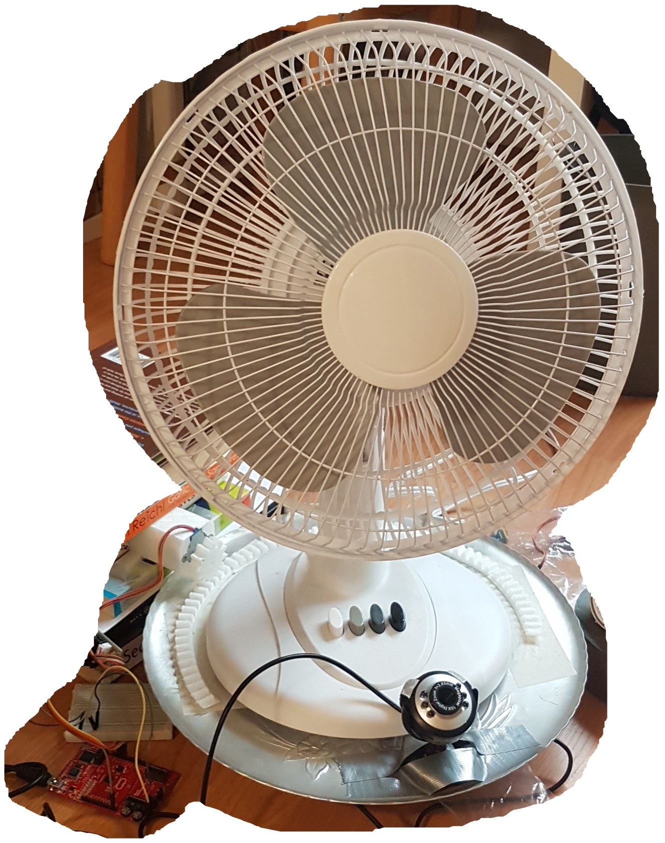
MECH423 Final Project: FanFaceTracker



Ratthamnoon Prakitpong
rprakitpong@gmail.com

0. Abstract:

The goal of this project is to make a fan to automatic directs the air from the fan to a moving target. The python software processes camera image using opencv and send movement commands to center the face found in the frame, or the user can send direct movement command using tkinter GUI. Movement command is a single byte of number of steps to step. The microcontroller unit receives commands via UART, scales up number of steps, then steps stepper accordingly. Pinion on stepper motor shaft turns rack on turntable, which adjusts the direction which the fan and the camera point to. Put together, the camera image capturing and processing features form a feedback control loop to control angular position of the turntable and the fan.

1. Objectives

The goal of this project is to make a fan to automatic directs the air from the fan to a moving target. I will design and execute the turntable's motor interface, motor software (firmware and controls), and computer vision software. I accomplished these goals as is set out in the proposal.

2. Rationale

Computer vision is an increasing popular and accessible method of gaining input signal, from self-driving cars to automation. However, it is not taught as a part of the mechatronics curriculum. This project would provide a good personal introduction to using this software tool.

While many past MECH423 projects have used CV, they tend to have an extra layer of complexity missing in this project. For example, the automated nerf gun has to do projectile calculations and face tracking. Another example is air hockey, which required 2-axis controller and a fast response time. This project simply turns the direction of the fan towards the target. This allows for reduced scope, appropriate for limited amount of help I can get from working online, and for a one-person project (instead of two like past years).

3. Summary of Functional Requirements

Functional Requirements	Brief Description	Technologies and Tools
FR#1: Fan platform with motor connection (mechanical)	Turntable that fan and camera sits on. A rack-and-pinion assembly is used to transfer movement from stepper motor to turntable.	SolidWorks, Blender, 3D printing, hand tools.
FR#2: UART Controlled stepper motor (electrical)	MCU receives bytes via UART and send step signals to pins connected to h-bridge, used to control stepper motor.	Microcontroller, stepper motor, h-bridge, electronic parts.
FR#3: Tracking software using computer vision (software)	Consists of three classes. Machine class, using serial, sends byte via UART to move motor. GUIcontroller class, using tkinter, displays angular displacement and has buttons to send movement command. Cvcontroller class, using opencv, gets image from camera and send movement commands till face (if in frame) is centered.	Python, numpy, cv2 (opencv), serial, tkinter, threading.

