Universidade Federal do Ceará - UFC
Centro de Tecnologia
Departamento de Engenharia Elétrica
Grupo de Pesquisa em Automação, Controle e Robótica

Criando ambiente de simulação no Gazebo

William S. Ferreira

2022

Sumário

1 Objetivos

Utilizando o software *Gazebo* criar um robô móvel e um ambiente de simulação que será usados em futuros testes.

Objetivos específicos:

- Ter robô móvel funcional no Gazebo (podendo ser criado ou utilizado algum);
- Adicionar recurso de Odometria;
- Adicionar sensor laser;
- Adicionar IMU;
- Criar um cômodo no Gazebo, para locomoção do robô;
- Conexão com o ROS2;
- Movimento do robô via ROS2;
- Visualização dos dados de odometria;
- Visualização dos dados do Laser.

2 Metodologia

A utilização do software *Gazebo* será realizada em conjunto com o *ROS2*, versão *foxy*. Inicialmente é apresentado o ambiente de simulação, para então buscar realizar os objetivos deste relatório.

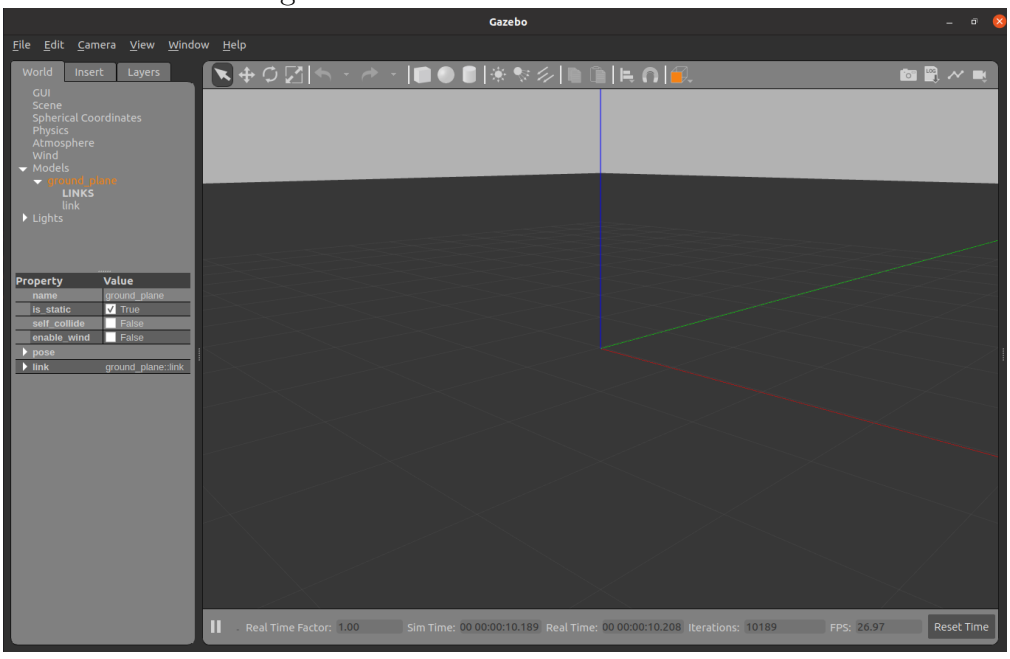
2.1 Gazebo

Ao se instalar a versão completa do *ROS2*, é instalado o software *Gazebo*. Para sua execução, digita-se no terminal:

```
$ gazebo
```

Na figura 1 é possível visualizar a interface do programa, que é o ambiente de simulação.

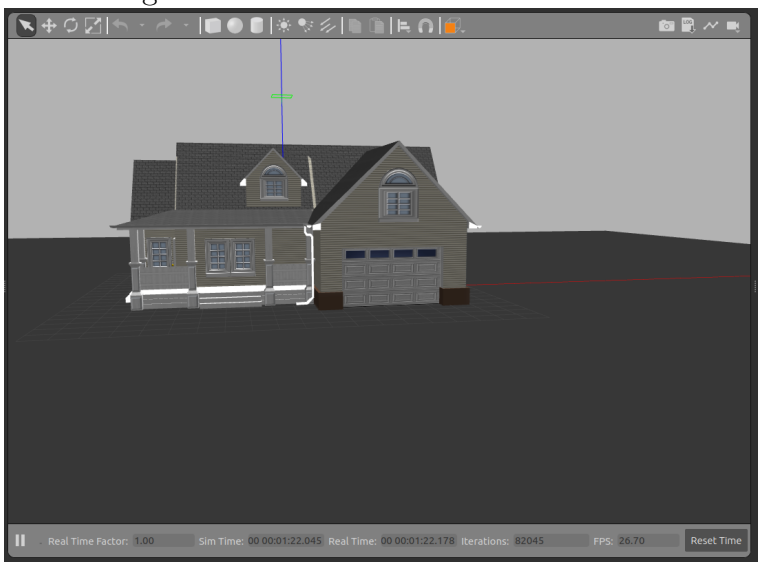
Figura 1: Interface Inicial do Gazebo



Fonte: O próprio autor

A partir deste software podemos adicionar estruturas, como por exemplo uma casa. Algumas destas construções são encontradas online.

