6D Pose Estimation - Morobot

1. Projektübersicht

In diesem Projekt soll ein Objekterkennungs- und Visualisierungssystem umgesetzt. Grundlage sind RGB-Bilder, Tiefenbilder und 3D-Modelle im .obj-Format.

Die Hauptschritte sind: Training eines YOLO-Modells zur Objekterkennung auf Basis der RGB-Bilder und deren Labels, Anwendung des trainierten Modells zur Detektion in neuen Bildern, Nutzung der 3D-Modelle, um die erkannten Objekte visuell zu überlagern, Speicherung der Ergebnisse in Form von Visualisierungen (PNG) und begleitenden Textdateien.

---

2. System- und Softwarevoraussetzungen

Das Projekt wurde unter folgender Umgebung getestet:
Betriebssystem: Windows 11 Home 64-bit
CPU: Intel(R) Core(TM) i5-12450H
GPU: NVIDIA GeForce RTX 3050
CUDA-Treiber: Version 12.5
Python: 3.10.18 (Conda, Environment happypose)
PyTorch: 2.7.1+cu118
Torchvision: 0.22.1+cu118
Torchaudio: 2.7.1+cu118
Ultralytics: 8.3.181 (für YOLO)

---

3. Installation

In meinem Projekt verwende ich Conda Version 25.5.1 ; Python 3.10.18

3.1 Conda-Umgebung erstellen

```
conda create -n happypose python=3.10
conda activate happypose
```

3.2 Installation der Abhängigkeiten

PyTorch mit CUDA-Unterstützung (angepasst an CUDA 11.8 Build):

```
pip install torch==2.7.1+cu118 torchvision==0.22.1+cu118 torchaudio==2.7.1+cu118 --index-url
https://download.pytorch.org/whl/cu118
```

Ultralytics (für YOLO):

pip install ultralytics==8.3.181

Zusätzliche Pakete:

pip install opencv-python numpy matplotlib pandas pillow pyyaml tqdm

Optional für 3D-Verarbeitung:

pip install trimesh open3d

---

4. Ablauf

Einleitend möchte ich erwähnen, dass ich im Verlauf dieses Projekts mehrfach zwischen unterschiedlichen Tools und Frameworks wechseln musste. Immer wieder gab es technische Sackgassen, an denen es scheinbar nicht mehr weiterging. Mein letzter Versuch, vollständig auf Ubuntu umzusteigen, brachte leider keine Abhilfe. Auch MegaPose stellte sich für mich als Endpunkt heraus, vor Allem aufgrund von Versionskonflikten und Schwierigkeiten bei der Nutzung meiner GPU. Für mich schien es, als würde es kurz peaken und dann evtl. aufgrund von nicht korrekt gestarteten Zusatzprozessen im unendlichen Loop verharren, die Auslastung der GPU blieb hierbei interessant niedrig.. Zusätzlich kam es wiederholt zu Terminal-Abstürzen. In der Fehlersuche habe ich weiters versucht, Zustandsvariablen direkt im Terminal zu setzen, jedoch ohne Erfolg. Selbst in den wenigen Fällen, in denen das Programm stabil durchlief, waren die Ergebnisse schlechter und insgesamt unbrauchbar im Vergleich zu meinen Versuchen unter Windows. Daher fiel die Entscheidung nach scheinbar unmöglicher Nutzung auf die am Ende erwähnte Methodik.

Um weitere Trainingsdaten zu gewinnen, habe ich zusätzlich ein eigenes Render-Skript geschrieben. Die Idee war, die vorhandenen CAD-Modelle einzulesen und daraus synthetische Bilder mit zugehörigen JSON-Annotationen zu erzeugen. Dazu werden alle .obj-Dateien aus dem models-Ordner geladen, normalisiert und zentriert, bevor jeweils ein Bild mit pyrender gerendert wird. Auf Basis dieser synthetischen Daten habe ich ein weiteres YOLO-Training gestartet, diesmal mit dem Modell yolov8m.pt. Hierfür legte ich ein eigenes YAML-File (synthetic_data.yaml) an, in dem 20 Klassen unterschieden werden – jeweils nach Bauteiltyp und Farbe, z. B. „morobot-s_Achse-1A_gray“. Das Training lief über 50 Epochen mit einer Bildgröße von 640 Pixeln und einer Batchgröße von 16. In der praktischen Anwendung erwies es sich jedoch als besonders schwierig, die so trainierten Modelle auf reale Aufnahmen zu übertragen. Auf den Box-Bildern konnten die Objekte kaum zuverlässig erkannt werden.

Für die eigentliche und finale Pose-Schätzung habe ich schließlich eine eigene Pipeline aufgebaut, in der ich mehrere Verfahren kombiniere. Die grobe Ausrichtung erfolgt zunächst über eine Registrierung mit RANSAC. Dabei werden Punktwolken-Features

(FPFH) extrahiert und miteinander verglichen, um ein erstes Alignment zwischen CAD-Modell und Tiefendaten herzustellen. Hierfür griff ich auf ältere RGB-D-Aufnahmen zurück, da eine erneute Datenerfassung aus Zeitgründen nicht mehr möglich war, meine zuvor erzeugten Bounding Boxes ließen sich in diesem Schritt vermutlich nicht nutzen. Auf diese RANSAC-Vorinitialisierung folgt eine klassische ICP-Optimierung, die die Pose schrittweise verfeinert, indem sie die Abstände zwischen korrespondierenden Punkten minimiert. Sobald Farb- und Texturinformationen vorliegen, setze ich zusätzlich Colored ICP ein, um die Genauigkeit zu erhöhen. Da die untersuchten Bauteile in vielen Bereichen sehr ähnlich geformt sind, prüfe ich zudem verschiedene 180°-Rotationen (Flips), um Orientierungsfehler zu erkennen und gegebenenfalls zu korrigieren.

