Team 14: Self-Parking Car

UC San Diego

JACOBS SCHOOL OF ENGINEERING

ECE/MAE 148, SP23
Team #14

The Team

Manoel Aguirre-Lara (MAE): Mechanical Engineering

Rohan Sreedhar (ECE): Computer Engineering

Allison Moya (ECE): Electrical Engineering

Shravan Suresh (MAE): Mechanical Engineering* (dropped the class)

Recap: Proposal

- The car will park itself in a free parking spot with lane detection and LIDAR
- Materials:
  - Multiple cameras
  - Other parts of the current car (Jetson, Battery, VESC)
- Different from previous projects (2019 Fall Team 2):
  - Using cameras and lidar, not other sensors (Adafruit TOF sensor)
- Stretch goal: parallel parking

What Was Promised

Must Have

- Car will park itself in a free parking spot, using image recognition
- Uses cameras

Nice to Have

- Parallel Parking
- Lidar integration

Demo! Proof of Concept

UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/MAE 148, SP23
Team #14

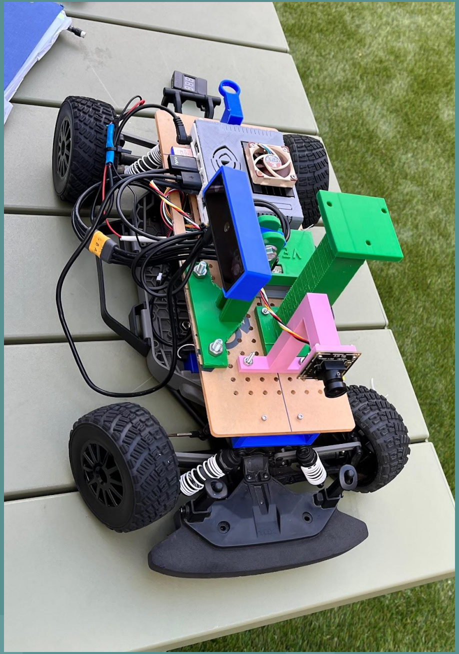
What Was Delivered

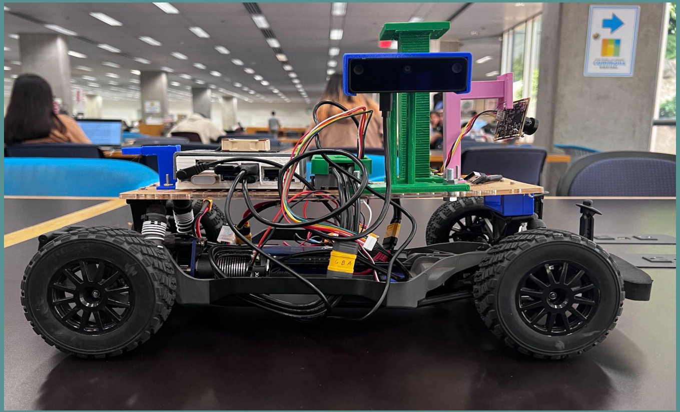
- Car moves forward, through the parking lot
- If it sees a handicap sign, it decides to park in that parking space
- It turns, throttles, then straightens out, throttles again, and stops

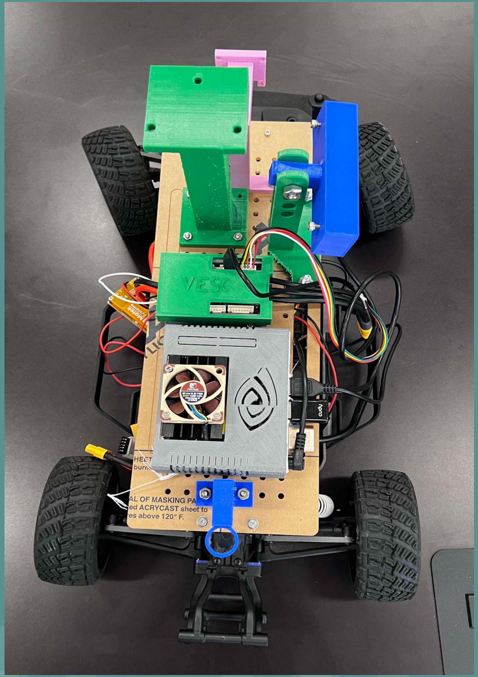
- Accomplished using Hough Lines, Template Matching, and PyVesc