Figura 2: Estrutura de casa no Gazebo

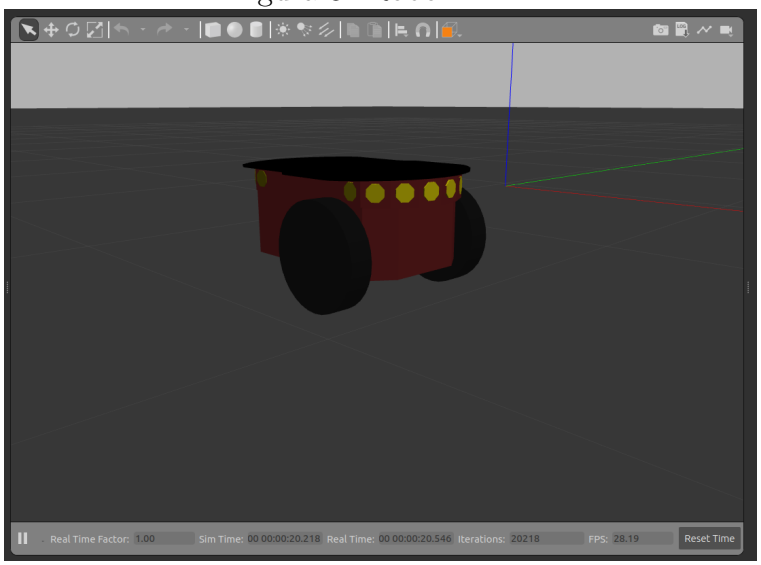


Fonte: O próprio autor

2.2 Robô P2DX

Para as simulações será utilizado a estrutura de um robô diferencial da biblioteca de modelos do Gazebo, o **P2DX**. O modelo do robô pode ser visto na figura 3.

Figura 3: Robô P2DX



Fonte: O próprio autor

Para realizar modificações, no **P2DX**, basta abrir o modelo *sdf* do mesmo. Nesse arquivo, é possível alterar as características físicas da estrutura do robô, além da física de cada componente, como eles reagem ao mundo. Um trecho de um arquivo *sdf* é mostrado a seguir, no mesmo, ele tem como intuito adicionar a roda direita de um robô móvel. É possível definir

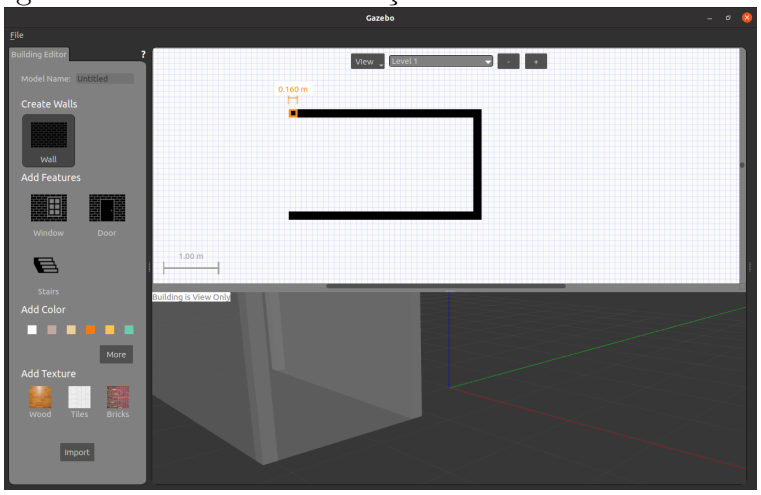
a posição que esse elemento será adicionado, além de definir seu visual. O mesmo acontece com o robô **P2DX**.

```
<link name="right_wheel">
        <pose>0.1 -0.13 0.1 0 1.5707 1.5707</pose>
        <collision name="collision">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </visual>
      </link>
```

2.3 Ambiente

A criação de ambientes no Gazebo é simples, se utilizado o recurso *Building Editor*, localizado na aba *Edit*. Esta ferramenta permite adicionar paredes, portas e janelas em um plano 2D, na qual é levado para o ambiente 3D.

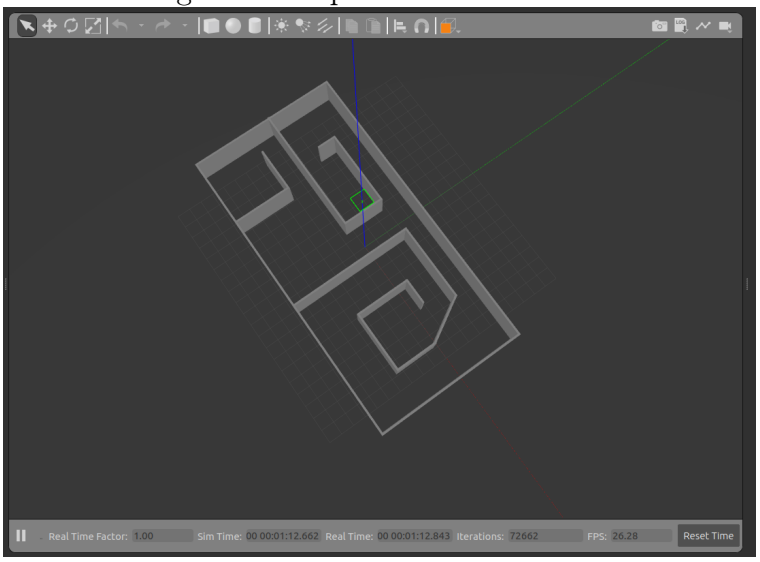
Figura 4: Recurso de construção de ambientes no Gazebo



