



Software-Projekt Mobile und Verteilte Anwendungen: Unterstützung für FRoST

Projektleiter: Prof. Dr. -Ing. Matthias Deegener

Team Objekterkennung

[Joao Paka Antonio \(1221655\)](#)

[Ante Maric \(1273904\)](#)

Inhaltsverzeichnis

- I. Einleitung
- II. Theoretische Grundlagen
- III. Einrichtung der Entwicklungsumgebung
- IV. Implementierung und Tests
- V. Ergebnisse
- VI. Diskussion und Analyse
- VII. Schlussfolgerungen und Ausblick



GitHub link zum Projekt: <https://github.com/Jakhaar/FRoST-Project>

I. Einleitung

Im Rahmen des Studienprojekts für das Modul „Software-Projekt Mobile und Verteilte Anwendungen“ widmen wir uns der Entwicklung einer Schlüsseltechnologie für das Mars Rover Team „FRoST“. Unser Projekt zielt darauf ab, eine zuverlässige Objekterkennungslösung zu entwickeln, die den Rover in der anspruchsvollen Umgebung des Mars unterstützen soll.

Die Herausforderung unseres Projekts liegt in der präzisen Identifizierung und Klassifizierung von Objekten sowie der exakten Tiefenerkennung, um den Rover sicher und effizient durch unwegsames Gelände navigieren zu können. Hierfür setzen wir zunächst auf das Jetson Board TX2 in Kombination mit der Stereo Kamera ZED 1, um die benötigten Bilddaten zu erfassen und zu verarbeiten. Die ZED 1 Kamera soll es uns ermöglichen, sowohl die Objekterkennung als auch die Tiefenmessung zu realisieren, indem sie stereoskopische Bilddaten liefert, die für unsere Algorithmen zur Objekterkennung und Tiefenschätzung unverzichtbar sind.

Das Hauptziel unseres Projekts besteht darin, ein robustes und genaues System zu entwickeln, das es dem FRoST-Team ermöglicht, Hindernisse wie Steine oder Löcher auf der Marsoberfläche zuverlässig zu erkennen. Diese Erkenntnisse sind von entscheidender Bedeutung, um den Rover sicher durch verschiedene Geländearten zu navigieren und mögliche Gefahren frühzeitig zu identifizieren. Durch die erfolgreiche Umsetzung unserer Lösung hoffen wir, einen wesentlichen Beitrag zur Wettbewerbsfähigkeit und Funktionsfähigkeit des FRoST Mars Rovers leisten zu können.

II. Theoretische Grundlagen

Spatial Mapping

Spatial Mapping ist ein Verfahren, das die dreidimensionale Erfassung und Modellierung physischer Umgebungen in Echtzeit ermöglicht. Es beruht auf der Erfassung von Tiefeninformationen, die durch Stereovision, Time-of-Flight (ToF), LIDAR oder andere Sensorik gewonnen werden. Die ZED-Kamera nutzt Stereovision, bei der zwei Kameras im festen Abstand zueinander verwendet werden, um die Tiefe eines jeden Bildpunkts durch Triangulation zu berechnen.

Die erfassten Tiefendaten werden in einer Punktfolge dargestellt, einer Sammlung von Punkten in einem dreidimensionalen Raum, die die Oberflächen der gescannten Umgebung repräsentieren. Diese Punktfolge wird dann in ein 3D-Mesh umgewandelt, das durch Polygone definiert wird und die Geometrie der Umgebung detailliert darstellt. Diese Modelle können für verschiedene Anwendungen wie Robotik, Augmented Reality (AR), virtuelle Realität (VR) und Architektur verwendet werden, wo eine präzise Darstellung der Umgebung erforderlich ist.

Funktionsweise der Objekterkennung

Die Objekterkennung in Computer Vision ist ein Prozess, bei dem Objekte innerhalb von Bildern oder Video Streams identifiziert und lokalisiert werden. Diese Technologie verwendet Algorithmen, die darauf trainiert sind, Muster und Merkmale zu erkennen, die spezifischen Objekten entsprechen. Heutzutage werden tiefe neuronale Netze, insbesondere Convolutional Neural Networks (CNNs), häufig für diese Aufgabe verwendet. CNNs sind darauf spezialisiert, visuelle Daten zu verarbeiten, indem sie Bildmerkmale hierarchisch von einfachen Kanten zu komplexeren Strukturen lernen.

Ein häufig verwendetes Modell für Objekterkennung ist YOLO (You Only Look Once), das auf Echtzeit-Performance optimiert ist und die Bilder in einem Durchgang analysiert. Andere gängige Modelle sind Faster R-CNN und SSD (Single Shot MultiBox Detector). Diese Modelle sind auf großen, diversifizierten Datensätzen vorgebildet und können für spezifische Erkennungsaufgaben feinabgestimmt werden.

Kombination von Spatial Mapping und Objekterkennung

Die Kombination von Spatial Mapping und Objekterkennung eröffnet neue Möglichkeiten in der Umweltwahrnehmung und Interaktion. Während Spatial Mapping eine präzise geometrische Repräsentation der Umgebung erstellt, liefert die Objekterkennung semantische Informationen, indem sie spezifische Objekte innerhalb dieser Umgebung identifiziert. Diese Kombination ermöglicht es Systemen, die physische Struktur einer Szene zu verstehen und gleichzeitig Objekte wie Möbel, Menschen oder Hindernisse zu erkennen und zu lokalisieren.

