Design Project #1

Graphical Synthesis of a Planar Linkage

by

Daniel Chao, 112412719
Dalton Lin, 112831251
Chengji Wang, 112609874

Group #11

Department of Mechanical Engineering
Stony Brook University
Stony Brook, New York — 11794

11/08/2021

Abstract

The goal of this assignment is to design a four-bar linkage that can shift an object from an initial position to a final destination without the object or any links touching the ground or an obstacle. In order to avoid contact with the ground, two more links will be added as a fully rotatable drive dyad to make the four-bar linkage into a six-bar linkage. The final linkage should be carefully designed to avoid any branch or circuit defect so that it can move continuously from the initial to the final position while having a good transmission angle as close to 90° as possible. The link length should also be reasonable in terms of intentions to build the mechanism, not too big or small relative to the size of the scoop. To create the design, an intermediate position of the movement will be selected. In the early designing process, several iterations were found to be non-Grashof, have a low transmission angle, or contain defects. After careful consideration and improvement in the designs, linkages with reasonable transmission angles and link lengths were created. In the hope of creating the best possible design, we created a program to find a design that has the highest possible transmission angle.

Table of Contents

List of Figures/Tables

Figures

1. Problem Statement

In this project, the geometry of a scoop is given in two positions and a mechanism needs to be designed to move the scoop between these two positions. The motion of the scoop and any links in the mechanism cannot hit the ground or a given rectangular obstacle in the problem. Because two points can define the location and orientation of the scoop, a four-bar mechanism can be used to move it. To ensure that the scoop does not hit the ground or obstacle, an intermediate position of the four-bar mechanism is used in a safe location. Three-point synthesis is used to achieve this motion, creating a four-bar mechanism which is then checked to satisfy the Grashof condition. This linkage must then be constrained to move only between the initial and final position without moving past either. To do this, the four-bar must be constrained with a dyad, which will make a six-bar mechanism. Finally, the mechanism must be tested to find any defects which could alter the motion of the mechanism. Once a mechanism that satisfies all of these conditions is made, the transmission angle is calculated and taken into account along with the link lengths to determine how viable the mechanism would be in real life. A larger transmission angle optimizes the force transmission and allows for a smoother motion. Since the maximum transmission angle is 90°, the minimum transmission angle must be as close to that as possible, which means that the design cannot be a double rocker or triple rocker since the transmission angles of those mechanisms are zero. To simulate the mechanism, reasonable link lengths have to be considered as links that are too small or large would be difficult or unwieldy to implement and use. Technical skills that are required for the project include knowledge in machine design, geometric drawings, and optional coding languages. In the project, a graphical synthesis and Python program was used to generate a possible solution. A reasonable link length and transmission angle was found using the method of graphical synthesis. However, using the coding method, the most optimal solution for the design was generated.

2. Design Iterations

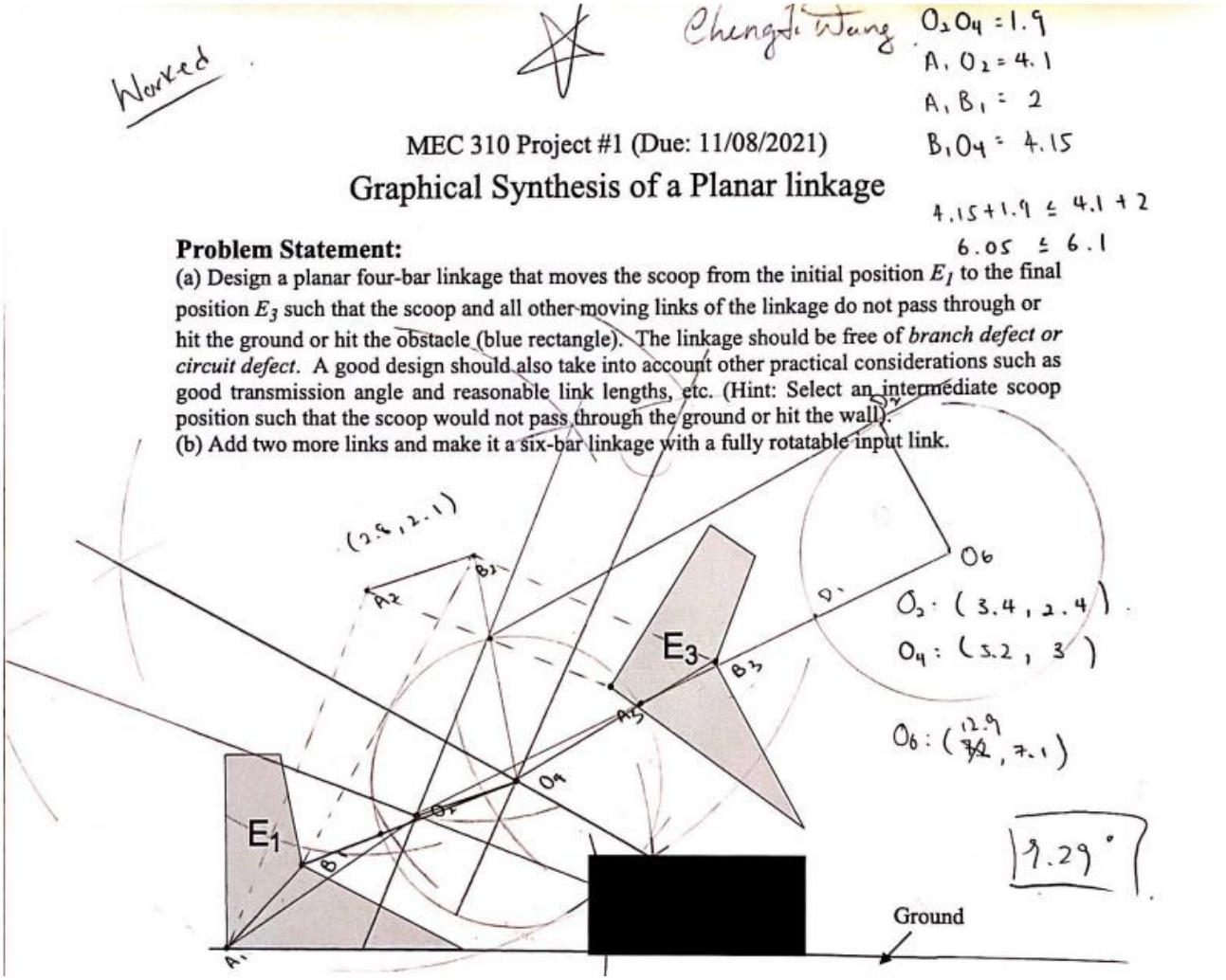

Figure 2.1: First Design Iteration

The first design iteration is derived from the method of graphical synthesis with reasonable link lengths. The sum of the shortest and longest link is less than the sum of other two links, which shows that it satisfies Grashof's law. Upon analyzing the links, the design is a double crank with a transmission angle of 9.29°. Using MotionGen to simulate the motion of the mechanism, it was found that the design will not hit the ground or the obstacle. The design was not selected due to the low value of the transmission angle.

