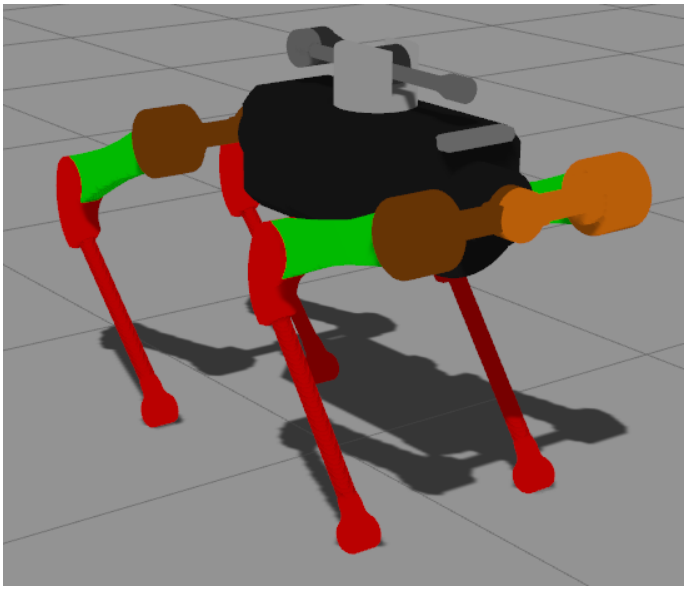
Mini Project Report

Christos Vasileios Kokas

11-10-2021



Contents

1 Abstract **3**

2 Modelling of ARGOS **3**
2.1 Champ Setup Assistant 3
2.2 Champ Setup Assistant Assumptions 4
2.3 URDF file 5
2.3.1 Mass Properties 5
2.3.2 Xacro Language 6
2.3.3 Creating the Model 7
2.3.4 Gazebo References and Plugins 21
2.4 ROS control 25

3 Stereo Camera to Laserscan **25**
3.1 Stereo Camera Topics 25
3.2 Packages Used to Produce Laserscan 26
3.2.1 Stereo_Image_Proc 26
3.2.2 Disparity_image_proc 27
3.2.3 Depthimage_to_Laserscan 28
3.3 Launch Files 28

4 Challenges of the Project **32**

5 Results **34**

6 Improvements that could be Made **39**

1 Abstract

The goal of this project is to build a simulation framework for ARGOS, which is a quadruped robot with legs of three actuated degrees of freedom. By using Champ and gmapping ROS packages the robot will be able to move to a designated spot on the map while avoiding obstacles detected by a stereo camera. This project contains in detail the steps taken to model the robot using the macro language xacro, to produce a URDF file and, following that, use Champ Setup Assistant to create the package for ROS. After creating the ROS package the pid gains and the gait parameters have to be tuned for the better performance of the robot. In order for the robot to be able to avoid obstacles while navigating the map, gmapping ROS packages will be used. The gmapping packages provide laser-based SLAM (Simultaneous Localization and Mapping) and as a result the output from the stereo camera has to be processed with the correct packages to produce a laserscan which can detect objects in close proximity of the robot.

2 Modelling of ARGOS

The first step in this project is the modelling of the robot and, following that, Champ Setup Assistant will be used to produce the ROS package.

2.1 Champ Setup Assistant

Champ Setup Assistant is a software that, if provided with a correct URDF file, can generate a configuration package containing all the files necessary to make a quadruped robot walk [1]. In Figure 1 the Champ Setup Assistant Software is shown.

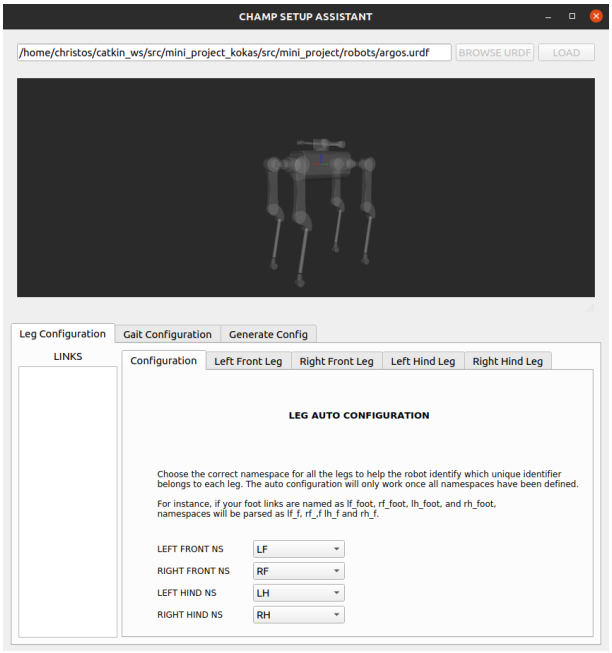


Figure 1: Champ Setup Assistant Software

2.2 Champ Setup Assistant Assumptions

In order to be able to use Champ Setup Assistant to create the ROS package some rules have to be obeyed so that the created package works as intended. The robot has to be modelled with the following assumptions :

- There are no rotation between frames (joint's origin-rpy are all set to zero).
- Hip joints rotate in the X axis.
- Upper Leg joints rotate in the Y axis.
- Lower Leg joints rotate in the Y axis.
- Origins of actuators' meshes are located at the center of rotation.
- All joints at zero position will results the robot's legs to be fully stretched towards the ground. From frontal and sagittal view, all legs should be perpendicular to the ground.

Figure 2 shows the above assumptions.

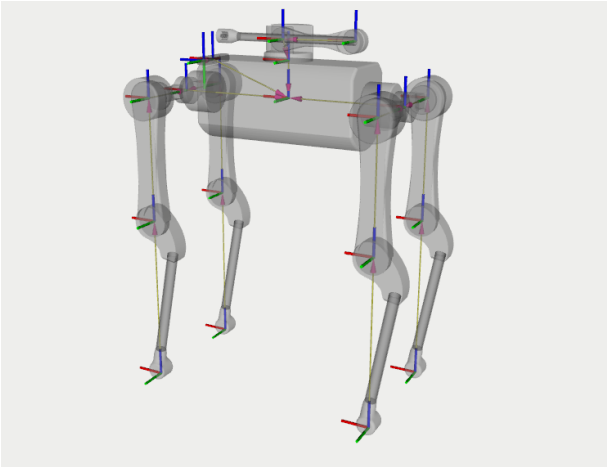


Figure 2: Assumptions for Champ Setup Assistant

2.3 URDF file

To use Champ Setup Assistant a URDF file needs to be created using the aforementioned assumptions. A URDF file is an XML file that describes the robot and contains values of each part such as mass or inertia. Additionally, it describes the position of each link and joint as well as any accessories that the robot has like a stereo camera. Lastly, gazebo references such as traction on parts, or plugins like gazebo ros control can be added.

2.3.1 Mass Properties

For the simulation to be realistic the mass properties have to be as close to the actual robot as possible. With the program Solidworks, the tool Mass Properties can be used to calculate the mass and inertia for the part, relative to the coordinate system selected. The mass properties for the body of the robot relative to the coordinate system used in gazebo are shown in Figure 3.