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Hardware



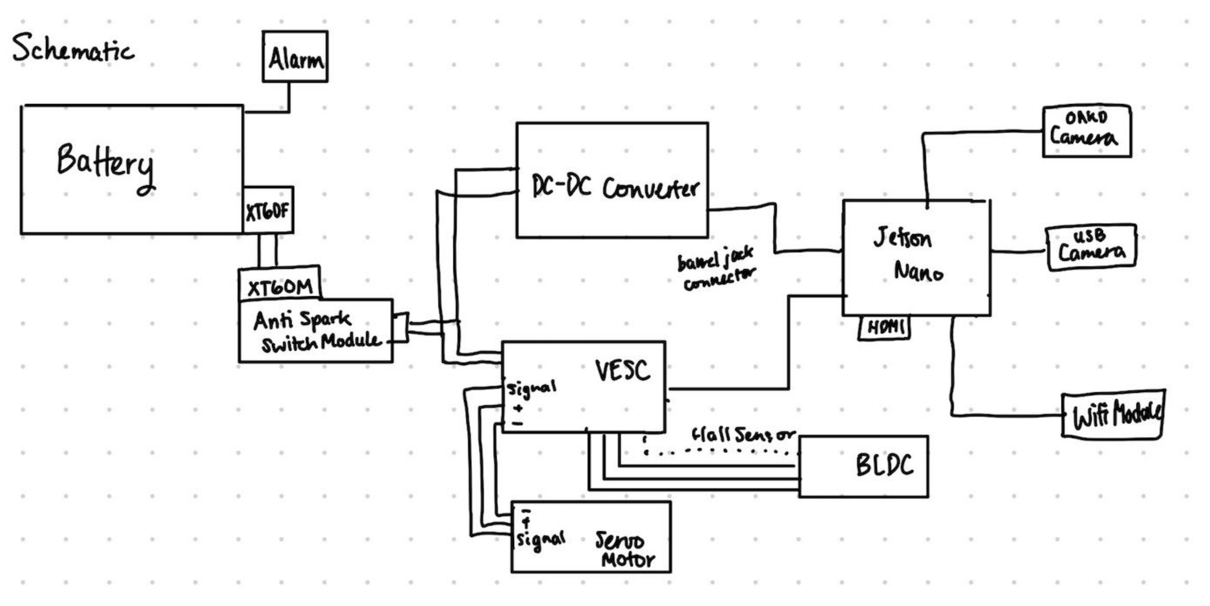




Hardware: Components

- OAK-D Camera: facing sideways
- USB Camera - unused
- Lidar + Lidar mount - unused
- Standard components: VESC, Jetson

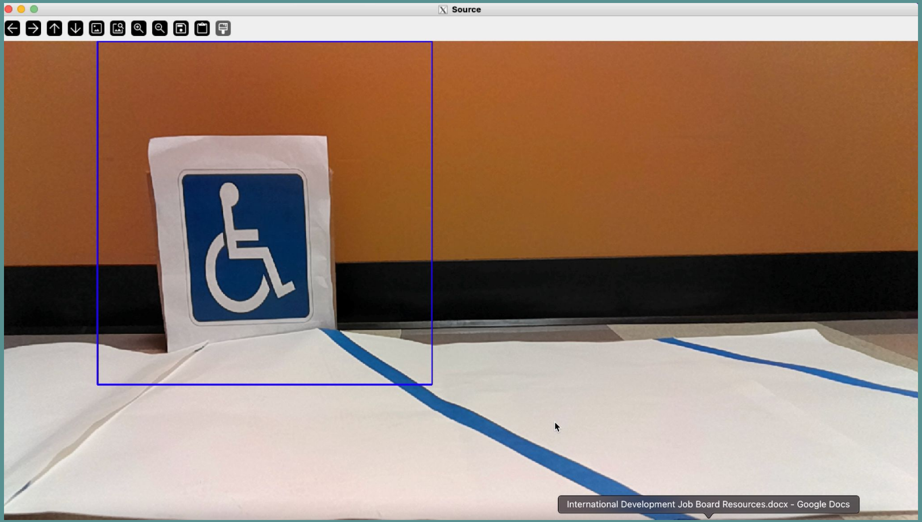
Wiring Schematic



Software

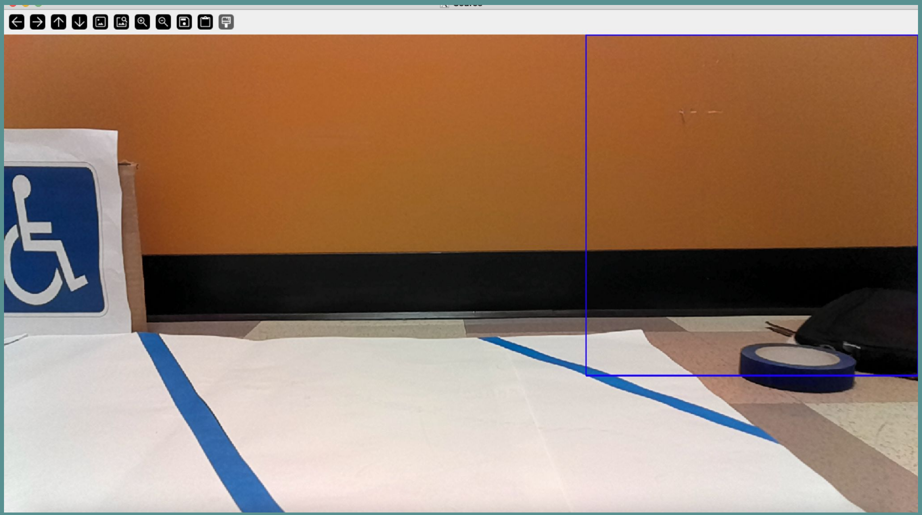
- Two OpenCV functions were used:
  - Hough Lines
  - Template Matching
- VESC was controlled using pyvesc python library

