Kinematics Design and Analysis for Recovery Evaluation of Spinal Cord Injury (KARESCI 2)

A Major Qualifying Project report

submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

degree of Bachelor of Science

By:

Apollinaris Rowe, Landen Kovens

Advisors:

Prof. Yuxiang Liu, Prof. Michael Engling

External Collaborator:

Prof. Wei Wu, Indiana University

Date:

3/25/2024

Abstract

This report iterates on a previous MQP, improving the design for a kinetic rehabilitative-assisted mechanism for recovery from spinal cord injury (SCI) induced paralysis, with an accompanying artificial intelligence model to track the joint positions and obtain data on healing progress. This AI (Artificial Intelligence) model tracks the hand of the mouse without markers to analyze the movement pattern of the mouse to understand the recovery process. The design is adjustable to fit mice of varied sizes, and the linkage motion path and range of motion can be adapted for mice in various stages of the healing process.

We believe that the novel combination of disciplines has the potential to increase the understanding and process of recovery from spinal cord injuries, as well as encourage future developments for applying a similar rehabilitative-assisted mechanism for use on humans.

Acknowledgements

First and foremost, we would like to thank Prof. Yuxiang Liu and Prof. Michael Engling for their continuous support and advisory throughout the year, both in and out of the project.

Additionally, we greatly appreciate all external collaborators from Indiana University, especially Prof. Wei Wu, for dedicating their time to provide generous feedback and testing.

We are incredibly grateful for the dedication it takes to provide guidance throughout such a project, and we greatly appreciate all guidance that was provided throughout.

Authorship

Chapter / Section #	Author(s)
1.	Apollo & Landen
2.	Apollo & Landen
2.3.	Apollo
2.4.	Landen
3.	Apollo & Landen
4.	Apollo
5.	Landen
6.	Apollo & Landen
7.	Apollo & Landen
8.	Apollo & Landen

Table of Contents

List of Figures

1. Introduction

In contemporary society, most people are familiar with the adoption of ChatGPT and other similar programs. However, the applications of Artificial Intelligence (AI) extend far beyond chatbots and image generation, as its ability to recognize patterns in data proves invaluable to researchers, both for spotting structures in complex data and for combing through vast quantities of data.

Researchers have been using AI to identify specific individuals in a photo from a population of animals, specifically whales, at an 87% accuracy. From this, the scientists are hoping to expand recognition to all whales in all regions, allowing scientists everywhere to spend less of their energy identifying individuals and more time on their research [1]. Similarly, there has also been a push to translate the language of sperm whales to English, again with AI being at the forefront of this investigation [2].

With these far-reaching scientific applications in mind, we wanted to explore the application of machine learning to identify mice with spinal cord injury (SCI) and how it could be used in the rehabilitation process, using machine learning to augment the data analysis aspect of the research. Though a previous years' MQP investigated the use of a linkage for mechanical rehabilitation of mice with an SCI, the addition of AI analyzation would allow for a more complete understanding of the recovery process and the mechanisms at play, allowing for the potential of a more complete recovery compared to previously implemented methods.

By analyzing videos of mice with and without spinal cord injury, two machine learning models could be run in parallel to identify whether the mouse has a spinal cord injury or not, and potentially, which vertebrae the injury occurred at. Additionally, the model could also potentially quantify the recovery, specifying what sort of rehabilitation method provides the greatest recovery benefit. The mechanism could then be adjusted to fit this ideal motion, allowing the mice to reach as complete of a recovery as possible.

The goal was for this project to build on the accomplishments of last year, combining computer science, mechanical engineering, and neuroscience disciplines. We aim to refine the kinetic adjustability and overall function of the physical mechanism while also introducing the use of machine learning to quantify rehabilitation results and maximize regained kinetic function, hopefully providing the groundwork for further research and accomplishments as the technology advances.

2. Literature Review

2.1. KARESCI-1, Previous Years' MQP Report – Findings and Summary

While unfortunately, as will be further discussed later, the previous years' team failed to accomplish their original goal of designing a complete functioning rehabilitation mechanism, their work proved invaluable in summarizing previous research, narrowing down hardware and design decisions, and giving lessons of their successes and failures.

First, and most importantly, their pre-project literature research concluded that rehabilitation-assisted motion devices have been shown to be effective in certain cases where mobility is lost. These devices are not only valuable to those whose movement can be rehabilitated, but also to neuroscientists and researchers hoping to better understand the neurorehabilitation process [3].

Since WPI is unable to perform laboratory experiments related to spinal cord injuries on rodents, the previous years' team had gained a working relationship with Indiana University, collaborating with them for videos, guidance, and testing. Neuroscience labs predominantly make use of laboratory mice because of their general medical similarity to humans and the broad genetic modifications they can exhibit [3]. While the larger size of rats could make the mechanism's design more forgiving and capable, Indiana Universities use of mice constrained the scale of the mechanism to fit only mice.

From their research, they concluded that optimum locomotive rehabilitation occurs when the animal is moving in its natural orientation, or in the case of this project with mice, quadrupedally. Additionally, while previous locomotive designs do exist, they are often rendered inadequate for complete testing as they were manufactured to fit only a single specific rodent, or they are simply designed as a concept without any physical mechanism being created [3]. These downfalls are part of what drove their project in the first place, attempting to mitigate the downfalls of these other devices with their own kinematic mechanism.

Their research throughout the term eventually settled to using stepper motors because of their high torque, accurate rotational positioning, and excellence at low speeds. An EWA2-CW3C encoder was attached to the steppers to integrate feedback into the system, with all of these being wired to an Arduino Uno because of its simplicity and effectiveness [3].

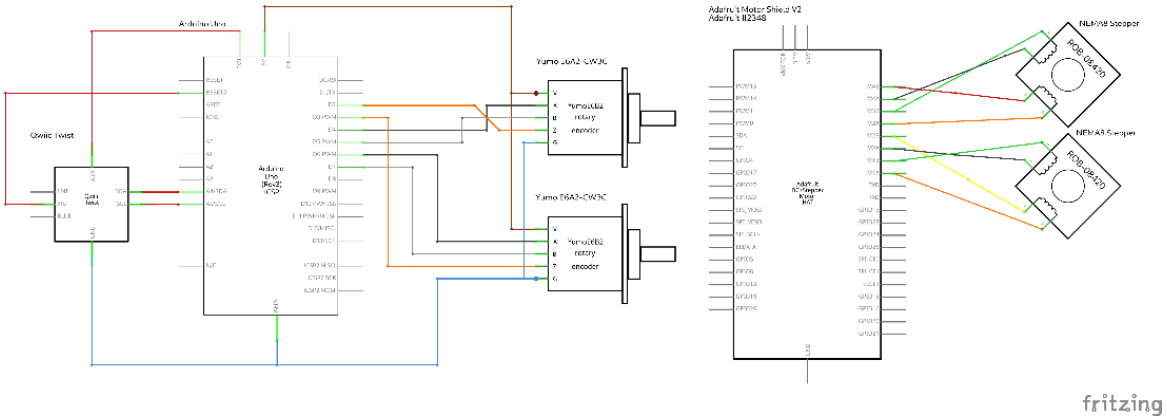

Figure 2-1: Wiring diagram of previous years' project [3]

Their mechanism consisted of a small linkage attached to the stepper motor, with gears connecting the stepper to the encoder. This linkage follows the motion path of the midpoint of a mouse's tibia, sliding along beams across each axis for adjustability to different sized mice. Every part of this design was 3D printed because of its low cost and ease of prototyping [3].

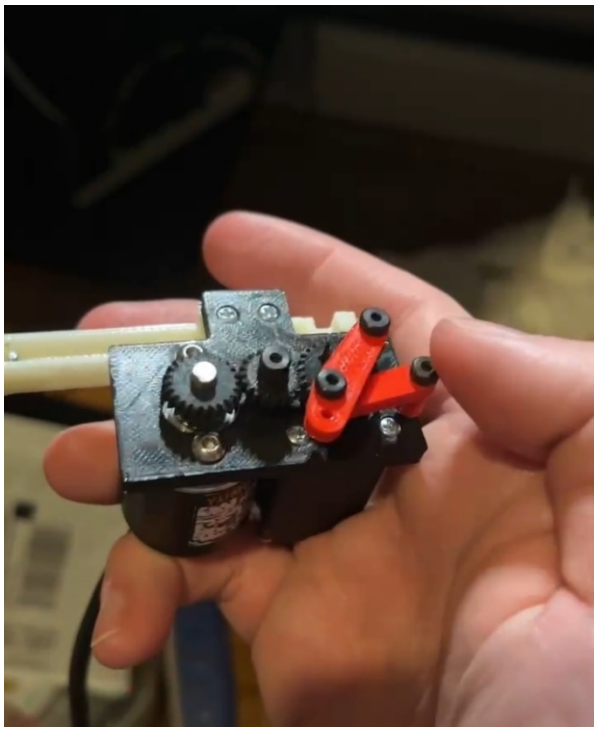

Figure 2-2: Final linkage mechanism from the previous year [3]

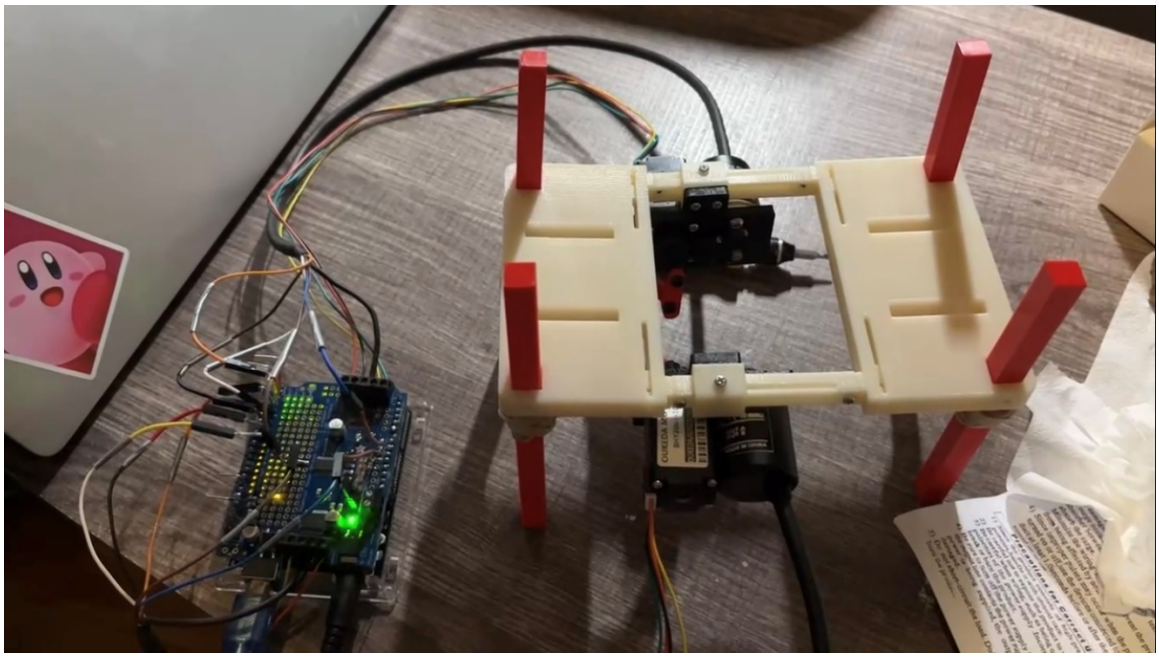

Figure 2-3: Previous years' entire mechanism [3]

While the team aspired to integrate automated data collection and other deeper computer science related behaviors, this aspect of their project ultimately fell through. Ultimately, the accomplishments of the CS team came down to programming the Arduino and assisting in the electronics control [3]. The previous mechanism functioned to the extent of providing mechanical motion to a specified part of a mouse's leg, however, the general motion, adjustability, mouse attachment, and measurement of rehabilitation effectiveness resulted in a device that was unfit for laboratory testing on injured mice.

2.2. Research Direction and Division

With the previous project in mind, the research going forward aimed to mitigate gaps in the previous years' research or accomplishments. These research duties would be split between the CS and the ME teams, with each focusing on the topics deemed relevant to their respective aspects of the project.

2.3. CS Literature Review

Before building a machine, learning model there were many different topics, we desired to research to build a robust and accurate model. Due to the previous group's focus on the mechanism, there was little to build off for creating a machine learning model.

2.3.1. Mouse Anatomy and Spinal Coord Injury

To support the knowledge of the team research into spinal cord injury was done to understand how to perform motion analysis. The reason for using a machine learning model to automate the motion analysis was to analyze a large amount of video data quickly and create a deterministic result. Previously, motion analysis done to determine spinal cord injury was done by a team of neuroscientists observing a mouse repeatedly accomplishing different tasks

For the classification of the mouse by its joints it was necessary to understand the anatomy of a mouse and what all the different joints were. So, by using the four major sections of the spine and the key features of both front and hind limbs a general skeletal model was created to guide the creation of the machine learning model [4].

Additionally, a general understanding of the spinal cord injury recovery process might be beneficial to the formation of the machine learning model. Knowing which motions to track and what the different motions of a mouse with and without spinal cord injury will be beneficial towards reinforcing the model during the training. So, by being familiar with the mouse locomotive process and what neurons help aid the recovery process of mice with spinal cord injury were useful for the creation of the model [5] [6].

2.3.2. Machine Learning with DeepLabCut

A general understanding of machine learning was also required to accomplish the goal of this project mentioned in the introduction. The Indiana University collaborators recommended a software package called DeepLabCut for marker less pose estimation of the mice. The software package is highly reputable for its accuracy and ease of use, however being to ensure that the right decision was being made more research into machine learning was needed [7]. DeepLabCut (DLC for short) creates and trains a machine learning model in the following workflow.

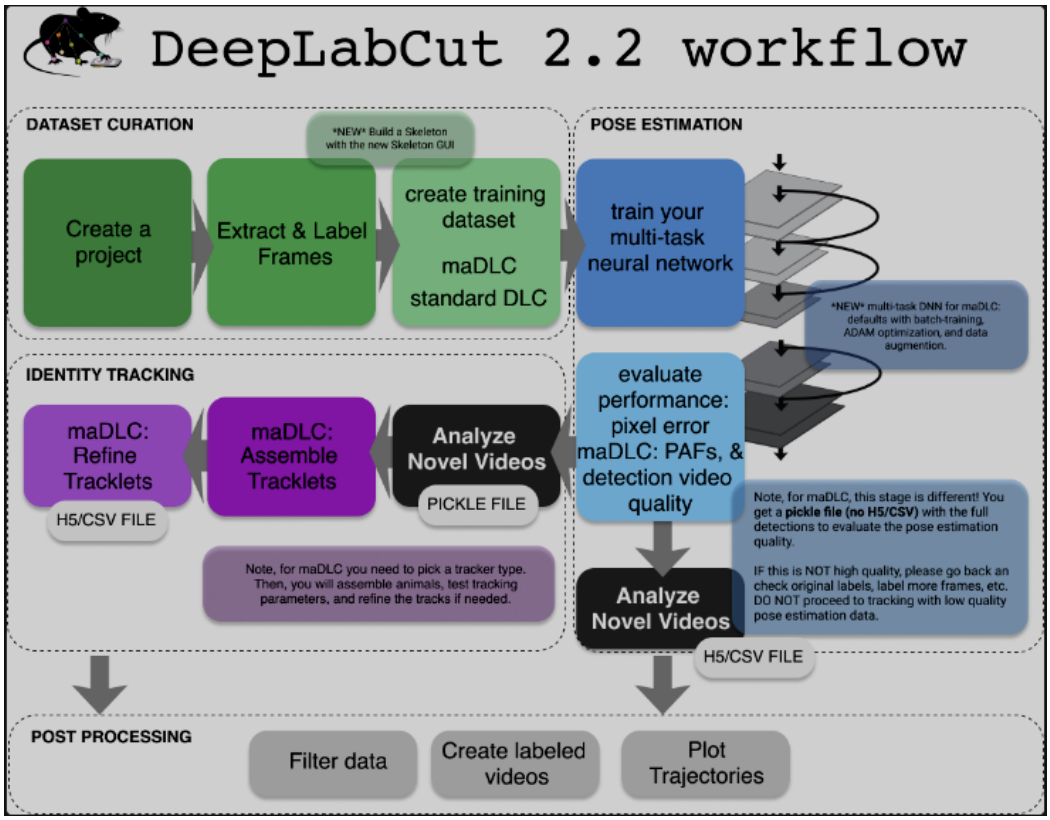

Figure 2-4: DeepLabCut machine learning model creation workflow [8]

Training a neural network requires three main steps, collection of data initially, label the data manually, and finally train the network. Collection of the data needed to train a model was done by the team at Indiana University since they had the laboratory set up to record mice initially from a top-down point of view. Later, the Indiana University team created a lab set up to record side and top-down views of the mice performing an action simultaneously. All instances of the video recordings for this project would be of a mouse performing a grabbing action.

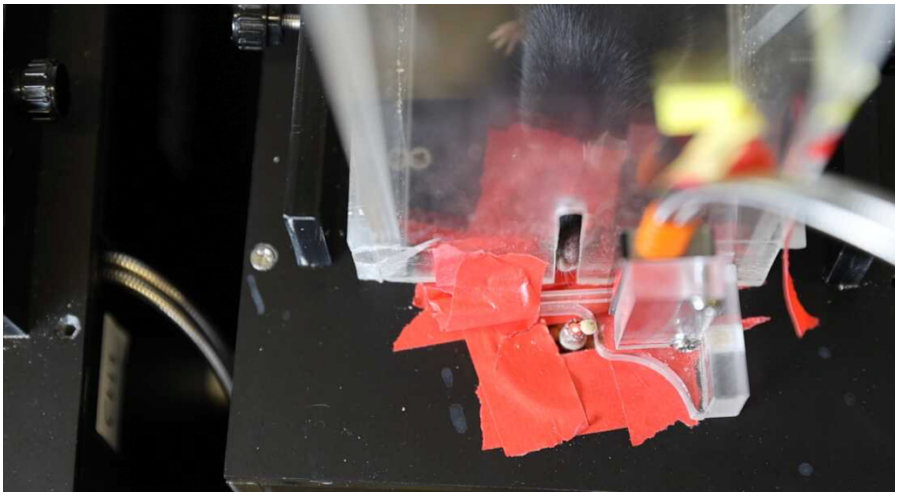

Figure 2-5: Example frame from an initial video from the Indiana University Team.

The data is then separated into a training data set and a testing data set. The training data set is then labeled by the user manually. Going through the skeleton that is defined in the configuration file the user will then manually put dots at each of the joints defined. A training data set is then created with a split of training to testing data which is defined in a project configuration file. The structure of the training data set can also vary, however the most common model architecture that was used was a resnet_50 architecture [9]. The training process of the neural network tunes the parameters of the model to track according to the labeled data. The resnet architecture contains 48 convolution layers which are transformation matrices full of parameters that transform the input layer into trackers on the output layer [10]. The size of each matrix is equivalent to the length of the input video times the width of the output frame. The output layer then is a probability matrix having the probabilities that the joint is in each part of the input video [11]. For the DeepLabCut software package, there are many dependencies that are used in the creation of the machine learning model. The package Tensorflow is the main machine learning package that does all the computation of training and processing a machine learning model [12].

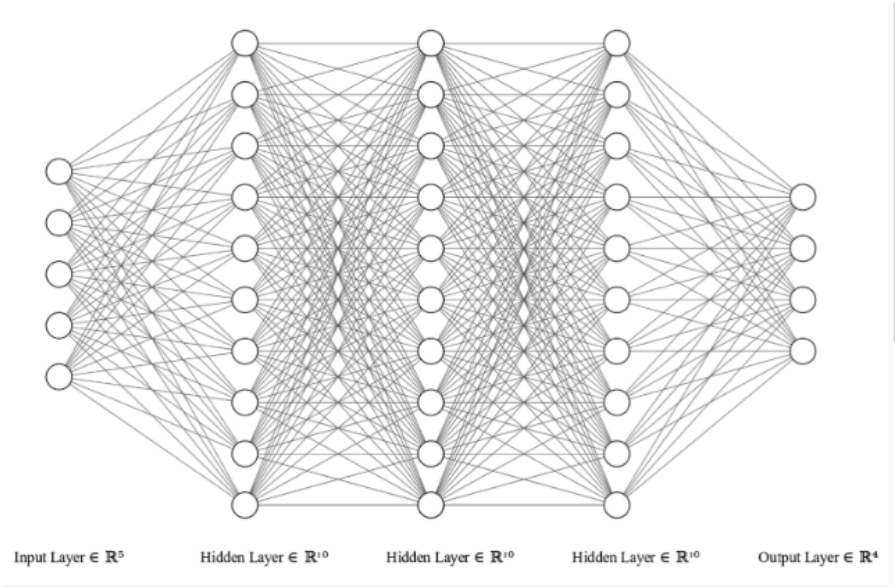

Figure 2-6: Decision tree of a neural network with three hidden layers [9]

Another key dependency that was research was Scikit-Learn, a statistics library that was used to create many of the summary statistics and tuning of the model [13]. When analyzing a video, the input frame is broken down and fitted to be the same size as the input layer of the model. Then running the input through the convolution layers the output is collected. The region with highest probability on each layer of tensor is then filtered into a data sheet. Those coordinates on the datasheet can be fitted back onto the input video.

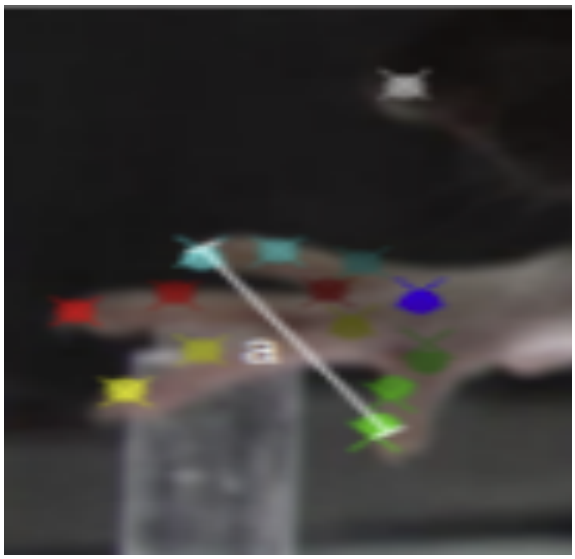

Figure 2-7: Example image of a mouse hand with data points fitted based on a neural network specified to track a mouse's hand (Sourced from Ahnsei Shon of Indiana University). The points correspond to joints in the hand of the mouse.

2.4. ME Literature

While the CS aspect of the project focuses predominantly on machine learning and is therefore entirely computer-based, the ME aspect is primarily physical with a kinematic mechanism. This mechanism consists of linkages, a mechanical system of interconnected beams which transmit motion along a predictable path. This path, or the coupler curve, can be designed to follow a specific trajectory that fulfills specific requirements, which in this instance, governs the motion path of a specific point on the mouse's limb.

2.4.1. Additional Linkage Investigation

Since the previous years' research confirmed the viability of a kinematic rehabilitation device for spinal cord injury induced paralysis in mice, while additionally investigating previously developed devices to emphasize the need for a new device and additional studies to fill in the shortcomings in previous research, the new research had the goal of confirming and refining the previously information. This research initially consisted of obtaining a better understanding of the native motion of mammalian limbs, the hopes of which would guide more specific direction of the smaller-limbed mice.

While specific classification of the function of a leg is possible, the complex interactions between each of the many components in a leg can often make simplification difficult. However, one model by which this simplification can be executed is by adapting a leg into a linkage, specifically two four-bar linkages connected into what's known as a 'pantograph' mechanism [14].

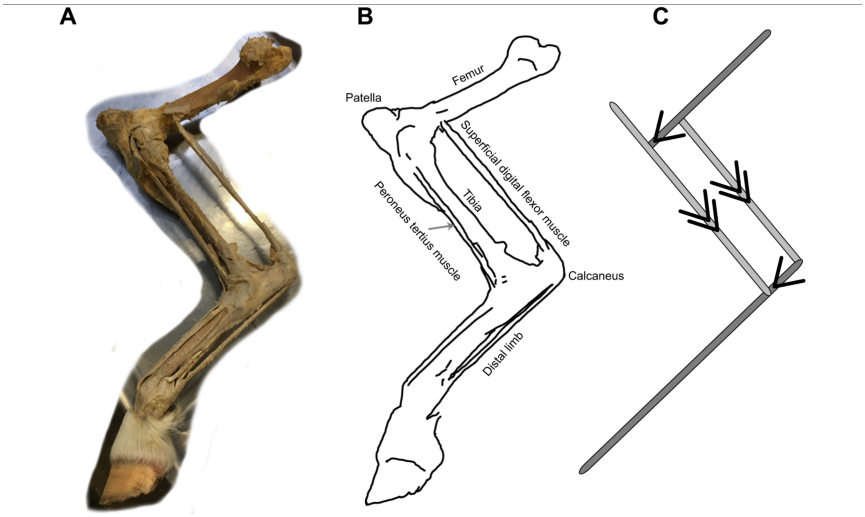

Figure 2-8: Gradual simplification of a horse hindlimb into a 'pantograph' mechanism [14]

The foot can be positioned by adjusting the angle of the femur and the angle of the tibia with respect to the femur. While employing much less control than is required in a real leg, this simplified mechanism accurately mimics the movement of the limb it is based on, allowing for realistic motion to be interpreted and recreated without understanding the precise positioning of each individual section of the hindlimb [14]. Additionally, the specific limb lengths in this linkage can be modified to accurately imitate the hindlimb motion of other animals, as was demonstrated with a dog [14].

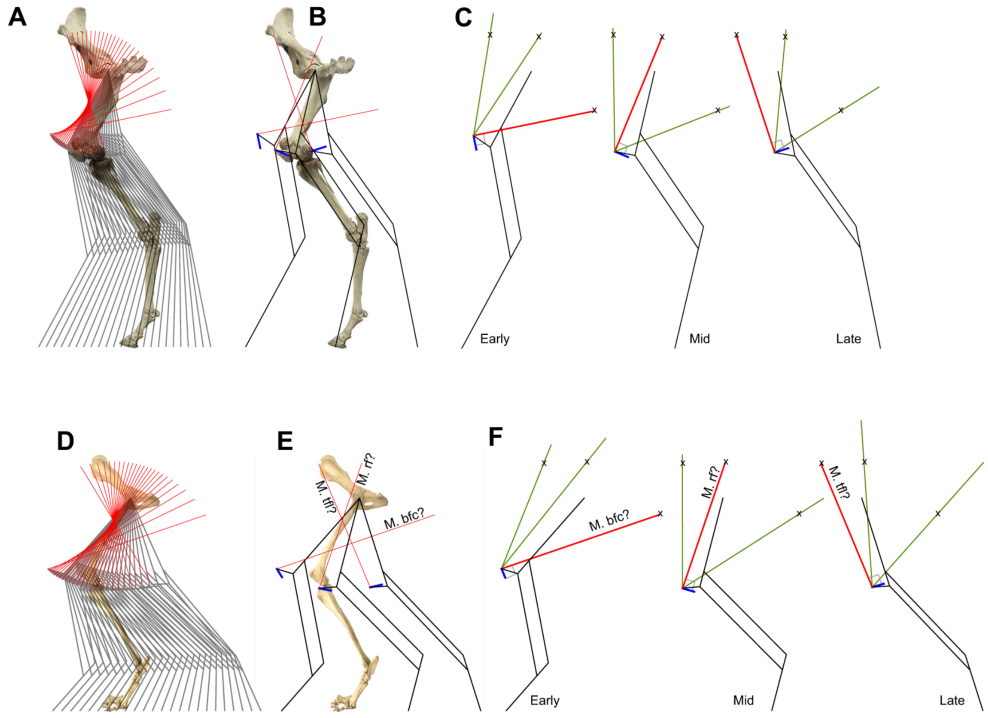

Figure 2-9: (A-C) is horse motion, (D-F) is dog motion [14]

As last years' research shows, complex mechanisms with multiple degrees of freedom already exist for kinematic rehabilitation, however, these all lack either versatility or a proper physical implementation [15]. Outside of the scope of rodent rehabilitation though, many 5-bar linkage designs already exist with their capabilities and limitations being well established.

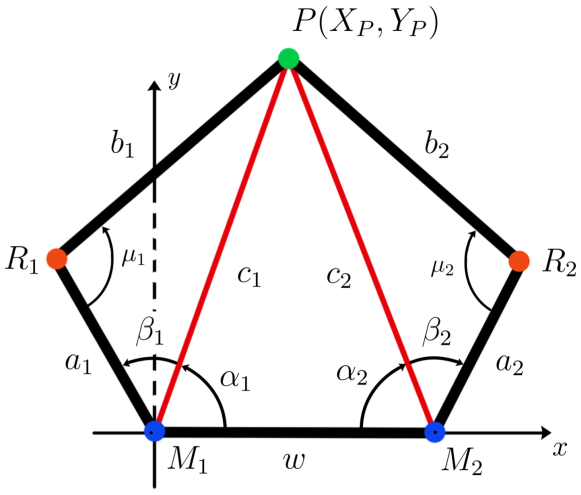

Figure 2-10: Notation used for a 5-bar linkage [15]

2.4.2. Rodent Limb Kinematic Investigations

While general mammalian motion has been investigated, the exact motion path of the mouse hind limb still required additional investigation, especially the variance that occurs between individual mice. This information is vital to understanding the degree to which adjustability must be considered, as well as the predictability of a specific mouse's expected gait.

Comparing wild mice to a specific breed of lab mice which lack muscle spindles (EGR3-KO), not only were there large differences in gait between the breeds, but there was a variance between the motion path of each mouse's hind limb, this being especially evident in the laboratory mice. The way each mouse reacts to an electrical stimulus also varies depending on the individual, with the motion similarly being more closely clustered in the wild mice than the laboratory mice [16]. Though this is an extreme example since muscle spindles provide sensory and positional feedback for the limbs, this study at least shows the variance that can be expected through different mice, especially since mice recovering from a spinal cord injury could have a changing sensitivity in their muscle spindles as their kinematic capability increases.

