Wireless Poo - Web Interface for the Brilliant Pad

Kelton Cheng

1. Abstract

The Brilliant Pad is the world's first self-cleaning doggy pad. However, the limited functionality and lack of wireless controls created inefficiencies in the pad's daily operations. The Brilliant Pad would occasionally roll too often or not enough during the day. This resulted in wasted pads or smelly houses. My goal is to expand the functionality with a browser based interface for controlling the Brilliant Pad. By using a Raspberry Pi based web server, I was able to implement a control interface for the Brilliant Pad. This greatly expanded the functionality and potential for future development.

2. Introduction

The Brilliant Pad was originally created to provide small pet owners a compact and automated solution for potty training. It added the benefit of allowing the pets a place to potty where they may not be space outside such as a yard. This greatly benefits those who live in urban areas where there is often little space outside as well as in the apartment. This solution also proved to be more efficient than traditional doggy pad training by optimizing the pad and using a roll. By using an interval system, it can roll the pad by itself to clear out any waste on it. There is an IR sensor located on the control module that will defer the rolling of the pad if there is a dog detected. Finally, the user can manually roll the pad in order to have a hands-free cleanup.

Despite this development, customers have reported some issues and suggested some quality of life improvements. One of these issues is that the interval choices available are not thorough enough. Currently, a user can only set the interval to 1,2, or 3 times in a day. Additionally, this will roll a full pad or one tick (4-5 inches) in this interval mode. Because of this, this can cause the Brilliant Pad to use either too much or not enough of the pad when rolling. This leads to wasted rolls or waste lingering on the Brilliant Pad. One other issue is that the IR sensor may not detect an animal on the pad properly. It has a very narrow field of view and only sees a small portion of the platform. This can cause the pad to roll even if a dog is on the pad. I may also detect stray anomalies past the platform. Finally, there were requests for possible remote use so users did not have to be in its vicinity to roll the pad.

I posed the solution to solve part of the problems with a web interface for the Brilliant Pad. The web interface will serve as platform to expand the limited functionalities of the Brilliant Pad. The primary goal was to demonstrate the feasibility of having an external control unit. As such, a Raspberry Pi was chosen to be the web server for this project. The Raspberry Pi had the ability to communicate with the Brilliant Pad via serial ports and also host a web server that also allowed use of the Raspberry Pi's board functions. These were the following parts for the project:

- Brilliant Pad and Modified Control Module from Brilliant Pad Team
- Raspberry Pi 2 Model B
- USB-TTL Cable
- Wireless USB Dongle
- Logitech C525 Webcam

3. Technical Material

The Raspberry Pi was the primary driving force for the entire project. By using the Python Flask library, I was not only able to host the web server using python as the backend, I can control the various functions on the Raspberry Pi such as the pins serial communication.

