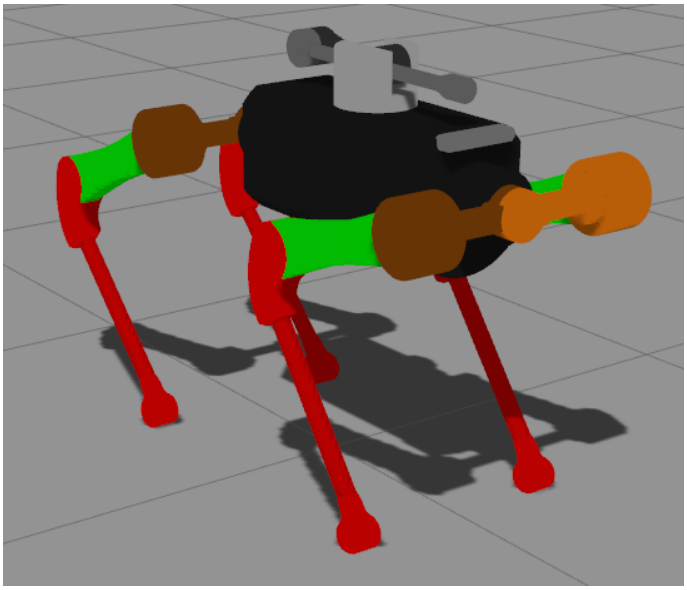
Mini Project Report

Christos Vasileios Kokas

11-10-2021



Contents

1 Abstract

The goal of this project is to build a simulation framework for ARGOS, a quadruped robot with three actuated degrees of freedom on each leg. In addition, with the use of 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 mounted on its body. This project covers in detail the steps taken to model the robot using the XACRO XML macro language and with the use of Champ Setup Assistant create a ROS package for the movement of ARGOS. After creating the ROS package the pid gains and the gait parameters have to be tuned. 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). Since the robot is equipped only with stereo cameras, a laserscan has to be produced with the use of the appropriate packages so that ARGOS can detect objects in close proximity.

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 Xacro Language

Xacro (XML Macros) Xacro is an XML macro language [1]. With xacro, you can construct shorter and more readable XML files by using macros that expand to larger XML expressions. 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 1.

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

Figure 1: Command to generate a URDF file from XACRO

2.2 URDF file

The URDF(Universal Robot Description Format) model is a collection of files that describe a robot's physical description to ROS. To model ARGOS a URDF file will be created. 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 equipment that the robot has e.g.: a stereo camera.

2.2.1 Mass Properties

For a realistic simulation the mass properties have to be as close to a robot of the same dimensions as possible. Using Solidworks, the Mass Properties tool can be used to calculate the mass and inertia for each part, relative to the coordinate system selected. The mass properties for the body of the robot relative to the coordinate system i.e: Mass, Center of Mass, Inertia Matrix, are shown in Figure 2.

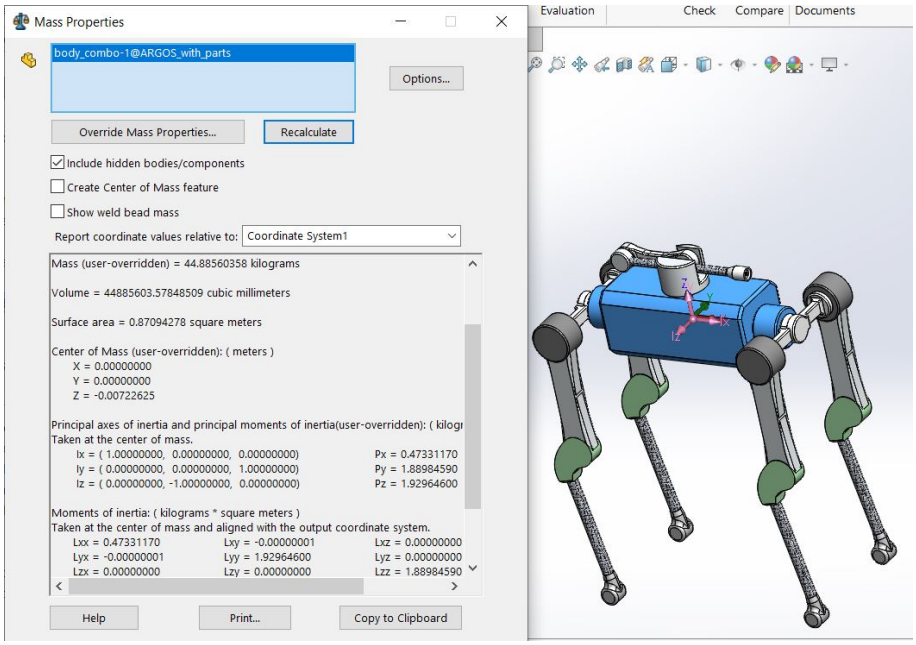


Figure 2: Body Mass Properties

The mass, the center of mass and the inertia matrix can be calculated

using the Mass Properties Tool. The mass properties are calculated relative to a coordinate system. The coordinate system of each part needs to be at the origin of the part, so a reference coordinate system was created.

2.2.2 Creating the Model

One of the advantages of the xacro language is the ability to create separate files ,e.g.: robot properties, legs, manipulator, and include them in one main file. The file containing all the robot properties like mass, center of mass from the origin, inertia values and the path to the mesh file are presented in Figure 3.

```
<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 3: Robot Properties Using Xacro

Links are connected with joints to build the robot model. Each Link has the following elements [4]:

- Inertial (Required for Gazebo Simulation) : The inertial properties of the link.
  - Origin : This is the pose of the inertial reference frame, relative to the link reference frame. The origin of the inertial reference frame needs to be at the center of gravity.
  - Mass : The mass of the link is represented by the value attribute of this element.

  - Inertia : The 3x3 rotational inertia matrix, represented in the inertia frame. Because the rotational inertia matrix is symmetric, only 6 above-diagonal elements of this matrix are specified here, using the attributes ixx, ixy, ixz, iyy, iyz, izz.

- Visual : The visual properties of the link. This element specifies the shape of the object (box, cylinder, etc.) for visualization purposes.
  - Origin : The reference frame of the visual element with respect to the reference frame of the link.
  - Geometry : The shape of the visual object. This can be one of the following:
    * Box
    * Cylinder
    * Sphere
    * Mesh File
  - Material : The material of the visual element.

- Collision : The collision properties of a link. To reduce computation time a simplified mesh can be used.
  - Origin : The reference frame of the collision element, relative to the reference frame of the link.
  - Geometry : The shape of the collision object.

The links are connected with joints and they follow a tree structure by using the kdl_parser [5]. The Kinematics and Dynamics Library (KDL) defines a tree structure to represent the kinematic and dynamic parameters of a robot mechanism. kdl_parser provides tools to construct a KDL tree from an XML robot representation in URDF. KDL does not support a root link with an inertia, so the root link (base_link) of the model cannot have inertia or mass properties. A secondary link (base_inertia) is created in the same position to have the mass properties of the body. The two links are displayed in Figure 4.

```
<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 4: Base Link with Secondary Link

ARGOS is a quadruped robot so it has four identical legs. The XACRO language gives the ability to create macros which are helpful for identical parts. Macros provide the ability to create a part once, and then use that part multiple times. In addition, if statements can be used to change values of variables of the macro depending on which leg is created. The variables changed is the distance from the hip to the body(position_hip), which changes value if the referenced leg is the front or the hind, and the side of the robot(side), which changes value if the the referenced leg is the right or the left. The if statement is shown in Figure 5, 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 5: Xacro If Statement

Each leg has 3 parts, the leg roll segment, which is connected to the body, the upper leg segment and the lower leg segment. These parts with their origins are displayed in Figures 6 - 9.

