Large Scale Cartesian Robot for automatic irrigation using URDF and ROS

Overview

This project involves the design and implementation of a high-scale cartesian robot using the Robot Operating System (ROS) and the Unified Robot Description Format (URDF) file. The robot has a size of 30x10x10 meters and is suitable for industrial applications that require large distances.

The robot's mechanical structure is defined in the URDF file, which includes the joint types, dimensions, and limits. The URDF file also defines the robot's visual and collision models, which are used for simulation and collision detection. The robot's kinematics and dynamics are modelled using the URDF-based robot description. The robot's control system is implemented using ROS, which provides a framework for developing and integrating software components.

Quick Start Guide

Requirements

For simulating the system, the following elements are required:

- Ubuntu 20.04.06 LTS (Focal Fossa): https://releases.ubuntu.com/focal/
- ROS Noetic: http://wiki.ros.org/noetic/Installation/Ubuntu
- RViz for Noetic: http://wiki.ros.org/rviz/UserGuide
- Xacro: http://wiki.ros.org/xacro

Setup

Download or clone contents of the ROS Folder to your designated Catkin Workspace source folder, in this example the folder is located at:

```
cd home/user/catkin_ws/src
```

Inside the cartesian_irrigator folder, the following folder structure should be included:

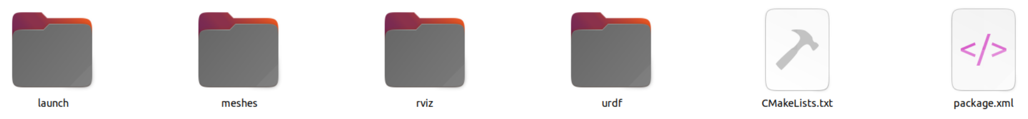


Figure. Folder Structure

Once downloaded execute the following command:

```
catkin_make
```

After that, run the setup.bash for the src:

```
source devel/setup.bash
```

Operation

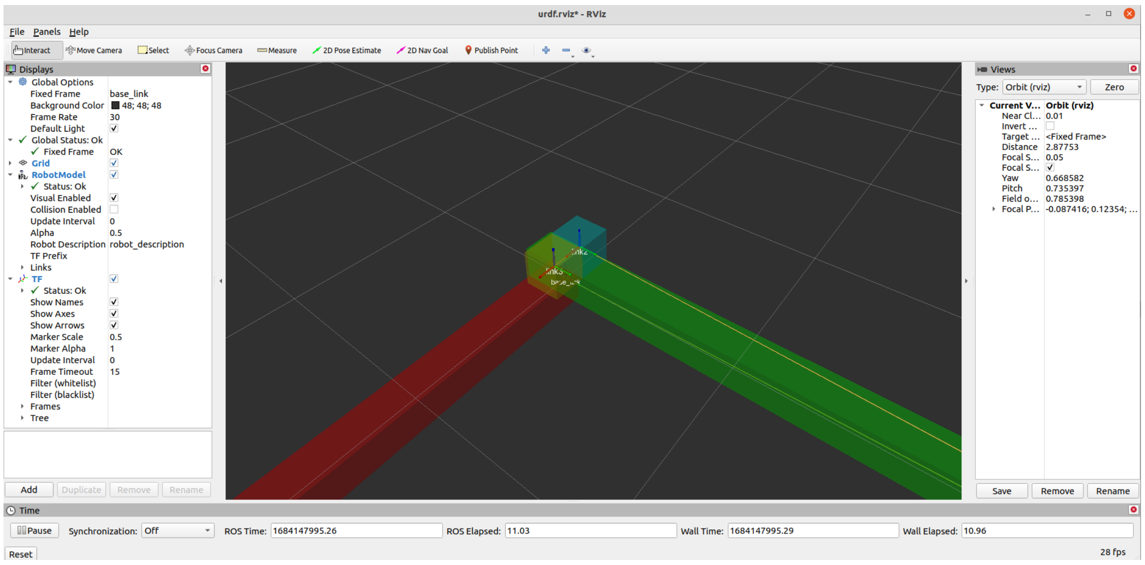
Open up a terminal and start ROS using this command:

```
roscore
```

