Inbetriebnahme des OpenMANIPULATOR-X und Bedienung mittels Handlungsplanung mit Partial Order Planning Forward

Fabian Claus

20130004

Bachelorarbeit

Fachbereich Informatik
und Medien
Technische Hochschule Brandenburg

Betreuer: Prof. Dr. Jochen Heinsohn
2. Betreuer: Dipl. Inform. Ingo Boersch

Brandenburg, den TT.MM.JJJJ
Bearbeitungszeit: TT.MM.JJJJ - TT.MM.JJJJ

Inhaltsverzeichnis

Zusammenfassung

Eine Kurzzusammenfassung der Vorgehensweise und der wesentlichen Ergebnisse.
Allgemeine Merkmale

- Objektivität: Es soll sich jeder persönlichen Wertung enthalten.
- Kürze: Es soll so kurz wie möglich sein.
- Verständlichkeit: Es weist eine klare, nachvollziehbare Sprache und Struktur auf.
- Vollständigkeit: Alle wesentlichen Sachverhalte sollen enthalten sein.
- Genauigkeit: Es soll genau die Inhalte und die Meinung der Originalarbeit wiedergeben.

Abstract

Obige Zusammenfassung in englischer Sprache.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

IDL Interface Definition Language

PlanSys2 ROS2 Planning System

PDDL Planning Domain Definition Language

POPF Partial Order Planning Forward

RCL ROS Client Library

ROS2 Robot Operating System 2

VM Virtuelle Maschine

ML Machine Learning

1 TODOS

- Neues Bild für Keyboard Teleop 5.2.5
- Bilder für Teleop Posen
- Bild für Servo Abdeckung vorgestanzt
- Überprüfen ob all Anhänge aktuell sind
- Entscheidung Bezeichnung für OMX, Abk., Greifarm, OpenManipulator-X, OpenMANIPULATOR X
- Fehlersuche Abstürze PlanSys2
- Zusammenfassen der Gripper Nodes zu einer parametrisierten Node
- baudrate änderungen für omx controller dokumentieren
- code kommentieren
- launch file für eigenes package beschreiben
- Move node umschreiben um pair anstelle mehrer maps zu nutzen
- PKG readme anpassen: start befehle sowie problem
- domänen dateien aufräumen, umbenennen
- Bilder für PlanSys2 Terminal

2 Einleitung

Dieser Teil der Arbeit sollte folgende Inhalte haben:

- Einführung in die Problemstellung
- Motivation und Herleitung des Themas
- Aufbau der Arbeit

Hinweis: Es hat sich als hilfreich erwiesen, die Einleitung mit der Zusammenfassung bzw. dem Abstract und der Schlussfolgerung zu vergleichen. Damit stellt man sicher, dass diese inhaltlich im Bezug auf Zielsetzung und Motivation übereinstimmen. Der Umfang sollte ca. 5 % der gesamten Arbeit betragen.

2.1 Aufgabe

Im Rahmen der Bachelorarbeit soll der Greifarm OpenMANIPULATOR-X der Firma Robotis in Betrieb genommen sowie die Möglichkeiten der Steuerung erprobt werden. Weiterhin soll die Steuerung Mittels Handlungsplanung ermöglicht werden. Hierfür sind bestehende Bibliotheken und Frameworks zu evaluieren und ein ausgewähltes zu implementieren.

2.2 Abgrenzung/Motivation

3 Grundlagen

3.1 ROS2

Robot Operating System 2 (ROS2) ist eine Sammlung von Bibliotheken und Werkzeugen für Robotik-Applikationen welche alle OpenSource sind. Die erste ROS2 Release-Version erschien im Dezember 2017 unter dem Namen Ardent Apalon **ros2docs**.
Es werden mehrere ROS Client Libraries (RCLs) zur Verfügung gestellt, welche den Zugriff auf die ROS2-API ermöglichen. Die RCLs für die Sprachen C++ und Python (rclcpp und rclpy) werden dabei direkt vom ROS2-Team verwaltet. Von der Community wurden weitere RCLs, unter anderem für die Sprachen C#, Swift und Rust, entwickelt.
Um die Entwicklung der RCLs zu vereinfachen und die Logik sprachen unabhängig zu machen werden Funktionalitäten als C Interfaces zugänglich gemacht, für welche in den RCLs Wrapper geschrieben werden.

3.1.1 Node

3.2 ROS2 Interfaces

In diesem Abschnitt werden die verschiedenen Arten von Interfaces beschrieben, welche für die im folgenden Abschnitt 3.3 erklärten Methoden zur Kommunikation zwischen mehreren Nodes genutzt werden.
ROS2 Interfaces sind Definitionen, welche Daten mit welchen Typen gesendet werden. Sie werden in der Interface Definition Language (IDL) geschrieben, welche es ermöglicht automatisch Source Code in verschiedenen Sprachen für diese zu generieren.
- Erklären was für den build nötig ist?

3.2.1 Message

Eine Message ist der einfachste Typ der ROS2 Interfaces, welcher gleichzeitig auch ein Baustein für die folgenden Interfaces sein wird.
Message Definitionen haben die Dateiendung `.msg` und werden per Konvention in einem Ordner `msg/` gespeichert. Jede `.msg` Datei besteht aus den folgenden Teilen:

- Felder
- Konstanten

Jedes Feld besteht aus einem Typ und einem durch ein Leerzeichen getrennten Namen: `typ name`, z.B. `bool isDone`. Als Typ können dabei die integrierten Typen (s. Tabelle 1) oder der Name anderer Messages genutzt werden. Zusätzlich kann jeder integrierte Typ als Array definiert werden (s. Tabelle 2).
Feldnamen müssen kleingeschrieben sein. Es können alphanumerische Zeichen sowie der Unterstrich zur Trennung von Wörtern genutzt werden. Das erste Zeichen muss ein Buchstabe sein und der Name darf nicht mit einem Unterstrich enden. Zusätzlich darf es keine aufeinander folgenden Unterstriche geben.

Konstanten behalten das Format der Felder bei. Der Name der Konstante wird komplett in Großbuchstaben geschrieben. Zusätzlich bekommen Konstanten einen Wert zugewiesen, welcher nicht innerhalb des Programms geändert werden kann. Die Zuweisung des Wertes erfolgt mit dem = Zeichen: `string EXAMPLE="test"`.

