Digitale Signalverarbeitung Hausarbeit Dokumentation

Levi Asmussen, 750410
Luca Schumacher, 750321
Carlo von Haw, 750236

WiSe 24/25

Inhaltsverzeichnis

1 Beschreibung der Eigenanteile

Wir haben dieses Projekt zusammen in gleichmäßigen Anteilen erstellt. Dies beinhaltet unter anderem:

- Programmierung der Dobot Bewegungen (Levi, Luca, Carlo)
- Programmierung der Gesture Recognition (Levi, Luca, Carlo)
- Entscheidung über Daten Analyse (Levi, Luca, Carlo)
- Schreiben der Dokumentation (Levi, Luca, Carlo)

Dafür haben wir uns überwiegend online getroffen.

2 Einleitung

Unser Programm soll eine menschengemachte Bewegung basierend auf den Daten eines Beschleunigungssensors ohne g erkennen und anschließend die passende Bewegung an einem Dobot ausführen können. Das Bedeutet:

- Bewegung eines Kreises → Zeichnen eines Kreises auf Papier
- Bewegung eines Dreiecks → Zeichnen eines Dreiecks auf Papier
- Bewegung eines Vierecks → Zeichnen eines Vierecks auf Papier

Dabei sollte bedacht werden, dass die Bewegung immer unter den gleichen Voraussetzungen ausgeführt wird:

1. Das Handy wird gerade vor den Körper gehalten, als würde man ein Foto machen
2. Die Bewegung wird über eine Dauer von 5 Sekunden kontinuierlich ausgeführt

3 Aufbau des Programmcodes

Unser Projekt ist wie folgt aufgebaut:

```
Disi Hausarbeit
 lib
    Dobot Python Dateien
 datasets
    circle
       Trainingsdaten
    square
       Trainingsdaten
    triangle
       Trainingsdaten
    Bewegungsdaten zur Analyse
 gesture_model.pkl
 bot_controller.py
 gesture_recognition.py
 main.py
```

Der trainierte Entscheidungsbaum befindet sich in `gesture_model.pkl`. Das Programm wird über die `main.py` gestartet und erwartet dann den Namen einer Datei, die eine aufgezeichnete Bewegung beinhaltet, die analysiert werden soll. Nach Bestätigung der Eingabe mit Enter, wird das Verzeichnis `datasets` nach dem neuen Bewegungsdatensatz durchsucht. Entsprechend der passenden Analyse in `gesture_recognition.py` wird dann die passende Bewegung in `bot_controller.py` aufgerufen.

4 Analyse der Daten

4.1 Vorverarbeitung

Bei einem Beschleunigungssensor ohne g stehen uns die x, y und z Achsen mit einem jeweiligen Zeitstempel zur Verfügung. Bei der Vorverarbeitung laden und filtern wir die Daten und wenden folgende Filter an:

1. Moving-Average-Filter, um die Daten zu glätten
2. Z-Score-Filter, um Ausreißer zu entfernen

4.2 Feature-Engineering

Basierend auf den vorverarbeiteten Daten bilden wir dann folgende Features:

- Durchschnitt (mean)
- Durchschnitt Differenz (mean_diff)
- Energie (energy), also Quadrieren der Werte, was Ausschlägen mehr Gewicht gibt
- Standardabweichung Differenz (std_diff)

Wir haben diese Features gewählt, da diese sich aussagekräftig zwischen den einzelnen Bewegungen unterscheiden.

4.3 Windowing

Nachdem wir die Features erfolgreich gebildet haben, wird der Datensatz in Abschnitte (Windows) aufgeteilt. Nach einigen Versuchen haben wir uns für eine Window Size von 100 und einem Overlap von 50 entschieden, da diese Einstellung die besten Ergebnisse geliefert hat. Jedes Window wird dann entsprechend analysiert und einer Bewegung zugeordnet.

4.4 Ergebnis

Am Ende gewinnt die Bewegung mit den meisten Windows (Majority-Vote).

5 Trainieren des Models und Bildung des Decision-Trees

Wir trainieren dann das Model mit aufgezeichneten Trainingsdaten und unserer Datenanalyse.

