FERMILAB-20-087-AD

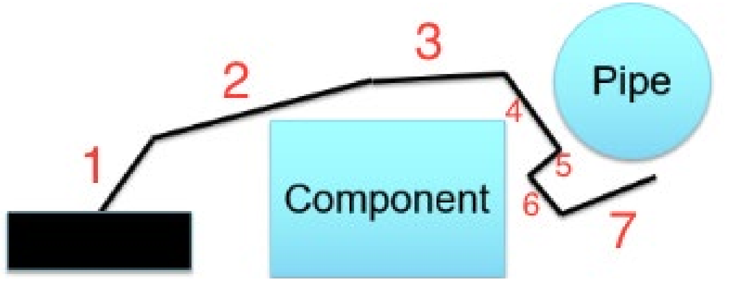
Long Reach Robotic Arm

Amanda Hoeksema & Brenda Sanchez
Under the mentorship of Kris Anderson
AD Robotics Initiative - CCI

5 August 2020

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Long Reach Robotic Arm



Problem

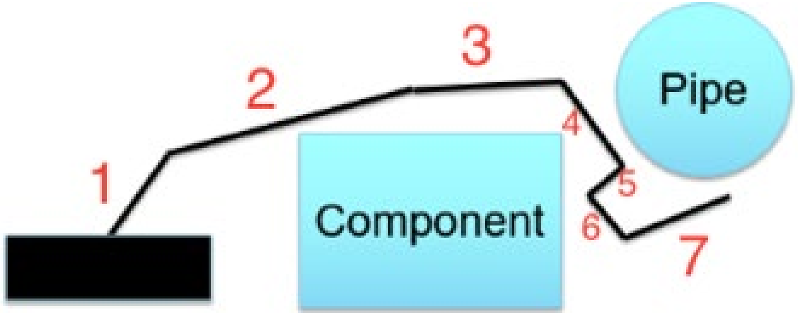
- Quadrupole magnets frequent failure
  - Water leaks
- Critical systems around the magnets that need access
  - Tight space of components make it difficult to view

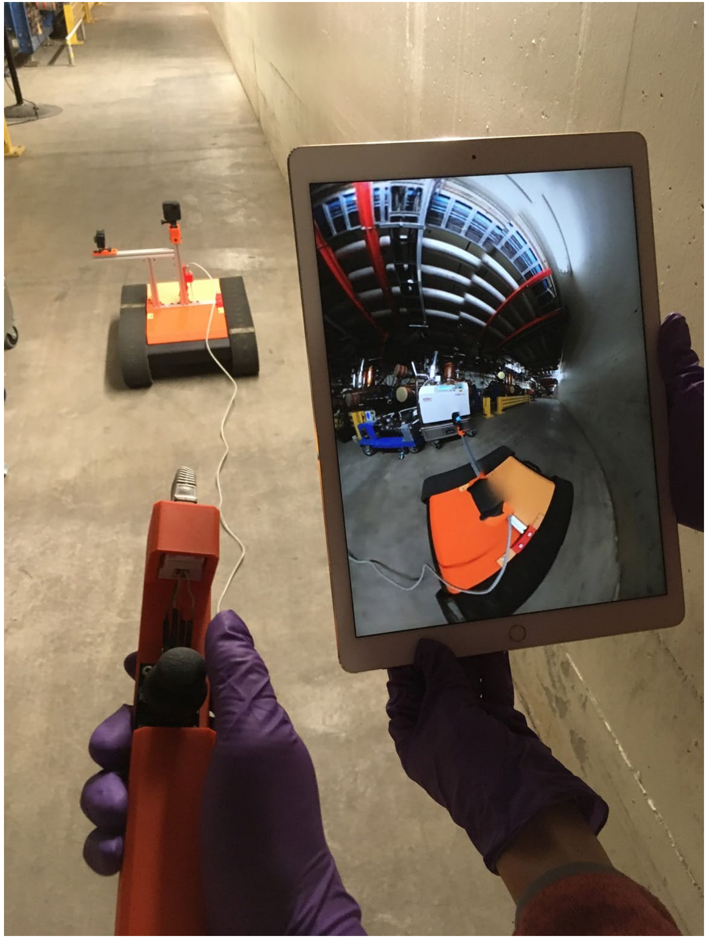
Solution

- Long robotic arm with many joints to extend and reach around components
- Camera attached at the arm’s end

Long Reach Robotic Arm Design - Split in Two

- **The Arm**
  - Control certain links of the arm
  - Move direction X,Y, & Z while avoiding components
    - “Snake” around obstructions
  - Compact design for mobility
- **The Base/Counterweight**
  - Needs to support the extended long arm
  - Large enough to provide stability but not consume too much space in small tunnel





Fermilab

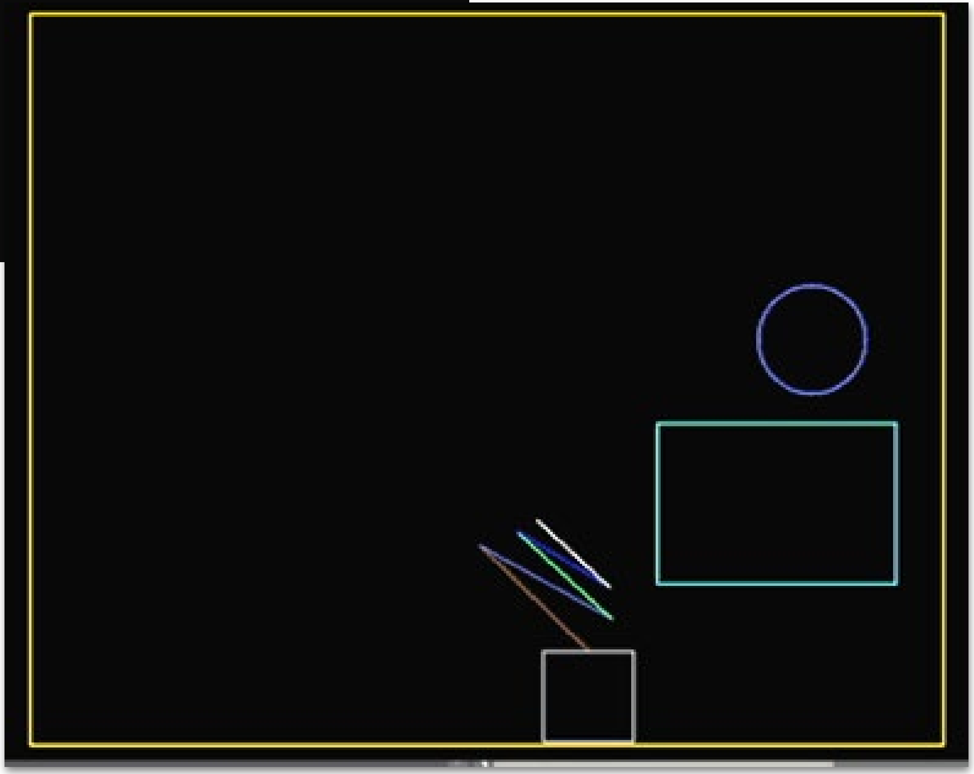
Coding

-

Program For Robot

Code for Robot

```
Enter distance between the center of the base and the wall (in inches):46
Enter number of arm links (1-6):5
Enter length of Link One (in inches):28
Enter length of Link Two (in inches):28
Enter length of Link Three (in inches)24
Enter length of Link Four (in inches):20
Enter length of Link Five (in inches):18
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <SDL.h>
#include <math.h>
```

Libraries used

Scaling

Scales window height & width to pixels

```
/// Robot Base, Tunnel, Magnet, and Pipe + Scaling
int border = 5;                                              // Window Border
int pixelwidth = windoww - (2 * border);
int pixelheight = windowh - (2 * border);

int basewidth = 12;                                          //Base width (inches)
int baseheight = 12;                                         //Base height (inches)

int tunnelwidth = 120;                                       // Tunnel Width (Inches)
int tunnelheight = 96;                                       // Tunnel Height (Inches)

int tunnelwpixels = 0;                                       // Tunnel Width (Pixels)
int tunnelhpixels = 0;                                       // Tunnel Height (Pixels)

int scaleheight = pixelheight / tunnelheight;                // Scaling Tunnel Height
int scalewidth = pixelwidth / tunnelwidth;                   // Scaling Tunnel Width

float scale = scaleheight;
if (scalewidth < scaleheight) {
    scale = scalewidth;
}

int basewidthpixels = (int)(basewidth * scale);              // Base Inches to Pixels
int baseheightpixels = (int)(baseheight * scale);            //Base  Inches to Pixels

int base_xcenter;                                            // Base- Inches Off the Wall

float base_ycenter = 13;                                     // Base- Inches Off the Ground
int base_ycenterpixels = (int)(base_ycenter * scale);        // Inches to Pixels

tunnelhpixels = (int)(tunnelheight * scale);                 // Inches to Pixels
tunnelwpixels = (int)(tunnelwidth * scale);
```

Scaling

```
/// Robot Base, Tunnel, Magnet, and Pipe + Scaling
int border = 5;                                          // Window Border
int pixelwidth = windoww - (2 * border);
int pixelheight = windowh - (2 * border);

int basewidth = 12;                                       //Base width (inches)
int baseheight = 12;                                      //Base height (inches)

int tunnelwidth = 120;                                    // Tunnel Width (Inches)
int tunnelheight = 96;                                    // Tunnel Height (Inches)

int tunnelwpixels = 0;                                    // Tunnel Width (Pixels)
int tunnelhpixels = 0;                                    // Tunnel Height (Pixels)

int scaleheight = pixelheight / tunnelheight;             // Scaling Tunnel Height
int scalewidth = pixelwidth / tunnelwidth;                // Scaling Tunnel Width

float scale = scaleheight;
if (scalewidth < scaleheight) {
    scale = scalewidth;
}

int basewidthpixels = (int)(basewidth * scale);           // Base Inches to Pixels
int baseheightpixels = (int)(baseheight * scale);         //Base  Inches to Pixels

int base_xcenter;                                         // Base- Inches Off the Wall

float base_ycenter = 13;                                  // Base- Inches Off the Ground
int base_ycenterpixels = (int)(base_ycenter * scale);     // Inches to Pixels

tunnelhpixels = (int)(tunnelheight * scale);              // Inches to Pixels
tunnelwpixels = (int)(tunnelwidth * scale);
```

Tunnel width and height (Inches)