Ein praktisches Beispiel dieser Integration findet sich in autonomen Robotern, die durch Räume navigieren und gleichzeitig Objekte wie Türen, Treppen oder Hindernisse identifizieren müssen. Auch in AR-Anwendungen ist diese Technologie essentiell, da virtuelle Objekte korrekt in die physische Umgebung eingebettet und mit realen Objekten interagieren müssen.

Herausforderungen und Lösungen in der Umsetzung

Die Implementierung von Spatial Mapping und Objekterkennung bringt einige Herausforderungen mit sich:

- **Rechenleistung und Echtzeitfähigkeit:** Die Verarbeitung großer Mengen an Tiefendaten und die parallele Ausführung von Objekterkennung Algorithmen erfordern erhebliche Rechenressourcen. GPUs spielen hierbei eine entscheidende Rolle, da sie die Verarbeitungsgeschwindigkeit erheblich steigern können.
- **Genauigkeit und Zuverlässigkeit:** Die Qualität der Ergebnisse hängt stark von den genutzten Algorithmen und der Kalibrierung der Kameras ab. Eine ungenaue Tiefenmessung kann zu fehlerhaften Meshes führen, während unzureichend trainierte Erkennungsmodelle Objekte falsch identifizieren oder übersehen werden können.
- **Integration und Kompatibilität:** Die Integration unterschiedlicher Software-Bibliotheken (wie das ZED SDK und TensorFlow) erfordert eine sorgfältige Abstimmung von Versionen und Abhängigkeiten. Probleme bei der Installation und Konfiguration der benötigten Softwarekomponenten können den Entwicklungsprozess verzögern.

Lösungen für diese Herausforderungen umfassen die Optimierung der Algorithmen, den Einsatz spezialisierter Hardware für die Berechnungen und die Verwendung gut dokumentierter und getester Bibliotheken, um Integrationsprobleme zu minimieren. Ein weiteres wichtiges Element ist die regelmäßige Kalibrierung der Kameras und die kontinuierliche Überwachung der Modellleistung, um eine hohe Genauigkeit und Zuverlässigkeit sicherzustellen.

III. Einrichtung der Entwicklungsumgebung

Linux (Ubuntu 18.04)

Einrichtung der Entwicklungsumgebung

Die erfolgreiche Einrichtung der Entwicklungsumgebung auf dem Jetson Board TX2 stellte den anfänglichen Schritt in unserem Projekt dar. Diese Phase umfasste die Installation und Konfiguration verschiedener Software- und Bibliothekskomponenten, um die notwendige Unterstützung für die Objekterkennungslösungen bereitzustellen. Die Installation und Konfiguration der Entwicklungsumgebung auf dem Jetson TX2 stellte sich als komplexe Aufgabe heraus.

Installation der NVIDIA SDK und weiterer Komponenten

Zunächst wurde das Jetson Board TX2 mit der NVIDIA SDK (JetPack) konfiguriert. Die Installation dieser Software-Suite ist unerlässlich, um die GPU-Beschleunigung und die Entwicklungsbibliotheken für maschinelles Lernen optimal zu nutzen. Der bereitgestellte Uni-Laptop von Prof. Dr. -Ing. Matthias Deegener ermöglichte uns den Zugriff auf die notwendigen Ressourcen und das Durchführen der Installation. Hierbei arbeiteten wir mit der JetPack-Version 4.6.1, die mit unserem System kompatibel war. Die konkrete Version des NVIDIA-L4T-Core-Pakets, welches für die grundlegende Systemarchitektur zuständig ist, betrug 32.7.4-20230608211515, wie durch den Befehl `dpkg-query --show nvidia-l4t-core` ermittelt.

Trotz dieser Vorbereitungen traten anfangs Herausforderungen auf. Besonders die Integration von YOLO (You Only Look Once) in unsere Umgebung erwies sich als problematisch, da die Kompatibilität der YOLO-Versionen mit dem Jetson TX2 und der JetPack-Version eingeschränkt war. Daher entschlossen wir uns, TensorFlow als Alternative zu wählen. Die TensorFlow-Version 2.7.0 wurde installiert, um den Anforderungen der Objekterkennung gerecht zu werden. Hierzu führten wir den Befehl zur Installation von TensorFlow mit der passenden Version und dem NVIDIA Framework Wheel aus:

```
sudo pip3 install --extra-index-url  
https://developer.download.nvidia.com/compute/redist/jp/v461/tensorflow==2.7.0+nv22.1
```

Trotz dieser Maßnahme traten erneut Versionskonflikte auf. Die Lösung bestand darin, das TensorFlow-Paket direkt als vorgefertigtes Wheel für Jetson TX2 herunterzuladen und zu installieren. Dies wurde durch die folgenden Schritte erreicht:

1. Herunterladen des TensorFlow-Pakets:

```
wget  
https://developer.download.nvidia.com/compute/redist/jp/v46/tensorflow/tensorflow-2.7.0+nv22.1-cp36-cp36m-linux\_aarch64.whl
```

2. Installation des Pakets:

```
pip3 install tensorflow-2.7.0+nv22.1-cp36-cp36m-linux_aarch64.whl
```

Konfiguration weiterer Entwicklungsbibliotheken

Neben TensorFlow wurden auch andere essentielle Bibliotheken wie OpenCV, Cython und NumPy für die Entwicklung und das Testen der Objekterkennungsalgorithmen installiert. Diese Bibliotheken sind unerlässlich für die Bildverarbeitung und numerische Berechnungen, die im Rahmen der Objekterkennung erforderlich sind.