Once it has mounted the instance, open up a new terminal and run the launch command created to launch RViz, as well as the necessary nodes with the URDF callback integrated, this file contains all the necessary components:

```
roslaunch cartesian_irrigator display.launch
```

The previous command will launch an RViz instance with the URDF robot displayed, such as the following figure describes:



```
Global Options
  Fixed Frame            base_link
  Background Color       48; 48; 48
  Frame Rate             30
  Default Light          ✓
✓ Global Status: Ok
  ✓ Fixed Frame          OK
Grid                     ✓
RobotModel               ✓
  ✓ Status: Ok
  Visual Enabled         ✓
  Collision Enabled      ☐
  Update Interval        0
  Alpha                  0.5
  Robot Description      robot_description
  TF Prefix
  Links
TF                       ✓
  ✓ Status: Ok
  Show Names             ✓
  Show Axes              ✓
  Show Arrows            ✓
  Marker Scale           0.5
  Marker Alpha           1
  Update Interval        0
  Frame Timeout          15
  Filter (whitelist)
  Filter (blacklist)
  Frames
  Tree

Type: Orbit (rviz)   Zero
Current V...   Orbit (rviz)
  Near Cl...   0.01
  Invert ...   ☐
  Target ...   <Fixed Frame>
  Distance     2.87753
  Focal S...   0.05
  Focal S...   ✓
  Yaw          0.668582
  Pitch        0.735397
  Field o...   0.785398
  Focal P...   -0.087416; 0.12354; ...

Synchronization: Off   ROS Time: 1684147995.26   ROS Elapsed: 11.03   Wall Time: 1684147995.29   Wall Elapsed: 10.96
28 fps
```

Figure. RViz

In order to move the robot in a simulation environment, an instance of the joint_state_pusblisher GUI window is needed, to run it use this command in a new terminal, the following figure will appear afterwards:

```
rosrun joint_state_publisher_gui joint_state_publisher_gui
```

