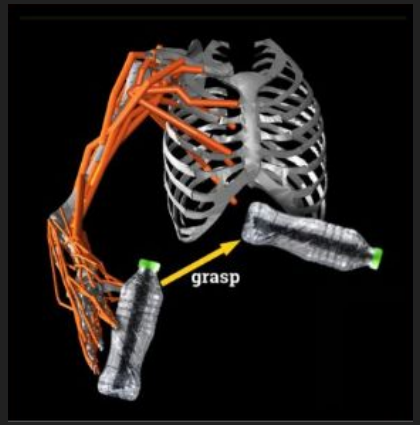
# ME 299 - PROJECT COURSE

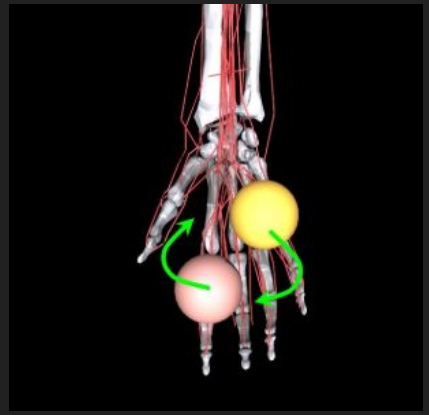
## Simulation of movement of musculoskeletal models of arm, elbow and finger

Hardik Jain(22110091)
Vidhi Shah(22110286)
Shrishti Mishra(22110246)

# OBJECTIVES

- The objective is to enhance the efficiency of musculoskeletal movement during tasks by optimizing the mechanics involved.
- Aimed to execute simple motions across different models such as MyoFinger, MyoArm, and MyoElbow.



# INITIAL WORK

Since the topics involved in this project were entirely new to us, we had to do learn the things involved in this project completely from scratch. We started by exploring and learning the main libraries involved and slowly started the implementation of the various features involved in the simulation of a model.

# INITIAL WORK

## Complete in-depth understanding of Myosuite

- MyoSuite is a collection of musculoskeletal environments and tasks simulated using the MuJoCo physics engine and wrapped in the OpenAI gym API to enable the application of Reinforcement Learning to biomechanical control challenges.

- Initially we started with by exploring and understanding the environments and models and then we got familiar with .xml files. The xml files have all the information about the muscles in a specific model.

# INITIAL WORK

- We looked at the various tasks that can be performed on the existing models and how variations can be induced into them.
- The different models that exist are:

  i) MyoFinger - 4 DoF and 5 muscle-tendon units

  ii) MyoElbow - 2 DoF and 6 muscle-tendon units

  iii) MyoHand - 39 muscle-tendon units

  iv) MyoArm - 27 DoF and 63 muscle-tendon units

  v) MyoLeg - 20 DoF and 80 muscle-tendon units

# INITIAL WORK

- The task variations that can be induced in these models for simulating real scenarios are:

  i) Sarcopenia

  ii) Fatigue

  iii) Tendon Transfer

  iv) Exoskeleton Assistance

  These variations cause a change in the normal behaviour of the model.

# INITIAL WORK

- By loading the xml files in MuJoCo, we looked at how the models behaved on actuating different types of muscles. We then analysed the mechanics of the muscles, tendons and bones in movement of our arms so that we can gain better understanding of all the muscles responsible for movement and corresponding activation parameter.


Varying the activations of the different muscles in the elbow model manually to understand the movements

# INITIAL WORK

- We started to comprehend example codes and in the process learn the different functions at our disposal and analyze different simulations and the reasons associated with each action, and adjusting parameters for visualization and graph analysis.
- After comprehending various codes, we attempted to take specific environments and play around with them.

# INITIAL WORK

- With all this mentioned in the previous slides, we were also learning how Gym environments are created and how those environments are simulated in MuJoCo. This was a crucial step as to work with any model, it is important to simulate it correctly and observe the results.

# IMPLEMENTATION OF WHAT WE LEARNT

# Modifying the existing model of an elbow and joint and execute various actions

We started with motivation to explore the ElbowRandomPose environment and tried to understand basics about it.