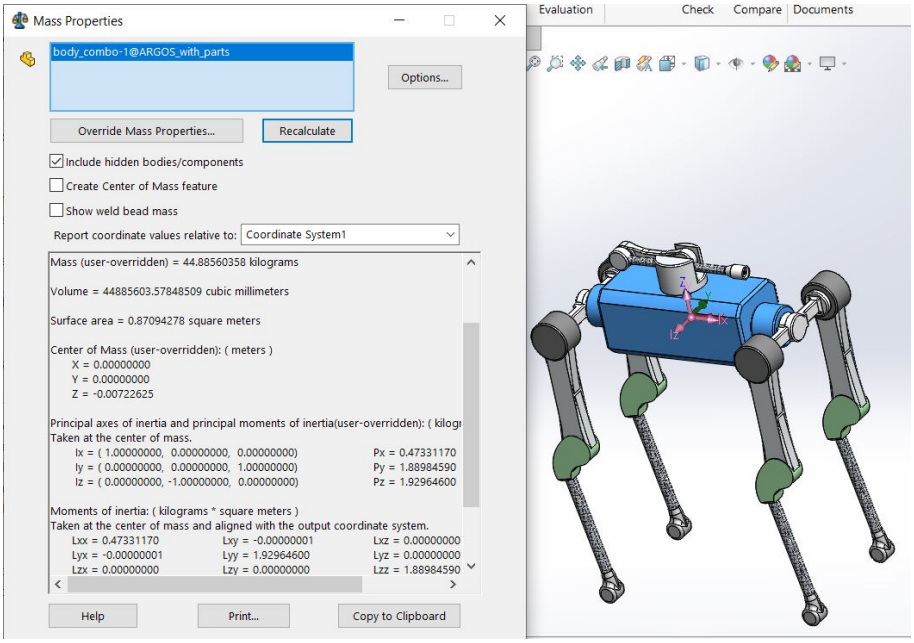


Figure 3: Body Mass Properties

The mass, the center of mass and the inertia matrix can be calculated using the Mass Properties Tool. The coordinate system of each part needs to be at the origin, so a reference coordinate system was inserted. Following that, the correct values can be calculated.

2.3.2 Xacro Language

With the aim of creating a urdf of the robot, the macro language xacro will be used which offers many helpful properties like including other xacro files so that the files are cleaner, and macros which can be used for identical parts. After creating all the xacro files needed for the model of the robot a urdf file can be generated using the command shown in Figure 4.

```
rosrun xacro xacro file.urdf.xacro > file.urdf
```

Figure 4: Command to create URDF file

Where file is the name of the xacro file that creates the model of the robot.

2.3.3 Creating the Model

One of the advantages of the xacro language is the ability to include files. So a file was created that contained all the robot properties like mass, center of mass from the origin, inertia values and the path to the mesh file as shown in Figure 5.

```
<xacro:property name="path_to_base" value =
    "package://mini_project/meshes/body.dae"/>
<xacro:property name="ixx_body" value = "0.47331170488629"/>
<xacro:property name="ixy_body" value = "-0.00000001187694"/>
<xacro:property name="ixz_body" value = "0.0"/>
<xacro:property name="iyy_body" value = "1.92964600184763"/>
<xacro:property name="iyz_body" value = "0.0"/>
<xacro:property name="izz_body" value = "1.88984590052932"/>
<xacro:property name="body_mass" value = "44.88560358"/>
<xacro:property name="body_center_of_mass" value = "0 0
    -0.00722624877"/>
```

Figure 5: Robot Properties Using Xacro

After calculating all the robot properties for each part, links and joints need to be created to start creating the model. Each link is required to have a visual, a collision and an inertial element. The visual element contains the mesh file and the position and orientation of the part relative to the origin. The collision element describes the geometry that is the collision of the specific part. The collision element for performance reasons can be a simplified version of the mesh file. The inertial element contains the values for the moments of inertia and the center of mass. Because the first link of the model is used for localization it cannot have inertia or mass properties so a secondary link is created in the same position to have the mass properties of the body. The two link are displayed is Figure 6.

```
<link name="base_link">
    <visual>
        <geometry>
            <mesh filename="${path_to_base}" scale = "0.001 0.001
                0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} 0 0"/>
    </visual>
    <collision>
        <geometry>
            <mesh filename="${path_to_base}" scale = "0.001 0.001
                0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} 0 0"/>
    </collision>
</link>

<link name="base_inertia">
    <inertial>
        <origin xyz="${body_center_of_mass}" rpy="0 0 0"/>
        <mass value="${body_mass}" />
        <inertia ixx="${ixx_body}" ixy="${ixy_body}"
            ixz="${ixz_body}"
                  iyy="${iyy_body}" iyz="${iyz_body}"
                  izz="${izz_body}" />
    </inertial>
</link>
```

Figure 6: Base Link with Secondary Link

These two links are then joined together with a fixed joint element which can be viewed in Figure 7.

```
<joint name="base_link_to_base_inertia" type="fixed">
    <parent link="base_link"/>
    <child link="base_inertia"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
</joint>
```

Figure 7: Joint of Base Link with Base Inertia

Because ARGOS is a quadruped robot, it has four parts that are identical. That is the reason why macros are really helpful. Using xacro a macro can be created that can be used however many times needed to create the model. In addition, if statements can be used so some variables can change depending on which leg is created as shown in Figure 8, where "RF" is Right Front, "LF" is Left Front, "RH" is Right Hind and "LH" is Left Hind.

```
<xacro:if value="${leg == 'RF'}">
    <xacro:property name="side" value = "-1"/>
    <xacro:property name="position_hip" value = "0.43"/>
</xacro:if>
<xacro:if value="${leg == 'LF'}">
    <xacro:property name="side" value = "1"/>
    <xacro:property name="position_hip" value = "0.43"/>
</xacro:if>
<xacro:if value="${leg == 'RH'}">
    <xacro:property name="side" value = "-1"/>
    <xacro:property name="position_hip" value = "-0.43"/>
</xacro:if>
<xacro:if value="${leg == 'LH'}">
    <xacro:property name="side" value = "1"/>
    <xacro:property name="position_hip" value = "-0.43"/>
</xacro:if>
```

Figure 8: Xacro If Statement

Each leg is composed of 3 parts, the hip, which is connected to the body, the upper leg and the lower leg. These parts with their origins are displayed in Figures 9 - 12.

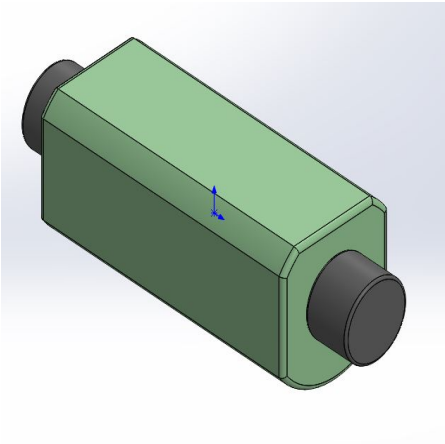


Figure 9: Body of the Robot

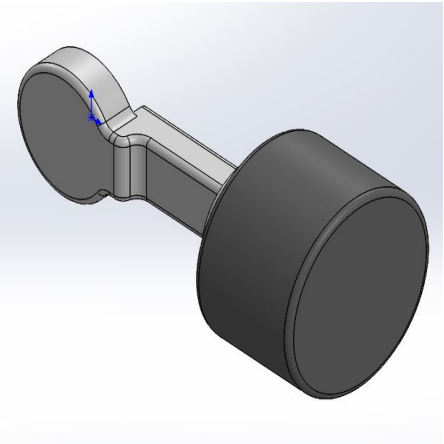


Figure 10: Hip

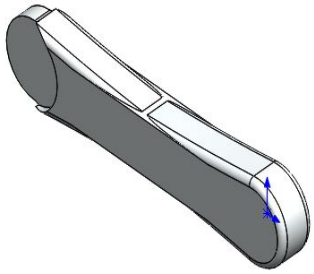


Figure 11: Upper Leg

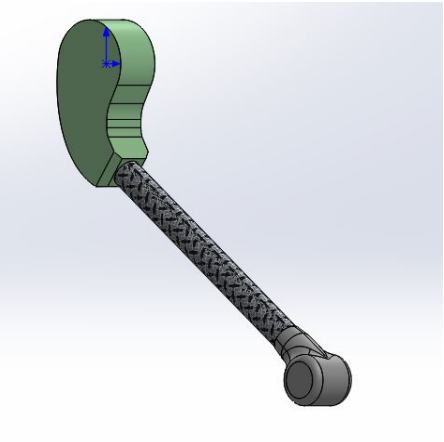


Figure 12: Lower Leg

The measure tool is used to calculate the distance from the origin of the body to the hip as displayed in Figure 13.