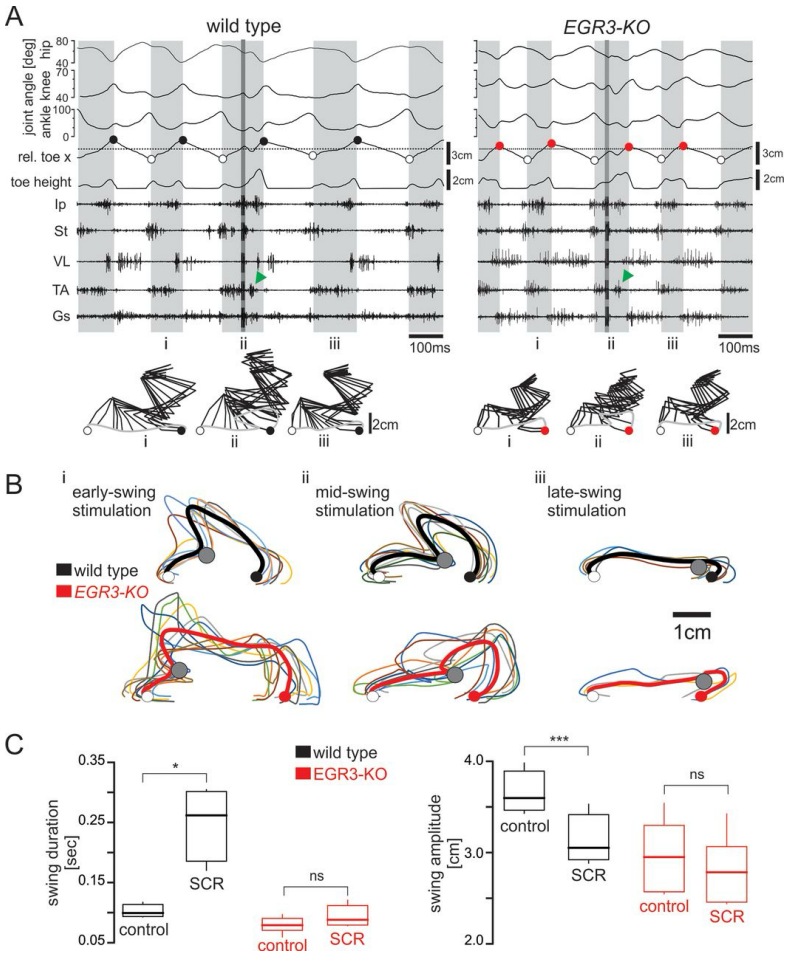

Figure 2-11: The motion paths of the foot of wild mice and laboratory mice, and how this motion is affected by electrical stimulus [16]

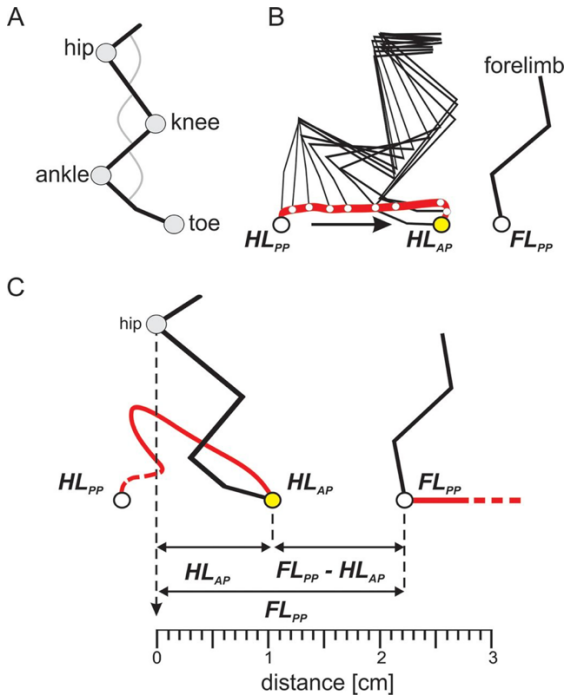

Figure 2-12: Plot showing the motion of the entire hind limb during walking [16]

Similarly, when the gait characteristics of a species of mice is compared to the same species bred for longer limbs, increasing the tibia length from 18.75mm to 21.44mm, some interesting changes were observed. The step length for the larger mice increased, as did the time each limb spent in contact with the ground, however, the swing duration remained consistent between all mice. This difference in stride frequency remained at the same 7-8% throughout all walking speeds. Similarly, there was no difference in gait sequence or relative timing of limb activation between the mice. There is a substantial difference in the limb angles throughout the motion range though, as the walking kinematics of the larger mice are not simply scaled up compared to the smaller mice [17]. This variance is important to keep in mind during the design to ensure that mice of varying walking patterns can be accounted for.

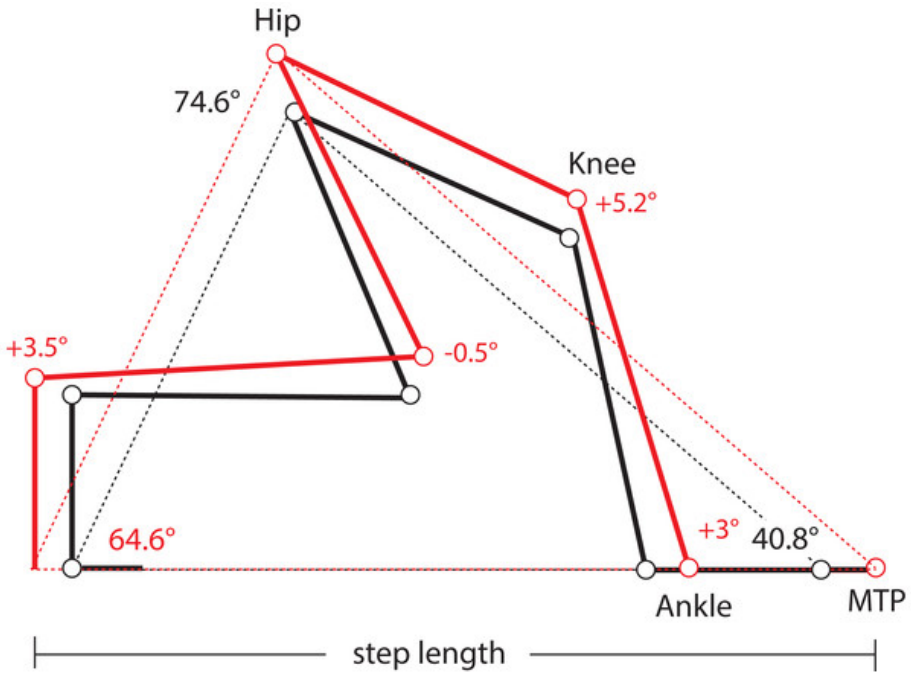

Figure 2-13: Difference in limb size and joint angles from the large vs. small mice [17]

Another study, which set out to create a muscular simulation of the hindlimb in mice, compared the data they gathered on joint deflection and variation to similar data from other studies. While the walking speed of their testing was higher than the other studies, resulting in slight variations in their data compared to others, the results of their experiments show how the data between each of these trials varies [18].

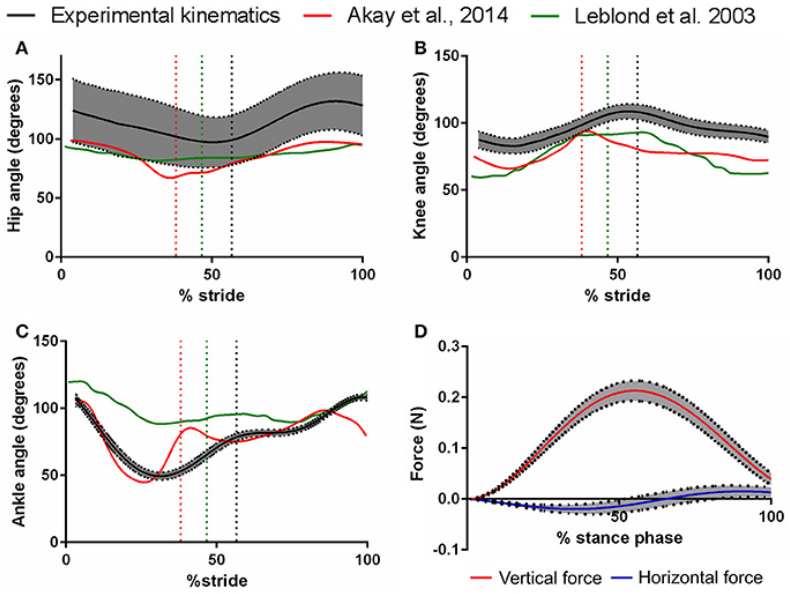

Figure 2-14: Three separate experiments for hindlimb joint deflection [18]

While this particular study investigated the kinematics of a rat instead of a mouse, the data is even more comprehensive compared to the previously mentioned articles, and the variance in results

still prove valuable in understanding how gait varies depending on the particular rodent being investigated. Even in the larger rats, a significant variation between individuals exists, and while it is more present in certain movements than others, the hip flexion, knee flexion, and ankle dorsiflexion, all the movements most important to a kinematic linkage mechanism, experience a large amount of variance throughout the entire gait cycle. Additionally, the article provides a frame by frame drawing of the limb movement, something many other articles fail to include [19].

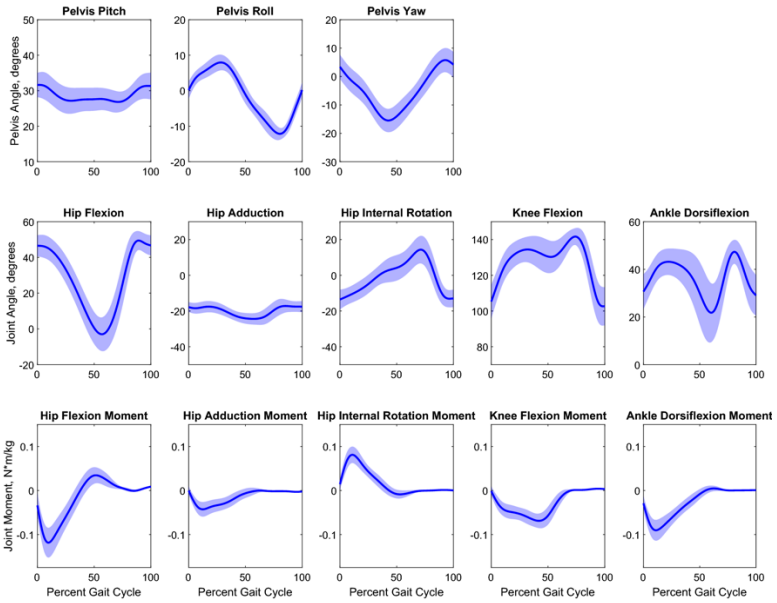

Figure 2-15: Complete kinematic analysis of rat hind limb [19]

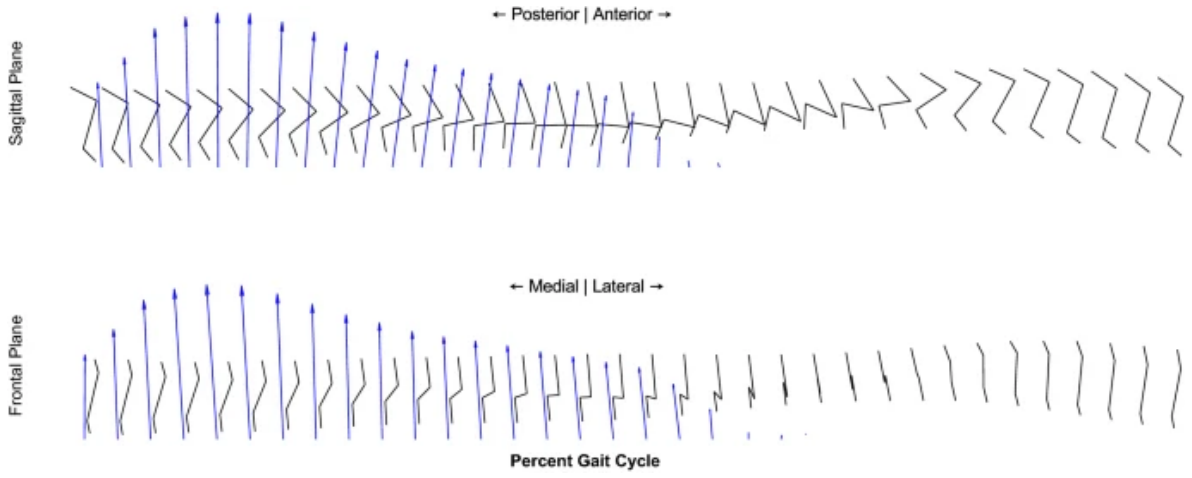

Figure 2-16: Step-by-step simplification of rat hind limb kinematics [19]

2.4.3. Summary of ME Reserach

Thankfully, the research done by the previous year immensely narrowed down the work required of the ME side. Having a basis from which to begin research and begin prototyping proved to be invaluable in making the project progress quickly and smoothly.

The new research done aids in clarifying how to approach the adjustability aspect of the mechanism, since that is one of the items stressed as being lacking from both previous mechanical designs and last years' final product. The results of this research seem to confirm the importance of adjustability, as individual walking kinematics can vary extremely from individual to individual with no way to predict or calculate the expected motion path.

Because of this, the primary focus of the ME aspect of this project will be to build on the lessons learned last year, designing and producing a product that is functional for kinematic testing, and ensuring the mechanism is adjustable to rodents of various sizes and gaits.

3. CS + ME Goal and Divergence Point

Expanding the scope of the project compared to the previous year, work began with an agreement on the goal of the project previously mentioned in the introduction, being a divergence point between the CS and ME teams. This was done to allow the CS team to focus on the machine learning goals of the project and the ME team to focus on improving the previous years' kinematic mechanism.

This goal was for a kinematic rehabilitation mechanism to be used on a mouse, with an accompanying software package to interpret the motion of the mice and quantify their recovery in terms of motor function. The approach to accomplish the goals separately was to approach the problems freely during the divergence yet constraining each other enough to ensure each team's work would support the other and mesh during the eventual reconvergence.

For the CS side of the project, the general direction was to create a robust machine learning model. The next step in the process during the divergence would be to test the model in the lab scenario and make changes if necessary. Then would converge back together with the ME side to implement the machine learning model onto the mechanism.

4. The CS Side of the MQP Work - Machine Learning-Based Image Analysis Based on DeepLabCut Codes for Injured Mouse Recovery Evaluation

4.1. Previous Years' Work

The 2022-23 team had made little progress on the CS side of things since the earlier team focused on the Arduino code used by the mechanism of the previous year's ME side. Most of the developments towards the goal of this year were made by the Indiana University team researching what processes and methodologies that a future computer science major could research.

The Indiana University team had done some surface level investigation into a tracker-less neural network training program called "DeepLabCut". Their investigation into how to use the tool to train a neural network model that could detect and track points on an image at a high-level.

The CS part of this MQP work will be described below and organized in the sequence of goals, requirements, specifications, a few iterations of different approaches to satisfy the goal, results, and key points that would be changed in future iterations and a brief conclusion.

4.2. Goals, Requirements, and Design Specifications

The high-level goal of the CS side is to classify whether a mouse has spinal cord injury or not. The secondary goal to during the convergence of the CS and ME sides was to include a live analysis of the movement pattern of a specific mouse using the trackers of the neural network model. With the analysis the movement pattern could be used to adjust different parameters in the mechanism. Additionally, the movement pattern could be used as a comparison point against the healthy mice to judge their recovery process with the mechanism and note any possible improvements.

While the high-level goal of the CS side is broad, it was broken down into further goals as complications arose during the project with major discoveries and refinements along the way. Requirements for completion include:

1. A neural network model to track a mouse.
    a. The neural network was specified to hand movements of mice since seeing and analyzing the locomotive process as well as all the limbs of a mouse can be quite difficult and beyond the limitations of a single person.
    b. A two-camera system was also used to observe two different perspectives of the mouse, therefore two different models had to be trained. Then using a 3D calibration and triangulation system, a 3D neural network model of the hand movement of the mouse.
2. A data refinement pipeline to extract the raw tracker data from the model and process into a movement pattern that can be processed and implemented by the ME side.

4.3. First Iteration

4.3.1. Setup

The first neural network that was created encompassed data points for the whole mouse body's skeletal structure. This was done since the assumption was that the videos were to be eventually of the entire mouse's body, and making a model where all the different points of the mouse could be tracked might save time in the future. A general skeleton model was drawn to visualize the skeletal structure of the mouse.

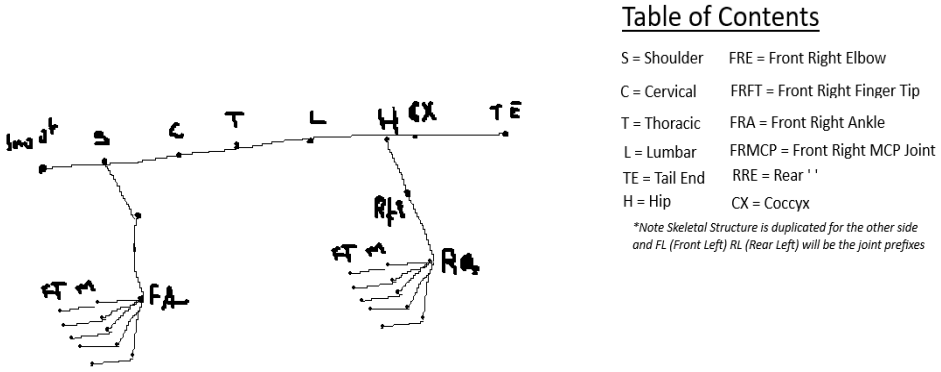

Figure 4-1: Joints to be tracked by the first neural network with a connection in a skeletal structured pattern.

```
skeleton:
- - - - - Snout
- - - - - Shoulder
  - - - - FrontRightElbow
    - - - FrontRightAnkle
      - - FrontRightMCP1
        - FrontRightFingerTip1
      - - FrontRightMCP2
        - FrontRightFingerTip2
      - - FrontRightMCP3
        - FrontRightFingerTip3
      - - FrontRightMCP4
        - FrontRightFingerTip4
      - - FrontRightMCP5
        - FrontRightFingerTip5
```

Figure 4-2: Code Representation of skeletal structure up to the front right limb joints (All 4 limbs and spine joints were included, 55 total joints)

Only one video was entered and split into multiple frames using k means clustering with thirty-nine frames to detect major changes between frames. All visible joints were labeled manually, and the x and y coordinates of the labels were recorded into a data table. The training to testing

split was 95 percent training data and 5 percent testing data for the dataset with a resnet 50 network architecture. In this case since all the frames were labeled, the testing data set would consist of random frames that were not included in the mean value clustering. The network was then trained for 100,000 iterations adjusting the parameters over each iteration. The DLC training parameters such as the alpha, and loss ratio. The total time to train took around 12 hours.

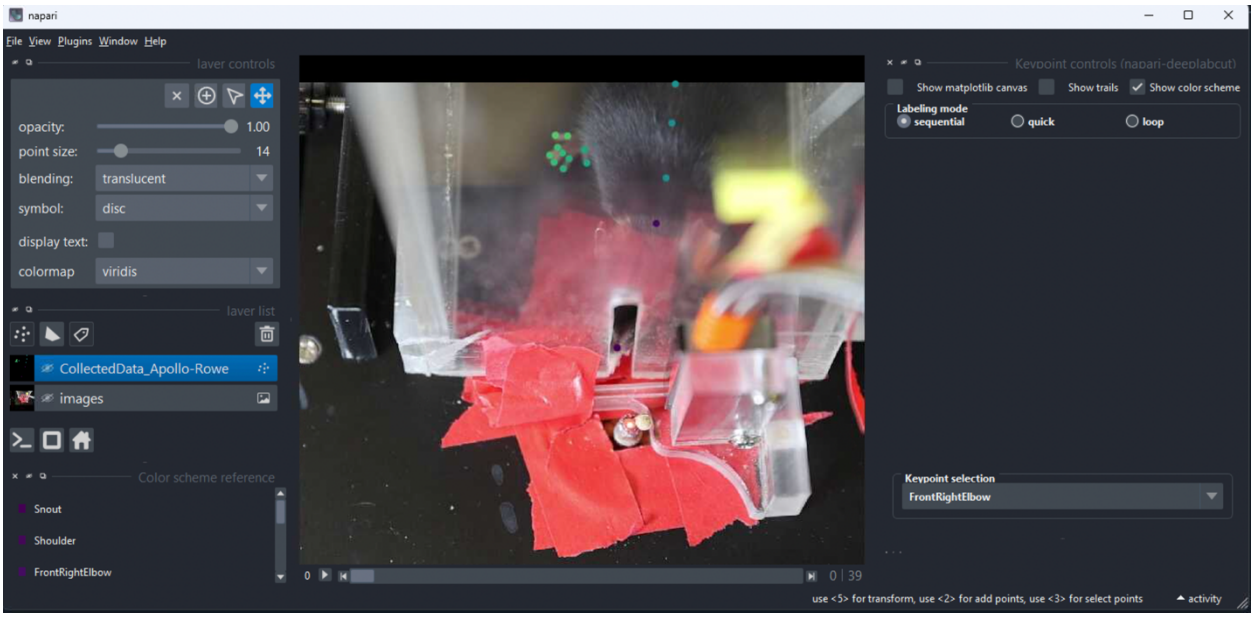

Figure 4-3: Example of the first labeled frame in the labeling software (Napari – A Plugin for DeepLabCut)

4.3.2. Results

Since the model was primitive and had low error values the reasoning for the results is that the train and test errors should be the same since none of the frames were left unlabeled. This is reflected in the p-cutoff error since the values are relatively the same (being 2.49% for train and 2.36% for test). The formula for the p-cutoff for train and test is how far off was the average prediction subtracted from the actual value in the form of a percentage based on 10000 guesses for both the testing dataset and the training dataset. Since this model was just a test to get a feel for the DeepLabCut program no further testing was done. Additionally, there were no further videos in the lab setup to test out the model on since the video used in the model was a couple months old.

Training Iterations	Percent of Training Data	Shuffle Number	Training Error (in pixels)	Test Error (in pixels)	P-Cutoff	Train error (as percentage)	Test Error (as percentage)
100,000	95%	1	2.82	5.94	60%	2.49%	2.36%

Figure 4-4: Results of the evaluation of the first model.

4.3.3. Lessons

Since the first iteration was just a testing to get used to the DeepLabCut interface and user experience, the model was crude, due to the low amount of training data, even though it had a low error. Replicating that low error would be crucial in the future for a high acuity model. I also believed that in the moment the wide view of the camera might pose an issue without cropping since the dead space on the sides of the video might hamper the training process and result in a low acuity model.

When trying to run the training process on a non-machine learning optimized training system significantly increases the time to train. Since the initial environment used to train the model had a compute power acceleration of 0, training 1000 iterations took one hour. However, training in an environment with a computer power acceleration of 7.5 decided not to wait 100 hours (about 4 days) to finish training took only 12 hours to complete. Also, there was an option of training on a computer cluster hosted by WPI investigated for the second model. The computer cluster would allow multiple models to be trained and re-trained at the same time, allowing better streamlining of the training process.

4.4. Second Iteration

4.4.1. Setup

The second iteration was setup using the compute cluster hosted by WPI. This was done to offload the processing so that the local environment wouldn't be blocked during regular hours due to the training of a network having a high demand of compute resources. The compute cluster was hosted in a Linux environment only usable from a command line interface. Multiple scripts created to help facilitate and to run the required code from DeepLabCut, however the GPU acceleration was not used to the fullest potential due to issues with the drivers being recognized by TensorFlow.

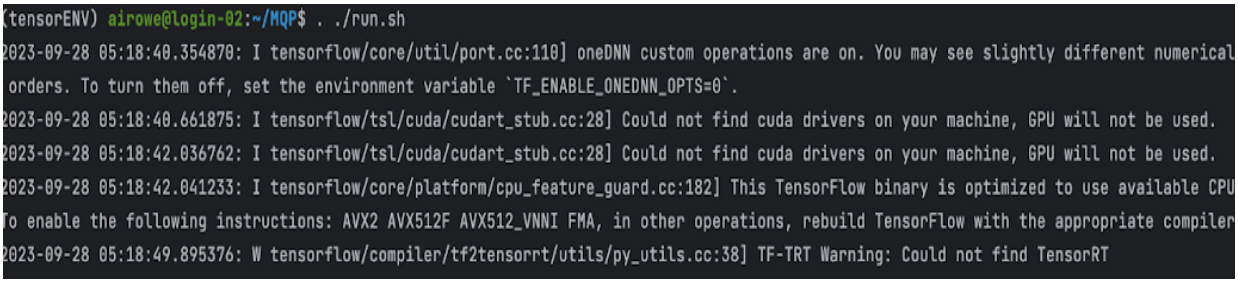

Figure 4-5: Output of the cluster failing to utilize GPU acceleration.

The two primary scripts that were used to run a certain step of the DeepLabCut workflow such as the creation of a training dataset or running the training of the model. Steps like the labeling of frames could not be done in the environment of the compute cluster since there is no graphical user interface. A script for was created for initializing the variables to be used by the python script and for activating the python virtual environment. The virtual environment had all the dependency libraries for DeepLabCut such as TensorFlow. The shell script file for this iteration only ran the python program [Appendix B] with the parameter for the amount iterations the model would train for.

```
#!/usr/bin/env bash

if [[ "$OSTYPE" == "linux-gnu"* ]];
then
  module load python/3.9.12/
  module load cuda12.1/blas/
  module load cuda12.1/fft/
  module load cuda12.1/toolkit/
  #SBATCH --job-name='train-model'
  #SBATCH --mem 4G
  #SBATCH -n 2
  #SBATCH -N 1
  #SBATCH --gres=gpu:1
  source tesnorENV/bin/activate
fi

python run.py 100000
```

Figure 4-6: Code of the shell script (sh file) that was used to run this model.

The version of the python program used for this iteration would handle some of the edge cases with the DeepLabCut program then train the model for the specified number of iterations by the run.sh script. This was done since the major time-consuming part of up to this point was the training of the model since training a model for 100,000 iterations would take over 3 days.

```
try:
    deeplabcut.train_network(
        config=config_path,
        maxiters=int(sys.argv[1]),
        saveiters=int(sys.argv[1]) / 10,
        displayiters=display_interval,
    )
# Yaml Fix (path issue most likely)
except FileNotFoundError:
    print('Attempting to fix config.yaml')

    with open(config_path) as fp:
        data = yaml.load(fp)

    updated = False
    for elem in data:
        print(elem)
        if elem['name'] is 'project_path':
            elem['value'] = project_path
            updated = True

    if not updated:
        print('No config.yaml changes aborting second run attempt')
        exit()

    print('Network Train Attempt 2')
    try:
        start = time.time();
        deeplabcut.train_network(
            config=config_path,
            maxiters=int(sys.argv[1]),
            saveiters=int(sys.argv[1]) / 10,
            displayiters=display_interval,
        )
        end = time.time()
    except:
        print('Unable to train network')
        exit()

print(f'Training completed in {end - start}ms')
```

Figure 4-7: Code snippet of the training function of the run.py file

For the pretraining steps: a set of thirty-four top-down view videos were sent by the Indiana University team and were split into frames with an average of nineteen frames per video using the k means clustering algorithm. The skeletal structure was trimmed down to only include the hand of the mouse since the videos were top down of the mouse's hand doing a grabbing motion of food.

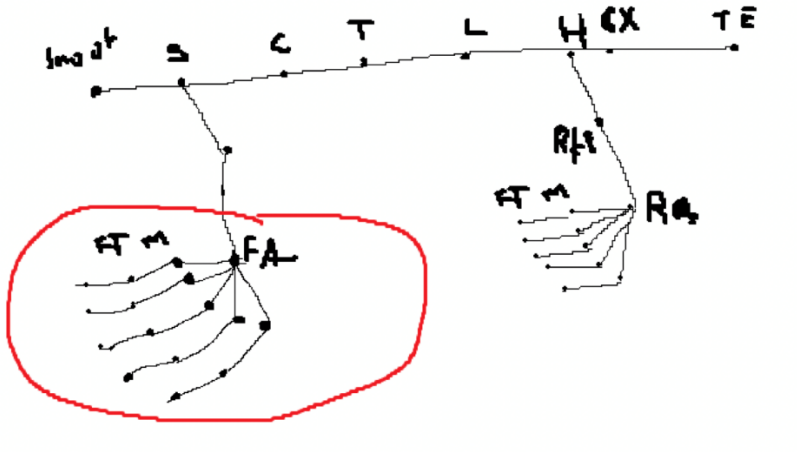

Figure 4-8: Joints circled in red are the joints to be tracked by the second model.

The videos were uncropped and all possible joints of the hand up until the elbow joint were labeled with finger 1 being the pinky finger and finger 5 being the thumb finger to be consistent across the right and left hands. The training dataset had a resnet_50 model architecture with a 95 – 5 training to testing split, however as like the previous iteration all possible frames were labeled so the testing data were randomly selected frames not included by the k means clustering. The training of the model was done twice in two different experiments, once for 100,000 iterations and another for 10,000 iterations on the cluster. This was done to find the sweat spot for iterations to find the point at which the acuity of the model no longer increased significantly. That point would be a difference of 1% or less.