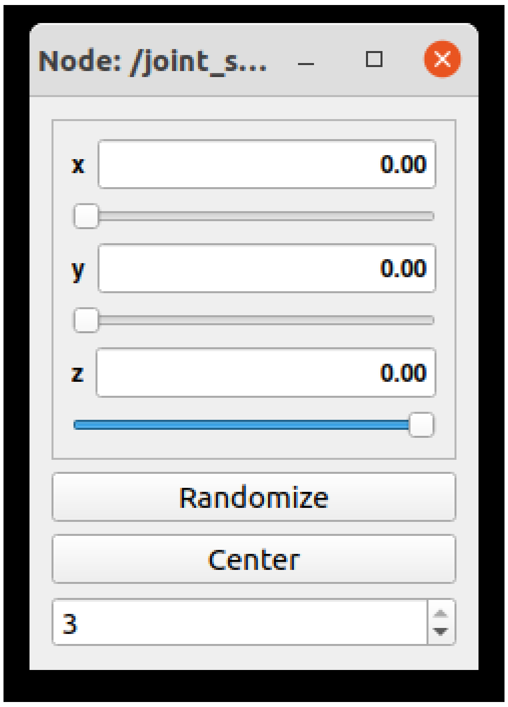
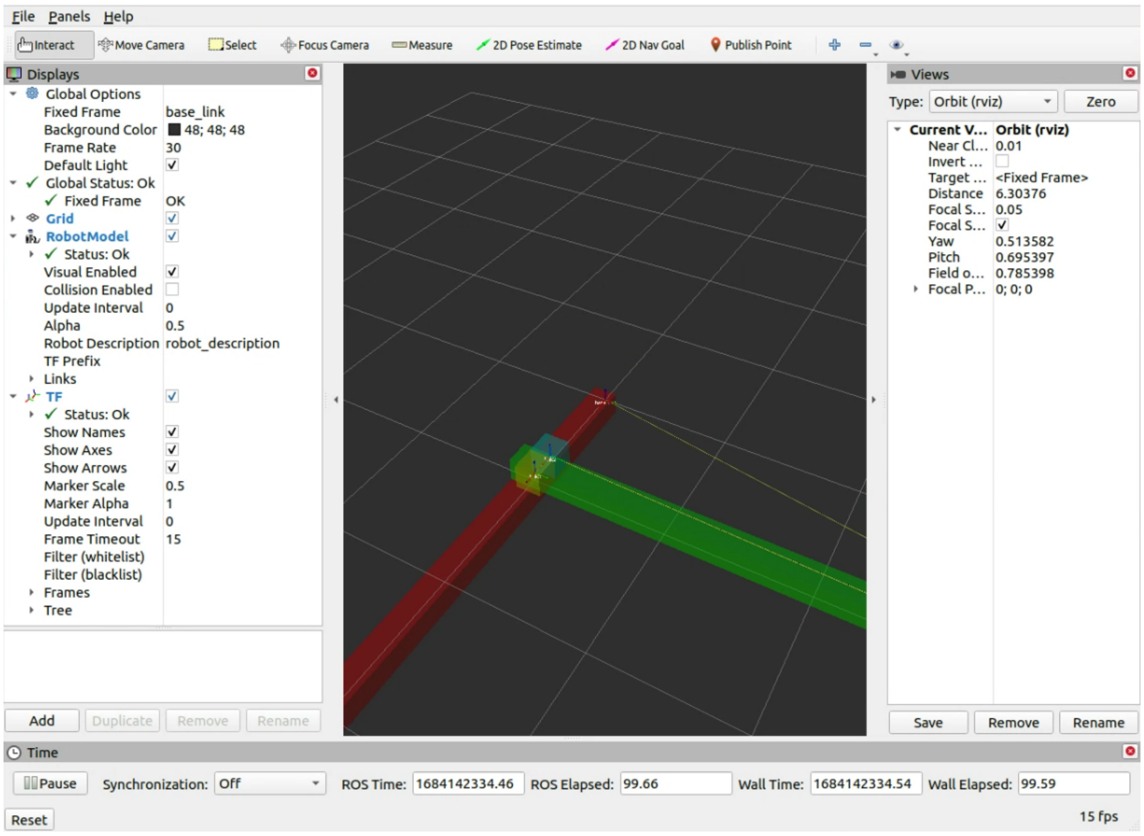


Figure. Joint State GUI

With the included sliders, you can move the robot in the X, Y and Z axis, with the available constraints and limits made by the URDF file itself, the following figures demonstrate the simulation with the sliders.

```
X Movement:
```



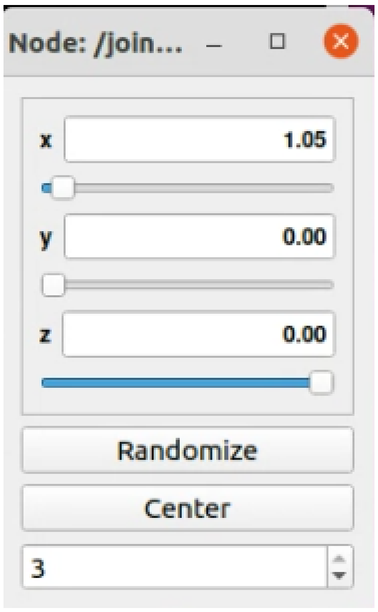
Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	0
Alpha	0.5
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
✓ Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	0.5
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Filter (whitelist)	
Filter (blacklist)	
Frames	
Tree	

Type: Orbit (rviz)

Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	
Target ...	<Fixed Frame>
Distance	6.30376
Focal S...	0.05
Focal S...	✓
Yaw	0.513582
Pitch	0.695397
Field o...	0.785398
Focal P...	0; 0; 0

Synchronization: Off ROS Time: 1684142334.46 ROS Elapsed: 99.66 Wall Time: 1684142334.54 Wall Elapsed: 99.59

15 fps



Node: /join...

