BHT Berliner Hochschule für Technik

Morse-Code Erkennung und Interpretation

Fachbereich VI

Studiengang Technische Informatik

Prüfer:

Prof. Dr.-Ing. Kristian Hildebrand

vorgelegt von:

Khaled Rafei – 930677

Ziad Elmhalawy - 108906

Abgabetermin:

27.09.2025

SS25

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Dieses Projekt beschäftigt sich mit der digitalen Umsetzung und Interpretation des Morsecodes. Morsecode ist ein Kommunikationssystem, bei dem Buchstaben und Zeichen durch kurze und lange Signale – traditionell dargestellt als Punkte (.) und Striche (–) – codiert werden. In dieser Umsetzung wurde eine vereinfachte binäre Darstellung verwendet: Der Punkt wird durch eine „0“ ersetzt, der Strich durch eine „1“. So ergibt sich beispielsweise für den Buchstaben „A“ die Sequenz „01“, für „B“ „1000“, für „C“ „1010“ usw.

Das Ziel des Projekts ist es, Videodaten auszuwerten, in denen eine Hand entweder offen oder geschlossen gezeigt wird. Dabei steht eine offene Hand für den Wert „1“ und eine geschlossene Hand für den Wert „0“. Das entwickelte Programm analysiert die Videosequenzen, erkennt den Zustand der Hand in jedem Frame und generiert daraus eine binäre Zeichenfolge. Im Anschluss wird diese Bitfolge dekodiert und in lesbaren Text – also Buchstaben und Wörter – umgewandelt.

1.1 Motivation

Solche Ansätze sind insbesondere für Assistive Technologien, lautlose Kommunikation in speziellen Situationen (z. B. Einsatzkräfte, stille Signale) oder Notfallszenarien interessant. Die Handgesten-basierte Übertragung von Morsecode verbindet eine sehr alte, robuste Codierungsmethode mit modernen Computer-Vision-Technologien.

1.2 Zielsetzung

Das Projekt verfolgt die konkrete Zielsetzung, eine möglichst robuste, automatisierte Pipeline zur Echtzeit-Dekodierung von Handgesten in Morsezeichen zu entwickeln. Wichtige Aspekte sind dabei Robustheit gegenüber unterschiedlichen Beleuchtungen, einfache Hardwareanforderungen (Standardkamera), und eine flexible Auswertung für verschiedene Testvideos.

2. Stand der Technik

Für die Erkennung der Handpose werden moderne Frameworks wie **MediaPipe Hands** eingesetzt. In diesem Projekt kommt MediaPipe zum Einsatz, da es Landmark-Punkte der Hand zuverlässig und in Echtzeit liefert. Der Handzustand (offen/geschlossen) wird anschließend über eine einfache heuristische Regel bestimmt, die Fingerkuppen und Gelenke vergleicht. Dadurch lässt sich mit geringem Rechenaufwand eine Binärfolge erzeugen, die Grundlage für die Morse-Interpretation ist.

Im Unterschied zu komplexeren ML-basierten Gestenerkennungssystemen wird hier ein regelbasierter Ansatz gewählt: MediaPipe liefert die Landmarks, die Heuristik bestimmt den Zustand, ein Postprocessing glättet die Sequenz, und ein Decoder übersetzt die Bitfolge in Klartext. Dieser modulare Ansatz ist transparent und ressourcenschonend, eignet sich jedoch vor allem für Prototypen und weniger für robuste Anwendungen unter variablen Bedingungen.

3. Daten & Versuchsanordnung

Hier steht eine Liste der Beispiel-Videos:

Dateiname	Dauer	Fps	Auflösung	Erwartete Ausgabe
Help.mp4	40s	29.92	720*1280	**[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0]** **'HELP' oder 'help'**
Hello01.mp4	44s	30.00	1920*1080	**[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]** **'HELLO' oder 'hello'**
Hello02.mp4	61s	29.85	720*1280	**[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]** **'HELLO' oder 'hello'**
Hallo.mp4	51s	30.00	1920*1080	**[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]** **'HALLO' oder 'hallo'**
Hilfe.mp4	45s	30.00	1920*1080	**[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]** **'HILFE' oder 'hilfe'**
Love.mp4	39s	30.00	1920*1080	**[0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0]** **'LOVE' oder 'love'**
Move.mp4	30s	30.00	1920*1080	**[1, 1, 1, 1, 1, 0, 0, 0, 1, 0]** **'MOVE' oder 'move'**
Arshia.mp4	50s	30.00	1920*1080	**[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]** **'ARSHIA' oder 'arshia'**
Elham.mp4	40s	30.00	1920*1080	**[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]** **'ELHAM' oder 'elham'**
Sadra.mp4	39s	29.95	720*1280	**[0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1]** **'SADRA' oder 'sadra'**
Kasra.mp4	42s	29.93	720*1280	**[1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1]** **'KASRA' oder 'kasra'**

