5. Robot arm

5.1 Introduction

A robotic arm is going to be used to manipulate the end effector through the toilet. In this case the DRV70E robot arm of the brand Delta Electronics is going to be used. Because this robot arm is not on the market yet, Delta Electronics provided a different robot arm to already start working on the software. In this stage of the project the DRV90L robot arm of the brand Delta Electronics is used to start working on the software for the robot arm. Delta Electronics stated that the software for both arms is very similar to each other. A force torque sensor is used for measuring the pressure that the end effector is applying on the toilet. In this way the robot can clean the toilet with the right amount of pressure.

5.2 Technology

5.2.1 Robot arm

The DRV90L robot arm is a six axis articulated robot with a maximum working range of 900 millimeters. The maximum payload that the robot arm can handle is 7 kilogram and has a repeatability of ±0.03 millimeters. Interfacing with robot controller can be done with the DRAStudio software which uses Ethernet communication. DRAStudio is the default robot software that Delta Electronics provides for controlling the robot. Another way to interface with the robot controller is to use the Modbus communication protocol. In this way it’s possible to communicate with the robot without using the DRAStudio software. The supported Modbus protocols are Modbus ASCII, Modbus RTU and Modbus TCP. Modbus ACII and Modbus RTU are both RS323/485 serial communication protocols and Modbus TCP is a TCP based protocol.

5.2.2 Force torque sensor

The fore torque sensor that is used it the FT300 of the brand Robotiq. This sensor can measure forces up to ±300 N in x, y and z direction. And moments in x, y and z direction up to ±30 Nm. The force torque sensor is mounted in between the mounting plate of the robot and the end effector. Interfacing with this sensor can be done using the Modbus RTU protocol. To interface with the sensor over USB a ACC-ADT-USB-RS485 signal converter is used.

5.2.3 ROS

“ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package

management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.” http://wiki.ros.org/ROS/Introduction

5.3 Design choices

5.3.1 Communication

For communication between the NUC and the robot arm it’s chosen that the Modbus TCP protocol will be used. This protocol has the highest transmission speed compared to the other compatible protocols. The transmission speed is important for the live visualization in ROS. With a low transmission speed the visualization will look jerky due to the lower update rates.

5.3.2 Path planning

For path planning it’s chosen to use the default path planner that is used inside the robot controller. This is because it’s not possible to implement a external path planner with this robot arm. To implement a external path planner, the robot arm has to be controlled by sending real time position values for each joint to the robot controller. This is not possible to do with this robot controller. By using the internal path planner in the robot controller there will be some limitations. For example when using a external path planner, it’s completely controllable how the robot moves to a certain point. In this way it’s possible to plan a path around obstacles.

5.4 Implementation

The complete code can be found at https://github.com/RemcoKuijpers/delta.

5.4.1 Robot control

In the sub repository “arm_driver” of the “delta” GitHub repository the code for controlling the robot can be found. In the text below the project structure of “arm_driver” is visible.

```
tree --dirsfirst
.
├── src
│   ├── end_effector.py
│   ├── joint_position_publisher.py
│   ├── remote_control.py
│   └── send_commands.py
├── srv
│   ├── power.srv
│   ├── reset_errors.srv
│   └── teach_position.srv
├── yaml
│   ├── joint_limits.yaml
│   └── poses.yaml
├── CMakeLists.txt
└── package.xml

3 directories, 11 files
```

The “arm_driver” package contains all the code that communicates with the robot.

5.4.1.1 Send_commands

In the send_commands.py python script (ROS node) a class “DRV90L” is defined for controlling the robot arm. In the table below the available functions are visible with a short description. For a more detailed description have a look at the docstrings inside the functions.

DRV90L	
Function	**Description**
enableRobot()	Enables robot
disableRobot()	Disables robot
sendPositionMove(x,y,z,rx,ry,rz,speed,frame)	Send end effector of robot to position
sendArcMove(p1,p2)	Send a arc move (to p2 through p1)
jogRobot(direction, stop)	Jog robot
goHome()	Home robot
writeDigitalOutput(output,state)	Write digital output
getUserDigitalInputs()	Read digital inputs
getUserDigitalOutputs()	Read digital outputs
getToolPosition()	Get tool position (x,y,z,rx,ry,rz) in mm
getJointPositions()	Get joint positions in radians
saveToolPose(name,pose)	Save current position to yaml file
teachCurrentToolPose(msg)	Callback for teach_position ROS service
getSavedToolPose(name)	Get a teached position (x,y,z,rx,ry,rz) in mm
resetUserDigitalOutputs()	Set all digital outputs to 0
resetErrors(msg)	Reset erros messages of robot arm

When a instance of DRV90L is created, three ROS services are created: /reset_robot, /power_robot and /teach_position. With the ROS service /reset_robot it’s possible to reset the errors in the robot, this has to be done for example after a emergency stop. With the ROS service /power_robot it’s possible to enable and disable the robot. With the ROS service /teach_position it’s possible to save the current position in a yaml file with a given name.

