

**DIP-CSE3081**

**COURSE-PROJECT-REPORT**

PROJECT TITLE

**Gesture Recognition System**

Group Members

1. Vipul Joseph Pinto (20201CSE0596)

2. Hemanth Kumar Shetty M(20201CSE0655)

3. Deepak V (20201CSE0673)

Supervisor

Mr. Lakshmisha S Krishna

May 2024

Table of Content

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| 1. | Abstract | 3 |
| 2. | Introduction | 3 |
| 3. | Objectives | 4 |
| 4. | Literature review | 4 |
| 5. | Methodology | 8 |
| 6. | Design | 9 |
| 7. | Block Diagram | 9 |
| 8. | System Flow Chart | 10 |
| 9. | Software Code | 11 |
| 10. | Evaluation and Results | 19 |
| 11. | Conclusion | 20 |
| 12. | References | 20 |
|  | Base Paper | 20 |
|  | Appendix | 20 |

Abstract

In the realm of assistive technology, GestureLink stands as a beacon of innovation and inclusivity. Developed using Python programming language and harnessing the power of cutting-edge libraries like MediaPipe and TensorFlow, GestureLink empowers individuals with speech impairments to communicate effortlessly through hand gestures. By identifying hand coordinates and recognizing patterns in real-time, GestureLink opens doors to a world where every gesture becomes a message, every movement a means of connection. In a world where verbal communication serves as the cornerstone of human interaction, individuals with speech impairments often face daunting challenges in expressing themselves and connecting with others. GestureSense emerges as a beacon of hope, offering a transformative solution that transcends linguistic barriers and empowers users to communicate with clarity and confidence. Developed with a profound sense of empathy and driven by cutting-edge technology, GestureSense harnesses the power of gesture recognition to create a platform where every movement becomes a message, every gesture a conduit for connection and understanding. Through the integration of Python, MediaPipe, and TensorFlow, GestureSense embodies a vision of inclusivity and empowerment, paving the way for a more inclusive and compassionate society.

Introduction

Communication lies at the heart of human experience, shaping our interactions, relationships, and sense of belonging. For individuals with speech impairments, however, the ability to communicate effectively can be elusive, leading to feelings of frustration, isolation, and disconnection. GestureSense seeks to address this fundamental need by providing a revolutionary platform that enables users to communicate through hand gestures. By leveraging the latest advancements in gesture recognition technology, GestureSense opens doors to a world where every gesture is understood, every movement meaningful. This introduction sets the stage for a comprehensive exploration of GestureSense, from its inception to its potential to transform the lives of its users. yet for individuals with speech impairments, expressing themselves can be a daunting task. GestureSense seeks to address this challenge by providing a novel solution that transcends traditional forms of communication. By leveraging cutting-edge technology and innovative design principles, GestureSense aims to redefine the way individuals with speech impairments connect with the world around them. This introduction sets the stage for a comprehensive exploration of GestureSense, from its inception to its potential impact on the lives of its users. GestureSense harnesses the power of gesture recognition to create a platform where every movement becomes a message, every gesture a conduit for connection and understanding. Through the integration of Python, MediaPipe, and TensorFlow, GestureSense embodies a vision of inclusivity and empowerment, paving the way for a more inclusive and compassionate society.

Objectives

# 1.Develop a robust hand gesture recognition system capable of accurately interpreting a diverse range of gestures in real-time, ensuring inclusivity and accessibility for users across varying contexts and environments.

# 2.Design an intuitive and user-friendly interface that invites users to engage with GestureSense with ease and confidence, fostering a sense of empowerment and agency.

# 3.Conduct thorough testing and evaluation to ensure that GestureSense meets the diverse needs and preferences of its users, fostering a culture of continuous improvement and innovation.

# 4.Explore opportunities for customization and personalization within GestureSense, allowing users to tailor the system to their unique communication styles and preferences.