Aufnahmebedingungen:

- Kameraposition: Frontkamera von 3 verschiedene Handys, Abstand ca. 0,5–1 m.
- Blickwinkel: Frontalaufnahme, die Hand ist vollständig sichtbar.
- Beleuchtung: Helle und halbhelle, gleichmäßige Beleuchtung ohne starke Schatten.
- Szenario: Alle Videos wurden mit einer Hand aufgenommen, um die binäre Unterscheidung (offen/geschlossen) konsistent umzusetzen.

4. Systemarchitektur & Pipeline

Das System folgt einer modularen Pipeline, die von der Videoaufnahme bis zur Textausgabe reicht:

Video → Frame-Extraktion → Handpose-Detection (MediaPipe) → State-Estimation (offen/ geschlossen) → Postprocessing & Glättung → Bitfolge-Erzeugung → Morse-Decoder → Ausgabe.

```
for value in frames:  # frames = deine Frame-Analyse
    if current_state is None:
        current_state = value
        frame_count = 1
        gap_count = 0
    else:
        if value == current_state:
            frame_count += 1
            gap_count = 0  # Unterbrechung beendet
        elif value == -1:
            # Kurze Unterbrechung innerhalb eines Zustands ignorieren
            gap_count += 1
            if gap_count > GAP_TOLERANCE:
                # Zustand beenden
                if (frame_count >= STABLE_MIN_FRAMES and current_state != -1):
                    final_sequence.append(current_state)

                frame_count = 0
                gap_count = 0
                current_state = None
        else:
            # Wechsel zu anderem Wert
            if frame_count >= STABLE_MIN_FRAMES and current_state != -1:
                final_sequence.append(current_state)

                frame_count = 1
                gap_count = 0

            current_state = value
```

Abbildung 1: Codearchitektur

4.1 Implementierung im Projekt

- Die Videoaufnahme erfolgt mit OpenCV (cv2.VideoCapture).
- MediaPipe Hands extrahiert Landmark-Punkte der Hand (einzelne Fingerkuppen und Gelenke).
- Über die Funktion **is_hand_open** wird heuristisch entschieden, ob die Hand offen oder geschlossen ist.

- Jede Frame-Analyse liefert einen Zustand (1, 0 oder -1 bei fehlender Hand). Ein Postprocessing-Algorithmus stabilisiert die Sequenz: Zustände müssen mindestens STABLE_MIN_FRAMES anhalten, kurze Aussetzer werden durch GAP_TOLERANCE toleriert.
- Die resultierende stabile Bitfolge wird mit einer Morsecodetabelle abgeglichen. Mithilfe des Lexikon-basierten Decoders (decode_with_lexicon_or_estimate) wird die Bitfolge in Wörter übersetzt.

4.2 Wichtige Parameter

frame_rate: Bildrate des Videos, automatisch mit OpenCV ermittelt.

STABLE_MIN_FRAMES: Minimale Anzahl aufeinanderfolgender Frames, um einen Zustand als stabil zu bestätigen (Standard: 31 Frames ≈ 1 Sekunde bei 30 fps).

GAP_TOLERANCE: Anzahl Frames, die als kurze Unterbrechung toleriert werden (Standard: 20 Frames).

5. Implementierung

5.1 Code-Struktur & Module

Im Projekt ist die Logik aktuell in einem Skript (Bits_Test.py) gebündelt, lässt sich jedoch klar in Module aufteilen:

- video_io.py – Einlesen des Videos über OpenCV, Ermittlung der frame_rate.
- state_estimator.py – Initialisierung von MediaPipe Hands, Extraktion von Landmark-Punkten, Funktion is_hand_open(landmarks) zur heuristischen Bestimmung des Handzustands, Stabilisierung der Rohsequenz, Glättung mithilfe von STABLE_MIN_FRAMES und GAP_TOLERANCE.
- decoder.py – Umsetzung der stabilisierten Bitfolgen in Text über Morsecodetabelle, Hamming-Distanz und Lexikon-basierte Suche.

