Setup of a robotarm under ROS for UPC Barcelona

Yannic Scholtyssek

Abstract—**The goal of this work is it to perform the necessary steps to control a robot arm using ROS and implement motion planning utilising the MoveIt! library. Hereby a four degree of freedom robotarm with servos in the joints has to be controlled. Firstly a simplified simulation of the robot is created using the universal robot description format(urdf). This way we can apply the MoveIt! library for the system and get a simulation that represents the real robot. On this simulation we can apply control strategies and the goal is it to get the joint states from the Encoders and use the current angles with ros_control and the desired position to execute a certain trajectory. Hereby a hardware interface has to be written and several control strategies will be tested. Furthermore the inverse kinematics of the robot has to be applied, to be able to set position goals. For this purpose the library ikfast will be used to obtain the inverse kinematics solution for the robot in the three dimensional space. Later on the end effector of the arm shall be controlled using a collaborative approach. This is achieved by incorporating distance sensors of the type VL53L0X. An array of distance sensors had to be designed and 3d-printed to obtain the best suited structural and functional sensor unit. Through this unit the robot perceives its surrounding and can be controlled using a humans hand and thus can be used to prevent collisions or injuries with humans working around it. The entire work is done using Ubuntu 20.04 with Ros noetic and python 3. All the practical coding work can be found on github under the url:**

`https://github.com/yscholty/yannic_robot/`

In the README on github further instructions on how to run the program and all the dependencies can be found. Furthermore the code is well commented and should be more or less straight forward to understand.

Index Terms—**ROS, MoveIt!, Arduino, Inverse Kinematiks, IK fast, Servos, Python3, Ubuntu 20.04**

I. Introduction

ROBOTS are an essential part of nowadays industry and production chains. The advantage of robotic systems is, that they can work non stop, that they are versatile and that they can work in for human-beings toxic work environments. Hence research in this area is moving fast. New solutions are found and one of the research field is the one looking at collaborative robots. Collaborative Robots means, that the strengths of both worlds are incorporated into one solution. The strengths of the robot regarding the latterly named, mainly precision and strength, and the strengths of the human namely its creativity and capability to have an overview. Hereby it is essential to provide a safe working environment for the human working along with the robot. One possibility to do so will be shown in this work.

Yannic Scholtyssek
July 22, 2021

II. Setup

Firstly we will set up the hardware components. To do so hook up the Arduino MEGA, the computer, the servos and distance sensors as can be seen in Figure 1.

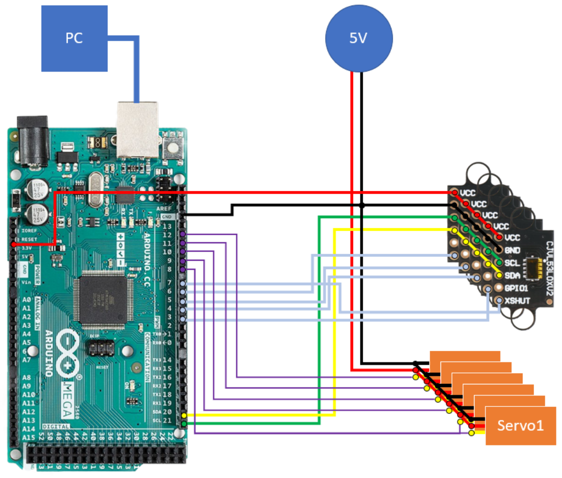


Fig. 1. The hardware setup.

To get started wit ROS noetic under Ubuntu 20.04 for this project, several dependencies have to be installed. Also to get the inverse kinematics solution for the robot, the library ikfast has to be used and set up. Therefore we have to install several programs and modules. First Ubuntu 20.04 has to be installed. To do so follow the steps on the official website, download the Desktop image and boot from a bootable device to install the operating system [1].

