Nat Curtis
ME 3313
9/10/2025

ME 3313 Honors Project

Introduction

For this project, I am analyzing the linkages of a skid-steer bucket loader, shown in Figure 1. I will explore each of the topics we explore in ME 3313, starting with mobility, and going on to topics including range of motion, position, velocity, acceleration, and forces, and use each of these topics to analyze the linkage. I will then explore synthesis and try to create my own linkage for the bucket loader.

Figure 1: Side view of a Skid

Mobility

Mobility is a measure of how many degrees of freedom a linkage has, and it is calculated with Equation (1), called the Grubler-Kutzbach Mobility equation, where M is the mobility, L is the number of links, J1 is the number of full joints, and J2 is the number of half joints.

$$M = 3(L-1) - 2J_1 - J_2 \tag{1}$$

A link is a single rigid body. A full joint is a joint with 1 degree of freedom, like a pin or two flat surfaces sliding along each other. A half joint is a joint with 2 degrees of freedom, like a pin in a joint, or two curved surface rolling or slipping along each other. On a kinematic diagram of a linkage, each link is marked with a unique plain number from 1 to the number of links, each full joint is marked with a unique circled number from 1 to the number of full joints, and each half joint is marked with a unique boxed number from 1 to the

number of half joints. As shown in Figure 2, the main linkage of the skid-steer has 6 links, 7 full joints, and 0 half joints. Using the mobility equation as shown in Equation (2), the mobility is calculated to be 1.

$$M = 3(L-1) - 2J_1 - J_2 = 3(6-1) - 2*7 - 0 = 1 \tag{2}$$

Because the mobility is 1, only one input is needed to control the linkage. In this case, the input is a hydraulic piston, which is a combination of links 4 and 6. Figure 3 shows the annotated linkage diagram overlain on the original image of the skid-steer.