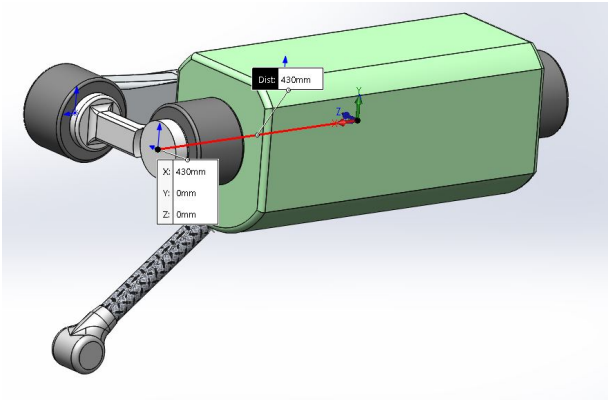


Figure 13: Distance from the Body to Hip

The body with one leg is shown in Figure 14.

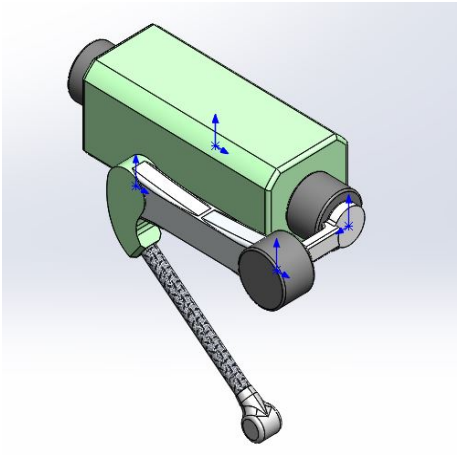


Figure 14: Body with One Leg

After calculating the position of each part and its properties the link can be created as displayed in Figures 15-17. Each link has a different colour to be easily visible.

```
<link name="${leg}_hip">
    <inertial>
        <origin xyz = "${hip_center_of_mass}" rpy="0 0 0"/>
        <mass value="${hip_mass}" />
        <inertia ixx="${ixx_hip}" ixy="${ixy_hip}" ixz="${ixz_hip}"
             iyy="${iyy_hip}" iyz="${iyz_hip}"
             izz="${izz_hip}" />
    </inertial>
    <visual>
        <geometry>
            <mesh filename="${path_to_hip}" scale = "0.001 0.001
                0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} 0.0 ${pi*side/2}" />
    </visual>
    <collision>
        <geometry>
            <mesh filename="${path_to_hip}" scale = "0.001 0.001
                0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} 0.0 ${pi*side/2}"/>
    </collision>
</link>
```

Figure 15: Hip Link

```
<link name="${leg}_upper_leg">
    <inertial>
        <origin xyz = "${thigh_center_of_mass}" rpy="0 0 0"/>
        <mass value="${thigh_mass}" />
        <inertia ixx="${ixx_thigh}" ixy="${side*ixy_thigh}"
            ixz="${ixz_thigh}"
             iyy="${iyy_thigh}" iyz="${side*iyz_thigh}"
             izz="${izz_thigh}" />
    </inertial>
    <visual>
        <geometry>
            <mesh filename="${path_to_upper_leg}" scale = "0.001
                0.001 0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} ${-pi/2} 0"/>
    </visual>
    <collision>
        <geometry>
            <mesh filename="${path_to_upper_leg}" scale = "0.001
                0.001 0.001"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="${pi/2} ${-pi/2} 0"/>
    </collision>
</link>
```

Figure 16: Upper Leg Link

```
<link name="${leg}_lower_leg">
    <inertial>
        <origin xyz = "${lower_leg_center_of_mass}" rpy="0 0 0"/>
        <mass value="${lower_leg_mass}" />
        <inertia ixx="${ixx_lower_leg}" ixy="${ixy_lower_leg}"
            ixz="${ixz_lower_leg}"
             iyy="${iyy_lower_leg}" iyz="${iyz_lower_leg}"
             izz="${izz_lower_leg}" />
    </inertial>
    <visual>
        <geometry>
            <mesh filename="${path_to_lower_leg}" scale = "0.001
                0.001 0.001"/>
        </geometry>
        <origin xyz="0 0.0 0" rpy="${pi/2} ${pi/2-0.182960291} 0"/>
    </visual>
    <collision>
        <geometry>
            <mesh filename="${path_to_lower_leg}" scale = "0.001
                0.001 0.001"/>
        </geometry>
        <origin xyz="0 0.0 0" rpy="${pi/2} ${pi/2-0.182960291} 0"/>
    </collision>
</link>
```

Figure 17: Lower Leg Link

Joints are used to connect the links and have the following elements.

- Parent Link : The First Link
- Child Link : The Link that connects to the parent Link
- Axis : On which axis the joint can turn
- Dynamics : An element specifying physical properties of the joint
- Limit : Which contains the following attributes
  - Effort : An attribute for enforcing the maximum joint effort

- Lower : An attribute specifying the lower joint limit
- Upper : An attribute specifying the upper joint limit
- Velocity : An attribute for enforcing the maximum joint velocity

The joints in the URDF file are shown in Figures 18-20.

```
<joint name="${leg}_hip_joint" type="revolute">
    <parent link="base_link"/>
    <child link="${leg}_hip"/>
    <axis xyz="1.0 0.0 0.0"/>
    <origin xyz="${position_hip} 0.0 0.0"/>
    <limit effort="150" lower="-${pi}" upper="${pi}" velocity="10"
        />
    <dynamics damping="0.0" friction="0.0" />
</joint>
```

Figure 18: Joint Base Link to Leg Hip

```
<joint name="${leg}_upper_leg_joint" type="revolute">
    <parent link="${leg}_hip"/>
    <child link="${leg}_upper_leg"/>
    <axis xyz="0.0 1.0 0.0"/>
    <origin xyz="0.0 ${side*0.233} 0.0"/>

    <limit effort="150" lower="-${pi}" upper="${pi}" velocity="10"
        />
    <dynamics damping="0.0" friction="0.0" />
</joint>
```

Figure 19: Joint Leg Hip to Upper Leg

```
<joint name="${leg}_lower_leg_joint" type="revolute">
    <parent link="${leg}_upper_leg"/>
    <child link="${leg}_lower_leg"/>
    <axis xyz="0.0 1.0 0.0"/>
    <origin xyz="0.0 0.0 -0.45"/>
    <limit effort="150" lower="-${pi}" upper="${pi}" velocity="10"
        />
    <dynamics damping="0.0" friction="0.0" />
</joint>
```

Figure 20: Joint Upper Leg to Lower Leg

The manipulator is created the same way as the previous parts. In order for Champ Setup Assistant to work correctly a foot link needs to be created at the toe. Because the file of the lower foot contains the toe an empty link is created with a fixed joint connecting it to the lower leg as shown in Figure 21.

```
<link name="${leg}_foot"/>
<joint name="${leg}_foot_joint" type="fixed">
    <parent link="${leg}_lower_leg"/>
    <child link="${leg}_foot"/>
    <origin xyz="0.0 0.0 -0.61"/>
</joint>
```

Figure 21: Foot Link with Joint

For the robot to be able to move, transmissions need to be added on all the revolute joints. The transmission element is an extension to the URDF robot description model that is used to describe the relationship between an actuator and a joint [2]. The Transmission element is displayed in Figure 22.

```
<transmission name="${leg}_hip_joint_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="${leg}_hip_joint">
        <hardwareInterface>hardware_interface/
            EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="${leg}_hip_joint_motor">
        <hardwareInterface>hardware_interface/
            EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

Figure 22: Transmission Element

After creating the macro for the legs and the manipulator in a different file they can be included in the main xacro file and be used to create the parts as in Figure 23.