ROS noetic has to be installed by following these steps [2]:

- First we need to set up the computer to accept installations from packages.ros.org:
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list´
- setting up the keys:
  curl -s https://raw.githubusercontent.com/ros/rosdistro /master/ros.asc | sudo apt-key add -
- updating the debian package:
  sudo apt update
- and then installing ros-noetic with: *sudo apt install ros-noetic-desktop-full*
- it is then also advised to add the sourcefile to the .bashrc:
  echo "source /opt/ros/noetic/setup.bash" >> /.bashrc
  source /.bashrc

Now we have to install the MoveIt! library. To do so, type: *sudo apt install ros-noetic-moveit*.

sudo apt-get install python3-catkin-tools . A known issue is, that catkin tools is broken. To install it either way, firstly pip3 has to be installed:

sudo apt-get -y install python3-pip and then catkin tools has to be installed with pip[3]:

pip3 install –user "git+https://github.com/catkin /catkin_tools.git#egg=catkin_tools" If at any point an errors comes up stating that dependencies are not met, we have to overwrite the corresponding files by typing: *sudo dpkg -i –force-overwrite <thefile>.deb*

ROSSERIAL: To enable the serial communication with the Arduino, we have to install rosserial in our src folder of the catkin workspace with [4]:

git clone https://github.com/ros-drivers/rosserial.git -b noetic-devel and also we have to install the libraries inside Arduino. To do so, we go up to Tools − > Manage libraries and find the library: ROSSERIAL ARDUINO LIBRARY and install it. Also we have to run:

rosrun rosserial_arduino make_libraries.py . and *pip3 install pyserial*. To install the sensor libraries again in the Manage libraries dialog find the library ADAFRUIT_VL53L0X and install it as well. You are now ready to upload the code to the Arduino board.

Now we can download the source files from the git and build the project.

- clone all the files of the github provided to your local machine by running *git clone https://github.com/yscholty/yannic_robot.git*
- since in the git only the src files are given, you will still have to build them. To do so go to the root folder of the git workspace and run *catkin build*. The building process should start and you should now have all the necessary libraries installed to run the setup for the robot. Now launch files can be started with: *roslaunch <rospackage>*
- the Arduino code has to be uploaded to the Arduino Mega. The Arduino source code can be found under */arduino_code/yannic_robot/yannic_robot.ino*. Note the serial port and change it in the demo.launch files or when you run the serialnode.

At UPC Barcelona I was provided with a robot arm with four degrees of freedom (dof). The end effector (endeff) can be used in three dimensional translational space and the orientation around one axis can be changed independently. The execution of the movement happens through servos of the type: Hitec HS-645MG[5] or similar. The robot and initial setup as given can be seen in Figure 2. To get the system to work with MoveIt!, a URDF model has to be created. In this case it can be fairly simple and just has to represent the exact lengths of the links and to avoid collision of the links it should also incorporate the correct thicknesses. Furthermore the transmission need to be added, to be able to control the robot through the Arduino. The URDF files are created in xml format. In our robot we have four joints plus the gripper. Therefore we create an URDF model with the corresponding amount of links connected by joints. The model for this case can be found under the path yannic_robot/urdf/yannic_robot.urdf.xacro and looks as can be seen in Figure 3.

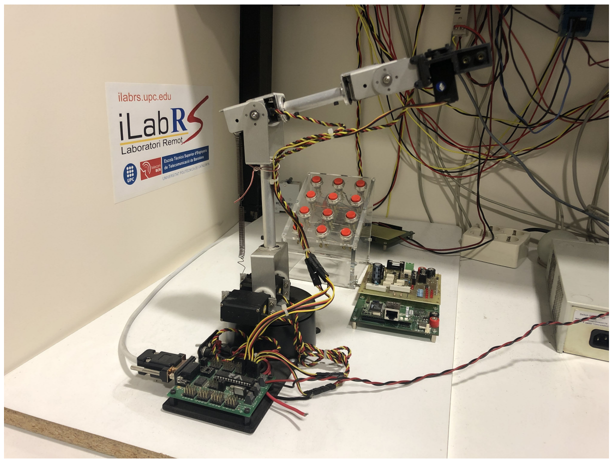


Fig. 2. The initial robot setup without controls.

