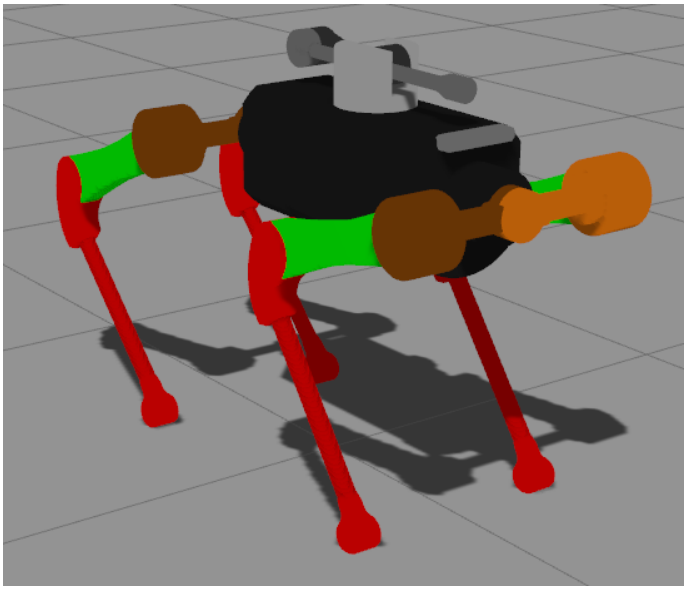
Mini Project Report

Christos Vasileios Kokas

11-10-2021



Contents

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.

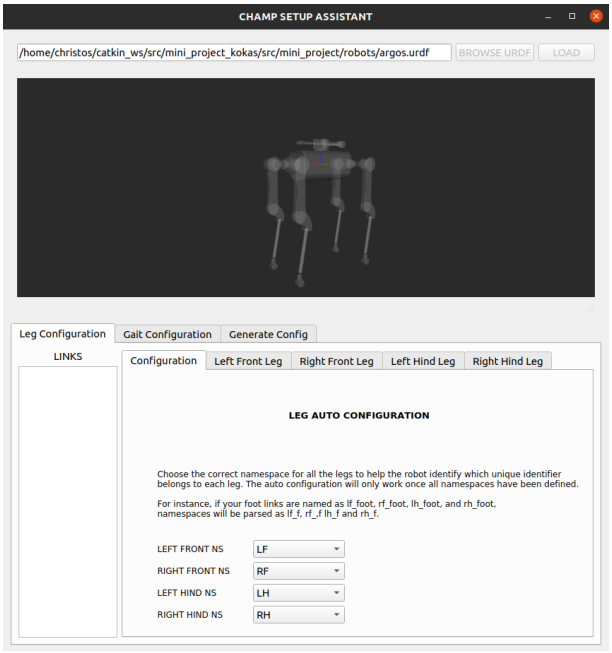


Figure 1: Champ Setup Assistant Software

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.

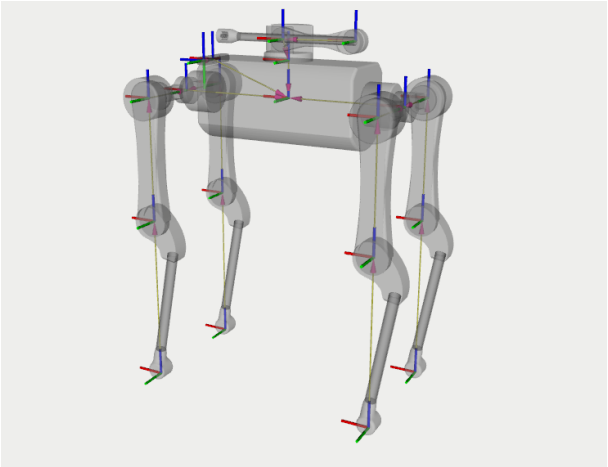


Figure 2: Assumptions for Champ Setup Assistant

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.

