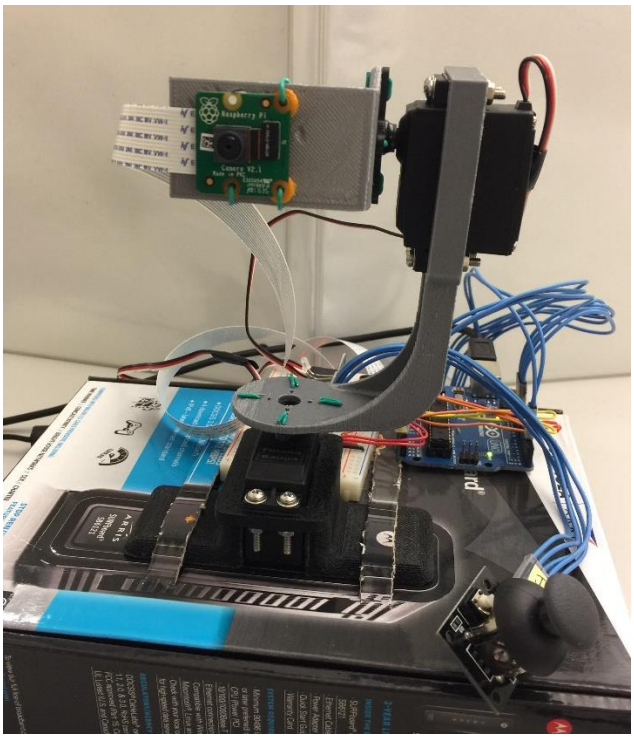
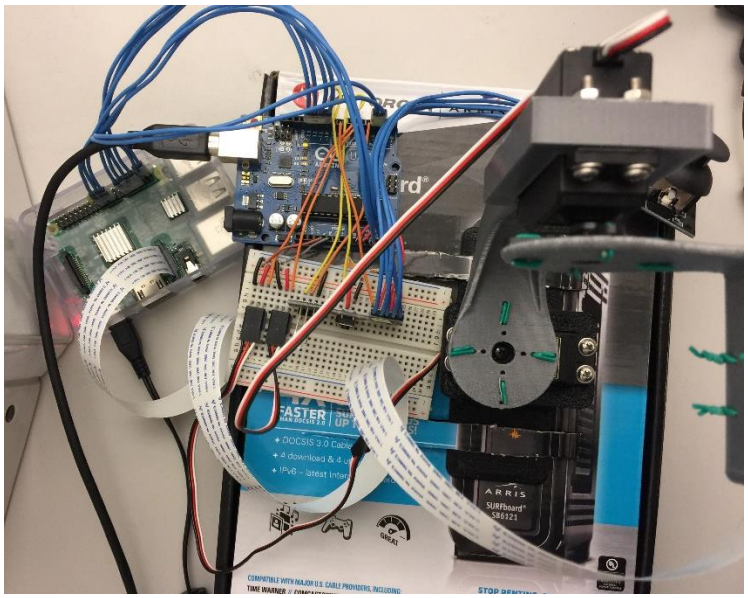
Face Tracking Camera
Tianchang Gu

Introduction

The purpose of this project is to implement a camera system that can track a face physically. The motivation of this project came from the possible need of tracking pet at your home, face centering during a live broadcast and video chat. With further hardware and software improvement, this project could lead to development of robotic eyes that simulate human eyes.





Related work

There are tons of project using OpenCV to do object detecting[1].There is one project on the website doing the face tracking camera [2], but the tracking is pretty slow and delayed because the author was using the Raspberry Pi controlling a pair of mini servos. Both RPi software PWM and mini servos lead to an inaccurate and low resolution (1.8 degrees) controlling. Many commercialized cloud camera with remote controls or motion detections[3]. With the existing face tracking camera project [2], I know the concept is achievable, and my implementation would only be better.

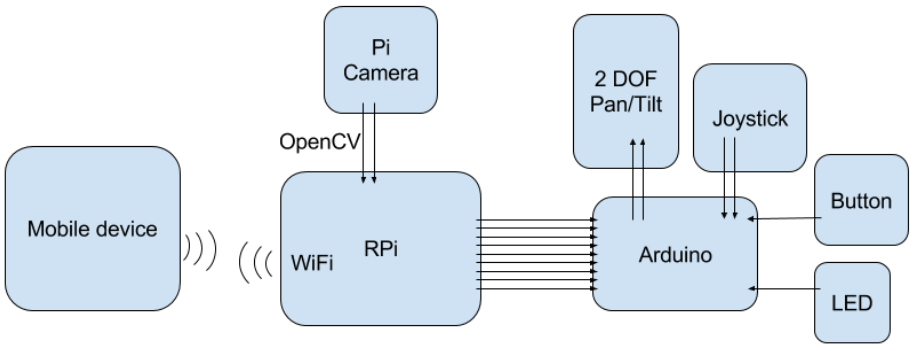
Hardware implementation

I used several very common hardware components that are available on Amazon, like the following.

