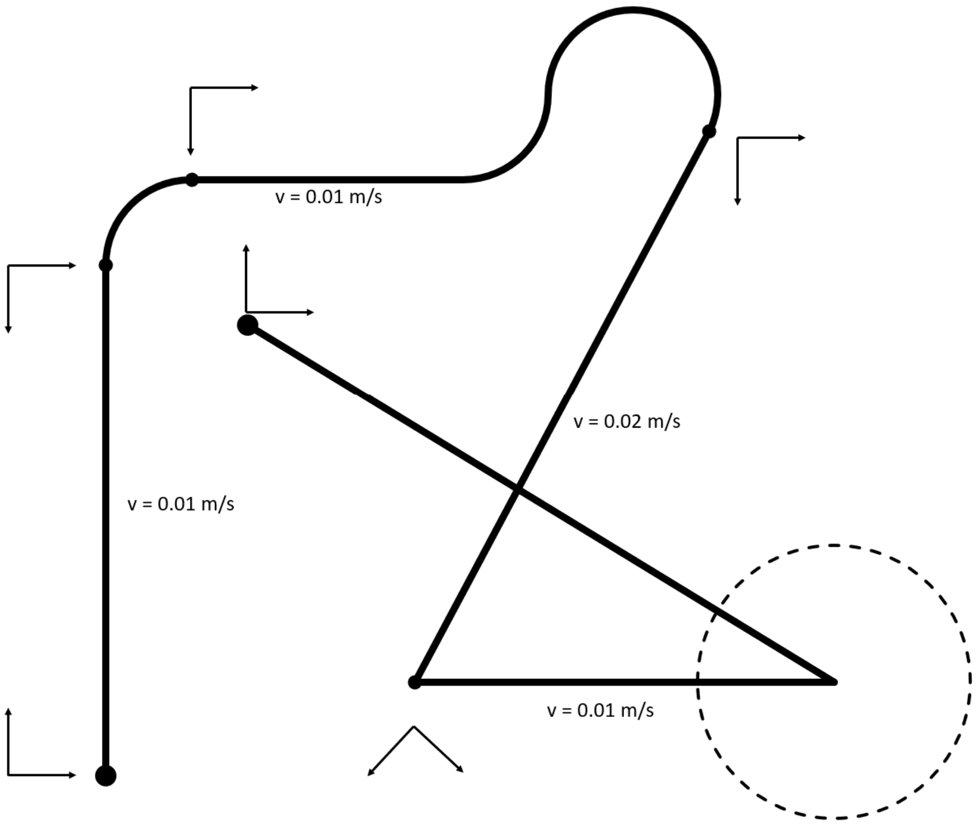
ROS Noetic Project - Developing a welding application with ARTag-guided path planning

image source

In this ROS project, you'll work in groups to perform a welding task with an Universal Robots 6-dof arm. The objective is to plan and execute the 2D welding curve. The task for your team is to develop the complete application.

The desired path gets teached point-wise by ar tags that are filmed from a camera. In the best case, you will be able to generate a weld like the one shown below using the ar tag teach guide. The tasks built on each other. It is recommended to first implement a linear movement between two points and expand from there.

Our goal is that you can work simultanously on the project and learn a lot by this welding application. This guide always mentions the Universal Robots 5 (UR5), but it's applicable to other robots from UR, of course! Adjust the commands according to your robot!



Example of a welding path

Starting point

- [x] ROS1 Noetic (latest ROS1 version) installed on Ubuntu 20.04
- [x] Necessary ROS packages are already installed.
- [x] *Terminator* or the 'standard' ubuntu terminal + basic understanding
- [x] VSCode for programming
- [x] Some knowledge in programming (ideally using Python or C++)
- [x] Ability to **search** for information in the official documentation, forums, tutorials, github issues and reviewing source code
- [x] Beneficial: Have a high frustration tolerance when learning ROS, as the learning curve initially tends to be shallow. Additionally, it is important to engage in extensive experimentation and enjoy working as a team.
- [x] You can access the additional files for the tasks in the share folder.
- [x] We've setup the robot with the corresponding laptop. This includes network settings and downloading + building the correct drivers in the workspace *ws_igmr*. **Don't use a laptop with a different robot than the one suggested by the numbering (Laptop & handwritten sticker on the robot should match)**
- [x] We provide you with a few *starting points*, mostly already created ros packages with some py-nodes/ launch files; it also includes the package *tf2_basics* which is a tutorial/ example for the usage of the ros internal system to work with frames and transformations.

