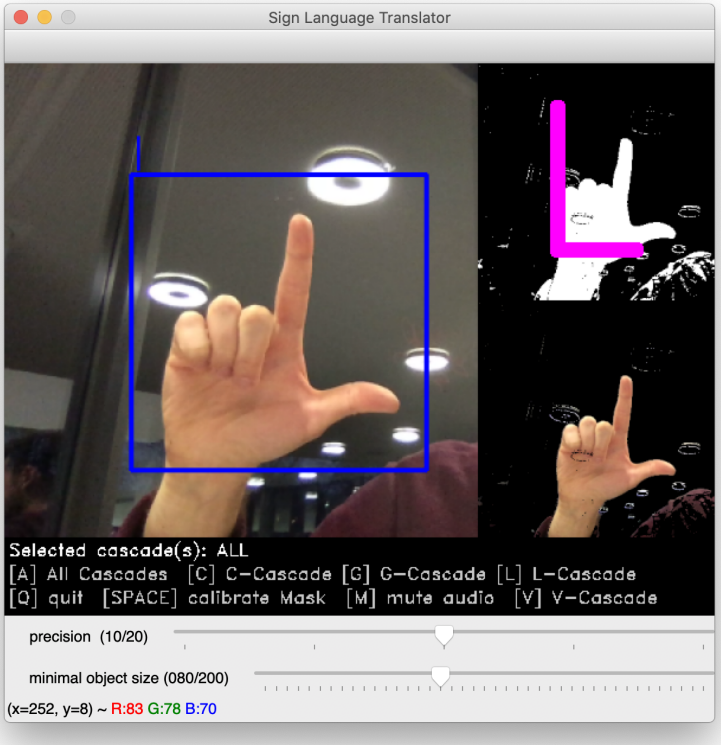
PROJEKTBERICHT: GEBÄRDENÜBERSETZER

1. Einführung

Im Rahmen der Wahlpflicht-Vorlesung Audio-Video-Programmierung sollte als Prüfungsleistung eine Anwendung zum Thema “Video steuert Audio” entwickelt werden. Die Umsetzung fand, nach Absprache mit Herrn Plaß, komplett in Python statt.
Da wir uns sowohl mit dem Machine Learning und der Lösung eines möglichst alltäglichen Problems befassen wollten, haben wir uns für die Realisierung eines Fingeralphabet-Übersetzers entschieden.

2. Projektbeschreibung

Über eine Webcam kann der Gebärdenübersetzer den gezeigten Buchstaben erkennen und sowohl in Textform, als auch gesprochen über die Lautsprecher ausgeben. Aufgrund sehr langer Rechenzeiten haben wir uns exemplarisch auf 4 Buchstaben konzentriert: C, G, L und V. Mit den angewandten Methoden lassen sich mit entsprechender Rechendauer und ggf. höherer Rechenleistung weitere Gebärden integrieren.
Innerhalb des Programms hat man die Möglichkeit Parameter der Objekterkennung einzustellen: Die Empfindlichkeit und die Größe der kleinsten zu analysierenden Struktur.
Des Weiteren lässt sich die Objekterkennung mittels der Tasten “C”, “G”, “L” und “V” auf die entsprechenden Gesten beschränken.
Auf der rechten Seite sieht man zur Demonstration des Hintergrund-Subtraktions-Verfahrens eine Vorschau der Subtraktions-Maske und darunter das letzten Endes analysierte Bild.
Durch Drücken der Leertaste lässt sich die Maske neu kalibrieren.
Mit der Taste “Q” wird die Anwendung beendet.