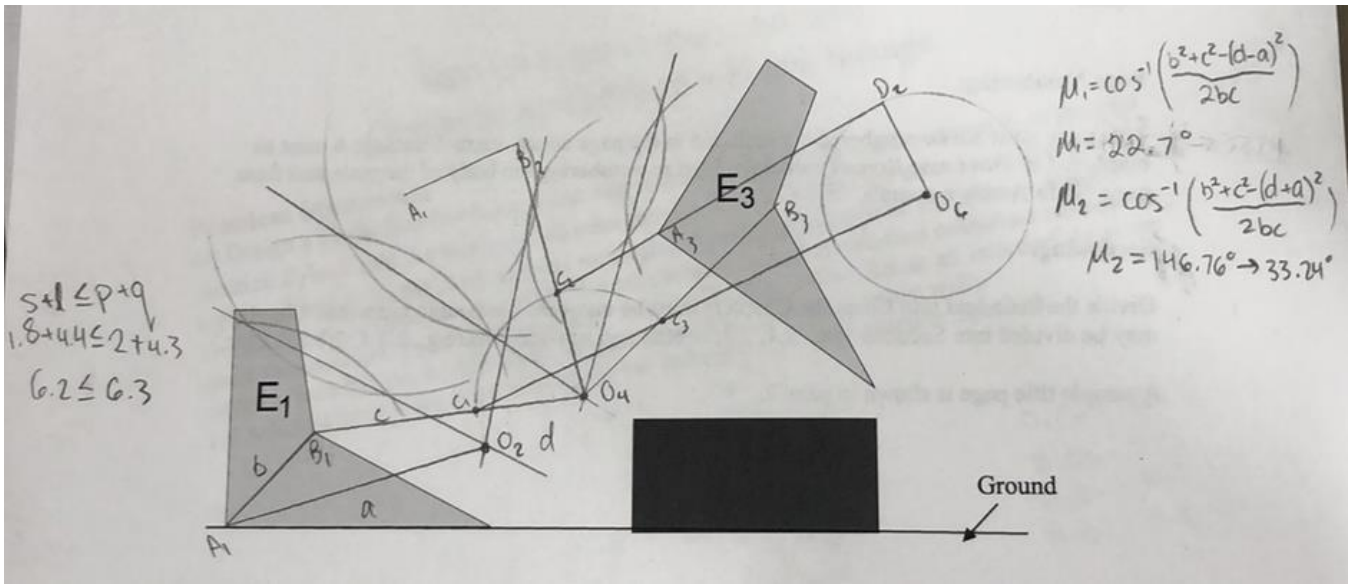

Figure 2.2: Second Design Iteration

The second design iteration was derived using the method of graphical synthesis, slightly modifying the design of the mechanism from the first iteration. The sum of the shortest and longest link is less than the sum of others shows that it satisfies Grashof's condition. After analyzing the links, the design is a double crank with a transmission angle of 22.7° and it contains no branch or circuit defect, which is better than the first iteration. Using MotionGen to simulate the motion, it was found that this design will not hit the ground or the obstacle. This is a viable design that has a reasonable transmission angle, but was not selected due to other better designs.