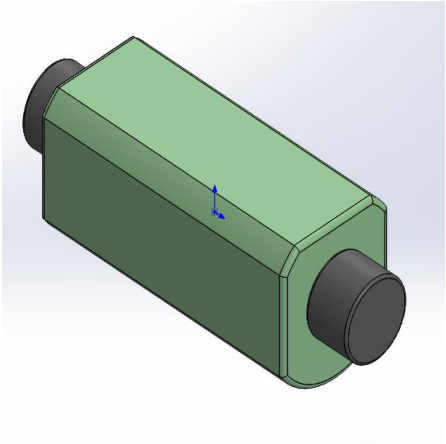


Figure 6: Body of the Robot

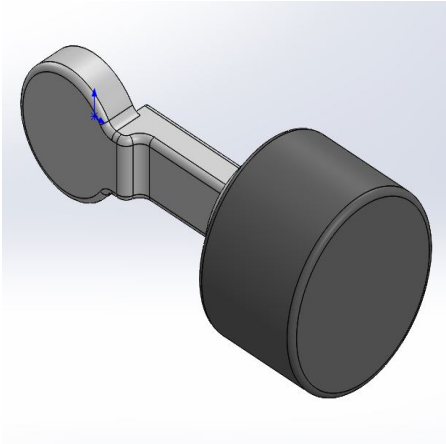


Figure 7: Leg Roll Segment

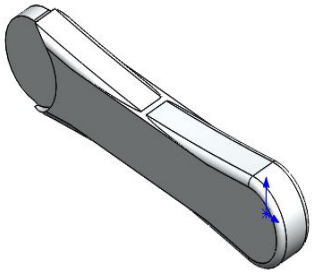


Figure 8: Upper Leg Segment

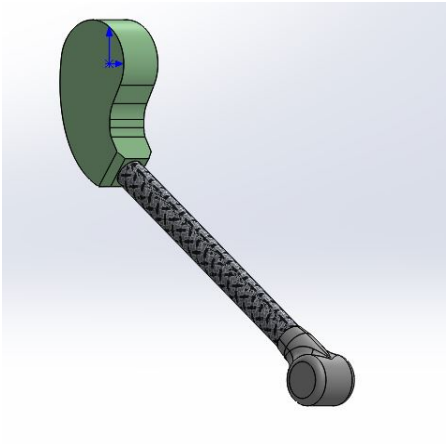


Figure 9: Lower Leg Segment

The measuring tool is used to calculate the distance from the origin of the body to the hip as displayed in Figure 10.

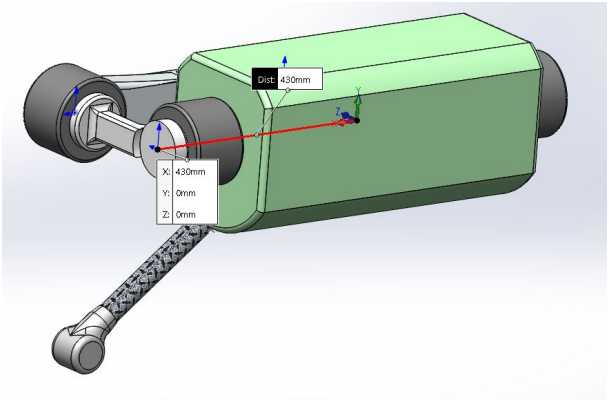


Figure 10: Distance from the Body to Hip

The body with one leg is shown in Figure 11.

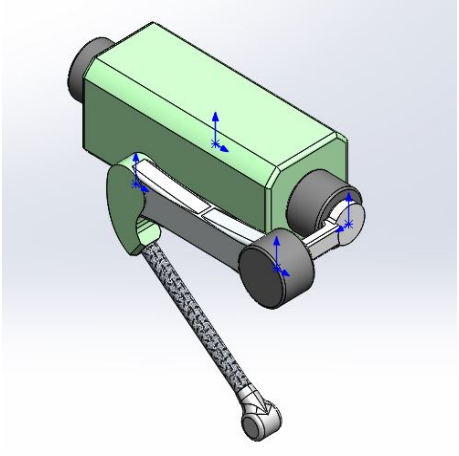


Figure 11: Body with One Leg

The links can be created for each part as displayed in Figures 12-14. Each link has a different colour to be easily distinguished.

```
<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 12: Leg Roll Segment 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 13: 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 14: Lower Leg Link

Joints are used to connect the links and have the following elements [6].

- Origin : This is the transform from the parent link to the child link. The joint is located at the origin of the child link.
  - xyz : Represents the offset. All positions are specified in meters.
  - rpy : Represents the rotation around fixed axis: first roll around x, then pitch around y and finally yaw around z. All angles are specified in radians.
- Parent Link (Required) : The name of the link that is the parent of this link in the robot tree structure.

- Child Link (Required) : The name of the link that is the child link.
- Axis : The joint axis specified in the joint frame. This is the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints.
- Calibration : The reference positions of the joint, used to calibrate the absolute position of the joint.
- Dynamics : An element specifying physical properties of the joint.
  - Damping : The physical damping value of the joint.
  - Friction : The physical static friction value of the joint.
- Limit (Required for revolute and prismatic joints) : An element can contain the following attributes :
  - Effort (Required) : 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 (Required) : An attribute for enforcing the maximum joint velocity.
- Mimic : This tag is used to specify that the defined joint mimics another existing joint.
- Safety_Controller : An element can contain the following attributes :
  - soft_lower_limit : An attribute specifying the lower joint boundary where the safety controller starts limiting the position of the joint.
  - soft_upper_limit : An attribute specifying the upper joint boundary where the safety controller starts limiting the position of the joint.
  - k_position : An attribute specifying the relation between position and velocity limits.
  - k_velocity : An attribute specifying the relation between effort and velocity limits.

Base_link and base_inertia links are connected together using a fixed joint element which can be viewed in Figure 15.

```
<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 15: Joint of Base Link with Base Inertia

The joints for the leg's links are presented in Figures 16-18.

```
<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 16: Base Link Connection to Leg Roll Segment

```
<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 17: Leg Roll Segment Connection 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 18: Upper Leg Connection to Lower Leg

A foot link is created at the toe to determine the contact point of the robot with the ground and because the mesh 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 19.

```
<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 19: 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 [7]. It allows modelling of concepts such as gear ratios and parallel linkages. A transmission transforms efforts/flow variables such that their product - power - remains constant. Multiple actuators may be linked to multiple joints through complex transmission. The transmission has one attribute :

- Name (Required) : Specifies the unique name of a transmission

And the following elements :

- Type : Specifies the transmission type.
- Joint : A joint the transmission is connected to. The joint is specified by its name attribute, and the following sub-elements:
  - HardwareInterface : Specifies a supported joint-space hardware interface. The value of this tag should be EffortJointInterface when this transmission is loaded in Gazebo and hardware_interface/EffortJointInterface when this transmission is loaded in RobotHW.
- Actuator : An actuator the transmission is connected to. The actuator is specified by its name attribute, and the following sub-elements:
  - MechanicalReduction : Specifies a mechanical reduction at the joint/actuator transmission.
  - HardwareInterface : Specifies a supported joint-space hardware interface. The ¡hardwareInterface¿ tag should only be specified here for ROS releases prior to Indigo. The correct place to specify this tag is in ¡joint¿ tag.

The Transmission element is displayed in Figure 20.