Tasks summary

- [ ] Use a camera driver & publish the video output in a topic

- [ ] Utilize the package ar_track_alvar for pose estimations of ARTags
- [ ] Start movegroup for simple planning & execution of the ur (check drivers & visualisation in RViz)
- [ ] Write a node that uses the pose estimates for teaching a path using the ARTags (you might want to check if the pose is reachable)
- [ ] Plan & execute the teached path (if possible)
- [ ] Implement safety features (avoid collision with the ground e.g.)
- [ ] Test & implement measures for a robust process
- [ ] Implement additional features, e.g. set the end effector orientation by interpolating between 2 angles

Learning outcomes

- [ ] **Collaboration and teamwork in a project-based environment**
- [ ] Understand and increase understanding of how ROS works (topics, nodes, packages)
- [ ] Controlling a robotic arm with ROS
- [ ] Using external packages for hardware (camera, ARTags)
- [ ] Learn the basics of path planning with MoveIt
- [ ] Visualization with RViz

Grades/ Evaluation

After your group has worked 2 days on this project, we want you to present and demonstrate your results in a few minutes. Key aspects (not ordered by importance) are:

- [ ] What and how you've implemented safety features (collision-avoidance)
- [ ] Robustness: Is it possible to change ar tags? What if you move the ar tag out of camera range?
- [ ] Using ROS-like features like launch file with arguments: You want to implement some arguments in your launch file for reusability of your work in previous tasks
- [ ] Clean code: You don't need to write hundreds of lines of python & it should be understandable

Table of content

Task 0 - A quick introduction in ros package management

You have already learned what a ROS package is. ROS is a popular and powerful tool because so many packages are open source. Often there are already required functionalities that only have to be configured and started correctly.

Once a package is released, you can install it with apt. Say, you read about a package **my_awesome_package** that does one specific task you need. It might be released, than you can install it with 1 easy command:

```
sudo apt install ros-noetic-my-awesome-package
```

The prefix **ros-noetic-** indicates your ros distro. Thats all, you are ready to use it because it's already compiled.

It can happen that the package you want is not released in general or not released for your distro. If you found the package on github (or something similar), just clone it in *your-workspace/src*. In your workspace:

```
cd src
git clone my-link-for-git-clone
```

This copies the source code in your src folder. To proceed, you need to build your workspace and source it afterwards. Now you can use the package. While the first option installs the package globally on your machine (you can use it in any workspace), in the second option you can only use it in your current sourced workspace.

Task 1 - Test robot connection & first movement

1. RViz + MoveIt demonstration

We first simulate a robot to plan and execute a trajectory with MoveIt. Open your terminal, source ROS, build and source your workspace:

```
source /opt/ros/noetic/setup.bash

cd ~/ws_igmr/

catkin build
source devel/setup.bash
```