Fonte: O próprio autor

Utilizando deste recurso, foi criado um ambiente fechado para testes futuros. O mesmo está ilustrado na figura 5

Figura 5: Mapa criado no Gazebo



Fonte: O próprio autor

2.4 Conexão com ROS2

Utilizando o robô **P2DX** no ambiente do Gazebo, se faz necessário obter os dados de localização do robô, ou adicionar sensores e captar essas informações. Além do controle manual, ou de controles de trajetórias que podem ser implementados. Para isso, conectou-se o ambiente de simulação com o ambiente do ROS2. Essa conexão é feita por meio de *plugins* [1] adicionados no arquivo *sdf* do robô.

Nesta etapa inicial, alguns *plugins* foram adicionados: *differential drive*, *ray sensor* e *imu sensor*.

2.5 Odometria

O primeiro *plugin* a ser apresentado é o que permite a odometria do robô móvel: *differential drive*.

```
<plugin name='diff_drive' filename='libgazebo_ros_diff_drive.so'>
        <ros>
          <namespace>/demo</namespace>
          <remapping>cmd_vel:=cmd_vel</remapping>
          <remapping>odom:=odom</remapping>
        </ros>

        <!-- wheels -->
```

```
    <left_joint>left_wheel_hinge</left_joint>
    <right_joint>right_wheel_hinge</right_joint>

    <!-- kinematics -->
    <wheel_separation>0.26</wheel_separation>
    <wheel_diameter>0.2</wheel_diameter>

    <!-- limits -->
    <max_wheel_torque>20</max_wheel_torque>
    <max_wheel_acceleration>1.0</max_wheel_acceleration>

    <!-- output -->
    <publish_odom>true</publish_odom>
    <publish_odom_tf>true</publish_odom_tf>
    <publish_wheel_tf>true</publish_wheel_tf>

    <odometry_frame>odom</odometry_frame>
    <robot_base_frame>base_footprint</robot_base_frame>
    <odometry_source>0</odometry_source>
  </plugin>
```

No código acima, é apresentado a estrutura utilizada no **P2DX**. A classe mais externa é do *plugin*, indicando qual o nome do arquivo que contém as informações de sua estrutura. Com isso, modifica-se os parâmetros envolvendo o *plugin*.

2.6 LIDAR

O mesmo procedimento é aplicado para adição do laser no **P2DX**. Neste caso, há um cuidado adicional, já que o mesmo é um elemento físico, e sua localização interfere nos dados capturados.

```
<link name="lidar">
    <pose>0.15 0 0.3 0 0 -2</pose>

    <sensor name="rplidar" type="ray">
      <always_on>true</always_on>
      <update_rate>30</update_rate>
      <visualize>true</visualize>
      <topic>scan</topic>
      <ray>
        <scan>
          <horizontal>
            <samples>360</samples>
            <resolution>1</resolution>
            <min_angle>0</min_angle>
```

```
            <max_angle>4.19</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.1</min>
          <max>5.6</max>
          <resolution>0.2</resolution>
        </range>
      </ray>
      <plugin name="laser" filename="libgazebo_ros_ray_sensor.so">
        <ros>
          <argument>~/out:=scan</argument>
        </ros>
        <output_type>sensor_msgs/LaserScan</output_type>
        <frame_name>laser</frame_name>
      </plugin>
    </sensor>
    <visual name="visual">
      <geometry>
        <mesh>
          <uri>model://hokuyo/meshes/hokuyo.dae</uri>
        </mesh>
      </geometry>
    </visual>
</link>
```

A localização do laser é definida no começo: *<pose>0.15 0 0.3 0 0 -2</pose>*. Na qual, são 6 indicações, três referente aos eixos, e as outras três referente aos ângulos de *roll, pitch* e *yaw*. Algumas características adotadas são retiradas das especificações do sensor *hokuyo*, e o mesmo foi utilizado como modelo visual do sensor da simulação. Esta parte está descrita no final do código, no tópico de *mesh*.