# 5.Lay the groundwork for future enhancements and iterations of GestureSense, with a focus on expanding its functionality, accessibility, and impact within the broader assistive technology community, while remaining responsive to evolving user needs and technological advancements.

Literature review –

Title - HAND GESTURE RECOGNITION

# Abstract

Hand gesture recognition system received great attention in the recent few years because of its manifoldness applications and the ability to interact with machine efficiently through human computer interaction. In this paper a survey of recent hand gesture recognition systems is presented. Key issues of hand gesture recognition system are presented with challenges of gesture system. Review methods of recent postures and gestures recognition system presented as well. Summary of research results of hand gesture methods, databases, and comparison between main gesture recognition phases are also given. Advantages and drawbacks of the discussed systems are explained finally.

Introduction

The essential aim of building hand gesture recognition system is to create a natural interaction between human and computer where the recognized gestures can be used for controlling a robot or conveying meaningful information [1]. How to form the resulted hand gestures to be understood and well interpreted by the computer considered as the problem of gesture interaction [2].

Human computer interaction (HCI) also named Man-Machine Interaction (MMI) [3][4] refers to the relation between the human and the computer or more precisely the machine, and since the machine is insignificant without suitable utilize by the human [3]. There are two main characteristics should be deemed when designing a HCI system as mentioned in [3]: functionality and usability. System functionality referred to the set of functions or services that the system equips to the users [3], while system usability referred to the level and scope that the system can operate and perform specific user purposes efficiently [3]. The system that attains a suitable balance between these concepts considered as influential performance and powerful system [3]. Gestures used for communicating between human and machines as well as between people using sign

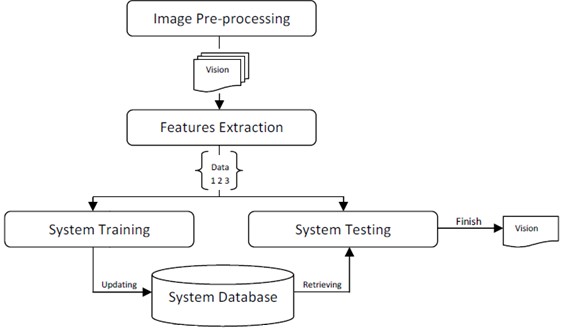
## Features Extraction

Good segmentation process leads to perfect features extraction process and the latter play an important role in a successful recognition process [6]. Features vector of the segmented image can be extracted in different ways according to particular application. Various methods have been applied for representing the features can be extracted. Some methods used the shape of the hand such as hand contour and silhouette [6] while others utilized fingertips position, palm center, etc. [6] created 13 parameters as a feature vector, the first parameters represents the ratio aspect of the bounding box of the hand and the rest 12 parameters are mean values of brightness pixels in the image. [14] used Self-Growing and Self-Organized Neural Gas (SGONG) neural algorithm to capture the shape of the hand, then three features are obtained; Palm region, Palm center, and Hand slope. [16] calculated the Center Of Gravity (COG) of the segmented hand and the distance from the COG to the farthest point in the fingers, and extracted one binary signal (1D) to estimate the number of fingers in the hand region. [15] divided the segmented image into different blocks size and each block represents the brightness measurements in the image. Many experiments were applied to decide the right block size that can achieve good recognition rate [15]. [17][18] used Gaussian pdf to extract geometric central moment as local and global features. Figure 3 shows some applications of feature extraction methods.

## Gestures Classification

After modeling and analysis of the input hand image, gesture classification method is used to recognize the gesture. Recognition process affected with the proper selection of features parameters and suitable classification algorithm [7]. For example edge detection or contour operators [9] cannot be used for gesture recognition since many hand postures are generated and could produce misclassification [9]. Euclidean distance metric used to classify the gestures [19][5][17]. Statistical tools used for gesture classification, HMM tool has shown its ability to recognize dynamic gestures [20][13]besides, Finite State Machine (FSM) [21], Learning Vector Quantization [22], and Principal Component Analysis (PCA) [23]. Neural network has been widely applied in the field of extracted the hand shape [14], and for hand gesture recognition

[24][25][26]. Other soft computing tools are effective in this field as well, such as Fuzzy CMeans clustering (FCM) [6], and Genetic Algorithms GAs [27]. Figure 4 explain the architecture of classification system.