Setting variable to zero

Scaling & converting inches to pixels

Fermilab

Scaling

```
/// Robot Base, Tunnel, Magnet, and Pipe + Scaling
int border = 5;                                          // Window Border
int pixelwidth = windoww - (2 * border);
int pixelheight = windowh - (2 * border);

int basewidth = 12;                                       //Base width (inches)
int baseheight = 12;                                      //Base height (inches)

int tunnelwidth = 120;                                    // Tunnel Width (Inches)
int tunnelheight = 96;                                    // Tunnel Height (Inches)

int tunnelwpixels = 0;                                    // Tunnel Width (Pixels)
int tunnelhpixels = 0;                                    // Tunnel Height (Pixels)

int scaleheight = pixelheight / tunnelheight;             // Scaling Tunnel Height
int scalewidth = pixelwidth / tunnelwidth;                // Scaling Tunnel Width

float scale = scaleheight;
if (scalewidth < scaleheight) {
    scale = scalewidth;
}

int basewidthpixels = (int)(basewidth * scale);          // Base Inches to Pixels
int baseheightpixels = (int)(baseheight * scale);        //Base  Inches to Pixels

int base_xcenter;                                         // Base- Inches Off the Wall

float base_ycenter = 13;                                  // Base- Inches Off the Ground
int base_ycenterpixels = (int)(base_ycenter * scale);    // Inches to Pixels

tunnelhpixels = (int)(tunnelheight * scale);              // Inches to Pixels
tunnelwpixels = (int)(tunnelwidth * scale);
```

Scaling & converting inches to pixels

Center of Base

Key Events

```
while (quit == 0)
{
    while (SDL_PollEvent(&event) != 0) {
        if (event.type == SDL_QUIT) {
            quit = 1;
        }
        if (event.type == SDL_KEYDOWN) {
            if (event.key.keysym.sym == SDLK_q) {
                rotate = -rotationspeed;
            }
            if (event.key.keysym.sym == SDLK_a) {
                rotate = rotationspeed;
            }
            if (event.key.keysym.sym == SDLK_w) {
                rotate2 = -rotationspeed;
            }
            if (event.key.keysym.sym == SDLK_s) {
                rotate2 = rotationspeed;
```

Moves arm link

```
        if (event.type == SDL_KEYUP) {
            if (event.key.keysym.sym == SDLK_q) {
                rotate = 0.0;
            }
            if (event.key.keysym.sym == SDLK_a) {
                rotate = 0.0;
            }
            if (event.key.keysym.sym == SDLK_w) {
                rotate2 = 0.0;
            }
            if (event.key.keysym.sym == SDLK_s) {
                rotate2 = 0.0;
            }
```

Stops movement

Link Info

```
if (linktotal == 2) {
    //LINE ONE INFO
    int lengthpixels = (int)(length * scale);
    //Centers Line One in the window
    LineLX = ((windoww)-(tunnelwpixels)+tunnelwpixels - base_xcenterpixels - (basewidthpixels / 2)) + (basewidthpixels / 2);
    LineLY = (windoww)-(tunnelwpixels)+tunnelhpixels - base_ycenterpixels - (baseheightpixels / 2);

    //Line One direction
    LineA = LineA + rotate;

    //Locations of Line One corners
    LineRX = LineLX + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0))) - ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0)));
    LineRY = LineLY + ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0))) + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0)));

    //Draw Line One
    SDL_SetRenderDrawColor(renderer, 220, 20, 60, 255);
    SDL_RenderDrawLine(renderer, LineLX, LineLY, LineRX, LineRY);


    //LINE TWO INFO
    int length2pixels = (int)(length2 * scale);
    //Line Two starting location
    LineLX2 = LineRX;
    LineLY2 = LineRY;

    //Line Two direction
    if (rotate == 0.0) {
        LineA2 = LineA2 + rotate2;
    }
    else {
        LineA2 = LineA2 + rotate;
    }

    //Locations of Line Two corners
    LineRX2 = LineLX2 + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0))) - ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0)));
    LineRY2 = LineLY2 + ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0))) + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0)));

    //Draw Line Two
    SDL_SetRenderDrawColor(renderer, 90, 79, 207, 255);
    SDL_RenderDrawLine(renderer, LineLX2, LineLY2, LineRX2, LineRY2);
```

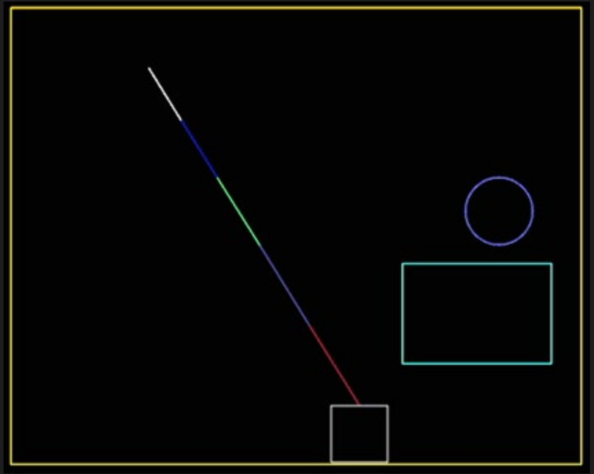
Scales link

```
Enter distance between the center of the base and the wall (in inches):46
Enter number of arm links (1-6):5
Enter length of Link One (in inches):28
Enter length of Link Two (in inches):28
Enter length of Link Three (in inches)24
Enter length of Link Four (in inches):20
Enter length of Link Five (in inches):18
```

Link Info

```
if (linktotal == 2) {
    //LINE ONE INFO
    int lengthpixels = (int)(length * scale);
    //Centers Line One in the window
    LineLX = ((windoww)-(tunnelwpixels)+tunnelwpixels - base_xcenterpixels - (basewidthpixels / 2)) + (basewidthpixels / 2);
    LineLY = (windoww)-(tunnelwpixels)+tunnelhpixels - base_ycenterpixels - (baseheightpixels / 2);

    //Line One direction
    LineA = LineA + rotate;

    //Locations of Line One corners
    LineRX = LineLX + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0))) - ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0)));
    LineRY = LineLY + ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0))) + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0)));

    //Draw Line One
    SDL_SetRenderDrawColor(renderer, 220, 20, 60, 255);
    SDL_RenderDrawLine(renderer, LineLX, LineLY, LineRX, LineRY);


    //LINE TWO INFO
    int length2pixels = (int)(length2 * scale);
    //Line Two starting location
    LineLX2 = LineRX;
    LineLY2 = LineRY;

    //Line Two direction
    if (rotate == 0.0) {
        LineA2 = LineA2 + rotate2;
    }
    else {
        LineA2 = LineA2 + rotate;
    }

    //Locations of Line Two corners
    LineRX2 = LineLX2 + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0))) - ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0)));
    LineRY2 = LineLY2 + ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0))) + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0)));

    //Draw Line Two
    SDL_SetRenderDrawColor(renderer, 90, 79, 207, 255);
    SDL_RenderDrawLine(renderer, LineLX2, LineLY2, LineRX2, LineRY2);
```

Sets line angle

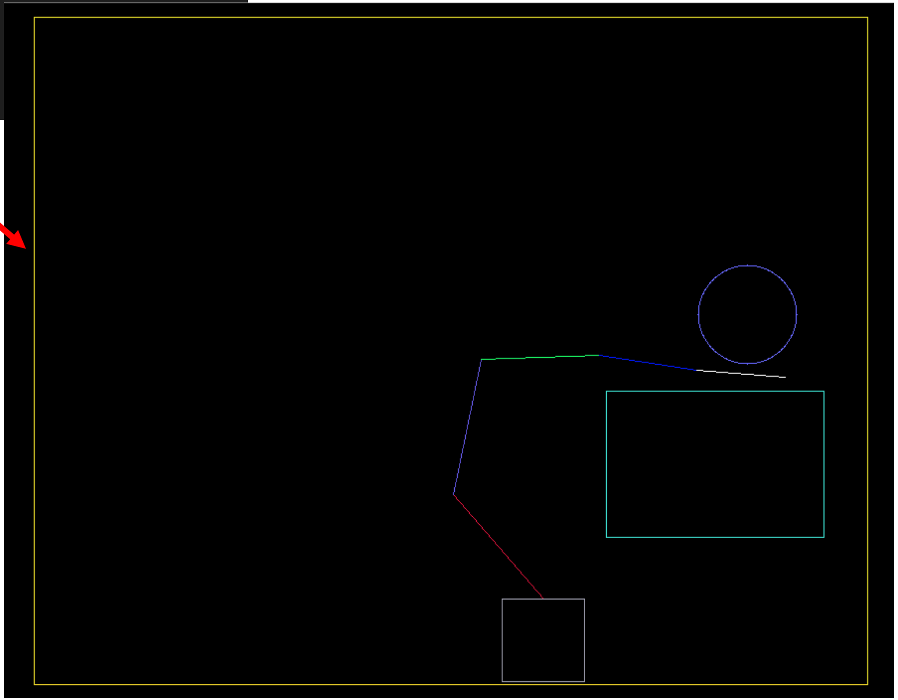


Link Info

```
if (linktotal == 2) {
    //LINE ONE INFO
    int lengthpixels = (int)(length * scale);
    //Centers Line One in the window
    LineLX = ((windoww)-(tunnelwpixels)+tunnelwpixels - base_xcenterpixels - (basewidthpixels / 2)) + (basewidthpixels / 2);
    LineLY = (windoww)-(tunnelwpixels)+tunnelhpixels - base_ycenterpixels - (baseheightpixels / 2);

    //Line One direction
    LineA = LineA + rotate;

    //Locations of Line One corners
    LineRX = LineLX + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0))) - ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0)));
    LineRY = LineLY + ((lengthpixels / 2.0) * sin(LineA * (pi / 180.0))) + ((lengthpixels / 2.0) * cos(LineA * (pi / 180.0)));

    //Draw Line One
    SDL_SetRenderDrawColor(renderer, 220, 20, 60, 255);
    SDL_RenderDrawLine(renderer, LineLX, LineLY, LineRX, LineRY);


    //LINE TWO INFO
    int length2pixels = (int)(length2 * scale);
    //Line Two starting location
    LineLX2 = LineRX;
    LineLY2 = LineRY;

    //Line Two direction
    if (rotate == 0.0) {
        LineA2 = LineA2 + rotate2;
    }
    else {
        LineA2 = LineA2 + rotate;
    }

    //Locations of Line Two corners
    LineRX2 = LineLX2 + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0))) - ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0)));
    LineRY2 = LineLY2 + ((length2pixels / 2.0) * sin(LineA2 * (pi / 180.0))) + ((length2pixels / 2.0) * cos(LineA2 * (pi / 180.0)));

    //Draw Line Two
    SDL_SetRenderDrawColor(renderer, 90, 79, 207, 255);
    SDL_RenderDrawLine(renderer, LineLX2, LineLY2, LineRX2, LineRY2);
```

Sets line endpoint

```
// Render Tunnel
SDL_SetRenderDrawColor(renderer, 255, 225, 0, 255);
dstrect.x = (windoww / 2) - (tunnelwpixels / 2);
dstrect.y = (windowh / 2) - (tunnelhpixels / 2);
dstrect.w = tunnelwpixels;
dstrect.h = tunnelhpixels;
SDL_RenderDrawRect(renderer, &dstrect);
```