```
<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 20: Transmission Element

The XACRO files for the legs and the manipulator can be included in the main xacro file and be used to create the parts as in Figure 21.

```
<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 21: Creating The Parts

Gazebo offers a tool to evaluate the model's mass properties. By spawning the model and choosing the option for inertias and center of mass from the view menu 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 22. The center of mass for each part is displayed in Figure 23.

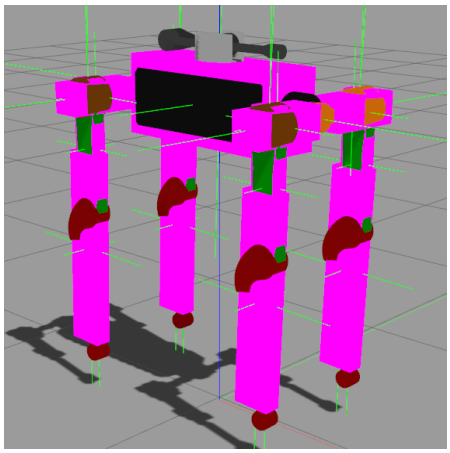


Figure 22: Inertia of Robot

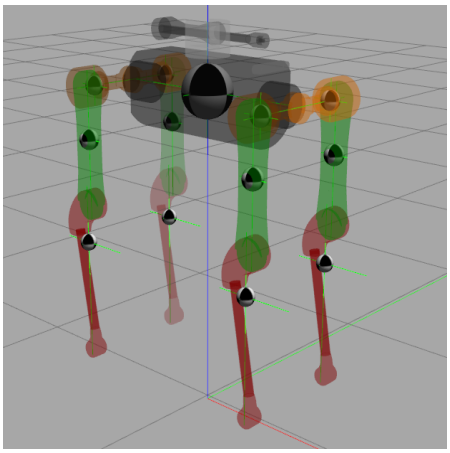


Figure 23: Centers of Mass

For obstacle detection and avoidance the model is equipped with a stereo camera. A stereo camera is a type of camera with two 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 [10]. For ARGOS

the ZED2 camera was chosen. ZED2 specifications are summarized in Figure 24 and Figure 25.

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 24: 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 25: ZED2 Depth Properties

To equip the model with the ZED2 camera, the center and the left camera frame are created as links and are connected to the body using joints as presented in Figure 26.

```
<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 26: Camera Links and Joints

To evaluate the relationship between the transforms(positions) of the links the ROS TF tree is used. The tf system in ROS keeps track of multiple coordinate frames and maintains the relationship between them in a tree structure. The base_link frame should be a parent of all the ARGOS parts and a child to the world frame (odom), which is the ultimate reference frame. Directly or indirectly, all other frames are defined with respect to the World frame. In Figure 27 the ROS TF frame tree is presented.

Figure 27: 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.

2.2.3 Gazebo References and Plugins

The toes of the robot need to have the appropriate friction to avoid sliding on the ground. Gazebo can add friction to a link by using the <gazebo> element [8]. The <gazebo> element is an extension to the URDF used for specifying additional properties needed for simulation purposes in Gazebo. The <gazebo> element for links has the following elements to add friction:

- kp,kd : Contact stiffness kp and damping kd for rigid body contacts as defined by the Open Dynamics Engine (ODE) [11]
- 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 28.

```
<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 28: Friction of Lower Leg Link

Gazebo additionally offers plugins that give your URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input [9]. Gazebo supports the plugin types :

- ModelPlugins : to provide access to the physics::Model API, e.g.: gazebo_ros_control (parses the transmission tags and loads the appropriate hardware interfaces and controller manager).
- SensorPlugins : to provide access to the sensors::Sensor API, e.g.: camera_controller (provides ROS interface for simulating cameras by publishing the CameraInfo and Image ROS messages).
- VisualPlugins : to provide access to the rendering::Visual API e.g.: display_video_controller (displays a ROS image stream on an OGRE Texture inside gazebo).

Sensors in Gazebo are meant to be attached to links, so the <gazebo> element describing that sensor must be given a reference to the camera link. Since ARGOS is equipped with a stereo camera, the sensor type that chosen was the multicamera (Figure 29).

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

Figure 29: multicamera sensor

The Left and Right camera specifications match the ZED2 camera specifications and are shown in Figures 30 and 31 (The Right camera has the same specifications except the pose).

