LEAP-Trainer: A Single-Frame Gesture Recording & Recognition System

Filip Klepsa
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
fklep001@fiu.edu

***Abstract*— The following paper proposes a single-frame gesture recording and recognition system utilizing the compact and affordable Leap Motion Controller. The English alphabet in American Sign Language is the set of gestures recorded, and recognition tested on. Applying K-nearest neighbor on recorded and trained frame vector data, experimental results obtained in this study show an average recognition rate of approximately 78% versus a test trained sample of 300 recorded frames per letter with accuracy increasing as the set of trained recorded frames grows. Additional topics covered in this paper include detailed discussions on the Leap Motion Controller and its attributes, data collection, features used for machine learning, the accuracy of specific letters, parameter settings influencing accuracy, user study results, and previous research performed.**

***Keywords*—American Sign Language, Leap Motion Controller, LEAP-Trainer, K-Nearest Neighbor Algorithm, Machine Learning Algorithm.**

I. Introduction

The Leap Motion Controller (LMC) is a commercialized, palm-sized, infrared sensor designed to make capturing hand and finger movements in 3D space affordable [1]. Applying machine learning methods, this paper proposes to extend the usefulness to, or re-validate use of the LMC device for single-frame gesture recording and recognition. As the backbone of the American deaf culture, gesture recognition on American Sign Language (ASL) could potentially become an educational tool for children and adults [2].

K-nearest neighbor algorithm (K-NN) is a non-parametric machine learning algorithm used for both classification and regression [5]. In K-NN, objects are classified by a majority comparison to its neighbors. K-NN is instance-based, meaning the function is approximated locally, and all computation is deferred until the moment just before classification. Due to its simplicity versus fellow machine learning algorithms (MLAs) as well as its consistency, error rates in recognition converge to zero as $n \to \infty$. Making it an ideal algorithm for this experiment and any future endeavors with this single-frame gesture recording and recognition software (henceforth to be known as the LEAP-Trainer).

Compared with existing ASL recognition systems using the LMC or other platforms, the LEAP-Trainer provides to the user the functionality to create and recognize any single-framed gesture imaginable within the limitations of the LMC device, all done in 3D and in real-time. Gesture recognition approaches utilizing optical, infrared sensors like the LMC have been performed in the past. Some achieved using a Parallel hidden Markov model approach [3]. Others similarly achieved using a K-NN approach with restricted applications in ASL [5]. To summarize, the following contributions have been made:

- Highlight the capabilities of the LEAP-Trainer as well as the experimentation results concluded.
- Show the applicability outside the scope of ASL with a comparable or better classification accuracy.
- Implementation done in C#, making it easily adapted to be compatible in Virtual Reality or Augmented Reality environments using the Unity engine.
- Vectors represented as a JSON, allowing the data to be used in a light-weight web application.

To facilitate flow, the format of this paper will take the following form: Begin with a detailed look at the LMC, its specifications and how it collects data. Followed shortly by the LEAP-Trainers training method. Then a glance at the features used for machine learning, the accuracy of specific letters, and parameter settings influencing accuracy. Finally concluded with the user study and its experimentation results.

II. Leap Motion Controller Specifications & Data Collection

A. Specifications

The LMC is an optical, infrared sensor boasting a 150-degree field of view. The valid range of the device extends approximately 25 to 600 millimeters above the device (1 inch to 2 feet) [1]. Tracking and detection perform best when the device has a bright, high-contrast view of the users hands. Its drivers combine sensor data with a built-in internal model of the human hand to assist with tracking.

The LMC utilizes a right-handed 3D Cartesian coordinate system, with the origin centered atop the device [1]. The X and Z axis are on the horizontal plane, while the Y-axis is vertical. For the X and Z axis, positive values increase to the right/toward the user and decrease to the left/away from the user, concerning the origin. The Y axis is always positive.

B. Data Collection

The LMC polls data with the following units unless otherwise noted: millimeters, microseconds, millimeters per second, radians [1]. As the device tracks the users' hands, it provides updates as a frame of data. Each frame is an object in the LMC Application Programming Interface (API) representing each part of the hand as its own tracked entity [6]. These entities include hands, fingers, tools, and factors that describe the overall motion within the scene. The LEAP-Trainer provides full user control of the frame rate, measured as capture rate ("Time Out" in the application) in milliseconds. Depending on the speed of the computers equipped processor, a target value of 100 to 50 milliseconds between frames (10-20 frames per second) while recording is recommended to ensure the data is captured and stored accurately.