We considered adjusting the joint's movement angle sequentially, starting from 90 degrees, then moving to 60 degrees, and finally to 45 degrees, with the aim of altering the motion.

But, after exploring the environment, we realized that it is not possible to define the joint angle (qpos) which the muscle should exhibit as for flexion and extension we have certain number of muscles responsible for all the movements (For e.g. TRIlong, TRIlat, etc.) and they are assigned with different numerical value for varying the motion.

We then simulate in MuJoCo, change the value associated with each of the flexion and extension muscles and then try to find the activation values required for getting the desired value of qpos.

```
[10]  env.step(np.array([0,0,0,0,0,0]))
```

```
(array([3.8241e-01, 8.7084e-03, 1.1180e+00, 9.9996e-04, 9.9996e-04,
        9.9996e-04, 9.9996e-04, 9.9996e-04, 9.9996e-04]),
 -1.11843442388312,
 False,
 {'time': 0.04000000000000002,
  'rwd_dense': -1.11843442388312,
  'rwd_sparse': -1.1180261910484042,
  'solved': False,
  'done': False,
  'obs_dict': {'time': array(0.04),
   'qpos': array(0.3824),
   'qvel': array(0.0087),
   'act': array([0.001, 0.001, 0.001, 0.001, 0.001, 0.001]),
   'pose_err': array(1.118)},
  'visual_dict': {},
  'proprio_dict': None,
  'rwd_dict': OrderedDict([('pose', array(-1.118)),
               ('bonus', array(0.)),
               ('penalty', array(-0.)),
               ('act_reg', array(-0.0004)),
               ('sparse', array(-1.118)),
               ('solved', array(False)),
               ('done', array(False)),
               ('dense', array(-1.1184))]),
  'state': {'time': 0.04000000000000002,
...
    'body_quat': array([[ 1.    ,  0.    ,  0.    ,  0.    ],
           [ 1.    ,  0.    ,  0.    ,  0.    ],
           [ 0.5004,  0.5   , -0.5   , -0.4996],
           [ 1.    ,  0.    ,  0.    ,  0.    ],
           [ 1.    ,  0.    ,  0.    ,  0.    ]])}})
```

Understanding the values that the env.step() function returns was really crucial as that contained all the information of all the parameters of a model at every step of the simulation of the model. From here, we can extract the relevant values (eg: joint angles, joint velocity, error in that step, etc.) and carry out further analysis.

```
for _ in range(1000):
    env.mj_render()
    env.step(np.array([random.random(), random.random(), random.random(), random.random(), random.random(), random.random()]))
```

- Controlled the trajectory of the motion of elbow and finger through various changes made in the mujoco environment like setting the various parameters.
- Varied the activation values of each muscle of both models at each step manually.
- Achieved random motion because of random activation values of each muscle at each step.

```
Kp = 1

for i in range(100):
      q_d = 90*np.pi/180
      q = obs[0]

      e = q_d-q
      act = np.array([-1, -1, 1, 1, 1, 1])*e*Kp # + Derivativee comp.

      obs, rew, done, info = env.step(act)

      frames.append(env.sim.renderer.render_offscreen(
                  width=400,
                  height=400,
                  camera_id=0))

      print(obs[0]*180/np.pi)
```

Here, instead of random values, we specified the activation values responsible for activating the tensors and flexors in the act array and implemented a simple control strategy to achieve the motion. The logic we implemented was correct but the desired motion could not be achieved due to inaccurate activation parameters.