Zur Überprüfung der Installation und Konfiguration testeten wir die ZED Explorer-Anwendung unter folgendem Pfad:

```
/usr/local/zed/tools/ZED_Explorer
```

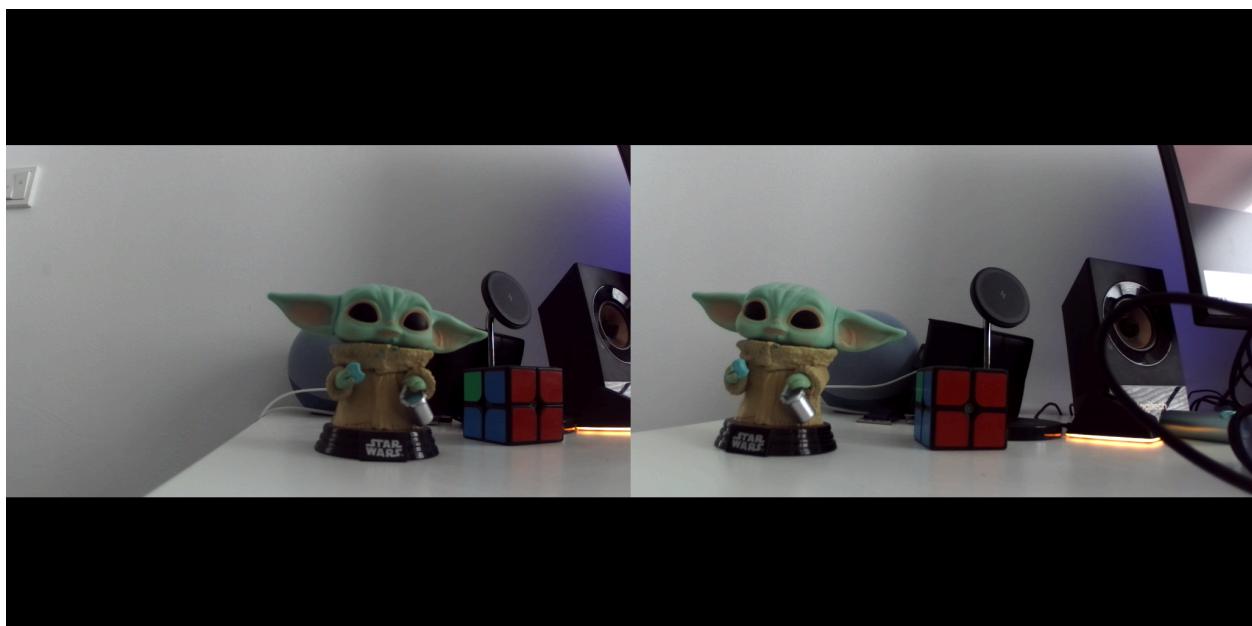
Diese Tests halfen uns sicherzustellen, dass die ZED Kamera ordnungsgemäß funktioniert und die notwendigen Bilddaten für die weiterführende Entwicklung bereitstellt.

Zusammenfassend war die Einrichtung der Entwicklungsumgebung auf dem Jetson Board TX2 ein komplexer Prozess, der sorgfältige Planung und Anpassung erforderte. Die erfolgreiche Installation und Konfiguration der benötigten Software- und Hardware-Komponenten bildet die Grundlage für die weitere Entwicklung der

Objekterkennungslösungen, die dem Mars Rover Team „FRoST“ zur Verfügung gestellt werden sollen.

Mit der konfigurierten Entwicklungsumgebung begannen wir die Implementierung der Objekterkennungslösung. Der entwickelte Code sollte die Bilddaten der ZED 1 Kamera verarbeiten und die gewünschten Objekte erkennen. Trotz erfolgreicher Bildübertragung zeigte die Kamera jedoch keine Objekterkennungsmarkierungen. Diese Diskrepanz deutete darauf hin, dass die ZED 1 Kamera nicht die erforderlichen Funktionen für eine effektive Objekterkennung bot.

```
nvidia@ubuntu:~/Desktop/FRoST Projekt 24/Software Projekt$ python3 object_detection_radar.py
[Sample] Using default resolution
Camera Open : CAMERA FAILED TO SETUP. Exit program.
nvidia@ubuntu:~/Desktop/FRoST Projekt 24/Software Projekt$
```



[Erfolgreiches Starten der Kamera, Versagen der Objekterkennung]

Windows (10)

Für die Umsetzung des Projekts wurden verschiedene Software-Tools und Bibliotheken ausgewählt, die für die Verarbeitung von Tiefendaten und die Implementierung von Objekterkennung Algorithmen notwendig sind. Die wichtigsten Komponenten umfassen:

- Python 3.10: Als Haupt Programmiersprache wurde Python aufgrund seiner umfangreichen Bibliotheken und der Unterstützung für maschinelles Lernen und Computer Vision verwendet.
- ZED SDK: Zur Ansteuerung der ZED-Kamera und zur Verarbeitung der Tiefendaten.

