UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

Traballo Fin de Mestrado

CURSO 2018/19

DESENVOLVEMENTO DUN MODELO DE SIMULACIÓN EN GAZEBO DUN ROBOT MÓBIL BASADO EN SMARTPHONE

Mestrado en Enxeñaría Industrial

ALUMNO

David Casal Freire

TITORES

Francisco Javier Bellas Bouza

Alejandro Romero Montero

DATA

XULLO 2019

TÍTULO E RESUMO

Título

Desenvolvemento dun modelo de simulación en Gazebo dun robot móbil basado en smartphone.

Resumo

O uso de modelos de simulación é unha práctica amplamente estendida no campo da robótica debido ás vantaxes que ofrece fronte á utilización dun robot real, tales como axilizar a realización de probas, eliminar os posibles fallos do hardware, dispoñer do espazo físico necesario, dispoñer das unidades de robots suficientes, redución de costes, etc. Estes modelos de simulación son de gran utilidade, tanto en labores de investigación como na docencia dentro deste campo, especialmente en niveis superiores de educación. Por outro lado, cada vez son máis empregados os estándares de ROS (Robot Operating System) para a programación de robots, principalmente no ámbito da investigación, sendo de gran interese contar con simuladores que permitan traballar con este sistema.

Este Traballo Fin de Mestrado ten como obxectivo desenvolver un modelo de simulación do robot Robobo para o simulador de código aberto Gazebo, que poida ser programado dende os estándares de ROS e que ofreza un comportamento afín ao ofrecido polo modelo real.

Título

Desarrollo de un modelo de simulación en Gazebo de un robot móvil basado en smartphone

Resumen

El uso de modelos de simulación es una práctica ampliamente extendida en el campo de la robótica debido a las ventajas que ofrece frente a la utilización de un robot real, tales como agilizar la realización de pruebas, eliminar los posibles fallos del hardware, disponer del espacio físico necesario, disponer de las unidades de robots suficientes, reducción de costes, etc. Estos modelos de simulación son de gran utilidad, tanto en labores de investigación como en la docencia dentro de este campo, especialmente en niveles superiores de educación. Por otro lado, cada vez son más empleados los estándares de ROS (Robot Operating System) para la programación de robots, principalmente en el ámbito de la investigación, siendo de gran interés disponer de simuladores que permitan trabajar con este sistema.

Este Trabajo Fin de Máster tiene como objetivo desarrollar un modelo de simulación del robot Robobo para el simulador de código abierto Gazebo, que pueda ser programado desde los estándares de ROS y que ofrezca un comportamiento afín al ofrecido por el modelo real.

Title

Development of a simulation model in Gazebo of a mobile robot based on smartphone

Summary

The use of simulating models is a wide extend practice in the robotics field due to the advantages it offers over the use of real robots, such as speeding up the execution of tests, avoid possible hardware failures, having the necessary physical space, having enough units of robot, reducing costs, etc. These simulation models are very useful, both in research work and in teaching, especially at higher levels of education. On the other hand, ROS (Robot Operating System) standards are increasingly used for programming robots, mainly in research, being of great interest to have simulators to allow working with this system.

This Master's Thesis aims to develop a simulation model of the Robobo robot for the Gazebo open source simulator, which can be programmed from the ROS standards and which offers a behavior like the offered by the real model.

Escola Politécnica Superior

TRABALLO FIN DE MESTRADO

CURSO 2018/19

DESENVOLVEMENTO DUN MODELO DE SIMULACIÓN EN GAZEBO DUN ROBOT MÓBIL BASADO EN SMARTPHONE

Mestrado en Enxeñaría Industrial

Documento

MEMORIA

ÍNDICE DO DOCUMENTO MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE TÁBOAS

1 INTRODUCIÓN

O presente TFM enmárcase no campo da robótica educativa e de investigación. En concreto, céntrase no desenvolvemento dun modelo en simulación para un robot de nova xeración que facilite o proceso da ensinanza de conceptos robóticos avanzados en estudos universitarios. Así mesmo, o devandito modelo será de gran utilidade para os investigadores do campo da robótica autónoma e os sistemas multi-robot.

O robot para o cal se levará a cabo o modelo en simulación denomínase Robobo [1]. A súa principal característica é que se compón de dous elementos: unha base móbil de baixo custo e tamaño reducido, e un smartphone que se acopla á base, dotándoa de capacidades avanzadas de sensorización, cómputo e comunicacións.

Con esta configuración, Robobo constitúe unha ferramenta educativa de gran utilidade en estudos de nivel universitario en que se impartan conceptos de robótica avanzados, xa que os alumnos poden utilizar cámaras, sensores táctiles, recoñecemento de voz, produción de sonido, etc.; a un prezo accesible para calquera centro educativo. Estes motivos son válidos así mesmo para o uso en tarefas de investigación onde se requiran varias unidades do robot a baixo custo.

O robot prográmase con diferentes linguaxes, dependendo do nivel de coñecementos do alumno, sendo un dos aspectos máis interesantes, para o nivel educativo superior, a súa compatibilidade con ROS [2], cuxas características serán detalladas máis adiante nesta memoria.

Na actualidade, Robobo xa está en uso en diversas universidades europeas, que mostraron o seu interese en contar cun modelo de simulación do robot compatible con Gazebo, o simulador oficial de ROS. Aínda que o custo é baixo e se poidan ter varias unidades dispoñibles, o uso dun simulador ten varias vantaxes a este nivel: por un lado, acelera a fase de proba dos algoritmos, xa que o uso de robots reais durante o desenvolvemento é pouco práctico, xa que poden resultar danados. Se ademais pensamos en tarefas que requiran varios robots, a súa posta en marcha e funcionamento tornase complexo, polo que retarda aínda máis o avance. Por outro lado, aínda que o número de robots na aula sexa alto, ter un modelo en simulación permite que os alumnos poidan realizar traballo de maneira autónoma fóra das aulas.

En consecuencia, este TFM ten como principal obxectivo o que se detalla no seguinte capítulo.

Estrutura da memoria

A memoria do presente traballo está estruturada en oito capítulos, sendo o primeiro deles esta ***Introdución***:

- Segundo capítulo: ***Obxectivos***. Neste capítulo indícanse os obxectivos principais deste TFM.
- Terceiro capítulo: ***Fundamentos tecnolóxicos***. Descríbese o robot Robobo, o sistema operativo para robots (ROS) e o seu simulador oficial Gazebo, así como outros conceptos importantes para o desenvolvemento deste traballo.
- Cuarto capítulo: ***Antecedentes***. Recóllense diferentes modelos de robots que xa contan con modelos de simulación utilizados tanto en labores de investigación como de educación, xustificando a importancia de contar cun modelo de simulación para Robobo.
- Quinto capítulo: ***Requisitos de deseño***. Neste capítulo enuméranse as bases de partida establecidas polos investigadores do GII que condicionan as solucións técnicas deste traballo.
- Sexto capítulo: ***Desenvolvemento do modelo de simulación***. Faise unha descrición detallada do desenvolvemento técnico do traballo, da metodoloxía seguida e os detalles da implementación do mesmo.
- Sétimo capítulo: ***Probas e validación***. Coméntase os resultados obtidos do desenvolvemento do modelo e as probas realizadas para confirmar a utilidade do mesmo.
- Oitavo capítulo: ***Conclusións e traballo futuro***. Neste derradeiro capítulo coméntase o resultado xeral do traballo ademais dos labores que deben ser realizados posteriormente.

2 OBXECTIVOS

O principal obxectivo deste Traballo Fin de Mestrado é levar a cabo o desenvolvemento dun modelo en simulación para o robot educativo Robobo no simulador Gazebo e que poida ser programado mediante ROS do mesmo xeito que o robot real. Quere isto dicir que un mesmo programa ten que poder ser executado tanto no modelo real como no modelo simulado, sen a necesidade de realizarlle ningún cambio.

O modelo de simulación debe comportarse de igual maneira que o robot real. En consecuencia, establécense os seguintes sub-obxectivos:

- Modelo físico do robot, que integrará tanto os elementos de visualización como os elementos de cálculo dinámico.
- Caracterización do comportamento dos catro motores que compoñen a base robótica.
- Implementación dos encoders dos motores.
- Deseño e caracterización dos oito sensores infravermellos que leva integrados o robot Robobo.
- Implementación da cámara frontal do smartphone.

Finalmente realizarase unha proba global na que se compare o modelo de simulación co robot real e se demostre que este último poida ser substituído polo devandito modelo.

3 FUNDAMENTOS TECNOLÓXICOS

Neste capítulo faise unha breve introdución ás bases teóricas a partir das cales se desenvolve o presente traballo académico. Describirase o robot Robobo, falarase do sistema operativo para robots ROS, do simulador Gazebo e do programa informático Blender, así como do formato de descrición robótica SDF.

3.1 Robobo

Robobo [1], [3] é unha base robótica móbil que se combina cun smartphone (Figura 1) para formar un robot educativo, desenvolvido por MINT, unha spin off da universidade da Coruña con ampla experiencia no deseño e fabricación de solucións robóticas industriais así como na educación universitaria nos campos da enxeñaría e a robótica.

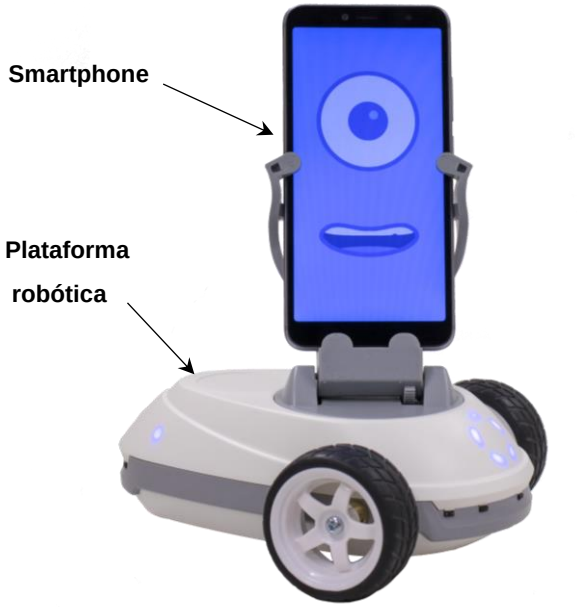


Figura 1. Robot Robobo

3.1.1 Hardware

A base robótica está formada por cinco pezas fundamentais (Figura 2): o corpo, dúas rodas, o pan e o tilt; permitindo o movemento relativo entre elas grazas a catro motores de corrente continua.

- **Corpo:** É a peza que fai de soporte de todos os compoñentes, dentro dela alóxanse todos os compoñentes electrónicos do robot. A versión comercial está conformada en plásticos que non permiten ver o interior, pero si distinguir as diferentes luces de cores que ofrecen os LEDs internos. Na súa parte traseira inferior dispón dun deslizador de teflón, que fai de terceiro punto de apoio e permítelle á base desprazarse por practicamente calquera superficie de interior.
- **Rodas**: Robobo dispón de dúas rodas na parte frontal, coa finalidade de desprazarse sobre superficies horizontais e incluso inclinadas. Cada roda conta co seu motor independente, para poder controlar cada unha por separado, de

forma que se poidan realizar xiros para efectuar a traxectoria desexada coa base robótica.

- **Pan**: Plataforma integrada na parte superior do corpo encargada de dar soporte á peza de suxeición do teléfono intelixente (tilt). Esta peza permite orientar o teléfono arredor do eixe vertical, cun radio de xiro que abarca dende os 11 ata os 344 grados.
- **Tilt**: Peza de suxeición do teléfono situada encima do pan. Está conformada en polímero flexible, o que proporciona un agarre firme para diferentes tamaños de teléfonos, sendo compatible coa maioría de dispositivos presentes no mercado. O tilt permite o movemento arredor do eixe horizontal, podendo inclinar o teléfono dende os 110 grados ata 5 grados, onde o dispositivo se encontraría nunha posición case horizontal e coa pantalla mirando cara arriba.

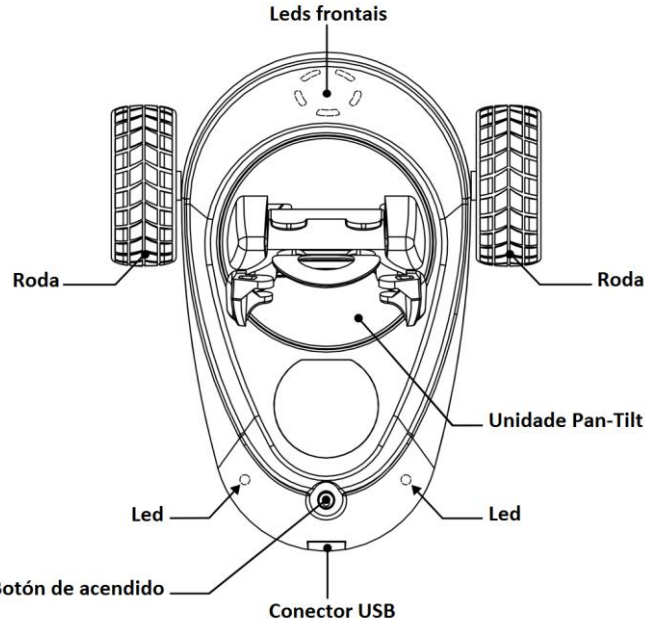


Figura 2. Principais compoñentes da base robótica Robobo

Dentro da parte electrónica cabe resaltar os catro motores que permiten os movementos do robot e os sensores infravermellos.

- **Motores**. A plataforma robótica incorpora catro motores de corrente continua do modelo TTF-N20VA-09220, equipados con caixa redutora e encoder. Tanto os motores das rodas como o do pan teñen unha redución de 150:1, ofrecendo un par nominal de 0.22 kg·cm e unha velocidade nominal de 81 rpm. Para o caso do tilt, o motor posúe unha redución de 1000:1 para que sexa capaz de levantar o teléfono dende unha posición horizontal, ofrecendo un par nominal de 1 kg·cm e unha velocidade nominal de 12 rpm. No caso particular do pan e o tilt, contan cunha relación de saída de 68:10 e 54:25 respectivamente, a causa da relación de engrenaxes. Os encoders permiten coñecer a posición de cada motor en todo momento.
- **Sensores infravermellos**. O corpo do robot está equipado con oito sensores infravermellos, cinco na parte frontal e tres na parte traseira. Estes dispositivos son empregados para a detección de obxectos a distancias inferiores a 30 cm. Os sensores están orientados de forma horizontal agás dous deles da parte frontal e un da parte traseira, que están inclinados aproximadamente 15 grados cara abaixo, deste xeito é posible detectar o chan e evitar que o robot caia dunha

mesa, por exemplo. Na Figura 3 pode verse a distribución dos sensores infravermellos da base Robobo.

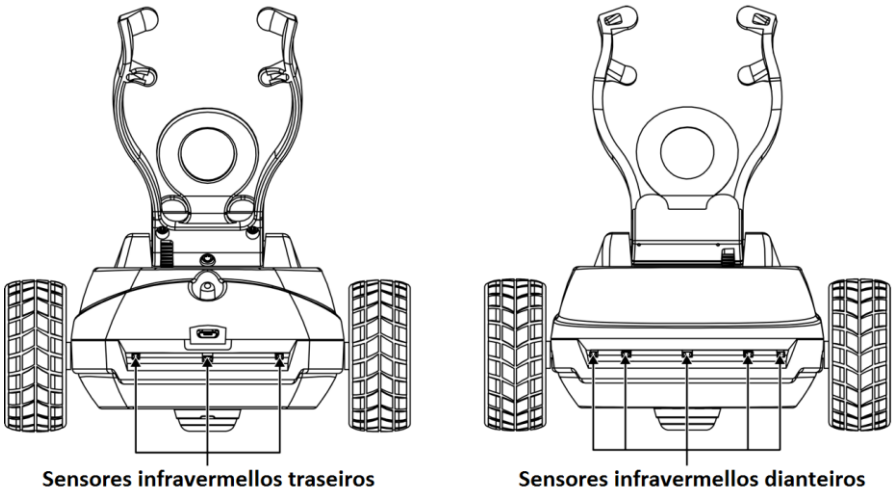


Figura 3. Disposición dos sensores infravermellos na base robótica

- **Luces LED**. Na parte frontal do corpo do robot sitúanse cinco luces LED (Light Emitting Diode) formando un círculo e na parte traseira sitúanse outros dous, un a cada lado do corpo. Estas luces serven para informar do estado do robot ao mesmo tempo que poden ser programadas de forma individual para iluminarse en oito cores diferentes.

Ademais dos elementos citados anteriormente, a plataforma robótica conta cun botón de acendido e apagado e unha batería de LiPo de 5000mAh recargable a través dun porto micro USB, ofrecendo unha autonomía aproximada de entre 8 e 10 horas, dependendo do uso.

A maiores dos sensores e actuadores que incorpora a base, hai que engadir os que posúen comunmente os teléfonos intelixentes como xiroscopio, acelerómetro, sensor de luz, cámara, pantalla táctil, altofalante ou micrófono; todos eles accesibles dende os módulos de programación. Esta é a principal potencialidade deste robot, xa que as funcionalidades que lle atribúe un smartphone elevan o conxunto a un nivel superior que os seus competidores, e fano óptimo para o seu uso uso na ensinanza da robótica intelixente.

As capacidades de procesamento de Robobo dependen do smartphone empregado, xa que en certo modo é o cerebro do robot. O dispositivo móbil enlázase coa base a través da conexión bluetooth e, ao mesmo tempo, conéctase cun ordenador ou con outros dispositivos a través dunha conexión WiFi.

3.1.2 Software

O desenvolvemento do software de Robobo pódese organizar en bloques de baixo nivel (firmware) e de alto nivel (arquitectura do software):

***Baixo nivel**:* a Figura 4 mostra a arquitectura do hardware xeral da base robótica, onde se representan as principais rutas de conexión. O núcleo do sistema é o microcontrolador PIC32, que se encarga de controlar os diferentes circuítos integrados, sensores e actuadores da base. Ademais, o PIC32 xestiona as comunicacións entre o microcontrolador e o módulo de bluetooth, o cal envía información ao smartphone e recibe do mesmo os comandos para os motores.

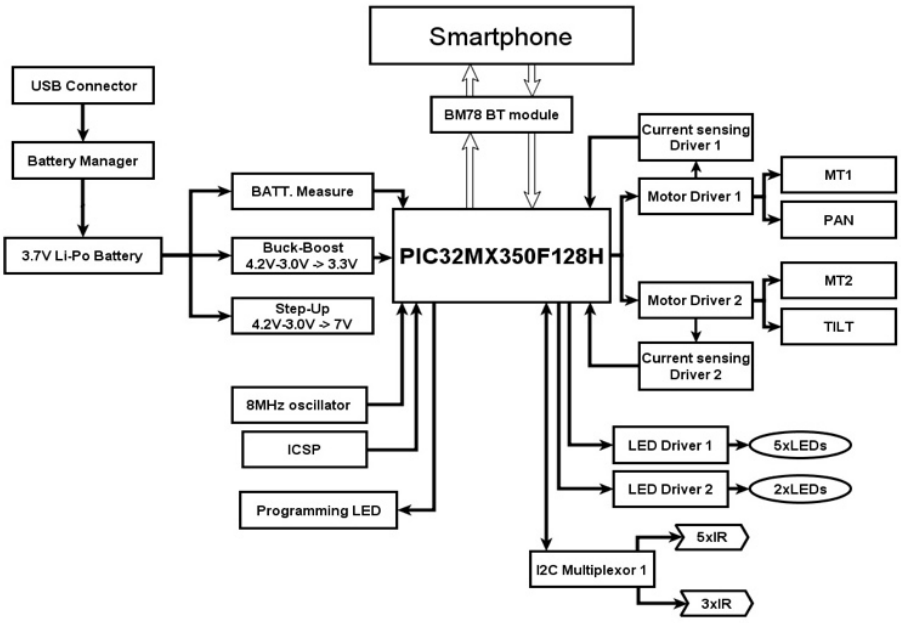


Figura 4. Arquitectura electrónica do Robobo

Alto nivel: Robobo conta cunha arquitectura de software completamente modular, mostrada na Figura 5. Basease no concepto do módulo Robobo e unha biblioteca lixeira, denominada Robobo Framework, que proporciona os mecanismos esenciais para administrar e executar os devanditos módulos nun teléfono intelixente Android. Na parte superior desta biblioteca pódense cargar dous tipos diferentes de módulos: módulos funcionais (en laranxa na Figura 5) e unha serie de módulos proxy (ros-proxy e remote-proxy na Figura 5).

