Department of Mechanical and Mechatronics Engineering

Statics of a Robotic Manipulator
0.23978Nm

A Report Prepared For:

The University of Waterloo

Prepared By:

Eric Zhao, Keith Lai, Tiger Ye

April 12, 2021

Overview

Robotic manipulators are composed of rigid links connected by joints with one fixed-base and one free-moving end. These links and joints make up the arm and body of the robotic manipulator while the gripper/wrist of the arm is attached to the free-moving end. This allows the arm to freely move within its environment to perform specific repetitive tasks such as moving an object from one location to another by using the wrist to pick up objects (if within range of the total arm length).

The goal of this project was to determine the 3 link lengths of a 3 DOF planar serial robotic manipulator that minimizes the torque generated on the base motor for 3 given configurations. The 3 different positions the arm must be in are detailed below in Table 1.

Table 1: Configurations of the Robotic Manipulator During Its Operation

	Gripper Location	Link 3 Orientation
Position 1	x = 0.75m, y = 0.1m	q_3 = -60° w.r.t the x-axis
Position 2	x = 0.5m, y = 0.5m	q_3 = 0° w.r.t the x-axis
Position 3	x = 0.2m, y = 0.6m	q_3 = 45° w.r.t the x-axis

When designing the manipulator arm, the material, width, and thickness of the links were already determined as the links' geometry constants were 4, 2 and 1 for link 1, 2 and 3, respectively. The gripper always carried a load of 5 kg at the end of the arm, and the gripper had to be capable of extending a minimum distance of 1 meter such that the sum of the 3 links lengths was at least 1 meter.

To determine the weight of the links to help determine the torque, Equation 1 below was used.

$$W_n = \alpha_n \times L_n \qquad (1)$$

In Equation 1, W_n is the weight, L_n is the length and α_n is the geometry constant of the nth link.

With all these considerations and constraints in mind, a variety of unique robotic manipulator designs were brainstormed. First, when approaching the problem, possible solutions were designed in Solidworks. To minimize the torque, the various link lengths were decreased/increased from a basic solution and the resulting torque generated was recalculated to determine whether the adjustment helped decrease the torque. Finally, it was decided that a program would be the best method to find the appropriate link lengths that generated the lowest torque possible. After creating the program, the 3 link lengths calculated were then used to create the drawing design of the robotic manipulator. The final design created is detailed in the Design section. It was assumed that the robotic manipulator could extend into both the –x and –y sections.

Code Rundown

To maximize efficiency while finding the most optimal solution to this problem, a program was created in C++ to simulate various valid design iterations. The code can be found in Appendix B. Classes were created for points, and links, to store x and y coordinates and link geometry constants, start and end points, respectively.

Since the force of the weights of all the links acts directly downwards, the torque from each of the links can be calculated by simply multiplying the horizontal x distance from the middle of the link to the origin, or base motor. Summing up these torques, the following Equation 2 can be found to calculate the total torque of each orientation, with Δx representing the change in x between the start and end joint positions of each link:

$$\tau = \left(\frac{\Delta x_{l1}}{2}\right) * W_1 + \left(\Delta x_{l1} + \frac{\Delta x_{l2}}{2}\right) * W_2 + \left(\Delta x_{l1} + \Delta x_{l2} + \frac{\Delta x_{l3}}{2}\right) * W_3 + (\Delta x_{l1} + \Delta x_{l2} + \Delta x_{l3}) * W_{Load} \quad (2)$$

This was coded as the “getTorque” function in the program, functions that would find the change in x for each of the links and was used to calculate the torque generated by each of the tested designs.

In addition to the individual torques, a “getTotalTorque” function was coded to calculate the parameter T based on the definition shown in Equation 3:

$$T = \sqrt{\tau_1{}^2 + \tau_2{}^2 + \tau_3{}^2} \quad (3)$$

As each given position provides a given gripper location, along with L3’s angle with respect to the x-axis, the only unknown for Link 3 is its length. Once L3 is found, the start-point of Link 3 can be calculated using Equation 4.

$$\mathrm{P2} = (\mathrm{x0} \pm \mathrm{L3} * \cos(\theta_{wrt-x}), y0 \pm \mathrm{L3} * \cos(\theta_{wrt-x})) \quad (5)$$

Whether the cos(θ) is added or subtracted depends on the angle, and the direction that the arm extends from the grabber point.

By inputting lengths for Link 2 and Link 1 as well, the cosine law can be used to calculate the angle shown in Figure 1 below, which is shown in Equation 5.

