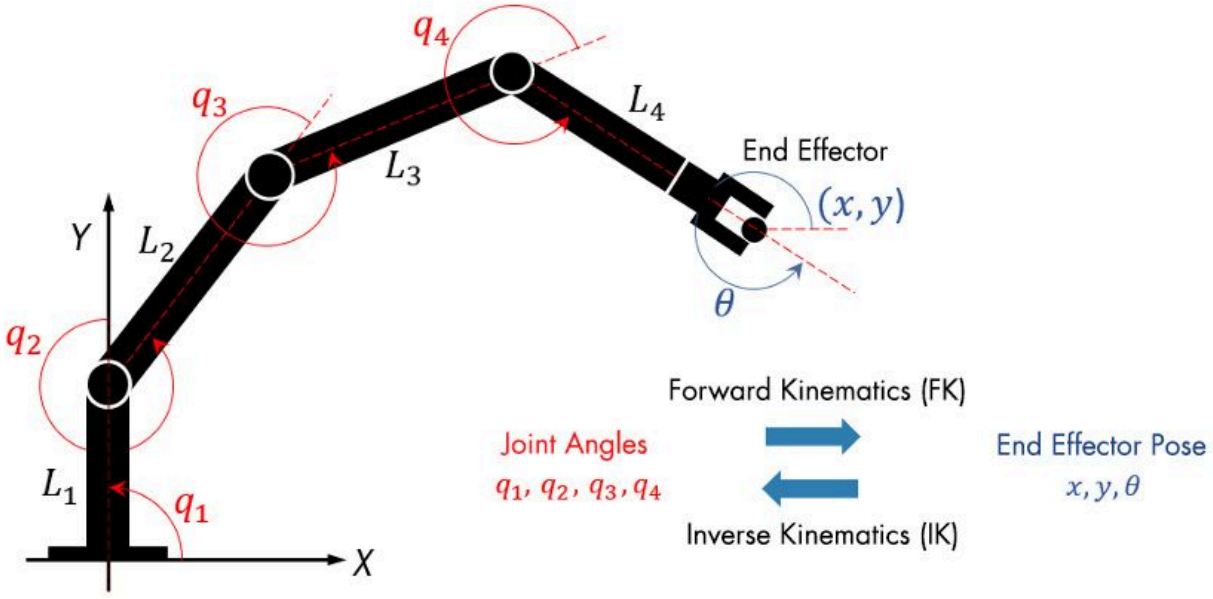
Period 1
Ian Shi
Group Name: abcdefghijklmnopqrstuvwxyz
Project Title: Inverse Kinematics (and PID control?)

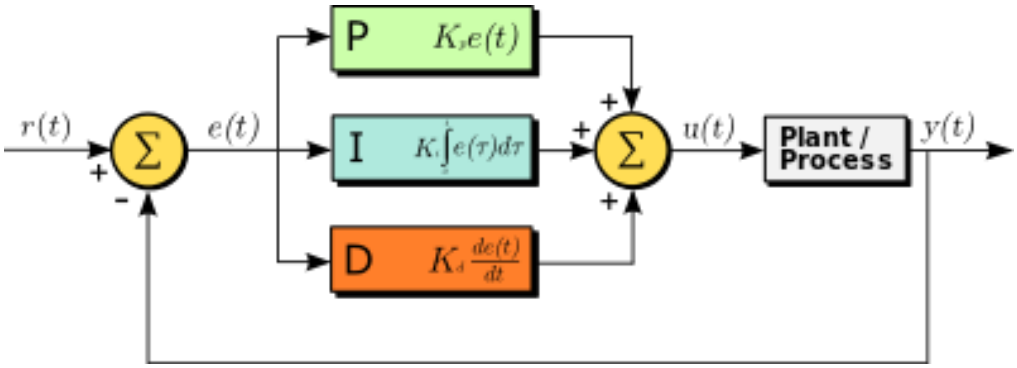
MEETING 1

Inverse kinematics is a process in robotics and computer graphics used to determine the joint parameters that provide a desired position for the end effector (the part of the robot or character that interacts with the environment, such as a hand or tool). It involves calculating the necessary angles or positions of the joints in a multi-jointed arm or limb so that the end effector reaches a specified target position and/or orientation.