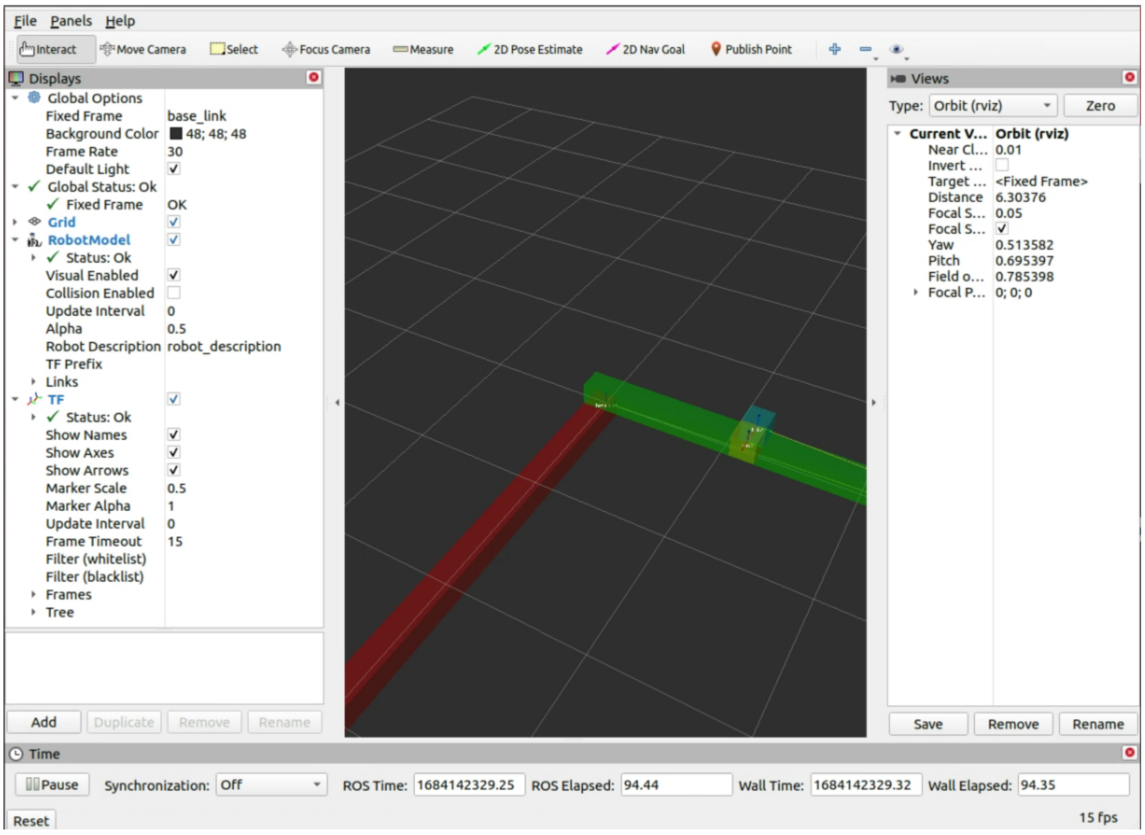
x 1.05

y 0.00

z 0.00

3

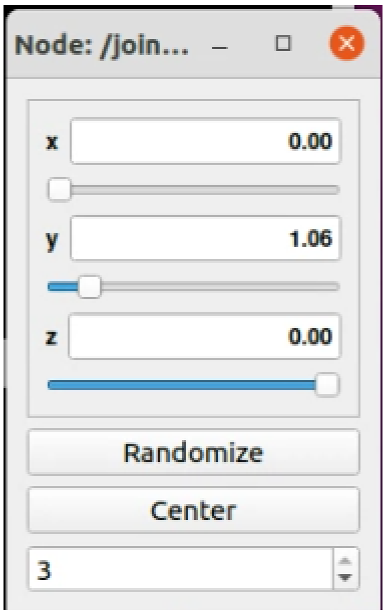
```
Y Movement:
```



Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Ok
Fixed Frame OK
Grid ✓
RobotModel ✓
Status: Ok
Visual Enabled ✓
Collision Enabled
Update Interval 0
Alpha 0.5
Robot Description robot_description
TF Prefix
Links
TF ✓
Status: Ok
Show Names ✓
Show Axes ✓
Show Arrows ✓
Marker Scale 0.5
Marker Alpha 1
Update Interval 0
Frame Timeout 15
Filter (whitelist)
Filter (blacklist)
Frames
Tree

