Facetracking Automated Camera Experiment (FACE)

CS122A: Fall 2018

Taylor Che

Github:

https://github.com/tche001/CS122A-Final-Project

Table of Contents

Introduction **2**

Hardware **3**

Parts List 3

Block Diagram 3

Pinout 4

Software **5**

Implementation Reflection **7**

Milestones 7

Milestone I 7

Milestone II 7

Completed components 8

Incomplete components 8

Youtube Links **8**

Testing **9**

Known Bugs **10**

Resume/Curriculum Vitae (CV) Blurb **10**

Future work **10**

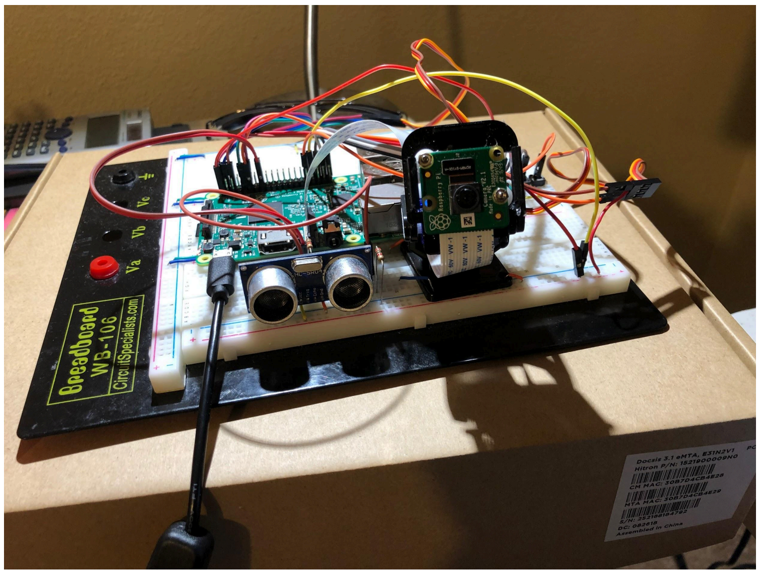
References **10**

Appendix **11**

Introduction

The objective of this project is to develop a face tracking system to keep the user's face in the centered on the camera's frame once they are in range. Additionally, a manual mode was added for debugging and repositioning of the frame. This project utilizes a Raspberry Pi 3B, as OpenCV's Haar cascade algorithm, among other face detection algorithms, are too computationally intensive for smaller or lower powered microcontrollers. The accompanying peripherals for this system are two servos, a Raspberry Pi camera, an ultrasonic sensor for distance detection, and four buttons for manual control.

The execution of the program is as follows: the ultrasonic sensor is polled, detecting if an object or person is within 20 centimeters of the camera. If the sensor finds an object in range, the face tracking algorithm is executed, searching for a face. Upon finding a face, the OpenCV camera output draws a green square around the found face and calculates the difference in the X and Y coordinates of the center of the camera compared to the center of the generated square. This difference is used in determining how far to turn and pan the camera to center the face on the camera's frame. However, pressing the to the manual movement buttons overrides any currently running process to move in the inputted direction. The entire program executes until cancelled in the terminal of the Raspberry Pi.