Tabelle 1: Integrierte Datentypen für Interfaces und deren C++ Equivalent

Typ	**C++**
bool	bool
byte	uint8_t
char	char
float32	float
float64	double
int8	int8_t
uint8	uint8_t
int16	int16_t
uint16	uint16_t
int32	int32_t
uint32	uint32_t
int64	int64_t
uint64	uint64_t
string	std::string
wstring	std::u16string

Tabelle 2: Array Typen und deren C++ Equivalent

Typ	**C++**
static array	std::array<T, N>
unbounded dynamic array	std::vector
bounded dynamic array	custom_class<T, N>
bounded string	std::string

3.2.2 Service

Service Definitionen beschreiben eine Anfrage und eine Antwort. Die Definition hat die Dateiendung `.srv` und wird im Ordner `srv/` gespeichert. Anfrage und Antwort werden innerhalb der Datei durch − getrennt. Für beide Teile gilt, dass sie gültig sind, wenn sie einer gültigen Message Definition entsprechen. Ein Beispiel einer einfachen Servicedefinition ist in Listing 1 zu sehen.

```
int32 request_int
string request_string
---
float32 response_float
```

Listing 1: Beispiel einer Service Definition

3.2.3 Action

Action Definitionen bestehen aus den 3 Teilen Anfrage, Ergebnis und Feedback, in dieser Reihenfolge. Wie auch für Service Definitionen gilt, das die Teile durch – getrennt sind und jeder einzelne Teil gültig ist, wenn er einer gültigen Message Definition entspricht.

3.3 ROS2 Node Kommunikation

Damit verschiedene Nodes untereinander kommunizieren können, gibt es verschieden Mechanismen, welche sich primär darin unterscheiden, in welche Richtungen Nachrichten gesendet werden und ob es eine direkte Reaktion gibt. Für alle Mechanismen gilt, das sie von einer Node unter einem bestimmten Namen sowie einem Typ zur Verfügung gestellt werden und von anderen Nodes durch eben diese genutzt werden können.

3.3.1 Topic

Topics entsprechen in etwa einem Newsletter System: eine Node veröffentlicht Daten, welche von allen anderen Nodes empfangen wird, die sich für diese registriert haben. Das Format der Daten entspricht einer gewählten Message Definition.

3.3.2 Service und Client

Services werden für Prozesse genutzt, in denen eine Anfrage gesendet und eine Antwort erwartet wird. Als Typ wird eine Servicedefinition genutzt.

3.3.3 Action Server und Client

Actions sind für länger andauernde Prozesse gedacht. Eine Node erstellt einen Action Server über welchen eine Action zur Verfügung gestellt wird. Andere Nodes können über einen Action Client eine Anfrage senden. Der Server bearbeitet die Anfrage und sendet bei Beendigung eine Antwort mit einem Ergebnis an den Client. Während der Dauer der Action kann der Server den Client optional mit Feedback Nachrichten über den aktuellen Status informieren.
Als Typ wird eine Action Definition genutzt.

3.4 OpenMANIPULATOR-X

Der OpenMANIPULATOR-X ist ein von der Firma ROBOTIS[1] nach den Prinzipien "OpenSoftware" und "OpenHardware" hergestellter Greifarm. OpenSoftware steht hierbei dafür, dass es ein OpenSource Projekt ist und auf dem OpenSource Projekt ROS2 basiert. OpenHardware steht dafür, dass die meisten Komponenten als STL-Dateien zur Verfügung stehen und als Ersatzteile oder zum Anpassen des Greifarms mittels eines 3D-Druckers selbst hergestellt werden können.
Der OMX(Greifarm?,Abk?) ist eine 5DOF (5 Degrees of Freedom) Plattform, welche mittels 5 Servomotoren[2] gesteuert wird. Dies ist aufgeteilt in 4DOF für den Arm sowie 1DOF für den Greifer. Es kann eine Last bis 500g getragen werden

3.5 Planung

3.5.1 STRIPS

3.5.2 Planning Domain Definition Language

Die Planning Domain Definition Language (PDDL) ist eine Sprache zur Modellierung von Planungs...

3.5.3 PDDL-Plugin für VS Code

- nach Implementierung verschieben?
- Installation
- Nutzung / Ausführung
- Pfad zur popf executable im foxy ordner
- Screenshot mit Bsp. für Planausgabe und Visualisierung

3.5.4 Partial Order Planning

3.5.5 Forward Chaining Partial Order Planning

3.5.6 Partial Order Planning Forward

(**popf**)

3.5.7 Behavior Tree

[1] http://en.robotis.com
[2] DYNAMIXEL XM430-W350-T

labe

4 Konzept

In diesem Abschnitt wird der Ansatz beschrieben, mit dem in dieser Arbeit vorgegangen werden soll.

4.1 Inbetriebnahme

Im ersten Schritt muss der OMX in Betrieb genommen werden. Dies beinhaltet den Zusammenbau und Konfiguration sowie die Installation nötiger Software. Ein besonderes Augenmerk wird dabei darauf gelegt, Details festzuhalten welche in den gegebenen Anleitungen nicht explizit angegeben werden oder leicht zu übersehen sind.

4.2 Struktur Nodes

Um eine einfache sowie übersichtliche Steuerung des Greifarms zu ermöglichen wird die Funktionalität in mehrere ROS2 Nodes aufgeteilt. Generell werden folgende Funktionalitäten benötigt: der Planer, die Speicherung des aktuellen Zustands der Welt, die Ausführung des Plans sowie die Möglichkeit Eingaben zu verarbeiten und an die entsprechenden Nodes weiterzuleiten.
Diese Aufteilung entspricht auch einer guten Aufteilung und Trennung der Verantwortungen um daraus ROS2 Nodes zu machen. Hier haben die Nodes folgende Verantwortlichkeiten:
Die Planungs-Node muss mit einer gegebenen Domäne und einem Problem einen Plan bestehend aus einer Reihe von Aktionen zurückgeben.
Die Welt-Node hält den aktuellen Zustand der Welt bzw. des Problems und muss diesen konsistent halten.
Die Ausführungs-Node muss entsprechend eines Plans die gegebenen Aktionen ausführen.
Die Eingabe-Node ermöglicht die Erstellung eines Problems mit einem Welt-Zustand und einem Ziel.

