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.

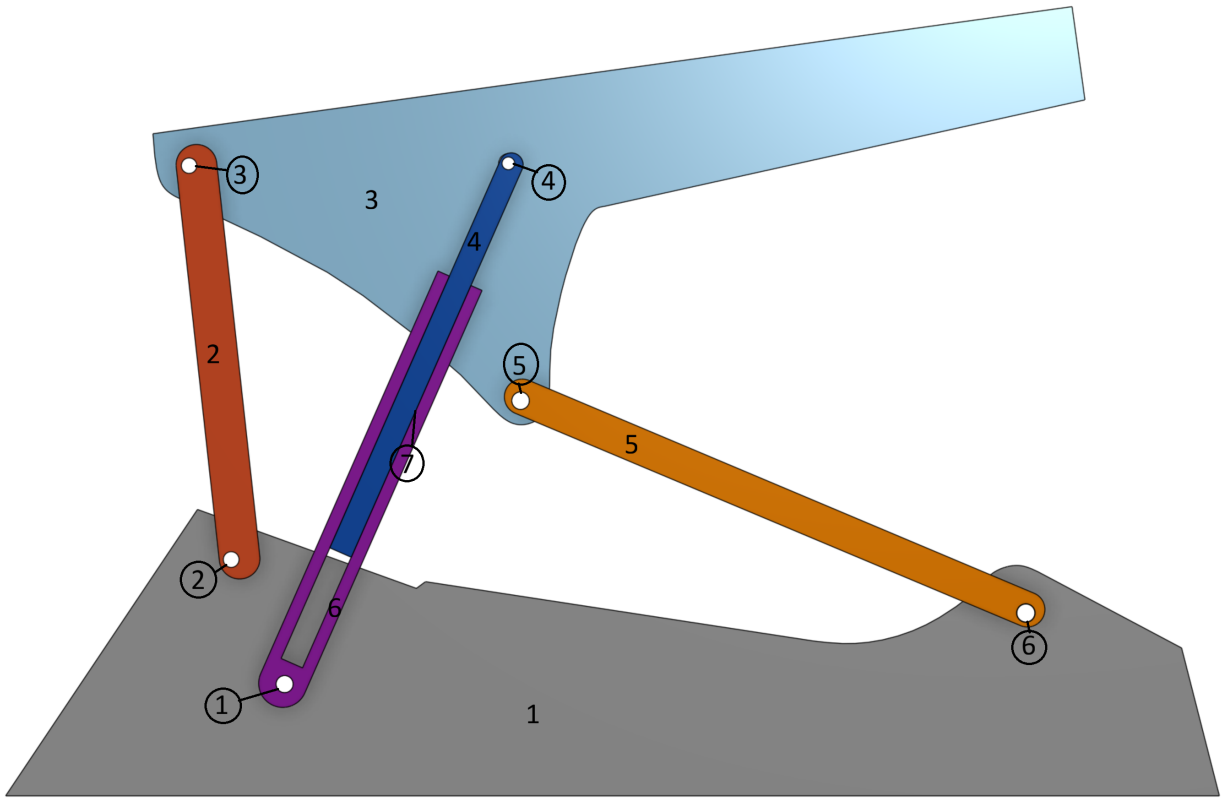

Figure 2: Annotated Linkage

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.