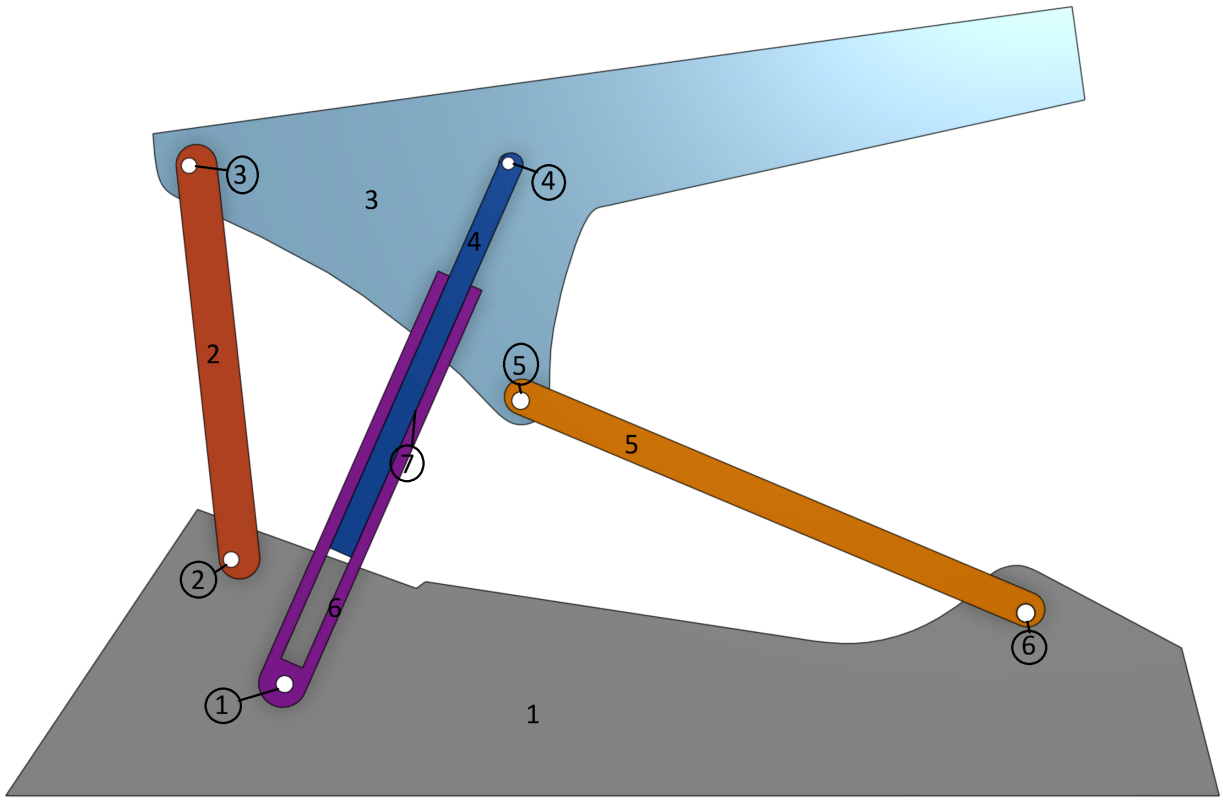

Figure 2: Annotated Linkage

Figure 3: Annotated Linkage Overlain On Skid-Steer

Multiloop Closure Equations

Mobility allows us to determine the degrees of freedom of a linkage, but usually we also need to determine the positions and angles of the individual links. To do this, we can use a method called Multiloop closure. It consists of drawing vectors between joints of each link, then writing equation summing loops of vectors, because the RHS of the equation has to equal zero. This is because the last vector ends where the first vector starts. These vector equations can then be turned into their scalar forms, which can be combined with equations relating rigid geometry of the links, providing a system of equations to solve. When done correctly, the number of unknowns minus the number of scalar and rigid geometry equations should equal the mobility. For the examined linkage, the vectors and loops are labeled as shown in Figure 4. The vector equations are shown below as eqs. (3) and (4).

$$\vec{R_{BA}} + \vec{R_{DB}} - \vec{R_{CA}} - \vec{R_{GC}} - \vec{R_{DG}} = 0 \tag{3}$$

$$\vec{R_{GC}} + \vec{R_{DG}} - \vec{R_{DE}} - \vec{R_{EF}} - \vec{R_{FC}} = 0 \tag{4}$$

These equations can be expanded into their scalar forms as shown in eqs. (5) to (10) below, where each θ is measured counter-clockwise from the horizontal to the tail of the corresponding vector.

$$R_{BA}\cos\theta_{BA} + R_{DB}\cos\theta_{DB} - R_{CA}\cos\theta_{CA} - R_{GC}\cos\theta_{GC} - R_{DG}\cos\theta_{DG} = 0 \tag{5}$$

$$R_{BA}\sin\theta_{BA} + R_{DB}\sin\theta_{DB} - R_{CA}\sin\theta_{CA} - R_{GC}\sin\theta_{GC} - R_{DG}\sin\theta_{DG} = 0 \tag{6}$$

$$R_{GC}\cos\theta_{GC} + R_{DG}\cos\theta_{DG} - R_{DE}\cos\theta_{DE} - R_{EF}\cos\theta_{EF} - R_{FC}\cos\theta_{FC} = 0 \tag{7}$$

$$R_{GC}\sin\theta_{GC} + R_{DG}\sin\theta_{DG} - R_{DE}\sin\theta_{DE} - R_{EF}\sin\theta_{EF} - R_{FC}\sin\theta_{FC} = 0 \tag{8}$$

$$\theta_{GC} - \theta_{DG} = 0 \tag{9}$$

$$\theta_{DB} + \beta_D - \theta_{DE} = 0 \tag{10}$$

In these equations, R_{BA}, R_{DB}, R_{CA}, θ_{CA}, R_{GC}, R_{DE}, R_{EF}, R_{FC}, θ_{FC}, and β_D are known from the geometry of the linkage and do not change. This leaves θ_{BA}, θ_{DB}, θ_{GC}, R_{DG}, θ_{DG}, θ_{DE}, and θ_{EF} as unknowns for a total of 7 unknowns. Taking 7 unknowns minus 6 equations yields 1 degree of freedom, which agrees with the earlier mobility analysis. Also, R_{DG} will be the input because it is controlled by a hydraulic piston.