Table 1: Summary of functional requirements

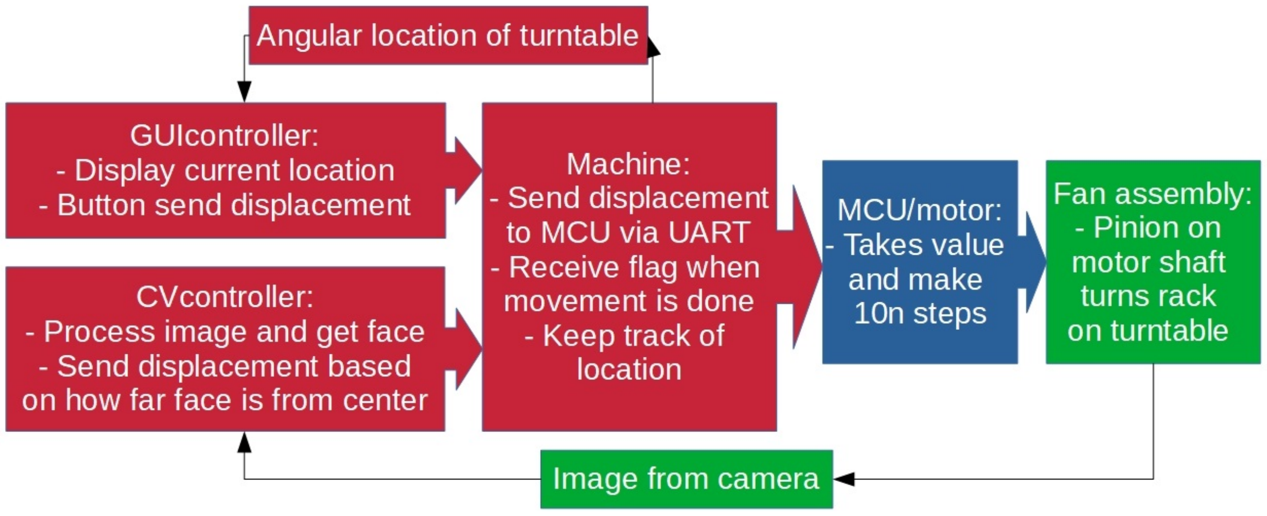

Angular location of turntable
GUIcontroller:
- Display current location
- Button send displacement
CVcontroller:
- Process image and get face
- Send displacement based on how far face is from center
Machine:
- Send displacement to MCU via UART
- Receive flag when movement is done
- Keep track of location
MCU/motor:
- Takes value and make 10n steps
Fan assembly:
- Pinion on motor shaft turns rack on turntable
Image from camera

Figure 1: Flow chart of signals are transferred and manipulated in the system between modules

4. Functional Requirement #1

4.1 Approach and Design

This module controls the angle that which the fan is facing. The image shows each parts in this mechanical module.

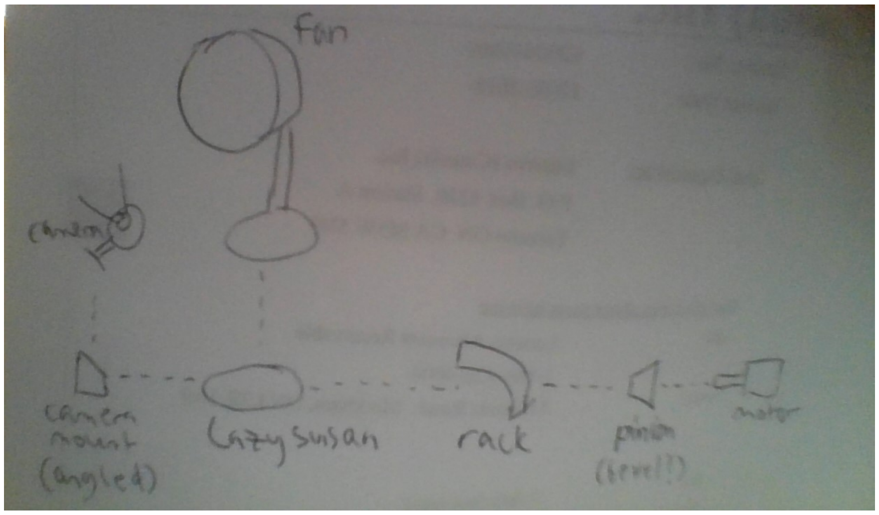


Figure 2: Mechanical components

Parts that are off-the-shelf are fans, motor, camera (camera mount came with camera), and turntable (Lazy Susan). Parts that need designing and manufacturing are rack (which is glued to turntable), pinion (which is attached to motor shaft), and motor mount (not pictured).

Since manufacturing is all done by 3D printing, we need stl files. To get rack and pinion 3D model, I used SolidWorks Toolbox, an add-on that allows user to give it parameters and generate parts based on those parameters. To get motor mount model, I found a mount for this motor on thingiverse.com.

To get them ready for 3D printing, I modified the stl files in Blender. For pinion, I did a boolean difference between it and the motor shaft's model (also from thingiverse.com), creating an appropriate hole for shaft to rest in. For rack, I did a boolean intersect between it and an arc model, cutting it into a slice like a cake, so that it's small enough to fit on 3D printer. For motor mount, I extended its base and added a flap for mounting. This process is summarized on Table 2.

After it's ready, it's printed on my personal printer. All parts are assembled using hand tools I have at home.

