Kantonsschule Baden
Informatik IDPA

Portfolioeintrag IDPA

Werteorientierter Schweizer Roboter: Sprachmodul

Projektname: WSR-Voice

Abb. 0: Institutsroboter als Projektmotiv für WSR-Voice.

Autor: Keanu Koelewijn
Praxisbetreuer: Andri Wild
Praktikumsbetrieb: IMVS, FHNW
Abgabe: 15.01.2026
Klasse: I4b

Aufgabenstellung:

Abb. 1: Der reale Roboter im Institut.

In diesem Projekt wird der Institutsroboter des IMVS (FHNW) um ein Sprachmodul erweitert. Der Roboter soll mit Besuchern und Mitarbeitenden im Institut in Echtzeit sprechen und, sofern technisch möglich, einfache Aufgaben ausführen können. Derzeit fehlt eine direkte Interaktionsmöglichkeit zwischen Nutzer und Roboter.

Dafür wird eine Künstliche Intelligenz *(LLM)* eingesetzt. Zusätzlich muss jederzeit nachvollziehbar sein, in welchem Arbeitsschritt sich der Roboter aktuell befindet.

Als Rahmenbedingung ist vorgegeben, ein bestimmtes Echtzeit-KI-Model *(OpenAI Realtime Model)* über eine Schnittstelle für Programm-zu-Programm *(API)* einzubinden und die Ablaufsteuerung als Ablaufplan mit Wenn-Dann-Schritten *(Behaviour Tree)* umzusetzen. Dabei sollen Realtime Model und Behaviour Tree über einen gemeinsamen Datenspeicher *(Blackboard)* Zustands- und Kontextinformationen austauschen. Die Lösung muss mit der bestehenden Architektur zur Robotersteuerung und Kommunikation *(ROS2)* kompatibel sein. Diese umfasst unter anderem Gesichts- und Objekterkennung oder Navigation, damit Kontextinformationen in die Konversation einfliessen und Aufträge ausgeführt werden können.

Ausführlichere Erklärungen zu Fachbegriffen, Abkürzungen und Abbildungen (inkl. vergrössert) sind im Glossar.

Ziele:

1. **ROS2-Kontext ins LLM integrieren**
   Ich integriere ROS2-Daten wie Gesichts- und Objekterkennung sowie Navigation strukturiert als Kontext in das LLM. Mindestens drei ROS2-Quellen (z.B. Gesichtserkennung, Objekterkennung, Navigation) werden in einer Demo nachweisbar genutzt.

2. **Behaviour Trees verstanden**
   Ich implementiere einen Behaviour Tree für Dialog- und Aufgabenabläufe und kann die Zustände RUNNING, SUCCESS und FAILURE erklären und begründe zentrale Designentscheidungen wie Selektor oder Sequenz. Der Nachweis erfolgt durch eine Demo mit sichtbaren Statuswechseln.

3. **LLM auf Robotikdomäne begrenzen**
   Ich gestalte den Prompt so, dass sich das System wie ein Empfangsroboter verhält, nur die vorgesehenen Robotikfunktionen und hinterlegten Daten verwendet und andere Themen oder Aktionen als nicht unterstützt zurückweist. Der Nachweis erfolgt durch den dokumentierten Prompt und eine Erläuterung.

4. **LLM und Behaviour Tree über Blackboard synchronisieren**
   Ich implementiere eine Blackboard-Struktur, über die LLM und Behaviour Tree Zustände konsistent austauschen. Die Zustandsänderungen im Blackboard werden angezeigt und sind nachvollziehbar.

5. **Robotik-Aktionen kontrolliert ausführen**
   Ich übersetze mithilfe des LLMs, Nutzerabsichten in sichere Aktionen, die ausschliesslich durch den Behaviour Tree ausgeführt werden. Der Nachweis erfolgt durch Architekturdiagramm, Validierungslogik im Code und Sequenzdiagramm.

Produkt:

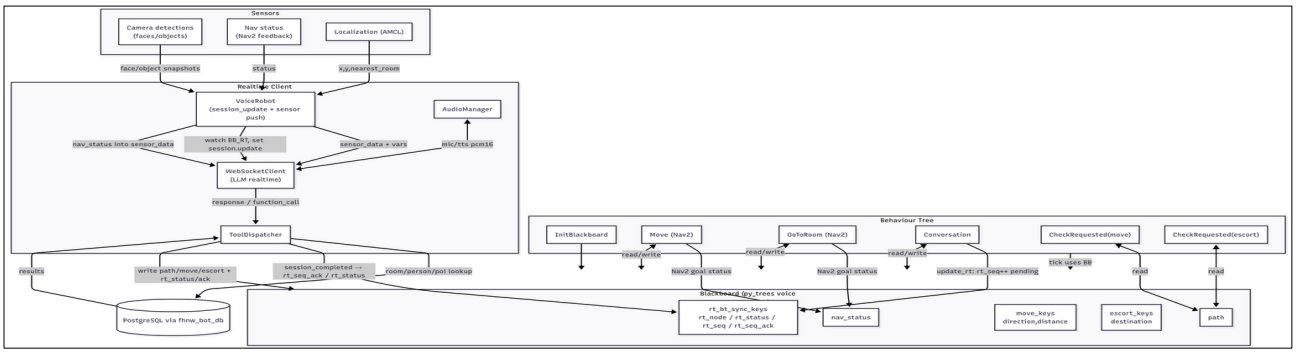

Abb. 2: Systemübersicht und Datenfluss zwischen ROS2, Realtime Model, Tools, Blackboard und Robotik-Aktionen.

Als Produkt wurde ein sprachbasierter Assistent für einen Institutsroboter im ROS2-Umfeld umgesetzt, der eine vollständige Interaktionskette von der Begrüssung bis zur Ausführung robotischer Aktionen abbildet. Nach dem Start kann eine Person direkt sprechen. Der Roboter reagiert sofort und führt durch eine klar strukturierte Abfolge aus Begrüssung, Namensabfrage und Klärung der Absicht *(Intent)*. Daraus leitet er konkrete Handlungen ab, etwa das Begleiten zu Räumen, das Geben von Auskünften oder eine Bewegung relativ zur aktuellen Position *(relative Bewegung)*. Fehlen Angaben oder ist eine Anfrage mehrdeutig, stellt der Assistent gezielte Rückfragen und bestätigt kritische Schritte, sodass der Nutzer jederzeit versteht, was als Nächstes passiert (Abb. 2).

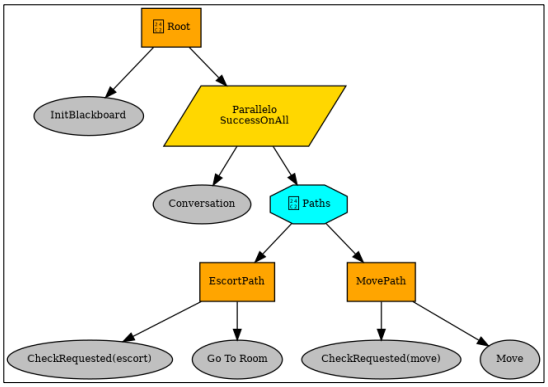

Abb. 3: Aktueller Behaviour Tree für Dialog und Aktionen.

Im Inneren ist die Dialogführung durch eine deterministische Ablaufstruktur organisiert, die festlegt, welcher Arbeitsschritt aktiv ist. Dieser Ablaufplan ist als Behaviour Tree modelliert und bildet die Aktionslogik als nachvollziehbare Struktur ab (Abb. 3). Behaviour Tree und Sprachkomponente tauschen Zustände über ein gemeinsames Blackboard aus, sodass Ergebnisse sauber übernommen werden und der Ablauf zuverlässig zum nächsten Schritt wechseln kann. Dadurch bleiben beide Komponenten synchron, Wiederholungen werden vermieden und Fehler konsistent behandelt (Abb. 2).