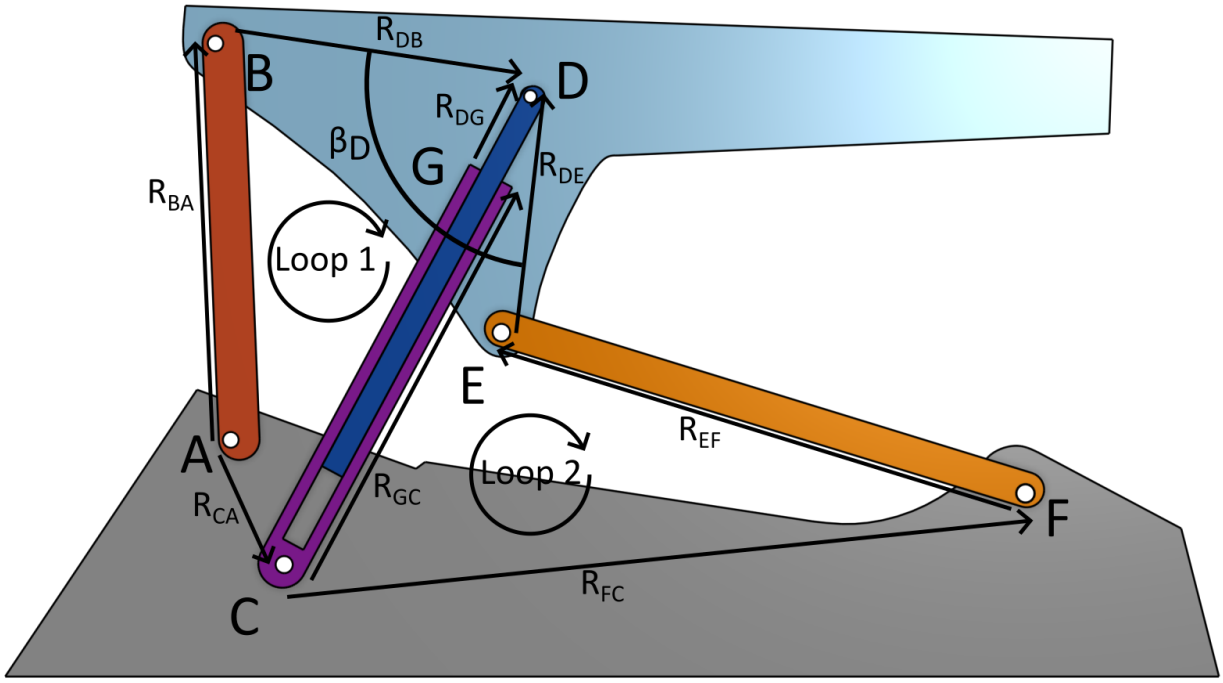

Figure 4: Diagram showing vector for Multiloop Equations

Code Introduction

To solve these equations, I used Python code taking advantage of the fsolve function from the SciPy library. This is a numerical solver function that takes in a set of equations and initial guesses and outputs a solution. Depending on the initial guess, this solution can be incorrect, or a valid solution but not the correct solution for the actual problem constraints. To increase the chancess significantly of a valid solution being generated, I generated a range of initial condition options for each unknown variable and then generated a solution for every combination of initial conditions. Taking these, I kept only ones that had little error when plugged back into the multiloop equations and each variable was within an estimated range of motion. The known link lengths that were plugged into the equations are shown below, and the input length is shown under that. These measurements are only estimates, however, because I found them by finding the length of the loader in its documentation, then measured the length in a side view image of the loader. I took the ratio of these, then measured each link length and and angle in the picture and multiplied each link length by the ratio to get the estimated actual link length. Angles however are not affected by scaling an image up or down though, so I used the measured angles as they were.

Link Measurements

$$
\begin{aligned}
R_{BA} &= 24.4 \text{ in.} \\
R_{DB} &= 19.8 \text{ in.} \\
R_{CA} &= 8.5 \text{ in.} \\
\theta_{CA} &= 293.04^\circ \\
R_{GC} &= 25.5 \text{ in.} \\
R_{DE} &= 14.7 \text{ in.} \\
R_{EF} &= 33.9 \text{ in.} \\
R_{FC} &= 45.9 \text{ in.} \\
\theta_{FC} &= 5.44^\circ \\
\beta_D &= 92.47^\circ
\end{aligned}
$$

Input Link Length

$$R_{DC} = 29.72 \text{ in.}$$

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     # Uncomment to use image link lengths instead of approximated actual lengths
# Known values (For the moment just measured in millimeters from image multiplied by ratio)
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
```

```python
rdc = 173*act_img_ratio

#print(rba, rdb, rca, rgc, rde, ref, rfc, rdc)
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 = []

num_equations = 6

# 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]
    ]

# 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]

# Create empty set to store unique solutions
gen_solutions = set()

# Check if solution is in approximate range of motion
def result_check(result) -> bool:
    if result[0] > 135 or result[0] < 45:

        return False
```

```
    if result[1] > 90 and result[1] < 270:

        return False
    if result[2] > 90:

        return False
    if result[3] > 90:

        return False
    if result[4] > 180:

        return False
    if result[5] < 90 or result[5] > 180:

        return False
    return True

# Check error of solution
def calc_err(result, eq=eqs):
    err = eq(result)
    err_sum = 0
    for i in range(len(err)):
        err_sum += float(np.abs(err[i]))
    return err_sum


# Populate the solution set with valid solutions
def solution_gen(current_guess: list[float] | None = None, guess_per_var: int =
↪   5, index: int = 0, initial_min: list[float] = initial_guess_min,
↪   initial_max: list[float] = initial_guess_max, output: set[tuple[float, ...]]
↪   = gen_solutions, eq=eqs, num_var: int = num_equations,
↪   unkn_len_ind=unknown_length_indices):
    if current_guess is None:
        current_guess = []
    if len(initial_min) ≠ num_var and len(initial_max) ≠ num_var:
        exit("Initial and/or final guess are wrong length")

    if index == num_var:
        with catch_warnings(record=True):
            sol = fsolve(eq, current_guess)
            curr_error = calc_err(sol, eq)
            for numb in range(len(sol)):
                if not numb in unkn_len_ind:
                    sol[numb] = math.degrees(sol[numb]) % 360
                sol[numb] = round(sol[numb], 1)
            if result_check(sol) and curr_error < 1:
                output.add(tuple(sol))
        return

    guesses = np.linspace(initial_min[index], initial_max[index], guess_per_var)

    # Finds solutions with all different combinations of guesses
    for guess in guesses:
```

```
123             solution_gen(current_guess=current_guess+[guess],
                ↪  guess_per_var=guess_per_var, index=index+1, initial_min=initial_min,
                ↪  initial_max=initial_max, output=output, eq=eq, num_var=num_var,
                ↪  unkn_len_ind=unkn_len_ind)
124         return
125
126     # Generate solutions
127     solution_gen()
128
129     # Turn set into a list so it is indexable
130     gen_solutions = list(gen_solutions)
131
132
133     # Grab first solution (There will probably only be one)
134     s = gen_solutions[0]
135
136
137     # Prints results
138     print(f"Theta_BA = {s[0]} degrees")
139     print(f"Theta_DB = {s[1]} degrees")
140     print(f"Theta_GC = {s[2]} degrees")
141     print(f"Theta_DG = {s[3]} degrees")
142     print(f"Theta_DE = {s[4]} degrees")
143     print(f"Theta_EF = {s[5]} degrees")
```

Multiloop Solver Output

```
PS C:\...\Honors> python code/main.py
Theta_BA = 82.2 degrees
Theta_DB = 335.1 degrees
Theta_GC = 52.9 degrees
Theta_DG = 52.9 degrees
Theta_DE = 67.6 degrees
Theta_EF = 170.2 degrees
```