ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

pd | z Product Development Group Zurich
Produktentwicklungsgruppe Zürich

Bachelor Thesis

Task Planning Interface for a Robot Arm in Mixed Reality

Documentation

Author

Yannick Pascal Huber

Supervisor

Sophokles Ktistakis

Professor

Prof. Dr. M. Meboldt

Zurich, July 13, 2023

Preface

This document accompanies the bachelor thesis 'Task Planning Interface for a Robot Arm in Mixed Reality' by Yannick Pascal Huber, written in the spring of 2023. However, it is not the official report of said bachelor thesis.

It acts as documentation for the 'Task Planning Interface' and all of its content, explaining how to work with the underlying elements and classes and what to look out for or pay special attention to.

Please refer to the report for a more in-depth view of the system architecture and implementation.

- Yannick Pascal Huber, Zurich, July 13, 2023

Contents

Nomenclature

Important Terms

Component	functional piece of GameObjects, containing properties that can be edited to define the GameObject's behavior. [1]
Constraint	restriction placed on a robotic system that narrows its achievable motion and function possibilities, e.g., maximal force or speed. Constraints can be applied globally or snippet-specific (only applied when a specific snippet is active). [2]
Dialog Menu	refers to a menu opened at runtime by the TPI to display information or let the operator make choices.
GameObject	in Unity, it refers to a fundamental object that represents characters, props, and scenery, which acts as a container for components [3]
Prefab	in Unity it refers to a prefabricated, configured object that is saved as an asset for reuse [4]
Sequence	in the TPI it refers to a list of snippets, where each snippet is run one after the other
Snippet	machine tasks like 'Move To' or 'Bring Me', whose parameters can be set up with the help of the TPI
Workflow	consists of snippets in a sequence and constraints and thus defines what the robot should do

Acronyms and Abbreviations

MR	Mixed Reality
MRTK	Mixed Reality Toolkit
ROS	Robot Operating System
TPI	Task Planning Interface
URDF	Universal Robot Description Format

1 Introduction

Welcome to the Task Planning Interface (TPI), a bachelor thesis by Yannick Huber.

The Task Planning Interface (TPI), created with the help of the Mixed Reality Toolkit v2.8.3 (MRTK) [5] and Unity3D for the Microsoft HoloLens 2 [6], acts as a framework for the creation of workflows, consisting of machine tasks, lovingly called snippets, and constraints, thus allowing you to define entire processes like the manufacturing of complex parts involving multiple steps before their execution. However, it can also be used with other Mixed Reality glasses supported by the MRTK.

Alongside the handling of the internal information in the background, the framework also comes with a graphical interface, which allows the operator to access and control the features of the TPI at runtime in the MR environment.

Furthermore, it gives you the option to connect to ROS [7], control the digital twin of the robot (initially set up for the FRANKA RESEARCH 3 robotic arm [8]), create dialog menus and much more.

This documentation guides you through the most important parts of the TPI, hopefully allowing you to gather some knowledge of how the framework works and how you can use it to efficiently create working workflows and tailor the experience to your liking.

The Task Planning Interface was developed with the help of the supervisor Sophokles Ktistakis, the help of Accenture, and the help of the Product Development Group Zurich ($pd|\mathbf{z}$) in the spring of 2023.

Throughout the project's code, you can find summaries and some tutorials that can be accessed by visiting the specific sections or by hovering over the names of functions and variable types created by the TPI framework like 'TPI_Snippet'. Thus, it usually gives you a short description of the underlying functionalities, sometimes containing short example code sequences. Additionally, if applicable, it provides you with information about the return value.

2 Installation and Setup

Installation

The first step to successfully set up the TPI is to install MRTK by following the tutorial found on the following website: `https://learn.microsoft.com/en-us/training/modules/learn-mrtk-tutorials/1-3-exercise-configure-unity-for-windows-mixed-reality?ns-enrollment-type=learningpath&ns-enrollment-id=learn.azure.beginner-hololens-2-tutorials`).
Please keep in mind when selecting a version that v2.8.3 of the MRTK was used when the TPI was designed, whereas MRTK v3.X might not work.

Afterward, please install the 'ROS-TCP-Connector' from the 'Unity Robotics Hub' and the 'URDF Importer' as described on this website: `https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/quick_setup.md`

Then, please install the 2D Sprite Unity Package in the 'Package Manager' of Unity. (Only the '2D Sprite Package' of the 'Sprite Editor' is needed for the TPI, the rest being optional to install.)
Window → Package Manager → Select 'Packages: Unity Registry' instead of 'Packages: in Project' → Scroll down to 'Packages' → Install '2D Sprite' v1.0.0 (or greater).

Once you have installed those packages, you have successfully prepared the project for the TPI!

Now, you can either copy the assets from the TPI directly into your folder structure or import them from GitHub using the link found in chapter A 'System Specifications' of the appendix.

The TaskPlanningInterface folder contains the prefab 'TPI_Manager', which is the essential part of the TPI that holds all the components and GameObjects needed for a successful setup. Drag it into your Hierarchy.

Additionally, you can find the prefab 'TPI_DebugTools', a collection of buttons with predefined functions, that can help you work with the TPI in the Editor, as the hand menus will not be shown there.

Setup

To configure the TPI, you have the following options:

Table 2.1: Overview of the available options to customize the TPI.

Component	Description of the Options
TPI_DialogMenuController	Options concerning the dialog menus
TPI_HandGestureController	Options concerning the hand gestures
TPI_MainController	Options concerning the TPI in general and the 'Workflow Hand Menu', i.e., icons and references to the different parts of the TPI
TPI_ObjectPlacementController	Options concerning the grid algorithm that strategically places the menus in the environment around the operator
TPI_RobotController	Options concerning the digital twin of the robot
TPI_ROSController	Options concerning ROS and the 'ROS Status Menu'
TPI_SequenceMenuController	Options concerning the 'Sequence Menu' and the sequence functions (e.g., 'StartSequence()')
TPI_TutorialController	Options concerning the tutorial feature
TPI_WorkflowConfiguration-Controller	Options concerning the 'Sequence Menu' and 'Building Blocks Menu', i.e., categories, snippet options, constraint options

Usually, the names of the input fields in the Unity inspector are self-explanatory. By hovering over the name of different input fields, you will see a short description, which can be used as a further point of reference for the underlying functionality.

3 Menu Descriptions

This chapter briefly describes and summarizes the functions of the various menus implemented in the TPI. Therefore, if you were to stumble over the names of one of these menus, you can look up their aim here.

3.1 Selection Menu

The 'Selection Menu' is used to organize the snippet and constraint templates into neat categories, each of those categories having its individual button. By selecting a category button, the corresponding content is opened in the 'Building Blocks Menu', which is described in section 3.2.
Furthermore, hovering over one of the buttons displays a short description, which can be set up for each category individually. If you do not want any text to appear, do not add a description when setting up the category.

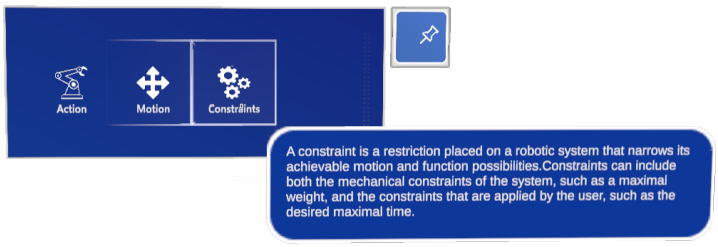


Figure 3.1: The figure shows a sample 'Selection Menu' with the categories 'Action', 'Motion', and 'Constraints'. Furthermore, it shows how a description can be displayed by hovering over the button. The categories can be manually added in the 'TPI_WorkflowConfigurationController'. The 'Selection Menu' automatically increases in size depending on the number of entries.

New categories, which will automatically be added to this menu, can be created in the 'TPI_WorkflowConfigurationController' component of the 'TPI_Manager' GameObject.

3.2 Building Blocks Menu

The 'Building Blocks Menu' displays the snippet and constraint templates belonging to the category selected in the 'Selection Menu'. Thus, it automatically shows buttons for each snippet or constraint. Furthermore, hovering over one of the buttons displays a short description. If you do not want any text to appear, do not add a description when setting up the snippet or constraint.

Figure 3.2: The figure shows a sample 'Building Blocks Menu' with 'Move To' and 'Bring Me' snippets. By hovering over an item, a description would be shown. The 'Building Blocks Menu' automatically increases in size depending on the number of entries.

New elements for the categories (e.g., snippets or constraints) can be set up in the 'TPI_WorkflowConfigurationController' component of the 'TPI_Manager' GameObject.

Initially, the constraints are all grouped into the same category called 'Constraints'. However, this can be changed to allow constraints to be placed in other categories.
To facilitate that, you would need to add a variable of type string (e.g., 'constraintCategoryID') to the 'TPI_ConstraintInformation' class. Furthermore, you would need to adapt the 'TPI_WorkflowConfigurationControllerEditor' to allow the selection of category IDs via a dropdown menu (it was already set up for the snippets and can thus be expanded). Finally, you would need to make a few changes in the 'TPI_WorkflowConfigurationController' class, such that the category 'Constraints' does not get created at startup and that the constraints will be placed in the category belonging to their individual category ID.

3.3 Sequence Menu

The 'Sequence Menu' is the central visual area that holds all the information of the configured snippets of the machine task sequence and the configured global or snippet-specific constraints.

Figure 3.3: The figure shows a sample 'Sequence Menu' for a workflow called 'SlantedWall Welding' with the snippets 'Weld Base Plate' and 'Drill Holes', and the constraint 'NoGoZone: Walls'. The template names 'Welding', 'Drilling', and 'NoGoZone' can be enabled and shortened if desired.

The lists of snippets and constraints are vertically scrollable, allowing you to navigate through them and search for the correct items easily.
You can make changes to the list entries with the help of the following gestures:

Table 3.1: Overview of the available gestures in the sequence menu.

Available Gestures		
Actions to be performed	Snippets	Constraints
Selection	Single Click	N/A
Change Sequence Order	Hold + Move vertically	N/A
Change Values	Double Click	Single Click
Deletion	Hold + Move to the right	Hold + Move to the right

Once the sequence has been started using either hand gestures or the 'Sequence Functions Menu', a group of spinning circles will appear next to the currently active and running snippet to indicate the status 'running'.

3.4 Sequence Functions Menu

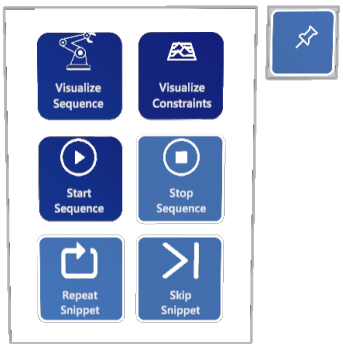


Figure 3.4: The figure shows the 'Sequence Functions Menu' before the sequence has been started. The 'Stop Sequence', 'Repeat Snippet', and 'Skip Snippet' buttons are currently locked. All the other buttons are unlocked and therefore useable.

The 'Sequence Functions Menu' controls the sequence of snippets, i.e., the sequence of machine tasks. Thus, it allows you to perform the central functions start, stop, pause, repeat, skip, go back, emergency stop, and restart. Some of those functions are only visible depending on the state of the sequence, e.g., pause is only visible if the sequence has been started.

Furthermore, it allows you to visualize the snippets by simulating what would happen if you were to start the sequence using the digital twin of the robot. Additionally, the visualization of constraints can be defined by you in the constraint template class (see chapter 7 'Creation of Constraints' for more details).
A potential visualization option would be to use spatial awareness to highlight no-go zones like a table or walls for a 'NoGo Constraint'.

3.5 ROS Status Menu

The 'ROS Status Menu' is activated if the ROS connection was enabled in the 'ROSController' GameObject. It allows you to display status messages received by ROS by subscribing to 'DiagnosticStatusMsg' ROS messages in the ROS topic 'tpi_diagnostic_status'.
Those status messages can be grouped into the general categories 'ok', 'warning', and 'error', each displayed with a different image in the status menu.

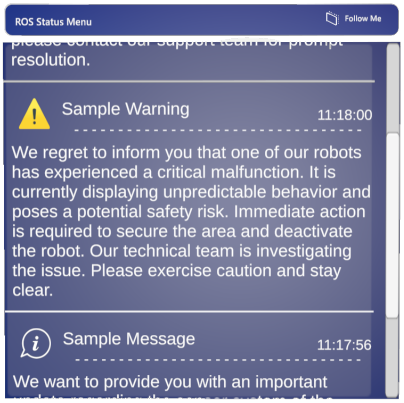


Figure 3.5: Depiction of the ros status menu

3.6 Workflow Hand Menu

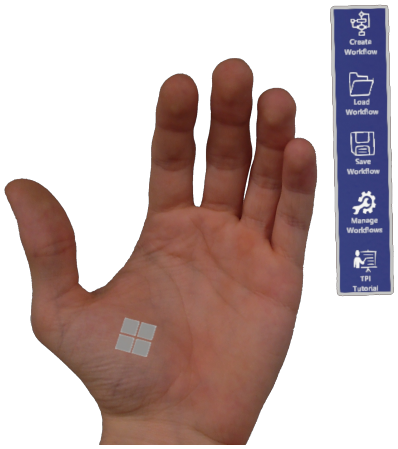


Figure 3.6: Depiction of the workflow hand menu

The 'Workflow Hand Menu' is usually the first thing the operator will see, located on the left-hand palm's ulnar side.
It controls the most general functions of the TPI, i.e., creating, loading, saving, or managing a workflow or starting the tutorial.
Once a workflow has been loaded or created, the 'Create Workflow' button changes into a button, which can be used to toggle the visibility of the whole TPI.
Furthermore, once the tutorial has been started, the 'Start Tutorial' button can be used again to stop it.