Die Sprachkommunikation erfolgt über eine persistente WebSocket Verbindung. Das Audio wird gestreamt und Antworten werden fortlaufend generiert, wodurch sich die Interaktion in Echtzeit anfühlt. Für jeden Arbeitsschritt wird ein kompakter Kontext aufgebaut. Dieser entählt Ziele, Bewegungsparameter, Navigationsstatus sowie relevante aktuelle Sensorinformationen. Modellantworten können Tool-Calls auslösen. Diese validieren Eingaben, erlauben nur freigegebene Operationen und schreiben strukturierte Resultate ins Blackboard zurück, sodass Dialog- und Aktionslogik konsistent bleiben. Der Ablauf vom Tool-Call bis zum Rückschreiben ins Blackboard ist in Abb. 4 dargestellt.

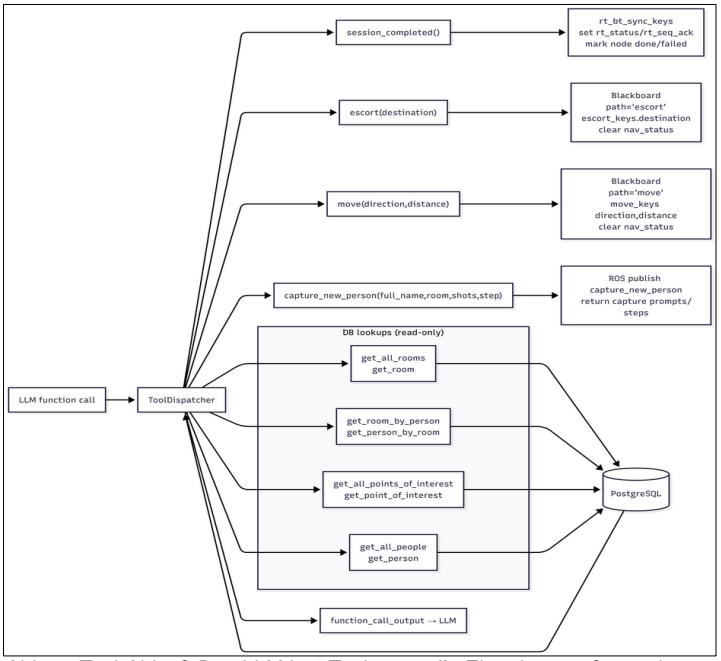

Abb. 4: Tool-Ablauf. Das LLM löst Tools aus, die Eingaben prüfen und Ergebnisse strukturiert ins Blackboard zurückschreiben.

Die Einbettung in ROS2 ermöglicht unterschiedliche Betriebsarten. Aktuell werden Behaviour Tree und Realtime-Komponente gemeinsam in einem ROS2-Node gestartet und ausgeführt. Die Konfiguration wird über Umgebungsvariablen geladen, damit Simulation und reale Hardware denselben Ablauf nutzen können (inklusive Audio-Ein-/Ausgabe, API-Zugang und Datenbankverbindung). Statuswechsel werden geloggt und im Viewer live visualisiert, sodass jederzeit ersichtlich ist, welcher Schritt aktiv ist und weshalb Entscheidungen getroffen oder Aktionen gestartet, wiederholt oder beendet wurden.

Demonstration:

Die Demonstration ist in drei separate Videos unterteilt:

1. **POV-Video:** zeigt eine reale Interaktion zwischen dem Roboter und mir. Der Roboter identifiziert die Person und reagiert mit Sprache und passenden Aktionen.
2. **CLI-Video:** gibt Einblick in das Terminalgeschehen während der Interaktion. Hier lassen sich die laufenden Sensordaten, interne Statusmeldungen sowie Funktionsaufrufe des Sprachmodells (z. B. Datenbankabfragen oder Navigationsanfragen) mitverfolgen.
3. **Behaviour Tree Viewer:** Veranschaulicht, in welchem Schritt sich der Roboter gerade befindet und wie der Verhaltensbaum auf Änderungen im Blackboard reagiert.

Verifizierung der Ziele:

1. **ROS2-Kontext ins LLM integrieren**

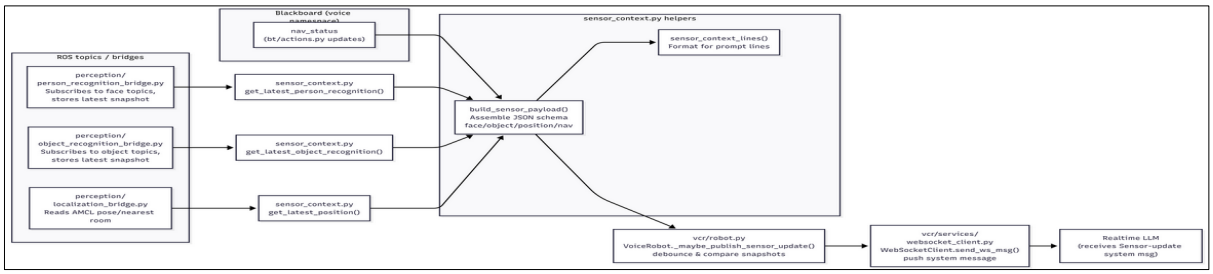

Abb. 5: Aufbau und Weitergabe von Sensoren an das Realtime Model. (ROS2, Sensor-Context, LLM)

Damit das LLM weiss, was rund um den Roboter passiert, binde ich Daten aus drei ROS2 Quellen ein. Dazu verwende ich drei Bridges für Gesichtserkennung, Objekterkennung und Lokalisierung. Die Bridges lesen die jeweiligen ROS2 Topics und formatieren die Daten als strukturiertes JSON. Dieses JSON wird als Kontext an das LLM gesendet und im Dialog verwendet. Datenfluss und Aufbau sind in Abb. 5 dargestellt.

Ein Gif zu den tatsächlich gesendeten Sensor daten ist hier.

2. **Behaviour Trees verstanden**

Ein Behaviour Tree Knoten liefert RUNNING, SUCCESS oder FAILURE. Diese Status bestimmen den weiteren Ablauf. In Abb. 6 sind „Check Known Person“ und „Greet Known Person“ erfolgreich, weshalb „KnownSequence“ SUCCESS zurückgibt. Der „PersonSelector“ wählt einen Pfad, sobald ein Kind SUCCESS liefert. Gibt „KnownSequence“ FAILURE zurück, wird im nächsten Tick auf „UnknownSequence“ gewechselt, wie Abb. 7 zeigt. Die Statuswechsel sind im Viewer sichtbar und machen den Ablauf nachvollziehbar.

Ein Gif zu den aktiven wechsel des Behaviour Trees ist hier.