5.2 Wichtige Funktion

```python
def is_hand_open(hand_landmarks):
    """Einfacher Heuristik-Ansatz: Ist die Hand offen?"""
    tips_ids = [8, 12, 16, 20]  # Fingerspitzen
    for tip_id in tips_ids:
        if hand_landmarks.landmark[tip_id].y > hand_landmarks.landmark[tip_id - 2].y:
            return False  # Mindestens ein Finger nicht ausgestreckt
    return True
```

Abbildung 2: is_hand_open-Funktion

5.3 Weitere zentrale Funktionen

- hamming_distance(a, b) – Berechnet die Hamming-Distanz zwischen zwei Bitfolgen.
- expand_lexicon(base_words, morse_table, max_len) – Generiert Wortkombinationen aus Basislexikon.
- decode_with_lexicon_or_estimate(bits, morse_table, lexicon, max_hamming) – Sucht passendes Wort im Lexikon oder liefert beste Schätzung via Beam Search.

5.4 Konfiguration

- frame_rate: wird mit OpenCV automatisch aus dem Video ermittelt.
- STABLE_MIN_FRAMES = 31 – Mindestanzahl Frames für stabilen Zustand (≈1 Sekunde bei 30 fps).
- GAP_TOLERANCE = 20 – Anzahl Frames, die als kurze Unterbrechung toleriert werden.
- MediaPipe-Parameter: max_num_hands=1, min_detection_confidence=0.7, min_tracking_confidence=0.5.

6. Postprocessing & Dekodierung

6.1 Postprocessing

Frames → States → Bits
Jeder Frame des Videos wird zunächst einem Zustand zugeordnet: 1 (Hand offen), 0 (Hand geschlossen), -1 (Hand nicht erkennbar/Pause). Die so entstehende Rohsequenz enthält viele kurze Schwankungen. Ein Postprocessing-Algorithmus glättet die Daten:

- Zustände gelten erst als **stabil**, wenn sie mindestens STABLE_MIN_FRAMES lang anhalten.
- Kurze Unterbrechungen werden über GAP_TOLERANCE ignoriert.
- Nur stabile Zustandsblöcke werden in die finale Bitfolge übernommen.

Beispiel: Eine Sequenz [1,1,1,1,-1,1,1,1] mit GAP_TOLERANCE=2 wird als durchgehende 1 erkannt.

6.2 Zeitschwellen

Im aktuellen Code erfolgt keine explizite Unterscheidung zwischen „Dot“ (kurz) und „Dash“ (lang). Stattdessen wird eine direkte Binärdarstellung genutzt, wobei jeder stabile Zustand in eine 0 (geschlossen) oder 1 (offen) übersetzt wird. Pausen (-1) markieren Trenner zwischen Zeichen oder Wörtern. Die Schwellenwerte in Frames lauten:

- STABLE_MIN_FRAMES = 31 (≈ 1 s bei 30 fps).
- GAP_TOLERANCE = 20 Frames (≈ 0,7 s bei 30 fps).

6.3 Morse-Tabelle

Das Projekt nutzt eine vordefinierte Tabelle, die Buchstaben auf Bitfolgen abbildet:

```
MORSE_TABLE_STR = {
    "A": "01",    "B": "1000",  "C": "1010", "D": "100",  "E": "0",
    "F": "0010",  "G": "110",   "H": "0000", "I": "00",   "J": "0111",
    "K": "101",   "L": "0100",  "M": "11",   "N": "10",   "O": "111",
    "P": "0110",  "Q": "1101",  "R": "010",  "S": "000",  "T": "1",
    "U": "001",   "V": "0001",  "W": "011",  "X": "1001", "Y": "1011",
    "Z": "1100",  " ": ""
}
```

Abbildung 3: Morse-Tabelle

Diese Tabelle wird in eine Dictionary-Struktur MORSE_TABLE überführt, die Listen von Integers (0 und 1) speichert.

