

Manual for the Workshop Seamless Engineering

Prof. Eric Sax iTIV

Prof. Kai Furmans IFL

1	FOREWORD.....	4
2	CONTENT OF THE WORKSHOP	4
2.1	ABOUT THIS MANUAL	4
2.2	STRUCTURE AND PROCEDURE	4
2.3	MILESTONES AND SUBMISSIONS	5
2.4	EVALUATION OF THE WORKSHOP.....	6
2.5	CURRENT INFORMATION, DATES AND DEADLINES.....	7
3	PREPARATION.....	8
3.1	WHAT IS BWLEHRPOOL AND WHY IS IT NEEDED FOR SEAMLESS ENGINEERING?	8
3.2	USING BWLEHRPOOL FOR SEAMLESS ENGINEERING	9
3.3	LINUX – UBUNTU	9
4	FIRST STEPS IN SIMULATION ENVIRONMENT	11
5	PROBLEM DEFINITION.....	13
5.1	HINTS.....	13
5.2	INTRODUCTION OF THE TASK	13
5.2.1	<i>Procedure of the flow of goods within the picking system.....</i>	<i>13</i>
5.3	SYSTEM REQUIREMENT SPECIFICATIONS FOR THE SYSTEM TO BE DEVELOPED	14
5.4	DESCRIPTION OF THE MOST IMPORTANT KEY FIGURES OF CONVEYOR LINES	16
5.4.1	<i>Throughput.....</i>	<i>16</i>
5.4.2	<i>Picking time.....</i>	<i>16</i>
5.5	PRESENTATION OF THE HARDWARE	17
5.5.1	<i>uArm Swift Pro</i>	<i>17</i>
5.5.2	<i>uArm Slider</i>	<i>18</i>
5.5.3	<i>uArm Conveyor.....</i>	<i>20</i>
5.5.4	<i>Tutlebot3 Burger</i>	<i>22</i>
5.6	IMPLEMENTATION ON REAL HARDWARE	24
5.7	IMPORTANT TOPICS, ACTIONS AND SERVICES OF THE HARDWARE	24
5.8	EXAMPLE FOR THE CONTROL OF UARM	26
6	TOOLING	27
6.1	GITLAB.....	27
6.1.1	<i>What is git?</i>	<i>27</i>
6.1.2	<i>How can i use GitLab?</i>	<i>27</i>
6.2	GIT BASH	34
6.3	WHY DOES THE WORKSHOP USE GIT?	35
6.4	ROBOT OPERATING SYSTEM (ROS).....	35
6.4.1	<i>ROS components</i>	<i>36</i>
6.4.2	<i>Important ROS concepts.....</i>	<i>37</i>
6.4.3	<i>ROS Terminology</i>	<i>37</i>
6.4.4	<i>Message communication</i>	<i>39</i>
6.4.5	<i>ROS Tutorials</i>	<i>40</i>
6.5	GAZEBO.....	41
6.6	RVIZ.....	41
6.7	PYTHON.....	41
6.8	OPENCV.....	42
6.9	NANO	42
6.10	VISUAL STUDIO CODE.....	43
6.11	OBS STUDIO	43
6.12	SIMPLE SCREEN RECORDER.....	44

7	PROGRAMMING GUIDELINES.....	45
7.1.1	<i>Motivation and application.....</i>	45
7.1.2	<i>Unity within the Group.....</i>	45
7.1.3	<i>Identifier.....</i>	45
7.1.4	<i>Formatting.....</i>	46
7.1.5	<i>Commenting.....</i>	46
7.2	COMMON MISTAKES	47
7.2.1	<i>Python mistakes.....</i>	47
7.2.2	<i>Python Debugging.....</i>	47
7.2.3	<i>ROS Troubleshooting.....</i>	47
8	FURTHER DOCUMENTS	48
9	APPENDIX	49

1 Foreword

Welcome to your team for the Seamless Engineering workshop. Your task over the next few weeks will be to creatively work on a logistics system consisting of software and hardware components. As you will be developing a new type of system in this workshop, you will be assisted by experienced developers (tutors).

You will get to know modern logistics hardware with your team members. This includes, among other things, robotic arms and automated guided vehicles. You will first control these in a simulation and then put the real hardware into operation to solve a logistics problem.

We wish you a lot of fun with this task and look forward to the next weeks!

2 Content of the workshop

2.1 About this manual

This document is divided into several large sections. In the first part you will learn everything you need to know about the structure and procedure of the workshop. Then, in the chapter Preparation, you will learn how to set everything up and ready for working. With this knowledge, you are then introduced to the task including the hardware used in the workshop. All the tools that you will need to successfully finish this workshop are explained afterwards. Finally, the manual finishes with programming guidelines and further documents

2.2 Structure and procedure

The workshop consists of the following components:

- Introduction meetings

The workshop uses several very practical tools that you may not have had to work with before. How to set them up properly and work with them effectively will be demonstrated to you in the introduction meetings. Therefore, we strongly recommend that you attend the introduction meetings with your notebook. The office hours are no substitute and no alternative!

- Working on the tasks

There are no compulsory dates when you have to complete the tasks. Ensure a fair distribution within your group and respect the deadlines and milestones of the workshop.

- Lectures

The workshop will be accompanied at the beginning by lectures that will help you progress through the workshop.

- Consultation hours

The weekly tutor consultations are designed to help you successfully complete the practical course. Please note, however, that the tutors will not write the code for you.

If you would like to come to the consultation hours, you must register for your desired slot in advance. Registration for the consultation hours takes place via ILIAS. The number of participants in the slots is limited due to the number of tutors. Please also note that booked appointments should also be kept or must be cancelled in good time before the appointment.

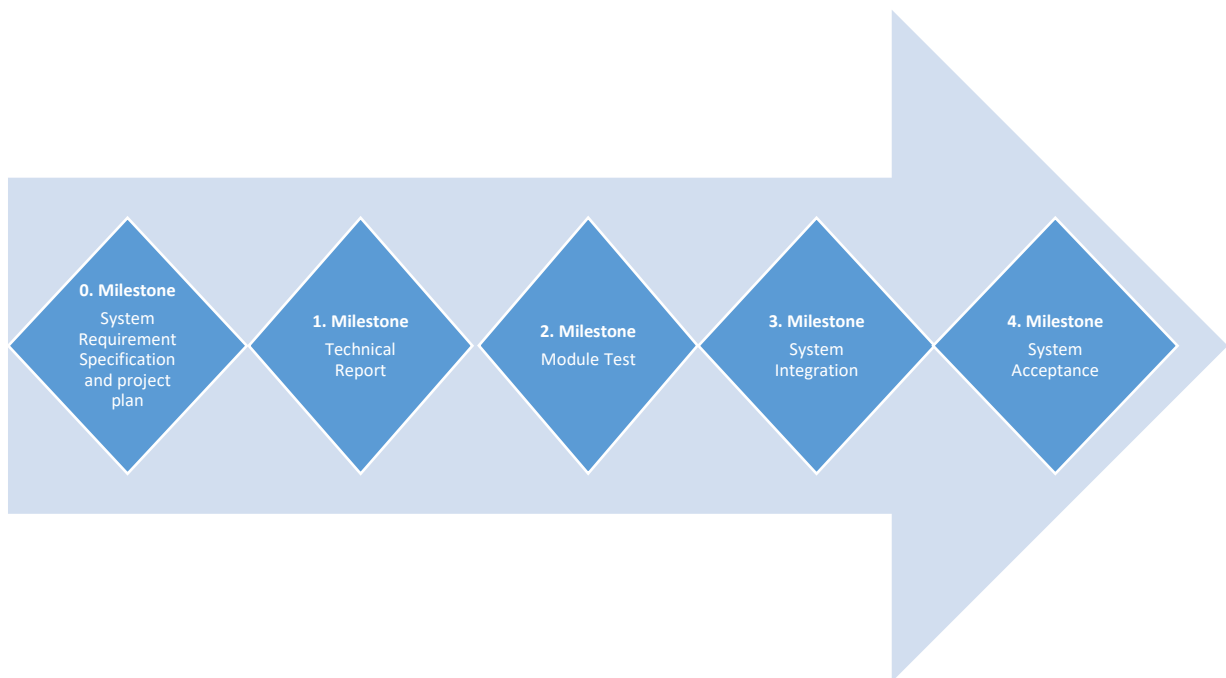
- Milestones

Throughout the workshop you will work with your group on milestones. The milestones are linked to submissions by the whole group. The exact times of the milestones will be given to you in the introduction meetings.

- Exam

The lecture block at the beginning of the workshop ends with an exam.

2.3 Milestones and submissions



All submissions must be uploaded to your group in ILIAS by Sunday 23:59 of the respective submission week. Milestones 2 to 4 also include a colloquium that takes place the week after the submission.

0. Milestone: System requirement specification and project plan

Project plan

The project plan in the form of a Gantt chart serves as an overview of all activities during the project and assigns tasks and deadlines to individual team members.

SWOT Analysis, a risk analysis

1. Milestone: Technical report

The first milestone is about defining your system architecture. Your task is to write a technical report that specifies possible use cases, system requirement specification and the system structure of your design. The technical report includes the following documents:

- System requirement specification. The client has created a list of functional and non-functional requirements in the system requirement specification. It is your task to transfer these requirements into a system requirement specification which describes how you want to fulfill these requirements in a technical way.
- Use Case Diagram
- Description of material flow

2. Milestone: Module Test

The second milestone is to demonstrate how individual actuators and sensors interact in the simulation environment. The focus here should be on the presentation of module tests in the simulation environment. For the demonstration, it is recommended to create a presentation with videos that show the individual functional units in action.

Concrete tasks:

- Turtlebot navigates autonomous between stations (video)
- Turtlebot can position itself at station using markers (video)
- Conveyor moves for specific length (video)
- Preparation of the results in the form of a presentation (10-15 minutes)

3. Milestone: System Integration

The third milestone is about transferring the results from the simulation to the real hardware. In a first step the flow of good takes place autonomously in the simulation. After that the transfer to the real hardware has to be realized on module level. For the demonstration, it is recommended to create a presentation with videos.

Concrete tasks:

- The flow of goods takes place autonomously in simulation (video)
- Transfer modules to real hardware (videos)
 - Turtlebot navigates between stations in real hardware
 - Turtlebot can position itself using markers
 - uArm can grasp different objects from Turtlebot and put them on conveyor
 - Preparation of results in the form of a presentation (10-15 minutes)

4. Milestone: System Acceptance

The fourth milestone is the system acceptance. On the real hardware, you demonstrate that your developed system is able to autonomously carry out a logistical flow of goods and meets the system requirements defined by the customer (See chapter 5.3). On the real demonstrator, you can show that your system is capable of handling any picking orders flexibly and efficiently.

Concrete tasks:

- The flow of goods takes place autonomously on the real hardware (video)
- Preparation of system acceptance test showing the whole flow of goods on real hardware at IfL workstation in a live demonstration

2.4 Evaluation of the workshop

The workshop is evaluated as follows:

- 50 % Exam. In the exam after the lecture blocks, the lecture contents are tested.
- 50 % Evaluation of the milestones:
 - 0. Milestone: Does not contribute to the overall grade à Submission mandatory
 - 1. Milestone: 10 % of the overall grade
 - 2. Milestone: 10 % of the overall grade
 - 3. Milestone: 10 % of the overall grade
 - 4. Milestone: 20 % of the overall grade

2.5 Current information, dates and deadlines

Necessary documents such as this handbook, as well as organizational and content-related information can be found in the ILIAS course.

The booking of the workstations with the hardware and the booking of the consultation hours takes place via ILIAS.

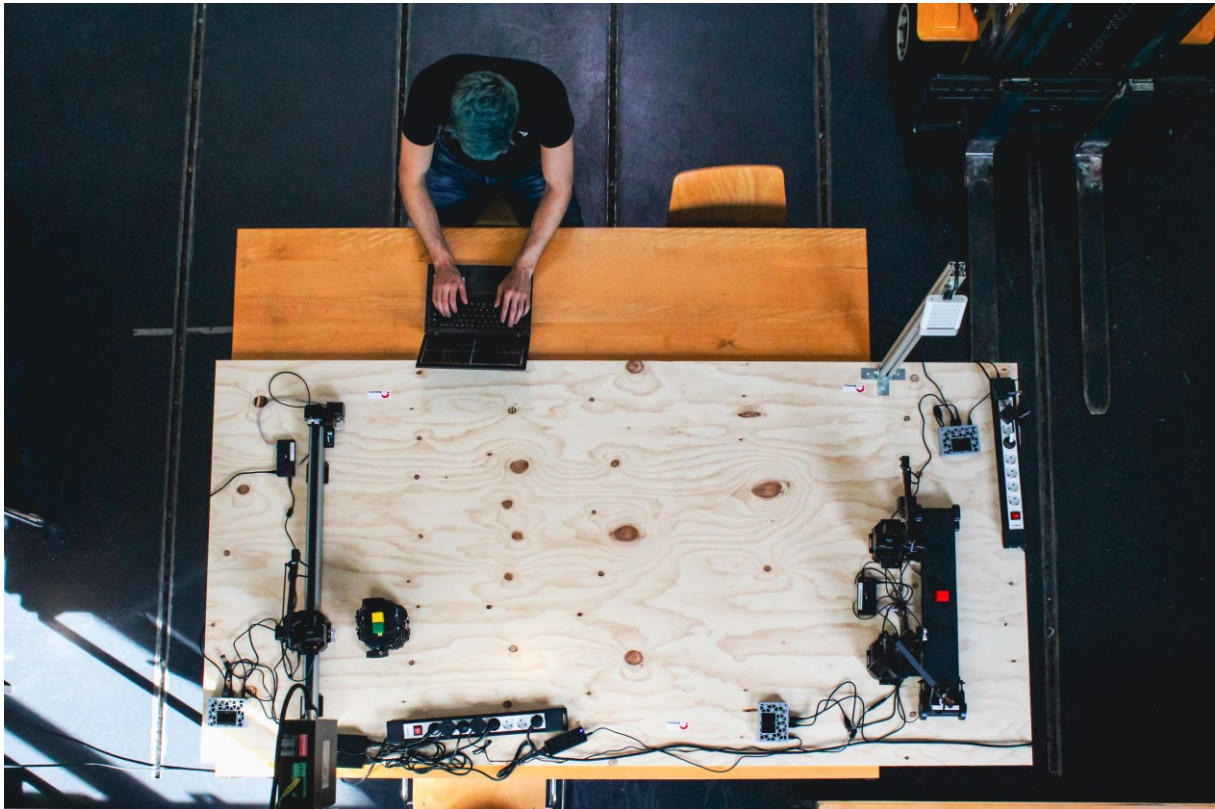


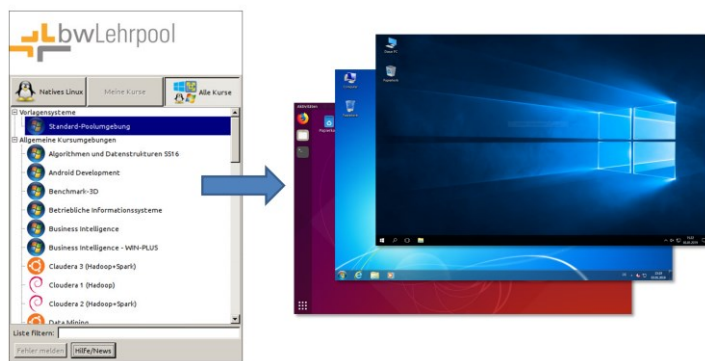
Figure 1: Workstation at the IFL

3 Preparation

3.1 What is bwLehrpool and why is it needed for Seamless Engineering?

BwLehrpool enables virtual teaching through the flexible and simple creation of virtual machines that are immediately available in either all or specific pool rooms of different universities. To read more about bwLehrpool visit:

https://www.bwlehrpool.de/wiki/doku.php/en/allgemein/was_ist_bwlehrpool



Seamless Engineering uses bwLehrpool for the easy deployment of our already set up virtual machine for the whole course. “Set up” in this context means that the virtual machine has everything preinstalled that you will need for the next weeks. This includes:

- Ubuntu 20.04
- Robotic Operating System (ROS)
 - Gazebo
 - rViz
- Python 3
 - OpenCV
- Visual Studio Code

A deeper description of the installed software can be found in chapter 0.