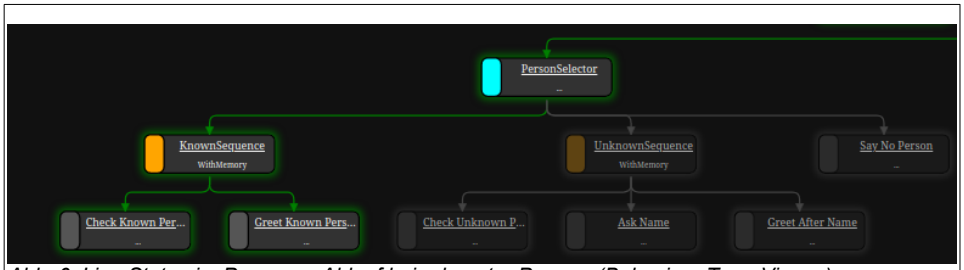

Abb. 6: Live-Status im Personen-Ablauf bei erkannter Person. (Behaviour Tree, Viewer)

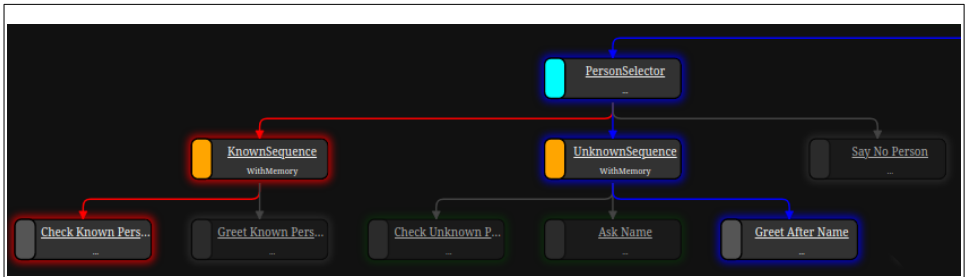

Abb. 7: Live-Status im Personen-Ablauf bei nicht erkannter Person. (Behaviour Tree, Viewer)

Im Projekt gibt es zwei Behaviour Tree Designs mit unterschiedlichen Zielen. Die ältere Version gibt der Realtime Komponente die Schritte strikt vor, dies ist für Demos stabil und gut kontrollierbar. In der neueren Version erhält das LLM mehr Entscheidungsfreiheit, wodurch der Ablauf flexibler wird. Die ältere Version ist in Abb. 9 dargestellt, die neuere in Abb. 8.

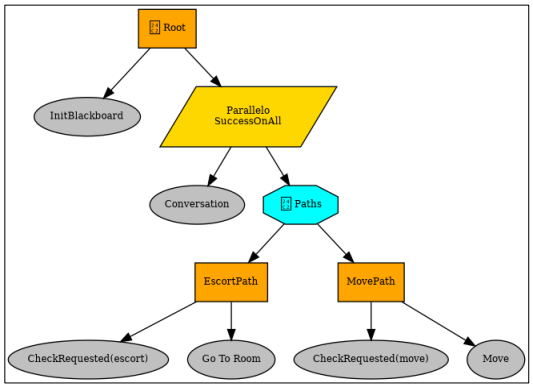

Abb. 8: Ausschnitt aus Abb. 2

Das Design nutzt Selektor, Sequenz und Parallelknoten, um Dialog und Aktionen zu kombinieren. In Abb. 8 startet der Parallelknoten die Konversation und überwacht gleichzeitig den Teilbaum „Paths“. Während das Gespräch läuft, kann das System auf „Escort“ oder „Move“ reagieren und den passenden Pfad ausführen. Der Parallelknoten wird erfolgreich, wenn die Konversation abgeschlossen ist und ein Pfad SUCCESS liefert.

Im Teilbaum „Paths“ wählt ein Selektor zwischen „EscortPath“ und „MovePath“. Beide Pfade sind als Sequenz aufgebaut, in der die Schritte nacheinander ausgeführt werden. Ein Beispiel dafür wäre „CheckDirection Set“, „CheckDistance Set“, „Move“ und „Clear Move“ in „MovePath“ (Abb. 9).

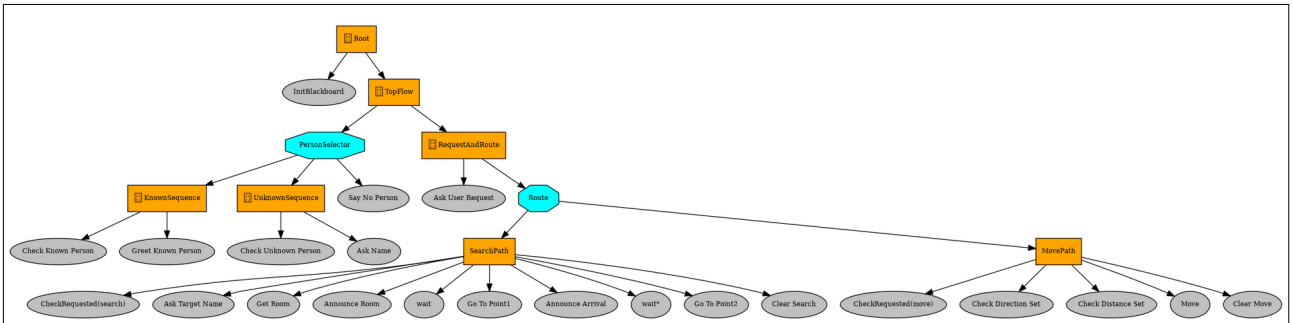

Abb. 9: Alter Behaviour Tree für Dialog und Aktionen.

3. LLM auf Robotikdomäne begrenzen

Der System Prompt definiert Rolle und Domäne als Empfangsroboter und beschränkt das Modell auf freigegebene Roboterfunktionen und hinterlegte Daten. Anfragen ausserhalb der Domäne werden zurückgewiesen.

1) Rolle Das LLM agiert als Empfangsroboter mit Begrüssung, Namensabfrage und Intent Erkennung.
2) Whitelist Das Modell darf ausschliesslich vordefinierte und implementierte Tools verwenden.
3) Stil, Sprache und Ton sind festgelegt. Zum Beispiel durchgehend Deutsch und freundlich formell.
4) Für die Kontrollierte Ausführung liefert das LLM Intent und Parameter. Der Behaviour Tree validiert die Eingaben und führt Robotikaktionen aus.
5) Definierte Regeln und keine Spekulationen. Bei Unklarheiten gezielt nachfragen. Ausserhalb der Domäne als nicht unterstützt zurückweisen.

Die Struktur folgt den Empfehlungen für Realtime Prompting mit den Abschnitten Rolle, Tools und Regeln sowie Sicherheit und ist im Prompt dokumentiert.

```
MAIN_INSTRUCTIONS_SUMMARY: list[str] = [
    "## Rolle & Ziel",
    "- Empfangsroboter am IMVS, erkennt Nutzerabsichten und löst passende Robotikaktionen über Tools aus.",

    "## Stil & Sprache",
    "- Kurz, freundlich, Schweizerdeutsch, keine Spekulation, nur echte Daten und Funktionen.",

    "## Turn-Taking",
    "- Nutzer darf jederzeit unterbrechen",

    "## Werkzeuge (Whitelist)",
    "- Nur freigegebene Tools verwenden; keine anderen erwähnen oder erfinden.",

    "## Tool-Entscheidung",
    "- Tools selbstständig einsetzen; {session_completed} nur bei klar beendetem Gespräch.",

    "## Multi-Step Aktionen",
    "- Mehrere Anweisungen in gegebener Reihenfolge ausführen (z. B. zwei Bewegungen hintereinander).",

    "## Tool-Kurzbeschreibungen",
    "- {escort}, {move}, Raum-/Person-/POI-Abfragen und {suppress_person_audio} mit definierten Parametern.",

    "## Gesprächsablauf",
    "- Begrüssen, Anliegen klären, fehlende Infos präzise nachfragen, passende Tools aufrufen.",

    "## Unmögliche / eingeschränkte Aktionen",
    "- Keine externen Infos, keine Termine, keine Telefonate, keine Objekte greifen, keine Bewegungen ohne klare
Anweisung.",

    "## Laufzeitkontext",
    "- Sensordaten (Gesichter, Position, Nav-Status) als Kontext; ersetzen keine Tool-Daten.",

    "## Sicherheit",
    "- Kein Raten; bei unklaren Situationen einmal nachfragen oder höflich beenden.",

    "## Kontext Personen",
    "- Nur echte Personen aus DB/{coworkers}; Namensabgleich bei Unsicherheit.",

    "## Kontext Räume & POIs",
    "- Bekannte Räume und POIs über {room_list} und {poi_list}.",

    "## Capture New Person",
    "- Geführter Ablauf zur Gesichtserfassung (Step 0–5) mit Namen, Position und klaren Anweisungen.",
]
```

