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 2(3 including acceleration)\*number of limbs + number of sensors = number of 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.
      * “Reset”: Call’s ResetHandPhysics() on the model. Resets the sequence.
      * “Quit”: to stop the C# script.
  + SEE TIME LINE BELOW FOR DETAILS.

Model details:

* Iteration #1:
  + 3 hidden layers (# of neurons per layer = # of inputs)
  + activation = “linear”

**Order of limbs (Protocol for storing limbs):**

* Thumb, index, middle, ring, pinky
  + For each: proximal, middle, distal limb
* Example:
  + Thumb proximal, thumb middle, thumb distal, index proximal, index middle, etc.

**C#-Python Training Protocol:**

|  |  |  |
| --- | --- | --- |
| Step: | C# Training Script | Python Training Script |
| Reset-0 | - Receives “Reset” from Python  - Prints to console the resetCount. | - Python sends “Reset” to C# |
| Reset-1 | - Resets all velocities and acceleration to 0.  - Sets all angles to start angles.  - Sets sequenceStartTimerMs = Time.CurrentMs() |  |
| Reset-3 | - Sends “Ready” to Python script | - Receives “Ready” from C# script |
|  |  |  |
| Loop-0 | - Parses and stored the nextFrameTime. | - Sends time of the next frame to C# |
| Loop-1 | - Waits until currentTime > nextFrameTime | - Waits (for data from C#) |
| Loop-2 | - Iterates through times and records:  + angles  + angular velocities | - Waits (for data from C#) |
| Loop-3 | - Sends currentTime to python | - Receives currentTime from C#  + Checks if localFrameTime < currentTime < nextFrameTime: |
| Loop-4 | - Sends recorded limb data | - Receives and parses the data from C# script. |
| Loop-5 | - Receives next command from Python | - If currentFrame# == totalFrames, sends “Quit”  + Saves the model. Exits.  - Checks the time condition above (from Loop-3):  + If the statement is true, sends “Next” (Only if it passes the threshold condition below too).  + If the statement is false, sends “Reset” and resets the training sequence (do not use data to correct the model).  - Compares received limb-data to training data:  + If current limb data is not within threshold, reset the training sequence, correct then model and ONLY THEN, send “Reset” to the C# script.  + If passed the threshold, send “Next” to C# script. |
| Loop-6 | Processes the received commands:  + If “Reset”, calls reset function.  + If “Quit”, stops the C# script  + If “Next”, receives torques from python script; applies the torques. | - If previous statement is “Next”:  + Run the data through the model and sends back the torques to apply to each limb. |
| Loop-7 | - Send “Ready” to python script | - Wait until receives “Ready” from C# script, then loops back. |

Keras model definitions:

* compile()
* fit(): Process the input/output, modifies the model (“fits” it).
* train\_on\_batch(): “Runs a single gradient update on a single batch of data”
* evaluate()
* predict()
* Training data set: Data used to train the model.
* Validation data set: Used along-side training data to evaluate the model while it is training.
* Test data set: Data which is used to evaluate the performance of the model when the model is “complete”.

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.