3.2 Using bwLehrpool for Seamless Engineering

The SCC pool computers are available to you for processing the workshop. You can use these remotely with your browser. The address for accessing the computers is: <https://pool-remote.scc.kit.edu>

The image shows two side-by-side screenshots of the bwLehrpool web interface. The left screenshot is the login page, featuring the bwLehrpool logo, a username field containing 'pi1102', a password field with masked characters, and a large 'Anmelden' button. The right screenshot shows the room selection page titled 'Wähle einen Raum aus'. It includes a note about browser window size and a table of available rooms. The 'SCC' row is highlighted in green.

Wähle einen Raum aus		
Stelle sicher, dass das Browser Fenster die gewünschte Größe hat. Die Auflösung des Clients wird dem entsprechend gesetzt.		
ITIV-Pool	2 verfügbar (1 offline)	
SCC	49 verfügbar (37 offline)	
IMI	0 verfügbar (15 offline)	
IMS	0 verfügbar (6 offline)	Passwortgeschützt
IFL	10 verfügbar (0 offline)	Passwortgeschützt
IMS-8	0 verfügbar (1 offline)	Passwortgeschützt

Buttons at the bottom: Ausloggen, Weiter

Then select the course "Seamless Engineering".

Important!

After closing the virtual machine, it is reset to its original state. All files that have not been saved are lost.

To save your own code in the virtual machine, a personal memory has been created for you at the SCC. You can find this under the following path: `/home/student/PERSISTENT_mov`

The simulation environment in the virtual machine

The simulation environment is already available for you in the virtual machine. You will find it under the following path: `/home/student/2021_xxxxx`

Here you will find the code to start the simulation environment and to load the hardware.

3.3 Linux – Ubuntu

Linux is a family of open-source operating systems based on the Linux kernel. It comes usually packed in a Linux distribution. A Linux distribution is a software collection based on the Linux Kernel and package management system and many more. Ubuntu is a Linux distribution based on Debian. Here we use Ubuntu 20.04 LTS.

For some parts of the lab you will need the so-called terminal. You will find it on the left side of the taskbar as shown in the Figure 2. Here you can launch various scripts and programs, as well as go through the file directory.

To make it easier for you to get started with the terminal, the most important commands are listed in Figure 2.



Figure 2 Linux getting started

Table 1: Linux Command Line

Command	Description
pwd	Show current directory
mkdir dir	Make directory dir
cd dir	Change directory to dir
cd ..	Go up a directory
ls	List files
rm file1	Delete file1
cp file1 file2	Copy file1 to file2
pdw	Show working directory

Source: <https://cheatography.com/davechild/cheat-sheets/linux-command-line/pdf/>

4 First steps in simulation environment

Starting the simulation environment on a prepared remote PC is a fairly simple process:

- first: open a terminal and navigate to the workspace's folder
- second: execute the *Start_Skript.sh* with the *Paramtert.sh* as argument (example terminal-command: `"bash Start_Skript.sh Parameter.sh"`)

Then the simulation environment should start and be visualized in RVIZ like in Figure 4. It is also possible to vizualize it in Gazebo like in Figure 3, but because of performance improvement, the simulation should be visualized in in RVIZ. After all models are spawned, you can start to work with the ROS-Topics.

Our start skript starts several ROS launchfiles using an application called *screen*. Open additional terminals and use screen to track the actual log data from the models for troubleshooting. You will find the necessary commands on <https://help.ubuntu.com/community/Screen>.

We also recommend you to get familiar with *rqt*. This tool can help you visualize, subscribe and manipulate all active nodes, topics, services and actions. You can use the tool to send action goals or call servies as well.

Gernal Advices:

To spawn a model individually, you can use following example command:

`"roslaunch MODEL_TYPE spawn.launch name:=NAME number:=NUMBER"` by pressing TAB you get the possible arguments printed.

To start a node you can use the command: `"roslaunch PACKAGE_NAME NODE_NAME"`

You can find example scripts in the documentation of the models.

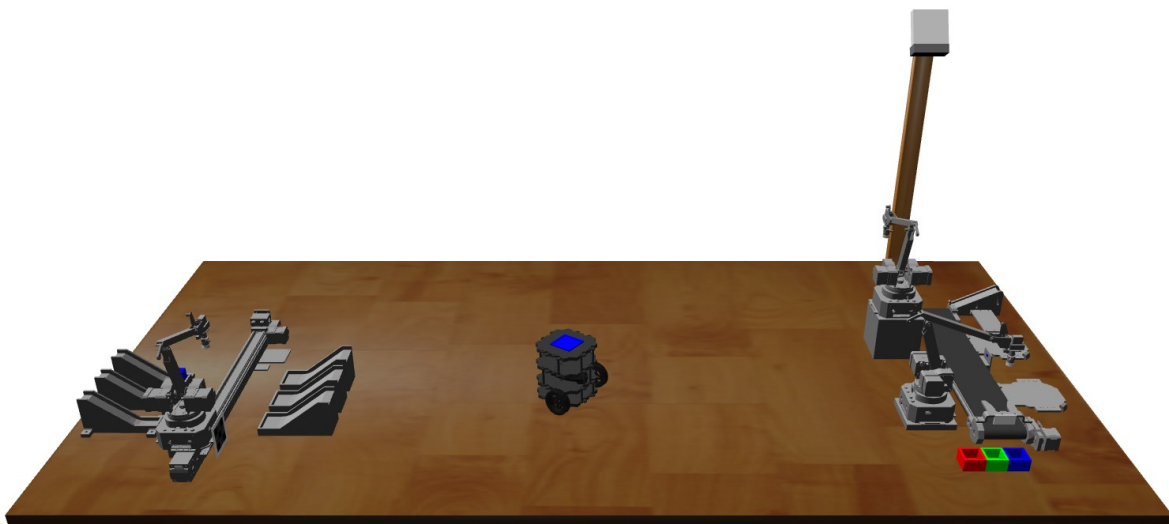


Figure 3: Gazebo Environment

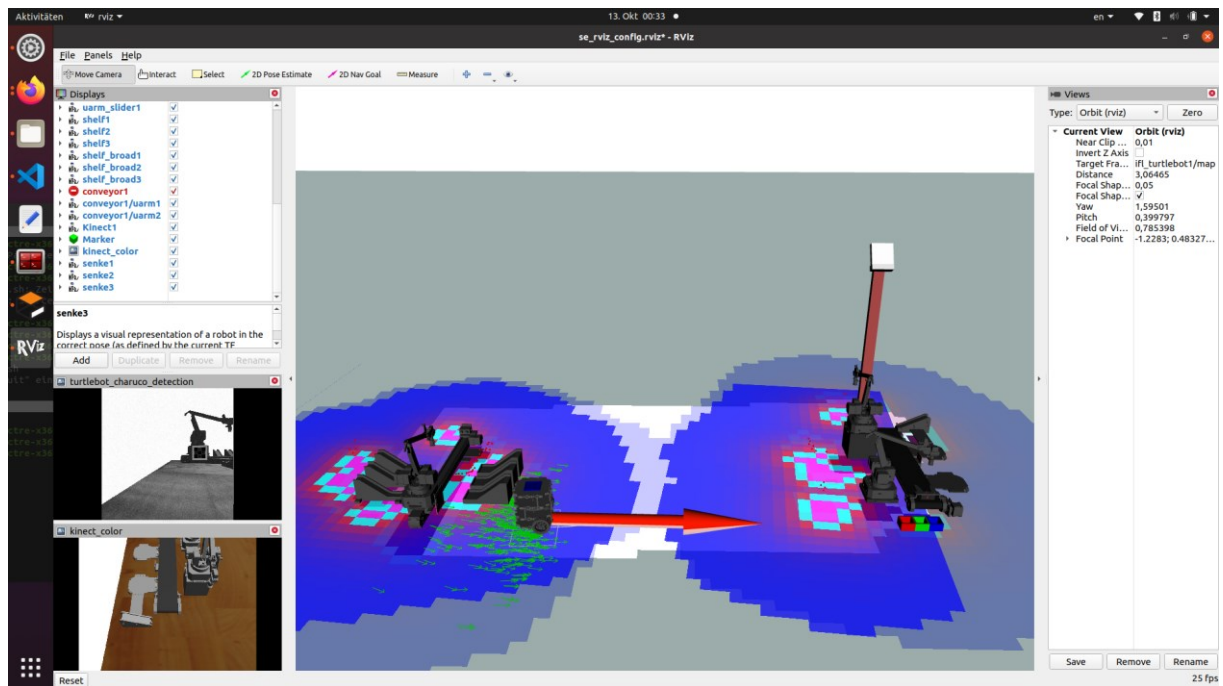


Figure 4: Rviz

5 Problem definition

5.1 Hints

This assignment was written with the aim of giving a good overview of the task. However, you are not expected to strictly follow a task and implement it 1:1, but to creatively deal with it and solve the requirements we have defined in the best possible way through research.

The following topics require that you have worked through the chapter Preparation in particular. You should also have an overview of the chapter Documents. The assignment does not have the function of explaining the required basics to you. Part of the workshop is for you to work out the required craft on your own - as is often necessary in practice.

5.2 Introduction of the task

Within the scope of the workshop, you will first solve a logistical task and develop an autonomous flow of goods purely simulative with the help of modern actuators for grasping and transporting objects, as well as with the help of sensors for the detection of objects. The task is based on an order picking system. The material flow system continuously receives new orders to deliver different objects in different colors and sizes. An example order could look as following:

- 1 orange object of size A
- 1 blue object of size A
- 1 orange object of size B

The system to be developed has to handle these sequentially incoming orders by picking the objects and putting them in boxes where one box represents one order.

After the system has been developed simulative, the next step is to go from simulation to real hardware. That means the code that you have implemented for the hardware in simulation should now run on the same real hardware. This real hardware is provided to you at workstations at the Institute for material handling and logistics.

5.2.1 Procedure of the flow of goods within the picking system

The material flow starts with a robotic arm (uArm Swift Pro) on a slider (uArm Slider) which has to pick objects according to the current order from different bins (see left side of Figure 3). Each bin contains a different object. The robotic arm then positions the objects on an automated guided vehicle (AGV: in our case a Turtlebot Burger) which has a defined loading area size. The AGV moves the objects to a conveyor belt (see right side of Figure 3). Here, the objects are placed on the conveyor belt with another robotic arm. The conveyor belt delivers the individual objects to the order disposition. At the order disposition, the objects are moved from the conveyor belt into stow boxes by a third robotic arm. Each stow box represents one order.

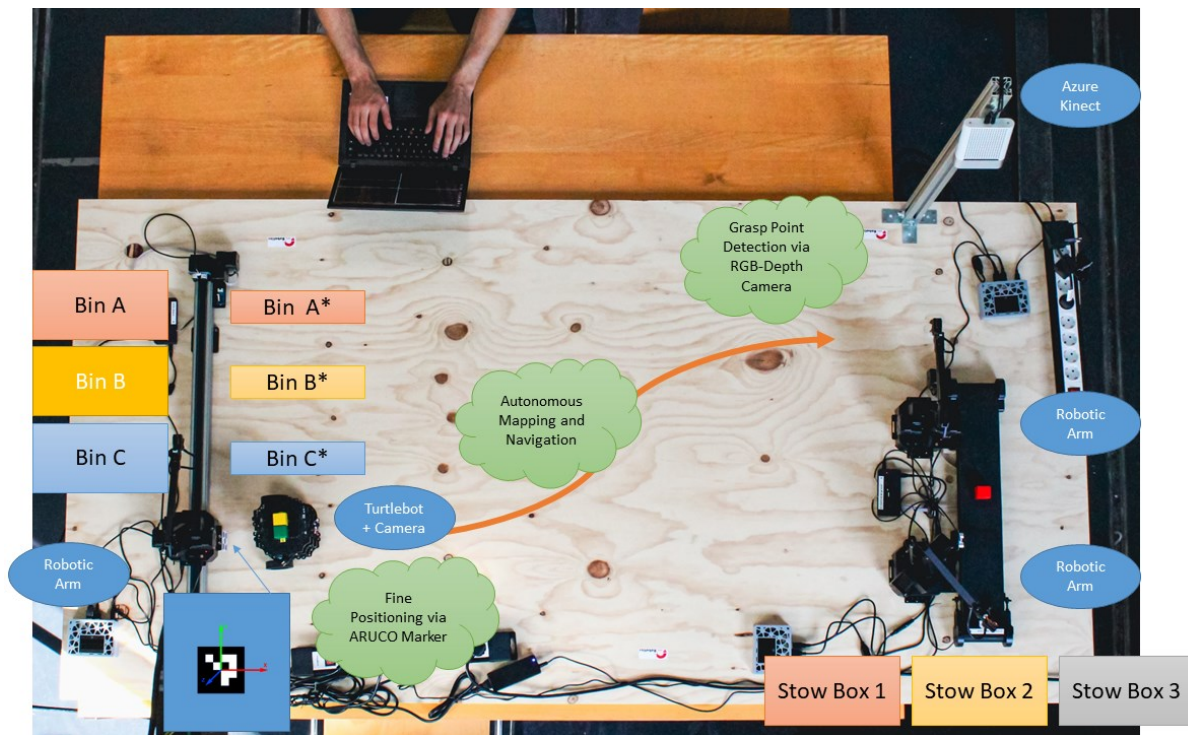


Figure 5: Structure of the system to be developed

5.3 System requirement specifications for the system to be developed

The Product Requirements for the system to be developed are summarized in the following tables. Your task is to use the following tables to create a System Requirement Specification (SRS) that serves as an elementary working basis for the project. Each Product Requirement can be refined by multiple System Requirements. In order to maintain traceability, each System Requirement must be clearly assignable to a Product Requirement. Therefore, no System Requirement should be created without an according Product Requirement.