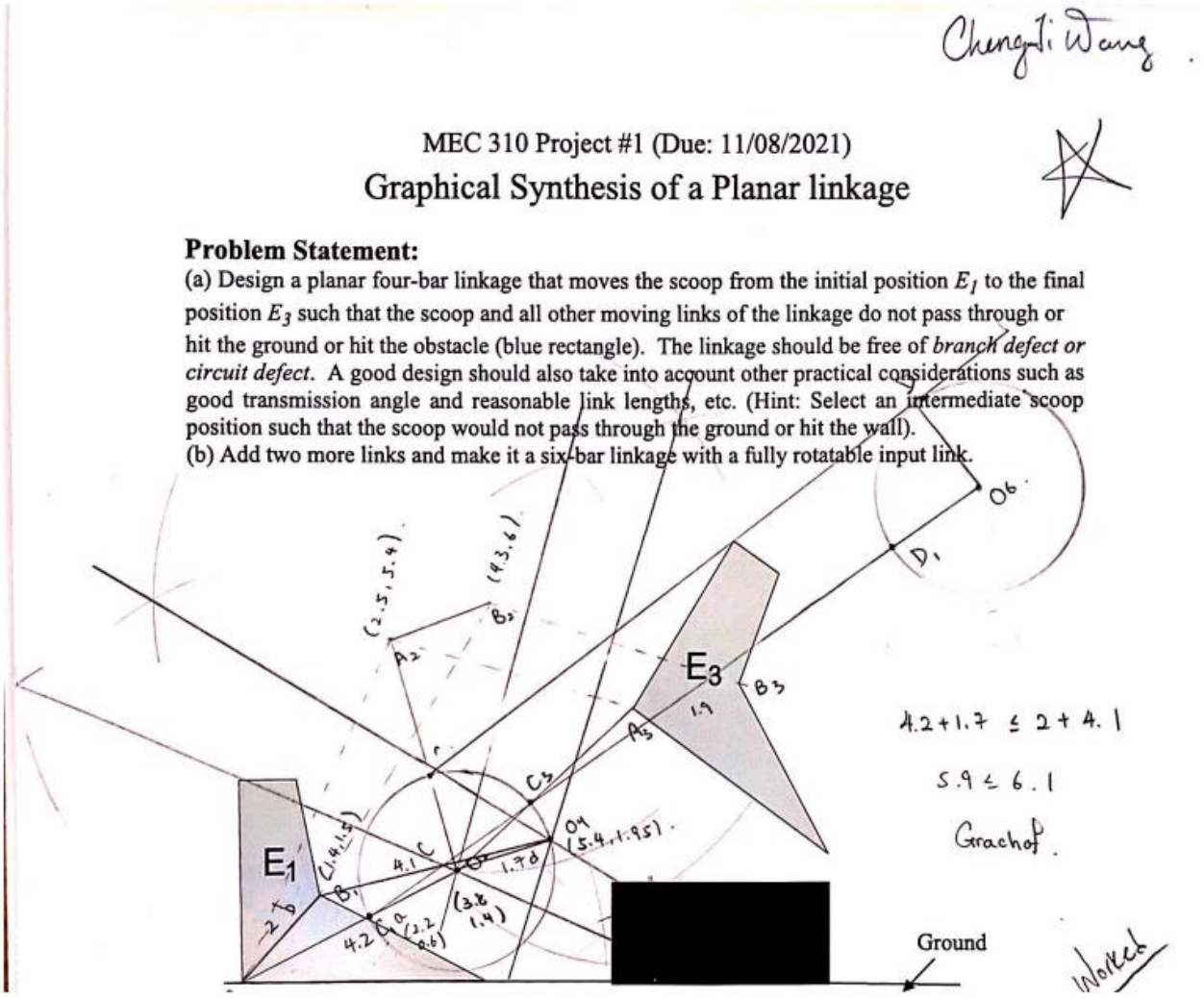

Figure 2.3: Third Design Iteration

The third design iteration is derived from the method of graphical synthesis, further improving upon the previous iterations using some patterns found in failed attempts. The sum of the shortest and longest link is less than the sum of others shows that it satisfies Grashof's law. Upon analyzing the links, the design is a double crank with a transmission angle of 27.40° and contains no branch or circuit defect. Using MotionGen to simulate the motion, it was found that the design will not hit the ground or the obstacle. The design has a reasonable transmission angle, but is not selected due to other better design.

Python Program

The code in the appendix tests various locations of the intermediate position and different coordinates of the moving pivots in the locally rotating coordinate system of the scoop. It rules out any possibilities that have the fixed pivots, links, or scoop intersect with either the ground or the rectangular obstacle and prints possible intermediate positions, coordinates of moving pivots in all three positions, corresponding coordinates of the fixed pivots, link lengths, and transmission angle. The interval at which the program tests can be varied to account for run time and number of trials. An example output is:

```
ANGLE2:  19
OFFSET2:  [4, 6]
A1:  [1.75, 0.25]
B1:  [2.0, 0.0]
A2:  [5.573265468684515, 6.806123914199853]
B2:  [5.891037151198634, 6.651136308914313]
A3:  [8.462726053229277, 3.8736300570819093]
B3:  [8.518033988749895, 3.524429495415054]
mu1:  82.34872373478004
mu2:  81.45148942561286
SIDE LENGTHS:  3.951956343280017 0.3535533905932738 3.933682242023945
0.049583587353350105
FP1:  [4.614969742430901, 2.9721144932104857]
FP2:  [4.630297901989583, 2.92495964553423]
81.45148942561286
```

where ANGLE2 is the angle of the LRCS of the intermediate position with respect to the GCS; OFFSET2 is the offset of the intermediate position compared to the initial position; A1, B1, A2, B2, A3, and B3 are the coordinates of the moving pivots in the GCS; mu1 and mu2 are the transmission angles at the extreme position; SIDE LENGTHS gives the lengths of the links, which can be used to check the Grashof condition; FP1 and FP2 are the coordinates of the fixed pivots corresponding to A and B, respectively; and the final number is the transmission angle of the four-bar mechanism. This output corresponds to the following four-bar mechanism:

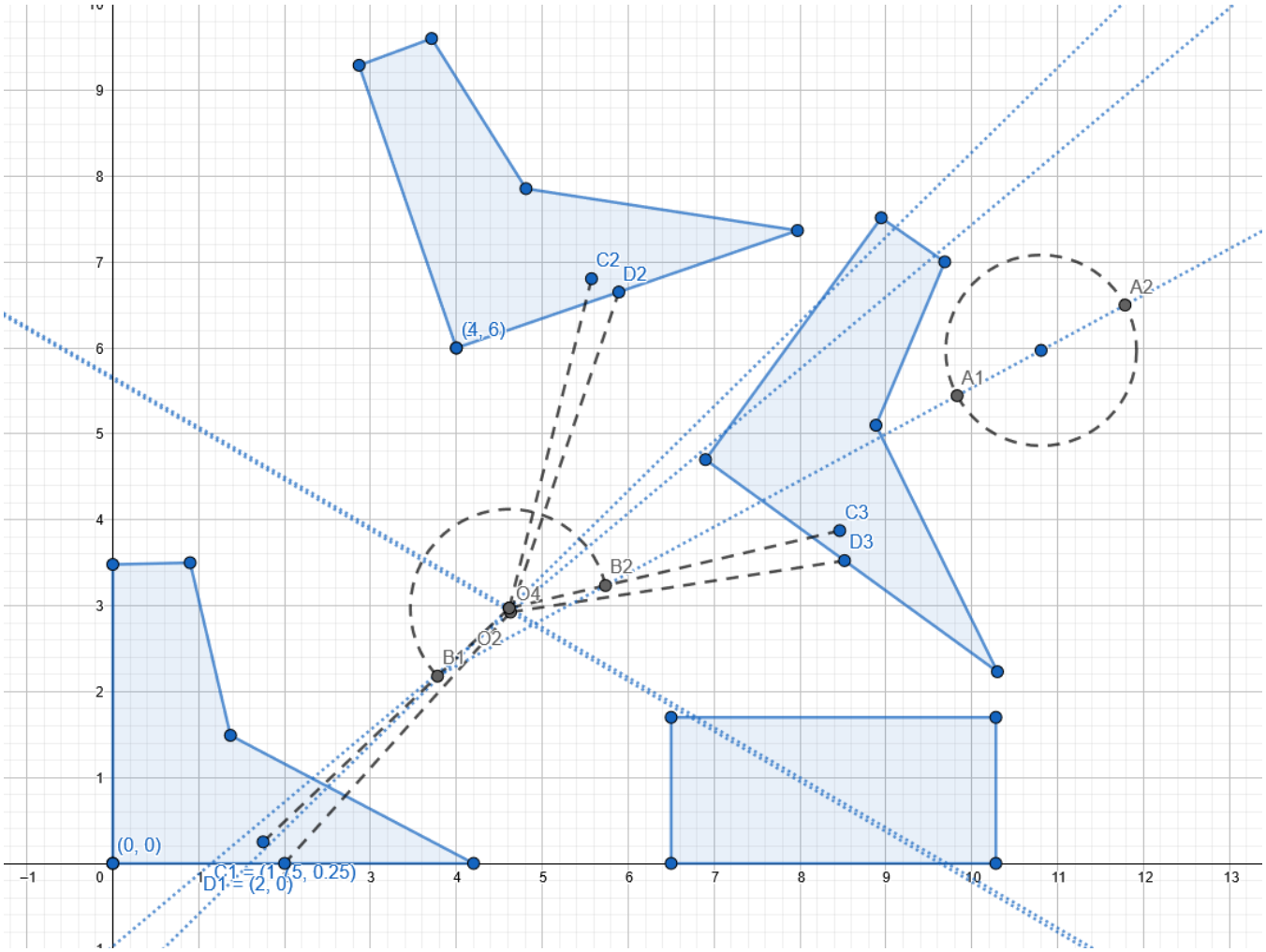

Figure 2.4: Fourth Design Iteration

The fourth design iteration comes from the Python program. The sum of the shortest and longest link is less than the sum of others, which is guaranteed by the program, showing that it satisfies Grashof's law. This is a double-crank mechanism with the maximum transmission angle of all the linkages found by the program, being 81°. This mechanism will not collide with the obstacle or the ground, but it has a circuit defect, which renders it unusable. While trying to optimize transmission angle, the resulting mechanism had a circuit defect which meant that it could not be used.

```
ANGLE2:  -11
OFFSET2:  [1, 3]
A1:  [0.0, 2.25]
B1:  [0.5, 2.75]
A2:  [1.4293202395972258, 5.2086611627572434]
B2:  [2.0155383290093303, 5.604070256792804]
A3:  [8.222516817658065, 6.520288237343632]
B3:  [8.920917940991774, 6.630904108384869]
mu1:  42.989278089028005
mu2:  43.81219921251797
SIDE LENGTHS:  5.778959933623614 0.7071067811865476 5.761853833333912
0.5122256428016766
FP1:  [5.703511879659491, 1.3192314181225757]
FP2:  [6.142477934785425, 1.5832085224152896]
42.98 9278089028005
```

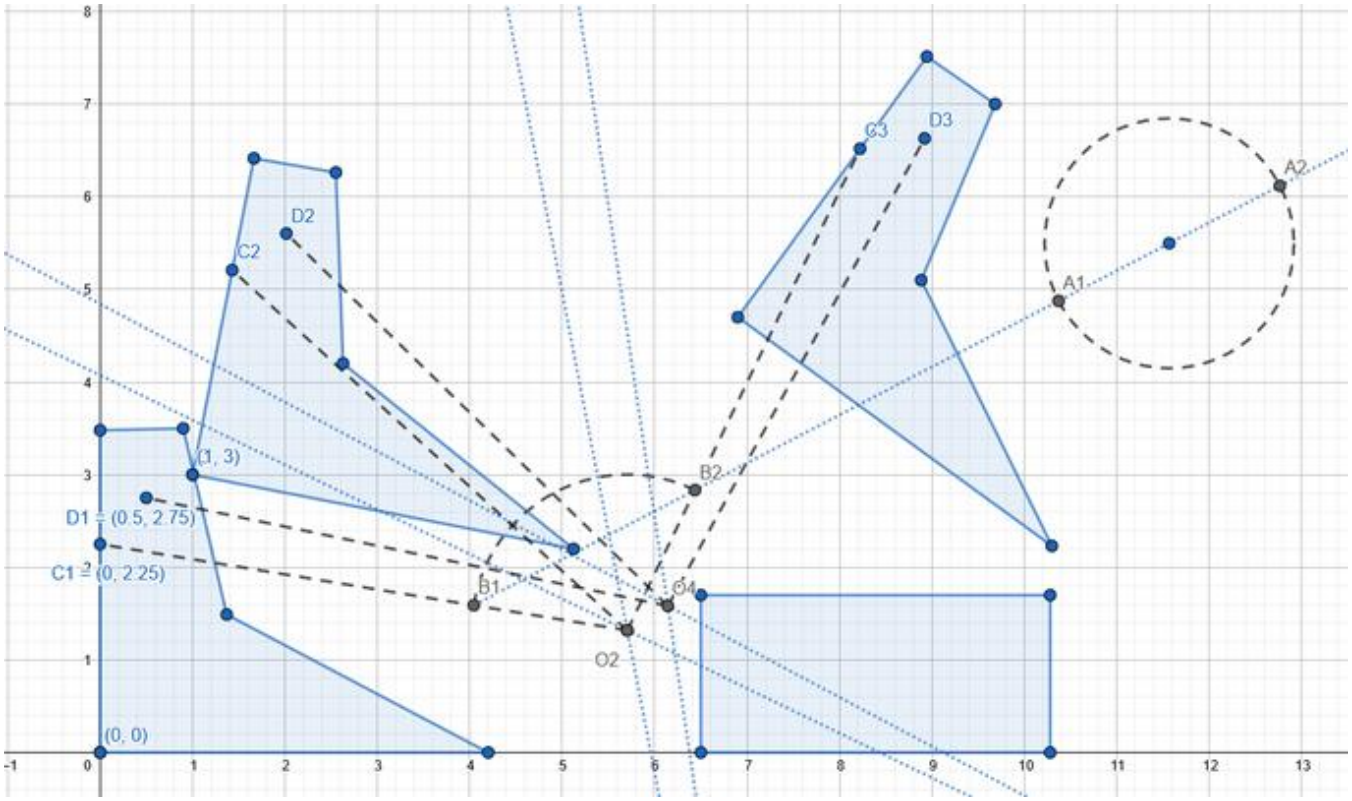

Figure 2.5: Final Design Iteration

The program output and corresponding image above are an example of a large transmission angle without any circuit or branch defect. Grahof's law is satisfied in this double-crank mechanism and the transmission angle is 42.99°. It contains no circuit or branch defect and all the link lengths are reasonable when compared to the scoop. This was the design selected to be simulated.

