Project report

Group 1 方致偉 B04901001 蘇峯廣 B04901070 陳紹羽 B04901179

Pokecan

Pokecan is a trash can that can automatically detect the level of trash inside itself, and if it is full, it will walk along the path that is set by user and will dump the trash into a larger trash can. After it dumps all trash out, the Pokecan will walk back to its original location.

Hardware implementation

1. Motors

- There are totally six motors.
- Two adhered to the base by hot melt glue gun to control the ups and downs
- Two are bolted on the tubes by screws to control dumping.
- And two are under the base to move Pokecan around.

2. Wheels

- The two of the wheels that we use are plastic wheels that printed by 3D printer. To strengthen the friction for the purpose of no slipping, we use sandpapering first and attach them to the rubber by resin.
- And the others, we utilize the steering wheels to change the direction with a low additional friction.
- In detail, as we put the wheels to the base, the degree of tightening screws is also a problem. If looser, they will be idle running. If tighter, the wheels will be broken since the plastic printed out was not strong enough.

3. Plastic tubes

- There are two plastic tubes on the left and right sides. And inside the tubes are two pulleys bolted on the up and down sides. We use a long line combined with two pulleys and connected with the motor below.
- To be true, the tubes are the most important ones that make our mechanical components stable. If they aren’t “perfectly” parallel, the motors cannot strike a balance. This means that the controller IC will be overloaded and things will breakdown (we burned down half side of a MCP23017!). Then we will be

freaked out because we need to disconnect all the things and “reconstruct” it.

4. Sensors

- In this project, we use three different sensors.
- The first is ultrasonic distance sensor. The function of this sensor is to detect height, so that the system will know whether the trash can is already full or not.
- The second sensor is the Hall effect sensor. The Hall effect sensors’ functions are to detect whether the movement of one side is faster than the another side or not and to determine the distance Pokecan has moved.
- The other one is a micro switch, but we used it like a sensor. It take the responsibility of checking the height of the trash can. There are totally two on the up and down side. When the trash can touched the upper sensor, it will stop and dump. On the other hand, when it touched the lower one, it will also stop and turn to the moving mode to the original location.

5. Battery Pack

- We thought that making a rechargeable Li-ion battery controller is cool, thus we checked out some circuits for charging a Li-ion battery, doing switching/load sharing between DC input/Battery power, and boosting the voltage of the battery pack (3.7V) to 5V which the MCU needs. Then we designed our own circuit. The main components are:
  i. Microchip MCP73833 charge management IC: to charge and protect the Li-ion battery.
  ii. TI LM2621 switching voltage boost IC: to boost the voltage from 3.7V to 5V.
  iii. P-Channel MOSFET: when an external power supply is provided, the power source of main system will switch to external supply by this PMOS.
- Then we got the PCB fabricated in 10pcs and ordered the components we needed. Finally the board is completed and it can really act as a USB battery pack to charge a phone! However it can only draw about 1A of current due to the restrictions of LM2621 and the inductor. It’s okay to use STM32F401 MCU with this battery pack; however Raspberry Pi draws about 2.5A so in the end we used a battery pack we bought online instead of the one we made ourselves.
- For the motors, we have 4 motors using 6V and 2 motors using 12V. Thus we have 3 non-rechargeable packs, each with 4 slots for 1.5V AA battery.

6. STM32F401

- This was the microcontroller unit we’ve used. It’s based on ARM’s Cortex M4 CPU architecture. All programs in it are written in C/C++. However just 5 days before the demo, the MCU died due to some shorted circuit. Thus we switched our platform to Raspberry Pi.

7. Raspberry Pi

- Despite its tiny size, it has full capabilities just like a desktop computer. We use it to control all other hardware components of Pokecan. The operating system running inside is Raspbian, which is based on Debian, a Linux distribution.

8. ICs

- L293D: This is a motor driver IC from STMicroelectronics/Texas Instruments. It takes 3 inputs: one to control the voltage by PWM signal, and the other 2 one to control the direction of the current. With these combined, we can easily control directions and speeds of the motors.

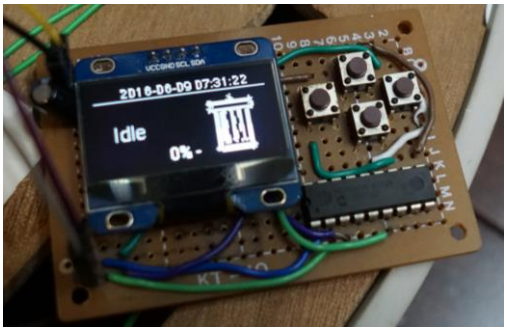
- PCA9685: It's a 16 channel PWM output IC from NXP. I2C bus is used to control it. By sending a PWM pulse signal to L293D, we can control the voltage output.