3.7 Object Placement Hand Menu

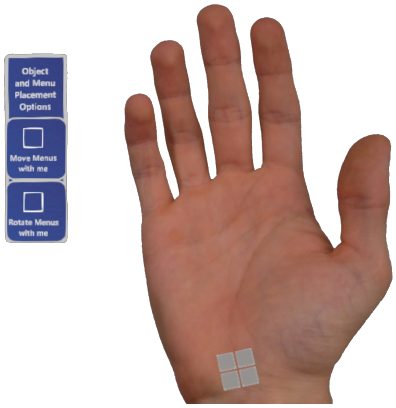


Figure 3.7: Depiction of the object placement hand menu

The 'Object Placement Hand Menu' on the ulnar side of the right-hand palm controls the central function of changing the position of the menus and objects attached to the object placement grid.
It displays a button, which can be used to attach the grid to the operator, therefore letting the grid move, but not rotate, with the operator. It also has a second button, which can be used to rotate the grid alongside the operator, but not move it. Both buttons can be activated at the same time.

3.8 Dialog Menus

One of the main features of the TPI is the ability to create and customize dialog menus at runtime, them being used to display key information or to give the operator the choice to configure pre-determined options. This can be done with the help of the 'TPI_DialogMenuController' component of the 'TPI_Manager' GameObject.

By default, dialog menus can contain the following features:

- Menu name
- Text fields
- Buttons
- Keyboard input fields

- Checkboxes
- Toggles
- Dropdown fields
- Point selection fields

If desired, more features can be added by extending the initial version of the 'TPI_DialogMenuController' script. Depending on the functionality, you would need to add a way to store the information, which is needed to correctly populate the dialog menu fields (easily achieved by adding a new class like the 'TPI_DialogMenuButton' class). Furthermore, you must add a section to the 'InstantiateDialogMenu' function to populate it. If the choice must be retrievable, add a section to the 'GetDialogMenuChoices' function.

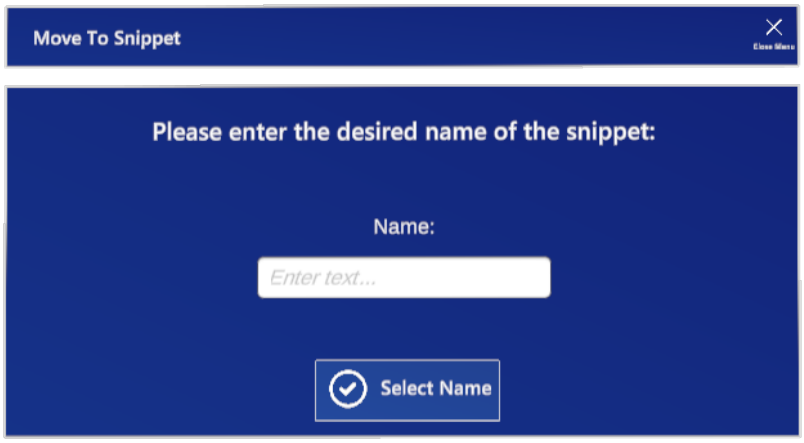


Figure 3.8: The figure shows a sample 'Dialog Menu', in this case, the one which asks for a name for a snippet of template 'Move To'. To top-right corner contains a button to close the menu. If the 'Select Name' button is pressed without setting a name, another dialog menu is opened that shows an error.

An extensive tutorial on how to add new templates, also called prefabs, how to create the dialog menus, how to delete dialog menus, and on how to retrieve the choices made in the dialog menu can be accessed in the 'TPI_DialogMenuController' script or in chapter 8 'Creation of a Dialog Menu'.

Finally, every dialog menu is automatically added to the object placement grid once it gets created.

4 Saving and Loading using the SaveController

The TPI contains a system that allows you to save and load workflows containing the snippets and constraints you have created. Furthermore, variables of said snippets and constraints are also saved, therefore radically reducing the effort for you.

The following informational bullet points highlight some do's and don'ts when working with the SaveController alongside mentioning some important advice:

- Suppose you rename the class of a snippet or constraint or their underlying subclasses, i.e., the 'SaveData' class. In that case, you must delete all workflows, snippets, and constraints containing said classes, as it prevents the 'SaveController' from loading it. Therefore, it is strongly suggested to pick a class name at the start and only rename it if necessary.
- As the content of the 'PersistenDataPath', a path to your 'LocalLow' folder in the 'AppData' directory that always stays the same, could get deleted once you redeploy your project to the HoloLens, the save folder of the TPI has been moved to the documents folder. Therefore, please follow the instructions in chapter 14 'Deployment' of this documentation on how to allocate the necessary permissions, as saving and loading would not work otherwise.

As mentioned before, the variables of your snippets and constraints can be saved alongside the internal information needed for the TPI to work. This can be done by creating a 'SaveData' class for each of your snippet and constraint templates, e.g., 'MoveTo_SaveData', which contains the variables that should be saved.

However, there are a few important points concerning them to be aware of:

- Not all variable types are supported. GameObjects, Dictionaries, and Components, for example, cannot be serialized properly. Please refer to the 'JSON Serialization Unity Documentation' for more information.
- To use lists inside a list, which under normal conditions is not supported by the JSON serialization, there is a workaround that you can do. Suppose you want to save the following: List<List<string>>. You can create a new class, e.g., with the name 'TestClass', that only contains a variable of type List<string>. This way, it is possible to have a variable of type List<TestClass> inside your SaveData class, which will be saved correctly.

The SaveController can switch between two modes: creating an entirely new workflow save and overwriting an existing workflow save.

5 Working with Assets

If you would like to add new assets like textures or sprites to your project, you are encouraged to do so. However, to permit the TPI to work correctly, it is necessary to make specific settings for some of those asset types.

If you would like to add new icons as 'Texture2D' for your snippets or constraints to your Unity project, please enable 'Read/Write' under 'Advanced' and set 'Compression' to 'None'. Furthermore, please only add icons of type 'png', as others could lead to errors.

The TPI already includes an extensive database of icons in black and white, some also in color, which you can use for your own buttons, snippets, constraints, etc. Those can be found in the 'TaskPlanningInterface' directory in the 'Textures/Icons/' folder.

6 Creation of a Snippet

Snippet templates, which will be configured during runtime by the operator, can easily be created with the help of two classes: 'TPI_Snippet' and 'TPI_SnippetSaveData'. The first handles the general setup of a snippet, and the second handles the saving and loading of the snippet-related data.

To make it easier for you, the TPI already includes a template script called 'TPI_SnippetFunctionTemplate' in the 'UseCases' folder of the 'TaskPlanningInterface' directory, which can easily be copied to have the general setup for your new snippet. Simply rename the file name, the class 'TPI_SnippetFunctionTemplate' and the 'TPI_SnippetFunctionTemplate_SaveData', the latter two being found inside the file, after copying it into a folder of your own making.

The template file contains a simple implementation of a feature that allows the operator to choose a name for the snippet after the snippet template was selected inside the 'Building Blocks Menu'.

It is important that your class, which derives from 'TPI_Snippet' contains the function 'UpdateSaveData()', which sets up the 'SaveData' class to permit successful saving. To work correctly, this function must include the code sample 6.1.

Code 6.1: Required code for the 'TPI_SnippetSaveData' class.

```
TPI_SnippetSaveData snippetSaveData = saveData;
if (snippetSaveData == null){
    // Replace "TPI_SnippetFunctionTemplate_SaveData" with your own class
    that derives from "TPI_SnippetSaveData"
    snippetSaveData =
        new TPI_SnippetFunctionTemplate_SaveData();
}
snippetSaveData.saveManager_functionObjectName =
    snippetInformation.functionObject.name;
snippetSaveData.information_snippetName =
    snippetInformation.snippetName;
snippetSaveData.information_snippetTemplateName =
    snippetInformation.snippetTemplateName;
snippetSaveData.information_snippetIcon =
    saveController.ConvertTextureToBytes(snippetInformation.snippetIcon);
snippetSaveData.information_snippetID =
    snippetInformation.snippetID;
this.saveData = snippetSaveData;

/*

Assign values here to the variables that you have created in
your own SaveData class that derives from TPI_SnippetSaveData

*/
```

IMPORTANT: Not all variable types are supported. Please refer to chapter 4 'Saving and Loading using the SaveController' for more information.

Furthermore, your snippet template class must contain the following functions, which support the general functioning:

ButtonPressed ()	This function should implement what happens when the button is pressed in the building blocks menu.
ChangeVariables ()	This function gets called when the operator double-clicks on a snippet in the sequence menu to change the variables' values.
SetupControllerReferences ()	This function sets up the references to the Task Planning Interface Controllers you will use most.
UpdateSaveData ()	This function correctly sets up and updates your save data script.

Additionally, your snippet template class can contain the functions from the following table, which controls what happens during different sequence events. Mandatory functions are red, and optional functions remain uncolored.

OnEmergencyStop ()	This function handles what happens if the operator signals with his hands to stop due to an emergency abruptly.
OnHasEnded ()	This function handles what happens when your snippet function has ended, i.e., when the sequence progress in the sequence menu reaches the next snippet.
RepeatSnippet ()	The operator indicated that he intends to let the snippet run again once the current iteration of the function has ended.
RunSnippet ()	This function handles what happens when the snippet is started, i.e., when the sequence progress in the sequence menu reaches your snippet. As a first action of the RunSnippet() function, please move to the starting position.
SkipSnippet ()	This function handles what happens when the operator wants to manually stop the snippet and go to the next one mid-execution.
StopSnippet ()	This function handles what happens when the operator wants to stop the sequence mid-execution manually.

Finally, as 'Unity Coroutines' are quite limited in some ways, it is not possible to pause the 'Coroutine' of the whole sequence in its entirety. Therefore, to still allow pauses, there is a simple workaround. Add the code sample 6.2 at those points inside your 'RunSnippet();' IEnumerator function, at which the 'Coroutine' should pause:

Code 6.2: Sample code to add pause points to your snippet function.

```
// Add the following code at the points where the TPI should pause if the
    'Pause Snippet' Button got pressed.
yield return new WaitUntil(() => TPI_SequenceMenuController.isPaused ==
    false);
```

After setting up both classes that derive from 'TPI_Snippet' and 'TPI_SnippetSaveData', it is time to register it in the TPI. Therefore, please create a new GameObject in Unity (e.g., right-click in the 'Hierarchy' + select 'Create Empty') and then add your script that contains the class, which derives from 'TPI_Snippet' as a component (e.g., click on the empty GameObject and select 'Add Component'). Furthermore, please rename the GameObject.

If this is the first time you are creating a snippet or constraint, please create a new folder '/Resources/TaskPlanningInterface/UseCases/', i.e., a folder called 'UseCases' in a folder called 'TaskPlanningInterface', which in turn is located in a folder called 'Resources'. Otherwise, you can skip this.

Now, create a new prefab of your GameObject by moving it inside the previously created folder '/Resources/TaskPlanningInterface/UseCases/'. After that, you can delete the instance of the GameObject in the 'Hierarchy', as it is no longer needed.

Finally, you can register it in the 'TPI_WorkflowConfigurationController' component on the 'TPI_Manager' GameObject. Add a new entry to the 'Snippet Templates' list and fill in the required information. The 'Function Script Prefab' refers to the just created prefab of the GameObject.

7 Creation of a Constraint

Constraint templates, which will be configured during runtime by the operator, can easily be created with the help of two classes: 'TPI_Constraint' and 'TPI_ConstraintSaveData'. The first one handles the general setup of a constraint, and the second handles the saving and loading of the constraint-related data.

To make it easier for you, the TPI already includes a template script called 'TPI_ConstraintFunctionTemplate' in the 'UseCases' folder of the 'TaskPlanningInterface' directory, which can easily be copied in order to have the general setup for your new constraint. Rename the file name, the class 'TPI_ConstraintFunctionTemplate' and the 'TPI_ConstraintFunctionTemplate_SaveData', the latter two being found inside the file, after copying it into a folder of your own making.

The template file contains a simple implementation of two features, which allow the operator to choose a name and constraint type (whether it is globally active or snippet-specific) for the constraint after the constraint template was selected inside the 'Building Blocks Menu'.

It is important that your class, which derives from 'TPI_Constraint' contains the function 'UpdateSaveData()', which sets up the 'SaveData' class in order to permit successful saving. To work correctly, this function must include the code sample 7.1.