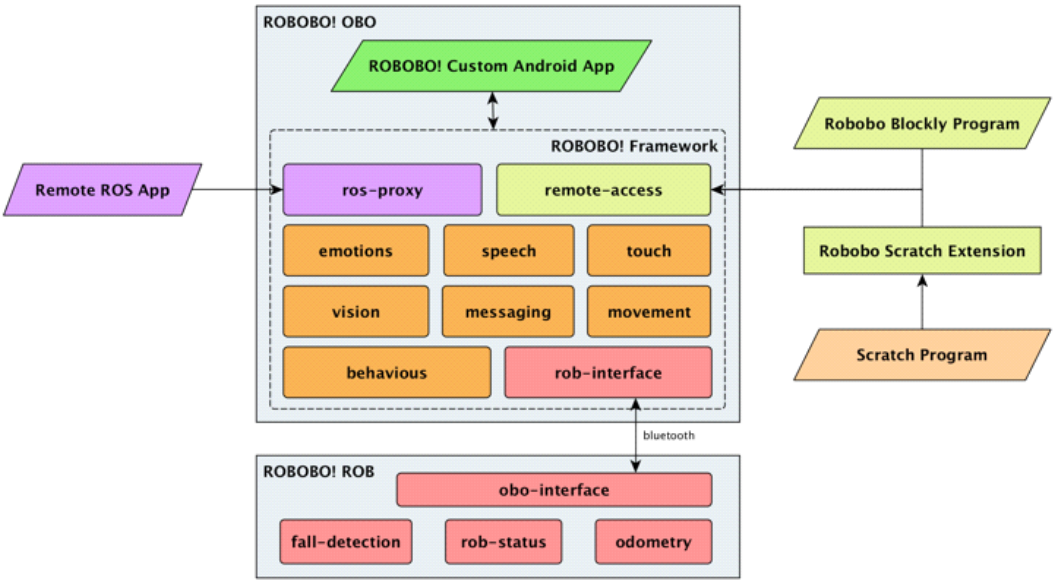


Figura 5. Diagrama de bloques da arquitectura do software do Robobo

Robobo ven por defecto cun conxunto particular destes módulos, que os usuarios avanzados poden personalizar e incluso implementar novos módulos que engadan novas funcionalidades ao robot.

Cabe sinalar que existe un módulo para conectar o *framework* Robobo e os seus módulos á base robótica. O módulo rob-interface, que se mostra en rosa na Figura 5.

Implementa o protocolo de comunicacións bluetooth da base e proporciona unha API (Aplication Programming Interface) de control para que outros módulos a utilicen.

3.1.3 Programación

En canto á programación, Robobo pode ser programado en Scratch 3, Javascript, Python e JavaAndroid, sendo tamén compatible con ROS. Deste xeito o abanico da súa utilización abrangue dende os usuarios máis inexpertos ata os usuarios máis avanzados no campo da programación.

A programación de Java é directamente complatible coa API nativa proporcionada polos diferentes módulos. Empregando Robobo Framework, os usuarios poden crear aplicacións de Android personalizadas que controlen o comportamento do robot. Estas aplicacións utilizan as interfaces de programación dos módulos de Robobo para acceder ás diferentes funcionalidades do robot e construír novos comportamentos ou resolver novas tarefas.

Para a programación de bloques, actualmente admítense dous enfoques diferentes: Scratch, e o IDE propio baseado nos bloques que utiliza Blockly. Ambos enfoques están conectados ao framework utilizando o mesmo proxy, o Protocolo de Acceso Remoto Robobo (RRAP). Este é un protocolo simple baseado en JSON, que permite o acceso remoto ás API dos módulos de Robobo.

Por último, Robobo tamén pode programarse empregando ROS, que é sistema máis empregado na ensinanza robótica a niveis de educación superior. ROS é compatible a través do módulo ros-proxy, encargado de enlazar as API nativas dos módulos cos mensaxes e temas de ROS.

3.2 ROS

As siglas ROS fan referencia a Robot Operating System [2], un framework de código aberto creado para o desenvolvemento de sofware de robótica. Consiste nunha colección de ferramentas, bibliotecas e convencións coa finalidade de simplificar os labores de creación de comportamentos robóticos complexos e robustos nunha ampla variedade de plataformas robóticas.

O software estruturase mediante un gran número de pequenos programas (nodos) que se comunican entre si a través de mensaxes. Esta estrutura permite a creación de módulos xenéricos aplicables a diferentes clases de hardware e sofware robótico, de xeito que se facilita o uso compartido de códigos para a súa reutilización entre a comunidade robótica global.

ROS non é un sistema operativo, polo tanto, debe ser executado sobre plataformas baseadas en UNIX. O software de ROS utilízase maioritariamente en plataformas Linux, sendo tamén compatible con Mac OS ou Microsoft Windows.

Grazas ao repositorio de software de ROS, o programador pode acceder a multitude de paquetes que proporcionan funcionalidades de forma simple, podendo centrarse no desenvolvemento do control do robot sen preocuparse da previa implementación dos algoritmos e técnicas máis comúns.

3.2.1 Principais características

As característica máis salientables deste sistema son as seguintes:

- **Computación distribuída:** ROS proporciona ferramentas que que simplifican a implementación das canles de comunicación necesarias entre os diferentes procesos que se executan nun ou incluso diferentes computadores.

- **Reutilización de software:** ROS permite a aplicación a novos contextos de algoritmos utilizados para a resolución de tarefas comúns, como a navegación, planificación de tarefas, etc., sen a necesidade de reimplementalos de xeito particular para cada caso. Isto é posible grazas ás funcionalidades citadas a continuación:
    1. Incorporación de paquetes con versións estables dos principais algoritmos utilizados na robótica.
    2. Non impón ningún tipo de restrición ao código desenvolvido noutros frameworks.
    3. Compatibilidade entre diferentes linguaxes, xa que ROS está implementado en C++, Python, Lisp, Java e Lua.
- **Facilidade para a realización de ensaios:** ROS permite reducir a cantidade de tempo durante a realización de probas de funcionamento, e incluso prescindir nelas dun robot real, grazas aos seguintes factores:
    1. A separación dos sistemas de alto nivel, orientados aos procesado e á toma de decisións, dos de baixo nivel, encargados dos controis básicos do hardware. Isto permite a substitución do hardware e dos sistemas de baixo nivel por un simulador, permitindo a comprobación do correcto funcionamento do software de alto nivel.
    2. Facilitación da gravación e posterior reprodución de datos de sensores ou outro tipo de mensaxes, permitindo un mellor aproveitamento do tempo invertido durante a realización de probas.
- **Popularidade:** o gran número de usuarios implica unha constante actualización dos repositorios, a disposición de novos algoritmos ou funcionalidades froito de investigacións, a corrección de erros que poidan xurdir na plataforma, etc.

3.2.2 Rede de comunicación en ROS

A Wiki de ROS [4] define o seu *grafo computacional* como unha rede *peer-to-peer* entre os procesos de ROS, que procesan os datos de maneira conxunta. Cabe destacar que existe un proceso mestre, denominado “Master”, encargado de centralizar as conexións.

A rede está composta polos seguintes elementos:

- **Nodo:** proceso que realiza unha computación, escrito mediante a utilización de librerías de cliente de ROS como rospy ou roscpp. Un sistema de control robótico adoita abarcar varios nodos.
- **Master:** proporciona os servizos de rexistro e consulta ao resto do grafo computacional, permitindo a conexión entre nodos.
- **Servidor de parámetros:** forma parte do Master e permite o almacenamento de datos baixo un índice nunha localización central.
- **Mensaxe:** os nodos comunícanse entre si a través das mensaxes, que son estruturas simples de datos que admiten os tipos primitivos estándar (integer, floating point, boolean, etc.) e incluso poden crearse mensaxes personalizados.
- **Tema:** canle a través da cal os nodos poden enviar ou recibir mensaxes. Xeralmente os nodos que transmiten información reciben o nome de *publishers* e os que a leen denominanse *subscribers*. Un mesmo tema pode ter múltiples *publishers* e *subscribers*.
- **Servizo:** mecanismo que permite o sistema de comunicación solicitude-resposta que a miúdo se requiren nun sistema distribuído. Os servizos defínense mediante un par de estruturas de mensaxe, unha para a solicitude e outra para a

resposta. Na Figura 6 móstrase a diferenza entre a comunicación de dous nodos mediante un tema e un servizo.

- **Bolsa:** formato para gardar e reproducir datos de mensaxes de ROS. As bolsas son un mecanismo importante para a reutilización de datos difíciles de recompilar pero necesarios para desenrolar e probar algoritmos.

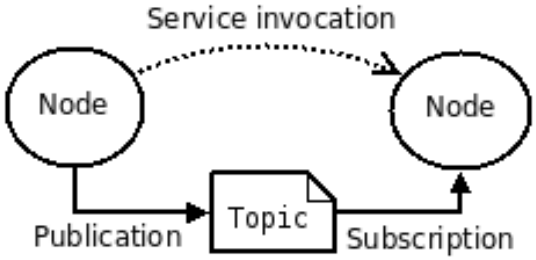


Figura 6. Sistema de comunicación entre nodos en ROS

3.2.3 Versión empregada

Para o desenvolvemento do presente traballo foi utilizada a versión ROS Kinetic, xa que é a recomendada para utilizar con Gazebo 7. A razón da utilización desta versión foi como consecuencia dos problemas xurdidos ao inicio deste TFM.

Nun primeiro momento instalouse unha versión de Gazebo 8 xunto con ROS Kinetic, sobre o sistema operativo Ubuntu 16.04, coa intención de importar os arquivos OBJ (Wavefront 3D Object File) dos que se dispoñía, xa que esta era a primeira versión de Gazebo que os soportaba. Debido a problemas á hora da visualización das mallas importadas dende Gazebo, decidiuse actualizar á versión estable mais recente, que no momento era Gazebo 9, coa que se recomendaba o uso de ROS Melodic. Neste caso foi necesario instalar o sistema operativo Ubuntu 18.04.

Finalmente, ao continuaren os problemas á hora de visualizar as mallas e, coa intención de non perder tempo neste asunto, tomouse a decisión de utilizar ROS Kinetic xunto coa versión recomendada de Gazebo instalados sobre Ubuntu 16.04.

De todas maneiras, está previsto que, no futuro, o modelo en simulación sexa adaptado a ROS 2, momento en que se pode aproveitar tamén, para realizar a migración a unha versión máis recente de Gazebo.

3.3 Gazebo

Gazebo [5] é un programa informático de sofware libre, desenvolvido maioritariamente por Open Source Robotics Foundation (OSRF). Trátase dun simulador de contornos 3D que permite simular e avaliar o comportamento dun ou máis robots sobre escenarios de diversa complexidade.

A pesar de nacer co propósito de poder simular robots en exteriores en condicións diversas, hoxe en día é tan utilizado en simulacións interiores como exteriores grazas a ofrecer a capacidade de poder crear mundos moi complexos para obter comportamentos o máis fieis posible ás condicións reais.

As principais características ou puntos fortes deste simulador cítanse a continuación:

- Permite crear e configurar mundos virtuais acorde ás necesidades do experimento.
- Capacidade para deseñar contornos e modelos robóticos propios, además da posibilidade de importar modelos xa desenvolvidos.

- Soporta a simulación dos sistemas físicos sobre corpos ríxidos, permitindo a interacción entre o robot e os diferentes elementos do contorno.
- Está integrado en ROS, que se encarga da comunicación co contorno e de onde se permite lanzar o simulador.

Gazebo utiliza por defecto ODE (Open Dynamics Engine) [6] como motor físico, podendo traballar con outros motores físicos como Bullet, Simbody ou DART.

Para a descrición dos contornos e dos modelos de simulación Gazebo utiliza o formato SDF. Cabe destacar que tamén soporta o formato URDF, utilizado en ROS para a descrición de robots. Sen embargo, desaconséllase o seu uso, xa que este formato só permite definir as propiedades dinámicas e cinemáticas dun so robot de forma illada, ademais de non poder especificar a posición do robot dentro do mundo, entre outras limitacións.

O simulador conta cunha interface de editor de modelos que permite a elaboración ou modificación dos mesmos dunha maneira máis intuitiva e amigable para quen non estea demasiado familiarizado coa descrición de robots no estándar SDF. Conta ademais, con editor do mundo, mediante o cal se permite crear o contorno de simulación segundo as necesidades do usuario, coa finalidade de acadar unhas características tanto físicas como visuais, o máis parecidas posible ás reais.

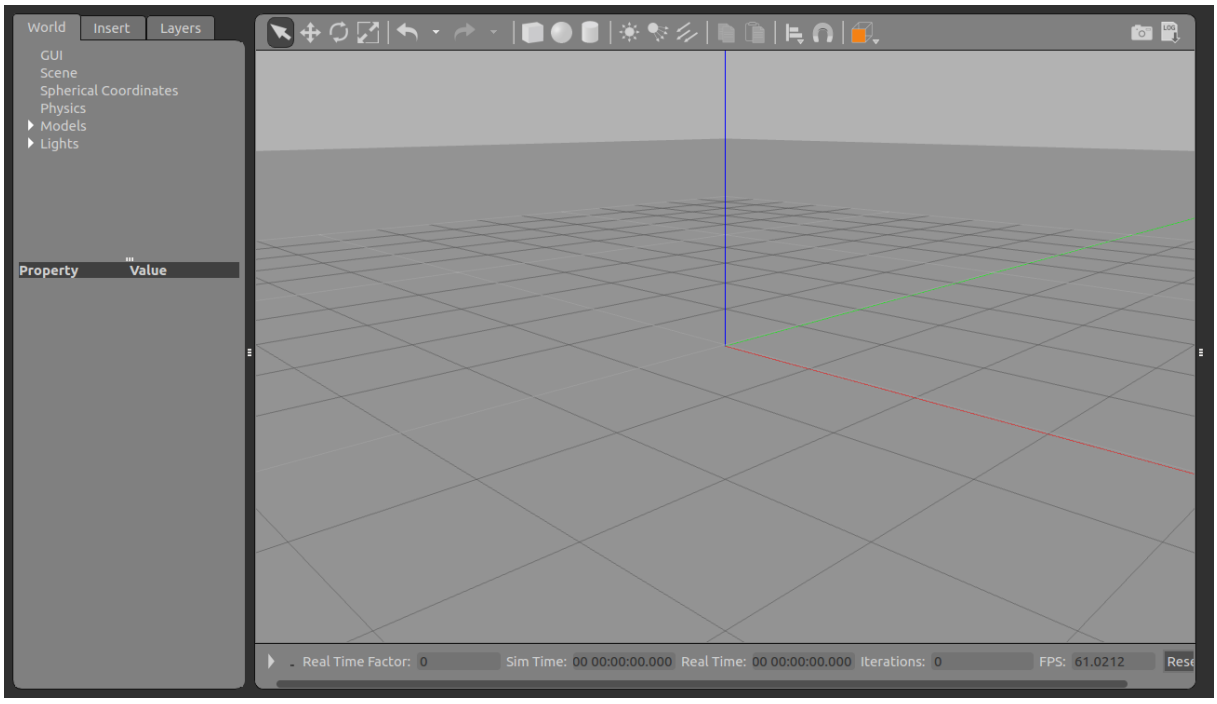


Figura 7. Interface gráfica de Gazebo durante a simulación

A interface de de usuario de Gazebo (Figura 7) permite visualizar a simulación en tempo real, ademais de poder pausala, importar modelos ou incluso modificar calquera parámetro do mundo ou modelos que forman parte da mesma.

3.4 Blender

Blender [7] é un programa informático de software libre de *creation suite* 3D dispoñible para sistemas operativos Linux, Windows e Mac OS. Entre as súas funcións destacan as de renderizado, modelado, simulación, edición de vídeo ou a creación de videoxogos.

No presente traballo a principal función empregada de dito programa é a de creación e edición de mallas 3D. As mallas son unha colección de vértices que se enlazan mediante bordes dando lugar a caras poligonais, xeralmente triángulos ou trapecios, que describen a

forma dun obxecto en tres dimensións. A partir destas mallas defínense as diferentes partes físicas do robot co fin de ser utilizadas tanto para a simulación física como para a representación dos elementos visuais.

Outro punto de importancia é que perite a exportación das mallas a arquivos COLLADA (.dae), único formato soportable por Gazebo ata a versión 8.2.

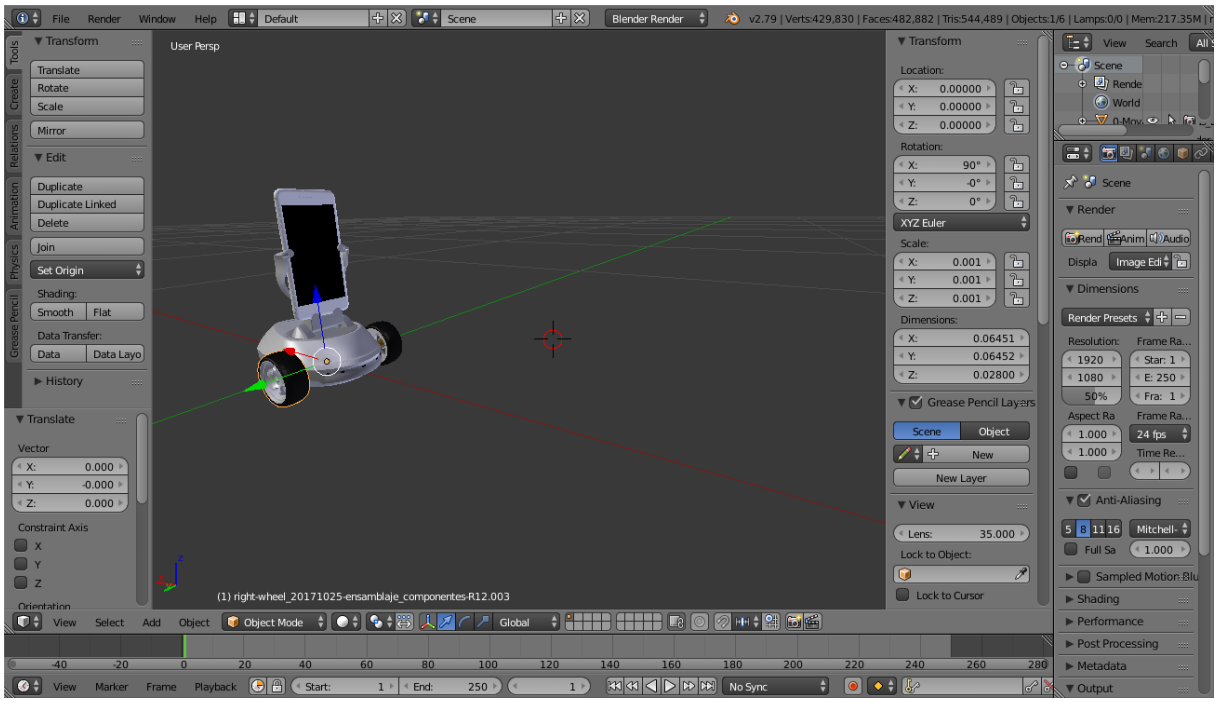


Figura 8. Interface gráfica do programa Blender

Na Figura 8 pode verse a interface gráfica de Blender durante a manipulación de mallas de Robobo.

3.5 SDF

O SDF [8], das siglas en inglés Simulation Description Format, é un formato XML que describe obxectos e contornos para a simulación de robots. Orixinalmente foi desenvolvido como parte do simulador Gazebo, tendo en conta as aplicacións dos robots científicos para o seu deseño. Co tempo, o SDF converteuse nun formato estable, robusto e extensible, capaz de describir todos os aspectos dos robots, dos obxectos estáticos e dinámicos, da iluminación, do terreo e incluso da física.

A continuación mostrase unha breve descrición dos principais elementos que forman o SDF:

- **World:** encapsula unha descrición completa do contorno de simulación, incluíndo o resto de elementos.
- **Scene:** fillo dun elemento *world*. Define a aparencia do contorno mediante propiedades como a tonalidade do ceo, do chan, a representación de brétema, entre outras.
- **Physics:** fillo dun elemento world. Establece o tipo de motor físico e os parámetros utilizados durante a simulación.
- **Light:** fillo dun elemento *world*. Describe unha fonte de luz.

- **Model:** fillo dun elemento *world*. Describe un robot completo ou calquera outro obxecto físico. Xeralmente os modelos defínense en arquivos independentes do *world* para poder lanzalos en diferentes mundos.
- **Link:** fillo dun elemento *model*. Define cada parte sólida na que se divide un modelo e contén as propiedades físicas e visuais que o definen.
- **Collision:** fillo dun elemento *link*. Define a xeometría utilizada polo motor físico para o cálculo de colisións. Permite a utilización de formas puras esféricas, cilíndricas ou cuboides; ou a importación de mallas se se pretende utilizar formas máis precisas.
- **Visual:** fillo dun elemento *link*. Define a xeometría que se representa na parte gráfica da simulación. Pode coincidir coa xeometría de colisión, ainda que normalmente impórtanse mallas de maior complexidade para obter unha calidade visual superior.
- **Joint:** fillo dun elemento *model.* Define a unión entre dous *links* con propiedades cinemáticas e dinámicas.
- **Sensor:** fillo dun elemento *link* ou *joint*. Describe o tipo e as propiedades do sensor.
- **Plugin:** fillo dun elemento *world*, *model*, ou *sensor*. Un plugin é un anaco de código que se executa dinamicamente.