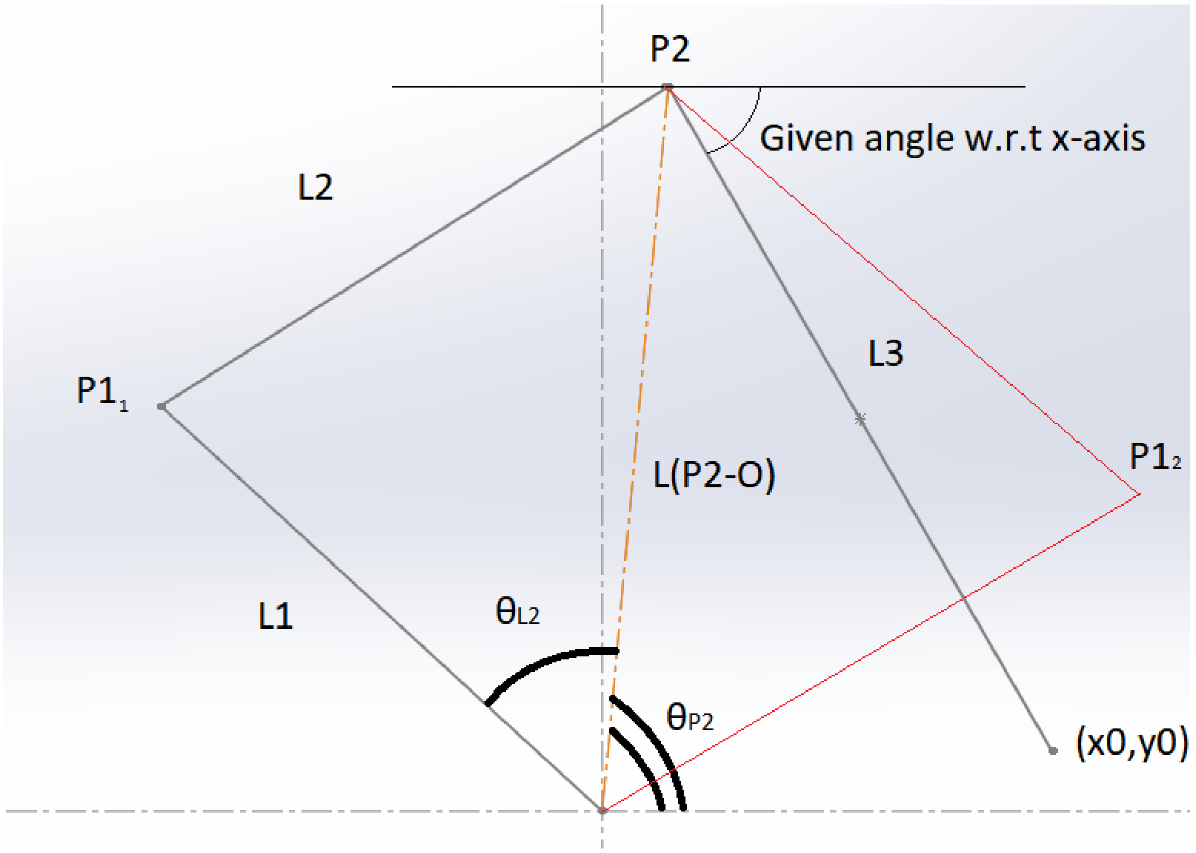


Figure 1: Labelled diagram of an example of how arm orientations were solved given L1, L2, and L3.

$$\theta_{L2} = \cos^{-1}\left(\frac{L2^2 - L1^2 - L_{P2}{}^2}{-2 * L1 * L_{P2}}\right) \quad (5)$$

By adding or subtracting this angle from the angle that P2 makes with the origin found by Equation 6, the entire system can be solved with Equations 7 and 8, leading to two possible arm orientations for each combination of 3 link lengths L1, L2, and L3 for each position.

$$\theta_{P2} = tan^{-1}\left(\frac{y_{P2}}{x_{P2}}\right) \quad (6)$$

$$P1_1 = (L1 * \cos(\theta_{P2} + \theta_{L2}), L1 * \sin(\theta_{P2} + \theta_{L2})) \quad (7)$$

$$P1_2 = (L1 * \cos(\theta_{P2} - \theta_{L2}), L1 * \sin(\theta_{P2} - \theta_{L2})) \quad (8)$$

With all the joint point locations and link locations found, the torque is calculated with Equation 2 for both possible orientations, and the lower magnitude of the two is returned for this specific combination of L1, L2, L3, and position in functions “firstPosition”, “secondPosition”, and “thirdPosition”.

However, as shown in Figure 1 with the red second orientation $P1_2$, there may be orientations where Link 1 and Link 3 would cross over each other, which would be impossible in this 2D scenario. Thus, before the minimum torque was found, both orientations were checked to ensure that no links intersect each other using function “doIntersect”.

With functions in place to implement all the calculations, all that was left to do was to iterate through as many possible link length measurements for each of the three positions and take the combination that gives the lowest torque parameter, calculated with Equation 3. For this, a max length variable was declared, representing the maximum total length of the arm. Nested for loops were used to iterate through all possible combinations of L1, L2, and L3 that added up to the max length, and the max length was also increased from 1m to 10m to check for all reasonable combinations, which is all done in function “combinations”.

By running the program as stated, it was found that a L1, L2, and L3 combination of 0.88m, 2.48m and 2.23m provided the lowest torque parameter of 0.23978Nm (with a max arm length of 5.59m) which was chosen to be the final design. The code used can be found in Appendix B.

Design

The final design consisted of the following lengths for the links, shown in Table 2:

Table 2: Final design link lengths.

Link Number	**Link Length [m]**
1	0.88
2	2.48
3	2.23

Free-body diagrams of each of the positions are shown below in Figures 2, 3, and 4. It can be seen that no links cross over each other, meaning that this is a valid 2-D arm design (the solution was calculated and modelled before the announcement about crossing the y-axis was made, so the solution does include some orientations with the links crossing below the y-axis):

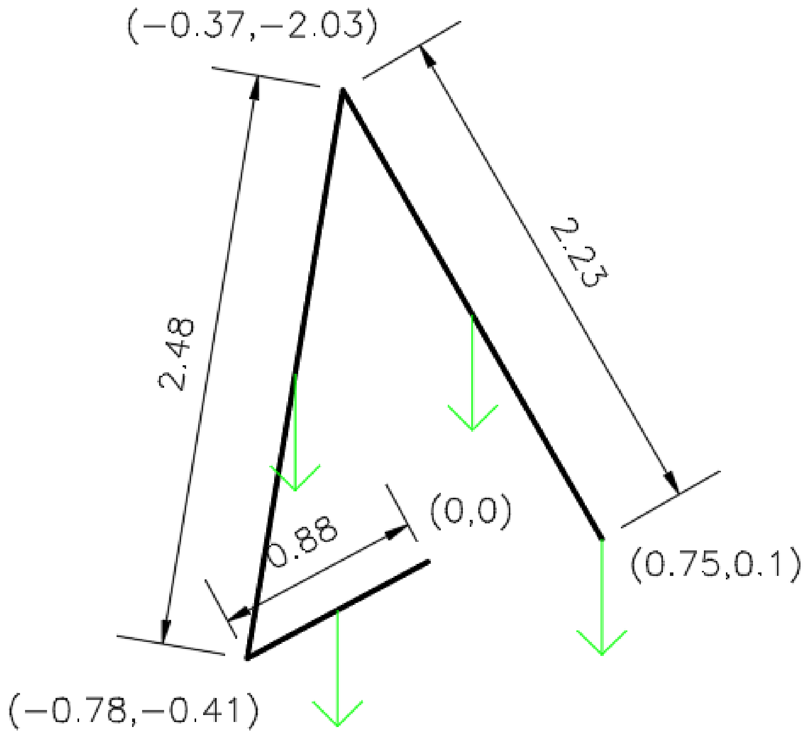


Figure 2: Free body diagram showing the final arm design in position 1.