The goal of this project is to create a simulation of a two or more jointed arm in processing and implement the necessary calculations to achieve the desired end effector position. The lengths of the arm segments will be configurable, and the end effector will try its best to track the position of the user's mouse on the screen.

The second goal of this project is to use a one-jointed arm to simulate a PID controller. PID control stands for Proportional-Integral-Derivative control, which is a widely used feedback control system. It combines three distinct control actions to maintain a desired output level or setpoint despite external disturbances or changes in system dynamics.



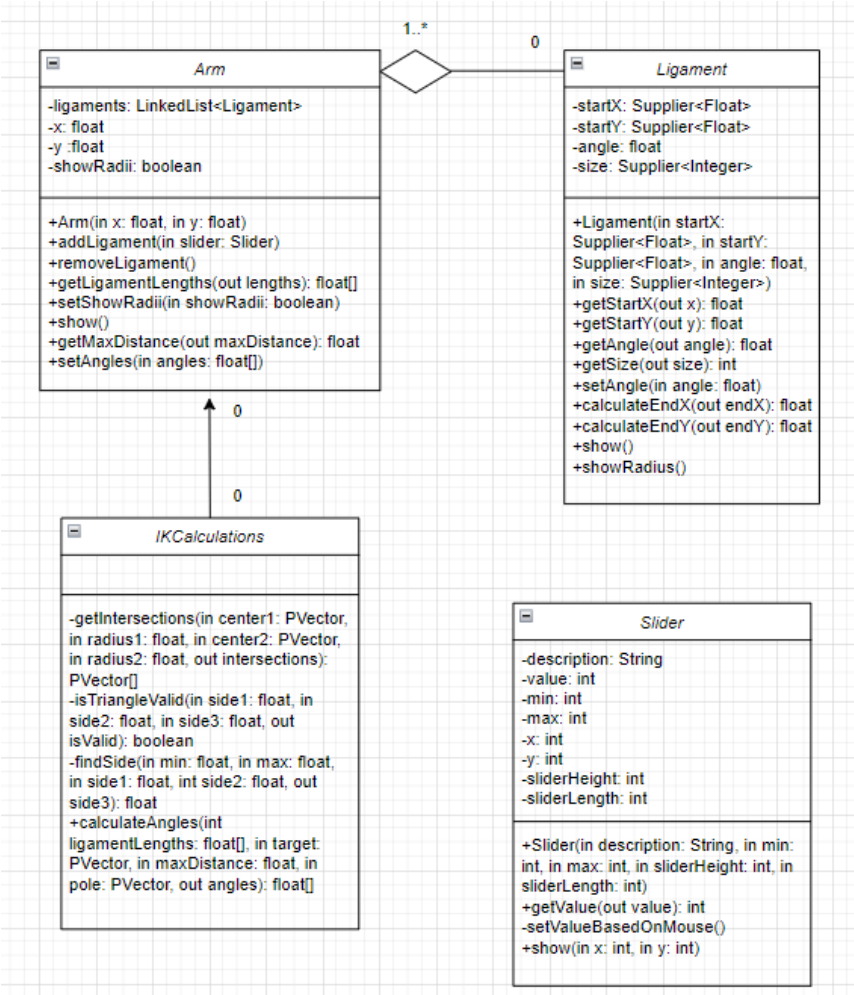
The first component is P (Proportional Control). The proportional term produces an output value that is proportional to the current error value.

The second component is I (Integral Control). The integral term produces an output value that is proportional to the accumulation of past errors. It sums up the error over time to eliminate any residual steady-state error that the proportional term alone cannot correct.

The last component is D (Derivative Control). The derivative term produces an output value that is proportional to the rate of change of the error.

The simulation will allow the user to configure the P, I, and D constants to allow the user to tune the PID controller.

Below is the UML Diagram for the Inverse Kinematics part of this project:



(UML Diagram and description for PID control will be added if I have the time to begin working on that)

How does it work?