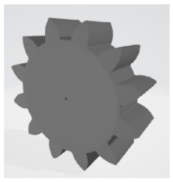
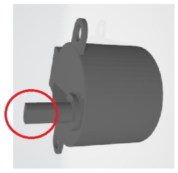
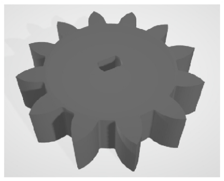
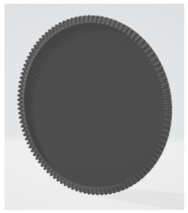
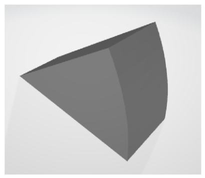
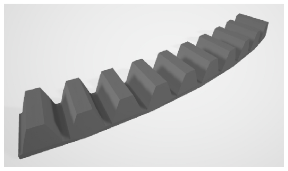
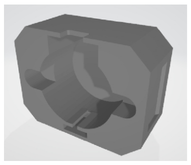
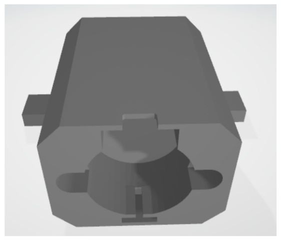
	Source stl models	+	Modifications		=	Final, 3D-print-ready models
Pinion	 SolidWorks Toolbox: RackTeeth=110, PinionTeeth=12 Module=3, PressureAngle=20deg		Boolean difference	 thingiverse.com/thing:151379		
Rack	 SolidWorks Toolbox: RackTeeth=110, PinionTeeth=12 Module=3, PressureAngle=20deg		Boolean intersect	 I designed this. Angle is 360*10/110 degrees.		
Motor mount	 thingiverse.com/thing:1215081		Extended and extrude part of back of mount.			

Table 2: Design steps for each part that needed designing

4.2 Inputs and Outputs

Input of this module is the rotation motion of motor shaft. Output is the angular position at which the fan is facing.

4.3 Parameters

Angular position of turntable and angular position of motor shaft are both measured in degrees. Since they are directed connected with rack and pinion, they're linearly proportional to step size and their relationships can be found (i.e. much many degrees turned per step). This calculation is discussed in detail in section 6.3. Angular position of turntable affects direction which fan and camera are pointed to; camera feed is fed back to controls software, which determines angular position of turntable. Module of rack and pinion is optimized such that they produce fine angular resolution, while remained easy to 3D print.

Since the rack is not fully circular, maximum travel is a mechanical parameter has to be considered when writing software; about 180 degrees was selected to give decent travel while not so much that cables are tangling from the movement. I ended up with a rack with 60 teeth, which is 196 degrees (360deg*60/110).

4.4 Development Plan

1. Acquire parts that are off-the-shelf.
2. Make measurements. Key measurements are:
   1. Turntable radius (use as rack size limit).
3. Get models available online:
   1. Stepper motor (use in pinion design process).
   2. Stepper motor mount.
4. Design custom parts:

    1. Rack.
    2. Pinion.
    3. Motor mount.
5. 3D print custom parts.
6. Assemble. Iterate if parts don't fit.

4.5 Test Plan

Off-the-shelf parts are likely working already, so testing will focus on designed parts.

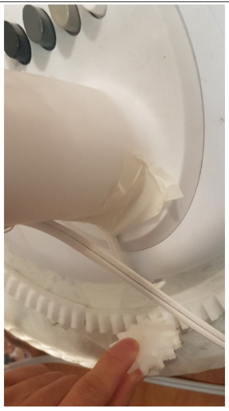
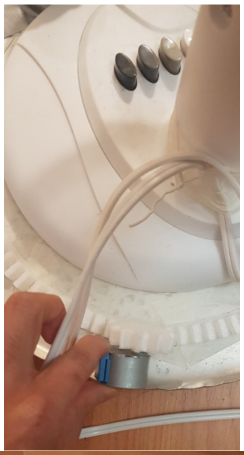
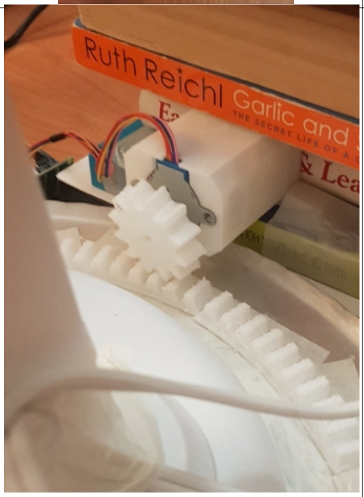
Part	Test plan	Successful if	Test observation	Test result
Rack	Use pinion to manually turn rack attached to turntable	Fan and turntable rotate without slipping or breaking		Pass: Has some slack due to non-industrial quality of turntable, but functionally fine
Pinion	Attach pinion to motor and use it to rotate turntable	Motor attached to pinion turns fan and turntable without slipping or breaking		Pass: Turned as expected
Motor mount	Put motor on motor mount and let the motor run	Positions motor such that rack and pinion teeth connect, and the turntable rotate when motor is on		Pass: Improved stability when weights are put on top of mount

Table 3: Test plans and results for mechanical module

5. Functional Requirement #2

5.1 Approach and Design
This module is used to control stepper motor speed and direction. The circuit is the same as Lab 3's exercise 2.