III. The Training Method

After the user has captured frame vector data that accurately represents the gesture they wish to train to the LEAP-Trainer, they must at that point convert that data into something a bit more classification friendly. Then and only then will the software be ready to interpret gestures made in real-time. It is essential to keep in mind that the frame data captured will contain all information about the hands in a well-formed, yet raw, JSON format. The JSON file will contain the frame ID, hand type, hand pitch/roll/yaw, arm direction, wrist/elbow position, hand center location, finger bone direction, finger bone start position, finger bone end position, for every single finger. When making this data classification-friendly, the software calculates the angle between the users' hands and each of their finger bones; then it finds the distance between these features. The result is saved to a new file within its corresponding gesture directory, represented by `frame#_angle.txt`. The LEAP-Trainer is now ready to be used to interpret the users' gestures. The calculation is achieved through the following three functions:

1) AngleBetween(): The function receives two parameters as input, `vector1` and `vector2`, where both are of type Vector. These vectors contain the information of how the hand exists in 3D space relative to the LMC. This returns the angle in degrees between `vector1` and `vector2` of type double.

2) DistanceBetween(): The function receives two Cartesian points as input, `vector1` and `vector2`, where both are of type Vector. The vectors contain the same information as used in $AngleBetween()$. This returns the magnitude of the distance between `vector1` and `vector2` of type double.

3) AngleBpointBetween3Point(): This function receives three Cartesian points as input, `A`, `B`, and `C`, where all three are of type Vector. These vectors contain the information of how each finger exists relative to the center of the hand. This returns the angle in degrees created between `BA` and `BC` of type double. For example, one sample angle created by LEAP-Trainer is shown in Figure 1.

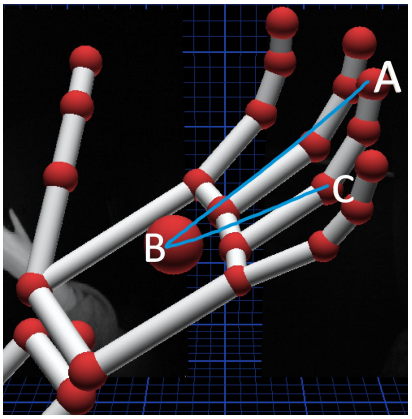

Fig. 1: LEAP-Trainer returns the angle created by `BA` and `BC`

IV. Machine Learning & the LEAP-Trainer

A. Features Used for Machine Learning

As touched on in the previous section, many features captured in a frame by the LMC API are not suitable for gesture recording and recognition. Therefore, it is essential to understand the data captured. There are four bones which are tracked by the LMC.

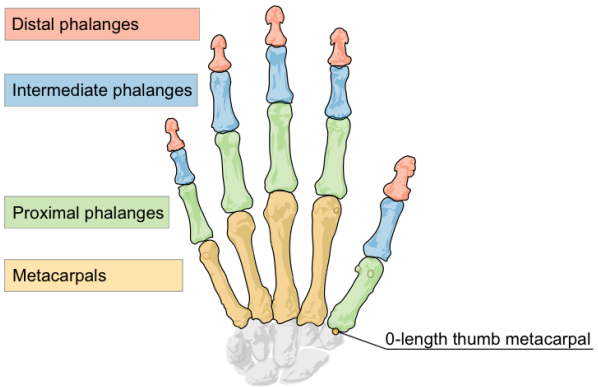

Fig. 2: Hand structure seen by LMC [1]

- The Metacarpal - the bone inside the hand connecting the finger to the wrist (except for the thumb).
- The Proximal Phalanx - the bone at the base of the finger, connected to the palm.
- The Intermediate Phalanx - the middle bone of the finger, between the tip and the base.
- The Distal Phalanx - the last bone at the end of the finger.

Post-training, when the angles have been calculated and stored, the user will want an interpretation. Classification is achieved by comparing the angles collected and its implementation is shown in Algorithm 1. When data is captured for the interpreter, it will get frame data (called `frame A`). It will convert `frame A` to an angle by using the methods in the previous section. Next, it is time to compare angle, θ, with each angle file for all trained gestures. To perform

the classification, the LEAP-Trainer needs the upper and lower limit value of the perceived finger variation within each capture, $upper$ and $lower$ respectively.