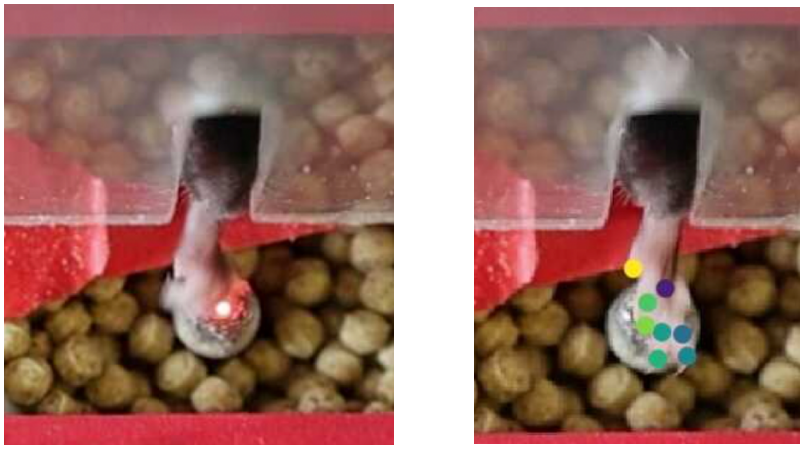

Figure 4-9: Cropped Example of the hand (without vs with labels) during a grabbing motion. (Dots represent the joint son the mouse hand)

4.4.2. Results

The model training that ran for 100,000 iterations on the cluster did not complete since there was a hard limit on the amount of time that a single program that can run for on the cluster (24 hours). The 10,000 iterations took about 20 hours to complete.

```
2023-10-06 04:10:18 iteration: 1000 loss: 0.0200 lr: 0.005
2023-10-06 06:29:59 iteration: 2000 loss: 0.0105 lr: 0.005
2023-10-06 08:52:05 iteration: 3000 loss: 0.0085 lr: 0.005
2023-10-06 11:11:49 iteration: 4000 loss: 0.0075 lr: 0.005
2023-10-06 13:33:22 iteration: 5000 loss: 0.0066 lr: 0.005
2023-10-06 15:55:49 iteration: 6000 loss: 0.0061 lr: 0.005
2023-10-06 18:14:33 iteration: 7000 loss: 0.0056 lr: 0.005
2023-10-06 20:34:15 iteration: 8000 loss: 0.0054 lr: 0.005
2023-10-06 22:54:26 iteration: 9000 loss: 0.0050 lr: 0.005
2023-10-07 01:18:02 iteration: 10000 loss: 0.0048 lr: 0.005
```

Figure 4-10: Log of the training step running for 10,000 iterations.

The model had a low accuracy due to a lot of parameters that I thought of. The main parameter was the number of iterations that the model was trained for. 10,000 iterations were not nearly enough iterations to lower the error enough passed the p-cutoff (60%) and was reflected in the train error being 5 times higher than the first iteration with 100,000 iterations.

Training Iterations	Percent of Training Data	Shuffle Number	Training Error (in pixels)	Test Error (in pixels)
10,000	95%	1	11.04	10.98

Figure 4-11: Evaluation of the model from the second iteration.

Another reason was due to the noise of the background of the images. With the subject of the frame being a small portion of each frame, the background could cause the model to not train efficiently. It also became an issue causing mistakes during the labeling process due to the subject being very small.

Figure 4-12: Full size frame of a picture from the labeling set with the subject circled in red.

Due to the high error, none of the videos could be tested and relabeled according to the p-cutoff since all joint's probabilities were less than 50% (p-cutoff was 60%). Lowering the p-cutoff to fit the data made the model produce results with high error.

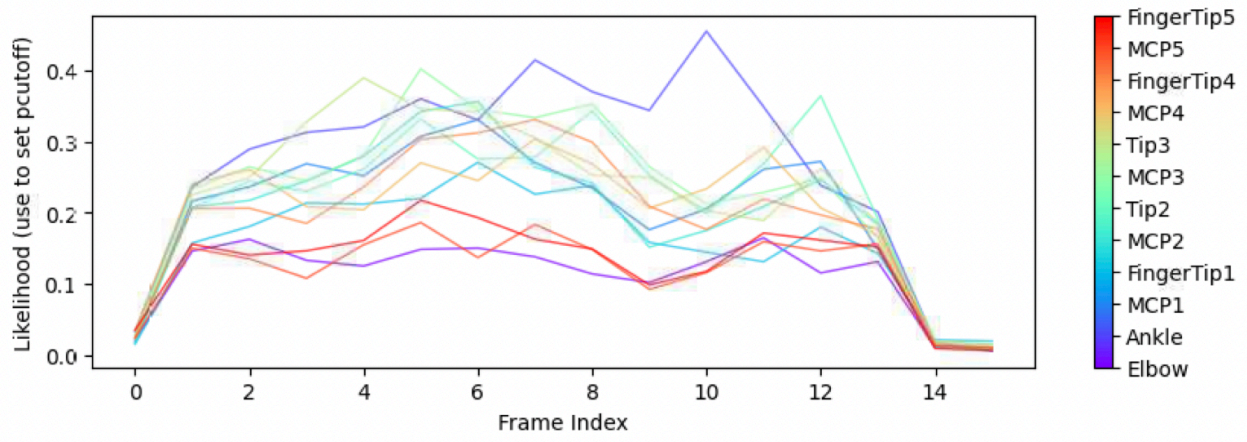

Figure 4-13: Joint probability graph for a video from the training dataset.

4.4.3. Lessons

The model had low acuity and would not be usable to identify a mouse. Several lessons could be adapted from these iterations such as the low amount of training iterations needed to be resolved

either by adjusting the compute cluster environment or using the local environment to train the model. Additionally cropping could solve the issue of the background becoming noise in a dataset where every frame has the same background as well as upscaling the resolution of each frame could allow the labeling process to be easier for each image. All these lessons were adapted to be included in the third iteration.

4.5. Third Iteration

4.5.1. Setup

In accordance with the previous iteration's lessons, two new preprocessing techniques were adapted for this iteration. This was in addition to running the model for more iterations locally. The first preprocessing technique is cropping the frames. Each video is recorded at 1080p, having a resolution of 1920x1080 (1920 pixels wide by 1080 pixels high). The crop would then be manually determined by looking at each set of videos and calculating the area the subject would reside in during the whole video. The crop is defined as a rectangle with x1, x2, y1, y2 parameters.

```
video_sets:
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining0.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining1.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining2.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining3.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining4.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining5.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining6.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining7.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining8.mp4:
    crop: 800, 1200, 350, 700
  E:\MQP\MQP-Apollo-Rowe-2023-11-25-3d/videos/sft4toptraining9.mp4:
    crop: 800, 1200, 350, 700
```

Figure 4-14: Crops of the top view videos.

The other preprocessing technique was image upscaling using an upscaling program called nunif. This program used another machine learning model to fill in pixels and upscale an image from native resolution to 2x or 4x native resolution. A virtual environment was also created to host all the package dependencies and an accompanying script file was used to create the batches of upscaled images [Appendix D]. The group decided to upscale the image 4x native resolution with

a noise level of 0. The noise level, which is on a scale of 0 to 4, is the blurriness of the low-resolution objects.

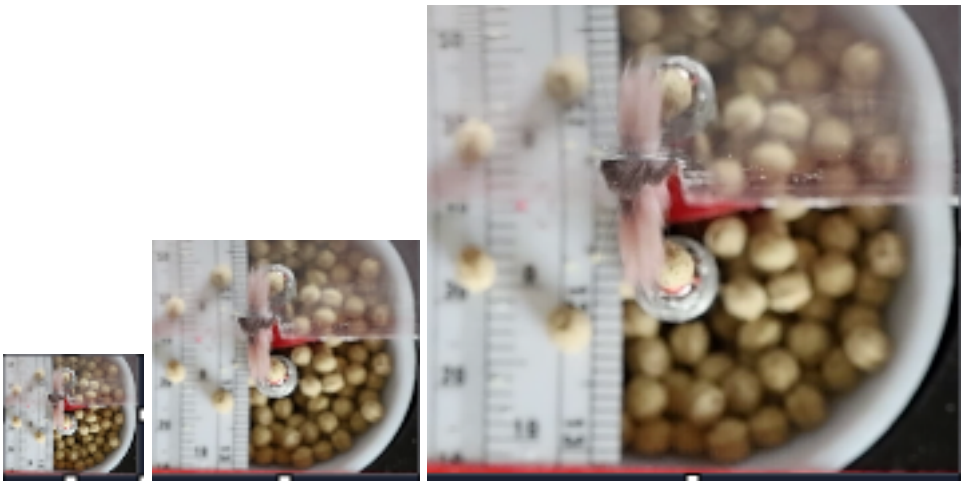

Figure 4-15: The same subjects at 1x scale, 2x scale and, 4x scale. 2x and 4x are with noise level 0 meaning the background has no noise.

The videos provided for this iteration were also different with the ability to have a 3D triangulation with a side point of view camera and a top point of view camera. Both side and top views were split into frames by the k means algorithm with an average of 19 frames per video. Frames for both side and top views were all labeled for all visible joints according to the second iteration's skeletal structure. The training to testing data split was 95 - 5 again. Both the side and top view frames were used in the training dataset. Another test for this iteration was to see the difference that a network architecture change would make for time vs accuracy. The model architectures that were tested were the resnet_50 architecture vs the mobilenet_v1 architecture. Both datasets were trained for 100,000 iterations with the mobilenet_v1 taking 4 hours and the resnet_50 dataset taking 9-10 hours.

Figure 4-16: Same frame of the mouse from the side and top views (without labels)

4.5.2. Results

The mobilenet_v1 architecture network had a higher error and a lower p-cutoff (40%) due to the increase in error. Due to the increased speed the cost of accuracy was too high. Thus, the model that was used for this iteration was the model with the resnet_50 architecture. However, to save in efficiency a shuffling of the training data was done and a sequential training step with a resnet architecture was done.

Training Iterations	Percent of Training Data	Shuffle Number	Training Error (in pixels)	Test Error (in pixels)	P-Cutoff	Train error (as percentage)	Test Error (as percentage)
100,000	95%	1	4.77	21.4	40%	3.86%	21.07%
100,000	95%	2	4.59	38.75	40%	4.2%	25.92%

Figure 4-17: Evaluation of the mobilenet_v1 and resnet_50 architecture model in sequential shuffles.

The assumption for the high testing error for the first model run was due to the mobilenet_v1 architecture and its' handling of testing data. Additionally, it is possible due to the training vs testing splits. For the second shuffle of the model changing the network architecture could also affect the testing error with higher testing errors for the resnet_50 architecture with similar training errors.

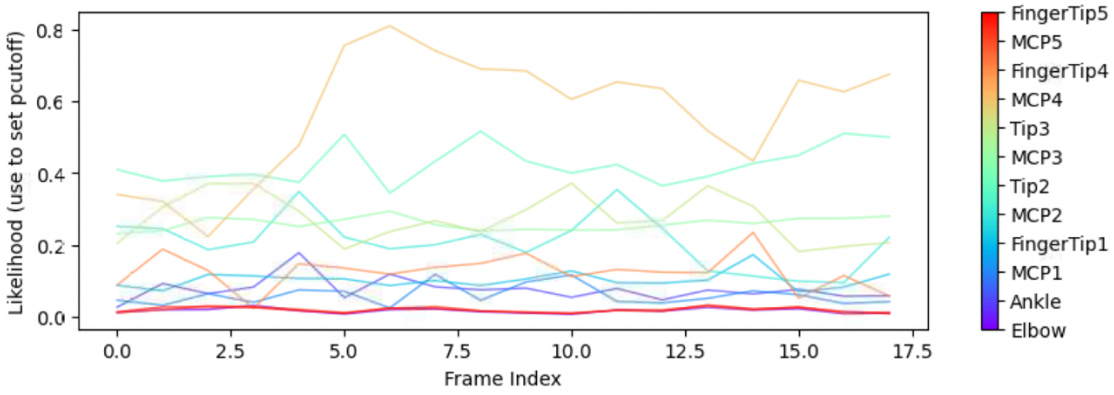

Figure 4-18: Joint probability graph for the side view using the mobilenet_v1 model.

Only one or two joints would be able to be tracked at a time due to the nature of the mouse movement as well as the joints looking similar. With the p-cutoff set at 40% it filters out a lot of the noise with the model, however only one or two of the data points would be tracked for a certain video.

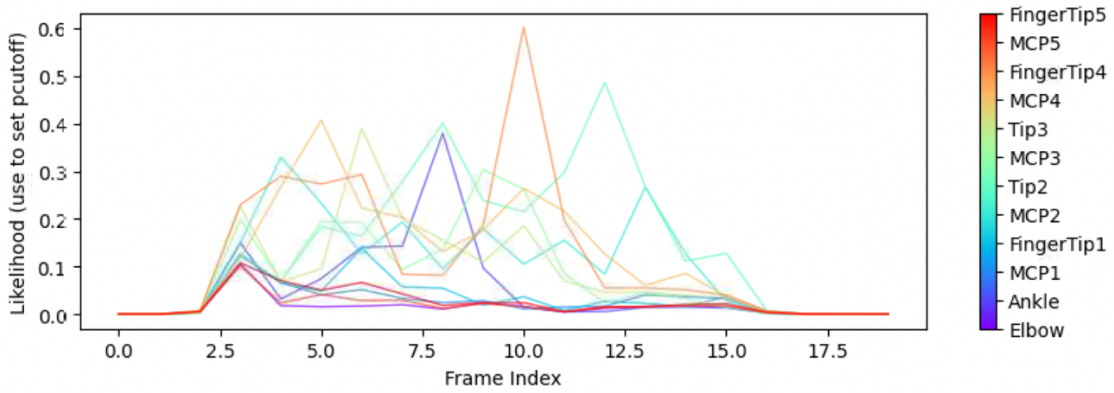

Figure 4-19: Joint probability graph for the top view of the same video using the resnet_50 model.

Additionally with the different architecture of the resnet_50 model the accuracy is slightly higher with different peaks when the different body parts are visible from the top view. Unfortunately for both cases the model was not accurate enough to track any of the body parts of the mouse on playback of the videos.

4.5.3. Lessons

The major lesson from this iteration was that preprocessing techniques and network architecture are crucial to increasing the accuracy of the model. Resnet_50 would be the best architecture to accomplish the goal of this project. Additionally, separating the side and top views of the model will also be beneficial since the cameras only capture two angles with the same background, variety in the background will muddy the dataset and confuse the model. Also, increasing the testing dataset size might also decrease the testing dataset error. Potentially, increasing the dataset size might also lower the error.

4.6. Fourth Iteration

4.6.1. Setup

The final iteration would be using a new set of videos from the Indiana University team this time focused on using the 3D triangulation method of DeepLabCut to analyze the mouse in three dimensions in accordance with the mechanism. The videos of the top and side view both have a cube visible in frame with a checkerboard print aimed at calibrating the camera's by determining the edges on which the camera's view intersect.

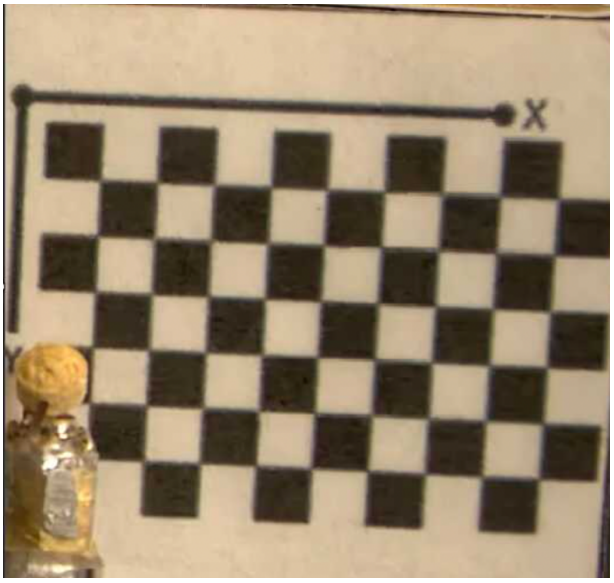

Figure 4-20: Checkerboard print visible in the sideview camera.

There were 60 side and top view videos in the new dataset that were split into a 75 – 25 training to testing dataset split. Additionally, some videos were left unlabeled to help facilitate the testing dataset.

Using the same preprocessing techniques as the third iteration, the video crops were manually and individually determined for both the side and top views. The cropped videos were then split into frames by the k means algorithm and upscaled. The upscaled images for the side view were then labeled and used to create a dataset with the resnet_50 architecture. The side view was then trained for 100,000 iterations taking approximately 9-10 hours.

A new method of adding additional training data to the pool was testing this iteration, using more videos in a different environment, and adding them to the dataset with the same preprocessing techniques.

Figure 4-21: Frames from sideview videos in different environments (left frame was flipped across the y axis and labeled)

4.6.2. Results

The model for this iteration had the highest accuracy out of all models with the lowest training and testing errors. The second shuffle lowered the testing errors and slightly increased the training errors due to the inclusion of additional data in a different environment and training for more shuffles.

Training Iterations	Percent of Training Data	Shuffle Number	Training Error (in pixels)	Test Error (in pixels)	P-Cutoff	Train error (as percentage)	Test Error (as percentage)
100,000	75%	1	3.53	8.52	40%	3.42%	7.66%
100,000	75%	2	3.93	7.05	40%	3.78%	6.63%

Figure 4-22: Evaluation results from model iteration 4.

Due to the high accuracy of the model the joints could be tracked, and the trajectory plots of the joints could be generated as well. Thus, leading to the successful creation of a machine learning model for tracking a mouse.

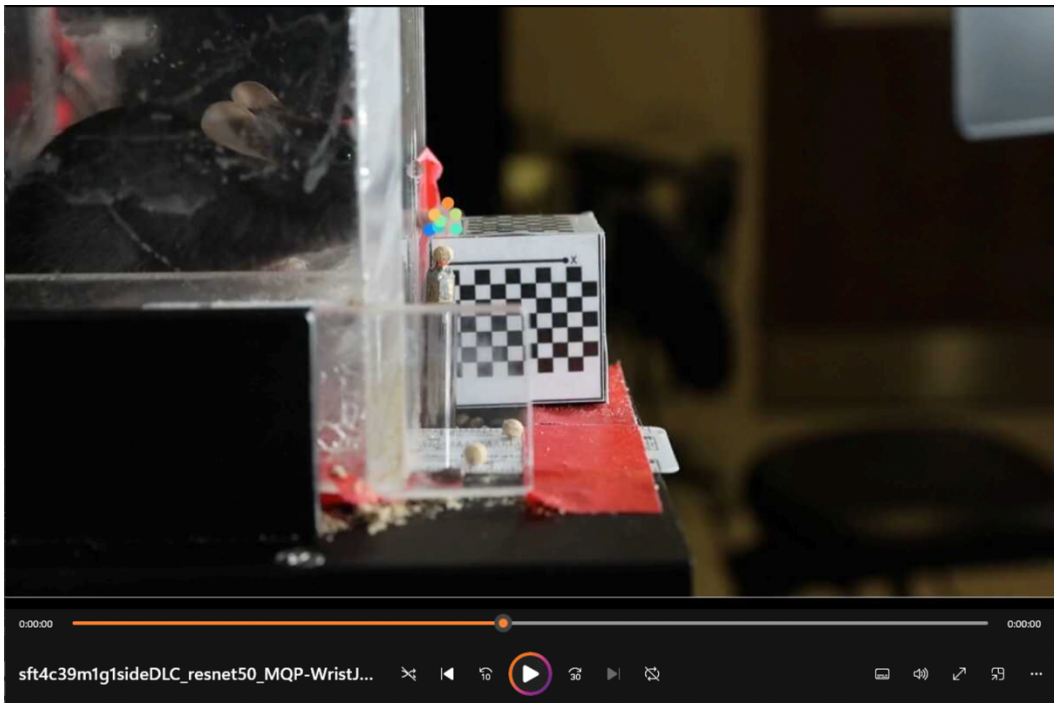

Figure 4-23: Video of the mouse with trackers from the side point of view

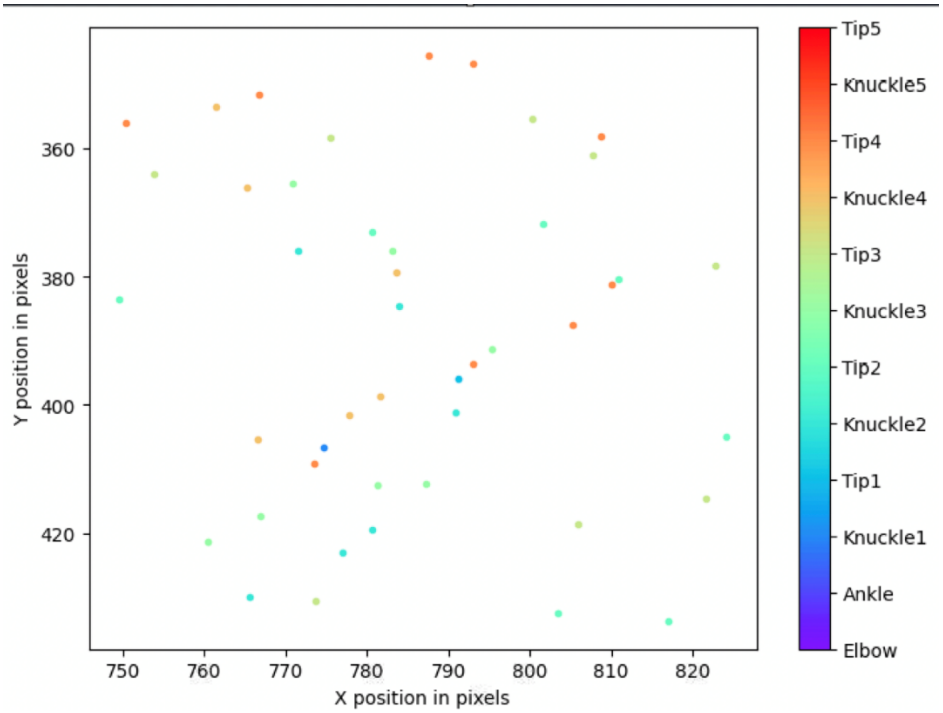

Figure 4-24: Trajectory plot mouse hand movement of the same video

4.6.3. 3D Triangulation

Due to the time constraints of this project the main objective of producing a 3D mapping of the movement of a mouse could not be completed due to iterations that needed to be done for the sake of the DeepLabCut software package. An unobstructed view of the grid is needed for the calibration of the camera, however in most images the grid is partially obstructed. Additionally, the same grid must be viewable from both the side and top points of view. New videos must have been created with unobstructed views from the Indiana University Team, however those videos could not have been created in enough time.

Figure 4-25: DeepLabCut triangulation example versus an example of a side view calibration image.

An attempt to block out the obstructed grid and orient the grid so that they would align was made to no avail since the grid must be the same from both camera views.

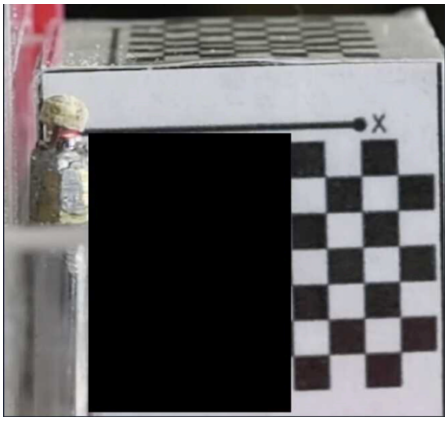

Figure 4-26: Previous calibration image with blocked out grid.

4.7. Application

Additionally, in hopes of a convergence with the ME side, an application was made to facilitate the usage of the machine learning model created in the fourth iteration. This application had two main features, a file input mode, and a live video feed mode. The file input mode would take in a file, run the model, and display a trajectory plot. The live video feed mode would display the camera input, analyze it, display the trackers on the feed, and display a trajectory plot too. Unfortunately, as well due to the inability of the Indiana University team to be able to test the application, no results were taken away. [Appendix E]

4.8. Summary and Recommendation

After several iterations on the fabrication of the machine learning model, fixing mistakes, and crafting a robust model. The process to create a machine learning model to track a mouse with and without spinal cord was successful and a substantial iteration from the previous year's accomplishments.

Unfortunately, due to time constraints of the Indiana University Further completion of the goal of a 3D tracking machine learning model and iterations on the application will not be possible.

Continuation of the iterations of this project could see a full-scale in-depth model with the ability to track a mouse and analyze several factors related to spinal cord injury.

5. The ME Side of the MQP Work - Design, Prototyping, and Testing of Kinematic Linkage System for Assisted Mouse Movement and Recovery

5.1. Previous Years' Work

The 2022-2023 team had the same overall goal as this years' team; to investigate the impact of a kinematic rehabilitation mechanism to aid neuroscientists' investigation into recovery from spinal cord injury. Their final mechanism consisted of a four-bar linkage connected to the midpoint of a mouse's calf with respect to the hip, with motion being driven with a NEMA-8 stepper motor, and the rotational orientation of the linkage being interpreted with a rotary encoder. Anterior/posterior and lateral/medial adjustment is handed by I-beams with sliders and screws to allow for use with mice of various shapes and sizes. The electronics are controlled by an Arduino Uno, with code linking the rotation of the steppers to the position of the encoders to ensure consistent positional accuracy.

This project had several shortcomings with its design and execution, however, and while it did reach many of the goals it set out to achieve, ultimately no experimental data was acquired upon its conclusion.

1. The assembly lacks any way to attach to the mouse, both to the body and to the limb. Verifying the mechanisms function in its current state would be impossible without redesign.
2. The assembly itself is structurally unsound and risks collapse. The structure must be robust to prevent collapse and harm to the mouse, while also being as rigid as possible to maintain accurate and consistent kinetic positioning.
3. The adjustment mechanism does not remain securely in place. If the adjustment slips, this could cause harm to the mouse by changing the limb trajectory in ways unnatural to its natural gait.
4. The mechanism only attaches to a single point on the limb. Control of only a single point creates a lack of specific angular control of the joint, while optimally, the mechanism would attach to at least two points to more precisely control the position and angle of the overall limb.

While not as immediately impactful to the mechanism's function, other areas for potential improvement were noted as well.

1. The gears being used to link the stepper to the encoder were beginning to show wear. A 3D printed gear can be durable, but on such a small scale, this strength is more difficult to ensure.
2. Control of the mechanism requires a computer. Uploading and running an Arduino sketch in a lab environment, while giving total control of the mechanism, might be daunting and out of the realm of knowledge for medical students.
3. The linkage is fixed and only mimics a single fixed gait. While the physical position of the mouse can be adjusted, as well as the position of the attachment point from the linkage to the leg, the trajectory of the linkage itself cannot be adjusted to better match the gait of mice of different sizes or walking characteristics. Additionally, the gait itself cannot be made 'smaller' to ease a paralyzed mouse into full motion from complete immobility.

5.2. Goals, Requirements, and Design Specifications

The goal of the ME side is to use the analyzed walking gait of a mouse to design a linkage that will closely mimic the natural walking motion. From this, a larger mechanism will be designed that will attach mice of different sizes to the linkage, using driven motion to rehabilitate the loss of motor control that results from a spinal cord injury. The greater intention is that this mechanism will aid neuroscientists in the study, understanding, capability, and limitations of how mice and other animals recover from spinal cord injuries.

While the goal remains largely the same as the previous year with similar design requirements, the approach will build off the shortcomings of the previous years' research to improve the design and function of the overall mechanism. Critically, the main objectives will satisfy the following overarching requirements:

1. The gait produced by the linkage is similar to a healthy mouse.
2. The mouse is positioned quadrupedally and not bipedally to more closely mimic the natural walking posture.
3. The mechanism can fit mice of various shapes and sizes.

In addition to these main objectives, the machine must also adhere to the following design specifications. These specifications take some ideas from the previous years' list, stressing the importance of improving specific areas within their design, while additionally building on them with a few new ideas.

1. Items from previous year
    a. The mechanism should provide proper support to the mouse and remain securely attached to the mechanism at all attachment points during motion; e.g. the body and legs.
    b. The mechanism's design should be modular, easy to assemble and disassemble, and able to have replacement parts printed on any readily available 3D printer.
    c. The mechanism must integrate two motor driven linkages to enable gait training for each of the mouse's hind limbs. Each linkage should satisfy the following specifications:
        i. The linkage should have at least one Degree of Freedom to mimic at least one of the healthy gait patterns.
        ii. The crank should be driven by a motor that provides the necessary power to drive the linkage.
        iii. Links should be connected with shoulder screws for reliability and durability of the mechanism.
2. Improvement areas
    a. The mechanism must be sturdy and stand secure with no risk of collapse or tipping.
    b. The transfer of rotation from the motor to the encoder will be durable and not show signs of wear and risk damage.
    c. The mechanism adjustment will hold securely in place and not slip during use.
    d. The electronics must be controlled with a contained unit independent of a computer.
    e. The linkage will be improved in several ways:
        i. The linkage will attach to more than one point on the limb to increase positional control.

ii. The motion path of the linkage will be adjustable to fit mice of varying gait styles and ranges of motion.
iii. The linkage will integrate some flexibility to ensure the gait isn't too constrained within a single plane of motion.

3. New items
    a. The mouse will be easy to attach and detach from the mechanism, minimizing the amount of effort required from the laboratory technician.
    b. The mechanisms must have accurate and consistent positional control to maintain proper kinematic relation between the two linkages.

As further prototypes are created, progress towards accomplishing each of these goals will be attempted, iterated upon, and completed. As many of these goals will be completed as possible to maintain alignment with project expectations.

5.3. Brainstorming

5.3.1. Motion Path of the Joints

Before a new design could be designed and fabricated, a new linkage to fit the updated design requirements was required. The previous year's group had experimented with other mechanisms besides the final rendition which attached to the midpoint of the calf, mainly one which attached to both the knee and the foot, but this design was too complex to be feasible within the given timeline.