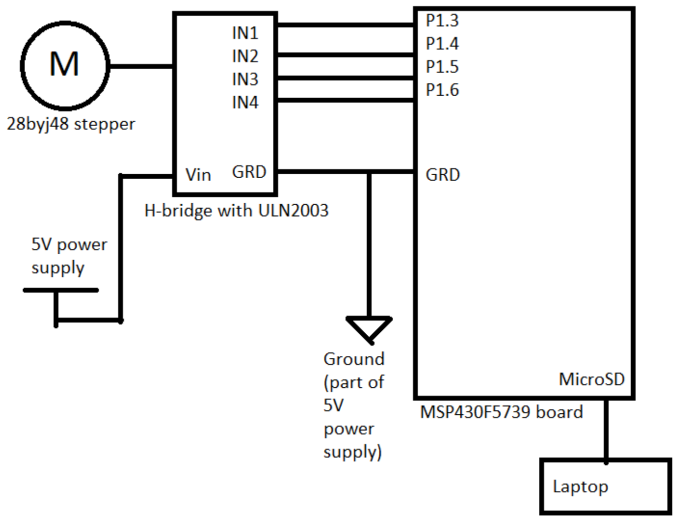


Figure 3: Electrical circuit diagram

The code used to run this stepper motor will be similar to the code in Lab 3's exercise 2 as well. It will be modified to receive number of steps and direction, instead of speed and direction.

5.2 Inputs and Outputs
The module outputs motor rotation according to the received commands. As input, this module receives bytes from UART communication, commanding number of steps to step. After movement is done, a byte value of 1 is returned as a flag to notify.

5.3 Parameters
An important parameter is value received to command number of steps. Ten times of difference between received byte and 127 is used as step count and direction for motor movement. For example, byte with value 90 sent will output 370 steps in the direction that will cause turntable to turn leftward (negative direction). It is formatted this way for optimized package size. If direction and maximum of 1270 steps are sent separately, it would require at least 4 bytes (1 for direction, 1 for upper bytes of step count, 1 for lower bytes of step count, 1 for escape byte), whereas this format needs only 1 byte received.

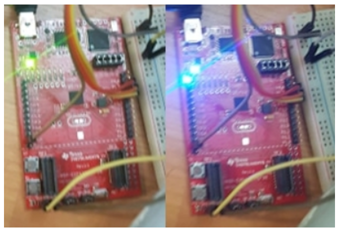
The stepper motor is controlled by signals from the board, so another parameter is stepping signals. To turn, we need to convert number of steps into signal to h-bridge. Since h-bridge has 4 input pins, I have 4 arrays that hold signal values. Stepping the motor meant looping through the array. The values in the arrays are for half-stepping; per the motor's hardware, a full revolution is 4096 half steps, which is why input signal needed to be multiplied by 10 to get a reasonable number of steps for stepping (127 * 10 = 1270 = ~1/4 revolution). Understanding motor hardware makes optimizing received package size doable.

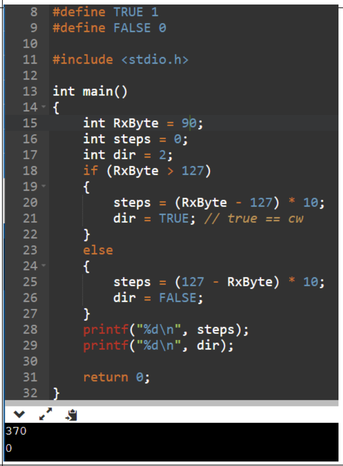
5.4 Development Plan

1. Assemble circuit according to diagram.
2. Write microprocessor code to:
    1. Send signal to h-bridge, based on half-step sequence of the stepper motor.
    2. Receive and store messages from the UART port.
    3. Parse messages to get number of steps to step the motor.
    4. Step motors accordingly.

5.5 Test Plan

Circuit assembly can be test by whether the wanted operations work or not. To test the code:

Step in development plan	Test plan	Successful if	Test observation	Test result
Send signals to h-bridge	Observe angular change of motor shaft	Motor shaft rotates by some hardcoded number of steps	Hardcoded 1024 steps, made a mark on pinion, observed that mark moved by about 1/4 turn	Pass: 1024 steps / 4096 steps = 1/4 turn
Receive UART messages	Toggle LED on message received	UART interrupt function is called after computer sends byte		Pass: LED toggled as expected

Parsing messages	Pause and see parameter in debug	Step count and direction are parsed from a byte and stored in some variables correctly	
			`#define TRUE 1`
			`#define FALSE 0`
			
			`#include <stdio.h>`
			
			`int main()`
			`{`
			`    int RxByte = 90;`
			`    int steps = 0;`
			`    int dir = 2;`
			`    if (RxByte > 127)`
			`    {`
			`        steps = (RxByte - 127) * 10;`
			`        dir = TRUE; // true == cw`
			`    }`
			`    else`
			`    {`
			`        steps = (127 - RxByte) * 10;`
			`        dir = FALSE;`
			`    }`
			`    printf("%d\n", steps);`
			`    printf("%d\n", dir);`
			