4 ANTECEDENTES

A día de hoxe existe unha gran variedade de modelos de simulación de robots reais educativos. A continuación describiranse brevemente algúns dos máis empregados en ensinanzas universitarias en Europa.

4.1 Turtlebot

Turtlebot é un kit robótico de baixo custo e código aberto creado en Willow Garage no 2010. Consiste nunha base robótica móbil, grazas a dúas rodas motorizadas, equipada con cámara e sensor de distancia e un Single Board Computer, como pode ser a coñecida Raspberry Pi. Este robot conta cun paquete de desenrolo de software, ademais de contar con bibliotecas con ferramentas útiles para a visualización, o control e a detección de erros. Actualmente encontrase na terceira xeración, e está dispoñible en tres versións diferentes, que se representan na Figura 9.

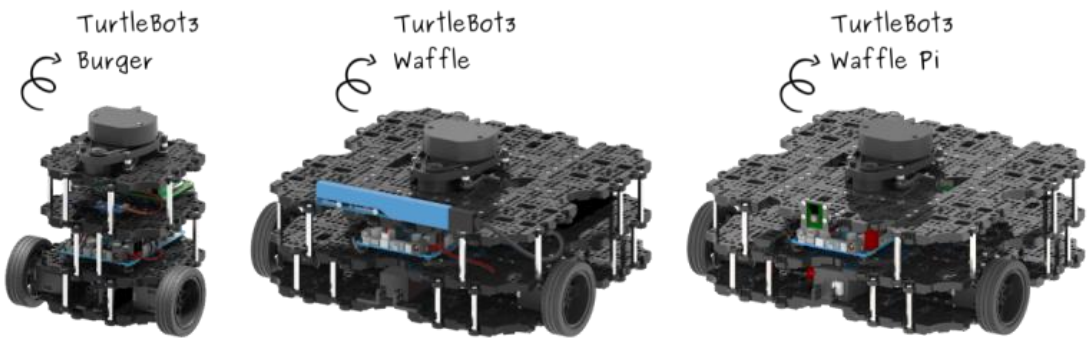


Figura 9. Formatos dispoñibles da última versión TurtleBot3.

Turtlebot conta cun modelo de simulación para Gazebo (Figura 10), que permite utilizalo en calquera das súas configuracións de hardware dispoñibles. A súa finalidade é poder ser utilizado en labores de investigación, podendo levar a cabo experimentos como o realizado por Lei Tai e Ming Liu [9], onde utilizaron dito modelo de simulación para o desenrolo dun método de aprendizaxe para a exploración de contornas por robots a partir dun único sensor de profundidade, baseándose no marco de traballo Deep Q-Network. A simulación de Turtlebot mediante Gazebo foi utilizada en diversos traballos de investigación en labores como o adestramento de robots [10], navegación robótica [11] [12] ou elaboración de sistemas de control [13].

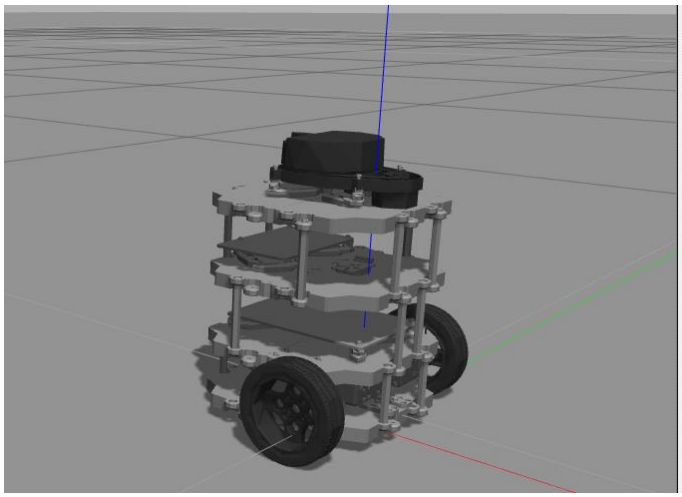


Figura 10. Modelo de simulación do Turtlebot3

4.2 Khepera

Khepera é un clásico entre os robots de dúas rodas diferenciais, desenvolvido dende principios dos 90 no LAMI (Laboratorio de Microinformática) da Escola Politécnica Federal de Lausana, en adiante EPFL. Nado inicialmente para o seu propio uso dentro da EPFL, o Khepera tivo verdadeiro éxito nas universidades a nivel mundial e vendendo unidades a máis de mil laboratorios de investigación. Froito deste proxecto nace a empresa que lle dá soporte, K-Team, fundada no 1995 como unha startup do EPFL co obxectivo de desenvolver e comercializar robots dedicados á educación.

Figura 11. Vista superior e inferior do Khepera IV

Actualmente o Khepera encóntrase na súa cuarta versión, con prestacións e dimensións superiores aos modelos iniciais. Trátase dun robot compacto deseñado para labores de investigación en interiores, requirindo de pouco espazo para ser usado en case calquera aplicación como navegación, intelixencia artificial, demostración, etc. Na parte superior incorpora un sistema de buses de extensión que lle proporcionan unha gran modularidade ao permitir implementar un gran número de configuracións. Algunhas das características técnicas máis salientables encóntranse na Táboa 1 e o seu prezo rolda os 2350 €.

Procesador	ARM Cortex-A8 de 800 MHz , DSP C64x e microcontrolador adicional para a administración de periféricos
Memoria	RAM: 512 mb; Flash: 512 mb + 4Gb para datos
Motores	2 motores paso a paso con redutora e encoder incremental
Sensores	12 sensores infravermellos, sensor de luz ambiental, 4 sensores ultrasónicos, acelerómetro, xiroscopio, micrófono e cámara
Batería	LiPo de 3400 mAh; Autonomía de hasta 7 horas
Dimensións	Diámetro: 140 mm; Altura: 58 mm; Peso: 540 g
Conectividade	USB, WiFi e Bluetooth
Programación	Compilador GNU C/C++ e Python 2.7.9

Táboa 1. Características técnicas Khepera IV

O Khepera so conta con versión oficial no simulador Webots (Figura 12).

Figura 12. Modelo de simulación do robot Khepera IV en Webots

4.3 e-puck

O e-puck [14] é un pequeno robot moi similar á primeira versión do Khepera, desenrolado tamén, na EPFL dende o 2004. Trátase dun robot deseñado para a educación a nivel universitario, coa intención de ser utilizado nunha ampla variedade de actividades de ensinanza. O seu pequeno tamaño faino propicio para realizar ensaios en espazos reducidos, como pode ser enriba dunha mesa ou dun escritorio. Na Figura 13 dáse conta das reducidas dimensións deste robot.

Figura 13. Unidade do robot e-puck

No 2018 sae a nova versión e-puck2, mellorando as funcionalidades da versión anterior e incorporando unha maior variedade de sensores. O prezo de mercado deste robot educativo rolda os 700 €, que o sitúa como un dos aparatos máis económicos da súa categoría. Na Táboa 2 amósanse as características principais deste robot.

Dimensións	Diámetro: 75 mm; Altura: 45 mm ; Peso: 130 g
Batería	LiPo 1800 mAh recargable por USB. 3 horas de autonomía
Procesador	32-bit STM32F407 @ 168 MHz (210 DMIPS), DSP e FPU, DMA
Memoria	RAM: 192 KB; Flash: 1024 KB
Motores	2 motores paso a paso con redución 50:1
Sensores	8 sensores infravermellos, acelerómetro, xiroscopio, magnetómetro, cámara, 4 micrófonos omnidireccionais
Conectividade	USB, Bluetooth, BLE, WiFi

Táboa 2. Características técnicas e-puck2

O e-puck está dispoñible en varios simuladores, tanto 3D como 2D. Ditos simuladores son Webots (Figura 14), Enki, V-REP e ARGoS.

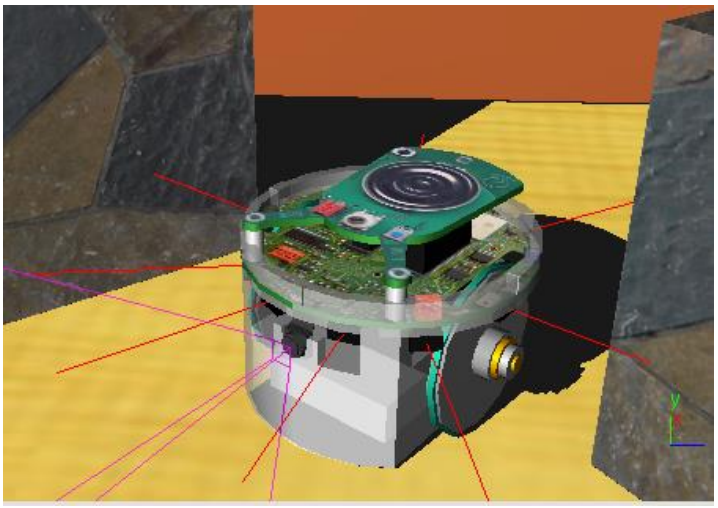


Figura 14. Modelo de simulación do robot e-puck en Webots

Este dispositivo non conta con modelo oficial para o simulador Gazebo, probablemente por non admitir a súa programación mediante ROS, sen embargo, é posible encontrar algún repositorio cun modelo do e-puck desenrolado por afeccionados á robótica para este simulador.

4.4 Lego Mindstorms EV3

Lego Mindstorms é unha liña de robótica para nenos fabricada pola compañía danesa LEGO dende 1998. O bloque EV3 (Figura 15) constitúe a terceira xeración da liña Lego Mindstorms e véndese xunto cun kit de pezas que permite fabricar de forma sinxela robots de pequena escala para fins didácticos. Entre os compoñentes deste kit pódense encontrar servomotores, sensores, a unidade central de procesamento e módulos de comunicación electrónica, ademais dos compoñentes mecánicos necesarios para o ensamblaxe como engrenaxes, correas de transmisión ou ladrillos Lego, entre outros.

Figura 15. Bloque programable Lego Mindstorms EV3

O bloque EV3 ven preparado para ser programado na súa propia linguaxe de programación, baseada en LabView, pero esta opción está mais enfocada a usuarios pouco avanzados no campo da robótica. A niveis de educación superior acostumase a programar en Python mediante a instalación da imaxe do MicroPython EV3 que o propio fabricante proporciona. Tamén é posible a súa programación mediante RobotC, que é unha linguaxe de programación para robótica baseada en C e cun entorno de desenvolvemento sinxelo de empregar.

A plataforma Robot Virtual Worlds [15] ofrece un simulador deste robot para ser programado mediante a linguaxe RobotC.

Dende o Grupo Integrado de Enxeñaría (GII) da Universidade da Coruña creouse un modelo de simulación en Gazebo da configuración “Lego Mindstorms Education EV3 Core Set (45544) Driving Base” [16], coa intención de poder ser utilizado na realización das prácticas da materia de robótica, presente no Grao en Enxeñaría Informática. A calidade da definición gráfica do modelo pódese apreciar na Figura 16.

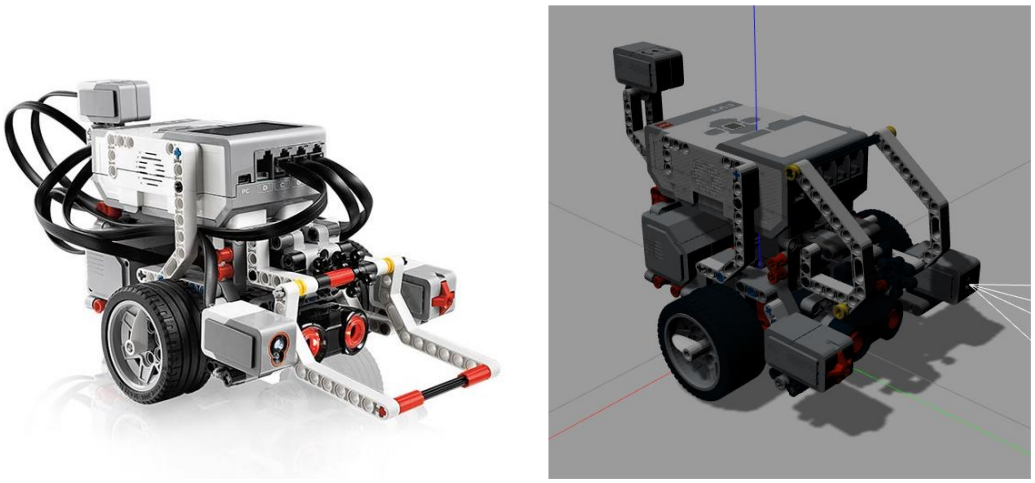


Figura 16. Modelo real fronte o modelo en simulación do Lego Mindstorms

4.5 Nao

Nao [17] é un robot humanoide autónomo e programable desenvolvido por Aldebaran Robotics, unha compañía de robótica francesa establecida en París. Está composto por multitude de sensores, motores e un software pilotado por un sistema operativo feito a medida baseado en Linux, o NAOqi OS.

Conta cunha versión académica desenvolvida para universidades e laboratorios para fins de investigación e educación. O robot Nao xa vai pola sexta xeración, chamada NAO[6], que foi lanzada no 2018 (Figura 17). Este robot humanoide conta cunha serie de capacidades que permiten crear unha interacción natural con usuarios humanos.

Figura 17. Robot humanoide NAO[6]

Entre as súas características destacan os 25 graos de liberdade de movemento e a cantidade de sensores, repartidos entre a cabeza, os pés e as mans. A comunicación con outros dispositivos realízaa mediante conexión WiFi, e pode conectarse a un ordenador persoal mediante conexión Ethernet.

Entre as súas desvantaxes está a capacidade de procesamento limitada e una pobre autonomía que non supera os 30 minutos de operación.

Dende a Wiki de ROS están dispoñibles varias plataformas de simulación para este robot, sendo Gazebo unha destas opcións. Este modelo de simulación foi empregado en

numerosos traballos de investigación tales como a comparación con outros simuladores [18], ou o desenvolvemento de novas ferramentas de programación [19]. O modelo de Nao en simulación tamén é utilizado para a docencia en estudos de nivel universitario no eido da robótica [20]. Na Figura 18 móstrase un grupo de Nao realizando unha simulación multi-robot.

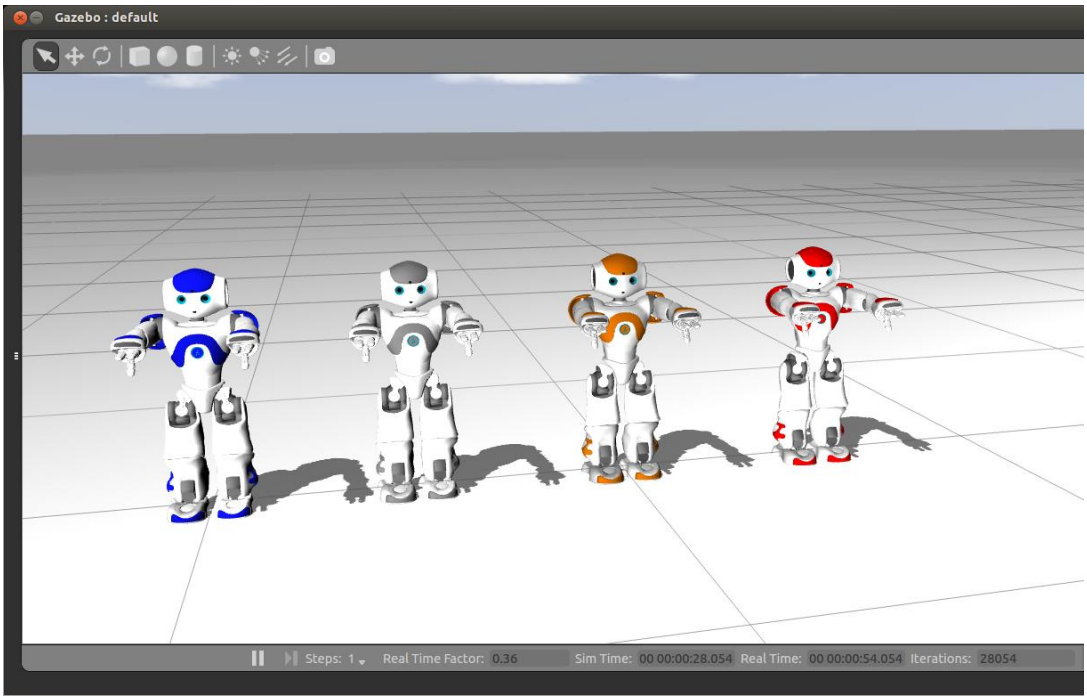


Figura 18. Simulación multi-robot en Gazebo utilizando Nao

4.6 Conclusión

Como se puido apreciar nas seccións anteriores deste capítulo, practicamente todos os robots educativos contan cun modelo de simulación, que permite realizar a maioría dos labores que no robot real pero na ausencia do mesmo. Ademais, todos os simuladores comentados foron empregados tanto en labores de investigación como na docencia, o que confirma a utilidade dos mesmos.

Gran parte dos robots comentados contan con modelo de simulación para Gazebo, o que os fai compatibles con ROS. No caso de empregar outros simuladores, a maior parte deles contan con APIs ou plugins que os fan compatibles con ROS. Isto demostra a utilidade deste sistema para a programación de robots.

Queda patente, polo tanto, que mediante o desenvolvemento dun modelo de simulación de Robobo que sexa compatible con ROS, obteríase unha ferramenta de gran potencial dentro do campo da robótica.

5 REQUISITOS DE DESEÑO

A continuación descríbense os requisitos a ter en conta na elaboración do modelo de simulación de Robobo en Gazebo realizado no presente traballo.

- Os comportamentos físicos do modelo de simulación deberán ser o máis fieis posibles aos do robot real, sen que comprometa de forma excesiva a velocidade de simulación.
- Os controis do robot virtual deberán ser os mesmos que os empregados no robot real, permitindo que o mesmo código poida ser executado en calquera dos modelos sen necesidade de modificacións.
- Gazebo é un simulador integrado en ROS, polo que para a realización do paquete de simulación empregarase dito sistema, utilizando unha correcta arquitectura dos directorios que o forman.
- Co fin de obter un modelo útil, tanto para investigación como para docencia, deberase garantir a compatibilidade con Python mediante ROS.

6 DESENVOLVEMENTO DO MODELO DE SIMULACIÓN

Neste capítulo faise unha descrición de todo o proceso de desenvolvemento do modelo de simulación levado a cabo durante a elaboración do presente traballo académico.

Definiranse todas as características físicas e visuais do modelo, das que dependeran os comportamentos dinámicos e de representación gráfica durante a simulación. Así mesmo, describiranse os plugins necesarios para a interacción co modelo en simulación e as calibracións necesarias para levar os comportamentos do modelo de simulación a que sexan o máis próximos posibles ao robot real. Por último farase unha breve descrición do directorio que conten o paquete de simulación Robobo.

6.1 Modelo 3D do robot

Como se explicou anteriormente no capítulo 3.3, a descrición do robot realizase mediante o formato SDF na linguaxe XML e recóllese nun arquivo ao que se debe denominar do xeito “nome.sdf”. Esta descrición debe incluír todas as características físicas do robot real, o mellor definidas posible, para que os comportamentos do modelo sexan o máis próximos posible aos reais que a simulación do mesmo poida ter unha utilidade práctica, a de substituír ao robot real. A descrición do modelo realizada no formato SDF recóllese no Anexo I: Descrición do modelo.

Para a elaboración do modelo de simulación partiuse do modelo de Robobo en Blender, desenvolvido por investigadores do GII. Este modelo está formado por mallas tridimensionais que definen todas as pezas físicas que conforman a base robótica e o modelo dun teléfono intelixente. Estas mallas foron creadas reproducindo fielmente e con precisión as pezas reais, o que proporciona unha gran resolución para a obtención dunha representación gráfica de calidade.