- MCP23008/MCP23017: It's a simple input/output expander using I2C from Microchip. MCP23008 has 8 channels and MCP23017 has 16 channels. Since we need many inputs/outputs, using these can avoid having many wires between Raspberry Pi and other components. For example, we have soldered a board with 4 L293D and a MCP23017 on it. Then we just need 5 wires (3V3, 5V, GND, SCL, SDA) instead of dozens. Unfortunately, half side of our MCP23017 has burned out, thus we utilized the unused channels in PCA9685 as a simple output signal by setting the PWM duty cycle to 100%.

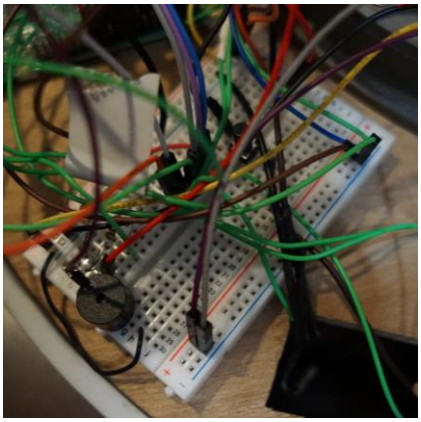
- US1881: This is a Hall effect sensor from Melexis. Its output will change when the polarity of the magnetic field around it changes. It takes a 5V input so we put a voltage divider to make the output voltage to 3V3. However we found out that the output was an open drain (floating/connected to ground) so just a pullup resistor instead of a voltage divider is needed.

- SSD1306: It's an OLED display controller using I2C/SPI (configurable) from Solomon Systech.

- HC-SR04: This is an ultrasonic distance sensor. The output is 5V so a voltage divider is needed. It will output a pulse signal so by multiplying the signal length by the speed of sound wave we get the distance from the sensor to the object in front of it. It's interesting that we didn't found out the name of the manufacturer of this IC/module!



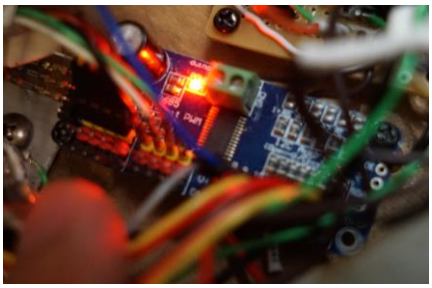
Our display module (SSD1306, OLED, buttons, MCP23008)



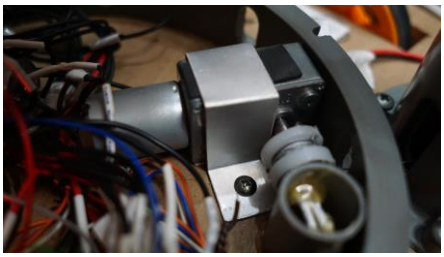
Breadboard connecting things together

Raspberry Pi

The motor controller board (L293D, half broken MCP23017) hidden under jungle of wires



PCA9685 PWM output module



Lifting motor bought from Taobao!

Micro switch determining the vertical position of the can

The wheel with a Hall effect sensor attached to it

Software implementation

Since there are many different components to control and many features to implement, we wrote programs for each component/function. Here's a list and descriptions of each program.

1. master: It stores the state (sensor readings, idle/move/dump status, etc) and configuration in memory. It also handles communications between different programs by a listening to a Unix domain socket (a socket file). Programs can connect to this socket and update or retrieve state information and configuration parameters by issuing commands in JSON format. This program is written in C++ to ensure the performance handling socket connections.

2. display: This program controls the user interface module (SSD1306 OLED controller and tactile buttons handled by MCP23008 GPIO expander. Each screen is a class and has 2 basic functions (render: takes a canvas and draw

the screen on it. key: takes a key number and do actions according to the key, returns the new screen ID or no screen transition). The program will first set the current screen to MainScreen, and check whether a button is pressed. If a button is pressed, it invokes the current screen's key function. Then it checks the return value from key function. If a screen transition is needed, it replaces the current screen with a new screen. Finally it calls the current screen's render function. Then the program will sleep for 50ms, and then loop to checking buttons again. The program is written in Python to make the development fast.