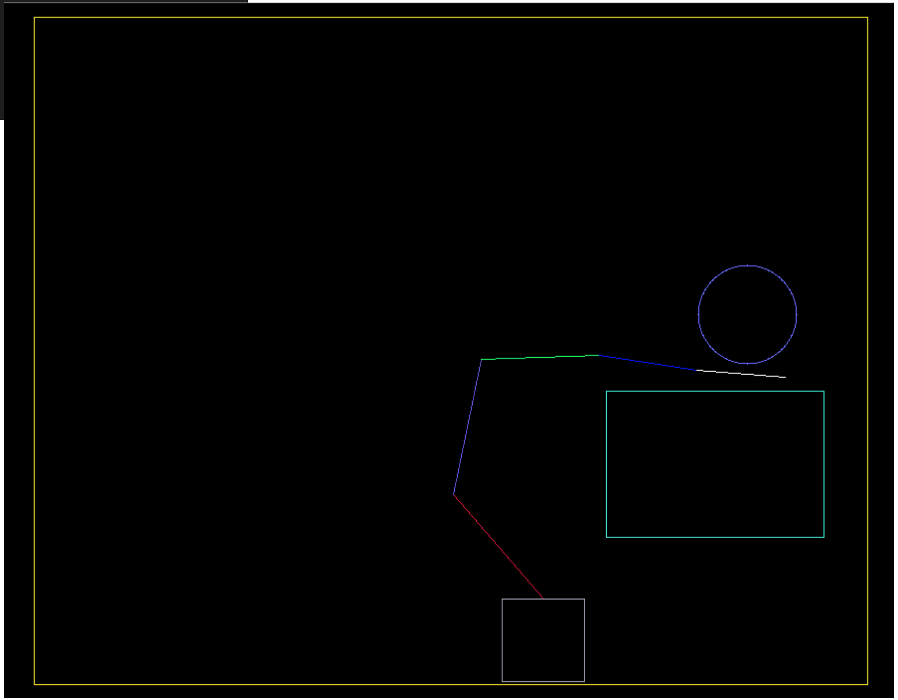


Rendering the Tunnel

```
// Render Tunnel
SDL_SetRenderDrawColor(renderer, 255, 225, 0, 255);
dstrect.x = (windoww / 2) - (tunnelwpixels / 2);
dstrect.y = (windowh / 2) - (tunnelhpixels / 2);
dstrect.w = tunnelwpixels;
dstrect.h = tunnelhpixels;
SDL_RenderDrawRect(renderer, &dstrect);
```

R,G,B, Opacity

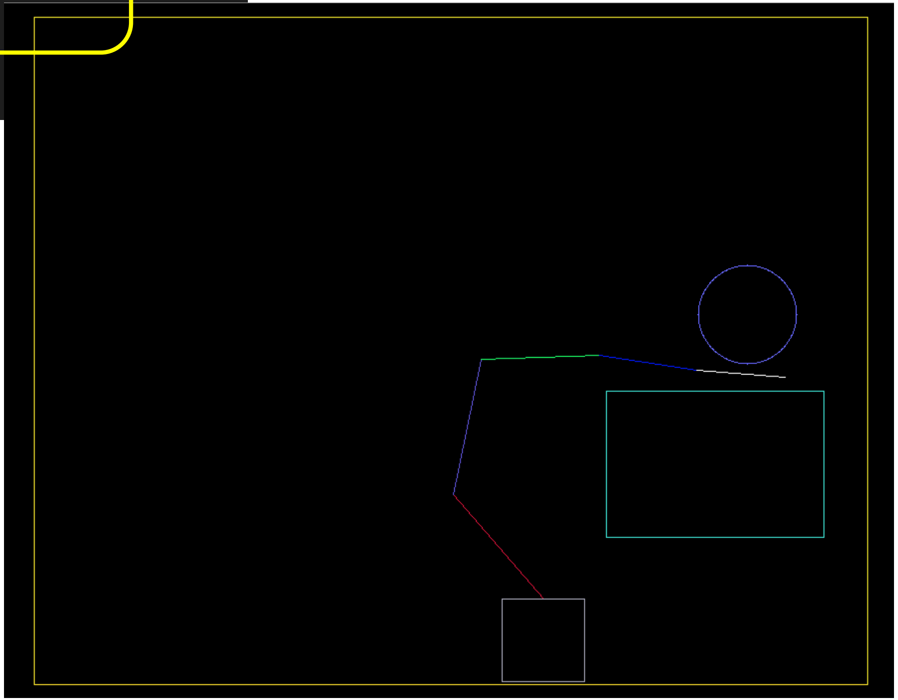
Rendering the Tunnel



```
// Render Tunnel
SDL_SetRenderDrawColor(renderer, 255, 225, 0, 255);
dstrect.x = (windoww / 2) - (tunnelwpixels / 2);
dstrect.y = (windowh / 2) - (tunnelhpixels / 2);
dstrect.w = tunnelwpixels;
dstrect.h = tunnelhpixels;
SDL_RenderDrawRect(renderer, &dstrect);
```

Set Location

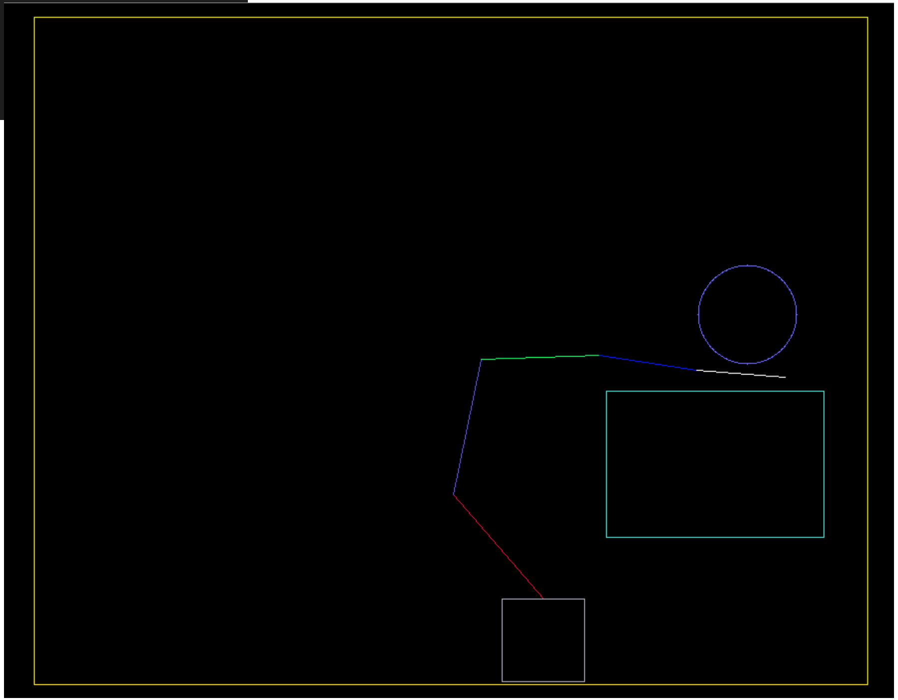
Rendering the Tunnel



```
// Render Tunnel
SDL_SetRenderDrawColor(renderer, 255, 225, 0, 255);
dstrect.x = (windoww / 2) - (tunnelwpixels / 2);
dstrect.y = (windowh / 2) - (tunnelhpixels / 2);
dstrect.w = tunnelwpixels;
dstrect.h = tunnelhpixels;
SDL_RenderDrawRect(renderer, &dstrect);
```