Outro ponto importante, é anexar o sensor ao robô. Para que com o movimento do **P2DX**, o sensor não caia.

```
<joint type="fixed" name="laser_joint">
        <child>lidar</child>
        <parent>base_link</parent>
  </joint>
```

2.7 IMU

O último *plugin* adicionado é o sensor inercial, este adiciona um acelerômetro e um giroscópio ao robô.

```
<link name = "imu">

        <gravity>true</gravity>
        <sensor name="imu_sensor" type="imu">
                <always_on>true</always_on>
                <update_rate>100</update_rate>
                <visualize>true</visualize>
                <topic>imu</topic>
        <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
                <initial_orientation_as_reference>false</initial_orientation
                <ros>
                  <namespace>/imu</namespace>
                  <remapping>~/out:=data</remapping>
                </ros>
        <bodyName>imu_link</bodyName>
        <updateRateHZ>10.0</updateRateHZ>
        <gaussianNoise>0.0</gaussianNoise>
        <xyzOffset>0 0 0</xyzOffset>
        <rpyOffset>0 0 0</rpyOffset>
        <output_type>sensor_msgs/Imu</output_type>
        <frame_name>imu_link</frame_name>
      </plugin>
    </sensor>
    </link>
```

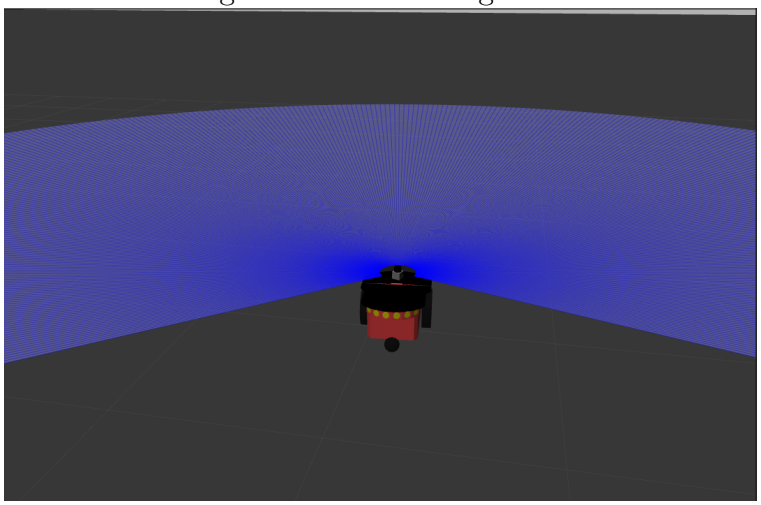
Assim como o laser, é utilizado o *joint* com o IMU. Pois é necessário indicar nossa referência para o sensor. Neste caso, vamos utilizar a base do robô.

```
<joint type="fixed" name="imu_joint">
                <child>imu</child>
                <parent>base_link</parent>
        </joint>
```

3 Resultados

Utilizando todos os *plugins*, com a estrutura física do **P2DX**, podemos verificar na figura 6 o resultado.

Figura 6: **P2DX** no gazebo



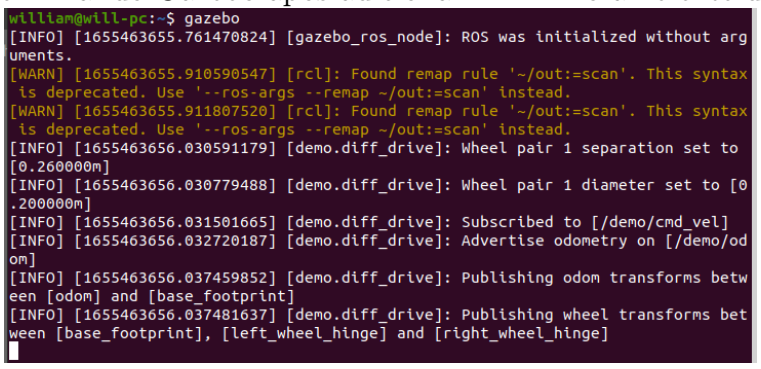
Fonte: O próprio autor

A conexão com o ROS2 é constatada pelo próprio terminal na qual inicializou-se o Gazebo, como mostra a figura 7. Outra forma de fazer essa verificação, é utilizando o comando:

```
$ ros2 topic list
```

O resultado do comando, considerando os *plugins* adicionados, deve indicar um tópico relacionado com a odometria, um com o laser e outro com os dados do IMU. Um exemplo de saída pode ser verificado na figura 8.

Figura 7: Terminal do Gazebo após adicionar P2DX no ambiente de simulação



Fonte: O próprio autor

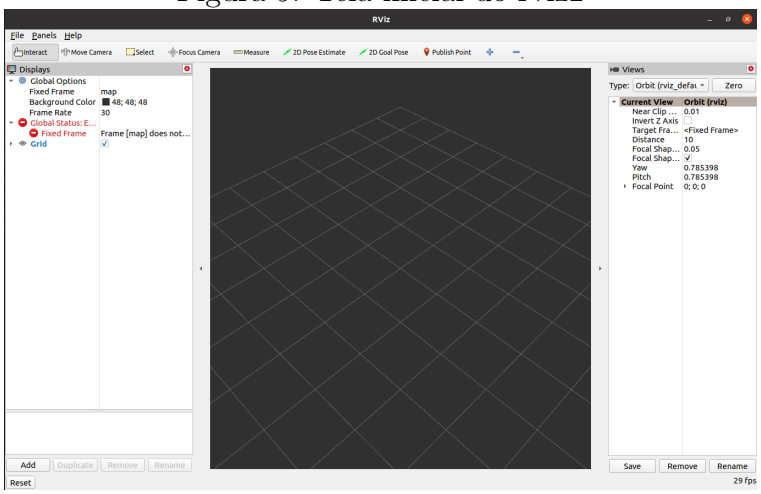
Figura 8: Tópicos do ROS2 com P2DX no Gazebo

```
william@will-pc:~$ ros2 topic list
/clock
/demo/cmd_vel
/demo/odom
/imu/data
/parameter_events
/rosout
/scan
/tf
```