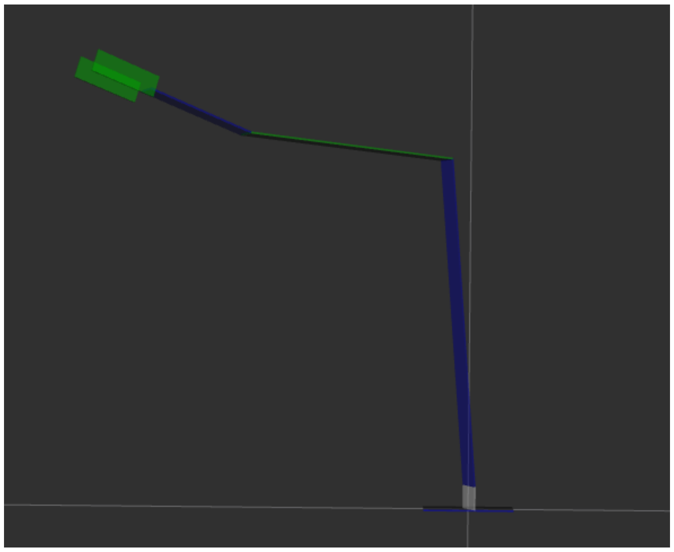


Fig. 3. URDF model of the robot arm.

To now run the demo files, navigate to the launch folder which is located under yannic_robot/launch. Here three demo files can be found.

roslaunch demo.launch will just start the moveit setup. to start the serial communication with the Arduino, rosserial has to be launched as well. To do so type in a new terminal:

rosrun rosserial_python serialnode.py /dev/<port> _baud:=250000

Hereby the port can be found in the Arduino IDE in the bottom right corner and is equal to the serial port the Arduino is connected to. in my case the Port was *ACM0*. To run the stop motion script with six sensors, you would have to run:

roslaunch yannic_robot_moveit_config demo_stop_motion_array.launch . In this launch file change the port number in the rosserial node to the one we have the Arduino connected to.

For the given robot all the inverse kinematics are already solved and saved as a plugin that is used by MoveIt!. Whenever the geometry or kinematic is changed of the robot, a new calculation has to be performed. To do so the library Openrave [6] was used. Since Openrave is not fully supported under Ubuntu 20.04 a Docker image was used. A Docker can be seen as a virtual environment where all the dependencies already come with this docker environment and compatibility is ensured. For this image as a base Ubuntu Indigo was used. Explicitly this means, that to install the Docker image we have to follow the following steps[7]. Firstly the docker environment has to be installed:

sudo apt-get install docker.io
sudo service docker start

Then the user has to be added to the group. If not, the docker image can not be seen:

sudo usermod -a -G docker $USER

Now we can start creating the inverse kinematic plugin. The Inverse Kinematics Type has to be set [8]. To initiate the IKfast plugin we first have to run:

roslaunch moveit_setup_assistant setup_assistant.launch

which starts a graphical interface that we have to go through and therefore set up the robot system for the inverse kinematics. In the setup we have to specify the location of the urdf file [9], [6]. The Setup automatically creates a configuration that can now be used for the IKfast plugin. We then run the IKfast algorithm by using:

rosrun MoveIt!_kinematics auto_create_ikfast_moveit _plugin.sh –iktype Translation3D $MYROBOT_NAME.urdf <planning_group> <base_link> <eef_link>

where we have to specify the *<planning_group>* ,*<base_link>* and *<eef_link>*. On Ubuntu 20.04 with ROS noetic the path to the plugin has to be specified[10]. For this robot the *<base_link>* is defined as *base_link* in the URDF file and the *<eef_link>* as *endeff*.

Another alternative is to write the Inverse Kinematics solver for this rather simple problem yourself. Hereby ROS allows to develop the IK around the $moveit\_core\ KinematicsBase$ class [11]. Though finding the analytical solution for the inverse kinematics of this four dof robot is not trivial at all, so I sticked to the IKfast solution.