Table 2: Functional Requirements

ID	Title	Description	Remark
F1	Throughput	The throughput of the system must be at least xx	
F2	Reliability	Objects must not be lost or damaged	
F3	Customer Satisfaction	The system must deliver the requested objects correctly	Examples for SRS Refinement: The system must detect the size and color of an object

			The system must localize the position of the shelf containing the object demanded by an order.
F4	Emergency stop	The system must stop all processes in case the Emergency stop button is pressed.	

Table 3:
Non-functional

requirements

ID	Title	Description	Remark
NF1	Easy usability of the system	The system must be intuitively usable	
NF2	Portability of the system	The system should be portable to the same hardware components without effort.	
NF3	Central settings area	All important settings should be centrally visible and adjustable	
NF4	Robustness against external interferences	External interference in the system (human intervention, etc.) should not lead to a failure of the system	
NF5	Optimized routes	The system should be resource efficient in terms of time and energy.	Example for SRS Refinement: The system should determine the shortest path between the loading area and the conveyor.
NF6	Efficient use of sensors and actuators	The total cost of logistics hardware must be as low as possible	
NF7	Scalability	The system should be expandable in terms of hardware and software	
NF8	Reusability	The system should be well and comprehensibly documented.	

5.4 Description of the most important key figures of conveyor lines

5.4.1 Throughput

On a conveyor line of length l , supply units (SU) move unhindered at speed v from a source (Q) to a sink (S). By making specific cuts - with sources or sinks at the interfaces - a networked material flow system can be broken down into simple conveyor sections.

The conveyor sections are not bound to a specific spatial position. For the throughput analysis, it is initially irrelevant which technical principle is used to move the conveyor units; only the knowledge of their speed-time behavior is important. In the simplest case, the throughput of a conveyor line is calculated from the speed v and the distance s between the conveyor units:

$$\lambda = \frac{v}{s} \left[\frac{1}{TVU} \right] \quad (s \text{ is the distance between the same points of two consecutive SUs, see also Figure 23})$$

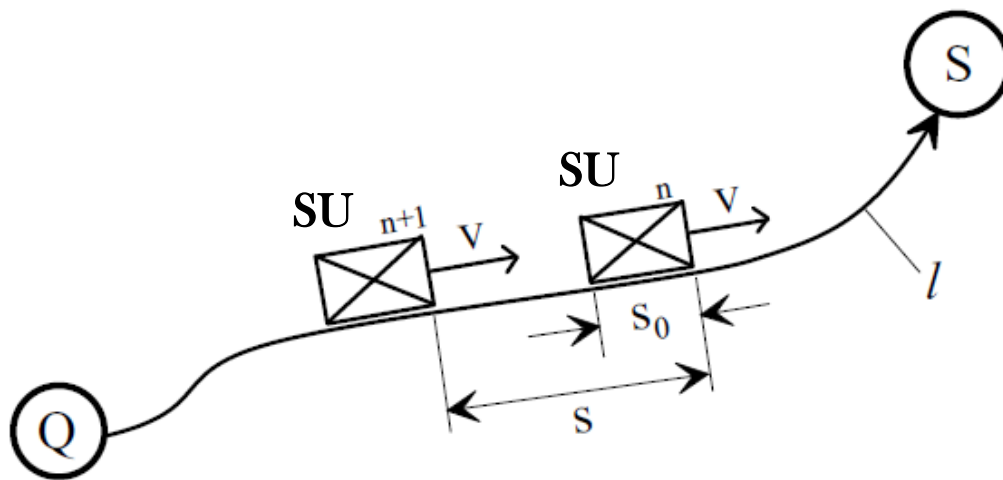


Figure 6: Supply units on a conveyor line of length l

The maximum throughput $\lambda_{Max} = \frac{\bar{v}}{s_0}$ is achieved when the conveyor units pass the conveyor line at maximum speed and without gaps, touching each other.

5.4.2 Picking time

Because of the direct dependence on consumption, a fast processing of the picking orders is usually desired.

The time for the processing of a picking order $r = 1, 2, \dots, n$ items is:

$$t_k = t_0 + \sum_{r=1}^n (t_{t,r} + t_{g,r} + t_{l,r})$$

Where:

t_0 = Base time: Time shares that are not dependent on the number of items, e.g. for providing and passing on a picking container, for receiving the order data, for printing out and attaching a label

$t_{t,r}$ = Dead time: Time shares to be spent for each position r at the withdrawal point in addition to the time for physical material movement, e.g. for reading, searching, numbers, acknowledging, correcting

$t_{g,r}$ = Grasp time: Time shares of the physical material movement, e.g. for picking up and putting down all items of a position

$t_{l,r}$ = Way time: Time shares for the movement of the order picker or the loading units

5.5 Presentation of the hardware

A variety of hardware components are available to you as part of the workshop. These include a variety of sensors as well as cameras and markers that you can use to solve the logistics problem.

5.5.1 uArm Swift Pro



Figure 7: uArm with vacuum gripper

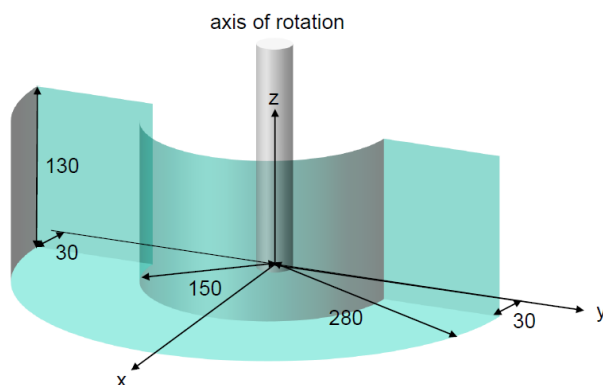


Figure 8: Workspace [mm] of an uArm on the ground

The uArm Educational Kit provides the central components of the workshop. The uArm Swift Pro is a robotic arm with four degrees of freedom. In our scenario it is equipped with a vacuum gripper. The workspace [mm] of the robot arm standing on the ground is shown in Figure 8. Placed in an elevated position it is also capable of reaching up to $z = -120$ mm below its platform. In addition to the three cartesian coordinates it is also capable of rotating the tool center point which is represented by the coordinate w . Also take account of, that the workspace of a robot arm is not cylindrical. So it's preferable not to push to the limits of its radius.

Workspace of the uArm Swift Pro:

- $30 \leq x \leq 280$ [mm]
- $-278 \leq y \leq 278$ [mm]
- $-120 \leq z \leq 130$ [mm]
- $0 \leq w \leq 180$ [degree]
- $150 \leq r \leq 280$ [mm]

Exemplary code and further instructions:

In the model's folder in our catkin workspaces
(~/2021_pnp_ws/src/models/uarm/scripts or ~/jetson_ws/src/models/uarm/scripts)
you will find a python script *very_simple_action_client_example.py*, explaining how to call the `uarm_move` action and the `uarm_set_pump` service.

Startup:

Simulation:

The following terminal command spawns a uArm model in the Gazebo environment and starts its controller nodes. It also optionally starts gazebo and can modify the model's name, number and coordinates. The parameters after launch are only necessary, if you want to change their values. The coordinate Y is the rotation around the z axis.

```
roslaunch uarm spawn.launch start_gazebo:=false name:=uarm number:=1 x:=0 y:=0 z:=0 Y:=0
```

Real model:

```
roslaunch uarm real.launch name:=uarm number:=1
```

5.5.2 uArm Slider



Figure 9: uArm Slider - simulation



Figure 10: uArm Slider - simulation model coordinate system (x/y/z)->(r/g/b)

Data:

- workspace [mm]: $-470 \leq y \leq 0$
- platform height [mm]: 70
- raised by another 70 mm in our configuration!
- sensors (only real model): ultrasonic sensor, limit switch

The uArm Slider is a wagon on a rail powered by a stepper motor. The position of the wagon can be determined by an ultrasonic sensor (here on the right in Figure 9). The model's coordinate system is introduced like shown in Figure 10 with the green arrow pointing in y direction.

Differences to the real model:

The real model of the uArm slider has a longer rail and the wagon's position is determined by an ultrasonic sensor (sensor is in Figure 9 on the right). To ease the transition from simulation to real world, we don't use the whole possible workspace on the larger rail, but kept the workspace of 470 mm. Furthermore, the model's coordinate system is introduced like in the simulation with one modification: In the simulation at position -470 the wagon is right at the ultrasonic sensor. Since the real ultrasonic sensor is unprecise at distances below 40 mm, we moved the coordinate system further away from the ultrasonic sensor, so it keeps a certain distance to the sensor even at position -470. This distance is determined by the variable *shift* which defaults to 40 mm.

In addition to the ultrasonic sensor, there is a limit switch placed near the sensor as well. We provide a ROS action *drive_to_limit_switch* to drive towards it until it is triggered.

Startup:

Simulation:

The following terminal command spawns a uArm Slider model in the gazebo environment and starts its controller nodes. It also optionally starts gazebo and can modify the model's name, number and coordinates. The parameters after launch are only necessary, if you want to change their values. The coordinate Y is the rotation around the z axis.

```
roslaunch slider spawn.launch start_gazebo:=false name:=slider number:=1 x:=0 y:=0 z:=0 Y:=0
```

Real model:

```
roslaunch slider real.launch name:=slider number:=1
```

Exemplary code and further instructions:

In the model's folder in our catkin workspaces

(~/2021_pnp_ws/src/models/slider/scripts or ~/jetson_ws/src/models/slider/scripts)

you will find a python script *very_simple_action_client_example.py*, explaining how to call the slider_move action.



Figure 11: uArm mounted on uArm Slider

5.5.3 uArm Conveyor

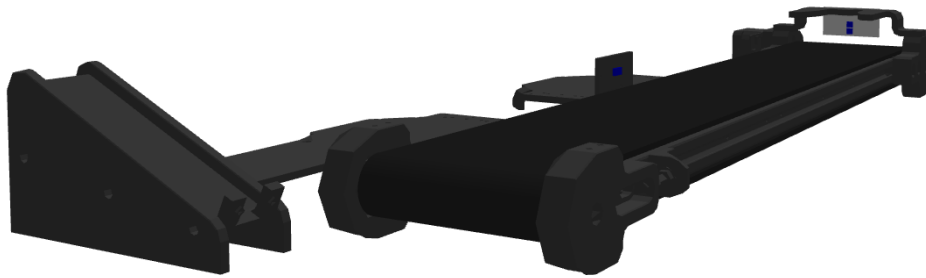


Figure 12: uArm Conveyor model - simulation

Robotis offers a buffering conveyor in the uArm's product line, as well. In our configuration it is equipped with sensors to detect objects near the stopper and at about the middle of the conveyor belt. The real model uses an ultrasonic sensor to detect objects along the conveyor belt and a color sensor at the stopper (topic: *NAMESPACE/state*). In the simulation we substituted the ultrasonic sensor with a laser sensor (*NAMESPACE/ultrasonic/scan*). At the stopper instead of the color sensor, we use a camera (*NAMESPACE/camera/image_raw*) and additionally attached another laser sensor (*NAMESPACE/ultrasonic_2/scan*) to it, so you can stop the objects at a certain distance to the stopper, because the cargo box colliding with the stopper while picking it up, is a common source of problems in the simulation.

Since the real model is not real-time capable, we provide an extra function *search_mode* to stop as quick as possible when the ultrasonic sensor detects an object.

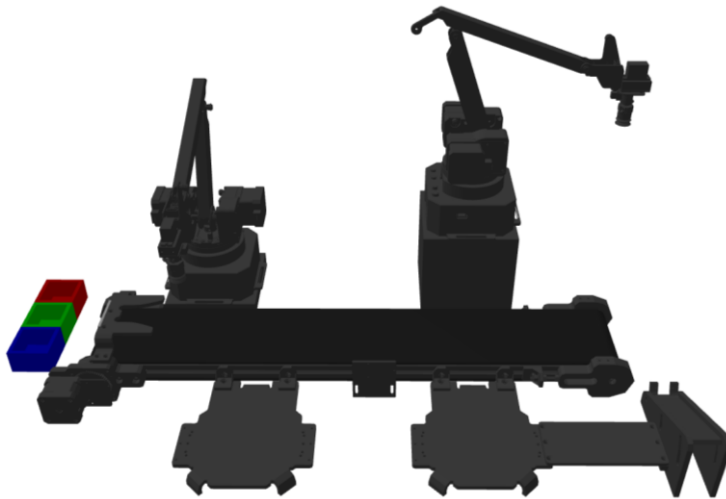


Figure 13: uArm Conveyor - SE-Configuration

Startup:

Simulation:

The following terminal command spawns a uArm Conveyor model in the SE-Configuration like in Figure 13 in the gazebo environment and starts its controller nodes. It also optionally starts gazebo and can modify the model's name, number and coordinates. The parameters after launch are only necessary, if you want to change their values.

The coordinate Y is the rotation around the z axis.

```
roslaunch conveyor spawn.launch start_gazebo:=false name:=conveyor number:=1 x:=0 y:=0 z:=0  
Y:=0
```

Real model:

```
roslaunch conveyor real.launch name:=conveyor number:=1
```

Exemplary code and further instructions:

In model's folder in our jetson catkin workspace (jetson_ws/src/models/conveyor/scripts) you will find a python script *very_example.py*, explaining how to call the *slider_move* action.