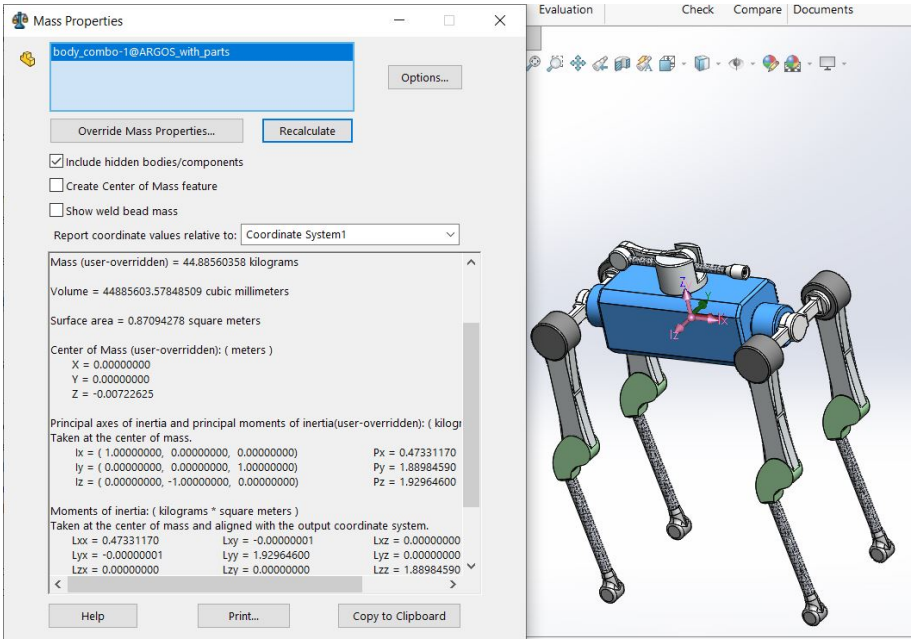


Figure 3: Body Mass Properties

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 :

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

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.

```
<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"/>
```

After calculating all the robot properties for each part, links and joints need to be created to start creating the model. Each link has 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 of mass properties so a secondary link is created in the same position to have the mass properties of the body. These two links are then joined together with a fixed joint element.

```
<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>
<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>
```

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.

```
<xacro:if value="${leg == 'LF'}">
    <xacro:property name="hip_center_of_mass" value =
        "${hip_center_x} ${hip_center_y} ${hip_center_z}"/>
    <xacro:property name="thigh_center_of_mass" value =
        "${-thigh_center_x} ${thigh_center_y} ${thigh_center_z}"/>
    <xacro:property name="lower_leg_center_of_mass" value =
        "${lower_leg_center_x} ${lower_leg_center_y}
        ${lower_leg_center_z}"/>
    <xacro:property name="ixz_hip" value = "${ixz_hip_d}"/>
    <xacro:property name="iyz_hip" value = "${iyz_hip_d}"/>
    <xacro:property name="orientation" value = "1"/>
```

```
    <xacro:property name="position_hip" value = "0.43"/>
</xacro:if>
```

After calculating the position of the hip and its properties a joint element is created to joint the body of the robot with each leg. The joint element is of revolute type because it is the first of three degrees of freedom.

```
<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
            ${pi*orientation/2}" />
    </visual>
    <collision>
        <geometry>
            <mesh filename="${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*orientation/2}"/>
    </collision>
</link>

<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="${position_hip} 0.0 0.0"/>
    <limit effort="500" lower="-${2*pi}" upper="${2*pi}"
        velocity="25" />
```

```
    <dynamics damping="0.0" friction="0.0" />
</joint>
```

All the other parts of the leg, as well as the manipulator, are created the same way as the hip. 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.

```
<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>
```

For the robot to be able to move, transmissions need to be added on all the revolute joints.

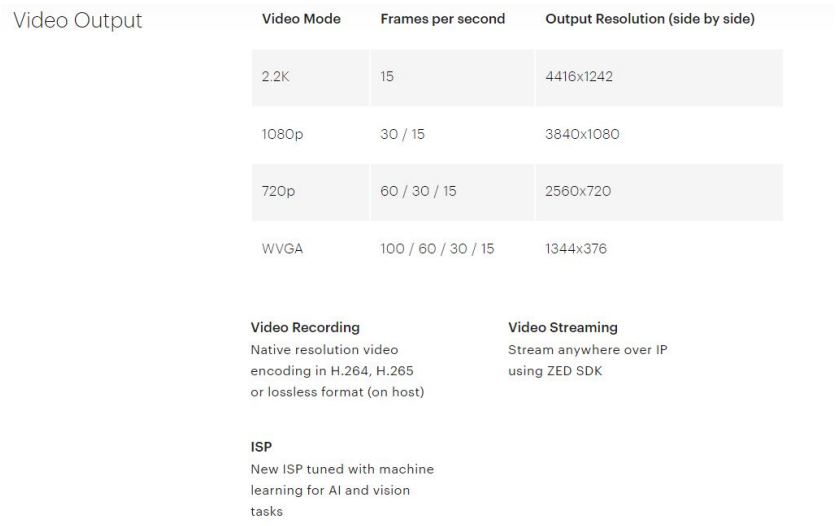
```
<transmission name="${leg}_hip_joint_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="${leg}_hip_joint">
        <hardwareInterface>hardware_interface/
            EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="${leg}_hip_joint_motor">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

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.