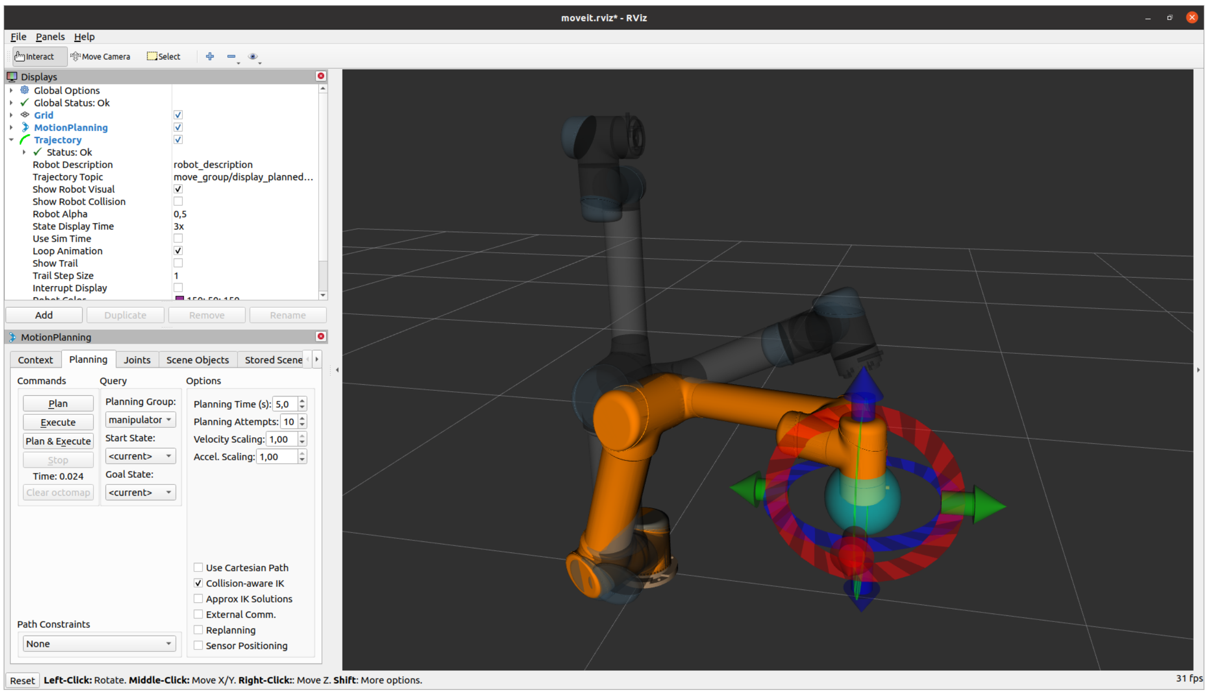
Afterwards, launch the robot launcher:

```
roslaunch robot_launcher launch_robot.launch sim:=true use_rviz:=true
```

HINT: Terminate the launch/ nodes file afterwards with CTRL + C (inside the corresponding terminal window).

Make sure that the argument sim is set to true. Otherwise the real robot starts!.

You should see an ur arm. In the left pannel, press *Add* and select *Trajectory* if it's not already added to your pannel. Drag the ball at the robot's end effector with your mouse to a desired position (See the image below).



Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Trajectory ✓
Status: Ok
Robot Description: robot_description
Trajectory Topic: move_group/display_planned...
Show Robot Visual: ✓
Show Robot Collision: ☐
Robot Alpha: 0,5
State Display Time: 3x
Use Sim Time: ☐
Loop Animation: ✓
Show Trail: ☐
Trail Step Size: 1
Interrupt Display: ☐

Time: 0.024
Planning Group: manipulator
Start State: <current>
Goal State: <current>
Planning Time (s): 5,0
Planning Attempts: 10
Velocity Scaling: 1,00
Accel. Scaling: 1,00
☐ Use Cartesian Path
✓ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options.

In the pannel, press *Plan*. You should see the planned trajectory. (Select *Loop Animation* in the Trajectory menue inside the left pannel to see the planned trajectory again).

After a certain trajectory is planned, execute it with the button *Execute*. That was your first movement of a (simulated) robot with rviz. We'll repeat it with the real one now.