5.5.4 Tuttlebot3 Burger

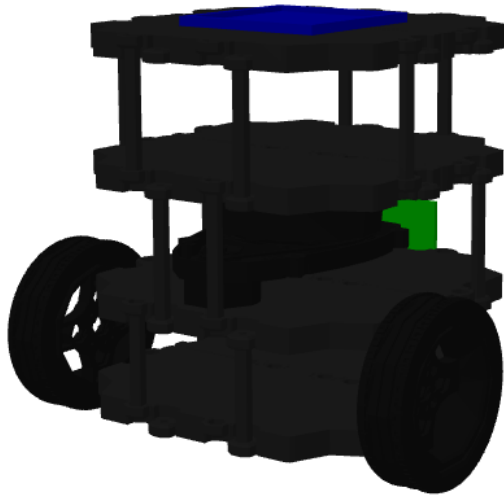


Figure 14: Turtlebot3 Burger

The Turtlebot3 Burger is a versatile robot differential drive platform for educational purposes and the standard robot for ROS. It is powered by a Raspberry Pi 3 and an OpenCR Controller. In our setting it is equipped with a 360 lidar sensor and a Raspberry Pi Camera (position in Figure 16).

Tutorials:

Being the standard robot platform for ROS there are plenty of ready to use ROS packages available. Robotis provides great beginner tutorials to get familiar with the Turtlebot.

<https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>

Especially important guides:

- Simulation
- SLAM (gmapping)
- Navigation
- Bringup (real model)

Most important commands from Turtlebot 3 tutorials (each in new terminal):

- start Gazebo with Turtlebot 3 World:
`roslaunch turtlebot3_gazebo turtlebot3_world.launch`
- control TB3 with keyboard:
`roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`
- start SLAM:
`roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping`
- save map:
`roslaunch map_server map_saver -f ~/map`
(you might want to edit path and name)
- start navigation:
`roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml`
(edit path to your map.yaml from step SLAM)

Startup:

Simulation:

Starting our gazebo environment, automatically starts a navigation node as well. So, you only have to call the *ifl_turtlebot1/move_base_simple* action to set a target position.

Start Real Model:

- To use the real model, you can follow the Robotis tutorials.
At first you have to establish a SSH connection with the Turtlebot.
name: pi, pw: turtlebot
- Next, you have to make sure a roscore is active on the main jetson.
- Then you can start the robot.launch and the camera node on the turtlebot:
roslaunch turtlebot3_bringup turtlebot3_robot.launch
and
roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch

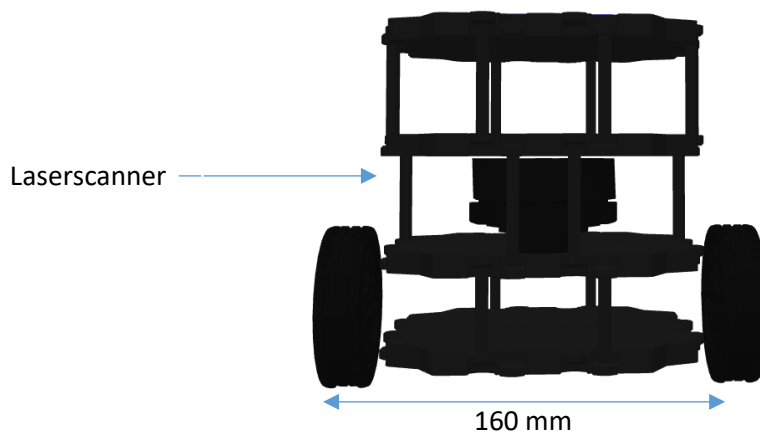
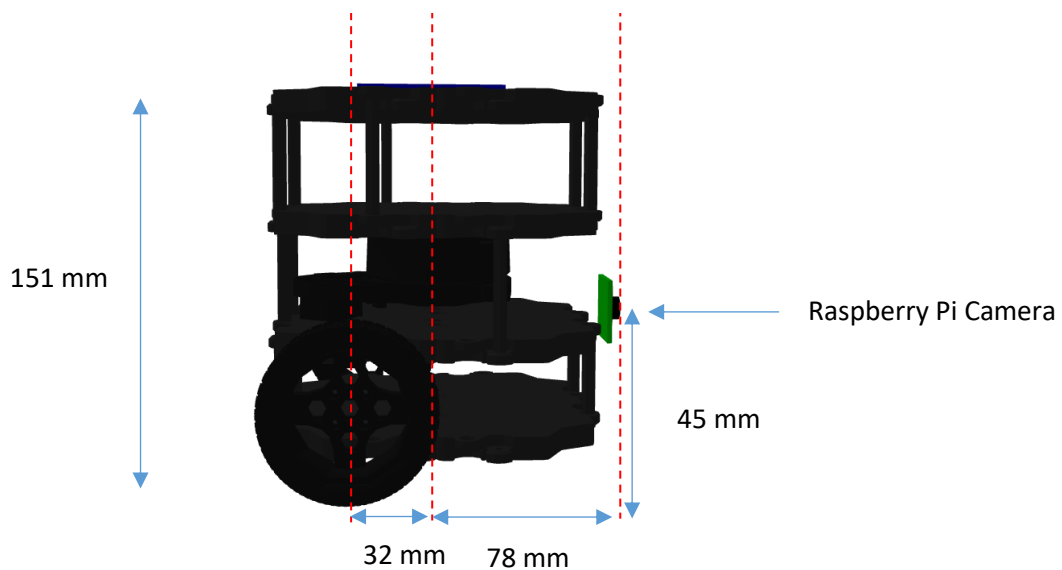


Figure 15: Turtlebot3 Front



5.6 Implementation on real hardware

After you have controlled the hardware in a simulation and the flow of goods works as desired, you can now put the real hardware into operation. The following section teaches you how to handle the real hardware and shows you how to check its functionality.

There are slight differences that must be taken account to compared to the simulation:

For example, after starting a model's server with ROS, you must wait until the server identified the model before entering commands. Look out for messages containing "... identified on port ..." and in case of the uArm "* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)" in the terminal.

For further information take a look into the doc folders. Those instructions also contain the most common sources of errors you will come across working with them. Most common source of errors

Figure 16: Turtlebot3 Side

are loose cables. Check them and verify by using the buttons on the Arduino controllers.

Further information for the start up of the hardware will be provided.

5.7 Important Topics, Actions and Services of the hardware

Table 4: Important Actions of the uArm Swift Pro

NAMESPACE/uarm_move	moves to target position
NAMESPACE/uarm_reset	Reset to position [180, 0, 130]

Table 5: Important Services of the uArm Swift Pro

NAMESPACE/uarm_set_pump/	Pump of vacuum gripper true/false
NAMESPACE/uarm_get_state/	Gives position, busy and pump status

Table 6: Important Topics of the uArm Swift Pro

NAMESPACE/busy	True, if currently working on task
----------------	------------------------------------

Table 7: Important Actions of the uArm Slider

NAMESPACE/slider_move	moves to target position y [-470,0]
NAMESPACE/drive_to_limit_switch	drives until detected by limit switch (only real model)

Table 8: Important Services of the uArm Slider

uarm_slider1/uarm_set_pump	pump of vacuum gripper true / false
uarm_slider1/uarm_get_state	gives position, busy and pump status

Table 9: Important topics of the uArm slider

uarm_slider1/uarm/busy	true if uArm currently working on task (sim)
uarm_slider1/slider/busy	true if slider currently working on task (sim)
uarm_slider1/state_raw	Raw and additional information from slider before transformed to fit the simulation (real model)

Table 10: Important actions of the turtlebot3

NAMESPACE /move_base_simple	moves to target position
-----------------------------	--------------------------

Table 11: Important topics of the turtlebot3

ifl_turtlebot1/camera/image	image of camera (in simulation)
ifl_turtlebot1/camera/image_charuco_pose	pose of recognized marker relative to camera (origin in bottom right corner of marker) (in simulation)
raspicam_node/image	image of camera (real model)
Raspicam_node/image_charuco_pose	pose of recognized marker relative to camera (origin in bottom right corner of marker, real model)

Table 12: Important services for uArm Conveyor

conveyor/conveyor/control	move conveyor belt (-15.0,0,15.0) (simulation)
NAMESPACE/move_forward (Int16 data)	moves belt forward with the duration of data (defaults to 0 – 0 means infinite) (real model)
NAMESPACE/move_backward (Int16 data)	moves belt backward with the duration of data (defaults to 0 – 0 means infinite) (real model)
NAMESPACE/stop	stops belt (real model)
NAMESPACE/start_search	stops the conveyor when the ultrasonic sensor detects an object. Deactivates itself when object detected.
NAMESPACE/stop_search	stop search mode

Table 13: Important topics for uArm Conveyor

NAMESPACE/state	Information about belt, mode, color sensor, time moving since last start command and (feed – not attached here)
-----------------	---

NAMESPACE/busy	Active while search mode activated
NAMESPACE/ultrasonic/scan	Ultrasonic sensor positioned beside the belt (simulation) [m]
NAMESPACE/ultrasonic_2/scan	Ultrasonic sensor positioned at the stopper (simulation) [m]
NAMESPACE/camera/image_raw	camera positioned at the stopper (simulation)

5.8 Example for the control of uArm

You set goal for offered ROS actions using in the terminal. The following figure shows how to set a target position for the uArm, setting the action goal of the *uarm_move* action to [180,0,110]

```
student@ubuntu2004:~/pnp_gazebo_ws$ rostopic pub /uarm_slider1/uarm_move/goal uarm_msgs/uarm_moveActionGoal "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
goal_id:
  stamp:
    secs: 0
    nsecs: 0
  id: ''
goal:
  x: 180
  y: 0
  z: 110"
publishing and latching message. Press ctrl-C to terminate
```

Within the uArm's terminal you will see that the server reacts to your request:

```
[INFO] [1615131692.879723, 460.898000]: uarm_slider1/: new goal received [180, 0, 110]
[INFO] [1615131693.226827, 460.982000]: uarm_slider1/: new goal reached [180, 0, 110]
```

In `/models/uarm/scripts/very_simple_action_client_example.py` you will find an exemplary script to call the *uarm_move* action as well as the *set_pump* service using python 3.

6 Tooling

6.1 GitLab

6.1.1 What is git?

Put simply, Git is the cloud for programmers. As soon as several developers work on the same source code (which is stored in a so-called "repository"), it is necessary to manage and merge the contributions ("commits") of the individual participants. In addition, Git allows for any number of development branches. For example, experimental functions can be tested in a branch. As soon as these functions are in a stable state, they can be merged with another branch. In doing so, the differences between the two source codes are compared and the most up-to-date parts are taken over. Another very important and practical feature is that every commit, i.e. every change to the source code, is saved. This means that it is always possible to return to any previous version without restriction.

Git offers - also thanks to the many extensions - numerous other functions. For daily use, however, Git as a command-line program is rather cumbersome, which is why a variety of platforms based on Git are freely available. First of all, these platforms provide a server with which developers can synchronize source code. They also provide a user interface. Almost all platforms offer an "issue" system (an issue is also called a "ticket"). This is a kind of forum for the repository with which bugs and possible improvements can be pointed out and discussed. The repository with the source files, the issue system and any other functions together form a project.

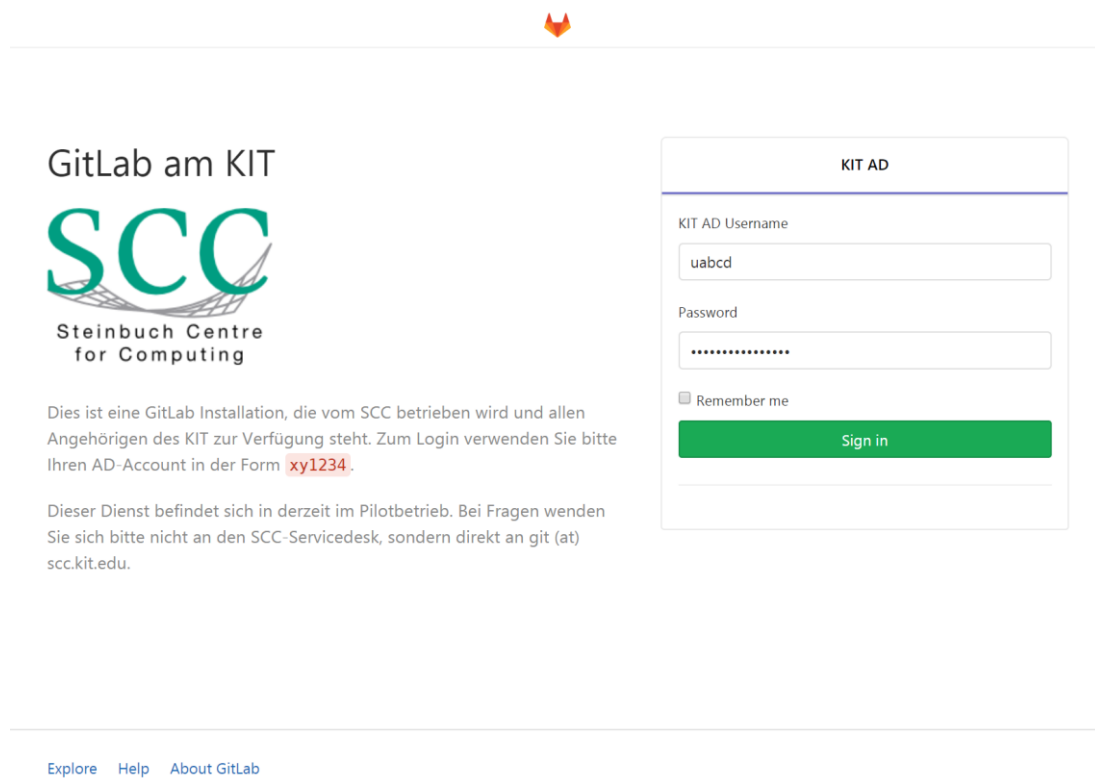
All these platforms do not touch the core, namely Git itself. This means that, as a rule, every client (i.e. the program on your computer with which you upload and download the code) is compatible with every platform. The GitLab used at KIT, which we use in the workshop, is aimed at software development within companies. It supports common user account management, so you can log in with your KIT account. In the context of this workshop, you only need a fraction of the available functions. You also only need to set up the minimum; as far as possible, everything has been pre-configured for you.

6.1.2 How can i use GitLab?

To use GitLab, you need to set up your GitLab account and a program that allows you to synchronize your group's repository with your computer.

6.1.2.1 Creating an account

Your GitLab account is automatically created the first time you log in. To do this, visit <https://git.scc.kit.edu> and log in with your SCC account.