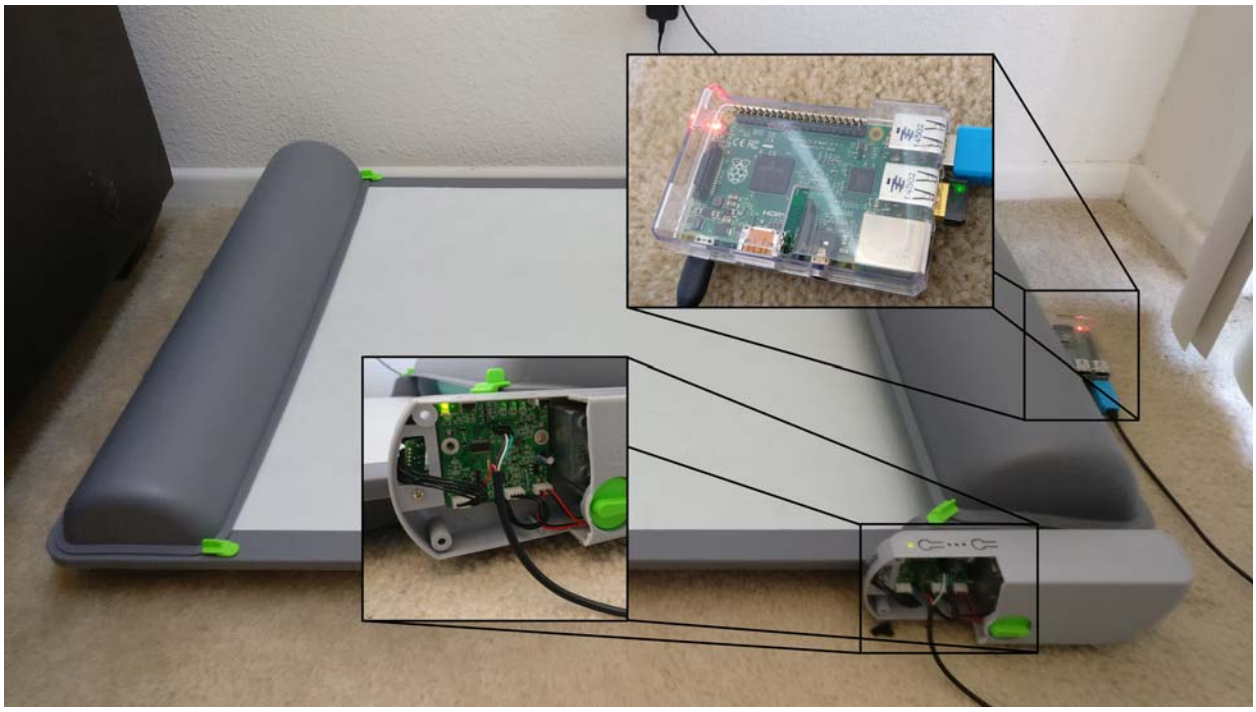
Firstly however, the Pi and the work environment needed to be configured before proceeding. I first had to static the IP address of the Pi in order to allow for a stable link for SSH, SFTP, and the web page. My local router was configured to assign the Pi a specific IP. I then had to purchase a wireless USB dongle for this Pi as it does not have onboard wifi. Using a monitor I enabled the pins and serial connections for the Pi as well as joining it to the local wifi network. Work was primarily conducted via SSH and SFTP from my main desktop. Majority of the times I had to directly use the Raspberry Pi was to manually start the web server if it had crashed during code updates.

Below is a snippet of the routing portion of the server:

```
@app.route('/')
def index():
    debugMsg("Accessing Index Page")
    return render_template('index.html',
            savedThreshold=config.get('WirelessPad','threshold'),
            intervalHour=config.get('WirelessPad','intervalHour'),
            rollAmount=config.get('WirelessPad','rollAmount'))

@app.route('/settings')
def settings():
    debugMsg("Accessing Settings Page")
    return render_template('settings.html',
            savedThreshold=config.get('WirelessPad','threshold'),
            intervalHour=config.get('WirelessPad','intervalHour'),
            rollAmount=config.get('WirelessPad','rollAmount'))

@app.route('/saveSettings', methods=['POST', 'GET'])
def saveSettings():
    debugMsg("Saving User Settings")
    if request.method == 'POST':
        print("In POST")
        newThreshold = request.form.get('slider')
        newInterval = request.form.get('intervalSelect')
        newRollAmount = request.form.get('rollAmountSelect')
        config.set('WirelessPad','threshold',newThreshold)
        config.set('WirelessPad','intervalHour',newInterval)
        config.set('WirelessPad','rollAmount',newRollAmount)
        write_config()
    return redirect('/')

@app.route('/consoleStream')
def consoleStream():
    return True

@app.route('/M')
def motorOn():
    debugMsg("Motor On")
    sendCmd('M')
    return redirect('/')
```