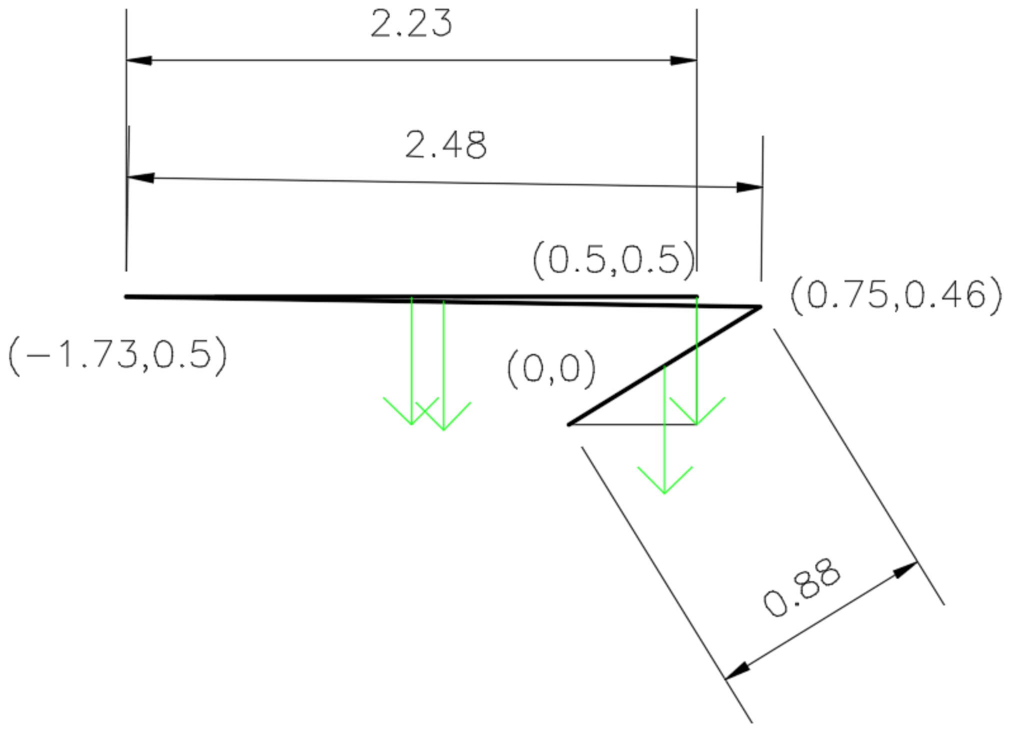


Figure 3: Free body diagram showing the final arm design in position 2.

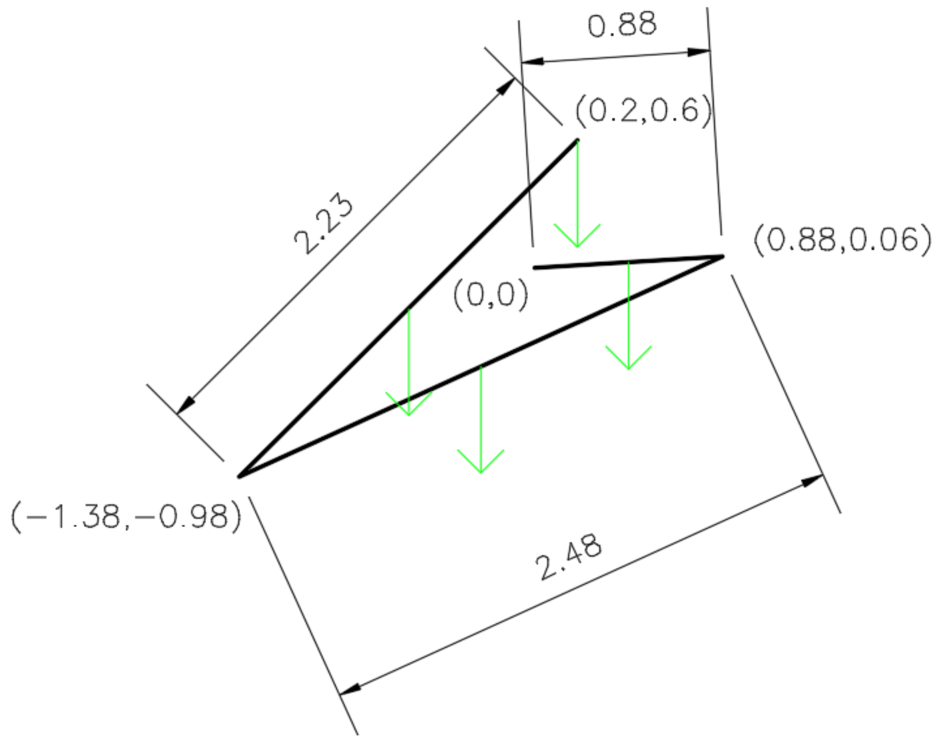


Figure 4: Free body diagram showing the final arm design in position 3.

From the program, the joint point locations were calculated and are displayed below in Table 3. Points A, B, and C represent the end points of Links 1, 2, and 3 respectively (C is the gripper location).

Table 3: Joint coordinates for each position, generated by the program.

Position Number	**Joint**	**Joint Coordinates [m]**
1	A	(-0.776311, -0.414417)
	B	(-0.365, 2.03124)
	C	(0.75, 0.1)
2	A	(0.749691, 0.46083)
	B	(-1.73, 0.5)
	C	(0.5, 0.5)
3	A	(0.878276, 0.0550497)
	B	(-1.37685, -0.976848)
	C	(0.2, 0.6)

Torques were calculated for the arm to be held in equilibrium for each of the positions, which are shown in Table 4 below:

Table 4: Torques generated at each position to keep the final design in equilibrium.

Position Number	**Torque [Nm]**
1	0.171501
2	0.165184
3	0.0282218

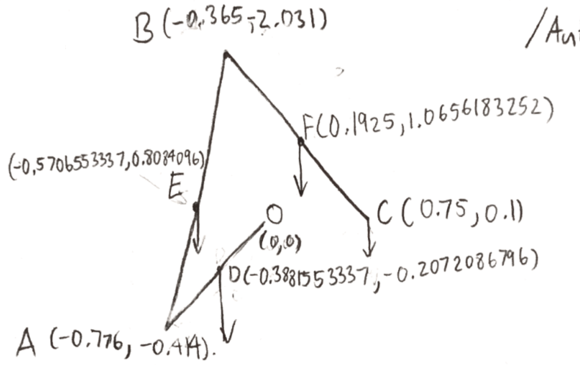
Which gives a final torque parameter of 0.23978Nm (according to Equation 3). Thus, this was the optimal solution as calculated by the program detailed above and is the final design of this report. Three other design iterations with varying max-lengths can be found in Appendix A.

Calculations

In order to verify the program's calculations, hand calculations were made to find the torques in each of the three positions, as shown in Figures 5 to 7 below:

Position 1:

(Point locations generated in SolidWorks /AutoCAD, which agreed with program calculations



$L_{OA} = 0.88\,m$

$L_{AB} = 2.48\,m$

$L_{BC} = 2.23\,m.$

For all 4 forces, as the weights act directly downwards, perpendicular to their x-distance, the torques generated can be calculated by:

$$\tau = P_x \cdot F_p$$

where: P_x = Horizontal distance from force point to origin / base. [m]

F_p = Force [N]

Force at D:

$F_D = 0.88 \times a \times 9.81$ (where a = geometry constant).

