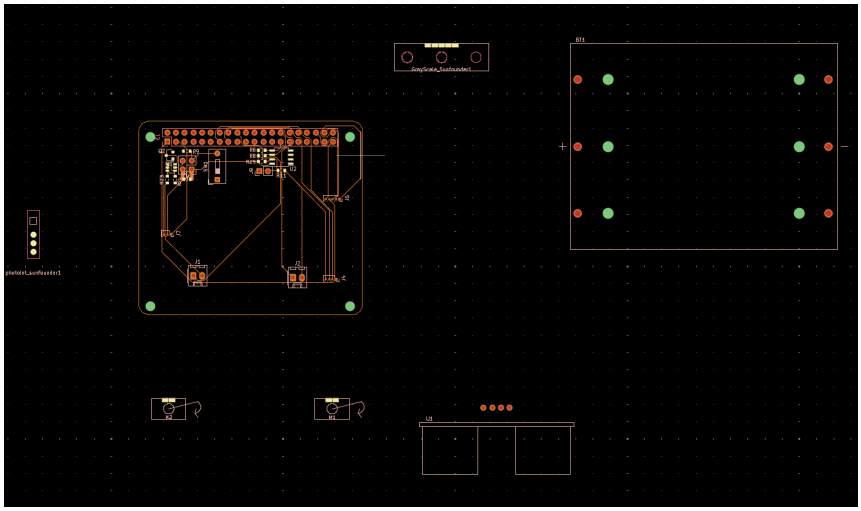
CS437: Internet of Things
Lab1 Part1

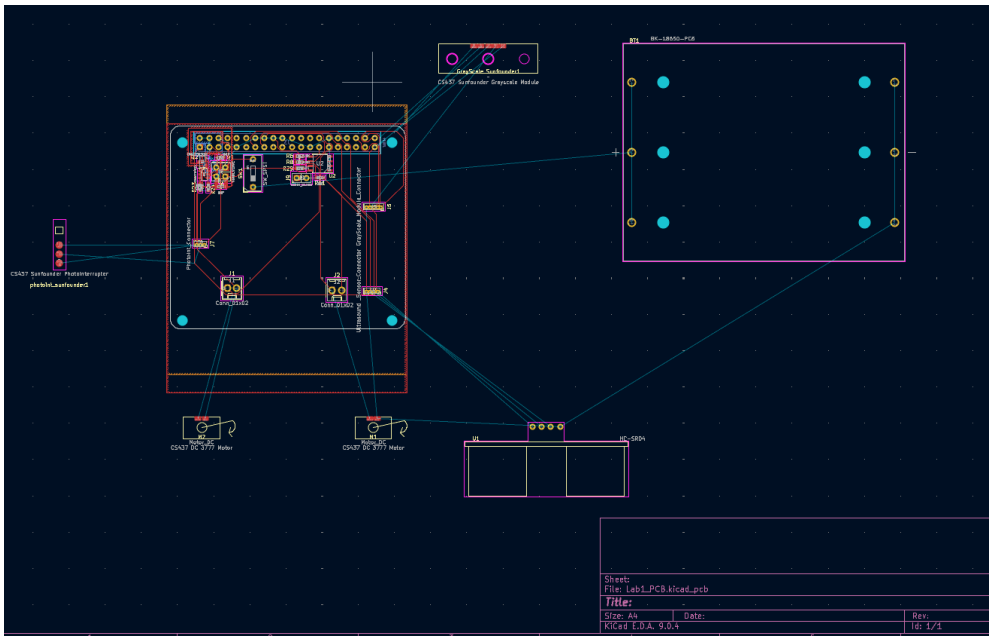
Name: Muhammad Amir, Saad Ahmad, Eraad Ahmd
NetIDs (include NetID of all group members): mamir6,saada4,eahme2
Late days used:1