4. **LLM und Behaviour Tree über Blackboard synchronisieren**
   Damit Realtime Teil und Behaviour Tree synchron bleiben, nutze ich im Blackboard ein gemeinsames Wörterbuch namens rt_bt_sync_keys. Es enthält den aktiven Schritt, Status, Resultat und zwei Zähler zur Zuordnung von Anfragen.
   - rt_node für den aktuell aktiven Schritt
   - rt_status für den Zustand des Schritts
   - rt_result für Ergebnis oder Rückmeldung
   - rt_seq als fortlaufende Nummer pro Dialogschritt
   - rt_seq_ack als Bestätigung, dass die Nummer verarbeitet wurde

   So ist klar, welcher Schritt aktiv ist und ob eine Antwort zum aktuellen Schritt gehört. rt_seq und rt_seq_ack verhindern doppelte oder verlorene Schritte, auch wenn Ereignisse schnell hintereinander auftreten.
   Ein Gif zu den Veränderungen im Blackboard ist hier.

5. **Robotik-Aktionen kontrolliert ausführen**
   Robotik Aktionen werden nur vom Behaviour Tree ausgeführt. Das LLM liefert die Absicht, zum Beispiel „escort“, sowie Parameter wie den Raumnamen. Der Behaviour Tree prüft, ob eine Aktion gesetzt ist, liest das Ziel aus dem Blackboard und validiert beim Schritt „Go To Room“ zusätzlich, ob der Raum in der Datenbank existiert, bevor die Navigation gestartet wird. So bleibt die Ausführung kontrolliert und sicher, während das LLM nur einfache Eingaben liefern muss.

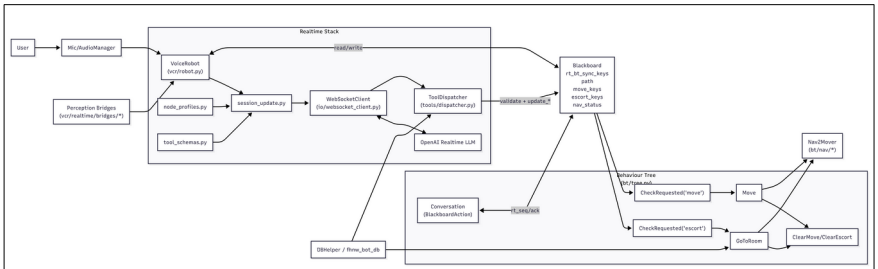

Abb. 10: Architekturdiagramm zur kontrollierten Ausführung von Robotik-Aktionen.

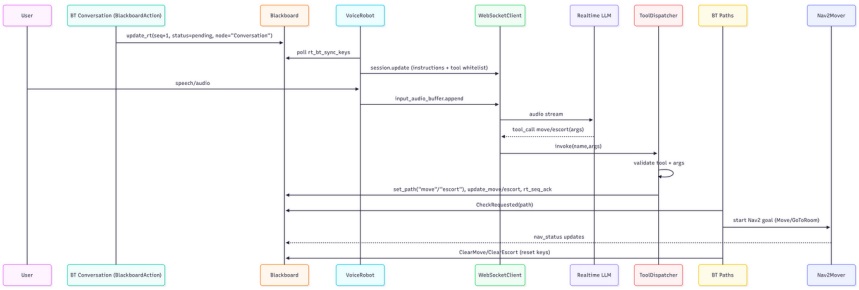

Abb. 11: Sequenz zur Validierung und Ausführung von Robotik-Aktionen.

```
# tool_schemas.py + session_update.py (vereinfacht)
TOOL_SCHEMAS = [{"name": "move"}, {"name": "escort"}, ...]
ALLOWED_TOOL_NAMES = tuple(s["name"] for s in TOOL_SCHEMAS)

def build_session_update_for_node(policy):
    allowed = policy.get("allowed_tools", list(ALLOWED_TOOL_NAMES))
    tools = [t for t in TOOL_SCHEMAS if t["name"] in set(allowed)]
    return {"session": {"tools": tools, "tool_choice": "auto"}}

# dispatcher.py (vereinfacht)
def invoke(name, args):
    if name not in self._tools:
        return {"ok": False, "error": f"unknown tool: {name}"}
    return self._tools[name](args)

def _move_intent(args):
    direction = parse_direction(args.get("direction"))  # left/right/forward/backward
    distance = parse_distance(args.get("distance"))    # number or "1.2m"
    if direction not in {"left","right","forward","backward"}:
        set_path(bb, None); raise ValueError("invalid direction")
    if distance is None or distance <= 0:
        set_path(bb, None); raise ValueError("invalid distance")
    set_path(bb, "move")
    update_move(bb, direction=direction, distance=distance)

# conditions.py + tree.py (vereinfacht)
if get_path(bb) == "move":
    Move()  # nur der BT startet Nav2-Ziele
```

Der Ausschnitt zeigt, wie Nutzerabsichten in geprüfte Robotikaktionen übersetzt werden. Beim Aufbau der Sitzung wird aus den verfügbaren Tools eine Zulassungsliste erstellt, die nur erlaubte Aktionen enthält. Das Realtime Modell kann dadurch nur diese Aktionen anfragen. Bei einem Tool Call prüft der Dispatcher zuerst, ob der Name erlaubt ist. Unbekannte oder gesperrte Tools werden abgewiesen. Für gültige Tools werden die Parameter validiert. Beim move Intent sind nur die vier Richtungen left, right, forward und backward erlaubt und die Distanz muss positiv sein. Bei Fehlern wird die Anfrage verworfen und der Pfad im Blackboard zurückgesetzt. Erst nach erfolgreicher Prüfung setzt das System den Pfad auf move oder escort und speichert die validierten Werte. Der Behaviour Tree startet die Navigation nur dann, wenn dieser Pfad gesetzt ist. Damit kann das LLM keine Fahrbefehle direkt auslösen, sondern liefert nur Absicht und Parameter.

Reflexion Ablauf:

Zu beginn gab es einige Schwierigkeiten. Ich musste mich zuerst in ROS2 und die ganze Roboter-Umgebung einarbeiten, und das war am Anfang einfach sehr viel auf einmal. In der ersten Woche habe ich vor allem Tutorials und Dokumentationen angeschaut, um die Grundlagen zu verstehen, zum Beispiel Koordinatentransformationen *(tf2)*. Damit ich es wirklich verstehe, habe ich ein kleines Mini-Projekt gemacht und Dinge in der Visualisierung *(RViz)* und der Simulation *(Gazebo)* ausprobiert.
Trotzdem war ich oft überfordert, weil neben ROS2 auch noch Realtime-Audio dazukam. Ich musste verstehen, wie Audiodaten gesendet, gepuffert und verarbeitet werden, und gleichzeitig die API-Integration hinbekommen. Parallel dazu habe ich sehr viel recherchiert, vor allem zu Behaviour Trees, zum OpenAI Realtime Model, zu LLMs und zu Prompting-Strategien, damit das System stabiler und verlässlicher reagiert. Um überhaupt ein Gefühl dafür zu bekommen, habe ich zuerst mit einfachen Function Calls experimentiert, bevor ich alles im System verbunden habe. Die ersten ein bis zwei Monate ging es darum nur langsam vorwärts. Es funktionierte immer so halb, es gab viele kleine Fehler, und das LLM machte nicht immer das, was ich wollte.