$F_D = 0.88 \times 4 \times 9.81$

$F_D = 34.5312$ N

Force at E:

$F_E = 2.48 \times 2 \times 9.81$

$F_E = 48.6576$ N

Force at F:

$F_F = 2.23 \times 1 \times 9.81$

$F_F = 21.8763$ N.

Force at C: $F_c = 5 \times 9.81$

$= 49.05$ N.

Since the link lengths are the same in all three positions, these forces will be the same at each point for the three positions.

As the links are uniform, the force of the weight of each link can be represented as a force acting in the centre of its respective length.

Figure 5: First page of hand calculations.

To find the total torque, the torques can be summed up:

$$\circlearrowright + \sum \tau = D_x \cdot F_D + E_x \cdot F_E + F_x + F_F + C_x \cdot F_C$$

((Clockwise is positive torque, since using the coordinates, $-x$ will give a negative torque value and CCW rotation about the origin).

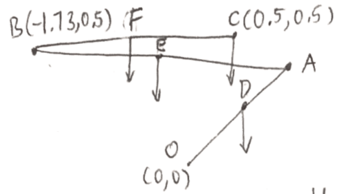
Thus, for position 1:

$$\circlearrowright + \sum \tau_1 = -0.3881553337 \cdot 34.5312 - 0.5706553337 \cdot 48.6576 + 0.1925 \cdot 21.8763 + 0.75 \cdot 49.05$$

$$\circlearrowright + \sum \tau_1 \approx -0.1715007 \text{ Nm}$$

$$\therefore \boxed{\tau_1 \approx 0.171501 \text{ N·m [CCW]}}$$ (Agrees with code).

Position 2:



Points found on SolidWorks

D(0.3748453212, 0.2304148112)

E(-0.4901546788, 0.4804148112)

F(-0.615, 0.50)

As the links are the same lengths, the forces are the same.

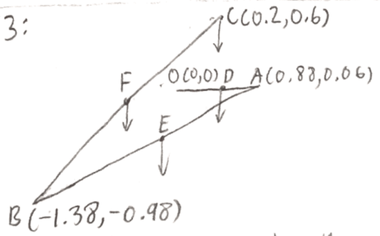
$$\therefore \circlearrowright \sum \tau = D_x \cdot F_D + E_x \cdot F_E + F_x \cdot F_F + C_x \cdot F_C$$

$$\circlearrowright \sum \tau_2 = 0.3748453212 \cdot 34.5312 - 0.4901546788 \cdot 48.6576 - 0.615 \cdot 21.8763 + 0.5 \cdot 49.05$$

$$\approx 0.16518395 \text{ Nm}$$

$$\boxed{\therefore \tau_2 \approx 0.165184 \text{ Nm [CW]}}$$ (agrees with code)

Position 3:



Points found on SolidWorks

D(0.4391382267, 0.0275248582)

E(-0.2492858343, -0.4608492028)

F(-0.588424061, -0.188424061)

The links have the same lengths as positions 1 and 2, so the forces are the same.

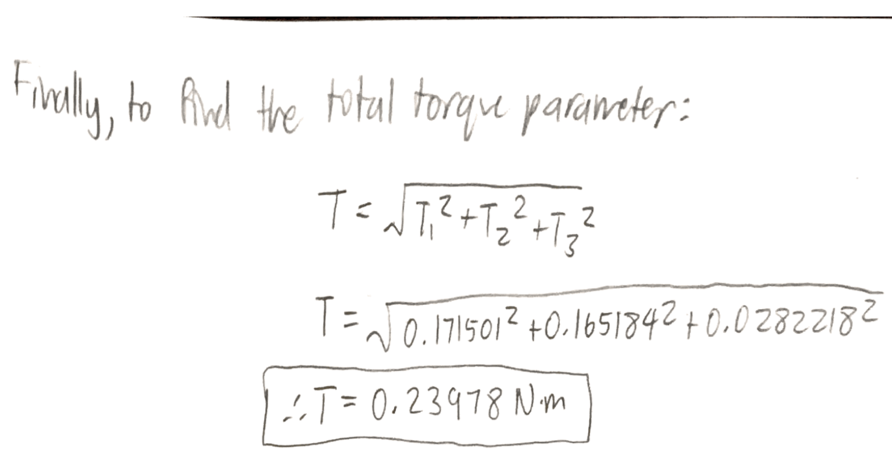
Thus, $\circlearrowright \sum \tau = D_x F_D + E_x F_E + F_x F_F + C_x F_C$

$$\circlearrowright \sum \tau_3 = 0.4391382267 \cdot 34.5312 - 0.2492858343 \cdot 48.6576 - 0.588424061 \cdot 21.8763 + 0.2 \cdot 49.05$$

$$= -0.028221763 \text{ Nm}$$

$$\boxed{\therefore \tau_3 \approx 0.0282218 \text{ Nm [CCW]}}$$ (agrees with code)

Figure 6: Second page of hand calculations.


Finally, to find the total torque parameter:

$$T = \sqrt{T_1^2 + T_2^2 + T_3^2}$$

$$T = \sqrt{0.171501^2 + 0.165184^2 + 0.028218^2}$$

$$\boxed{\therefore T = 0.23978\ N \cdot m}$$

Figure 7: Third and final page of hand calculations.

Summary/Conclusion

After creating a program that would help calculate the torque given the 3 link lengths, 3 unique designs were created by looping through the different possible link lengths. It was determined that the minimum torque parameter generated was 0.23978Nm with link lengths of 0.88m, 2.48m and 2.23m for links 1, 2 and 3, respectively. This was arrived at by looping through the possible maximum lengths and selecting the one arrangement that minimized the torque generated. Then, it was verified to be a plausible solution by creating the free body diagrams and drawings of the design in SolidWorks and AutoCAD. Hand calculations for the total torque were performed and matched the program's expected torque values which confirms that this design minimizes the total torque. Thus, Figures 2, 3, and 4 show the final and most optimal solution for the project at hand.

The other 3 design iterations created are shown in Appendix A. The first alternative design is optimized to have the lowest torque possible while using the least amount of material possible with a total length of 1m. With a total length of 1m, the torque parameter is equal to 50.251Nm. This design has link lengths of 0.24, 0.59 and 0.17m for links 1, 2 and 3, respectively. The second alternative design has a total length of 3.3m (sum of the 3 link lengths), which is in the middle of the minimum length, and the optimized torque length. It has a torque parameter of 19.310Nm with link lengths 0.98, 1.32 and 1m for links 1, 2 and 3, respectively. The third alternative design has a total length of 6m. It has a torque parameter of 5.625Nm with link lengths 0.96, 2.72 and 2.32m for links 1, 2 and 3, respectively. Appendix B displays the program code created in C++ to figure out the lowest torque possible with the given specifications.