Fonte: O próprio autor

Os dados do tópico */scan* são do laser, e o mesmo pode ser visualizado utilizando o *rviz2*. Digitando *rviz2* no terminal, considerando que o mesmo é instalado com o ROS2, abre uma janela, na qual podemos fazer a visualização dos dados do laser, como mostra a figura 9. Porém, para isso precisamos adicionar o tópico dos dados a serem lidos.

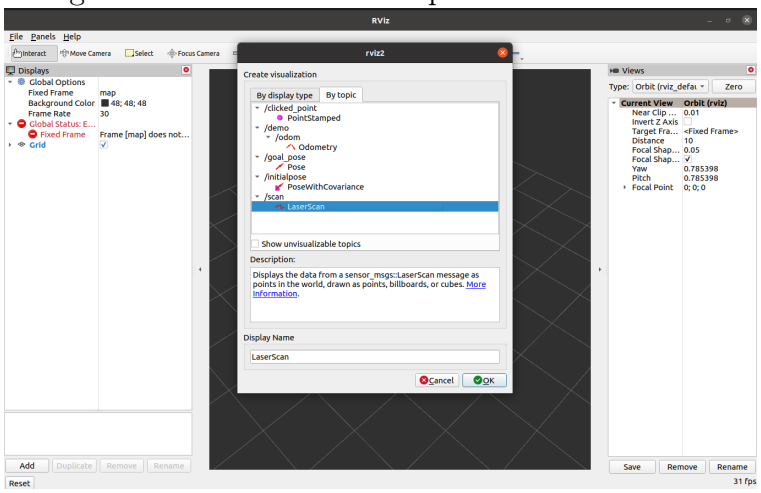
Figura 9: Tela inicial do rviz2



Fonte: O próprio autor

No canto inferior esquerdo, há um item indicando *Add*. Clicando no mesmo, abre uma janela com duas abas: *By display type* e *By topic*. Como temos um tópico com a informação do sensor laser, então clicamos nesse item. E finalmente, clicamos no tópico que as informações estão sendo publicadas, no caso */scan*, como indica a figura 10.

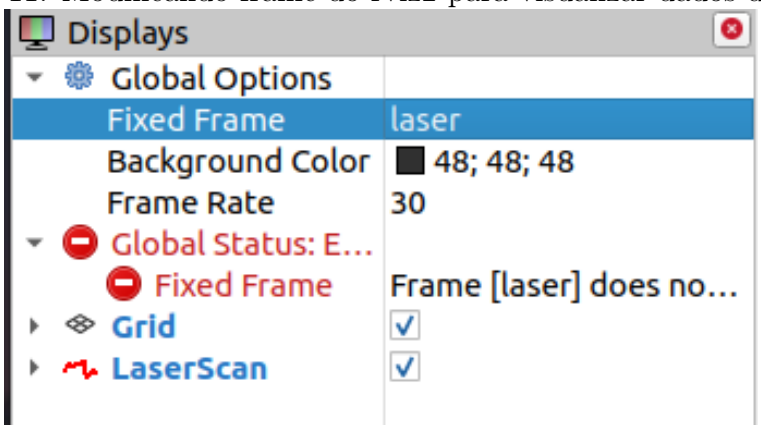
Figura 10: Adicionando o tópico do laser no rviz2



Fonte: O próprio autor

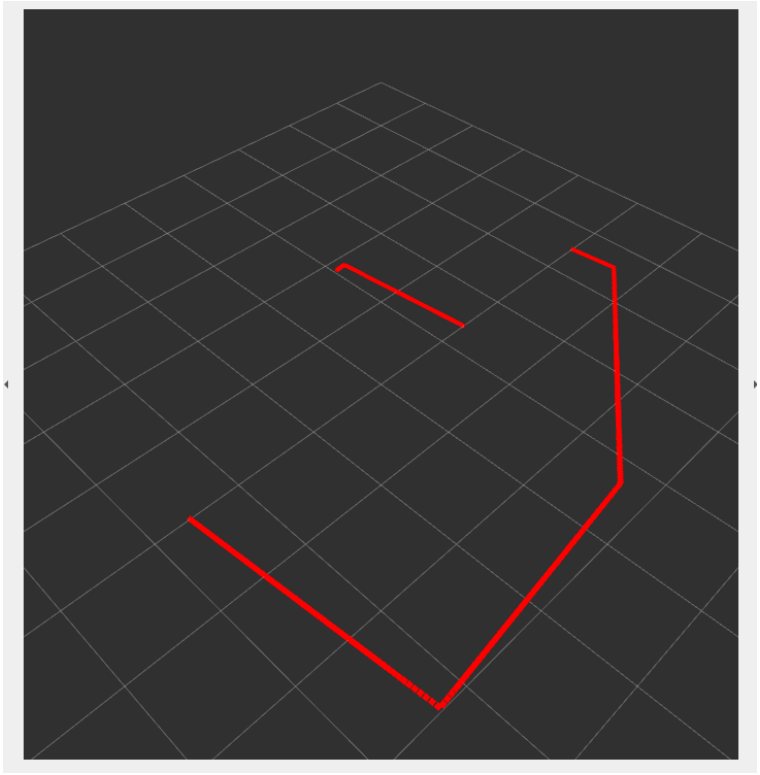
Porém, antes de a informação ser mostrada na tela do rviz2, é necessário modificar o frame. Na figura 11 indica onde fazer a modificação, é colocado o nome *laser*, que é o frame do sensor. Com isso, e o **P2DX** adicionado no Gazebo, devemos começar a visualizar listras vermelhas na tela do rviz2. Se o laser não tiver nenhum obstáculo, ou seja, indo para infinito, não será mostrado nada. Para isso, colocou-se um ambiente no Gazebo, para que os raios do laser possam ser refletidos. Na figura 13 é o ambiente e o robô no Gazebo, enquanto que na figura 12 é o resultado no rviz2.