The solving factor was it to copy the generated plugin: $<\ robot\_description\ >\ \_ikfast\_plugin$ to the plugin location of the move_it-workspace under */catkin_ws/src/moveit_resources*

It actually just has to be outside another catkin package. So just move the plugin to the workspace that holds the other catkin packages.

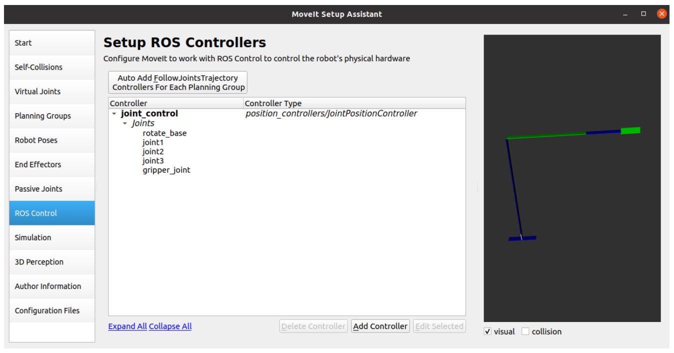
As long as the links are not changed, but only its length is adjusted, running the moveit_setup is obsolete. Though when adding a new link or joint, or changing the constraint of a joint, the moveit_setup has to be run again. This will reload the .yaml files. To do so, the URDF file will be imported by the moveit_setup_assistant. Firstly we need to convert the .urdf.xacro file to a normal .urdf file by running:

rosrun xacro xacro -o yannic_robot.urdf yannic_robot.urdf.xacro

Now the moveit_setup_assistant can be launched with [9]:

roslaunch moveit_setup_assistant setup_assistant.launch

A user interface 4 will come up where we can specify the simulation parameters. The collision matrix can be set. Planning groups will be created so that in the final simulation we can adjust parameters only for one planning group at a time which makes it more user friendly. Furthermore robot poses can be specified. In this example three poses are specified. One for the open and closed gripper in the planning group for the hand and one for the default home position of the robot arm in the planning group arm. With virtual joints we ”mount” the base_link to the reference frame so that the robot_arm does not move freely in space. Furthermore the endeffector is specified and we can inlude ros_controls. Hereby we have several options and in our case chose *position_controllers/JointPositionController*. Once the


MoveIt Setup Assistant

Setup ROS Controllers

Configure MoveIt to work with ROS Control to control the robot's physical hardware

Controller	Controller Type
joint_control	position_controllers/JointPositionController
Joints	
rotate_base	
joint1	
joint2	
joint3	
gripper_joint	

Fig. 4. Moveit setup assistant

setup is done, everything can be exported. This will create the corresponding .yaml files in *yannic_robot_moveit_config*. It has to be checked now that in *kinematics.yaml* the solver is set to the earlier created ikfast plugin.

III. Implementing Controls

To control the robotic arm there are countless possibilities. In this project we will look at one approach, that incorporates a distance sensor. We want to take a look at how we can create a setup, where the robot only moves as long as there is no object in close proximity. Furthermore a custom made sensor mount was printed, which offers space for up to six sensors. The array of sensors makes sure that we can look in all directions. In this study we make use of the distance sensor VL53LOX[12]. To include the VL53LOX, a ROS node has to be written that publishes to a distance topic. This will be done using Arduino again. In case of multiple distance sensors, we can then publish the distances to an array and use the values later on. To initialise multiple distance sensors, a start sequence has to be performed. Hereby I oriented myself along the example given in the Arduino for the VL53L0x. For an Arduino Mega we can publish the distances using a Range sensor message [13] which automatically has the right format. For the sensor array a different message type had to be chosen, because the Range message only offers room for one sensor. The chosen message type was an Int16MultiArray.

