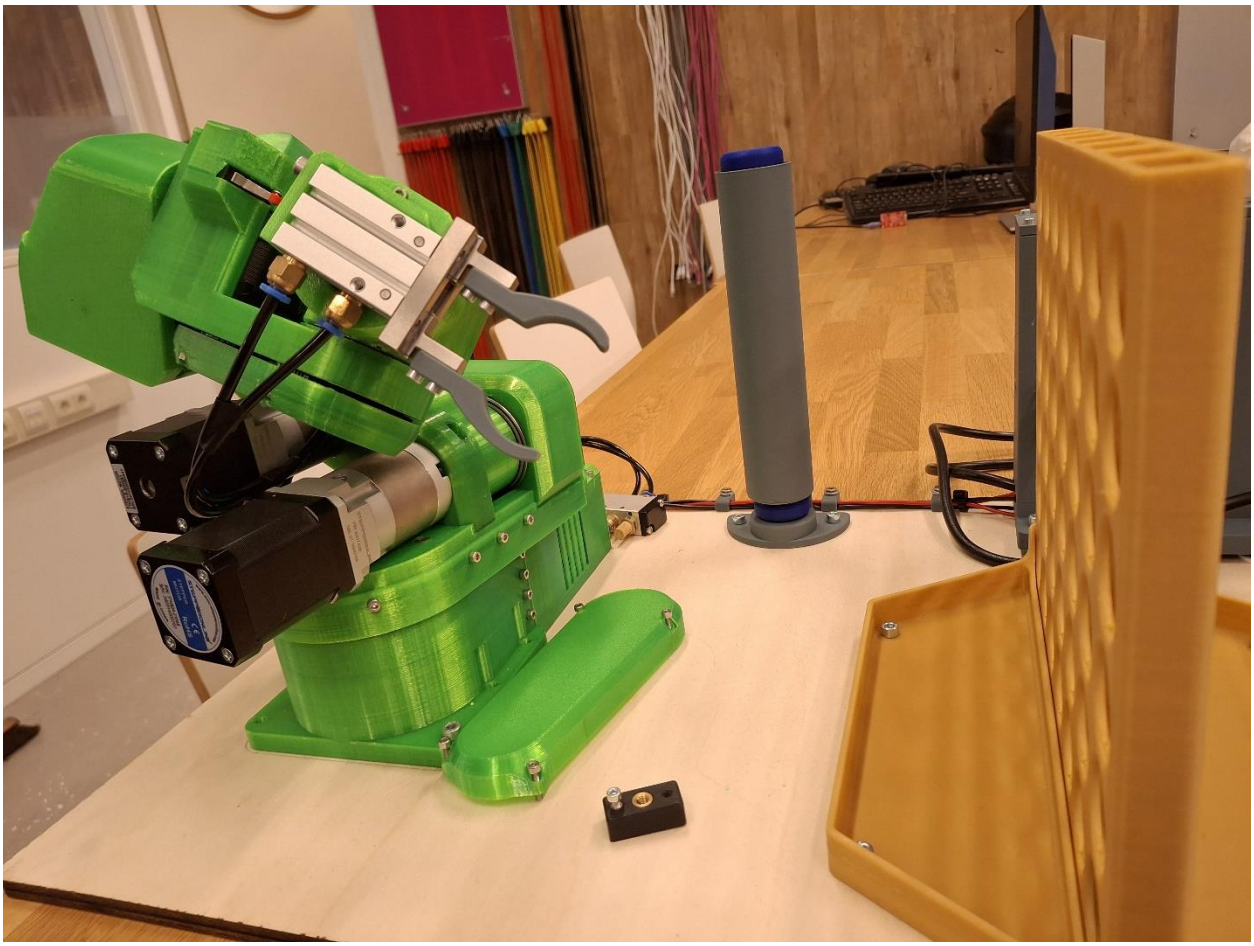
Assignment – PAROL6 model scene for “connect-4 ” game

The PAROL6 robot is a simple robot that can be used to play “connect-4”. The idea is that it will pick up a disk from and tube and puts the disk into the game board (see lab). It is the idea that the Robot plays against a human being.

The whole software has to be implemented in ROS2. The first necessary step is to set up a model in ROS2 with the correct physical dimensions of the robot. For this a package(s) has to be implemented in your ROS2 environment.

See example of real set up in lab.



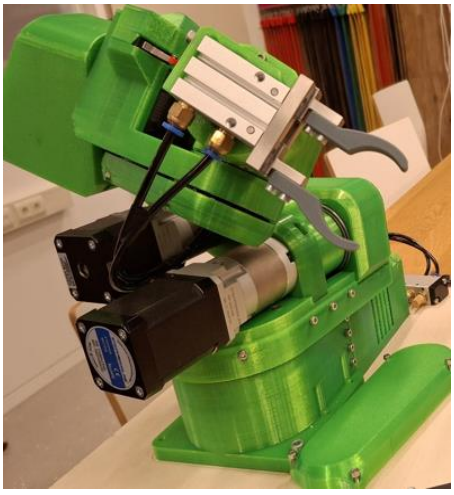
Setting up Robot model – task1

In this assignment a correct robot model for PAROL6 in your own ROS environment has to be set up.