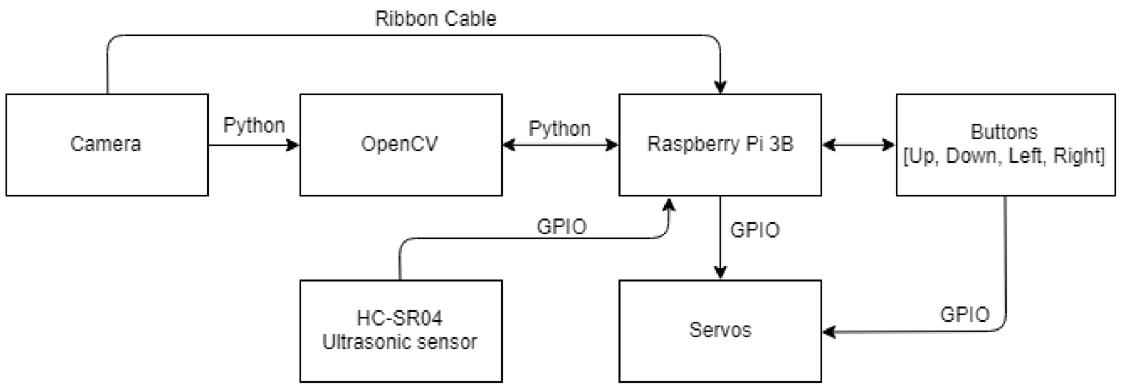


Hardware

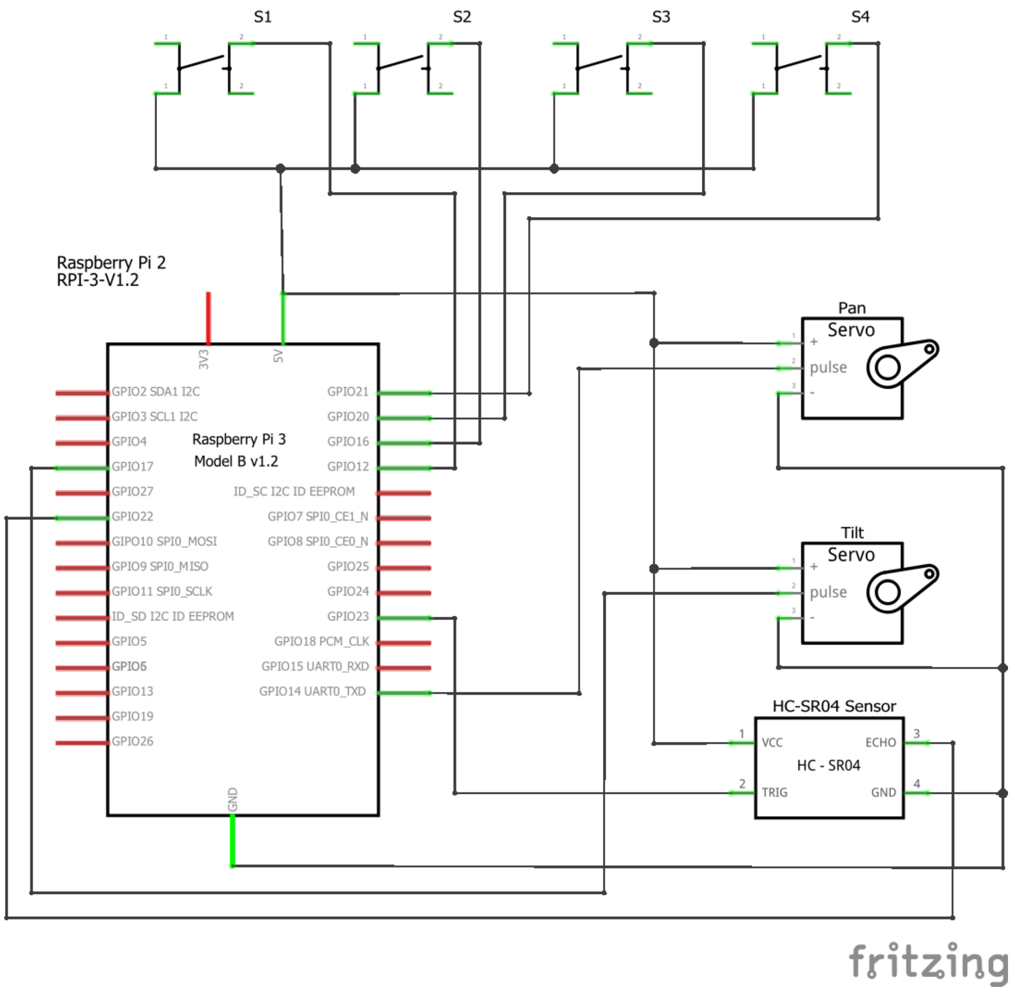
Parts List

Part	Part #	Quantity
Raspberry Pi 3B	Raspberry Pi 3B	1
Servo	SG-90	2
Raspberry Pi Camera	V2.1	1
Ultrasonic Sensor	HC-SR04	1
Button	Button	4