4.3 Planungsmodell

Für die Erstellung von Plänen wird eine Domäne im PDDL Format erstellt. In der Domäne werden die Aspekte des Stapelns von Blöcken mit denen des Greifers kombiniert. Zusätzlich werden *durative actions* genutzt, um die Dauer der einzelnen Aktionen zu modellieren.

4.4 Erstellung von Übungsbeispielen

Abschließend soll eine Übersicht, mit Möglichkeiten den OMX in die Lehre einzubinden, erstellt werden. Dies umfasst spezifische Beispiele sowie allgemeine Ansatzpunkte, um die Benutzung des OMX sowie dessen grundlegenden Konzepte zu vermitteln.

5 Implementierung

5.1 Inbetriebnahme Greifarm

Der OMX wird als Bausatz geliefert. Für die Inbetriebnahme ist daher der Zusammenbau und die Installation der entsprechenden Software nötig.

5.1.1 Zusammenbau

-Bausatz ca 40 Teile (ohne Schrauben)
-Rausbrechen Plastik bei Vorbereitung Servos
-Servos einzeln anschließen und per Dynamixel Wizard ID setzen
Der Bausatz des OMX besteht aus ca. 60 Teilen (ohne Schrauben, s. Abbildung 1).Einige der mit den Servomotoren mitgelieferten Teile werden dabei nicht benötigt, da der Bausatz des OMX diese auch enthält oder ersetzt (z.B. längere Kabel). Von allen Schrauben wurde außerdem Ersatz mitgeliefert.
Der Zusammenbau erfolgte nach der auf der Webseite verfügbaren Bauanleitung VERWEIS ANLEITUNG. Zu beachten ist, dass hier vorausgesetzt wird, dass den Servos bereits die IDs 11 (Basis des Greifarms) bis 15 (Greifer) zugewiesen wurden. Dies kann über die Software DYNAMIXEL Wizard[3] gemacht werden: die Servos einzeln über das U2D2 (s. Abschnitt 5.2.1) an den PC anschließen, die ID setzen und den Servo entsprechend markieren oder die ID merken. Weiterhin müssen bei den Abdeckungen der Servos 12 und 14 die vorgestanzten Abdeckungen herausgebrochen werden. Dies ist in der Anleitung leicht zu übersehen. Weiterhin wird angenommen, dass das Horn der Servos bereits angebracht ist. Hierbei ist darauf zu achten, dass die Einkerbung an Horn und Servo übereinstimmen.

5.1.2 Virtuelle Maschine

-Ubuntu 20.04
– Auflösung Einstellung "keine"
-Installation ROS2 Foxy
-Installation OMX
Zur Nutzung des Greifarms wurde eine VM mit VirtualBox [4] von Oracle aufgesetzt. Als Betriebssystem der VM wurde das für ROS2 Foxy empfohlene (**foxyreq**) Ubuntu 20.04 [5] gewählt. Danach wurde entsprechend der Anleitung für den OpenMANIPULATOR-X (**foxyinstall**) zuerst ROS 2 Foxy über das Installations-Script von ROBOTIS und im Anschluss die für den Greifarm benötigten Packages installiert.

[3] VERWEIS ODER LINK
[4] https://www.virtualbox.org
[5] `https://releases.ubuntu.com/20.04/`

Abbildung 1: Bausatz für den OpenMANIPULATOR-X

5.2 Steuerung OMX

Die Steuerung des OMX erfolgt über die vom *open_manipulator_x_controller* `open_manipulator_x_controller` zur Verfügung gestellten Topics und Services.

5.2.1 Anschluss über U2D2

Der OMX wird über das U2D2 und das U2D2 Power Hub Board (s. Abbildung 2) per USB an den PC angeschlossen.
Das U2D2 konvertiert die Signale der DYNAMIXEL und ermöglicht die Kontrolle über den PC. Zur Nutzung des U2D2 in der VM muss das Gerät FTDI XXXXXXXX über das Geräte-Menü der VM an diese gebunden werden (s. Abbildung 3).

5.2.2 OMX-Controller

Der OMX-Controller ist ein Package, welches automatisch beim Installieren der für den OMX benötigten Software installiert wird. Er kann über die entsprechende Launch-Datei mit dem Befehl

```
ros2 launch open_manipulator_x_controller
     open_manipulator_x_controller.launch.py
```

gestartet werden.

Abbildung 2: U2D2 Power Hub Board mit montiertem U2D2 (LINKS/RECHTS im Bild)

Abbildung 3: Geräte-Menü der VM mit gebundenem U2D2

5.2.3 Topics

5.2.4 Kinematik

5.2.5 Teleop

Mit einem laufenden OMX-Controller kann der OMX auch ohne extra Programmierung direkt ferngesteuert werden. Mögliche Geräte zur Steuerung sind die Tastatur sowie Playstation- und XBOX-Controller. Für ROS2 Foxy wird aktuell allerdings nur die Steuerung über die Tastatur unterstützt.
Der OMX kann dabei sowohl im Task Space als auch im Joint Space kontrolliert werden (s. Abbildung 4). Zusätzlich gibt es 2 vordefinierte Posen (Init und Home) in die der OMX bewegt werden kann (s. Abbildung 5).