Architecture of gesture recognition system

# APPLICATION AREAS OF HAND GESTURES SYSTEM

Hand gestures recognition system has been applied for different applications on different domains, as mentioned in [7][9] including; sign language translation, virtual environments, smart surveillance, robot control, medical systems etc. overview of some hand gesture application areas are listed below[7][8].

A. Sign Language Recognition:

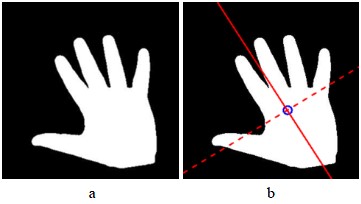
B. Robot Control:

C. Graphic Editor Control:

D. Virtual Environments ( VEs):

# LITERATURE REVIEW OF GESTURE RECOGNITION SYSTEMS

Hasan [17] applied multivariate Gaussian distribution to recognize hand gestures using nongeometric features. The input hand image is segmented using two different methods [18]; skin color based segmentation by applying HSV color model and clustering based thresholding techniques [18]. Some operations are performed to capture the shape of the hand to extract hand feature; the modified Direction Analysis Algorithm are adopted to find a relationship between statistical parameters (variance and covariance) [17] from the data, and used to compute object (hand) slope and trend [17] by finding the direction of the hand gesture [17], As shown in Figure 5.



# DRAWBACKS

In this section, drawbacks of some discussed methods are explained: Orientation histogram method applied in [19] have some problems which are; similar gestures might have different orientation histograms and different gestures could have similar orientation histograms, besides that, the proposed method achieved well for any objects that dominate the image even if it is not the hand gesture [19]. Neural Network classifier has been applied for gestures classification [28][8] but it is time consuming and when the number of training data increase, the time needed for classification are increased too [8]. In [28] the NN required several hours for learning 42 characters and four days to learn ten words [28]. Fuzzy c-means clustering algorithm applied in [6] has some disadvantages; wrong object extraction problem raised if the objects larger than the hand. The performance of recognition algorithm decreases when the distance greater than 1.5 meters between the user and the camera. Besides that, its variation to lighting condition changes and unwanted objects might overlap with the hand gesture. In [16] the system is variation to environment lighting changes which produces erroneous segmentation of the hand region. HMM tools are perfect for recognition dynamic gestures [13] but it is computational consuming.

# CONCLUSIONS

In this paper various methods are discussed for gesture recognition, these methods include from Neural Network, HMM, fuzzy c-means clustering, besides using orientation histogram for features representation. For dynamic gestures HMM tools are perfect and have shown its efficiency especially for robot control [20][16]. NNs are used as classifier [8][25] and for capturing hand shape in [14]. For features extraction, some methods and algorithms are required even to capture the shape of the hand as in [15][17][18], [17] applied Gaussian bivariate function for fitting the segmented hand which used to minimize the rotation affection [17][18]. The selection of specific algorithm for recognition depends on the application needed. In this work application areas for the gestures system are presented. Explanation of gesture recognition issues, detail discussion of recent recognition systems are given as well. Summary of some selected systems are listed as well.

Methodology

The methodology section outlines the systematic approach taken in the design, development, and testing of GestureSense, encompassing a series of interconnected stages and activities aimed at achieving project objectives. Beginning with a deep dive into user requirements and a review of existing research, the methodology progresses through iterative cycles of design, prototyping, and testing. Python serves as the canvas upon which GestureSense is painted, while MediaPipe and TensorFlow provide the colors that bring it to life. Through a relentless pursuit of excellence and a commitment to user feedback, GestureSense emerges as a testament to the power of human-centered design in shaping technology for the better.