2. Safety notice

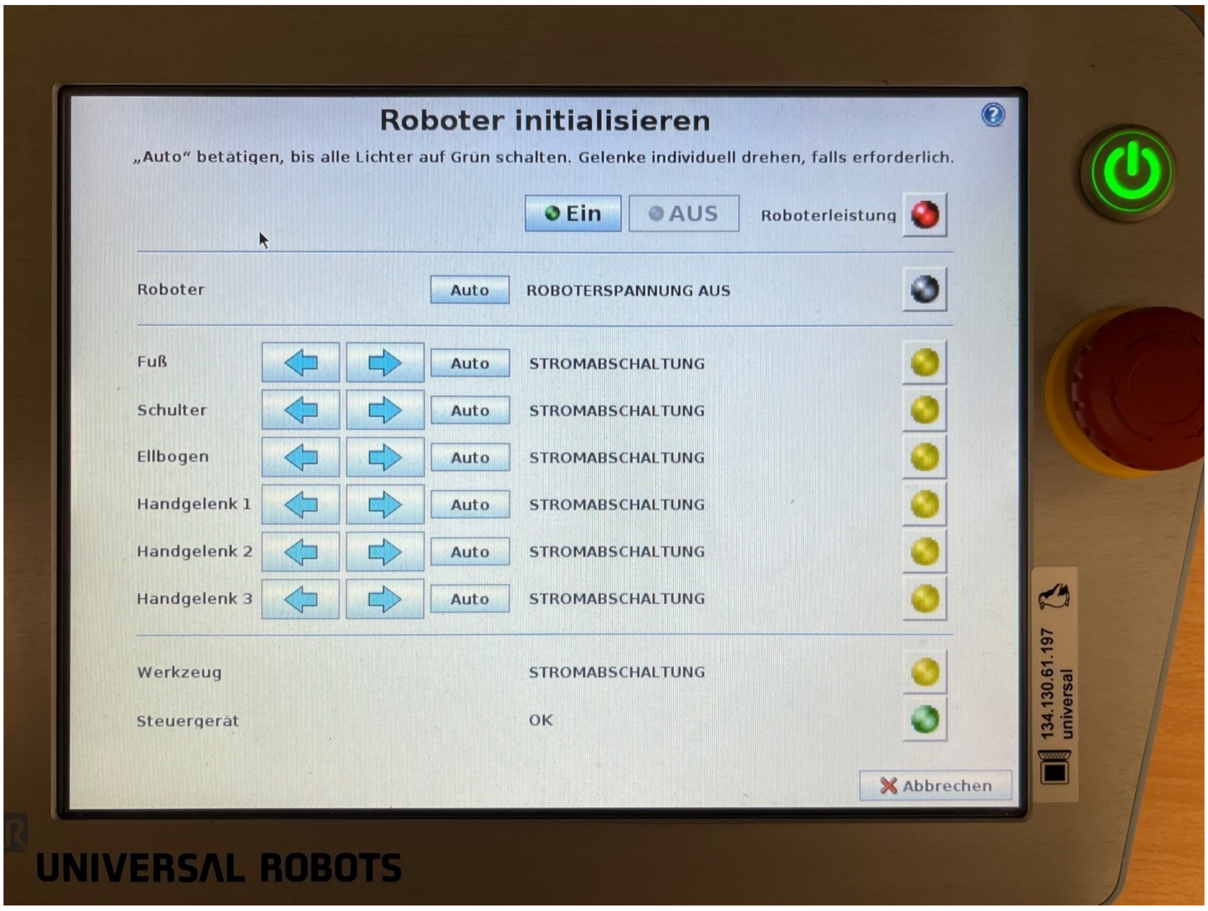
Read the following instructions carefully!

- Emergency Stop: Ensure that the emergency stop button is easily accessible and can be pressed at any time to halt the robot's movement.
- Collision Avoidance: Clear the workspace of any obstacles to avoid collisions. Be aware: The arm itself cannot collide with itself, but with the ground plate or the end effector, if they are not part of the urdf or planning scene. Always check the planned path in rviz before executing any motion.
- Always read the entire paragraph first before starting to implement it.

3. Initialize robot

On Startup, the teach pendant asks to initialize your robot. If your robot is on PolyScope >= 3.x, you just need to *enable* and press *start*. Otherwise you need to move every axis until the green led is activated:

Navigate back to the home screen click on *Roboter EINSTELLEN* (*SETUP Robot*) and then *Roboter INITIALISIEREN* (*INITIALISE Robot*). If not done already, unlock the emergency button. Your screen should look like the image below:



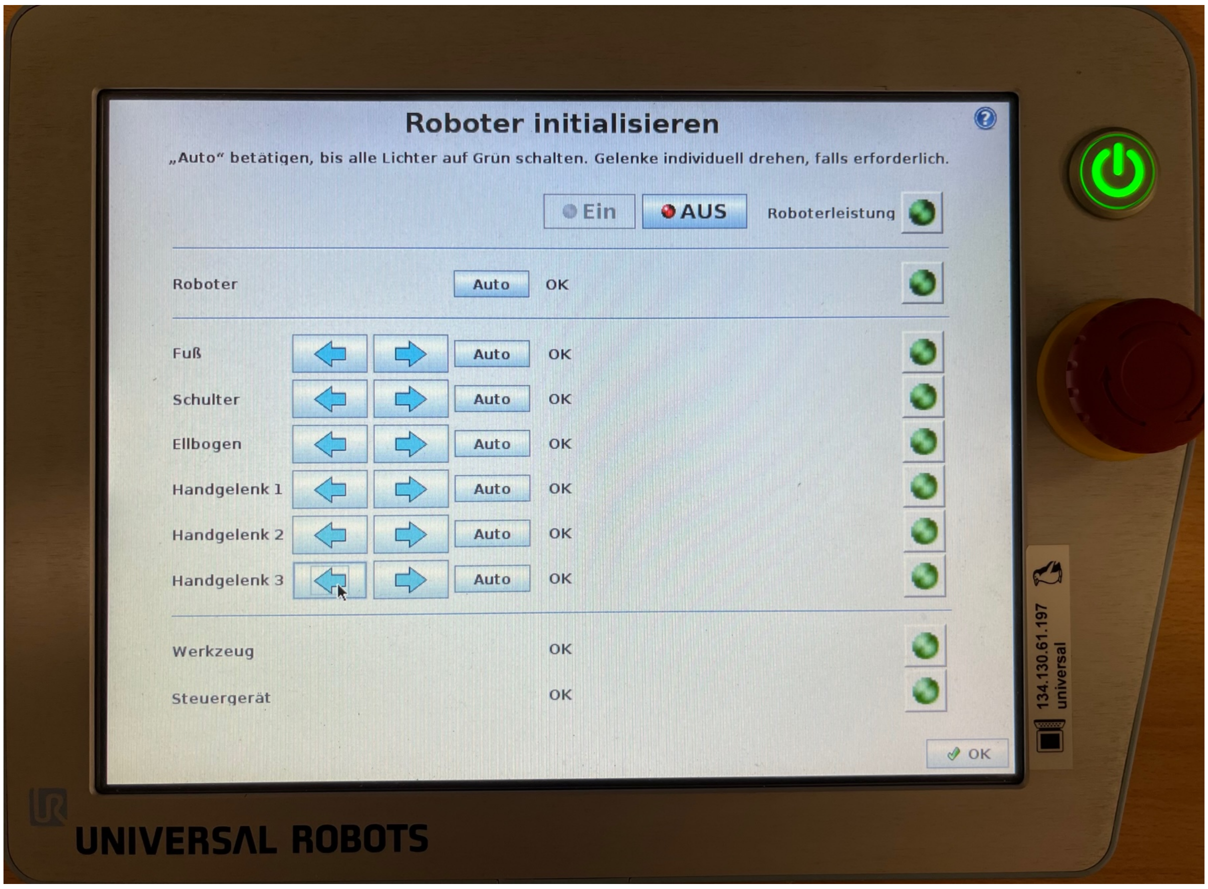
Roboter initialisieren

„Auto" betätigen, bis alle Lichter auf Grün schalten. Gelenke individuell drehen, falls erforderlich.

● Ein / ● AUS	Roboterleistung	●
Roboter (Auto)	ROBOTERSPANNUNG AUS	●
Fuß (Auto)	STROMABSCHALTUNG	●
Schulter (Auto)	STROMABSCHALTUNG	●
Ellbogen (Auto)	STROMABSCHALTUNG	●
Handgelenk 1 (Auto)	STROMABSCHALTUNG	●
Handgelenk 2 (Auto)	STROMABSCHALTUNG	●
Handgelenk 3 (Auto)	STROMABSCHALTUNG	●
Werkzeug	STROMABSCHALTUNG	●
Steuergerät	OK	●

Press *Ein* (*On*) in the first row. You now need to enable every joint. You can do so by press (and hold) *Auto*, **but watch out for collisions**! The robot stops immediately when you release the button.

You can also move each joint separately until the green dot appears. If your screen looks like the following image, press *OK*. The initialization process is now finished.



Roboter initialisieren

„Auto“ betätigen, bis alle Lichter auf Grün schalten. Gelenke individuell drehen, falls erforderlich.