Code 7.1: Required code for the 'TPI_ConstraintSaveData' class.

```
TPI_ConstraintSaveData constraintSaveData = saveData;
if(saveData == null){
    // Replace "TPI_ConstraintFunctionTemplate_SaveData" with your own
    class that derives from "TPI_ConstraintSaveData"
    constraintSaveData =
        new TPI_ConstraintFunctionTemplate_SaveData();
}
```

```
constraintSaveData.saveManager_functionObjectName =
    constraintInformation.functionObject.name;
constraintSaveData.information_constraintName =
    constraintInformation.constraintName;
constraintSaveData.information_constraintTemplateName =
    constraintInformation.constraintTemplateName;
constraintSaveData.information_constraintType =
    constraintInformation.constraintType;
constraintSaveData.information_constraintIcon =
    saveController.ConvertTextureToBytes(constraintInformation.
    constraintIcon);
constraintSaveData.information_snippetID =
    constraintInformation.snippetID;
constraintSaveData.information_constraintID =
    constraintInformation.constraintID;
this.saveData = constraintSaveData;

/*

Assign values here to the variables that you have
created in your own SaveData class that derives
from TPI_ConstraintSaveData

*/
```

IMPORTANT: Not all variable types are supported. Please refer to chapter 4 'Saving and Loading using the SaveController' for more information.

Furthermore, your constraint template class must contain the following functions, which support the general functioning:

ButtonPressed ()	This function should implement what happens when the button is pressed in the building blocks menu.
ChangeVariables ()	This function gets called when the operator clicks on a Constraint in the Sequence Menu to change the variables' values.
SetupControllerReferences ()	This function sets up the references to the Task Planning Interface Controllers you will use most.
StopVisualization ()	This function handles what happens when the operator no longer wants to visualize the constraint.
UpdateSaveData ()	This function correctly sets up and updates your save data script.
VisualizeConstraint ()	This function handles what happens when the operator wants to visualize the constraint.

Additionally, your constraint template class must contain the following functions, which are needed to control the functionality of the constraint:

ApplyConstraint ()	This function handles what happens when the constraint starts to be applied,
StopConstraint ()	This function handles what happens when the operator either wants to stop the snippet sequence or when the snippet has ended, and the snippet-specific constraint thus should no longer be applied

After setting up both classes that derive from 'TPI_Constraint' and 'TPI_ConstraintSaveData', it is time to register it in the TPI. Therefore, please create a new GameObject in Unity (e.g., right-click in the 'Hierarchy' + select 'Create Empty') and then add your script that contains the class, which derives from 'TPI_Constraint' as a component (e.g., click on the empty GameObject and select 'Add Component'). Furthermore, please rename the GameObject.

If this is the first time you are creating a snippet or constraint, please create a new folder '/Resources/TaskPlanningInterface/UseCases/', i.e., a folder called 'UseCases' in a folder called 'TaskPlanningInterface', which in turn is located in a folder called 'Resources'. Otherwise, you can skip this.

Now, create a new prefab of your GameObject by moving it inside the previously created folder '/Resources/TaskPlanningInterface/UseCases/'. After that, you can delete the instance of the GameObject in the 'Hierarchy', as it is no longer needed.

Finally, you can register it in the 'TPI_WorkflowConfigurationController' component on the 'TPI_Manager' GameObject. Add a new entry to the 'Constraint Templates' list and fill in the required information. The 'Function Script Prefab' refers to the just created prefab of the GameObject.

8 Creation of a Dialog Menu

The 'TPI_DialogMenuController' helps you to easily create complex and good-looking 'Dialog Menus' (see section 3.8) to your liking. All you need to create them is a reference to the 'TPI_DialogMenuController' component located on the 'TPI_Manager' GameObject. You can for example achieve this by using the following code sample:

```
TPI_DialogMenuController dialogMenuController =
    GameObject.FindGameObjectWithTag("TPI_Manager").GetComponent<
    TPI_DialogMenuController>();
```

This tutorial can also be found in the 'TPI_DialogMenuController' script. Furthermore, the script also contains a summary of all functions with a description of all their input parameters and their potential return value.

For a quick overview of all the functions and variables available to you, visit section D.1 of appendix D 'Overview of the TPI Classes'.

A) *Creation and Spawning of a Dialog Menu:*

1. 'Inspector + using the functions given in this script'

   The easiest way to create a dialog menu is by creating and filling out a 'Dialog Menu Template' in the 'TPI_DialogMenuController' component of the 'TPI_Manager' GameObject in the Unity inspector.
   This allows you to easily decide with the help of a graphical interface how your dialog menu should be populated.

   Once you have set up the template via the Unity inspector, you can call either the 'SpawnDialogMenu(string dialogMenuID)' function or the 'SpawnDialogMenu(int dialogMenuIndex)' function at the right

place in your code in order to create the dialog menu.
The 'dialogMenuID' or 'dialogMenuIndex' can be found in the 'Dialog Menu Template' list, by either determining the index of your template entry (starting at 0) or by simply copying the content of the 'Dialog Menu ID' field for the 'dialogMenuID' (right-click + Copy).

2. 'Coding all the way'

   If you do not want to use the Unity inspector, you have to create a new 'TPI_DialogMenuInformation' instance and assign every value yourself. Furthermore, you then have to use the function 'SpawnDialog-Menu(TPI_DialogMenuInformation dialogMenuInformation)' to create your menu.

   An example of how to create such a 'TPI_DialogMenuInformation' instance, showing all the possible options you can utilize, can be seen in code sample 8.1.

Code 8.1: Sample code to create a dialog menu.

```
// Set the menuPrefab in the inspector or assigned it otherwise
GameObject menuPrefab;
// OPTIONAL: Set up an icon for the button in the inspector or assigned it
    otherwise
Texture2D buttonIcon;

// Set up the general information
TPI_DialogMenuInformation dialogMenuInfo =
    new TPI_DialogMenuInformation();
dialogMenuInfo.dialogMenuName = "Put the menu name here. It will replace
    the field [Name]";
dialogMenuInfo.dialogMenuPrefab = menuPrefab;
// Decide if there should be a "Close Menu" Button in the title bar
dialogMenuInfo.showCloseMenuButton = true;

// Add text fields
dialogMenuInfo.dialogMenuTexts.Add("Put the first text here. It will
    replace the field [Textfield0]");
dialogMenuInfo.dialogMenuTexts.Add("Put the second text here. It will
    replace the field [Textfield1]");

// Add a button
TPI_DialogMenuButton button1 = new TPI_DialogMenuButton();
```

```
button1.buttonText = "Put the button name here. It will replace the field
    [Button0]";
// OPTIONAL: icon visible on the button
button1.buttonIcon = buttonIcon;
// What should happen if the button is clicked?
button1.buttonOnClick.AddListener(delegate { PutYourFunctionHere(); });
dialogMenuInfo.dialogMenuButtons.Add(button1);

// Add a keyboard input field
dialogMenuInfo.keyboardInputFieldTitles.Add("Put the title of the keyboard
     input field here. It will replace the field [KeyboardInput0]");

// Add a checkbox field
dialogMenuInfo.checkboxTitles.Add("Put the title of the checkbox field
    here. It will replace the field [Checkbox0]");

// Add a toggle field
dialogMenuInfo.toggleTitles.Add("Put the title of the toggle field here.
    It will replace the field [Toggle0]");

// Add a dropdown field
// Create the correct List of options for the Dropdown
// Set the dropdown options up in inspector or assign them otherwise
List<string> dropdownOptions;
List<TPI_DialogMenuDropDown.OptionData> dropdownList =
    new List<TPI_DialogMenuDropDown.OptionData>();
// Create the correct list of options
for (int i = 0; i < dropdownOptions.Count; i++) {
    // instead of null you can also assign a Texture2D
    dropdownList.Add(new TPI_DialogMenuDropDown.OptionData(dropdownOptions
    [i], null));
}
dialogMenuInfo.dialogMenuDropdowns.Add(new TPI_DialogMenuDropDown("Put the
     dropdown title here. It will replace the field [Dropdown0]",
    dropdownOptions));

// Add a point selection field
// instead of null you can also assign a Texture2D
dialogMenuInfo.pointSelections.Add(new TPI_DialogMenuPointSelection("Put
    the title of the point selection field here. It will replace the field
     [PointSelection0]", null));

// Create the Dialog Menu
GameObject.FindGameObjectWithTag("TPI_Manager").GetComponent<
    TPI_DialogMenuController>().SpawnDialogMenu(dialogMenuInfo);
```

IMPORTANT: Generally speaking, when planning on having multiple dialog menus in a sequence, them being connected by the click on a button, please first delete the old dialog menu before you spawn in the new one as problems with the 'ObjectPlacementController' otherwise might arise.

B) *Retrieval of the Choices made in a Dialog Menu:*

The 'TPI_DialogMenuController' makes it extremely easy for you to retrieve the choices made by the operator, as you only have to know the 'dialogMenuID' of the dialog menu you want to get the choices of.
This can then be achieved by performing the code from code sample 8.2 BEFORE you delete the dialog menu:

Code 8.2: Sample code to retrieve the dialog menu choices.

```
TPI_DialogMenuChoices choices = GameObject.FindGameObjectWithTag("
    TPI_Manager").GetComponent<TPI_DialogMenuController>().
    GetDialogMenuChoices(dialogMenuID);
```

Then, you can extract the desired information from each option (e.g., keyboard input field) by using the code in code sample 8.3.
Short reminder: the identifier for e.g., '[KeyboardInput0]' would be 0, and the one of '[KeyboardInput3]' would be 3!

Code 8.3: Sample code to extract the values from the dialog menu choices.

```
choices.keyboardTexts[index] // returns the string of the text belonging
    to the keyboard input field with the identifier "index"

choices.checkboxes[index] // returns the bool belonging to the checkbox
    field with the identifier "index"

choices.toggles[index] // returns the bool belonging to the toggle field
    with the identifier "index"

choices.dropdown[index] // returns the index of the selected option
    belonging to the dropdown field with the identifier "index"

choices.selectedPoints[index] // returns the Vector3 of the position
    belonging to the point selection field with the identifier "index"
```

C) *Deletion of a Dialog Menu:*

1. 'Coding all the way':

   There are two ways how you can delete a dialog menu by code:

   - First of all, you can save the dialogMenuID when you spawn the dialog menu and then plug it into the 'UnSpawnDialogMenu(string dialogMenuID)' function. The dialogMenuID can be obtained by using the following line: 'dialogMenuInfo.dialogMenuID;'

   - Otherwise, you could also save the 'TPI_DialogMenuInformation' instance and then plug it into the 'UnSpawnDialogMenu(TPI_DialogMenuInformation dialogMenuInformation)' function once you want to delete the dialog menu.

2. 'Inspector + using the functions given in this script':

   If you want to delete a dialog menu, which was registered in the 'Dialog Menu Template' list in the Unity inspector, you can achieve this by looking up the respective 'dialogMenuID' and plugging it into to following function: 'UnSpawnDialogMenu(string dialogMenuID)'
   You can right-click and select 'Copy' while hovering over the 'Dialog Menu ID' field to obtain it.

3. Deleting the last element added:

   To delete the last element added, you can call the 'UnSpawnLastDialogMenu()' function.

4. Delete all dialog menus:

   If you want to delete all active dialog menus simultaneously (usually, you have just one open), then you can call the 'ClearDialogMenus()' function.

D) *Creation of Dialog Menu Prefabs:*

Finally, to set up a new dialog menu prefab (the GameObject that is instantiated and then populated with the correct information), you can either start from scratch or copy and adapt an already existing one provided to you by the TPI. Keep in mind, it is always good to name your dialog menu prefabs in a way that makes it instantaneously clear what that template offers or where it is used, e.g., you could name a prefab with two text fields, a button, and a keyboard input field: 'DMP: 2TXT 1BUT 1KIF Variation1', where 'DMP' refers to 'DialogMenuPrefab', '2TXT' refers to '2 text fields', '1BUT' refers to '1 button' and '1KIF' refers to '1 keyboard input field'. Finally, if you have multiple versions of the same prefab, each looking a bit different but containing the same elements, then it might make sense to also add a 'Variation1' tag after it to signal that.

IMPORTANT: The TPI can only guarantee that the population of information works with the provided 'Dialog Menu Building Blocks' prefabs found in the 'Prefabs' folder of the TaskPlanningInterface directory.
Short reminder: the identifier for '[KeyboardInput0]' would be 0, and the one for '[KeyboardInput3]' would be 3!

NAME: To specify the place where the name of the dialog menu should be located, move a text field template to your desired position and set the displayed text to '[Name]' without the apostrophes.

TEXT FIELD: To specify where a text field should be located, move a text field prefab to your desired position and set the displayed text to the correct identifier, e.g., '[Textfield0]' without the apostrophes.

BUTTON: To specify where a button should be located, move a button prefab to your desired position and set the displayed text to the correct identifier, e.g., '[Button0]' without the apostrophes.

KEYBOARD INPUT FIELD: To specify the place where a keyboard input field should be located, move a keyboard input field prefab to your desired position and set the displayed text to the correct identifier, e.g., '[KeyboardInput0]' without the apostrophes.

CHECKBOX: To specify where a checkbox field should be located, move a checkbox prefab to your desired position and set the displayed text to the correct identifier, e.g., '[Checkbox0]' without the apostrophes.

TOGGLE: To specify where a toggle field should be located, move a toggle prefab to your desired position and set the displayed text to the correct identifier, e.g., '[Toggle0]' without the apostrophes.