```
<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 30: Left Camera

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

Figure 31: Right Camera Pose

To publish the stereo camera outputs on topics a plugin is needed that is shown in Figure 32.

```
<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 32: 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 33.

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

Figure 33: ROS Control Plugin

2.3 Champ

CHAMP is an open source development framework for building new quadrupedal robots and developing new control algorithms [2]. Core Features :

- Fully Autonomous (using ROS navigation Stack).
- Setup-assistant to configure newly built robots.
- Collection of pre-configured URDFs like Anymal, MIT Mini Cheetah, Boston Dynamic's Spot and LittleDog.
- Gazebo simulation environment.
- Compatible with DIY quadruped projects like SpotMicroAI and Open-Quadruped.
- Demo Applications like TOWR and chicken head stabilization.
- Lightweight C++ header-only library that can run on both SBC and micro-controllers.

2.3.1 Champ Setup Assistant

Champ Setup Assistant is a software that, if provided with a correct URDF file, auto generates a configuration package to make a quadruped robot walk [3]. In Figure 34 the Champ Setup Assistant Software is presented.

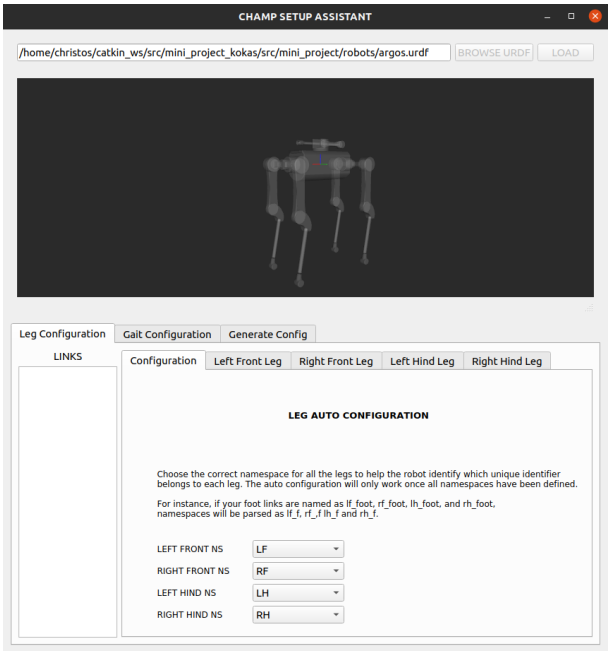


Figure 34: Champ Setup Assistant Software

2.3.2 Champ Setup Assistant Assumptions

The following assumptions have been made to avoid fragmentation across different robots as there can be thousands of ways to create robot's URDF :

- 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.

Figures 35-38 shows the above assumptions. AXES: +X:Red, +Y:Green, +Z:Blue.

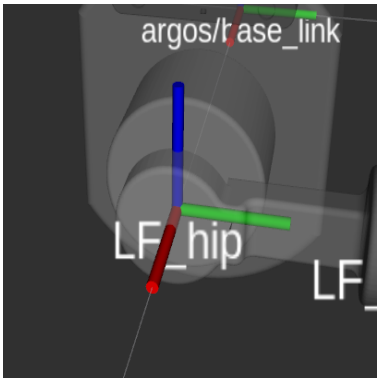


Figure 35: Hip Joint Rotation on X Axis

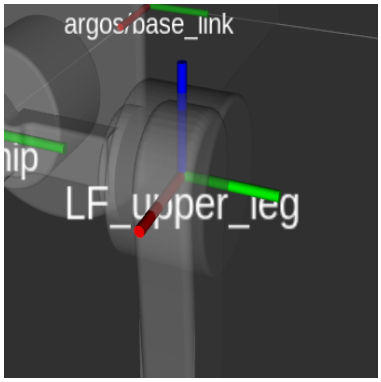


Figure 36: Upper Leg Joint Rotation on Y Axis

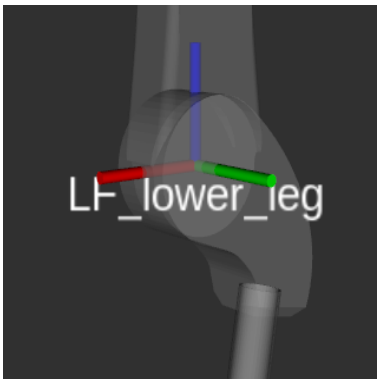


Figure 37: Upper Leg Joint Rotation on Y Axis

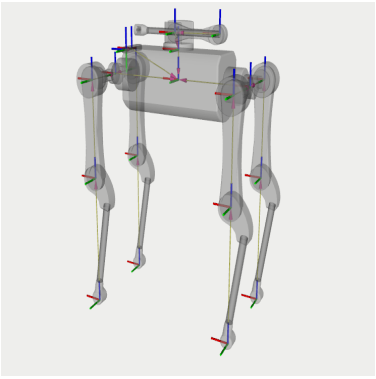


Figure 38: ARGOS Following the Above Assumptions

2.4 ROS control

The ros_control [12] package was chosen to control ARGOS. The ros_control packages takes as input the joint state data from your robot's actuator's encoders and an input set point. It uses a generic control loop feedback mechanism, typically a PID controller, to control the output, typically effort, sent to your actuators. The transmission element was needed to link

actuators to joints. Following that, a yaml file (YAML is a data serialization language that is often used for writing configuration files) 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. The data flow diagram of ROS Control is presented in Figure 39.

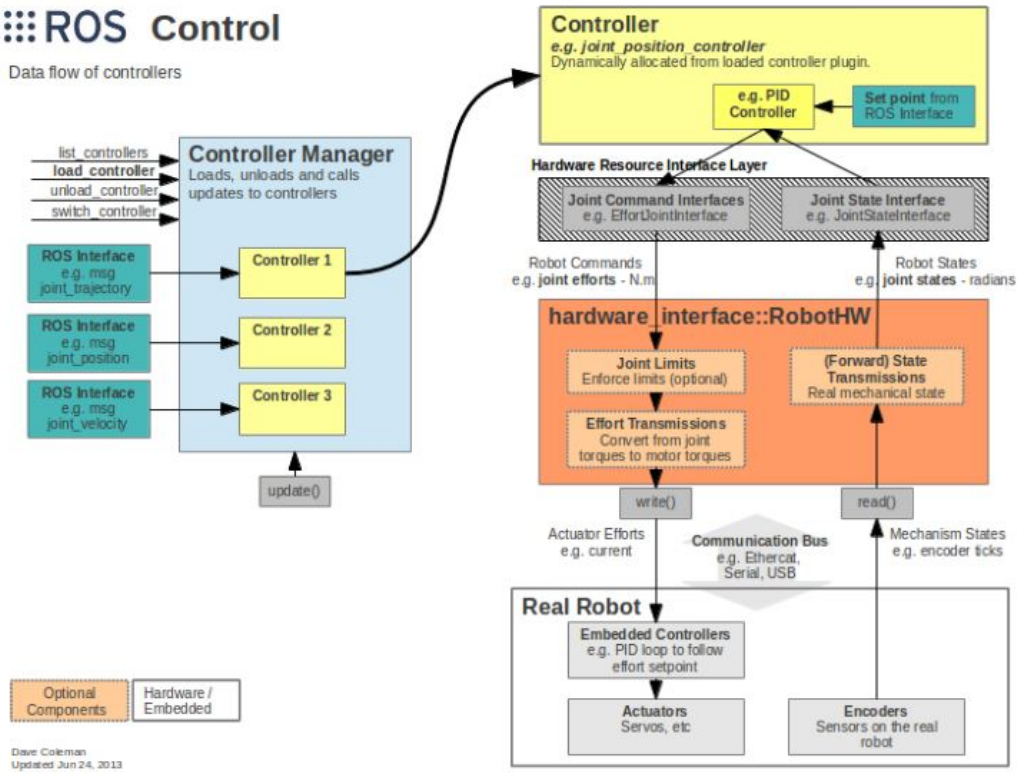


Figure 39: ROS Control Data Flow Diagram

3 Stereo Camera to Laserscan

Gmapping packages require laserscans as input to produce SLAM (Simultaneous Localization and Mapping) [13]. ARGOS is equipped with a stereo

camera which offers the topics presented in Figure 40.

3.1 Stereo Camera Topics

The stereo camera publishes its outputs of the left camera with namespace /camera/left and the outputs of the right camera with namespace /camera/right.

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

Figure 40: Stereo Camera Outputs

All topics with their corresponding 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 published topics /argos/camera/left/image_raw and /argos/camera/right/image_raw are used to produce a laserscan message and the packages needed are :

- **stereo_image_proc** : Processes the images from 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 41.

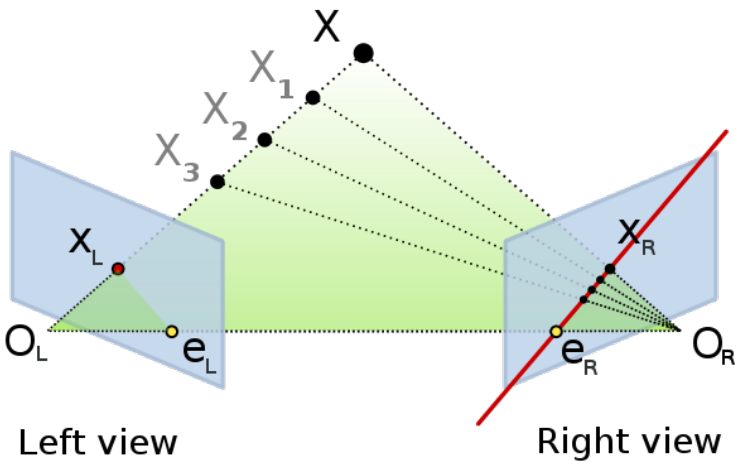


Figure 41: Disparity

- **disparity_image_proc** : 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.

To Convert the produced disparity image from the stereo_image_proc package, the package disparity_image_proc is used. The raw image and the produced depth image from the disparity_image_proc package is displayed in Figure 42.

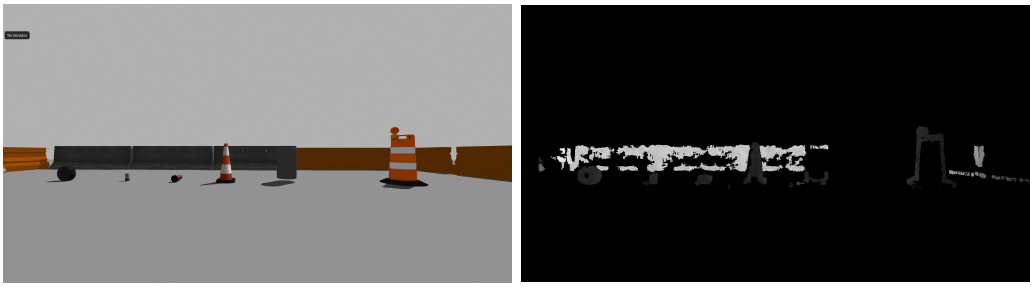


Figure 42: Raw and Produced Depth Image

- **depthimage_to_laserscan** [16] : This package generates a 2D laser scan from a depth image based on the provided parameters.

The ROS Graph of the communication of each of these packages are presented in Figures 43-45.

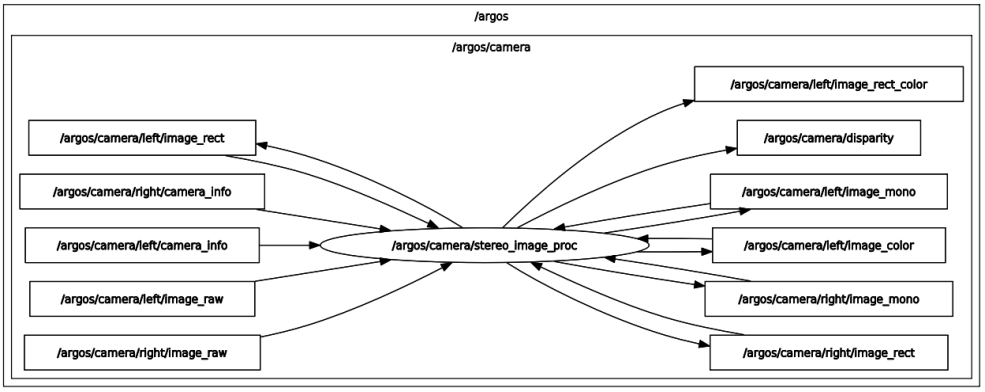


Figure 43: stereo_image_proc

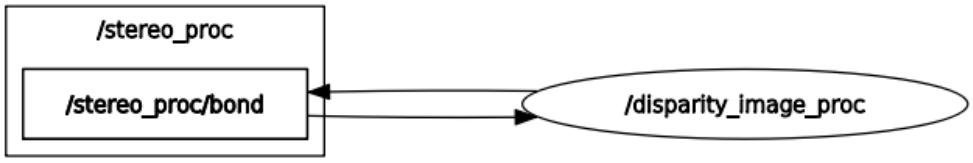


Figure 44: disparity_image_proc

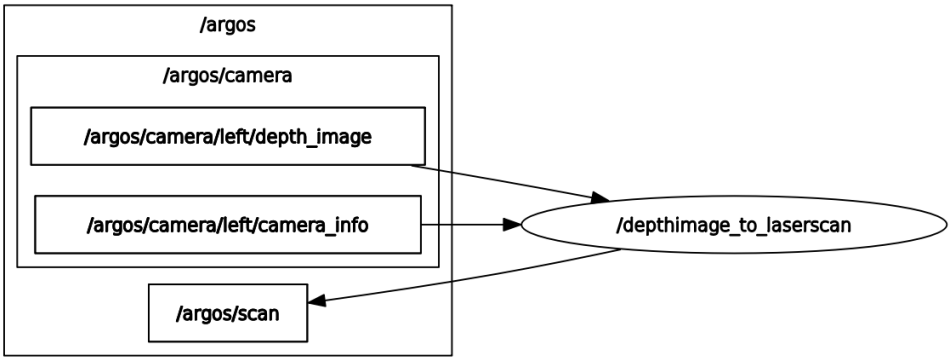


Figure 45: depth_to_laserscan

The Complete ROS Graph is presented in Figure 46.

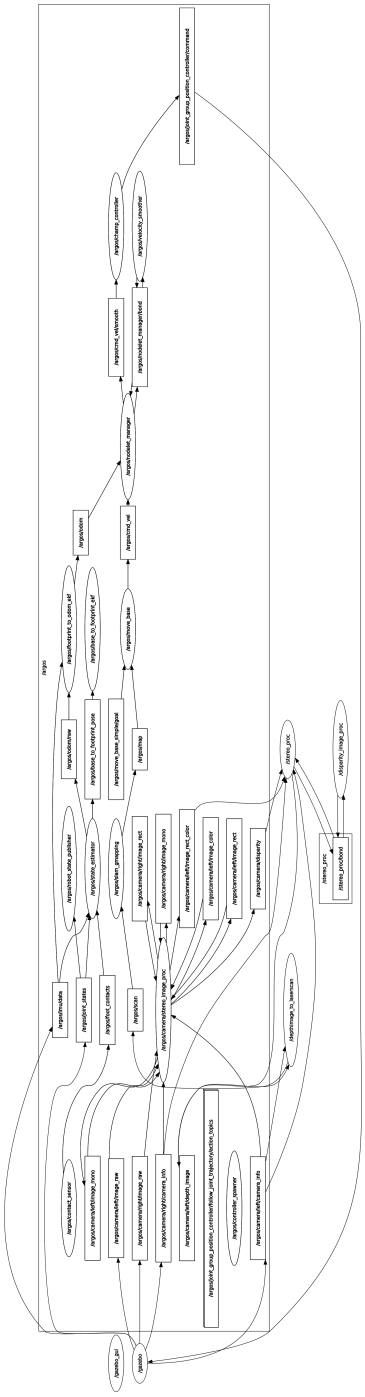


Figure 46: Complete ROS Graph

3.3 Launch Files

To use the stereo_image_proc package the namespace of the cameras has to be defined on the launch file as shown in Figure 47.

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

Figure 47: Stereo Image Proc Launch

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 48.