Contributions

Keith Lai: Overview and conclusion

Tiger Ye: FDB Diagrams, Code to find Torque

Eric Zhao: Code to find intersections, hand calculations

Appendix A – Design Iterations

Figures 8 to 16 show three design iterations in the 3 different positions, each with a different max length, which were generated using the program.

Design #1, Least Cost and Material (Total Length = 1m, Torque Parameter = 50.2508Nm):

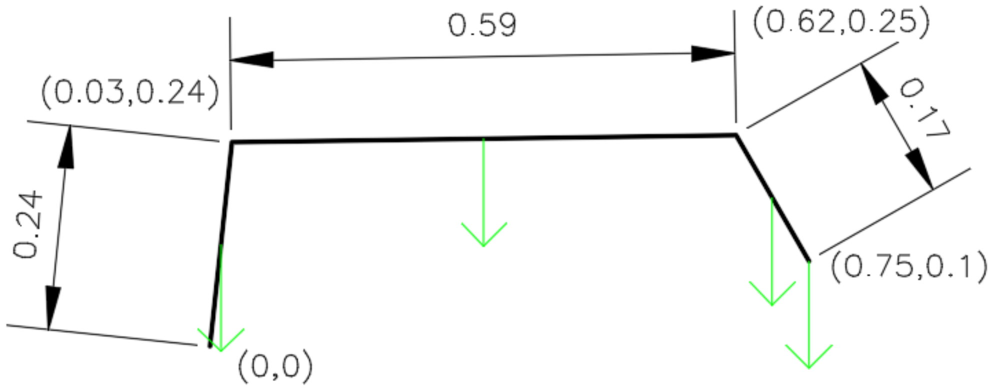


Figure 8: Free body diagram showing the lowest cost design in position 1. The torque generated in this position is 42.6069Nm.

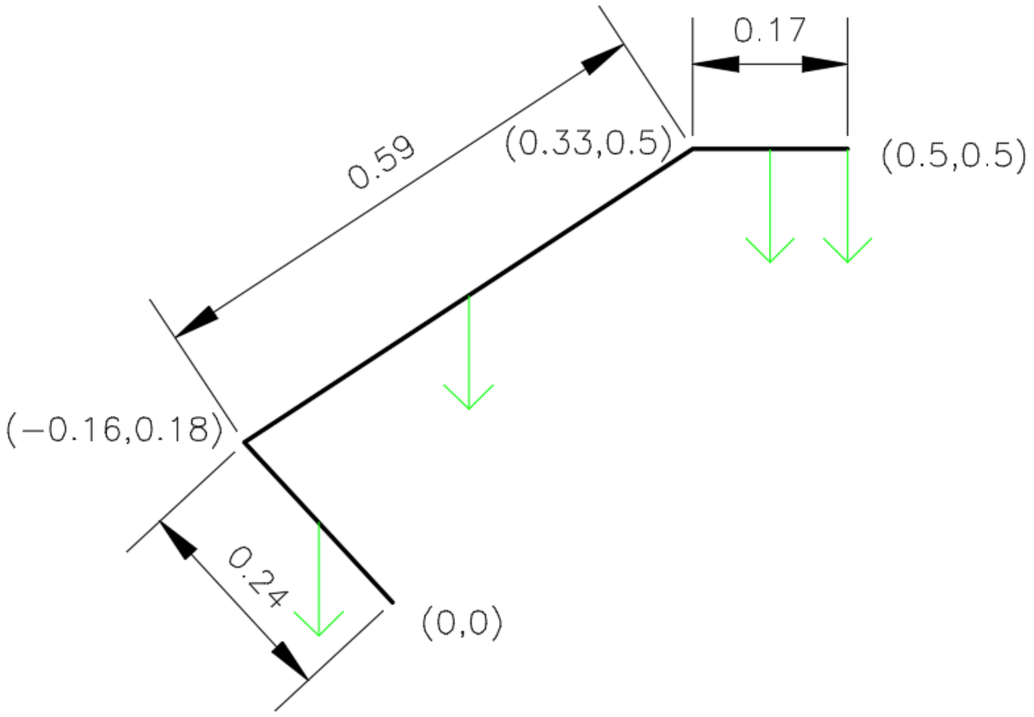


Figure 9: Free body diagram showing the lowest cost design in position 2. The torque generated in this position is 25.415Nm.

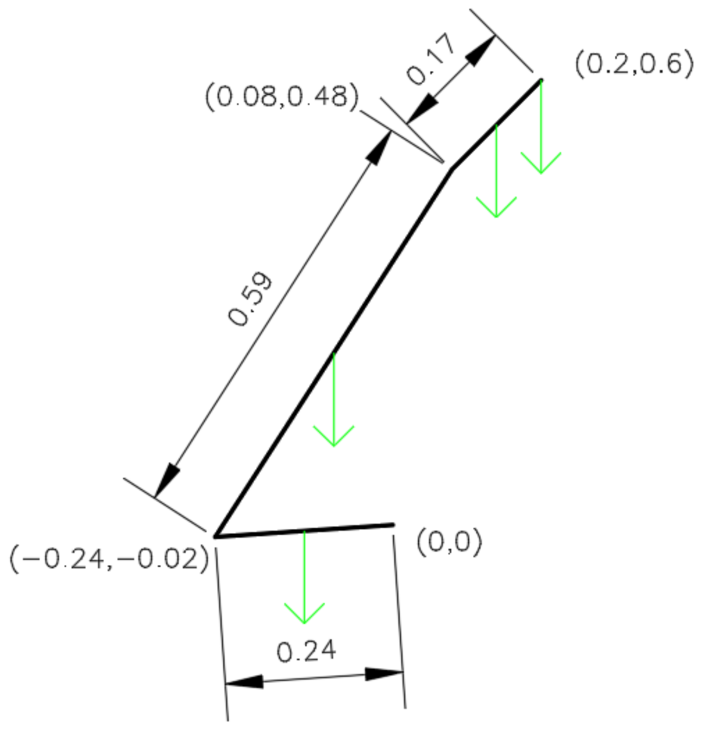


Figure 10: Free body diagram showing the lowest cost design in position 3. The torque generated in this position is 7.9918Nm.