From this image, you can see one of the few pieces that make Flask a very powerful and appropriate web server for this project. Firstly, defining a link was very simple. By using @app.route('/'), I am able to define the main home page. Inside, that and every block I have a debug message that is printed to let me know what the server was trying to access. Originally,

this specific routing served only to host several buttons that would redirect the user to various other links and sections to access more functionality. However, it has been updated to have a more unified look. Another useful feature of Flask is passing in variables from the python code to HTML. In the '/' page, you can see that after I render an HTML template, the main page in this case, I pass in a few variables that I will get to later. Further down at @app.route('/A'), this is where I combine the web interface portion with the serial communication aspect of the Raspberry Pi. When the user presses the "Advance X Ticks" button on the main page, it will call this particular function. The Flask library retrieves the user set value on the main page and advances the pad how many ever defined tick marks the user set.



In the above picture, you can see the configuration of the project. The Raspberry Pi uses a USB-TTL cable to send commands to the Control Module of the Brilliant Pad. By default, the Brilliant Pad cannot accept commands in this method. Their team developed a new firmware and sent me a module with the update to allow control in this fashion. Below are the following commands available:

- 'A' - Advance sheet. Same as double click (almost same behavior as double click)
- 'S' - Single mark advance. Motor turns on and stops at the next pad mark.
- 'M' - Turns the motor ON, regardless of any other machine state. Still detects overcurrent and stops if so.
- 'm' - Turns the motor OFF.
- 'U' - UART ayt kind of test. Always returns "Cmd: UART OK\n"

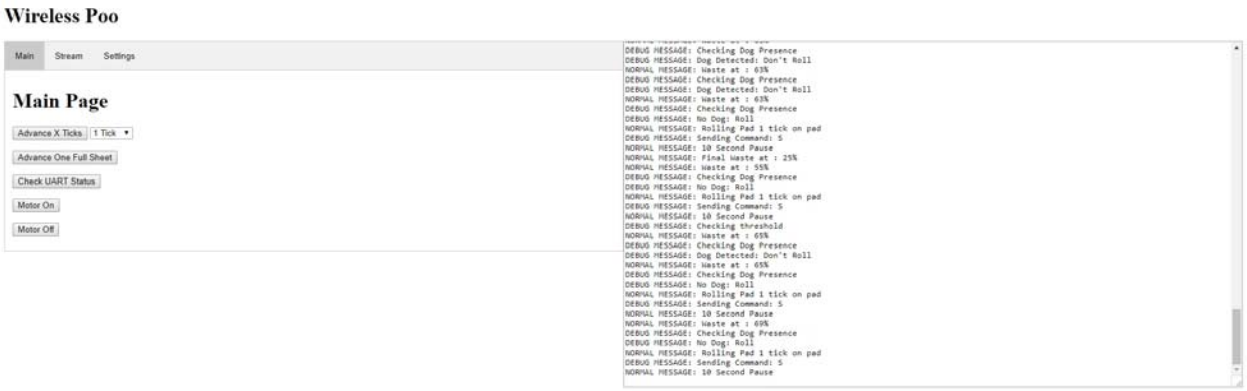
The Raspberry Pi does come with its own TX and RX pins on the GPIO pins, but I found that those pins seem produce a lot of garbage signal thus not allowing me to properly communicated with the control module of the Brilliant Pad. The USB-TTL also had issues with garbled signals, but it was more successful and thus the preferred method of communication. Even then, this often led to unresponsive commands.

```python
def getWaste():
    return random.randint(1,75)

def isDogThere():
    debugMsg('Checking Dog Presence')
    return random.randint(0,1)

def rollPad(ticks):
    if(int(ticks) == 6):
        normalMsg('Rolling until clean')
    elif(int(ticks) == 5):
        normalMsg('Rolling Full Pad')
        sendCmd('A')
    else:
        normalMsg('Rolling Pad 1 tick on pad')
        for i in range(0,int(ticks)):
            sendCmd('S')
            normalMsg('10 Second Pause')
            time.sleep(10)
    return True

def sendCmd(command):
    debugMsg('Sending Command: ' + command)
    ser.write(command)
```

```python
def saveData(dataType, data):
    currTime = datetime.now().strftime("%H:%M:%S")
    with open('wasteLog.csv', 'a') as f:
        writer = csv.writer(f)
        writer.writerow([currTime,dataType,data])
    return True

def checkThreshold():
    # Vision Team will pass their current value in
    wasteAmount = getWaste()
    config.read('config.ini')
    userThreshold = config.get('WirelessPad','threshold')
    saveData(0,wasteAmount)
    while(int(userThreshold) < wasteAmount):
        normalMsg('Waste at : '+str(wasteAmount)+'%')
        if(isDogThere()):
            debugMsg('Dog Detected: Don\'t Roll')
        else:
            debugMsg('No Dog: Roll')
            rollPad(1)
            # Update with new Waste Amount
            wasteAmount = getWaste()
```