5.4.1.2 joint_angle_publisher

A continuous stream of the actual joint angles of the robot arm get received inside the joint_angle_publisher.py python script (ROS node). The angles are received in degrees and get converted to radians. Then the joint angles in radians get published to the ROS topic /joint_states. This will make the ROS model in Rviz move like the real robot arm. Rviz is a 3D robot visualizer for the Robot Operating System (ROS) framework. More about the visualization will be explained in chapter *5.4.3 ROS visualization*.

5.4.1.3 end_effector

All the code for controlling the end effector from the NUC side is in the end_effector.py python script. This file contains a class named EndEffector. This class contains all the functions to control the end effector. This class only controls the high level control for the end effector by sending commands over serial communication to the low level controller for the end effector.

EndEffector	
Function	**Description**
startDispensing()	Start dispensing liquid, opens the valve
stopDispensing()	Stop dispensing liquid, closes the valve
startBrushing()	Starts rotating the brush
stopBrushing()	Stops rotating the brush

5.4.1.4 remote_control

Code for remote controlling the robot arm including the end effector is in the remote_control.py python script. A Sony PlayStation 4 wireless controller is used for controlling the robot. The PS4 controller can connect to the NUC via Bluetooth, in this way it's possible to wireless control the robot. For now the remote control is only working for the robot arm and end effector. But in a later stage with a working AGV, remote control for the AGV can be added. Currently it's possible to control all the functions of the end effector and some functions of the robot arm:

- Jogging robot in all directions (x,y,z,rx,ry,rz), one direction at the time.
- Enable robot
- Disable robot
- Homing robot
- Reset errors

5.4.2 Force torque sensor

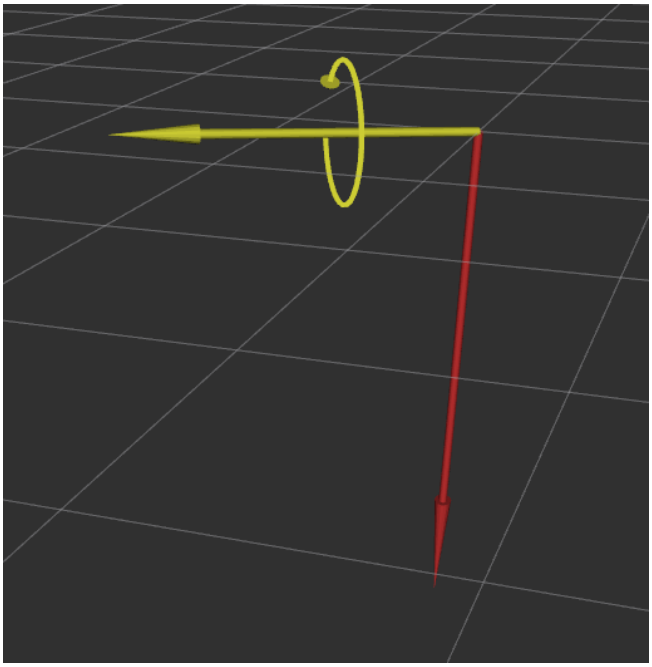
In the sub repository "ft300_sensor" of the "delta" GitHub repository the code for the force torque sensor can be found. This sub project's structure can be found below.

```
tree --dirsfirst
.
├── src
│   └── ft300.py
├── yaml
│   └── calibration.yaml
├── CMakeLists.txt
└── package.xml

2 directories, 4 files
```

This project contains one python script and a yaml file for saving calibration values. The ft300.py python script (ROS node) contains the code for FT300 force torque sensor of Robotiq. When running the script the script will receive the force values in x,y,z direction and the torque in rx,ry,rz direction. These values will be published to the ROS topic /ft_data and /stamped_ft_data. The message type is Wrench for /ft_data and WrenchStamped for /stamped_ft_data. The /stamped_ft_data ROS topic is used for visualizing the force torque data in Rviz. It's also possible to calibrate the sensor with the python script. When calibrating the actual force and torque values will be set as the "zero" values. These values are saved in the calibration.yaml file.