DROPDOWN: To specify where a dropdown selection field should be located, move a dropdown prefab to your desired position and set the displayed text to the correct identifier, e.g., '[Dropdown0]' without the apostrophes.

POINT SELECTION: To specify where a point selection field should be located, move a point selection prefab to your desired position and set the displayed text to the correct identifier, e.g., '[PointSelection0]' without the apostrophes.

E) *Pre-Made Functions:*

Multiple functions are already implemented in the 'TPI_DialogMenuController' class, which will be used a lot to configure a snippet or constraint. Therefore, this should make it easier and faster for you, preventing repetition and errors.

ShowConstraintTypeMenu	Creates a Dialog Menu that allows the operator to choose the type of a Constraint (whether it is globally active or snippet-specific)
ShowDropdownSelectionMenu	Creates a Dialog Menu that allows the operator to select an item from a list of strings (styled as a dropdown).
ShowErrorMenu	Creates a Dialog Menu that points out an error to the operator or provides the operator with information.
ShowErrorMenu_TwoButtons	Creates a version of the 'ShowErrorMenu' Dialog Menu containing two buttons. This can be useful to give the operator a choice between two options.
ShowObjectNameMenu	Creates a Dialog Menu that allows the operator to select a Name for an Object (e.g., Snippet or Constraint)

For a more detailed description, especially of their parameters, please visit the implementations in the 'TPI_DialogMenuController script.

9 ObjectPlacementController

The TPI contains a feature that automatically strategically places menus and interfaces in the environment around the operator. It achieves this by overlaying a cylindrical-shaped MRTK 'GridObjectCollection' grid over the viewport of the operator.

The 'TPI_ObjectPlacementController', found as a component on the 'ObjectPlacementController' GameObject, allows you to set the number of free spots and the number of columns, automatically calculating the number of rows for you. To further customize it, you can determine the width and height of an element inside the grid.

With the help of the 'Object Placement Hand Menu' (ulnar side of right-hand palm), the operator can move and rotate the grid alongside the viewport. For a more detailed view of said hand menu, please either visit the 'TPI_ObjectPlacementController' script or section 3.7 'Object Placement Hand Menu'.

You can use different algorithms with different inputs to determine a free position in the grid. The available search algorithms include:

- Horizontally
- Vertically
- Closest Position

You can also determine whether there should be a preferred search direction for the horizontal and vertical search directions. The available search directions include: 'both ways', 'Only Upwards or Leftwards', and 'Only Downwards or Rightwards'.

Finally, you have the option to choose a specific starting position (applicable for each search algorithm), which is visualized in the following table:

Table 9.1: Overview of the available starting positions for the ObjectPlacementController.

Upper Left	Upper Center	Upper Right
Middle Left	Middle Center	Middle Right
Bottom Left	Bottom Center	Bottom Right

Spots inside the grid can be managed with the help of the following functions:

ClearAllSpots ()	Frees up all the spots in the grid.
FindAndReservePosition	Finds a spot that fits best to the provided anchor point (and search algorithm and search direction) and reserves it for the GameObject, returning the position and rotation of said spot.
FindUnoccupiedSpot	Finds a suitable unoccupied spot according to your anchor point, search algorithm, and search direction and returns the index.
FreeUpSpot	Frees up a spot in the grid by providing the TPI_PositionAndRotation of said spot.
ReserveSpot	Reserves a spot for a GameObject by providing the TPI_PositionAndRotation of the desired spot.

IMPORTANT: The functions 'FindAndReservePosition' and 'ReserveSpot' do not automatically set the position and rotation of your GameObject to the position and rotation of the free spot. You can either do this yourself, or you can set the optional parameter 'applyPose' to 'true' (e.g., add 'applyPose:true' without the apostrophes to the function's parameters).

For a full list of public variables and public functions of the 'TPI_ObjectPlacementController', please visit section D.4 of the appendix.

10 ROS Controller

The TPI uses the 'Unity Robotics Hub' [9] to connect to ROS and send and receive ROS messages. As the code of the 'Unity Robotics Hub' does not contain any documentation and as it is not possible to find good documentation on their website, the TPI contains documentation for all public functions of the 'ROSConnection' script of the 'Unity Robotics Hub', also including some sample codes for the most important features like publishing, subscribing and services. This documentation can be found in the 'TPI_ROSController' component located on the 'ROSController' GameObject.

Furthermore, the 'TPI_ROSController' comes with some ready-to-use functions, which can be directly used in any of your scripts.

An overview of all the public functions of the 'ROSConnection' script, including the ready-to-use functions of the TPI, is accessible in section D.6.

IMPORTANT: As the coordinate systems of ROS and Unity are not the same, it is important to translate between the coordinate systems directly before publishing it or after receiving it via the topic subscription. This makes it much easier on the ROS side, as one does not have to care about the correct transformations anymore [10].
To translate the Unity coordinates into ROS coordinates, you can use the '***To<FLU>()***' function, as can be seen in the code sample 10.1.

Code 10.1: Sample code to convert coordinates from Unity to ROS.

```
// Convert Unity coordinates into the ROS coordinate system
Vector3<FLU> rosPositon = desiredPosition.To<FLU>();

// Create a new PointMsg from the coordinates
PointMsg pointMsg =
    new PointMsg(rosPositon.x, rosPositon.y, rosPositon.z);
```

On the other hand, to get from the ROS coordinates to Unity coordinates, one can use the '***To<RUF>()***' function as is shown in the code sample 10.2.

Code 10.2: Sample code to convert coordinates from ROS to Unity.

```
Vector3 unityPosition = (Vector3)new Vector3<FLU>(
    (float)poseMsg.position.x,
    (float)poseMsg.position.y,
    (float)poseMsg.position.z).To<RUF>();

Quaternion unityRotation = (Quaternion)new Quaternion<FLU>(
    (float)poseMsg.orientation.x,
    (float)poseMsg.orientation.y,
    (float)poseMsg.orientation.z,
    (float)poseMsg.orientation.w).To<RUF>();
```

Both functions work for Vector3 positions and for Quaternion rotations.

Furthermore, it might make sense for you to translate the global coordinates of a specific point in the Unity environment into the relative local coordinates of the base of your robot (usually called link0). Especially if you are working with a robotic arm, this makes it much easier, as you thus can calculate the different joint angles of each joint link without caring about the correct coordinate system translations. This can be achieved by calling the '***GetRelativePose(Vector3 initialPosition, Quaternion initialRotation)***' function in the 'TPI_RobotController' script, accessible with the 'robotURDF' variable from the 'TPI_MainController' component located on the 'TPI_Manager'. An example of this can be seen in code sample 10.3.

Code 10.3: Sample code to get the coordinates relative to the base link.

```
PoseMsg msg =
 mainController.robotURDF.GetComponent<TPI_RobotController>().
   GetRelativePose(desiredPosition, desiredRotation);
```

Finally, if the ROS connection is enabled at startup, the 'ROS Status Menu', described in section 3.5, will automatically be activated. It is used to display status messages of type 'ok', 'warning', and 'error' directly in the mixed reality environment on your glasses.

11 Robot Visualization

The TPI uses the 'URDF Importer' [11] to import and display a digital twin of a robot in the Unity environment. By default, the FRANKA RESEARCH 3 robotic arm is imported and correctly set up for use, as it was used during the development of the TPI. However, the TPI framework was also designed to work with other robots and robotic arms.

Furthermore, to steer the digital twin visible in the mixed reality environment, the TPI includes the 'TPI_RobotController' script, which subscribes to the joint angles published by ROS and automatically sets the robot's joint links to the correct angles. This, again, was initially set up for the FRANKA RESEARCH 3 robotic arm but can easily be adapted to work with others.

If desired, there is also the possibility of sending and receiving the location of the base link of the robotic arm, which can be configured in the Unity Inspector of the 'TPI_RobotController' component of the 'robotArm_FrankaEmika_Panda' GameObject.

Especially when trying to calibrate the base link of the robotic twin, it might make sense to use the 'TPI_Photo' script to take a picture with the HoloLens webcam and send it to ROS (see section D.13 for more details). This allows the detection of 'ArUco markers' in the captured picture (done on the ROS side), after which the position of said marker can be sent back to Unity to move the base link position.

12 Creation of a Tutorial

The 'TPI_TutorialController' script helps you to easily create a functioning tutorial for the operator, consisting of multiple steps in a sequence.
The operator can start said tutorial by pressing the "Start Tutorial" button in the 'Workflow HandMenu'.

A slightly modified version of this section can also be found in the 'TPI_TutorialController' script.

A) *How it works:*

1. The easiest way to create the tutorial is by filling in the 'Tutorial Steps' List in the Inspector of the 'TPI_TutorialController' component, located on the 'TPI_Manager' GameObject. Otherwise, you can also add additional tutorial dialogs at runtime (see subsection B: 'Creation of new Tutorial Steps at Runtime' for a detailed view).

2. Once the tutorial has started, it will automatically instantiate the prefab, which belongs to the first tutorial step, and invoke all the function(s) that were added to the 'Tutorial Start Event' field.

3. Then, it waits for the 'TriggerNextTutorialStep(string tutorialID)' function with the correct tutorialID of the first step to be invoked somewhere in your code.

4. Once this happens, it destroys the previous tutorial prefab, invokes the function(s) that were added to the 'Tutorial End Event' field, and starts the next tutorial step, looping through this process until the end of the list of tutorial steps is reached.

To sum up:

Please add the 'TriggerNextTutorialStep(string tutorialID)' function at the correct spot in your code where you want the 'TPI_TutorialController' to proceed to the next tutorial step.
You can get the tutorialID by either copying it (right-click + Copy) from the Unity inspector or by extracting it from your tutorial dialog that was created at runtime (see subsection B: 'Creation of new Tutorial Steps at Runtime').

B) *Creation of new Tutorial Steps at Runtime:*

New tutorial steps can easily be created by using the code illustrated in code sample 12.1.

Code 12.1: Sample code to create a tutorial step.

```
// Assign the Prefab in the Inspector or by other means
GameObject tutorialPrefab;

// Create a new TutorialDialog instance
TPI_TutorialController.TutorialDialog myTutorialStep = new TutorialDialog(
    "Insert Tutorial Title", "Insert Tutorial Text", tutorialPrefab, new
    UnityEvent());

// Configure the Action that should be invoked once the Tutorial Step is
    started
myTutorialStep.tutorialStartEvent.AddListener(AddYourFunctionNames);

// Configure the Action that should be invoked once the Tutorial Step has
    ended (immediately before the next one starts)
myTutorialStep.tutorialEndEvent.AddListener(AddYourFunctionNames);

// Extract the ID of the Tutorial Step
string tutorialID = myTutorialStep.tutorialID;

TPI_TutorialController tutorialController = GameObject.
    FindGameObjectWithTag("TPI_Manager").GetComponent<
    TPI_TutorialController>().AddTutorialStep(myTutorialStep);
```

C) *Creation of Tutorial Prefabs:*

Finally, we discuss how you can set up a tutorial prefab (the GameObject that is instantiated). You can either start from scratch or copy and then adapt an already existing one provided to you by the TPI. Keep in mind, it is always good to name your tutorial prefabs in a way that makes it instantaneously clear what that template offers or where it is used.

As the Tutorial Prefabs do not differ that much from the 'Dialog Menu' system implemented in the 'TPI_DialogMenuController', it is advised to use the 'Dialog Menu Building Blocks' for this process, them being located in the 'Prefabs' folder of the 'TaskPlanningInterface' directory.

NAME: To specify where the tutorial's name should be located, move a text field template to your desired position and set the displayed text to '[Name]' without the apostrophes.

TEXT FIELD: To specify where the text field should be located, move a text field prefab to your desired position and set the displayed text to '[Textfield]' without the apostrophes.

Future Extension:

A future extension of this feature could involve the ability to create and choose between different tutorials. Currently, it is only possible to create one tutorial, which can hold as many tutorial steps as you would like.