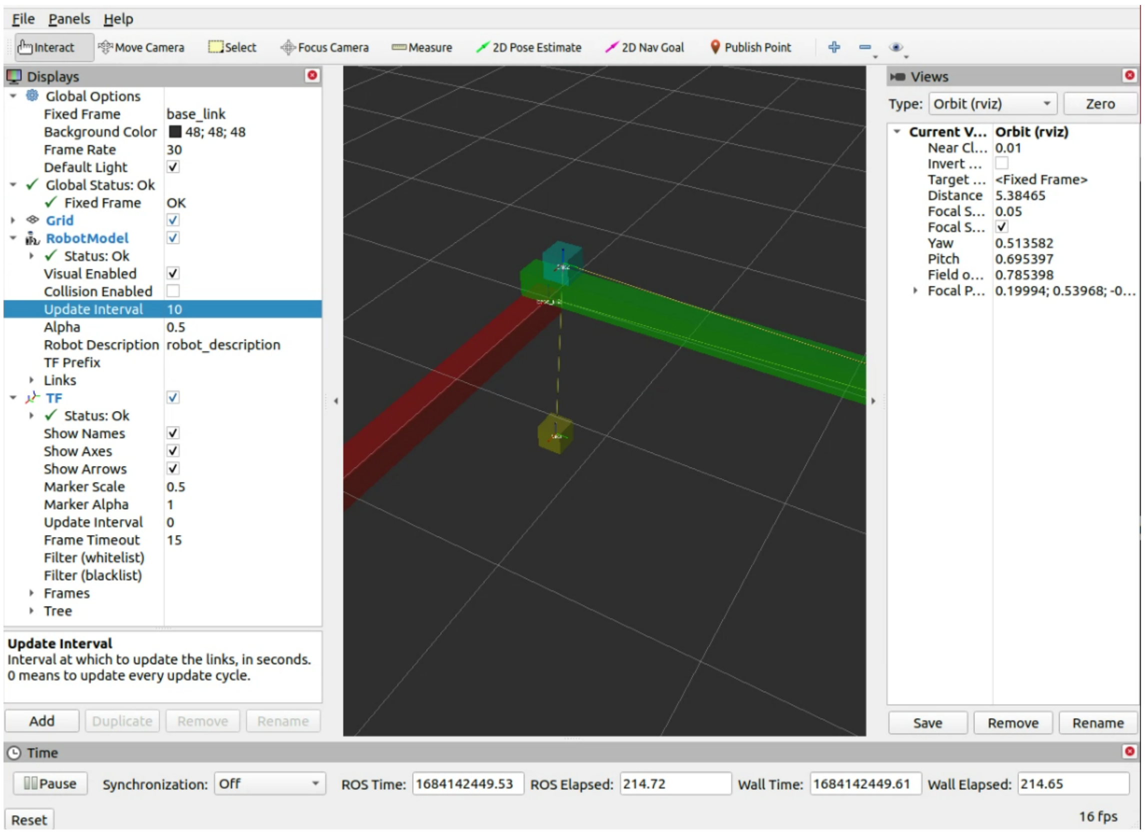
Type: Orbit (rviz)
Current V... Orbit (rviz)
Near Cl... 0.01
Invert ...
Target ... <Fixed Frame>
Distance 6.30376
Focal S... 0.05
Focal S... ✓
Yaw 0.513582
Pitch 0.695397
Field o... 0.785398
Focal P... 0; 0; 0

Time
Synchronization: Off ROS Time: 1684142329.25 ROS Elapsed: 94.44 Wall Time: 1684142329.32 Wall Elapsed: 94.35
15 fps