Design #2, Balance Between Length and Torque (Total Length = 3.3m, Torque Parameter = 19.3103Nm):

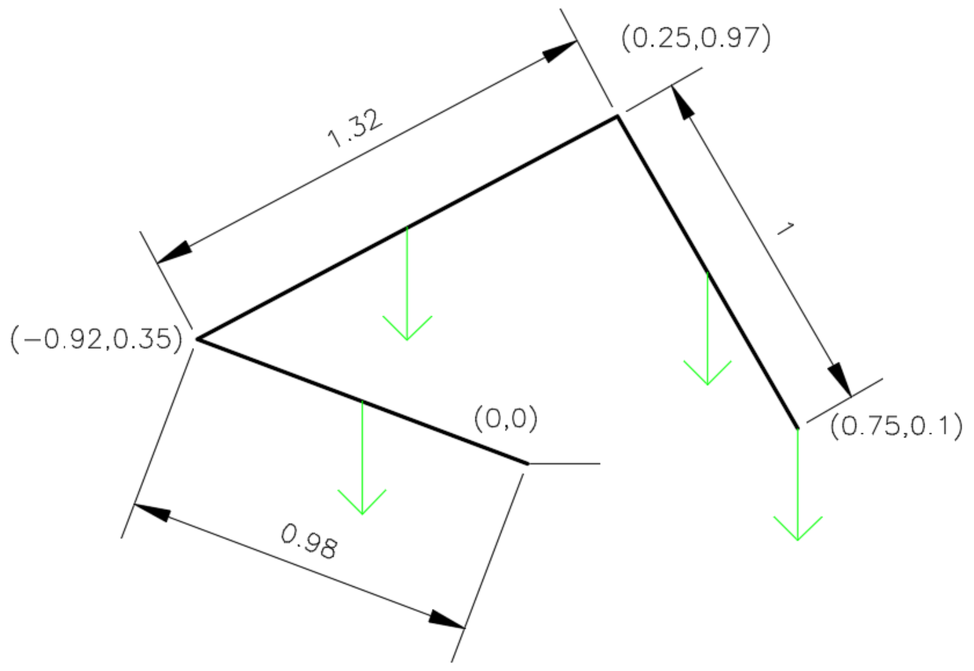


Figure 11: Free body diagram showing design #2 in position 1. The torque generated in this position is 14.7904Nm.

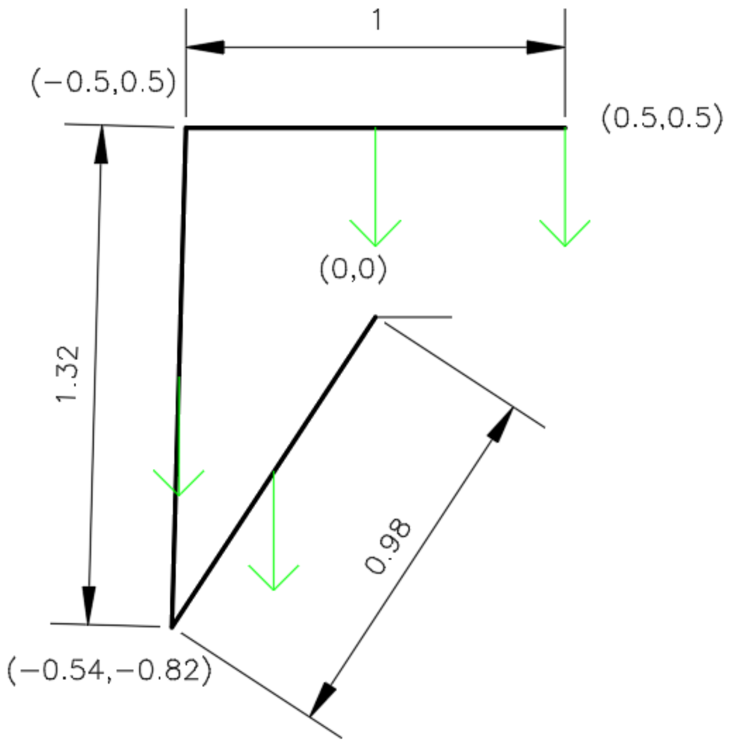


Figure 12: Free body diagram showing design #2 in position 2. The torque generated in this position is 0.403009Nm.

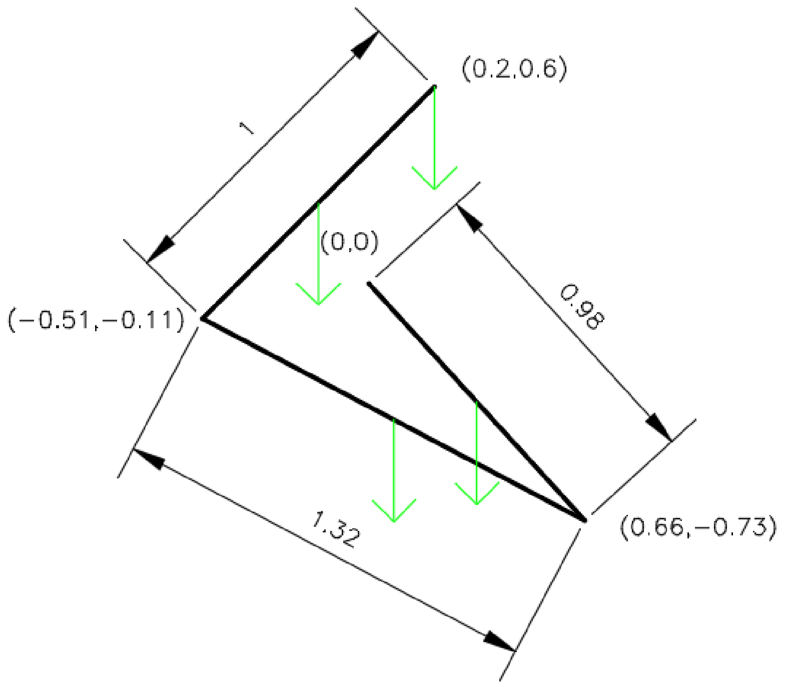


Figure 13: Free body diagram showing design #2 in position 3. The torque generated in this position is 12.4084Nm.