			`    return 0;`
			`}`
			`370`
			`0`
				Pass:
				Tested parser separately and it parsed various values correctly

Step motors from command	Observe angular change of motor shaft	Motor steps, and number of steps are successfully changed from hardcoded values to said variables' values	Observed pinion on motor shaft turn when received UART message	Pass: Pinion's angle change correlated to steps sent by command

Table 4: Test plans and results for electrical module

6. Functional Requirement #3

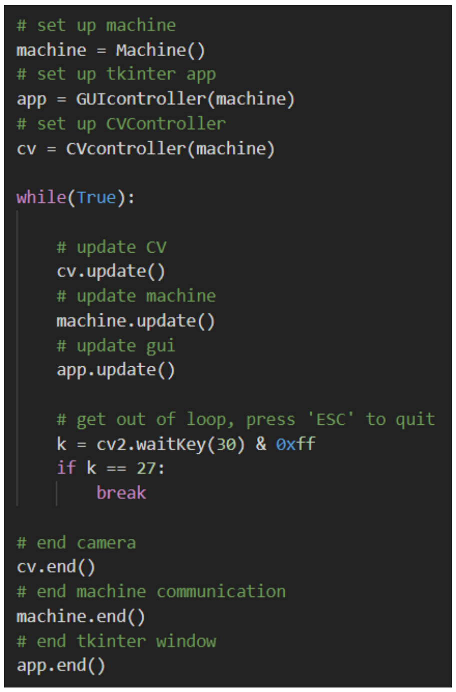
6.1 Approach and Design

This module's objective is to turn the motor such that the fan is directed towards a face, while also provide a manual way to move motors (i.e. buttons). The pseudo-code can be like shown:

```
CreateManualControlToggle()
StartUART()
while(1):
  if started:
    image = getImage()
    if face is in image:
      x, y = getFacePosition(image)
      x is left and motorIsNotMoving:
        sendUART(farther left, more negative steps)
      x is right and motorIsNotMoving:
        sendUART(farther right, more positive steps)
```

A refactoring is done after a functional iteration is finished, to improve readability and maintainability of the code. A sample method is to encapsulate related functionalities into classes.

In the final version, CV, GUI, and serial communication are refactored into their own class (Cvcontroller, GUIcontroller, and Machine, respectively). An instance of each class is instantiated, and their update method are called in a loop, and their end methods are called after loop is ended. The final design of the software module is shown in the UML class diagram below, along with the main code being run.



```
# set up machine
machine = Machine()
# set up tkinter app
app = GUIcontroller(machine)
# set up CVController
cv = CVcontroller(machine)

while(True):

    # update CV
    cv.update()
    # update machine
    machine.update()
    # update gui
    app.update()

    # get out of loop, press 'ESC' to quit
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

# end camera
cv.end()
# end machine communication
machine.end()
# end tkinter window
app.end()
```