3. Umsetzung

Den Gebärdenübersetzer haben wir komplett in Python 3 entwickelt. Mittels OpenCV wurden die Videoverarbeitung und zuvor das Machine Learning realisiert. Die Umsetzung lässt sich in folgende zwei Abschnitte Unterteilen:

Cascade-Training

Das Haar-Cascade-Training erforderte vorab eine Reihe von Vorbereitungen.
Zuerst haben wir für verschiedener Gesten des Gebärden-Fingeralphabets (C,G,L und V) je 6 Bilder aufgenommen, wobei wir uns um eine Variation der Haltung und Lichtverhältnisse bemüht haben. Im Laufe der Trainingsläufe hat es sich als qualitätssteigernd erwiesen die Bild-Hintergründe zu schwärzen.
Zusätzlich haben wir eine Reihe von zufälligen Bildern aus dem Internet heruntergeladen (2100 Stück) und diese mittels eines eigenen Hilfsskriptes in Graustufenbilder konvertiert, systematisch benannt und leistungsbedingt auf 100x100 px zugeschnitten. Die originalen Gesten-Bilder wurden auf 50x50 px zugeschnitten.
Die Verwendung des “createsamples”-Skripts ermöglichte es uns mit verhältnismäßig wenig Aufwand für jede Geste 4200 Positivbilder zu generieren. Die 2100 Internet-Bilder wurden als Negativbilder genutzt.
Die Bildgenerierung und das tatsächliche Training benötigten Textdateien mit Dateipfaden und weiteren Informationen, welche wir mittels weiterer eigener Hilfsskripte erzeugt haben.

Da für das Training eine Linux-Umgebung erforderlich war und sich tagelange Rechenzeiten ergaben (teils über 60h), entschieden wir uns dazu trotz Leistungseinbußen einen Raspberry Pi 3B zu nutzen, da wir diesen ungestört zu Hause rechnen lassen konnten und mittels SSH online die Trainingsläufe überwachen und steuern konnten.

Anwendung

Um die erhaltenen Kaskaden zu benutzen, haben wir die Klasse “detection.py” geschrieben. Diese bietet die Möglichkeit mehrere Kaskaden zu verwalten und gesammelt auf ein Bild anzuwenden.
Aufgrund der unterschiedlichen Qualitäten der Kaskaden lassen sich die Empfindlichkeiten mit einem Offsetwert kalibrieren.
Des Weiteren bauten wir eine Fehlerkorrektur ein: Da lediglich eine Hand zur Zeit erwartet wird, entnehmen wir innerhalb eines Frames den am häufigsten vorkommenden Treffer und vergleichen diesen mit

den zehn letzten Durchläufen. Erst dann wird das Ergebnis als “mostCommonLetter” übergeben.

Das Skript “main_script.py” vereint alle Funktionen (Detektion, Audioausgabe, Visualisierung) und beinhaltet die Haupt-Programmschleife.
In der ausgeführten Anwendung entsprechen die im linken Bild erscheinenden Kästen den Ergebnissen der Detektion *vor* der Fehlerkorrektur und der Buchstabe im oberen rechten Bild, sowie die Audioausgabe, sind das Ergebnis *nach* der Fehlerkorrektur.
Als weiteren Optimierungsschritt wird der Webcam-Hintergrund vor der Objekterkennung subtrahiert.
Mit den beiden Reglern lassen sich zum Erkunden der Objekterkennung die Empfindlichkeit und die Größe des kleinsten zu berücksichtigenden Objekts variieren.

Die Soundausgabe haben wir mittels der in “audio_playback.py” ausgelagerten Klasse unter Zuhilfenahme der “Simpleaudio”-Bibliothek verwaltet. Die Audiodateien wurden mit einem Text-To-Speech Hilfsskript generiert.

4. Fazit

Mit OpenCV lassen sich in Python fertige Kaskaden mit nur wenig Aufwand integrieren und weiter verarbeiten. Es hat sich jedoch als sehr zeit- und arbeitsaufwändig erwiesen eigene Kaskaden zu trainieren. Des Weiteren lassen die Ergebnisse, zumindest für Gestenerkennungen, u.a. aufgrund ihrer großen Abhängigkeit von der Lichtrichtung zu Wünschen übrig.
In sich starre Objekte lassen sich wahrscheinlich besser erfassen.
Durch die nachgeschobene Fehlerkorrektur konnten wir die Trefferquoten jedoch erheblich verbessern.
In einem nächsten Schritt wäre es sehr interessant eigene Kaskaden mit dem moderneren “Tensorflow” zu trainieren.
In jedem Fall konnten wir sehr viel von diesen ersten Schritten mit OpenCV und dessen Machine Learning unter Python für zukünftige Projekte lernen.