Design #3 (Total Length = 6m, Torque Parameter = 5.62458Nm):

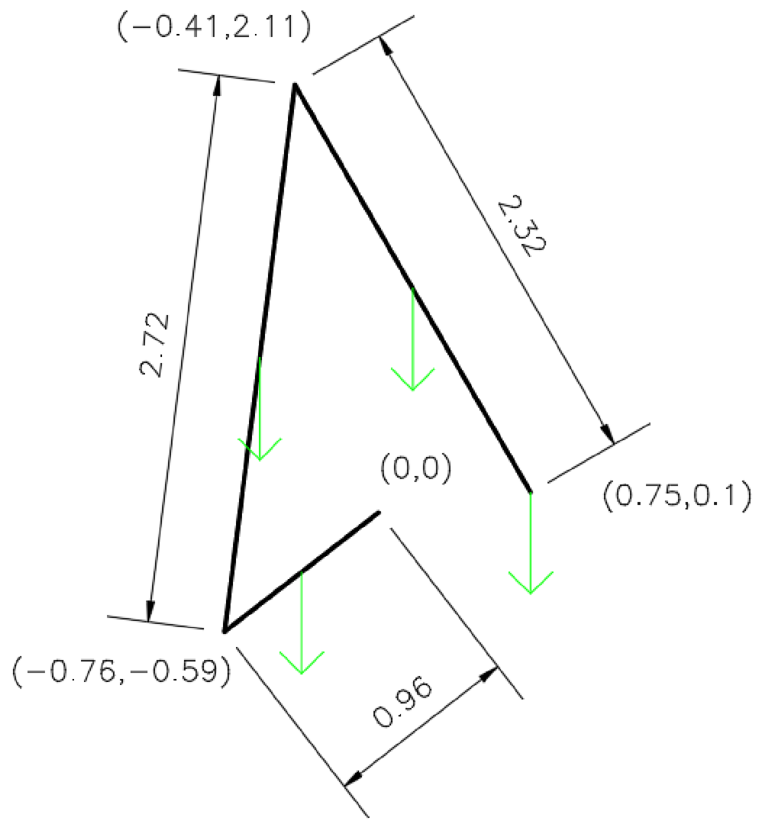


Figure 14: Free body diagram showing design #3 in position 1. The torque generated in this position is 4.81125Nm.

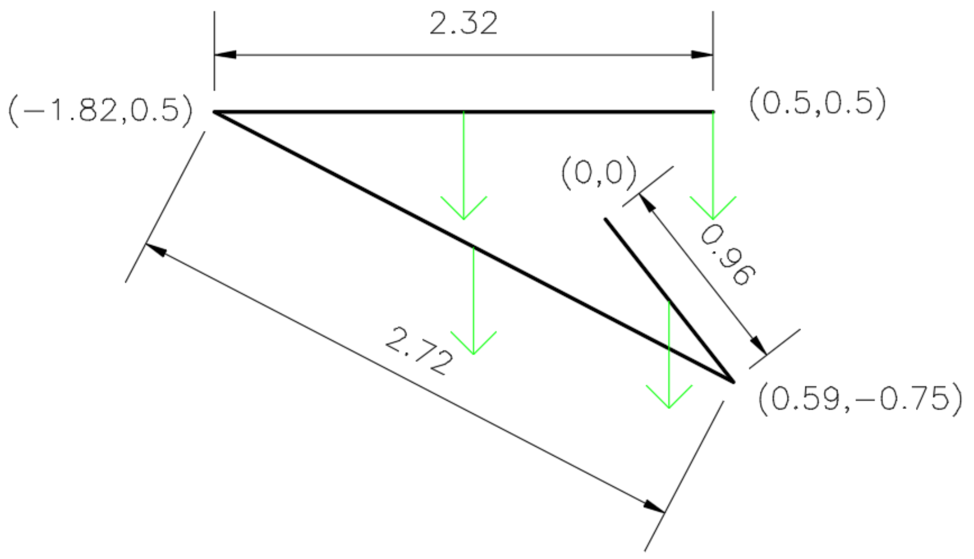


Figure 15: Free body diagram showing design #3 in position 2. The torque generated in this position is 1.70755Nm.

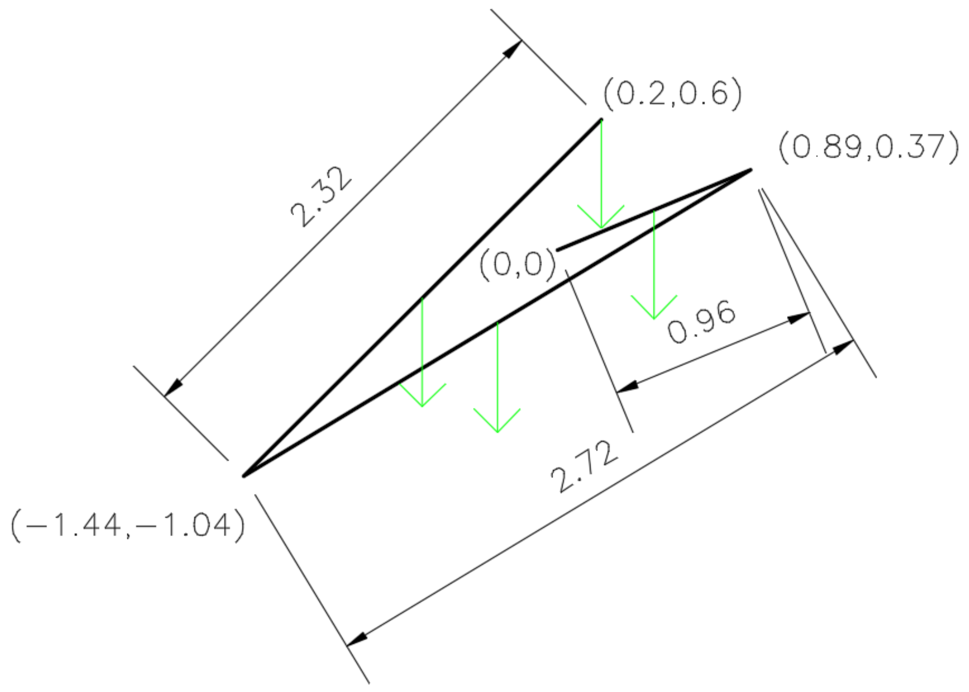


Figure 16: Free body diagram showing design #3 in position 3. The torque generated in this position is 2.36052Nm.