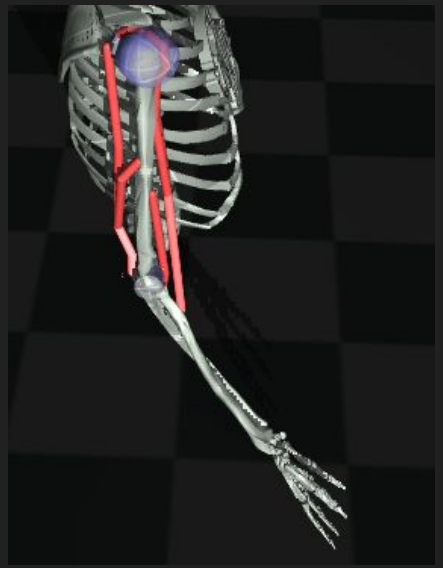
# Failure of the method involving manually setting the activations in each step

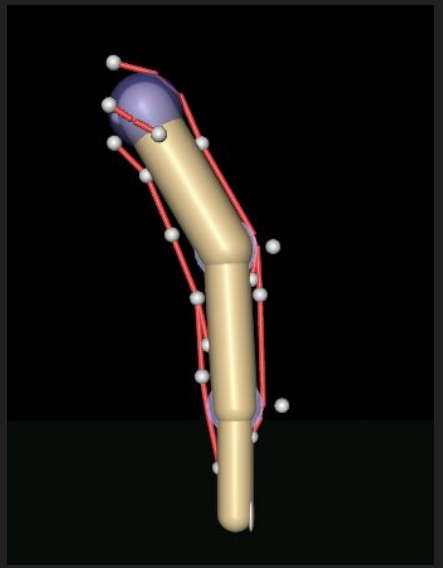
- It was difficult to define break condition, as we encountered issues during simulation. When the code failed to break the loop, it resulted in erratic vibration between two random points.
- We could not define the exact target angle using qpos and it was difficult to determine the exact muscle and required activation parameter responsible for certain movements and achieve the desired angle manually. For example, we could not determine the activations of the muscles if we wanted the elbow model to exhibit a 60 degree joint angle. For that, we needed RL training.

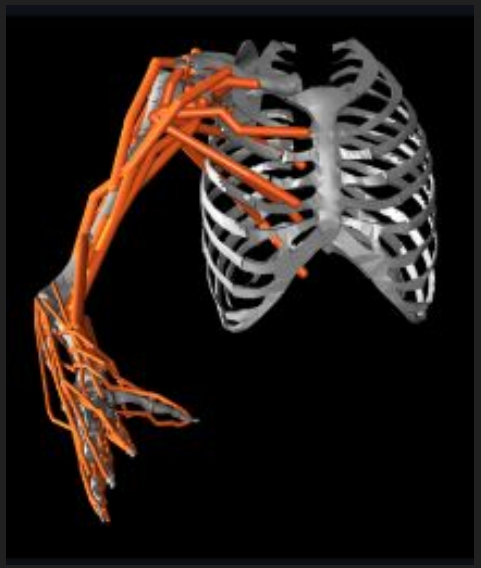
# RL IMPLEMENTATION (MAIN OBJECTIVE OF THE PROJECT)

# Model Selection

- Elbow
- Finger
- Arm





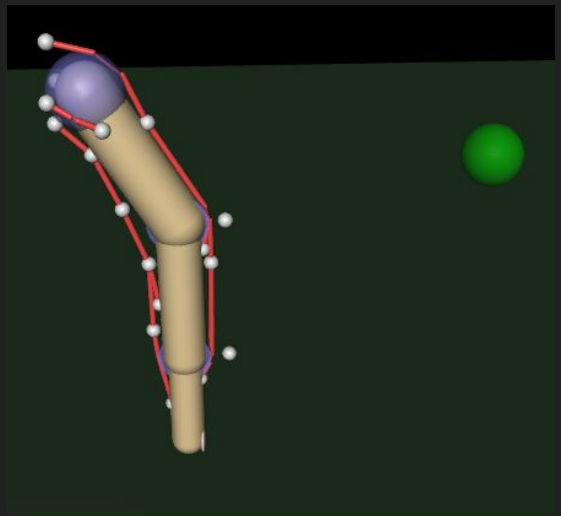


# Identification of different parts of the model

Using the XML file of the model, identify the different geometries, activation parameters, sites and responsible muscles present in the model and then train those parameters in order to reach the desired target.

## Task selection

Create a Reach environment - To make one of the sites of the model to reach a specific location. Example: In the elbow model, the wrist should reach the coordinates (x, y, z).