In the picture below measurements are shown as a three dimensional vector. The yellow arrow is the three dimensional vector of the moments, and the red arrow is the three dimensional vector of the forces.



5.4.3 ROS visualization

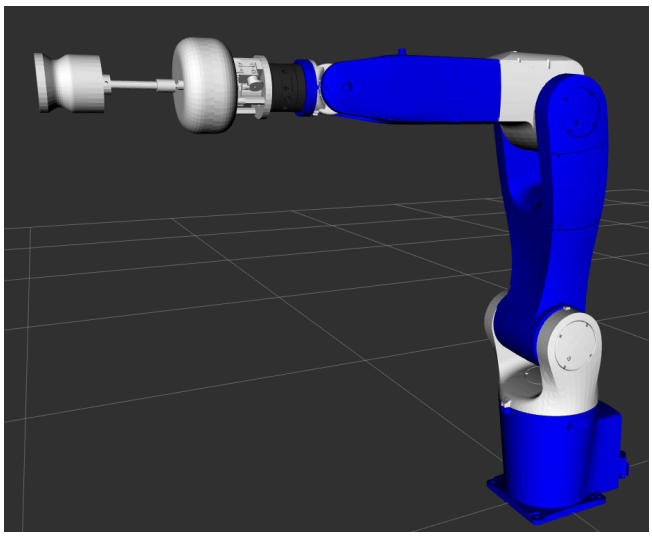
In the sub repository “arm_description” of the “delta” GitHub repository the sources for the ROS visualization can be found. This repository contains all files regarding to the 3D ROS model of the robot. The 3D ROS model is built with URDF (Universal Robot Description Format) files. A URDF file contains of links and joints, where links are connected together with joints. For this 3D ROS model two URDF files are connected together. The robot arm and the end effector are two separate URDF files.

5.4.3.1 URDF robot arm

The URDF file of the robot arm consists of seven links and six joints. Each link is coupled to a 3D model of the link. All six joints are revolute joints in this robot, so the links can rotate with respect to each other around a certain axis of the joint. In appendix 1 a visual representation of the robot arm URDF is shown.

5.4.3.2 URDF end effector

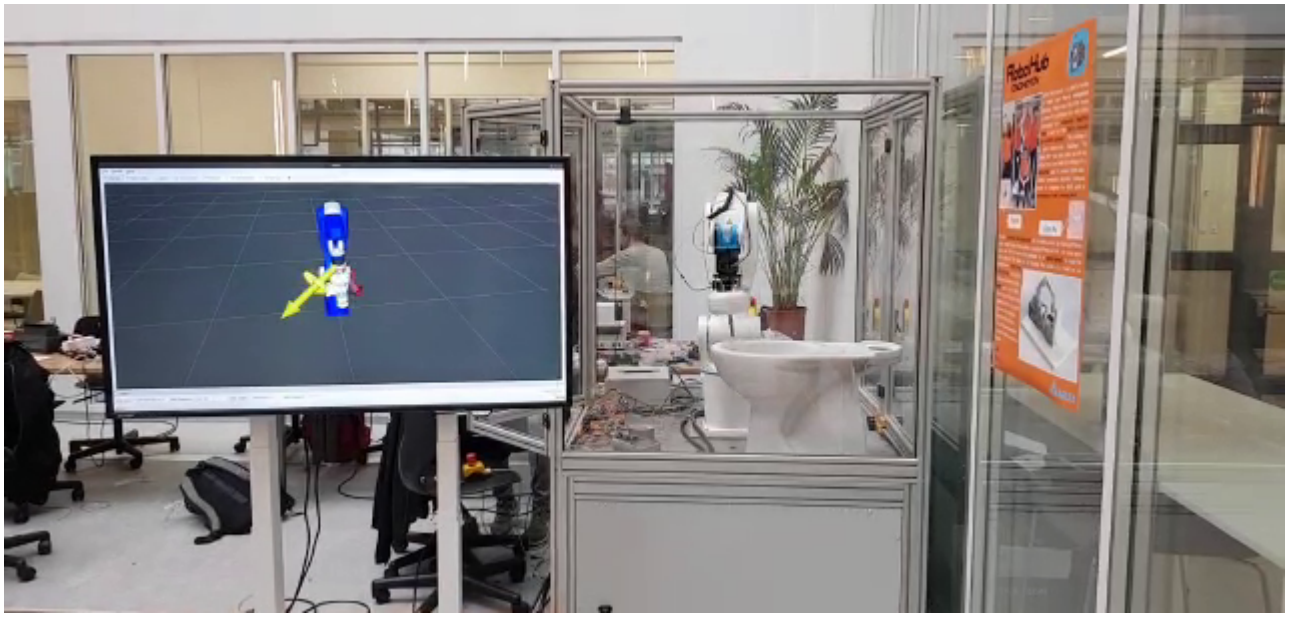
The URDF file of the end effector consists of three links and two joints. In this case of the end effector both joints are fixed. There is no need to have a moving end effector in this case, because when the brush is rotating the brush will stay at the same place because of the symmetry of the brush. In appendix 2 a visual representation of the end effector URDF is shown.
The robot model with the robot arm and end effector connected together is shown in the picture below.