```
<xacro:include filename="$(find
    mini_project)/robots/legs.urdf.xacro"/>
<xacro:argos_leg leg="LF"/>
<xacro:argos_leg leg="RF"/>
<xacro:argos_leg leg="LH"/>
<xacro:argos_leg leg="RH"/>
<xacro:include filename="$(find
    mini_project)/robots/manipulator.urdf.xacro"/>
<xacro:manipulator base_name="base_link"/>
```

Figure 23: Transmission Element

To make sure the properties chosen for each part are correct gazebo can be used. By spawning the model and choosing the option for inertia and center of mass it is possible to check if the values are realistic. The purple boxes are the inertia of each part and they should cover most of the part as shown in Figure 24. The center of mass for each part is displayed in Figure 25.

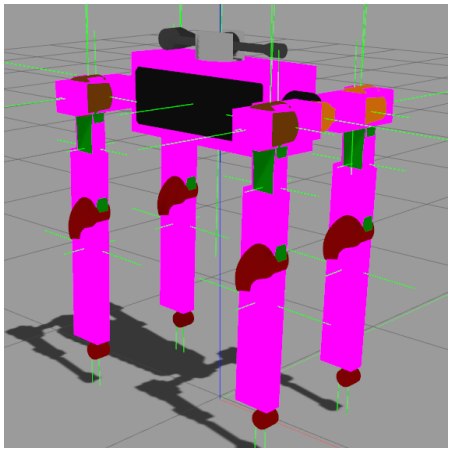


Figure 24: Inertia of Robot

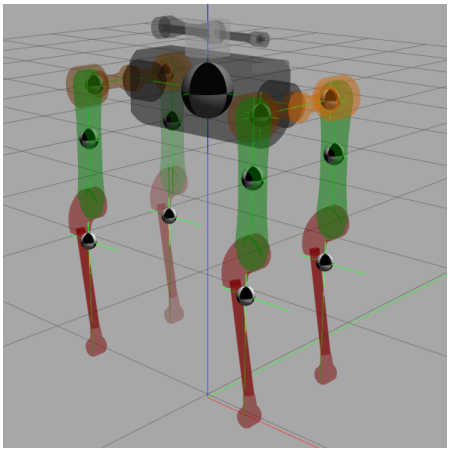


Figure 25: Centers of Mass

In order to be able to detect and avoid obstacles a stereo camera is added to the model. A stereo camera is a type of camera with two or more lenses with a separate image sensor or film frame for each lens. This allows the camera to simulate human binocular vision, and therefore gives it the ability to capture three-dimensional images, a process known as stereo photography [3]. For the model the camera chosen was the ZED2 with the video and depth properties that are shown in Figure 26 and Figure 27 respectively.

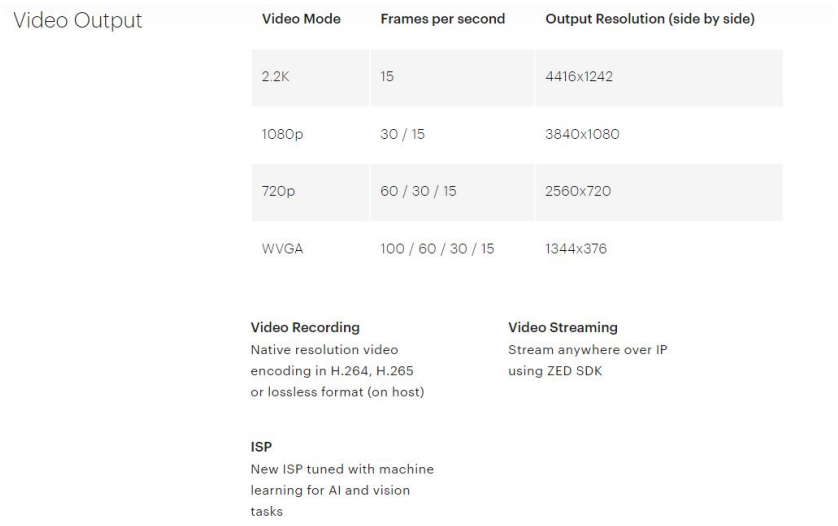


Figure 26: ZED2 Video Properties

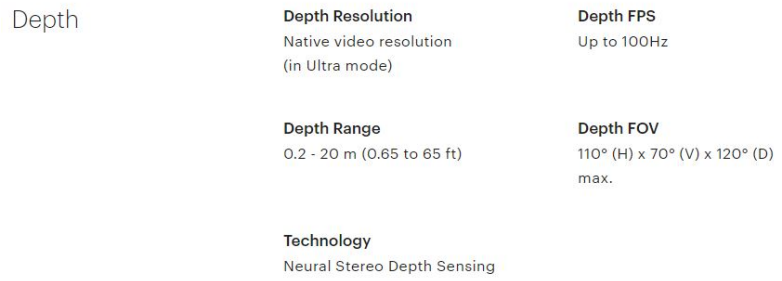


Figure 27: ZED2 Depth Properties

To add the camera on the model, the center of the camera and the left camera frame is created as links and are joined with the body using joints as viewed in Figure 28.

```
<link name="camera_center">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://mini_project/
            meshes/${model}.stl" />
        </geometry>
        <material name="${model}_mat" />
    </visual>
</link>
<link name="left_camera_frame" />
<link name="left_camera_optical_frame"/>
<joint name="camera_center_joint" type="fixed">
    <parent link="$(arg base_frame)"/>
    <child link="camera_center"/>
    <origin xyz="$(arg cam_pos_x) $(arg cam_pos_y) $(arg
        cam_pos_z)" rpy="$(arg cam_roll) $(arg cam_pitch) $(arg
        cam_yaw)" />
</joint>
<joint name="left_camera_joint" type="fixed">
    <parent link="camera_center"/>
    <child link="left_camera_frame"/>
    <origin xyz="0 ${baseline/2} 0" rpy="0 0 0" />
</joint>
<joint name="left_camera_optical_joint" type="fixed">
    <origin xyz="0 0 0" rpy="-${M_PI/2} 0.0 -${M_PI/2}"/>
    <parent link="left_camera_frame"/>
    <child link="left_camera_optical_frame"/>
```

Figure 28: Camera Links and Joints

In Figure 29 the ROS TF frame tree is shown. The TF frame tree shown the parents and children of each link following a tree structure.

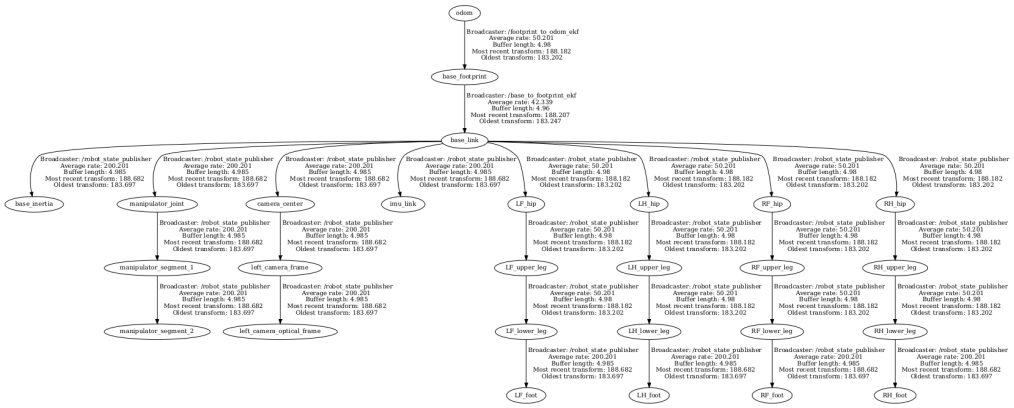


Figure 29: ROS TF Frame Tree

To add the correct properties of the camera as well as the control of the robot gazebo references and plugins will be used.

2.3.4 Gazebo References and Plugins

Because the simulation will be done in Gazebo, there are some useful references and plugins that gazebo offers. Firstly, to avoid sliding on the ground, the toes of the robot need to have traction. This is achieved with a gazebo reference which adds traction. This Gazebo reference has the following elements.

- kp,kd : Contact stiffness kp and damping kd for rigid body contacts as defined by the Open Dynamics Engine (ODE) [4]
- mu1,mu2 : Friction coefficients μ for the principal contact directions along the contact surface as defined by ODE
- minDepth : minimum allowable depth before contact correction impulse is applied
- maxContacts : Maximum number of contacts allowed between two entities

The values choosen are shown in Figure 30.