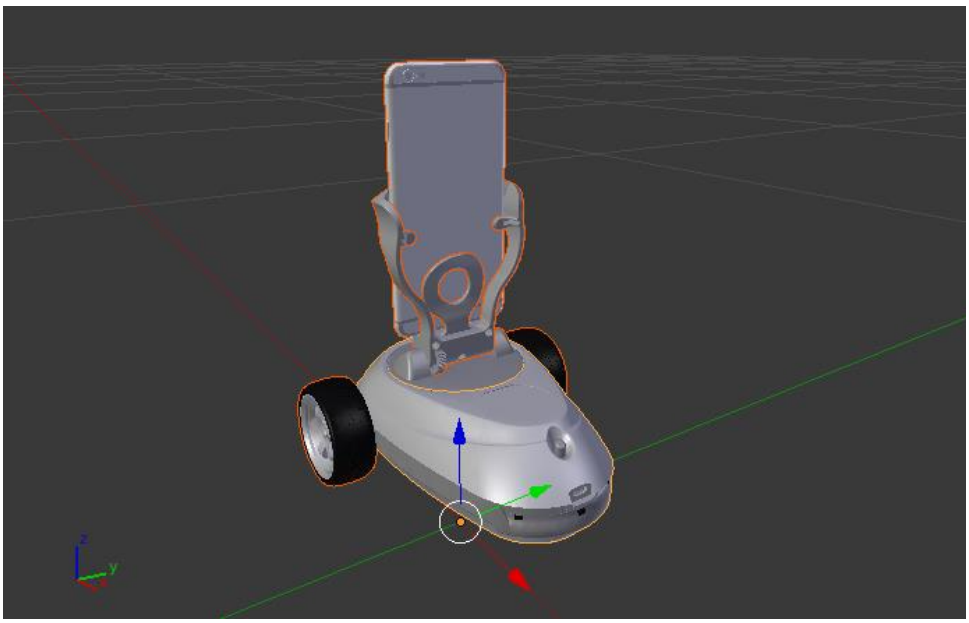


Figura 19. Orixe de coordenadas do modelo de Robobo en Blender

É importante destacar que o sistema de referencia para a definición do modelo é o mesmo que o empregado no modelo de Blender. Na Figura 19 móstrase a orixe e a

orientación dos eixos de coordenadas para dito modelo. Para definir a orientación dos elementos tense en conta que as rotacións de balanceo (roll) teñen lugar arredor do eixo X, as rotacións de cabeceo (pitch) teñen lugar arredor do eixo Y e as rotacións de guiñada (yaw) teñen lugar ao redor do eixo Z.

6.1.1 Elementos visuais e de colisión

Para a definición das diferentes partes que conforman o robot real utilízanse os elementos *link* do SDF. Estes elementos incorporan á súa vez outros dous elementos principais, o *collision* e o *visual*, que definen a xeometría para o cálculo das colisións e a aparencia visual durante a simulación, respectivamente. Existen dúas maneiras diferentes de especificar a xeometría destes elementos, a máis sinxela é definindo unha das tres formas básicas que ofrece o simulador: unha esfera, unha caixa ou un cilindro. A outra opción é importar unha xeometría definida por unha malla 3D definindo a ruta na que se encontra o arquivo que a contén. Xeralmente, co fin de restarlle peso á simulación, defínense os elementos de colisión mediante xeometrías básicas, especialmente nos casos que non é relevante a detección das colisións, como pode ser o caso dun brazo robótico, onde se poden restrinxir os movementos de cada peza a uns rangos determinados. Unha vez definida a xeometría de colisión o máis simple posible sóese incluír mallas cun grao de definición elevado para os elementos visuais, deste xeito conséguese unha aparencia do robot moi realista á vez que non se compromete a velocidade de simulación.

Como se expón no capítulo 3.1, a base robótica Robobo está formada por cinco pezas principais: o corpo, dúas rodas, o pan e o tilt. Polo tanto, utilizarase un elemento de tipo *link* para a descrición de cada unha destas pezas. A posición de cada *link* está documentada na táboa Táboa 3. Gazebo utiliza o Sistema Internacional de Unidades polo tanto, os valores de cada posición estarán en metros e os valores dos ángulos de orientación en radiáns.

Link	**Posición**			**Orientación**		
	X	**Y**	**Z**	**Roll**	**Pitch**	**Yaw**
body	- 0.055984	0.034729	0.0308	0	0	0
left_wheel	- 0.082628	- 0.038096	0.03225	0	0	0
right_wheel	- 0.082628	0.10746	0.03225	0	0	0
pan	- 0.059743	0.034729	0.060151	0	0	3.14159
tilt_smartphone	-0.149486	0.03499	0.03499	0	0	3.14159

Táboa 3. Coordenadas de posición e orientación dos *links*

6.1.1.1 Corpo

O corpo é a peza que da soporte ao resto de compoñentes. Isto quere dicir que o resto de *links* se van mover de forma relativa a este, polo que a posición do modelo será a mesma que a do corpo. Este *link* denomínase “body”.

Esta peza posúe unha forma específica que difire moito das formas simples (cilindro, esfera ou cuboide) que o simulador permite empregar. Por esta razón compre utilizar unha malla o máis simplificada posible como elemento de colisión, de xeito que non comprometa a velocidade de simulación á vez que posúa unha precisión mínima para o cálculo das colisións.

En primeiro lugar, utilizouse una malla simplificada polo simulador V-REP (Figura 20), exportada dun modelo de simulación que o GII dispoñía nese momento para dito programa.

O problema desta malla era que a causa da simplificación perdíase a xeometría da parte do deslizador, polo que era necesario engadir unha forma simple que representase esta parte. Durante a simulación o deslizador é a parte mais importante do corpo, xa que é o apoio este ten sobre o chan, permitindo que se manteña na posición correcta.

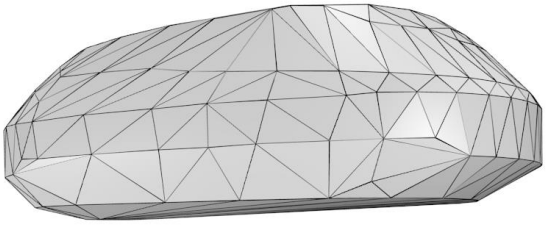


Figura 20. Xeometría exportada do simulador V-REP

Para evitar problemas futuros e que a descrición do robot fose máis simple, decidiuse crear unha malla de xeito manual en Blender partindo do modelo existente neste programa. Desta maneira obtívose unha malla simplificada da xeometría do corpo pero coa cara de rozamento do deslizador situada na posición real, garantindo así que o corpo se sitúe na mesma posición que o real.

No caso do elemento visual, emprégase unha malla exportada directamente do modelo de Blender. Esta xeometría móstrase, xunto coa de colisión, na Figura 21.

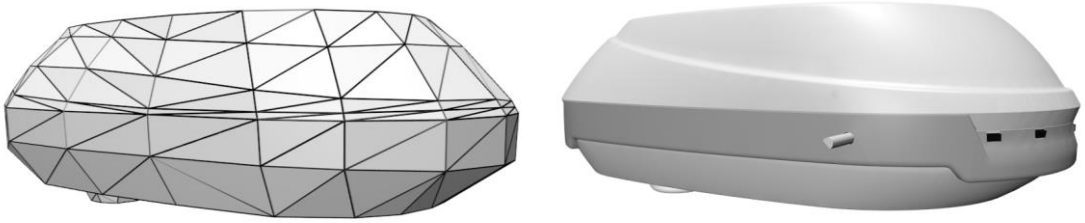


Figura 21. Elemento de colisión e elemento visual para o corpo do robot

Cómpre mencionar que ambas mallas, tanto a visual como a de colisión, están centradas no seu centro volumétrico. Tendo en conta que a posición do *link* é a mesma que a do centro volumétrico destas mallas respecto do sistema de referencia do modelo, non é necesario definir a posición do elemento visual nin de colisión.

6.1.1.2 Rodas

As rodas están compostas polos links denominados “left_wheel” e “right_wheel”.

Para o modelado dos elementos de colisión emprégase un cilindro de 32.25 mm de radio e unha altura de 26 mm. Desta maneira redúcese o peso de cálculo da simulación en comparación á utilización de mallas 3D. De xeito experimental elaborouse unha malla simplificada para a utilización como elemento de colisión das rodas (Figura 22). Os resultados non foron satisfactorios, polo que finalmente se descartou a súa utilización.

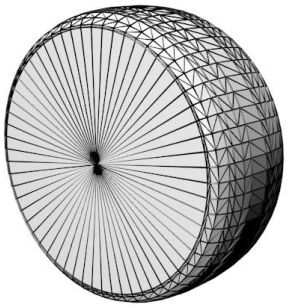


Figura 22. Malla simplificada a partir dunha roda de Robobo

Como elementos visuais utilízanse as mallas exportadas do modelo de Blender. A orixe destas mallas non coincide exactamente coa dos respectivos elementos de colisión, polo que é necesario definir a posición dos elementos visuais desprazados 2.4 mm no eixo Y cara o exterior do robot. Na Figura 23 represéntase a xeometría utilizada para o elemento visual xunto coa de colisión, empregadas para as rodas.

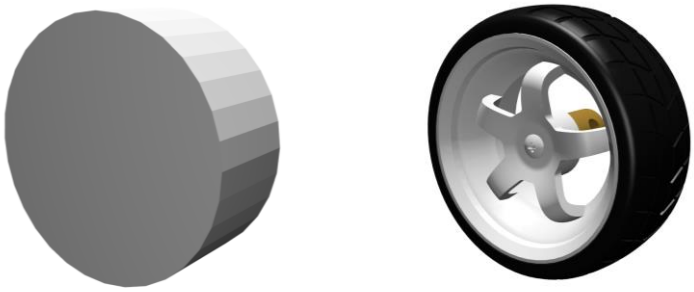


Figura 23. Elemento de colisión e visual para as rodas

6.1.1.3 Pan

No caso do pan, emprégase como elemento de colisión un cilindro de 44 mm de radio e unha altura de 3 mm. Como elemento visual utilízase unha malla exportada do modelo de Blender. Ambas xeometrías coinciden na súa orixe, polo que non é necesario especificar a posición do elemento de colisión nin do visual. Ambos elementos móstranse na Figura 24. As dúas xeometrías coinciden na súa orixe, polo que non é necesario especificar a posición do elemento de colisión nin do visual.

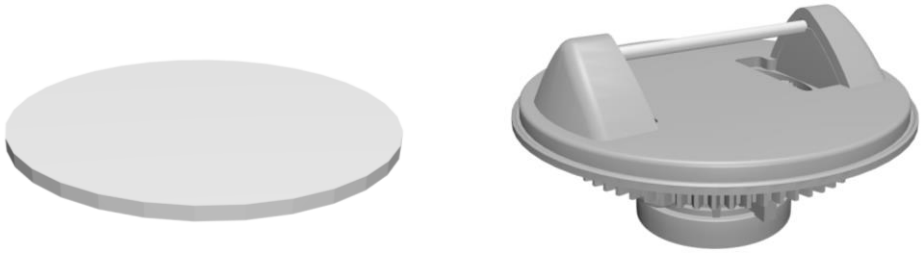


Figura 24. Elemento de colisión e visual para o pan

6.1.1.4 Tilt e smartphone

O modelo de simulación de Robobo debe imitar ao robot real durante o seu funcionamento, polo tanto, débese ter en conta que vai estar colocado un teléfono móbil na plataforma robótica. Deste xeito, modelouse este último *link* cun elemento de posición

formado por un cuboide con dimensións similares ás dun teléfono móbil: *158.1 x 77.8 x 7.7* mm.

Para o elemento visual extraeuse unha malla conxunta do smartphone xunto co soporte do modelo de Blender (Figura 25). Neste caso é necesario definir a posición do elemento visual como: $X = -0.079749, Y = -0.002249$ *e* $Z = -0.004338$.

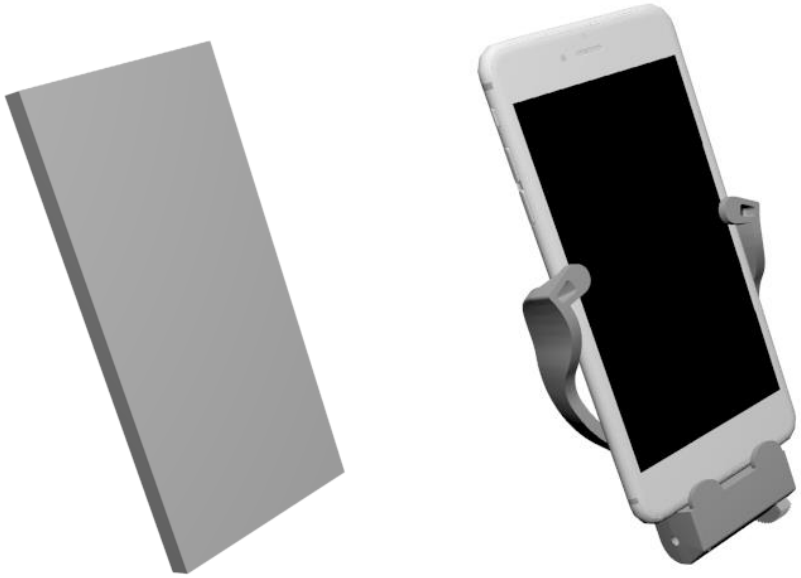


Figura 25. Elemento de colisión e visual para o conxunto tilt-smartphone

6.1.2 Consideracións importantes

Durante a elaboración do modelo, un dos principais problemas afrontados foi a inestabilidade deste durante a simulación. O robot deslizábase sobre a superficie con movementos cambiantes, sen aplicarlle ningunha forza externa. Logo de experimentar a variar numerosos parámetros de simulación e de investigar a través de foros especializados nestes temas, finalmente deuse co motivo deste contratempo. Gazebo, no seu defecto, toma como 0 o parámetro *min_depth*, que se encontra definido dentro dos elementos de colisión e que restrinxe a profundidade mínima permisible antes de que sexa aplicado un impulso de corrección de contacto. Polo tanto, é necesario definir unha tolerancia mínima que evite que o simulador estea aplicando continuamente forzas de corrección que resulten nun comportamento inadecuado. O parámetro *min_depth* só resulta necesario definilo naquelas pezas que van estar en contacto co chan, que neste caso son os *links* que compoñen as rodas e o corpo. Ademais do devandito parámetro, cómpre definir os coeficiente de Poisson e o módulo elástico, ou módulo de Young, de cada peza, xa que están compostas de distintos materiais e, en calquera caso, o valor que toma Gazebo por defecto pode non estar moi próximo ao real.

Por último, débese introducir tamén, o valor dos coeficientes de rozamento dos compoñentes que están en contacto co chan. No caso das rodas, para que non patinen á hora de impoñerlles un par determinado, e no caso do deslizador para que non ofreza demasiada resistencia e impida o correcto desprazamento do robot. O deslizador está integrado no corpo, polo tanto, será toda esta peza a que leve as características do mesmo. Os valores definidos no modelo do presente traballo son os que se mostran na Táboa 4, onde os valores de profundidade mínima se representan en metros e os do módulo elástico en Pascais. Para a obtención destes valores tomouse como referencia o caucho, para as rodas, e o teflón para o deslizador.

Elemento	*min_depth*	Coeficiente de Poisson	Módulo elástico	Coeficiente de fricción
Deslizador	$4 \cdot 10^{-4}$	0.46	$7 \cdot 10^{8}$	0.2
Rodas	$8 \cdot 10^{-4}$	0.5	$7 \cdot 10^{6}$	0.8

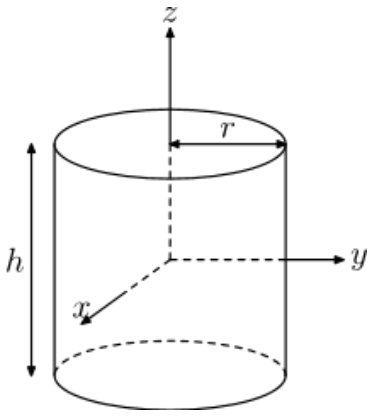
Táboa 4. Parámetros de contacto do modelo

6.1.3 Propiedades inerciais

Un dos requisitos a ter en conta no desenvolvemento deste traballo era que o robot virtual se comportase o máis parecido posible á realidade. Polo tanto, compre definir as propiedades inerciais de cada *link* de xeito adecuado, para que o motor físico de Gazebo proporcione uns resultados de comportamento similares aos reais.

En primeiro lugar, co fin de coñecer os valores das masas dos cinco conxuntos de pezas que compoñen o robot, elaborouse unha coidadosa medición utilizando unha báscula dixital. No caso do smartphone pesáronse catro dispositivos diferentes (Huawei P9, iPhone 6s, Samsung Galaxy J7 e Huawei P8 Lite) obtendo unha media aproximada de 150 gramos que, xunto co peso do tilt suman un total de 180 gramos.

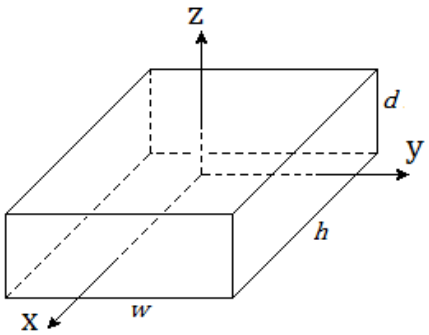
Ademais das masas de cada peza é necesario introducir os valores de cadanseu tensor de inercia. Un tensor de inercia é un tensor simétrico de segundo orde que caracteriza a inercia rotacional dun sólido ríxido. Está formado por unha matriz simétrica, polo que só se necesitan definir os cinco elementos diferentes do tensor. Os tensores de inercia das partes asimiladas como cilíndricas, neste caso as rodas e o pan, foron calculados a partir do tensor de inercia dun cilindro, como se amosa na Figura 26. No caso das rodas a orientación non é a mesma que o pan, xa que as caras planas sitúanse perpendiculares ao eixo Y.



$$I = \begin{bmatrix} \frac{1}{12}m(3r^2 + h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2 + h^2) & 0 \\ 0 & 0 & \frac{1}{12}mr^2 \end{bmatrix}$$

Figura 26. Tensor de inercia para un cilindro de radio *r* e altura *h*

O cálculo do tensor de inercia do conxunto formado polo tilt e o smartphone realizouse para a xeometría de colisión, como se amosa na Figura 27.



$$I = \begin{bmatrix} \frac{1}{12}m(w^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(d^2+h^2) & 0 \\ 0 & 0 & \frac{1}{12}m(h^2+w^2) \end{bmatrix}$$

Figura 27. Tensor de inercia para un paralelepípedo de dimensións *h* x *w* x *d*

O tensor de inercia do corpo de Robobo resulta difícil de calcular de xeito manual, xa que se trata dunha xeometría mais complexa que as anteriores. Polo tanto, recorreuse ao software de deseño Meshlab [21] para a obtención dos momentos de inercia. O funcionamento do programa consiste en realizar os cálculos tendo en conta densidade unitaria, o que quere dicir que asume que o valor de masa do corpo importado é o mesmo que o volume. Unha vez importada a malla de colisión do corpo en Meshlab, obtívose o seguinte tensor:

$$I = \begin{bmatrix} 9.9589 \cdot 10^{-7} & 1.2 \cdot 10^{-10} & 2.811 \cdot 10^{-8} \\ 1.2 \cdot 10^{-10} & 2.51868 \cdot 10^{-6} & -6 \cdot 10^{-11} \\ 2.811 \cdot 10^{-8} & -6 \cdot 10^{-11} & 2.98177 \cdot 10^{-6} \end{bmatrix}$$

Coñecido o volume e a masa do corpo, obtéñense os valores reais do tensor de inercia multiplicando cada termo pola masa e dividindo por o volume. O valor da masa do corpo é de 477 gramos e o valor do volume é de $9.79 \cdot 10^{-4}$ m^3, este último é calculado tamén, por Meshlab.

Na Táboa 5 móstranse todos os valores das propiedades inerciais de cada link que se deben introducir na descrición do modelo de Robobo. Como se mencionou anteriormente, Gazebo utiliza o Sistema Internacional de Unidades, polo que todos os valores da táboa están en unidades de dito sistema.

Link	**Masa**	**Inercias (valores multiplicados por 10^6)**					
		Ixx	**Ixy**	**Ixz**	**Iyy**	**Iyz**	**Izz**
body	0.48	485.229	0.058	13.696	1227.181	- 0.029	1452.813
left_wheel	0.046	14.552	0	0	23.921	0	14.552
right_wheel	0.046	14.552	0	0	23.921	0	14.552
pan	0.077	37.326	0	0	37.326	0	74.536
tilt_smartphone	0.18	91.682	0	0	375.824	0	465.727

Táboa 5. Propiedades inerciais de cada *link*

Cabe destacar que Gazebo calcula as propiedades inerciais dende o eixo de referencia de cada peza, polo que todas elas teñen a orixe no centro volumétrico da súa xeometría, tendo en conta que esta localización é coincidente ou, polo menos, aproximada ao seu centro de masas.

6.1.4 Articulacións

Unha vez definidos todos os *links* do modelo é necesario crear algún tipo de relación entre eles, xa que, no caso contrario, comportaríanse como corpos independentes durante a simulación e non se manterían na súa posición. Estas relacións establécense mediante os elementos denominados *joints*. Para a definición dun *joint* é necesario establecer un link como pai e outro como fillo, de xeito que este último moverase relativamente respecto do primeiro.

No caso de Robobo, as relacións que se deben impoñer entre as cinco pezas que definen o modelo correspóndense cos catro motores que leva instalado o robot. Estes motores permiten o xiro das rodas, ademais da orientación do pan e o tilt. Tendo isto en conta, os *joints* encargados de definir estas relacións serán de tipo revolución.