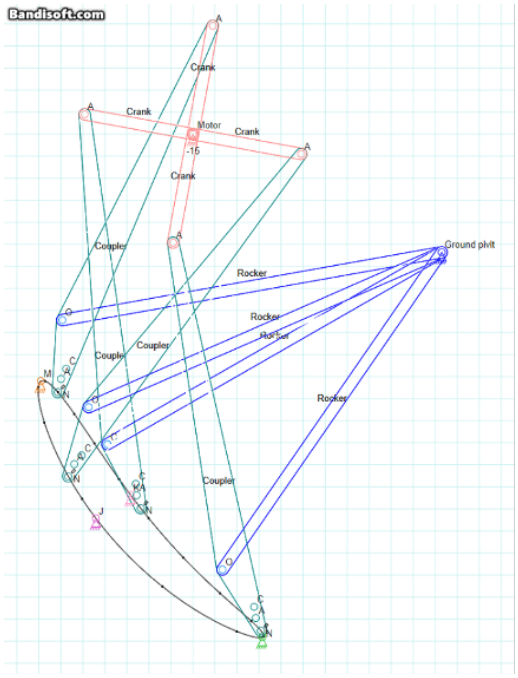

Figure 5-1: Previous years' final linkage design [3]

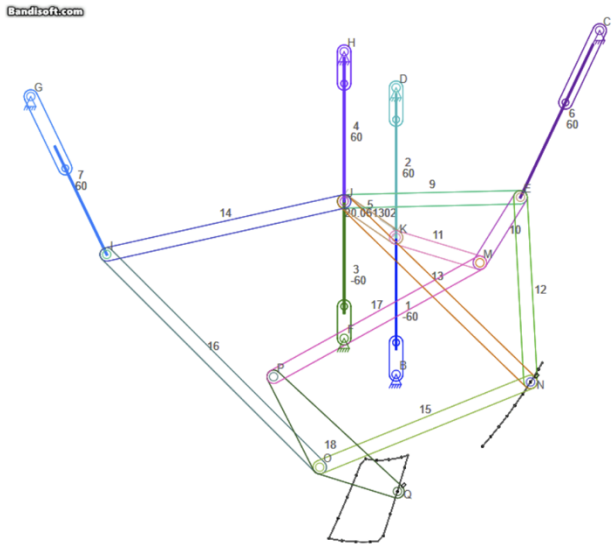

Figure 5-2: Previous years' idea for a more complete but complex linkage [3]

To accomplish this first goal, the trajectory of a rodent limb must first be analyzed, from which an associated linkage will be created. This linkage must be simple enough to function in a compact mechanism and have as few links and joints as possible, yet also follow the respective position and velocity of the target positions as closely as possible.

Using a combination of the previous years' investigation and a few journal articles investigating the location of joints during movement, several approximations were created that followed the movement of the knee, ankle, and foot of various rodents.

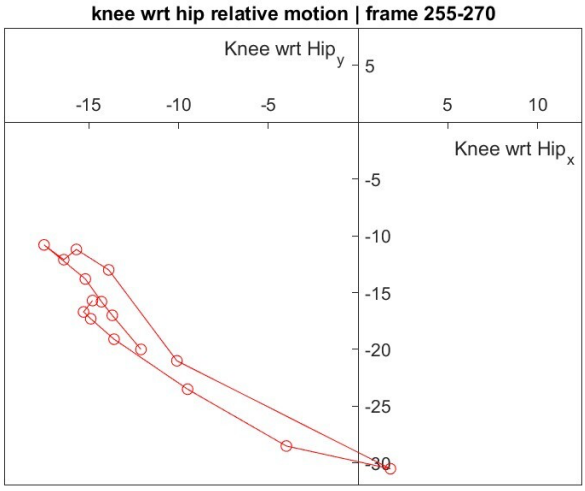

Figure 5-3: Previous years' plotted motion path of the mouse's knee with respect to its hip [3]

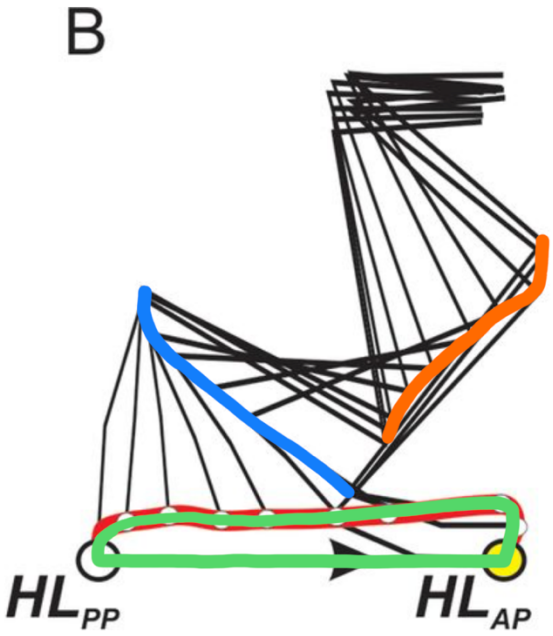

Figure 5-4: Image above overlaid with approximated trajectories for knee, ankle, and foot from image 2-8

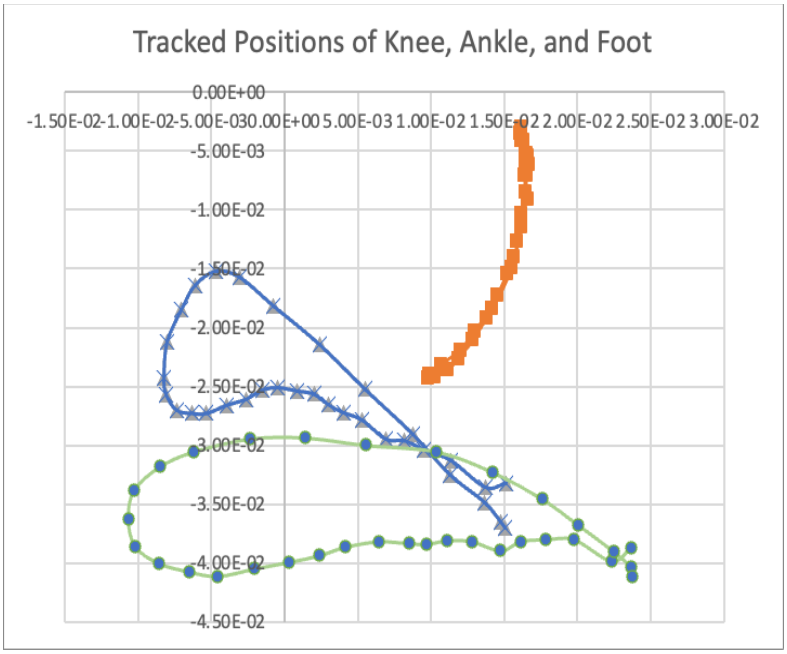

Figure 5-5: Plotted positions of each joint from image 2-12

5.3.2. Initial Linkage Brainstorming

With these images, the was now a more concrete direction to which the project should head. Like the previous year, it was decided that the mechanism must avoid using sliders as they would prove exceedingly complex on a small scale compared to the ease of implementing rotationally controlled joints. Otherwise, the mechanism will attach to two points on the mouses leg to constrain the movement of two joints to a specified path.

A four-bar linkage follows a specified coupler curve that cannot be adjusted without physically altering the mechanism, and while additional links can result in a more complex motion path, they all ultimately follow the same path for each motion cycle. A five-bar linkage, however, uses two motors in tandem to drive a point to a location specified by code, is ideal in achieving unlimited adjustability along a planar surface. As stated in the literature review, a variation of this mechanism is already the model by which some scientists simplify the walking motion of animals [14].

Since a traditional 5 bar linkage only controls the position of a single point, and the ideal outcome of this project attaches to two points on the mouses limb, two of these linkages would need to be placed side by side, with a bar connecting the two to constrain their movement in relation to the other. This would give a stable platform on which one of the limbs could be attached to, and in tandem with the CS side of the project, use an input of size and motion path characteristics to tailor an individualized walking path for each specific mouse. Also, this mechanism's flexibility would allow for the possibility to control each individual limb segment, slowly progressing rehabilitation from the attaching to the calf to attaching to the foot.

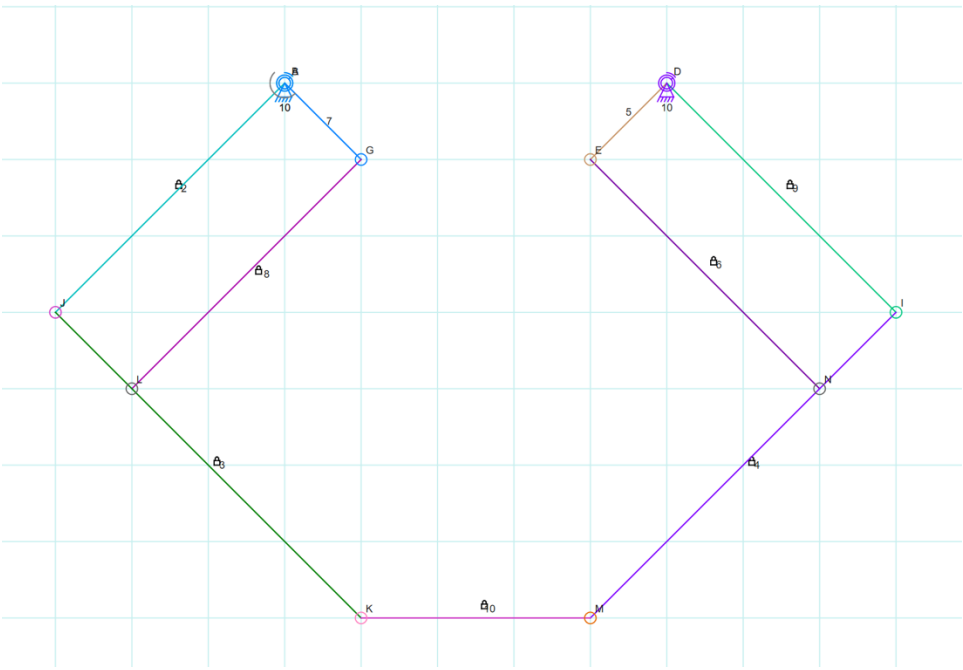

Figure 5-6: Initial idea for dual 5-bar linkage

This approach, however, was decided to be too complex to fit within this project's scope. The synchronization of two separate five-bar linkages, especially on such a small scale, would likely take up a large portion of the available project time, without considering the other required design aspects required of the project. Also, it would cost much more than the allocated budget and shift the project towards being heavily CS focused. To make this heavily ambitious idea more realistic, it was decided to reapproach the problem with a simpler linkage using only a single driven crank, similar to the one used in the previous year's project.

First, however, it was necessary to decide which segment of the limb the linkage would attach to, using image 4-3, 4-4, and 4-5 as references from which to assist in judging the viability of motion paths. While attaching to each end of foot would provide the most complete control over the entire limb, it would also be the most difficult, as both the more complex green and blue motion paths must both be followed. After experimentation with possible designs to fit this criterion, a simple linkage that could generate each of these motions in sync with each other with just a single driven crank quickly fell out of the realm of possibility on such a small scale.

Instead, a linkage that could generate the motion path of just one of the joints would be far simpler. Neither the toe nor the ankle paths themselves are too complex, but this still ignores the vital requirement of attaching to two points. A non-driven dummy leg could be attached to the coupler curve of the driven joint, providing accurate motion of the limb segments above the chosen joint without much additional complexity. This would be ideal to combine with a linkage that emulates the motion of the toe, however, this resulting three-link dummy leg would be non-deterministic. On the other hand, a linkage that emulates the motion of the ankle would only require a two-link dummy leg, making this an ideal candidate on which to base further progress. This linkage also

has the added benefit of being the most closely constrained to the mouse's body, minimizing the risk of potential harm to the mouse by limb hyperextension.

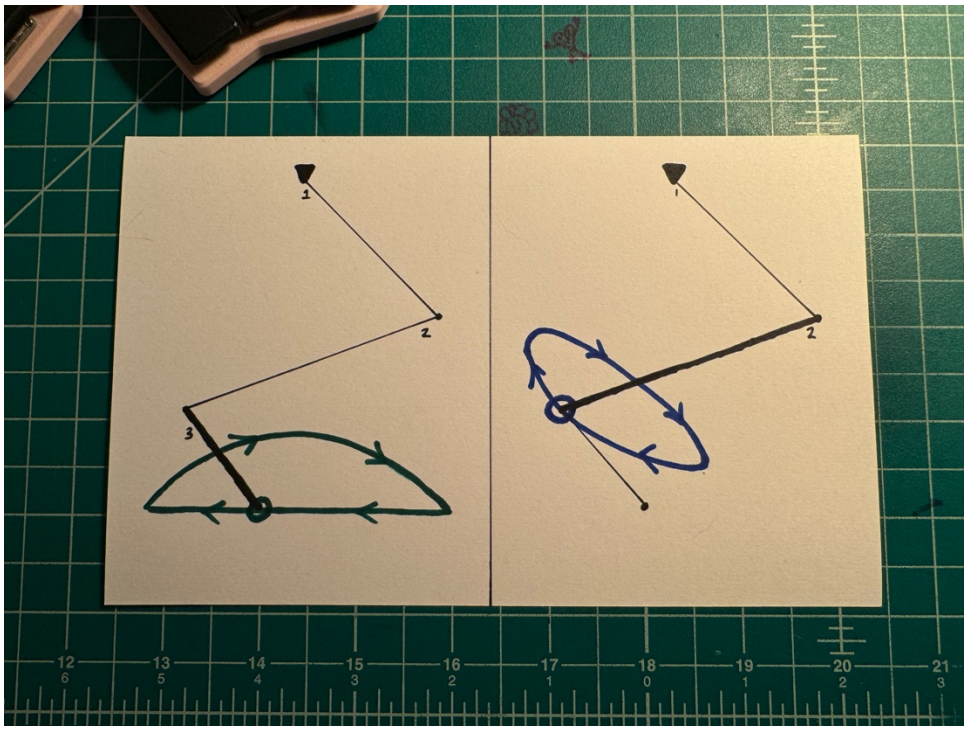

Figure 5-7: Rough sketch of the motion path for the toes and ankles and their associated dummy legs

5.4. First Design Iteration

5.4.1. Mechanical Design

With a general linkage approach in mind using a dummy leg, the mechanism's greater design can now begin to be explored. First, the linkage design must trace the path of the ankle, while also ensuring that there is a generous surplus of space within the linkage to allow for the placement of the dummy leg. Additionally, the leg's motion should have a consistent velocity along the entire motion range without any sharp accelerations or decelerations, both to mimic the natural motion and prevent injury. This three-bar linkage would extend the coupler far off the axis that connects the crank to the rocker to strike a balance between steady ankle velocity, a wide enough and fairly accurate coupler curve, and space for the dummy leg.

Through much trial and error, an initial linkage that satisfied all these design requirements was found. This design used a short crank with a longer offset rocker and a large triangular coupler, the far point of the triangle being the ankle. From this, a dummy leg was attached to the ankle and was adjusted until the knee coupler curve closely matched the ones previously generated, making sure no interference was generated between the dummy leg and the crank or rocker.

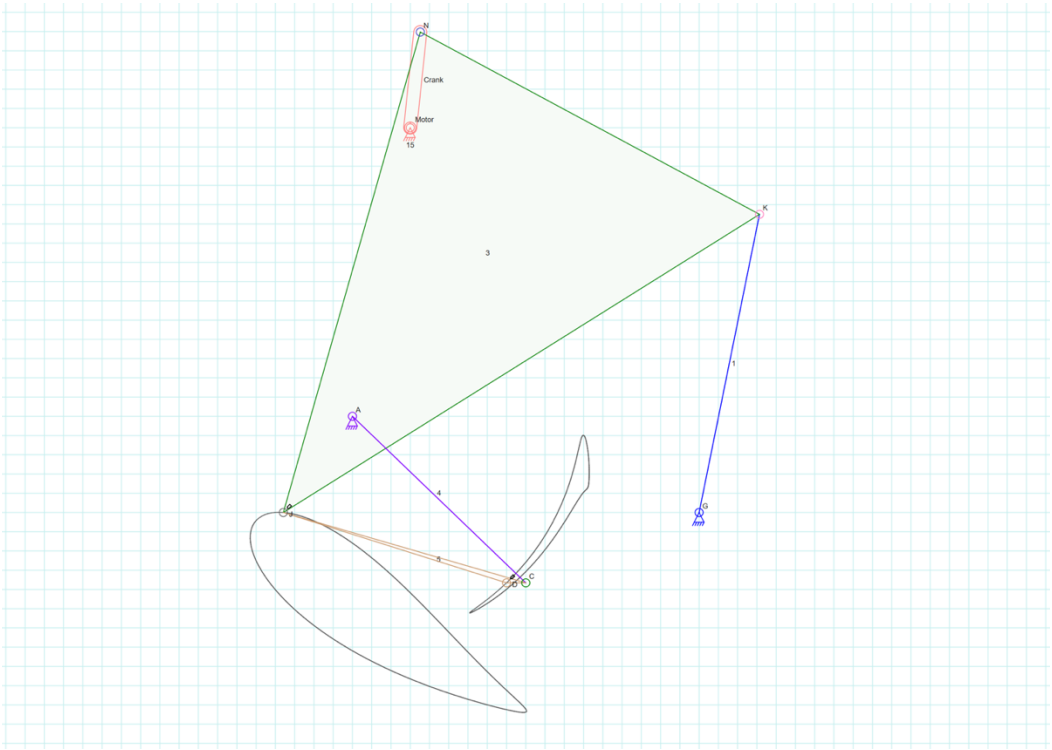

Figure 5-8: Initial linkage design with the corresponding coupler curves of both the knee and ankle

As seen in the image, the top orange link is the driven crank, the right blue link is the rocker, and the large green triangle is the coupler, with the far bottom point acting as the ankle. The purple link is the upper dummy leg while the yellow link is the lower dummy leg, with a slightly offset point on the yellow link near their joint acting as the knee. Each of the black traces are the coupler curves of both the ankle and the knee. Comparing these black traces to previously gathered joint

motion data shows that the motion from this linkage is fairly accurate, while additionally, the linkage successfully left room for the dummy leg and showed fluid motion, fulfilling each of the linkage design requirements.

From this initial linkage, before an overall mechanism could be designed, the linkage itself had to be verified physically. Each of the three ground joints were bound to a central plane, and the large triangular coupler was instead designed to be a wide arcing piece, allowing the linkage to function in fewer overlapping layers and removing all worries of interference with the dummy leg. A combination of 3mm and 6mm M2 shoulder screws were used at all the joints, each of them being secured into an M2 nut, and all parts being 3D printed.

https://www.mcmaster.com/products/shoulder-screws/metric-alloy-steel-shoulder-screws/?s=m2+shoulder+screw

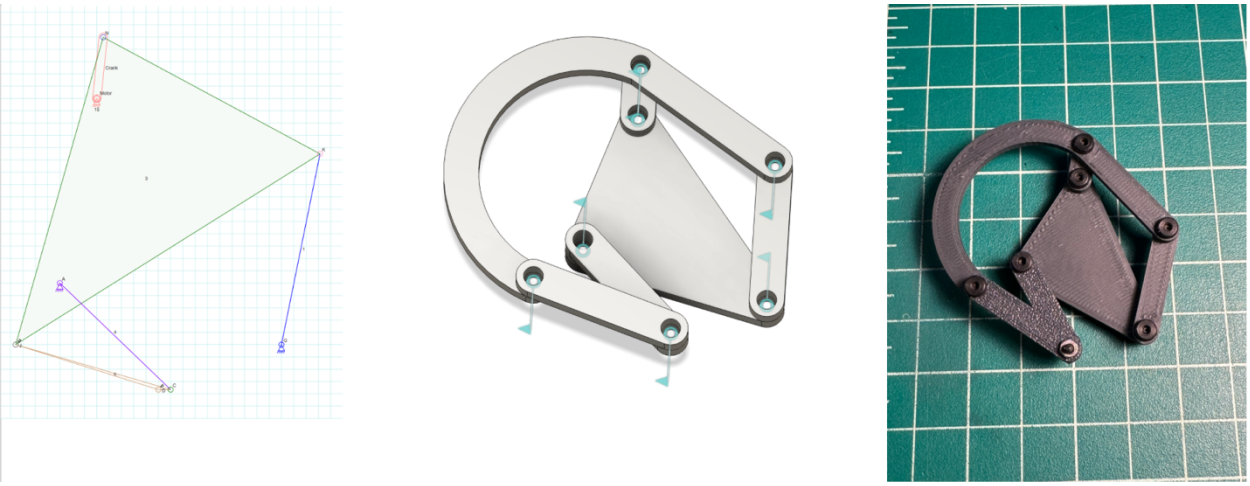

Figure 5-9: Progression showing linkage design, CAD model, and 3D print

Though this model didn't complete a full rotation due to unplanned interferences, it did verify the function of the linkage and the use of 3D printing as a manufacturing method. Now, this linkage could be implemented into an encompassing mechanism, the first step of which was changing the link between the stepper and the encoder from gears to a belt.

The previous year used a 28 tooth gear on the stepper and a 24 tooth gear on the encoder with a smaller gear connecting the two, causing them to both spin in the same direction. Because of this, a similar result could simply be achieved by using a 28 tooth pulley on the stepper and a 24 tooth pulley on the encoder, connected by a small GT2 belt. While this change is not completely necessary to the function of the mechanism, the change from small gears to a pulley would provide a more durable connection, especially during the long continuous operation expected while the mouse undergoes kinematic rehabilitation.

It was also decided to use the same stepper motors and encoders as the previous year. This would save on hardware costs, save time in researching and waiting for newly ordered parts to arrive, and save time in troubleshooting and programming the code to implement this hardware. Specifically, this is a BLANK BLANK stepper motor and a BLANK BLANK encoder.

The initial plan was to use laser cut steel front and back plates on which to secure the other components, as it would be rigid and easy to manufacture. The plates would sandwich a 1515 aluminum extrusion for its combination of availability, rigidity, and the slots which would allow for simple sliding adjustment. The stepper, with four mounting screws, would be secured in place, while the encoder, with only two mounting screws, would be allowed to pivot for adjusting the belt tension.

A parametric GT2 pulley design was used to generate the basic design for each of the pulleys, which were then modified to fit the overall design.

https://grabcad.com/library/parametric-gt2-pulley-1

Each pulley extended through the inner plate to hold it securely in place during motion, with the larger 28 tooth pulley acting as the crank for the linkage. Two of the shoulder screws would screw directly into the metal plate, one would screw into the aforementioned 3D printed pulley, and three would screw into other 3D printed links.

The dummy calf, which would attach to the mouse's leg, was made larger than the other links to account for glued padding, and two slots were created so that twist ties or string could slide through and secure the leg.

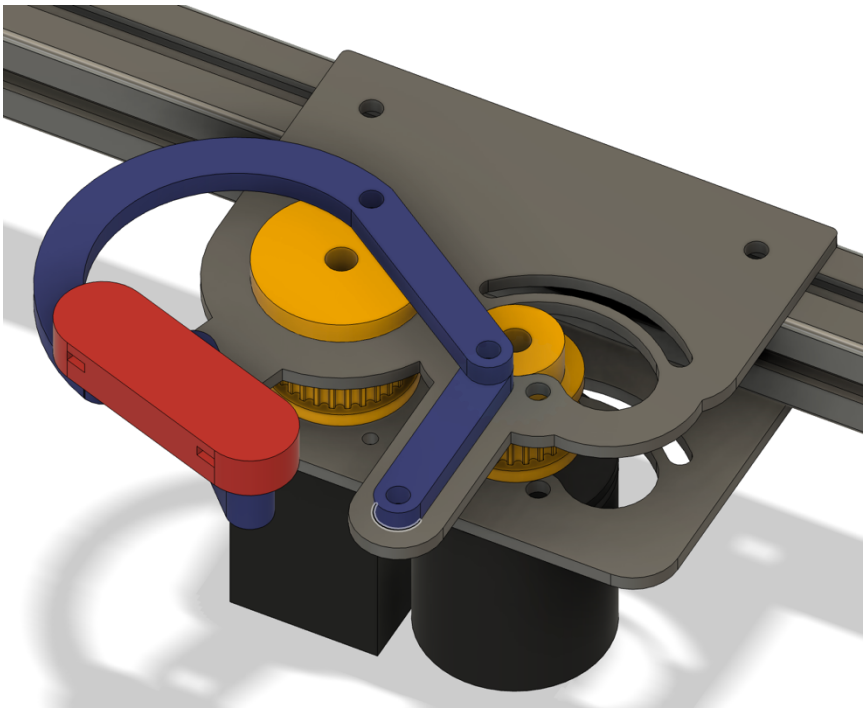

Figure 5-10: CAD model of the initial design of the linkage mechanism

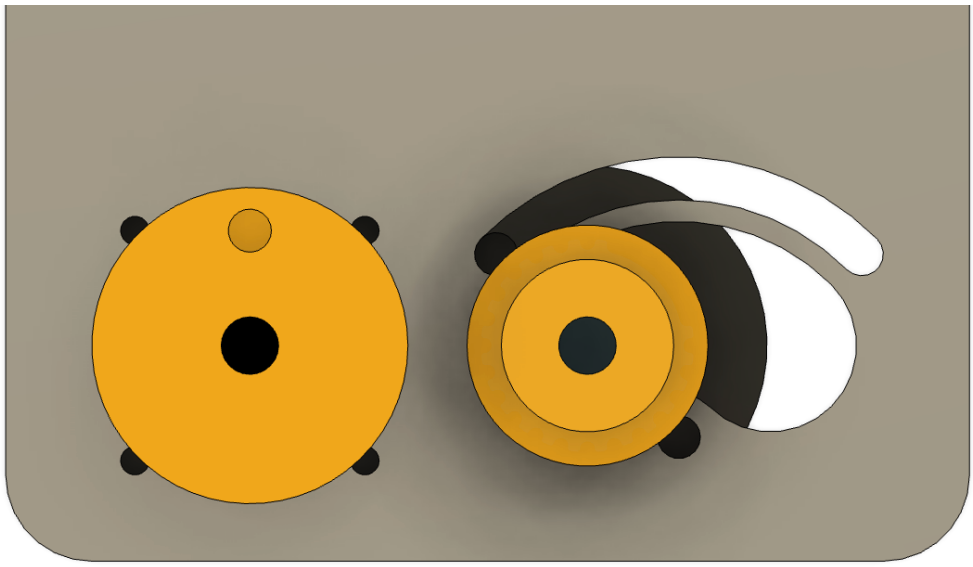

Figure 5-11: The pivoting tension system

Though the top and bottom mounting plates were designed to be laser cut steel, all these initial parts were 3D printed to check the fitment and overall design.

5.4.2. Evaluation

This design, being the first prototype, unsurprisingly had a number of problems. First, when the encoder was swiveled to put the correct amount of tension on the belt, the long arced slot and the thin strip below allowed for the encoder to tilt inwards, causing excessive friction in the rotation. This tension also caused each pulley to be pulled towards the other, pulling them out of their expected location and contacting the inner plate, creating even more friction. Additionally, with no way to maintain their position relative to each other, the slightest misalignment between the top and outer plates would result in more contact with the pulleys and additional friction. Some of the links were also made too thin, with the threads of the shoulder screw extending completely through the link contacting the plate. Because of this, the mechanism generated far too much friction and interference and was therefore unable to rotate.

Moving forward, it was decided that the mechanism would be entirely 3D printed, ditching the laser cut top and bottom pieces for easier repairability and reproducibility. Additionally, it would incorporate bearings to help mitigate the friction from inward pulling of the belt on the pulleys, and the belt tensioning system would be removed to improve stiffness.

5.5. Second Design Iteration

5.5.1. Mechanical Design

The second design iteration built on the fundamental ideas brought up during the first iteration, using the same hardware but making all the adjustments deemed necessary in the previous section.

Firstly, the front and back metal plates were changed into a larger 3D printed assembly. With this, readily available skateboard bearings could be implanted within the top section, while thicker sections above and below the pulleys could ensure that each plate remains aligned with the other. The larger pulley was split into two separate parts that screw together to sandwich the bearing. While the pulley adjustment was removed, the inclusion of multiple belt lengths ensured the correct tension could still be achieved, with ultimately the 112mm belt being the optimal choice for the arbitrarily chosen spacing. Finally, the links were also thickened so that the shoulder screws no longer protruded completely through.