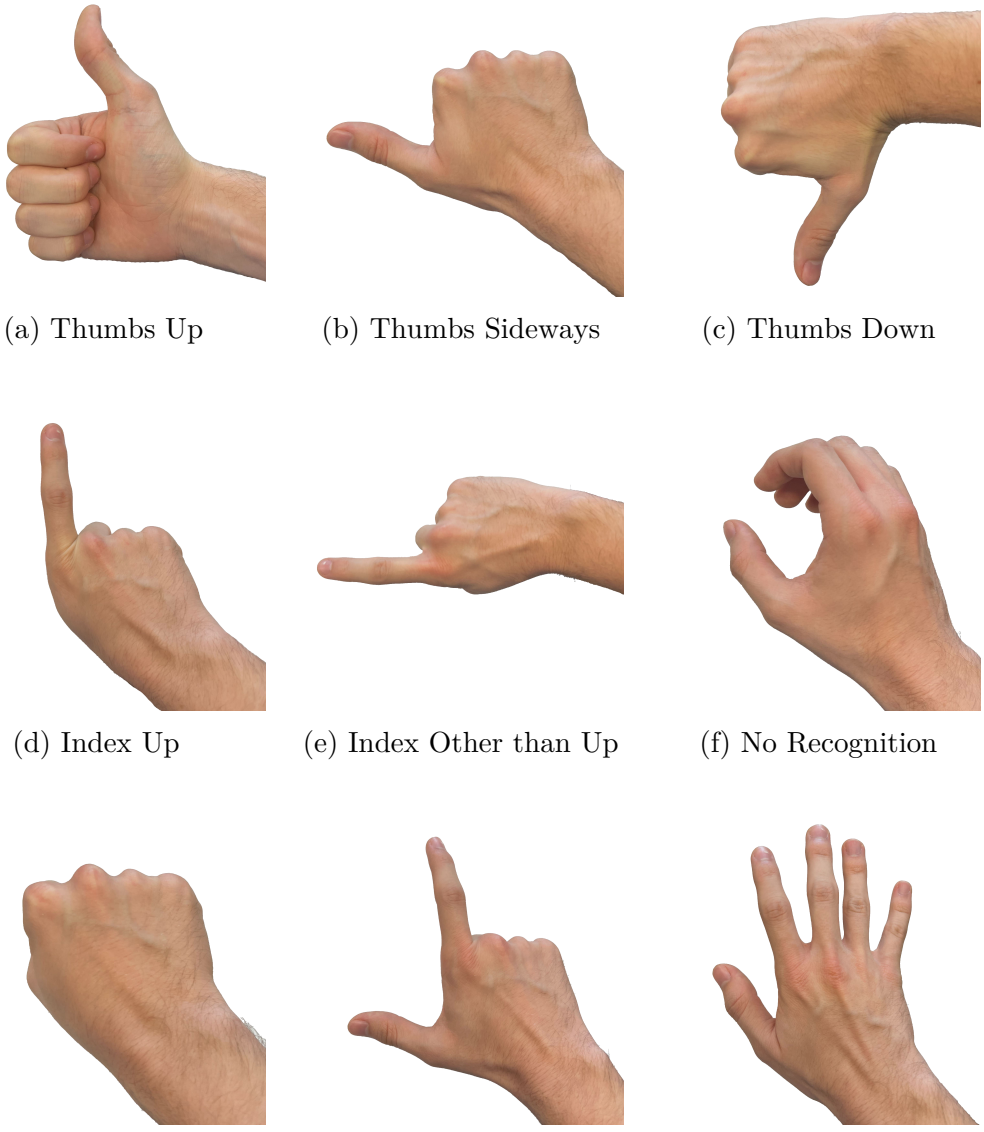
13 Hand Gesture Recognition

The TPI contains a hand gesture recognition system, which you can configure yourself. It works by checking the curl of each finger and comparing it with specific threshold values to determine the shown hand gesture [12].
Once a known hand gesture is recognized, it invokes the corresponding function, which you can set up in the Unity inspector of the 'TPI_HandGestureController' component of the 'TPI_Manager' GameObject. Alongside the option to designate the called functions, it also comes with other options, like threshold values for the curls, that can be customized by you.
This script originally stemmed from Manuel Koch's Master's Thesis and was heavily adapted to work with the TPI and to provide more features. [13]

The following hand gestures are recognized by default:

- Thumbs Up
- Thumbs Sideways
- Thumbs Down
- Index Up
- Index Other than Up
- Fist (all fingers curled)
- Two (Thumb + Index)
- Five (all fingers straight)

Refer to figure 13.1 for a visual display of all the recognized hand gestures.



(a) Thumbs Up (b) Thumbs Sideways (c) Thumbs Down

(d) Index Up (e) Index Other than Up (f) No Recognition

(g) Fist (h) Two (i) Five

Figure 13.1: Overview of all hand gestures that the TPI recognizes

If desired, you can extend the list of recognized hand gestures by adapting the 'TPI_HandGestureController' script.

14 Deployment

Unity related section

Once you are ready to deploy your project to the HoloLens 2 for example, it is important to make sure that you have correctly setup MRTK according to the MRTK2 tutorial found on the following website: `https://learn.microsoft.com/en-us/training/modules/learn-mrtk-tutorials/1-3-exercise-configure-unity-for-windows-mixed-reality?ns-enrollment-type=learningpath&ns-enrollment-id=learn.azure.beginner-hololens-2-tutorials` (accessed: 15.06.2023).

Afterward, please make sure that your project is set up according to the following points:

- In the 'Build Settings': Add the correct scenes, set 'Platform' to 'Universal Windows Platform', set 'Architecture' to 'Arm 64-bit', and set 'Build Type' to 'D3D Project'.
- In 'Project Settings' in the 'Player' tab: Set a 'Company Name', 'Product Name', and 'Version'.
- In 'Project Settings' in the 'Player' tab in the 'Publishing Settings' section: Setup 'Package name' and 'Version' and set the 'Capabilities' to: 'InternetClient, InternetClientServer, PrivateNetworkClientServer, WebCam, Microphone, SpatialPerception, GazeInput' (can also include more than that). Finally, set the 'Supported Device Families' to 'Holographic'.

IMPORTANT: If you plan on using ROS, please ensure you have set up the correct IP address and port number in the 'Unity Robotics Hub'. Both can be accessed in the network settings of the ROS host computer.

Once you have completed the previous steps, you can deploy it in the 'Build Settings' by clicking 'Build' and selecting a folder. Do not use 'Build and Run' as this will not work.

It is good practice to always deploy your project to the same folder without deleting the contents first, as the settings made to the build solution using Visual Studio will not change if you build your project again in Unity. Furthermore, the building and deployment of your project will happen faster.

Possible problems encountered while building in Unity:

- 'AssetDatabase' does not work outside the Unity environment; please use 'Resources' or 'AssetBundles' to load assets at runtime
- 'using UnityEditor' at the top of scripts, which are not purely designated for the Unity Editor / Inspector, will throw errors and cannot be used. Thus, please remove it. (Custom Editors / Inspectors scripts or scripts in an Editor folder are acceptable as they will only be called in the Unity Editor)
- Some imported packages might not be compatible with the 'Universal Windows Platform' and 'ARM 64-bit'

Visual Studio related section

Now, head to the folder, which contains your build files, and open the file of type 'Visual Studio Solution' ('.sln') with Visual Studio (e.g., Right-Click + Open With → Microsoft Visual Studio or double-click on the file).

Once it has been opened, make sure that the 'Universal Windows' solution in the Solutions Explorer is set up as 'Startup Project' (it should be highlighted in bold). If that is not the case, you can use Right-Click + 'Set as Startup Project'.

To save and load workflows, you have to make some changes to the project's permissions. Open the 'Package.appxmanifest' found in the 'Universal Windows' application section (extend the arrow) using Right Click + 'Open with' and select 'XML (Text) Editor'. First of all, add the code from code sample 14.1 to the <Capabilities> at the end:

Code 14.1: Capability needed for the SaveController

```
<uap:Capability Name="documentsLibrary" />
```

Secondly, paste the code from code sample 14.2 above </Application> (Application ***without the s*** at the end!):

Code 14.2: Extensions needed for the SaveController

```
<Extensions>
  <uap:Extension Category="windows.fileTypeAssociation">
    <uap:FileTypeAssociation Name="jpeg">
      <uap:SupportedFileTypes>
        <uap:FileType>.jpeg</uap:FileType>
      </uap:SupportedFileTypes>
    </uap:FileTypeAssociation>
  </uap:Extension>
  <uap:Extension Category="windows.fileTypeAssociation">
    <uap:FileTypeAssociation Name="mp4">
      <uap:SupportedFileTypes>
        <uap:FileType>.mp4</uap:FileType>
      </uap:SupportedFileTypes>
    </uap:FileTypeAssociation>
  </uap:Extension>
  <uap:Extension Category="windows.fileTypeAssociation">
    <uap:FileTypeAssociation Name="txt">
      <uap:SupportedFileTypes>
        <uap:FileType>.txt</uap:FileType>
      </uap:SupportedFileTypes>
    </uap:FileTypeAssociation>
  </uap:Extension>
</Extensions>
```

Once you have done that, you can close the 'Package.appxmanifest' file and select the 'Universal Windows' application again using Left Click. This concludes the settings needed for the 'SaveController'.

Now, for the final settings, before you can deploy your program to the HoloLens 2, you must set up the 'IP-Address' of the HoloLens2. This can be done by choosing Project → Properties → Configuration Properties → Debugging, where you have to select 'Remote Machine' as 'Debugger to launch', 'Arm64' or 'Active(Arm64)' as 'Platform' and 'Release', 'Active(Release)', 'Debug' or 'Active(Debug)' as 'Configuration'. Grab your HoloLens 2 and navigate to Settings → Update & Security → For Developers. Please write down the IP-Address (only the numbers) found in the 'Wi-Fi' field under the

'Device Portal' section. Finally, change the 'Machine Name' to the IP-Address, which you have just retrieved.
Please be aware that this number differs for each HoloLens and foreach Network that the HoloLens is in! (It could cause an error while deploying if you have not set the 'Machine Name' to the correct value.)
Now, close the settings window again.

As a last step, select either 'Release' or 'Debug' in the toolbar, depending on what you want to achieve when you deploy. The debug mode can be helpful if you have errors in your program, as it displays what the program is doing in the Visual Studio 'Output Console' while the program is running. Therefore you should not disconnect the HoloLens from your computer when choosing the debug mode.
Furthermore, select 'ARM64' in the toolbar and 'Device' as the Output rather than 'Remote Machine', found when extending the field with the full green arrow in the taskbar.

Finally, to deploy your program, select either 'Start without Debugger' (only the outline of the green arrow is visible) if you have selected 'Release' or select 'Start with Debugger' (the full green arrow is visible) if you have selected 'Debug' as the mode.
Please wait until there is a confirmation text in the 'Output Console' of Visual Studio before disconnecting the HoloLens from the computer. The confirmation text should state that the deployment was either successful or that there were errors while deploying. Depending on your computer, the deployment can take quite some time (even 10 minutes is possible).

How to solve possible problems encountered while deploying in Visual Studio:

- If you try to open the 'Visual Studio Solution' ('.sln') with Visual Studio and it throws an error saying that there are two 'assimp.dll' files, please delete the one that is located in the 'Plugins' folder of your build files.
- Please make sure you have entered the correct 'Machine Name' in the Project Properties.
- Please ensure the HoloLens is connected with a USB cable supporting data transfer.
- Please make sure that the cable is connected to both devices properly.
- Some packages imported in Unity might prevent deployment or cause errors.

A System Specifications

The framework was built with the help of the following components:

- Build Machine: Windows 10 Home 22H2, Intel(R) Core(TM) i7-1065G7 CPU (1.30GHz), 16.0GB RAM
- Development Environment:
  - Game Engine Editor: Unity3D v2021.3.19f1 [14]
  - MRTK v2.8.3 [5]
  - Unity Robotics Hub v0.7.0-preview [9]
  - URDF importer v0.5.2-preview [11]
  - Unity 2D Sprite v1.0.0 [15]
- Programming Language: C#
- Devices:
  - Microsoft HoloLens 2
  - FRANKA RESEARCH 3 Robotic Arm
- Network: Smartphone wireless hotspot

The code of the framework, as well as the installation instructions and the documentation of the TPI, can be found in the following GitHub repository: `https://github.com/Donnickis/TaskPlanningInterface`

B Used ROS Messages

The following ROS message types and topic names were used throughout the TPI framework. Oftentimes, the topic names can be changed in the Unity inspector of their respective class.Red entries are used to publish messages, blue ones are used to receive messages.

Table B.1: Overview of used ROS Messages and Topics in the TPI

Usage Location	Topic Name	Message Type
TPI_Photo	tpi_camera_image	ImageMsg
TPI_Photo	tpi_camera_transform	TransformMsg
TPI_RobotController	tpi_robot_basepose	TransformMsg
TPI_RobotController	tpi_robot_joints	JointStateMsg
TPI_RobotController	tpi_robot_pose	PoseMsg
TPI_RobotController	tpi_robot_posereachable	BoolMsg
TPI_ROSController	tpi_should_execute_instructions	BoolMsg
TPI_ROSController	tpi_visualization_speed	Float32Msg
TPI_ROSController	tpi_robot_destination	PoseMsg
TPI_ROSController	tpi_pose_change	PoseMsg
TPI_ROSController	tpi_diagnostic_status	DiagnosticStatusMsg

C ROS Message Definitions

In this chapter, you can find the definitions of all the ROS message types that were used in the TPI framework.

All those definitions stem from the Unity Robotics Hub [9], but can also be applied on the ROS side.