Requirements:

* Hand gesture as input data for the training data.
* A formatted csv file to store the student’s name and entry time.

Hardware:

* Web Cam

Software:

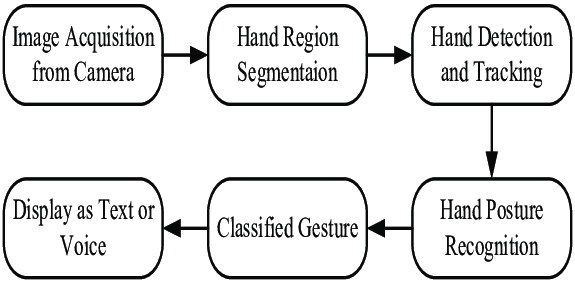
* Mediapipe
* TensorFlow
* Python

Design

GestureSense's design embodies a synthesis of technological innovation, user-centered design principles, and human-computer interaction considerations, with a focus on maximizing functionality, usability, and accessibility. The system architecture is meticulously crafted to facilitate seamless integration of hardware and software components, with each element playing a vital role in the gesture recognition process. MediaPipe's hand tracking module captures the essence of hand movements, while TensorFlow's machine learning models bring them to life with unparalleled accuracy and precision. The user interface serves as a window into GestureSense's soul, inviting users to explore a world where every gesture is a message, and every message is a connection.

At the heart of GestureSense lies a design philosophy rooted in simplicity, accessibility, and elegance. The system architecture is carefully crafted to ensure seamless integration of hardware and software components, with each element playing a vital role in the gesture recognition process. MediaPipe's hand tracking module captures the essence of hand movements, while TensorFlow's machine learning models bring them to life with unparalleled accuracy and precision. The user interface serves as a window into GestureSense's soul, inviting users to explore a world where every gesture is a message, and every message is a connection.

Block diagram



Explanation:

**Preprocessing and Segmentation:**

Enhance image quality and segment the hand from the background for clearer analysis.

**Hand Detection and Tracking:**

Identify and follow the hand's position and movement in real-time.

**Hand Posture Recognition:**

Recognize specific static hand postures by analyzing the hand's shape and orientation.

**Hand Gesture Classification:**

Classify recognized postures into meaningful gestures using machine learning models.

**Message Generation:**

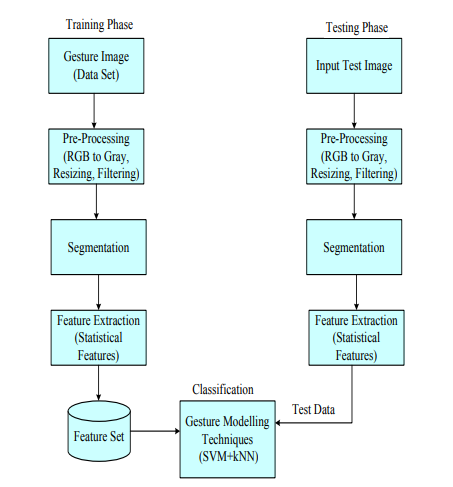
Map classified gestures to predefined messages for interpretation.

**Output Display:**

Display or play the interpreted messages through a screen or speaker, allowing effective communication.

This summary and block diagram outline the key stages involved in a hand gesture recognition system, demonstrating the flow from initial image processing to the final message output.

System Flow chart



Explanation:

* **Training Phase:**

1. **Gesture Image (Data Set):** Collection of gesture images for model training.
2. **Pre-Processing:** Converts images to grayscale, resizes, and filters them.
3. **Segmentation:** Isolates the hand gesture from the background.
4. **Feature Extraction:** Extracts statistical features from segmented images.
5. **Feature Set:** Compiles extracted features into a comprehensive set.
6. **Gesture Modelling Techniques (SVM + kNN):** Trains the model using SVM and kNN algorithms.