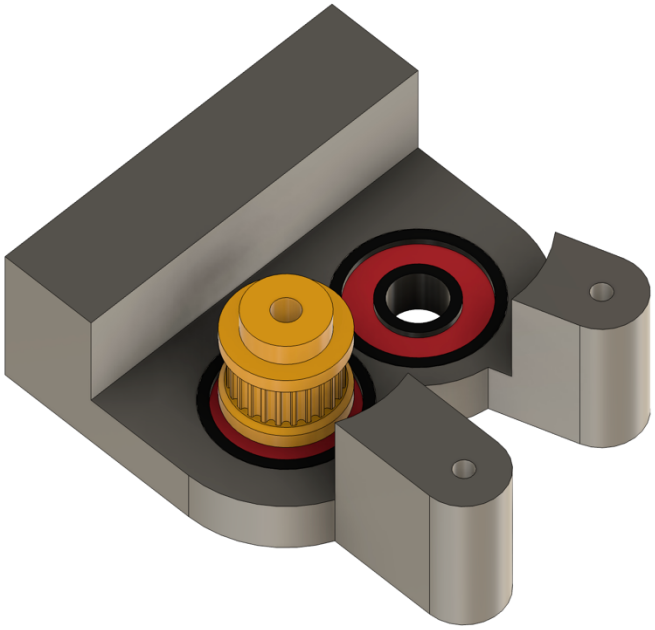

Figure 5-12: Inside the second iteration, showing the recessed bearings and one pulley

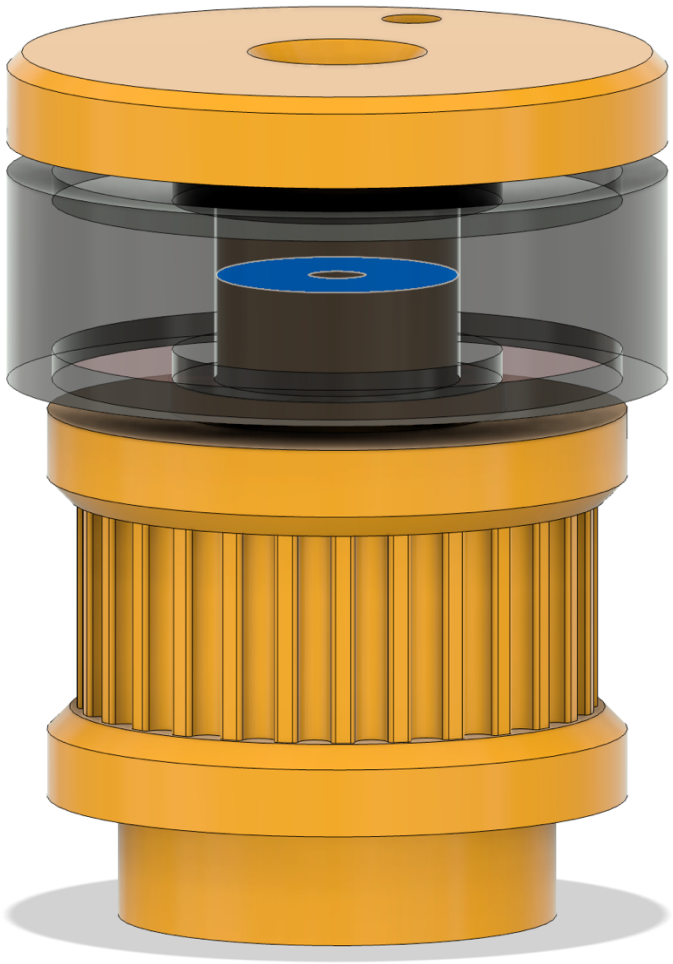

Figure 5-13: CAD drawing showing the split pulley and crank design and how it sandwiches around the bearing

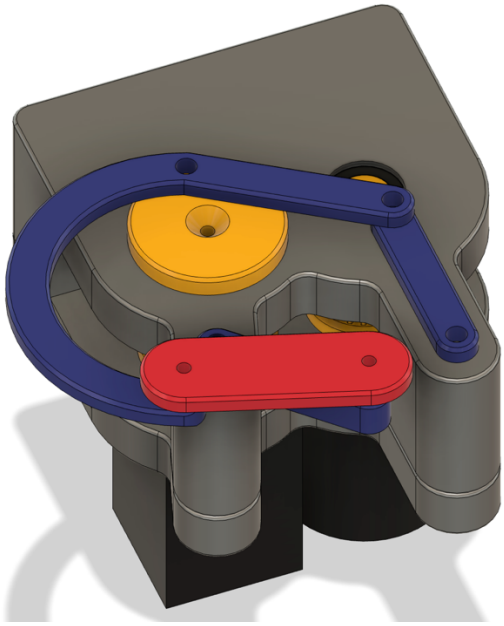

Figure 5-14: CAD drawing of second model's entire linkage mechanism

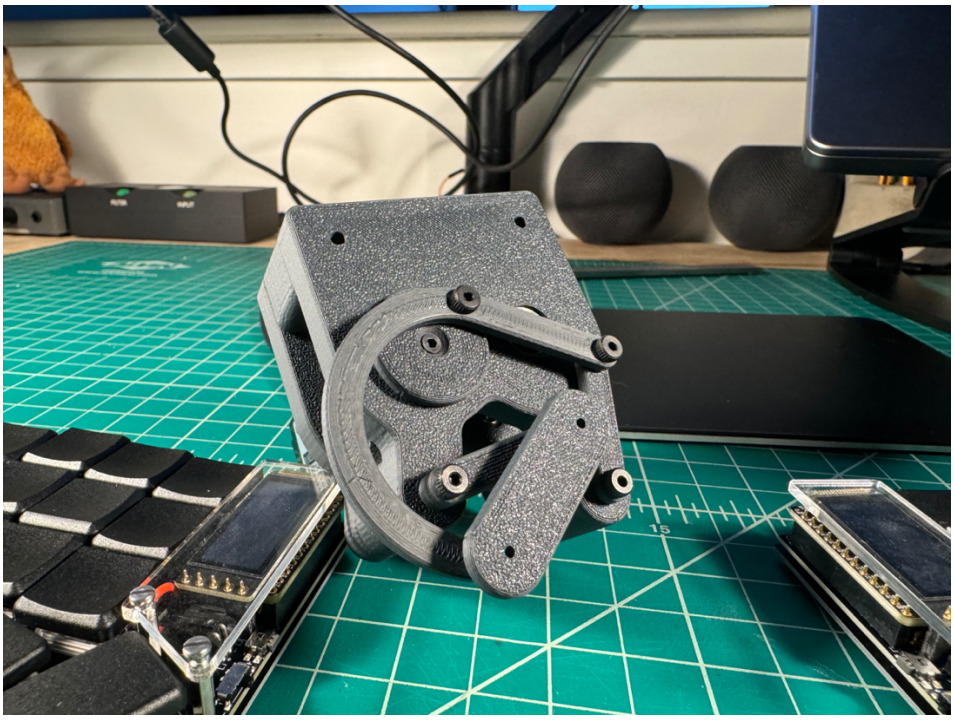

Figure 5-15: Image of the second trial's entire 3D printed linkage mechanism

Though this revision did not incorporate any changes to the leg attachment, a design for the back attachment and body adjustment was drafted. This would appoint a sliding adjustment to each of the X, Y and Z axis, allowing total range for the mechanism to adapt to the mouse's body. A curved, padded section would attach to the mouse's body with straps under the belly, providing both cushion and support.

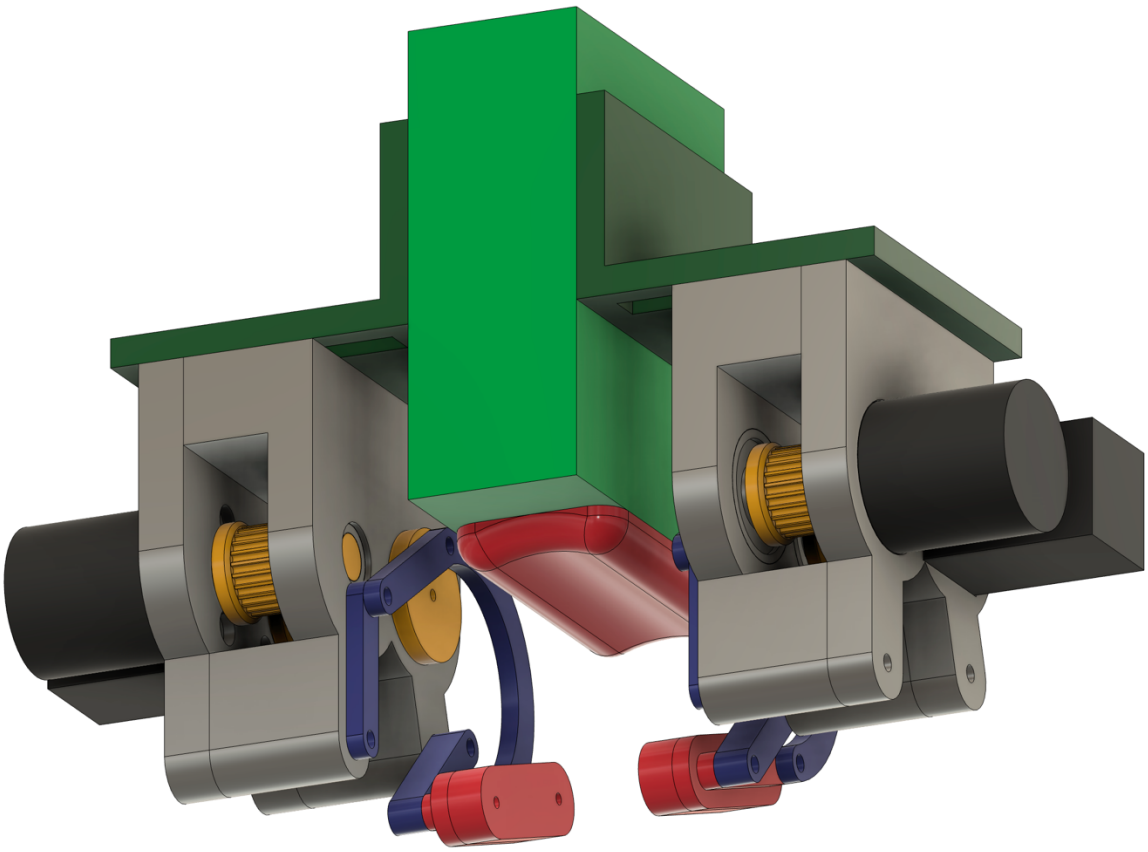

Figure 5-16: Rough idea for the mechanism adjustment

5.5.2. Electrical Design

Using the same hardware as the previous year, the stepper and encoder could simply be plugged into the existing electronics to provide motion and gain feedback. This consisted of an Arduino Uno with an Arduino Motor Shield v2.0, programmed with the final code used by the previous year. With the stepper motor plugged into M1 and M2 of the Motor Shield, the stepper completed continuous motion in a singular direction. The wiring for the encoder in their code was unknown, so while it would be useful for future tests, it was left unwired for the time being.

5.5.3. Evaluation

This revision, while solving the problems of the previous model, introduced many more that would necessitate a near complete redesign. The implementation of bearings, thicker top and outer plates with connecting supports, removal of encoder adjustment, and proper thickness links combined to fix the friction problem completely.

However, the previous year's stepper, while sufficient for a smaller linkage, was lacking the torque to drive a more complex linkage with a larger lever arm and would stall with even the slightest bit of added force. Additionally, the motor would often backdrive the Arduino because of this,

causing it to momentarily freeze until it reset back to its initial state. This was also likely exemplified by a lack of power from USB and subpar cooling without any heatsinks.

It was also at this point when it was realized that, without the original code used for the Arduino, it would be impossible to correctly implement and modify the control for the steppers and encoders. The previous year used serial input to control the steppers and serial output to read the state of the motors, and without the code from which to flash the Arduino directly, only the existing motion of the mechanism could be obtained using their existing implementation. When testing the stepper motor, it also only worked on a single side of the Motor Shield, meaning that they never implemented dual motor control and it would be impossible to drive more than a single stepper at a time.

Because of these issues, a more powerful stepper motor would be used for the next revision to overcome the torque issues. Some method to attach the linkage system to the mouse's leg still had to be developed, as well as a way to attach the linkage system to an overall mechanism, while this overall mechanism required refinement from the rough design previously outlined. A new method to power the system would also be implemented, as well as heatsinks to prevent any overheating. Additionally, between the previous year's code being controlled only by a computer, not being available, and being incomplete, it was necessary to also rewrite the code completely.

5.6. Third Design Iteration (Prototype 1)

5.6.1. Mechanical Design

As this revision was planned to be a working product that would be sent to and used by the Indiana University team, more thought was put into this revision to ensure it would fulfill all the requirements requested by them.

While the previous design used a NEMA-8 stepper with 0.02 N-m of holding torque, the new design would use a NEMA-17 stepper with 0.4 N-m. Though this new stepper motor is more than twice as large, the mechanism can avoid becoming too large by moving the crank from the stepper pulley to the encoder pulley. To help counteract some of the additional weight and heat generation from the new stepper motor, the plates for the linkage mechanism were made thicker, with additional contact area securing the top and outer plates together.

https://grobotronics.com/stepper-motor-8hy2001-10-0.20kg.cm.html?sl=en

https://azurefilm.com/product/creality-stepper-motor-42-34/

The vertical rocker link was changed to recess into the inner plate, allowing the rest of the linkage to sit more flush with the top surface, both adding some stability and removing potential interference points with the mouse. All screws to secure the plates together were also moved from the outer plate to the inner plate which might help visually simplify the mechanism for the Indiana University researchers.

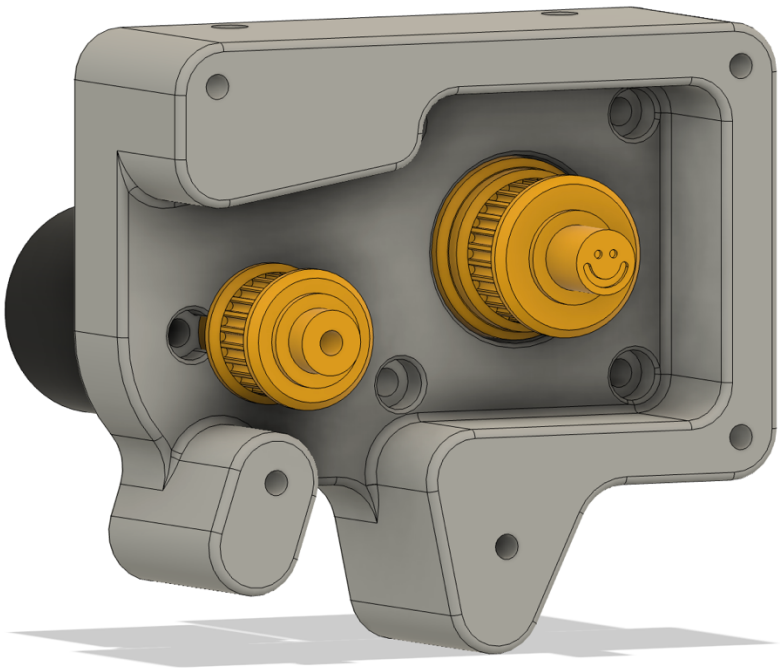

Figure 5-17: CAD model of the internal pulleys in the first prototype's linkage mechanism

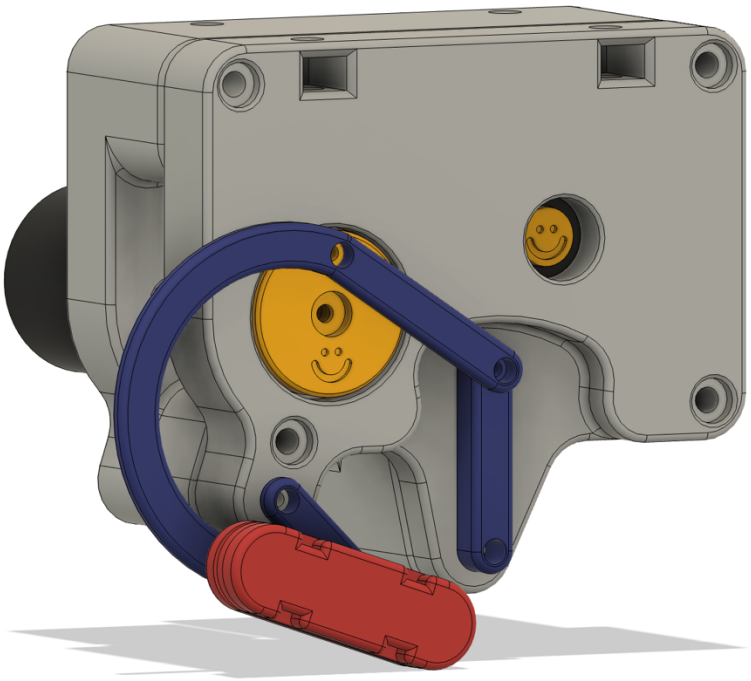

Figure 5-18: CAD model of the entire first prototype's linkage mechanism

To attach the linkage mechanism to the greater adjustment mechanism, slots along the upper end of both the inner and outer plate were added so that nuts could be slid inside, allowing for screws to be repeatedly secured vertically. These screws could be loosed to slide the linkage mechanisms inward and outward along a slot, then tightened again to be secured into place, with four on each to keep it aligned and secure. Similar implementations were used for the forward and backward adjustment, with nuts being inserted into a center piece and two screws per side protruding through slots. The vertical adjustment, however, has the screws remain stationary and the nuts slide through an internal channel. Each of the slots have gradients placed along their lengths to allow for repeatable and symmetrical adjustment. The section with the inward/outward adjustment is extended to each side to allow legs to be attached underneath.

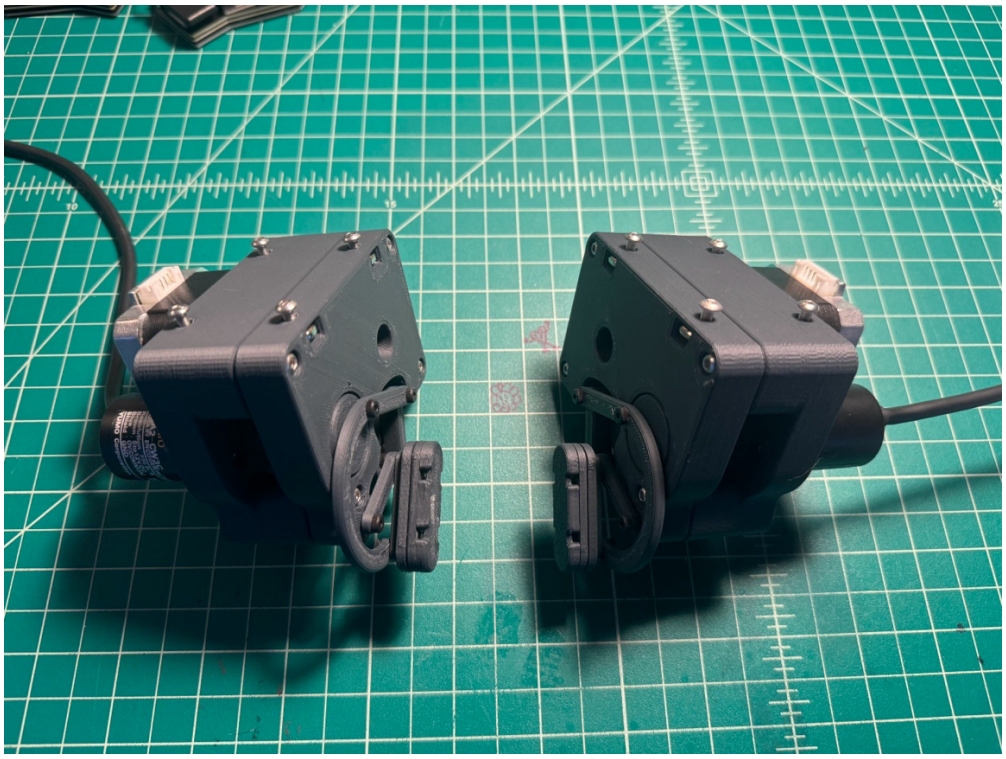

Figure 5-19: The left and right 3D printed linkage mechanisms from the first prototype with the adjustment slider screws inserted

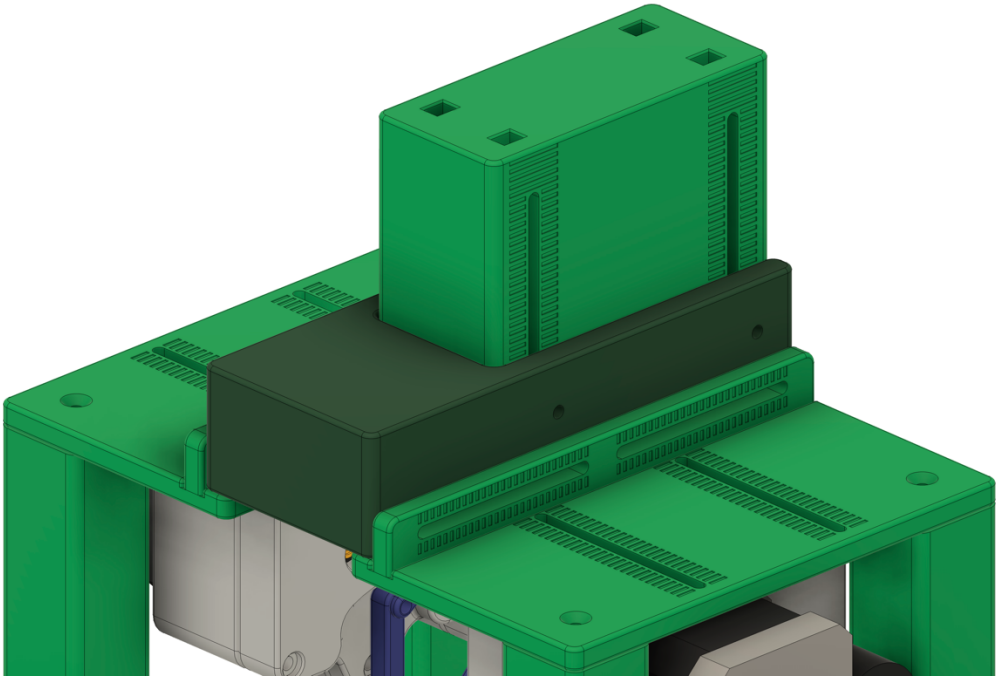

Figure 5-20: View of all adjustment sliders on CAD model of first prototype

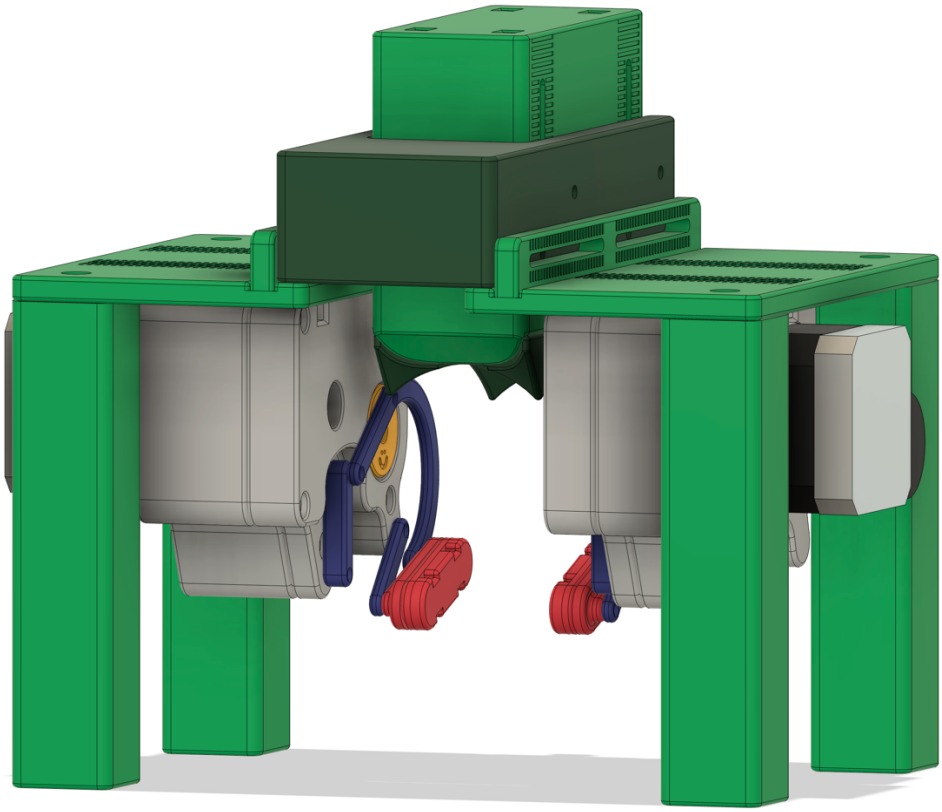

Figure 5-21: Front view of CAD model of first prototype

As for attaching the mouse, since it might be difficult to navigate around the parts of the mechanism while making sure not to harm the delicate mouse limbs, some way to make this process easier was required. It was thought that the mouse could be attached to removable pieces outside of the mechanism, with these pieces then magnetizing to their respective places on the mechanism after attachment was already completed. This would give the researchers much better access and visibility to ensure the mouse is attached correctly, making them less likely to harm the mouse or accidentally damage the mechanism.

The leg attachment would use four small 3/8” x 1/16” neodymium magnets with 1/8” center holes, with two exposed on the linkage and two hidden inside of the removable piece. A short M3 screw was used to keep the removable piece securely centered within the holes of the magnets on the linkage, with the screws helping to resist accidental removal from forces in any direction other than pulling directly outwards. Thick felt was cut to the shape of the removable piece a gap running down the middle, acting as a channel to hold the leg in place, a small Velcro strap wrapping around the calf to easily secure it in place. This felt piece also has cutouts at each end to allow the upper and lower sections of the limb to move without interference.

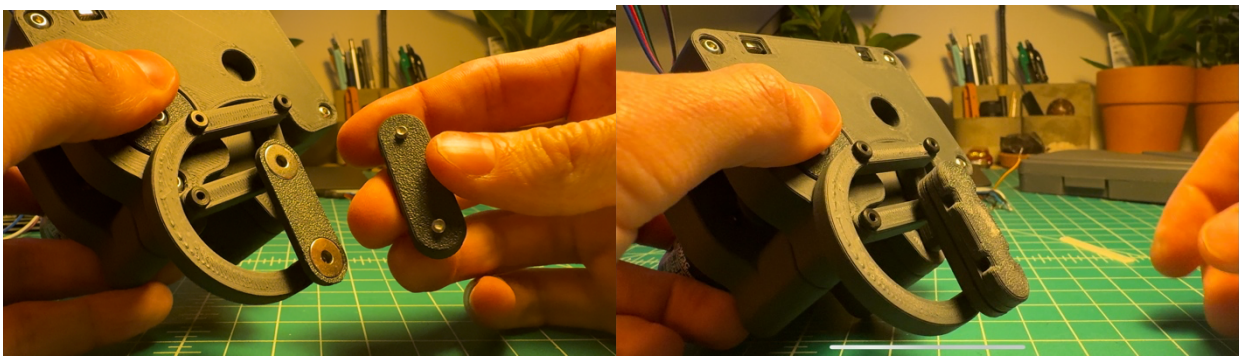

Figure 5-22: Magnetically securing the leg attachment piece to the mechanism

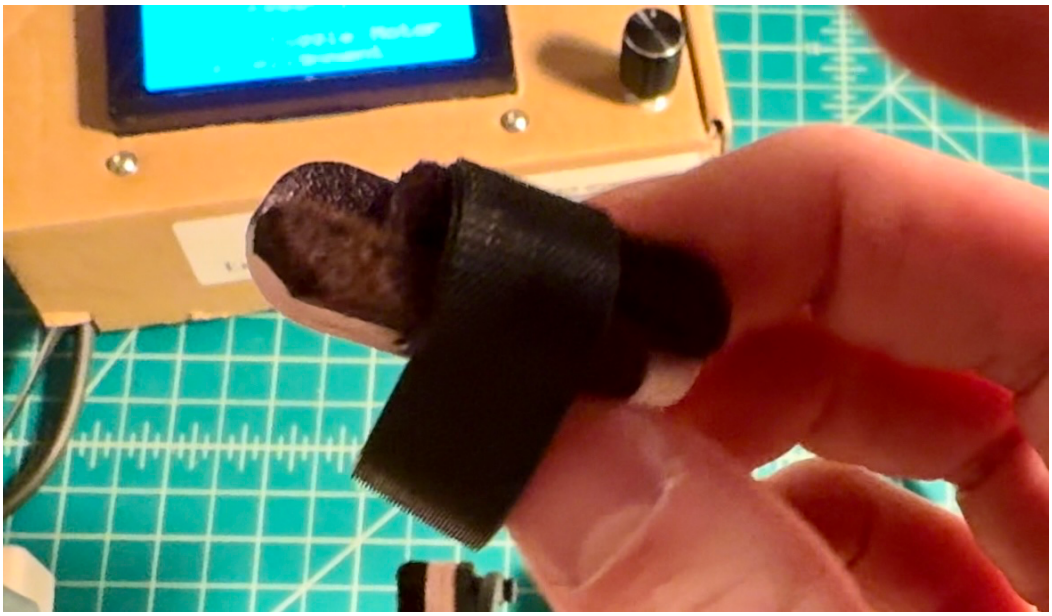

Figure 5-23: The completed leg attachment piece with felt and Velcro

Larger 3/4” x 1/16” magnets were also used on the piece that attaches to the mouse’s body, again allowing for easy attachment outside of the mechanism. This piece was curved to contour to the mouse's body, with half of a 3/4” pipe foam insulator providing cushioning and support to the mouse. Three pieces of Velcro attach the mouse to this piece; one further forward, one between the legs, and one supporting the base of the tail. A curved cutout was made at the front of this piece to ensure it remained centered and correctly positioned when fully inserted, and an optional piece could be placed on the rear to lock it in place and prevent any pivoting, again with a magnet to make it easier to secure. This piece may or may not be necessary depending on the amount of side-to-side movement the researchers prefer the mechanism to have.