The image shows the GitLab login page for the Steinbuch Centre for Computing (SCC) at KIT. On the left, the SCC logo is displayed above the text "GitLab am KIT". Below this, a paragraph explains that the installation is operated by the SCC and available to all KIT members, with login credentials in the form of `xy1234`. Another paragraph mentions the pilot status and provides contact information for the SCC-Servicedesk. On the right, the "KIT AD" login form is shown, featuring fields for "KIT AD Username" (containing "uabcd") and "Password" (masked with dots). There is a "Remember me" checkbox and a green "Sign in" button. At the bottom, navigation links for "Explore", "Help", and "About GitLab" are visible.

Figure 17: GitLab Login site

A GitLab account will automatically be created for you and you will be redirected to the start page.

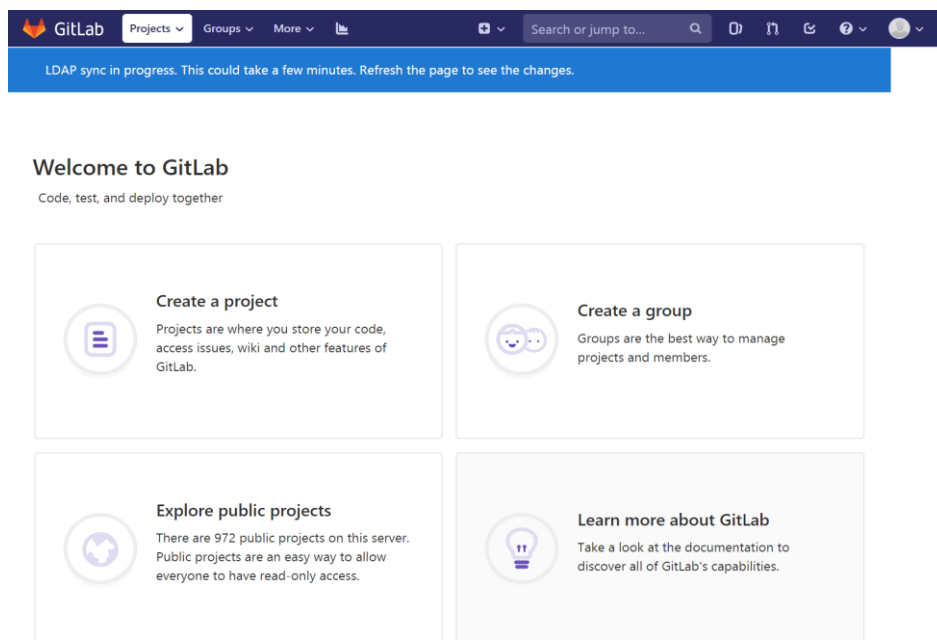


Figure 18: GitLab after login

If you receive a warning regarding SSH, you can ignore it as you do not need this function.

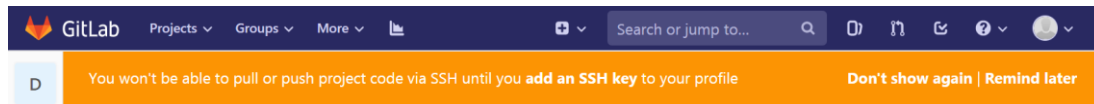


Figure 19: SSH Key Warning GitLab

6.1.2.2 Preferences

You can access the preferences via the selection menu in the top right-hand corner.

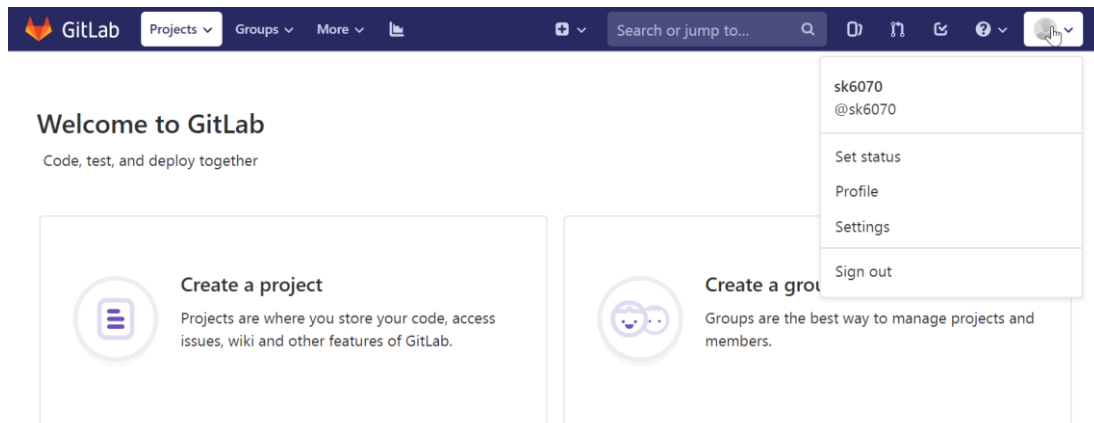


Figure 20: Access to the GitLab preferences

There you can change your user name or your profile picture, among other things. In the bar on the right, you will find numerous other settings pages. Under "Preferences", for example, you can change the appearance and language of GitLab.

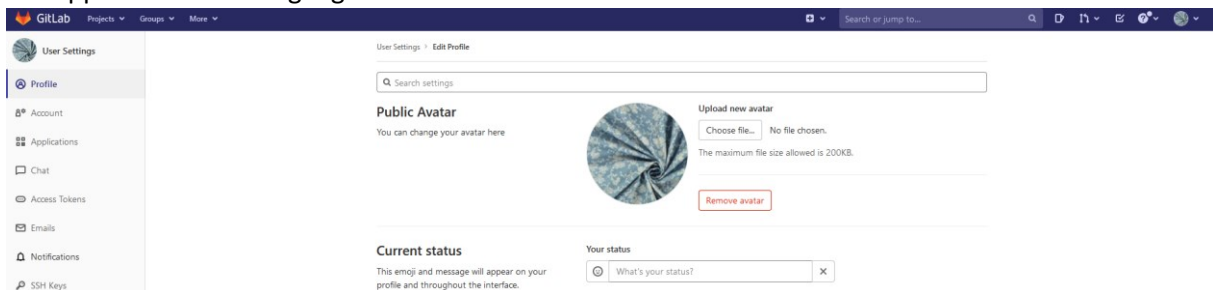


Figure 21: GitLab Preferences

6.1.2.3 Your project

After grouping, you will find a ready-made Git project for your group in the workshop group (see above). To synchronize this repository, you need the HTTPS link shown under "Clone".

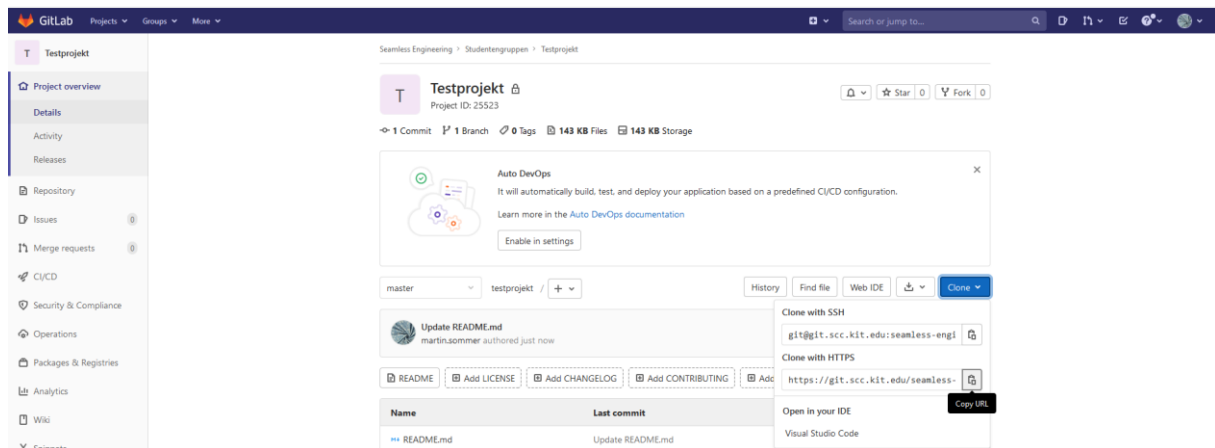


Figure 22: HTTPS Link for cloning in GitLab

6.1.2.4 Installing GitHub Desktop

Of the numerous clients available, one free program stands out for its simplicity: GitHub Desktop. As the name suggests, this client is actually designed to work with the GitHub platform, but it can also work with any other platform.

You can download the program for macOS and Windows: <https://desktop.github.com/>.

During installation, you will be asked for a GitHub account. If you don't have one, or don't want to provide one, skip this step.

If you do not have an account, you will be asked to specify the name and email address under which your commits should be submitted. Both name and email address should be identical to the information on GitLab, but they do not have to be.

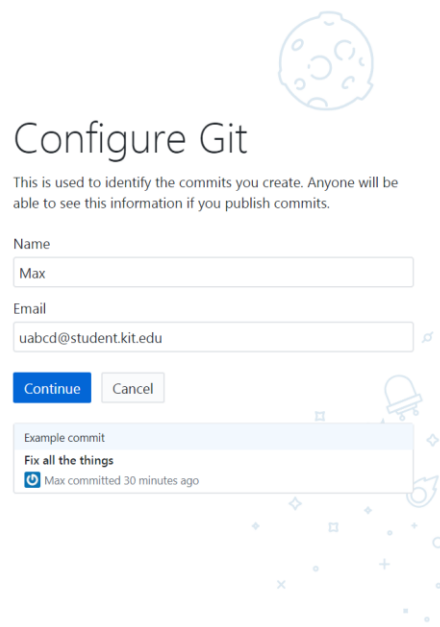
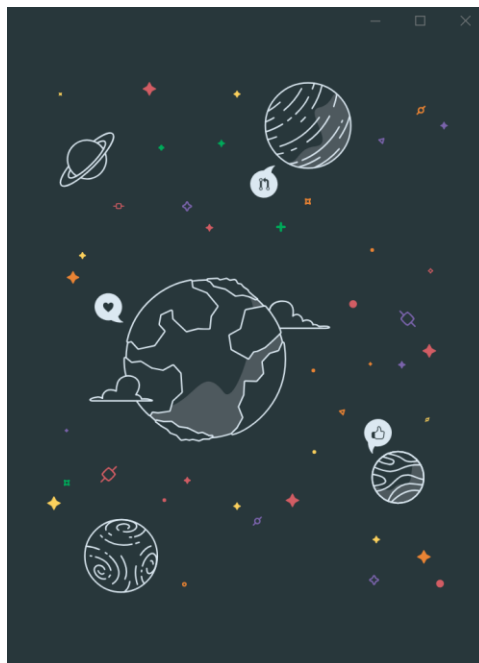


Figure 23: GitHub Desktop Configuration

6.1.2.5 Setting up GitHub Desktop

After the installation, you will be taken to the GitHub Desktop start page. There you select "Clone a repository from the Internet...".



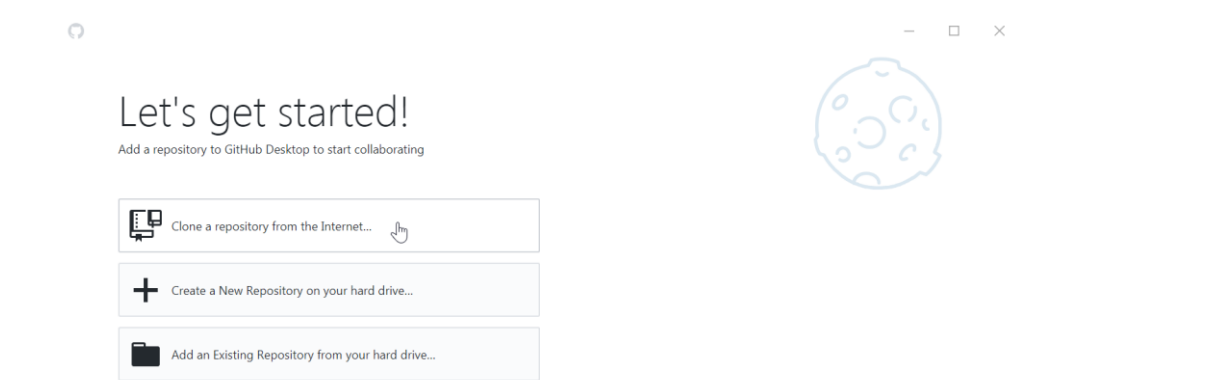


Figure 24: Clone repo in GitHub Desktop

In the window that appears, select the "URL" tab and enter the link you copied from your GitLab project. You can also specify where the repository should be stored.

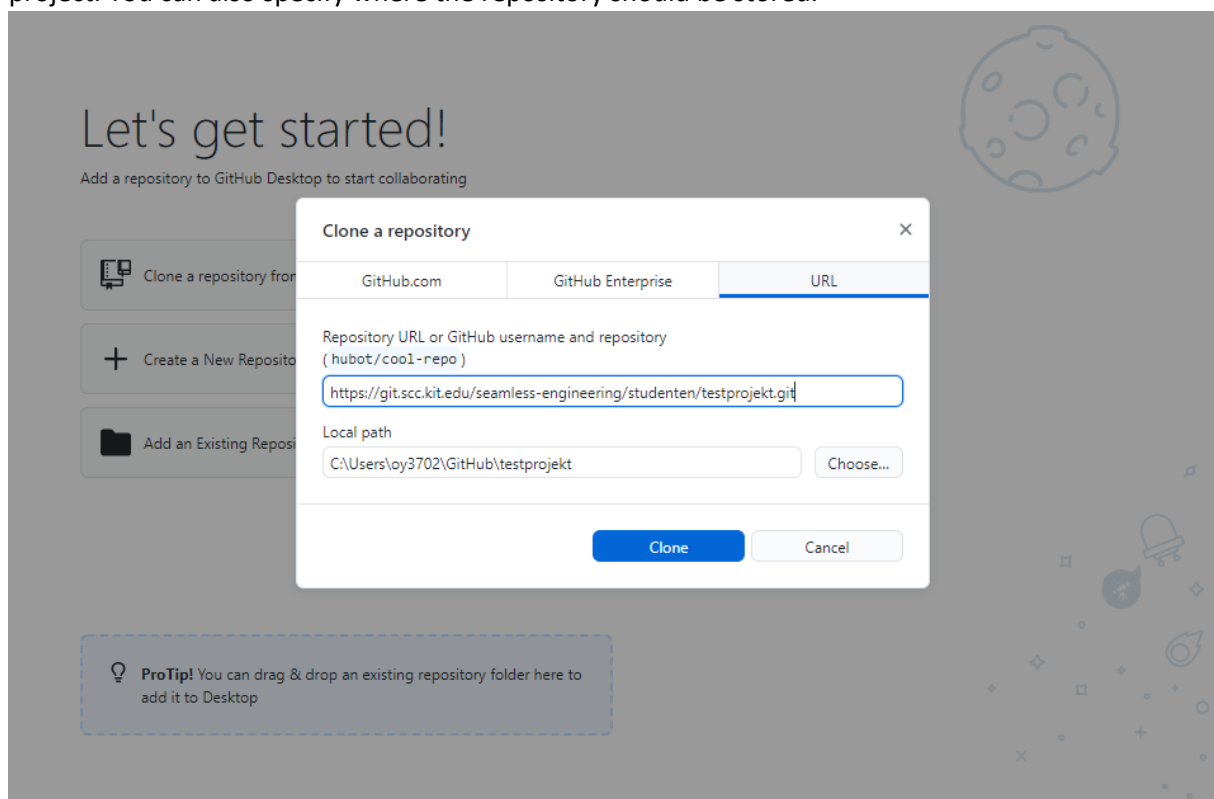


Figure 25: URL for clone process in GitHub Desktop

Shortly after, GitHub Desktop will ask for login details. Enter your GitLab login details, i.e. your SCC account.