Node: /join...
x 0.00
y 1.06
z 0.00
3

Z Movement:



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Fixed Frame	OK
Grid	✓
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	10
Alpha	0.5
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	0.5
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Filter (whitelist)	
Filter (blacklist)	
Frames	
Tree	

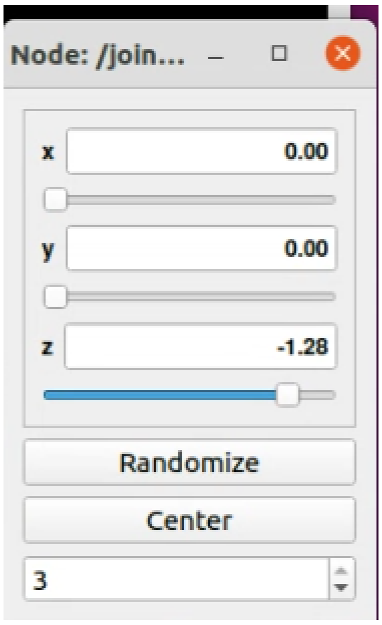
Update Interval
Interval at which to update the links, in seconds. 0 means to update every update cycle.

Type: Orbit (rviz) Zero

Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	
Target ...	<Fixed Frame>
Distance	5.38465
Focal S...	0.05
Focal S...	✓
Yaw	0.513582
Pitch	0.695397
Field o...	0.785398
Focal P...	0.19994; 0.53968; -0...

Synchronization: Off ROS Time: 1684142449.53 ROS Elapsed: 214.72 Wall Time: 1684142449.61 Wall Elapsed: 214.65

16 fps



x	0.00
y	0.00
z	-1.28

3

In order to use the robot in a real-life environment, connect to the node **robot_state_publisher** using your language of preference.

Project Design

Robot Design

The design of the high-scale cartesian robot for automatic irrigation with dimensions of 30x10x10m was intended to provide a solution that facilitates the use of robotics in agriculture and botanics. With the increasing need for automation in the agriculture industry, the robot's design was tailored to provide a scalable solution that could be used in different applications.

The robot's size is a significant advantage, as it can cover a large area and perform its functions without requiring significant manual intervention. The robot's ability to navigate through fields and cover large distances makes it ideal for use in a variety of crop management applications. The robot's design was also intended to address the challenges associated with traditional irrigation methods. With the colour recognition system, the robot can identify plants that need irrigation, reducing the need for water wastage and optimizing plant development. This functionality significantly reduces the cost of labour associated with manual irrigation and improves overall plant health.

The size of the joints was measured based on the material for the implementations that being, a 1x1 inch Steel PTR, giving both structural integrity as well as low weight compared to more standardised PTR sizes. The following parts define the sizes described in the URDF:

```xml
<robot name="prismatic_cartesian_robot">
  <link name="base_link">
    <visual>
      <origin rpy="0 0 0" xyz="15 0 0"/>
      <geometry>
        <box size="30 0.025 0.025"/>
      </geometry>
    </visual>
  </link>
```

To standardize, the proposed implementation uses the same PTR to save on costs and logistics, this size was used due to the fact of having around 1.3 to 1.46 kg of weight per meter, focusing on the lowest weight but incorporating industry standards such as Steel PTR, it was defined a 1x1 transversal PTR for the Y axis was optimal since the system should only really carry the y axis element with the x-axis motor, with the proposed solution of an AZM98AC-HS100+AZD-A+CC005VZF motor capable of sustaining the load in a Rack-and-Pinion Drive Mechanism with 52 N per meter with a 90mm flange. The proposed load plus all the other motor falls around 16kg which at 10m it should need around 32N per meter having plenty enough for more applications or Addons.