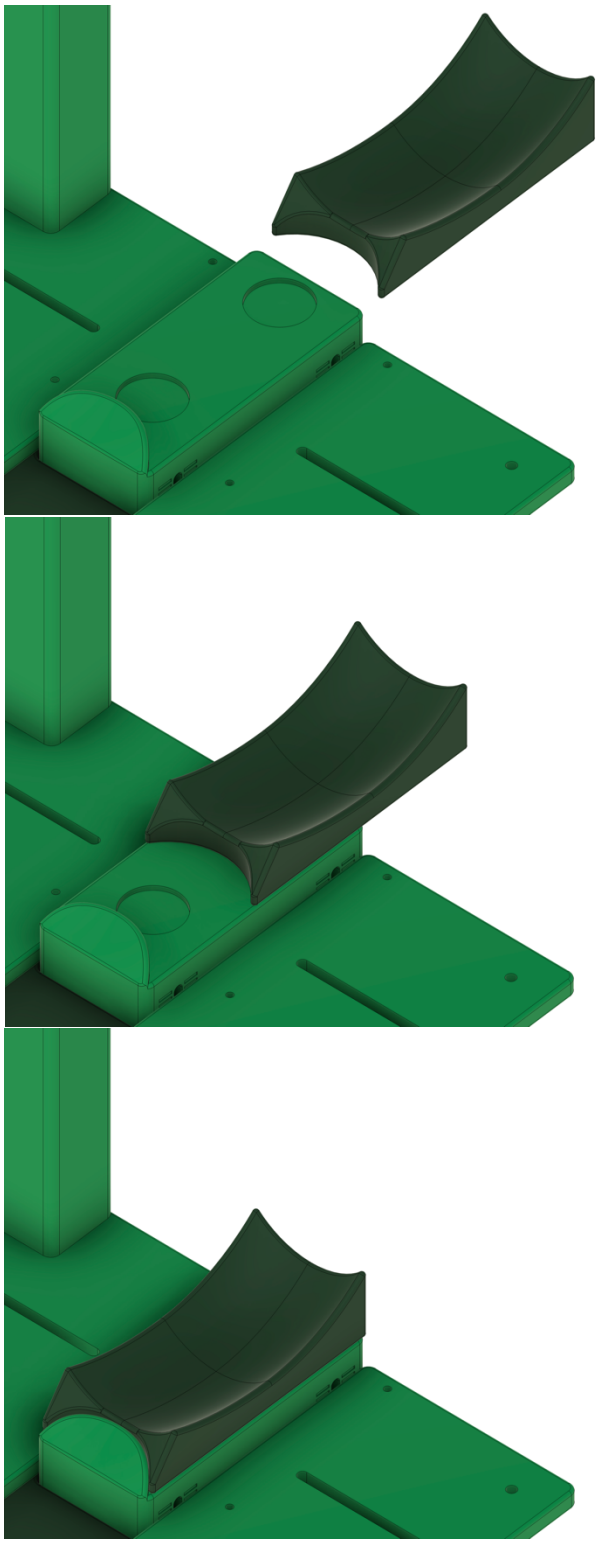

Figure 5-24: Progression of images showing the process of sliding the body attachment piece in to the mechanism

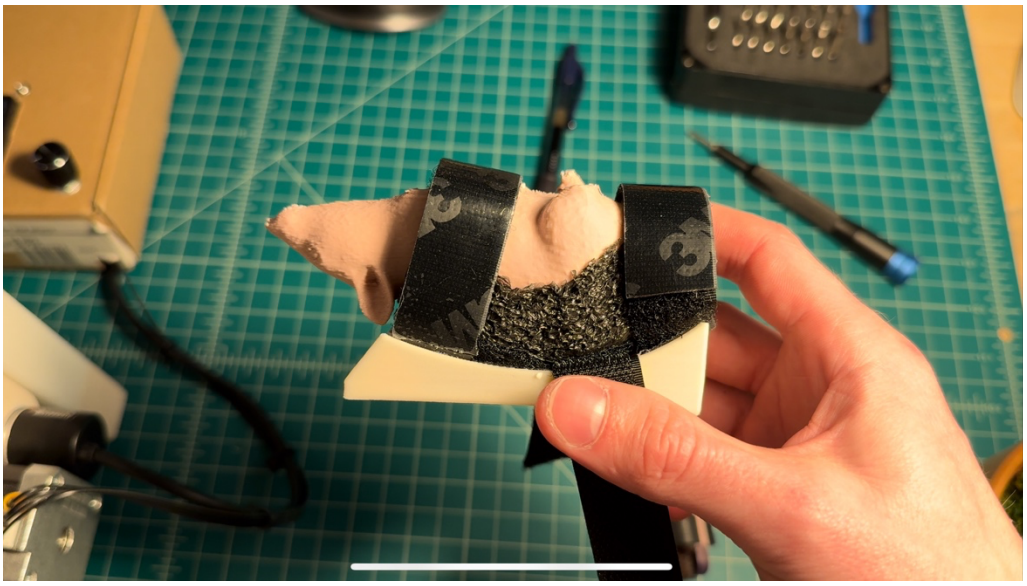

Figure 5-25: Mouse secured in body attachment piece

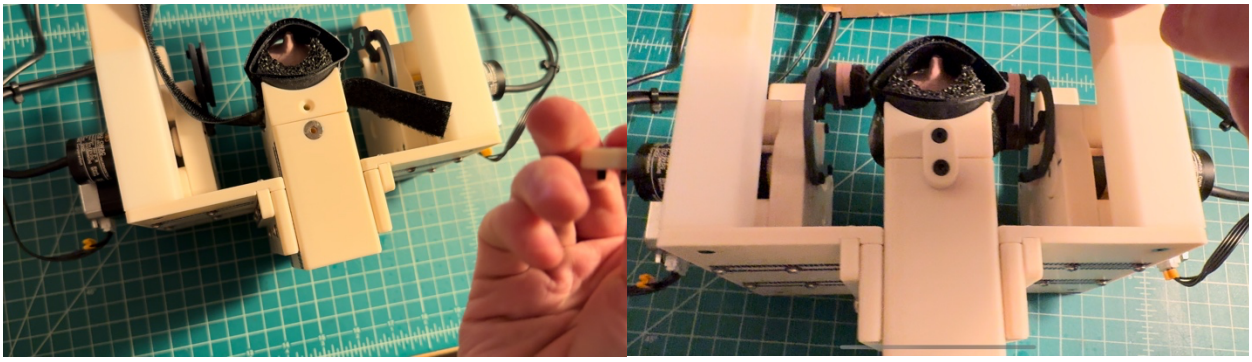

Figure 5-26: Inserting the securing piece into the mechanism

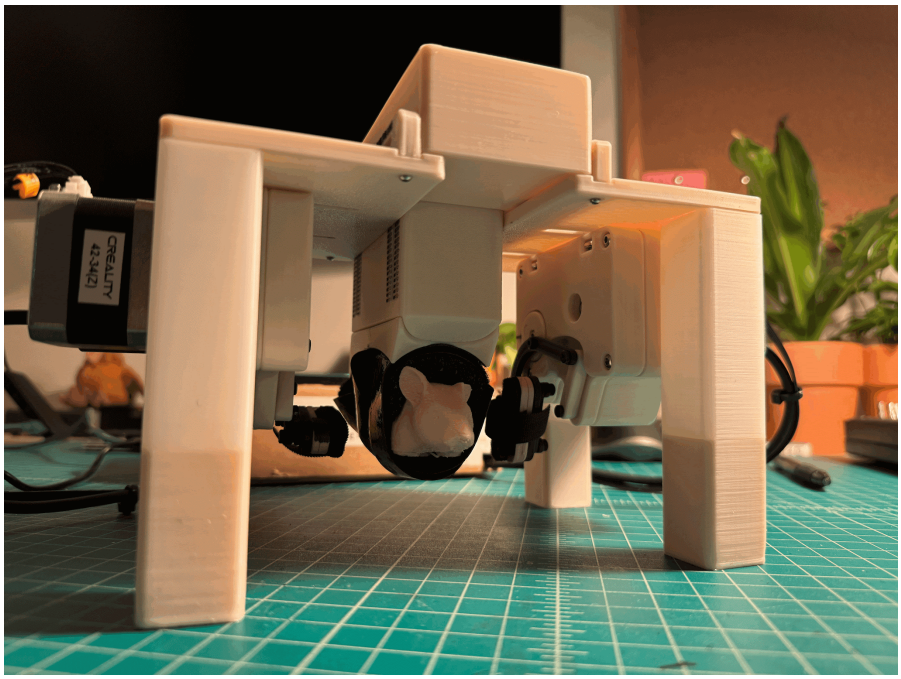

Figure 5-27: Front image of the 3D printed first prototype

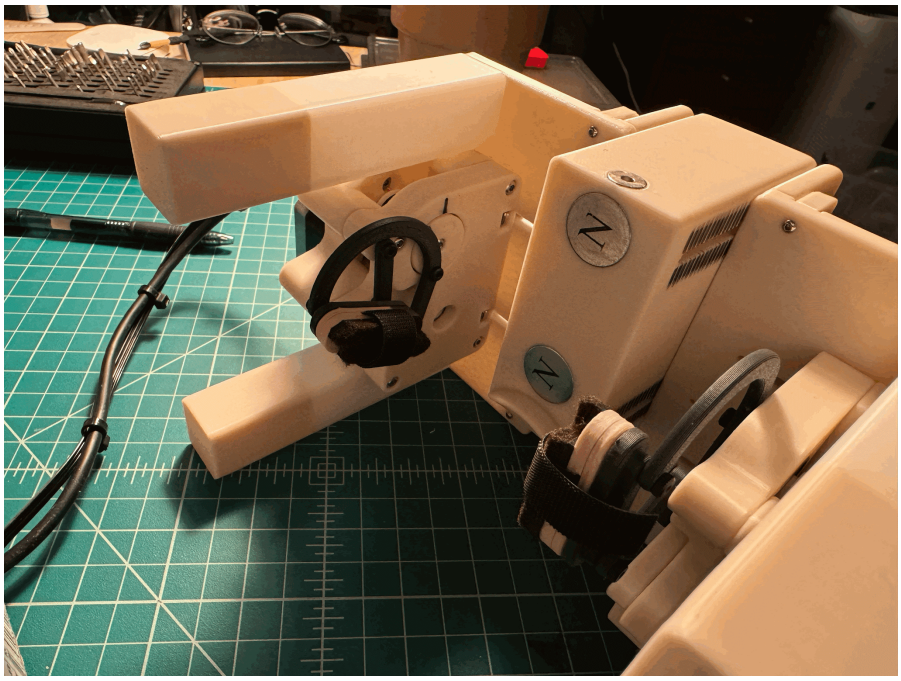

Figure 5-28: Bottom image of the 3D printed first prototype

5.6.2. Electrical Design

First, the issues regarding power required addressing. While USB was unable to provide sufficient power to drive the mechanism, especially with the larger stepper motors, a 12V power supply plugged directly into the Arduino fixed this issue, at least briefly until the Arduino overheated. Even with a heatsink on the MOSFET, function would last less than 10 seconds before thermal

protection kicked in. This same power supply, however, plugged into the Motor Shield, with the USB still plugged into the Arduino, solved all the previous power and temperature issues. Additional heatsinks were added to the power MOSFET on the Motor Shield and each of the two motor control ICs to maintain temperatures during longer operation.

When switching from computer control to some sort of external control, there needed to be a way to visualize the current system state and execute any changes or adjustments. A 4x20 I2C LCD is straightforward to integrate and could display all necessary information, while a singular clickable rotary encoder, or knobs, could be used to perform menu navigation and mechanism control. The LCD was wired to SDA and SCL, and CLT, DT, and SW of the knob were wired to pins 8, 9, and 10 respectively. The left stepper was wired to M1 and M2, and the right stepper was wired to M3 and M4.

The two C6A2-CW3C encoders required 10K pull-up resistors on each of the three phase-terminals, which were soldered onto the prototyping board built into the Motor Shield. Phase A, B, and Z of the left encoder were wired to pins 7, 6, and 5, while A, B, and Z of the right encoder were wired to 4, 3, and 2.

However, upon testing the encoders, it was found that only one was functional. Upon investigation, it appeared that two out of the three encoders were defective, and no amount of troubleshooting resulted in any solution. With the prototype needing to be shipped out as soon as possible, the decision was made to make use of the mechanism as is without ordering new encoders to replace the broken ones. The mechanism could be adapted to operate properly without the encoders, and as was realized while programming and testing, not lose any functionality vital to the function of the mechanism. Without the constant feedback from the encoders providing positioning data though, it's crucial to ensure the steppers always remain 180 degrees out of phase with each other to mimic realistic walking motion.

All these parts were secured in a cardboard housing, the LCD and knob accessible through the top, and the wires routed through holes in areas convenient relative to their destinations.

Figure 5-29: External view of the first prototype electronics box

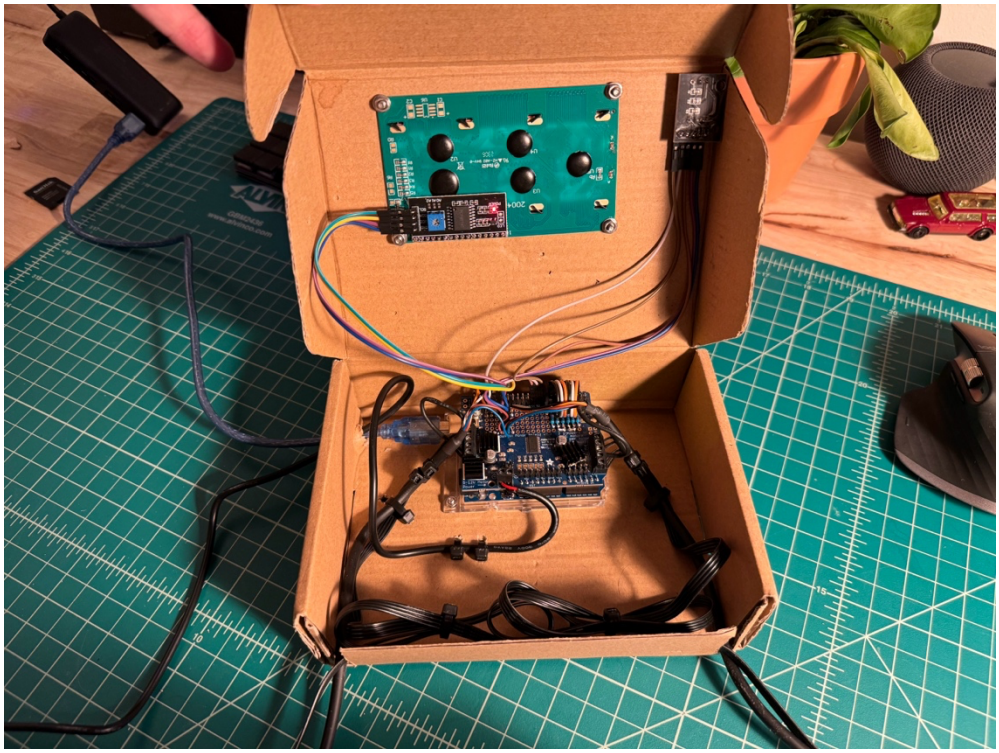

Figure 5-30: Internal view of the first prototype electronics box

The control box can be cycled between three screens: Alignment, Microstep, and Full Speed. Alignment sets the position of the left and right steppers, Microstep allows the mechanism to be better adjusted to fit the mouse with as little potential for harm, and Full Speed moves the legs at full speed for kinematic rehabilitation.

The Alignment screen begins with prompt Aligning Left, with the rotation of the knob corresponding to the rotation of the crank. Once the black line on the crank is aligned with the similar line on the inner plate, the knob can be pushed to repeat the same procedure with the right leg. When complete, the left leg will offset 180 degrees from the right leg, and the screen will progress to slow rotation.

Figure 5-31: Image showing the properly adjusted alignment lines

The Microstep screen uses the microstep function within the Arduino Motor Shield PWM Servo Driver library to produce slow, fluid motion. A microstep divides a single step into 16 smaller steps, therefore reducing the rotation speed so each motor spins at about 3 rpm. Pressing the knob will start both steppers rotating in the same direction, while pressing the knob again will stop the motion. To keep the stepper motors exactly 180 degrees offset, each stepper takes one step before the encoder button state is read, ensuring that they travel the same number of steps before the motors are stopped.

Pressing and holding the knob for more than one second will progress the menu to Full Speed. This changes the motors from operating with microsteps to fullsteps, therefore increasing the speed by a factor of 16 to about 48rpm. Like the Microstep screen, pressing the knob will enable the motors rotation, while pressing it again will disable their rotation, again ensuring each motor has completed an equal number of steps before changing their state.

A long press of the knob on the Full Speed screen reset back to the Alignment screen, allowing for any possible misalignments to be corrected. While the motors are in motion, the screen will display "Running", while otherwise "Stopped" will be shown.

5.6.3. Evaluation and Feedback

As is expected with the first prototype, especially since the WPI and Indiana University teams were only collaborating digitally, there were a handful of problems with the design that caused it to not be functional during real world tests. Additionally, there were a handful of minor complaints

and changes that, while weren't as important to address in the next revision, would provide quality of life changes during operation.

First, and most importantly, the mechanism doesn't fit the mouse for a handful of reasons. The dummy limbs and the overall movement of the mechanism was too large compared to Indiana University's lab mice. As it turns out, the paper used to investigate the lengths of mouse limbs proved to be too long compared to those used in the lab, and therefore, the entire mechanism required scaling down by about 25%. The mechanism doesn't adjust far back enough for the leg to be properly positioned to fit into the mechanism, so the forward backward adjustment needed to be extended to remedy this issue.

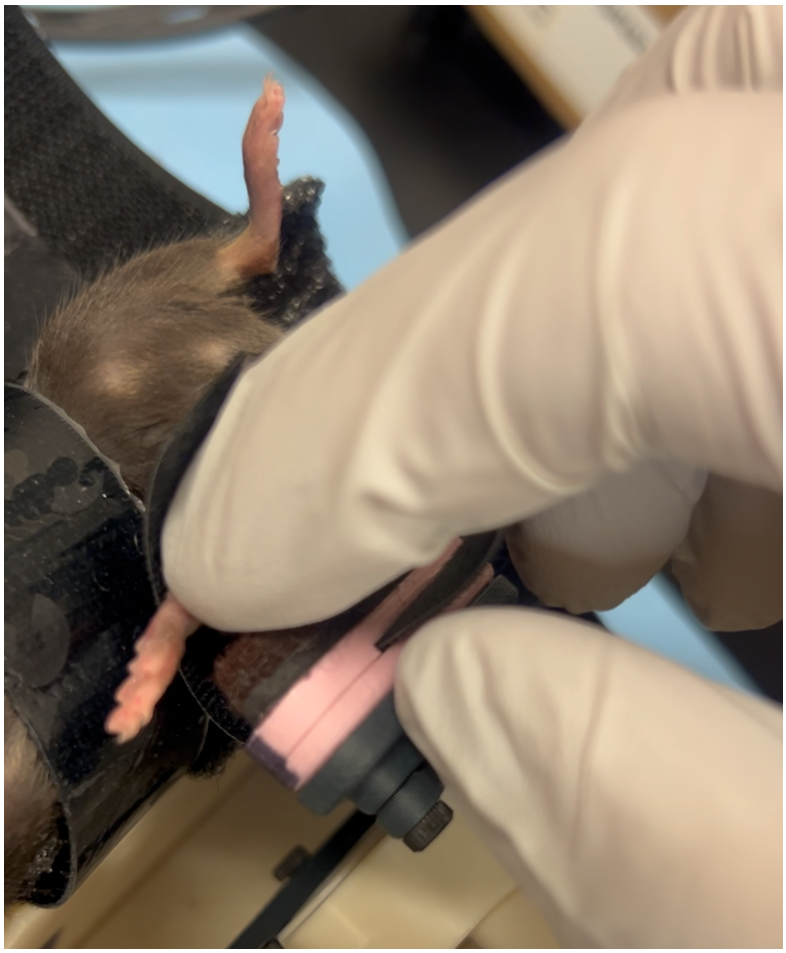

Figure 5-32: Image of the mouse in the Indiana University lab, with the technician struggling to position the leg correctly

Lastly, the middle of the leg is unable to attach to the mechanism because it is covered in a thick, loose skin that becomes buried in its stomach and interferes with the attachment section of the linkage. This skin, colloquially named the pants, covers nearly the entire lower leg down to the ankle. The current design overlooked the existence of the pants and therefore requires the attachment point be shifted from mid-limb to the bald section near the ankle to bypass this skin.

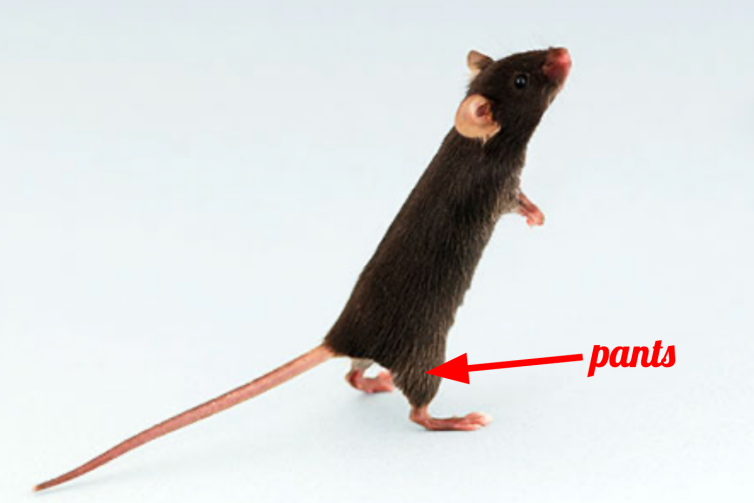

Figure 5-33: Arrow pointing to the specific area of the mouse dubbed the 'pants'

There are also a handful of smaller quality of life improvements that could be made to the mechanism. First, the legs felt slightly delicate and could use some reinforcement. Second, reducing the amount of screws required to adjust the mechanism would make the process less cumbersome. Removing the encoder would also allow the mechanism to be more compact, as well as simplify the wiring coming out of the control box.

However, the Indiana University team continued to stress the importance of the adjustability aspect of the mechanism, of which this revision completely lacks. Currently, the coupler curve follows only a single set trajectory, continuously tracing the set path at only two set speeds. With the wide variation of mouse sizes, along with the reduction in their limb flexibility post-injury, it is required that the mechanism be adjustable to broaden its abilities and be applicable to as many mice and stages of healing as possible.

Electronics wise, the cardboard box feels cheap and flimsy. The motors are also loud, and when on the fullstep mode, vibrate quite a bit. Even with the checks to ensure the motors remain in sync, sometimes spinning the knob will cause them to become misaligned. On a similar note, sometimes the rapid spinning of the knob to align the motors will, for some reason, overload the Arduino, briefly causing the screen to turn off.

Otherwise, the mechanisms function seems promising and was a welcome step up from the previous year.

5.7. Fourth Design Iteration (Prototype 2)

5.7.1. Mechanical Design

With the feedback received, the first step was shrinking the linkage by 25%. This decreased the length of the lower dummy leg from 25.30mm to 18.97mm, decreasing the size of the coupler curve to something that would better match the size of the mice in the lab. It seems as if the limb length of the initial linkage was accidentally misinterpreted, however, this new smaller linkage is more in line with the results from a study by Leah M. Sparrow, with the lower leg of a mouse being 18.75mm on the lower end and 21.44mm on the upper end [17]. This smaller linkage, coupled with the removal of the encoder, allows for a much more compact linkage assembly with fewer pieces.

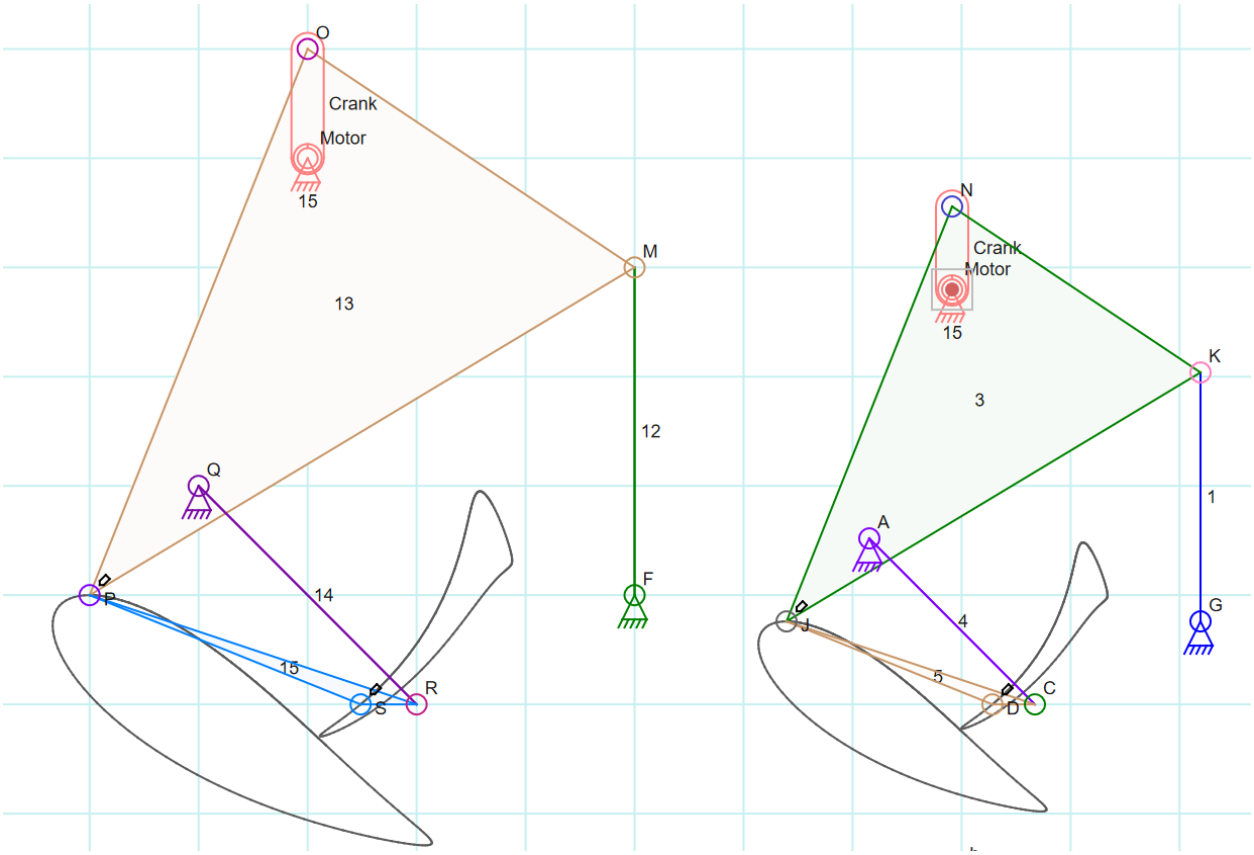

Figure 5-34: Previous linkage is shown on the left, new linkage is shown on the right. The new linkage is scaled down 25% compared to the previous version.

Without the need for the encoder, each of the pulleys could be removed and the inner and outer plates could be combined into one singular piece. The crank was instead made into a long cone that directly connects to the stepper, the slight taper allowing the entire plate to be removed from the stepper while ensuring the crank remains correctly positioned. Similar to the previous design, there is a gap to the one side of the crank to give the large arched coupler space to deflect, while a recessed section on the other acts as a stable platform for the rocker. The links themselves, where

possible, have also been reinforced around the shoulder screw locations in an attempt to make them more durable.

The same slotted holes with inset nuts were added to the mechanism for slider adjustability, however, the number of total adjustment screws was reduced from 16 to 6, just one per axis direction on both the left and right side. A recessed track was added to keep the inward outward adjustment mechanisms in line with the fewer screws, the forward backward adjustment piece slides along a parallel lower piece to keep it horizontal, and the vertical adjustment piece slides through a hole to keep it from rocking. The gradients remain along each adjustment slot to again allow for consistent positioning.

The forward backward adjustment was increased from 50mm to 100mm, and the upper section was reinforced to prevent the larger slot from bending out of shape. To accommodate this larger movement and shifting center of mass, each pair of legs was connected into a singular piece and extended outwards to increase the stability.

Since the way the back attachment functioned properly, the only change made there was the removal of the Velcro straps deemed unnecessary. However, since the leg attachment interfered with the mouse pants, the contact area was shifted downwards away from the pants and the wide Velcro piece was replaced with a thinner rubber band. This rubber band can also easily be swapped out for a piece of string if the technician believes tying it to the link would be easier than pushing it through the band. Both the back and leg attachment remain magnetically attached, again allowing for them to be secured outside of the mechanism.

The adjustability aspect of the mechanism, however, remained unsolved. The first idea was to replace some of the links with small turnbuckles, though with how compact the 3D printed links are, the torque required to adjust the turnbuckles could potentially damage them. Additionally, even the smallest turnbuckles prove to be too large, with the longest section between straight joints being just above 20mm. Instead, the turnbuckle could be replaced for a stiff section of wire, with pliers being used to bend the wire, easily adjusting its length. However, precisely adjusting the mechanism using this method would be exceedingly difficult and matching each side to the other would prove to cause similar challenges, while bending the wire could cause the linkage to be non-parallel and move in unplanned ways.

Because of the difficulties of implementing infinite adjustability into such a small linkage, the decision was made to instead have a fixed number of adjustment points, which are switched between by adjusting the location of the linkage joints, unscrewing and replacing the shoulder screw in a different location. The total number of these points were limited to reduce the risk of thin walls causing weak threads in the plastic, as the extremely small screws and their corresponding holes, especially in a 3D printed part, are fairly delicate.

Two adjustment points, located on the vertical rocker, are designated S for short and L for long. While S is the original joint location, L comparatively steepens the motion of the ankle and makes the knee swoop down less. The other two adjustment points, located on the circular crank, are designated I for inner and O for outer. These keep the general shape of the S and L coupler curves

but instead adjust their magnitude, allowing for the overall size of the motion path to be constrained.

