General structure:

* Obtaining Data:
  + Programs:
    - Single python script (or multiple if python does not support hand tracking).
  + What it records:
    - Sensor readings (capped at 10Hz because of sensors)
    - Hand limb angles (capped at 30Hz because of camera)
  + Stores the data into TensorFlow/machine learning-compatible format.
  + Options:
    - Specify the sapling rate of the sensors and camera.
    - Add profiles to store which limbs are being recorded for which user.
* Training:
  + Programs:
    - Unity/C# program
      * For simulating the test results.
    - Python machine learning script
      * For processing the data.
  + Unity/C# Environment details (for increase speed):
    - Runs at default 50 fixed updates per second (Time.fixedDeltaTime = 0.01f = 100Hz).
    - Runs at default time scaling (Time.timeScale = 100f).
      * Time faster than normal = (0.02/Time.fixedDeltaTime) \* (Time.timeScale)
        + So 0.01f fixedDeltaTime and 100f timescale would yield 200x speed.
    - **Note**: Increasing the time scale speeds up the entire program. So a %50 to the timescale would also increase the default 50Hz physics calculations to 75Hz.
* Use:
  + Programs:
    - Unity/C# program
      * For obtaining physics references.
    - Python machine learning script
      * For computing the forces to be applied within Unity.
  + Unity/C# Environment details:
    - Runs at default 50 fixed updates per second (Time.fixedDeltaTime = 0.02f = 50Hz).
    - Runs at default time scaling (Time.timeScale = 1f).
* General information about Unity/C# Program:
  + Computes the angles of the limbs.
  + Applies the forces directly given by the AI.
  + Does not have direct connection access to the sensor readings.
* General information about the Server/AI Python script:
  + Had direct connection to the Arduino and the sensor readings.
  + Computes the forces to be directly applied on which limbs (and in which direction).

Program AI training algorithm:

* Obtain features for feature layer:
  + Sensor readings
  + Limb angles
  + Limb angular velocities
  + Limb angular accelerations
* Computes and predicts output:
  + Outputs the result to the engine.
  + If (prediction is within +-5 degrees of expected)
    - then, continue to the next frame.
  + Else if (prediction outside +-5 degrees of expected)
    - then, restart update the model and restart the sequence.
* Repeat above until the entire data set is gone through within the satisfactory angle bounds.

Unity/Python detailed interaction:

* For training:
  + Unity runs the C# program which calls the python training script as a sub-process.
    - Creates model with 3\*number of limbs inputs.
    - Training script loads in the dataset.
    - Returns acknowledgement to C# (“Ready”)
    - Sends list of starting angles for the Unity model.
  + C# saves the starting angles.
    - Sets up the reset function (vel = 0, acc = 0)
  + Once C# is ready, repeats the following, until python script sends end of training message (“Done”):
    - C# sends set of angles, angular velocities, angular accelerations, indexed for every finger limb, separated by spaces.
      * E.g. “ang1 vel1 acc1 ang2 vel2 acc2 ang3 vel3 acc3…”
      * **NOTE:** **For the first iteration, no accelerations.**
    - Python receives the message, runs the numbers through the model.
    - Python returns one of the following messages:
      * “Next”: Algorithm is successful, pass in the next data point.
      * “Failed”: Call’s ResetHandPhysics() on the model. Resets the sequence.
      * “Quit”: to stop the C# script.

First iteration of the program:

* Python computer vision reads and stores finger angles and sensor readings at **10Hz**.
  + No profiling, limited details on the data.
* Unity/C# Script runs at default speeds (both in training and in use).

Second iteration of the program:

* Python data gathering procedure is now scalable and can be profiled for different users.
* Unity/C# Script can training speeds can be set to a variable speed grater than 10Hz.