Template Matching


Source
International Development Job Board Resources.docx - Google Docs

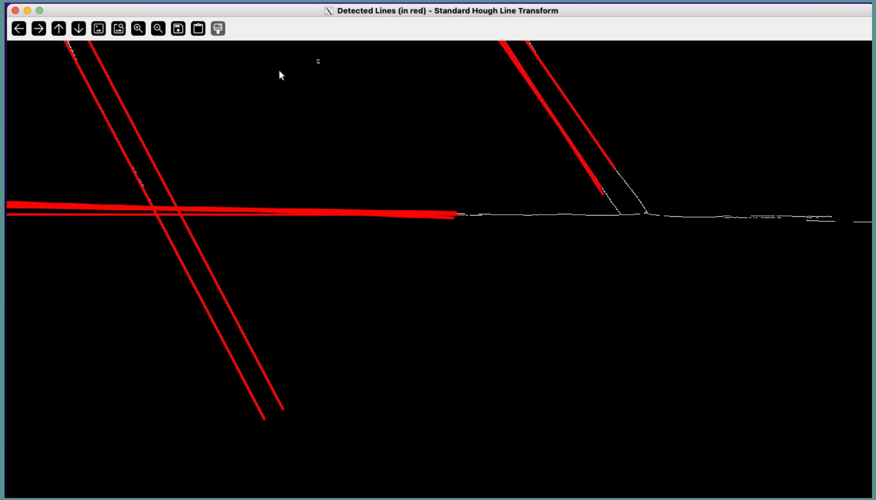
- The concept:
  - Template Matching finds a given template in another image
- Template: that blue Handicapped Parking sign

Template Matching



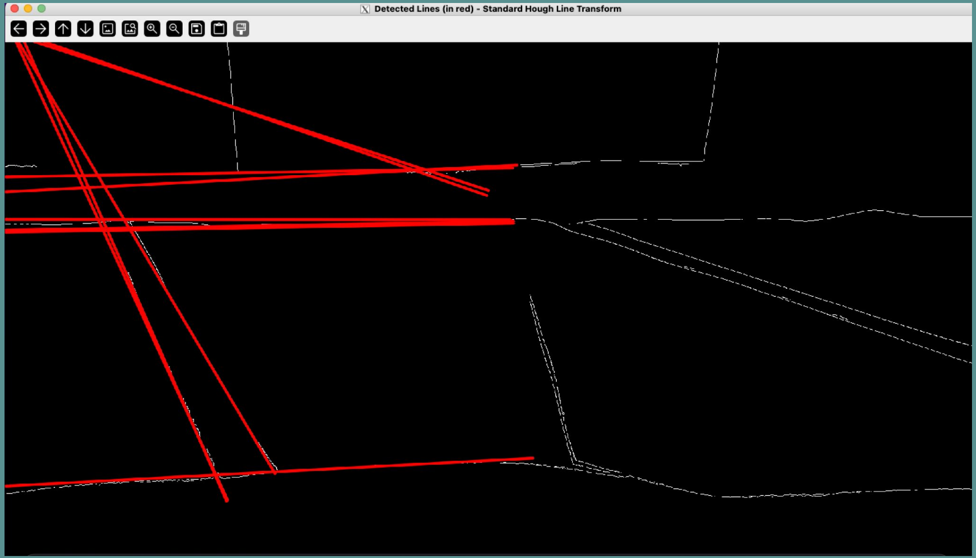
- Very unreliable!
- Many false positives
- Had to set threshold very high - 0.9999/1 - to recognize the correct part of the image

Hough Lines and Lane Detection



- The concept:
  - Based on the number of discrete points, found using polar coordinates (r, θ), detect lines in the given image
- Goal: detect parking lines (like lane detection)
  - Detect angle of lines, for turn angle

Hough Lines and Lane Detection



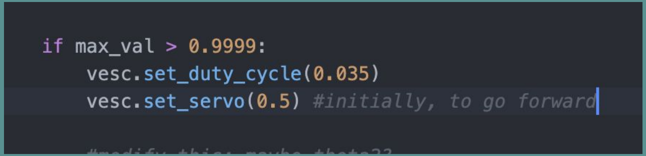
- Not perfect
  - Restricted angle between 1.75 and 2.8 radians
- Efficiency: houghLinesP (probabilistic)
  - Didn’t work!