Video Link: https://youtube.com/shorts/HngZWCo6HTM

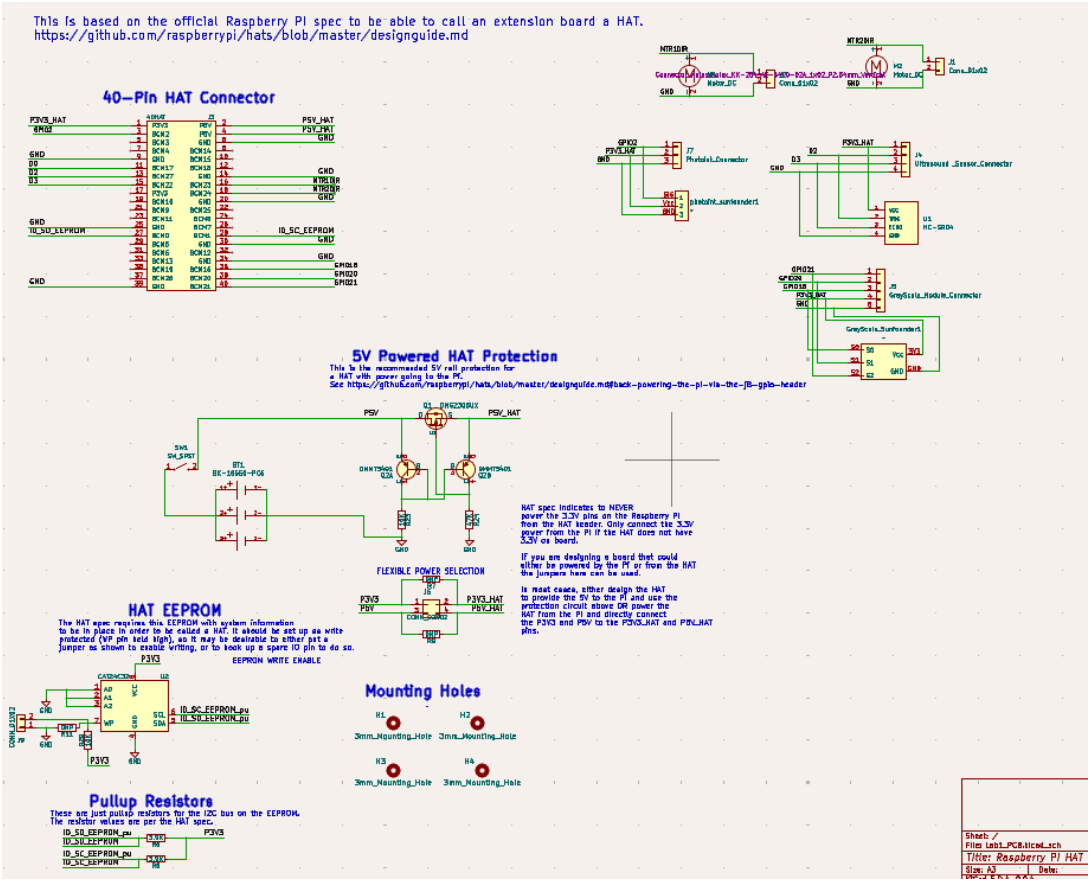
Kicad related files submitted on Coursera in folder
Pictures of the Kicad Results:
Gerber:



PCB:



Schematic:



Design Consideration:

When assembling and configuring the PiCar-X, we encountered an issue with the steering servo not responding correctly. To resolve this, we changed the steering servo control from PWM pin 2 to PWM pin 3. This adjustment fixed the steering issue and ensured that the car could properly navigate turns during testing.

We also considered sensor placement for mapping. The grayscale sensor was positioned to provide reliable line/contrast detection for maze tracking, while the camera was mounted forward-facing to capture visual data for future extensions such as lane detection or object recognition. The ultrasonic sensor was mounted on a servo to allow dynamic distance measurements directly in front of the car.

These design choices allowed the car to move reliably, detect obstacles, and execute basic avoidance behaviors while leaving room for more advanced navigation features in later parts of the lab.

Chassis Assembly:

We assembled the PiCar-X kit according to the provided guide. Key steps included mounting the motors, attaching the Raspberry Pi, and installing the ultrasonic sensor on the servo mount. We ensured the camera was facing forward to capture a clear field of view for potential computer vision tasks.