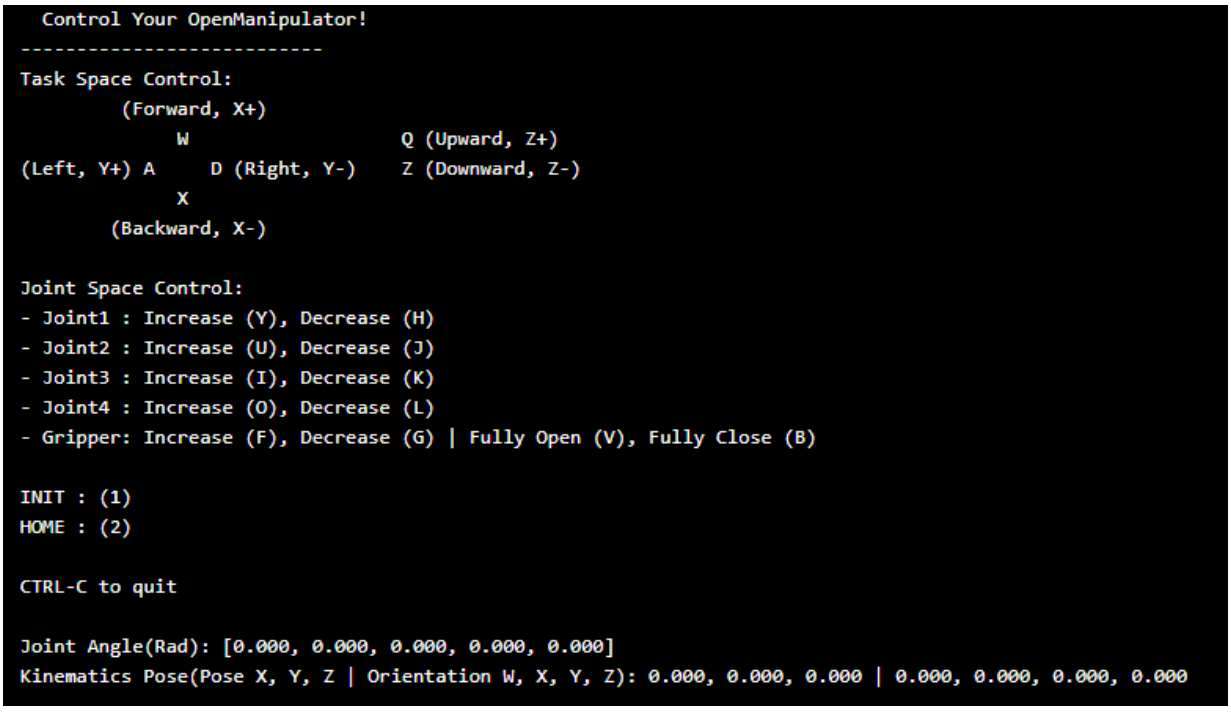

Abbildung 4: Fernsteuerung des OMX über die Tastatur

5.3 PlanSys2

plansys

Für die Implementierung der in Abschnitt 4.2 genannten Funktionalitäten wird das Framework ROS2 Planning System (PlanSys2) genutzt. Es übernimmt dabei alle Funktionalitäten: die Verwaltung der Daten zur Domäne und dem aktuellen Problem erfolgt über die domain-expert und problem-expert Nodes. Die planer Node ist für die komplette Planung zuständig und die Executioner Node für die Ausführung des Plans.
Es müssen hier lediglich die Domäne erstellt sowie die tatsächliche Funktionalität der

Abbildung 5: Home (l.) und Init (r.) Pose der Tastatur Teleop

einzelnen Aktionen implementiert werden. Das Wort Action im Namen der Nodes und den folgenden Kapiteln bezieht sich hier auf die Aktionen der Domäne, nicht auf ROS2 Actions.

5.3.1 PDDL-Domain

-Durative Actions
-Gripper + Blockworld
- keine existential/negative Preconditions
Beim erstellen der Domäne ist zu beachten, das Partial Order Planning Forward (POPF) nicht alle Funktionen unterstützt, die mit PDDL beschrieben werden können. Dies beinhaltet unter anderem existentielle sowie negative Vorbedingungen. Die vollständige Liste ist auf **??** zu finden.
Für den Blockworld Teil der Domäne gibt es zunächst den Typ `box` und Prädikate, die das Stapeln von Blöcken darstellen. Dies umfasst das Verhältnis von Blöcken aufeinander (`(box_on ?b_above ?b_below - box)`) sowie ob ein Block der oberste eines Stapels und damit frei zum bewegen ist (`(clear ?b - box)`). Als Aktionen werden das Nehmen sowie Ablegen eines Blocks benötigt. Da in den Bedingungen und Effekten einer Aktion nur mit Parametern gearbeitet werden kann, welche auch Parameter der Aktion selbst sind, wird hier weiter aufgeteilt in Nehmen/Ablegen eines Blocks von/auf einen anderen Block sowie von/auf einen leeren Stapel. Für das Bewegen von Blöcken ergeben sich die 4 Aktionen `GRAB, PLACE, STACK, UNSTACK`.
Während für das reine Stapeln der Blöcke die exakte Position dieser nicht wichtig ist, wird diese jedoch für den Greifer benötigt, welcher vor diesen Aktionen an die richtige Stelle bewegt werden muss. Hier werden die weiteren Typen `location` und `stack` sowie die Prädikate `(box_at ?b - box ?l - location)`,
`(gripper_at ?g - gripper ?l - location)`,
`(location_above ?l_above ?l_below - location)` und
`(is_base_loc ?l - location ?s - stack)` eingeführt.

Das Prädikat `clear` ist notwendig, da POPF weder existentielle noch negative Vorbedingungen unterstützt. Es wird genutzt um zu überprüfen, ob ein Block der oberste eines Stapels ist, was es ermöglicht diesen zu nehmen oder einen anderen auf diesem

abzulegen. Durch existentielle und negative Vorbedingungen könnte dies für die Aktion den Block B zu greifen, durch eine Bedingung der Form "es existiert kein Block C für den gilt:(box_on C B)" ersetzt werden.

Da alle Aktionen eine Dauer haben und PlanSys2 auch nur diese unterstützt, werden alle Aktionen als `durative-action` modelliert. Es wird jeder Aktion eine Dauer zugeordnet (`:duration (= ?duration 0.25)`) welche auch für Metriken genutzt werden kann. Zusätzlich bekommen Bedingungen und Effekte einen temporalen Zusatz: `at start` und `at end` können sowohl für Bedingungen als auch Effekte benutzt werden, um festzulegen zu welchem Zeitpunkt der Aktion diese Bedingung zutreffen oder der Effekt eintritt. Für Bedingungen gibt es außerdem `over all` damit eine Bedingung über die komplette Dauer der Aktiion zutreffen muss damit sie gültig ist.
Im aktuellen Szenario wird `over all` dafür genutzt, allen Block-Aktionen die Bedingung zu geben, dass sich der Greifer über die komplette Zeit an der gleichen Stelle wie der entsprechende Block befinden muss.
Nur in der `MOVE-GRIPPER` Aktion wird `at start` für einen Effekt genutzt: `(at start(not (gripp`
Sobald die Aktion beginnt befindet sich der Greifer nicht mehr an der Ausgangsposition
und erst am Ende der Aktion wird die neue Position als Fakt gesetzt (`(at end(gripper_at ?g ?l_`
Würden beide Effekte erst am Ende der Aktion eintreten, würde der Planer mehrere `MOVE-GRIPPER` Aktionen parallel ausführen (s. Abbildung **??**). Dies ist zum einen physisch nicht möglich, hinterlässt die Welt aber auch in einem ungültigen Zustand, da der Greifer sich dann an mehreren Positionen gleichzeitig befinden würde.