* **Testing Phase:**

1. **Input Test Image:** New gesture image for classification.
2. **Pre-Processing:** Standardizes the test image format.
3. **Segmentation:** Isolates the gesture in the test image.
4. **Feature Extraction:** Extracts features from the segmented test image.
5. **Classification:** Uses the trained model to classify the test image, resulting in the recognized gesture.

This architecture ensures that GestureSense effectively processes and classifies hand gestures, enabling reliable communication for users with speech impairments.

Software Code

**MainActivity.kt**

package com.google.mediapipe.examples.gesturerecognizer

import android.os.Bundle

import androidx.activity.viewModels

import androidx.appcompat.app.AppCompatActivity

import androidx.navigation.fragment.NavHostFragment

import androidx.navigation.ui.setupWithNavController

import com.google.mediapipe.examples.gesturerecognizer.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

private lateinit var activityMainBinding: ActivityMainBinding

private val viewModel: MainViewModel by viewModels()

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

activityMainBinding = ActivityMainBinding.inflate(layoutInflater)

setContentView(activityMainBinding.root)

val navHostFragment =

supportFragmentManager.findFragmentById(R.id.fragment\_container) as NavHostFragment

val navController = navHostFragment.navController

activityMainBinding.navigation.setupWithNavController(navController)

activityMainBinding.navigation.setOnNavigationItemReselectedListener {

// ignore the reselection

}

}

override fun onBackPressed() {

finish()

    }

}

**GestureRecognizerHelper.kt**

package com.google.mediapipe.examples.gesturerecognizer

import android.content.Context

import android.graphics.Bitmap

import android.graphics.Matrix

import android.media.MediaMetadataRetriever

import android.net.Uri

import android.os.SystemClock

import android.util.Log

import androidx.annotation.VisibleForTesting

import androidx.camera.core.ImageProxy

import com.google.mediapipe.framework.image.BitmapImageBuilder

import com.google.mediapipe.framework.image.MPImage

import com.google.mediapipe.tasks.core.BaseOptions

import com.google.mediapipe.tasks.core.Delegate

import com.google.mediapipe.tasks.vision.core.RunningMode

import com.google.mediapipe.tasks.vision.gesturerecognizer.GestureRecognizer

import com.google.mediapipe.tasks.vision.gesturerecognizer.GestureRecognizerResult

class GestureRecognizerHelper(

var minHandDetectionConfidence: Float = DEFAULT\_HAND\_DETECTION\_CONFIDENCE,

var minHandTrackingConfidence: Float = DEFAULT\_HAND\_TRACKING\_CONFIDENCE,

var minHandPresenceConfidence: Float = DEFAULT\_HAND\_PRESENCE\_CONFIDENCE,

var currentDelegate: Int = DELEGATE\_CPU,

var runningMode: RunningMode = RunningMode.IMAGE,

val context: Context,

val gestureRecognizerListener: GestureRecognizerListener? = null

) {

// For this example this needs to be a var so it can be reset on changes. If the GestureRecognizer

// will not change, a lazy val would be preferable.

private var gestureRecognizer: GestureRecognizer? = null

init {

setupGestureRecognizer()

}

fun clearGestureRecognizer() {

gestureRecognizer?.close()

gestureRecognizer = null

}

// Initialize the gesture recognizer using current settings on the

// thread that is using it. CPU can be used with recognizers

// that are created on the main thread and used on a background thread, but

// the GPU delegate needs to be used on the thread that initialized the recognizer

fun setupGestureRecognizer() {

// Set general recognition options, including number of used threads

val baseOptionBuilder = BaseOptions.builder()

// Use the specified hardware for running the model. Default to CPU

when (currentDelegate) {

DELEGATE\_CPU -> {

baseOptionBuilder.setDelegate(Delegate.CPU)