Figura 11: Modificando frame do rviz2 para visualizar dados do laser



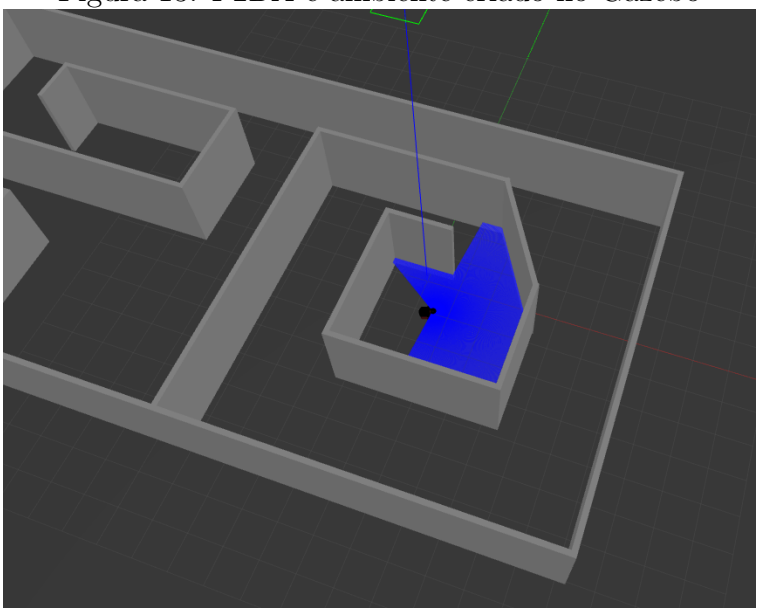
Fonte: O próprio autor

Figura 12: Dados do laser no rviz2



Fonte: O próprio autor

Figura 13: P2DX e ambiente criado no Gazebo



Fonte: O próprio autor

O controle do robô no Gazebo é feito via ROS2, na qual seu código está disponível no github do GPAR, e nos anexos deste arquivo.

4 Conclusão

Com o suporte do software Gazebo é possível a simulação de robôs móveis, além de sensores. Isso permite um ambiente de simulação na qual pode ser testado controles, códigos, de forma fácil, antes de serem levados para o mundo real. Nesta primeira parte de criação do ambiente de simulação, foi adicionado sensores essenciais para as pesquisas em andamento, como o *encoder*, que permite a odometria do robô. Porém, um problema foi encontrado com este sensor. A localização dada pela odometria, é em relação ao mundo, e não ao robô. Há um parâmetro do *plugin* que faz essa alteração, porém o mesmo não funcionou durante os testes.

Foi possível adicionar o sensor laser, este que se torna importante para uma aplicação SLAM em 2D. Nos parâmetros do *plugin* é possível modificar frequência de amostragem, o alcance do sensor, ângulos máximos e mínimos, entre outros. Isto permite adequar o sensor para aquele que será utilizado nos experimentos reais. Outro ponto, é que a leitura pode ser realizada de forma horizontal, além de vertical, o que permite a criação de ambientes em 3D. Pode ser aproveitado para o trabalho de nuvem de pontos futuramente.

O sensor IMU também foi adicionado no robô P2DX, este recurso será importante caso seja necessário fazer alguma fusão sensorial, com os dados do SLAM ou da odometria. Nos parâmetros do IMU, é possível adicionar ruído, o que pode ser vantajoso para testar a fusão. Uma das fusões que seria importante, é a que resulte no ângulo *yaw* do robô. Já que uma orientação mais precisa é importante em aplicações práticas.

Este foi só o começo das aplicações envolvendo o ambiente do Gazebo. O próximo passo, é utilizar o robô, com seus sensores, para testar códigos e controladores. Além disso, futuramente, buscar criar ambientes dinâmicos, com pessoas circulando. Outros robôs, como braços ou drones. E adicionar mais sensores, como a câmera.