C.1 BoolMsg

```
public const string k_RosMessageName = "std_msgs/Bool";
public override string RosMessageName => k_RosMessageName;

public bool data;

public BoolMsg(){
    this.data = false;
}

public BoolMsg(bool data){
    this.data = data;
}
```

C.2 DiagnosticStatusMsg

```
public const string k_RosMessageName = "diagnostic_msgs/
   DiagnosticStatus";
public override string RosMessageName => k_RosMessageName;

//  This message holds the status of an individual component
   of the robot.
```

```
//  Possible levels of operations.
public const sbyte OK = 0;
public const sbyte WARN = 1;
public const sbyte ERROR = 2;
public const sbyte STALE = 3;
//  Level of operation enumerated above.
public sbyte level;
//  A description of the test/component reporting.
public string name;
//  A description of the status.
public string message;
//  A hardware unique string.
public string hardware_id;
//  An array of values associated with the status.
public KeyValueMsg[] values;

public DiagnosticStatusMsg(){
    this.level = 0;
    this.name = "";
    this.message = "";
    this.hardware_id = "";
    this.values = new KeyValueMsg[0];
}

public DiagnosticStatusMsg(sbyte level, string name, string
   message, string hardware_id, KeyValueMsg[] values){
    this.level = level;
    this.name = name;
    this.message = message;
    this.hardware_id = hardware_id;
    this.values = values;
}
```

C.3 Float32Msg

```
public const string k_RosMessageName = "std_msgs/Float32";
public override string RosMessageName => k_RosMessageName;

public float data;

public Float32Msg(){
    this.data = 0.0f;
}

public Float32Msg(float data){
    this.data = data;
}
```

C.4 ImageMsg

```
public const string k_RosMessageName = "sensor_msgs/Image";
public override string RosMessageName => k_RosMessageName;

//  This message contains an uncompressed image
//  (0, 0) is at top-left corner of image
public Std.HeaderMsg header;
//  Header timestamp should be acquisition time of image
//  Header frame_id should be optical frame of camera
//  origin of frame should be optical center of cameara
//  +x should point to the right in the image
//  +y should point down in the image
//  +z should point into to plane of the image
//  If the frame_id here and the frame_id of the CameraInfo
//  message associated with the image conflict
//  the behavior is undefined

public uint height;
//  image height, that is, number of rows
public uint width;
//  image width, that is, number of columns
//  The legal values for encoding are in file src/
   image_encodings.cpp
//  If you want to standardize a new string format, join
//  ros-users@lists.ros.org and send an email proposing a new
    encoding.
public string encoding;
//  Encoding of pixels -- channel meaning, ordering, size
//  taken from the list of strings in include/sensor_msgs/
   image_encodings.hpp

public byte is_bigendian;
//  is this data bigendian?
public uint step;
//  Full row length in bytes
public byte[] data;
//  actual matrix data, size is (step * rows)

public ImageMsg(){
    this.header = new Std.HeaderMsg();
    this.height = 0;
    this.width = 0;
    this.encoding = "";
    this.is_bigendian = 0;
    this.step = 0;
    this.data = new byte[0];
}
```

```
public ImageMsg(Std.HeaderMsg header, uint height, uint width
    , string encoding, byte is_bigendian, uint step, byte[]
    data){
     this.header = header;
     this.height = height;
     this.width = width;
     this.encoding = encoding;
     this.is_bigendian = is_bigendian;
     this.step = step;
     this.data = data;
}
```

C.5 PointMsg

```
public const string k_RosMessageName = "geometry_msgs/Point";
public override string RosMessageName => k_RosMessageName;

//  This contains the position of a point in free space
public double x;
public double y;
public double z;

public PointMsg(){
    this.x = 0.0;
    this.y = 0.0;
    this.z = 0.0;
}

public PointMsg(double x, double y, double z){
    this.x = x;
    this.y = y;
    this.z = z;
}
```

C.6 PoseMsg

```
public const string k_RosMessageName = "geometry_msgs/Pose";
public override string RosMessageName => k_RosMessageName;

//  A representation of pose in free space, composed of
    position and orientation.
public PointMsg position;
```

```
public QuaternionMsg orientation;

public PoseMsg(){
    this.position = new PointMsg();
    this.orientation = new QuaternionMsg();
}

public PoseMsg(PointMsg position, QuaternionMsg orientation){
    this.position = position;
    this.orientation = orientation;
}
```

C.7 QuaternionMsg

```
public const string k_RosMessageName = "geometry_msgs/
    Quaternion";
public override string RosMessageName => k_RosMessageName;

//  This represents an orientation in free space in
    quaternion form.
public double x = 0;
public double y = 0;
public double z = 0;
public double w = 1;

public QuaternionMsg(){}

public QuaternionMsg(double x, double y, double z, double w){
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
}
```

C.8 TransformMsg

```
public const string k_RosMessageName = "geometry_msgs/
    Transform";
public override string RosMessageName => k_RosMessageName;

//  This represents the transform between two coordinate
    frames in free space.
```

```
public Vector3Msg translation;
public QuaternionMsg rotation;

public TransformMsg(){
    this.translation = new Vector3Msg();
    this.rotation = new QuaternionMsg();
}

public TransformMsg(Vector3Msg translation, QuaternionMsg
   rotation){
    this.translation = translation;
    this.rotation = rotation;
}
```

C.9 Vector3Msg

```
public const string k_RosMessageName = "geometry_msgs/Vector3
   ";
public override string RosMessageName => k_RosMessageName;

//  This represents a vector in free space.
//  This is semantically different than a point.
//  A vector is always anchored at the origin.
//  When a transform is applied to a vector, only the
   rotational component is applied.
public double x;
public double y;
public double z;

public Vector3Msg(){
    this.x = 0.0;
    this.y = 0.0;
    this.z = 0.0;
}

public Vector3Msg(double x, double y, double z){
    this.x = x;
    this.y = y;
    this.z = z;
}
```

D Overview of the TPI Classes

This chapter provides an overview of all the public variables and functions from the most important TPI classes. For a detailed view with a description of all variables and a description of the return value (if applicable), please visit the definition of the functions in their respective classes.

D.1 TPI_DialogMenuController

Namespace: TaskPlanningInterface.Controller

This script aims to assist you in creating 'Dialog Menus' to your liking. You can choose from multiple pre-made templates (prefabs) or create your own, which will automatically be populated with the correct information for you.

Properties

dialogMenuPrefabList	Add, alter or remove Dialog Menu Templates by changing their information in this list.

Functions

ClearDialogMenus ()	Remove all active Dialog Menus that were previously created
GetDialogMenuChoices (string)	Get the choices made by the operator during runtime. This includes for example the text input by a keyboard.
GetDialogMenuCount ()	Returns how many dialog menus are currently active
SpawnDialogMenu (TPI_DialogMenuInformation)	Create a Dialog Menu by plugging in the dialogMenuInformation.

SpawnDialogMenu (int)	Create a Dialog Menu by plugging in the index of the respective Prefab in the "Dialog Menu Prefab List" that you want.
SpawnDialogMenu (string)	Create a Dialog Menu by plugging in the dialogMenuID of the respective Template you want.
UnSpawnDialogMenu (TPI_DialogMenuInformation)	Remove an active Dialog Menu by plugging in the dialog-MenuInformation.
UnSpawnDialogMenu (string)	Remove an active Dialog Menu by plugging in the dialog-MenuID of the respective Prefab that you want.
UnSpawnLastDialogMenu ()	Remove the last Dialog Menu that was created.

Pre-made Functions

The pre-made functions of the dialog menus can have a different amount of variables (as some variables will be automatically set if left out). Therefore, looking up the definitions of the individual functions at the bottom of the TPI_DialogMenuController is highly suggested to decide which version of the respective function to use.

ShowConstraintTypeMenu	Creates a Dialog Menu that allows the operator to choose the type of a Constraint (whether it is globally active or snippet-specific)
ShowDropdownSelectionMenu	Creates a Dialog Menu that allows the operator to select an item from a list of strings (styled as a dropdown).
ShowErrorMenu	Creates a Dialog Menu that points out an error to the operator or provides the operator with information.
ShowErrorMenu_TwoButtons	Creates a version of the 'ShowErrorMenu' Dialog Menu containing two buttons. This can be useful to give the operator a choice between two options.
ShowObjectNameMenu	Creates a Dialog Menu that allows the operator to select a Name for an Object (e.g. Snippet or Constraint)

D.2 TPI_HandGestureController

Namespace: TaskPlanningInterface.Controller

This script aims to recognize the hand gestures performed by the operator. This script originally stems from Manuel Koch's Master's Thesis and was heavily adapted to work with the TPI and to provide more features. [13]

Properties

curled_threshold	Determine the curled threshold, i.e. after which the finger counts as a curled finger.
enableHandGestures	Determine whether Hand Gestures should be recognized and whether the underlying UnityEvents should thus be invoked.
fistEvent	Determine what happens if the Hand Gesture 'fist' is recognized.
fiveFingersEvent	Determine what happens if the Hand Gesture 'five fingers' is recognized.
gesture	This field tells you what Hand Gesture is currently getting recognized by the TPI_HandGestureController.
gestureCheckFrequency	Select how many times per second the hand gesture should be checked.
gestureVerificationAmount	Select how many times a Hand Gesture must be recognized in a row before it is ultimately selected as an active Hand Gesture, and thus before the underlying event is invoked. -> reduces the chance of accidentally recognized gestures
gestureVisualizationPrefab	Prefab that should be visible if a hand gesture is recognized (only important if visualizeRecognizedGesture was set to true).
indexOtherEvent	Determine what happens if the Hand Gesture 'index other' is recognized.
indexUpEvent	Determine what happens if the Hand Gesture 'index up' is recognized.
selectedHand	Select on which Hand the Gestures should be detected.
straight_threshold_finger	Determine the straight threshold of a finger, i.e. before which the finger counts as a straight finger.
straight_threshold_thumb	Determine the straight threshold of a thumb, i.e. before which the thumb counts as a straight finger.
thumbsDownEvent	Determine what happens if the Hand Gesture 'thumbs down' is recognized.

thumbsSidewaysEvent	Determine what happens if the Hand Gesture 'thumbs sideways' is recognized.
thumbsUpEvent	Determine what happens if the Hand Gesture 'thumbs up' is recognized.
twoFingersEvent	Determine what happens if the Hand Gesture 'two fingers' is recognized.
visualizeRecognizedGesture	Determine whether a text should appear on the hand with the name of the gesture once a hand gesture is recognized.

Functions

AssignUnityActionToGesture (Gestures, UnityAction)	Assign a UnityAction that should be invoked once the specific gesture is triggered.
AssignUnityEventToGesture (Gestures, UnityEvent)	Assign a new UnityEvent that should be invoked once the specific gesture is triggered.
getCurrentGesture ()	Returns the currently recognized Hand Gesture.

The 'TPI_HandGestureController' script also includes modified versions of all sequence functions (e.g. 'Start Sequence'), specifically designed for the use with hand gestures.

D.3 TPI_MainController

Namespace: TaskPlanningInterface.Controller

The MainController handles the TPI in general and the 'Workflow Hand Menu'.

Properties

abortButtonIcon	Abort icon that will be shown on the button in the Dialog Menu that gives the operator the option to manage the workflows once the ManageWorkflows Button in the Hand-Menu is pressed.
createNewSaveButtonIcon	'Create a new save' icon that will be shown on the button in the Dialog Menu that gives the operator the either overwrite an existing save or create a new one.
createWorkflowButtonIcon	Icon that will be shown on the CreateWorkflow Button in the HandMenu to create a new workflow.
currentWorkflow	Reference to the information of the currently active work-flow.
deleteButtonIcon	Delete icon that will be shown on the button in the Dialog Menu that gives the operator the option to manage the workflows once the ManageWorkflows Button in the Hand-Menu is pressed.
hideButtonIcon	Icon that will be shown on the ToggleTPI Button in the HandMenu to hide the TaskPlanningInterface.
isOpen	This bool states whether the TPI has been opened. Please do not change it yourself.
loadButtonIcon	Icon that will be shown on the button in the Dialog Menu that asks the operator to select and load a Workflow once the LoadWorkflow Button in the HandMenu is pressed.
overwriteButtonIcon	Overwrite icon that will be shown on the button in the Dialog Menu that gives the operator the either overwrite an existing save or create a new one.
reassignButtonIcon	Icon that should be shown on the button that asks the operator to reassign something.
showButtonIcon	Icon that will be shown on the ToggleTPI Button in the HandMenu to show the TaskPlanningInterface if it is hid-den.
startButtonIcon	Icon that will be shown on the button in the Dialog Menu that asks the operator to enter a Workflow Name once the CreateWorkflow Button in the HandMenu is pressed.

References to other GameObjects

buildingBlocksMenu	Reference to the 'Building Blocks Menu' GameObject
constraintFunctionContainer	Reference to the 'Constraint Function Container' GameObject
dialogMenuContainer	Reference to the 'Dialog Menu Container' GameObject
handMenu_objectPlacement	Reference to the 'Object Placement Hand Menu' GameObject
handMenu_workflow	Reference to the 'Workflow Hand Menu' GameObject
objectPlacementController	Reference to the 'ObjectPlacementController' GameObject
robotURDF	Reference to the digital twin of the robot GameObject
rosController	Reference to 'ROS Controller' GameObject
rosStatusMenu	Reference to the 'ROS Status Menu' GameObject
selectionMenu	Reference to the 'Selection Menu' GameObject
sequenceFunctionsMenu	Reference to the 'Sequence Functions Menu' GameObject
sequenceMenu	Reference to the 'Sequence Menu' GameObject
snippetFunctionContainer	Reference to the 'Snippet Function Container' GameObject

Functions

DeleteWorkflow (int)	Deletes the workflow with a specific index from the list of saved workflows (SaveController).
LoadWorkflow ()	Opens a dialog menu if a workflow has been loaded before or if one has been created in order to confirm that the operator wants to delete the previous workflow.
ManageWorkflows ()	Opens a dialog menu that allows the operator to reset the current workflow (if applicable) or delete saved workflows.
ResetTPI ()	Call this function if you want to reset all changes and settings made in the TPI Menu.
SaveWorkflow ()	Opens a dialog menu where the operator can choose to either create a new workflow save or overwrite an existing (if this workflow was loaded before).
ShowTutorial ()	Toggles the tutorial (whether it is active or not).
ToggleTPI ()	Create a new Workflow (only if using for the first time) or toggle the visibility of the Task Planning Interface by using this function.