}

DELEGATE\_GPU -> {

baseOptionBuilder.setDelegate(Delegate.GPU)

}

}

baseOptionBuilder.setModelAssetPath(MP\_RECOGNIZER\_TASK)

try {

val baseOptions = baseOptionBuilder.build()

val optionsBuilder =

GestureRecognizer.GestureRecognizerOptions.builder()

.setBaseOptions(baseOptions)

.setMinHandDetectionConfidence(minHandDetectionConfidence)

.setMinTrackingConfidence(minHandTrackingConfidence)

.setMinHandPresenceConfidence(minHandPresenceConfidence)

.setRunningMode(runningMode)

if (runningMode == RunningMode.LIVE\_STREAM) {

optionsBuilder

.setResultListener(this::returnLivestreamResult)

.setErrorListener(this::returnLivestreamError)

}

val options = optionsBuilder.build()

gestureRecognizer =

GestureRecognizer.createFromOptions(context, options)

} catch (e: IllegalStateException) {

gestureRecognizerListener?.onError(

"Gesture recognizer failed to initialize. See error logs for " + "details"

)

Log.e(

TAG,

"MP Task Vision failed to load the task with error: " + e.message

)

} catch (e: RuntimeException) {

gestureRecognizerListener?.onError(

"Gesture recognizer failed to initialize. See error logs for " + "details",

GPU\_ERROR

)

Log.e(

TAG,

"MP Task Vision failed to load the task with error: " + e.message

)

}

}

// Convert the ImageProxy to MP Image and feed it to GestureRecognizer.

fun recognizeLiveStream(

imageProxy: ImageProxy,

) {

val frameTime = SystemClock.uptimeMillis()

// Copy out RGB bits from the frame to a bitmap buffer

val bitmapBuffer = Bitmap.createBitmap(

imageProxy.width, imageProxy.height, Bitmap.Config.ARGB\_8888

)

imageProxy.use { bitmapBuffer.copyPixelsFromBuffer(imageProxy.planes[0].buffer) }

imageProxy.close()

val matrix = Matrix().apply {

// Rotate the frame received from the camera to be in the same direction as it'll be shown

postRotate(imageProxy.imageInfo.rotationDegrees.toFloat())

// flip image since we only support front camera

postScale(

-1f, 1f, imageProxy.width.toFloat(), imageProxy.height.toFloat()

)

}

// Rotate bitmap to match what our model expects

val rotatedBitmap = Bitmap.createBitmap(

bitmapBuffer,

0,

0,

bitmapBuffer.width,

bitmapBuffer.height,

matrix,

true

)

// Convert the input Bitmap object to an MPImage object to run inference

val mpImage = BitmapImageBuilder(rotatedBitmap).build()

recognizeAsync(mpImage, frameTime)

}

// Run hand gesture recognition using MediaPipe Gesture Recognition API

@VisibleForTesting

fun recognizeAsync(mpImage: MPImage, frameTime: Long) {

// As we're using running mode LIVE\_STREAM, the recognition result will

// be returned in returnLivestreamResult function

gestureRecognizer?.recognizeAsync(mpImage, frameTime)

}

// Accepts the URI for a video file loaded from the user's gallery and attempts to run

// gesture recognizer inference on the video. This process will evaluate

// every frame in the video and attach the results to a bundle that will be

// returned.

fun recognizeVideoFile(

videoUri: Uri,

inferenceIntervalMs: Long

): ResultBundle? {

if (runningMode != RunningMode.VIDEO) {

throw IllegalArgumentException(

"Attempting to call recognizeVideoFile" +

" while not using RunningMode.VIDEO"

)

}

// Inference time is the difference between the system time at the start and finish of the

// process

val startTime = SystemClock.uptimeMillis()

var didErrorOccurred = false

// Load frames from the video and run the gesture recognizer.