Ein wichtiger Schritt war, dass ich den Behaviour Tree zuerst stark verkleinert habe, also nur das Nötigste gemacht habe, bis das stabil lief. Danach konnte ich ihn schrittweise erweitern. Die Synchronisation zwischen Realtime Model und Behaviour Tree war eine grosse Hürde. Ich brauchte mehrere Ansätze, bis es zuverlässig lief. Das Ganze war oft mehr „ausprobieren und testen“ als einfach nur umsetzen, aber genau das hat mir auch gefallen.

Sobald eine stabile Version verfügbar war, begann ich mit Tests auf dem realen Roboter. Dabei zeigte sich schnell, dass die reale Umgebung neue Herausforderungen mit sich bringt, die in der Simulation nicht auftreten. Ich musste lernen, Probleme gezielt zu analysieren und die richtigen Stellen im System anzupassen. Dazu gehörten zum Beispiel das Dockerisieren der Umgebung und das Verständnis, wofür die einzelnen Container zuständig sind. Zusätzlich besteht der Roboter aus zwei Rechnern. Dadurch waren bestimmte Schritte nur auf dem einen System möglich und andere nur auf dem zweiten. Am Anfang wurde ich dabei eng von meinem Praxisbetreuer begleitet, bis ich diese Abläufe selbstständig umsetzen konnte. Als der Roboter dann wirklich zu einem Raum fahren konnte und die Tests funktionierten, war das ein richtig gutes Gefühl.

Was mache ich das nächste Mal besser:

Nächstes Mal starte ich früher mit einem MVP, das schnell stabil läuft. Ich plane die Einarbeitung mehr in Etappen, statt zu viele neue Themen gleichzeitig zu machen. Und ich gehe früher auf den echten Roboter, damit Probleme nicht erst am Schluss sichtbar werden.

Reflexion Erkenntnisse:

Aus dem Projekt nehme ich vor allem mit, wie viele Bausteine sauber zusammenspielen müssen, damit ein sprachgesteuerter Roboter im echten Umfeld zuverlässig wirkt. ROS2 war dabei die Grundlage für Navigation und Sensorik, während die Dialoglogik über einen Behaviour Tree stabil und nachvollziehbar blieb. Die grösste Erkenntnis war, dass ein LLM-Sprachmodell zwar sehr gut für Sprache ist, aber ohne klare Regeln und Grenzen schnell unzuverlässig wird. Darum waren klar formulierte Prompts an das Modell, eine Liste erlaubter Funktionen *(Tool-Whitelist)* und eine kontrollierte Ausführung über vordefinierte Funktionen entscheidend, damit das Modell nicht „frei" handelt, sondern nur den Intent des Nutzers liefert und das System immer nach klaren Regeln reagiert *(deterministisch)*. Auf der technischen Seite habe ich gelernt, dass Live-Audio viel mehr ist als „Audio rein, Audio raus", denn automatische Erkennung von Spracheingang und Pausen *(VAD)*, Pufferung und stabile Streaming-Ausgabe sind nötig, damit es ohne Störungen und ohne spürbare Verzögerung *(Latenz)* funktioniert. Dazu kommt die Kontrolle von Eingabelänge und Kontext *(Token- und Kontext-Management)*, also dass Kontext nicht einfach immer grösser werden darf, sondern gezielt verdichtet, zwischengespeichert *(Caching)* und begrenzt werden muss, damit das System schnell bleibt und die Kosten kontrollierbar sind. Genau daraus ist auch die Idee entstanden, eine Aktivierungswort-Erkennung einzubauen *(Hotword-Modell)*, damit nicht dauerhaft Audio an OpenAI gestreamt werden muss, sondern der Dialog erst nach einem Aktivierungswort startet. Dabei lerne ich aktuell viel über das Trainieren eines eigenen KI-Modells *(Machine Learning)* und merke, wie entscheidend gute Daten sind, also saubere Zuordnungen *(Labels)*, genügend Beispiele, ähnliche Aufnahmebedingungen und eine klare Trennung zwischen Trainings- und Testdaten. Ich würde sagen eines der wichtigsten Erkenntnisse ist, dass Robustheit weniger durch einzelne Features entsteht, sondern durch klare Schnittstellen, saubere Zustände und gutes Debugging, das in kleinen stabilen Schritten erweitert wird.

Zusätzliche (weniger ernstzunehmende) Erkenntnisse:

- Linux > Windows
- Vim macht mir Angst.
- Docker ist cool, aber manchmal wünschte ich mir, dass „it works on my computer" genügen würde.
- Speech-to-Speech-Modelle sind sehr teuer!
- ROS 2 ist Black Magic.
- Arbeiten ohne Kaffee = unmöglich.
- Dieser Roboter wird uns noch alle in den Wahnsinn treiben.

Glossar:

Abkürzungen:

Begriff	Definition	Kontext im Projekt
IMVS	Das Institut für Mobile und Verteilte Systeme (IMVS) ist ein multidisziplinäres Institut der Hochschule für Informatik FHNW.	Das Projekt wird am IMVS umgesetzt und direkt in die dortige Roboter- und Softwareumgebung integriert.
FHNW	Die **Fachhochschule Nordwestschweiz** (FHNW) ist eine Fachhochschule in der Schweiz und ist in der Lehre, Forschung, Weiterbildung und Dienstleistung tätig.	Das IMVS ist ein Institut der Fachhochschule Nordwestschweiz.
LLM	Ein **Large Language Model (LLM)** ist ein grosses Sprachmodell, das durch Lernen aus umfangreichen Textdaten die Fähigkeit zur automatischen Texterzeugung besitzt. Es handelt sich um ein computerlinguistisches Wahrscheinlichkeitsmodell, das komplexe Zusammenhänge in Texten versteht und neuen Text generiert.	Das LLM verarbeitet die Spracheingaben, formuliert passende Antworten und liefert strukturierte Absichten, aus denen der Roboter die nächsten Schritte ableitet.
API	Eine **Programmierschnittstelle (API)** ist ein Teil einer Software, der anderen Programmen vorgibt, wie sie mit dieser Software kommunizieren und Funktionen nutzen können. Sie stellt Funktionen oder Daten eines Systems so zur Verfügung, dass andere Programme sie ansprechen und verwenden können	Über die API verbindet sich das System mit externen Diensten wie dem Sprachmodell und ruft definierte Funktionen auf, um Ergebnisse kontrolliert zurückzubekommen.
ROS2	**ROS 2 (Robot Operating System 2)** ist ein Open-Source-Framework für Robotikanwendungen. Es bietet Bibliotheken und Tools zur Entwicklung und Ausführung von Roboterprogrammen. ROS 2 ist dabei nicht ein echtes Betriebssystem, sondern ein flexibles Software-Framework für Robotikprojekte	Der Roboter läuft in einer ROS2-Umgebung. Dadurch kann er über ROS2-Komponenten mit Navigation, Sensorik und weiteren Modulen kommunizieren und diese im Dialog nutzen.
VAD	**Voice Activity Detection (VAD)** ist ein Verfahren zur Sprachaktivierungserkennung. Es überprüft ein Audiosignal und erkennt, ob gerade gesprochen wird. VAD erkennt das Vorhandensein von Sprache und markiert den Beginn sowie das Ende der Sprachaktivität	VAD erkennt, ob gerade gesprochen wird, und steuert damit, wann Audio verarbeitet oder weitergeleitet wird. So werden unnötige Audioübertragungen vermieden und der Dialog wirkt reaktionsschnell.