A. *Stop motion*

To implement this stop motion capability of the robot, the movement of the robot has to be inhibited in certain situations To start the process, the corresponding launch file has to be

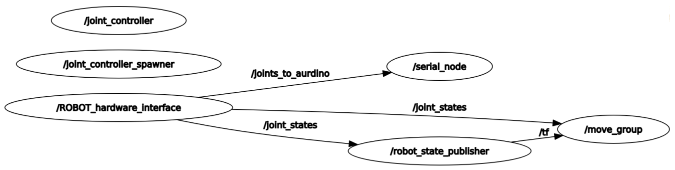


Fig. 5. rqt_graph for the system.

called with:

```
roslaunch yannic_robot_moveit_config \
demo_stop_motion_array.launch
```

The stop motion approach works as follows. Firstly we have the general robot model and robot loaded. Now an additional script has to be created, to incorporate a situation specific action. Firstly a dialog is given where different actions can be selected. A marker can be moved with the mouse and then like it can be seen in 6 via right click a context menu can be found. Here three options are given.

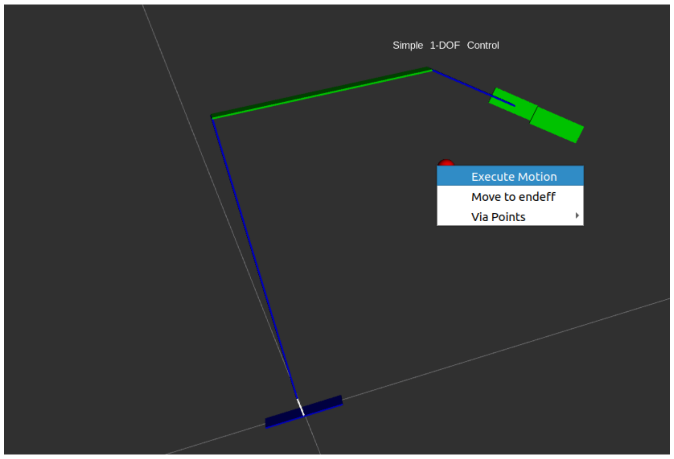


Fig. 6. Context menu for the marker.

Execute motion will make the endeffector move to the specified location. Move to endeff will move the marker to the location of the current position of the endeffector and via points makes available to set a path by specifying several points. With *Via Points/Execute Motion* this path can then be executed[14]. In this case whenever the distance sensor is sensing an object in its range, a flag is published to a topic called *execution_flag*. Another python script is subscribing to this topic and stops the robot motion as soon as the flag changes to ”1”. Stopping the motion is realised by calling *moveit_commander.MoveGroupCommander(”robot_arm).stop()*. Whenever the flag goes back to ”0”, the motion is resumed by setting the initial motion planning goal and executing this motion.

IV. Realtime control of the robot

To include the sensors in all directions, a mount for the link between elbow and hand joint was 3d printed. This way six sensors can be mounted to the robot arm. The design of it is as can be seen in Figure 7. The sensors are mounted with screws to the printed part and the cables are then mounted with zip ties to the robot arm to not have them interfere with the distance sensors themselves.

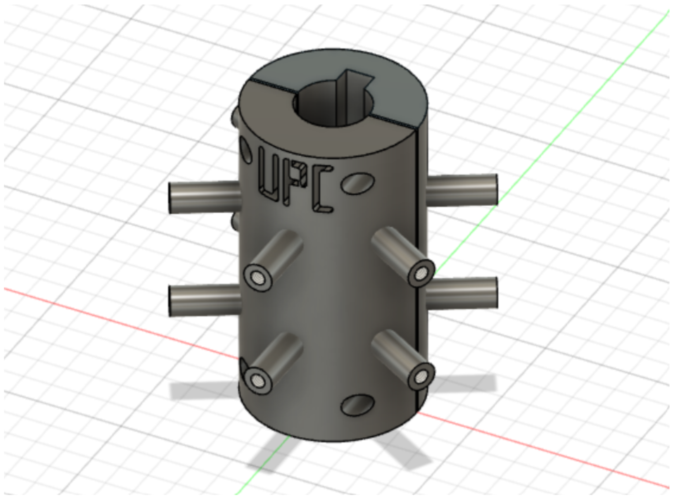


Fig. 7. The distance sensor mount.