1. Two standard-sized Futaba ball bearing servos
2. 3D printed mounting parts [4]
3. Arduino Uno
4. Raspberry Pi 3
5. PiCamera 2 module
6. LED, button, joystick sensors

The reason of using standard-sized servos is to gain the benefit of fine servo step control and further the speed control. Since RPi GPIO uses software PWM, which is not accurate at all, it is not correct to use RPi controlling servos. I decide to use Arduino to control servos with real PWM signals. Since PiCamera is native on the Raspberry Pi, I believe the connection speed and reliability are much better than normal USB camera.

In some case that the camera goes to an extreme position and it is not correct to rotate the servo by hand due to the possible damage to the hardware, I use LED and button to switch the mode between autonomous tracking and manual control, and I use the joystick for manual control and position reset.

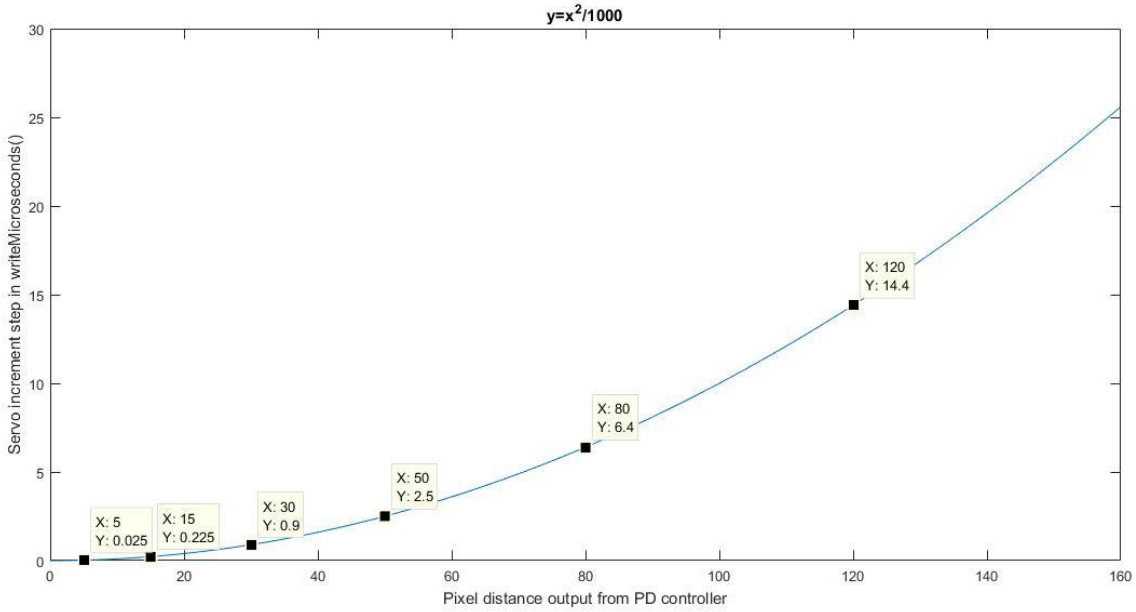


Wiring and Transmission

For the data transmission from Raspberry Pi to Arduino, I use totally nine wires (8 normal pinouts and 1 interrupt pinout). The reason of using binary transmission will be explained in the section of issues and challenges. The following shows the wiring specification.

- 1 Bit of x-direction
- 1 Bit of y-direction
- 3 Bit for x speed
- 3 Bit of y speed
- 1 Bit for interrupt that indicates face detected

Pixel distance between the center of detected face and center of the image is passing into a PD (proportional and derivative) controller, the output of the PD controller is quantized using a 3 Bit second order linear quantizer.



As you can see in the figure, larger pixel distance will lead to larger servo increment step, and as the servo moves towards to the target, the servo increment step will decrease dramatically to slow down the speed.

Software platform and tools

The software tools I used are Python IDE, OpenCV-python 3.0.0 [5][6], Arduino IDE, and MJPG Streamer [19][21]. To properly install the packages, I followed the instructions in the corresponding links.