D.4 TPI_ObjectPlacementController

Namespace: TaskPlanningInterface.Controller

The ObjectPlacementController was created to easily control the position of objects and dialog menus in the environment around the operator.

Properties

attachToOperator	Decide whether the GridObjectCollection should automatically follow the operator around the environment.
distanceToCamera	Decide whether the GridObjectCollection should automatically rotate with the operator.
rotateWithOperator	Determine the distance of the GridObjectCollection to the operators view
rotationSpeed	How fast should the GridObjectCollection rotate
smoothTime	How much smooth time should the GridObjectCollection have while following the operator?

Functions

Some of the functions are overloaded with at least one other version that accepts other parameters. → Those do not have any brackets with values in this list. Furthermore, some also permit optional parameters. For a more in-depth view of the description of all the parameters, please visit the TPI_ObjectPlacementController script.

ClearAllSpots ()	Frees up all the spots in the grid.
ConvertAnchorToPosition (StartingPosition)	Converts the provided anchor point to its index in the list.
ConvertSpotIntToVector3 (int)	Converts the index (int) of a spot into the Vector3 of it at that index.
ConvertSpotVector3ToInt (Vector3)	Converts the Vector3 of a spot into the position index (int) of it.
FindAndReservePosition	Finds a spot that fits best to the provided anchor point (and search algorithm and search direction) and reserves it for the GameObject, returning the position and rotation of said spot.

FindUnoccupiedSpot	Finds a suitable unoccupied spot according to your anchor point, search algorithm, and search direction and returns the index.
FreeUpSpot	Frees up a spot in the grid by providing the TPI_PositionAndRotation of said spot.
GetIndexOfGameObject (GameObject)	Looks for the GameObject in the Grid and return the index of it.
GetNumUnoccupiedSpots (two versions overloaded)	Calculates the number of unoccupied spots for a given starting position, search algorithm, and search direction, and returns the value.
GetNumUnoccupiedSpots ()	Calculates the total number of unoccupied spots and returns the value.
GetPreviousPositionIndex ()	Return the position index of the last GameObject that was removed from the Grid.
GetSpotDown (int, int)	Moves from an initial position downwards along a column and return the index of that spot.
GetSpotLeft (int, int)	Moves from an initial position leftwards along a row and return the index of that spot.
GetSpotRight (int, int)	Moves from an initial position rightwards along a row and return the index of that spot.
GetSpotUp (int, int)	Moves from an initial position upwards along a column and return the index of that spot.
ReserveSpot	Reserves a spot for a GameObject by providing the TPI_PositionAndRotation of the desired spot.
ResetPreviousPositionIndex ()	Resets the previous position index, so that new dialog menus will be spawned in the Center Middle.
SetAttachmentState (bool)	This function activates the feature that the GridObjectCollection follows the operator.
SetRotationState (bool)	This function activates the feature that the GridObjectCollection rotates with the operator.
SwapSpotContents	Exchanges the contents of two spots.

D.5 TPI_RobotController

Namespace: TaskPlanningInterface.Controller

This script handles the movements of the digital twin in the MR environment. Some parts of this script were adapted from code from Manuel Koch's master's thesis, but heavily rewritten to better work with the TPI and to provide more features. [13]

Properties

poseNotReachableEvent	Designate what happens if a pose is not reachable.
poseReachableEvent	Designate what happens if a pose is reachable.
poseReachableTopic	Topic name to which Unity will subscribe in order to receive the information whether a pose is reachable or not.
robotBaseTopic	Topic name under which the robot base pose will be either published or to which Unity will subscribe.
robotBase_publishFrequency	In the case of publishRobotBasePose being set to automatically publishing, set the frequency how many times each seconds the robot base pose should be published.
robotJointTopic	Topic name to which Unity will subscribe in order to display the movement of the robot.
robotLinkNames	Add the names of all the links that contain movable joints. Please pay attention, as the order is important!

robotPoseTopic	Topic name under which a pose should be published to ROS.
_robotBase	Reference to the ArticulationBody Component on the robot base link. If you do not add anything, it will automatically set it to the 'FRANKA RESEARCH 3' standard settings.

Functions

GetRelativePose (two versions overloaded)	This function converts the given world position and rotation into the relative local coordinates of the robot base link.
isRobotActive ()	Return whether the robot is currently visible (not disabled).
PublishRobotBase ()	This function is used to publish the current pose of the robot base in the robotBaseTopic (publishes a TransformMsg).
PublishRobotPose (three versions overloaded)	This function can be used to publish the current Pose of the ArticulationBody you provide.
SetRobotActive (bool)	This function allows you to enable (true) or disable (false) the visibility of the robot.
SetRobotBase (TransformMsg)	This function is used as the Callback function for the robotBaseTopic and sets the position and rotation of the robot base link.

D.6 TPI_ROSController

Namespace: TaskPlanningInterface.Controller

The ROSController handles the ROS Integration on the unity side to facilitate successful interactions, including a working messaging system.

Properties

AllTopics	Returns all the topics that are currently in the topics list (IEnumerable).
OnConnected	This action is invoked if the connection to ROS was successfully established WITHOUT errors. Add whatever action should be executed in that case.
OnDisconnected	// This action is invoked if the connection to ROS was stopped or errors occurred. Add whatever action should be executed in that case.
RegisterDataStream	Register Publishers and Subscribers using this action → they will be registered once the connection was successful. After registering, this action is automatically cleared and ready for new registrations.

General Functions

EnterROSAddress (string, int)	Used to enter the ROS IP and port, and start the ROS connection (only used in the case of rosConnectionType == ROSConnectionType.requestIP).
GetROSConnection ()	Returns the instance of the ROSConnection that is currently active
hasConnectionError ()	Return whether the connection to ROS has errors and was thus not successful.
hasConnectionThread ()	Return whether ROS tried to initiate a connection. If you want to check if the ROS Connection was successful or not, please use 'hasConnectionError()'
IsROSConnectionDeactivated ()	Returns whether Unity tries to create a ROS Connection on Startup.

Topics related Functions

GetOrCreateTopic (string, string)	Searches for a topic by name and returns it (RosTopicState).
GetTopicAndTypeList (Action<Dictionary<string, string>>)	Allows you to perform actions on the entire topic and type lists by providing the function.
GetTopicList (Action<string[]>)	Allows you to perform actions on the entire topic list by providing the function.
GetTopic (string)	Searches for a topic by name and returns it (RosTopicState).
ListenForTopics (Action<RosTopicState>)	Allows you to add a function as a listener, which will get called if a new topic is recognized.
RefreshTopicsList ()	Makes sure that the topics list in general and each member of it is setup correctly.

Publishing related Functions

Publish (string, Message)	Publishing is used to send messages from Unity to ROS.
QueueSysCommand (string, object)	System commands consist of two parts, the actual command, which always starts with '___' to distinguish them from ros topics, followed by a message.
RegisterPublisher (string, string)	Before you can actually send a message, you must set up the Publisher (a topic for a specific message).
RegisterPublisher <MessageType> (string)	Before you can actually send a message, you have to set up the Publisher (a topic for a specific message).

Subscriber related Functions

HasSubscriber (string)	Return whether this specific Topic has any Subscribers.
SubscribeByMessageName (string, string, Action<Message>)	Subscribing is used to receive messages from ROS in Unity.
Subscribe <MessageType>(string, Action<MessageType>)	Subscribing is used to receive messages from ROS in Unity.
Unsubscribe (string)	Once you no longer wish to receive the messages of a topic, you can unsubscribe from it.

Service related Functions

ROS Services can be used to set up a request-reply system, which in turn is defined by a pair of messages.

ImplementService <MessageRypeRequest, MessageTypeResponse> (string, Func<MessageRypeRequest, MessageTypeResponse>)	The ImplementService function is used to RESPOND to requests made by ROS.
ImplementService <MessageTypeRequest, MessageTypeResponse> (string, Func<MessageTypeRequest, Task<MessageTypeResponse>>)	The ImplementService function is used to RESPOND to requests made by ROS.
RegisterRosService (string, string, string)	The RegisterRosService function is used to make REQUESTS from ROS by Unity.
RegisterRosService<TRequest, TResponse> (string)	The RegisterRosService function is used to make REQUESTS from ROS by Unity.
SendServiceMessage <MessageTypeResponse> (string, Message)	Once the Listener for the Service Response from ROS has been set up using RegisterRosService, you can call the SendServiceMessage function to actually send requests to ROS.
SendServiceMessage <MessageTypeResponse> (string, Message, Action<MessageTypeResponse>)	Once the Listener for the Service Response from ROS has been set up using RegisterRosService, you can call the SendServiceMessage function to actually send requests to ROS.

UI related Functions

DrawConnectionArrows (bool, float, float, float, float, bool, bool, bool)	This function can be used in order to easily draw the connection arrows (indicate the status of the connection).
GetConnectionColor (float, bool, bool)	Returns the correct Color that belongs to each connection status: noConnection -> gray, hasErrors -> red, elapsedTime is smaller than 0.03f -> bright, else -> fading color from bright to dark

TPI related Functions

AddStatusEntry (DiagnosticStatusMsg)	Call this function in order to add a DiagnosticStatusMsg to the ROS Status Menu.
GetStatusEntries ()	Returns the list of entries in the ROS Status Menu.
GetStatusEntryCount ()	Returns the number of entries in the ROS Status Menu.
PublishRobotDestination (Vector3, Quaternion, bool)	This function sends a PoseMsg in order to tell ROS that the robot arm has to move to a specific position with a specific rotation.
PublishShouldExecuteInstructions (bool)	This function sends the message of the shouldExecuteInstructions bool. The shouldExecuteInstructions BoolMsg is used to tell ROS whether the given machine instructions should be actually performed or whether they should only be simulated.
PublishVisualizationSpeed (float)	This function sends the massage of the visualizationSpeed float. The VisualizationSpeed Float32 is used to tell ROS at what speed the visualization of Snippets should happen, i.e. if it should be sped up. (only needed in the case that the shouldExecuteInstructions bool is false)
RegisterRobotDestinationPublisher ()	This function registers the PoseMsg in order to, later on, tell ROS that the robot arm has to move to a specific position with a specific rotation.
RegisterShouldExecuteInstructionsPublisher ()	This function registers the publisher for the shouldExecuteInstructions bool message.
RegisterVisualizationSpeedPublisher ()	This function registers the publisher for the visualizationSpeed float message.
RemoveStatusEntry (two versions overloaded)	Call this function in order to remove a DiagnosticStatusMsg from the ROS Status Menu.
SubscribeToDiagnosticStatus (Action<DiagnosticStatusMsg>)	Call this function in order to subscribe to the diagnostic status messages. This function is mainly used for the ROS Status Menu of the Task Planning Interface.
SubscribeToPoseChange (Action<PoseMsg>)	Call this function in order to subscribe to a pose change of the robot arm. A pose consists of the position (Vector3) and of the rotation (Quaternion).

D.7 TPI_SaveController

Namespace: TaskPlanningInterface.Controller

The SaveController acts as a data manager, which saves and loads the relevant data.

Properties

activeWorkflowID	Indicates if a workflow has been loaded or saved during this session, showing the ID of that workflow. If it is empty, a new one has been created or it has not been saved before.
workflowSaveData	This List contains all the workflows that have been saved in the past and were thus loaded at startup, and all the workflows that the operator has saved during the runtime of this session.

Functions

DeleteConstraint (string)	Deletes a constraint when given the constraint ID.
DeleteSnippet (string)	Deletes a snippet when given the snippet ID.
DeleteWorkflow (two versions overloaded)	Deletes a specific workflow when given the ID or reference.
GetConstraint (string, string)	Returns the constraint when given the constraint ID.
GetSnippet (string, string)	Returns the snippet when given the snippet ID.
LoadConstraint (string, string)	Loades the constraint when given the snippet ID and adds it to the constraint list in the sequence menu.
LoadSnippet (string, string)	Loades the snippet when given the snippet ID and adds it to the sequence list in the sequence menu.
LoadWorkflow (string)	Loads and opens a specific workflow by providing the desired workflow ID.
ResetSaveController ()	H
SaveConstraint (two versions overloaded)	Saves a constraint when given the ID or reference
SaveSnippet (two versions overloaded)	Saves a snippet when given the ID or reference.
SaveWorkflow (bool)	Saves the currently active Workflow
SaveWorkflow (TPI_Workflow)	Saves the provided workflow as a new workflow by assigning a new workflowID.

D.8 TPI_SequenceMenuController

Namespace: TaskPlanningInterface.Controller

The SequenceMenuController is used to control the sequence menu. Furthermore, it controls the sequence functions, which steer the sequence of snippets, e.g. 'StartSequence()' or 'StopSequence()'.