```
<xacro:include filename="$(find
    mini_project)/robots/legs.urdf.xacro"/>

<xacro:argus_leg leg="LF"/>
<xacro:argus_leg leg="RF"/>
<xacro:argus_leg leg="LH"/>
```

```
<xacro:argus_leg leg="RH"/>

<xacro:include filename="$(find
    mini_project)/robots/manipulator.urdf.xacro"/>

<xacro:manipulator base_name="base_link"/>
```

In order to be able to detect and avoid obstacles a stereo camera is added to the model. The ZED2 camera was chosen for the model that has the below video and depth properties.



Video Output

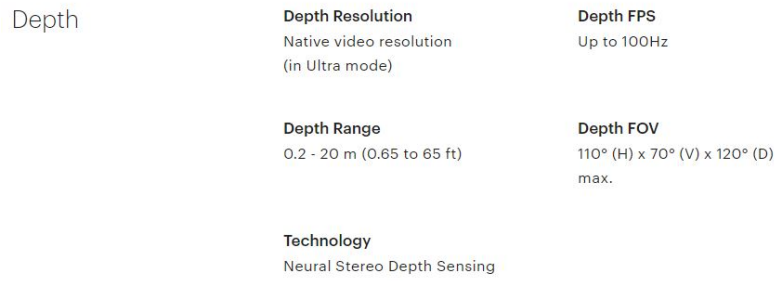
Video Mode	Frames per second	Output Resolution (side by side)
2.2K	15	4416x1242
1080p	30 / 15	3840x1080
720p	60 / 30 / 15	2560x720
WVGA	100 / 60 / 30 / 15	1344x376

Video Recording
Native resolution video encoding in H.264, H.265 or lossless format (on host)

Video Streaming
Stream anywhere over IP using ZED SDK

ISP
New ISP tuned with machine learning for AI and vision tasks

Figure 4: ZED2 Video Properties



Depth

Depth Resolution
Native video resolution (in Ultra mode)

Depth FPS
Up to 100Hz

Depth Range
0.2 - 20 m (0.65 to 65 ft)

Depth FOV
110° (H) x 70° (V) x 120° (D) max.

Technology
Neural Stereo Depth Sensing

Figure 5: 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.

```
<!-- Camera Center -->
<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>

<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>

<!-- Left Camera -->
<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>

<link name="left_camera_frame" />

<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"/>
</joint>

<link name="left_camera_optical_frame"/>
```

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 easily achieved with a gazebo reference which adds traction as is shown below.

```
<gazebo reference="${leg}_lower_leg">
    <kp>10000000.0</kp>
    <kd>1.0</kd>
    <mu1>0.99</mu1>
    <mu2>0.99</mu2>
    <maxVel>1000.0</maxVel>
    <minDepth>0.1</minDepth>
    <material>Gazebo/Red</material>
</gazebo>
```

Gazebo references can also create sensors, like multicamera (stereo camera). 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">
    <update_rate>30.0</update_rate>
    <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>
    <camera name="right">
        <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>
    <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>
    </sensor>
</gazebo>
```

As it is show above, a plugin is also needed to control the stereo camera. Lastly, to control the movement and the stability of the robot the gazebo ros control plugin is used.

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

After creating all the xacro files needed with the below command a urdf file is produced.

```
rosrun xacro xacro argos.urdf.xacro > argos.urdf
```

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 with the robot the transmission element, which was added previously, 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.

```
<rosparam file="$(arg ros_control_file)" command="load"/>
<node name="controller_spawner" pkg="controller_manager"
    type="spawner" respawn="false" output="screen" args="
    joint_states_controller
    joint_group_position_controller
    ">
</node>
```

3 Stereo Camera to Laserscan

To use gmapping packages a laserscan is needed for navigation. ARGOS has a stereo camera which has to be configured with the correct packages so that

in produces a laserscan message. The packages that were used to create the laserscan message are :