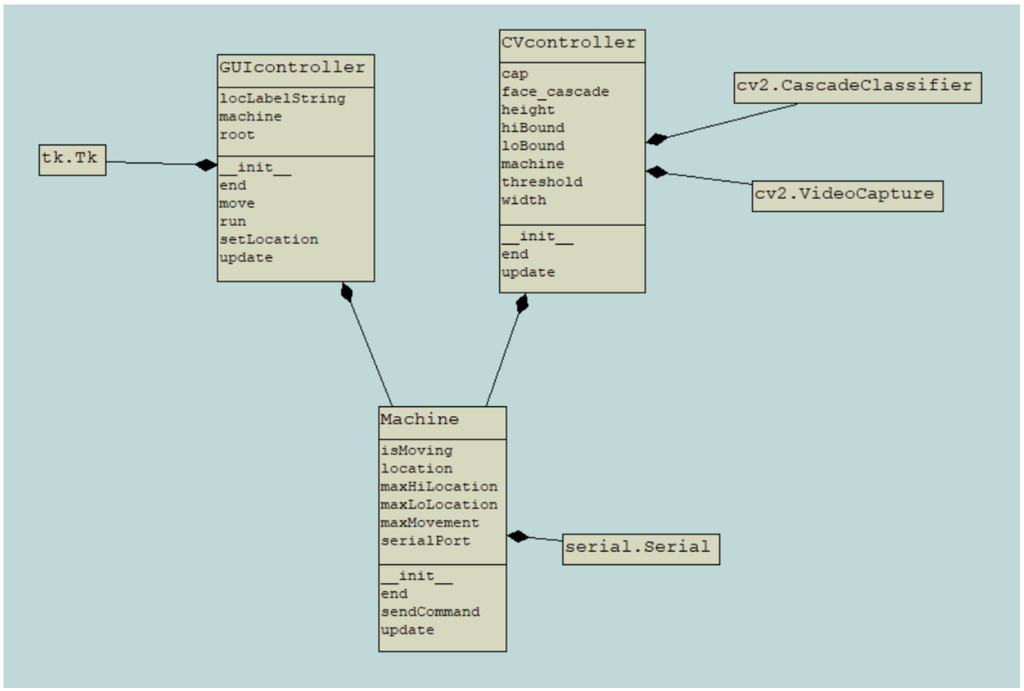
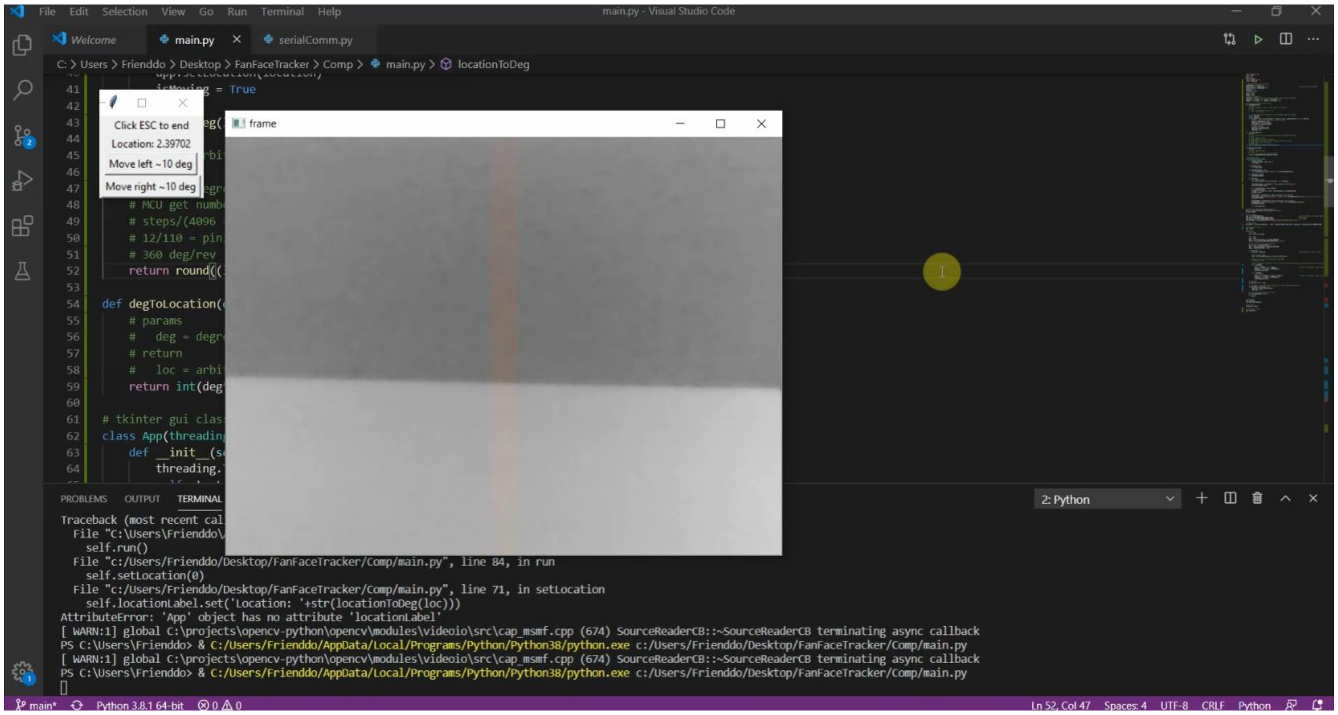


Figure 4: Main code and UML diagram of software module

One GUI window is from GUI class and has the buttons, while the other window shows what is processed by CV class.



```
            isMoving = True

        # MCU get numb
        # steps/(4096 
        # 12/110 = pin
        # 360 deg/rev
        return round((

    def degToLocation(
        # params
        #   deg = degr
        # return
        #   loc = arbi
        return int(deg

    # tkinter gui clas
    class App(threadin
        def __init__(s
            threading.
```

```
Traceback (most recent cal
  File "C:\Users\Frienddo\
    self.run()
  File "c:/Users/Frienddo/Desktop/FanFaceTracker/Comp/main.py", line 84, in run
    self.setLocation(0)
  File "c:/Users/Frienddo/Desktop/FanFaceTracker/Comp/main.py", line 71, in setLocation
    self.locationLabel.set('Location: '+str(locationToDeg(loc)))
AttributeError: 'App' object has no attribute 'locationLabel'
[ WARN:1] global C:\projects\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (674) SourceReaderCB::~SourceReaderCB terminating async callback
PS C:\Users\Frienddo> & C:/Users/Frienddo/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Frienddo/Desktop/FanFaceTracker/Comp/main.py
[ WARN:1] global C:\projects\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (674) SourceReaderCB::~SourceReaderCB terminating async callback
PS C:\Users\Frienddo> & C:/Users/Frienddo/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Frienddo/Desktop/FanFaceTracker/Comp/main.py
```

Figure 5: Sample GUI window

6.2 Inputs and Outputs

Input is the live image feed from a camera connected to the laptop, as well as buttons that manually instruct some fixed amounts of angular change to turntable. Output is direction and the number of steps for motors, sent to MSP board by UART. See section 5.3 for format of output byte and its optimizations.