val retriever = MediaMetadataRetriever()

retriever.setDataSource(context, videoUri)

val videoLengthMs =

retriever.extractMetadata(MediaMetadataRetriever.METADATA\_KEY\_DURATION)

?.toLong()

// Note: We need to read width/height from frame instead of getting the width/height

// of the video directly because MediaRetriever returns frames that are smaller than the

// actual dimension of the video file.

val firstFrame = retriever.getFrameAtTime(0)

val width = firstFrame?.width

val height = firstFrame?.height

// If the video is invalid, returns a null recognition result

if ((videoLengthMs == null) || (width == null) || (height == null)) return null

// Next, we'll get one frame every frameInterval ms, then run recognizer

// on these frames.

val resultList = mutableListOf<GestureRecognizerResult>()

val numberOfFrameToRead = videoLengthMs.div(inferenceIntervalMs)

for (i in 0..numberOfFrameToRead) {

val timestampMs = i \* inferenceIntervalMs // ms

retriever

.getFrameAtTime(

timestampMs \* 1000, // convert from ms to micro-s

MediaMetadataRetriever.OPTION\_CLOSEST

)

?.let { frame ->

// Convert the video frame to ARGB\_8888 which is required by the MediaPipe

val argb8888Frame =

if (frame.config == Bitmap.Config.ARGB\_8888) frame

else frame.copy(Bitmap.Config.ARGB\_8888, false)

// Convert the input Bitmap object to an MPImage object to run inference

val mpImage = BitmapImageBuilder(argb8888Frame).build()

// Run gesture recognizer using MediaPipe Gesture Recognizer

// API

gestureRecognizer?.recognizeForVideo(mpImage, timestampMs)

?.let { recognizerResult ->

resultList.add(recognizerResult)

} ?: {

didErrorOccurred = true

gestureRecognizerListener?.onError(

"ResultBundle could not be returned" +

" in recognizeVideoFile"

)

}

}

?: run {

didErrorOccurred = true

gestureRecognizerListener?.onError(

"Frame at specified time could not be" +

" retrieved when recognition in video."

)

}

}

retriever.release()

val inferenceTimePerFrameMs =

(SystemClock.uptimeMillis() - startTime).div(numberOfFrameToRead)

return if (didErrorOccurred) {

null

} else {

ResultBundle(resultList, inferenceTimePerFrameMs, height, width)

}

}

// Accepted a Bitmap and runs gesture recognizer inference on it to

// return results back to the caller

fun recognizeImage(image: Bitmap): ResultBundle? {

if (runningMode != RunningMode.IMAGE) {

throw IllegalArgumentException(

"Attempting to call detectImage" +

" while not using RunningMode.IMAGE"

)

}

// Inference time is the difference between the system time at the

// start and finish of the process

val startTime = SystemClock.uptimeMillis()

// Convert the input Bitmap object to an MPImage object to run inference

val mpImage = BitmapImageBuilder(image).build()

// Run gesture recognizer using MediaPipe Gesture Recognizer API

gestureRecognizer?.recognize(mpImage)?.also { recognizerResult ->

val inferenceTimeMs = SystemClock.uptimeMillis() - startTime

return ResultBundle(

listOf(recognizerResult),

inferenceTimeMs,

image.height,

image.width

)

}

// If gestureRecognizer?.recognize() returns null, this is likely an error. Returning null

// to indicate this.

gestureRecognizerListener?.onError(

"Gesture Recognizer failed to recognize."