- stereo_image_proc
- disparity_image_proc
- depthimage_to_laserscan

Stereo_image_proc is an image processing package that can take the output of the stereo camera on ARGOS and produce a disparity image. To use this package the namespace of the cameras need to be given. Because on the stereo_camera_controller the cameraName that was given is camera, then the namespace (ns) to be used is :

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

Afterwards, the disparity image is converted to a depth image using the disparity_image_proc package. To use this package some changes need to be made on the launch file regarding the correct topics to be assigned.

```
<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>
```

Following the changes, a new launch file is created and is launched from the gazebo.launch file.

```
  <include file="$(find
     argos_config)/launch/disparity_to_depth.launch" />
```

The produced depth image is shown below.

Figure 6: Produced Depth Image

With the depth image produced, using the depthimage_to_laserscan package, a laserscan message can be generated. After changing the topic and the scan height it is also launched from the gazebo.launch file.

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

```
<include file="$(find argos_config)/launch/depth_to_laser.launch"
    />
```

The resulting laserscan can be viewed on rviz as shown below.

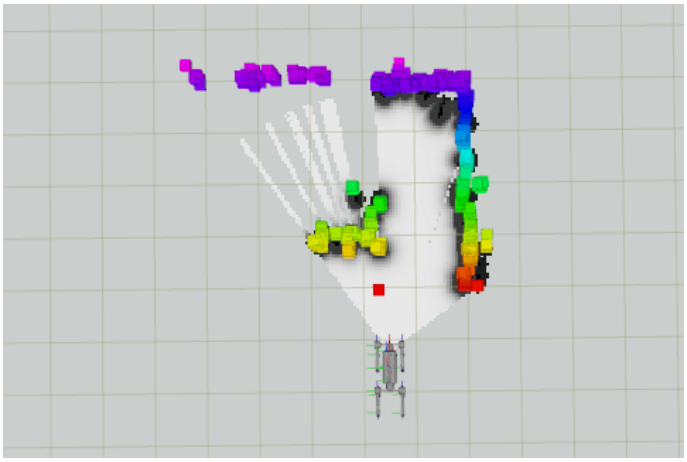


Figure 7: Produced LaserScan

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 many trials and errors the parameters for the pid controllers were chosen as follows.

```
gains:
  LF_hip_joint : {p: 800, d: 100.0, i: 300}
  LF_upper_leg_joint : {p: 800, d: 100.0, i: 300}
  LF_lower_leg_joint : {p: 800, d: 100.0, i: 300}
  RF_hip_joint : {p: 800, d: 100.0, i: 300}
  RF_upper_leg_joint : {p: 800, d: 100.0, i: 300}
  RF_lower_leg_joint : {p: 800, d: 100.0, i: 300}
  LH_hip_joint : {p: 800, d: 100.0, i: 300}
  LH_upper_leg_joint : {p: 800, d: 100.0, i: 300}
  LH_lower_leg_joint : {p: 800, d: 100.0, i: 300}
  RH_hip_joint : {p: 800, d: 100.0, i: 300}
  RH_upper_leg_joint : {p: 800, d: 100.0, i: 300}
  RH_lower_leg_joint : {p: 800, d: 100.0, i: 300}
```

Figure 8: 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 chosen are shown below.

```
knee_orientation : ">>"
pantograph_leg : false
odom_scaler: 1.0
max_linear_velocity_x : 2.5
max_linear_velocity_y : 1.5
max_angular_velocity_z : 1.0
com_x_translation : -0.14
swing_height : 0.1
stance_depth : 0.06
stance_duration : 0.75
nominal_height : 0.7
```

Figure 9: 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 because Champ Setup Assistant is generally used for smaller robots. Furthermore, the goal tolerance had to be increased due to the size of the robot.

```
min_vel_trans: 0.1

xy_goal_tolerance: 0.5
yaw_goal_tolerance: 0.5
```

As mentioned before some assumptions have been made in the urdf so that Champ Setup Assistant works correctly. The assumption that the legs are stretched touching the ground was a challenge because the lower part of the leg follows an arc and is not a straight line. Because it is not straight the rotation of the part had to be calculated using Solidworks.
Lastly, one more challenge was getting the correct inertia parameters for each part. On the urdf files the inertia parameters are relative to the origin of each part, and because Solidworks uses different coordinate system than gazebo, new coordinate systems had to be created to get the correct values.