Fachbegriffe:

Begriff	Definition	Kontext im Projekt
OpenAI Realtime Model	Das **OpenAI Realtime Model** (auch *gpt-realtime* genannt) ist ein neues multimodales Sprachmodell von OpenAI, das speziell für Echtzeitanwendungen entwickelt wurde. Es kann simultan Audio- und Texteingaben verarbeiten und ermöglicht Sprachinteraktionen mit sehr geringer Verzögerung.	Das Realtime Model ermöglicht dem Institutsroboter, Sprache in Echtzeit zu verstehen und ohne spürbare Verzögerung zu antworten.
Behaviour Tree	Ein **Behaviour Tree** (Verhaltensbaum) ist eine hierarchische Ablaufsteuerung in Form eines Baumdiagramms. Er dient zur Strukturierung komplexer Abläufe und funktioniert wie ein erweiterter Zustandsautomat. Solche Bäume werden z.B. in der Spieleprogrammierung und Robotik genutzt, um das Verhalten übersichtlich zu regeln.	Der Behaviour Tree steuert den Dialog und die Aktionen des Roboters als klarer Ablaufplan, damit jeder Schritt nachvollziehbar und kontrolliert abläuft.
Blackboard	Ein **Blackboard** ist ein gemeinsamer „Schreibtafel“-Bereich in einem KI-System. Mehrere spezialisierte Module (Wissensquellen) arbeiten dort zusammen: Jedes Modul trägt Teilergebnisse zur Problemlösung auf dieses Blackboard bei. So entsteht in mehreren Schritten eine Gesamtlösung.	Über das Blackboard teilen Robotiklogik und Sprachkomponente Zustände, Ziele und Ergebnisse, damit der Roboter jederzeit weiss, was gerade läuft und was als Nächstes kommt.
Gesichts- und Objekterkennung	**Gesichtserkennung** und **Objekterkennung** sind Bildverarbeitungsverfahren. Dabei werden in Fotos oder Videos automatisch menschliche Gesichter bzw. beliebige Gegenstände erkannt und identifiziert. Es werden Algorithmen eingesetzt, die Bildmerkmale analysieren und z.B. Klassen von Objekten oder Menschen zuordnen.	Erkannte Personen und Objekte liefern Kontext für den Dialog, zum Beispiel wer gerade vor dem Roboter steht oder was in der Umgebung sichtbar ist.
Intent	Ein **Intent** (Nutzerabsicht) bezeichnet bei Sprachassistenten oder Chatbots das Ziel bzw. die beabsichtigte Aktion hinter einer Nutzeranfrage. Er beschreibt, was der Nutzer erreichen möchte (z.B. „Wecker stellen“ oder „Wetter abfragen“).	Aus einer Aussage des Nutzers wird ein Intent abgeleitet, damit der Roboter entscheiden kann, ob er informieren, nachfragen oder eine Aktion starten soll.
Relative Bewegung	**Relative Bewegung** bedeutet, dass sich ein Roboter (oder Objekt) relativ zu seiner aktuellen Position bewegt. Das heisst, die neue Position wird auf Basis der bisherigen Koordinaten berechnet, nicht anhand eines festen Bezugssystems. Mit Relativbewegungen verschiebt sich die Position des Roboters relativ zu seiner momentanen Lage.	Für einfache Befehle wie „ein Stück vor“ oder „nach links“ nutzt der Roboter relative Bewegungen, also Richtungen und Distanzen bezogen auf seine aktuelle Position.
Deterministische Ablaufstruktur	Eine **deterministische Ablaufstruktur** ist ein Ablauf, bei dem bei den gleichen Eingaben immer derselbe Prozess und dasselbe Ergebnis zustande kommen. Jeder Schritt ist fest vorgegeben, es gibt keine zufälligen oder unbestimmten Entscheidungen. Bei den gleichen Startbedingungen erhält man also stets dasselbe Ergebnis.	Der Ablauf ist bewusst deterministisch aufgebaut, damit der Roboter bei gleichen Eingaben gleich reagiert und Fehler oder Abbrüche reproduzierbar sind.
Tools	**Tools** sind in diesem Kontext zusätzliche Funktionen oder Dienste, die ein Sprachmodell nutzen kann. Dazu zählen etwa Websuchen, Dateiabrufe oder andere spezialisierte Dienste. Das Modell kann diese Tools aufrufen, um Informationen zu erhalten oder Aufgaben	Tools sind die kontrollierten Systemfunktionen, über die der Roboter Informationen abfragt oder Aktionen ausführt, ohne dass das Sprachmodell direkt

	zu lösen (z.B. Wetter abfragen, Dateien lesen).	eingreifen muss.
Tool-Call	Ein **Tool-Call** (Funktionsaufruf) ist eine vom Modell ausgelöste Anfrage, eine externe Funktion oder ein externes Tool zu nutzen. Das Modell erkennt, wenn es zusätzliche Daten braucht, und gibt dann einen sogenannten Tool-Call zurück. Beispielsweise könnte das Modell auf die Frage „Wie ist das Wetter in Berlin?“ einen Tool-Call des Typs get_weather(“Berlin”) erzeugen.	Wenn das Sprachmodell eine Aktion oder Abfrage braucht, löst es einen Tool-Call aus, der dann vom System ausgeführt und als Ergebnis zurückgemeldet wird.
Tool-Whitelist	Eine **Tool-Whitelist** ist eine Vorgabe, welche Tools einem KI-Agenten erlaubt sind. Nur die auf der Whitelist aufgeführten Tools darf das Modell aufrufen; alle anderen Tools sind gesperrt. So wird sichergestellt, dass der Agent nur mit autorisierten Funktionen arbeitet.	Damit der Roboter nur sichere und erlaubte Aktionen ausführen kann, sind Tools auf einer Whitelist freigegeben und alles andere wird blockiert.
Prompts	**Prompts** sind Eingabeaufforderungen oder Anweisungen, die man einem KI-Modell gibt, um eine Antwort zu erhalten. Ein Prompt kann eine Frage, ein Befehl oder auch ein Beispieltext sein. Er enthält den Kontext oder die gewünschte Aufgabe; das Modell verarbeitet den Prompt, um eine passende Ausgabe zu erzeugen.	Prompts geben dem Sprachmodell pro Schritt klare Regeln und Kontext, damit der Roboter konsistent antwortet und bei Unsicherheit lieber nachfragt.
Debugging	**Debugging** (Fehlersuche) ist der Prozess, bei dem Entwickler in einem Programm Fehler (Bugs) suchen und beheben. Dabei wird das Programm schrittweise analysiert, um unerwartetes Verhalten oder Abstürze zu finden, und diese Probleme werden behoben.	Beim Debugging werden Probleme im Dialog, im Audiostream oder in der Navigation systematisch gesucht, geloggt und behoben, bis der Roboter stabil läuft.
Glitches	**Glitches** sind kurze Störungen oder Fehler im Programmablauf (häufig bei Spielen). Wörtlich ist ein Glitch „eine kleine Störung im Programmablauf“: Der Benutzer bemerkt einen Fehler (z.B. Grafikfehler), das Programm kann aber meist weiterlaufen. Glitches gehen meist nach einem Neustart oder Update wieder weg.	Bei Echtzeit-Audio oder Streaming können kurze Störungen auftreten, die durch Pufferung, Wiederholungen oder erneute Schritte abgefangen werden.
Latenz	**Latenz** bezeichnet die zeitliche Verzögerung bei der Übertragung oder Verarbeitung von Daten. Zum Beispiel ist die Netzwerklatenz die Zeitspanne, die ein Datenpaket benötigt, um von einem Punkt zu einem anderen zu gelangen. Hohe Latenz bedeutet also, dass es länger dauert, bis Daten gesendet und empfangen werden.	Eine geringe Latenz ist wichtig, damit der Roboter natürlich wirkt und Antworten nicht erst nach spürbarer Wartezeit kommen.
Hotword-Modell	Ein **Hotword-Modell** (auch *Wake-Word-Modell*) ist ein spezielles Spracherkennungsmodell, das ständig auf ein bestimmtes Schlüsselwort hört (z.B. „Hey Siri“ oder „OK Google“). Sobald dieses Hotword erkannt wird, aktiviert es den Sprachassistenten und signalisiert, dass nun Sprachbefehle folgen können.	Ein Hotword-Modell kann den Roboter erst nach einem Aktivierungswort starten, damit nicht dauerhaft Audio gestreamt werden muss.
Machine Learning	**Maschinelles Lernen (ML)** ist ein Teilbereich der Künstlichen Intelligenz. Dabei werden Algorithmen eingesetzt, die aus Trainingsdaten Muster „lernen“ und auf neue Daten anwenden. ML-Modelle können so Vorhersagen treffen oder Entscheidungen treffen, ohne dass jeder Schritt explizit programmiert ist.	Mehrere Komponenten des Roboters basieren auf Machine Learning, zum Beispiel Erkennung von Personen oder Objekten und die Sprachverarbeitung.
Labels	**Labels** (Beschriftungen) sind in Trainingsdaten die vorgegebenen Kategorien oder Tags, mit denen Beispiele versehen werden. Beim Data Labeling wird jedem Datensatz ein Label zugewiesen (z.B. „Hund“ oder „Katze“ für ein Bild). Durch diese Beschriftungen lernt das Modell, welche Merkmale zu welchem Label	Labels sind nötig, um Trainingsdaten sauber zu beschriften, damit Erkennungsmodelle zuverlässig lernen und später im Roboter korrekt funktionieren.