)

return null

}

// Return running status of the recognizer helper

fun isClosed(): Boolean {

return gestureRecognizer == null

}

// Return the recognition result to the GestureRecognizerHelper's caller

private fun returnLivestreamResult(

result: GestureRecognizerResult, input: MPImage

) {

val finishTimeMs = SystemClock.uptimeMillis()

val inferenceTime = finishTimeMs - result.timestampMs()

gestureRecognizerListener?.onResults(

ResultBundle(

listOf(result), inferenceTime, input.height, input.width

)

)

}

// Return errors thrown during recognition to this GestureRecognizerHelper's

// caller

private fun returnLivestreamError(error: RuntimeException) {

gestureRecognizerListener?.onError(

error.message ?: "An unknown error has occurred"

)

}

companion object {

val TAG = "GestureRecognizerHelper ${this.hashCode()}"

private const val MP\_RECOGNIZER\_TASK = "gesture\_recognizer.task"

const val DELEGATE\_CPU = 0

const val DELEGATE\_GPU = 1

const val DEFAULT\_HAND\_DETECTION\_CONFIDENCE = 0.5F

const val DEFAULT\_HAND\_TRACKING\_CONFIDENCE = 0.5F

const val DEFAULT\_HAND\_PRESENCE\_CONFIDENCE = 0.5F

const val OTHER\_ERROR = 0

const val GPU\_ERROR = 1

}

data class ResultBundle(

val results: List<GestureRecognizerResult>,

val inferenceTime: Long,

val inputImageHeight: Int,

val inputImageWidth: Int,

)

interface GestureRecognizerListener {

fun onError(error: String, errorCode: Int = OTHER\_ERROR)

fun onResults(resultBundle: ResultBundle)

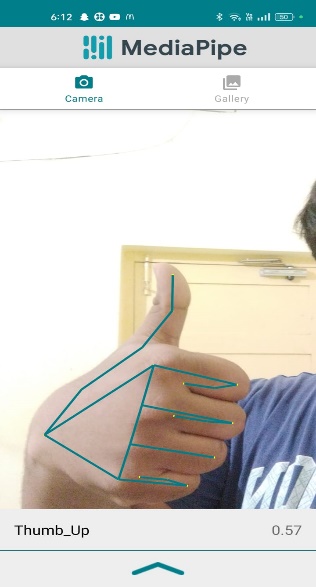
    }

}

Evaluation and Results

Following are the examples which the user can use with the help of Mediapy –

Example Outputs -

Conclusion

As GestureSense embarks on the final leg of its journey, it stands tall as a beacon of human compassion and technological prowess in the landscape of assistive technology. Each line of code, each meticulously crafted algorithm, is a testament to the unwavering dedication and empathy of the team behind it. With every gesture it recognizes, GestureSense extends a lifeline to those whose voices are often lost in the cacophony of the world.

In the symphony of human connection, GestureSense orchestrates a melody of understanding and belonging for individuals with speech impairments. It transforms mere movements into profound messages, bridging the gap between silence and expression with grace and precision. As we bid adieu to this chapter of our journey, we do so with hearts brimming with gratitude and eyes shimmering with the hope of a brighter future.

For in the tapestry of human experience, GestureSense is more than just a technological marvel—it is a testament to the boundless capacity of the human spirit to innovate, to empathize, and to uplift. And as it continues to illuminate the path forward, we stand in awe of its legacy, knowing that it will forever shape the landscape of assistive technology and redefine the way we connect, communicate, and understand each other.

References

<https://www.researchgate.net/publication/284626785_Hand_Gesture_Recognition_A_Literature_Review>

<https://www.researchgate.net/publication/356400301_A_Review_of_the_Hand_Gesture_Recognition_System_Current_Progress_and_Future_Directions>

Base Paper

<https://www.sciencedirect.com/topics/computer-science/gesture-recognition>

<https://www.st.com/content/st_com/en/campaigns/gesture-recognition.html>

Appendix

Nestled within the appendix lies a treasure trove of insights, specifications, and snapshots that capture the essence of GestureSense's journey. From detailed component datasheets to immersive project images, each appendix item offers a glimpse into the heart and soul of GestureSense, showcasing the passion, dedication, and innovation that have gone into its creation and evolution. From detailed component datasheets to immersive project images, each appendix item offers a glimpse into the heart and soul of GestureSense, showcasing the passion, dedication, and innovation that have gone into its creation and evolution.