3. motor: This program retrieves the motor states from master every 100ms and sends signals to MCP23017 GPIO expander and PCA9685 PWM controller to control the motors' voltage. The program is written in Python.

4. route: This program manages the whole moving and dumping process. The motors are not directly controlled by it, instead it control motors by sending commands to master. Every 500ms the program checks whether a trash dump is requested. If there is a request, it starts the dumping process. First it sets the status of Pokecan from idle to dump to prevent from getting more dumping request. Then it retrieves the preconfigured route from master and moves along it. When moving, it reads the Hall effect sensor to know how far Pokecan moved. After getting to the target position, it lifts the trash can. While lifting, it listens to the top micro switch. If the switch is triggered, then it means the can has reached its top positon, thus it will stop the lifting motor. And then it starts the dumping motor to dump trashes inside. Then the previous process is done again in a reverse order, to make Pokecan go back to its original position. After all processes are done, it updates the state back to idle. The program is written in Python.

5. web: This is a tiny HTTP server that will read the state from master and render it to a web page. Also it can handle a GET request and then request for a dump from master. The server is written in JavaScript since I'm more used to write web applications in Node.js.

We've modified `/etc/rc.local` file to make the program start in background as Raspberry Pi boots. You may now find that most of the programs are written in Python. Because Python is easy to learn, many users recommends/uses Python in Raspberry Pi, and thus there are many 3rd party libraries written in Python for hardware components available. Using these libraries saved much of our time.

Language	files	blank	comment	code
C++	1	43	2	315

Language	files	blank	comment	code
Python	20	188	10	971

We wrote about 1300 lines of code!

The C++ version of source code (incomplete, used on STM32) is at https://github.com/dodo0822/pokecan

And the final Python version of our code is at https://github.com/dodo0822/pokecan-rpi

Problems & Future

We've encountered many problems integrating the hardware and software together.

1. When we're putting things together, things start to break down time over time. However most of the time it's just a wire came down and it wasted much of our time. We need to be more careful with wires in the future.

2. The Hall effect sensor is not so reliable, for example in two wheels turning in same speed, one reported 100 turns and the other one reported 500! So we only used one side of the sensors and tuned the voltage of two motors to ensure they spin at the same speed. Even we've spent hours tuning the sensors and motors they are not so accurate. During demo you can see that Pokecan can't move back to its original position after dumping trashes. In the future maybe we will replace them with infrared sensors and more expensive motors.

3. Bottom motors are not powerful enough. Though it seems good at demo, sometime it would free spin and the screws need to be tightened. When we found this there was no enough time to buy another one and install it.

4. Currently the only method to charge the battery pack is to connect directly to USB. Installing a wireless charging module would make it more convenient!

5. The web server is currently hosted in Raspberry Pi. This means users need to be in the same LAN with Pokecan, or have a static IP and port forwarding enabled to use the web interface. In the future, we may set up an external server, and have Pokecan push its status to the server and pull requests from it, thus users can manage their Pokecans from our server.

6. The appearance is not so appealing now. Maybe an outer shield will make it look much better.

Conclusion

The main part of our project is successful, as we implemented its basic functions to move, dump, and interact. However there are also many things that we can improve. Things are not so perfect as we expected, but every time we face a problem, we can think of a way to fix it! In this project, we learned a lot about using many different electrical components and the process integrating hardware and software. We want to give special thanks to Prof.張紹光 from department of veterinary medicine, NTU. He gave us suggestions on the mechanics of Pokecan and done many works that he thinks too dangerous for us to do (cutting holes in the wood and tubes, making the special connectors between the can and the tube). Without him, we may have been stuck in many mechanics problems now. We've gotten ideas about choosing components for different applications (which motor, which IC, which battery, …) and making mechanical structures (pulleys, locking things together). Next time we're going to make something requiring integration between hardware and software, we're sure that we can do it better!

References

1. https://cdn-shop.adafruit.com/datasheets/MCP23008.pdf

2. http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf
3. https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf
4. http://www.melexis.com/Asset/DataSheet-old-118-DownloadLink-4816.aspx
5. http://www.ti.com/lit/ds/symlink/l293.pdf
6. http://atceiling.blogspot.tw/2014/02/raspberry-pi-l293d.html
7. http://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi
8. https://github.com/rm-hull/ssd1306
9. https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library
10. https://github.com/adafruit/Adafruit_Python_GPIO
11. https://pymotw.com/2/socket/uds.html
12. https://github.com/nlohmann/json