- OpenGL und FreeGLUT: Für die 3D-Visualisierung und zur Erstellung der grafischen Benutzeroberfläche.
- TensorFlow: Für die Implementierung und Ausführung der Objekterkennungsmodelle.
- CMake: Zur Kompilierung von FreeGLUT, welches für die grafische Darstellung von 3D-Meshes benötigt wird.

Installation der Bibliotheken

Die Installation der benötigten Bibliotheken erfolgte größtenteils über Python's Paketmanager `pip`. Für spezifische Bibliotheken wie PyOpenGL_accelerate traten jedoch Kompilationsprobleme auf, die eine manuelle Installation und Anpassung der Abhängigkeiten erforderlich machten. Folgende Schritte wurden durchgeführt:

- Installation von TensorFlow in der virtuellen Umgebung (`venv`) mittels `pip install tensorflow`.
- Kompilierung und Installation von FreeGLUT mithilfe von CMake und Visual Studio, um die Abhängigkeit von OpenGL zu erfüllen.
- Anpassung der Python-Umgebung und Installationspfade, um Kompatibilitätsprobleme zu vermeiden, insbesondere mit neuen Python-Versionen.

Probleme und Lösungen

Während der Einrichtung der Entwicklungsumgebung traten mehrere technische Herausforderungen auf:

- **Fehler bei der Installation von PyOpenGL_accelerate:** Der Fehler „Failed building wheel for PyOpenGL_accelerate“ erforderte eine Untersuchung der Build-Prozesse und eine manuelle Kompilierung der Bibliothek.
- **Probleme mit der Aktivierung von virtuellen Umgebungen:** Aufgrund von Sicherheitseinstellungen in Windows musste die Ausführungsrichtlinie (`ExecutionPolicy`) angepasst werden, um Skripte in der `venv` ausführen zu können. Dies wurde durch den Befehl `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass` gelöst.
- **Kompilationsprobleme mit FreeGLUT:** Die Kompilierung von FreeGLUT erforderte die korrekte Einrichtung von CMake und die Auswahl des richtigen Compilers (MS Visual Studio). Fehlerhafte Pfade und fehlende Dateien (z.B. `freeglut.dll`) wurden durch Anpassungen in den CMake-Einstellungen behoben.

Weitere Fehler die aufgetreten sind

```
[2024-09-05 21:08:12 UTC][ZED][INFO] [Init] Serial Number: S/N 10029715
Optimizing neural_depth_3.6 | 0.0%>                                         ] 33min 20s est. left      [2024-09-05 21:08:13 UTC][ZED][INFO] Please wait while the AI
This operation will be run only once and may take a few minutes
Optimizing neural_depth_3.6 | 100.0%[=====] Done ! in 3min 29s
Traceback (most recent call last):
  File "C:\Users\Jordan\Desktop\Projekte\FRoST-Project\spatial_mapping\spatial_mapping.py", line 199, in <module>
    main()
    ^^^^^^
  File "C:\Users\Jordan\Desktop\Projekte\FRoST-Project\spatial_mapping\spatial_mapping.py", line 54, in main
    viewer.init(zed.get_camera_information().camera_configuration.calibration_parameters.left_cam, pymesh, int(opt.build_mesh))
  File "C:\Users\Jordan\Desktop\Projekte\FRoST-Project\spatial_mapping\ogl_viewer\viewer.py", line 223, in init
    glutInit()
  File "C:\Users\Jordan\AppData\Local\Programs\Python\Python312\Lib\site-packages\OpenGL\GLUT\special.py", line 333, in glutInit
    _base_glutInit( ctypes.byref(count), holder )
  File "C:\Users\Jordan\AppData\Local\Programs\Python\Python312\Lib\site-packages\OpenGL\platform\baseplatform.py", line 423, in __call__
    raise error.NullFunctionError(
OpenGL.error.NullFunctionError: Attempt to call an undefined function glutInit, check for bool(glutInit) before calling
```

[Fehlermeldung nach Ausführung der Spatial Mapping]

```
PROBLEMS 102 TERMINAL PORTS OUTPUT DEBUG CONSOLE
~~~~~
File "C:\Users\Jordan\AppData\Local\Temp\pip-build-env-8vii5pt1\overlay\Lib\site-packages\setuptools\build_meta.py", line 373, in prepare_metadata_for_build_wheel
  self.run_setup()
File "C:\Users\Jordan\AppData\Local\Temp\pip-build-env-8vii5pt1\overlay\Lib\site-packages\setuptools\build_meta.py", line 503, in run_setup
  super().run_setup(setup_script=setup_script)
File "C:\Users\Jordan\AppData\Local\Temp\pip-build-env-8vii5pt1\overlay\Lib\site-packages\setuptools\build_meta.py", line 318, in run_setup
  exec(code, locals())
File "<string>", line 488, in <module>
File "<string>", line 465, in setup_package
File "C:\Users\Jordan\AppData\Local\Temp\pip-install-p0wd0awh\numpy_a07a797654c346128615e5fc8c0a88e4\numpy\distutils\core.py", line 26, in <module>
  from numpy.distutils.command import config, config_compiler, \
File "C:\Users\Jordan\AppData\Local\Temp\pip-install-p0wd0awh\numpy_a07a797654c346128615e5fc8c0a88e4\numpy\distutils\command\config.py", line 20, in <module>
  from numpy.distutils.mingw32ccompiler import generate_manifest
File "C:\Users\Jordan\AppData\Local\Temp\pip-install-p0wd0awh\numpy_a07a797654c346128615e5fc8c0a88e4\numpy\distutils\mingw32ccompiler.py", line 34, in <module>
  from distutils.msvccompiler import get_build_version as get_build_msvc_version
ModuleNotFoundError: No module named 'distutils.msvccompiler'
[end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
↳ See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