Hardware and software integration

Starting from the Raspberry Pi, I am feeding my program with the 640x480 image at framerate of 40 fps. For each frame, front faces, right profile faces or left profile faces are searched in the sequence if the previous search has no result. In the case of no face found, the program will continue with the next available frame. When there are faces found, the program will go through the list of available faces and pick the largest face as the target of tracking.

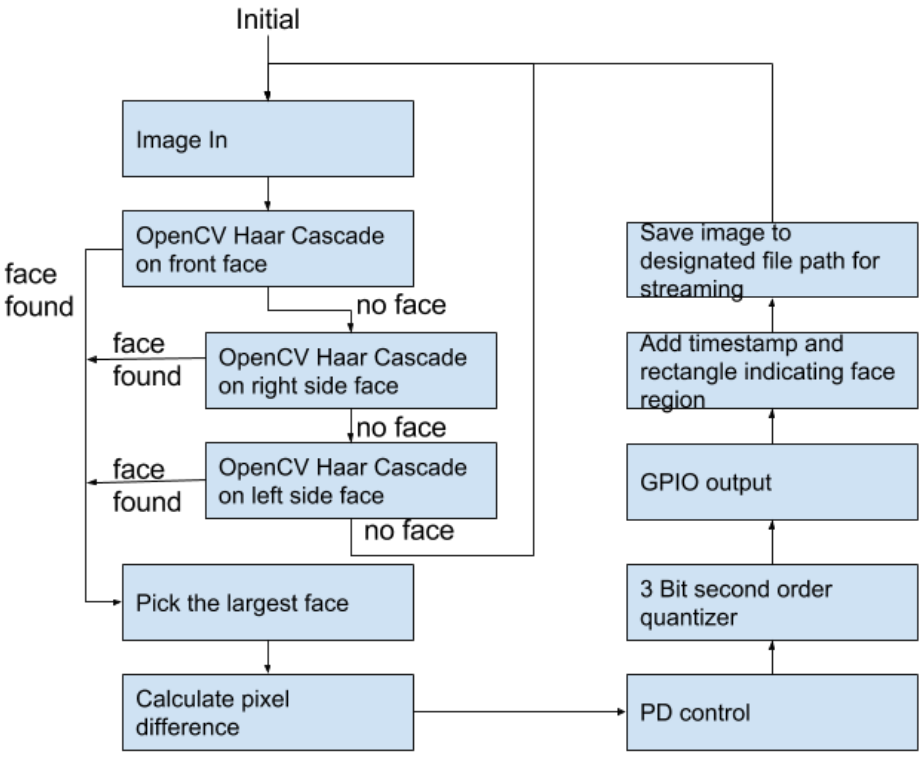
After there is a target face, the horizontal and vertical distances from the center of the face target to the center of the image are calculated, noted as deltaX and deltaY. Then the difference between current deltaX, deltaY and deltaX, deltaY in the previous frame are calculated, noted as diffX and diffY.

Then deltaX, deltaY, diffX, and diffY are inputted into the PD (proportional and derivative) controller. The output distX and distY are then inputted into a 3 Bit second order linear quantizer as mentioned before.

The quantized outputs distX and distY (encoded in binary) are transmitted to Arduino using raspberry pi GPIO python library. This signal will be used by Arduino to control the servo steps, in another term, the speed. Depend on the sign of deltaX and deltaY, 2 Bit signal contains the x-

