2019

Echtzeit Bewegungsklassifizierung mithilfe von Zeitfaltungsnetzen auf Basis von Gelenkerkennung mit Tf-Pose

POL ZEIMET, HOCHSCHULE KARLSRUHE – TECHNIK UND WIRTSCHAFT

Inhalt

Abstract

Diese Projektarbeit beschäftigt sich mit der Erkennung von bestimmten Bewegungen anhand des Gelenkverlaufes bei Personen. Die hierfür benötigten Gelengpositionen sollen mithilfe einer Gelenkerkennung in Echtzeit aus einer 3d-Kameraufnahme entnommen werden. Das Ganze soll auf einer embedded Plattform umgesetzt werden, in diesem Falle eines Jetson TX2 von NVIDIA. Die Verarbeitungsgeschwindigkeit einzelner Bilder beträgt dabei 7-8 FPS und es wird eine Boxbewegung erkannt, die aus dem Bereithalten einer Deckung sowie Zuschlagen besteht.

Einführung

Dieses Projekt ist eine Weiterführung des im WiSe1819 durchgeführten Automatisierungslabor bei dem, Mithilfe von OpenPose und Dynamic Time Warping, eine Kniebeugen Bewegung erkannt wurde. Diese Fortführung setzt dabei auf einen performanteren Embedded Chip mit GPU Funktionalität sowie ein leichtgewichtigeres Modell zur Gelenkerkennung, um höhere Frameraten zu gewährleisten. Die extrahierten Gelenkpunkte sollen dann durch ein Zeitfaltungsnetz einer bestimmten Geste zugeordnet werden, alles in Echtzeit. Die zum Trainieren des Modells benötigten Daten werden aus dem AMASS Datenset generiert.

Ablauf

Der Ablauf der Arbeit unterteilt sich in mehrere Phasen, bei denen jeweils andere Komponenten des Projektes in den Schwerpunkt rücken.

Installation

Unter Installation versteht sich in diesem Zusammenhang die Auflösung der Abhängigkeiten zwischen den Verschiedenen benötigten Libraries und deren Installation. Dies ist insbesondere dadurch erschwert, dass es sich beim Jetson TX2 Board um einen ARM Prozessor handelt. Diese Prozessor-Architektur wird von vielen Libraries nicht von Haus aus unterstützt, was das Aufsetzen einer funktionierenden Arbeitsumgebung deutlich verkompliziert. Für folgende benötigte Libraries waren somit individuelle Lösungen notwendig:

- Python Wrapper für die Realsense Kamera
- Tensorflow
- Tf-Pose
- OpenCV

Die genauen Schritte, die zum Bauen dieser Libraries notwendig waren, finden sich in den Ressourcen, die unter dem Punkt "Das Projekt aufsetzen" in der im Git Repository hinterlegten Dokumentation aufgelistet wurden. Um die Notwendigen Abhängigkeiten an beispielsweise CUDA und cudnn, die zur Nutzung der GPU Beschleunigung der ML Modelle benötigt werden, zu gewährleisten, war auch eine komplette Aktualisierung des Jetson TX2 notwendig. So musste das Board zuerst geflashed und mit der aktuellsten JetPack Version, Version 4.2, aufgesetzt werden. Diese Installation lieferte auch gleich OpenCV mit.

Extrahieren von Gelenkpunkten

Nachdem Kamera, OpenCV und TF-Pose am Laufen sind, können nun die Koordinaten der erkannten Gelenkpunkte extrahiert werden. Dieser Vorgang umfasst jedoch mehrere wichtige Schritte:

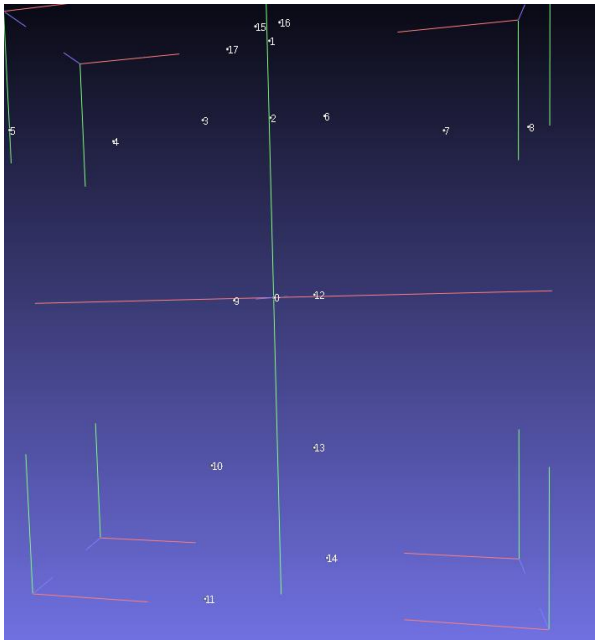
1. Auswahl der benötigten Gelenkpunkte:
   Je nach Anwendungsfall werden nicht alle Gelenkpunkte benötigt. Um Rechenleisung einzusparen werden nicht benötigte Keypoint weggelassen.
2. Berechnen eines Körpermittelpunktes:
3. Mithilfe ausgewählter Gelenkpunkte der linken und rechten Hüftseite wird ein Körperschwerpunkt berechnet.
4. Berechnung der z-Koordinate der 2D-Gelenkpunkte:
5. Überlappung des 2d-Farbbildes mitsamt erkannten 2D-Gelenkkoordinaten mit dem Tiefenbild der Realsense Kamera um 3D Koordinaten für jeden Keypoint zu berechnen
6. Umrechnen der Pixelkoordinaten in das Koordinatensystem der Kamera
7. Transformation der Keypoint-Koordinaten ins Koordinatensystem des Körpers:
   Um die Daten jeder Bewegung kohärent und unabhängig von der Position des Menschen im Raum darzustellen, transformieren wir die Keypoints in ein Körpereigenes Koordinatensystem. Dieses bauen wir auf, indem wir eine x-Achse über den Körperschwerpunkt und die linke Hüft-Seite aufziehen und eine y-Achse vom Körperschwerpunkt nach oben zum Halsansatz. Die z-Achse ergibt sich dann aus dem Kreuzprodukt der beiden anderen Achsen. Die so entstandenen Vektoren werden normiert und stellen dann die Normal-Vektoren unseres neuen Koordinatensystems dar. Nebeneinander aufgestellt ergeben sie die Rotationsmatrize, mithilfe derer wir alle Keypoints in das neue Koordinatensystem transformieren können.

Die Gleiche Vorgehensweise verfolgen wir bei der Extrahierung von Keypoints aus dem AMASS Datensatz, der uns als Trainingsdatensatz dienen wird.

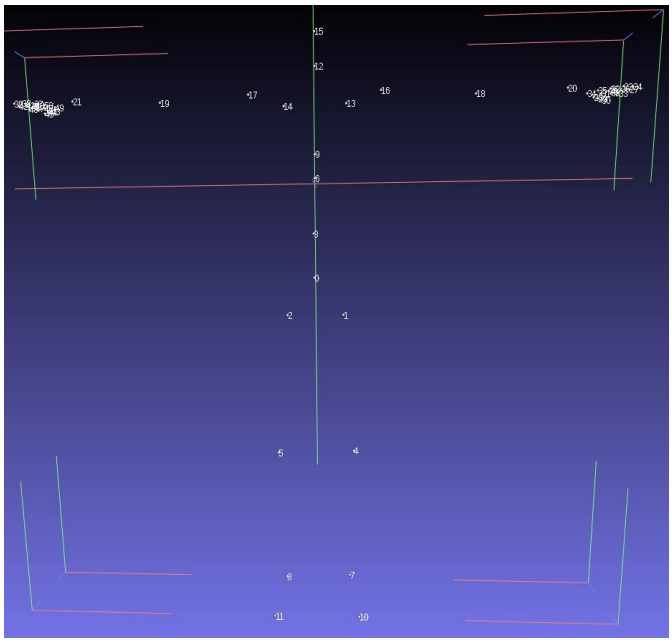
Wichtig hierbei ist, dass Keypoints gewählt werden, die ein Mapping zwischen den von AMASS verwendeten SMPL-H Körpermodellen und dem Tf-pose Modell erlauben. Beim Abspeichern der Daten ist somit auch die Reihenfolge der Keypoints in der Datenstruktur ausschlaggebend. Ein solches Mapping sieht in unserem Projekt wie folgt aus:

Tf-Pose	SMPL-H
1	15
2	(17+16)/2
3	17
4	19
5	21
6	16
8	18
8	20
9	2
10	5
11	8
12	1
13	4
14	7