6.4 Decoder-Strategie

Die finale Bitfolge wird mit Hilfe eines **Lexikon-gestützten Decoders** interpretiert:

1. **Lexikon-Erweiterung:** Mit expand_lexicon werden Wort- und Phrasenkombinationen aus einer Menge von Basiswörtern gebildet (bis max. 4 Wörter).
2. **Harte Suche:** Die Funktion decode_with_lexicon_or_estimate prüft, ob ein Wort aus dem Lexikon mit der gegebenen Bitfolge übereinstimmt (unter Berücksichtigung einer maximal

7. Evaluation

7.1 Versuchsaufbau

Für die Bewertung wurden mehrere Testvideos genutzt:
(**Hello01.mp4, Hello02.mp4, Hello03.mp4, Hilfe.mp4, Love.mp4, Move.mp4, Arshia.mp4, Elham.mp4, Kasra.mp4, Sadra.mp4**).
Jedes Video enthält Gesten, die einem Wort aus dem vordefinierten Lexikon entsprechen.

```
# --- Wörter ---
BASE_WORDS = {"HALLO", "HELLO", "WELT", "TEST", "HILFE", "MORSE", "CODE",
              "ICH", "MEIN", "NAME", "IST", "BRAUCHE", "ARSHIA", "ELHAM",
              "LOVE", "MOVE", "HELP", "SADRA", "KASRA"}

# Lexikon erweitern (mit dynamischen Kombinationen bis 4 Wörter)
LEXICON = expand_lexicon(BASE_WORDS, MORSE_TABLE, max_len=4)
```

Abbildung 4: Base_Words-Tabelle

Das Programm erzeugt eine finale Bitfolge, die anschließend mit dem Decoder auf ein Wort abgebildet wird.

7.2 Metrik

Das Projekt verwendet eine einzige zentrale Metrik: die **Hamming-Distanz** zwischen erkannter Bitfolge und der erwarteten Referenzbitfolge. Diese gibt die Anzahl der Bitpositionen an, an denen Unterschiede auftreten. Eine geringe Hamming-Distanz bedeutet eine hohe Genauigkeit. Zusätzlich kann auf dieser Basis abgeschätzt werden, ob das Zielwort mit zulässiger Abweichung (max_hamming) korrekt erkannt wurde.

7.3 Ergebnisse

Die Decoder-Ausgaben zeigen, dass alle Wörter wie *HELP, HELLO, HALLO* usw., die sich in dem vordefinierten Lexikon befinden, in zuverlässig erkannt wurden, aber bei den Wörtern wie Informatik, die außerhalb der vordefinierten Lexikonen waren, kam es hingegen zu einer fehlerhaften Dekodierung. Hauptursachen für Fehler könnte die Untersuchungsmethode zur Dekodierung sein. Insgesamt bestätigt die Evaluation, dass die Kombination aus stabilisierten Bitfolgen und lexikonbasiertem Decoder eine funktionierende, aber noch fehleranfällige Lösung darstellt.
Außerdem wurde die Funktionalität des Codes mit verschiedenen Parametern überprüft**, Abstand, Helligkeit und Hintergrundfarbe**, und die Ergebnisse lauten:

Abstand (Cm)	Bis 110	110-150	150-200
Weiß-hell	perfekt	gut	gut
Weiß-halb hell	perfekt	gut	×
Schwarz-hell	perfekt	gut	×
Schwarz-halb hell	perfekt	×	×

8. Ergebnisse

Um die Codeergebnisse im Docoder-Bereich besser auswerten zu können, wurden mehrere verschiedene Beispiel-Bitfolgen eingegeben und deren Ausgabe mit den ursprünglichen Bitfolgen und ihren Wörtern verglichen:

--- Test ---

0: **HELLO**
test0 = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**

1: **HALLO**
test1 = **[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**

2: **HILFE**
test2 = **[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]**

3: **HELP**
test3 = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0]**

Und so weiter.

8.1 Korrekt erkannte Wörter

Mehrere Testvideos führten zu klar erkannten Bitfolgen und korrekter Dekodierung:

- video_path: **Hello01.mp4**
  → Erwartete = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → Ausgabe = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → Beste Wort-Dekodierung: **'HELLO'** oder **'hello'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Hello02.mp4**
  → Erwartete = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → Ausgabe = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → Beste Wort-Dekodierung: **'HELLO'** oder **'hello'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Hallo.mp4**
  → **Erwartete = [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → **Ausgabe = [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1]**
  → Beste Wort-Dekodierung: **'HALLO'** oder **'hallo'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Hilfe.mp4**
  → Erwartete = **[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]**
  → Ausgabe = **[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]**
  → Beste Wort-Dekodierung: **'HILFE'** oder **'hilfe'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Help.mp4**
  → Erwartete = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0]**
  → Ausgabe = **[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0]**
  → Beste Wort-Dekodierung: **'HELP'** oder **'help'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Arshia.mp4**
  → Erwartete = [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
  → Ausgabe = [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
  → Beste Wort-Dekodierung: **'ARSHIA'** oder **'arshia'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Elham.mp4**
  → Erwartete = **[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]**
  → Ausgabe = **[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]**
  → Beste Wort-Dekodierung: **'ELHAM' oder: 'elham'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Kasra.mp4**
  → Erwartete = [1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1]
  → Ausgabe = [1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1]
  → Beste Wort-Dekodierung: **'KASRA'** oder **'kasra'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Sadra.mp4**
  → Erwartete = **[0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1]**
  → Ausgabe = **[0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1]**
  → Beste Wort-Dekodierung: **'SADRA'** oder **'sadra'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Love.mp4**
  → Erwartete = **[0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0]**
  → Ausgabe = **[0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0]**
  → Beste Wort-Dekodierung: **'LOVE'** oder **'love'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- video_path: **Move.mp4**
  → erwartete = **[1, 1, 1, 1, 1, 0, 0, 0, 1, 0]**
  → Ausgabe = **[1, 1, 1, 1, 1, 0, 0, 0, 1, 0]**
  → Beste Wort-Dekodierung: **'MOVE'** oder **'move'**
  → Distanz: 0 | Genauigkeit: **100.0%**

8.2 Korrekte Test-Wörter

- test_Text: **MEINNAME**
  → Erwartete = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]**
  → Ausgabe = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]**
  → Beste Wort-Dekodierung: **'MEIN NAME'** oder **'mein name'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- test_Text: **MEINNAMEIST**
  → Erwartete = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]**
  → Ausgabe = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]**
  → Beste Wort-Dekodierung: **'MEIN NAME IST' oder 'mein name ist'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- test_Text: **ICHBRAUCHEHILFE**
  → Erwartete = **[0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]**
  → Ausgabe = **[0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]**
  → Beste Wort-Dekodierung: **'ICH BRAUCHE HILFE' oder 'ich brauche hilfe'**
  → Distanz: 0 | Genauigkeit: **100.0%**

- test_Text: **MEINNAMEISTARSHIA**
  → Erwartete = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]**
  → Ausgabe = **[1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]**
  → Beste Wort-Dekodierung: **'MEIN NAME IST ARSHIA'** oder **'mein name ist arshia'**
  → Distanz: 0 | Genauigkeit: 100.0%

8.3 Fehlgeschlagene Dekodierungen

Die Wörter außerhalb der **BASE_WORDS-Dictionary**, wie die Folgende, lieferten die richtigen Bitfolgen aber fehlerhafte Wörter:

- test_Text: **Informatik**
  → Erwartete = **[0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1]**
  → Ausgabe = **[0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1]**
  → Beste Wort-Dekodierung: **'ELHAM WELT'** oder **'elham welt'**
  → Distanz: 7 | Genauigkeit: **70.83%**

9. Diskussion

9.1 Interpretation der Ergebnisse

Die Evaluation zeigt, dass alle Wörter aus dem vordefinierten Lexikon ***(HELLO, HALLO, HILFE, HELP, ARSHIA, ELHAM, KASRA, SADRA, LOVE, MOVE,*** *sowie Phrasen wie* ***MEIN NAME IST ARSHIA)*** fehlerfrei erkannt wurden. In diesen Fällen betrug die Hamming-Distanz stets 0, was einer Genauigkeit von 100 % entspricht. Dies bestätigt, dass die Pipeline aus Handzustands-Erkennung, Stabilisierung und Lexikon-Decoder zuverlässig funktioniert, solange die Zielwörter im Lexikon enthalten sind.

9.2 Limitationen

Ein zentrales Problem tritt bei Eingaben auf, die nicht im **BASE_WORDS**-Lexikon definiert sind (z. B. **INFORMATIK**). Obwohl die Bitfolge korrekt generiert wurde, konnte der Decoder das Wort nicht finden und lieferte ein falsches Ergebnis (**ELHAM WELT**). Hier zeigt sich die Abhängigkeit vom Lexikon. Darüber hinaus ist die Gestenerkennung anfällig für kurze Bewegungsunterbrechungen, wechselnde Beleuchtung und Handrotation. Die aktuelle Heuristik (is_hand_open) prüft nur die y-Koordinaten der Finger und ist daher nicht robust gegenüber allen Handhaltungen.