```
<gazebo reference="${leg}_lower_leg">
    <kp>10000000.0</kp>
    <kd>1.0</kd>
    <mu1>0.9</mu1>
    <mu2>0.9</mu2>
    <minDepth>0.005</minDepth>
    <maxContacts>1</maxContacts>
    <material>Gazebo/Red</material>
</gazebo>
```

Figure 30: Traction of Lower Leg

Gazebo references can also create sensors, like the multicamera (stereo camera, Figure 31). In these references many properties can be entered like height, width or field of view (fov).

```
<gazebo reference="camera_center">
    <sensor type="multicamera" name="stereo_camera">
```

Figure 31: multicamera sensor

The Left and Right camera properties are shown in Figures 32 and 33 (The Right camera has the same properties except the pose which is shown in 33).

```
<camera name="left">
    <pose>0 0.06 0 0 0 0</pose>
    <horizontal_fov>1.91986218</horizontal_fov>
    <vertical_fov>1.22173048</vertical_fov>
    <diagonal_fov>2.0943951</diagonal_fov>
    <image>
    <width>1920</width>
    <height>1080</height>
    <format>B8G8R8</format>
    </image>
    <clip>
    <near>0.2</near>
    <far>20</far>
    </clip>
    <noise>
    <type>gaussian</type>
    <mean>0.0</mean>
    <stddev>0.007</stddev>
    </noise>
</camera>
```

Figure 32: Left Camera

```
    <pose>0 -0.06 0 0 0 0</pose>
```

Figure 33: Right Camera Pose

To control the stereo camera a plugin is needed that is shown in Figure 34.

```
<plugin name="stereo_camera_controller"
   filename="libgazebo_ros_multicamera.so">
   <alwaysOn>true</alwaysOn>
   <updateRate>0.0</updateRate>
   <cameraName>camera</cameraName>
   <imageTopicName>image_raw</imageTopicName>
   <cameraInfoTopicName>camera_info</cameraInfoTopicName>
   <frameName>camera_center</frameName>
   <!--<rightFrameName>right_camera_optical_frame</
      rightFrameName>-->
   <hackBaseline>0.07</hackBaseline>
   <distortionK1>0.0</distortionK1>
   <distortionK2>0.0</distortionK2>
   <distortionK3>0.0</distortionK3>
   <distortionT1>0.0</distortionT1>
   <distortionT2>0.0</distortionT2>
</plugin>
```

Figure 34: Plugin for Stereo Camera Control

Lastly, to control the movement and the stability of the robot the gazebo ros control plugin is used that is displayed in Figure 35.

```
<gazebo>
   <plugin filename="libgazebo_ros_control.so"
      name="gazebo_ros_control">
      <legacyModeNS>true</legacyModeNS>
   </plugin>
</gazebo>
```

Figure 35: ROS Control Plugin

With the completed urdf Champ Setup Assistant can be used to produce the ROS package argos_config which contains all files needed to launch gazebo and navigate using gmapping packages.

2.4 ROS control

To be able to use ros_control [5] with the robot the transmission element is needed to link actuators to joints. Following that, a yaml file needs to be configured which will publish joint states and have pid values for all joints with actuators. To control the robot the yaml file needs to be loaded to the parameter server, load all the controllers for each actuator and convert joint states to TF transforms.

3 Stereo Camera to Laserscan

Gmapping packages require laserscans as input to produce SLAM (Simultaneous Localization and Mapping) [6]. ARGOS is equipped with a stereo camera which has many outputs.

3.1 Stereo Camera Topics

The stereo camera publishes its outputs of the left camera in the topics shown in Figure 36 (the outputs of the right camera are published in topics with namespace /camera/right).