	gehören.	
WebSocket	WebSocket ist ein Netzwerkprotokoll, das nach einem initialen Verbindungsaufbau eine **dauerhafte, bidirektionale** Kommunikation zwischen Client und Server ermöglicht (beide Seiten können jederzeit Daten senden).	Die Echtzeitkommunikation für Sprache und Status erfolgt über eine WebSocket-Verbindung.
JSON	**JSON (JavaScript Object Notation)** ist ein **kompaktes, textbasiertes Datenformat** zum Austausch strukturierter Daten zwischen Anwendungen; es ist **programmiersprachenunabhängig**.	Nachrichten, Konfigurationen und Ergebnisse werden in JSON-Strukturen übertragen.
Audio-Ein-/ Ausgabe	Form der **Ausgabe**, bei der ein System nach interner Verarbeitung **Informationen als Audiosignal** über ein Ausgabegerät bzw. eine Schnittstelle **ausgibt** (z. B. an Lautsprecher/Kopfhörer).	Spracheingabe und Sprach-ausgabe sind zentrale Bestandteile der Interaktion.
Live-Audio	**Audiosignal-Übertragung in Echtzeit** (live) als Form des Livestreamings; das Audiosignal wird dabei in einen Datenstrom umgewandelt und unmittelbar übertragen. **Audio-Eingabe (Audio Input):** Form der **Eingabe**, bei der ein System **Audiosignale empfängt/aufnimmt** (typisch über Mikrofon oder Audio-Schnittstelle), um sie weiterzuverarbeiten oder zu speichern.	Audiodaten werden fortlaufend verarbeitet, um unmittelbare Reaktionen zu ermöglichen.
Streaming	„Streaming“ bezeichnet die **gleichzeitige Übertragung und Wiedergabe** von Audio- und/oder Videodaten über ein Netzwerk **als Datenstrom** (ohne dass zwingend zuerst eine vollständige Kopie beim Nutzer gespeichert wird).	Audio- und Statusinformationen werden als kontinuierlicher Datenstrom verarbeitet.
Puffering	Das **Zwischenspeichern** von Daten in einem Puffer, um z. B. Unterschiede in Verarbeitungsgeschwindigkeit auszugleichen oder kurze Übertragungs-/Lastspitzen abzufangen.	Kurze Audioabschnitte werden zwischengespeichert (gepuffert), bevor sie weitergeleitet werden.
Caching	Technik, bei der Daten **in einem schnellen Zwischenspeicher (Cache)** abgelegt werden, um **wiederholte Zugriffe auf langsamere Speicher** oder **aufwendige Neuberechnungen** zu vermeiden und spätere Zugriffe zu beschleunigen.	Häufig benötigte Informationen werden im Cache abgelegt, um Antworten zu beschleunigen.
Token-Management	Strategien, um innerhalb des Tokenlimits zu bleiben (z. B. Kürzen, Zusammenfassen, nur relevante Teile mitsenden).	Eingaben und Antworten werden so aufgebaut, dass Kontextfenster und Längenbeschränkungen eingehalten werden.
Turn-Taking	Mechanismus in Gesprächen, der beschreibt, **wie Redebeiträge (Turns) aufgebaut** und **das Rederecht zwischen Gesprächsteilnehmenden verteilt** wird; ein Sprecherwechsel wird an geeigneten Übergabestellen möglich.	Der Assistent steuert den Sprecherwechsel (Turn-taking) zwischen Zuhören und Antworten.
Prompting-Strategien	Systematisches Formulieren/Optimieren von Prompts, um **zuverlässigere und passendere** Modellantworten zu erhalten.	Pro Arbeitsschritt werden kurze, klare Anweisungen generiert.
Aktivierungswort	Ein festes Schlüsselwort, das einen Sprachassistenten **„aufweckt“** und erst danach die eigentliche Anfrage verarbeiten lässt.	Optional wird der Assistent erst nach einem Aktivierungswort (Wake Word) aktiv.
ROS2-Node	Eine Recheneinheit im ROS-2-System (Teil des „ROS Graph“), die **eine klar abgegrenzte Aufgabe** übernimmt und mit anderen Nodes kommuniziert.	Der Assistent läuft als ROS-2-Node und tauscht Daten innerhalb des Systems aus.
Bridges	Software, die Daten eines Systems **in ein Format übersetzt**, das ein anderes System verarbeiten kann	Schnittstellen können Robotikkomponenten und

	(Überbrückung inkompatibler Systeme/Formate).	externe Dienste koppeln.
Topics	Ein **benannter Kanal** im Publish/Subscribe-Modell; Publisher senden Nachrichten auf ein Topic, Subscriber empfangen diese Nachrichten über denselben Namen.	Status- und Sensordaten werden über Topics publiziert und abonniert.
Lokalisation	Bestimmung/Schätzung der **Position und Orientierung** (Pose) eines Roboters in seiner Umgebung.	Die Lokalisierung unterstützt Kontextaufbau und Navigation.
Navigation	Eine Variable im **Prozess-/Betriebssystem-Umfeld**, die Programme zur **Konfiguration** (z. B. Pfade, Schlüssel, Modi) auslesen können.	Der Assistent kann Navigationsziele anfahren und den Fortschritt melden.
Nav2	Ein ROS-2-Navigationsframework, das Funktionen wie **Planung, Steuerung und Lokalisierung** bereitstellt, damit mobile Roboter zu Zielpositionen navigieren können.	Bewegungen werden über den Navigationsstack ausgelöst.
Dispatcher	Eine Komponente bzw. ein Muster, das **eingehende Befehle/Anfragen** entgegennimmt und sie an die **passenden Handler** weiterleitet (Dispatching).	Funktionsaufrufe werden über einen Dispatcher an die passenden Aktionen weitergeleitet.
POI	Ein „Ort von Interesse“ als **punktförmiges Geoobjekt** mit Koordinate und optionalen Zusatzinfos, relevant z. B. für Karten/Navigationssysteme.	Orte wie Räume oder Ziele werden als Points of Interest (POIs) verwaltet.
Umgebungsvariablen	Eine Variable im **Prozess-/Betriebssystem-Umfeld**, die Programme zur **Konfiguration** (z. B. Pfade, Schlüssel, Modi) auslesen können.	Wichtige Einstellungen und Zugangsdaten werden über Umgebungsvariablen gesteuert.