PyVesc

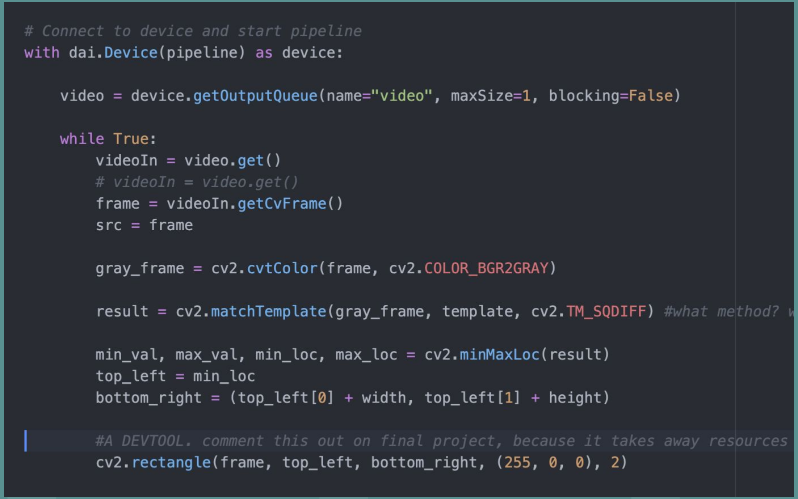
- Straightforward
  - set_servo(angle)
    - 0-1, 0.5-1 is to the right
  - set_duty_cycle(speed)
    - 0-1, 0.035 was selected
- Too easy to throttle too much

Software

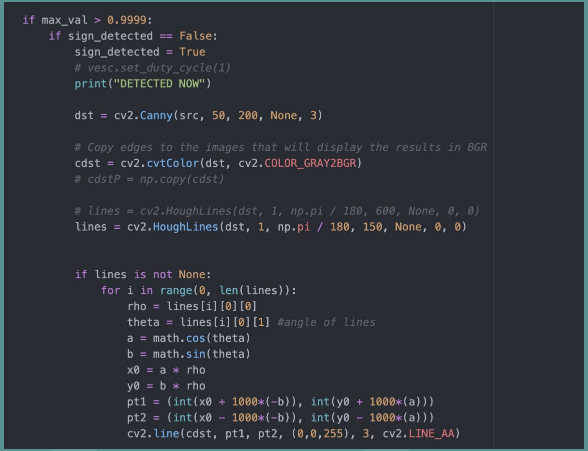
- HoughLines and Template Matching



```
if max_val > 0.9999:
    vesc.set_duty_cycle(0.035)
    vesc.set_servo(0.5) #initially, to go forward

    #modify this: maybe theta??
```



```
# Connect to device and start pipeline
with dai.Device(pipeline) as device:

    video = device.getOutputQueue(name="video", maxSize=1, blocking=False)

    while True:
        videoIn = video.get()
        # videoIn = video.get()
        frame = videoIn.getCvFrame()
        src = frame

        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        result = cv2.matchTemplate(gray_frame, template, cv2.TM_SQDIFF) #what method? w

        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
        top_left = min_loc
        bottom_right = (top_left[0] + width, top_left[1] + height)

        #A DEVTOOL. comment this out on final project, because it takes away resources
        cv2.rectangle(frame, top_left, bottom_right, (255, 0, 0), 2)
```



```
if max_val > 0.9999:
    if sign_detected == False:
        sign_detected = True
        # vesc.set_duty_cycle(1)
        print("DETECTED NOW")

        dst = cv2.Canny(src, 50, 200, None, 3)

        # Copy edges to the images that will display the results in BGR
        cdst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)
        # cdstP = np.copy(cdst)

        # lines = cv2.HoughLines(dst, 1, np.pi / 180, 600, None, 0, 0)
        lines = cv2.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)


        if lines is not None:
            for i in range(0, len(lines)):
                rho = lines[i][0][0]
                theta = lines[i][0][1] #angle of lines
                a = math.cos(theta)
                b = math.sin(theta)
                x0 = a * rho
                y0 = b * rho
                pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
                pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
                cv2.line(cdst, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)
```

Challenges

- VESC stopped working, required replacement
- Switch stopped working, unable to use anti-spark module
- Hough lines and template matching not always succeeding in detection; unreliable
- Jetson randomly turning off

If we had another week…

- Use YOLO instead of template matching to detect cars/signs
- Stretch goals of parallel parking and lidar integration for calculating distance

Thank You! Questions?

UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/MAE 148, SP23
Team #14