I created various methods to facilitate incorporation with the Vision System Team's designs. The Vision System's goals were to detect a dog and the amount of waste on the pad using a camera and image analysis. It's aim is to improve on the single IR sensor and provide a better and more accurate coverage of the pad. During development, we were split and just created requirements for each other when we potentially integrate in the future. I am to receive a boolean on whether or not a dog is present on the pad. This will always be checked if the pad is about to roll. Next, is the amount of waste detected on the pad. The user have various settings that they can set. One setting is a threshold for the amount of waste on the pad. When checking how much waste is on the pad, the vision system will return a number indicating the coverage on the pad. If the waste is greater than the user's settings, then it will advance the pad. Currently, both of the checks are generated by a random number in lieu of not having the vision system incorporated into the project.

The next section will talk about the actual web interface itself. First, the home page.



On the right half is a debug window screen displaying the output of the Raspberry Pi. Since the Raspberry Pi's output can only be seen by the window that initiated the server, whether that is through SSH or on the Pi directly, it made sense to add this to allow viewing of the output if the main window is inaccessible. Initially, I wanted it to stream directly from the python server to the text box, but that proved to be more complicated than what I had hoped for. I moved to a simpler method. By outputting the python messages directly to a file, I can use Javascript to pull the information contents of the file and display them into the text box. By adding in a periodic scrolling to the bottom of the text box, allowed the most recent messages to be shown first.

The left half is a tabbed section giving access to the Main, Video Stream, and Settings page. The Main page shows various commands that can be sent to the Brilliant Pad. Each one leads to its corresponding python routing shown from before. In the python block, the command is sent over serial communication over the USB-TTL cable. This gives users the ability to control the Brilliant Pad from their phone or web browser without having to be in the device's vicinity. As stated before, there were issues with the serial communication not giving proper feedback. This occurred with various cables and the on board serial communication pins. Without an extra Raspberry Pi, I wasn't able to confirm if that was the source of the issue. Below, you can see output the communication when sending it commands. The left shows garbled and missing text when compared to the right. The Brilliant Pad team's tests showed that it is working fine, so there may be an issue with my configuration.

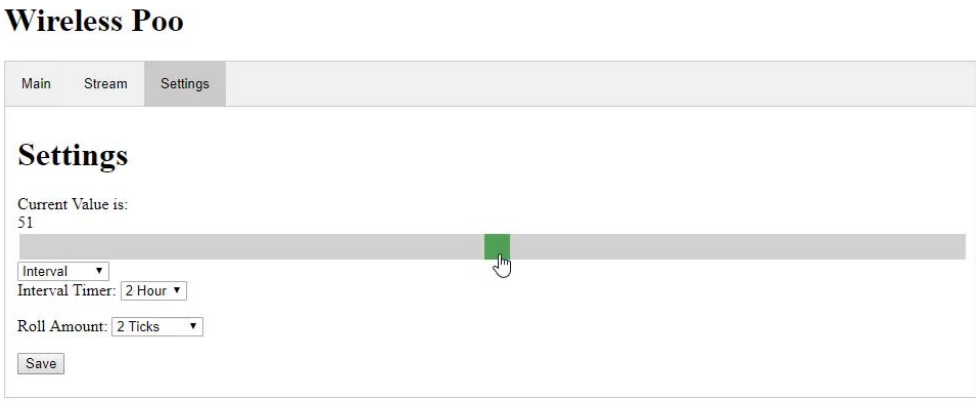
```
1 @raspberrypi:~ sudo python serial_U.pysh
2
3 pi@raspberrypi:~ $ sudo python serial_test.py
4 Cmd: SINGLE
5
6 PAD: pad_seek_mark
7
8 Motor: START
9
10 MOTORz▒?▒▒▒▒▒▒▒▒▒▒**
11
12 Queue = 20 20 20 20 20 20 20 20
13
14 tected at 15
15
16 Motor: STOP
17
18 pi@raspberrypi:~ $ sudo python serial_test.py
19 ▒
```