```
/camera/left/image_raw
/camera/left/image_raw/compressed
/camera/left/image_raw/compressed/parameter_descriptions
/camera/left/image_raw/compressed/parameter_updates
/camera/left/image_raw/compressedDepth
/camera/left/image_raw/compressedDepth/parameter_descriptions
/camera/left/image_raw/compressedDepth/parameter_updates
/camera/left/image_raw/theora
/camera/left/image_raw/theora/parameter_descriptions
/camera/left/image_raw/theora/parameter_updates
```

Figure 36: Stereo Camera Outputs

Each topic with its message type is shown in Table 1.

Topic	Message Type
image_raw	sensor_msgs/Image
compressed	sensor_msgs/CompressedImage
compressed/parameter_descriptions	dynamic_reconfigure/ConfigDescription
compressed/parameter_updates	dynamic_reconfigure/Config
compressedDepth	sensor_msgs/CompressedImage
compressedDepth/parameter_descriptions	dynamic_reconfigure/ConfigDescription
compressedDepth/parameter_updates	dynamic_reconfigure/Config
theora	theora_image_transport/Packet
theora/parameter_descriptions	dynamic_reconfigure/ConfigDescription
theora/parameter_updates	dynamic_reconfigure/Config

Table 1: Message Type of Each Topic

3.2 Packages Used to Produce Laserscan

The packages needed to produce a laserscan from the published topics are :

- stereo_image_proc
- disparity_image_proc
- depthimage_to_laserscan

3.2.1 Stereo_Image_Proc

- stereo_image_proc [7] : performs the duties of image processing for both cameras, undistorting and colorizing the raw images and will also compute disparity images.

A disparity image is an image that shows the distance between two corresponding points in the left and right image of a stereo camera as shown in Figure 37.

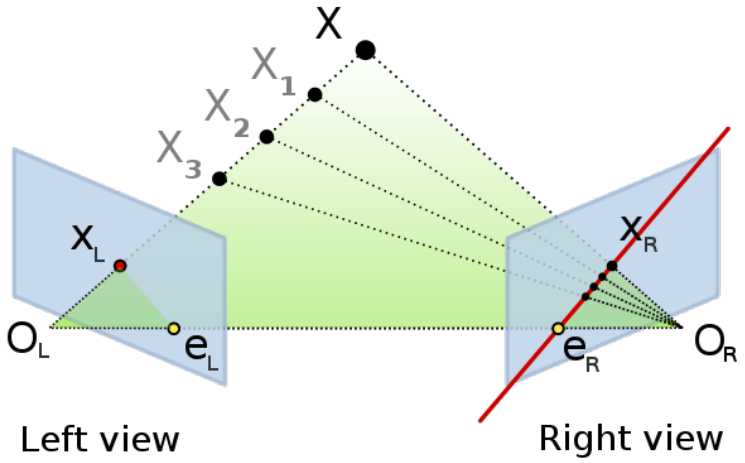


Figure 37: Disparity

The produced depth image from the stereo_image_proc package is displayed in Figure 38.

Figure 38: Produced Depth Image

3.2.2 Disparity_image_proc

After producing the Disparity image the disparity_image_proc package is used.

- disparity_image_proc [8]: Package that can convert ROS disparity images to depth images. The depth image produced has values that represent the distance of the object from the stereo camera.

3.2.3 Depthimage_to_Laserscan

Lastly, the depthimage_to_laserscan package is used to convert the depth image to a laserscan message.

- depthimage_to_laserscan [9]: takes a depth image and generates a 2D laser scan based on the provided parameters.

The ROS Graph of the communication of these packages is shown in Figure 39.

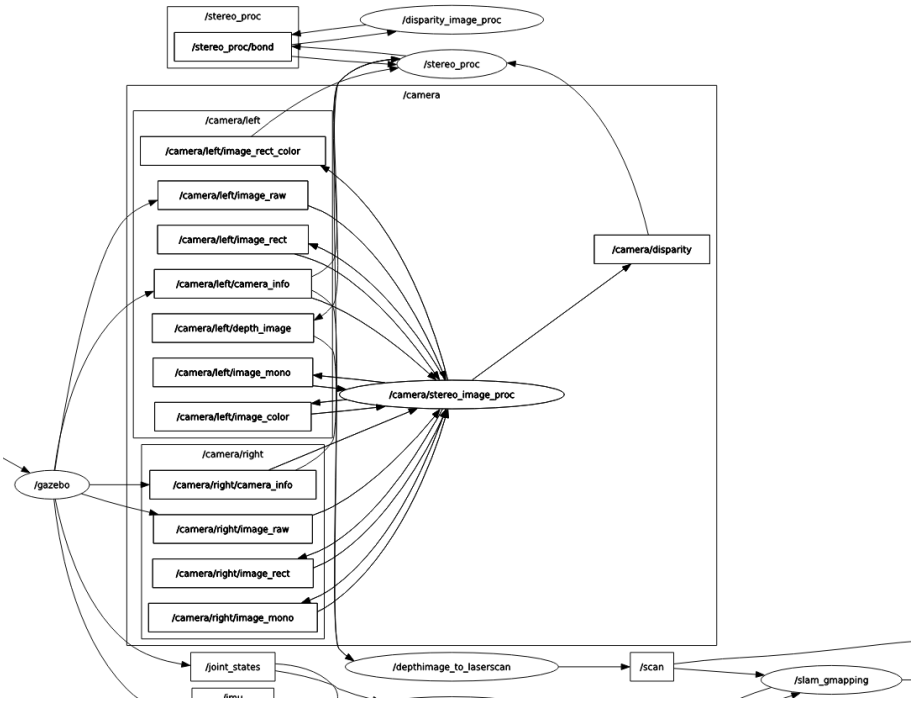


Figure 39: ROS Graph of the Packages

3.3 Launch Files

To use the stereo_image_proc package the namespace of the cameras need to be given on the launch file as shown in Figure 40.

```
<node ns="camera" pkg="stereo_image_proc" type="stereo_image_proc"
   name="stereo_image_proc"/>
```

Figure 40: Stereo Image Proc Launch

Afterwards, the disparity image is converted to a depth image using the disparity_image_proc package. To use this package the topics of the launch file need to be remapped to agree with the topics published from the stereo_image_proc package as displayed in Figure 41.

```
<node name="disparity_image_proc" pkg="nodelet" type="nodelet"
    args="load disparity_image_proc/depth_image stereo_proc"
    output="screen">
  <remap from="/right/camera_info" to="/camera/right/camera_info"/>
  <remap from="/left/camera_info" to="/camera/left/camera_info"/>
  <remap from="/left/image_rect_color"
      to="/camera/left/image_rect_color"/>
  <remap from="/disparity" to="/camera/disparity"/>
  <remap from="/depth_image" to="/camera/left/depth_image"/>
</node>
```

Figure 41: Disparity Image Proc Launch

For the depthimage_to_laserscan package to produce a better laserscan the scan height was changed in the launch file in Figure 42.

```
<remap from="image"     to="/camera/left/depth_image"/>
<param name="scan_height"  type="int"   value="1080"/>
```

Figure 42: Scan Height in DepthImage to Laserscan Launch file

Following the changes, a new launch file is created. Firstly, values are chosen for some arguments as shown in Figure 43.

```
<arg name="robot_name"      default="/"/>
<arg name="rviz"            default="false"/>
<arg name="lite"            default="false" />
<arg name="ros_control_file" default="$(find
    argos_config)/config/ros_control/ros_control.yaml" />
<arg name="gazebo_world"    default="$(find
    argos_config)/worlds/outdoor_new.world" />
<arg name="gui"             default="true"/>
<arg name="world_init_x"    default="0.0" />
<arg name="world_init_y"    default="0.0" />
<arg name="world_init_z"    default="1.08" />
<arg name="world_init_heading" default="0.0" />
<param name="use_sim_time" value="true" />
```

Figure 43: Arguments in Launch File

Following that, the bringup.launch file and the gazebo.launch file are launched to spawn the model, start the controllers and start gazebo as shown in Figure 44.

```
<include file="$(find argos_config)/launch/bringup.launch">
    <arg name="robot_name"          value="$(arg robot_name)"/>
    <arg name="gazebo"              value="true"/>
    <arg name="lite"                value="$(arg lite)"/>
    <arg name="rviz"                value="$(arg rviz)"/>
    <arg name="joint_controller_topic"
        value="joint_group_position_controller/command"/>
    <arg name="hardware_connected"  value="false"/>
    <arg name="publish_foot_contacts" value="false"/>
    <arg name="close_loop_odom"     value="true"/>
</include>
<include file="$(find champ_gazebo)/launch/gazebo.launch">
    <arg name="robot_name"      value="$(arg robot_name)"/>
    <arg name="lite"            value="$(arg lite)"/>
    <arg name="ros_control_file" value="$(arg ros_control_file)"/>
    <arg name="gazebo_world"    value="$(arg gazebo_world)"/>
    <arg name="world_init_x"    value="$(arg world_init_x)" />
    <arg name="world_init_y"    value="$(arg world_init_y)" />
    <arg name="world_init_z"    value="$(arg world_init_z)" />
    <arg name="world_init_heading" value="$(arg
        world_init_heading)" />
    <arg name="gui"             value="$(arg gui)" />
</include>
```

Figure 44: Gazebo and Controllers Launch

Lastly, the packages for the image processing are launched to produce the laserscan for the gmapping packages in Figure 45.

```
<node ns="camera" pkg="stereo_image_proc" type="stereo_image_proc"
    name="stereo_image_proc"/>
<include file="$(find
    argos_config)/launch/disparity_to_depth.launch" />
<include file="$(find argos_config)/launch/depth_to_laser.launch"
    />
```

Figure 45: Image Processing Packages

The resulting laserscan can be viewed on rviz as shown in Figure 46.

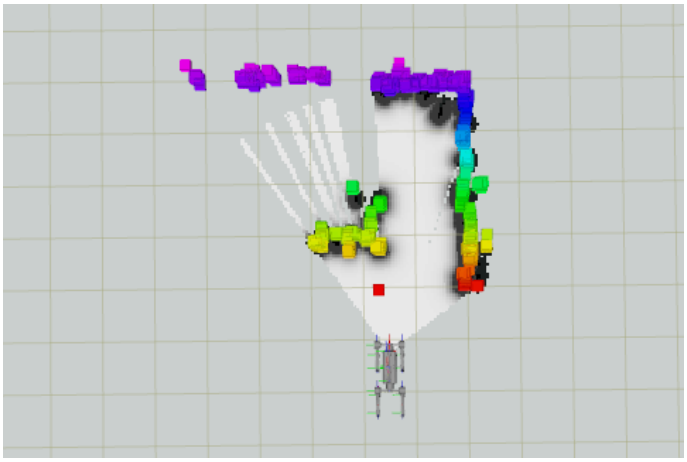


Figure 46: Produced LaserScan

4 Challenges of the Project

One of the challenges faced during the project was choosing the parameters for the controllers as well as the parameters for the movement of the robot. After choosing a value for the P gain so the robot was stable, the D gain was increased in order to reduce oscillation of the robot and the I gain was chosen to remove any errors that could be formed. The values chosen are displayed in Figure 47.