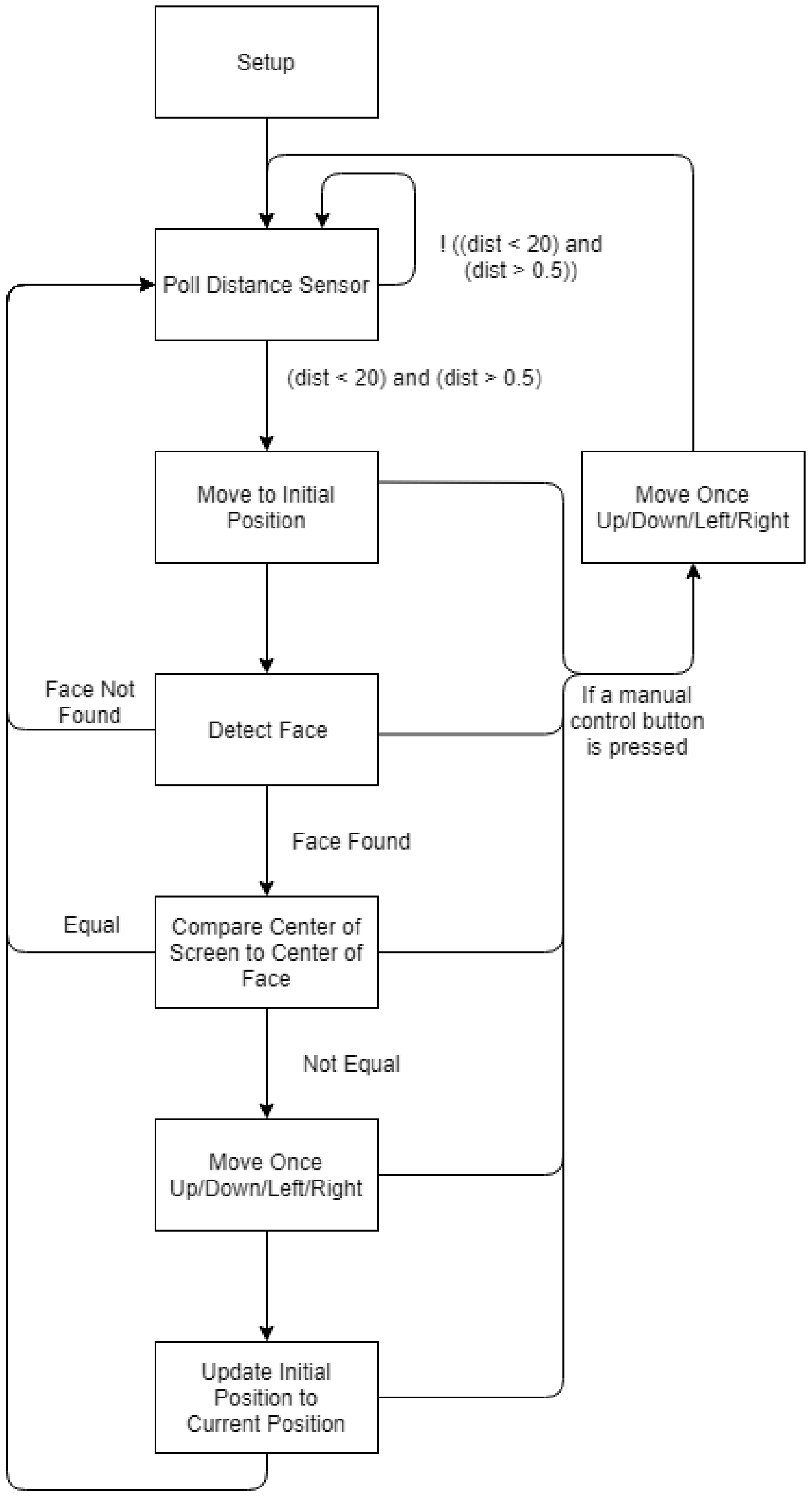
Block Diagram



Pinout


S1
S2
S3
S4
Raspberry Pi 2
RPI-3-V1.2
3V3
5V
Raspberry Pi 3
Model B v1.2
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4
GPIO17
GPIO27
GPIO22
GIPO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
ID_SD I2C ID EEPROM
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
GPIO21
GPIO20
GPIO16
GPIO12
ID_SC I2C ID EEPROM
GPIO7 SPI0_CE1_N
GPIO8 SPI0_CE0_N
GPIO25
GPIO24
GPIO23
GPIO18 PCM_CLK
GPIO15 UART0_RXD
GPIO14 UART0_TXD
GND
Pan
Servo
pulse
Tilt
Servo
pulse
HC-SR04 Sensor
HC - SR04
VCC
ECHO
TRIG
GND
fritzing

Software Flowchart



Setup
Poll Distance Sensor
! ((dist < 20) and (dist > 0.5))
(dist < 20) and (dist > 0.5)
Move to Initial Position
Move Once Up/Down/Left/Right
Face Not Found
Detect Face
If a manual control button is pressed
Face Found
Equal
Compare Center of Screen to Center of Face
Not Equal
Move Once Up/Down/Left/Right
Update Initial Position to Current Position

Task Functionality Description:

- Setup

  In this task the GPIO pinouts are defined for usage, the OpenCV video output is created, the camera is initialized, and the servos move to their initial position.