Ein | AUS | Roboterleistung

Roboter	Auto	OK
Fuß	Auto	OK
Schulter	Auto	OK
Ellbogen	Auto	OK
Handgelenk 1	Auto	OK
Handgelenk 2	Auto	OK
Handgelenk 3	Auto	OK
Werkzeug		OK
Steuergerät		OK

OK

4. Launch real robot

You already have the necessary drivers/ packages installed in your workspace ~/ws_igmr. Make sure, that your laptop is only connected with the ethernet cable that goes to the switch. **ONLY CONNECT THIS CABLE TO THE USB-ETHERNET-ADAPTER** and plug it in one usb port of your laptop. Make sure there is no other cable in the integrated ethernet interface of your laptop, otherwise you cannot connect to the robot.

```
roslaunch robot_launcher launch_robot.launch sim:=false use_rviz:=true
```

HINT: Terminate nodes afterwards with CTRL + C (inside corresponding terminal window).

As in the simulation, you should be able to drag the interactive marker with your mouse to a new position.

FOR NEWER ROBOTS WITH POLYSCOPE >= 3.x: You need to start the URCap program now before you can make movements:

- In the top bar, press *OPEN* -> *Program*
- Open the program 00_IGMR/ros.urp
- Press the *play* button in the bottom bar and select *Starting at the beginning*

BEFORE YOU START PLANNING/ EXECUTING change the velocity & acceleration scaling down to 0.2 or something similar. You don't want to plan/ execute with 100% velocity/ acceleration in the beginning. Now, as in the simulation, you can *Plan* a movement and *Execute* it within the MotionPlanning widget in rviz.

Task 2 - Embed a webcam in your ROS stack

You'll use a webcam to estimate the pose of an ARTag. To do so, we need to embed the video stream of our webcam in ROS. We'll benefit from the rich ecosystem and use the package **usb_cam**, it's already installed on your ubuntu laptop.

- [ ] Skim the linked documentation
- [ ] Create a new ros package with a directory *config* and copy the file *camera_calibration.yaml* inside.
- [ ] Write a ROS launch file, which starts the necessary nodes (with its parameters) and shows the video stream in rviz. Save the rviz configuration so that you can use it in your launch file (means that rviz starts up with your individual window setup).

HINTS

- You might want to implement some arguments in the launch file, so that your work is reusable in later tasks. Reasonable arguments could be:
  - id of your camera device (string)
  - launch rviz node (bool)
- Find a *default* camera calibration file in the provided course material.
- As a starting point the package **webcam_launch** can be used
- The cam node needs the following parameters: video_device, camera_name, camera_frame_id, camera_info_url
- Make sure that the cam_node finds the camera calibration file **(no warning in the terminal!)**
- After you have added widgets to rviz, you can safe this window setup of rviz (configuration) as a file. On launching, you can pass the path of the config file to rviz node - it will load your previous configured window.
- Use tf to publish a frame (see *frame_example.launch* in the pkg *webcam_launch*)
- The documentation for tf's static_transform_publisher can be found here

Task 3 - Tracking of AR Tags

We use our camera implementation to estimate poses of ar tags. For this task we use the package ar_track_alvar. Since its not released as a binary, we need to clone & build it ourselves. Within your workspace, execute the following command:

```
git clone -b noetic-devel https://github.com/ros-perception/ar_track_alvar.git src/ar_track_alvar
```

It clones the source code of that package from the branch *noetic-devel* for our distro noetic (hence, *-b noetic-devel*) inside the folder *src*. Now, build (and source) your workspace, then you are ready to work with ar_track_alvar.

- ☐ Skim the linked documentation
- ☐ Write a launch file which uses ar_track_alvar and your previous work from task 2. Rviz should launch with the widget *Marker* to display the ar tags.
- ☐ Echo the alvar-specific topics in an additional terminal window.

HINTS

- You can use/ include the launch file `pr2_indiv_no_kinect.launch` within your launch files. Necessary arguments are:
  - cam_image_topic: Name of the topic that contains the camera video stream
  - cam_info_topic: Name of the topic that containts information about the camera (among other things the cam calibration file)
  - output_frame: Name of the camera frame
- For the development, you can use a pre-recorded video stream instead, provided in the package **camera_bag**. See *README.md* of that package for further instructions.
- You can extend the launch file from the previous exercise or create a new one. If you write a new launch file, include the previous work (launch file) with
- If you need further information on frame transformations, look out for **tf_basics**, a ros package, which includes a little tutorial.

```
<include file="$(find my_package_name)/launch/my_previous_launchfile.launch">
  <arg name="arg_name_1" value="a_value">
  <arg name="arg_name_2" value="$(arg my_launch_argument_val)">
</include>
```

Task 4 - Implement safety features

An important part of our application are safety features. For example, we need to make sure that the robot will not collide with it's environment. While MoveIt takes care of self-collisions of the arm, we need to complete the planning scene with our welding nozzle and other obstacles to avoid collision with the nozzle/ the robot with it's environment.