5.3.2 Action Nodes

Obwohl in der Domäne mehr als 2 Aktionen beschrieben sind, beschränken sich die ausgeführten Aktionen auf ein Bewegen des OMX sowie die Kontrolle des Greifers. Für diese wird jeweils eine Node erstellt. Damit diese Nodes von PlanSys2 genutzt werden können, müssen sie von der Basisklasse `plansys2::ActionExecutorClient` erben. Da diese von PlanSys2 nur in C++ zur Verfügung gestellt wird, müssen auch die Aktionen in C++ implementiert werden.
In jeder Aktion muss die Methode `void do_work()` implementiert werden. Diese wird mit einem bestimmten Zeitintervall aufgerufen während die Aktion aktiv ist. Das Intervall wird im Konstruktor gesetzt (s. Zeile **??** in **??**). Das Mapping einer Node zu einer Aktion erfolgt durch das Setzen des Parameters `action_name` nach dem Erstellen der Node (s. Zeile **??** in **??**).

5.3.3 Control Gripper Aktion

Die Logik Node zur Steuerung des Greifers wird in der Klasse `ControlGripperAction` (s. **??**) implementiert. Da es vom OMX kein Feedback gibt, ob oder wann etwas gegriffen wurde, wird die Aktion mit einer fixen Wartezeit implementiert: wenn die Aktion gestartet wird, wird ein Request zur Steuerung des Greifers erzeugt (s. Zeile **??** in **??**) und gesendet und nach einer bestimmten Zeit die Aktion beendet. Um unnötige Aufrufe der

Methode während des Wartens zu verhindern wird die Wartezeit über das Zeitintervall zur Ausführung der Node gesetzt. Die Aktion wird hierdurch immer beim 2. Aufruf der Methode `do_work` beendet.
Zum Senden des Requests wird ein ROS2 Client für den Service `goal_tool_control` mit dem Typ `open_manipulator_msgs::srv::SetJointPosition` erstellt. Um sowohl die Aktionen zum Öffnen sowie Schließen des Greifers mit einer Node implementieren zu können wird ein Parameter eingeführt, welcher über die Launch-Datei gesetzt wird und die Bewegung des Greifers bestimmt.

5.3.4 Move Gripper Aktion

Die Logik Node zur Bewegung des OMX wird in der Klasse `MoveGripperAction` (s. **??**) implementiert. Insgesamt umfasst dies das Senden des Bewegungsrequests an den vom OMX-Controller erstellten Service `goal_task_space_path_position_only` mit dem Typ `open_manipulator_msgs::srv::SetKinematicsPose` (s. Zeile **??** in**??**) sowie das Beenden der Aktion wenn der OMX an der Zielposition angekommen ist. Ob die Bewegung beendet ist wird über den im Topic `states` gesendeten Bewegungsstatus geprüft. Dieser kann die beiden Werte `IS_MOVING` und `STOPPED` annehmen. Der aktuelle Status wird mit dem letzten empfangenen verglichen: war der letze Status `IS_MOVING` und der aktuelle `STOPPED`, waren wir in einer Bewegung, die jetzt beendet ist.
Um die Zielkoordinaten der Bewegung zu erhalten, muss der String Parameter der Aktion in Zahlenwerte konvertiert werden. Dies erfolgt über mehrere Stufen mittels mehrerer Maps. Zunächst wird der String, welcher im Format `s<Nummer des Stapels>l<Höhe innerh` vorliegt zu 2 Integer im Bereich 1 bis 3 gemappt. Diese werden im 2. Schritt über weitere Maps zu Y und Z Koordinaten konvertiert. Die X Koordinate ist im Rahmen dieser Arbeit fest auf den Wert 0,25 gesetzt.
Um eine kollisionsfreie Bewegung sicherzustellen findet diese nicht direkt von der aktuellen zur Zielposition statt. Stattdessen werden mehrere Zwischenpositionen errechnet und entsprechende Bewegungen zuerst ausgeführt. Zunächst wird der OMX bei gleichbleibenden X und Y Koordinaten auf eine Z Koordinate von 0,25 bewegt. Dies entspricht eine Höhe in der keine Blöcke liegen können. Die zweite Zwischenposition bleibt auf der gleichen Höhe (Z Koordinate 0,25) und bewegt sich zur X und Y Koordinate der Zielposition. Diese Zwischenposition befindet sich also exakt oberhalb der Zielposition, sodass eine vertikale Bewegung zur Zielposition ohne Kollision nötig ist.
Diese Zwischenpositionen werden übersprungen, wenn die aktuelle Position des OMX bereits dem korrekten Stapel der Zielposition entspricht. Hierbei werden die aktuellen Koordinaten auf 2 Dezimalstellen gerundet.
Für alle benötigten Positionen wird ein Request erzeugt und einer Queue hinzugefügt. Der Auslöser, den nächsten Request zu senden ist jeweils das Ende der vorherigen Bewegung.
Um Zugriff auf die aktuelle Position des OMX zu erhalten wird ein weiterer Subscriber für das Topic `kinematics_pose` benötigt, welches eine Message vom Typ `open_manipulator_msgs`

veröffentlicht.

5.3.5 Package

- Launch file
- text datei mit commands
- pkg build config

Alle erstellten Komponenten werden in einem eigenen ROS2 Package gebündelt. Dies umfasst die C++ Dateien für die Aktionen sowie die Domänendatei. Zusätzlich wurde eine Launch Datei erstellt um das komplette System über einen Befehl starten zu können. In der Launch Datei wird zum einen PlanSys2 und zum anderen die Nodes für alle Aktionen gestartet. Weiterhin wird hier festgelegt, welche Domäne genutzt wird.
Das Package enthält außerdem eine Readme Datei in der die nötigen Schritte beschrieben sind um das gesamte System, einschließlich OMX, zu starten. Sie enthält auch die nötigen Befehle, um über das PlanSys2 Terminal ein Problem zu initialisieren, welches 3 Stapel mit je 3 möglichen Höhenpositionen hat, sowie einige platzierte Blöcke.