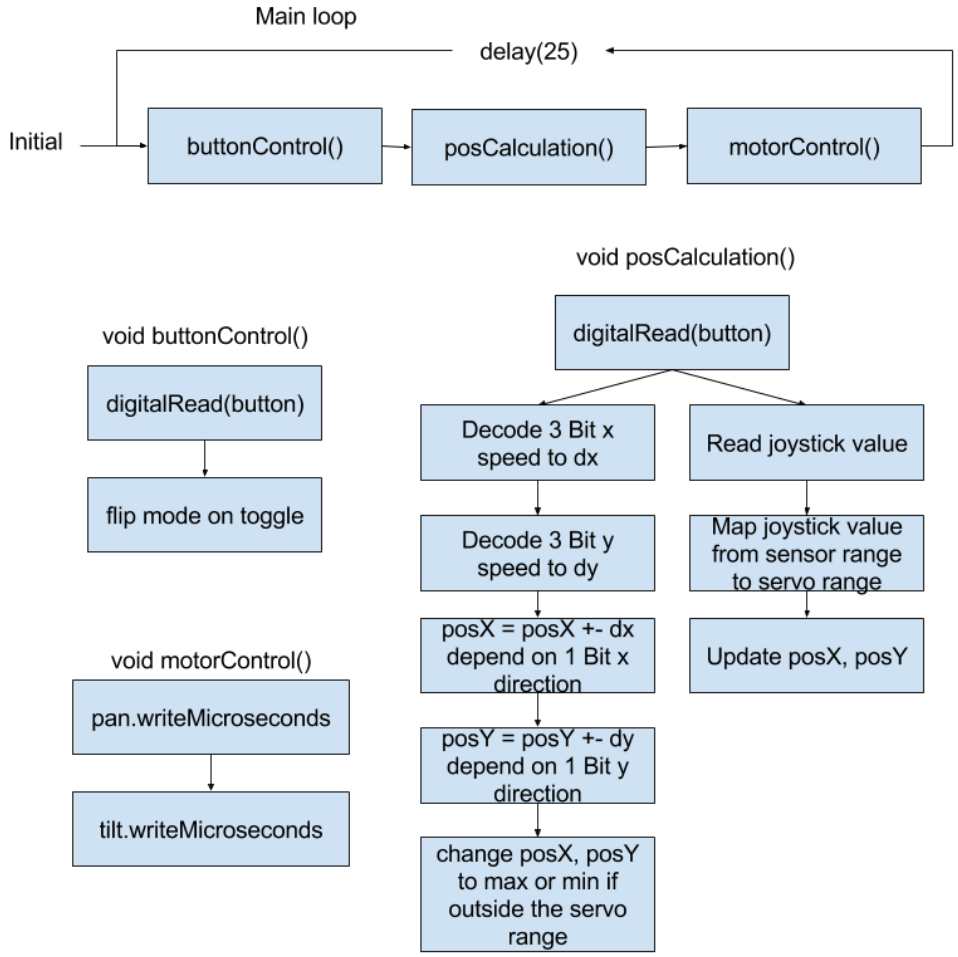
axis, and y-axis servo directions are transmitted to Arduino. Then, a Bit channel is toggled to indicate the Arduino there is a face found so the Arduino can update the servo position.

In the end, a timestamp and a rectangle of the target face region are added to the frame. The frame image is saved as a jpg image to a location designed in the MJPG Streamer.



In a separate terminal, the MJPG Streamer is launched with specified jpg image location. Anyone on the same WiFi network can access to the live streaming via the address http://<ip-address-of-RPi>:8080/stream.html.

In Arduino, buttonControl, posCalculation and motorControl functions run at a rate of 0.025 sec. With the looping rate of 0.025 sec, I found my servo step sizes work smooth and give a quick tracking response. The button changed the state of autonomous tracking to manual controlling and vice versa. The LED indicated when the auto tracking is on. In manual control, Arduino read the analog value of the joystick and map the value to the range of servo movement. With a click on the joystick, the servo moved to a specific position.



Issues and challenges

The hardest issues I ran into were communication between RPi and Arduino, and the calibration of the feedback control.

In the first issue, I tried sending servo steps and direction data via software PWM in the raspberry pi python library. Because the PWM signal is actually digital signal and the analog pin on the Arduino can not read it as an analog signal. I tried to measure the pulse width and calculate the PWM duty cycle using interrupt pin. The result was that the soft PWM signal from RPi was not accurate at all. With a fixed PWM setting, the measurement on Arduino varied more than 10% in duty cycle.

Then I try sending servo controlling data (9 ASCII chars at a time) via USB serial port. The solution worked fine, however, the delay in the tracking response was obvious and uncomfortable. Occasionally, due to the failure of receiving serial data, the servo failed to slow down near the target and move to extreme position quickly. Also, because serial write took more CPU resources (large data package) than GPIO output, the on-screen monitoring froze sometimes. Those were the reasons that I choose parallel data transmission via GPIO.

In the second issue of calibration, I was using first order linear quantizer to quantize the pixel distance and just use proportional control. It turned out with unhappy overshoot and unstable behaviour near the target. Because the OpenCV will never give a statical face detection result on a statical face, I decided to use PD control to increase the stability near the target while keeping

the accuracy of the face centering. Then I went back to the theory and decided my response should look like a 2nd order linear curve, so I chose the quantized level based on the 2nd order quantizer to implement the functionality of accelerating and slowing down. It resulted in very quick and accurate tracking behaviour.

Experiments and results analysis

1. Experiment one was to measure the looping rate in the Arduino and RPi. I calculated and printed the time difference between the current timestamp and previous iteration timestamp. The results are the following.

- Arduino looping rate: 0.025443 seconds based on 1509 samples
- Raspberry Pi looping rate: 0.186 seconds based on 559 samples
- Arduino position updating rate: 0.216236 seconds based on 2359 samples

Arduino looping rate is not very meaningful since I was adding 25 microseconds delay in the loop. That value was picked simply because it worked well with the quantized servo step values.