The correct gesture name is returned by Algorithm 2. First, Algorithm 2 retrieves the locations where the angle data is stored, $signAngleTxt$. Then it parses the data in $signAngleTxt$ for all the files containing `_angle` within their name. For all data, the training object, $trainObj$, and the comparison Object, $compareObj$, is loaded which was created by Algorithm 1. Then, it finds all the occurrences where the data is $false$ for all values in the training set verses the comparison set and chooses the gesture which has the least amount of fail counts stored in $failCount$.

Algorithm 1 compareTestAndTrainData

Input: Captured data converted to a JSON object type, $testObj$, and trained data converted to a JSON object type, $trainObj$.
Output: JSON object containing comparative parameters, $compareObj$.

$alpha = 0$
for each key in $keys$ **do**
  if $count == 0$ **then**
    $alpha = testValue - trainValue$
    compareObj.Add(key, $alpha$)
  else
    $testAlpha = trainValue + alpha$
    $change = \frac{testValue}{trainValue} * 100$
    compare.Obj.Add(key, $change$)
    if $lower < change < upper$ **then**
      $countAccurate++$
    else
      $countInaccurate++$
    end if
  end if
  $count++$
end for
if $countAccurate - countInaccurate > 12$ **then**
  compareObj.Add("isSign", $true$)
else
  compareObj.Add("isSign", $false$)
end if
return $compareObj$

Algorithm 2 SignNameFromLMFrameByListData

Input: LMFrame containing captured data, $lmFrame$.
Output: String containing gesture math and comparison result, $returnSignName$.

$currentFailCount$ = 2000
for each $trainKey$ in $dictTrainTesult.Keys$ **do**
  for each $hand$ in $hands$ **do**
    $dataAtKey$ = getListFileFromSign($trainKey$)
    for each $trainedPath$ in $dataAtKey$ **do**
      if trainedPath.Contains("`_angle`") **then**
        $signAngleTxt$ = File.ReadAllText($trainedPath$)
        $preTrainObj$ = JOBject.Parse($signAngleTxt$)
        $handTrainkeys$ = $preTrainObj$.Properties().Select($p \rightarrow p.Name$).ToList()
        for each $handTrain$ in $handTrainkeys$ **do**
          $trainObj$ = preTrainObj[$handTrain$]
          $compareObj$ = compareTestAndTrainData($testObj$, $trainObj$)
          $isSign$ = compareObj["isSign"]
          if $!isSign$ **then**
            $failCount++$
          end if
          $total++$
        end for
      end if
    end for
  end for
  builder.AppendLine($trainKey$, $failCount$, $total$)
  if $failCount < currentFailCount$ **then**
    $lessFailCountkey$ = $trainKey$
    $currentFailCount$ = $failCount$
  end if
end for
$returnSignName$ = $lessFailCountKey$ + builder.ToString()
return $returnSignName$

B. Accuracy

When attempting interpretation users will notice that several letters will share failCounts, this occurs due to the similarity of the angle magnitudes. Unfortunately, this occurs due to the limitations of the LMC as well as the algorithms approach. As mentioned in section II, the LMC performs best when it has a bright, high-contrast view of the users hands. Many letters require the user to position one finger above another which can cause occlusion. Occlusion forces the LMC to make an approximation of how the users fingers could be positioned. Sometimes it gets it right while other times it can be dramatically off.

The letters J and Z are completely unclassifiable within a single frame because they require motion to be represented. The accuracy of M, N, and T are unreliable due to occlusion. G and Q can be confused with each other, similarly With "A" and "S", due to their resemblance in their representation, which causes similar angles to be generated. If they manage to fall within the acceptable boundaries, then the LEAP-Trainer will consider it a pass even if it wasn't the users' intended gesture.