Das YOLO-Modell (yolov8l) wurde anhand eines Youtube-Tutorials trainiert. Die dafür notwendigen Annotationen habe ich mit dem Online-Tool CVAT erstellt. Alle Objekte wurden dabei manuell in den Bildern markiert. Im Verlauf gab es mehrere Versuche, den Datensatz robuster zu gestalten, letztlich kam jedoch aufgrund gescheiterter Ansätze ein relativ kleiner Datensatz mit den RGB-Bildern zum Einsatz.

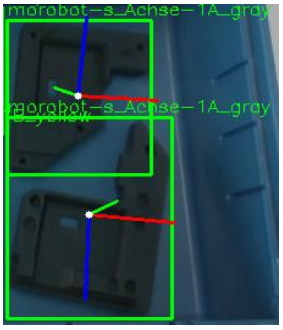
Yolo-Training:
https://youtu.be/Bzv58L6xYGc?si=LoRBARK1VhavOIdU

Annotations:
Online Tool “CVAT” (https://www.cvat.ai/)

5. Ergebnisse und Diskussion

Die finale Objektdetektion erfolgte mit YOLOv8 (Ultralytics, Modellvariante yolov8l). Die so erzeugten Bounding Boxes dienten anschließend als Grundlage für die ICP-Pipeline, in der die eigentliche Pose-Schätzung umgesetzt wurde.

Die Ergebnisse sind in Form visualisierter Posen gespeichert (out_vis/). Dabei zeigt sich, dass die Orientierungen teilweise stark springen, obwohl auf einzelnen Aufnahmen fast korrekte Posen erkennbar zu sein scheinen, wie z.B. hier ->



Technisch entsteht die Pose so, dass zunächst aus der Bounding Box eine grobe Position geschätzt wird, bevor RANSAC das Modell an der Punktwolke ausrichtet und ICP die Pose weiter verfeinert. Am Ende resultiert eine 4×4-Transformationsmatrix, mit der das Koordinatensystem ins Objekt gelegt und im RGB-Bild über Achsen dargestellt wird.

Die häufig falsche oder instabile Ausrichtung lässt sich auf mehrere Faktoren zurückführen: Unterschiede in der Skalierung und Orientierung der CAD-Modelle, ungenaue Tiefenwerte als Initialschätzung, abweichende Kameraparameter zwischen Rendering und Erkennung sowie die allgemeine Anfälligkeit von ICP. Zusammengenommen führen diese Einflüsse dazu, dass die Pipeline zwar prinzipiell durchläuft, aber keine konsistenten Ergebnisse liefert.

Aus zeitlichen Gründen habe ich keine weitergehende Normalisierung der Modelle oder Optimierung der Registrierungsparameter mehr umgesetzt. Besonders die Versuche mit FoundationPose und MegaPose hielt ich zunächst für sehr vielversprechend, verlangten am Ende jedoch viel Zeit ab, ohne zu verwertbaren Resultaten zu führen. Oft scheiterte schon der Durchlauf, oder die erhaltenen Ergebnisse enthielten schlicht keine gültigen Detections – oder eine falsche.

---

9. Zusammenfassung

In diesem Projekt wurde ein System zur Objekterkennung und 6D-Pose-Schätzung für den Morobot entwickelt, basierend auf RGB-Bildern, Tiefendaten und 3D-CAD-Modellen im .obj-Format. Die Objektdetektion erfolgte mit YOLOv8 (yolov8l), wobei die Bounding Boxes als Grundlage für die anschließende Pose-Schätzung dienten. Für die finale Pose-Schätzung wurde eine Pipeline aufgebaut, die zunächst eine grobe Ausrichtung über RANSAC-Registrierung mit Punktwolken-Features (FPFH) vornimmt, anschließend klassische ICP-Optimierung durchführt und, wenn Farb- und Texturinformationen vorliegen, Colored ICP verwendet. Zusätzlich werden verschiedene 180°-Rotationen geprüft, um Orientierungsfehler zu erkennen und zu korrigieren.

Zur Datenerweiterung wurden synthetische Trainingsbilder aus CAD-Modellen erzeugt und für ein separates YOLO-Training verwendet. Diese Modelle erwiesen sich jedoch als nur bedingt übertragbar auf reale Bilder und wurden nicht für die finale Pose-Schätzung eingesetzt.

Die Ergebnisse zeigen, dass die Pipeline prinzipiell funktioniert: Aus der Bounding Box wird eine grobe Position geschätzt, RANSAC richtet das CAD-Modell an der Punktwolke aus, und ICP verfeinert die Pose zu einer 4×4-Transformationsmatrix, die das Objektkoordinatensystem im RGB-Bild abbildet. Die Ausrichtung ist jedoch teilweise instabil, verursacht durch Unterschiede in CAD-Modellen, ungenaue Tiefendaten, abweichende Kameraparameter und die Anfälligkeit von ICP bei ähnlichen Bauteilen. Trotz dieser Limitationen zeigt das Projekt, wenn auch fehlerhaft, die Methodik und die grundsätzliche Funktion einer kombinierten Detektions- und Pose-Schätzpipeline. Besonders zuletzt viel Zeit in das Projekt gesteckt im Versuch, die jeweiligen Tools zur Funktionstüchtigkeit überzuleiten – leider ohne Erfolge.