Even though I was feeding 40 frames per second, Raspberry Pi looping rate was 0.186 seconds (5.4 frames per second) due to the time cost of face detection. Based on the fact that Arduino position updated at the rate of face detection, which should be close to RPi looping rate, the data transmission between RPi and Arduino was very fast.

2. Experiment two was to measure the live streaming delay. I showed the frames on the RPi monitor and showed the live streaming on my laptop next to the monitor. Then I took 10 photos with several seconds interval. I calculated the time difference between the two timestamps and found the average.

- Streaming delay: 0.295 second

The streaming delay is about one and a half iterations in Raspberry Pi. I would say it was a short delay comparing to the VLC video streamer.

3. Experiment three was to measure the tracking response delay. I covered my face with my hand and moved myself to the position that was farthest to the close but still had the whole face in the monitor. I showed my face and started the stopwatch at the same time. Moreover, I stopped the stopwatch when I saw myself at the center of the image. This method could give me a rough idea of my tracking response. I repeated this experiment for 17 times. The result is the following.

- Tracking response delay: 0.643 second (std=0.218)

I would say the tracking response was pretty fast, thanks to my PD controller and quantizer.

4. Experience four was to measure the target accuracy. In this experience, I physically maintained my head in a statical position, and I printed out the pixel difference between the face center and image center, deltaX and deltaY. Then I calculated the mean and standard deviation on the samples. The results are the following.

- Average pixel difference in x axis:7.69 pixels (std=1.80) based on 198 samples
- Average pixel difference in y axis:3.51 pixels (std=2.21) based on 206 samples

The results show the target accuracy was very good.

Conclusion

As I proposed in the project proposal, I want to implement an auto tracking camera with all kinds of delay under 1 second. Also, I want to give the user a fluent user experience. I believed that I

succeeded. The most important parts of my project were controlling and data communication. With the camera, servo, circuit and PD control, my system was a typical embedded system.

During the demo in the lab, I realized that my system could improve on the target selection (a specific face) due to the fact my camera looked at different faces. I want to improve the controlling behaviour with PID controller. Also, the face detection speed could be improved with GPU acceleration so that the whole system could speed up. Lastly, my pan/tilt mechanism introduced vibration which could be improved with a better 2-DOF joint mechanical design.

References

1. http://www.instructables.com/id/RasPi-OpenCV-Face-Tracking/
2. http://www.instructables.com/id/Pan-Tilt-face-tracking-with-the-raspberry-pi/
3. https://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Daps&field-keywords=cloud+camera
4. http://www.thingiverse.com/thing:2066115
5. http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/
6. http://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/
7. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection
8. https://www.raspberrypi.org/documentation/usage/camera/python/README.md
9. http://picamera.readthedocs.io/en/release-1.13/
10. https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/
11. https://sourceforge.net/p/raspberry-gpio-python/wiki/Outputs/
12. https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/
13. https://www.arduino.cc/en/Tutorial/PWM
14. https://www.arduino.cc/en/Reference/AnalogRead
15. http://www.benripley.com/diy/arduino/three-ways-to-read-a-pwm-signal-with-arduino/
16. https://www.arduino.cc/en/Reference/attachInterrupt
17. https://oscarliang.com/connect-raspberry-pi-and-arduino-usb-cable/
18. http://robotic-controls.com/book/export/html/10
19. https://blog.miguelgrinberg.com/post/how-to-build-and-run-mjpg-streamer-on-the-raspberry-pi
20. https://www.raspberrypi.org/forums/viewtopic.php?t=72918&p=524934
21. https://blog.miguelgrinberg.com/post/stream-video-from-the-raspberry-pi-camera-to-web-browsers-even-on-ios-and-android
22. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html
23. https://media.readthedocs.org/pdf/opencv-python/latest/opencv-python.pdf
24. http://stackoverflow.com/questions/36218385/parameters-of-detectmultiscale-in-opencv-using-python
25. https://www.arduino.cc/en/Reference/ServoWriteMicroseconds
26. https://www.linux.com/learn/writing-simple-bash-script
27. http://linuxcommand.org/writing_shell_scripts.php
28. http://superuser.com/questions/176783/what-is-the-difference-between-executing-a-bash-script-vs-sourcing-it/176788#176788
29. https://www.maketecheasier.com/run-bash-commands-background-linux/
30. http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html#puttext
31. http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html#cascadeclassifier-detectmultiscale