6.3 Parameters

The main parameter of this module is the output byte telling how many steps the motor should step.

A way to get that is via position of the face in an image, another parameter in the module. Due to many CV library's abstraction and the high complexity behind the abstraction, as well as issue of tolerances of the mechanical assembly, it may be difficult to optimize the accuracy of the position. We get around this by specifying a threshold parameter. For example, if threshold is 0.10, then as long as face is within 10% of center, we are happy. This way, we can ignore the accuracy, and still get the face to roughly center on frame. This threshold is shown in GUI, where the colored area on frame is the ok range, and black-and-white area is the move-motor-more range (it is 10% in Figure 5).

Another parameter used to get that output byte is user interface software. There are buttons to input a fixed angular displacement in degrees, and that value is used to generate output byte. To get degrees from step, we need a conversion factor. We know that rack-to-pinion ratio is 110/12, per number of teeth, and in one revolution, there are 4096 steps and 360 degrees. Therefore, multiplying degrees by (110/12)*(4096/360) will get us steps. Note that we need to divide by 10 is we want it to be in the MCU's format.

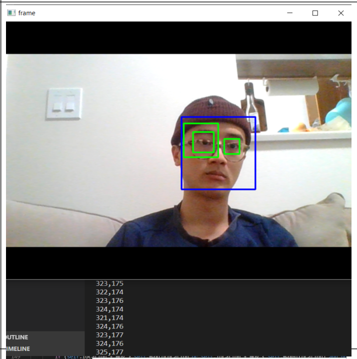
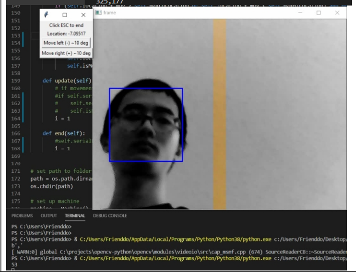
Whether a step command should be sent or not depends on whether stepping those steps would let pinion leave the rack (i.e. out of travel space on rack); if pinion would be outside of rack after stepping, then don't send the command. To determine that, we need to keep track of the current location of the pinion; I picked using 1/10th of step displacement as unit of location, so it's consistent with format of byte sent to MCU. We can also display this parameter (after it is converted to degress) on GUI for user's convenience.

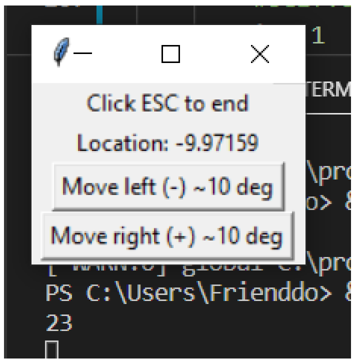
To check if where on rack is out of bounds, we need to know max location. This can be calculated. Our rack has 60 teeth, which means maximum travel is 30 teeth each way, resulting in 2.5 pinion revolutions to travel maximally in a direction (30/12=2.5). Given that a revolution is 4096 steps, we get 10240 steps (4096*2.5=10240) as max travel, or 1024 in location unit (10240/10=1024). Given that maximum movement that can be send in a byte is 127 (half of 255, largest number in a byte), we can say that max location is about 8 times max movement. This is experimentally confirmed to be true. In our software, however, we'll specify it as 7 times, to leave some margin to ensure pinion never falls off rack.

6.4 Development Plan

1. Write code to get face coordinates from a live image feed.
2. Get formatted byte of motor steps from coordinates of face.
3. Get formatted byte of motor steps from button press.
4. Start UART connection.
5. Send formatted byte to UART.
6. Refactor as needed.

6.5 Test Plan

Step in development plan	Test plan	Success if	Test observation	Test result
Live facial recognition	Create test software to get live camera feed and face coordinates	Print coordinates of face in image in real time	 323,175 322,174 323,176 324,174 321,174 324,176 323,177 324,176 325,177	Pass: Correct coordinates printed
Get byte from recognition	Put face in camera's view and print formatted step	Printed steps is proportional to distance from threshold line	 Click ESC to end Location: -7.09517 Move left (-) ~10 deg Move right (+) ~10 deg PS C:\Users\Frienddo> PS C:\Users\Frienddo> PS C:\Users\Frienddo> & C:/Users/Frienddo/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Frienddo/Desktop/ b',' [WARN:0] global C:\projects\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (674) SourceReaderCB::~SourceReader PS C:\Users\Frienddo> & C:/Users/Frienddo/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Frienddo/Desktop/ 53	Pass: Printed reasonable steps, also confirmed when whole assembly is tested later

Get byte from button	Compare algorithm’s output steps and direction to expected steps and direction as formatted and printed	Print expected steps and direction	 Click ESC to end
			Location: -9.97159
			Move left (-) ~10 deg
			Move right (+) ~10 deg
			`PS C:\Users\Frienddo>`
			`23`
				Pass:
				(127-23)*10steps*(360deg/4096steps)*(12/110) = 9.972deg = ~10deg
				Limitation exists due to rounding because steps has to be integer