- Calibration: Verified motor directions using keyboard_control.py and adjusted configuration so "forward" motion was consistent.

- Power: Used a rechargeable battery pack to supply sufficient current to both the Pi and motors.

- Testing: Ran servo sweeps to confirm full rotation and tested ultrasonic readings for reliable distance detection

Naive Mapping:

For environment awareness, we implemented a simple grayscale mapping algorithm together with the onboard camera.

- **Grayscale sensor:** Used to detect line contrast and provide basic feedback on track or maze boundaries.

- **Camera:** Mounted forward-facing, providing visual input for potential extensions such as object recognition or lane detection. For Part 1, it was mainly validated to ensure proper mounting and image capture.

- **Algorithm:**

    1. Drive forward until an obstacle is detected within a set threshold (≈20 cm) using the ultrasonic sensor.

    2. Stop and back up briefly.

    3. Choose a new direction (random left/right turn).

    4. Resume forward driving.

- **Testing:** Placed cardboard boxes and other household items in a simple test course to validate detection reliability. The grayscale sensor was verified by running along black/white surface sections, while the camera was tested by displaying objects in front of the car and confirming they appeared correctly in the feed.

Naive Self-driving:

We extended the mapping logic into a basic self-driving routine that mimics a “Roomba-style” avoidance system.

- Behavior: The car avoids collisions by dynamically rerouting whenever an obstacle is detected. It performs stop, backup, random turn, and forward motion.

- Performance: In a small maze-style setup, the car was able to navigate around multiple objects without collisions in most trials.

- Limitations: Occasionally struggled with shiny or angled surfaces that reflected the ultrasonic signal incorrectly. Navigation speed had to be kept moderate (20–40% duty cycle) to ensure accurate detection and stopping distance.

Name	Contribution
mamir6	Built the PiCar-X and helped with calibration; assisted with the maze track setup.
saada4	Built the PiCar-X and helped with calibration; wrote the maze navigation code.
eahme2	Completed the PCB assignment and performed the car assembly; helped debug the code.

```
from picamera2 import Picamera2
from picarx import Picarx
import time
import random

# Camera settings
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration())
picam2.start()
time.sleep(2)

# Car and movement setup
px = Picarx(servo_pins=['P0','P1','P3'])
SPEED = 15
SafeDistance = 30   # > 40 safe
DangerDistance = 20 # > 20 && < 40 turn around,
BACKUP_TIME = 1.25
BACKUP_SPEED = 20
TURN_TIME = 1.1
TURN_ANGLE = 45
STRAIGHT_ANGLE = 0

def turn_left(speed, duration):
   px.set_dir_servo_angle(-TURN_ANGLE)
   px.forward(speed)
```

```
    time.sleep(duration)
    px.stop()
    px.set_dir_servo_angle(STRAIGHT_ANGLE)

def turn_right(speed, duration):
    px.set_dir_servo_angle(TURN_ANGLE)
    px.forward(speed)
    time.sleep(duration)
    px.stop()
    px.set_dir_servo_angle(STRAIGHT_ANGLE)

turn_directions = [turn_left, turn_right]

def random_turn(px, speed, duration):
    turn_func = random.choice(turn_directions)
    turn_func(speed, duration)

try:
    while True:
        dist = px.ultrasonic.read()
        print(f"Distance: {dist:.2f} cm")

        # Capture image each cycle (optional)
        picam2.capture_file("latest.jpg")
        if dist >= SafeDistance:
            px.set_dir_servo_angle(0)
            px.forward(SPEED)
        elif dist >= DangerDistance:
            random_turn(px, SPEED//2, TURN_TIME)
            px.forward(SPEED)
            time.sleep(0.1)
        else:
            random_turn(px, SPEED//2, TURN_TIME)
            px.backward(SPEED)
            time.sleep(0.5)
        time.sleep(0.06)

except KeyboardInterrupt:
    px.stop()
```

```
    picam2.stop()
    print("Stopped")
```