```
<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="/argos/camera/right/camera_info"/>
  <remap from="/left/camera_info"
      to="/argos/camera/left/camera_info"/>
  <remap from="/left/image_rect_color"
      to="/argos/camera/left/image_rect_color"/>
  <remap from="/disparity" to="/argos/camera/disparity"/>
  <remap from="/depth_image" to="/argos/camera/left/depth_image"/>
</node>
```

Figure 48: Disparity Image Proc Launch

The produced laserscan message from the depthimage_to_laserscan package is published by default on the topic /scan. ARGOS has a namespace of /argos and as a result the gmapping packages search for the laserscan message at topic named /argos/scan. For that reason the /scan topic has to be remapped to /argos/scan. To accomplish that, the source file DepthImageToLaserScanNodelet was loaded from the arguments on the launch file, and following that, the /scan topic can be remapped to /argos/scan as presented in Figure 49.

```
<node name="depthimage_to_laserscan"
    pkg="depthimage_to_laserscan" type="depthimage_to_laserscan"
    args="load
    depthimage_to_laserscan/DepthImageToLaserScanNodelet">
  <remap from="image"     to="/argos/camera/left/depth_image"/>
  <remap from="camera_info" to="camera/color/camera_info"/>
  <remap from="scan" to="argos/scan"/>
```

Figure 49: Scan topic Remapped in DepthImage to Laserscan Launch File

The Scan height parameter (Number of pixel rows used to generate laser scan) was additionally changed to the value of 1080 pixels to match the stereo camera resolution as presented in Figure 50.