Bad Response

```
21 pi@raspberrypi:~ $ sudo python serial_U.py
22 Cmd: UART OK
23
24 pi@raspberrypi:~ $ sudo python serial_test.py
25 Cmd: SINGLE
26
27 PAD: pad_seek_mark
28
29 Motor: START
30
31 MOTOR: Mark #1 detected at 18
32
33 Motor: STOP
```

Good Response

The video stream displays a live video from the connected webcam at 640x480 @ 15fps. Because of the middle resolution and lower processing power of the Raspberry Pi, the video frame rate could not be as high. Reduction of the resolution could lead to faster frame rates, but I wasn't able to confirm or test this. The camera was not mounted in a reasonable fashion as it would be in the real world. My camera was placed on a tripod and aimed at the Brilliant Pad. This was acceptable as the Vision Team was developing camera mounting solutions for the Brilliant Pad. The video stream would give users an opportunity to manually assess the Pad's coverage.



For the final page, we have the Settings Page. There are various settings available for the user to change that I mentioned previously. First, there's the threshold settings. This will determine the maximum amount of detected waste on the Brilliant Pad before the Pad will roll the pad. This solves the issue for users where the pad can roll too often or not enough. With the threshold level, it can be tuned to be used as much as the user would like to have it roll. The next setting is the interval timer. This is very similar to the current existing function built into the control module for setting the intervals for rolling the Brilliant Pad. With the web interface, it's much more expandable to nearly whatever option a user would like. Currently, the user can roll it anywhere between 1 to 5 hours. The final setting you can see is the Roll Amount. This determines how many ticks to roll it by after the given interval timer activates or the system detects waste over a certain amount. A user can choose anywhere between 1 to 5 ticks, where 5 ticks is a full sheet, or an option called 'until clean'. This will roll the pad until the pad's surface is below the user's threshold. The system will roll once, poll the vision system to determine the waste level, and then roll again if it's not below the threshold yet. To note, the threshold detection and interval timers are unrelated. The user can choose which cleaning option they would prefer. The threshold will check every 5-10 minutes to see if the waste is above a certain threshold and roll accordingly. The interval timer on the other hand is just a simple timer that will every X amount of time. At the very end, the user can save all of the settings that they selected. The settings are saved into a configuration file. This configuration file is read on the page load and prepopulates the settings page with the users last known settings.

1	19:02:07	0	48	19	19:03:01	0	44
2	19:02:10	0	54	20	19:03:04	0	5
3	19:02:13	0	54	21	19:03:07	0	42
4	19:02:16	0	28	22	19:03:10	0	54
5	19:02:19	0	51	23	19:03:13	0	67
6	19:02:22	0	8	24	19:03:16	0	67
7	19:02:25	0	10	25	19:03:19	0	12
8	19:02:28	0	26	26	19:03:22	0	50
9	19:02:31	0	32	27	19:03:25	0	74
10	19:02:34	0	25	28	19:03:28	0	16
11	19:02:37	0	43	29	19:03:31	0	54
12	19:02:40	0	30	30	19:03:34	0	74
13	19:02:43	0	18	31	19:03:37	0	19
14	19:02:46	0	37	32	19:03:40	0	73
15	19:02:49	0	44	33	19:03:43	0	61
16	19:02:52	0	31	34	19:03:46	0	40
17	19:02:55	0	23	35	19:03:49	0	37
18	19:02:58	0	25	36	19:03:52	0	46