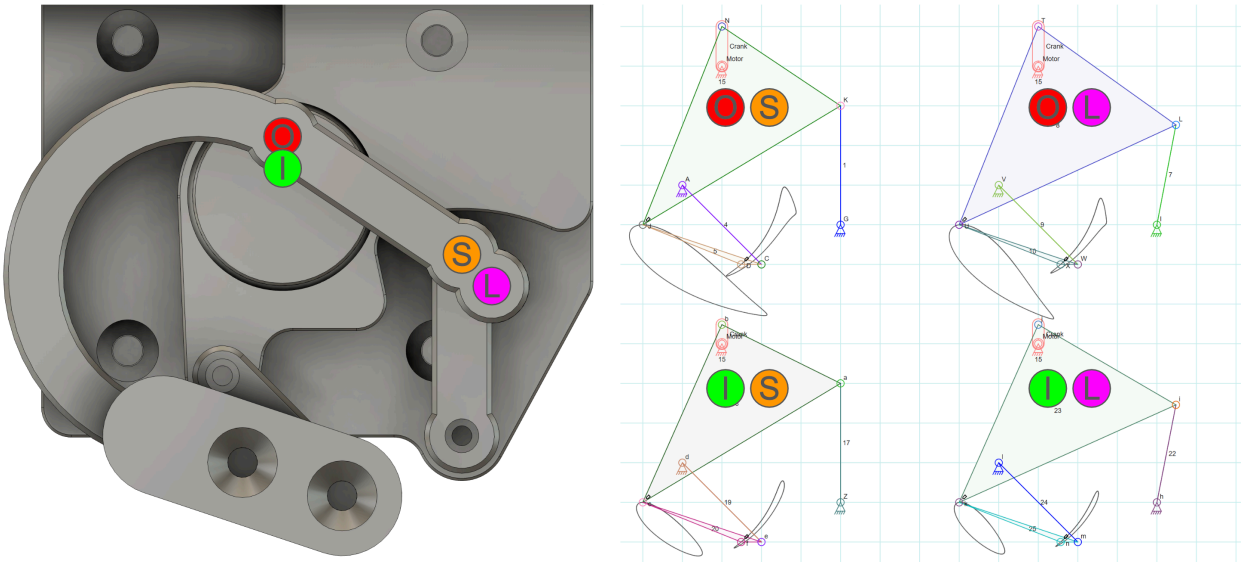

Figure 5-35: Combination of all possible joint locations and their effects on their corresponding coupler curves

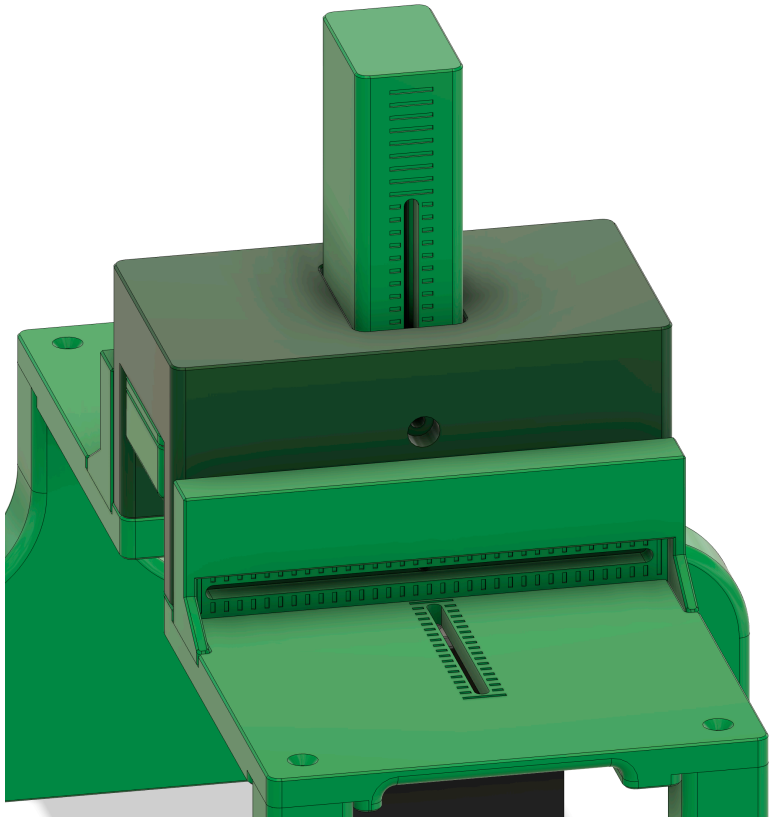

Figure 5-36: View of all adjustment sliders on CAD model of second prototype

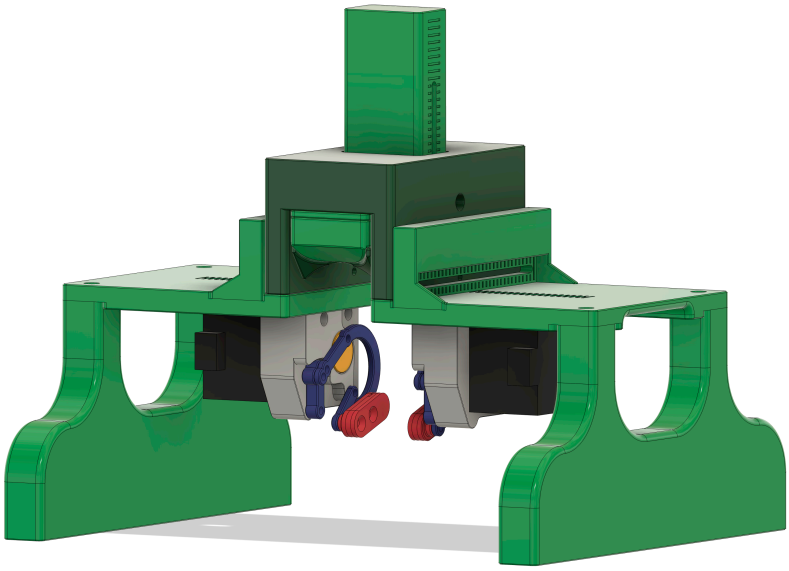

Figure 5-37: Front view of CAD model of second prototype

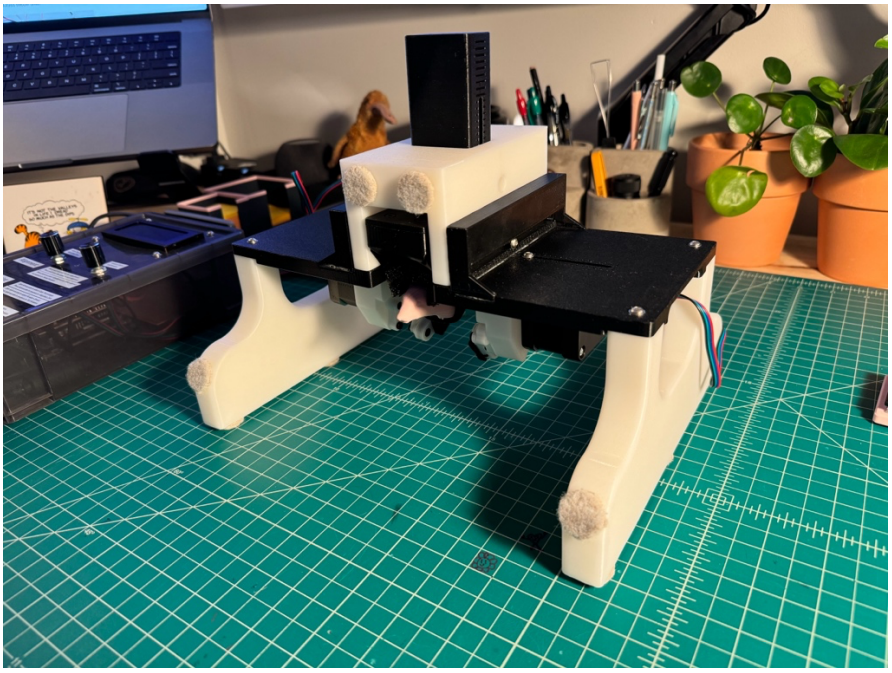

Figure 5-38: Front image of 3D printed second prototype

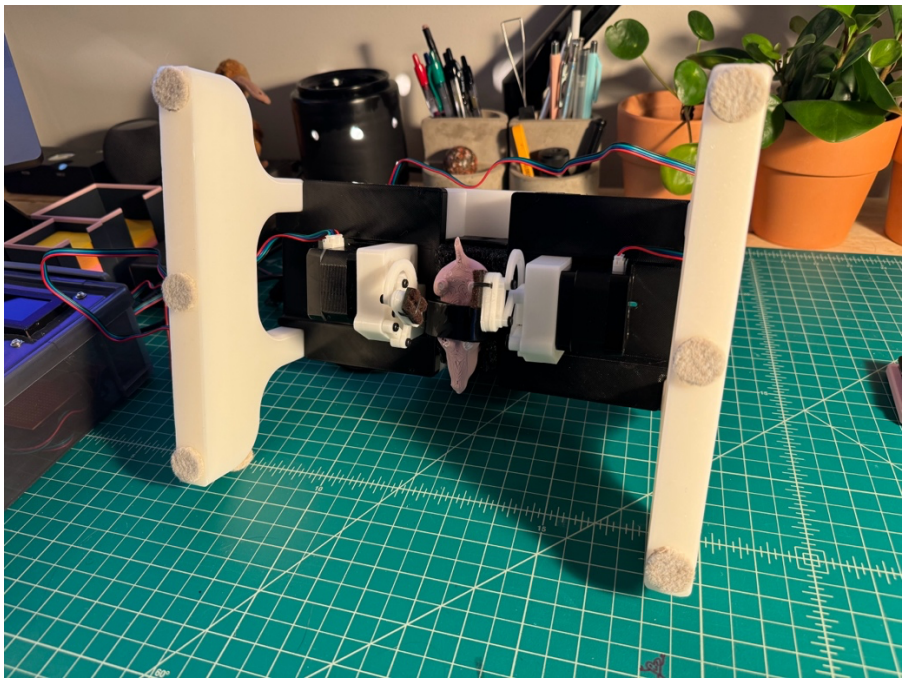

Figure 5-39: Bottom image of 3D printed second prototype

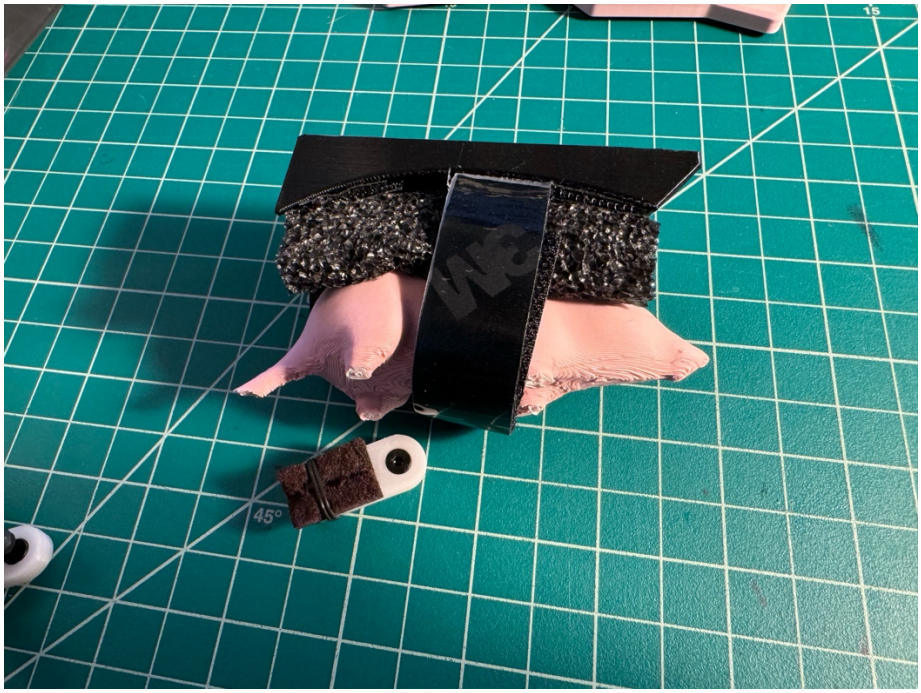

Figure 5-40: Mouse secured to back attachment device outside of mechanism

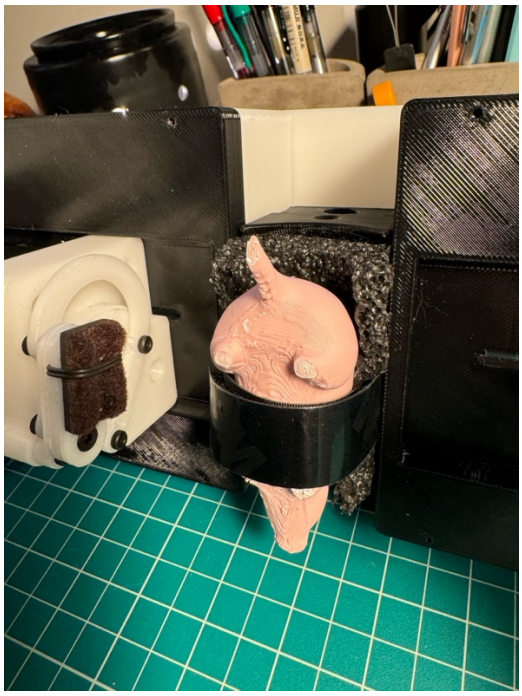

Figure 5-41: Mouse slid into mechanism while secured to back attachment device

5.7.2. Electrical Design

The motor drivers built into the Motor Shield are meant more for small motors than steppers. While steppers are able to be driven, their functionality and performance are limited by these drivers, as is visible by their loud and jittery motion and lack of control options in code. The CNC Shield, however, designed for homebrew CNC machines and 3D printers, supports swappable modern stepper motor drivers, allows for more customizable motion control in code and the ability to integrate individually chosen stepper motor drivers. The TMC2209, a stepper motor driver available for the CNC Shield, is advertised to have silent operation, automatic microstepping, and lower power consumption and heat generation than similar older drivers.

With stepper motors on the CNC Shield, a large power supply with ample current is advantageous to consistent operation over long periods of time, and therefore the previous 12V 1A power supply was upgraded to one with 12V and 10A. The Arduino, like the last version, is powered separately through USB. Since the encoders were removed for this revision, many pins on the Arduino were freed up, and therefore there was now plenty of space to integrate an additional knob to make navigation through the menus more intuitive. All of the electronics are housed in a more durable and more aesthetically pleasing plastic container. The screen and both knobs are accessible from above, while the stepper wires and power cables extend out of the back and the front of the box respectively.

While these additions are all advantageous to the function of the mechanism, changing from the Motor Shield to the CNC Shield necessitated a near complete rewrite of the underlying code. However, this allowed for the feature set to be reconsidered, giving the opportunity to implement new motion mechanics.

The new electronics implement four unique control screens: Alignment, Partial Rotation, Adjusting Zero, and Full Rotation.

Figure 5-42: External view of the second prototype electronics box

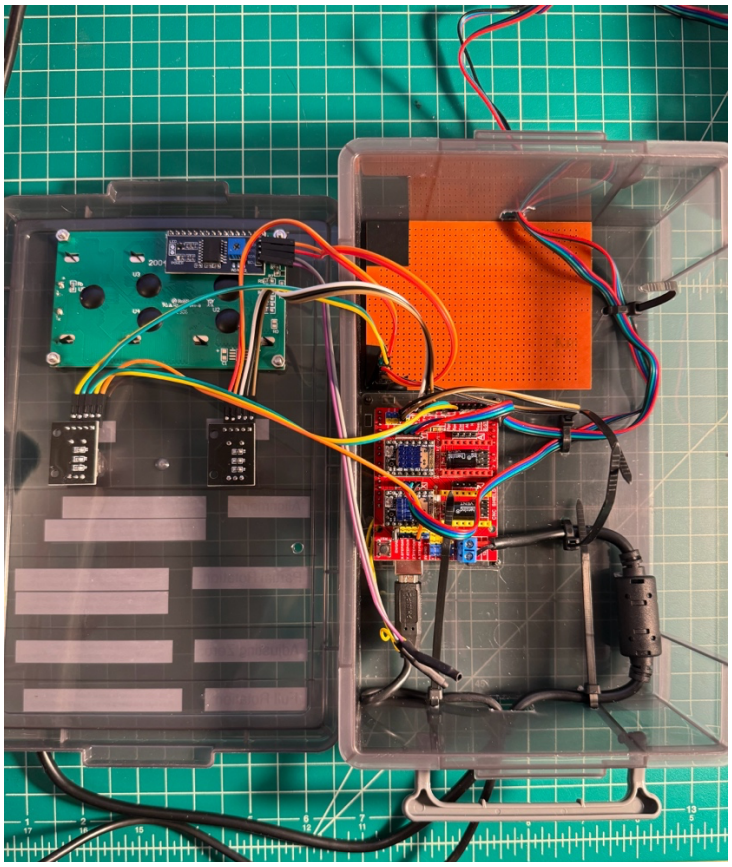

Figure 5-43: Internal view of the second prototype electronics box

In Alignment, rotating the left knob rotates the left stepper, while rotating the right knob rotates the right stepper. Similar to the first prototype, the two sides are considered aligned when the black line on the crank is aligned with the similar line on the housing. When these lines are close, pressing the respective knob will switch from coarse to fine adjustment, allowing for the lines to be more accurately positioned. Pressing the knob again will lock the rotation, allowing for the other side to be adjusted without the worry of accidentally moving the completed side. When each side is locked, both knobs can be pressed simultaneously to progress to the Partial Rotation screen.

Unlike the first prototype, however, where these lines were arbitrarily placed, this line ensures that each crank is positioned vertically, ensuring the dummy leg is positioned in its furthest tucked-up position that is closest to the body, as this is similar to how a newly paralyzed mouse would have its legs positioned.

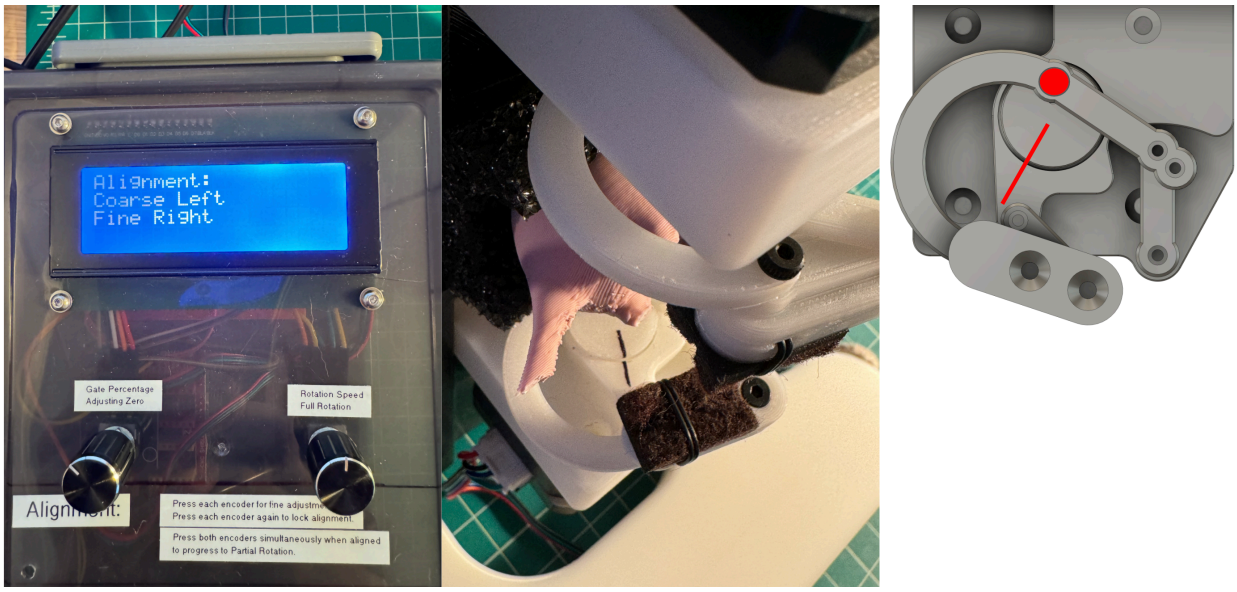

Figure 5-44: Image of the Alignment screen and, the alignment line, and the corresponding vertically aligned crank

Partial Rotation is an attempt to mitigate the downfalls of having a mechanism with limited adjustability. Instead of completing continuous rotations to simulate walking, Partial Rotation oscillates each leg back and forth along the set zero point to stimulate the mouse's legs and slowly work up from a complete lack of motion to a full motion range. The left knob controls the gait Percentage, adjustable from 0-10, with 0 being no rotation, 10 being 55% of a complete rotation, and each number in-between being an equal step between those two values. Similarly, the right knob controls the Rotation Speed, again adjustable from 0-10, with each number increasing the number of steps per second by 100, from 0 to 1000. Since each rotation has 1600 effective steps with the built in microstepping from the TMC2209, this corresponds to a max speed of 37.5 RPM.

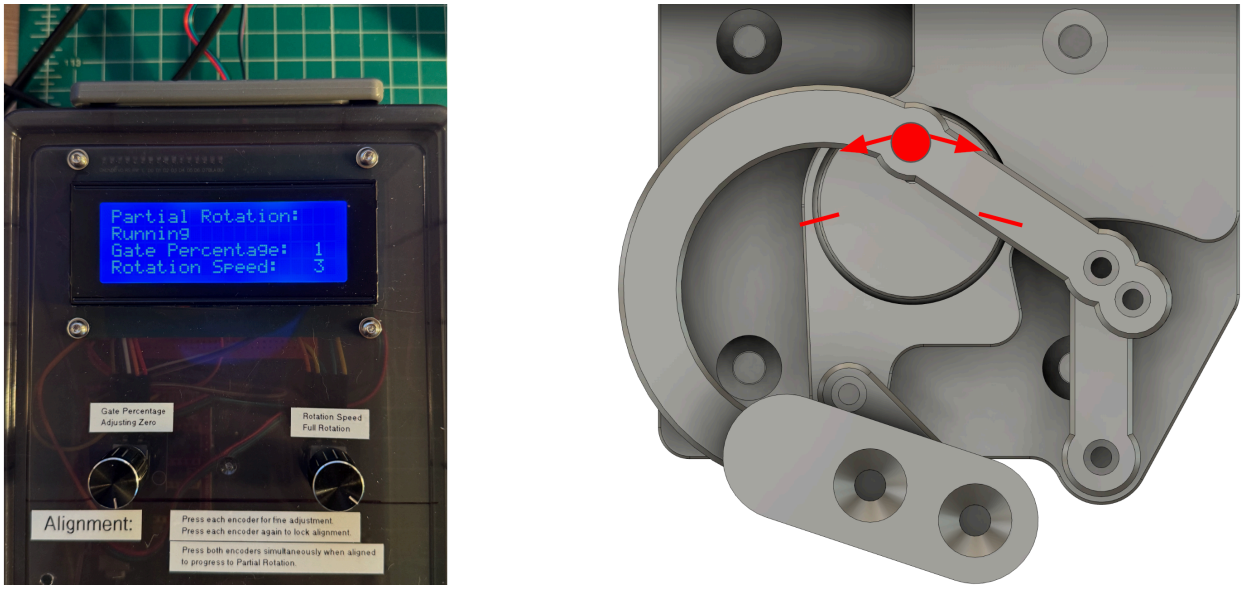

Figure 5-45: Image of the Partial Rotation screen and the corresponding linkage motion

The zero point about which the Partial Rotation oscillates, by default, is the topmost position of the crank; however, the specific location of this zero point can be changed. After pressing the left knob, which progresses the screen to Adjusting Zero, rotating the knob will similarly rotate each of the steppers, while pressing the knob again will set the current position as the new zero and progress back to Partial Rotation. This allows for various motion ranges to be individually exercised: for example, slowly moving the zero point from the top of the crank to the bottom, and therefore easing the mouse from minimum to maximum leg deflection.

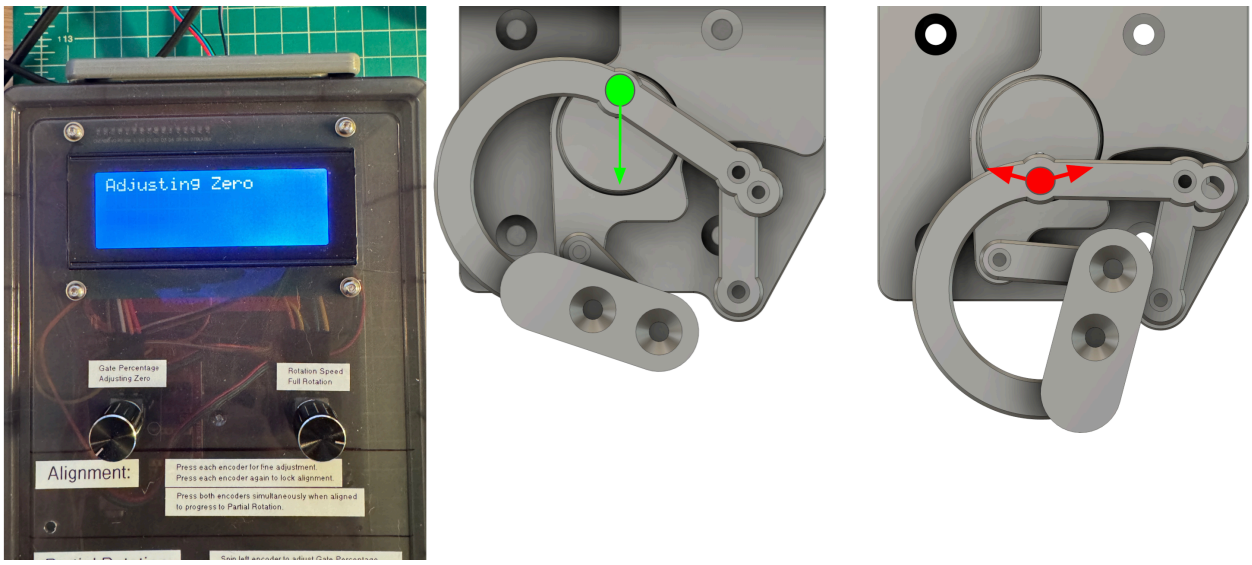

Figure 5-46: Image of the Adjusting Zero screen and an example of what occurs when assigning a new zero location

When the mouse is expected to have regained full limb mobility along its entire walking motion path, the right knob can be pressed to progress from Partial Rotation to Full Rotation. In this, the left stepper will offset a half rotation, with each stepper rotating continuously in one direction, simulating a walking motion. Spinning the right knob will adjust the speed from 0-10, again increasing by 100 steps per second from 0 to 1000, with the max speed setting 10 being equal to 37.5 RPM. While in this mode, pressing the right encoder again will bring the screen back to Partial Rotation.

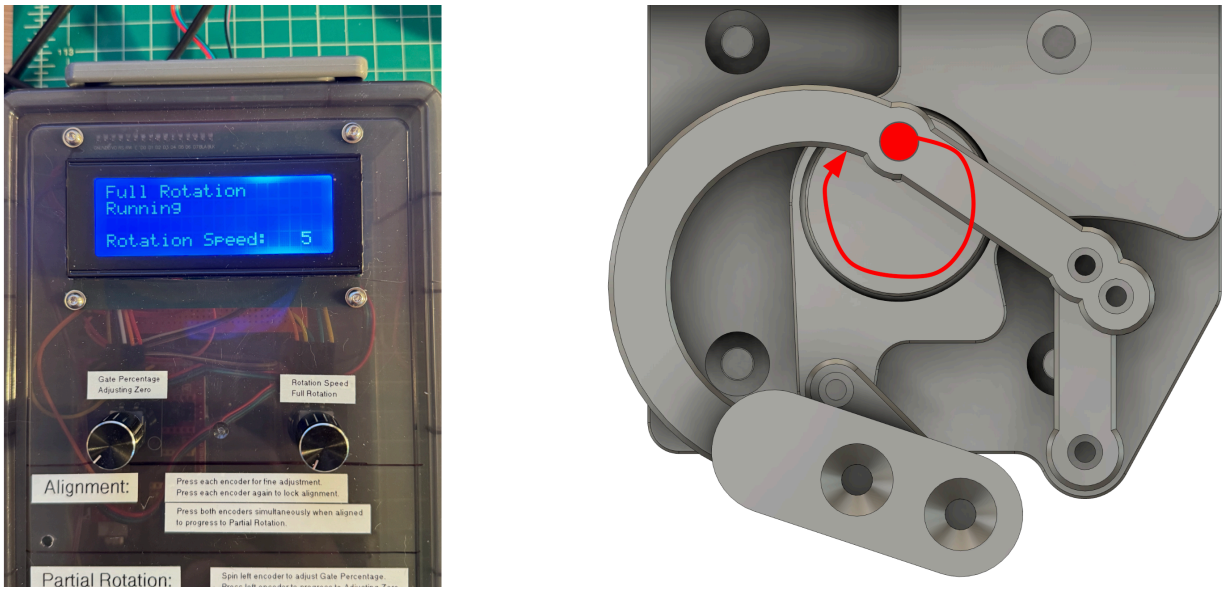

Figure 5-47: Image of the Full Rotation screen and the corresponding linkage motion

The previous code would move each stepper motor a single step at a time, sequentially outputting the movement commands which would sometimes get interrupted by an encoder rotation, leading to the motor desync. The new code instead generates the motion for the left motor relative to the stored zero position, while simultaneously driving the right motor to match the current position of the left motor. Additionally, any inputs from the knobs are only executed after each stepper has completed its queued movements, and if the menu is ever changed, the motors will slowly reset back to their stored zero positions and display WAIT… RE-ZEROING. These two ideas combine to ensure that the motors never lose their zero position or their position relative to the other.

Figure 5-48: Image of the WAIT... RE-ZEROING screen during a menu screen change

5.7.3. Summary and Recommendations

After a number of designs iterating on feedback from team members and collaborators, a final version was designed, manufactured, and delivered to the Indiana University team. This design met all the requirements outlined in section 5.2., being a successful and substantial iteration from the mechanism created in the previous year.

Unfortunately, it was at this point where the Indiana University team became too busy performing surgeries to complete testing on the mechanism within the required timeframe. No further communications were received from them, and therefore, no feedback was received on this version of the mechanism.

That being said, there are a handful of ideas related to the design that would prove beneficial if further iterations were to continue. First, adapting the mechanism for use in rats would allow for greater mechanism precision due to their larger size. This would potentially allow for an infinitely adjustable linkage by use of turnbuckles or similar device, or a more complex linkage to move the attachment point from the calf to the foot. More ambitiously, a 5-bar linkage could be programmed via. DLC output data from AI analyzation of a pre-injury walking video, allowing for bespoke kinematics to be applied on an individual-by-individual basis.

6. CS + ME Convergence Point

It was at this point that the CS and ME teams reconverged, but the timeline didn't allow Indiana University to provide proper testing, feedback, and results before the term's end. It is believed that a concrete result could have been reached given additional time and the necessary data from Indiana University.

Several aspects of the project would have benefited from closer communication with the Indiana team, as the physical distance and lack of required data from them often made it difficult to accomplish the stated goals.

On the CS side, the lack of videos of wider body motions could've provided a more useful model in relation to the ME side as the spinal cord rehabilitation mechanism focused on the locomotion of a mouse, the machine learning model focused on the grabbing motions of a mouse. Additionally, in relation to the 3D model of the mouse an improper lab setup for the specifications of DeepLabCut provided a roadblock.

On the ME side, the lack of having direct walking videos of their mice made creating a linkage difficult, since without reference footage of mice in their lab, a motion path instead had to be estimated from outside studies. Additionally, much of the struggle with fitment that arose from the first prototype could have been avoided with a mouse on which to intermittently test the sizing. While the second prototype was sent in two weeks before the term's end, the Indiana University team was so busy with performing surgeries that any further collaboration was unable to occur.

Because of this, even with communication to ensure each side of the project retained parallel trajectories, the paths were diverted just enough so that significant collaborative results within the proper timeframe became unattainable. If the motion of the mechanism were to simply be analyzed with the CS software to prove each of their ability, it being in possession of Indiana University would make this impossible. Even if a video were taken, the results would be inconclusive since the DLC trained exclusively on the forelimbs of real mice in a specific lab environment. This invariance in training data makes it so the only way to achieve a meaningful collaborative result, ranked in order of adherence to the original goal, by:

1. Doing additional training on the DLC model with similar videos of paralyzed mice. Then, the rehabilitation mechanism could be used on a paralyzed mouse, with its progress in the grabbing test being continuously analyzed in DLC during the rehabilitation process.
2. Retraining the DLC model with full body walking videos, then analyzing the kinematics of a mouse in the rehabilitation mechanism in DLC compared to the unconstrained natural movement.
3. Using a pre-made DLC training dataset to analyze the motion path of the rehabilitation mechanism, comparing the results to the pre-generated coupler curves.

However, lack of the necessary training data, lack of time to perform experiments, and the absence of the mechanism itself make obtaining a collaborative result of any degree by the project due date of the question.