9.3 Verbesserungspotenzial

Für zukünftige Arbeiten sollte das Lexikon erweitert oder dynamisch anpassbar gestaltet werden, um flexiblere Vokabulare zuzulassen. Zudem könnten ML-basierte Klassifikatoren für Handzustände (statt reiner Heuristik) die Robustheit steigern. Auf Decoder-Seite wären probabilistische Ansätze (z. B. Viterbi-Suche) oder n-gramm-Modelle denkbar, um auch unbekannte Wörter mit hoher Wahrscheinlichkeit korrekt zu rekonstruieren. Schließlich könnte eine feinere Unterscheidung zwischen Punkt und Strich die Nähe zum klassischen Morsecode erhöhen und zusätzliche Ausdrucksmöglichkeiten schaffen.

10. Fazit & Ausblick

Fazit:
Das Projekt hat gezeigt, dass Handgesten zuverlässig in Morsecode-ähnliche Bitfolgen übersetzt und anschließend mit einem Lexikon-Decoder in Klartext überführt werden können. Die erreichte Genauigkeit von 100 % für alle Wörter im Lexikon unterstreicht die Praxistauglichkeit des Ansatzes in einfachen Szenarien.

Ausblick:
Zukünftige Arbeiten sollten sich auf die folgenden Aspekte konzentrieren:

- Erweiterung des Lexikons und Unterstützung freier Eingaben.
- Robustere Handzustandserkennung, z. B. mit neuronalen Netzen.
- Integration probabilistischer Dekoder zur Fehlertoleranz.
- Nähere Anlehnung an den klassischen Morsecode durch Zeit-basierte Punkt/Strich-Unterscheidung.

Damit eröffnet das Projekt Perspektiven für Anwendungen in assistiven Technologien, stiller Kommunikation und Notfallszenarien.

11. Anhang

11.1 Quellcode

- Vollständiger Python-Code liegt in der Datei **Bits_Test.py** und **Final_Code.py**
- Bereitstellung über GitHub-Repository für einfacheren Zugriff und Versionierung: „https://github.com/Arshia-R13/Morse_Code“
- Strukturierung für eine Modulaufteilung:
    - **video_io.py**
    - **state_estimator.py**
    - **decoder.py**

11.2 Testfälle

- Im Code sind mehrere Testvideos eingebunden (z. B. Hello01.mp4, Help.mp4, Elham.mp4) und ein zusätzlicher **Ueben.py-**Code.
- Zusätzlich wurden Arrays mit erwarteten Bitfolgen definiert, die für Evaluation und Debugging genutzt werden können:
    - test0 = [0,0,0,0,0,0,1,0,0,0,1,0,0,1,1,1] → HELLO
    - test1 = [0,0,0,0,0,1,0,1,0,0,0,1,0,0,1,1,1] → HALLO
    - test2 = [0,0,0,0,0,0,0,1,0,0,0,0,1,0,0] → HILFE
    - … bis zu test13 (siehe Quellcode).

11.3 Konfiguration

- Alle relevanten Parameter sind direkt im Skript definiert:
    - frame_rate – automatisch ermittelt mit OpenCV.
    - STABLE_MIN_FRAMES = 31 – Mindestanzahl stabiler Frames für eine Zustandsänderung.
    - GAP_TOLERANCE = 20 – Frames, die als kurze Unterbrechung toleriert werden.
    - interval = 3 – Sekunden bis zum Timeout bei Handverlust.
    - MediaPipe-Parameter: max_num_hands=1, min_detection_confidence=0.7, min_tracking_confidence=0.5.
- Beispielkonfigurationsdatei: requirements.txt mit Abhängigkeiten (opencv-python, mediapipe, numpy).

12. Literatur

Liste der referenzierten Arbeiten, Tutorials, Media Pipe-Dokumentation:

- https://www.geeksforgeeks.org/python/morse-code-translator-python
- https://github.com/google-ai-edge/mediapipe/tree/master/mediapipe
- https://how.dev/answers/how-to-write-a-morse-code-translator-in-python
- https://stackoverflow.com/questions/71099952/how-to-decode-morse-code-in-a-more-pythonic-way
- Example scenario, M. Shamsi, Le Mans University, 2024