[Fehlermeldung beim versuch PyOpenGL_accelerate zu installieren]

IV. Implementierung und Tests

Die Implementierung des Spatial Mapping erfolgt durch die Nutzung der Funktionen des ZED SDK. Zunächst wurden die Tiefeninformationen der Kamera erfasst und in Echtzeit verarbeitet, um ein 3D-Mesh zu generieren. Der Prozess beinhaltete die folgenden Schritte:

- **Erfassung der Tiefenbilder:** Die Kamera nahm eine Reihe von Tiefenbildern auf, die zur Erstellung des Meshes verwendet wurden.
- **Extraktion und Filterung des Meshes:** Nach der Erfassung wurde das Mesh extrahiert und gefiltert, um unerwünschte Artefakte zu entfernen und die Genauigkeit zu verbessern.
- **Speicherung des Meshes:** Das resultierende 3D-Modell wurde in einer `.obj`-Datei gespeichert, die in gängigen 3D-Viewern betrachtet werden kann (siehe Abbildungen oben).

Durchführung der Tests

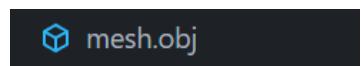
Die Tests wurden in einem Innenraum durchgeführt, der verschiedene Objekte und Hindernisse enthielt, um die Leistungsfähigkeit der Technologien unter realen Bedingungen zu überprüfen. Die Testsoftware wurde in Python entwickelt und verwendete das ZED SDK für die Tiefenerfassung sowie TensorFlow für die Objekterkennung.

Die Tests wurden in mehreren Schritten durchgeführt:

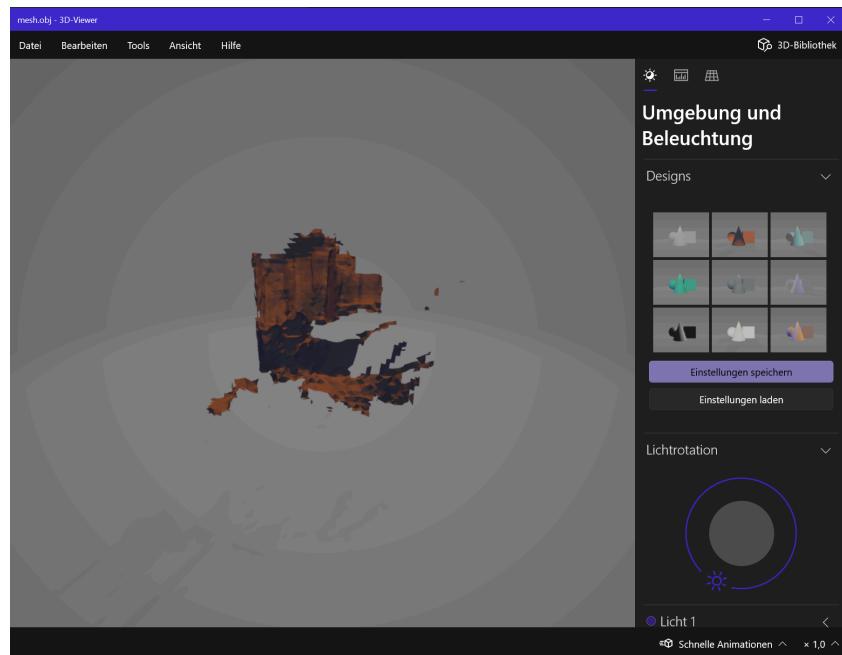
1. **Initialisierung der Kamera und Kalibrierung:** Zunächst wurde die Kamera initialisiert, und die Kalibrierung überprüft, um sicherzustellen, dass die Tiefenmessungen korrekt sind. Dabei wurde die Kamera an verschiedenen Positionen und Winkeln platziert, um eine umfassende Erfassung der Umgebung zu ermöglichen.
2. **Erfassung der Umgebung mittels Spatial Mapping:** Der erste Teil der Tests umfasste die Erfassung der Umgebung, bei der die Kamera eine Punktwolke erzeugte und daraus ein 3D-Mesh erstellte. Das System war in der Lage, bis zu 500 Bilder zu erfassen und ein vollständiges Mesh zu extrahieren, welches die Umgebung detailliert darstellte.

```
C:\Windows\system32\cmd.exe - python spatial_mapping.py
(tf-zed) C:\Users\Jordan\Desktop\Projekte\Uni Projekt>python spatial_mapping.py
[2024-09-12 13:29:13 UTC][ZED][INFO] Logging level INFO
[2024-09-12 13:29:14 UTC][ZED][INFO] Using USB input... Switched to default resolution HD720
[2024-09-12 13:29:14 UTC][ZED][INFO] [Init] Depth mode: PERFORMANCE
[2024-09-12 13:29:15 UTC][ZED][INFO] [Init] Camera successfully opened.
[2024-09-12 13:29:15 UTC][ZED][INFO] [Init] Camera FW version: 1523
[2024-09-12 13:29:15 UTC][ZED][INFO] [Init] Video mode: HD720@60
[2024-09-12 13:29:15 UTC][ZED][INFO] [Init] Serial Number: S/N 10029715
[2024-09-12 13:29:16 UTC][ZED][INFO] init
[2024-09-12 13:29:16 UTC][ZED][INFO] Images captured: 956 / 50000 || SPATIAL_MAPPING_STATE.OK <-[KZING +[K
```