Os sentidos de xiro especificáronse de xeito que se os motores se comporten como no robot real. Isto quere dicir que os valores de posición se incrementen cando a peza xira no mesmo sentido que no robot real. Na Figura 28 aparece a disposición dos catro *joints* cos que conta o modelo.

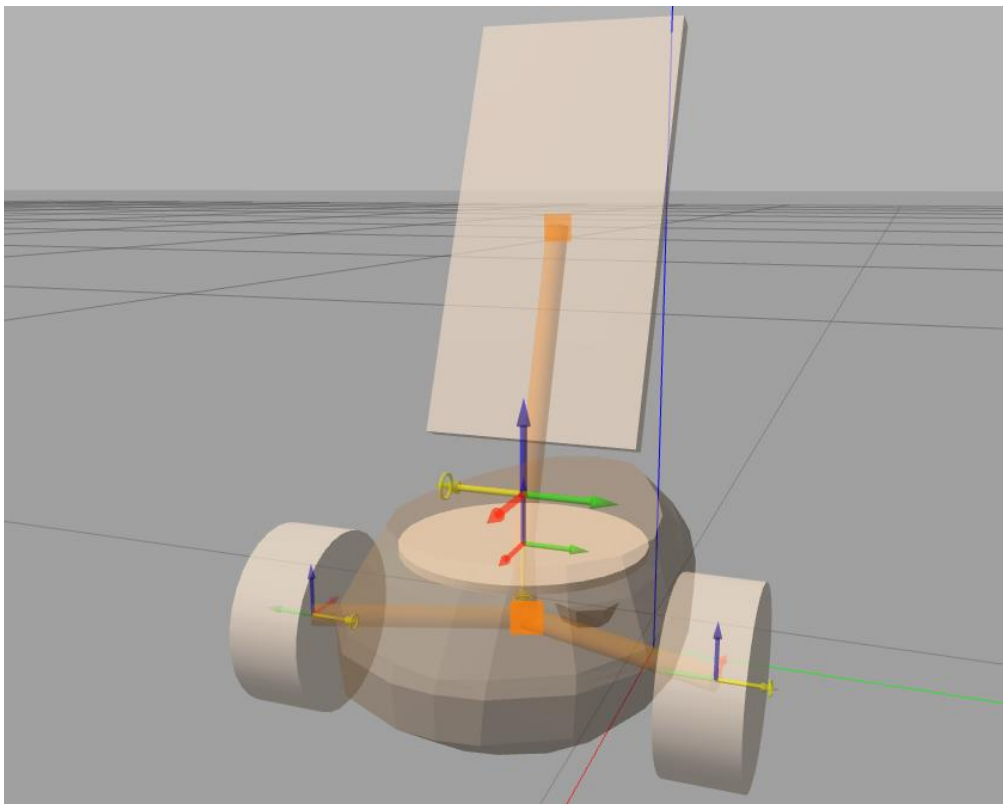


Figura 28. Situación dos *joints* na xeometría de cálculo do modelo

Para evitar que as articulacións se movan idealmente, existe un parámetro de fricción que restrinxe o movemento simulando os rozamentos ou forzas ás que poden estar sometidos na realidade. Estas fricción son as causadas por pequenos rozamentos nos rodamentos, caixas de engrenaxes ou mesmo o par que ofrece un motor eléctrico en baleiro. Aínda que non é un parámetro que tome relevancia durante a simulación, xa que Gazebo só os ten en conta cando a velocidade do *joint* é cero, no modelo definíronse como os valores de par nominal que ofrece cada motor de Robobo.

Na Táboa 6 amósanse as propiedades dos catro *joints* empregados na descrición do modelo.

Joint	***Link* pai**	***Link* fillo**	**Fricción**	**Posición**		
				X	**Y**	**Z**
left_motor	body	left_wheel	0.022	0	0	0
right_motor	body	right_wheel	0.022	0	0	0
pan_motor	body	pan	0.15	0	0	0
tilt_motor	pan	tilt_smartphone	0.22	- 0.089743	0	-0.005

Táboa 6. Parámetros de cada *joint* do modelo

Os *links* do pan e o tilt están orientados de xeito que se facilite a utilización da súa posición á hora da elaboración dos controis dos motores do modelo. O que se pretende conseguir é que os valores de posición dos seus *joints* sexan exactamente os mesmos que os valores dos comandos que se introducen para o control do modelo, evitando así ter que realizar ningún tipo de conversión, tanto na elaboración dos controis como na lectura dos encoders.

A pesar de que se pode especificar os valores do límite superior e inferior dun *joint* dentro do arquivo XML que define o robot, non é aconsellable utilizalo deste xeito. Esta recomendación débese a que, na versión de Gazebo empregada para o desenvolvemento deste traballo, ao definir ditos límites no XML, ocasionaba problemas cando se pretendía variar eses valores durante a simulación por medio dun plugin.

6.1.5 Sensores infravermellos

Mediante o elemento sensor do SDF descríbese o tipo e as propiedades do sensor que se desexa incluír no modelo. No caso de querer simular sensores infravermellos emprégase o tipo de sensor *ray*. Este tipo de sensor devolve o valor da distancia á que un corpo interseca coas rectas, ou raios, que o definen.

O funcionamento dun sensor de tipo *ray* é sinxelo: o simulador crea unha especie de abanico no plano horizontal ao longo do ángulo que recorre dende o valor mínimo ata o valor máximo especificado no SDF, e repartindo de xeito equidistante o número de raios indicados para o plano horizontal. Á súa vez, despregа esta configuración tantas veces como raios se indicaran no plano vertical, e repartíndoos, tamén, de forma equidistante entre o ángulo mínimo e máximo especificados para o plano vertical. Deste xeito os raios dispóñense formando unha especie de volume piramidal, cuxo vértice se sitúa no punto mínimo do rango especificado, tomando como referencia a posición do sensor. A lonxitude de cada raio ven definida loxicamente, pola diferenza do punto máximo e mínimo do rango do sensor.

O simulador asigna un índice a cada un dos raios que conforman o sensor. Este índice serve para acceder ao valor de distancia mínimo en que se corta dito raio en calquera instante da simulación. O sistema de numeración de Gazebo foi obtido de xeito experimental, coñecendo finalmente que, dado o plano vertical YZ, a numeración comezaría dende a orixe de coordenadas e incrementase ao longo da fila horizontal no sentido crecente do eixo Y, ata que salta á seguinte fila no sentido crecente do eixo Z. Así, para o caso do sensor implementado no modelo de Robobo, a numeración dos índices sería a da Figura 29.

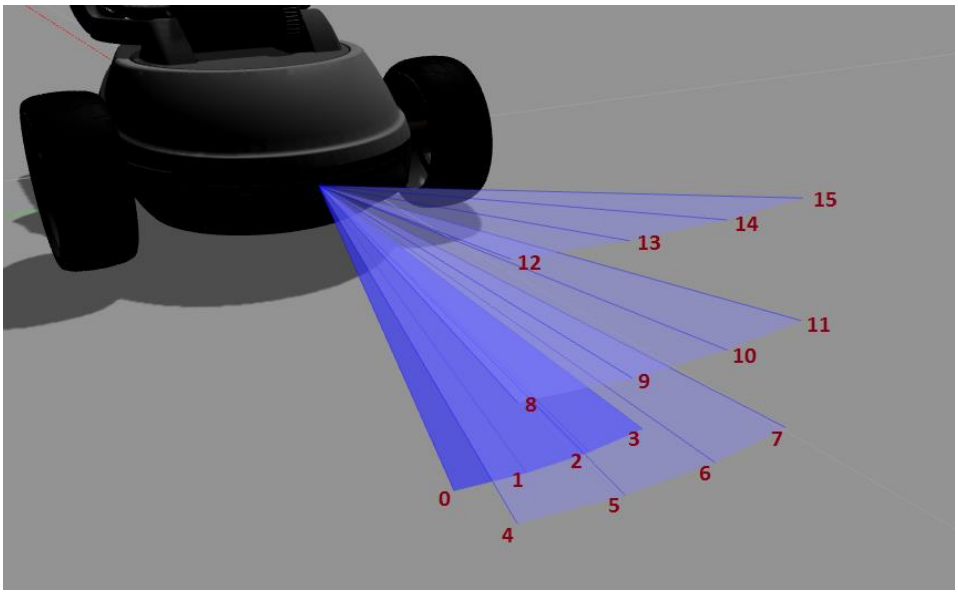


Figura 29. Índices de numeración dos raios que forman un sensor tipo *ray*

A descrición dun elemento de tipo sensor conta cunha propiedade que permite introducir ruído. O que fai é variar de xeito aleatorio o valor de distancia que devolve cada raio. Para o desenvolvemento deste modelo non se utiliza dito parámetro xa que, no caso de querer introducir ruído aos sensores, é conveniente utilizar a funcionalidade do plugin empregado, que introduce a variación no valor de saída final do sensor.

Durante a elaboración do modelo, considerouse que era suficiente a utilización de 4 raios en cada plano horizontal por catro filas no plano vertical. Esta configuración suma un total de 16 raios e ofrece unha resolución axeitada.

Os sensores infravermellos presentan unha variación considerable do seu funcionamento en función da superficie do material que detecten, xa que cada material posúe unha reflectividade diferente. Coa intención de estandarizar os métodos de medición destes sensores realizados dende o GII, para todos os resultados obtidos de xeito experimental foi empregado un material de superficie plana, liso, de cor branco e mate. Polo xeral empregáronse láminas de papel para revestir as superficies de medición. No caso de pretender ter un comportamento dos mesmos moi aproximado ao real, sería preciso realizar calibracións para calquera tipo de material que poida ser empregado durante experimentos reais. Este labor require dun longo e minucioso traballo que non resulta factible para este tipo de robot.

Para calcular a superficie que debería abarcar o sensor botouse man dos resultados de medicións experimentais realizadas en anteriores traballos de investigación sobre Robobo. A intención á hora de calcular os ángulos máximo e mínimo que require a descrición do sensor, era que os raios exteriores cortasen o perímetro da área de detección real aproximadamente no punto onde esta comeza a estreitecerse. Na Figura 30 represéntanse superpostas ambas áreas de detección, onde a orixe do gráfico sería o punto onde se sitúa o sensor.

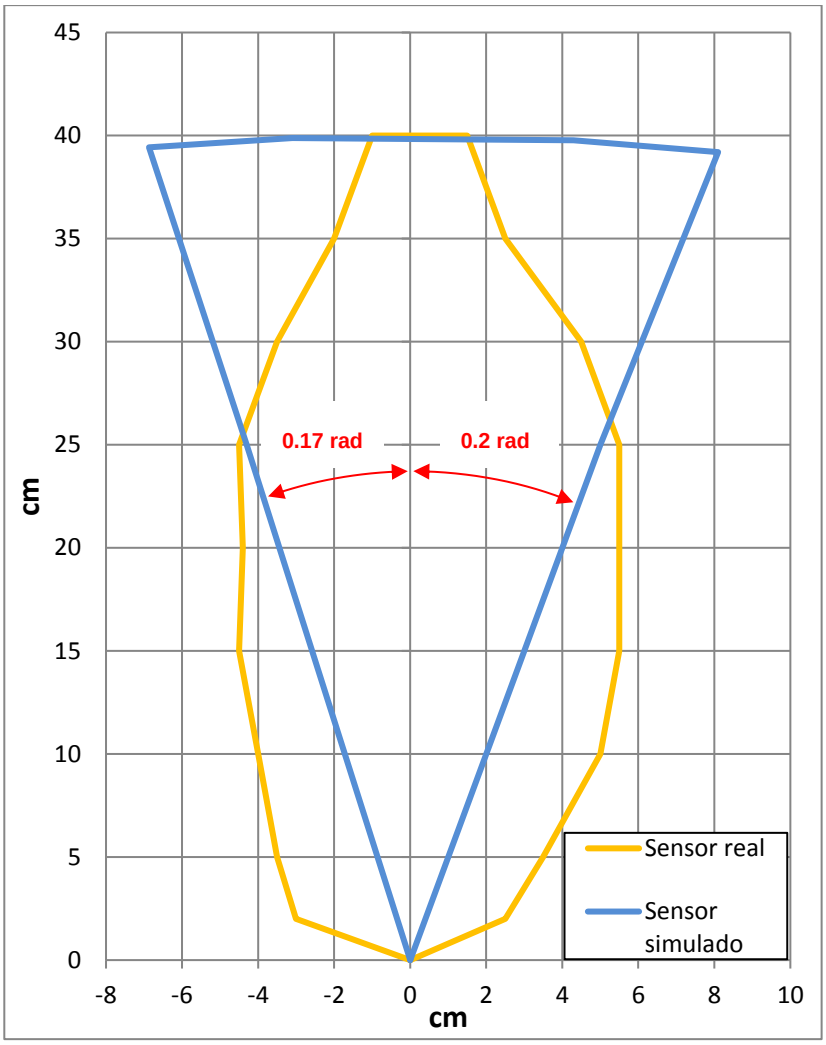


Figura 30. Gráfico coas áreas de detección no plano horizontal

A pesar de que ambas superficies non aparenten ter demasiada coincidencia, en Gazebo resulta complexo elaborar aproximacións máis exactas. Sen embargo, esta configuración ofreceu resultados satisfactorios durante o desenvolvemento do modelo. Hai que ter en conta que nas simulacións que se poidan realizar no futuro con este modelo non se acostuma empregar obxectos de dimensións moi pequenas, polo que sería moi difícil que un corpo fose capaz de colarse entre os sensores sen cortar ningún raio.

Gazebo utiliza, para a definición dos ángulos máximos e mínimos, o incremento positivo no sentido de xiro contrario ao das agullas do reloxo. Polo tanto, definíronse os valores como 0.17 e -0.2 radiáns para os ángulos máximo e mínimo do plano horizontal, respectivamente. Do mesmo xeito, no plano vertical os valores serían 0.16 e -0.12 radiáns. No caso dos sensores encargados de detectar o chan, o valor do ángulo mínimo no plano vertical redúcese a -0.085 radiáns.

Todos os sensores infravermellos están situados no corpo da base robótica, polo que na descrición do modelo serán fillos deste *link*. A disposición dos mesmos respecto deste elemento amósanse na Táboa 7, onde os valores de posición están en metros e os de orientación en radiáns.

Sensor	Posición			Orientación		
	X	Y	Z	Roll	Pitch	Yaw
front_c	- 0.08	0	0.0012	0	0	3.1415
front_l	-0.0727	- 0.026	0.0012	0	0.21	3.526
front_ll	- 0.0625	- 0.0405	0.0012	0	0	3.927
front_r	-0.0727	0.026	0.0012	0	0.21	2.758
front_rr	- 0.0625	0.0405	0.0012	0	0	2.356
back_c	0.12	0	0	0	0.21	0
back_l	0.1055	-0.0308	0.001	0	0	- 0.524
back_r	0.1055	0.0308	0.001	0	0	0.524

Táboa 7. Disposición dos sensores infravermellos no modelo de simulación

O ángulo de inclinación dos sensores encargados de detectar o chan é lixeiramente inferior ao que presentan os do robot real. Esta variación, así como a de teren un ángulo de apertura vertical dos raios menor que o resto de sensores, ten a finalidade de que os valores de intensidade que devolvan os devanditos sensores sexan próximos aos reais.

A definición de sensor conta tamén cun parámetro que, mediante unha variable booleana, activa ou desactiva a representación gráfica dos mesmos durante a simulación. Esta función permite ao usuario comprobar que o modelo se está executando do xeito desexado.

6.1.6 Cámara

O Robobo real fai uso da cámara frontal do smartphone, presente por regra xeral en tódolos dispositivos deste tipo que se encontran actualmente no mercado. Esta funcionalidade é amplamente empregada para a realización de probas no desenvolvemento da docencia nos campos da robótica así como en labores de investigación como, por exemplo, o desenvolvemento de métodos de aprendizaxe. É por isto polo que o modelo simulado debería contar coa posibilidade de captar imaxes do mundo da simulación en tempo real.

Gazebo permite simular unha ou máis cámaras a través do elemento *camera*, fillo dun elemento tipo *sensor* que, á súa vez é fillo dun elemento tipo *link*. Neste caso, o devandito elemento debe ser fillo do *link* que contén o smartphone, de xeito que se mova solidario ao mesmo, como ocorre na realidade. A localización da cámara situouse de xeito aproximado a partir do modelo 3D de smartphone empregado. Débese ter en conta que cada dispositivo ten unha localización diferente da cámara, pero estas pequenas variacións non debían influir severamente no enfoque da imaxe que proporcionan. A posición do elemento *camera* está definida como $X = 0.0695$, $Y = 0.0105$ e $Z = 0.004$, e a orientación como $Roll = 0$, $Pitch = -1.5708$ e $Yaw = 3.1415$.

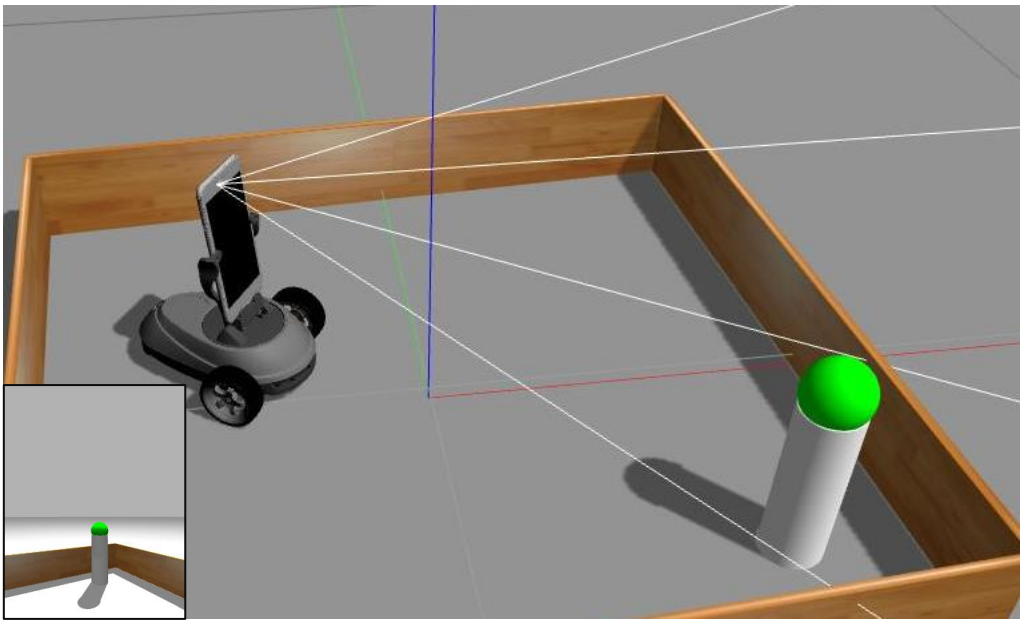


Figura 31. Situación da cámara no modelo xunto con detalle da imaxe captada

Na Figura 31 móstrase a disposición da cámara no modelo de simulación do Robobo xunto coa imaxe que xera dentro do simulador.

6.2 Plugins

Mediante a interface gráfica de Gazebo permítese interactuar durante a simulación variando parámetros nos *joints,* como pode ser a velocidade, cambiar a posición dun modelo, etc.; pero este método non é práctico cando se queren comprobar diferentes comportamentos dun robot. Para iso é necesario a utilización de plugins, que se cargan de forma dinámica durante a execución do sdf e permiten a interacción co propio modelo mediante ROS. Deste xeito, é posible coñecer os diferentes estados dos sensores así como introducir comandos de actuación sobre o robot a través dos temas e servizos de ROS.

Gazebo está implementado na linguaxe de programación C++, polo tanto, os plugins teñen que ser elaborados nesta linguaxe. Pódese dicir que se trata dunha linguaxe de programación multiparadigma, xa que engade facilidades de programación xenérica ademais dos paradigmas de programación estruturada e programación orientada a obxectos. C++ non é unha linguaxe interpretada, polo que é necesario o uso dun compilador para poder executar o código. Neste caso, para a o desenvolvemento do modelo e conseguinte elaboración dos plugins, foi utilizado o sistema de compilación Catkin. Catkin é o sistema de compilación oficial de ROS, e combina macros de CMake e scripts de Python para proporcionar algunha funcionalidade sobre o fluxo de traballo normal de CMake.

O simulador Gazebo conta cunha documentación da súa API, accesible dende a propia páxina web. Nesta documentación proporcionase información útil sobre os módulos de programación, así como unha lista de todas as clases existentes en Gazebo. Outro aspecto interesante é que informa das funcións que están obsoletas e por cales foron substituídas, especialmente útil se se desexa migrar de versión do simulador.