Draw to Screen

Rendering the Tunnel



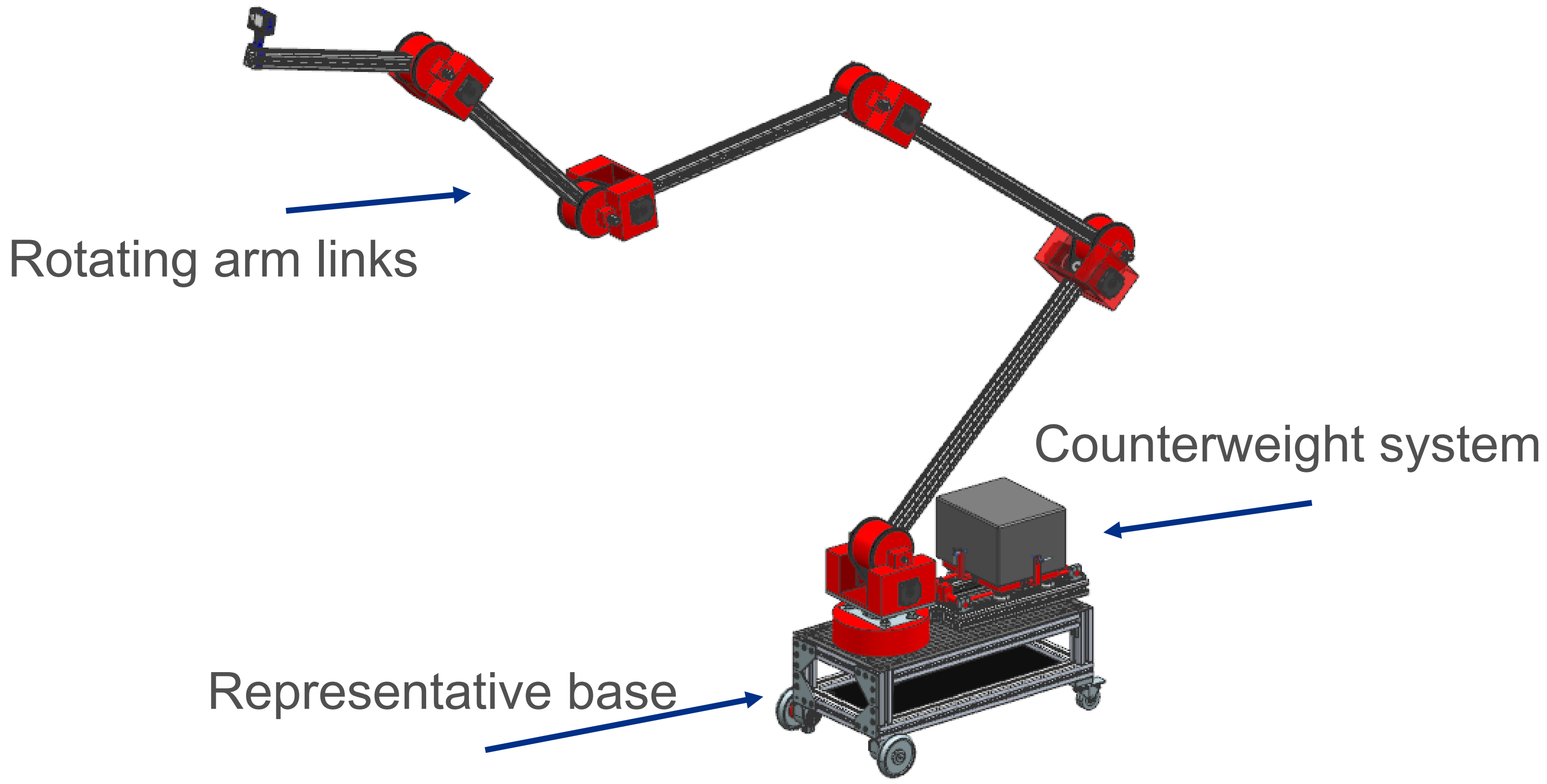
Fermilab

NX CAD

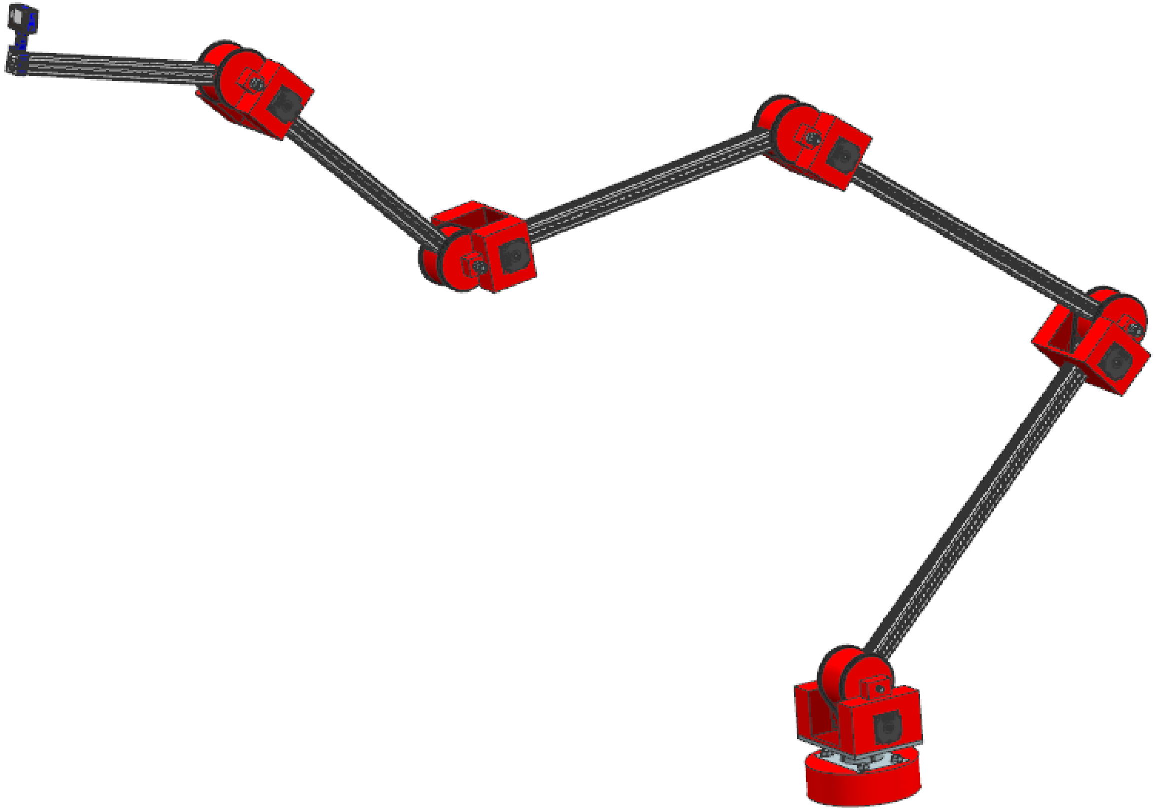
-

Robot Design

Long Reach Robotic Arm

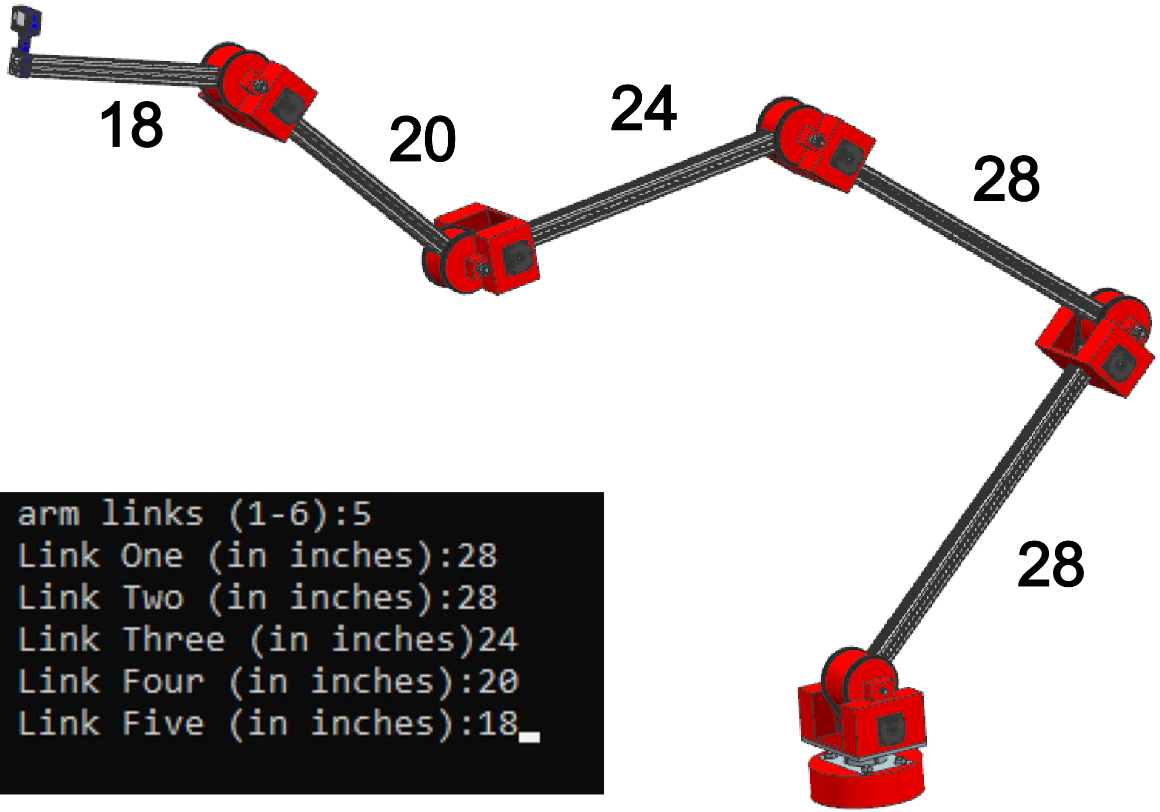


Robot Arm Assembly



Fermilab

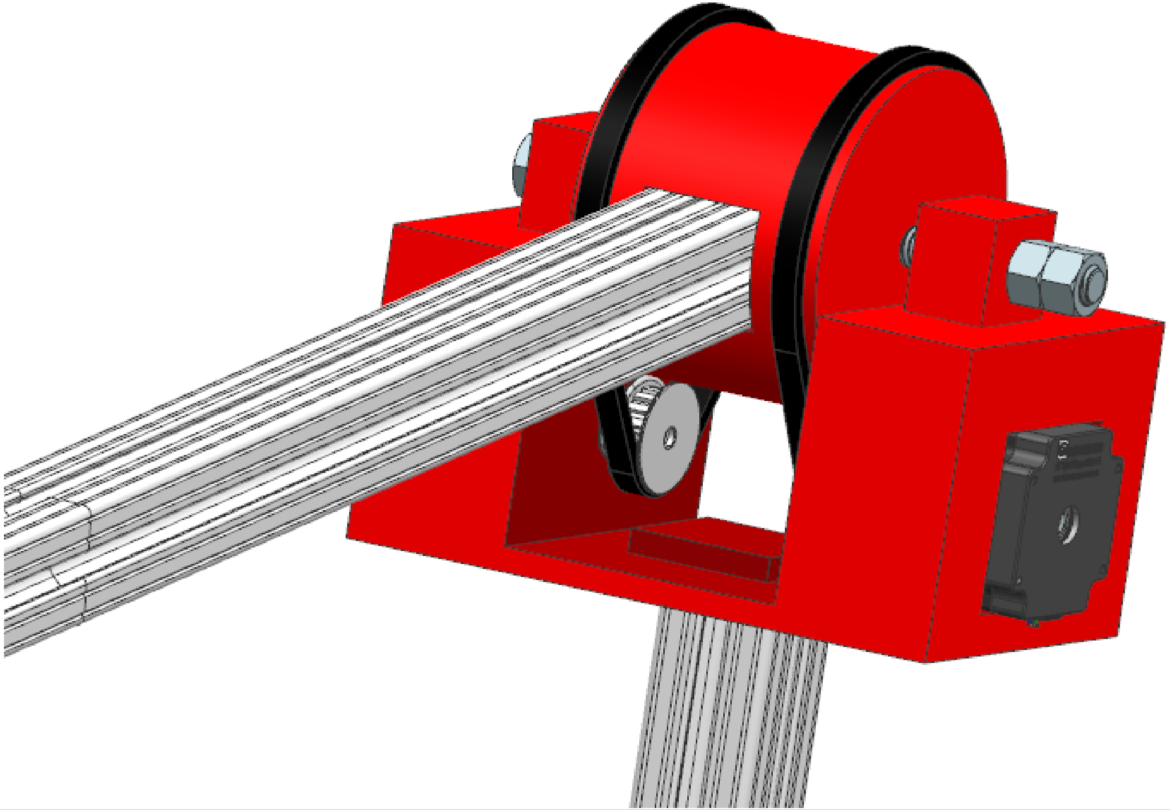
Robot Arm Links




```
Enter number of arm links (1-6):5
Enter length of Link One (in inches):28
Enter length of Link Two (in inches):28
Enter length of Link Three (in inches)24
Enter length of Link Four (in inches):20
Enter length of Link Five (in inches):18
```