And finally, the system will also log and track usage data. Every time the system polls the vision system for waste levels, it can record the amount of waste detected and also whether or not the dog was on the pad. This can provide customers with useful information such as how often they use it, how much waste they leave, and when they like to use the pad. With more information, the user can tune and adjust the settings to match their needs and their dog's needs.

4. Milestones

- Week 2: Completed
    - View the video stream of the Raspberry Pi on a Web page
    - Control the Raspberry Pi through the webpage. Output will be either console or LED output to demonstrate the control
    - I was able to create a simple webpage that hosted a video stream. The video stream just referenced a different port on the Raspberry Pi's ip address where the camera was connected to. The web page contained simple buttons that when activated, there were corresponding python functions to enable specific Raspberry Pi Pins. In addition, I created output logs to record what had happened. This would form much of the foundation for the fundamental objective of this project.
- Milestone 2: Week 3
    - Accept data input from Vision System and any other sensor data that may be present. Use Random number generator to simulate the input.
    - Methods were created to handle the polling of the Vision System for data. Because the Vision Team created their project in Python as well, it would allow future integration to be very easy. I would simply call their method that returns desired the values and store them in existing variables. Because of this, adding and exchanging modules would be easy. In the meantime, the data was created with random numbers to simulate possible inputs in order to procede with testing.
    - There were issues with using random numbers. I sometimes had to wait for the right numbers to be generated in order to optimally test the system. Further testing should most likely be conducted with a series of predefined input that would more resemble real life usage than random numbers
- Milestone 3: Week 5
    - Add user settings for the following items:
        - Scheduling
        - Timer
        - Waste Threshold
        - Roll Amount
    - After a discussion with Alan Cook, he gave more specific requirements for the user interface which are listed above. I was able to complete most of the objectives. Most of these settings were defined in the settings page. After they were saved, the config file was accessible by other parts of the python server to allow for accurate representation of the users preferences.
    - The timer system was originally done through the Advanced Python Scheduler library, but the library was very buggy. I transitioned into using separate threads for tracking the times and intervals that were defined. This also made it easier to alter the timing of the interval as I would just have to pass in a new value to the thread.
    - I was unable to create a scheduling system where by the user could choose times of day for when the pad can roll versus the current system of just set intervals.
- Milestone 4: Week 8
    - Add data logging and tracking abilities to see animal statistics such as usage and waste.
    - Data was saved to a CSV file whenever the vision system was polled. The data held no real use at this current point since they were all random numbers, but with the existing framework there, it can be used in the future.

    - Future iterations can aim to creating data visualization to interpret the data. A CSV file is difficult to see any trends and correlation, so data visualization can make this aspect more useful.
- Milestone 5: Week 10
    - Communicate with the Brilliant Pad via onboard serial to control its motors and other functions
    - (Stretch) Merge project with vision system to create a single system
    - After receiving the updated control module, I started development with the onboard Raspberry Pi pins. I wasn't able to get clean communication with that method, so I used the USB-TTL cable provided by the Brilliant Pad team. This was still only partially successful still. When the communication did work, it worked flawlessly, but a majority of the time, commands weren't sent properly and the received response was not expected.
    - I was unable to meet my stretch goal of integrating with the Vision Team as there was not enough time to do so in the quarter.

5. Conclusion

This project was fairly successful. I was able to create a working proof of concept that demonstrates the capabilities of joining a Raspberry Pi with the Brilliant Pad's control module. While I was unable to get consistent communication with the Control Module, I think that in further testing, this can be hashed out. This groundwork provides many opportunities and direction for improvement into the future of this development. The web interface can be further expanded to enable more options because of the Raspberry Pi.