Fehleranalyse und Best Practices
Extensions und harte Pfade in Startskripten

Teil A: Extension-Konflikte oder fehlende Abhängigkeiten

Problemstellung

Beim Start lädt Kit ein Set an Extensions. Jede Extension kann weitere Extensions und Python-Pakete voraussetzen. Beim Wechsel zwischen Szenen werden häufig unterschiedliche Extension-Sets aktiviert. Kit verwendet eine eigene Python-Runtime, das heißt Pakete aus dem System-Python sind dort nicht automatisch vorhanden.

Typische Symptome

- Kit startet und bricht beim Aktivieren einer Extension ab.
- `No module named <paket>`, `ModuleNotFoundError`, `ImportError: cannot import name <X>`.
- Versionskonflikte, z. B. `ContextualVersionConflict`.

Häufige Ursachen

- Unterschiedliche Extension-Sets zwischen Szenen.
- Fehlende Python-Pakete in der Kit-Runtime.
- Versions-Drift zwischen Paketen.
- Persistente Caches oder Settings reaktivieren alte Extensions.

Fix-Patterns

Pattern A: Minimales Extension-Set Nur benötigte Extensions aktivieren, alles andere deaktivieren.

Pattern B: Einheitliche `requirements.txt` Eine zentrale `requirements.txt` pflegen und alle Pakete mit der Kit-Python installieren.

Pattern C: Preflight-Check Vor dem Start prüfen, ob alle benötigten Pakete in der aktiven Runtime vorhanden sind.

Pattern D: Sauberer Neustart Beim Wechsel der Szene Kit komplett beenden; Caches bei Bedarf löschen.

Pattern E: Versions-Pinning Kritische Pakete strikt pinnen, z. B. `numpy==1.26.4`.

Konkrete Befehle

```
# Installation in der Kit-Runtime (Beispielpfad)
cd ~/Apps/isaac-sim
./kit/python.sh -m pip install --upgrade pip
./kit/python.sh -m pip install -r requirements.txt
./kit/python.sh -m pip install transforms3d==0.4.1
```

Preflight in Python

```
REQUIRED = ["numpy", "transforms3d", "opencv-python"]
import importlib
missing = []
for m in REQUIRED:
    try:
        importlib.import_module(m)
    except Exception:
        missing.append(m)
if missing:
    raise RuntimeError(f"[preflight] missing packages: {missing}")
```

Debug-Playbook

1. Logs unter `~/.cache/ov` und `~/.local/share/ov` lesen.
2. Problem-Extension identifizieren und testweise deaktivieren.
3. In der Kit-Python `import <paket>` prüfen.
4. Paketversionen mit `pip freeze` vergleichen, ggf. pinnen.
5. Caches löschen, sauber neu starten.

Quick-Checklist

- Minimales Extension-Set definiert.
- `requirements.txt` zentral und gepinnt.
- Installation mit `./kit/python.sh -m pip ...`.
- Preflight ohne Missing-Module.
- Nach Szenenwechsel immer Neustart.

Teil B: Harte Pfade in Skripten

Problemstellung

Harte absolute Pfade (`/home/user/...`, `C:\Users\...` oder `omniverse://old-host/...`) koppeln den Start an eine konkrete Ordnerstruktur. Schon kleine Änderungen (anderer Benutzername, anderer Szenenname, umsortierte Verzeichnisse) führen zu Fehlern beim Bootstrap.

Symptom

```
FileNotFoundError: [Errno 2] No such file or directory: '/old/path/pick_oranges/assets
    /bowl.usd'
```

Häufige Ursachen

- Absolute Pfade im Code oder in USD-Dateien.
- Copy-Paste aus alter Szene (z. B. `pick_oranges` Reste).
- OS-spezifische Pfade, hart gesetzte Startflags im Launcher.
- Abhängigkeit vom wechselnden `cwd`.

Fix-Patterns

Pattern A: Projektroot deterministisch ermitteln

```
from pathlib import Path
PROJECT_ROOT = Path(__file__).resolve().parents[2]
ASSETS_DIR = PROJECT_ROOT / "assets"
SCENES_DIR = PROJECT_ROOT / "scenes"
```

Pattern B: Pfade parameterisieren

```
import argparse, os
ap = argparse.ArgumentParser()
ap.add_argument("--scene", default=os.environ.get("SCENE_USD", "scenes/pick_apples/
    main.usd"))
ap.add_argument("--task", default=os.environ.get("TASK_NAME", "pick_apples_v1"))
args = ap.parse_args()
```

Pattern C: OS-neutrales Path-Building

```
from pathlib import Path
BOWL = ASSETS_DIR / "bowl.usd"
if not BOWL.is_file():
    raise FileNotFoundError(f"missing␣asset:␣{BOWL}")
```

Pattern D: Resolver-Funktion mit Checks

```
from pathlib import Path
def resolve_asset(project_root: Path, rel_or_abs: str) -> Path:
    p = Path(rel_or_abs)
    p = p if p.is_absolute() else (project_root / p).resolve()
    if not p.is_file():
        raise FileNotFoundError(str(p))
    return p
```

Pattern E: Startflags übergeben

```
python launch.py --task pick_apples_v1 --scene scenes/pick_apples/main.usd
```

Beispiel-Startskript

```
from pathlib import Path
import argparse, os, sys
PROJECT_ROOT = Path(__file__).resolve().parents[1]
ASSETS = PROJECT_ROOT / "assets"
SCENES = PROJECT_ROOT / "scenes"

ap = argparse.ArgumentParser()
```

```
ap.add_argument("--task", default=os.environ.get("TASK_NAME", "pick_apples_v1"))
ap.add_argument("--scene", default=os.environ.get("SCENE_USD", "scenes/pick_apples/
    main.usd"))
args = ap.parse_args()

scene_abs = (PROJECT_ROOT / args.scene).resolve()
if not scene_abs.is_file():
    raise FileNotFoundError(f"scene␣not␣found:␣{scene_abs}")
print("[ok]␣task:", args.task)
print("[ok]␣scene:", scene_abs)
```

Quick-Checklist

- Alle Pfade projektrelativ aus `PROJECT_ROOT`.
- Task und Scene als Flags oder ENV.
- Existenzcheck zu Startbeginn.
- USD-References aktualisiert.
- Startskripte funktionieren auf Linux und Windows.