Coa intención de ofrecer un modelo de simulación que poida ser controlado da mesma maneira que o real, os temas e servizos de ROS utilizados nos plugins e que sexan para a utilización dos usuarios e non a nivel interno, foron creados a semellanza dos empregados no robot real. A información dos temas e servizos do Robobo encóntrase correctamente documentada na súa Wiki [22]. Tendo en conta o anterior, utilizáronse as propias mensaxes

que se utilizan no robot real, polo que para poder executar o modelo de simulación é necesario ter instalado no espazo de traballo o paquete de mensaxes de Robobo [23].

Todos os temas de ROS creados dende os plugins están nomeados de xeito que se poidan lanzar varias unidades do mesmo modelo sen que se solapen temas co mesmo nome. Este método consiste en introducir un prefixo que asigna nome do nodo de cada modelo ao nome do tema , da seguinte maneira: /nome_do_nodo/nome_do_tema.

O código desenvolvido nos seguintes plugins atópase dispoñible no seguinte enlace: https://github.com/david-casal/robobo_gazebo.

6.2.1 Motores das rodas

Robobo conta con dúas rodas na súa base robótica que se poden mover de maneira independente, permitíndolle desprazarse en liña recta ou realizar xiros de ata 360º. Dende ROS, o tipo de servizo “robobo_msgs/MoveWheels” é o encargado de comandar aos motores das rodas que estas se movan á velocidade especificada durante o tempo desexado. As variables de velocidade e tempo son de tipo enteiro, debendo introducirse o valor de tempo en milisegundos e os valores de velocidade dentro do rango de 0 a 100. Coa finalidade de conseguir esta funcionalidade no robot real, creouse un plugin encargado de xerar dito servizo de xeito que a forma de comandar os movementos das rodas sexa exactamente a mesma no modelo de simulación que no modelo real. Ao lanzar o modelo de simulación nun nodo determinado, xérase un servizo co nome “nome_do_nodo/moveWheels”.

En Gazebo existen tres maneiras de establecer a velocidade dun joint:

1. Establecer a velocidade de forma instantánea.
2. Crear un controlador PID.
3. Configurar un motor.

O primeiro caso é un método pouco práctico, xa que o obxecto comeza a moverse á velocidade desexada de forma inmediata, sen que se apliquen forzas ou pares sobre os mesmos. Como consecuencia, as forzas ou pares poden cambiar a velocidade do obxecto despois de que fose establecida, sendo preciso establecer a velocidade cada paso de tempo para manter o obxecto a unha velocidade constante. O segundo método consiste en configurar un controlador PID encargado de aplicar as forzas e pares necesarios para alcanzar a velocidade desexada. A principal desvantaxe deste método é que require unha especificación das constantes diferente para cada obxecto que se precise mover, sendo necesaria a realización dun proceso de axuste ata conseguir o comportamento desexado. O terceiro e último método consiste en simular o funcionamento dun motor real, de xeito que, unha vez definido o par nominal do motor, o simulador emprega a forza exacta (inferior ou igual ao par nominal) sobre o *joint* no que está implementado para alcanzar a velocidade desexada. Este método so está dispoñible empregando o motor de física ODE.

Tendo en conta que ODE é o motor de física empregado por defecto en Gazebo, e que foi o motor empregado para o desenvolvemento do presente traballo, considerouse que o método máis oportuno para establecer a velocidade das rodas da base robótica no modelo de simulación sería realizando a configuración dun motor para cada *joint*. Deste xeito utilízase a función *SetParam*, pertencente á clase *Joint* da API de Gazebo. A través desta función permítese establecer os valores dos atributos de velocidade obxectivo e a forza máxima que será aplicada sobre o *joint*. Nun primeiro momento estableceuse o par máximo como o valor do par nominal que ofrecen os motores do robot real pero, durante a realización das probas púidose comprobar que o comportamento non era o correcto. Cando se aplicaba unha velocidade determinada a unha soa roda, a base robótica comezaba a xirar intermitentemente entre dúas velocidades diferentes, como se tivese algunha restrición que o frease en determinados momentos. Posiblemente fose debido a cuestións relacionadas coas forzas de contacto. Como solución optouse por definir o par máximo dos

motores das rodas como 0.5 N·m, xa que deste xeito desaparecía o problema e non se apreciou un comportamento diferente á hora de alcanzar a velocidade obxectivo.

6.2.2 Motores pan e tilt

Os motores do pan e o tilt permiten orientar o cabeceo e a guiñada do smarphone respecto da base robótica. Os valores dos ángulos que definen a orientación destas pezas increméntanse conforme o esquema representado na Figura 32. Estes motores son controlados a través do tipo de servizo "robobo_msgs/MovePanTilt", a través do cal se introducen os comandos de velocidade e posición para cada peza. Este control permite establecer calquera destas dúas pezas de xeito independente na posición desexada e a unha velocidade determinada. Cada vez que se lance un modelo xérase un servizo co nome "nome_do_nodo/movePanTilt", a través do cal se controlan ambas pezas do mesmo xeito que no robot real. Os rangos dos valores de posición son de 11º a 344º para o pan e de 5º a 110º para o tilt En canto aos valores de velocidade, ao igual que o caso das rodas, o valor vai entre o rango de 0 a 100.

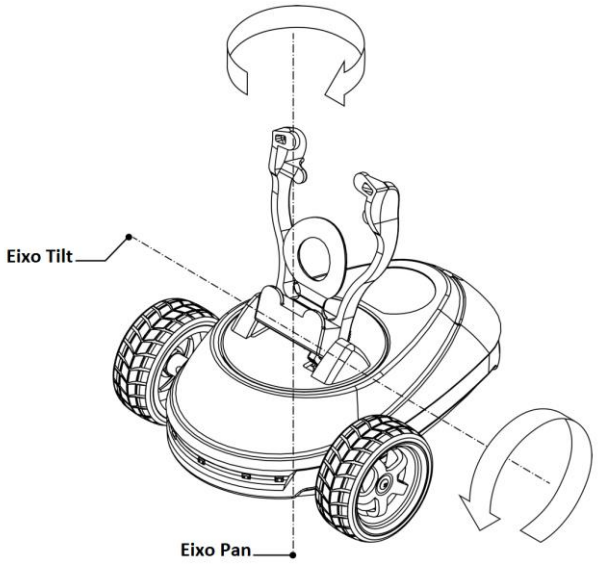


Figura 32. Eixos de rotación do pan e o tilt da base robótica

Como se explicou no apartado anterior, o método de actuar sobre os *joints* destes obxectos é configurando un motor mediante a función *SetParam*. Neste caso, o parámetro de par máximo que emprega cada *joint* á hora de establecer a velocidade obxectivo é o do par nominal de cada motor multiplicado pola relación de engrenaxes. Estes valores son de 0.15 N·m para o pan e de 0.22 N·m para o tilt, e ofreceron uns resultados de funcionamento satisfactorios durante a simulación.

Dende este plugin realizase un control por posición para cada peza, de xeito que se establece a velocidade indicada no servizo en valor positivo ou negativo, dependendo da posición real do elemento e a comandada, ata que ambas posicións coincidan. Cando a velocidade é cero, é necesario bloquear estes *joints*, especialmente o do tilt, xa que no caso contrario acaba por caer a causa das forzas de gravidade. A pesar de que existen varios métodos para bloquear o movemento destes elementos, tras diferentes probas o único método que demostrou ser efectivo foi o de establecer o límite inferior e o límite superior do *joint* na posición en que se encontra no instante que a velocidade sexa nula. Para establecer o límite inferior e o límite superior utilízanse as funcións *SetLowStop* e *SetHighStop,* respectivamente.

Cómpre mencionar que, como se explicou anteriormente, a colocación dos *links* está convenientemente establecida na posición cero dos ángulos de xiro do pan e do tilt. Cando se arranca o robot Robobo, por defecto, orienta o smartphone cara o frente e inclínao a 70º. Por esta razón, cando se inicializa o modelo este plugin é o encargado de colocar o pan e o tilt en dita posición.

6.2.3 Encoders

Os encoders son os elementos encargados de especificar a posición absoluta na que se encontran os motores que compoñen a base robótica. A función deste plugin é a de crear os temas de ROS onde posteriormente se publicará as posicións de ditos motores. Deste xeito o plugin xera tres temas:

1. "nome_do_nodo/pan".
2. "nome_do_nodo/tilt".
3. "nome_do_nodo/wheels".

O primeiro e o segundo tema son do tipo "std_msg/Int16" e neles publícase a posición do pan e do tilt en graos respectivamente. O terceiro é un tema de tipo "robobo_msgs/Wheels", onde se publica a posición e a velocidade de cada roda da base robótica.

O código deste plugin atópase recollido a modo de exemplo no Anexo II: Plugin dos encoders.

6.2.4 Sensores infravermellos

Dende a Wiki de ROS está dispoñible o paquete "gazebo_ros_pkgs", onde se pode encontrar unha serie de plugins estándar desenvolvidos para Gazebo. Entre eles encontrase o plugin "gazebo_ros_range", que mediante un sensor do tipo *ray* devolve un sinal como se se tratase dun sensor infravermello ou dun sonar. Este plugin xera un tema de ROS do tipo "sensor_msg/Range" onde publica directamente a distancia máis curta á que un raio do sensor é cortado por outro obxecto.

Partindo do plugin "gazebo_ros_range", realizáronselle as modificacións necesarias para que, en primeiro lugar, o valor que devolva cada sensor sexa de intensidade e non de distancia. Para conseguir este fin, utilízase a ecuación obtida tras a calibración dos sensores infravermellos (Capítulo 6.4.3). En segundo lugar cambiouse o nome do tema en que publica, de xeito que utilice o nome do nodo do modelo da seguinte maneira: "nome_do_nodo/nome_do_sensor". Dito plugin conta tamén cunha opción para introducir ruído sobre a lectura do sensor, para o cal, basta con definir a porcentaxe de ruído que se desexa engadir no parámetro *gaussianNoise* do SDF. No caso deste modelo decidiuse non introducir ruído, xa que no robot real existe unha función que filtra o sinal. Ademais desta funcionalidade tamén é posible variar a frecuencia coa que se publica no tema de ROS, mediante o parámetro *updateRate*. No caso de non definir calquera destes dous parámetros, tómase cero como valor por defecto.

Dentro da definición de cada sensor no SDF necesítase introducir este plugin, isto quere dicir que vai existir un tema diferente para cada sensor, non sendo posible a agrupación dos mesmos nun só tipo de mensaxe de ROS.

6.2.5 Cámara

No caso da cámara utilízase o plugin do paquete de ROS "gazebo_ros_camera", encargado de publicar nun tema do tipo "sensor_msgs/CameraInfo". Dende este tema os usuarios poden empregar estas imaxes para procesalas ou velas dende aplicacións como *rqt_image_view* ou *rviz*. Este plugin foi convenientemente adaptado para que cree o tema co nome do nodo de cada modelo, garantindo así poder lanzar varias unidades de Robobo á vez.

6.3 Uso do modelo en ROS con python

Un dos requisitos que debe cumprir o modelo de simulación é que os controis deben ser exactamente os mesmos que os implementados no robot real. No caso dos sensores infravermellos, o plugin empregado só permite publicar cada sensor nun tema individual, mentres que no robot real de Robobo os valores destes sensores publícanse nun tema tipo "robobo_msgs/IRs". Este tema, á súa vez, está formado polas oito mensaxes que conteñen o valor individual de cada sensor.

Coa intención de poder englobar o total de oito temas que crean os sensores nun só, creouse un novo nodo en ROS no cal se executa un programa encargado de agrupar esas oito mensaxes e publicalas no tema "nome_do_nodo/irs", do tipo "robobo_msgs/IRs". Para a realización deste programa empregouse a linguaxe de programación Python. Esta é unha linguaxe de programación interpretada, multiplataforma e cunha filosofía que fai fincapé nunha sintaxe que favoreza un código lexible. Trátase dunha linguaxe de programación multiparadigma, xa que soporta orientación a obxectos, programación imperativa e, en menor medida, programación funcional. Para a programación en ROS utilízase a biblioteca de cliente rospy, cuxa API habilita aos programadores de Python a interconectar rapidamente cos Temas, Servizos e Parámetros de ROS.

6.4 Calibracións

Unha vez conseguido o modelo de simulación é necesario caracterizar aqueles elementos que van definir os comportamentos do robot, tales como os motores ou os sensores infravermellos. Para poder asimilar o funcionamento do modelo de simulación ao do robot real cómpre realizar unha serie de medicións que permitan obter funcións matemáticas para a calibración dos movementos e a lectura dos sensores.

6.4.1 Motores das rodas

Como se explicou en apartados anteriores, as rodas do modelo de simulación móvense a unha velocidade determinada durante un tempo especificado. Os valores comandados sitúanse no rango de 0 a 100, pero este valor de comando debe ser extrapolado a un valor de velocidade angular á que se debe mover a roda. Co fin de coñecer o valor de velocidade angular que se debe introducir no simulador para cada caso é necesario caracterizar a resposta da base robótica real a través dun proceso de calibración. Dito proceso consiste na realización dunha serie de medicións das respostas ofrecidas polo robot para uns comandos determinados e, unha vez obtidos todos os datos que se consideren oportunos, aproximar os resultados a unha función matemática que relacione os valores de entrada cos de saída coa maior precisión posible.

A calibración dos motores das rodas da base robótica xa foi realizada con anterioridade noutro traballo de investigación [24], polo que se aproveitaron os resultados obtidos para o desenvolvemento deste modelo. Dita calibración consiste na caracterización do movemento do robot real segundo a distancia percorrida en función dos comandos de velocidade e tempo. Considerouse que o ángulo percorrido por unha roda podíase expresar como unha ecuación do tipo: $\theta = A \cdot t + B$, onde A sería o termo da velocidade angular, t o tempo e B o termo independente. A e B consideráronse polinomios de terceiro grao que dependen da porcentaxe de velocidade P. Deste xeito obtense a ecuación final da velocidade que se debe introducir no simulador:

$$\omega = \frac{\theta}{t} = (a \cdot P^3 + b \cdot P^2 + c \cdot P + d) + \frac{(e \cdot P^3 + f \cdot P^2 + g \cdot P + h)}{t}$$

Os coeficientes da ecuación anterior foron calculados mediante o complemento *solver* de Microsoft Excel, de xeito que o erro cuadrático medio entre o valor do ángulo percorrido

polas rodas do robot real e o do ángulo calculado fose mínimo. Os resultados obtidos para ditos coeficientes móstranse na Táboa 8.

a	b	c	d	e	f	g	h
-0.00005	0.00522	6.35708	51.36677	-0.00033	0.04285	-2.06405	-17.69727

Táboa 8. Coeficientes da ecuación de axuste da velocidade das rodas

Mediante a expresión obtida, o valor da velocidade resulta en graos por segundo, polo que é necesario converter o valor a radiáns por segundo antes de introducilo no comando de Gazebo.

6.4.2 Motores do pan e o tilt

A pesar de que se trata do mesmo tipo de actuadores que os tratados no apartado anterior, para este caso o control destes elementos realízase por posición. Isto quere dicir que a posición que debe alcanzar o elemento é coñecida e, polo tanto, non debe haber erro entre a posición acadada no simulador e a acadada na realidade. Neste caso precísase coñecer tamén o valor da velocidade, pero para tratar de conseguir que o tempo que tarda o elemento en alcanzar a posición no modelo simulado sexa o mesmo que no robot real.

Ao igual que no caso das rodas, a calibración do pan e do tilt tamén foi realizada durante outras tarefas de investigación [24]. Partindo da mesma ecuación que a descrita no apartado anterior, realizouse un axuste dos coeficientes mediante *solver* para encontrar unha solución en que o erro cuadrático medio entre o tempo cronometrado na base robótica e o tempo teórico sexa mínimo. Tras o axuste, os coeficientes obtidos para cada un destes elementos encóntranse na Táboa 9.

Elemento	a	b	c	d	e	f	g	h
Pan	-0.00003	0.00388	0.84747	8.05468	-0.00002	0.00106	-0.33034	-0.89074
Tilt	0.00001	-0.00260	0.48094	3.18153	-0.00009	0.01233	-0.54953	4.73795

Táboa 9. Coeficientes da ecuación de axuste da velocidade para o Pan e o Tilt

6.4.3 Sensores infravermellos

Como se mencionou en apartados anteriores, o modelo de sensor que se pode implementar en Gazebo só devolve o valor de distancia ao que corta cada un dos raios que o forman. É por isto que se debe achegar unha función matemática que, en función da distancia de todos os raios que forman un mesmo sensor, devolva un valor de intensidade.

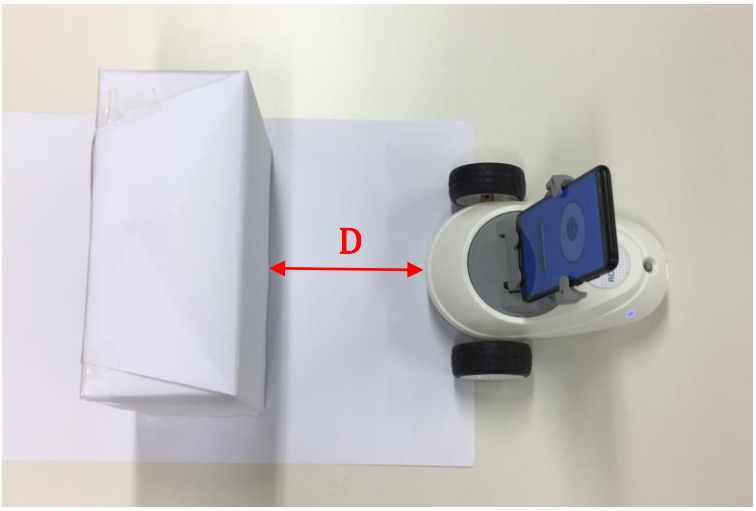


Figura 33. Calibración do sensor infravermello central

Tendo en conta que todos os sensores cos que conta o robot Robobo son do mesmo modelo, sería suficiente con realizar a calibración dun deles. Neste caso decidiuse realizar a calibración do sensor situado no centro da parte central da base robótica, por ser o máis convenientemente situado á hora de realizar os experimentos. Para a realización desta calibración empregouse unha caixa de cartón de dimensións: 23 cm de ancho, 10 cm de longo e 18.5 cm de alto; que foi situada a diferentes distancias ao longo do eixo central deste sensor. Os sensores infravermellos teñen un comportamento que varía de forma apreciable en función da reflectividade da superficie dos obxectos que detecte en cada momento. Polo tanto, co fin de empregar un material estándar para a obtención dos resultados, as medicións foron realizadas enriba dunha lámina de papel de cor branco. Así mesmo, a caixa de cartón foi recuberta deste mesmo material. Os resultados obtidos tras a realización das medicións recóllense na Táboa 10.

Distancia (cm)	Intensidade
5	415
10	117
15	57
20	35
25	26
30	21
35	18

Táboa 10. Valores das medicións realizadas para un sensor infravermello

En traballos anteriores considerouse que a expresión matemática á que se poden aproximar estes valores en función da distancia é da seguinte forma:

$$IR = \sum_{i=0}^{n} a_i \cdot {D_i}^{b_i}$$

Onde IR é o valor de intensidade, D a distancia á que corta cada raio e i o índice de cada raio do sensor do modelo de simulación. Os coeficientes a e b foron calculados

mediante o complemento *solver* de Microsoft Excel co obxectivo de minimizar o erro cuadrático medio entre o valor real medido polo infravermello e o valor calculado, utilizando as restricións de que os valores do coeficiente a fosen positivos e os valores do coeficiente b fosen negativos, ademais de que sexan os mesmos para todos os raios. Decidiuse que todos os raios ponderasen igual xa que, tendo en conta que o valor de lectura do sensor depende de diversos factores e principalmente un deles é a reflectividade, non ten sentido perder demasiado tempo en buscar unha aproximación perfecta para este caso, cando os resultados ao detectar outro tipo de material poidan ser considerablemente diferentes. Finalmente os valores obtidos para ditos coeficientes foron $a = 0.124$ e $b = -1.7823$.

6.5 Paquete Robobo

O resultado do desenvolvemento deste traballo é a obtención do paquete de ROS *robobo*, que está estruturado co seguinte directorio de carpetas:

- **robobo:** carpeta xeral do paquete
  - **include:** recolle os ficheiros *.h* dos plugins implementados en C++.
  - **launch:** recolle os ficheiros de lanzamento dos nodos necesarios para a simulación.
  - **models:** nesta carpeta deben depositarse os modelos que sexan empregados durante a simulación.
  - **nodes:** recolle os ficheiros para a creación de novos nodos.
  - **scripts**: aquí serán depositados os programas que se realicen para programar os comportamentos do modelo.
  - **src:** esta carpeta contén o ficheiro de Python co código encargado de agrupar as mensaxes dos sensores infravermellos nunha soa.
  - **worlds:** nesta carpeta deben depositarse os mundos que sexan empregados na simulación.