Referências

[1] Gazebo. *Tutorial: Using Gazebo plugins with ROS*. 2014. URL: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#collapseOne.

Anexo A - Código sdf P2DX

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="p2dx">
  <static>false</static>


  <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>
  <link name="base_footprint"/>


    <link name="base_link">
      <pose>0 0 0.16 0 0 0</pose>
      <inertial>
        <mass>5.67</mass>
        <inertia>
          <ixx>0.07</ixx>
          <iyy>0.08</iyy>
          <izz>0.10</izz>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyz>0</iyz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>0.445 0.277 0.17</size>
          </box>
        </geometry>
      </collision>
      <collision name="castor_collision">
        <pose>-0.200 0 -0.12 0 0 0</pose>
        <geometry>
          <sphere>
            <radius>0.04</radius>
          </sphere>
        </geometry>
        <surface>
          <friction>
            <ode>
              <mu>0</mu>
              <mu2>0</mu2>
              <slip1>1.0</slip1>
```

```
            <slip2>1.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="visual">
      <pose>0 0 0.04 0 0 0</pose>
      <geometry>
        <mesh>
          <uri>model://pioneer2dx/meshes/chassis.dae</uri>
        </mesh>
      </geometry>
    </visual>
    <visual name="castor_visual">
      <pose>-0.200 0 -0.12 0 0 0</pose>
      <geometry>
        <sphere>
          <radius>0.04</radius>
        </sphere>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/FlatBlack</name>
        </script>
      </material>
    </visual>
  </link>
  <link name="right_wheel">
    <pose>0.1 -.17 0.11 0 1.5707 1.5707</pose>
    <inertial>
      <mass>1.5</mass>
      <inertia>
        <ixx>0.0051</ixx>
        <iyy>0.0051</iyy>
        <izz>0.0090</izz>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyz>0</iyz>
      </inertia>
    </inertial>
    <collision name="collision">
      <geometry>
        <cylinder>
          <radius>0.11</radius>
```

```
          <length>0.05</length>
        </cylinder>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>100000.0</mu>
            <mu2>100000.0</mu2>
            <slip1>0.0</slip1>
            <slip2>0.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="visual">
      <geometry>
        <cylinder>
          <radius>0.11</radius>
          <length>0.05</length>
        </cylinder>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/FlatBlack</name>
        </script>
      </material>
    </visual>
  </link>
  <link name="left_wheel">
    <pose>0.1 .17 0.11 0 1.5707 1.5707</pose>
    <inertial>
      <mass>1.5</mass>
      <inertia>
        <ixx>0.0051</ixx>
        <iyy>0.0051</iyy>
        <izz>0.0090</izz>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyz>0</iyz>
      </inertia>
    </inertial>
    <collision name="collision">
      <geometry>
        <cylinder>
```

```
          <radius>0.11</radius>
          <length>0.05</length>
        </cylinder>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>100000.0</mu>
            <mu2>100000.0</mu2>
            <slip1>0.0</slip1>
            <slip2>0.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="visual">
      <geometry>
        <cylinder>
          <radius>0.11</radius>
          <length>0.05</length>
        </cylinder>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/FlatBlack</name>
        </script>
      </material>
    </visual>
  </link>
  <joint type="revolute" name="left_wheel_hinge">
    <pose>0 0 -0.03 0 0 0</pose>
    <child>left_wheel</child>
    <parent>base_link</parent>
    <axis>
      <xyz>0 1 0</xyz>
      <use_parent_model_frame>true</use_parent_model_frame>
    </axis>
  </joint>
  <joint type="revolute" name="right_wheel_hinge">
    <pose>0 0 0.03 0 0 0</pose>
    <child>right_wheel</child>
    <parent>base_link</parent>
    <axis>
      <xyz>0 1 0</xyz>
```

```
      <use_parent_model_frame>true</use_parent_model_frame>
    </axis>
  </joint>


  <!-- *** odometry *** -->
 <!-- Controls the differential drive robot -->
     <plugin name='diff_drive' filename='libgazebo_ros_diff_drive.so'>
       <ros>
         <namespace>/demo</namespace>
         <remapping>cmd_vel:=cmd_vel</remapping>
         <remapping>odom:=odom</remapping>
       </ros>

       <!-- wheels -->
       <left_joint>left_wheel_hinge</left_joint>
       <right_joint>right_wheel_hinge</right_joint>

       <!-- kinematics -->
       <wheel_separation>0.26</wheel_separation>
       <wheel_diameter>0.2</wheel_diameter>

       <!-- limits -->
       <max_wheel_torque>20</max_wheel_torque>
       <max_wheel_acceleration>1.0</max_wheel_acceleration>

       <!-- output -->
       <publish_odom>true</publish_odom>
       <publish_odom_tf>true</publish_odom_tf>
       <publish_wheel_tf>true</publish_wheel_tf>

       <odometry_frame>odom</odometry_frame>
       <robot_base_frame>base_footprint</robot_base_frame>
       <odometry_source>0</odometry_source>
     </plugin>



