HÁSKÓLINN Í REYKJAVÍK

COMPUTER VISION
T-869-COMP

Assignment 3

Sindri Þór Harðarson
sindrih18@ru.is

Instructor
Torfi Þórhallsson

November 29, 2023

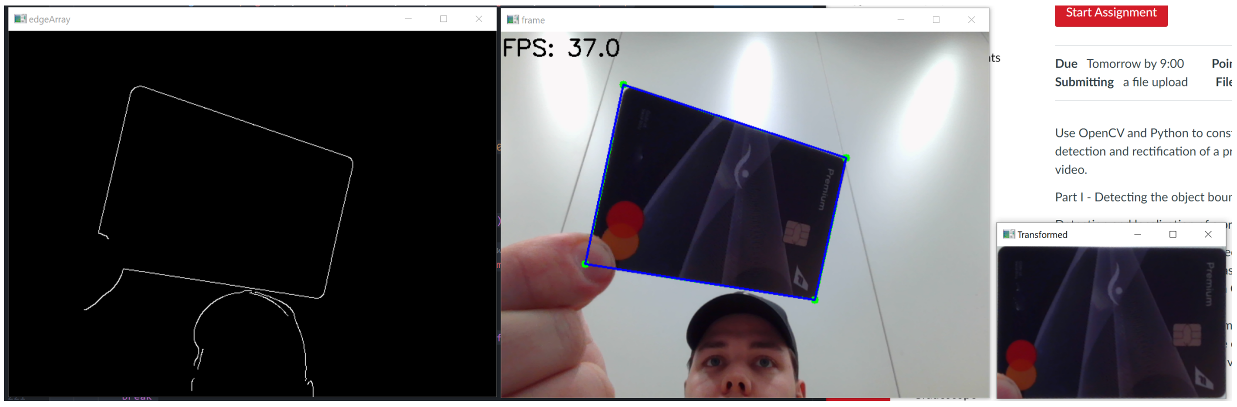

Figure 1: Perspective change

`https://github.com/SindriTh/ComputerVision`

1 Questions

1.1 How well your straight line detector follows the edge of a sheet of paper moved across the field of view of the camera.

It depends on the background. If a white background is used, the CANNY function loses it completely. However, if there is slightly more contrast between the background and foreground, it works quite well. The reason why the sheet gets lost if the background is white, is that the Canny function is working hard to only keep relevant edges (more contrasty) edges.

1.2 How well it detects other straight lines in your environment.

It has been tuned quite heavily to ignore those, but it does detect them. The HoughLinesP has rho set to 4, and theta to $3\pi/180$ or 3 degrees. Minimum line length is 90 pixels, and it needs to have 120 inliers, with a maximum gap of 45 pixels. This ensures that only "real" lines are captured.

1.3 The processing time for one video frame or image.

The processing time varied between 10ms to 30ms, depending on the amount of "noise" picked up by the Canny function.
If a scheme was really busy, I got up to 30ms of processing time, but down to 10ms when the scene was simpler.