7 PROBAS E VALIDACIÓN

Neste capítulo expoñeranse os resultados de funcionamento para as diferentes probas realizadas sobre o modelo de simulación desenvolvido neste traballo, co fin de garantir o correcto funcionamento do mesmo. Ademais farase unha breve explicación dos pasos a seguir para realizar unha simulación do devandito modelo.

7.1 Utilización do paquete *robobo*

O paquete *robobo* debe ser introducido nun espazo de traballo (*workspace*) de ROS, dentro da carpeta *src* correspondente. Como se explicou en capítulos previos, para o funcionamento do modelo empréganse os tipos de mensaxes de ROS que se utilizan no robot real, polo tanto, é preciso introducir tamén o paquete *robobo_msgs* [23]. Unha vez se dispoña de ambos paquetes no entorno de traballo, antes da primeira utilización é necesario compilar o código mediante a ferramenta *catkin*. Esta acción consiste en abrir un terminal novo dentro do entorno de traballo e, a continuación, introducir o comando *catkin_make*. Unha vez se teña compilado o código, a utilización do modelo de simulación é moi sinxela:

1. Crear un ficheiro de lanzamento (*nome.launch*) que conteña un mundo e os modelos necesarios para a simulación. Ademais deberá conter o nodo de Python onde se executan certas funcionalidades do modelo. Para facilitar esta tarefa pódese partir do ficheiro de lanzamento básico co que conta o paquete
2. Iniciar un novo terminal, entrar no *workspace* e introducir o comando *"source devel/setup.bash"*.
3. Lanzar a simulación introducindo o comando "*roslaunch robobo nome.launch"* no terminal.

Tras a realización destes pasos, ábrese unha xanela nova de Gazebo para a simulación, tal e como se aprecia na Figura 34, e xéranse os temas e servizos de ROS que permiten controlar o robot. Cada vez que se desexe acceder a estes controis mediante un terminal debe repetirse o paso 2.

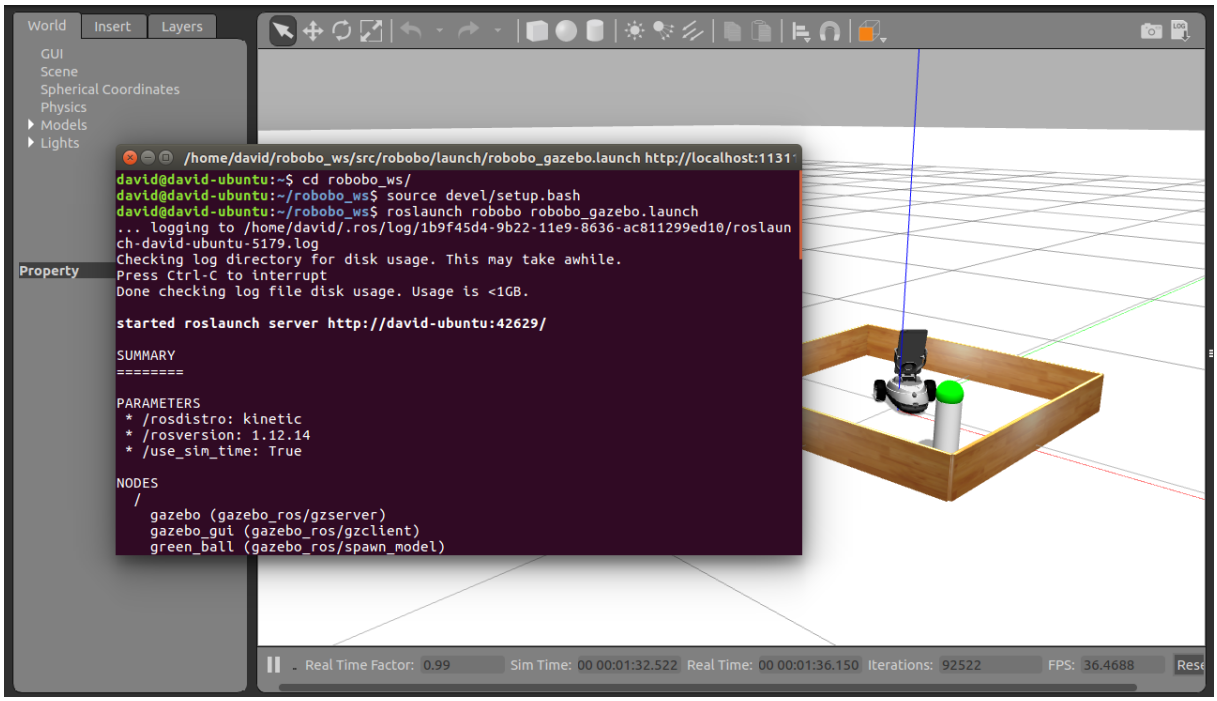


Figura 34. Representación gráfica do lanzamento dunha simulación

No caso de precisar introducir varios Robobo na simulación, basta con lanzar tantos nodos como robots se precisen con dito modelo. Estes nodos descríbese no ficheiro de lanzamento, e cada un deles debe ter un nome diferente.

7.2 Movemento das rodas

Co fin de corroborar que o desprazamento da plataforma robótica é o mesmo que no caso do robot real, realizáronse unha serie de probas mediante o comando *moveWheels*. Para esta comprobación partiuse dos mesmos valores experimentais dos que se dispoñía para a calibración dos motores das rodas, polo que só foi necesario realizar os experimentos no simulador. A proba consistiu en mover o robot en liña recta a unha velocidade e durante un tempo especificados. Na Táboa 11 recóllense os resultados deste experimento.

Velocidade	Tempo (s)	Distancia percorrida (cm)	
		Robot real	Robot simulado
2	1	3.0	2.4
2	3	10.2	9.6
10	1	4.1	4.5
10	3	16.8	17.4
60	1	21.7	21.4
60	3	72.0	70.5
90	1	30.5	29.9
90	3	101.0	100.0

Táboa 11. Comprobación dos desprazamentos da base robótica

Como se pode apreciar, ambos valores son moi semellantes, grazas ao axuste realizado.

7.3 Movemento do pan e o tilt

Ao igual que no caso anterior, a comprobación dos movementos do pan e do tilt realizouse partindo dos resultados das medicións do comportamento do robot real das que se dispoñía para a calibración. A comparación realizase entre o tempo que tardan ambos modelos en alcanzar unha posición determinada a unha velocidade determinada. Os resultados obtidos para o pan e o tilt móstranse na Táboa 12 e na Táboa 13, respectivamente.

Velocidade	Desprazamento de 45º		Desprazamento de 90º	
	Robot real	Robot simulado	Robot real	Robot simulado
2	4.55	4.77	9.39	9.4
6	3.57	3.6	7.00	6.99
10	2.84	2.91	5.95	5.57
30	1.52	1.53	2.82	2.78
60	1.04	1.00	1.67	1.68
80	0.88	0.89	1.41	1.42

Táboa 12. Comprobación dos desprazamentos do pan

Velocidade	Desprazamento de 45º		Desprazamento de 90º	
	Robot real	Robot simulado	Robot real	Robot simulado
2	9.87	10.9	20.67	21.17
6	7.03	7.69	14.43	14.85
10	5.8	6.59	11.83	11.63
30	3.13	3.15	6.08	5.96
60	1.91	1.9	3.68	3.62
80	1.54	1.59	2.84	2.97

Táboa 13. Comprobación dos desprazamentos do tilt

No caso destes dous elementos os resultados volven ser satisfactorios.

7.4 IR

Para a comprobación dos sensores infravermellos realizouse unha nova medición co robot real para o sensor *front_l*, coa intención de comprobar un dos sensores inclinados, para os cales non se realizou unha calibración específica. O experimento levouse a cabo do

mesmo xeito que para a calibración realizada co sensor central. A Figura 35 mostra a disposición dos modelos para realizar as medicións oportunas.

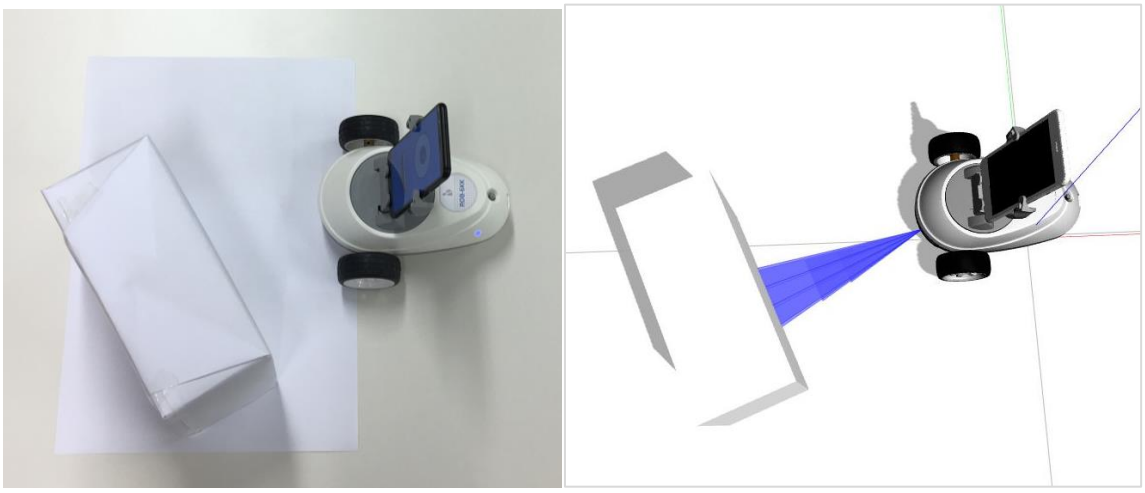


Figura 35. Disposición para a medición do sensor *front_l* en ambos modelos

Os resultados obtidos móstranse na Táboa 14. Os resultados obtidos no modelo simulado acércanse bastante aos do robot real. Débese ter en conta que o valor devolto polos sensores infravermellos é moi variante en función das diferentes reflectividades dos materiais, polo que o importante é que a resposta dinámica do sensor real e simulado sexan parecidas, de forma que o valor de intensidade devolto creza ou decreza nun rango similar, de forma que os programas realizados no simulador poidan ser axustados de maneira sinxela na realidade en función do material utilizado.

Distancia (cm)	IR real	IR simulado
5	435	395
10	138	117
15	80	70
20	62	57
25	53	52
30	49	49
35	47	48

Táboa 14. Comprobación dun sensor infravermello inclinado

7.5 Proba global

O principal obxectivo deste traballo é a obtención dun modelo en simulación de Robobo que poida ser utilizado no defecto do robot real. Quere isto dicir que, no caso de non alcanzar un modelo de simulación cuxos comportamentos sexan iguais ou moi próximos ao modelo real, non se cumpriría o principal requisito deste TFM. Para poder corroborar a utilidade do devandito modelo de simulación é necesario realizar unha comprobación de que, baixo as mesmas ordes, o robot virtual teña un comportamento similar ao robot real. Con este fin, elaborouse un programa en Python que será executado en ambos modelos e que requirirá que o robot utilice varios dos seus sensores, como a cámara e os sensores

infravermellos, ademais dos catro motores dos que dispón a base robótica. O código deste programa está dispoñible no seguinte enlace: https://github.com/david-casal/robobo_gazebo.

7.5.1 Descrición da proba

A proba consiste en situar o robot Robobo no interior dun recinto de madeira cadrado de 80 cm de lado, ademais dunha pelota de cor verde, situada sobre dun soporte o suficientemente alto para que a cámara do robot a poida alcanzar. Dentro desta disposición ordenaráselle ao robot cumprir dous obxectivos:

1. Evitar choques: o robot deberá avanzar en liña recta ata que detecte unha parede e, en función de que sensores a detecten, deberá xirar aproximadamente 90º cara un lado ou cara o outro, e avanzar de novo.
2. Buscar a pelota verde: no momento en que o robot detecte unha pelota de cor verde mediante a cámara, deberá dirixirse cara ela e pararse xusto en fronte da mesma.

Na Figura 36 móstrase a disposición dos diferentes obxectos para a realización desta proba, tanto na realidade como no simulador.

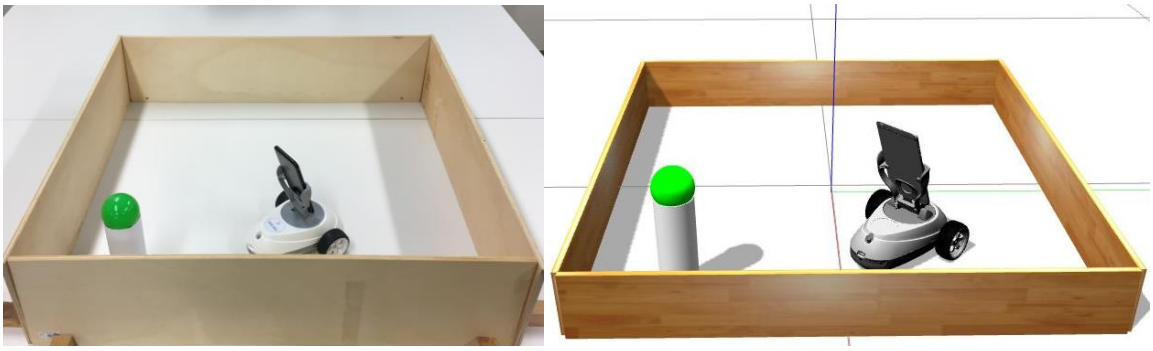
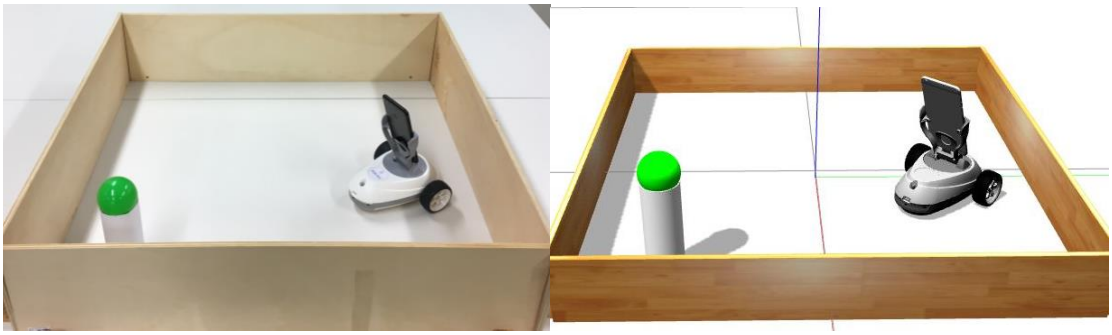


Figura 36. Montaxe da proba global

7.5.2 Resultados

Unha vez realizada a proba os resultados foron moi semellantes. Ambos modelos conseguiron alcanzar a pelota de cor verde sen colidir con ningunha das paredes, realizando traxectorias moi semellantes. Cabe destacar que sería prácticamente imposible que realizasen exactamente os mesmos movementos, xa que durante o desenvolvemento desta proba intervellen numerosos factores, tales como a posición e orientación exacta de partida ou unha pequena variación ao realizar o xiro xa provocan cambios importantes na traxectoria que seguen.



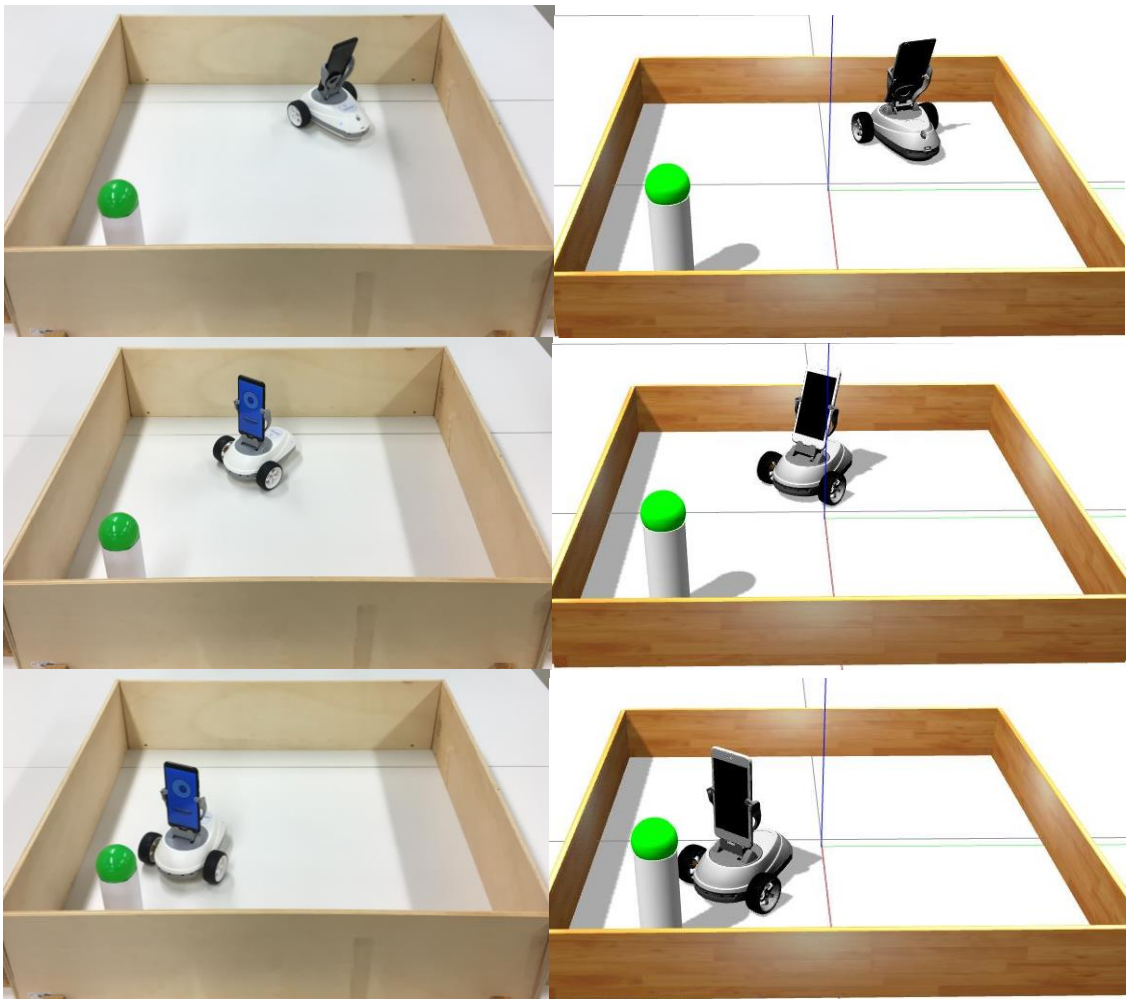


Figura 37. Traxectoria seguida por ambos modelos na proba global

Na Figura 37 móstrase a traxectoria que seguiron o modelo de simulación e o Robobo real durante a execución desta proba. Finalmente, cando recoñecen a pelota verde e se paran enfronte dela, as imaxes que mostraban as cámaras eran as da Figura 38.

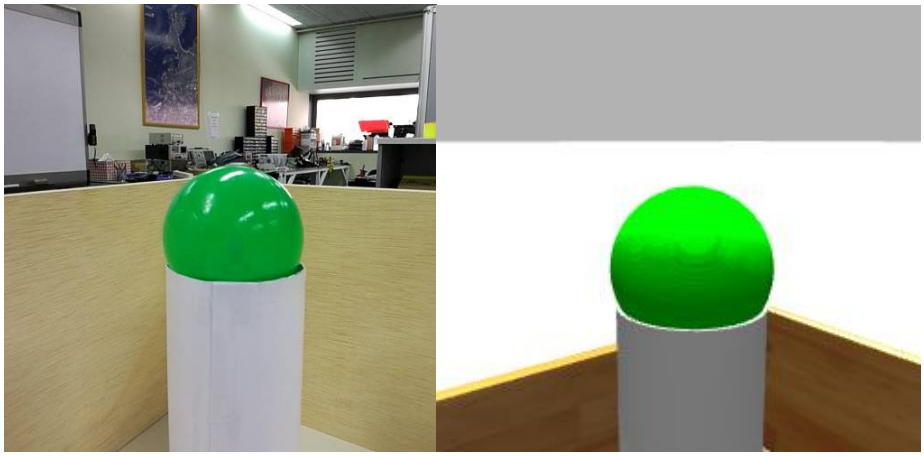


Figura 38. Comparación das imaxes das cámaras de ambos modelos de Robobo

8 CONCLUSIÓNS E TRABALLO FUTURO

Unha vez comprobado o funcionamento do modelo de simulación, pódense sacar as seguintes conclusións acerca das diferentes tarefas desenvolvidas neste traballo:

- **Modelo físico do robot:** creouse un modelo de Robobo acorde ás especificacións do robot real, e que conta cunha definición precisa das súas características que ofrece un comportamento dinámico apropiado e acorde ao real.
- **Caracterización do comportamento dos motores**: contando coas calibracións existentes conseguiuse unha definición dos catro motores que compoñen a base robótica, de forma que ofrecen uns resultados de comportamento máis que satisfactorios.
- **Implementación dos encoders**: grazas ás funcións con que conta a API de Gazebo foi relativamente sinxelo crear un plugin que ofrecese a posición de cada *joint* cos que conta o modelo. Isto permite coñecer o valor exacto da posición de cada roda ao igual que na base robótica real.
- **Deseño e caracterización dos sensores infravermellos:** conseguiuse obter un deseño eficiente dos sensores infravermellos que ofrecen uns valores razoablemente semellantes aos reais, dotando ao modelo de simulación de funcionalidade de cara á súa utilización.
- **Implementación da cámara:** foi implementada a cámara frontal do smartphone no modelo de simulación, conseguindo obter imaxes do mundo simulado de xeito semellante ao robot real.