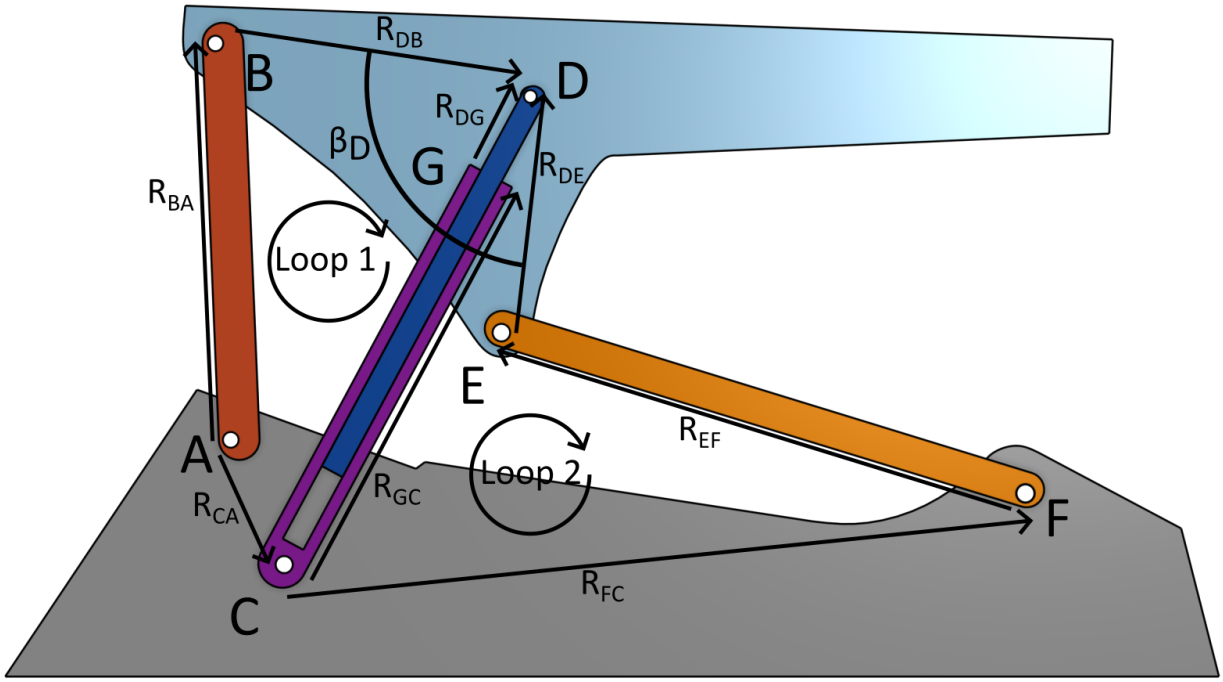

Figure 4: Diagram showing vector for Multiloop Equations

Multiloop Solver Code

```
import numpy as np
import math
from scipy.optimize import fsolve
from warnings import catch_warnings

#act_img_ratio = 0.1718 # Approximate ratio of length without bucket from specs in inches over length without bucket from image in mm

act_img_ratio = 1
# Known values (For the moment just measured in millimeters from image)
rba = 142.27*act_img_ratio
rdb = 115.2*act_img_ratio
rca = 49.46*act_img_ratio
```

```
thetaca = math.radians(293.04)
rgc = 148.5*act_img_ratio # Can't really tell from picture, but should not affect results as long as counted for accordingly
rde = 85.65*act_img_ratio
ref = 197.12*act_img_ratio
rfc = 267.12*act_img_ratio
thetafc = math.radians(5.44) # angle from positive x-axis to ground link, which is Rea
betad = math.radians(92.47) # Angle from DB to DA

# Inputs
rdc = 173

rdg = rdc - rgc # Given input length

# Finds most common element in a list
def most_common(lst: list):
    return max(set(lst), key=lst.count)


"""
Unknowns:
x[0]: thetaba
x[1]: thetadb
x[2]: thetagc
x[3]: thetadg
x[4]: thetade
x[5]: thetaef
"""
unknown_length_indices = []

"""# System of scalar equations to solve
def eqs(x):
    return [

        rba*np.cos(x[0]) + rdb*np.cos(x[1]) - rca*np.cos(thetaca) - rgc*np.cos(x[2]) - rdg*np.cos(x[3]),
        rba*np.sin(x[0]) + rdb*np.sin(x[1]) - rca*np.sin(thetaca) - rgc*np.sin(x[2]) - rdg*np.sin(x[3]),
        rgc*np.cos(x[2]) + rdg*np.cos(x[3])  -rde*np.cos(x[4]) - ref*np.cos(x[5]) - rfc*np.cos(thetafc),
        rgc*np.sin(x[2]) + rdg*np.sin(x[3]) - rde*np.sin(x[4]) - ref*np.sin(x[5]) - rfc*np.sin(thetafc),
        x[2]-x[3],
        x[1]+betad-x[4]
    ]"""

# test
def eqs(x):
    return [

        rba*np.cos(x[0]) + rdb*np.cos(x[1]) - rca*np.cos(thetaca) - rgc*np.cos(x[2]) - rdg*np.cos(x[3]),
        rba*np.sin(x[0]) + rdb*np.sin(x[1]) - rca*np.sin(thetaca) - rgc*np.sin(x[2]) - rdg*np.sin(x[3]),
```

```
        rgc*np.cos(x[2]) + rdg*np.cos(x[3])  -rde*np.cos(x[4]) -
        ↪  ref*np.cos(x[5]) - rfc*np.cos(thetafc),