2 Code

```
import itertools
import numpy as np
import cv2
import time

FRAME_WIDTH = 640
FRAME_HEIGHT = 480
CANNY_THRESHOLD1 = 250
CANNY_THRESHOLD2 = 390
RANSAC_ITERATIONS = 150
RANSAC_THRESHOLD = 10 # The distance from the line that a point can be and still be consi
RANSAC_INLIER_THRESHOLD = 120 # The number of inliers required to stop the RANSAC algorit

def line_slope(line):
    """Calculate the slope of a line."""
    x1, y1, x2, y2 = line[0]
    return (y2 - y1) / (x2 - x1) if (x2 - x1) != 0 else float('inf')

def line_intersection(line1, line2):
    """Find the intersection point between two lines."""
    x1, y1, x2, y2 = line1[0]
    x3, y3, x4, y4 = line2[0]

    denom = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)
    if denom == 0:
        return None  # Lines are parallel

    intersect_x = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4)) / c
    intersect_y = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 * x4)) / c
    return (intersect_x, intersect_y)

def is_point_within_viewport(point):
    """Check if a point is within the viewport."""
    x, y = point
    return 0 <= x < FRAME_WIDTH and 0 <= y < FRAME_HEIGHT

def find_intersections(lines):
    """Find all intersections between lines within the viewport and calculate angles."""
    intersection_points = []
    for line1, line2 in itertools.combinations(lines, 2):
        intersection_point = line_intersection(line1, line2)
        if intersection_point and is_point_within_viewport(intersection_point):
            angle = angle_between_lines(line1, line2)
            if 65 <= angle <= 115:  # Slightly broader range to capture near-90 degree ir
                intersection_points.append(intersection_point)
    # Filter out points that are too close to each other
    filtered_points = filter_close_points(intersection_points)

    return filtered_points

def angle_between_lines(line1, line2):
    """Calculate the angle between two lines in degrees."""
    slope1 = line_slope(line1)
    slope2 = line_slope(line2)

    tan_angle = abs((slope2 - slope1) / (1 + slope1 * slope2))
    angle = np.arctan(tan_angle) * (180 / np.pi)  # Convert radian to degree

    return angle

def filter_close_points(points, min_distance=20):
    """Filter out points that are too close to each other."""
    filtered_points = []
    for point in points:
        if all(np.linalg.norm(np.array(point) - np.array(other_point)) >= min_distance
               for other_point in filtered_points):
            filtered_points.append(point)
    return filtered_points


def order_points(pts):
    """Sort the points based on their x-coordinates and y-coordinates."""
    # Check if there are exactly four points
    if len(pts) != 4:
        raise ValueError("There must be exactly four points to order")

    # Sort the points based on their x-coordinates.
    xSorted = pts[np.argsort(pts[:, 0]), :]

    # Get the two left-most points and the two right-most points.
    leftMost = xSorted[:2, :]
    rightMost = xSorted[2:, :]

    # Now, sort the left-most according to their y-coordinates to grab the top-left and b
    leftMost = leftMost[np.argsort(leftMost[:, 1]), :]
    (tl, bl) = leftMost

    # Get the top-right and bottom-right points by finding the farthest and nearest point
    # to the top-left from the rightMost array
    distance = np.linalg.norm(rightMost - tl[np.newaxis], axis=1)
    (br, tr) = rightMost[np.argsort(distance)[::-1], :]

    # Return the points in order: top-left, top-right, bottom-right, bottom-left
```

```python
    return np.array([tl, tr, br, bl], dtype="float32")


def extract_polygon(frame, intersection_points):
    """Extract the outer polygon based on the intersection points."""
    # Convert intersection points to a NumPy array
    points = np.array(intersection_points, dtype="float32")

    # Filter out points that are too close to each other
    filtered_points = filter_close_points(points)

    if len(filtered_points) >= 4:
        hull = cv2.convexHull(np.array(filtered_points))
        hull_points = hull[:, 0, :]
        # Sort the hull points
        if len(hull_points) == 4:
            ordered_hull_points = order_points(hull_points)
            return ordered_hull_points
    return filtered_points

def perform_perspective_transform(frame, src_points):
    # Define a fixed size for the destination rectangle
    width, height = 300, 200  # For example, a standard size for a credit card

    if src_points is None or len(src_points) != 4:
        return None

    src_points = np.array(src_points, dtype="float32")  # Ensure dtype is float32

    # Define the destination points for the fixed size rectangle
    dst_points = np.array([
        [0, 0],
        [width - 1, 0],
        [width - 1, height - 1],
        [0, height - 1]
    ], dtype="float32")

    # Compute the perspective transform matrix
    M = cv2.getPerspectiveTransform(src_points, dst_points)

    # Perform the warp perspective
    transformed = cv2.warpPerspective(frame, M, (width, height))
    return transformed

def main():
    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_WIDTH)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT)
    cap.set(cv2.CAP_PROP_AUTO_WB, 0.0) # Disable automatic white balance
    cap.set(cv2.CAP_PROP_WB_TEMPERATURE, 4200) # Set manual white balance temperature to

    while True:
        start_frame_time = time.time()
        _, frame = cap.read()
        cleanframe = frame.copy()
        edges = cv2.Canny(frame, CANNY_THRESHOLD1, CANNY_THRESHOLD2)
        cv2.imshow('edgeArray', edges)


        # Hough Line Transform
        lines = cv2.HoughLinesP(edges, 4, 3 * np.pi / 180, 120, minLineLength=90, maxLine

        if lines is not None:
            for line in lines:
                x1, y1, x2, y2 = line[0]
                cv2.line(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            # Show intersection points
            intersection_points = find_intersections(lines)
            if intersection_points is not None:
                for point in intersection_points:
                    cv2.circle(frame, (int(point[0]), int(point[1])), 5, (0, 255, 0), -1)

            polygon = extract_polygon(frame, intersection_points)
            # Draw the polygon on the frame
            if polygon is not None and len(polygon) == 4:
                cv2.polylines(frame, [np.int32(polygon)], True, (255, 0, 0), 2)

                # Perform perspective transform and display it in another window
                transformed_frame = perform_perspective_transform(cleanframe, polygon)
                if transformed_frame is not None:
                    cv2.imshow('Transformed', transformed_frame)

        # print(f"Processing time: {time.time() - start_frame_time:.4f}")
        cv2.putText(frame, f"FPS: {1 / (time.time() - start_frame_time):.1f}", (1, 31), c
        print(f"Frame time: {(time.time() - start_frame_time)*1000:.4f}ms")
        cv2.imshow('frame', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```