additionally, the lab showed the importance of software like STM32CubeIDE and how it can be used to simulate real physical systems.

5.0 References

[1] R. Muresan and K. Dong “ENGG4420: Real-Time Systems Design – Lab Manual” Courselink, https://courselink.uoguelph.ca/d2l/le/content/972947/viewContent/4257371/View (accessed Nov. 18, 2025)

[2] R. Muresan and K. Dong “ENGG4420 Real Time System Design – Lab 2: RTOS basics” Courselink, https://courselink.uoguelph.ca/d2l/le/content/972947/viewContent/4263977/View (accessed Nov. 18 2025)

[3] K. Dong “ENGG4420 Real Time System Design – Lab 2 Hot Air Plant Control With RTOS” Courselink, https://courselink.uoguelph.ca/d2l/le/content/972947/viewContent/4264009/View (accessed Nov. 18 2025)

6.0 Appendix

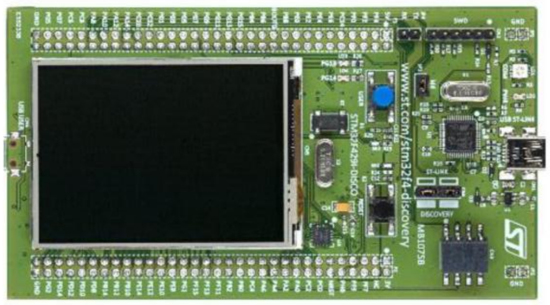
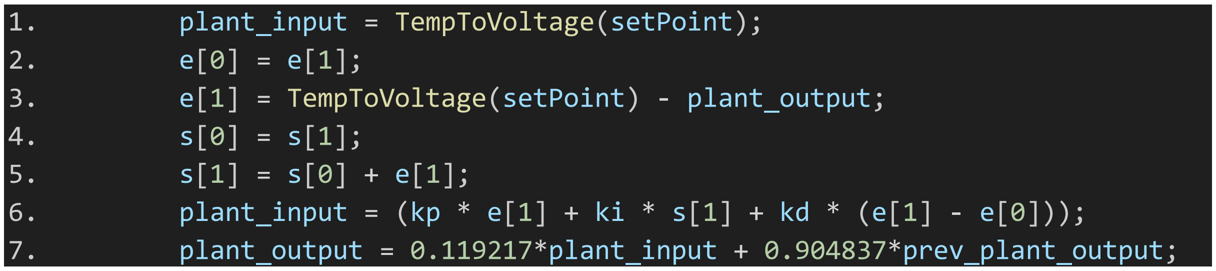


***Figure 1:** STM32F429 Discovery Kit*



```c
plant_input = TempToVoltage(setPoint);
e[0] = e[1];
e[1] = TempToVoltage(setPoint) - plant_output;
s[0] = s[1];
s[1] = s[0] + e[1];
plant_input = (kp * e[1] + ki * s[1] + kd * (e[1] - e[0]));
plant_output = 0.119217*plant_input + 0.904837*prev_plant_output;
```