Appendix B – Program Code

The program coded in C++ is shown below:

```
#include <iostream>
#include <cmath>

using namespace std;

const double WL = 5;
const double GRAVITY = 9.81;
const double MAX_LENGTH = 500;

class Point {
private:

    double x;
    double y;

public:

    Point(){
        x = 0;
        y = 0;
    }

    Point(double x0, double y0){
        x = x0;
        y = y0;
    }

    double getx()const{
        return x;
    }

    double gety()const{
        return y;
    }

    double distOrigin()const{
        return sqrt(pow(x,2) + pow(y,2));
    }

    double angleOrigin()const{
        return atan2(y,x);
    }
    void print(){
        cout << "(" << x << "," <<y <<")" << endl;
    }
};

class Link {
private:

    Point start;
    Point end;
    double length;
    double width;

public:

    Link(Point start0, Point end0, double width0){
        start = start0;
        end = end0;
        length = sqrt(pow(end.getx() - start.getx(),2) + pow(end.gety() - start.gety(),2));
        width = width0;
    };

    Point getStart()const{
        return start;
    }

    Point getEnd()const{
        return end;
    }

    double getLength()const{
        return length;
    }
    //radians wrt x axis
    double getAngle()const{
        return atan2(end.gety() - start.gety(),end.getx() - start.getx());
    }

    double xDist()const{
        return end.getx() - start.getx();
    }

    double getWeight()const{
        return length*width*GRAVITY;
    }
};

double getTorque(Link links[3]){
    return links[0].xDist()*links[0].getWeight()/2
           + (links[0].xDist() + links[1].xDist()/2)*links[1].getWeight()
           + (links[0].xDist() + links[1].xDist() + links[2].xDist()/2)*links[2].getWeight()
           + (links[0].xDist() + links[1].xDist() + links[2].xDist())*WL*GRAVITY;
}

bool onSegment(Point p, Point q, Point r)
{
    if (q.getx() <= max(p.getx(), r.getx()) && q.getx() >= min(p.getx(), r.getx()) &&
        q.gety() <= max(p.gety(), r.gety()) && q.gety() >= min(p.gety(), r.gety()))
       return true;

    return false;
}


int orientation(Point p, Point q, Point r)
{
    double val = (q.gety() - p.gety()) * (r.getx() - q.getx()) -
              (q.getx() - p.getx()) * (r.gety() - q.gety());

    if (val == 0) return 0;

    return (val > 0)? 1: 2;
}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false;
}

double firstPosition(double l1, double l2, double l3){
    Point end = Point(0.75,0.10);
    Point p2 = Point(0.75 - l3*cos(M_PI/3), 0.10 + l3*sin(M_PI/3));
    if(l1 + l2 < p2.distOrigin())
        return 1000;

    double angle1 = acos((pow(l2,2) - pow(l1,2) - pow(p2.distOrigin(),2))/(-2*l1*p2.distOrigin()));

    Point p11 = Point(l1*cos(angle1+p2.angleOrigin()),l1*sin(angle1+p2.angleOrigin()));
    Point p12 = Point(l1*cos(p2.angleOrigin()-angle1),l1*sin(p2.angleOrigin()-angle1));

    Link links1[] = {Link(Point(0,0),p11,4), Link(p11,p2,2), Link(p2,end,1) };
    Link links2[] = {Link(Point(0,0),p12,4), Link(p12,p2,2), Link(p2,end,1) };

    double torque1 = getTorque(links1);
    double torque2 = getTorque(links2);

    if(doIntersect(links1[0].getStart(),links1[0].getEnd(),links1[2].getStart(),links1[2].getEnd()))
        torque1 = 1000;

    if(doIntersect(links2[0].getStart(),links2[0].getEnd(),links2[2].getStart(),links2[2].getEnd()))
        torque2 = 1000;

    return min(fabs(torque1),fabs(torque2));
}

double secondPosition(double l1, double l2, double l3){
    Point end = Point(0.50,0.50);
    Point p2 = Point(0.50 - l3, 0.50);

    if(l1 + l2 < p2.distOrigin())
        return 1000;

    double angle1 = acos((pow(l2,2) - pow(l1,2) - pow(p2.distOrigin(),2))/(-2*l1*p2.distOrigin()));

    Point p11 = Point(l1*cos(angle1+p2.angleOrigin()),l1*sin(angle1+p2.angleOrigin()));
    Point p12 = Point(l1*cos(p2.angleOrigin()-angle1),l1*sin(p2.angleOrigin()-angle1));

    Link links1[] = {Link(Point(0,0),p11,4), Link(p11,p2,2), Link(p2,end,1) };
    Link links2[] = {Link(Point(0,0),p12,4), Link(p12,p2,2), Link(p2,end,1) };

    double torque1 = getTorque(links1);
    double torque2 = getTorque(links2);

    if(doIntersect(links1[0].getStart(),links1[0].getEnd(),links1[2].getStart(),links1[2].getEnd()))
        torque1 = 1000;

    if(doIntersect(links2[0].getStart(),links2[0].getEnd(),links2[2].getStart(),links2[2].getEnd()))
        torque2 = 1000;

    return min(fabs(torque1),fabs(torque2));
}

double thirdPosition(double l1, double l2, double l3){
    Point end = Point(0.20,0.60);
    Point p2 = Point(0.20 - l3*cos(M_PI/4), 0.60 - l3*sin(M_PI/4));
    if(l1 + l2 < p2.distOrigin())
        return 1000;

    double angle1 = acos((pow(l2,2) - pow(l1,2) - pow(p2.distOrigin(),2))/(-2*l1*p2.distOrigin()));

    Point p11 = Point(l1*cos(angle1+p2.angleOrigin()),l1*sin(angle1+p2.angleOrigin()));
    Point p12 = Point(l1*cos(p2.angleOrigin()-angle1),l1*sin(p2.angleOrigin()-angle1));

    Link links1[] = {Link(Point(0,0),p11,4), Link(p11,p2,2), Link(p2,end,1) };
    Link links2[] = {Link(Point(0,0),p12,4), Link(p12,p2,2), Link(p2,end,1) };

    double torque1 = getTorque(links1);
    double torque2 = getTorque(links2);

    if(doIntersect(links1[0].getStart(),links1[0].getEnd(),links1[2].getStart(),links1[2].getEnd()))
        torque1 = 1000;

    if(doIntersect(links2[0].getStart(),links2[0].getEnd(),links2[2].getStart(),links2[2].getEnd()))
        torque2 = 1000;

    return min(fabs(torque1),fabs(torque2));
}

double getTotalTorque(double t1, double t2, double t3)
{
    return sqrt(pow(t1,2) + pow(t2,2) + pow(t3,2));
}
```

```
double combinations(){
    double least = 1000;
    for(int max_length = 330; max_length < 331; max_length++)
    {
        for(int i = 1; i < max_length - 2; i++)
        {
            for(int k = 1; k < max_length - i - 1; k++)
            {
                int j = max_length - k - i;
                double j1 = j/100.0;
                double i1 = i/100.0;
                double k1 = k/100.0;
                double total_torque = getTotalTorque(firstPosition(j1,k1,i1), secondPosition(j1,k1,i1), thirdPosition(
j1,k1,i1));
                if(total_torque < least && total_torque > 0)
                {
                    cout << j << " " << k << " " << i << endl;
                    least = total_torque;
                }
            }
        }
    }
    return least;
}

int main() {

    cout << combinations() << endl;
    return 0;
}
```