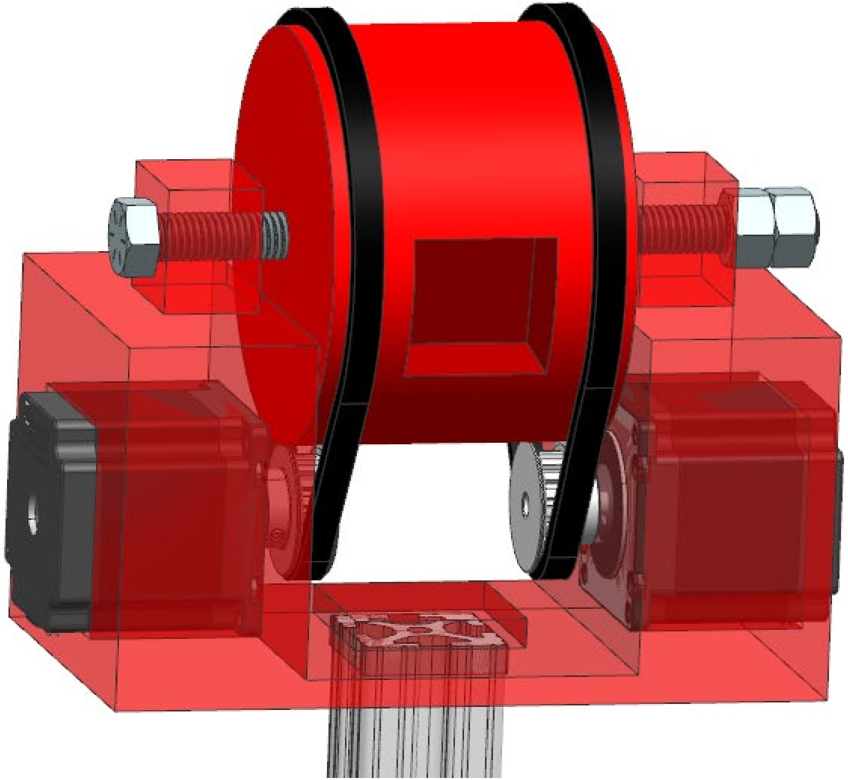

Fermilab

Joint Assembly - Closer View



Joint Assembly - How it Works

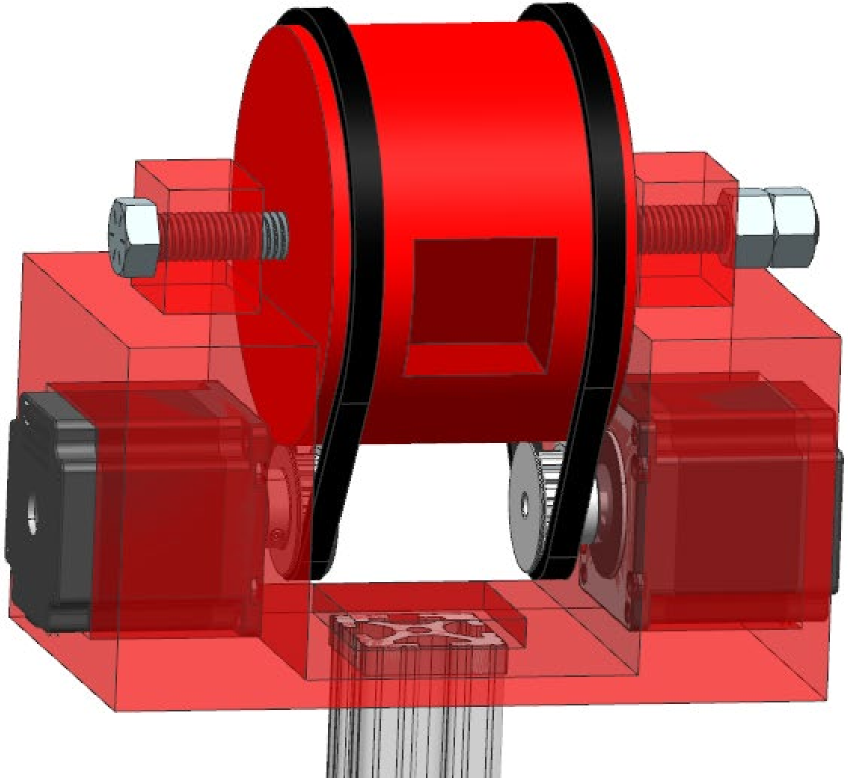
*All red parts are 3D printed



Joint Assembly - How it Works

*All red parts are 3D printed

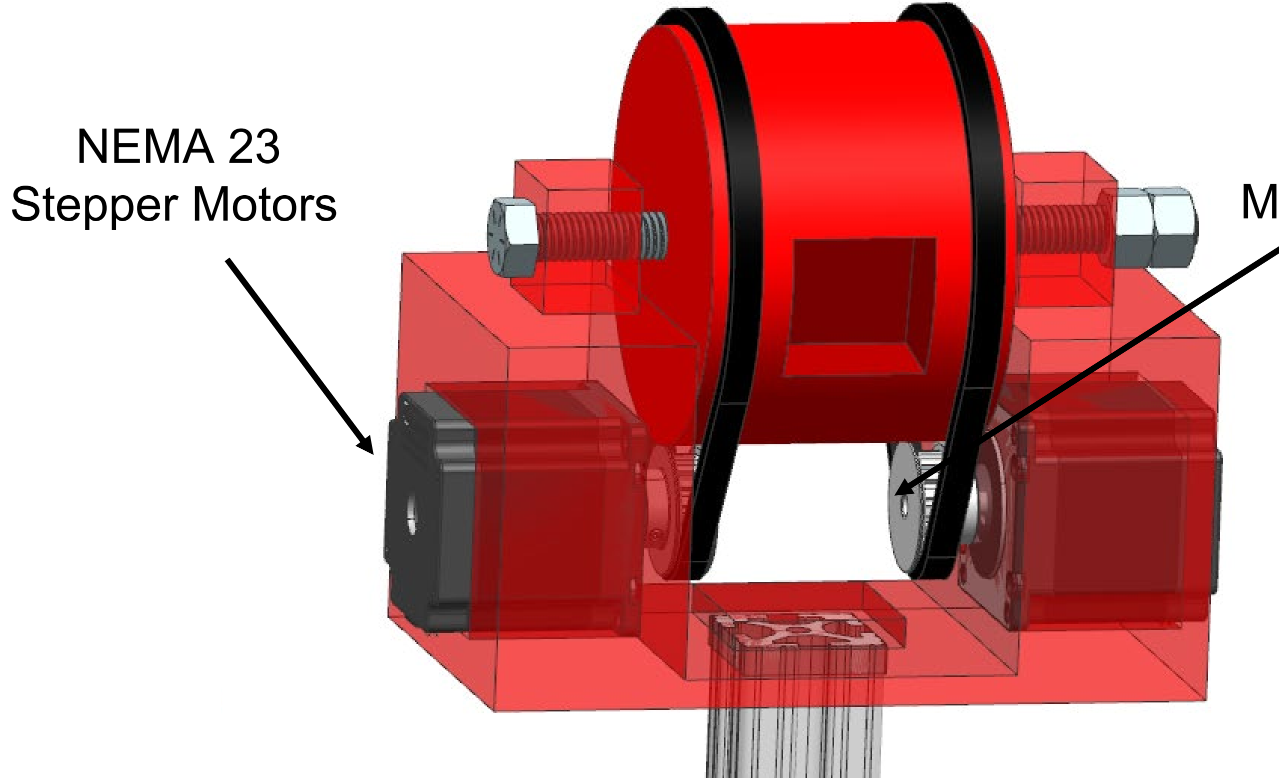
NEMA 23
Stepper Motors



Fermilab

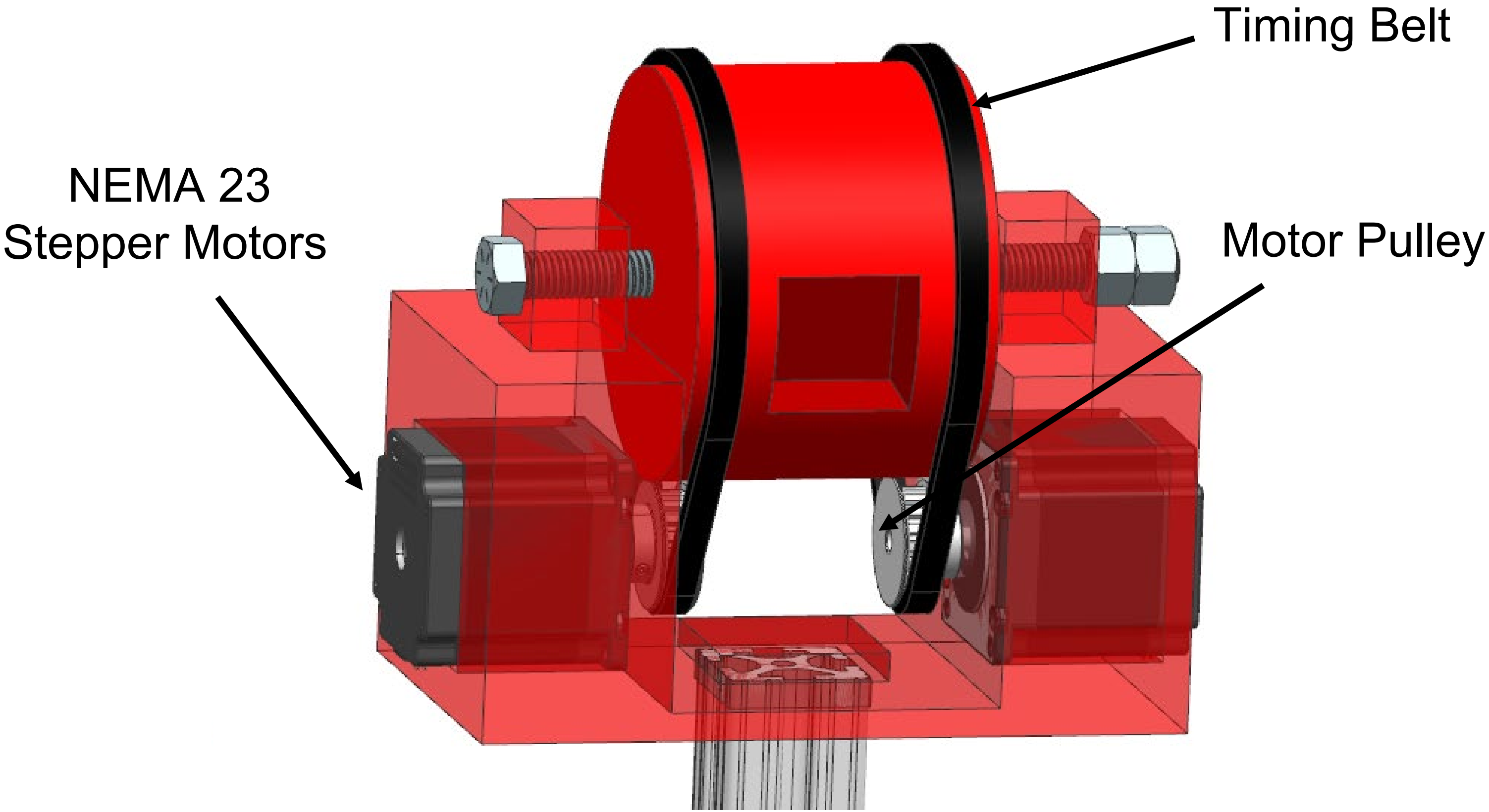
Joint Assembly - How it Works

*All red parts are 3D printed



Joint Assembly - How it Works

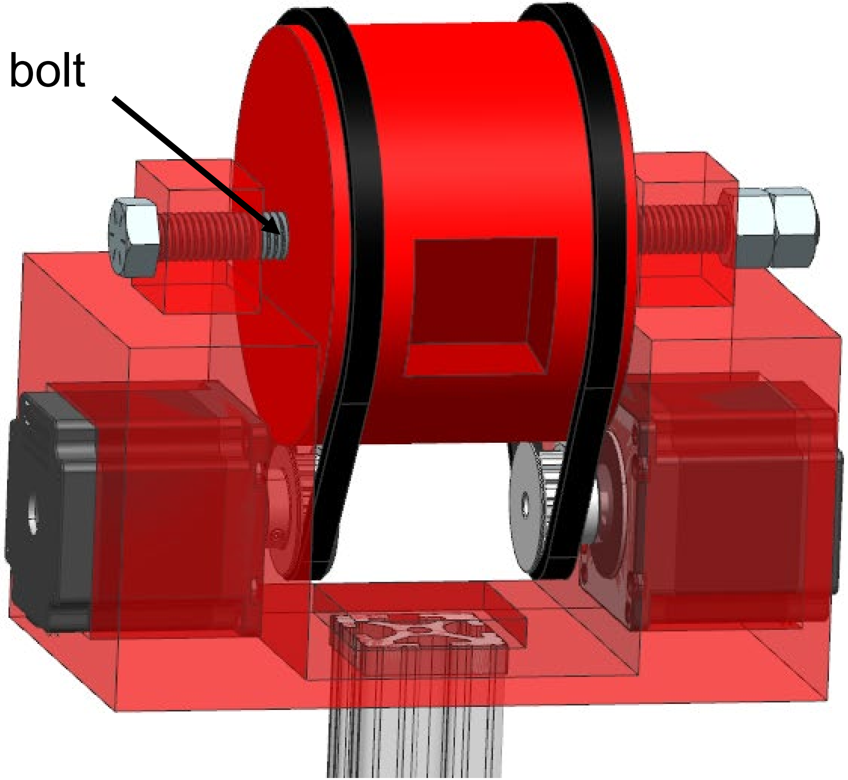