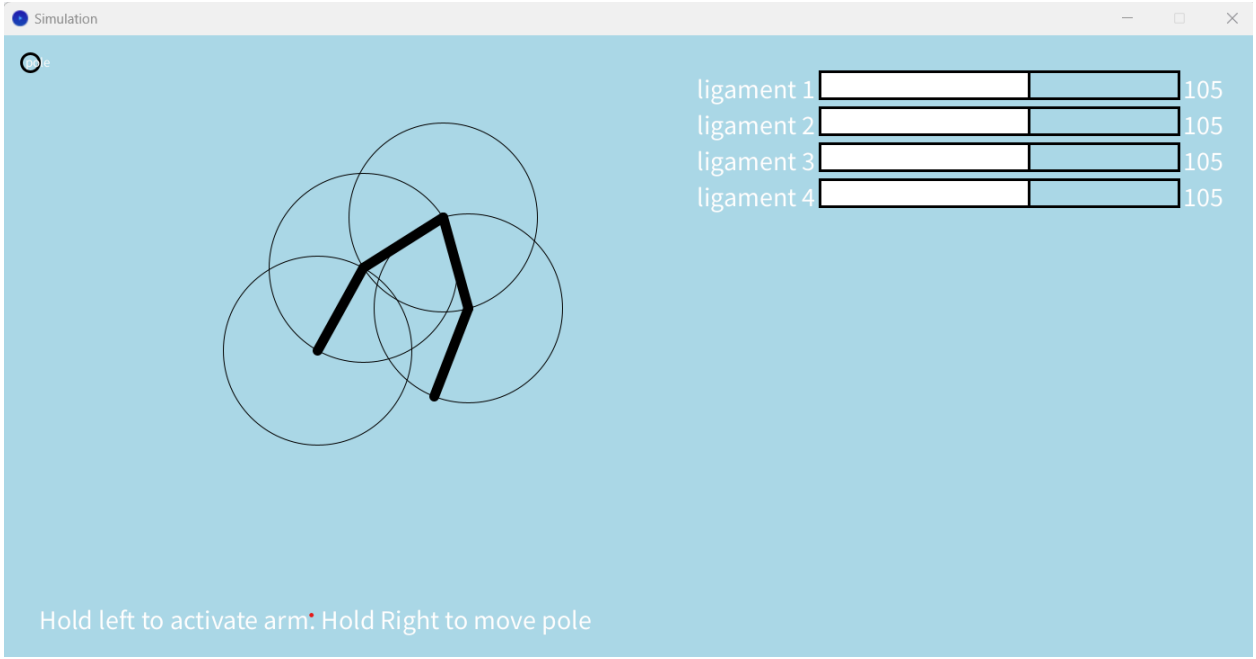
Inverse Kinematics:

As the user moves their mouse across the screen, the end effector of the arm will try its best to get to where their mouse is. I will add certain key binds or a slider that allows the user to change the length of each of the ligaments, which enables the arm to behave differently.

PID:

Similar to Inverse Kinematics, the arm will try its best to reach a certain setpoint. This time, instead of the mouse's position, the setpoint is configurable with a slider. The user will be able to tune PID constants to attempt to give the arm the best characteristics to reach its setpoint as it fights gravity. The user will also be able to configure the length of the arm to see how the length affects the required PID constants.

Functionalities/Issues



I've successfully implemented the calculations required to reach the desired angles to allow the end effector of the arm to achieve the desired position. I've also created sliders that make the length of the ligaments configurable.

When the user holds left click, the arm tries to track the user's mouse. When the user holds right click, the user can move the pole around.

The pole is an important part of the calculations required to reach the desired joint configurations. Without getting too deep into the math, the position of the pole alters the behavior of the arm at certain points.

One issue is that the arm is jittery, I'm not sure if this is how it's supposed to behave (maybe the desired joint configurations really do change that fast…) Unfortunately, this could be a side effect of the simpler method that I used to calculate desired angles. Instead of using linear algebra, I used only vector math and circles.

Another issue is this logic right here:

```
// not sure why this works lol
private static boolean isTriangleValid(float side1, float side2, float side3) {
  return (side1 + side2) >= side3 || (side1 + side3) >= side2 || (side2 + side3) >= side1;
}
```

For a triangle to be valid, all three of the conditions must be satisfied. For some reason, the simulation works when any of the three conditions are met. I plan to examine why this is.

Plans for the next meeting:

- Add buttons to add and remove ligaments
    - Automate the positioning of corresponding sliders
- Start with PID simulation
    - Create new branch
    - Create a new class that extends from Ligament that allows the ligament to have an angular velocity and acceleration
    - Create methods to apply torque (will be useful for applying gravity and PID outputs)