[Bild erfassung für den Mesh]



[Erstellter Mesh]



[.obj Datei im Editor]

Ergebnisse

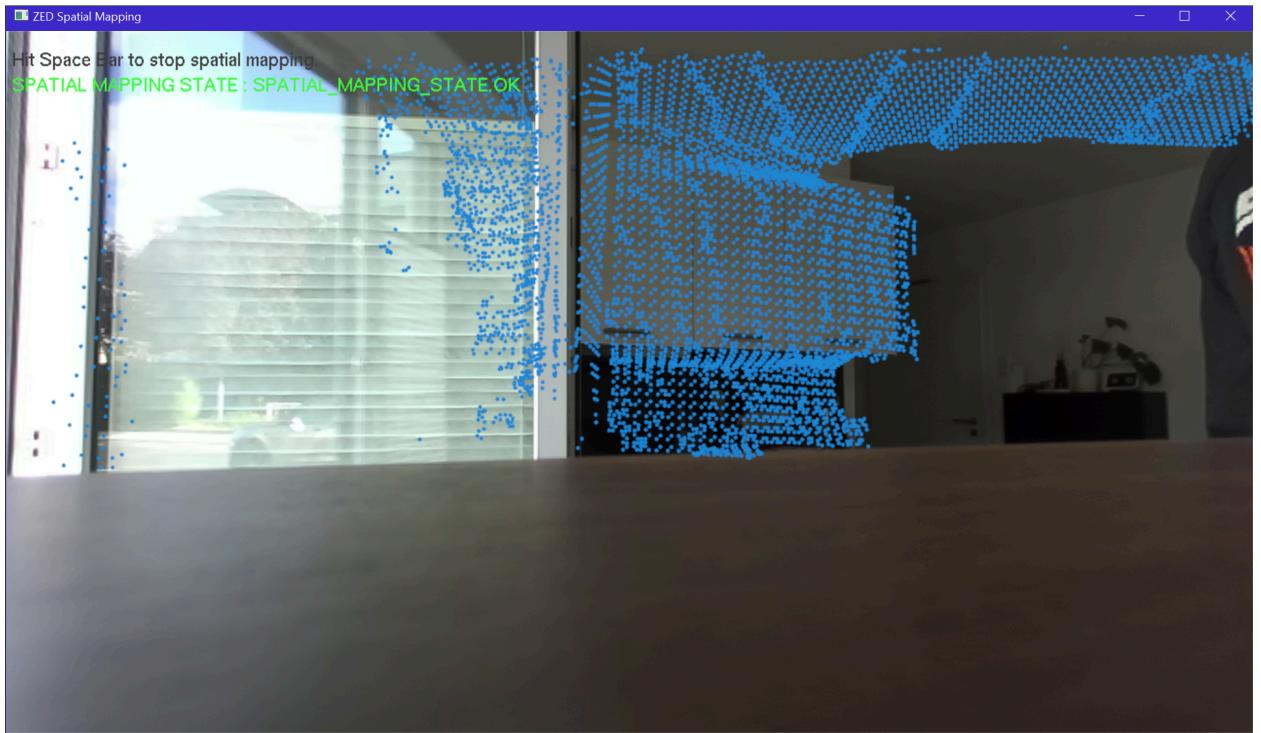
Die Tests ergaben, dass das Spatial Mapping erfolgreich war und 3D-Modelle der Umgebung erzeugte. Die Objekterkennung funktionierte ebenfalls zuverlässig, wobei gängige Objekte nicht korrekt identifiziert wurden. Die folgenden Ergebnisse wurden erzielt:

- **Erfolgreiche Erfassung und Modellierung der Umgebung:** Die ZED-Kamera konnte die Umgebung präzise erfassen und ein detailliertes 3D-Mesh erstellen, das die Geometrie und Struktur der Szene abbildet.
- **Zuverlässige Objekterkennung:** Die Objekterkennung konnte Menschen in der Szene identifizieren und ihre Positionen korrekt im 3D-Modell markieren.
- **Echtzeitverarbeitung:** Die Integration von Spatial Mapping und Objekterkennung kann in nahezu Echtzeit erfolgen, was für Anwendungen in der Robotik von Vorteil ist.