*All red parts are 3D printed



Joint Assembly - How it Works

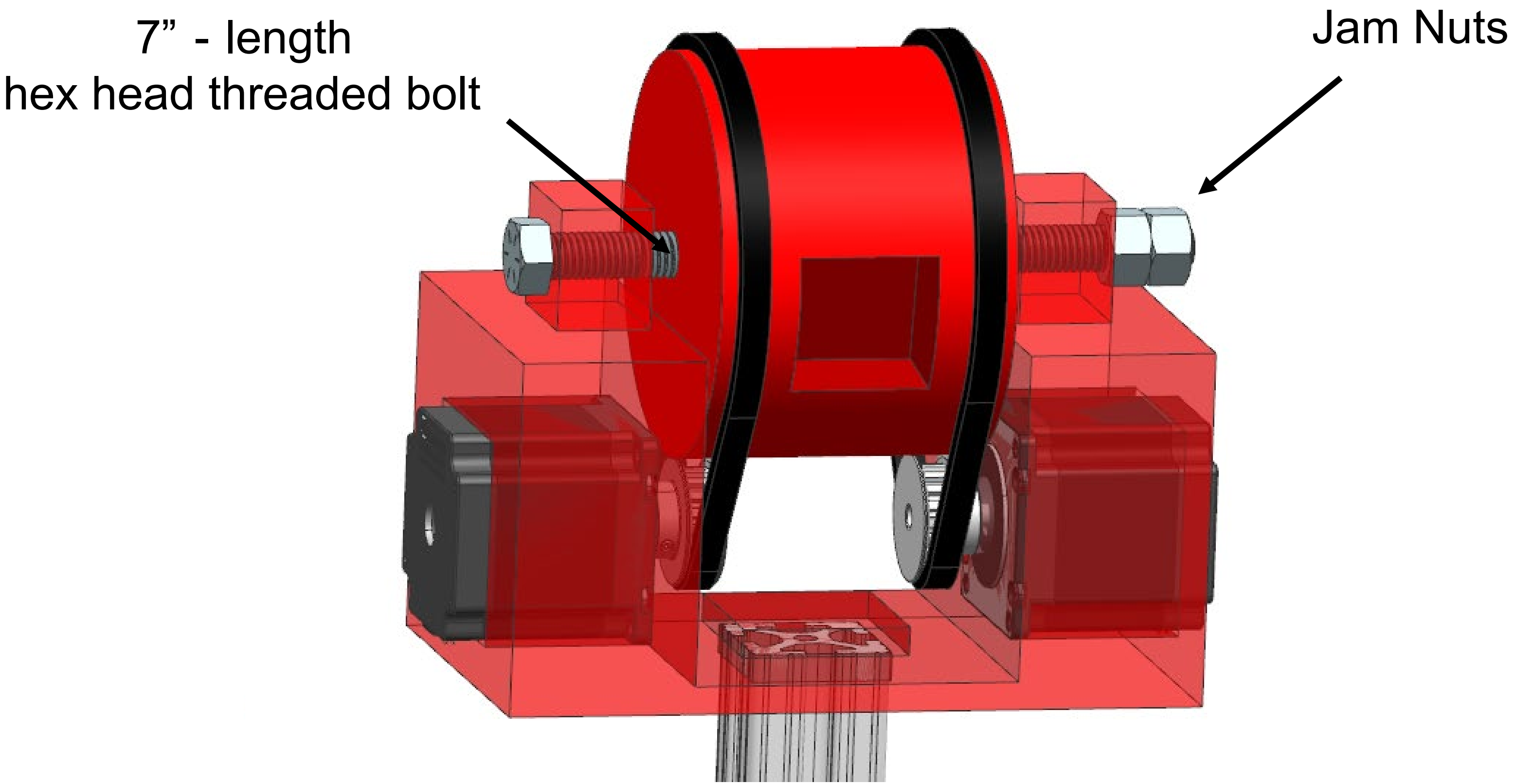
*All red parts are 3D printed

7” - length
hex head threaded bolt



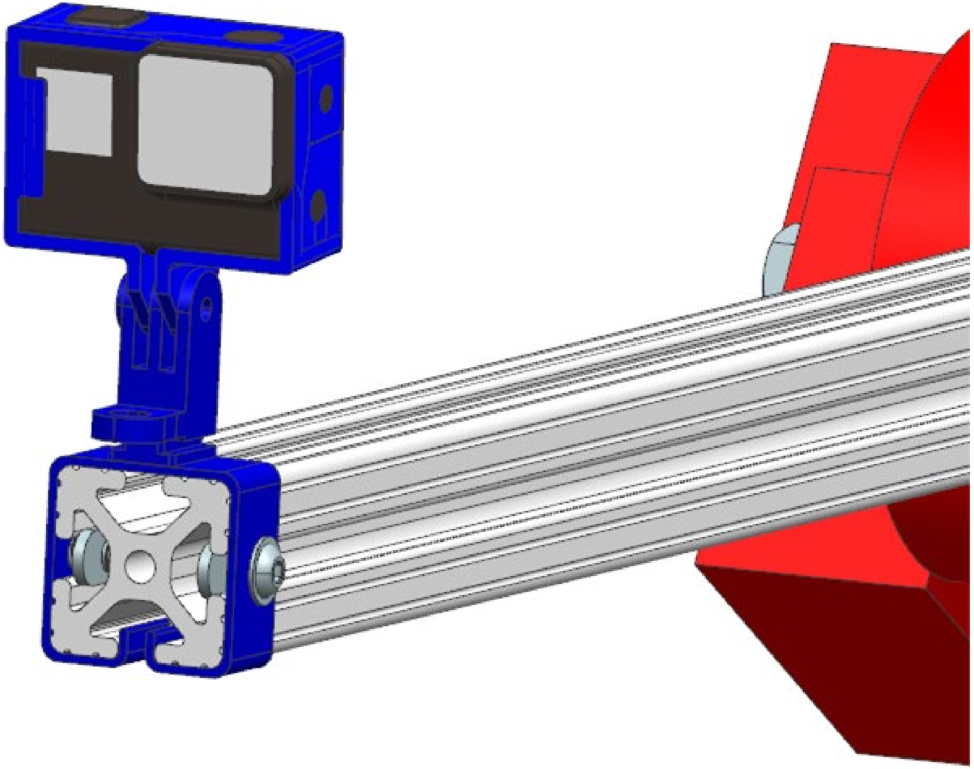
Joint Assembly - How it Works

*All red parts are 3D printed



GoPro Camera Mount

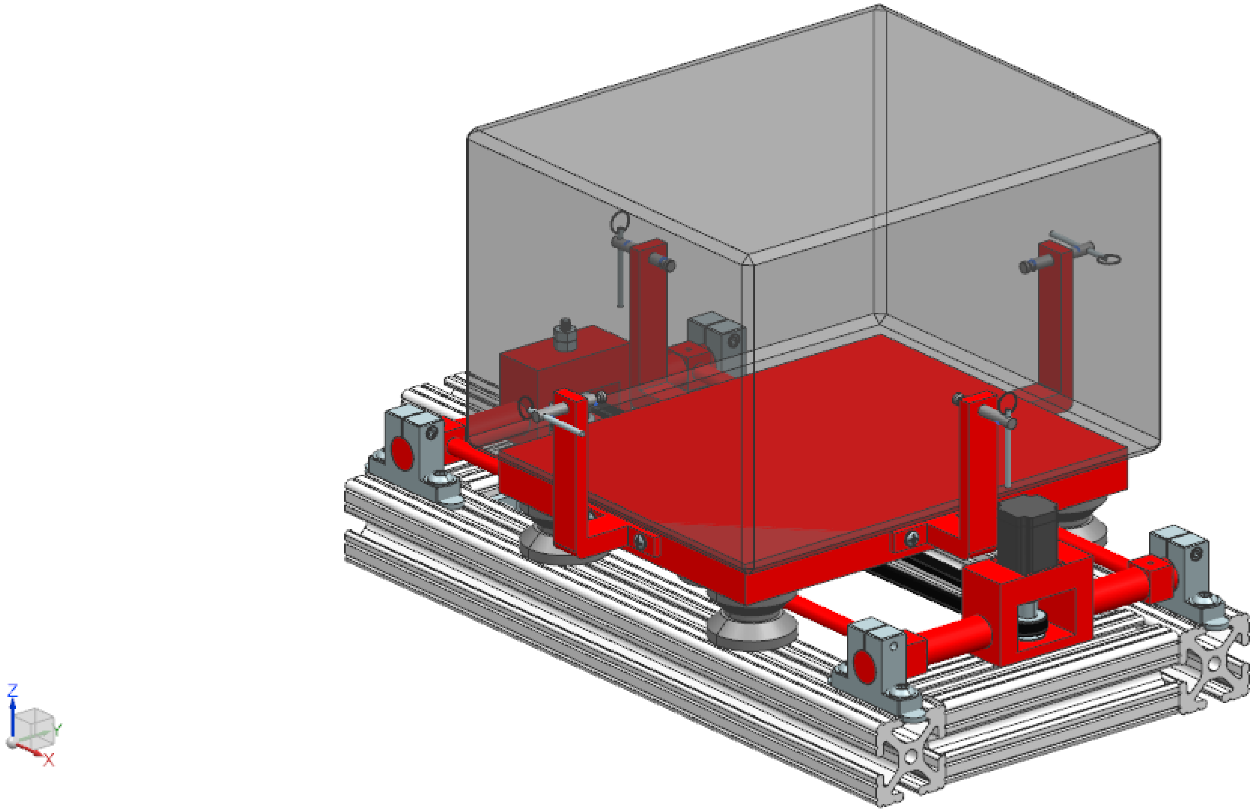
*All blue parts are 3D printed



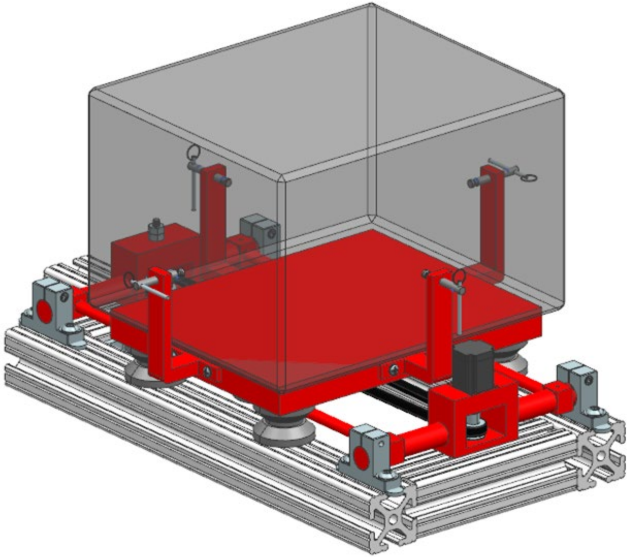
The Counterweight Design

Counterweight Full Assembly