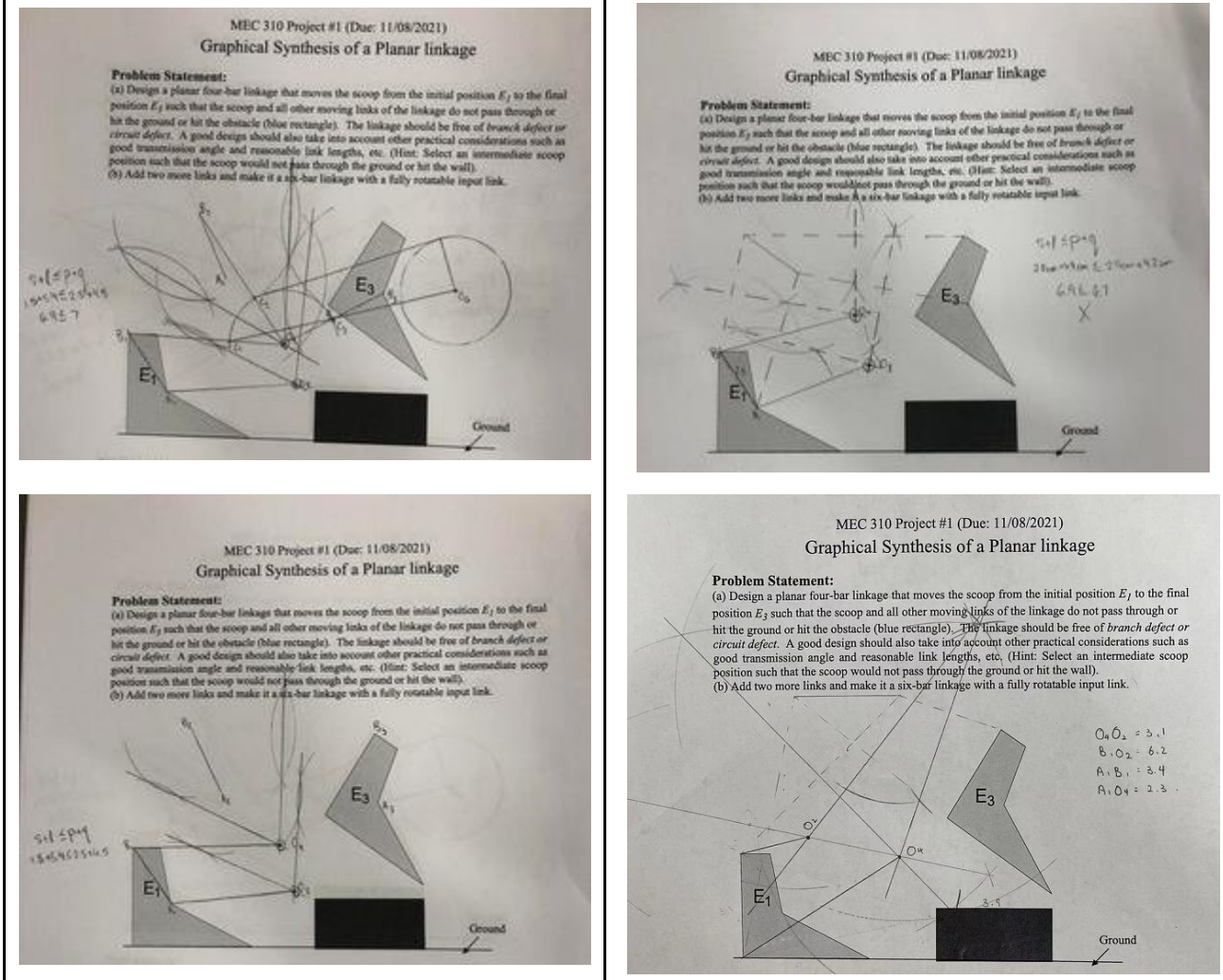

Figure 2.6: Failed Iterations Due to Not Satisfying Grashof's Law

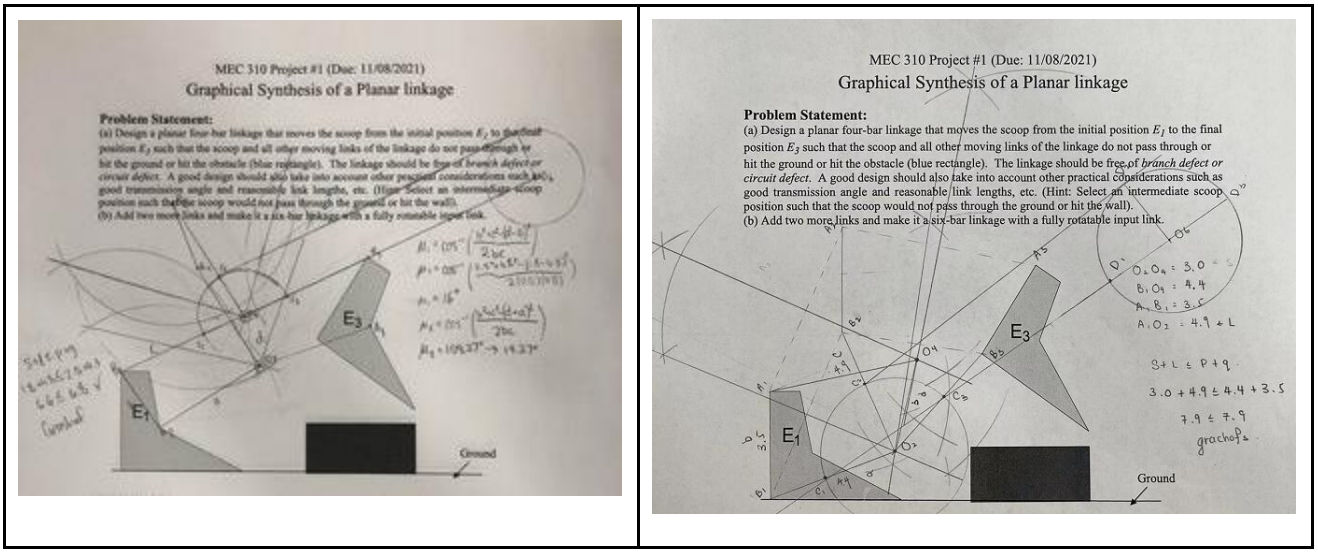

Figure 2.7: Failed Iterations Due to Transmission Angles Equal Zero

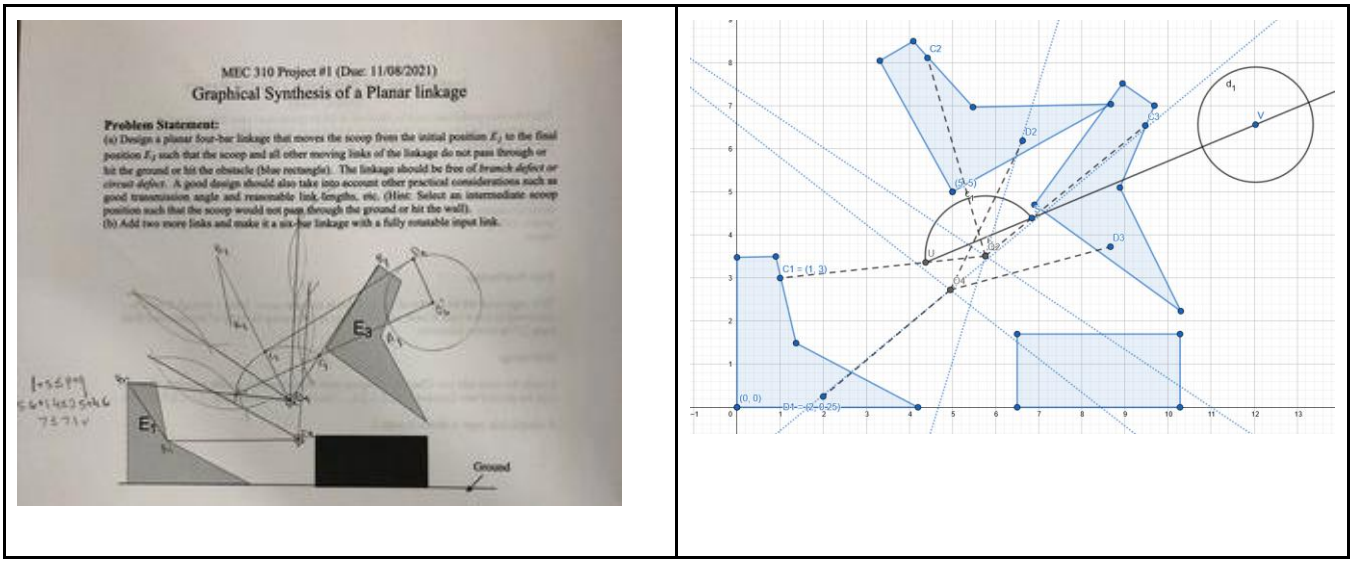

Figure 2.8: Failed Iterations Due to Defects

3. Design Analysis and Simulation

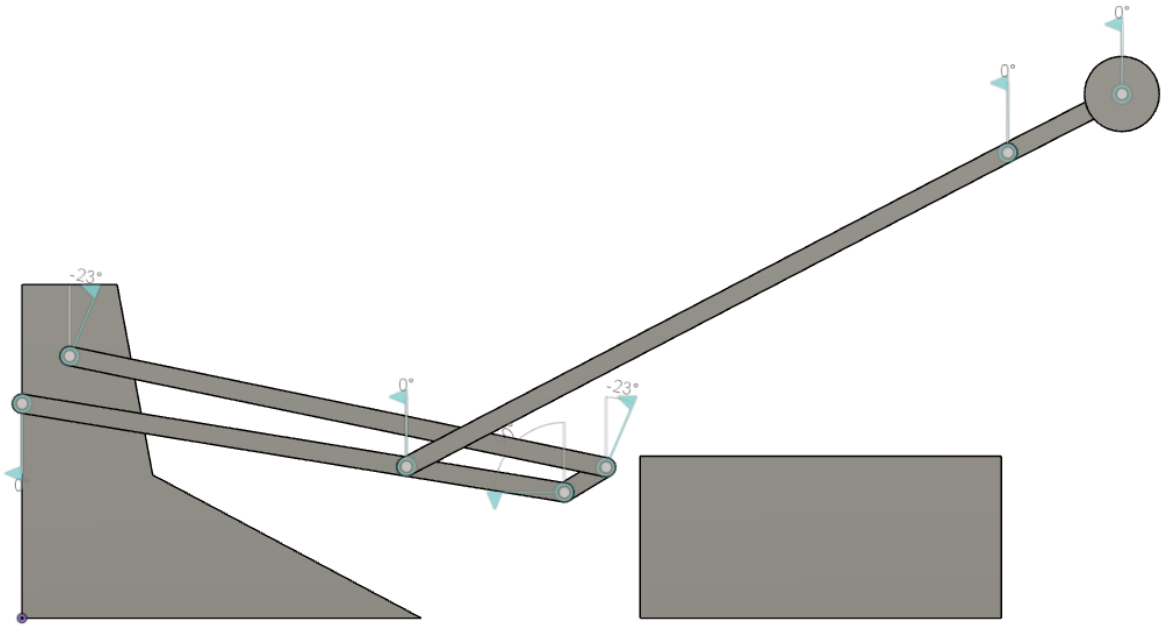

Figure 3.1: The initial position of the scoop during motion

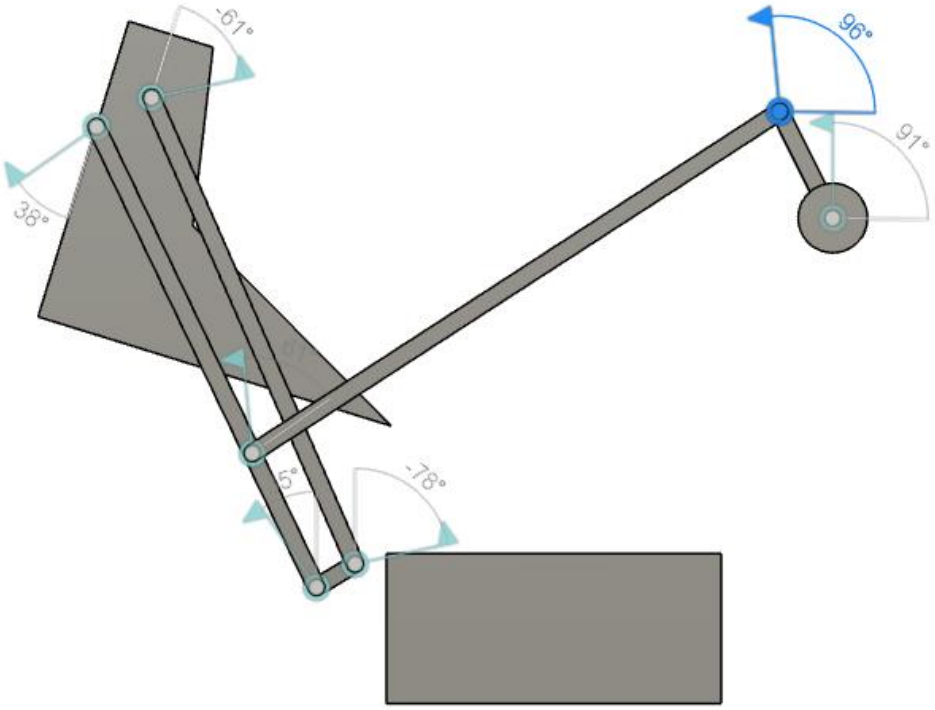

Figure 3.2: The Intermediate Position of The Scoop During Motion

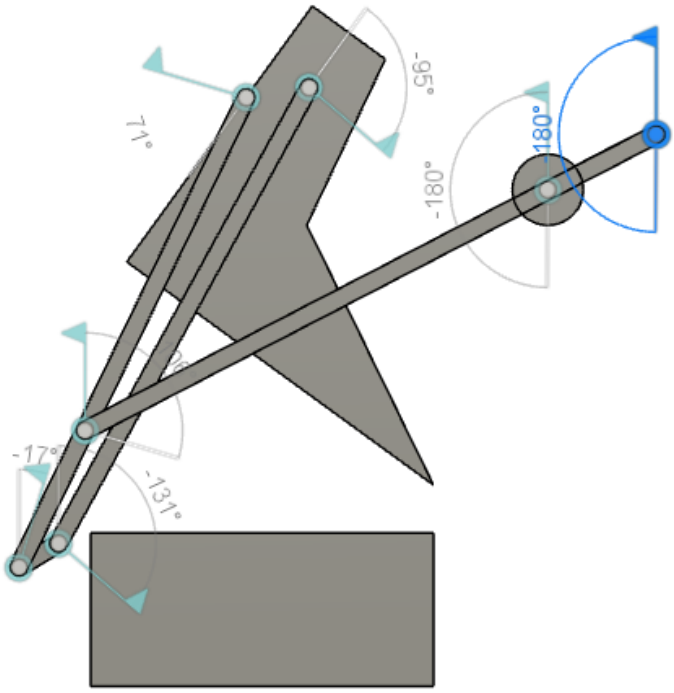

Figure 3.3: The Final Position of The Scoop During Motion

4. Discussions and Conclusions

In this assignment, our group was able to determine the optimal configuration of the six-bar mechanism to accomplish our goal. In the early stages of the project, numerous attempts were made graphically to determine the configuration of the six-bar linkage that would not conflict with the ground and the obstacle. Most of the iterations contain either a branch or circuit defect or did not satisfy the Grashof criterion. After using many failed attempts, around five of the iterations from all of the group members satisfied specifications and design criteria. Initially, iteration with a reasonable link length was created, however, the transmission angle was found to be zero. Thus, several attempts were made to improve on the initial designs. After careful consideration and selection of the intermediate points, an iteration with a non zero transmission angle while maintaining the desired motion was generated. However, the design was not up to our expectation due to a small transmission angle of around 9°.