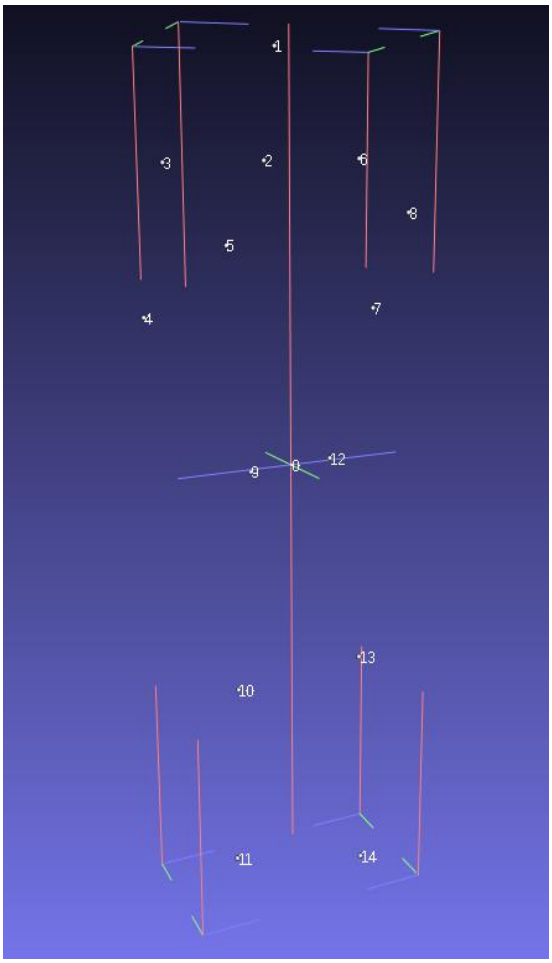
Die Gelenkpunkte/Keypoints:



Das Tf-Pose Gelenkmodell



Das SMPL-H Gelenkmodell



Das SMPL-H Modell nach Transformation in Box-Pose

Generieren von Trainingsdaten

Mit der Möglichkeit geschaffen, für jeden Frame die Keypoint-Koordinaten zu extrahieren, kann nun auf dieser Basis mit aufgenommenen Bewegungssequenzen ein Trainingsdatenset generiert werden. Das AMASS Datenset bietet hier eine breite Auswahl an verschiedenen Bewegungsmustern, aufgenommen mit modernen Motion-Capture Geräten und von Personen verschiedenster Statur. SMPLH Modelle erlauben sogar, darüber hinaus mit den gleichen Aufnahmen weitere Sequenzen zu generieren, indem die Körperproportionen verändert werden. Dies wird jedoch in dieser Arbeit direkt nicht weiter betrachtet.

Die Datengenerierung läuft nun so ab, dass für jede Sequenz, je nach aufgenommener Person, das entsprechende SMPL-H-Modell gewählt wird. Dieser Modell werden die in der Sequenz hinterlegt Körperproportionen übergeben. Für jedes Frame in der Sequenz wird nun dem Modell die passende Pose angeheftet, das Modell geladen und die Keypoints extrahiert. So entsteht eine Abfolge an Keypoints für jede Bewegung. In jede Trainingssequenz werden jeweils 15 Frames geladen. Um eine höhere Sequenzmenge zu bilden, werden jeweils 2 Trainingssequenzen gleichzeitig gefüllt, jeweils mit einem Abstand von 8 Frames zueinander. Da die Sequenzen mit 120 FPS aufgenommen werden, wir aber nur maximal 7-8 FPS mit unserer Gelenkerkennung, sampeln wir die Datensätze langsamer und nehmen nicht jedes Frame, sondern jedes 120/7 = 17te Frame. Jede Sequenz wird dann noch annotiert, ob es sich um die zu erkennend Bewegung, handelt oder nicht.

Die Datenstruktur sieht dann für eine Sequenz von 15 Frames folgendermaßen aus:

[[[keypoint0],[keypoint1]...,[keypoint10]],Klasse]

Mit: [[keypointx] = [x1,y1,z1,x2,y2,z2,.....,x15,y15,z15]

Ein Zweidimensionales Array mit Höhe 11, Breite 45 und Tiefe 1: (11,45,1).