```
<param name="scan_height"  type="int"   value="1080"/>
```

Figure 50: Scan Height Parameter

A launch file was created to include all parameters, remapping and arguments. Firstly, values are chosen for arguments as presented in Figure 51.

```
  <arg name="robot_name"      default="argos"/>
  <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 51: 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 presented in Figure 52.

```
<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 52: Gazebo and Controllers Launch

Lastly, the packages for the image processing are launched to produce the laserscan for the gmapping packages in Figure 53.

```
<node ns="argos/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 53: Image Processing Packages

The Gazebo world with the resulting laserscan is presented in Figure 54.

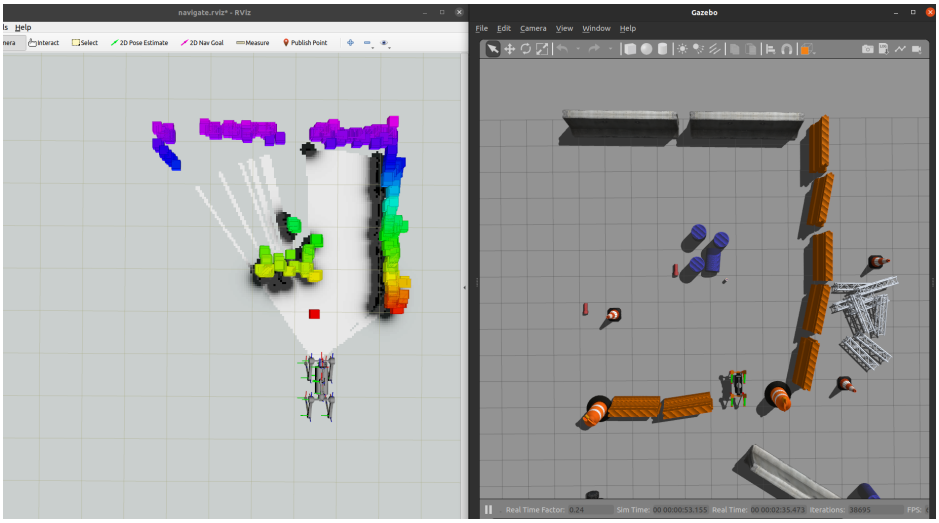


Figure 54: Produced LaserScan

4 Project Challenges

Choosing the parameters for the controllers as well as the parameters for the movement of the robot was challenging. Each actuator is controlled by a PID controller. For ARGOS the PD controller was chosen. The P gain for each controller was chosen with a value of 1500 to be able to counteract the weight of the robot. Lower values for the P gain resulted in stabilization at a lower nominal height or instability, and higher values resulted in abrupt movement resulting in instability. The D gain for each controller was chosen

with a value of 80 to remove unwanted oscillations. Lower values meant that ARGOS would oscillate for a long period of time while higher values resulted in instability. The values chosen for each actuator are displayed in Figure 55.

```
gains:
  LF_hip_joint : {p: 1500, d: 80.0, i: 0.0}
  LF_upper_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  LF_lower_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  RF_hip_joint : {p: 1500, d: 80.0, i: 0.0}
  RF_upper_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  RF_lower_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  LH_hip_joint : {p: 1500, d: 80.0, i: 0.0}
  LH_upper_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  LH_lower_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  RH_hip_joint : {p: 1500, d: 80.0, i: 0.0}
  RH_upper_leg_joint : {p: 1500, d: 80.0, i: 0.0}
  RH_lower_leg_joint : {p: 1500, d: 80.0, i: 0.0}
```

Figure 55: PID Gains

The way the robot moves is controlled by the gait parameters which are the robot's walking parameters. The gait parameters are :

- Knee Orientation : How the knees should be bent. There are 4 configurable orientations : .>> .>< .<< .<> where dot is the front side of the robot.
- Max Linear Velocity X (meters/second) : Robot's maximum forward/reverse speed.
- Max Linear Velocity Y (meters/second) : Robot's maximum speed when moving sideways.
- Max Angular Velocity Z (radians/second) : Robot's maximum rotational speed.
- Stance Duration (seconds) : How long should each leg spend on the ground while walking.
- Leg Swing Height (meters) : Trajectory height during swing phase.
- Leg Stance Height (meters) : Trajectory depth during stance phase.

- Robot Walking Height (meters) : Distance from hip to the ground while walking.
- CoM X Translation (meters) : This parameter is used to compensate for the weight of the robot if the center of mass is not in the middle of the robot.
- Odometry Scaler : This parameter can be used as a multiplier to the calculated velocities for dead reckoning. This can be useful to compensate odometry errors on open-loop systems.

The gait parameters are summarized in Figure 56.

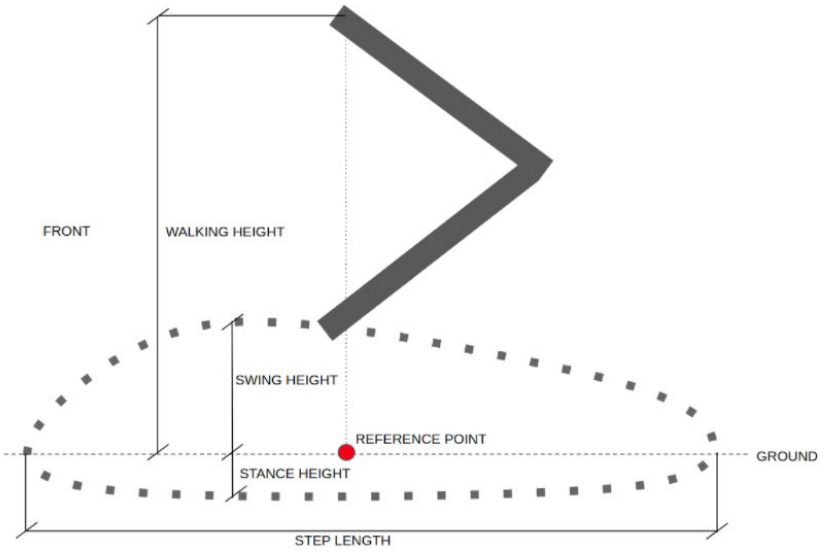


Figure 56: Gait Parameters

The parameter values that were chosen for ARGOS are presented in Figure 57.

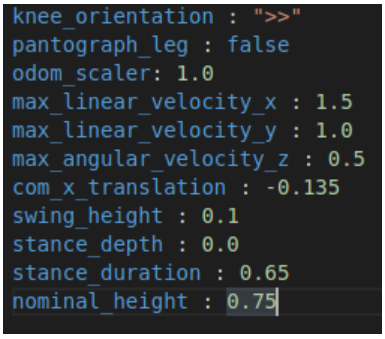


Figure 57: Chosen Gait Parameters

5 Results

5.1 Robot's Stability and Movement

In this section the results are shown of the robot's stability on four legs, and moving while avoiding obstacles detected by the stereo camera. The simulation is launched using the command in Figure 58.

```
rolaunch argos_config gazebo.launch
```

Figure 58: Gazebo Launch

This launch file launches the gazebo world and spawns the robot from height of 1.08 meters (which is the height of the robot with stretched legs) and launches the packages needed for the conversion of the stereo camera messages to the laserscan. To verify that the laserscan massage is being published on the /argos/scan topic, the command in Figure 59 can be used.

```
rostopic list | grep scan
```

Figure 59: Command for Scan Topic

In Figure 60 the robot is at its starting position with the command showing the scan topic.