*All red parts are 3D printed

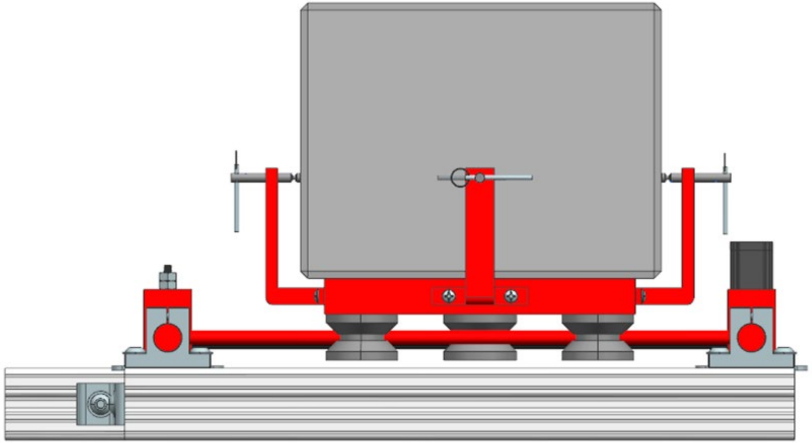


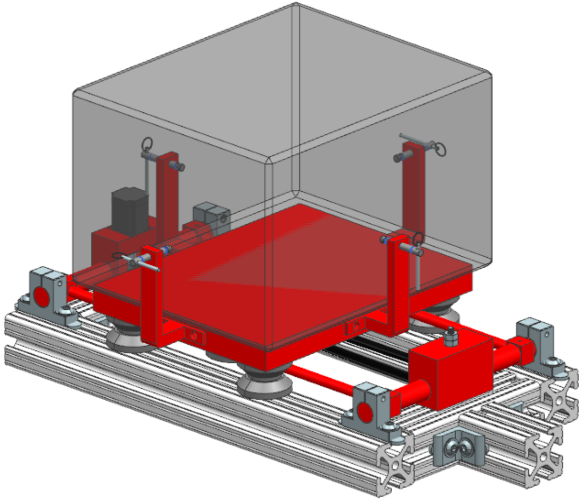
Counterweight



Front view

Side view



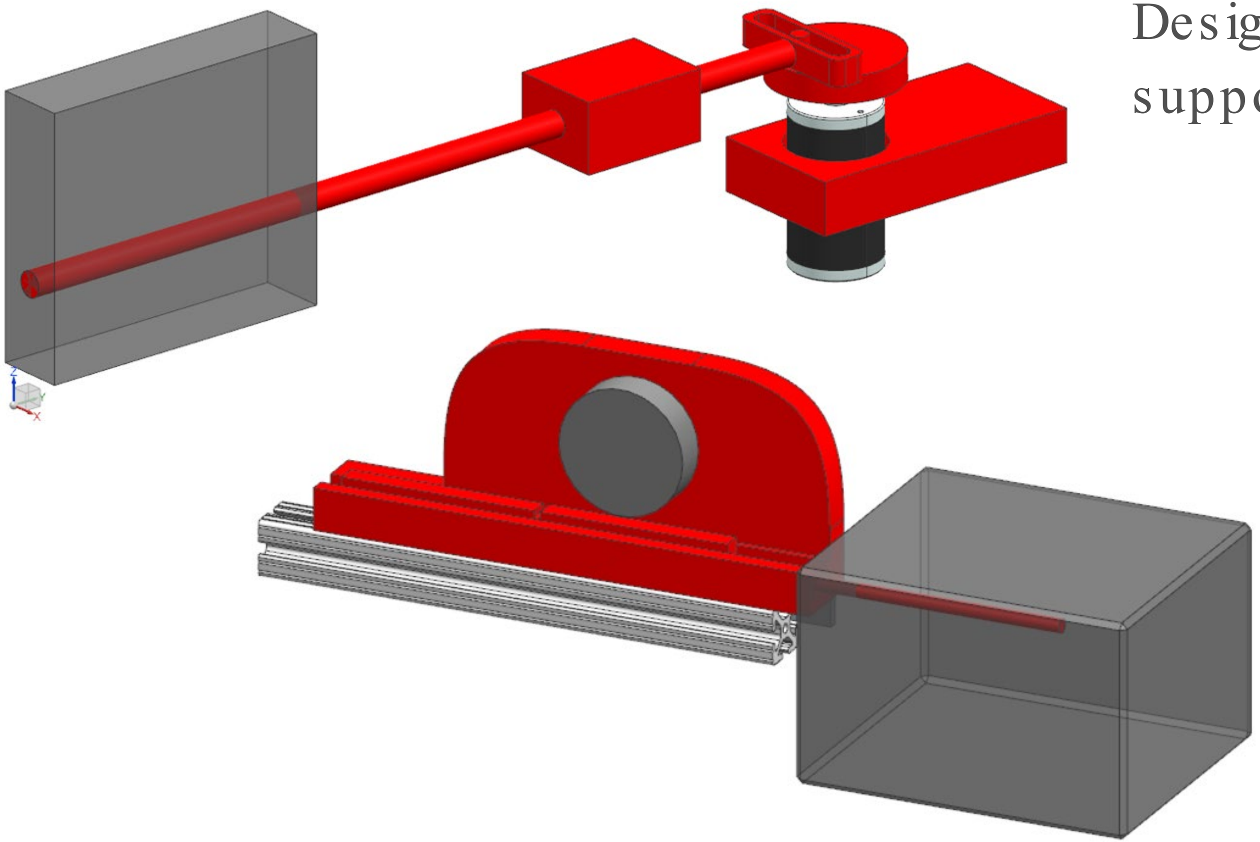


Back view

*All red parts are 3D printed

Counterweight Iteration

*All red parts are 3D printed



Design 1: insufficient support for weight

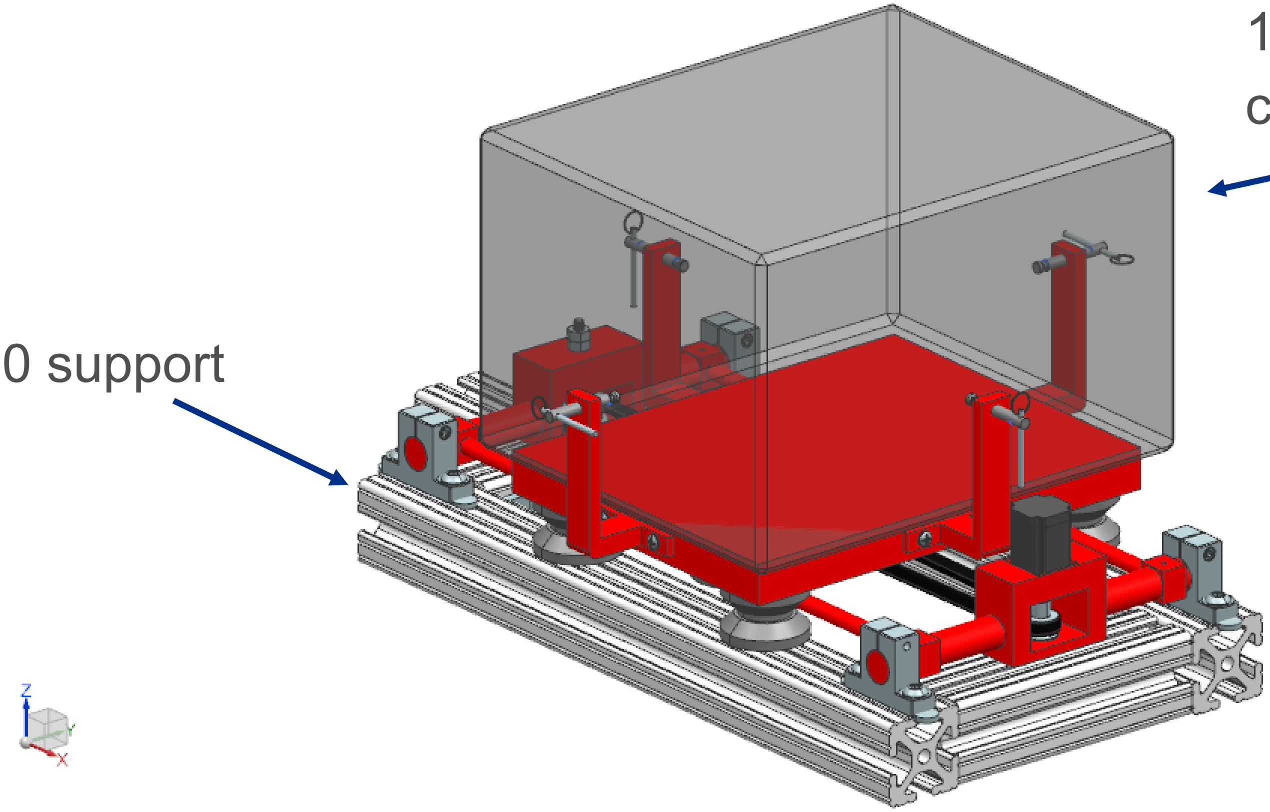
Design 2: limited horizontal movement

Counterweight Assembly

*All red parts are 3D printed