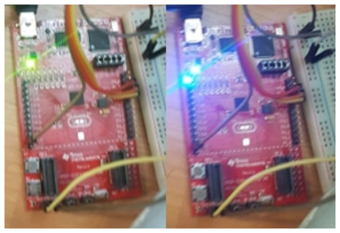
Start UART connection	Toggle LED on message received	Python software runs without freezing while LED is toggled		Pass: LED toggled as expected
Send byte to UART	Send byte to UART and observe turn angle	Turntable moved by table according to steps given		Pass: Turned as expected
Refactoring	Code review	Minimized software issues (ex: are functionalities encapsulated?)	Functionalities are encapsulated into CVcontroller, GUIcontroller, and Machine classes	Pass: Code is reasonably clean and commented

Table 5: Test plans and results for software module

7. System Evaluation

Test case	Test plan	Success if	Test observation	Test result
Full turn to left without falling off rack	Walk as far left as the turntable will follow	Location display on GUI is a little more than -889 (-7*127), and pinion is still on rack		Pass: Turned and displayed without falling off
Full turn to right without falling off rack	Walk as far right as the turntable will follow	Location display on GUI is a little less than 889 (7*127), and pinion is still on rack		Pass: Turned and displayed without falling off
Right button rotates to left by 10 degrees	Click on button	Pinion rotates by about 1/4 revolution ((110*10/360)/12), and display shows about 10 degrees moved	Pinion rotated by about 1/4 revolution, and display showed about 10 degrees moved	Pass: Turned and displayed as expected
Left button rotates to left by 10 degrees	Click on button	Pinion rotates by about -1/4 revolution, and display shows about -10 degrees moved	Pinion rotated by about -1/4 revolution, and display showed about -10 degrees moved	Pass: Turned and displayed as expected

Table 6: Test plans and results for full system

10. Reflections

10.1 What worked and what didn’t work? Why? What would you do differently if you could do it again?

Limiting scope is what worked best in this project. My project has only three custom parts (all easily 3D-print-able) and one degree of freedom (only one motor needed), with a controls software that can be Frankensteined from tutorials and forum posts. Limiting scope made the project feel manageable while juggling all other classes’ work. Because of this decision working, nothing in this project stood out as not working.

That said, if I had more time and I could do it differently, I might make the project scope bigger. Maybe add another degree of freedom to system (camera tilt angle controlled by another stepper?). Maybe use machine learning to identify custom, odd-shaped objects.

10.2 Identify 3 things you learned in MECH 423 that you consider the most useful and why.

1. Reaffirm my belief that I shouldn’t do an electrical-related job. Had a wire snap, bad jumper, and a board short in this class. I didn’t have time to exercise 4 & 5 from lab 3. In previous classes, I’ve had ICs short, breadboard not working, etc. I don’t think I fated to do this.
2. Relearned useful mechatronic concepts in lectures, like Mosfets or DC motors. Ties together many previous classes neatly. It’d be nice if you can upload all the lectures to Youtube and send us the playlist link as a term-end gift.
3. Losing the battle to win the war. A friend of mine told me this but I didn’t understand it, until I spent about twice the time compared to my friends on exercise 4 of lab 3, just to get it to not work for seemingly random reasons (motor wire snap is completely random and shorting might be because my window was opened and something blew into the circuit(?)). A lot of other classes’ work were piling up in backlog. I had to let it go so I could finish everything else. In the end, even though I didn’t have a submission, I think I learned what I was supposed to learn.

10.3 What are some limits of your knowledge and expertise as a mechatronic engineer?

As previously mentioned, my weakest area is electrical. From co-ops, I would say that software is my strongest suite; I’d say that I can write cleaner code in more languages faster than other mechatronics students, while writing better hardware-related code than computer science students.

Identify 3 things you would like to learn going forward. What is your strategy to acquire knowledge in these areas?

While my electrical skill is weakest, I don’t think it’ll help me to learn more about it if I don’t plan on working in that field.

Personally, I find it hard to come up with ideas of what to learn directly, but I often get a project idea, and through that, I find out what skills I need to execute on that idea. The idea for this project came from watching videos of people making useless robots. Instead of going from what-to-learn to strategy, I go from strategy (project idea) to what-to-learn, so I can’t identify 3 things I want to learn, but I can list 3 projects I want to do:

1. Wealth visualizer of Thai royal family, similar to https://mkorostoff.github.io/1-pixel-wealth/.
2. Integrating 3D printing with a novel 3D modelling software. May be marketable. Doesn’t feel particularly hard for me to do, but it requires deeper knowledge of two very different fields (mechanical and software), which is why I think no one has done this.

3. A novel visualizer that helps mechanical design. I think it’ll be very useful, and I’ve not seen anyone do it even though it’s not hard to do. Again, likely because not a lot of mechatronics people are around.

Honestly, the most exciting part of this Christmas break is I’ll have some free time from class to work on some ideas I’ve had for a while. Sorry for not being able to give a straight answer.