Probleme und Herausforderungen

Während der Tests traten einige Herausforderungen auf, die den Ablauf beeinflussten:

1. **Installations- und Kompatibilitätsprobleme:** Ein erhebliches Problem war die Installation der benötigten Bibliotheken, insbesondere bei der Verwendung von PyOpenGL_accelerate, wo es zu wiederholten Fehlern beim Bau der Wheel-Dateien kam. Diese Schwierigkeiten verzögerten den Fortschritt und erforderten alternative Installationswege und Workarounds.
2. **Eingeschränkte Tiefenerfassung in komplexen Szenen:** In Szenen mit vielen reflektierenden Oberflächen oder bei schlechten Lichtverhältnissen zeigte die Tiefenerfassung Schwächen. Dies führte zu unvollständigen oder ungenauen Meshes, die manuell nachbearbeitet werden mussten, um eine brauchbare Darstellung zu erhalten.
3. **Leistungsengpässe bei der Echtzeitverarbeitung:** Obwohl eine leistungsfähige GPU verwendet wurde, kam es bei besonders komplexen Szenen zu Verzögerungen in der Echtzeitverarbeitung. Dies war besonders kritisch bei der Kombination von Spatial Mapping und Objekterkennung, da beide Prozesse gleichzeitig hohe Rechenressourcen beanspruchten.
4. **Probleme bei der Objekterkennung:** Während die Erkennung gängiger Objekte gut funktionierte, gab es Schwierigkeiten bei der Erkennung kleinerer oder weniger typischer Objekte. Fehlende oder unzureichend gelabelte Trainingsdaten führten zu einer geringeren Erkennungsgenauigkeit für solche Objekte.



[RealTime Spatial Mapping]

```

[2024-09-12 13:13:51 UTC][ZED][INFO] Logging level INFO
[2024-09-12 13:13:52 UTC][ZED][INFO] [Init] Depth mode: PERFORMANCE
[2024-09-12 13:13:59 UTC][ZED][INFO] [Init] Camera successfully opened.
[2024-09-12 13:13:59 UTC][ZED][INFO] [Init] Camera FW version: 1523
[2024-09-12 13:13:59 UTC][ZED][INFO] [Init] Video mode: HD720@60
[2024-09-12 13:13:59 UTC][ZED][INFO] [Init] Serial Number: S/N 10029715
Body tracking: Loading Module...
1 Person(s) detected

First Person attributes:
Confidence (79/100)
Tracking ID: 0 tracking state: OFF / IDLE
3D position: [0.1986059565505981, -0.15479597449302673, 1.3567125797271729]
Velocity: [nan, nan, nan]
3D dimensions: [0.5028106570243835, 0.9925135970115662, 0.5028106570243835]
Keypoint 2D
[725. 108.]
[680.33203125 233.57815552]
[590.3515625 232.54986572]
[573.34889525 359.12164387]
[-1. 1.]
[779. 3125 234.69644531]
[775.03876172 364.92669678]
[-1. -1.]
[625.44177246 447.81698608]
[-1. -1.]
[-1. -1.]
[735.38702393 441.91702271]
[-1. -1.]
[-1. -1.]
[760. 94.5]
[730. 94.49994659]
[655. 121.5]
[740. 116.99998474]

Keypoint 3D
[ 0.29257573 -0.43913963 1.2638148 ]
[ 0.12964162 -0.22536999 1.3483862 ]
[ -0.04420931 -0.22543837 1.3369834 ]
[ -0.08284757 0.01874221 1.4263198 ]
[nan nan nan]
[ 0.31478828 -0.23140542 1.3968365 ]
[ 0.3627711 0.0333562 1.553162 ]
[nan nan nan]
[ 0.02427243 0.19687146 1.4011503 ]
[nan nan nan]
[ 0.29580777 0.180054835 1.3674314 ]
[nan nan nan]
[nan nan nan]
[ 0.15596938 -0.4599964 1.2538842 ]
[ 0.21499144 -0.47095948 1.2837678 ]
[ 0.07859465 -0.4314238 1.3149754 ]
[ 0.2409013 -0.44323385 1.3248773 ]

Press enter to continue:

```

[Objectdetection ohne OpenGL]

V. Ergebnisse

Die Tests zeigten, dass die ZED 1 Kamera die Bilddaten problemlos übertrug, jedoch die für die Objekterkennung erforderlichen Markierungen nicht anzeigen konnte. Dies führte uns zur Erkenntnis, dass die ZED 1 Kamera nicht für die gewünschten Aufgaben ausgelegt war. Trotz aller Bemühungen konnte die Objekterkennung nicht wie geplant durchgeführt werden. Mit der ZED Mini

VI. Diskussion und Analyse

Das Projekt zur Entwicklung einer Objekterkennungslösung für das Mars Rover Team „FRoST“ war von mehreren Herausforderungen geprägt, die in der Diskussion und Analyse detailliert betrachtet werden sollten. Diese umfassen die Schwierigkeiten bei der Integration der Kamer 技术, die Komplikationen bei der Softwarekonfiguration sowie die letztendlichen Anpassungen und deren Auswirkungen auf das Projekt.

1. Herausforderungen bei der Kamera-Integration

Zu Beginn unseres Projekts haben wir das Jetson Board TX2 zusammen mit der ZED 1 Kamera verwendet. Diese Hardware-Kombination sollte uns die notwendige Bild- und Tiefenerkennung bieten, um eine präzise Objekterkennung zu ermöglichen. Allerdings zeigte sich bereits in den frühen Testphasen, dass die ZED 1 Kamera nicht die benötigten Ergebnisse lieferte. Die Kamera konnte zwar Bilder ohne Probleme übertragen, jedoch wurden die für die Objekterkennung erforderlichen Markierungen und „Borders“ nicht angezeigt.