You can use as starting point the information from

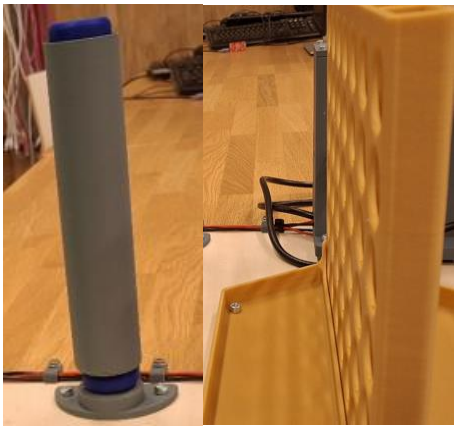
(GitHub - PCrnjak/PAROL6-Desktop-robot-arm: BOM, STL files and instructions for PAROL6 3D printed robot arm)

- The STLs from the website can be used
- The URDF should follow the same structure as the robot in the lecture
- All necessary collision using simple geometry should be implemented (note: for sure the collisions for the disks need to be added)
- The color of the links should be yellow and orange (alternating). The gripper should be grey.
- The model should be successfully visualized in RVIZ2.



Adding objects to your Robot scene (world) – task2

Add the three “fixed” objects to the Robot scene (world). There are several ways to add these to our robot scene in RVIZ. For the three options described below you have implement a solution. For each option please use a separate package. The Stl of the objects you can get from the course (moodle).



a) “fixed” link (environment object) in your robot urdf
    i. Robot urdf
    ii. RIVZ = visualization
b) Mesh marker
    i. Mesh marker publisher node
    ii. RIVZ2 = visualization
c) Static transform (zie appendix)
    i. Separate urdf
    ii. Static transform publisher node = places object in world
    iii. Robot state publisher = publishes urdf to RVIZ2
    iv. RVIZ2 = visualization

For b,c the location in the scene should be parameters.

Note: keep always in mind

- CMakeList.txt
- Launch.xml
- User interface action in RVIZ

Other implemented options can be considered as bonus ...

Note: For the assessment of this assignment you need to demonstrate all solutions and you should be able to explain all of them

Appendix

Note: Code snippets are retrieved by AI ... thus it might be not correct

adding object to visualization: mesh marker

✅ Step 1: STL File

Make sure you have an STL file (e.g., `object.stl`) placed in your ROS2 package, typically under `share/<your_package>/meshes/`.

✅ Step 2: C++ Node

C++

```cpp
// file: src/mesh_marker_publisher.cpp

#include "rclcpp/rclcpp.hpp"
#include "visualization_msgs/msg/marker.hpp"

class MeshMarkerPublisher : public rclcpp::Node
{
public:
    MeshMarkerPublisher() : Node("mesh_marker_publisher")
    {
        publisher_ = this->create_publisher<visualization_msgs::msg::Marker>("visualization_marker", 10);
        timer_ = this->create_wall_timer(std::chrono::seconds(1), std::bind(&MeshMarkerPublisher::publish_marker, this));
    }
```

adding object to visualization: option Static Transform Publisher

These code snippets are retieved from Copilot examples of version b)

✅ Step 2: Static Transform Publisher (C++)

Create a node `static_tf_publisher.cpp`:

C++

```cpp
#include "rclcpp/rclcpp.hpp"
#include "tf2_ros/static_transform_broadcaster.h"
#include "geometry_msgs/msg/transform_stamped.hpp"

class StaticTFPublisher : public rclcpp::Node
{
public:
    StaticTFPublisher() : Node("static_tf_publisher")
    {
        broadcaster_ = std::make_shared<tf2_ros::StaticTransformBroadcaster>(this);

        geometry_msgs::msg::TransformStamped transform;
        transform.header.stamp = this->get_clock()->now();
        transform.header.frame_id = "map";
        transform.child_frame_id = "object_link";

        transform.transform.translation.x = 2.0;
        transform.transform.translation.y = 1.0;
        transform.transform.translation.z = 0.5;

        transform.transform.rotation.x = 0.0;
        transform.transform.rotation.y = 0.0;
        transform.transform.rotation.z = 0.0;
        transform.transform.rotation.w = 1.0;
```

✅ Step 3: Robot State Publisher (C++)

Create a node `robot_state_publisher_node.cpp`:

C++

```
            return;
        }

        auto rsp = std::make_shared<robot_state_publisher::RobotStatePublisher>(model);
        rsp->publishFixedTransforms(true);
    }
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<URDFPublisher>());
    rclcpp::shutdown();
    return 0;

```