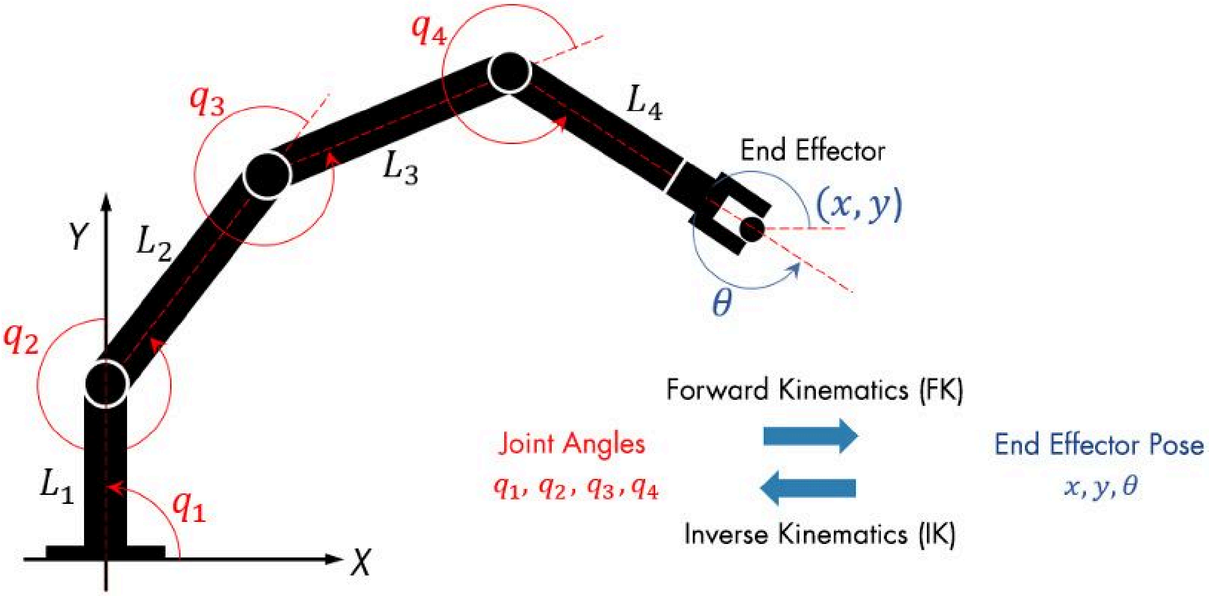
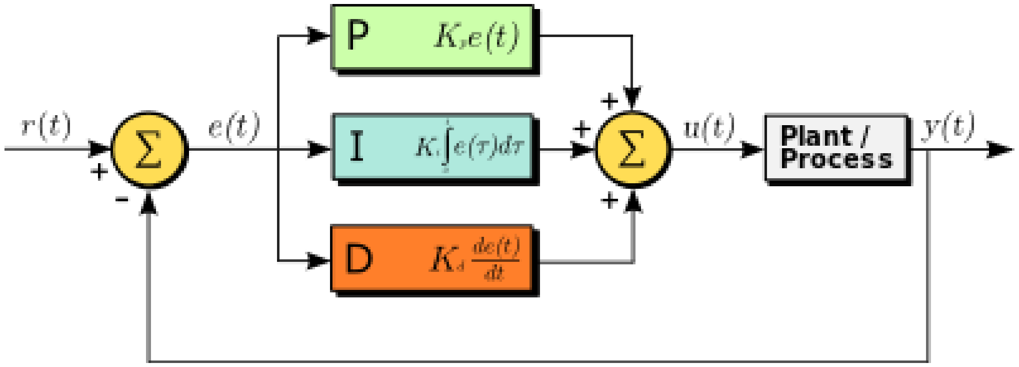
Period 1
Ian Shi
Group Name: abcdefghijklmnopqrstuvwxyz
Project Title: Inverse Kinematics (and PID control?)

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.

Next, if there is time, I plan 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.



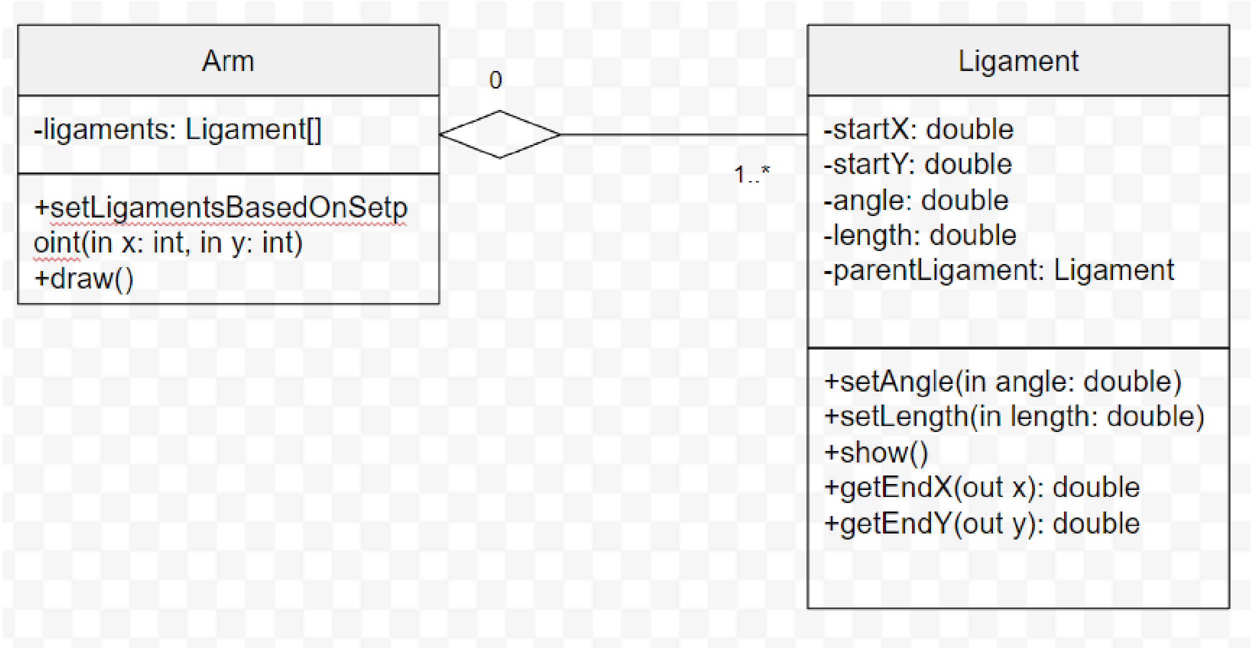
The first component is P (Proportional Control). The proportional term produces an output value that is proportional to the current error value.

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.

I plan the make the simulation allow for 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:



Arm

-ligaments: Ligament[]

+setLigamentsBasedOnSetpoint(in x: int, in y: int)
+draw()

Ligament

-startX: double
-startY: double
-angle: double
-length: double
-parentLigament: Ligament

+setAngle(in angle: double)
+setLength(in length: double)
+show()
+getEndX(out x): double
+getEndY(out y): double

How does it work?

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.

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