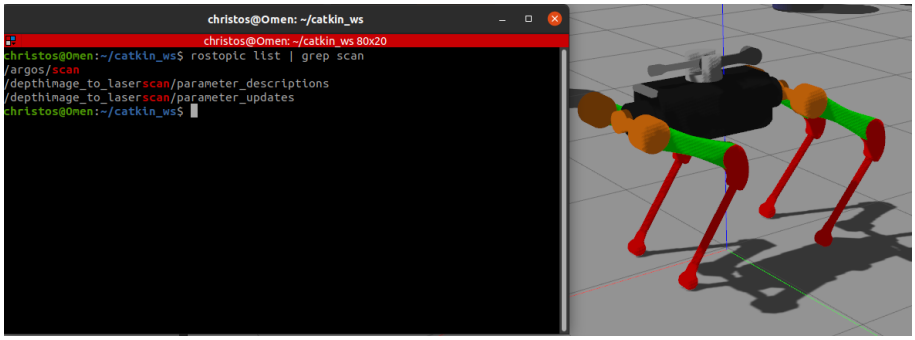


Figure 60: Starting Position of the Robot

To launch the obstacle avoidance using gmapping packages the command in Figure 61 is executed.

```
roslaunch argos_config slam.launch
```

Figure 61: Command for Obstacle Avoidance with Gmapping Packages

This command launches 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 ARGOS is shown in Figure 62.

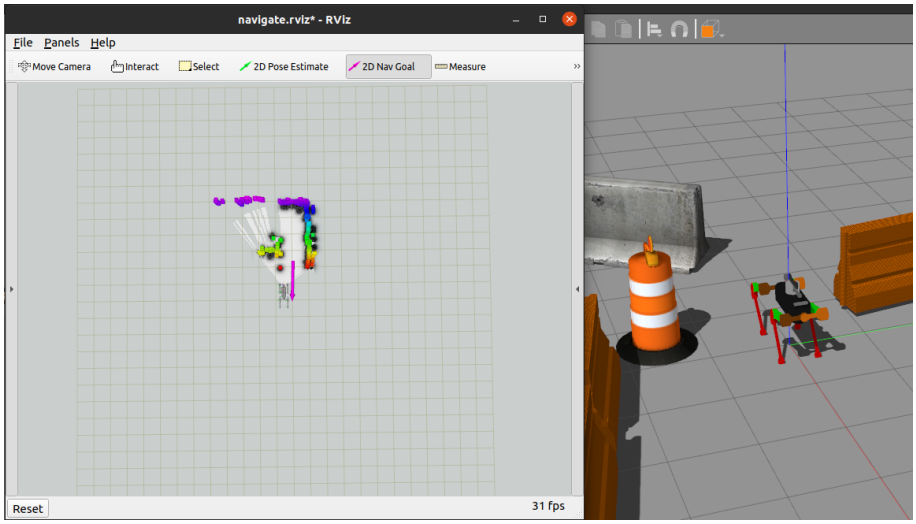


Figure 62: Robot Goal and Orientation Position

The robot after given the instruction creates a path to follow and begins the movement towards the goal as shown in Figure 63 (left to right, top to bottom).

Figure 63: Movement of the Robot Towards Goal

The terminal is displayed in Figure 64 as the robot reached the goal.

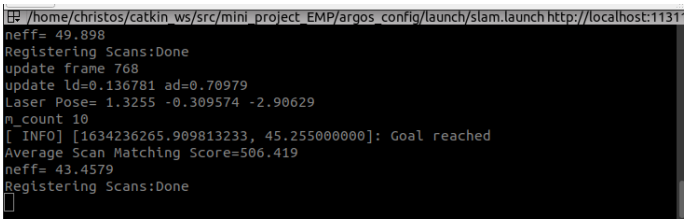


Figure 64: Goal Reached

The path planning also avoids obstacles detected by the laserscan as shown in Figure 65.

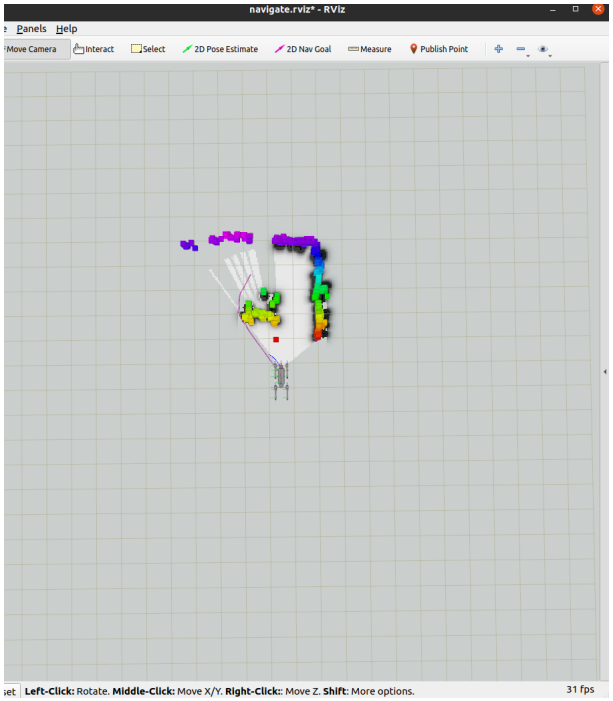


Figure 65: Obstacle Avoidance

5.2 Obstacle Detection

In this section the results for obstacle detection using the stereo camera are presented. To distinguish the obstacles that can be detected, a new gazebo world was created with obstacles of different size (biggest obstacle is a barrel,

smallest obstacle is a drink can). The gazebo world is presented in Figure 66.

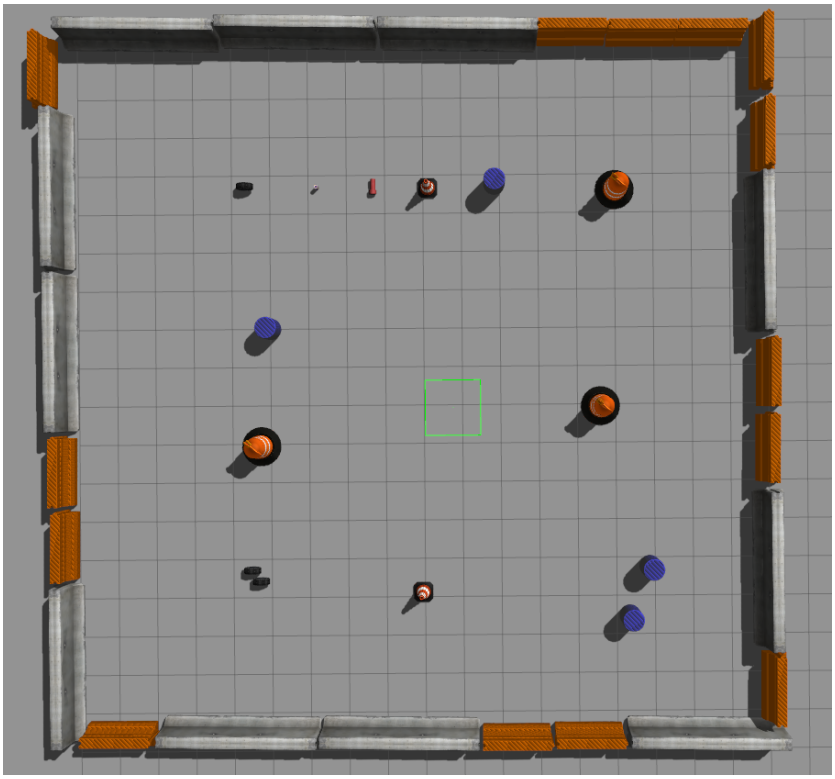


Figure 66: Gazebo Obstacle World

To spawn ARGOS in the new gazebo world the argument gazebo_world in the launch file is changed to match the newly created world as presented in Figure 67.

```
<arg name="gazebo_world"    default="$(find
    argos_config)/worlds/arena_world.world" />
```

Figure 67: Gazebo World Argument

The commands in Figure 68 are executed in separate terminals to launch the gazebo world, spawn ARGOS and launch the gmapping packages.

```
roslaunch argos_config gazebo.launch

roslaunch argos_config slam.launch
```

Figure 68: Commands to Launch Gazebo and Gmapping

The gazebo obstacle world with ARGOS is presented in Figure 69.