- Poll Distance Sensor

  The HC-SR04 is polled in this task, but requires time to process the results. First the sensor is flushed by holding trigger is held high for 0.01 milliseconds and the time function is tested. Next, a counter is created to prevent a infinite loop created by the sensor not detecting distance correctly. After, the elapsed time is calculated and the distance is calculated using the speed of sound divided by 2

- Move to Initial Position

  This function is used to ensure that the motor is in the correct location.

- Detect Face

  This task handles the Haar cascade face tracking, where a frame is read from the video stream from the camera, an attempt to find the face is made. If a face is found, OpenCV draws a green rectangle around the face. From the coordinates of the rectangle, the center of the rectangle is calculated and returned. If a face is not found, the function skips the previous step. Regardless if a face is found or not, the processed frame is output to an OpenCV window.

- Compare Center of Screen to Center of Face

  Here the result of detect, the center of the user's face, is compared to the center of the camera's screen in both X and Y directions. The resulting difference is used to determine the direction in which to turn the camera.

- Move Once

  The difference calculated in the previous task is used to pan or tilt the camera one step in the determined direction. This uses the move function which sends the ServoBlaster library which servo to turn, and how far to turn.

- Update Initial Position to Current Position

  In this task, the current position is set as the new initial position which is passed to the *Move to Initial* task to ensure the servo is in the correct position when the next tick is processed.

- Move Once [Up, Down, Left, Right]

  At every tick, the "pull up" buttons are polled, checking the status of each button. If an input is detected, the program is interrupted to move the servo one step in the indicated direction until the button is depressed.

Implementation Reflection

In my opinion, my project met the both the milestone goals set at the basic level. Most of the functionality was implemented successfully, barring a few issues and redesigns. The core feature of the project, face tracking, works perfectly in most lighting conditions and the servos do respond as expected, albeit at a slower rate than expected. The tertiary peripherals functioned correctly as well, the ultrasonic sensor, buttons, and joystick provided the correct output for the project.

However, the project could have been further optimized. I attempted to speed up the face tracking algorithm through threading, using knowledge from the GPU class I previously attended, but I could not figure out how to utilizes the threads on the Raspberry Pi in Python. In addition to increasing the speed of the program, I would have replaced the buttons with a different joystick implementation using an ADC to interface with the Pi.

The portion of my project that I liked the most was the Haar cascade detection. I was intrigued in furthering my understanding of image processing and how object and image detection functions. Additionally, I have never worked with servos before, so interacting with simple and accurate motors was interesting to say the least.

Milestones

Milestone I

For this milestone, basic functionality and features were to be built and partially implemented. The servos were mounted to a breadboard and the servo movements were programmed with servoblaster. The camera was tested with OpenCV and Haar cascades were implemented and integrated with movement.

A function of manual mode was attempted using a joystick and USART connecting the Raspberry Pi and Atmega1284, but did not work due to an issue with the baud rate of the Raspberry Pi being heavily dependent on its current voltage. The solution to the issue was to replace the joystick with four buttons. A simpler alternative would have been to use an MCP3008 ADC to retain the joystick functionality.

Milestone II

The object of this milestone, was to implement live video tracking, a trigger to start the automated program, and multi face detection. The implementation of live video tracking improved the framerate of the video playback since, in the previous milestone, the camera would capture a frame from the camera instead of from a video stream, adding considerable delay due to the camera having to restart for every capture. The trigger for the system is a ultrasonic distance sensor, constantly polling the distance between the sensor and closest object and preventing the camera portion of the project from executing. I did not complete the multi face tracking portion of the project, as I felt it was too ambitious of an objective. To complete that portion of the project, I did not have the time to commit to reworking the Haar cascade function to center multiple faces.

Completed components

The majority of the project was completed, as facetracking in OpenCV, a method of triggering the program was implemented, and servo control was achieved. However, the methodology has changed for certain parts, such as the joystick portion where buttons were substituted. The missing parts of the project was multiple object detection and the joystick.

Incomplete components

I did not complete multi face detection that I set out to complete in milestone II, due to my underestimation of how difficult determining the center of the two faces would be. Currently, the Harr cascade detects multiple faces, but only tracks the center of a single face. The program determines the center of the detected face using the vertices of the rectangle generated from the cascade. From my testing, I found that the function only returns the center value of the first detected face and does not interact with any following detected face. Given a few more days, I believe that I could fix this issue by doing an additional pass of the detected faces and averaging the resulting “centers”.

