1. detect_lines()
    a. Given the image, run it through canney edge detection and houghlinesP to get the edges from the images.
2. get_pairwise_intersections()
    a. Given the detected lines in the image, get the cross product of each pair of lines.
    b. Check if intersection point lines at infinity (z=0).
        - If yes, discard
        - If no, save point into the intersections array.
3. get_support_mtx()
    a. For each intersection point(i), calculate the shortest distance to each line(j).
        - Using formula $\text{distance}(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}.$
    b. For each distance calculated, check if within the max threshold value.
        - If yes, set support matrix[i][j] = 1.
        - Else, set support matrix[i][j] = 0.
4. get_vanishing_pts()
    a. Given the support matrix, find the intersection with the most supporting lines (row with most 1s).
    b. Set the columns of the support matrix where the chosen intersection has the value ==1 to 0.
    c. Repeat process for the given number of vanishing points needed.
5. get_vanishing_line()
    a. Given 2 vanishing points, the vanishing line is the cross products of these 2 vanishing points.
6. get_target_height()
    a. Calculate vanishing point u = (b_1 x b_2) x I and scale to form (x, y, 1).
    b. Calculate transferred point t'_1 = (t_1 x u) x (v x b_2) and scale to form (x, y, 1).
    c. Calculate distances of t'_1, t2, v from b2 using $\sqrt{\Delta x^2 + \Delta y^2}$
    d. Calculate distance ratio, $\frac{d_1}{d_2} = \frac{t_1'(v-t_2)}{t_2(v-t_1')}$
    e. Calculate target height by dividing query distance by ratio.