```
gains:
  LF_hip_joint : {p: 800, d: 90.0, i: 300}
  LF_upper_leg_joint : {p: 800, d: 90.0, i: 300}
  LF_lower_leg_joint : {p: 800, d: 90.0, i: 300}
  RF_hip_joint : {p: 800, d: 90.0, i: 300}
  RF_upper_leg_joint : {p: 800, d: 90.0, i: 300}
  RF_lower_leg_joint : {p: 800, d: 90.0, i: 300}
  LH_hip_joint : {p: 800, d: 90.0, i: 300}
  LH_upper_leg_joint : {p: 800, d: 90.0, i: 300}
  LH_lower_leg_joint : {p: 800, d: 90.0, i: 300}
  RH_hip_joint : {p: 800, d: 90.0, i: 300}
  RH_upper_leg_joint : {p: 800, d: 90.0, i: 300}
  RH_lower_leg_joint : {p: 800, d: 90.0, i: 300}
```

Figure 47: PID Gains

There are also some parameters that describe the way the robot moves. The most notable are :

- Nominal Height : The height of the robot standing still
- Leg Swing Height : The height at which the leg reaches during the movement
- Leg Stance Height : The difference between the nominal height and the height at which the robot drops to while moving
- CoM X Translation : If the robot's center of mass is not at the center of the body then changing this parameter moves the coordinate system on the x axis to counter the difference

The parameters were chosen to fit argos and are shown in Figure 48.

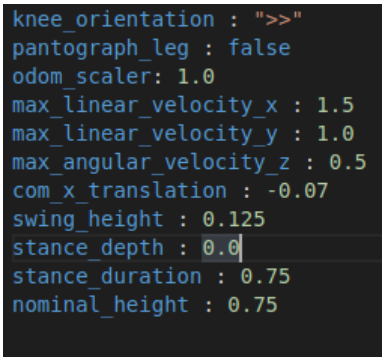


Figure 48: Gait Parameters

Another challenge was finding the correct parameters to avoid the robot moving slowly and additionally reach the goal that was assigned. By changing the values of the file **base_local_planner_holonomic_params.yaml** the above can be achieved. The value **min_vel_trans** is the minimum movement speed which was set too low by default. Furthermore, the goal tolerance had to be increased due to the size of the robot. In Figure 49 these parameters are displayed.

```
min_vel_trans: 0.1

xy_goal_tolerance: 0.5
yaw_goal_tolerance: 0.5
```

Figure 49: Movement of Argos Parameters

5 Results

In this section the results are shown of the robot standing still, and moving avoiding obstacles detected by the stereo camera. Firstly, the simulation begins with the command in Figure 50.

```
rolaunch argos_config gazebo.launch
```

Figure 50: Gazebo Launch

This launch files starts the gazebo world and spawn the robot from 1.08 meters. It additionally starts the packages needed for the conversion of the stereo camera messages to the laserscan which can be verified with the command in Figure 51.

```
rostopic list | grep scan
```

Figure 51: Command for Scan Topic

In Figure 52 the robot is at its starting position with the command showing the scan topic.

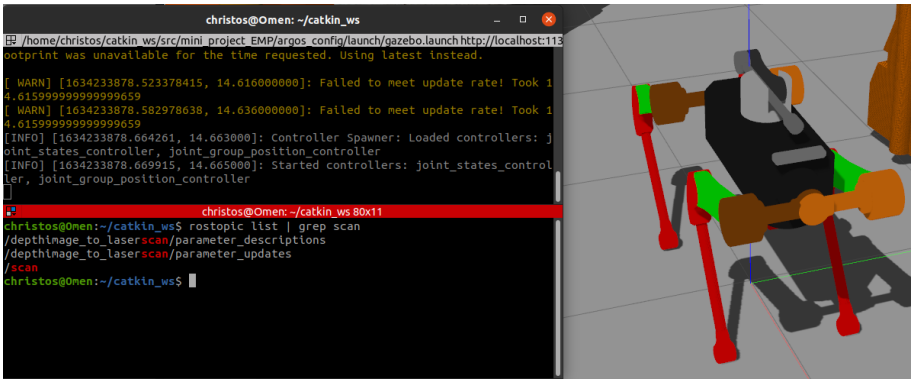


Figure 52: Starting Position of the Robot

To start the obstacle avoidance using gmapping packages the command in Figure 53.

