MECH423 Project Proposal

Ratthamnoon Prakitpong (#63205165)

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 the fan platform and its motor connection interface, motor controls software, and computer vision software. I will already have a stepper motor and a laptop, so the only thing I need to build is the fan platform and its interfaces. The final product will a combination of these three components.

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. List of Functional Requirements

Functional Requirements	% Effort
FR#1: Fan platform with motor connection (mechanical)	35
FR#2: UART Controlled stepper motor (electrical)	20
FR#3: Tracking software using computer vision (software)	45

In a project that starts from zero, FR#2 will take up more effort. However, since I can reuse motor set-up from Lab 3, the effort spent on that FR will be less.

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.

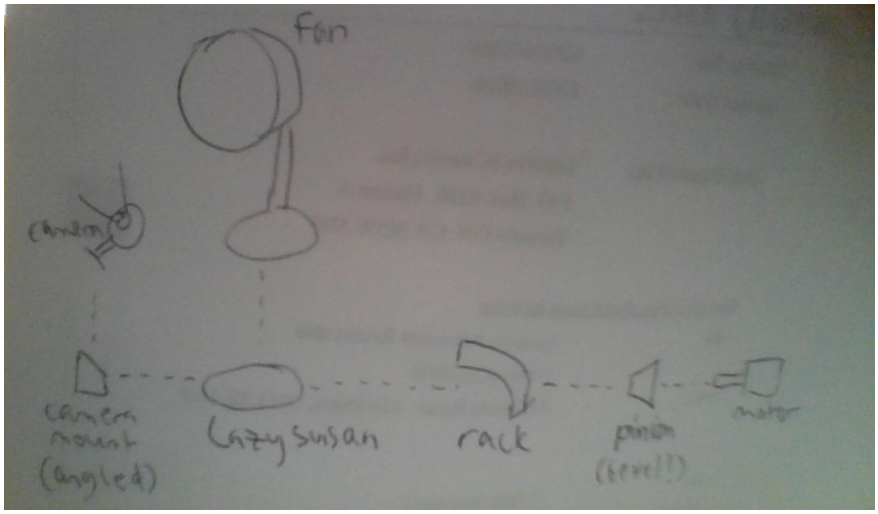

Fig 1: Mechanical components

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

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 fan and angular position of motor shaft are both measured in degrees. Since they are directed connected with rack and pinion, they're linearly proportional and their relationships can be found experimentally.

4.4 Development Plan

1. Acquire parts that are off-the-shelf.
2. Make measurements. Key measurements are:
    1. Turntable radius (use to design rack).
    2. Motor shaft cross section (use to design pinion).
    3. Camera mounting point dimension (use to design camera mount).
3. Design and 3D print these parts.
4. 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
Camera mount	Use it to get sample image	Software can determine position of face when I stand in front of camera
Rack	Use pinion to manually turn rack attached to turntable	Fan and turntable rotate without slipping or breaking
Pinion	Attach pinion to motor and use it to rotate turntable	Motor attached to pinion turns fan and turntable without slipping or breaking
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

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.

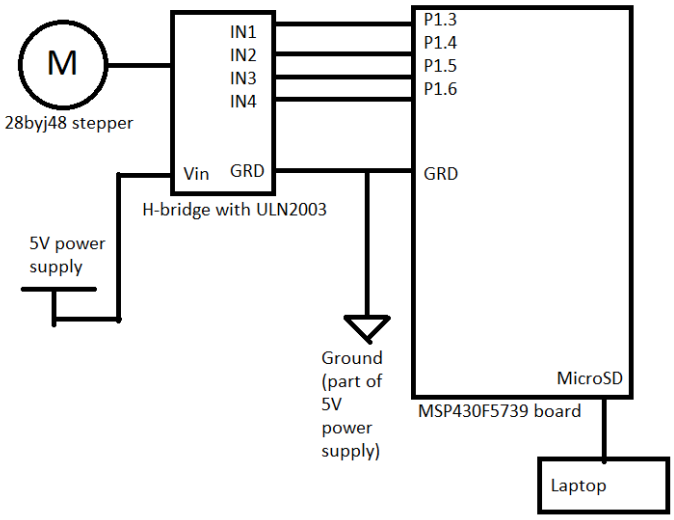

Fig 2: Electrical circuit diagram

The code used to run this stepper motor will be similar to the code in Lab 3's exercise 2 as well.

5.2 Inputs and Outputs

The module outputs motor rotation according to the received commands. As input, this module receives data package from UART communication, commanding the speed and direction of the motor. Format of the package is the same as Lab 2. Direction can be set with command byte.

Start byte	Command byte	Data byte #1	Data byte #2	Escape byte
0xFF	0x01			

Fig 3: UART package format

5.3 Parameters

The stepper motor is controlled by signals from the board. How quickly the signals are sent sets the speed of the motor; maximum speed is set by the physical limits of the motor itself. In a different project setup, we can test different motors and optimize speed of motor vs other parameters. How quickly the speed and direction can be changed is dependent on UART baud rate, as well as the speed at which the message is parsed. This operation's speed can be optimized by changing baud rate and writing better code.