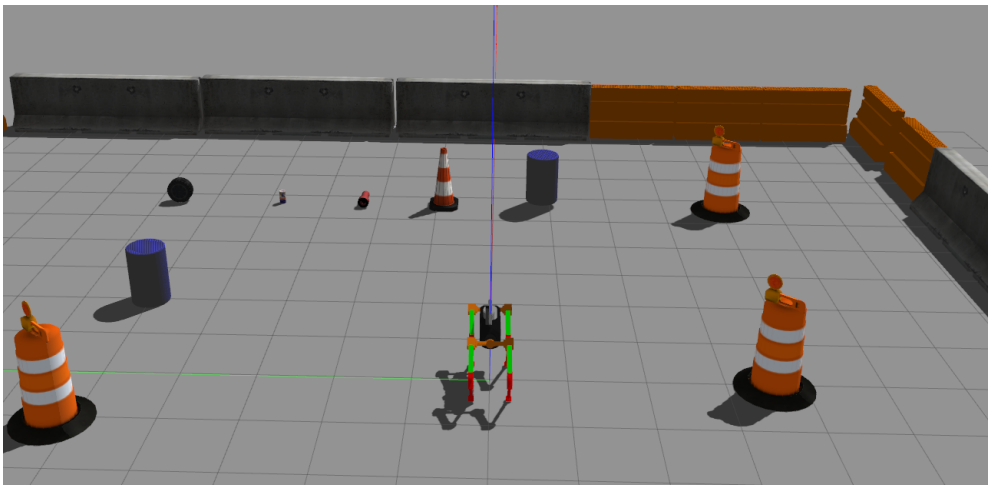


Figure 69: ARGOS in Gazebo Obstacle World

In Figure 70 ARGOS is at starting position and the laserscan can detect all obstacles of different size.

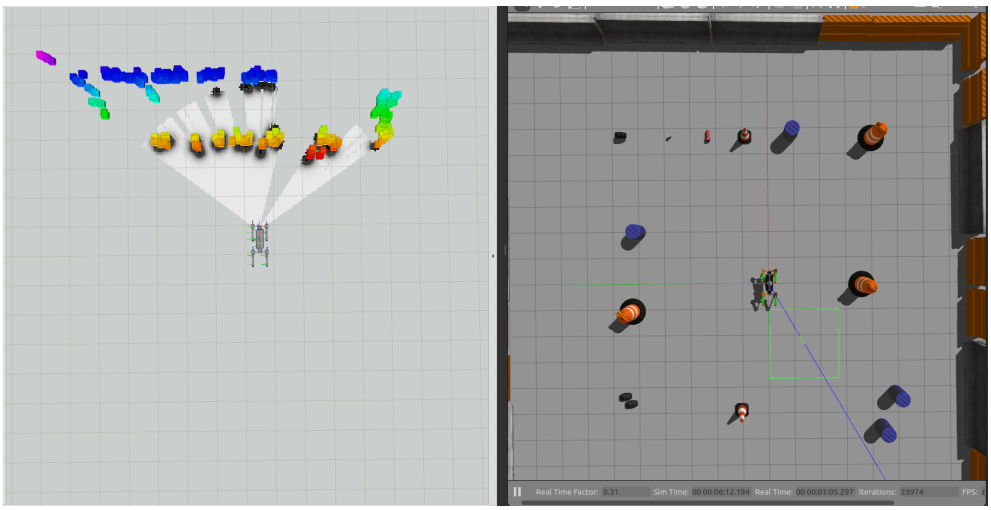


Figure 70: Obstacle Detection : Starting Position

In Figures 71-72 ARGOS is instructed to move closer to the obstacles to verify that the laserscan can still detect them from closer range.

Figure 71: Obstacle Detection : Big Obstacles

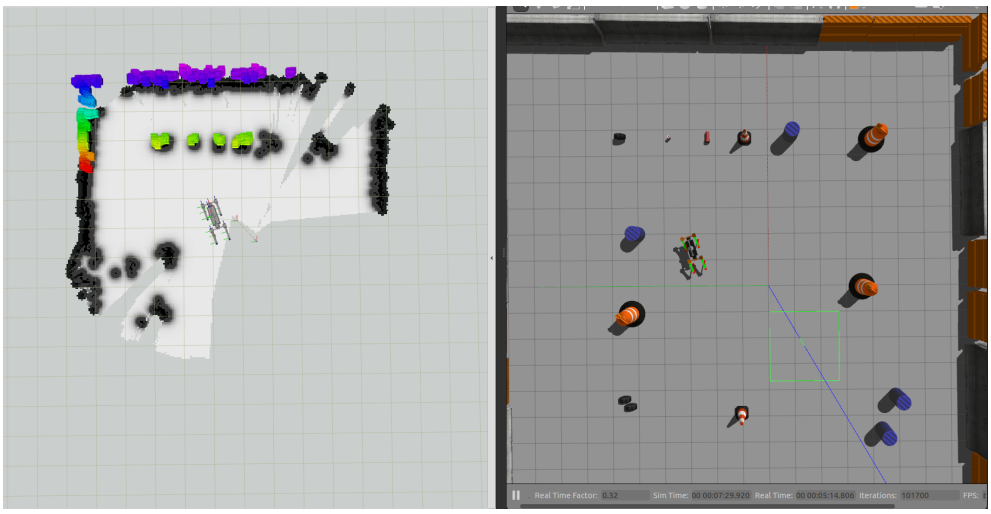


Figure 72: Obstacle Detection : Small Obstacles

In can be verified from Figures 70-72 that the laserscan can detect all obstacles in this gazebo world.

6 Improvements that could be Made

One improvement that could be made is to find correct values for the parameters to make the movement of ARGOS more accurate. By using 3 packages to convert the stereo images to a laserscan it is highly probable for errors to be produced, so appropriate packages could be selected to minimize the errors that are produced.

List of Figures

References

[1] XACRO XML macro language, ROS Wiki.

[2] Champ,GitHub.

[3] Champ Setup Assistant, GitHub.

[4] Link, ROS Wiki.

[5] kdl_parser, ROS Wiki.

[6] Joint, ROS Wiki.

[7] Transmission in URDF, ROS Wiki.

[8] Gazebo Reference Tag, GazeboSim.

[9] Gazebo Plugins, GazeboSim.

[10] Stereo Camera, Wikipedia.

[11] Open Dynamics Engine, ODE.

[12] Gazebo ROS Control, ROS Wiki.

[13] Gmapping Packages, ROS Wiki.

[14] Stereo Image Proc package, GitHub.

[15] Disparity Image Proc package, GitHub.

[16] Depthimage To Laserscan package, GitHub.