With further improvements on the iterations, two functional solutions with reasonable link lengths and transmission angles of 23° and 27.4° were obtained. Using the MotionGen program, the motions and defects of the iterations can be observed through simulations. As a result, none of our functional iterations contain circuit or brand defects. However, one of the group members wrote a program that finds the locations of the scoop and points for the pivots in order to maximize the transmission angle of the mechanism. The program was able to generate a mechanism with a transmission angle of around 81°, which is extremely close to the optimal angle of 90°. However, the ground link lengths are not reasonable. Other mechanisms with high transmission angles, between 50° and 80° were generated, but most contained circuit or branch defects. Thus, we chose a final design with a transmission angle of around 42.99° and reasonable link lengths. According to the data and results obtained, we have successfully designed an optimal six-bar linkage with a fully rotatable input link to complete our task of moving a scoop from an initial to a final position without hitting the ground or the obstacle.

This project required several factors to be taken into account to try to create the best mechanism to move a scoop. An improvement that could be made to the procedure could be to generalize the graphical synthesis to test faster and with fewer failed attempts. Another possible improvement to the procedure of this project would be to optimize the transmission angle not by testing many trials, but rather to systematically zone in on the largest transmission angle using something like a binary search. It would be interesting to continue this by looking into how different parts of the mechanism affect its performance — such as the distance or angle between pivots or positions of the scoop affecting transmission angle. Overall, this project served to introduce students to the design process, trial and error involved, and various aspects that need to be kept in mind throughout the procedure.