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 speed and direction of motor.
    4. Change speed and direction 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
Send signals to h-bridge	Turning the motor on	Motor shaft rotates at some hardcoded speed and direction
Receive UART messages	Pause and see parameter in debug	UART messages are stored in circular buffer as they are sent
Parsing messages	Pause and see parameter in debug	Speed and direction are parsed and stored in some variables correctly
Change speed and direction	Observe motor speed and direction	Speed and direction are successfully changed from hardcoded values to said variables' values

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. The pseudo-code can be like shown:

```
CreateStartStopToggle()                  # set started
while(1):
  if started:
    image = getImage()
    if face is in image:
      x, y = getFacePosition(image)
      if x is centered:
        sendUART(0)                      # set speed to 0, stop moving
      elif x is left:
        sendUART(some speed, turn left) # I will experiment later whether CW or CCW motor shaft turn
                                         # will turn fan left or right
      else:                               # x has to be left
        sendUART(some speed, turn right)
```

6.2 Inputs and Outputs

Input is the live image feed from a camera connected to the laptop, as well as a button that starts and stops the program. Output is the speed and direction of the motors, sent to MSP board by UART. Format of package is discussed in section 5.2.

6.3 Parameters

Some parameters of this module include position of the face in an image. Due to many CV library's abstraction and the high complexity behind the abstraction, it may be difficult to optimize the accuracy of the position. Another parameter is the data to be sent to UART. To optimize the speed at which the data can be processed and sent, we can write better code. For example, using a simple check in a while loop as shown in the pseudo-code can be slow since the software has to perform this check at every iteration of the loop. A better implementation may use events to start and stop a image processing thread.

6.4 Development Plan

1. Write code to get face coordinates from a live image feed.
2. Get appropriate speed and direction of motor from coordinates of face.
3. Start UART connection.
4. Send speed and direction to UART.

6.5 Test Plan

Step in development plan	Test plan	Success if
Live facial recognition	Create test software to get live camera feed and face coordinates	Print coordinates of face in image in real time
Convert coordinates to speed and direction	Compare algorithm's output speed and direction to expected speed and direction	Conversion returns expected speed and direction
Start UART connection	Run board software, pause, and see changes in debug	A dummy message sent should be received and stored in circular buffer
Send motor parameters to UART	Run board software, pause, and see changes in debug	Correct values as sent in a formatted package is received and stored in circular buffer

7. Most Critical Module

The skills for FR#1 and FR#2 are taught in classes (CAD, prototyping, motors, firmware), whereas FR#3 is not, the module required to satisfy FR#3 becomes the most critical module. I will be following online tutorials on some CV tools available online (starting with OpenCV) and observe its accuracy, precision, input, output, and other parameters to verify how feasible they are for use in this project.

So far, I've put together a test script for getting coordinates of a face, as well as highlighting it in the video stream, from a camera input. The script is also attached as appendix.

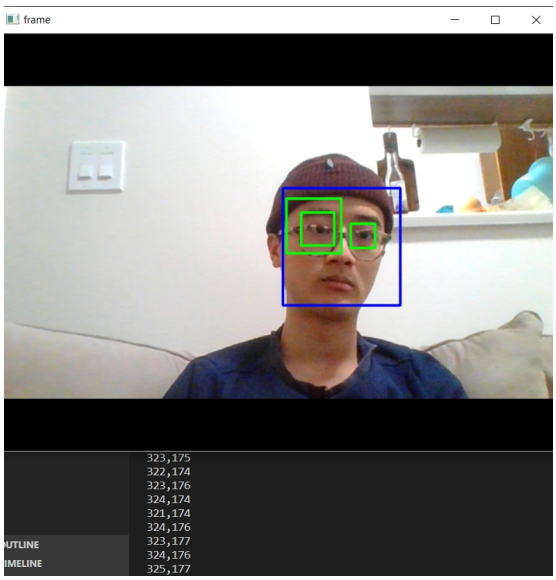

Fig 4: Test script for image recognition

This test script is able to accurately track the position of a face from a video input. This quick preliminary experiment proves that it is feasible to use CV to satisfy FR#3, our most critical module.

8. Risks and Countermeasures

For each module of the project (broken down as functional requirement), there are potential conditions associated with them that could undermine the project's success, as well as actions I can take to lessen potential impacts. Some sample risks and countermeasures are listed in the table below:

Associated FR	Potential Issue	Countermeasure(s)
1	Parts not arriving on time (bearings, bolts, screws, etc.).	• Go to a physical store to buy parts (ex: Lee's Electronics). • Find/design a 3D-print-able alternative.
1	3D printer breaking down.	• Borrow a friend's 3D printer. • Prototype with other materials like cardboard.
2	Electrical parts breaking down.	• Use my own Arduino board and 28byj48 stepper motors.
2	Stepper motor can't output enough torque.	• Design gear train to increase torque from motor to fan platform. • Use alternative stepper motors (I have NEMA17 from MECH45X).
3	OpenCV is too complex to use, or doesn't work.	• Did preliminary experiments to observe feasibility of CV module.
3	Python's serial communication library not compatible with MSP430 board.	• Go back to creating application with WinForm and use IronPython to run OpenCV from the C# app. • Find alternative serial communication library.

9. Learning Objectives

Software tools like computer vision will be increasing popular and accessible, so gaining this skill now is important for future mechatronics projects (even though it is not taught in the current curriculum). I hope to learn how to use computer vision modules available online, and learn how to adapt it for use in mechatronics context. Other parts of the project such as prototyping or motor controls can help me revise and apply what I've learned from classes.

Appendix

OpenCV test code:

```
import numpy as np
import cv2

# https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html#face-detection

cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Height
while(True):
    ret, frame = cap.read()

    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

    img = frame
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        print(str(x) + ',' + str(y))
        img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)




    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)

    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        break
cap.release()
cv2.destroyAllWindows()
```