5.4 Praktische Anwendung

Um das System praktisch anwenden zu können wurden mehrere farbige Würfel aus LEGO gebaut (s. Abbildung **??**). Auf der Basisplatte des OMX wurden die Positionen der 3 Stapel markiert.

5.4.1 PlanSys2 Terminal

Das PlanSys2 Terminal ist eine separate Node, welche es ermöglicht, das System durch Eingabe über die Konsole zu steuern. Die wichtigsten Befehle sind hierbei `get`, `set` und `run`. Mit `get` können die Typen, Prädikate und Aktionen der geladenen Domäne sowie des aktuellen Problems angezeigt werden. Der `set` Befehl wird genutzt, um das aktuelle Problem zu verändern. Es können Instanzen, Prädikate sowie das Ziel erstellt und geändert werden. Über den Befehl `run` lässt sich der Plan für das aktuelle Ziel ausführen. Es ist auch möglich nur einen Plan zu suchen, ohne ihn direkt auszuführen. Dies ist mit `get plan` möglich.
Während der Ausführung eines Plans wird im Terminal die aktuelle Aktion sowie dessen Fortschritt (gesteuert über den Feedback Kanal innerhalb der Aktion) angezeigt.

6 Ergebnis

6.1 Zusammenfassung

- off. Unterstützung von Foxy des OMX erst während der Bearbeitung

6.2 Ausblick

6.2.1 MoveIt

Um den in der dieser Arbeit verwendeten Ansatz der Bewegungsplanung robuster und dynamischer zu machen empfiehlt sich die Implementierung der Bibliothek MoveIt [6]. Zum Zeitpunkt der Bearbeitung wird diese vom OMX beim Betrieb mit ROS2 Foxy allerdings noch nicht offiziell unterstützt.
MoveIt bietet mittels virtueller Szenen die Möglichkeit effiziente und kollisionsfreie Wege für die Bewegung des OMX zu planen und auszuführen.

6.2.2 Vorschläge für Übungen in der Lehre

Um die Verwendung des OMX in der Lehre näher zu bringen gibt es in dem für diese Arbeit entstandenen Package mehrere Stellen, an denen angesetzt werden kann.
Grundsätzlich lassen sich diese in folgende Bereiche aufteilen:

- ROS2
- Domäne und Problem in PDDL
- ...

Für jeden dieser Bereiche besteht die Möglichkeit die Aufgaben in die folgenden Richtungen zu führen:

- Fehlersuche mit sinnvoll platzierten Fehlern
- Erkennen und Korrigieren weggelassener Abschnitte, welche durch das Verstehen des Codes erkannt werden können
- die Anpassung des Codes an ein ähnliches Szenario

Ein Beispiel für die Domäne ist, den Effekt der `MOVE-GRIPPER` Aktion der die Startposition löscht erst am Ende der Aktion auszuführen, was dazu führt, dass zum Start des Plans alle nötigen Bewegungen gleichzeitig ausgeführt werden.
Eine Aufgabe für das Schreiben/ Anpassen eines Problems in PDDL, welche sich anbietet, ist eine vorgegeben Problemdatei, welche nur die grundlegende Struktur über Stapel und Positionen enthält so zu erweitern, dass eine von mehreren möglichen, vorgegebenen Varianten von gestapelten Blöcken beschreibt. Die Aufgabe kann dadurch erleichtert werden, dass das gegebene Problem bereits eine bestimmte Stapelung von Blöcken darstellt und nur angepasst werden muss.

[6] MOVEIT URL

Für alle Aufgaben die PDDL betreffen bietet es sich an, VS-Code mit dem PDDL-Plugin zu nutzen um schnelle Iterationen und eine Visualisierung der Pläne zu ermöglichen.

A Quellcode

A.1 Planungsdomäne

```
(define (domain blockworld)
(:requirements :strips :typing :adl :fluents :durative-actions)

;; Types ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:types
    box
    gripper
    location
    stack
);; end Types ;;;;;;;;;;;;;;;;;;;;;;;

;; Predicates ;;;;;;;;;;;;;;;;;;;;;;;
(:predicates

    (gripper_at ?g - gripper ?l - location)
    (box_at ?b - box ?l - location)
    (box_on ?b_above ?b_below - box)
    (gripper_open ?g - gripper)
    (is_holding ?g - gripper ?b - box)
    (clear ?b - box)
    (stack_empty ?s - stack)
    (location_above ?l_above ?l_below - location)
    (is_base_loc ?l - location ?s - stack)

);; end Predicates ;;;;;;;;;;;;;;;;;;;
;; Functions ;;;;;;;;;;;;;;;;;;;;;;;;
(:functions

);; end Functions ;;;;;;;;;;;;;;;;;;;
;; Actions ;;;;;;;;;;;;;;;;;;;;;;;;;;

(:durative-action GRAB
    :parameters (
        ?g - gripper
        ?b - box
        ?l - location
        ?s - stack
    )
    :duration (= ?duration 0.25)
    :condition (
        and
            (over all(gripper_at ?g ?l))
            (at start(box_at ?b ?l))
            (at start(gripper_open ?g))
            (at start(clear ?b))
            (at start(is_base_loc ?l ?s))
    )
```

```
    :effect (
        and
            (at end(not (box_at ?b ?l)))
            (at end(not (gripper_open ?g)))
            (at end(is_holding ?g ?b))
            (at end(stack_empty ?s))
            (at end(not(clear ?b)))
    )
)

(:durative-action PLACE
    :parameters (
        ?g - gripper
        ?b - box
        ?l - location
        ?s - stack
    )
    :duration (= ?duration 0.25)
    :condition (
        and
            (at start(is_holding ?g ?b))
            (over all(gripper_at ?g ?l))
            (at start(is_base_loc ?l ?s))
            (at start(stack_empty ?s))
    )
    :effect (
        and
            (at end(box_at ?b ?l))
            (at end(not (is_holding ?g ?b)))
            (at end(gripper_open ?g))
            (at end(clear ?b))
            (at end(not(stack_empty ?s)))
    )
)

(:durative-action STACK
    :parameters (
        ?g - gripper
        ?b ?b2 - box
        ?l ?l2 - location
    )
    :duration (= ?duration 0.25)
    :condition (
        and
            (at start(clear ?b2))
            (at start(is_holding ?g ?b))
            (over all(gripper_at ?g ?l))
            (at start(location_above ?l ?l2))
            (at start(box_at ?b2 ?l2))
    )
    :effect (
```

```
        and
            (at end(not(clear ?b2)))
            (at end(box_on ?b ?b2))
            (at end(clear ?b))
            (at end(box_at ?b ?l))
            (at end(not(is_holding ?g ?b)))
            (at end(gripper_open ?g))
    )
)

(:durative-action UNSTACK
    :parameters (
        ?g - gripper
        ?b ?b2 - box
        ?l ?l2 - location
    )
    :duration (= ?duration 0.25)
    :condition (
        and
            (at start(clear ?b))
            (at start(gripper_open ?g))
            (at start(box_on ?b ?b2))
            (over all(gripper_at ?g ?l))
            (at start(box_at ?b ?l))
            (at start(box_at ?b2 ?l2))
    )
    :effect (
        and
            (at end(not(clear ?b)))
            (at end(not(box_on ?b ?b2)))
            (at end(clear ?b2))
            (at end(is_holding ?g ?b))
            (at end(not(box_at ?b ?l)))
            (at end(not(gripper_open ?g)))
    )
)

(:durative-action MOVE-GRIPPER
    :parameters (
        ?g - gripper
        ?l_from ?l_to - location
    )
    :duration (= ?duration 1)
    :condition (
        and
            (at start(gripper_at ?g ?l_from))
    )
    :effect (
        and
            (at start(not (gripper_at ?g ?l_from)))
            (at end(gripper_at ?g ?l_to))
```

```
    )
)

);; end Domain ;;;;;;;;;;;;;;;;;;;;;;;;
```