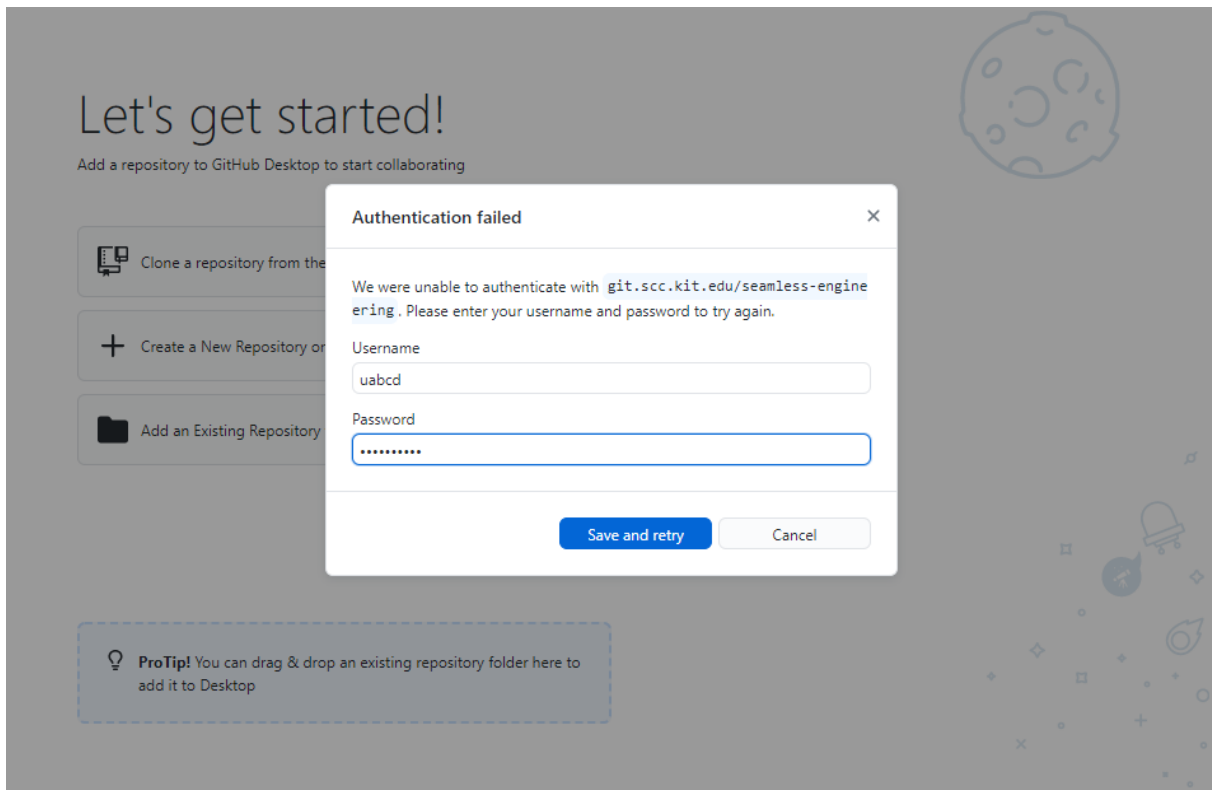


Figure 26: Entering GitLab login data in GitHub Desktop

If you prefer not to enter your password, you can also create a so-called token in the GitLab settings and enter it instead of the password. You can create the token under "Settings → Access Token". Give it a name, an expiry date if necessary and tick "api". You can then use it to log in and revoke it at any time.

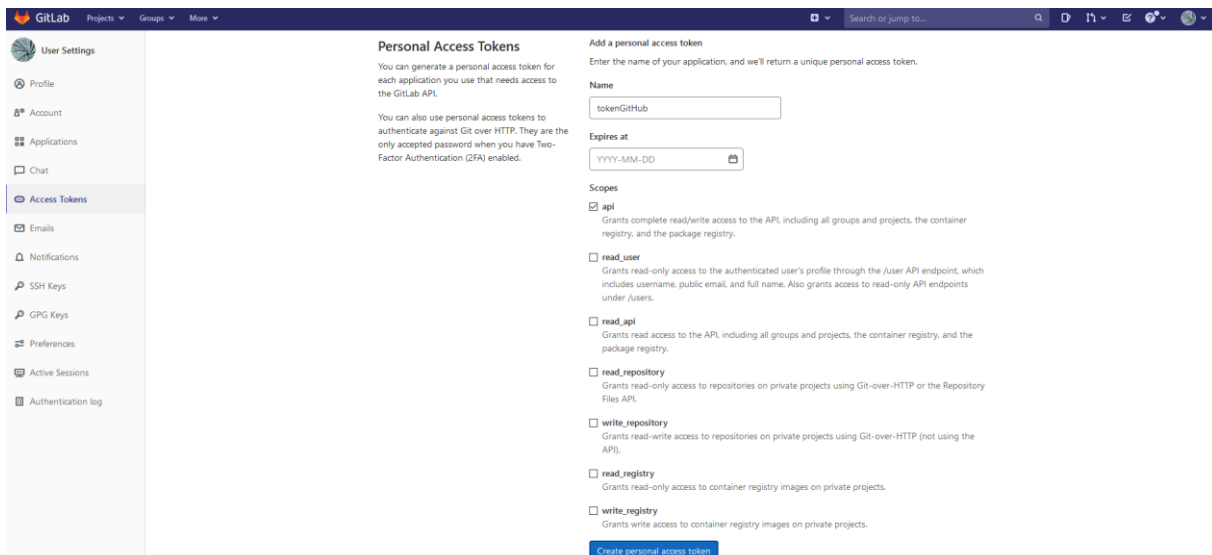


Figure 27: (Optional) Creating a access token in GitLab

After confirming your login details, GitHub Desktop will synchronize the repository and take you to the overview page.

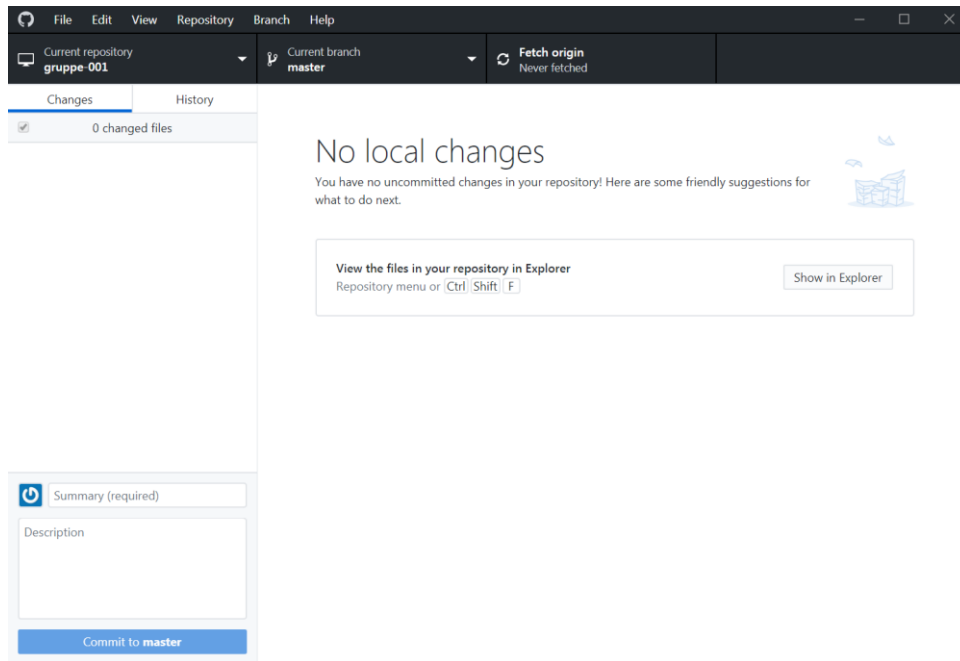


Figure 28: GitHub Desktop overview

6.1.2.6 Working with GitHub Desktop

The basic procedure for working with GitHub Desktop is as follows:

- Make sure the correct branch is selected ("Current branch" in the top center). "master" is the main branch of your project.
- Make sure the local version is up to date (top right "Fetch origin").
- If necessary, download changes ("Pull origin").
- Work on the code
- Create a commit with the changes
- Upload the changes ("Push origin")

To avoid conflicts caused by different versions of the files, make sure you always follow this procedure. In addition, make sure that several group members never work on the same files in the same branch at the same time. If necessary, create several branches and merge them afterwards.

Always use GitHub Desktop to download the source code and the documents and info repository from GitLab to your computer. Do not use the download function of your browser for this. This ensures that all files are always stored in the correct directory and are up-to-date.

When you have worked on the code, the changes are displayed in GitHub Desktop. If you now want to upload them, enter a short description of what you have changed in the bottom left-hand corner, as well as an optional more detailed description. Finally, you can create the commit ("Commit to ...").

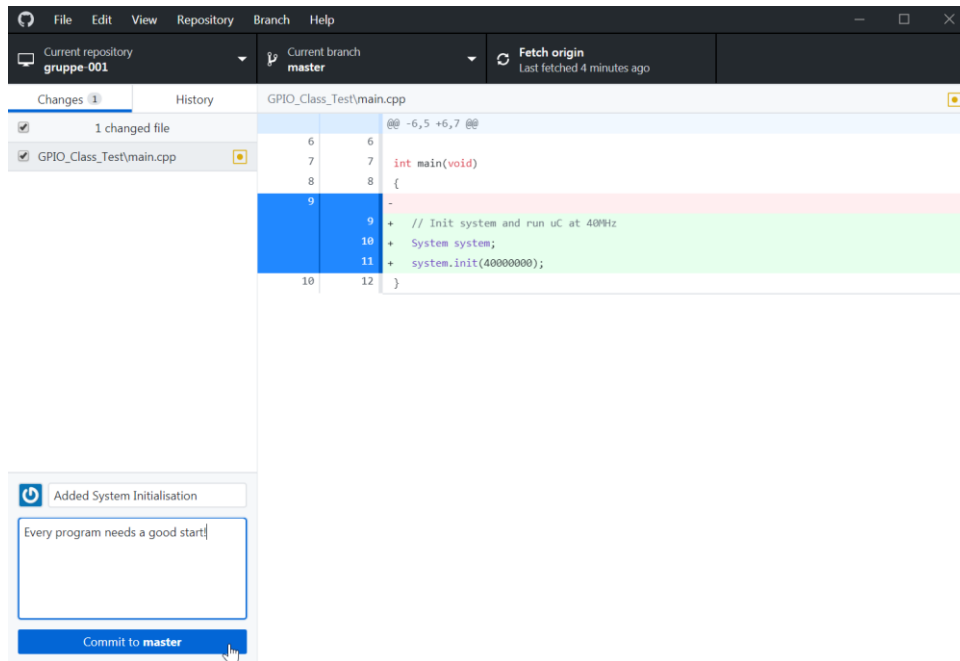


Figure 29: Creating a commit in GitHub Desktop

As long as the commit has not been uploaded, you can still undo it ("Undo" at the bottom left), but not afterwards. Upload it with "Push origin".

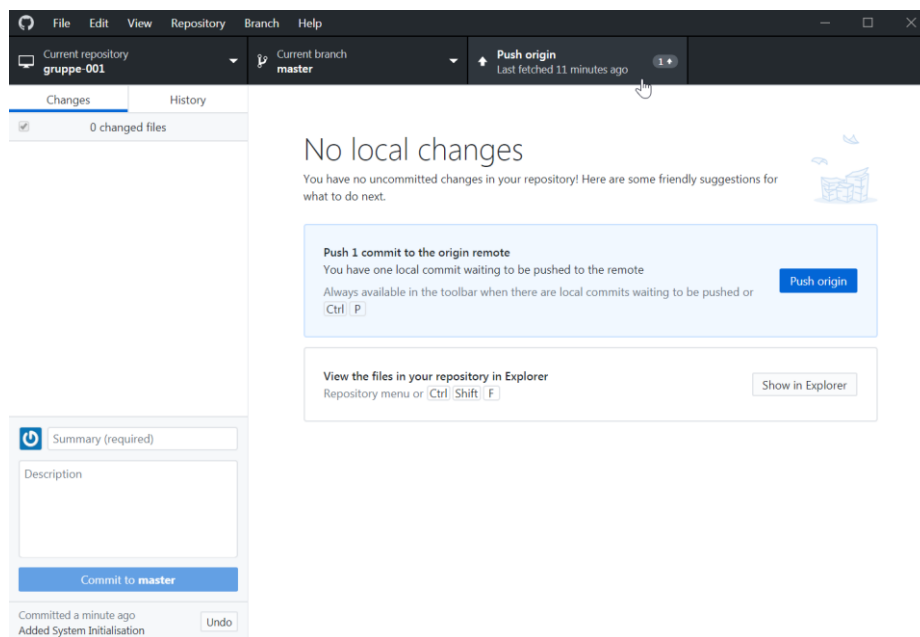


Figure 30: Pushing commit in GitHub Desktop

6.2 Git Bash

Alternatively, to the graphical variant there is also the possibility to use git directly from the terminal. This is especially necessary for using the GIT on the VM. Here you can do the same things as already presented.

You just must type "git" followed by the command in the command line.