Abbildungen:

Quellen:

Institut für Mobile und Verteilte Systeme (IMVS), FHNW, https://www.fhnw.ch/plattformen/imvs/, abgerufen am: 14.01.2026.

Fachhochschule Nordwestschweiz, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Fachhochschule_Nordwestschweiz&oldid=261623850, abgerufen am: 14.01.2026.

Programmierschnittstelle, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Programmierschnittstelle&oldid=259470413, abgerufen am: 14.01.2026.

Robot Operating System (ROS2) – Basis Kurs, Scientifica (Netzwerk Frauen.Innovation.Technik Baden-Württemberg), https://scientifica.de/kurse/robot-operating-system-ros2-basis-kurs/, abgerufen am: 14.01.2026.

Sprechpausenerkennung, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Sprechpausenerkennung&oldid=260858599, abgerufen am: 14.01.2026.

gpt-realtime, OpenAI API, https://platform.openai.com/docs/models/gpt-realtime, abgerufen am: 14.01.2026.

Gesichtserkennung, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Gesichtserkennung&oldid=259694532, abgerufen am: 14.01.2026.

Objekterkennung, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Objekterkennung&oldid=250560216, abgerufen am: 14.01.2026.

Intents (Dialogflow), Google Cloud, https://docs.cloud.google.com/dialogflow/es/docs/intents-overview, abgerufen am: 14.01.2026.

Absolute vs. relative Bewegungen, VEX Robotics Education, https://education.vex.com/stemlabs/de/cte/introduction-to-the-6-axis-arm/absolute-vs-relative-movements/introduction, abgerufen am: 14.01.2026.

Using tools, OpenAI API, https://platform.openai.com/docs/guides/tools, abgerufen am: 14.01.2026.

Function calling, OpenAI API, https://platform.openai.com/docs/guides/function-calling, abgerufen am: 14.01.2026.

Verzögerung (Telekommunikation), in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Verz%C3%B6gerung_(Telekommunikation)&oldid=227522981, abgerufen am: 14.01.2026.

Keyword spotting, in: Wikipedia, https://en.wikipedia.org/w/index.php?title=Keyword_spotting&oldid=1310435157, abgerufen am: 14.01.2026.

Maschinelles Lernen, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Maschinelles_Lernen&oldid=261741389, abgerufen am: 14.01.2026.

WebSockets, MDN Web Docs, https://developer.mozilla.org/de/docs/Glossary/WebSockets, abgerufen am: 15.01.2026.
JSON, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=JSON&oldid=262296639, abgerufen am: 15.01.2026.
Streaming Media, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Streaming_Media&oldid=263218240, abgerufen am: 15.01.2026.
Puffer (Informatik), in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Puffer_%28Informatik%29&oldid=259909934, abgerufen am: 15.01.2026.
Umgebungsvariable, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Umgebungsvariable&oldid=262421694, abgerufen am: 15.01.2026.
Tokenisierung, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Tokenisierung&oldid=262988957, abgerufen am: 15.01.2026.
Prompt Engineering, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Prompt_Engineering&oldid=263112866, abgerufen am: 15.01.2026.
Lokalisierung (Robotik), in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Lokalisierung_%28Robotik%29&oldid=236571174, abgerufen am: 15.01.2026.
Point of Interest, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Point_of_Interest&oldid=251288592, abgerufen am: 15.01.2026.
Datenströme, Gesellschaft für Informatik (GI) – Informatiklexikon, https://gi.de/informatiklexikon/datenstroeme, abgerufen am: 15.01.2026.

Konversationsanalyse, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Konversationsanalyse&oldid=262796805, abgerufen am: 15.01.2026.

Cache, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Cache&oldid=261878366, abgerufen am: 15.01.2026.

Livestreaming, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Livestreaming&oldid=261485972, abgerufen am: 15.01.2026.

Eingabe und Ausgabe, in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Eingabe_und_Ausgabe&oldid=262447004, abgerufen am: 15.01.2026.

Ausgabe (Computer), in: Wikipedia, https://de.wikipedia.org/w/index.php?title=Ausgabe_(Computer)&oldid=261226867, abgerufen am: 15.01.2026.
Nodes, ROS 2 Documentation, https://docs.ros.org/en/iron/Concepts/Basic/About-Nodes.html, abgerufen am: 15.01.2026.
Topics, ROS 2 Documentation, https://docs.ros.org/en/humble/Concepts/Basic/About-Topics.html, abgerufen am: 15.01.2026.
Nav2 Documentation, https://docs.nav2.org/, abgerufen am: 15.01.2026.
What are tokens and how to count them?, OpenAI Help Center, https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them, abgerufen am: 15.01.2026.
Context window, IBM, https://www.ibm.com/think/topics/context-window, abgerufen am: 15.01.2026.
The Command Dispatcher Pattern (PDF), The Hillside Group (PLOP), https://hillside.net/plop/plop2001/accepted_submissions/PLoP2001/bdupireandebfernandez0/PLoP2001_bdupireandebfernandez0_1.pdf, abgerufen am: 15.01.2026.
Wie funktionieren Sprachassistenten?, Universität Würzburg – MOTIV, https://www.mcm.uni-wuerzburg.de/motiv/sprachassistenten-portal/wie-funktionieren-sprachassistenten/, abgerufen am: 15.01.2026.

ChatGPT(5.2 Research): „Recherchiere für die folgenden Begriffe (Abkürzungen und Fachbegriffe) passende Definitionen auf Deutsch. Die Definitionen sollen für Laien verständlich sein. Verwende Wikipedia, offizielle Projektseiten oder offizielle Dokumentationen (z. B. FHNW/IMVS-Seite, ROS-Doku, OpenAI-Doku).

Wichtig: Wenn möglich, soll der Definitionstext 1:1 aus der Quelle übernommen werden. Gib zu jeder Definition den Link zur Quelle an.

Abkürzungen:
IMVS, FHNW, LLM, API, ROS2, VAD

Fachbegriffe:
OpenAI Realtime Model, Behaviour Tree, Blackboard, Gesichts- und Objekterkennung, Intent, Relative Bewegung, Deterministische Ablaufstruktur, Tools, Tool-Call, Tool-Whitelist, Prompts, Debugging, Glitches, Latenz, Hotword-Modell, Machine Learning, Labels, WebSocket, JSON, Audio-Ein-/Ausgabe, Live-Audio, Streaming, Puffering, Caching, Token-Management, Turn-Taking, Prompting-Strategien, Aktivierungswort, ROS2-Node, Bridges, Topics, Lokalisation, Navigation, Nav2, Dispatcher,POI, Umgebungsvariablen“, **14.01.2026. Text übernommen und sprachlich überarbeitet**