Green dot: Target location

Task - Tip of the finger should reach the green dot

# Environment creation

```
ENV_NAME = 'Reach-v0'
MODEL_PATH = "C:/Personal/Second year/Sem 2/Project Course/myo_sim/finger/myofinger_v0.xml"
register(id=ENV_NAME,
        entry_point='myosuite.envs.myo.myobase.reach_v0:ReachEnvV0',
        max_episode_steps=200,
        kwargs={
            'model_path': MODEL_PATH,
            'target_reach_range': {'IFtip': ((0.2, 0.05, 0.20), (0.2, 0.05, 0.20)), },
            'normalize_act': True,
        }
)
```

Name of the custom environment

Max number of steps per episode

Site which needs to reach a specific location

Coordinates of the target point

# Initialising the environment

```
env = gym.make('Reach-v0', seed=1)
obs = env.reset()

eval_env = gym.make('Reach-v0', seed=1)
eval_env.reset()

# Callback
eval_callback = EvalCallback(eval_env, best_model_save_path='./logs/',
                             log_path='./logs/', eval_freq=500,
                             deterministic=True, render=False)
```

Setting up the environment

Setting up a training env for assessing performance

Performance Evaluation

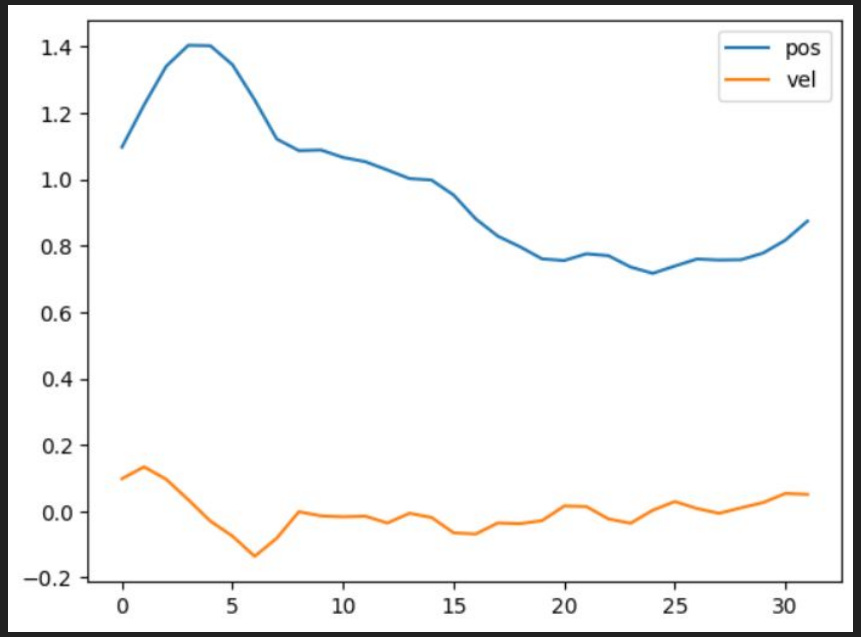
# Testing Trained RL Model

```
from time import sleep

model = PPO("MlpPolicy", env, verbose=1, device=device)
model.learn(total_timesteps=50000, callback=eval_callback)
model.save("myofinger_model")
model.load("myofinger_model")

done = False
while not done:
    act, _ = model.predict(obs)
    sleep(0.1)
    obs, reward, done, info = env.step(act)
    env.mj_render()
    print(done, act, obs)
```