7. Social Implications

As the previous project group reported, 250,000 to 500,000 people suffer from spinal cord injuries worldwide every year, affecting their ability to function in society and completely uprooting their previously established lives [20]. While kinematic training remains the preferred method for rehabilitation, results continue to be less than optimal, and whether this is because of a lack of understanding, technology, or execution remains to be known.

Though our project group lacked any medical knowledge, our project provided a novel approach to the issue of recovery from spinal cord injury, with the potential to use AI to recognize and diagnose any minute changes during the recovery process. If done right, this AI characterization paired with the adjustability of the mechanism could lead to breakthroughs in understanding exactly how to optimize this recovery process for complete pre-injury recovery.

Our aspiration is to garner interest in continued research from higher qualified scientists, eventually solving the mystery of complete recovery from spinal cord injury. Additionally, we hope to inspire students to participate in interdisciplinary collaboration to solve complex problems that would otherwise be out of scope.

8. Conclusion

Overall, the MQP team created the groundwork for a novel combination of computer science and mechanical engineering for medical research, using AI to analyze the recovery and possible effectiveness of an adjustable rehabilitative-assisted mechanism for mice suffering from spinal cord injury induced paralysis.

While, due to time constraints, no animal experiments were conducted to test the effectiveness of the combination of the two aspects of the project, the project showed promising progress compared to the previous years' work, giving insight into the potential capability of possible future studies. Continuation on the project could focus on the direct results from testing the mechanism, finalizing training on the DLC model to diagnose the severity of the SCI and quantify recovery, while also enhancing the mechanism to have an infinitely adjustable linkage. Hopefully, the scope of the research can eventually expand to include the exploration of application in human patients.

References

[1] C. Khan *et al.*, "Artificial intelligence for right whale photo identification: from data science competition to worldwide collaboration," *Mammalian Biology*, vol. 102, no. 3, pp. 1025–1042, Jun. 2022, doi: https://doi.org/10.1007/s42991-022-00253-3.

[2] J. Andreas *et al.*, "Cetacean Translation Initiative: a roadmap to deciphering the communication of sperm whales," *arXiv:2104.08614 [cs, eess]*, Apr. 2021, Available: https://arxiv.org/abs/2104.08614.

[3] Z. Tang, H. Yan, Y. Dai, and X. Jia, "Design of Adjustable Rehabilitative-Assisted Mechanism for Rodents Suffering from Spinal Cord Injury," *Digital WPI*, Mar. 24, 2023. https://digital.wpi.edu/concern/student_works/t722hd21m?locale=en.

[4] Suckow, Mark. *The Laboratory Mouse*. 3rd ed., CRC Press, 2023.

[5] J. Dienes, B. Hicks, C. Slater, K. D. Janson, G. J. Christ, and S. D. Russell, "Comprehensive dynamic and kinematic analysis of the rodent hindlimb during over ground walking," *Scientific Reports*, vol. 12, no. 1, Nov. 2022, doi: https://doi.org/10.1038/s41598-022-20288-3.

[6] Kathe, Claudia, et al. "The Neurons That Restore Walking after Paralysis." *Nature*, no. 7936, Springer Science and Business Media LLC, Nov. 2022, pp. 540–47. *Crossref*, doi:10.1038/s41586-022-05385-7.

[7] Nath, Tanmay, et al. "Using DeepLabCut for 3D Markerless Pose Estimation across Species and Behaviors." *Nature Protocols*, no. 7, Springer Science and Business Media LLC, June 2019, pp. 2152–76. *Crossref*, doi:10.1038/s41596-019-0176-0.

[8] "DeepLabCut for Multi-Animal Projects — DeepLabCut," *deeplabcut.github.io*. https://deeplabcut.github.io/DeepLabCut/docs/maDLC_UserGuide.html (accessed Mar. 26, 2024).

[9] Koonce, Brett. "ResNet 50." *Convolutional Neural Networks with Swift for Tensorflow*, Apress, 2021, pp. 63–72, http://dx.doi.org/10.1007/978-1-4842-6168-2_6.

[10] O'Shea, Keiron, and Ryan Nash. "An Introduction to Convolutional Neural Networks." *ArXiv*, Cornell University, Nov. 2015, doi: https://doi.org/10.48550/arXiv.1511.08458.

[11] Féraud, Raphael, and Fabrice Clérot. "A Methodology to Explain Neural Network Classification." *Neural Networks*, no. 2, Elsevier BV, Mar. 2002, pp. 237–46. *Crossref*, doi:10.1016/s0893-6080(01)00127-7.

[12] Pang, Bo, et al. “Deep Learning With TensorFlow: A Review.” *Journal of Educational and Behavioral Statistics*, no. 2, American Educational Research Association (AERA), Sept. 2019, pp. 227–48. *Crossref*, doi:10.3102/1076998619872761.

[13] Hao, Jiangang, and Tin Kam Ho. “Machine Learning Made Easy: A Review of *Scikit-Learn* Package in Python Programming Language.” *Journal of Educational and Behavioral Statistics*, no. 3, American Educational Research Association (AERA), Feb. 2019, pp. 348–61. *Crossref*, doi:10.3102/1076998619832248.

[14] J. R. Usherwood, “Legs as linkages: an alternative paradigm for the role of tendons and isometric muscles in facilitating economical gait,” *Journal of Experimental Biology*, vol. 225, no. Suppl_1, Mar. 2022, doi: https://doi.org/10.1242/jeb.243254.

[15] B. Gamble, “5-Bar Linkage Kinematic Solver and Simulator 5-Bar Linkage Kinematic Solver and Simulator,” 2020. Accessed: Nov. 05, 2022. [Online]. Available: https://scholarworks.uvm.edu/cgi/viewcontent.cgi?article=1416&context=hcoltheses.

[16] W. P. Mayer and T. Akay, “The Role of Muscle Spindle Feedback in the Guidance of Hindlimb Movement by the Ipsilateral Forelimb during Locomotion in Mice,” *eNeuro*, vol. 8, no. 6, Nov. 2021, doi: https://doi.org/10.1523/ENEURO.0432-21.2021.

[17] L. M. Sparrow, E. Pellatt, S. S. Yu, D. A. Raichlen, H. Pontzer, and C. Rolian, “Gait changes in a line of mice artificially selected for longer limbs,” *PeerJ*, vol. 5, p. e3008, Feb. 2017, doi: https://doi.org/10.7717/peerj.3008.

[18] J. Charles, Ornella Cappellari, and J. W. Hutchinson, “A Dynamic Simulation of Musculoskeletal Function in the Mouse Hindlimb During Trotting Locomotion,” vol. 6, May 2018, doi: https://doi.org/10.3389/fbioe.2018.00061.

[19] J. Dienes, B. Hicks, C. Slater, K. D. Janson, G. J. Christ, and S. D. Russell, “Comprehensive dynamic and kinematic analysis of the rodent hindlimb during over ground walking,” *Scientific Reports*, vol. 12, no. 1, Nov. 2022, doi: https://doi.org/10.1038/s41598-022-20288-3.

[20] W. H. Organization and I. S. C. Society, *International Perspectives on Spinal Cord Injury.* World Health Organization, 2013.

Appendix A: Arduino Code for Mechanism Control

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <AccelStepper.h>


LiquidCrystal_I2C lcd(0x27, 20, 4);

AccelStepper stepperLeft(AccelStepper::DRIVER, 2, 5); // Pins for left leg stepper
AccelStepper stepperRight(AccelStepper::DRIVER, 3, 6); // Pins for right leg stepper

const int encoderLeftCLK = 9;
const int encoderLeftDT = 10;
const int encoderLeftSW = 11; // Switch pin for left encoder
int leftLastCLK;
int leftAdjustmentState = 1;
int leftStepSize = 100;
bool leftLegAligned = false;

const int encoderRightCLK = 4;
const int encoderRightDT = 13;
const int encoderRightSW = 12; // Switch pin for right encoder
int rightLastCLK;
int rightAdjustmentState = 1;
int rightStepSize = 100;
bool rightLegAligned = false;

int gatePercentage = 0;
int rotationSpeed = 0;
bool movingForward = false;

bool offsetDone = false;

bool menuCheck = false;
bool motorRunning = false;
int menuScreen = 1;

//////////////////////////////////////////////////

void setup() {

  lcd.init();
  lcd.backlight();
  updateLCD();

  // Setup encoder pins
  pinMode(encoderLeftCLK, INPUT);
```

```
  pinMode(encoderLeftDT, INPUT);
  pinMode(encoderLeftSW, INPUT_PULLUP);
  leftLastCLK = digitalRead(encoderLeftCLK);

  pinMode(encoderRightCLK, INPUT);
  pinMode(encoderRightDT, INPUT);
  pinMode(encoderRightSW, INPUT_PULLUP);
  rightLastCLK = digitalRead(encoderRightCLK);

  stepperLeft.setMaxSpeed(3000);
  stepperLeft.setAcceleration(2000);
  stepperRight.setMaxSpeed(3000);
  stepperRight.setAcceleration(2000);

  Serial.begin(9600);
}

//////////////////////////////////////////////////

void loop() {

  if (menuScreen == 1) {
    Alignment(1);
    Alignment(2);
    stepperLeft.run();
    stepperRight.run();
  } else if (menuScreen == 2) {
    partialRotation();
  } else if (menuScreen == 3) {
    fullRotation();
   } else if (menuScreen == 4) {
    setZero();
    stepperLeft.run();
    stepperRight.run();
  } else {
    //motorRelease();
  }
}

//////////////////////////////////////////////////

void Alignment(int encoder) {
  int CLK, DT, SW, currentCLK, lastCLK;

  if (encoder == 1) {
    CLK = encoderLeftCLK;
    DT = encoderLeftDT;
    lastCLK = leftLastCLK;
```

```
} else { // encoder == 2
  CLK = encoderRightCLK;
  DT = encoderRightDT;
  lastCLK = rightLastCLK;
}

currentCLK = digitalRead(CLK);
if (currentCLK != lastCLK && currentCLK == HIGH) {
  if (digitalRead(DT) != currentCLK) {
    if (encoder == 1) {
      stepperLeft.move(leftStepSize);
    } else {
      stepperRight.move(rightStepSize);
    }
  } else {
    if (encoder == 1) {
      stepperLeft.move(-leftStepSize);
    } else {
      stepperRight.move(-rightStepSize);
    }
  }
}
if (encoder == 1) {
  leftLastCLK = currentCLK;
} else { // encoder == 2
  rightLastCLK = currentCLK;
}

if (digitalRead(encoderLeftSW) == LOW) {
  delay(100); // Debounce delay
  if (digitalRead(encoderRightSW) == LOW) {
    if (leftAdjustmentState && rightAdjustmentState == 3) {
      menuScreen = 2;
      leftStepSize = 100;
      leftAdjustmentState = 1;
      rightAdjustmentState = 1;
      stepperLeft.setCurrentPosition(0);
      stepperRight.setCurrentPosition(0);

      updateLCD();
      exit;
    }
  }
  if (digitalRead(encoderLeftSW) == LOW) {
    if (leftAdjustmentState == 1) {
      leftStepSize = 10;
    } else if (leftAdjustmentState == 2) {
      leftStepSize = 0;
```

```
    } else {
      leftStepSize = 100;
    }
    if (leftAdjustmentState == 3) {
      leftAdjustmentState = 1;
    } else {
      leftAdjustmentState++;
    }
    updateLCD();
  }
 }
 if (digitalRead(encoderRightSW) == LOW) {
  delay(100); // Debounce delay
  if (digitalRead(encoderLeftSW) == LOW) {
   if (leftAdjustmentState && rightAdjustmentState == 3) {
     menuScreen = 2;
     leftStepSize = 100;
     leftAdjustmentState = 1;
     rightAdjustmentState = 1;
     stepperLeft.setCurrentPosition(0);
     stepperRight.setCurrentPosition(0);

     updateLCD();
     exit;
    }
  }
  if (digitalRead(encoderRightSW) == LOW) {
   if (rightAdjustmentState == 1) {
     rightStepSize = 10;
    } else if (rightAdjustmentState == 2) {
     rightStepSize = 0;
    } else {
     rightStepSize = 100;
    }
    if (rightAdjustmentState == 3) {
     rightAdjustmentState = 1;
    } else {
     rightAdjustmentState++;
    }
    updateLCD();
  }
 }
}

//////////////////////////////////////////////////

void partialRotation() {
```

```
  updatePartialRotation(1);
  updatePartialRotation(2);

  stepperLeft.setMaxSpeed(rotationSpeed*100);
  stepperRight.setMaxSpeed(rotationSpeed*100);

  if (movingForward == false) {
    stepperLeft.moveTo(gatePercentage*45);
    stepperRight.moveTo(-stepperLeft.currentPosition());
  } else {
    stepperLeft.moveTo(-gatePercentage*45);
    stepperRight.moveTo(-stepperLeft.currentPosition());
  }

  if (stepperLeft.distanceToGo() != 0) {
    stepperLeft.run();
    stepperRight.run();
  } else {
    movingForward = !movingForward;
  }
}

////////////////////////////////////////////////

void updatePartialRotation(int encoder) {
  int CLK, DT, SW, currentCLK, lastCLK;

  if (encoder == 1) {
    CLK = encoderLeftCLK;
    DT = encoderLeftDT;
    lastCLK = leftLastCLK;
  } else { // encoder == 2
    CLK = encoderRightCLK;
    DT = encoderRightDT;
    lastCLK = rightLastCLK;
  }

  currentCLK = digitalRead(CLK);
  if (currentCLK != lastCLK && currentCLK == HIGH) {
    if (digitalRead(DT) != currentCLK) {
      if (encoder == 1) {
        if (gatePercentage != 10)
          gatePercentage++;
      } else {
        if (rotationSpeed != 10)
          rotationSpeed++;
      }
    } else {
```

```
    if (encoder == 1) {
      if (gatePercentage != 0)
        gatePercentage--;
    } else {
      if (rotationSpeed != 0)
        rotationSpeed--;
    }
  }
  if ((rotationSpeed == 0) || (gatePercentage == 0)) {
    motorRunning = false;
  } else {
    motorRunning = true;
  }
  updateLCD();
}
if (encoder == 1) {
  leftLastCLK = currentCLK;
} else { // encoder == 2
  rightLastCLK = currentCLK;
}

if (digitalRead(encoderLeftSW) == LOW) {
  delay(100); // Debounce delay
  if (digitalRead(encoderLeftSW) == LOW) {
    menuScreen = 4;
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("WAIT... RE-ZEROING");

    stepperLeft.setMaxSpeed(100);
    stepperRight.setMaxSpeed(100);

    stepperLeft.moveTo(0);
    stepperRight.moveTo(0);

    while ((stepperLeft.distanceToGo() != 0) || (stepperRight.distanceToGo() != 0)) {
      stepperLeft.run();
      stepperRight.run();
    }
    gatePercentage = 0;
    rotationSpeed = 0;
    updateLCD();
  }
}

if (digitalRead(encoderRightSW) == LOW) {
  delay(100); // Debounce delay
  if (digitalRead(encoderRightSW) == LOW) {
```

```
    menuScreen = 3;
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("WAIT... RE-ZEROING");

    stepperLeft.setMaxSpeed(100);
    stepperRight.setMaxSpeed(100);

    stepperLeft.moveTo(0);
    stepperRight.moveTo(0);

    while ((stepperLeft.distanceToGo() != 0) || (stepperRight.distanceToGo() != 0)) {
      stepperLeft.run();
      stepperRight.run();
    }
    gatePercentage = 0;
    rotationSpeed = 0;
    motorRunning = false;
    updateLCD();
   }
  }
}

////////////////////////////////////////////////

void fullRotation() {
  if (offsetDone == false) {
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("WAIT... RE-ZEROING");
   stepperLeft.setMaxSpeed(100);
   stepperLeft.moveTo(800);

   while (stepperLeft.distanceToGo() != 0) {
     stepperLeft.run();
   }
   offsetDone = true;
   updateLCD();
  }

  updateFullRotation(2);

  stepperLeft.setMaxSpeed(rotationSpeed*100);
  stepperRight.setMaxSpeed(rotationSpeed*100);
  stepperLeft.setSpeed(rotationSpeed*100);
  stepperRight.setSpeed(rotationSpeed*100);

  stepperLeft.runSpeed();
```

```
  stepperRight.runSpeed();

  stepperLeft.run();
  stepperRight.run();

  if (stepperLeft.currentPosition() == 1600) {
    stepperLeft.setCurrentPosition(0);
  }
  if (stepperRight.currentPosition() == 1600) {
    stepperRight.setCurrentPosition(0);
  }
}

////////////////////////////////////////////////

void updateFullRotation(int encoder) {
  int CLK, DT, SW, currentCLK, lastCLK;

  CLK = encoderRightCLK;
  DT = encoderRightDT;
  lastCLK = rightLastCLK;


  currentCLK = digitalRead(CLK);
  if (currentCLK != lastCLK && currentCLK == HIGH) {
    if (digitalRead(DT) != currentCLK) {
      if (rotationSpeed != 10)
        rotationSpeed++;
    } else {
      if (rotationSpeed != 0)
        rotationSpeed--;
    }
    if (rotationSpeed == 0) {
      motorRunning = false;
    } else {
      motorRunning = true;
    }
    updateLCD();
  }

  rightLastCLK = currentCLK;

  if (digitalRead(encoderRightSW) == LOW) {
    delay(100); // Debounce delay
    if (digitalRead(encoderRightSW) == LOW) {
      menuScreen = 2;
      lcd.clear();
      lcd.setCursor(0,0);
```

```
    lcd.print("WAIT... RE-ZEROING");

    stepperLeft.setMaxSpeed(100);
    stepperRight.setMaxSpeed(100);
    stepperLeft.setSpeed(100);
    stepperRight.setSpeed(100);

    stepperLeft.moveTo(0);
    stepperRight.moveTo(0);
    while ((stepperLeft.distanceToGo() != 0) || (stepperRight.distanceToGo() != 0)) {
      stepperLeft.run();
      stepperRight.run();
    }
    gatePercentage = 0;
    rotationSpeed = 0;
    offsetDone = false;
    motorRunning = false;
    updateLCD();
   }
  }
}

////////////////////////////////////////////////

void setZero() {
  int CLK, DT, SW, currentCLK, lastCLK;

  CLK = encoderLeftCLK;
  DT = encoderLeftDT;
  lastCLK = leftLastCLK;

  stepperLeft.setMaxSpeed(100);
  stepperRight.setMaxSpeed(100);

  currentCLK = digitalRead(CLK);
  if (currentCLK != lastCLK && currentCLK == HIGH) {
   if (digitalRead(DT) != currentCLK) {
    stepperLeft.move(25);
    stepperRight.move(25);
   } else {
    stepperLeft.move(-25);
    stepperRight.move(-25);
   }
  }
  leftLastCLK = currentCLK;

  if (digitalRead(encoderLeftSW) == LOW) {
   delay(100); // Debounce delay
```

```
    if (digitalRead(encoderLeftSW) == LOW) {
      menuScreen = 2;
      lcd.clear();

      while ((stepperLeft.distanceToGo() != 0) || (stepperRight.distanceToGo() != 0)) {
        stepperLeft.run();
        stepperRight.run();
        lcd.setCursor(0,0);
        lcd.print("WAIT... RE-ZEROING");
      }

      stepperLeft.setCurrentPosition(0);
      stepperRight.setCurrentPosition(0);

      updateLCD();
    }
  }
}

////////////////////////////////////////////////

void updateLCD() {
  lcd.clear();
  if (menuScreen == 1) {
    lcd.setCursor(0,0);
    lcd.print("Alignment:");
    lcd.setCursor(0, 1);
    if (leftAdjustmentState == 1) {
      lcd.print("Coarse Left");
    } else if (leftAdjustmentState == 2) {
      lcd.print("Fine Left");
    } else {
      lcd.print("Left Adjusted");
    }
    lcd.setCursor(0, 2);
    if (rightAdjustmentState == 1) {
      lcd.print("Coarse Right");
    } else if (rightAdjustmentState == 2) {
      lcd.print("Fine Right");
    } else {
      lcd.print("Right Adjusted");
    }
  } else if (menuScreen == 2) {
    lcd.setCursor(0, 0);
    lcd.print("Partial Rotation:");
    lcd.setCursor(0, 2);
    lcd.print("Gate Percentage: ");
    lcd.setCursor (18,2);
```

```
    lcd.print(gatePercentage);
    lcd.setCursor(0, 3);
    lcd.print("Rotation Speed: ");
    lcd.setCursor (18,3);
    lcd.print(rotationSpeed);
    lcd.setCursor(0, 1);
    if (motorRunning == true) {
      lcd.print("Running");
    } else {
      lcd.print("Stopped");
    }
  } else if (menuScreen == 3) {
    lcd.setCursor(0, 0);
    lcd.print("Full Rotation");
    lcd.setCursor(0, 3);
    lcd.print("Rotation Speed:");
    lcd.setCursor (18,3);
    lcd.print(rotationSpeed);
    lcd.setCursor(0, 1);
    if (motorRunning == true) {
      lcd.print("Running");
    } else {
      lcd.print("Stopped");
    }
  } else if (menuScreen == 4) {
    lcd.setCursor(0, 0);
    lcd.print("Adjusting Zero");
  } else {
    lcd.setCursor(0, 0);
    lcd.print("Motors Released");
    lcd.setCursor(0, 2);
    lcd.print("Realignment Required");
  }
}
```

Appendix B: DeepLabCut Python Script

```python
import sys      apollo, 5 months ago • App skeleton directory and updates to the pyt
import os
from deeplabcut import auxiliaryfunctions, train_network, create_training_dataset, ev
import time


def detect_shuffle(path):
    models = path + '/dlc-models'
    if not os.path.exists(models):
        print('Path does not exist')
        return -1
    iteration = os.scandir(models)
    max_shuffle = -1
    for directory in iteration:
        directory_name = directory.name
        if directory.is_dir() and directory_name.__contains__("iteration"):
            shuffle_number = ""
            for char in directory_name:
                if char.isnumeric():
                    shuffle_number = shuffle_number + char
            max_shuffle = max(int(shuffle_number), max_shuffle)
    return max_shuffle


def verify_paths(cfg, slash):
    print('Verifying Video Paths')
    p = cfg['project_path']
    cfg_yaml = p + slash + 'config.yaml'

    cfg_videos = dict(cfg['video_sets'].copy())

    repairs: int = 0
    for vid in cfg_videos.copy():
        vid = str(vid)
        video_index = str(vid)
        reconstruct = p + vid[video_index - 1:]
        if vid.find(p) == -1 and os.path.exists(reconstruct):
            cfg_videos[reconstruct] = cfg_videos.get(vid)
            cfg_videos.pop(vid, None)
            repairs = repairs + 1

    auxiliaryfunctions.edit_config(cfg_yaml, {'video_sets': cfg_videos})
    print(f'Paths verified with {repairs} repairs')
    return True


start = sys.argv[1].lower()
```

```
48  print(f'Using absolute path: {os.getcwd()}')
49  abs_path = os.getcwd()
50
51  slash_char = '/'
52  project = sys.argv[2]
53  if project[0] == '.':
54      project = project[1:]
55  if project[0] == '\\':
56      slash_char = '\\'
57  print(f'Project: {project}')
58
59  project_path = os.getcwd() + project
60  print(f'Project Path: {project_path}')
61  if not os.path.exists(project_path):
62      print('Project specified does not exist')
63      exit()
64
65  config_path = project_path + 'config.yaml'
66  if not project_path[-1] == '\\' and not project_path[-1] == '/':
67      config_path = project_path + slash_char + 'config.yaml'
68  print(f'Configuration File Path: {config_path}')
69  if not os.path.exists(config_path):
70      print('Configuration file not found')
71      exit()
72
73  config = auxiliaryfunctions.read_config(configname=config_path)
74  verify_paths(config, slash_char)
75
76  if start == "train":
77      max_iters = int(sys.argv[3])
78      display_iters = int(sys.argv[3]) / 25
79
80      print(f'Max iterations: {max_iters}')
81      print(f'Display interval: {display_iters}')
82
83      if not os.path.exists(project_path + 'training-datasets'):
84          print('Training dataset not found...')
85          print('Creating training dataset')
86
87          res = create_training_dataset(
88              config=config_path,
89              net_type='resnet_152',
90              augmenter_type='tensorpack'
91          )
92
93          if res is None:
94              print('Failed to create training dataset')      You,
95              exit()
```

```
97      print('Training network')
98
99      shuffle = detect_shuffle(project_path)
100     print(f'Shuffle number: {shuffle}')
101
102     start = time.time()
103     train_network(
104         config=config_path,
105         maxiters=max_iters,
106         saveiters=display_iters,
107         displayiters=display_iters,
108         shuffle=shuffle,
109         keepdeconvweights=True
110     )
111     end = time.time()
112
113     print(f'Training completed in {end - start}ms')
114
115 elif start == "eval":
116     print('Attempting Evaluate...')
117
118     start = time.time()
119
120     shuffle = detect_shuffle(project_path)
121
122     evaluate_network(
123         config=config_path,
124         Shuffles=[shuffle],
125         plotting=True,
126     )
127
128     end = time.time()
129
130     print(f'Finished eval in {end - start}ms')
131
132 elif start == 'analyze':
133     print('Attempting Analyze...')
134
135     videos = []
```

```
136
137     videos.append(project_path + slash_char + 'videos')
138     print('Analyzing all videos')
139
140     analyze_videos(
141         config=config_path,
142         videos=videos,
143         save_as_csv=True,
144         videotype='mp4'
145     )
146         apollo, 3 months ago • run script changes
147 elif start == 'calibrate':
148     calibrate_cameras(
149         config=config_path,
150         calibrate=True
151     )
152
153 elif start == 'extract':
154     if project.__contains__('3d'):
155         print('3d project detected!')
156         camera = sys.argv[3]
157         if camera is None:
158             raise 'Camera not specified'
159         i = 0
160
161         for cam in config['camera_names']:
162             if camera == cam:
163                 break
164             i = i + 1
165
166         video_set = []
167         print(f'Using video subset for: {camera} camera index {i}')
168         for video in config['video_sets']:
169             if str(video).__contains__(camera):
170                 video_set.append(str(video))
171                 print(f'{str(video)} with crop: {config["video_sets"][video].get("crop")}')
172
173         extract_frames(
174             config=config_path,
175             algo='kmeans',
176             mode='automatic',
177             userfeedback=False,
178             crop=True,
179             config3d=True,
180             extracted_cam=i,
181             videos_list=video_set,
```

```
182         )
183
184     extract_frames(
185         config=config_path,
186         algo='kmeans',
187         mode='automatic',
188         userfeedback=False,
189         crop=True
190     )
191
192 elif start == 'create':
193     video = [project_path + slash_char + 'videos' + slash_char + sys.argv[3]]
194     create_labeled_video(
195         config=config_path,
196         videos=video,
197     )
```

Appendix C: Compute Cluster Shell Script

```
#! /bin/bash
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --mem 1G

module load python/3.9.12/
module load cuda11.6/blas/
module load cuda11.6/fft/
module load cuda11.6/toolkit/

. ./tensorENV/bin/activate

python run.py extract /MQP-Apollo-Rowe-2023-11-25-3d side
#python run.py eval /MQP-Wrist-Mouse-Apollo-Rowe-2023-10-05
#python run.py train /MQP-Wrist-Mouse-Apollo-Rowe-2023-10-05 50000
#python run.py analyze /MQP-Wrist-Mouse-Apollo-Rowe-2023-10-05 *
```

Appendix D: Compute Cluster Shell Script Generator

```
import sys
import time

start = time.time()
project = sys.argv[1]
print(f'Project: {project}')
if project[-1] == '/' or project[-1] == '\\':
    project = project[:-1]

sh_script = f'''
#! /bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --mem 1G

module load python/3.9.12/
module load cuda11.6/blas/
module load cuda11.6/fft/
module load cuda11.6/toolkit/

. ./tensorENV/bin/activate

#python run.py extract {project}
#python run.py eval {project}
#python run.py train {project} 50000
#python run.py analyze {project} *
'''

with open(f'run_{project}.sh', 'w') as rsh:
    rsh.write(sh_script)

print(f'Created in {time.time() - start}ms')
```

Appendix E: Nunif Shell Script

```bash
#!/usr/bin/env bash

cd nunif || exit
. ./venv/Scripts/activate

current_directory=$(pwd)
search_dir="$current_directory"/tmp/images
out_dir="$current_directory"/tmp/out

if [ -d "$out_dir" ]; then
  echo "Output directory detected clearing space..."
  rm -rf "$out_dir"
  mkdir "$out_dir"
  mkdir "$out_dir/n-0"
  mkdir "$out_dir/n-1"
  mkdir "$out_dir/n-2"
  mkdir "$out_dir/n-3"
else
  echo "Making output directories at $out_dir"
  mkdir "$out_dir"
  mkdir "$out_dir/n-0"
  mkdir "$out_dir/n-1"
  mkdir "$out_dir/n-2"
  mkdir "$out_dir/n-3"
fi

if [ -d "$search_dir" ]; then
  for entry in "${search_dir}"/*; do
    substring="$(echo "$entry" | awk -F "/images/" '{print $2}')"

    output_directory="$out_dir/n-0/$substring"
    mkdir "$output_directory"
    echo "Upscaling 4x $substring directory noise param = 0"
    python -m waifu2x.cli --style photo -m noise_scale -n 0 -i "$entry" -o "$output_directory"
```

```
#    output_directory="$out_dir/n-1/$substring"
#    mkdir "$output_directory"
#    echo "Upscaling 4x $substring directory noise param = 1"
#    python -m waifu2x.cli --style photo -m noise_scale4x -n 1 -i "$entry" -o "$output_directory"
#
#    output_directory="$out_dir/n-2/$substring"
#    mkdir "$output_directory"
#    echo "Upscaling 4x $substring directory noise param = 2"
#    python -m waifu2x.cli --style photo -m noise_scale4x -n 2 -i "$entry" -o "$output_directory"
#
#    output_directory="$out_dir/n-3/$substring"
#    mkdir "$output_directory"
#    echo "Upscaling 4x $substring directory noise param = 3"
#    python -m waifu2x.cli --style photo -m noise_scale4x -n 3 -i "$entry" -o "$output_directory"
  done
else
  echo "Directory not found: $search_dir"
fi
```