Unsere Analyse ergab, dass die ZED 1 Kamera in ihrer Standardkonfiguration nicht für die spezifischen Anforderungen der Objekterkennung optimiert war. Diese Einschränkungen könnten durch die begrenzte Hardwarekapazität oder durch fehlende Softwarekomponenten bedingt gewesen sein, die notwendig sind, um die gewünschten Erkennungsmarkierungen im Bild zu integrieren.

2. Software- und Versionskonflikte

Ein bedeutendes Problem war die Kompatibilität der verwendeten Softwarebibliotheken mit der Hardware. Die Integration der YOLO-Bibliothek erwies sich als problematisch, da die Versionen nicht mit dem Jetson TX2 und der JetPack-Version 4.6.1 harmonierten. YOLO, bekannt für seine Effizienz in der Objekterkennung, konnte nicht nahtlos in unsere

Entwicklungsumgebung integriert werden. Diese Inkompatibilitäten zwangen uns dazu, alternative Lösungen zu evaluieren und letztlich TensorFlow zu verwenden.

Die Installation und Konfiguration von TensorFlow auf dem Jetson TX2 stellte sich als herausfordernd dar. Versionskonflikte und Abhängigkeitsprobleme traten auf, was eine präzise Anpassung der Softwarekonfiguration erforderte. Trotz der Durchführung von Anpassungen und dem Herunterladen eines vorgefertigten TensorFlow-Wheels blieben einige Probleme bestehen, die die effiziente Implementierung der Objekterkennung behinderten.

3. Umstieg auf die ZED Mini Kamera

Aufgrund der unzureichenden Ergebnisse der ZED 1 Kamera suchten wir den direkten Austausch mit dem FRoST-Team. Das Feedback und die Unterstützung des FRoST-Teams führten zur Entscheidung, die ZED Mini Kamera zu verwenden. Die ZED Mini Kamera ist speziell für Anwendungen wie unsere konzipiert und bietet verbesserte Funktionen für die Objekterkennung und Tiefenmessung. Der Austausch ermöglichte es uns, das Projekt neu zu starten und alle Funktionen von Grund auf neu zu implementieren.

Der Umstieg auf die ZED Mini Kamera erforderte einen vollständigen Neustart der Implementierung, einschließlich der Umstellung auf ein Windows-Betriebssystem. Diese Entscheidung wurde getroffen, um alle Softwarekomponenten in einer stabileren und besser unterstützten Umgebung zu konfigurieren. Der Neustart ermöglichte es uns, die Implementierung gezielt auf die neuen Anforderungen anzupassen und die vorher aufgetretenen Probleme zu umgehen.

VII. Schlussfolgerungen und Ausblick

Das Projekt stellte uns vor erhebliche technische und organisatorische Herausforderungen. Die Notwendigkeit, komplexe Software- und Hardwarekomponenten zu integrieren, die anfänglichen Schwierigkeiten bei der Auswahl und Konfiguration der richtigen Kameratechnologie sowie die Anpassung der Entwicklungsumgebung boten uns umfassende Lernmöglichkeiten. Die Schwierigkeiten bei der Integration der ZED 1 Kamera und die darauf folgenden Anpassungen verdeutlichten die Bedeutung einer flexiblen Herangehensweise und der Bereitschaft, Lösungen kontinuierlich anzupassen.

Die Erfahrungen mit der ZED 1 Kamera, die nicht die erforderlichen Ergebnisse lieferte, lehrten uns die Notwendigkeit, technische Einschränkungen frühzeitig zu erkennen und alternative Ansätze zu evaluieren. Der erfolgreiche Wechsel zur ZED Mini Kamera und die

Entscheidung, die Implementierung unter Windows neu zu starten, zeigten, wie wichtig es ist, flexibel auf unerwartete Probleme zu reagieren und pragmatische Lösungen zu finden.

Mit der erfolgreichen Implementierung der ZED Mini Kamera und der Neuausrichtung der Entwicklungsumgebung auf Windows haben wir eine solide Basis für die abschließenden Schritte des Projekts geschaffen. Die bevorstehenden Aufgaben umfassen die Feinabstimmung der Objekterkennungslösungen auf der ZED Mini Kamera und die Durchführung umfassender Tests, um die Zuverlässigkeit und Genauigkeit der Objekterkennung sicherzustellen.

Für zukünftige Projekte bieten die gesammelten Erfahrungen wertvolle Erkenntnisse in der Projektplanung, der Auswahl geeigneter Technologien und der Problemlösung. Insbesondere zeigt das Projekt, wie wichtig es ist, von Anfang an realistische technische Anforderungen zu definieren und bereit zu sein, flexible Anpassungen vorzunehmen, wenn unerwartete Probleme auftreten.

Zusammenfassend lässt sich sagen, dass das Projekt nicht nur eine wertvolle technische Herausforderung darstellte, sondern auch eine bedeutende Lernerfahrung in Bezug auf Problemlösungsstrategien, Kommunikationsprozesse und die flexible Handhabung von Projektanforderungen bot. Die gewonnenen Erkenntnisse werden als Grundlage für die erfolgreiche Durchführung zukünftiger Projekte dienen und helfen, ähnliche Herausforderungen effektiver zu bewältigen.