To clone a repository from the git local type:

```
git clone <repo> <directory>.
```

This will clone the repository at <repo> into the ~<directory>! folder on the local machine. For example:

```
git clone ssh://john@example.com/path/to/my-project.git cd my-project
```

To commit changes simply type:

```
git commit -m "commit message"
```

To push changes just type in

```
git pull or git push
```

6.3 Why does the workshop use git?

There are several situations where a well-maintained Git repository comes in handy.

- Nothing gets lost

If you hit a dead end, or your code stops working and you can't find the bug, you can just go back to a previous commit where everything was fine. Better than having to go back, however, is to have a branch with only finished code (usually the "master" branch) and create a separate development branch (e.g. "feature" branch) for each development step. In this branch, new functions and ideas can be developed and tested without affecting the stable code in the "master" branch.

- Clear and conflict-free collaboration possible

Each group member creates their own branch for the section they want to develop. If he has successfully completed his part, this branch is merged with the "master" branch ("merge"). If another group member has edited another section in the same file (in his branch), this is usually not a problem when merging, because Git automatically recognizes what goes where and so there is no need to "patch together" lines of code. However, the more often one has tinkered with existing code, the higher the risk that another group member has edited the same place and Git can no longer automatically decide which version should now be used. For this reason, the features or code sections edited in a branch should be as precisely delimited as possible and should be merged in time (only when it works, of course).

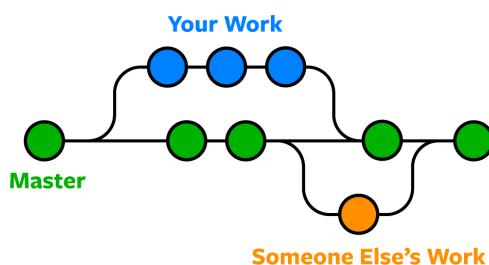


Figure 31: Branches in Git

6.4 Robot Operating System (ROS)

ROS is a Meta-Operating System. Although Meta-Operating System is not a defined term in the dictionary, it describes a system that performs processes such as scheduling, loading, monitoring and error handling by using the virtualization layer between applications and distributed computing resources.

As a meta-operating system, ROS develops, manages and deploys application packages for various purposes, and it has formed an ecosystem that distributes user-developed packages. As described in Figure 17, ROS is a supporting system for controlling a robot and sensor with a hardware abstraction and developing robot applications based on existing conventional operating systems.



Figure 32: ROS as a Meta OS

6.4.1 ROS components

As shown in Figure 18, ROS consists of a client library to support various programming languages, a hardware interface for hardware control, communication for data transmission and reception, the Robotics Application Framework to create various robotics applications, the robotics application, which is a service application based on the Robotics Application Framework, simulation tools that can control the robot in virtual space, and software development tools.



Figure 33: ROS components

6.4.2 Important ROS concepts

To develop a robot related to ROS, it is necessary to understand the essential components and concepts of ROS. This chapter introduces the terminology used in ROS and the important concepts of ROS.

6.4.3 ROS Terminology

Master

The master acts as a name server for node-to-node connections and message communication.

The `roscore` command is used to run the master, and when you run the master, you can register the name of each node and get information if needed. The connection between nodes and Message communication such as Topics and Services are not possible without the master.

The master communicates with the slaves via XMLRPC (XML Remote Procedure Call), an HTTP-based protocol that does not maintain connectivity. In other words, the slave nodes can only access when they need to register their own information or request information from other nodes. The connection status of each other's nodes is not checked regularly.

When you run ROS, the master is configured with the URI address and port configured in the `ROS_MASTER_URI`. By default, the URI address uses the IP address of the local PC and the port number 11311, unless otherwise changed.

Node

A node refers to the smallest processing unit that runs in the ROS. Think of it as a single executable program. ROS recommends creating a single node for each purpose, and it is recommended to design for easy reusability. For example, in mobile robots, the program to operate the robot is divided into specialized functions. A specialized node is used for each function such as sensor data conversion, obstacle detection, motor drive, encoder input and navigation.

At start-up, a node registers information such as the name, message type, URI address and port number of the node. The registered node can act as a publisher, subscriber, service server or service client based on the registered information, and nodes can exchange messages via topics and services.

At startup, a node registers information such as the name, message type, URI address and port number of the node. The registered node can act as a publisher, subscriber, service server or service client based on the information.

or service client based on the registered information, and Node can exchange messages via Topics and Services.

Package

A package is the basic unit of ROS. The ROS application is developed on a package basis, and the package contains either a configuration file for starting other packages or nodes. The package also contains all the files needed to run the package, including the ROS dependency libraries for running various processes, data sets and the configuration file.

Message

A Node sends or receives data between nodes via a message. Messages are variables such as integer, floating point and Boolean. A nested message structure can be used in the message, which contains another message or an array of messages. The TCPROS and UDPROS communication protocol is used for message delivery. Topic is used in unidirectional message delivery, while bidirectional message delivery uses a service involving request and response.

sensor_msgs/Image Message

File: `sensor_msgs/Image.msg`

Raw Message Definition

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#
Header 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 camera
                        # *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

uint32 height          # image height, that is, number of rows
uint32 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.sourceforge.net and send an email proposing a new encoding.

string encoding         # Encoding of pixels -- channel meaning, ordering, size
                        # taken from the list of strings in include/sensor_msgs
                        # /image_encodings.h

uint8 is_bigendian      # is this data bigendian?
uint32 step            # Full row length in bytes
uint8[] data            # actual matrix data, size is (step * rows)
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

autogenerated on Fri, 15 Jan 2021 03:18:41

Figure 34: ROS Message

Topic

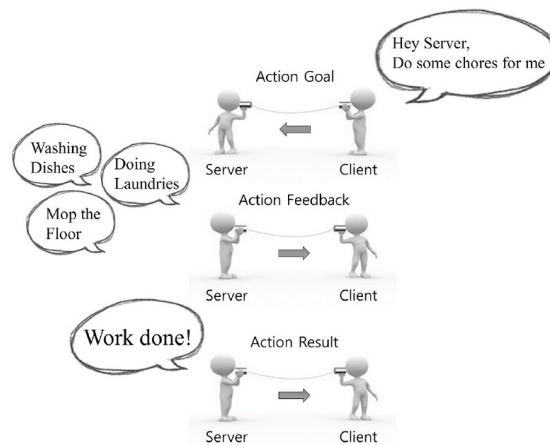
The Topic is literally like a Topic in a conversation. The publisher node first registers its topic with the master and then starts publishing messages about a topic. Subscriber nodes receive topic request information from the publisher node that corresponds to the name of the topic registered in the master. Based on this information, the subscriber node connects directly to the publisher node to exchange messages as a topic.

Service

The service is a synchronous bidirectional communication between the service client, which requests a service for a specific task, and the service server, which is responsible for responding to requests.

Action

Action is another message communication method for asynchronous bidirectional communication. Action is used when it takes longer to respond after receiving a request and intermediate responses are required until the result is returned. The structure of the action file is also like that of the service. However, the feedback data section for intermediate responses is added along with target and result data section which are represented as request and response in and service respectively. There is an action client that sets the target of the action and an action server that executes the action specified by the target and returns feedback and result to the action client.



6.4.4 Message communication

As described earlier, ROS is developed in the unit of Nodes, which is the minimum unit of executable program collapsed for maximum reusability. The node exchanges data with other nodes via messages that form a large program as a whole. The key concept here is the methods of message communication between nodes. There are three different methods of message exchange: the Topic (unidirectional), the Service (bidirectional) and the Action (bidirectional). Figure xxx shows these different methods.

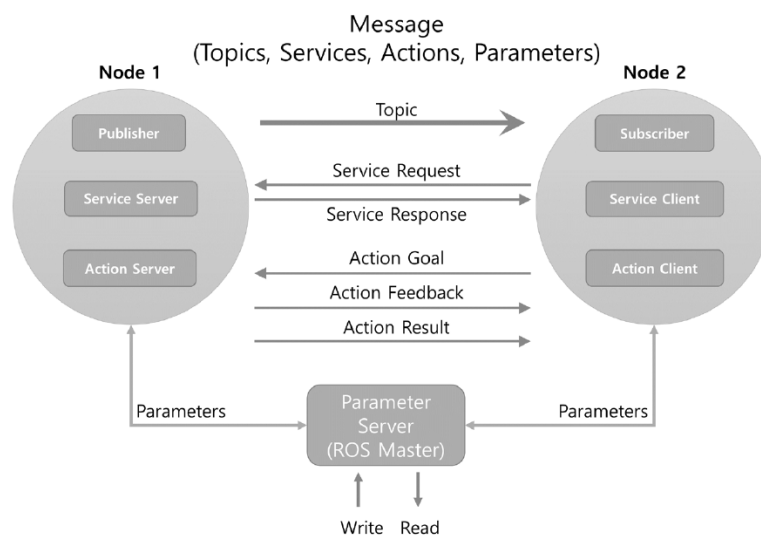


Figure 35: Message communication between nodes

Type	Feature	Description
Topic	Asynchronous unidirectional	Used when exchanging data continuously
Service	Synchronous bi-directional	Used when request processing requests and responds current states
Action	Asynchronous bidirectional	Used when it is difficult to use the service due to

		long response times after the request or when an intermediate feedback value is needed
--	--	--

6.4.5 ROS Tutorials

To get started with ROS, the ROS organization offers tutorials that introduce ROS beginners to the topic step-by-step. You can find these tutorials here: <http://wiki.ros.org/ROS/Tutorials>

Below you will find all the tutorials for beginners listed on the website.

1.1 Beginner Level

1. [Installing and Configuring Your ROS Environment](#)
This tutorial walks you through installing ROS and setting up the ROS environment on your computer.
2. [Navigating the ROS Filesystem](#)
This tutorial introduces ROS filesystem concepts, and covers using the `roscd`, `rosls`, and `rospack` commandline tools.
3. [Creating a ROS Package](#)
This tutorial covers using `roscat` or `catkin` to create a new package, and `rospack` to list package dependencies.
4. [Building a ROS Package](#)
This tutorial covers the toolchain to build a package.
5. [Understanding ROS Nodes](#)
This tutorial introduces ROS graph concepts and discusses the use of `roscat`, `rosls`, and `roslaunch` commandline tools.
6. [Understanding ROS Topics](#)
This tutorial introduces ROS topics as well as using the `rostopic` and `rqt_plot` commandline tools.
7. [Understanding ROS Services and Parameters](#)
This tutorial introduces ROS services, and parameters as well as using the `rosservice` and `rosparam` commandline tools.
8. [Using rqt_console and roslaunch](#)
This tutorial introduces ROS using `rqt_console` and `rqt_logger_level` for debugging and `roslaunch` for starting many nodes at once. If you use ROS `fuerte` or earlier distros where `rqt` isn't fully available, please see this page with [this page](#) that uses old `rx` based tools.
9. [Using roset to edit files in ROS](#)
This tutorial shows how to use `roset` to make editing easier.
10. [Creating a ROS msg and srv](#)
This tutorial covers how to create and build msg and srv files as well as the `rosmmsg`, `rossrv` and `roscp` commandline tools.
11. [Writing a Simple Publisher and Subscriber \(C++\)](#)
This tutorial covers how to write a publisher and subscriber node in C++.
12. [Writing a Simple Publisher and Subscriber \(Python\)](#)
This tutorial covers how to write a publisher and subscriber node in python.
13. [Examining the Simple Publisher and Subscriber](#)
This tutorial examines running the simple publisher and subscriber.
14. [Writing a Simple Service and Client \(C++\)](#)
This tutorial covers how to write a service and client node in C++.
15. [Writing a Simple Service and Client \(Python\)](#)
This tutorial covers how to write a service and client node in python.
16. [Examining the Simple Service and Client](#)
This tutorial examines running the simple service and client.
17. [Recording and playing back data](#)
This tutorial will teach you how to record data from a running ROS system into a .bag file, and then to play back the data to produce similar behavior in a running system
18. [Reading messages from a bag file](#)
Learn two ways to read messages from desired topics in a bag file, including using the `ros_readbagfile` script.
19. [Getting started with roswtf](#)
Basic introduction to the `roswtf` tool.
20. [Navigating the ROS wiki](#)
This tutorial discusses the layout of the ROS wiki (wiki.ros.org) and talks about how to find what you want to know.
21. [Where Next?](#)
This tutorial discusses options for getting to know more about using ROS on real or simulated robots.

Figure 36: ROS Tutorials for beginners

6.5 Gazebo

Gazebo is a 3D simulator that provides robots, sensors and environment models, 3D simulation required for robot development and offers realistic simulation with its physics engine.

Gazebo is characterized by the following features:

- Dynamics simulation: various physics engines such as Bullet, Simbody and DART.
- Sensor simulation: Laser range finder (LRF), 2D/3D camera, depth camera, contact sensor, force-torque sensor and more are supported, and noise can be added to the sensor data like the real environment.
- Plug-ins: APIs are provided to allow users to create robots, sensors and environment control
- Benefit: Integration with ROS, digital twin, simulation is provided.

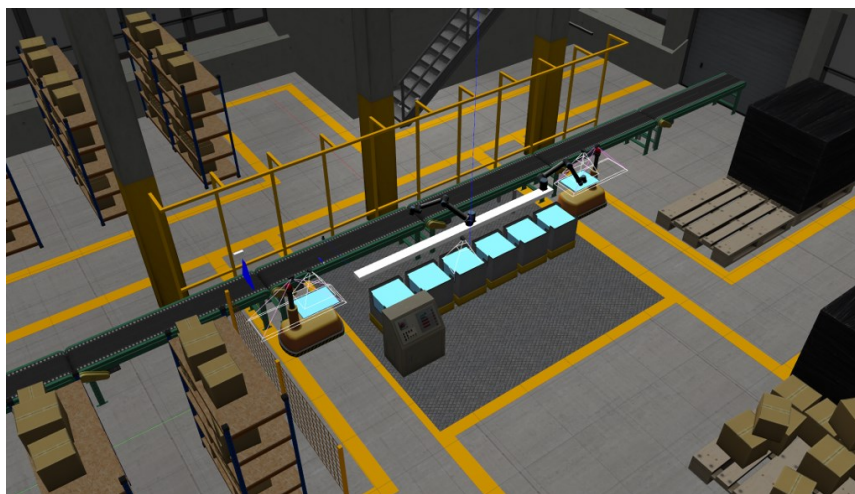


Figure 37: Simulated warehouse in gazebo

6.6 RVIZ

rviz is a 3D visualization tool for ROS applications. It provides a view of your system model, captures sensor information and renders the captured data. It can display data from cameras, lasers, 3D and 2D devices including images and point clouds.

rviz needs a running roscore, when one is running you can simply start rviz with

```
roslaunch rviz rviz
```

After you will see an empty window. To visualize your simulation, you need to add the sensors and robots in the list on the left side.