```
roslaunch argos_config slam.launch
```

Figure 53: Command for Obstacle Avoidance with Gmapping Packages

This command starts the gmapping packages needed, as well as rviz to be able to move the robot. With the command 2D Nav Goal the robot can be instructed to move to a position in the map with a certain orientation. The goal that was set for the robot is shown in Figure 54

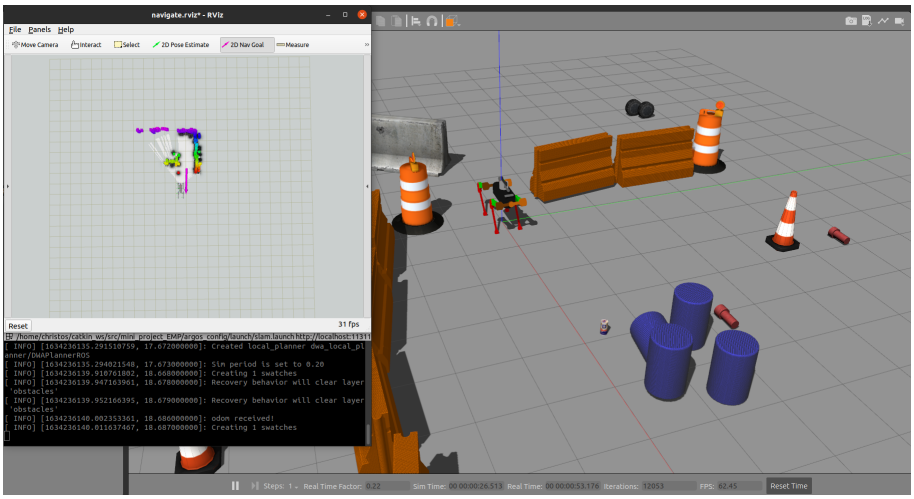


Figure 54: Robot Goal and Orientation Position

The robot after given the instruction creates a path to follow and begins the movement towards the goal as shown in Figure 55 (left to right, top to bottom).

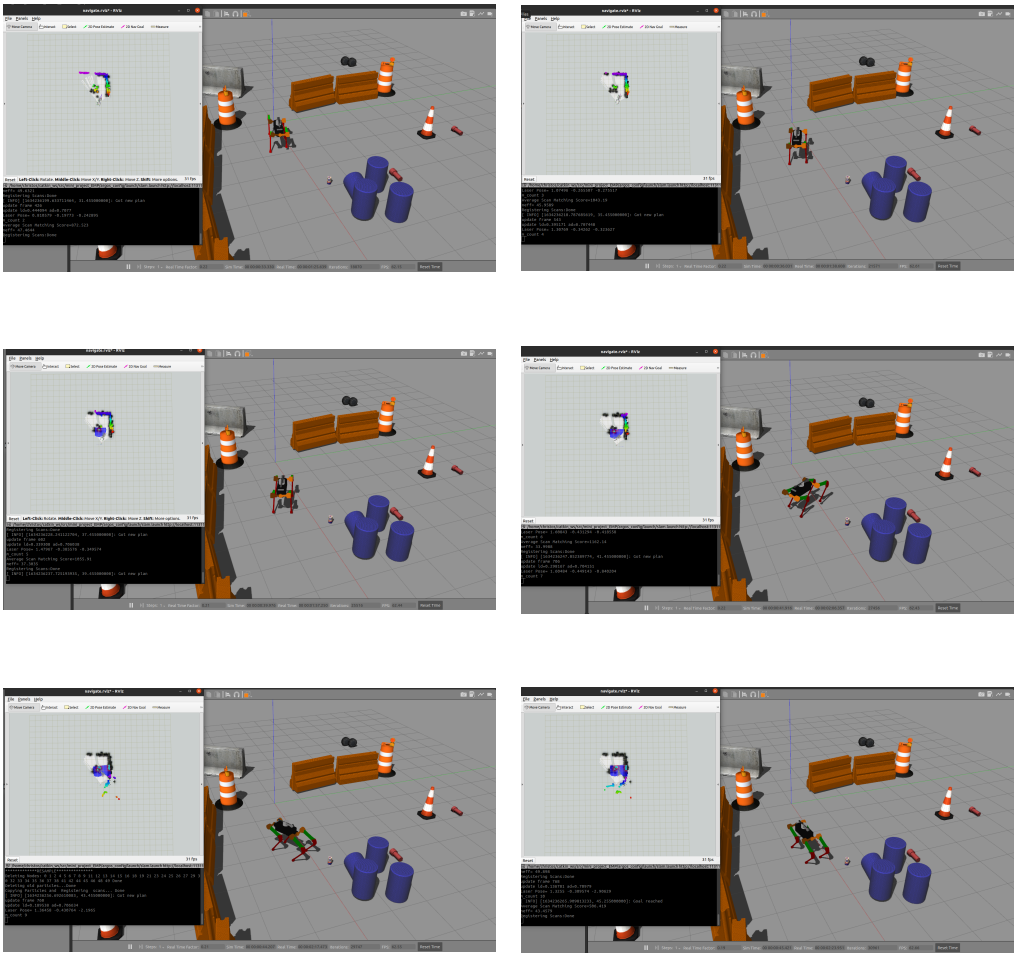


Figure 55: Movement of the Robot Towards Goal

The terminal is displayed in Figure 56 as the robot reached the goal.

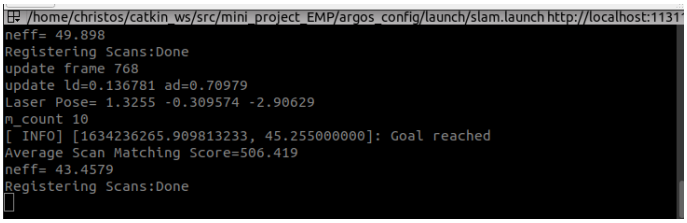


Figure 56: Goal Reached

The path planning also avoids obstacles detected by the laserscan as shown in Figure 57.

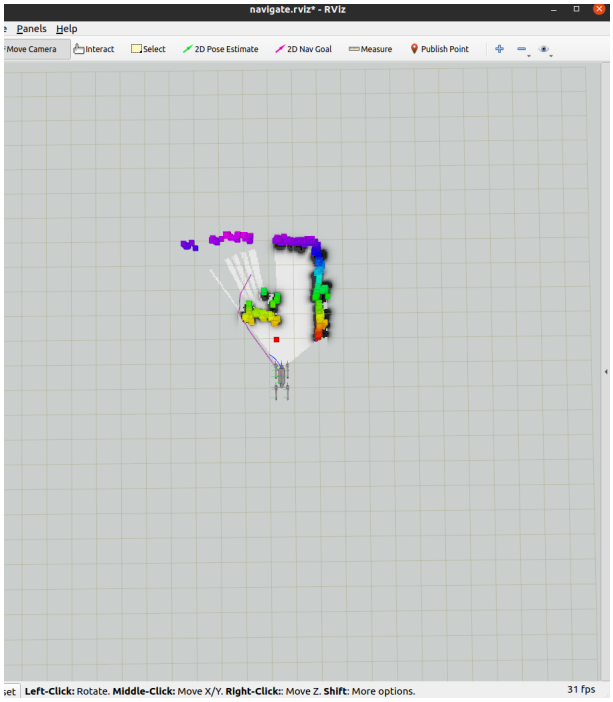


Figure 57: Obstacle Avoidance

6 Improvements that could be Made

Although the robot moves and avoids obstacles as intended there are always improvements that could be made to make it better. One of these improvements is to find better values for the parameters to make the movement more accurate. Additionally, it was observed that small objects are not detected perfectly, mostly because, by using so many packages to convert the stereo image to laserscan it is impossible for errors not to accumulate. So some other combination of packages could be chosen so that all the objects can be detected.

List of Figures

1 Champ Setup Assistant Software 4
2 Assumptions for Champ Setup Assistant 5
3 Body Mass Properties 6
4 Command to create URDF file 6
5 Robot Properties Using Xacro 7
6 Base Link with Secondary Link 8
7 Joint of Base Link with Base Inertia 9
8 Xacro If Statement 9
9 Body of the Robot 10
10 Hip 10
11 Upper Leg 10
12 Lower Leg 10
13 Distance from the Body to Hip 11
14 Body with One Leg 11
15 Hip Link 12
16 Upper Leg Link 13
17 Lower Leg Link 14
18 Joint Base Link to Leg Hip 15
19 Joint Leg Hip to Upper Leg 15
20 Joint Upper Leg to Lower Leg 16
21 Foot Link with Joint 16
22 Transmission Element 17
23 Transmission Element 17
24 Inertia of Robot 18
25 Centers of Mass 18
26 ZED2 Video Properties 19
27 ZED2 Depth Properties 19
28 Camera Links and Joints 20
29 ROS TF Frame Tree 21
30 Traction of Lower Leg 22
31 multicamera sensor 22
32 Left Camera 23
33 Right Camera Pose 23
34 Plugin for Stereo Camera Control 24
35 ROS Control Plugin 24
36 Stereo Camera Outputs 25

37 Disparity 27
38 Produced Depth Image 27
39 ROS Graph of the Packages 28
40 Stereo Image Proc Launch 29
41 Disparity Image Proc Launch 29
42 Scan Height in DepthImage to Laserscan Launch file 29
43 Arguments in Launch File 30
44 Gazebo and Controllers Launch 31
45 Image Processing Packages 32
46 Produced LaserScan 32
47 PID Gains 33
48 Gait Parameters 34
49 Movement of Argos Parameters 34
50 Gazebo Launch 35
51 Command for Scan Topic 35
52 Starting Position of the Robot 35
53 Command for Obstacle Avoidance with Gmapping Packages . 36
54 Robot Goal and Orientation Position 36
55 Movement of the Robot Towards Goal 37
56 Goal Reached 38
57 Obstacle Avoidance 38

References

[1] Champ Setup Assistant, GitHub.

[2] Transmission in URDF, ROS Wiki.

[3] Stereo Camera, Wikipedia.

[4] Open Dynamics Engine, ODE.

[5] Gazebo ROS Control, ROS Wiki.

[6] Gmapping Packages, ROS Wiki.

[7] Stereo Image Proc package, GitHub.

[8] Disparity Image Proc package, GitHub.

[9] Depthimage To Laserscan package, GitHub.