***Figure 2:** PID Control Logic in C (Plant Task)*

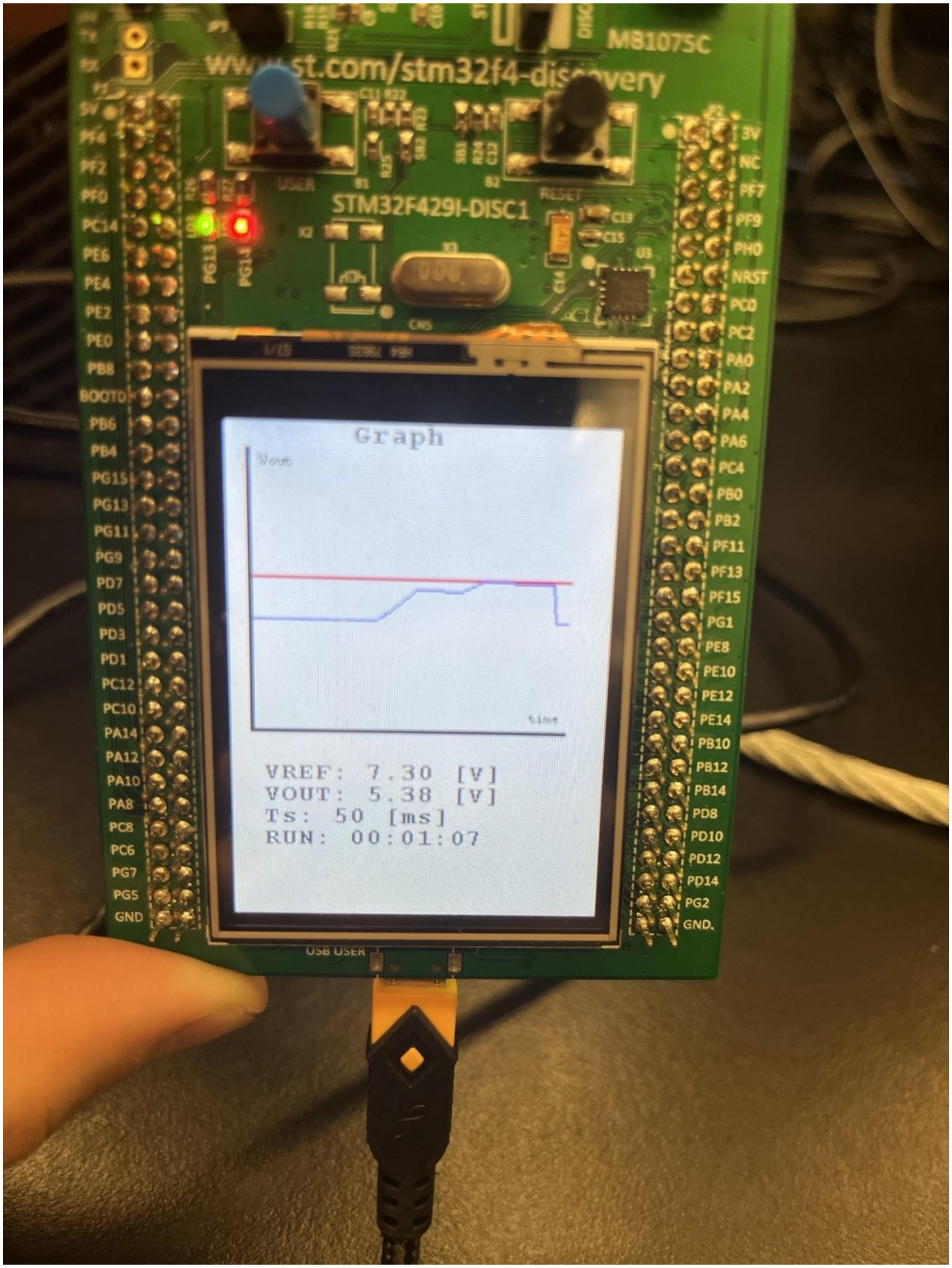
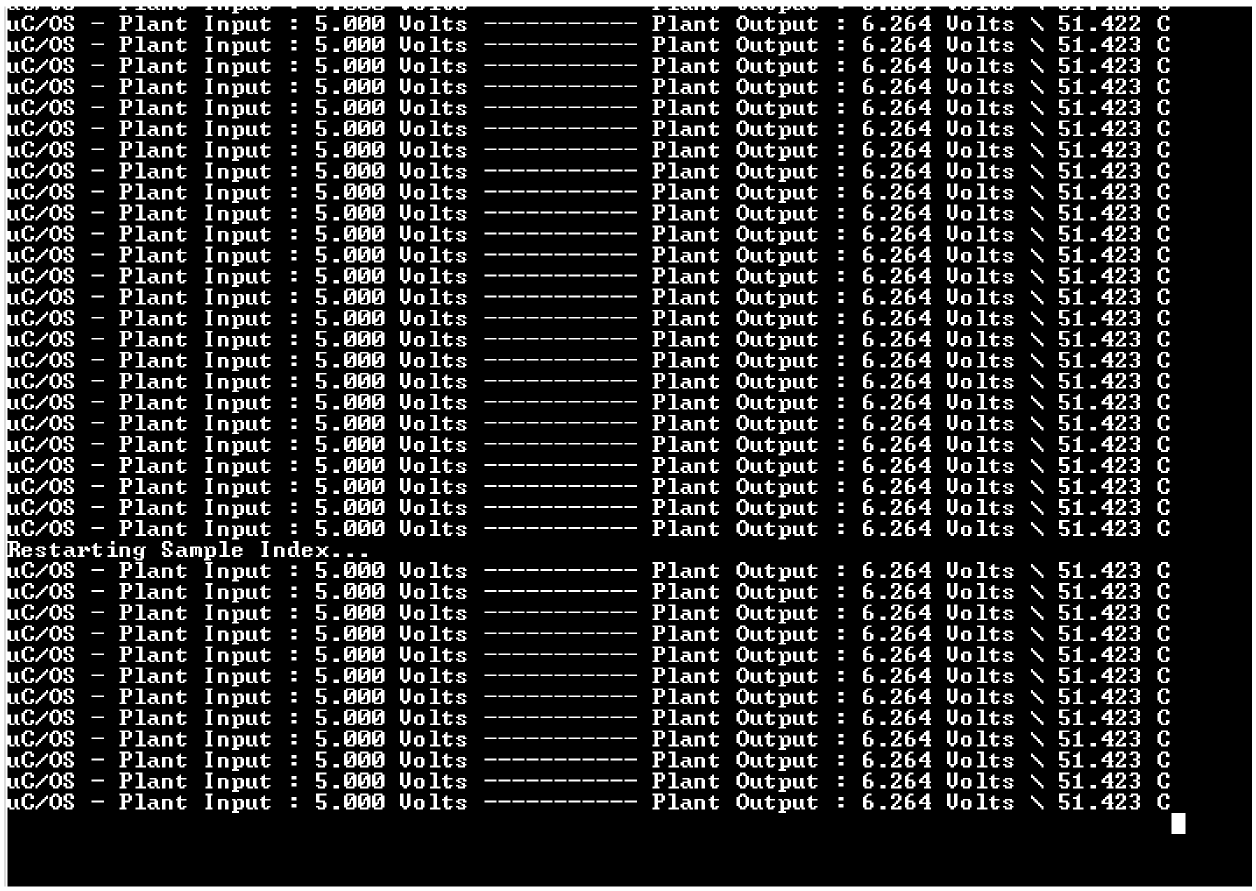


Figure 3: *System Response Plot on LCD*



```
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.422 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
Restarting Sample Index...
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
uC/OS - Plant Input : 5.000 Volts ------------ Plant Output : 6.264 Volts \ 51.423 C
```

Figure 4: *System Terminal Output*