Figure. AZ Series Motor

The following code was contructed to define the robot constraints and limits, as well as the type of joint utilized:

```
<?xml version="1.0" ?>
<!--
=================================================================================== --
>
<!-- |    This document was autogenerated by xacro from robot.xacro
| -->
<!-- |    EDITING THIS FILE BY HAND IS NOT RECOMMENDED
| -->
<!--
=================================================================================== --
>
<robot name="prismatic_cartesian_robot">
  <link name="base_link">
    <visual>
      <origin rpy="0 0 0" xyz="15 0 0"/>
      <geometry>
        <box size="30 0.025 0.025"/>
      </geometry>
    </visual>
  </link>
  <joint name="x" type="prismatic">
    <origin rpy="0 0 0" xyz="0.0125 4.91 0.025"/>
    <axis xyz="1 0 0"/>
    <limit effort="100" lower="0" upper="30" velocity="1"/>
    <parent link="base_link"/>
    <child link="link1"/>
  </joint>
  <link name="link1">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.18 10 0.18"/>
      </geometry>
      <material name="Green1">
        <color rgba="0 1 0 1"/>
      </material>
    </visual>
  </link>
  <joint name="y" type="prismatic">
    <origin rpy="0 0 0" xyz="0 -4.73 0.025"/>
    <axis xyz="0 1 0"/>
    <limit effort="100" lower="0" upper="9.46" velocity="1"/>
    <parent link="link1"/>
    <child link="link2"/>
  </joint>
  <link name="link2">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.025 0.025 0.025"/>
      </geometry>
      <material name="Cyan1">
        <color rgba="0 0.9 0.9 1.0"/>
      </material>
    </visual>
  </link>
  <joint name="z" type="prismatic">
    <origin rpy="0 0 0" xyz="0.025 0 0"/>
    <axis xyz="0 0 1"/>
    <limit effort="100" lower="-9.64" upper="0" velocity="1"/>
    <parent link="link2"/>
    <child link="link3"/>
  </joint>
  <link name="link3">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.18 0.18 0.18"/>
      </geometry>
      <material name="Yellow2">
        <color rgba="0.8 0.8 0 1.0"/>
      </material>
    </visual>
  </link>
</robot>
```

The important parts to note about this code is that is a 3 joint cartesian robot, so each joint is based of two links in a prismatic type of implementation, defined by the joint type called "prismatic". Another part to emphasize is the visual elements, elements which in a real life implementation wouldnt be considered for the transformations. The robot uses offsets for their elements since the sliding elements for the rack would take space and the robot would collide, soi the calculated workspace of the proposed solutions falls onto a 29.5x9.5x10m space.

MATLAB Implementation

For the proposed solution, the colour recognition algorithm was constructed in MATLAB, which recognizes a colour label used to determine the plant used and the necessary components needed, in this case, irrigation, the recognition saves the centroid of the element and returns the position from Home that the label acquires. For more information, visit the following repo:

https://github.com/Githubense/raspberryColorVision

Connection

The proposed solution is composed of three major elements, the MATLAB Code, the ROS files, and the connection between those environments. Utilizing the ROS Toolbox, the MATLAB code can convert the x,y,z data from the image sensor to x,y,z positions in the world, and from that the robot can communicate through the robot_state_publisher and the joint_state_publisher to calculate the movements needed to reach that location and have it saved. The ROS part of the project has already implemented the use of those nodes, so the implementation not only with the MATLAB code but with any interface for ROS can utilise the present repo to modify it for new applications. The MATLAB code with ROS integration automatically handles the nodes as well as the remote connection to the ROS controller.

References

Corke, P. (2017) Robotics, Vision & Control, Springer, ISBN 978-3-319-54413-7. Ivanov, I. (2021) Raspberry Pi 4 + Raspberry Pi Camera case, Sketchfab, retrieved from: https://sketchfab.com/3d-models/raspberry-pi-4-raspberry-pi-camera-case-7daba0cf330844b0aa425e5d85525436 The MathWorks, Inc. (2023) Instalar productos con conexión a Internet, MathWorks, retrieved from: https://la.mathworks.com/help/install/ug/install-products-with-internet-connection.html Pimienta, A. (2023) Color Image Recognition with Kinematic Positioning and modular feed for MATLAB®, Github, retrieved from: https://github.com/Githubense/raspberryColorVision