C. Parameter Settings Influencing Accuracy

Despite many of the shortcomings, there are a few ways in which accuracy can be enhanced to acceptable

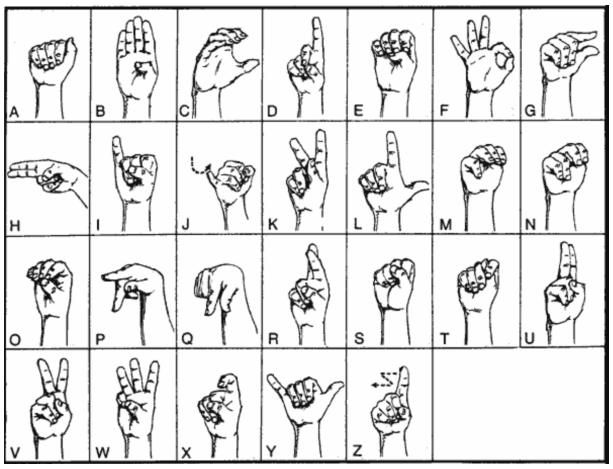

Fig. 3: ASL hand-shape chart [7]

levels. Primarily, the user should ensure the lighting in the room is adequate. At which point the user should use the LMCs built-in calibration tool. The most intuitive way to improve accuracy would be to increase the trained set size, providing more angles that can be referenced during classification. Additionally, the user can modify the `compare_bottom_percent` and `compare_top_percent` limits located in $AppSettings.txt$, widening or narrowing the allowable discrepancy between the angle generated during interpretation and the angles generated during training.

V. Experiment Design

A. User Study

LEAP-Trainer was evaluated in a total of 23,400 gestures collected from three participants over the 26 letters in the ASL alphabet (7,800 exclusive recordings per participant). Each letter was requested to be trained a total of 300 times. Additionally, the participants were requested to train only their dominant hand. The ages of the participants are 21, 25, and 30 respectively.

- A male majority of $\frac{2}{3}$.
- $\frac{2}{3}$ are right hand dominant.
- All three participants are working toward a degree in Computer Science.

B. Results

The experiment was performed live using the LEAP-Trainer paired with the LMC on a Windows 10 i7 Surface Book. Each participant was requested to test the interpretation of each letter ten times and record if what was displayed by the interpreter was the gesture they intended. Even with the classification difficulties spoken in the accuracy section, when using a `compare_bottom_percent` of 80 and a `compare_top_percent` of 120, results obtained showed an average recognition rate of approximately 78% except for the letters discussed in section IV-B.

VI. Conclusion

This paper has described the LEAP-Trainer, a single-frame gesture recording and recognition software that utilizes the Leap Motion Controller optical, infrared sensor, designed by Leap Motion to perform detection and tracking of the human hand and its features. This paper has demonstrated that the LEAP-Trainer paired with the Leap Motion Controller, the right environment and application settings can accurately classify single-frame gestures in American Sign Language in real-time with an average accuracy of 78% per letter. Additionally, no detail has been spared in explaining the acquisition of data, the explanation of its uses, as well as the algorithms incorporated. Future work includes the possibility of extending the software to utilize more than one controller and to introduce a system that can support multi-framed gestures.

Acknowledgments

With support from Dr. Francisco Ortega and the AR-VR-VE for Computer Science Education team, the School of Computing and Information Sciences, Florida International University, and the user study testers: Ms. Sheila Alemany, Mr. Alberto Camacho, and Mr. Armando Carrasquillo.

References

[1] Leap Motion, http://www.leapmotion.com (last access: Dec 2017)

[2] American sign language National Association of the Deaf http://nad.org/issues/american-sign-language (last access: Dec 2017)

[3] Fok, Kai-Yin, et al. "A real-time asl recognition system using leap motion sensors." Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on. IEEE, 2015.

[4] Chuan, Ching-Hua, Eric Regina, and Caroline Guardino. "American sign language recognition using leap motion sensor." Machine Learning and Applications (ICMLA), 2014 13th International Conference on. IEEE, 2014.

[5] Cover, Thomas, and Peter Hart. "Nearest neighbor pattern classification." IEEE transactions on information theory 13.1 (1967): 21-27.

[6] Leap Motion CSharp API, https://developer.leapmotion.com/documentation/v2/csharp/devguide/`Leap_Overview`.html (last access: Dec 2017)

[7] ASL hand-shape chart, http://resources.seattlecentral.edu/faculty/bbernstein/ASL101/`handshape_chart`.gif (last access: Dec 2017)