Properties

buttonHoldTime	Time it takes for Unity to register the OnHold Event (the smaller the value, the faster)
isPaused	This bool states whether the sequence is currently paused.
requireDeletionConfirmation	Decide whether the operator has to confirm his choice of deleting a snippet or constraint.
requireVariableChangeConfirmation	Decide whether the operator has to confirm his choice of changing the values of the underlying variables of a snippet or constraint.
standardButtonPressedCooldown	Time given to the operator in order to perform a double click on a Snippet in seconds
_constraintObjects	Information of all the Constraints (both global and snippet-specific) of the current Workflow. You should not make any alterations to this list as this should be read-only.
_sequenceState	This enum states the current status of the Sequence. You cannot change the value manually.
_snippetObjects	Information of all the Snippets of the current Workflow sorted by their correct position in the Sequence. You should not make any alterations to this list as this should be read-only.

General Functions

ResetSequenceMenu ()	This function resets the sequence menu to it's initial state (it will be like it was when the TPI was started for the first time)

Snippet related Functions

AddSnippet (TPI_Snippet)	Allows you to add a snippet at the end of the sequence list by providing the snippet function component.
ClearSnippets ()	Allows you to remove all snippets from the sequence list.
GetSnippetAt (int)	Allows you to get a snippet at a specific position (index starts at 0).
GetSnippetSequenceLength ()	Allows you to get the snippet count in the sequence list.
GetSnippetSequence ()	Allows you to get the entire snippet function sequence (not the visual objects spawned in the sequence menu though).
HandleSnippetOnHoldButtonEvents (TPI_SequenceMenuButton)	Handles the OnHold Button Event in order to properly move the position of a Snippet inside the Sequence with drag and drop and to delete a snippet properly. Usually, you will not use this function.
InsertSnippetAt (TPI_Snippet, int)	Allows you to insert a snippet at a specific position in the sequence list by providing the snippet function component and the position.
RemoveLastSnippet ()	Allows you to remove the last snippet from the sequence list.
RemoveSnippetAt (int)	Allows you to remove a snippet from the sequence list by providing the position in the list (index starts at 0).
RemoveSnippet (three versions overloaded)	Allows you to remove a snippet from the sequence list.
SelectSnippet (int)	Handles what happens if a Snippet is selected. Usually, you will not use this function.
SwitchSnippets (two versions overloaded)	Allows you to switch two snippets in the sequence list.
UpdateSnippetVisuals (TPI_Snippet)	Updates the visual snippet representation in the sequence menu by providing the TPI_Snippet reference and the visual snippet GameObject itself.
VisualizeSnippets ()	Visualizes the Snippets by moving the digital twin of the robot arm along the planned path.

Constraint related Functions

AddConstraint (TPI_Constraint)	Allows you to add a new constraint to the constraints list by providing the constraint function component.
AddConstraint (TPI_Constraint, TPI_ConstraintType)	Allows you to add a new constraint to the constraints list by providing the constraint function component and the constraint type (global or snippet-specific).
ClearConstraints ()	Allows you to remove all constraints from the constraints list.
GetConstraintCount ()	Allows you to get the constraints count (both global and snippet-specific).
GetConstraints ()	Allows you to get all constraints (both global and snippet-specific).
GetGlobalConstraintAt (int)	Allows you to get the global constraint at a specific position in the constraints list (index starts at 0).
GetGlobalConstraintCount ()	Allows you to get the global constraint count.
GetGlobalConstraintList ()	Allows you to get all global constraints.
GetSpecificConstraintAt (int)	Allows you to get the snippet-specific constraint at a specific position in the constraints list (index starts at 0).
GetSpecificConstraintCount ()	Allows you to get the snippet-specific constraint count.
GetSpecificConstraintList ()	Allows you to get all snippet-specific constraints.
GetSpecificConstraintsOfSnippet (string)	Allows you to get all snippet-specific constraints that belong to the snippet with the given snippetID.
HandleConstraintOnHoldButtonEvents (TPI_SequenceMenuButton)	Handles the OnHold Button Event in order to delete the held Constraint. Usually, you will not use this function.
RemoveConstraint (two versions overloaded)	Allows you to remove a constraint from the constraints list.
UpdateConstraintVisuals (TPI_Constraint)	Updates the visual constraint representation in the sequence menu by providing the TPI_Constraint reference and the visual constraint GameObject itself.
VisualizeConstraints ()	Visualizes the global or snippet-specific constraints.

Sequence related Functions

EmergencyStopSnippet ()	In case of an emergency, it stops the sequence of snippets and the constraints. This function is different from the StopSequence function in the regard that the users of the TPI can define special behaviors. Please only use this function if the sequence is running, otherwise, use 'StopSequence();'.
RepeatSnippet ()	Repeates the current snippet by adding a copy of it at the next position in the sequence. Please only run this function if the sequence is either running or paused.
RestartSequence ()	Restarts the whole sequence after it was stopped by the StopSequence function. Please only run this when the sequence was stopped beforehand.
ReturnToPreviousSnippet ()	Stops the current snippet and starts the previous one. Please only run this function if the sequence is either running or paused.
SkipSnippet ()	Skips the current snippet and starts the next one. Please only run this function if the sequence is either running or paused.
StartSequence ()	Starts the Sequence of snippets and enables the coroutine and respective constraints.
StopSequence ()	Manually Stops the Sequence of snippets and disables the coroutine and constraints. Please only run this function if the sequence is either running or paused.
TogglePauseSnippet ()	Pauses or unpauses the sequence of snippets. Please only run this function if the sequence is either running or paused.

D.9 TPI_TutorialController

Namespace: TaskPlanningInterface.Controller

The TutorialController handles the creation and progression of the Tutorial.

Properties

There are no public variables in this class.

Functions

AddTutorialStep (TutorialDialog)	Adds a tutorial dialog to the end of the list.
ClearTutorialDialogList ()	Removes all entries of the tutorial dialog list
GetAllTutorialDialogs ()	Returns the list of all tutorial dialogs.
GetRemainingTutorialSteps ()	Returns how many tutorial dialogs are left to be executed (not including the current one).
GetTotalTutorialSteps ()	Returns how many tutorial dialogs there are.
GetTutorialDialogFromList (string)	Returns the tutorial dialog with the specified ID from the list if it exists
InsertTutorialStep (TutorialDialog, int)	Inserts a tutorial dialog at a specific position in the list.
IsSpecificTutorialActive (string)	Returns whether the tutorial with the given tutorialID is currently active.
IsTutorialActive ()	Returns whether the tutorial is active.
RemoveAtTutorialStep (int)	Removes the tutorial dialog found at the specified position from the list.
RemoveTutorialStep (TutorialDialog)	Removes a specific tutorial dialog from the list.
ResetTutorial ()	Resets all the Tutorial variables to their initial states.
ToggleTutorial ()	Toggles the Tutorial. If the Tutorial should be deactivated, it resets it completely.
TriggerNextTutorialStep (string)	Triggers the next tutorial step if the conditions are met.

D.10 TPI_WorkflowConfigurationController

Namespace: TaskPlanningInterface.Controller

The WorkflowConfigurationController handles the "Selection Menu" and the "Building Blocks Menu" in order to set up the snippet and constraint templates.

Properties

categories	Add, alter or remove Categories by changing the information contained in this list
charactersShowed_constraint	How many characters of the Template Name should be shown? To show the full name, set it to -1.
charactersShowed_snippet	How many characters of the Template Name should be shown? To show the full name, set it to -1.
constraintTemplates	Add, alter or remove Constraints by changing the information contained in this list
showTemplateName_constraint	Should the Constraint Template name be shown (e.g. NoGoZone)?
showTemplateName_snippet	Should the Snippet Template name be shown (e.g. MoveTo)?
snippetTemplates	Add, alter or remove Snippets by changing the information contained in this list.

Functions

CloseBuildingBlocksMenu ()	Allows you to close the building blocks menu.
IsBuildingBlocksMenuOpen ()	Returns whether the building blocks menu is currently open.

D.11 TPI_Constraint

Namespace: TaskPlanningInterface.Workflow

The TPI_Constraint class is used as a base class for the inheritance to work. It defines variables and functions, which are important for constraints, so that other child classes can derive those.

Properties

constraintInformation	Information belonging to the constraint
dialogMenuController	Reference to the TPI_DialogMenuController
saveController	Reference to the TPI_SaveController
saveData	Class that saves all variables that you want to be saved and loaded by the TPI
sequenceMenuController	Reference to the TPI_SequenceMenuController

Functions

ApplyConstraint ()	This function handles what happens when the constraint starts to be applied,
ButtonPressed ()	This function should implement what happens when the button is pressed in the building blocks menu.
ChangeVariables ()	This function gets called when the operator clicks on a Constraint in the Sequence Menu in order to change the values of the variables.
SetupControllerReferences ()	This function sets up the references to those Task Planning Interface Controllers, which you will use the most.
StopConstraint ()	This function handles what happens when the operator either wants to stop the snippet sequence or when the snippet has ended and the snippet-specific constraint thus should no longer be applied
StopVisualization ()	This function handles what happens when the operator no longer wants to visualize the constraint.
UpdateSaveData ()	This function correctly sets up and updates your save data script.
VisualizeConstraint ()	This function handles what happens when the operator wants to visualize the constraint.

D.12 TPI_Snippet

Namespace: TaskPlanningInterface.Workflow

The TPI_Snippet class is used as a base class for the inheritance to work. It defines variables and functions, which are important for snippets, so that other child classes can derive those.

Properties

dialogMenuController	Reference to the TPI_DialogMenuController
saveController	Reference to the TPI_SaveController
saveData	Class that saves all variables that you want to be saved and loaded by the TPI
sequenceMenuController	Reference to the TPI_SequenceMenuController
snippetInformation	Information belonging to the snippet

Functions

ButtonPressed ()	This function should implement what happens when the button is pressed in the building blocks menu.
ChangeVariables ()	This function gets called when the operator double-clicks on a snippet in the sequence menu to change the variables' values.
OnEmergencyStop ()	This function handles what happens if the operator signals with his hands to stop due to an emergency abruptly.
OnHasEnded ()	This function handles what happens when your snippet function has ended, i.e., when the sequence progress in the sequence menu reaches the next snippet.
RepeatSnippet ()	The operator indicated that he intends to let the snippet run again once the current iteration of the function has ended.
RunSnippet ()	This function handles what happens when the snippet is started, i.e., when the sequence progress in the sequence menu reaches your snippet.
SetupControllerReferences ()	This function sets up the references to the Task Planning Interface Controllers you will use most.

SkipSnippet ()	This function handles what happens when the operator wants to manually stop the snippet and go to the next one mid-execution.
StopSnippet ()	This function handles what happens when the operator wants to stop the sequence mid-execution manually.
UpdateSaveData ()	This function correctly sets up and updates your save data script.

D.13 TPI_Photo

Namespace: TaskPlanningInterface.Helper

The TPI_Photo class can take a picture with the HoloLens Webcam, which will then be sent to ROS. This behavior can be used to calibrate the base link of the robot.
This script originally stems from Manuel Koch's Master's Thesis and was slightly adapted to work with the TPI. [13]

Properties

topic_Image	Topic name under which the HoloLens camera image (ImageMsg) should be published to ROS.
topic_CameraTransform	Topic name under which the camera matrix (TransformMsg) should be published to ROS.

Functions

GetCamToWorld ()	Returns the camera position when executing the function in the Unity editor (not applicable on the HoloLens).
TakePhoto ()	Takes a picture with the HoloLens RGB camera and sends it to ROS.

List of Figures

List of Tables

List of Codes

Bibliography

[1] Unity: Components. https://docs.unity3d.com/Manual/Components.html (accessed: 15.06.2023).

[2] Dan Dockter. What are robotic constraints and optimizations? https://www.energid.com/blog/what-are-robotic-constraints-and-optimizations (accessed: 15.06.2023), Sep 2019.

[3] Unity: Gameobject. https://docs.unity3d.com/Manual/GameObjects.html (accessed: 15.06.2023).

[4] Unity: Prefabs. https://learn.unity.com/tutorial/prefabs-e (accessed: 15.06.2023).

[5] MRTK v2.8.3. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05 (accessed: 15.06.2023).

[6] HoloLens 2 Website. https://www.microsoft.com/en-us/hololens (accessed: 21.06.2023).

[7] ROS Website. https://www.ros.org/ (accessed: 15.06.2023).

[8] FRANKA RESEARCH 3 Website. https://www.franka.de/research/ (accessed: 21.06.2023).

[9] Unity Robotics Hub. https://github.com/Unity-Technologies/Unity-Robotics-Hub (accessed: 15.06.2023).

[10] RosGeometry component. https://github.com/Unity-Technologies/ROS-TCP-Connector/blob/main/ROSGeometry.md (accessed: 21.06.2023).

[11] URDF Importer. https://github.com/Unity-Technologies/URDF-Importer (accessed: 15.06.2023).

[12] MRTK Hand Tracking. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/input/hand-tracking?view=mrtkunity-2022-05 (accessed: 21.06.2023).

[13] Manuel Koch, Sophokles Ktistakis, and Mirko Meboldt. Human-Robot Object Handovers using Mixed Reality, December 2022. Written at the Product Development Group Zurich.

[14] Unity Website. https://unity.com/ (accessed: 21.06.2023).

[15] Unity: 2D Sprite. https://docs.unity3d.com/Packages/com.unity.2d.sprite@1.0/manual/index.html (accessed: 15.06.2023).