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 :

```
rolaunch argos_config gazebo.launch
```

This launch files starts the gazebo world and spawn the robot from 1.5 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 :

```
rostopic list | grep scan
```

Below is a figure of the robot at starting position with the command showing the scan topic.

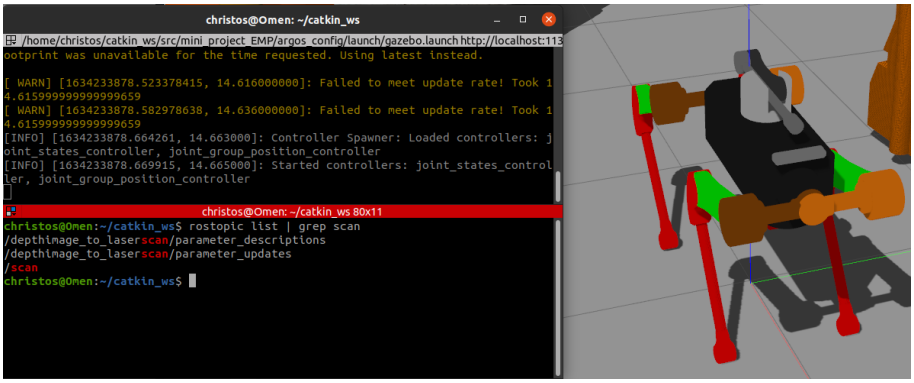


Figure 10: Starting Position of the Robot

To start the obstacle avoidance using gmapping packages the below command is used.

```
roslaunch argos_config slam.launch
```

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. Below is the goal that was set for the robot.

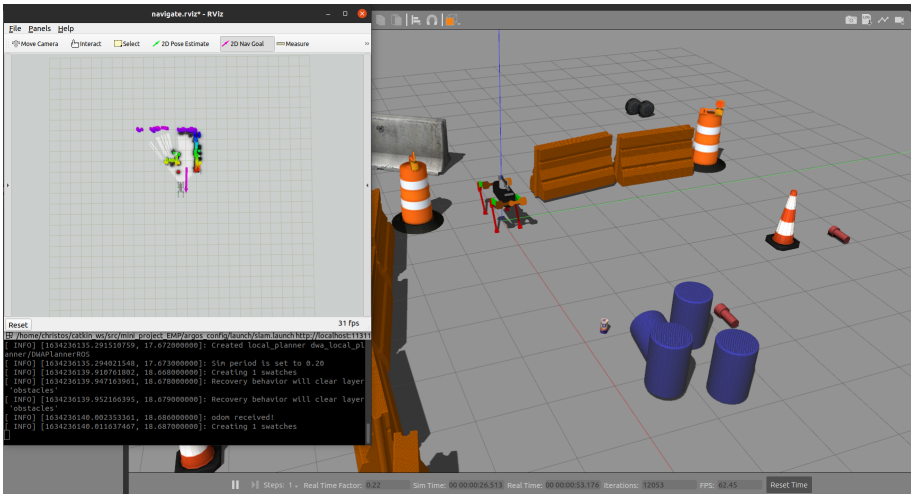


Figure 11: Robot Goal and Orientation Position

The robot after given the instruction creates a path to follow and begins the movement towards the goal.