A.2 Gripper Package Launch Datei

```
# Copyright 2019 Intelligent Robotics Lab
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, IncludeLaunchDescription,
    SetEnvironmentVariable
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


def generate_launch_description():
    # Get the launch directory
    example_dir = get_package_share_directory('blockworld_gripper')
    namespace = LaunchConfiguration('namespace')

    declare_namespace_cmd = DeclareLaunchArgument(
        'namespace',
        default_value='',
        description='Namespace')

    stdout_linebuf_envvar = SetEnvironmentVariable(
        'RCUTILS_LOGGING_BUFFERED_STREAM', '1')

    plansys2_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(
            get_package_share_directory('plansys2_bringup'),
            'launch',
```

```
            'plansys2_bringup_launch_monolithic.py')),
        launch_arguments={
          'model_file': example_dir + '/pddl/test-domain.pddl',
          'namespace': namespace
          }.items())

    # Specify the actions
    move_gripper_cmd = Node(
        package='blockworld_gripper',
        executable='move_gripper_action_node',
        name='move_gripper_action_node',
        namespace=namespace,
        output='screen',
        parameters=[])

    grab_cmd = Node(
        package='blockworld_gripper',
        executable='grab_action_node',
        name='grab_action_node',
        namespace=namespace,
        output='screen',
        parameters=[])

    place_cmd = Node(
        package='blockworld_gripper',
        executable='place_action_node',
        name='place_action_node',
        namespace=namespace,
        output='screen',
        parameters=[])

    stack_cmd = Node(
        package='blockworld_gripper',
        executable='stack_action_node',
        name='stack_action_node',
        namespace=namespace,
        output='screen',
        parameters=[])

    unstack_cmd = Node(
        package='blockworld_gripper',
        executable='unstack_action_node',
        name='unstack_action_node',
        namespace=namespace,
        output='screen',
        parameters=[])
    # Create the launch description and populate
    ld = LaunchDescription()

    # Set environment variables
    ld.add_action(stdout_linebuf_envvar)
```

```
    ld.add_action(declare_namespace_cmd)

    # Declare the launch options
    ld.add_action(plansys2_cmd)

    ld.add_action(move_gripper_cmd)
    ld.add_action(grab_cmd)
    ld.add_action(place_cmd)
    ld.add_action(stack_cmd)
    ld.add_action(unstack_cmd)

    return ld
```