5. Contributions

Daniel contributed design iterations by writing the Python program to test a large number of mechanisms quickly. Many iterations did not work due to defects, unreasonable link lengths, or intersection of links with the rectangular obstacle, but these were dealt with by altering the program. The final iteration using this method was chosen to be simulated and was modeled on MotionGen. He also helped in writing as well as revising the final paper.

Dalton contributed six design iterations and worked with Chengji on the MotionGen and Fusion 360 animations. Only one iteration worked with the given parameters, as the other eight either did not satisfy Grashof's Law, had a transmission angle of zero, or had circuit defects. One of his iterations had a transmission angle of 22.7° which was the third highest. He made this iteration in an attempt to improve on the first iteration which had a transmission angle of 9.29°. Dalton mainly worked on the six bar portion of the Fusion 360 design, and helped Chengji with making the four bar linkage mechanism. Dalton also wrote and revised the final paper.

Chengji contributed five design iterations, in which two of them satisfied the design specification and design criteria and worked with Dalton to model the final design on Fusion 360 . The best design out of the iterations came out to have a reasonable length and a maximum transmission angle of 27.4° with no circuit or branch defects. With the help of Dalton, the CAD model that was created using fusion 360 turns out to be working perfectly. He also helped in writing up the final report for the project. Chengji mainly worked on the design iterations of the six-bar mechanism, writing up the report, and the design of the CAD for the final iteration.