        rgc*np.sin(x[2]) + rdg*np.sin(x[3]) - rde*np.sin(x[4]) -
        ↪  ref*np.sin(x[5]) - rfc*np.sin(thetafc),
        x[2]-x[3],
        x[1]+betad-x[4]
    ]

# Rename pi for shorter call
pi = math.pi

# For finding right guesses
initial_guess_min = [0, 0, 0, 0, 0, 0]
initial_guess_max = [2*pi, 2*pi, 2*pi, 2*pi, 2*pi, 2*pi]
current_guess = [0, 0, 0, 0, 0, 0]

var_choices = []


# Finds best guess for each variable
for var in range(len(current_guess)):
    # Generates list of guesses from min to max
    guesses = np.linspace(initial_guess_min[var], initial_guess_max[var], 1000)

    results = [] # Holds all results including Nones
    results_wo_none = [] # Holds all results except for Nones


    # Finds results from each guess, converting to degrees and constraining to
    ↪  360 degrees for angles

    for guess in guesses:
        current_guess[var] = guess


        # If no
        with catch_warnings(record=True) as w:
            sol = fsolve(eqs, current_guess)
            if w:
                results.append(None)
                continue
        if not var in unknown_length_indices:
            sol[var] = math.degrees(sol[var]) % 360
        results.append(float(np.round(sol[var], 3)))
        results_wo_none.append(float(np.round(sol[var], 3)))

    res_set = set(results_wo_none)
    var_choices.append(list(res_set))
    # Sets guess for current variable to guess resulting in most common solution
    current_guess[var] = guesses[results.index(most_common(results_wo_none))]
```

```
112
113 # Finds solution for guesses
114 s = fsolve(eqs, current_guess)
115
116 # Converts angles to degrees from 0 to 360 and rounds all results to 3 decimal
    ↪ places
117 for numb in range(len(s)):
118     if not numb in unknown_length_indices:
119         s[numb] = math.degrees(s[numb]) % 360
120     s[numb] = round(s[numb], 3)
121
122 # Prints results
123 print(f"Theta_BA = {s[0]} degrees")
124 print(f"Theta_DB = {s[1]} degrees")
125 print(f"Theta_GC = {s[2]} degrees")
126 print(f"Theta_DG = {s[3]} degrees")
127 print(f"Theta_DE = {s[4]} degrees")
128 print(f"Theta_EF = {s[5]} degrees")
```