A.3 Move Gripper Action Node

```
#include <memory>
#include <algorithm>
#include <queue>

#include "plansys2_executor/ActionExecutorClient.hpp"

#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"

#include "open_manipulator_msgs/msg/open_manipulator_state.hpp"
#include "open_manipulator_msgs/msg/kinematics_pose.hpp"
#include "open_manipulator_msgs/srv/set_kinematics_pose.hpp"

using namespace std::chrono_literals;
using std::placeholders::_1;

class MoveGripperAction : public plansys2::ActionExecutorClient {
public:
    MoveGripperAction()
            : plansys2::ActionExecutorClient("move_gripper", 100ms) {
        isStarted = false;
        isCurrentMovementFinished = false;
        manipulator_state_subscription_ = this->create_subscription<
    open_manipulator_msgs::msg::OpenManipulatorState>(
                "states", 10, std::bind(&MoveGripperAction::
    manipulator_state_callback, this, _1));
        kinematics_pose_subscription_ = this->create_subscription<
    open_manipulator_msgs::msg::KinematicsPose>(
                "kinematics_pose", 10, std::bind(&MoveGripperAction::
    kinematics_pose_callback, this, _1));
        kinematicsPoseClient = this->create_client<open_manipulator_msgs::
    srv::SetKinematicsPose>(
                "goal_task_space_path_position_only");
        while (!kinematicsPoseClient->wait_for_service(1s)) {
            if (!rclcpp::ok()) {
```

```
                RCLCPP_ERROR(rclcpp::get_logger("rclcpp"), "Interrupted
    while waiting for the service. Exiting.");
                break;
            }
            RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "KinematicsPose
    service not available, waiting again...");
        }
    }

private:
    void manipulator_state_callback(const open_manipulator_msgs::msg::
    OpenManipulatorState::SharedPtr msg) {
        //std::cout << msg->open_manipulator_moving_state << std::endl;
        last_moving_state = current_moving_state;
        current_moving_state = msg->open_manipulator_moving_state;
        if(last_moving_state == STATE_MOVING && current_moving_state ==
    STATE_STOPPED){
            isCurrentMovementFinished = true;
            queueTasksDone++;
        }
    }
    void kinematics_pose_callback(const open_manipulator_msgs::msg::
    KinematicsPose::SharedPtr msg) {
        //std::cout << &msg << std::endl;
        kinematicsPose = msg;
    }

    void do_work() {

        if (!isStarted) {
            isStarted = true;
            isCurrentMovementFinished = true; //we want to immediately start
     the next movement
            auto args = get_arguments();
            int level = -1;

            std::map<std::string, int>::const_iterator iterLevel = levelMap.
    find(args[2]);
            if (iterLevel != levelMap.end()) {
                std::cout << iterLevel->first << "->" << iterLevel->second
    << " ";
                level = iterLevel->second;
            }
            int stack = -1;
            std::map<std::string, int>::const_iterator iterStack = stackMap.
    find(args[2]);
            if (iterStack != levelMap.end()) {
                std::cout << iterStack->first << "->" << iterStack->second;
                stack = iterStack->second;
            }
```

```
            create_movement(stack, level);
            queueLength = requestQueue.size();
            queueTasksDone = 0;
            send_feedback(float(queueTasksDone) / queueLength, "Move started
");
        }
        if(isCurrentMovementFinished){
            if (!requestQueue.empty()) {
                auto nextRequest = requestQueue.front();
                requestQueue.pop();
                isCurrentMovementFinished = false;
                auto result = kinematicsPoseClient->async_send_request(
nextRequest);
                send_feedback(float(queueTasksDone) / queueLength, "Move
started");
            }
            else if(isStarted){
                //we did start and no more movement enqueued
                finish(true, 1.0, "Move completed");

                isStarted = false;
                std::cout << std::endl;
            }
        }


        std::cout << "\r\e[K"  << std::flush;
        std::cout << "Moving ... [" << float(queueTasksDone) / queueLength
<< "]  " << std::flush;
    }

    void create_movement(int stack, int level){
        //clear the queue by swapping with an empty one
        std::queue<std::shared_ptr<open_manipulator_msgs::srv::
SetKinematicsPose::Request>> emptyQueue;
        std::swap(requestQueue, emptyQueue);
        if(stack == -1 || level == -1 || kinematicsPose == nullptr){
            std::cout << "Error detected when creating movement stack: " <<
stack << " level: " << level << " Pose null: " << (kinematicsPose ==
nullptr);
            return;
        }
        //a position above the current that is clear from collisions
        auto currentPosition = kinematicsPose->pose.position;
        std::cout << "Current (" << currentPosition.x << ", " <<
currentPosition.y << ", " << currentPosition.z << ")" << std::flush;
        std::cout << "Target (" << STACK_POS << ", " << stackPosMap[stack]
<< ", " << heightMap[level] << ")" << std::flush;
        if(roundTo2DecimalPlaces(currentPosition.x) != STACK_POS ||
roundTo2DecimalPlaces(currentPosition.y) != stackPosMap[stack]){
```

```
            auto currentClearRequest = create_request(currentPosition.x,
    currentPosition.y, CLEAR_HEIGHT);

            //a position above the target position that is clear from
    collisions
            auto targetClearRequest = create_request(STACK_POS,stackPosMap[
    stack], CLEAR_HEIGHT);
            requestQueue.push(currentClearRequest);
            requestQueue.push(targetClearRequest);
        }

        //the position we want to end up at
        auto targetPointRequest = create_request(STACK_POS, stackPosMap[
    stack], heightMap[level]);
        requestQueue.push(targetPointRequest);
    }

    std::shared_ptr<open_manipulator_msgs::srv::SetKinematicsPose::Request>
    create_request(float x, float y, float z) {
        auto request = std::make_shared<open_manipulator_msgs::srv::
    SetKinematicsPose::Request>();
        request->end_effector_name = "gripper";
        request->kinematics_pose.pose.position.x = x;
        request->kinematics_pose.pose.position.y = y;
        request->kinematics_pose.pose.position.z = z;
        request->path_time = 2.5;

        return request;
    }

    float roundTo2DecimalPlaces(float input){
        return roundf(input * 100) / 100;
    }

    rclcpp::Subscription<open_manipulator_msgs::msg::OpenManipulatorState>::
    SharedPtr manipulator_state_subscription_;
    rclcpp::Subscription<open_manipulator_msgs::msg::KinematicsPose>::
    SharedPtr kinematics_pose_subscription_;
    rclcpp::Client<open_manipulator_msgs::srv::SetKinematicsPose>::SharedPtr
     kinematicsPoseClient;
    std::shared_ptr<open_manipulator_msgs::msg::KinematicsPose>
    kinematicsPose;
    std::queue<std::shared_ptr<open_manipulator_msgs::srv::SetKinematicsPose
    ::Request>> requestQueue;
    bool isStarted;
    bool isCurrentMovementFinished;
    int queueLength;
    int queueTasksDone;
    //int level = -1;
    std::string current_moving_state;
    std::string last_moving_state;
```

```
    const std::string STATE_MOVING = "IS_MOVING";
    const std::string STATE_STOPPED = "STOPPED";
    const float STACK_POS = 0.25;
    const float CLEAR_HEIGHT = 0.25;
    std::map<std::string, int> levelMap = {
            {"s1l1", 1},
            {"s2l1", 1},
            {"s3l1", 1},
            {"s1l2", 2},
            {"s2l2", 2},
            {"s3l2", 2},
            {"s1l3", 3},
            {"s2l3", 3},
            {"s3l3", 3},
    };
    std::map<std::string, int> stackMap = {
            {"s1l1", 1},
            {"s1l3", 1},
            {"s1l2", 1},
            {"s2l1", 2},
            {"s2l3", 2},
            {"s2l2", 2},
            {"s3l2", 3},
            {"s3l1", 3},
            {"s3l3", 3},
    };
    std::map<int, float> heightMap = {
            {1, 0.04},
            {2, 0.085},
            {3, 0.13},
    };
    std::map<int, float> stackPosMap = {
            {1, -0.1},
            {2, 0.0},
            {3, 0.1},
    };

};

int main(int argc, char **argv) {
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MoveGripperAction>();

    node->set_parameter(rclcpp::Parameter("action_name", "move-gripper"));
    node->trigger_transition(lifecycle_msgs::msg::Transition::
   TRANSITION_CONFIGURE);

    rclcpp::spin(node->get_node_base_interface());

    rclcpp::shutdown();
```

```
    return 0;
}
```