In addition to not completing the multiple face detection I could not figure out how to interface USART and SPI between the Atmega 1284 and Raspberry Pi. I tried many troubleshooting options to fix this issue, but have not been able to figure out how to operate the Raspberry Pi as a slave, receiving joystick input from the Atmega 1284. From my understanding, the baudrate of the Raspberry Pi’s GPIO ports varies with voltage, thus with any shift of voltage, whether it being connecting to wifi or the activation of servos, the baud rate between the two controllers become unsynchronized and the data sent would become corrupted or lost. My current solution to this issue is to replace the joystick with 4 pull down buttons, simplifying the process. However, a future solution would be to supplement the Raspberry Pi with a MCP-3008 ADC, which the RPi lacks, allowing the joystick to directly communicate with the Raspberry Pi. Integration into the system would take only a few hours as there exists documentation of using the MCP-3008 specifically for a joystick.

Youtube Links

- Overview Video:
    - https://www.youtube.com/watch?v=5MhvC-dVeIQ
- Detailed Video:
    - https://youtu.be/HG4MHmVDAbg

Testing

- Camera

    The camera was tested using the function *Raspistill* every startup, ensuring that the camera was detected and correctly attached to the board. An issue in testing I encountered was the camera's driver, *bcm2835-v4l2*, not running during boot. This was mediated by editing the modules file to include the driver, to ensure it started correctly.

- Raspberry Pi

    The Raspberry Pi was a challenge to set up correctly for this project, as many libraries needed to be installed. Additionally, my unfamiliarity with setting up virtual environments made it a challenge to test the installed libraries. This was because certain libraries were reliant on an out of date version of another, while another library relies on the most up to date version. After flashing my SD card multiple times, I managed to find a stable setup from which to base my project from.

- Servos

    The servos were fitted into a mount and tested individually utilizing ServoBlaster. By creating programs to test pan and tilt functionality simultaneously. The ServoBlaster library made servo interfacing trivial since the library was robust.

- HC-SR04 Ultrasonic Sensor

    The ultrasonic sensor was challenging to interface, as it operates at 3.3V logic, and the rest of the project operates at 5V. This was tested by running a specific program repeatedly whilst changing the values of the voltage divider used to step down the project until the results of the sensor stopped outputting errors.

- Joystick

    The joystick attached to the Atmega1284 was debugged by attaching LEDs and assigning values to X and Y input ports and associating them to four LEDS to indicate what direction that the joystick is currently in. USART was tested on the Raspberry Pi by connecting the Pi's Rx and Tx pins together and running a program that sends values through them while polling the Rx pin for received values. Testing the Atmega 1284's USART was done by reading values on the second microcontroller and assigning it to the LEDs as was done earlier.

Known Bugs

- HC-SR04 Ultrasonic Sensor

  The bugs with the ultrasonic sensor were issues with detection as well as indefinite polling. The ultrasonic sensor would occasionally output a value close to zero or a value around 2,000. This was probably caused by the program polling the sensor too fast and not clearing the GPIO pins after every execution, such the rebounding signal is lost until erroneously received during the next tick. This was somewhat fixed by filtering out values that are too high or low for use in the program. In addition to this, a counter was added to break out of the distance function if it counted past 2,000ms, effectively preventing the program from getting stuck infinitely within the loop. With more time, completely rewiring the project may result in fewer errors, but due to the inherent low quality of the sensor itself. Thus, without spending additional money on a better sensor, the errors are inevitable.

Resume/Curriculum Vitae (CV) Blurb

The Facetracking Automated Camera Experiment (FACE) is an automated camera with the objective is to constantly center the user's face in the frame of the camera. This is based on the Raspberry Pi platform in Python utilizing Haar cascade face detection in OpenCV and two servos to pan and tilt the camera to keep the face centered in the camera's frame. This project utilizes a ultrasonic sensor to trigger the automatic face tracking if it detects an object or a user within 20 centimeters of the sensor. Additionally, four buttons can directly control the servos, panning or tilting them one degree every second.

Future work

In future iterations of the project, this project would feature multi-face tracking as well as be exported to an FPGA to speed up its processing speed, as it is currently running very slowly. Additionally, by using a different sensor to trigger the automated portion of the code, it would further increase the speed of the program. Since this project has few peripherals, encasing the project would simple. This type of project would be useful for the company I did my research project for, since Raytheon uses servos in many of their projects, as well as object tracking and detection.

References

OpenCV: Face Detection using Haar Cascades
https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html
ServoBlaster: PiBits
https://github.com/richardghirst/PiBits/tree/master/ServoBlaster

HC-SR04 Ultrasonic Sensor: modmypi.com
https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

Using PullUp and PullDown Resistors on the Raspberry Pi:
https://grantwinney.com/using-pullup-and-pulldown-resistors-on-the-raspberry-pi/

Getting Started with Videos: OpenCV
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html