  <!-- *** IMU *** -->
  <link name = "imu">

      <gravity>true</gravity>
      <sensor name="imu_sensor" type="imu">
              <always_on>true</always_on>
              <update_rate>100</update_rate>
```

```
            <visualize>true</visualize>
            <topic>imu</topic>
      <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
        <initial_orientation_as_reference>false
        </initial_orientation_as_reference>
            <ros>
              <namespace>/imu</namespace>
              <remapping>~/out:=data</remapping>
            </ros>
      <bodyName>imu_link</bodyName>
      <updateRateHZ>10.0</updateRateHZ>
      <gaussianNoise>0.0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <output_type>sensor_msgs/Imu</output_type>
      <frame_name>imu_link</frame_name>
    </plugin>
</sensor>
</link>


<!-- *** LIDAR *** -->
<link name="lidar">
  <pose>0.15 0 0.3 0 0 -2</pose>

  <sensor name="rplidar" type="ray">
    <always_on>true</always_on>
    <update_rate>30</update_rate>
    <visualize>true</visualize>
    <topic>scan</topic>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples>
          <resolution>1</resolution>
          <min_angle>0</min_angle>
          <max_angle>4.19</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>5.6</max>
        <resolution>0.2</resolution>
      </range>
    </ray>
```

```
      <plugin name="laser" filename="libgazebo_ros_ray_sensor.so">
        <ros>
          <argument>~/out:=scan</argument>
        </ros>
        <output_type>sensor_msgs/LaserScan</output_type>
        <frame_name>laser</frame_name>
      </plugin>
    </sensor>
    <visual name="visual">
      <geometry>
        <mesh>
          <uri>model://hokuyo/meshes/hokuyo.dae</uri>
        </mesh>
      </geometry>
    </visual>
</link>


    <joint name="base_joint" type="fixed">
      <parent>base_footprint</parent>
      <child>base_link</child>
      <pose>0 0 0 0 0 0</pose>
    </joint>


    <joint type="fixed" name="laser_joint">
      <child>lidar</child>
      <parent>base_link</parent>
    </joint>

    <joint type="fixed" name="imu_joint">
            <child>imu</child>
            <parent>base_link</parent>
    </joint>


  </model>
</sdf>
```

Anexo B - Código Gazebo Move

```
#include <memory>
#include <chrono>
#include <functional>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
#include "geometry_msgs/msg/twist.hpp"
#include "termios.h"
#include <unistd.h>

#include <iostream>
using namespace std::chrono_literals;

#define KEYCODE_RIGHT 0x43
#define KEYCODE_LEFT 0x44
#define KEYCODE_UP 0x41
#define KEYCODE_DOWN 0x42
#define KEYCODE_SPACE 0x20

int kfd = 0;
char c;
struct termios cooked,raw;


class RobotMove: public rclcpp::Node{
    private:

        rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr pub;
        rclcpp::TimerBase::SharedPtr timer;

        double linear,angular;
    public:
        RobotMove()
        : Node("gazebo_move"){
            pub = this->create_publisher
            <geometry_msgs::msg::Twist>("/demo/cmd_vel",10);
            timer = this->create_wall_timer
            (10ms, std::bind(&RobotMove::publish, this));

            tcgetattr(kfd, &cooked);
                memcpy(&raw, &cooked, sizeof(struct termios));
                raw.c_lflag &=~ (ICANON | ECHO);
                // Setting a new line, then end of file
                raw.c_cc[VEOL] = 1;
```

```
        raw.c_cc[VEOF] = 2;
        tcsetattr(kfd, TCSANOW, &raw);

    puts("Reading from keyboard");
        puts("---------------------------");
        puts("Use arrow keys to move the robot.");
}

void publish(){
  geometry_msgs::msg::Twist msg;
  if(read(kfd, &c, 1) < 0){
    perror("read():");
    exit(-1);
 }
    switch(c){
            case KEYCODE_RIGHT:
                    RCLCPP_INFO(this->get_logger(),"Move to right")
                    angular=-0.2;
                    linear=0.0;
            break;
            case KEYCODE_LEFT:
                    RCLCPP_INFO(this->get_logger(),"Move to left");
                    angular=0.2;
                    linear=0.0;
            break;
            case KEYCODE_UP:
                    RCLCPP_INFO(this->get_logger(),"Move to up");
                    linear=0.2;
                    angular=0.0;
            break;
            case KEYCODE_DOWN:
                    RCLCPP_INFO(this->get_logger(),"Move to down");
                    linear=-0.2;
                    angular=0.0;
            break;
            case KEYCODE_SPACE:
                    RCLCPP_INFO(this->get_logger(),"Stop");
                    linear=0;
                    angular=0;
            break;

            }
     msg.linear.x = linear;
     msg.angular.z = angular;
```

```
            pub->publish(msg);
        }

        ~RobotMove(){
            tcsetattr(kfd, TCSADRAIN, &cooked);
            //To fix this part.
            //Allow the terminal show what you write on it.
        }


};

int main(int argc, char* argv[]){
    rclcpp::init(argc,argv);
    rclcpp::spin(std::make_shared<RobotMove>());
    rclcpp::shutdown();
    return 0;
}
```