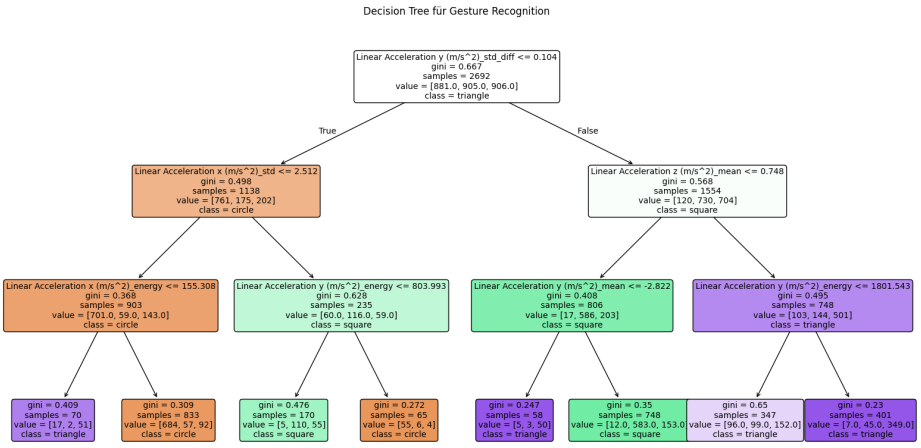
```
Accuracy: 0.7988888888888889
              precision    recall  f1-score   support

      circle       0.85      0.92      0.88       324
      square       0.78      0.84      0.81       297
    triangle       0.74      0.62      0.68       279

    accuracy                           0.80       900
   macro avg       0.79      0.79      0.79       900
weighted avg       0.79      0.80      0.79       900
```

Der Klassifikationsreport

Mit Hilfe der Analyse von Trainingsdaten wird auch ein Entscheidungsbaum gebildet, der dann die Bewegungen erkennen soll. Hierbei haben wir eine Tiefe von 3 und einen Random State von 42 gewählt, da wir immer ein gleiches Ergebnis haben wollen. Dies sowohl beim DecisionTreeClassifier als auch bei dem Aufteilen in Train und Test.



Unser Entscheidungsbaum

6 Diskussion

Die Präsentation unseres Projekts hat noch ein paar Probleme hervorgehoben, die wir im Anschluss verbessert haben:

1. Der Entscheidungsbaum hatte keine Grenze bei der Tiefe, was dazu führen kann dass dieser Trainingsdaten überanpasst und nicht gut auf neue Daten generalisieren kann. Diese haben wir nun auf 3 gesetzt.

2. Wir hatten sehr kleine Windows und somit nur sehr kurze Abschnitte der Bewegung, was dazu geführt hat, dass es immer ein knappes Rennen zwischen zum Beispiel einem Kreis und einem Viereck gab. Das haben wir korrigiert, indem wir jetzt eine Window Size von 100 mit einem Overlap von 50 haben.

3. Unser Entscheidungsbaum wurde jedes Mal bei Programmstart erstellt, was bei komplexeren Programmen extrem inperformant und zeitraubend wäre. Dies würde bei sehr großen Trainingsdaten dazu führen, dass das Progamm sehr lange zum Laden benötigt. Nun trainieren wir den Entscheidungsbaum nur wenn das Model noch nicht vorhanden ist und wenden ihn dann auf neue Datensätze an.

4. Wir hatten zuerst eine Bewegungserkennung für eine schüttelnde Bewegung eingeführt, die das Homing des Dobots auslösen sollte. Das führte

dazu, dass die schüttelnde Bewegung immer als erstes ausgeführt werden musste, damit der Dobot überhaupt gehomed ist. Das Programm haben wir jetzt so umgebaut, dass der Dobot einmal zu Beginn das Homing ausführt und anschließend Bewegungsdaten entgegennimmt.

Zusätzlich haben wir uns entschieden unsere Trainingsdaten zu erweitern. Wir hatten erst ausschließlich Trainingsdaten, die die jeweilige Bewegung immer nur einmal beinhalteten. Nun haben wir Trainingsdatensätze mit unterschiedlichen Längen und unterschiedlichen Wiederholungen der gleichen Bewegung, um die Robustheit des Models zu erhöhen.