Appendix

Python Program

```
# To model the various shapes, lines, and points involved in the design process, the shapely Python project was used.
import math
from shapely import geometry
from shapely.geometry import Point
from shapely.geometry import Polygon
from shapely import affinity

# This function determines whether a point is inside a polygon. It takes into account if the point lies on the boundary or is a vertex.
def in_polygon(point, polygon):
    return polygon.contains(Point(point)) or point in list(zip(*polygon.exterior.coords.xy)) or geometry.LineString(list(zip(*polygon.exterior.coords.xy))).contains(Point(point))

# This function converts a point in LRCS of scoop into coordinates in the GCS given the angle and offset of the scoop.
def LRCStoGCS(LRCSpoint, angle, offset):
    point = list(affinity.translate(affinity.rotate(Point(LRCSpoint), angle, Point(0,0)), offset[0], offset[1]).coords)[0]
    return [point[0], point[1]]

# This function returns the distance between two points
def dist(pt1, pt2):
    return math.sqrt((pt2[0]-pt1[0])**2 + (pt2[1]-pt1[1])**2)

# This function finds fixed pivots corresponding to three moving pivots(GCS coords) using three-point analytical synthesis
def findPivot(vectorC1, vectorC2, vectorC3):
    vectorH2 = [vectorC1[0]-vectorC2[0], vectorC1[1]-vectorC2[1]]
    vectorH3 = [vectorC1[0]-vectorC3[0], vectorC1[1]-vectorC3[1]]
    H2x = vectorH2[0]
    H2y = vectorH2[1]
    H3x = vectorH3[0]
    H3y = vectorH3[1]
    F2 = 0.5 * (vectorC1[0]*vectorC1[0]+vectorC1[1]*vectorC1[1] - vectorC2[0]*vectorC2[0]-vectorC2[1]*vectorC2[1])
    F3 = 0.5 * (vectorC1[0]*vectorC1[0]+vectorC1[1]*vectorC1[1] - vectorC3[0]*vectorC3[0]-vectorC3[1]*vectorC3[1])
    Gx = (F3-((H3y/H2y)*F2))/(H3x-(H3y*H2x/H2y))
    Gy = (F2-H2x*Gx)/H2y
    return [Gx, Gy]

# GLOBAL VARIABLES
LRCScoords = []
maxtransmissionangle = 0

# CONSTANTS
# scoop in intial position
```

```
E1points = [(0,0),(0,3.48),(0.90,3.50),(1.37,1.49),(4.20,0)]
E1 = Polygon([[0,0],[0,3.48],[0.90,3.50],[1.37,1.49],[4.20,0]])
E1line = geometry.LineString(E1points)

# rectangular obstacle
Rpoints = [(6.48,0),(6.48,1.70),(10.28,1.70),(10.28,0)]
R = Polygon(Rpoints)
Rline = geometry.LineString(Rpoints)

# scoop in final position
angle3 = -36
offset3 = [6.9,4.7]
E3 = affinity.translate(affinity.rotate(E1, angle3, Point(0,0)),offset3[0],offset3[1])
E3points = list(zip(*E3.exterior.coords.xy))
E3line = geometry.LineString(E3points)

# coordinates of the ground
groundpoints = [(-100, -0.1), (100, -0.1)]
ground = geometry.LineString(groundpoints)

# find moving pivots in locally rotating coordinate system (precalculated for runtime)
for LRCSx in range(0, 16):
    LRCSx = LRCSx/4
    for LRCSy in range(0, 14):
        LRCSy = LRCSy/4
        if(not in_polygon([LRCSx, LRCSy], E1)):
            break
        LRCScoords.append([LRCSx, LRCSy])

# find angle/offsets that give shape that does not intersect with ground or rectangle
for angle2 in range(0, 18): # 90/5
    angle2 = angle3 + angle2 * 2
    for o2x in range(1, 7):
        for o2y in range(1, 7):
            offset2 = [o2x, o2y]
            E2 = affinity.translate(affinity.rotate(E1, angle2, Point(0,0)), offset2[0], offset2[1])
            E2points = list(zip(*E2.exterior.coords.xy))
            E2line = geometry.LineString(E2points)
            if(not (E2.intersection(R).is_empty)):
                break
            if(not (E2.intersection(ground).is_empty)):
                break

# find moving pivots in global coordinate system
            GCS1 = []
            GCS2 = []
            GCS3 = []
            for coord in LRCScoords:
                GCS1.append(coord)
                GCS2.append(LRCStoGCS(coord, angle2, offset2))
                GCS3.append(LRCStoGCS(coord, angle3, offset3))
```

```
# find fixed pivot given 3 positions of moving pivot
            pivots = []
            i = 0
            while i < len(GCS1):
                p = findPivot(GCS1[i], GCS2[i], GCS3[i])
                # if the coordinates are too far, skip
                if(p[1] < 0):
                    GCS1.pop(i)
                    GCS2.pop(i)
                    GCS3.pop(i)
                    continue
                elif(p[0] > 10 or p[1] > 10):
                    GCS1.pop(i)
                    GCS2.pop(i)
                    GCS3.pop(i)
                    continue
                else:
                    i += 1
                    pivots.append(p)

            for i in range(0, len(GCS1)):
                for j in range(i+1, len(GCS1)):
                    moving_pivot1 = GCS1[i]
                    moving_pivot2 = GCS1[j]
                    aa = GCS2[i]
                    ab = GCS2[j]
                    ac = GCS3[i]
                    ad = GCS3[j]
                    fixed_pivot1 = pivots[i]
                    fixed_pivot2 = pivots[j]
                    line1 = geometry.LineString([Point(fixed_pivot1), Point(moving_pivot1)])
                    line2 = geometry.LineString([Point(fixed_pivot2), Point(moving_pivot2)])
                    # if it intersects with the rectangle, skip
                    if(line1.intersects(R) or line2.intersects(R)):
                        continue
                    O2C2 = geometry.LineString([Point(fixed_pivot1), Point(aa)])
                    O4D2 = geometry.LineString([Point(fixed_pivot2), Point(ab)])
                    if(O2C2.intersects(O4D2)):
                        continue
                    a = dist(fixed_pivot1, moving_pivot1)
                    b = dist(moving_pivot1, moving_pivot2)
                    c = dist(moving_pivot2, fixed_pivot2)
                    d = dist(fixed_pivot1, fixed_pivot2)
                    s = min(a, b, c, d)
                    l = max(a, b, c, d)
                    # if not grashof, skip
                    if((l + s) > (a+b+c+d-l-s)):
                        continue
                    # if the shortest length is too small, skip
                    if(s < 0.5):
                        continue
```

```
                else:
                    if((b**2+c**2-(d-a)**2)/(2*b*c) > 1):
                        continue
                    mu1 = math.acos((b**2+c**2-(d-a)**2)/(2*b*c))
                    mu2 = math.acos((b**2+c**2-(d+a)**2)/(2*b*c))
                    if(mu1 > math.pi/2):
                        mu1 = math.pi - mu1
                    if(mu2 > math.pi/2):
                        mu2 = math.pi - mu2
                    mu = min(mu1, mu2)
                    if(mu > maxtransmissionangle):
                        maxtransmissionangle = mu
                        print("ANGLE2: ", angle2)
                        print("OFFSET2: ", offset2)
                        print("A1: ", moving_pivot1)
                        print("B1: ", moving_pivot2)
                        print("A2: ", aa)
                        print("B2: ", ab)
                        print("A3: ", ac)
                        print("B3: ", ad)
                        print("mu1: ",  mu1*180/math.pi)
                        print("mu2: ", mu2*180/math.pi)
                        print("SIDE LENGTHS: ", a, b, c, d)
                        print("FP1: ", fixed_pivot1)
                        print("FP2: ", fixed_pivot2)
                        print(math.degrees(mu))
                        print()
                                    print()
```