One thing that MoveIt! lacks so far is the capability to control in realtime. You would always have to plan and then execute the motion. This limits the speed of the execution and parameters can not be adapted while a motion is executed. In the stop motion control the robot motion was basically stopped to then replan and reexecute the motion at the highest possible rate. The maximum rate on my computer and my setup was at about 10 Hz and the yielding motion was jittery. When trying to make the system faster than that, the robot would just not move anymore. This is because To face this problem and solve the motion control for a realtime system, two approaches can be used. This would either be the Jogarm by Picknik [15] which still uses the MoveIt! library or the cartesian controllers package distributed by fzi - Fraunhofer Forschungszentrum Informatik [16]. Both these approaches have in common, that they allow to modify or adjust the goal position of a robot and then find a smooth trajectory and velocity to reach this given position. Though the Arduino seems to have problems running too many requests and nodes at a time and at some point gets stuck. To solve this problem in a future research, the Arduino should be replaced with a more powerful hardware solution. A possibility could be the Jetson Nano with I2C communication to the Arduino. The I2C bus is generally speaking more reliable then the approach that was used here, creating a Service Server on the Arduino.

V. Conclusion

In this work, it was shown how to set up a robotic arm that uses servos under ROS with Ubuntu 20.04 and Python3. Hereby firstly a simplified urdf model was created. Then the

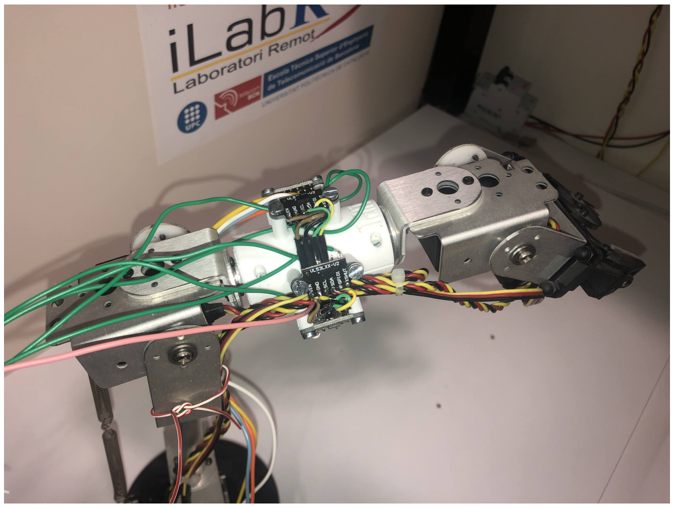


Fig. 8. The robot with the distance sensor array.

inverse kinematics had to be found using the library openrave-With openrave it was now possible to move the endeffector and based on that calculate the joint angles. Then a marker script was created so that a user can specify the desired coordinate and execute the movement. When an object was detected using the 3D printed distance sensor array, the movement was stopped to then only be resumed once the object was removed from the workspace. The robot movement is still jittery, because the Arduino can not handle too many requests at once which is why delays had to be incorporated.

VI. Outlook

Further Investigations could be of the following nature:

- Use the given setup to perform simple pick and place actions
- implement a reinforcement learning setup to make the robot work out its controls automatically
- replace the servos with high quality stepper motors to get a more rigid system
- use more distance sensors, to get a seamless coverage of the entire robot. Alternatively a smart setup could be thought of to minimise the number of sensors and have their detection area overlap.
- newly developed sensors at UPC using nanospheres as proximity sensors could be used instead of the VL53LOX
- use a more powerful Hardware setup like the Jetson Nano for example

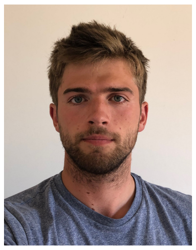
Furthermore it would be very interesting to implement the same robot control with MATLAB to see how the workflow for MATLAB would be in comparison to Python and cpp. For me it was partly very difficult to find good documentation on some of the subjects, wheres MATLAB usually provides an extremly detailed and in depth explanation.