- [ ] Recreate your robot's environment, so that obstacles are considered for collision-checking. For that, write a python node, that adds these items to the planning scene.
- [ ] Create a launch file that combines your so far completed task. It makes sense to have an argument for launching the simulated or the real robot. Items to be launched are:
    - Start the webcam launch file
    - Start the ar track launc file
    - Start the just created node
    - Add the robot launcher

HINTS

- *tf_conversions* can help with transformations between different orientation formats (Matrix, Euler angles, Quaternion). See the provided tutorial/ example **tf2_basics** for how to use/ transform tf's in ros. Feel free to add additional frames.
- MoveIt's planning scene is responsible for collision objects.
- We provide the package *robot_environment* as a starting point.
- The nozzle (stl) is provided in the course materials folder.
- You might need to make the .py-file executable:

```
sudo chmod +x ~/ws_igmr/src/path_to_my_file.py
```

- Necessary PlanningScene Documentation
- There are some minor differences in the documentation and the actual implementation
- You can verify your implementation by launching the simulated robot. After starting your node you should see the changes in the planning scene
- If you feel like you need some overview of MoveIt concepts, see some information here
- This tutorial might also be helpful

Task 5 - Movement action server

The planning and execution is done in this node. It acts as an action server which gets the poses as inputs and plans/ executes the motion afterwards.

- [ ] create a new package for planning/ execution of the robot
- [ ] Implement the action server (meaning: Write a node): It waits for a incoming action. Then if plans the movement. If the plan is valid, execute it.
- [ ] Write a little test node that can call this server for testing purpose.

HINTS

- Planning & execution can be done with the MoveGroupCommander Python Interface
- We have not much time to check the planning for collision, therefore: Make sure that no collision can happen (welding nozzle, ground plane, camera setup). Correctly implemented safety features (task 4) will take care of this.
- Action definitions are usually done in a separate package. We provide you one (with cmake.txt and package.xml already setup correctly) called *welding_robot_msgs*. For the action server itself, we provide you again a template called *robot_mover*.

Task 6 - Teaching Node

We can read pose estimations through some topics by now. It's time now to develop our sophisticated teaching system. The input is the constant stream of artag pose estimations (or none if there is no ar tag in the field-of-view) through a specific topic. Keep in mind that in the end you want to send some poses which corresponds to the tcp (named *tool0*) frame so you need to transform the pose estimations to fit to this tcp frame.

- [ ] Write a node which captures user inputs for saving the current pose as a goal pose. You might need to make transformations of the frames to obtain correct tcp goal poses.
- [ ] Make sure that the nozzle moves along the teached path with a given z-offset to the XY-plane (= your table).

ADDITIONAL TASKS

- implement your solution in terms of stability checking, e.g. are the poses reachable
- Try to implement the possibility for the user to use an arc shape between some points instead of a straight line. How could this be implemented?
- In the teaching process, check if the goal pose for the robot is reachable. If not, print a feedback to the user and ignore this point.
- (Difficult) Extend your implementation so that the user can enter the cartesian nozzle speed between 2 captured poses.

HINTS

- Hardware-dimensions are provided in the additional material.
- As bevor, see **tf_basics** for a starting point into tf2 library.
- See **robot_teaching** for a starting point.
- The nozzle's endpoint should always be orthogonal towards the XY-plane.
- for better results in the pose estimation, it might be good to re-calibrate your camera once you placed it.
- There are different possible approaches for planning and possibly different solutions.

Task 7 - Merging all components

Congratulations, the hardest part is done! Now merge your work so that it can be launched with a single command.

- [ ] Merge your work into one launch file & test your application for edge cases.
- [ ] Make sure that common parameters like id of the artag etc. are not hardcoded but instead can be changed dynamically through launch file arguments for example.