5.4 Results

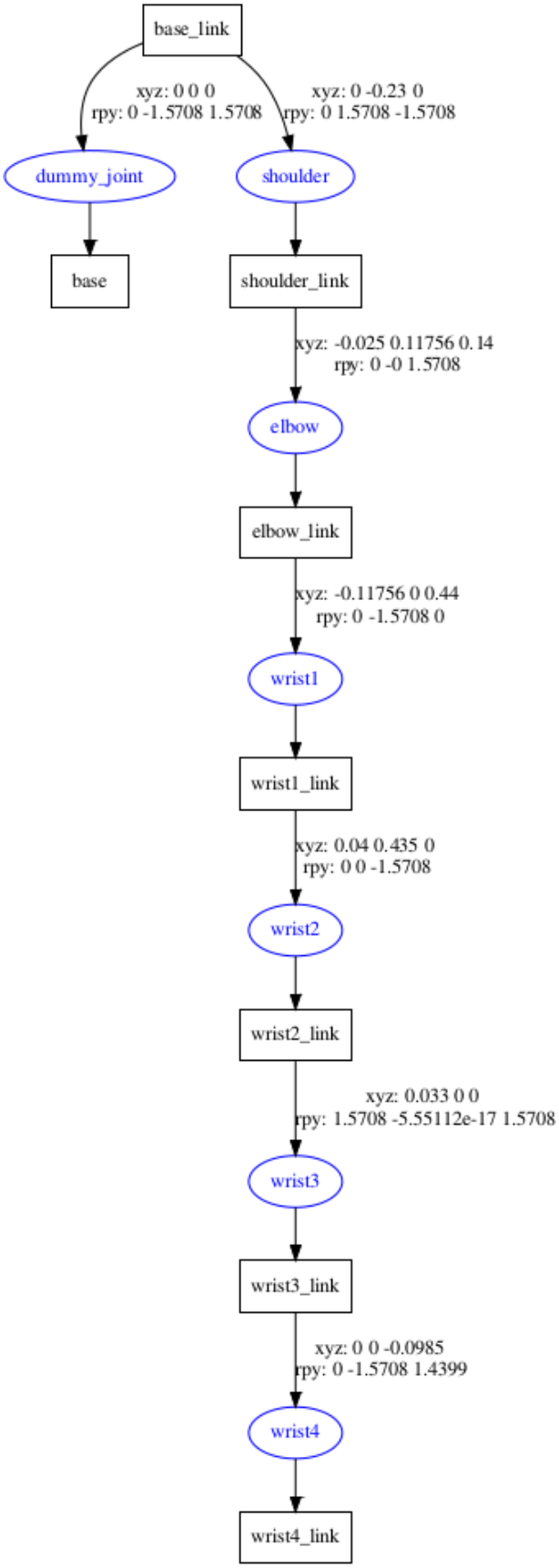
When the end effector, force torque sensor and the robot arm are combined together, the robot arm is able to brush a toilet at a hard coded position. The moves that the robot arm does, are hard coded too. In the picture below it's visible that the robot arm is brushing the toilet with the end effector. The forces and moments that the force torque sensor is measuring are visible by the red and yellow arrows. The yellow arrow is clearly bigger then red arrow. This is because the force on the brush is creating a moment on the force torque sensor. The force on the brush can be calculated by dividing the moment with the distance from sensor to end effector. For now it's not implemented yet, because the final design of the end effector is not yet on the robot arm, and it's still unknown what the required force is to properly clean the toilet.

$$F_{brush} = \frac{M_{sensor}}{d_{brush}}$$



Appendix

1. Visual representation of robot URDF



2. Visual representation of end effector URDF