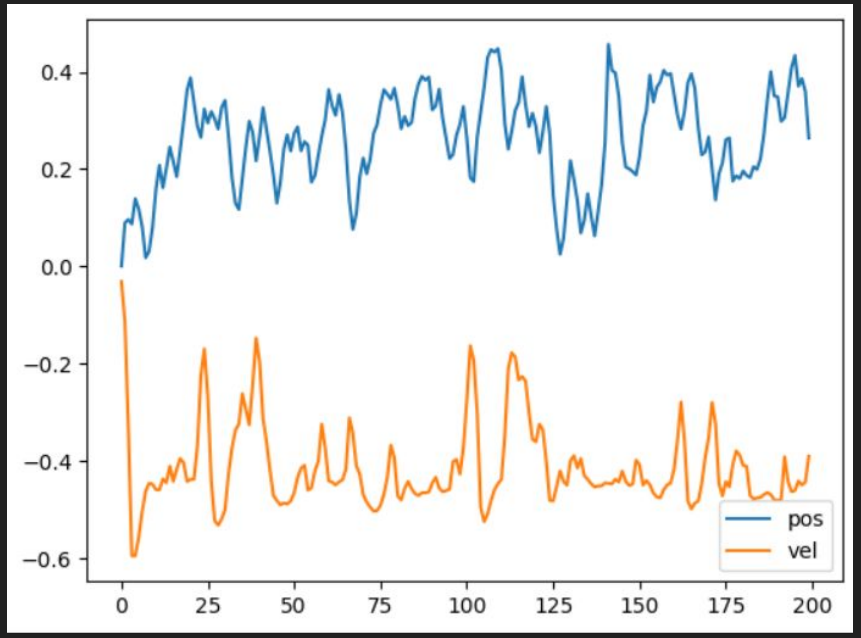
PPO Algorithm guides agent's learning by iteratively interacting with the environment, observing states, taking actions, and receiving rewards

In the above code, we implemented Reinforcement Learning in the finger model. Here RL part of the code trains a Proximal Policy Optimization (PPO) agent using the Stable Baselines3 Library.