6.7 Python

The programming language used in this workshop is Python 3. It is an easy-to-use language that makes it easy to get a program running.

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional

programming. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants including Linux and macOS, and on Windows. (<https://docs.python.org/3/faq/general.html#what-is-python>)

In Python no brackets or semicolons are needed to use functions. The structuring is only done by indenting with 4 spaces. A good summary about the most important Python commands can be found here: <https://www.pythoncheatsheet.org/>

6.8 OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. It is one of the most widespread libraries for image processing and provides more than 2500 algorithms. One of the strengths of this is the high speed of these. It is also available for Python.

6.9 Nano

Nano is a free text editor for Unix and similar systems, such as Linux. It is easy to use, unlike other command line applications.

```

student@ubuntu2004: ~/2021_pnp_ws/src/pnp_ws_environm...
GNU nano 4.8      tf_example.py
#!/usr/bin/env python3

import rospy
import tf
from geometry_msgs.msg import PoseStamped

if __name__ == '__main__':
    rospy.init_node('tf_echo')

    listener = tf.TransformListener()

    rate = rospy.Rate(0.5)

    while not rospy.is_shutdown():
        try:
            (translation, rotation) = listener.lookupTransform('/charuco', '/it',
            print("Translation: " + str(translation))
            print("Rotation: " + str(rotation) + "\n")

        pose = PoseStamped()
    [ 32 Zeilen gelesen ]
^G Hilfe    ^O Speichern ^W Wo ist    ^K Ausschneid ^J Ausrichten ^C Textmarke
^X Beenden  ^R Datei öffn ^\ Ersetzen ^U Text einfü ^T Rechtschr. ^_ Zu Zeile geh

```

Nano can be used to quickly edit files directly in the command window. In the following the most important commands for nano are presented. To open a file with nano, simply navigate to the folder where the file is located (cf. ...). After that you type:

```
nano file_to_open.py
```

Ctrl+S , Save current file	Alt+F, Run a formatter/fixer/arranger
----------------------------	---------------------------------------

Ctrl+O, Offer to write file ("Save as")	Ctrl+A, To start of line
Ctrl+K , Cut current line into cutbuffer	Ctrl+E, To end of line
Ctrl+U, Paste contents of cutbuffer	Ctrl+P, One line up
Alt+3, Comment/uncomment line/region	Ctrl+N, One line down
Alt+U, Undo last action	Ctrl+Y, One page up
Alt+E, Redo last undone action	Ctrl+V, One page down
Ctrl+W, Start forward search	Ctrl+G, Display help text
Ctrl+H, Delete character before cursor	Alt+N, Turn line numbers on/off
Ctrl+T , Execute some command	Ctrl+L, Refresh the screen
Alt+B, Run a syntax check	Ctrl+Z, Suspend nano
Ctrl+X Exit nano	

<https://nano-editor.org/dist/latest/cheatsheet.html>

6.10 Visual Studio Code

For coding with an GUI we recommend: <https://code.visualstudio.com/>.

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

A good overview of the first steps can be found here:

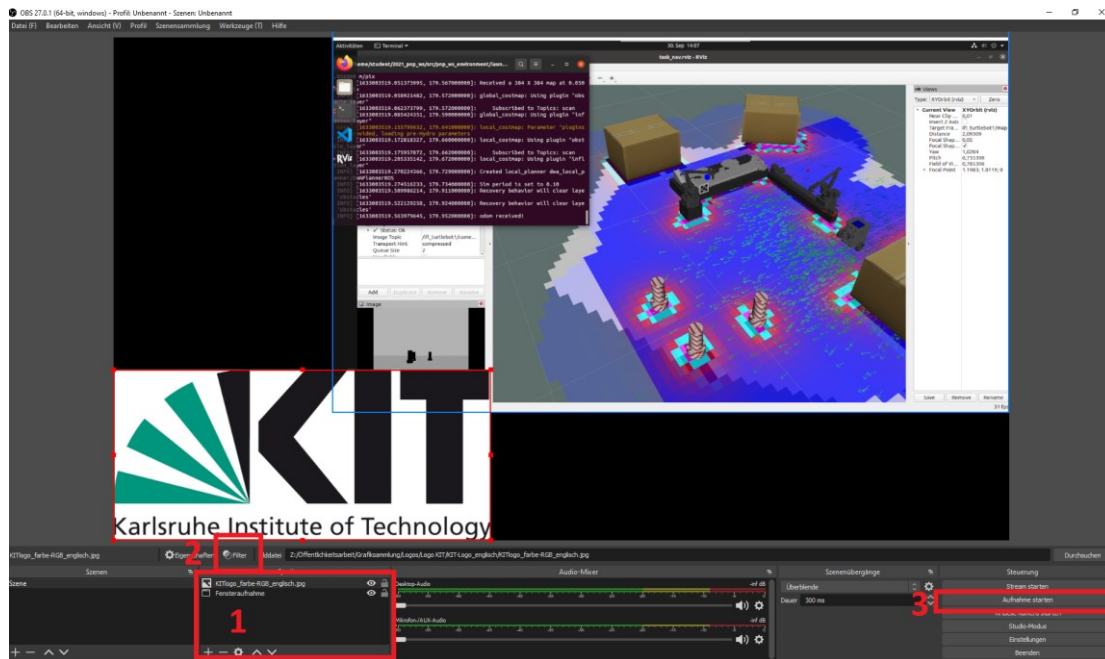
<https://code.visualstudio.com/docs/getstarted/introvideos>

6.11 OBS Studio

OBS Studio is a free and open-source software suite for recording and live streaming. Here you can record multiple windows at once. You can download it under: <https://obsproject.com/de/download>

To get your first recording started please have a look at: <https://obsproject.com/wiki/OBS-Studio-Quickstart>

To capture your screen, you must first select the source (1), then you can arrange the different sources on the screen. After that you can apply different filters to the source, for example to crop it (2). To start recording, simply press Start recording (3).



6.12 Simple Screen Recorder

The program can be started via the entry "Multimedia → SimpleScreenRecorder". For the user guidance, the program author has opted for a five-step wizard. If there are no special requests, the defaults can simply be adopted:

1. Welcome	2. Recording area/sound	3. Codecs	4. Start recording	5. Goodbye

The following is a brief description of what is relevant in steps 2, 3 and 4.

Recording area and audio input

- Capture area: whole screen (full screen), window or user-defined section.
- Record mouse pointer (yes/no)?

Audio and video codec

- File name for saving. Large files can be split automatically if desired.

Recording

- Recording control: Start, Pause, Save (=Stop).
- Shortcut keys for recording. Default: Ctrl + R
- Preview function

7 Programming guidelines

7.1.1 Motivation and application

„The purpose of a defined programming style is to facilitate the work of all team members involved in a programming project. This refers in particular to the readability, comprehensibility and maintainability of program source code and the elimination of avoidable sources of error in programs.

In terms of comprehensibility and maintainability, a guideline can restrict or completely prohibit the use of program constructs that are permitted in programming language (but are "unclean"). Adherence to predefined nomenclatures for variables, procedures and class names can significantly improve the readability and maintainability of a program code.

For the maintenance of program code, adherence to a defined programming style is even more important than during development. As a guideline, 80% of the lifetime of a software system is spent on maintenance. Only rarely is a system maintained by the original developer. It is therefore all the more important that a good programming style is used right from the start.“

[\(Source\)](#)

The following guidelines are divided into two categories: Mandatory Guidelines (M) and Recommendations (R). The former must be followed, the latter are intended primarily as an aid for those with less programming experience and are not mandatory. A helpful tool is the auto-formatting of xxx (highlight the code and then enter the key combination CTRL + I). It complies with the rules, but cannot implement all the points listed here.

7.1.2 Unity within the Group

(M) It is important that you agree on exactly one programming style within your group and stick to it.

7.1.3 Identifier

(M) Choose meaningful names. The only allowed exception are count variables.

Bad:

```
a = 42
for countingVariable in range(3):
```

Good:

```
int answerToEverything = 42
for index in range(3):
```

(M) Use only one language in the source code. All class, method, and variable names must be written in the same language.

Variable and method names always start with a lowercase letter. If a name consists of several words, each subsequent word begins with an uppercase letter.

Example:

```
speed = 98
```

Class names always begin with a capital letter. If a name consists of several words, each subsequent word begins with an uppercase letter.

Example:

```
class Timer
```

7.1.4 Formatting

Ⓜ A space is placed before and after operators, after control statements and after commas. The only exceptions are the ++ and -- operators. There is no space between method name and parentheses, and directly after an open parenthesis (or directly before a closed parenthesis).

Ⓜ Comments are never at the end of the line, but always in the line above. Method comments can be placed before the method as well as at the beginning of the method.

🔗 Comments longer than two lines are written with """.... """ , not with #

🔗 Insert blank lines to divide your code into blocks. You can also insert a blank line before a comment to emphasize the affiliation with the next line.

7.1.5 Commenting

Ⓜ Each method has a comment that explains the function of the method and, if present, the meaning of the passing parameters as well as the return value.

Ⓜ All source code must be commented in such a way that no follow-up questions are necessary to understand the functionality. You can assume that the reader is familiar with the syntax of Python, but not that he knows what exactly you have thought about. Because of this, the comments contain less description of what the line says, but why.

Bad:

```
# Set current and pin type
current = 2MA
pinType = PIN_TYPE_STD
```

Good:

```
#The default current and pin type (also see pin.pdf on page 22)
gpioCurrent = 2MA
pinType = PIN_TYPE_STD
```

Ⓜ Be sure to keep your comments up to date! Especially when debugging, it is common to comment out or change lines. However, debugging also involves cleaning up afterwards.

🔗 If you use information from external sources such as datasheets, then write your source as well as the page number in the comment.

7.2 Common mistakes

7.2.1 Python mistakes

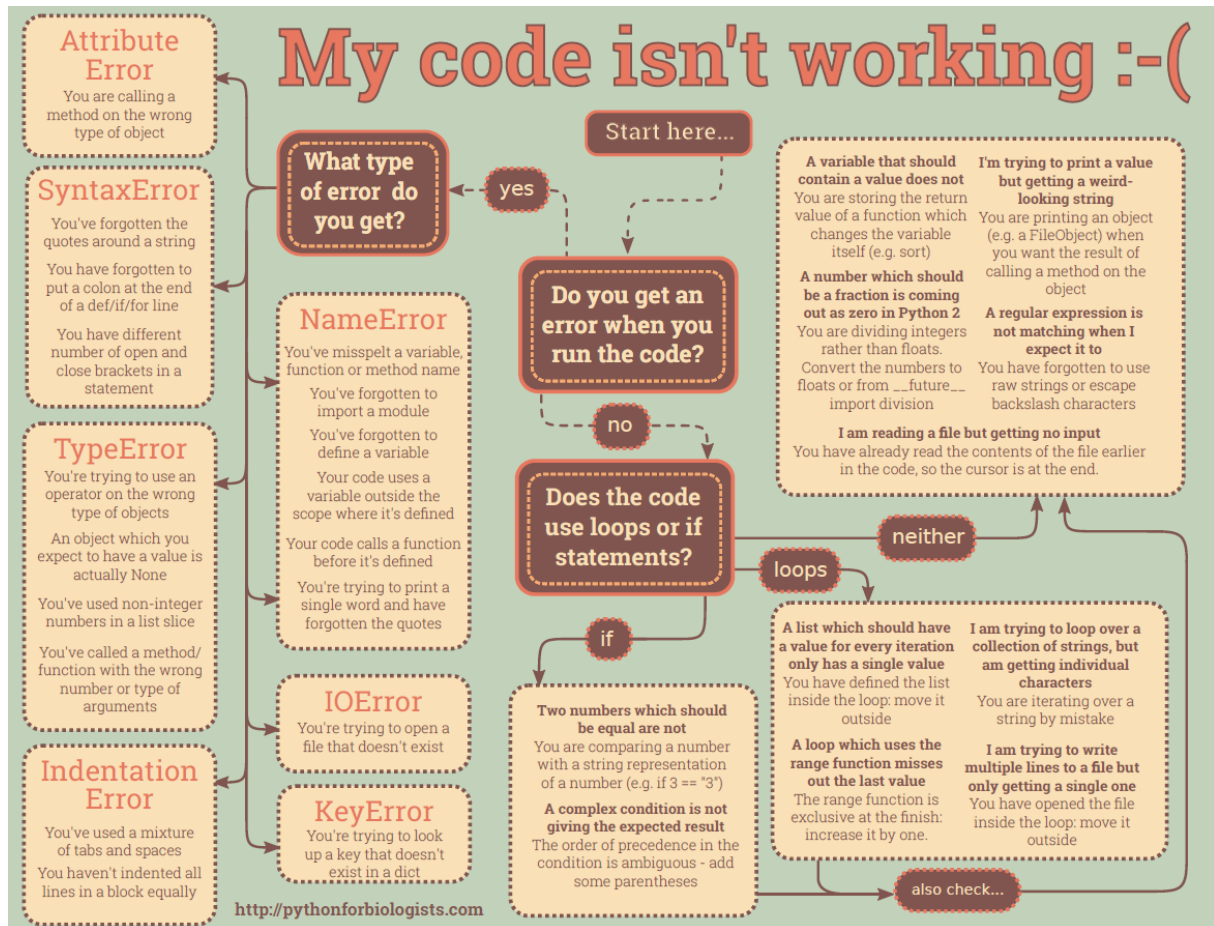


Figure 38: Typical Python mistakes

7.2.2 Python Debugging

The following website offers very helpful debugging tips you may need during the course

<https://medium.com/techtofreedom/six-debugging-techniques-for-python-programmers-cb25a4baaf4b>

7.2.3 ROS Troubleshooting

ROS bietet ein kleines Troubleshooting Tutorial, das Sie hier finden können:

<http://wiki.ros.org/ROS/Troubleshooting>

8 Further documents

If you have little or no programming experience in Python, you will find below the most important concepts you need to know how to deal with in the workshop.

There is a detailed manual for dealing with ROS that you can find here:

http://wiki.ros.org/Books/ROS_Robot_Programming_English

Also you can find very good video tutorials about ROS on YouTube:

<https://www.youtube.com/watch?v=0BxVPCInS3M&t=1106s>

Of course, you are free to use other materials (books, websites) to learn the programming language.

9 Appendix

The uArm dimensions for kinematics can be seen here:W