Modell trainieren

Das Trainieren des Modells erwies sich nach einer kurzen Vertraumachung mit Tensorflow und Keras als recht unkompliziert. Lediglich die genause Struktur des Netzes sowie die Dimensionen der Zeitfaltung erforderten genauere Überlegungen.

So soll nur über die Zeit hinweg gefaltet werden, um den Verlauf von Bewegungen zu analysieren. Nicht aber soll über die verschiedenen Keypoins hinweg gefaltet werden. Der erste Falsungskern sieht dementsprechend folgendermassen aus:

Ein (11,9) Faltungskern, der über 3 frames hinweg faltet und sich ein jeweils ein Frame(Also um eine Koordinate mit 3 Werten) weiter bewegt.

x1,y1,z1,x2,y2,z2,x3,y3,z3	x4,y4,z4,...,x15,y15,z15
x1,y1,z1,x2,y2,z2,x3,y3,z3	x4,y4,z4,...,x15,y15,z15
…	…
x1,y1,z1,x2,y2,z2,x3,y3,z3	x4,y4,z4,...,x15,y15,z15
x1,y1,z1,x2,y2,z2,x3,y3,z3	x4,y4,z4,...,x15,y15,z15

Modell Zusammenfassung:

Das Netz insgesamt hat folgenden Aufbau:

- Faltung mit (11,9) Kernel und 32 Filtern Output
- Maxpooling mit (1,3) Kernel
- Faltung mit (1,3) Kernel und 64 Filtern Output
- Flattening der Daten
- 4 versteckte Layer für voll verbundenes neuronales Netzwerk mit 100 Schichten runter auf 20
- 1 Ausgangs-Layer für voll verbundenes NN mit einer Ausgangsschicht und Sigmoid Aktivierung für binäre Klassifizierung der Bewegung.

Schichten:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 1, 13, 32)	3200
max_pooling2d (MaxPooling2D)	(None, 1, 4, 32)	0
conv2d_1 (Conv2D)	(None, 1, 2, 64)	6208
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 100)	12900
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 20)	1020
dense_3 (Dense)	(None, 20)	420
dense_4 (Dense)	(None, 1)	21

Total params:	28,819
Trainable params:	28,819
Non-trainable params:	0

Modell Metriken:

Genauigkeit auf Trainingsdaten: 0.9189

Konfusions Matrix:

Cl1	283	6
Cl2	24	57
Is/predicted	Cl1	Cl1

Genauigkeit auf Testdaten: 0.9692

Konfusions Matrix:

Cl1	50	1
Cl2	1	13
Is/predicted	Cl1	Cl1

Resultate und Fazit

Das Modell an sich wurde überraschend genau. In der Echtzeiterkennung liegt die Framerate bei 7 FPS und Boxen wird oft korrekt erkannt, obwohl das Datenset in der ersten Iteration vergleichsweise klein war, mit 435 Sequenzen insgesamt, davon 95 für die Boxbewegung. Das Modell reagiert besonders auf eine hoch gehaltene Deckung, weniger auf das 'Boxen' mit der Faust. Boxen mit der linken Hand wird in 50% der Fälle erkannt, mit der rechten Hand hingegen so gut wie nie.

Dies liegt wahrscheinlich an den Trainingsdaten, welche in jedem Fall eine hochgehaltene Deckung und oftmals keine Schläge beinhalten. Auch scheint es mehr Schläge mit der linken als mit der rechten Hand gegeben zu haben.

Alles in Allem ist die Leistung es Modelles im Rahmen der Projektarbeit allemal zufriedenstellend.

Mögliche Erweiterungen und Verbesserungen

Da das volle Potential des AMASS Datensatzes nicht ausgenutzt wurde, sondern nur der HumanEva Datensatz zum Training verwendet wurde, kann davon ausgegangen werden, dass das Modell noch enorm verbessert werden kann. Auch eine Optimierung bei der Koordinatentransformation und anderen Stellen in der Echtzeitauswertung würde durchaus die Framerate bei der der Echtzeiterkennung verbessern. Es bietet sich an, das Modell zur Bewegungserkennung auf mehrere Bewegungen auszuweiten.

Um die Wartezeit beim Starten der Echtzeitauswertung zu verringern könnte die TensorRT Engin, die jedes Mal neu gebaut wird, abgespeichert und geladen werden. Auch das Modell zur Bewegungserkennung könnte potentiell mit TensoRT GPU-beschleunigt werden.