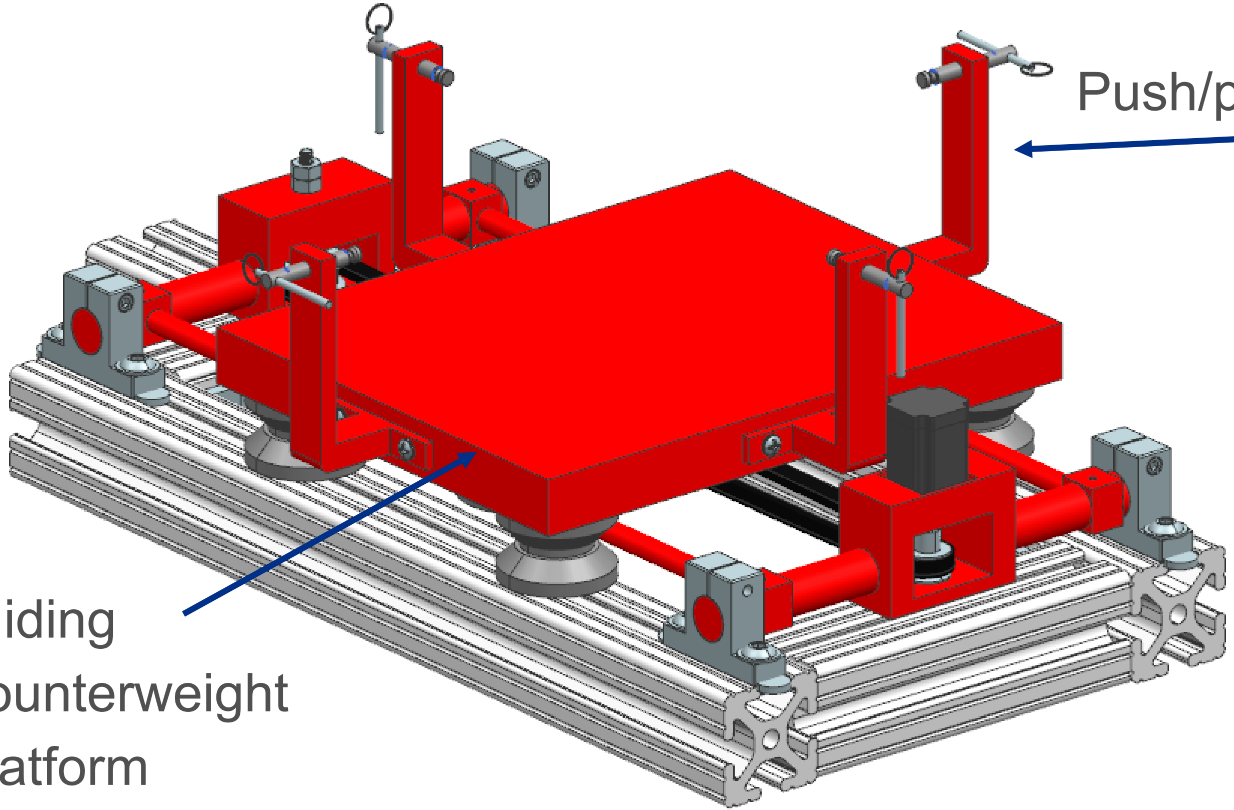
110 lbs steel counterweight

80/20 support



Counterweight Assembly

*All red parts are 3D printed

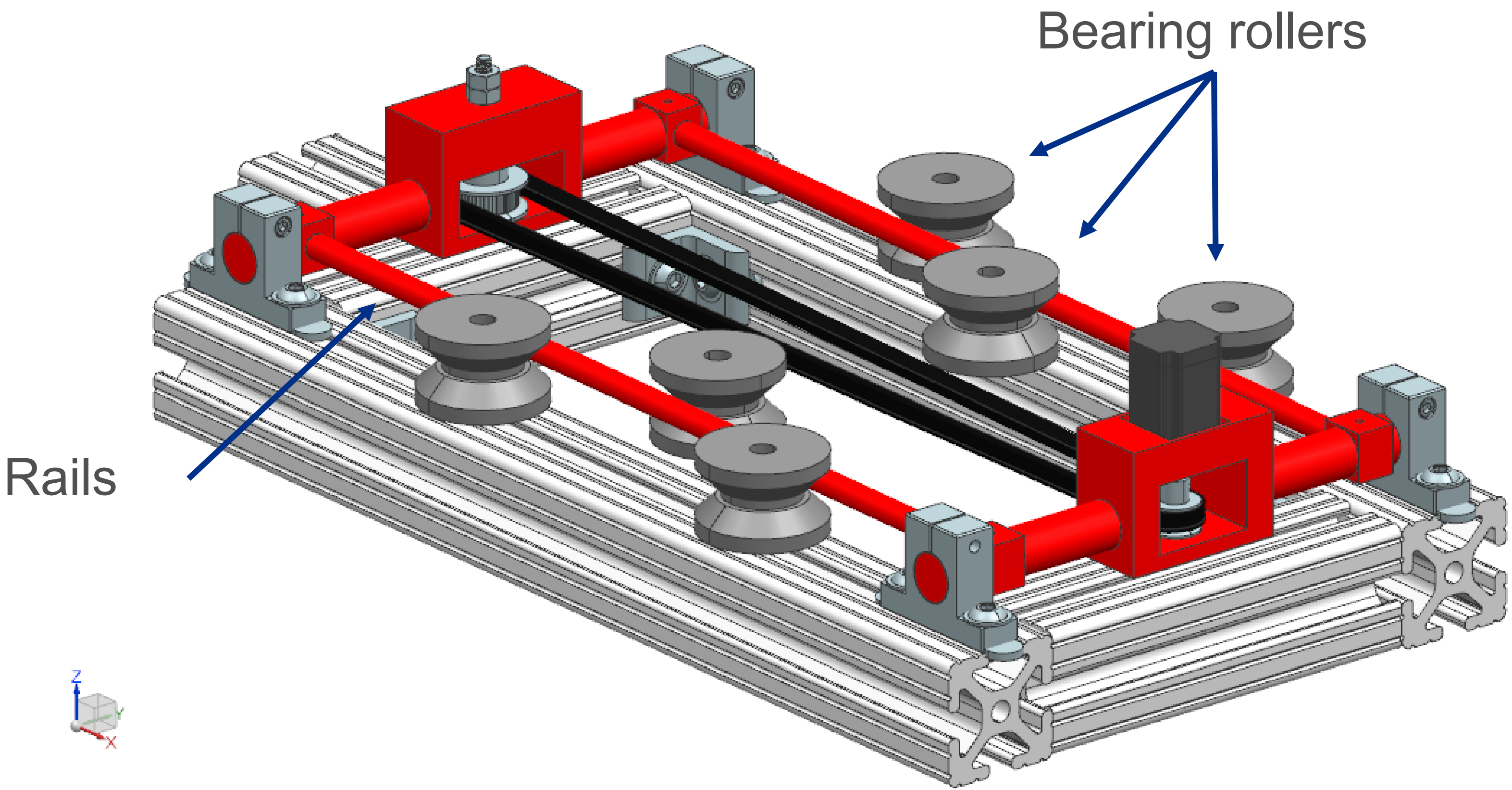


Push/pull support

Sliding counterweight platform

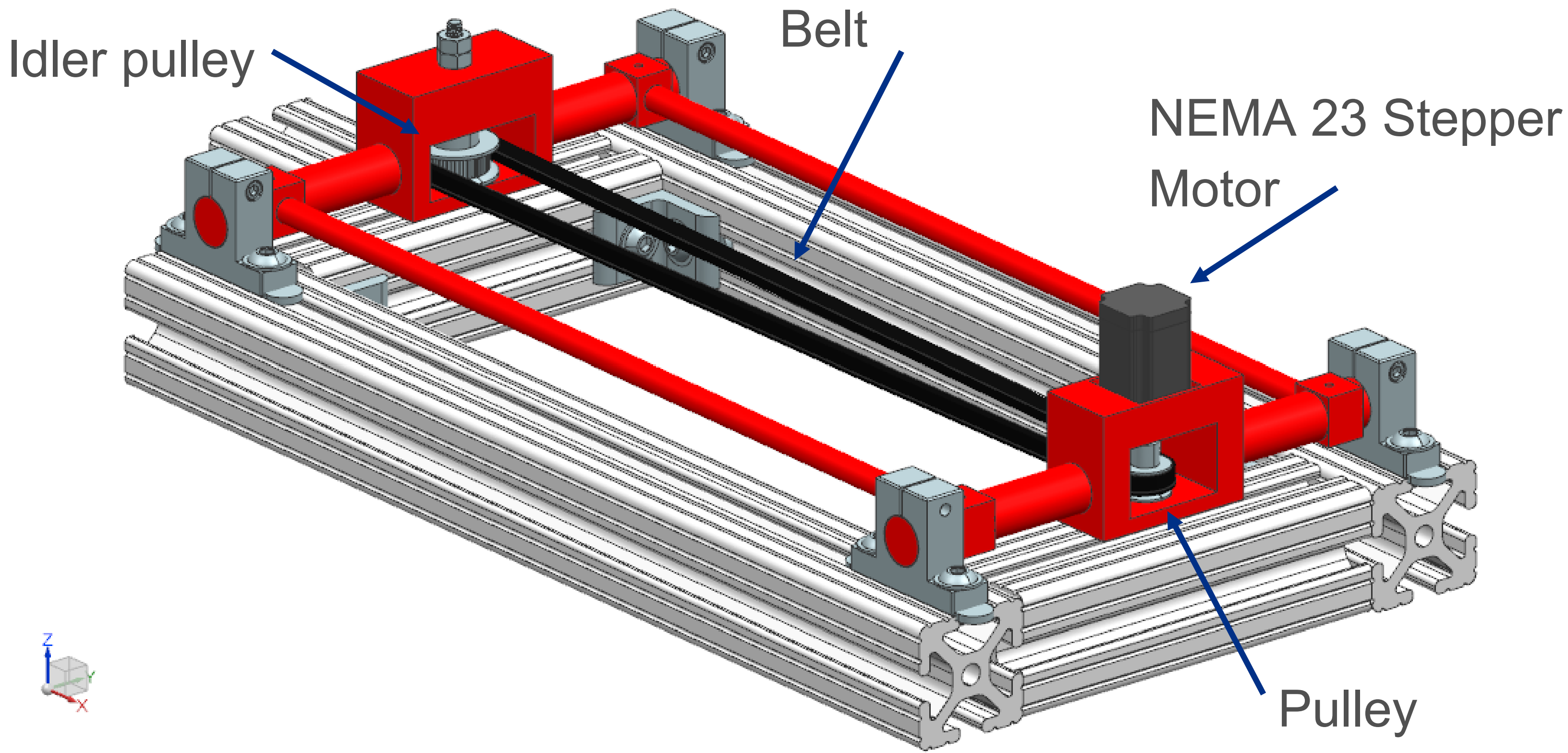
Counterweight Assembly

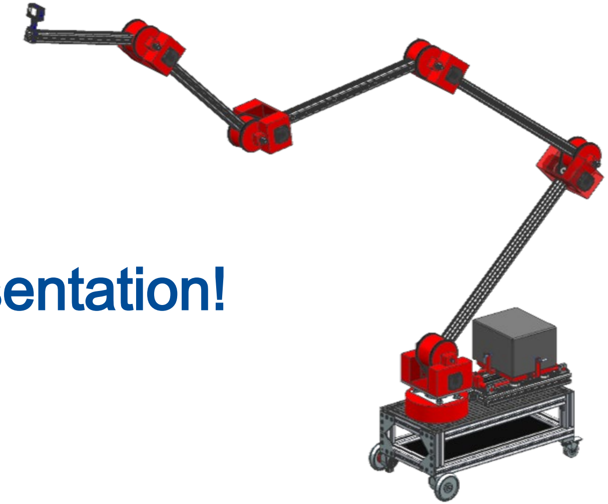
*All red parts are 3D printed



Counterweight Assembly

*All red parts are 3D printed





Thank you for attending our presentation!

-

Any questions?