# Results

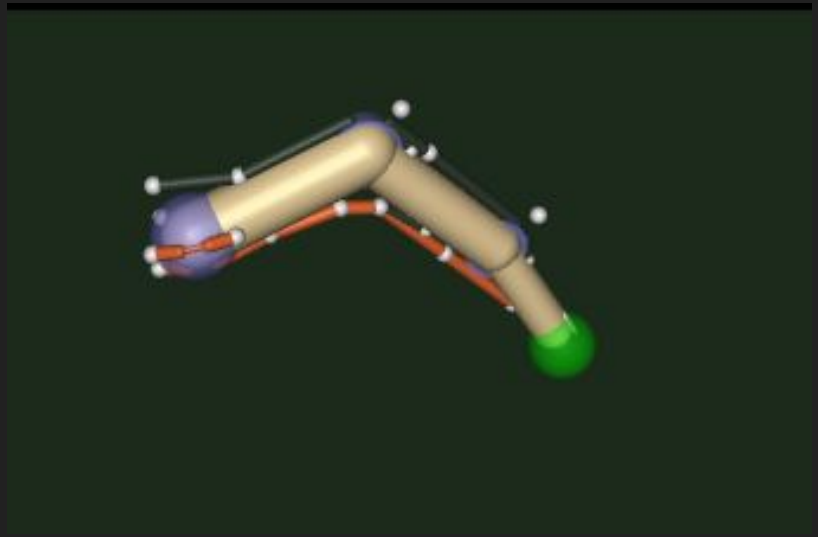



Elbow

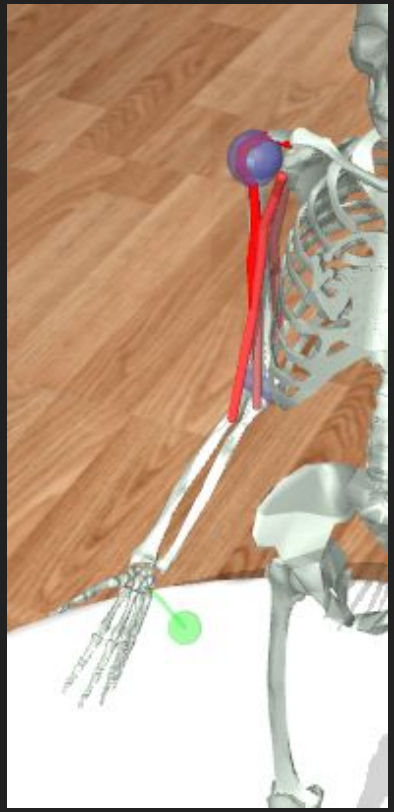



Finger

# Results



Final position of the finger



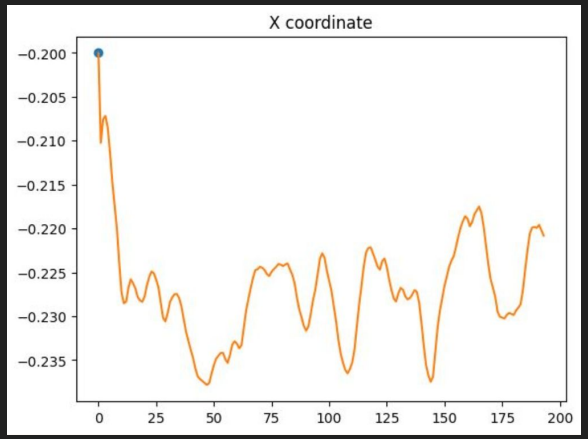
Final position of the elbow model

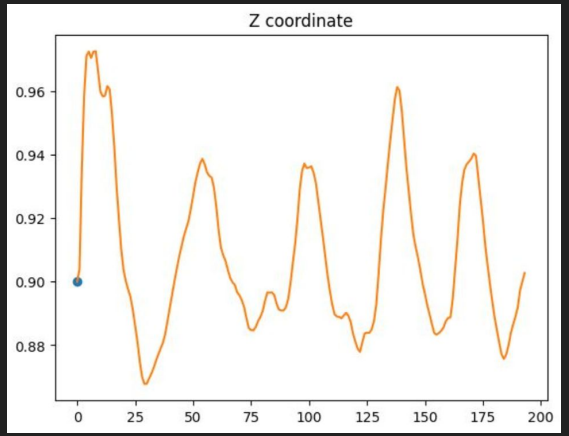
# Working on Myo Arm

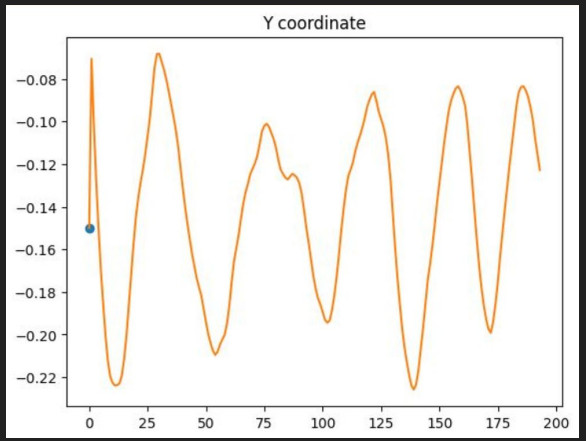
Task: Utilizing the Myo Arm model for drawing a circle on a 2D plane

How we did it: Implemented an RL algorithm, specifically the PPO baseline, with the Myo Arm model to like we did in other models to achieve task of drawing circle on the plane. We did not achieve the circular motion of the as we could not find a way of tracing the coordinates of the circle. Instead, we trained an RL model on the arm model and made it reach a specific target point.

# Results

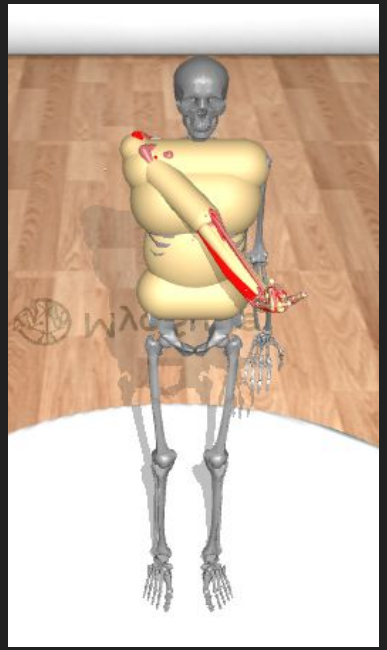



Myo Arm

# Results



Myo Arm

# THANK YOU!!