Acknowledgment

Finally I would like to thank Professor Jose Antonio Lazaro for always directing me in the right way and giving feedback wherever it was needed. Also i want to mention Valerio Magnano who was helping out with coding specific questions.

References

[1] Ubuntu 20.04.2.0 lts (focal fossa). https://releases.ubuntu.com/20.04/. (Accessed on 06/17/2021).
[2] noetic/installation/ubuntu - ros wiki. http://wiki.ros.org/noetic/Installation/Ubuntu. (Accessed on 06/15/2021).
[3] catkin build in ubuntu 20.04 noetic? - ros answers: Open source q&a forum. https://answers.ros.org/question/353113/catkin-build-in-ubuntu-2004-noetic/. (Accessed on 06/16/2021).
[4] rosserial - ros wiki. http://wiki.ros.org/rosserial. (Accessed on 06/16/2021).
[5] hs225bb.pdf. https://www.robotshop.com/media/files/pdf/hs645mg.pdf. (Accessed on 06/17/2021).
[6] Introduction to ikfast and prerequisite — framefab_irb6600_support documentation. https://choreo.readthedocs.io/en/latest/doc/ikfast_tutorial.html#openraveinstallation. (Accessed on 04/15/2021).
[7] Ikfast kinematics solver — moveit_tutorials noetic documentation. https://ros-planning.github.io/moveit_tutorials/doc/ikfast/ikfast_tutorial.html. (Accessed on 04/29/2021).
[8] Openrave — ikfast module — openrave documentation. http://openrave.org/docs/latest_stable/openravepy/ikfast/#ik-types. (Accessed on 04/29/2021).
[9] Moveit! setup assistant tutorial — moveit_setup_assistant_tutorial documentation. http://docs.ros.org/en/hydro/api/moveit_setup_assistant/html/doc/tutorial.html. (Accessed on 06/08/2021).
[10] Plugins — moveit. https://moveit.ros.org/documentation/plugins/#kinematicsbase. (Accessed on 04/21/2021).
[11] Writing an inverse kinematics solver for ros — by rohin malhotra — nymble — medium. https://medium.com/nymble/writing-an-inverse-kinematics-solver-for-ros-17b90c8677b5. (Accessed on 04/20/2021).
[12] World's smallest time-of-flight ranging and gesture detection sensor. https://www.st.com/resource/en/datasheet/vl53l0x.pdf. (Accessed on 05/18/2021).
[13] Ros vl53l0x range sensor based on arduino mega 2560 and rosserial — by robotics weekends — robotics weekends — medium. https://medium.com/robotics-weekends/ros-vl53l0x-range-sensor-based-on-arduino-mega-2560-57922804a419. (Accessed on 05/18/2021).
[14] bandasaikrishna/3-dof_manipulator: This repo contains the a 3-dof manipulator ros project. it covers few topics in ros like ros_control, moveit, ikfast plugin and interactive marker. https://github.com/bandasaikrishna/3-DOF_Manipulator. (Accessed on 06/09/2021).
[15] Jogarm: Realtime cartesian motion with moveit — picknik. https://picknik.ai/control/realtime/moveit/2020/05/18/jogarm-realtime-cartesian-motion-with-moveit.html. (Accessed on 05/28/2021).
[16] fzi-forschungszentrum-informatik/cartesian_controllers: A set of cartesian controllers for the ros-control framework. https://github.com/fzi-forschungszentrum-informatik/cartesian_controllers. (Accessed on 05/28/2021).



Yannic Scholtyssek performed his master studies at Technical University Berlin and during his semester abroad at UPC Barcelona he did research on how robotic systems could be implemented using ROS at hand with a simple robot arm.