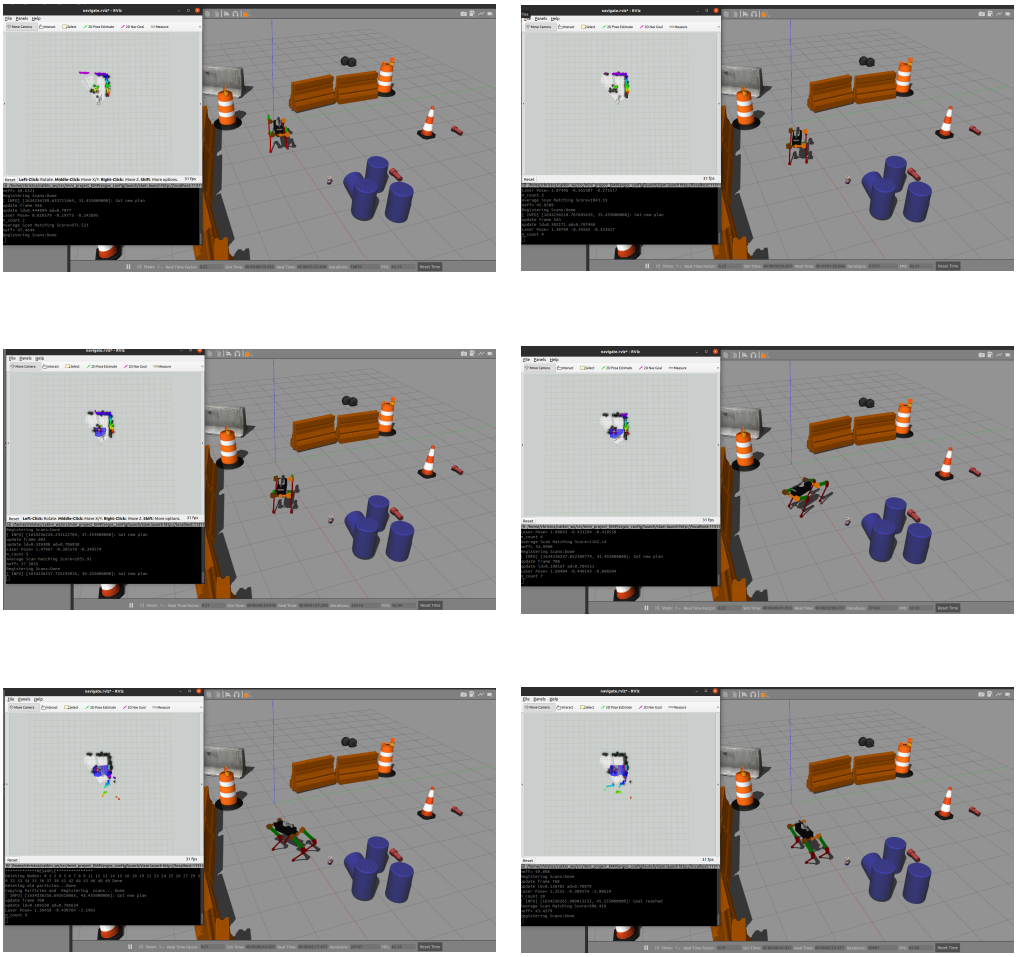


Figure 12: Movement of the Robot Towards Goal

As is shown on the terminal the robot reached the goal.

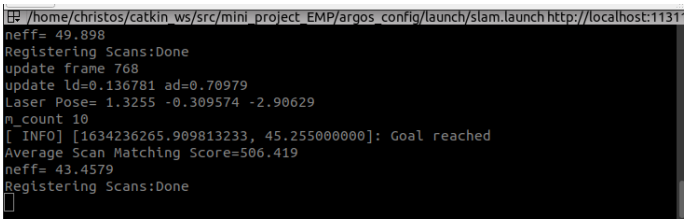


Figure 13: Goal Reached

The path planning also avoids obstacles detected by the laserscan as shown below.

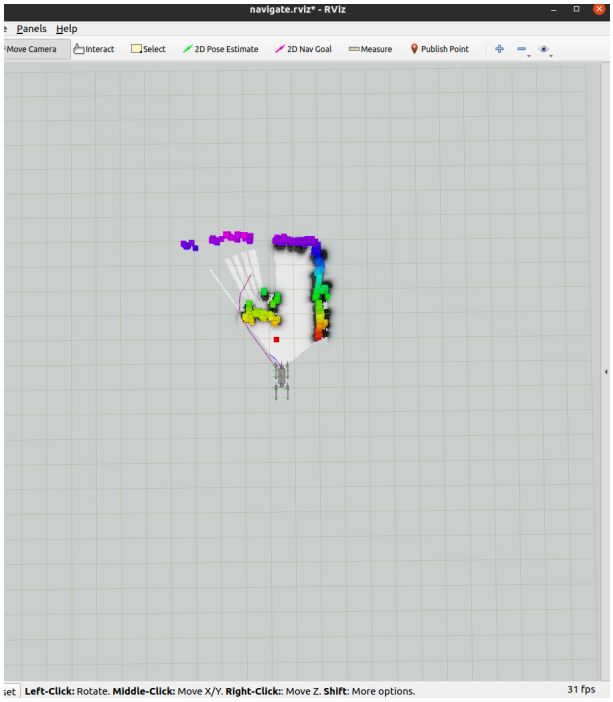


Figure 14: Obstacle Avoidance

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 mentioned above to make to movement better. 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

References

[1] Champ Setup Assistant, Github.