Ademais, o modelo foi elaborado para que puidese ser programado da mesma maneira que o Robobo real, para poder empregar o mesmo código en calquera dos dous modelos sen a necesidade de modificalo.

Finalmente foi realizada unha proba na que se combinaron o uso de varios sensores e de varios actuadores, de xeito que se fixo uso de gran parte das utilidades do robot para a solución de dúas tarefas, non chocar e atopar a pelota verde. Esta proba foi resolta de xeito satisfactorio, corroborando que o modelo de simulación conta cun comportamento afín ao real.

Como conclusión, tras a finalización deste traballo obtívose un modelo de simulación do Robobo programable en ROS, completamente compatible co sistema de programación do robot real e que ofrece uns comportamentos moi similares ao mesmo. Polo tanto, cumpriuse co obxectivo de contar cun modelo de simulación que poida substituír ao Robobo en labores de investigación e na docencia de robótica a niveis superiores de educación.

8.1 Traballo futuro

Unha vez finalizado o traballo, aínda que nas comprobacións realizadas se obtiveran resultados satisfactorios, é mais que probable que o modelo poida ter algún defecto. Estes defectos xeralmente saen á luz durante a súa utilización, por iso é de vital importancia levar conta de todas as incidencias que poidan ocorrer con este modelo e tratar de solucionalas, para finalmente obter como resultado unha versión estable e fiable coa que traballar.

Por outro lado, existen unha serie de tarefas que sería interesante que fosen levadas a cabo:

- Adaptación do modelo a unha versión mais recente de Gazebo, xa que sempre é recomendable poder traballar coas últimas versións estables.
- Engadir o resto de sensores cos que conta un smartphone e que son utilizados para a programación de Robobo.
- Realizar unha migración a ROS 2, que é unha versión nova de ROS que neste momento se encontra en desenvolvemento, polo que pronto acabará substituíndo á versión antiga.
- Deseñar mundos de simulación estándares, como por exemplo unha mesa con diferentes obxectos, coa intención de facilitar o seu uso en usuarios que descoñezan o funcionamento deste simulador ou o formato SDF.
- Engadir un parámetro que permitise modificar a calibración dos sensores infravermellos en función do tipo de material dunha forma xenérica, de xeito que o usuario puidese escoller entre un listado de materiais predefinidos, tales como madeira, plástico, metal, etc.

REFERENCIAS

[1] F. Bellas *et al.*, “The Robobo Project: Bringing Educational Robotics Closer to Real-World Applications,” *RIE*, pp. 226–237, 2017.

[2] “ROS.org | Powering the world’s robots.” [Online]. Available: https://www.ros.org/. [Accessed: 25-Jun-2019].

[3] F. Bellas *et al.*, “Robobo: The Next Generation of Educational Robot,” *Robot 2017 Third Iber. Robot. Conf. Adv. Intell. Syst. Comput.*, vol. 694, pp. 359–369, 2018.

[4] “es - ROS Wiki.” [Online]. Available: http://wiki.ros.org/es. [Accessed: 25-Jun-2019].

[5] “Gazebo.” [Online]. Available: http://gazebosim.org/. [Accessed: 25-Jun-2019].

[6] “Open Dynamics Engine.” [Online]. Available: https://www.ode.org/. [Accessed: 25-Jun-2019].

[7] “blender.org - Home of the Blender project - Free and Open 3D Creation Software.” [Online]. Available: https://www.blender.org/. [Accessed: 25-Jun-2019].

[8] “SDF Home.” [Online]. Available: http://sdformat.org/. [Accessed: 25-Jun-2019].

[9] T. Lei and L. Ming, “A robot exploration strategy based on Q-learning network,” *2016 IEEE Int. Conf. Real-Time Comput. Robot. RCAR 2016*, pp. 57–62, 2016.

[10] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, “Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo,” no. August, pp. 1–6, 2016.

[11] N. Kumar, Z. Vamossy, and Z. M. Szabo-Resch, “Robot obstacle avoidance using bumper event,” *SACI 2016 - 11th IEEE Int. Symp. Appl. Comput. Intell. Informatics, Proc.*, pp. 485–490, 2016.

[12] N. Kumar, Z. Vamossy, and Z. M. Szabo-Resch, “Robot path pursuit using probabilistic roadmap,” *CINTI 2016 - 17th IEEE Int. Symp. Comput. Intell. Informatics Proc.*, pp. 139–144, 2017.

[13] C. Bensaci *et al.*, “Nonlinear Control of a differential wheeled mobile robot in real time-Turtlebot 2 To cite this version : HAL Id : hal-02014895 Nonlinear Control of a Differential Wheeled Mobile Robot in Real Time -Turtlebot 2,” 2019.

[14] M. Bonani *et al.*, “The e-puck, a Robot Designed for Education in Engineering,” *Proc. 9th Conf. Autnomous Robot Syst. Compet.*, vol. 1, no. 1, pp. 59–65, 2009.

[15] “Robot Virtual Worlds | ROBOTC | VEX & NXT Simulator.” [Online]. Available: http://www.robotvirtualworlds.com/. [Accessed: 25-Jun-2019].

[16] S. Palma Morena, “Desarrollo de un modelo computacional de simulación para el sistema robótico Lego Mindstorms EV3,” Tecnológico de Costa Rica, 2018.

[17] “NAO the humanoid robot | SoftBank Robotics EMEA.” [Online]. Available: https://www.softbankrobotics.com/emea/en/nao. [Accessed: 25-Jun-2019].

[18] S. Michieletto, D. Zanin, and E. Menegatti, “NAO robot simulation for service robotics purposes,” *Proc. - UKSim-AMSS 7th Eur. Model. Symp. Comput. Model. Simulation, EMS 2013*, pp. 477–482, 2013.

[19] B. Menéndez, R. Salamanqués, and J. Canas, “Programming of a Nao humanoid in Gazebo using Hierarchical FSM,” *Svn.Jderobot.Org*, vol. 1, no. 1, 2013.

[20] J. M. Cañas, L. Martín, and J. Vega, “Innovating in robotics education with Gazebo simulator and JdeRobot framework,” no. 1, 2014.

[21] M. Corsini, P. Cignoni, and R. Scopigno, “Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes,” *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 6, pp. 914–924, Jun. 2012.

[22] “mytechia / robobo-programming / wiki / Home — Bitbucket.” [Online]. Available: https://bitbucket.org/mytechia/robobo-programming/wiki/Home. [Accessed: 26-Jun-2019].

[23] “GitHub - mintforpeople/robobo-ros-msgs: Repository for the ROS package that contains the messages and services of the Robobo educational robot.” [Online]. Available: https://github.com/mintforpeople/robobo-ros-msgs/. [Accessed: 26-Jun-2019].

[24] R. Boado, “Desarrollo de un modelo en simulación en V-REP de un robot móvil basado en smartphone y soporte al lenguaje Python,” Universidade da Coruña, 2019.

ANEXOS

Anexo I: Descrición do modelo

```
<?xml version="1.0" ?>
<sdf version="1.6">
    <model name="Robobo">
        <static>0</static>
        <!-- LINKS -->
        <link name="body">
            <pose>-0.055984 0.034729 0.0308 0 0 0</pose>
            <inertial>
                <mass>0.48</mass>
                <inertia>
                    <ixx>0.000485229</ixx>
                    <ixy>0.000000058</ixy>
                    <ixz>0.000013696</ixz>
                    <iyy>0.001227181</iyy>
                    <iyz>-0.000000029</iyz>
                    <izz>0.001452813</izz>
                </inertia>
            </inertial>
            <collision name="body_collision">
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/body_collision.dae
                        </uri>
                    </mesh>
                </geometry>
                <surface>
                    <friction>
                        <ode>
                            <mu>0.2</mu>
                            <mu2>0.2</mu2>
                        </ode>
                    </friction>
                    <contact>
                        <poissons_ratio>0.46</poissons_ratio>
                        <elastic_modulus>7e8</elastic_modulus>
                        <ode>
                            <min_depth>0.0004</min_depth>
                        </ode>
                    </contact>
                </surface>
            </collision>
            <visual name="body_visual">
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/body_visual.dae
                        </uri>
                    </mesh>
```

```
                </geometry>
            </visual>
            <!--IR SENSORS-->
            <sensor type="ray" name="front_c">
                <pose>-0.08 0 0.0012 0 0 3.1415</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.12</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="front_l">
                <pose>-0.0727 -0.026 0.0012 0 0.21 3.526</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.085</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="front_ll">
                <pose>-0.0625 -0.0405 0.0012 0 0 3.927</pose>
                <visualize>false</visualize>
                <ray>
```

```
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.12</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="front_r">
                <pose>-0.0727 0.026 0.0012 0 0.21 2.758</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.085</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="front_rr">
                <pose>-0.0625 0.0405 0.0012 0 0 2.356</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
```

```
                            <samples>4</samples>
                            <min_angle>-0.12</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="back_c">
                <pose>0.12 0 0 0 0.21 0</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.085</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="back_l">
                <pose>0.1055 -0.0308 0.001 0 0 -0.524</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.12</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
```

```
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
            <sensor type="ray" name="back_r">
                <pose>0.1055 0.0308 0.001 0 0 0.524</pose>
                <visualize>false</visualize>
                <ray>
                    <scan>
                        <horizontal>
                            <samples>4</samples>
                            <min_angle>-0.2</min_angle>
                            <max_angle>0.17</max_angle>
                        </horizontal>
                        <vertical>
                            <samples>4</samples>
                            <min_angle>-0.12</min_angle>
                            <max_angle>0.16</max_angle>
                        </vertical>
                    </scan>
                <range>
                    <min>0</min>
                    <max>0.4</max>
                </range>
                </ray>
                <plugin name="infrared_range"
filename="libinfrared_range.so">
                    <frameName> </frameName>
                    <fov>0</fov>
                </plugin>
            </sensor>
        </link>
        <link name="left_wheel">
            <pose>-0.082628 -0.038096 0.03225 0 0 0</pose>
            <inertial>
                <mass>0.046</mass>
                <inertia>
                    <ixx>0.000014552</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.000023921</iyy>
                    <iyz>0</iyz>
                    <izz>0.000014552</izz>
                </inertia>
            </inertial>
            <collision name="lw_collision">
                <pose>0 0 0 1.5708 0 0</pose>
                <geometry>
                    <cylinder>
                        <radius>0.03225</radius>
                        <length>0.026</length>
                    </cylinder>
                </geometry>
                <surface>
                    <friction>
                        <ode>
```

```
                            <mu>0.8</mu>
                            <mu2>0.8</mu2>
                        </ode>
                    </friction>
                    <contact>
                        <poissons_ratio>0.5</poissons_ratio>
                        <elastic_modulus>7e6</elastic_modulus>
                        <ode>
                            <min_depth>0.0008</min_depth>
                        </ode>
                    </contact>
                </surface>
            </collision>
            <visual name="lw_visual">
                <pose>0 -0.0024 0 0 0 0</pose>
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/left_wheel_visual.dae
                        </uri>
                    </mesh>
                </geometry>
            </visual>
        </link>
        <link name="right_wheel">
            <pose>-0.082628 0.10746 0.03225 0 0 0</pose>
            <inertial>
                <mass>0.046</mass>
                <inertia>
                    <ixx>0.000014552</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.000023921</iyy>
                    <iyz>0</iyz>
                    <izz>0.000014552</izz>
                </inertia>
            </inertial>
            <collision name="rw_collision">
                <pose>0 0 0 1.5708 0 0</pose>
                <geometry>
                    <cylinder>
                        <radius>0.03225</radius>
                        <length>0.026</length>
                    </cylinder>
                </geometry>
                <surface>
                    <friction>
                        <ode>
                            <mu>0.8</mu>
                            <mu2>0.8</mu2>
                        </ode>
                    </friction>
                    <contact>
                        <poissons_ratio>0.5</poissons_ratio>
                        <elastic_modulus>7e6</elastic_modulus>
                        <ode>
                            <min_depth>0.0008</min_depth>
                        </ode>
                    </contact>
                </surface>
```

```
            </collision>
            <visual name="rw_visual">
                <pose>0 0.0024 0 0 0 0</pose>
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/right_wheel_visual.dae
                        </uri>
                    </mesh>
                </geometry>
            </visual>
        </link>
        <link name="pan">
            <pose>-0.059743 0.034729 0.060151 0 0 3.14159</pose>
            <inertial>
                <mass>0.077</mass>
                <inertia>
                    <ixx>0.000037326</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.000037326</iyy>
                    <iyz>0</iyz>
                    <izz>0.000074536</izz>
                </inertia>
            </inertial>
            <collision name="pan_collision">
                <geometry>
                    <cylinder>
                        <radius>0.044</radius>
                        <length>0.003</length>
                    </cylinder>
                </geometry>
            </collision>
            <visual name="pan_visual">
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/pan_visual.dae
                        </uri>
                    </mesh>
                </geometry>
            </visual>
        </link>
        <link name="tilt_smartphone">
            <pose>-0.149486 0.03499 0.0843 0 0 3.14159</pose>
            <inertial>
                <mass>0.18</mass>
                <inertia>
                    <ixx>0.000091682</ixx>
                    <ixy>0</ixy>
                    <ixz>0</ixz>
                    <iyy>0.000375824</iyy>
                    <iyz>0</iyz>
                    <izz>0.000465727</izz>
                </inertia>
            </inertial>
            <self_collide>1</self_collide>
            <collision name="tilt_collision">
                <geometry>
```

```
                    <box>
                        <size>0.1581 0.0778 0.0077</size>
                    </box>
                </geometry>
            </collision>
            <visual name="tilt_visual">
                <pose>-0.079749 -0.002249 -0.004338 0 0 0</pose>
                <geometry>
                    <mesh>
                        <uri>

model://robobo/models/robobo/meshes/tilt_smartphone_visual.dae
                        </uri>
                    </mesh>
                </geometry>
            </visual>
            <sensor type="camera" name="front_camera">
                <pose>0.0695 0.0105 0.004 0 -1.5708 3.1415</pose>
                <visualize>false</visualize>
                <camera>
                    <image>
                        <width>250</width>
                        <high>320</high>
                    </image>
                </camera>
                <plugin name="camera_controller"
filename="libgazebo_ros_camera.so">
                    <cameraName>robobo</cameraName>
                </plugin>
            </sensor>
        </link>
        <!-- JOINTS -->
        <joint name="left_motor" type="revolute">
            <parent>body</parent>
            <child>left_wheel</child>
            <pose>0 0 0 0 0 0</pose>
            <axis>
                <xyz>0 -1 0</xyz>
                <dynamics>
                    <friction>0.022</friction>
                </dynamics>
            </axis>
        </joint>
        <joint name="right_motor" type="revolute">
            <parent>body</parent>
            <child>right_wheel</child>
            <pose>0 0 0 0 0 0</pose>
            <axis>
                <xyz>0 -1 0</xyz>
                <dynamics>
                    <friction>0.022</friction>
                </dynamics>
            </axis>
        </joint>
        <joint name="pan_motor" type="revolute">
            <parent>body</parent>
            <child>pan</child>
            <pose>0 0 0 0 0 0</pose>
            <axis>
                <xyz>0 0 -1</xyz>
                <dynamics>
```

```
                    <friction>0.15</friction>
                </dynamics>
            </axis>
        </joint>
        <joint name="tilt_motor" type="revolute">
            <parent>pan</parent>
            <child>tilt_smartphone</child>
            <pose>-0.089743 0 -0.005 0 0 0</pose>
            <axis>
                <xyz>0 -1 0</xyz>
                <dynamics>
                    <friction>0.22</friction>
                </dynamics>
            </axis>
        </joint>
        <!-- PLUGINS OF MODEL-->
        <plugin name="move_wheels" filename="libmove_wheels.so"/>
        <plugin name="move_pan_tilt" filename="libmove_pan_tilt.so"/>
        <plugin name="encoders" filename="libencoders.so"/>
    </model>
</sdf>
```

Anexo II: Plugin dos encoders

```
#ifndef _ENCODERS_HH_
#define _ENCODERS_HH_

#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/transport/transport.hh>
#include <gazebo/msgs/msgs.hh>
#include <gazebo/physics/Joint.hh>
#include <gazebo/physics/Model.hh>
#include <gazebo/physics/PhysicsTypes.hh>
#include <std_msgs/Int16.h>
#include <gazebo_msgs/LinkStates.h>

#include <thread>
#include "ros/ros.h"
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include "std_msgs/Float64.h"
#include "robobo_msgs/Wheels.h"

namespace gazebo
{

    class Encoders : public ModelPlugin
    {

        public: Encoders() {}

        private: ros::Publisher pubWheels;
        private: ros::Publisher pubPan;
        private: ros::Publisher pubTilt;
        private: robobo_msgs::Wheels msgWheels;
        private: std_msgs::Int16 msgPan;
        private: std_msgs::Int16 msgTilt;
        private: int RWPos;
        private: int RWVel;
        private: int LWPos;
        private: int LWVel;
        private: int pan;
        private: int tilt;

        //public: physics::JointControllerPtr jointController;

        public: virtual void Load(physics::ModelPtr _model, sdf::ElementPtr
_sdf)
        {
            // Safety check
            if (_model->GetJointCount() == 0)
            {
            std::cerr << "Invalid joint count, model not loaded\n";
            return;
            }

            this->model = _model;

            // Initialize ros
```

```
            if (!ros::isInitialized())
            {
                int argc = 0;
                char **argv = NULL;
                ros::init(argc, argv, "gazebo_client",
ros::init_options::NoSigintHandler);
            }

            //Create node handler
            this->rosNode.reset(new ros::NodeHandle("gazebo_client"));

            // Subscribe to topic
            ros::SubscribeOptions so =
ros::SubscribeOptions::create<gazebo_msgs::LinkStates>("/gazebo/link_states
",1,
                boost::bind(&Encoders::Callback, this, _1),ros::VoidPtr(),
&this->rosQueue);

            // Create topics to publish
            this->pubWheels = this->rosNode-
>advertise<robobo_msgs::Wheels>("/" + this->model->GetName()
                + "/wheels", 1);
            this->pubPan = this->rosNode->advertise<std_msgs::Int16>("/" +
this->model->GetName() + "/pan", 1);
            this->pubTilt = this->rosNode->advertise<std_msgs::Int16>("/" +
this->model->GetName() + "/tilt", 1);

            // Declare the node as a subscriber
            this->rosSub = this->rosNode->subscribe(so);

            this->rosQueueThread = std::thread(std::bind
(&Encoders::QueueThread, this));
        }

        public: void Callback(const gazebo_msgs::LinkStates::ConstPtr& msg)
        {
            // Read and transform values to degrees of right wheel joint
            RWPos = int (round(this->model->GetJoint("right_motor")-
>GetAngle(0).Radian() * 180 / M_PI));
            RWVel = int (round(this->model->GetJoint("right_motor")-
>GetVelocity(0) * 180 / M_PI));

            // Read and transform values to degrees of left wheel joint
            LWPos = int (round(this->model->GetJoint("right_motor")-
>GetAngle(0).Radian() * 180 / M_PI));
            LWVel = int (round(this->model->GetJoint("right_motor")-
>GetVelocity(0) * 180 / M_PI));

            // Save data in Wheels msg
            this->msgWheels.wheelPosR.data = RWPos;
            this->msgWheels.wheelSpeedR.data = RWVel;
            this->msgWheels.wheelPosL.data = LWPos;
            this->msgWheels.wheelSpeedL.data = LWVel;

            // Publish msg in topic
            this->pubWheels.publish(this->msgWheels);

            // Read an transform values to degrees of pan and tilt
            pan = int (round(this->model->GetJoint("pan_motor")-
>GetAngle(0).Radian() * 180 / M_PI));
            tilt = int (round(this->model->GetJoint("tilt_motor")-
```

```
>GetAngle(0).Radian() * 180 / M_PI));

            // Save data in msg
            this->msgPan.data = pan;
            this->msgTilt.data = tilt;

            // Publish msg in topic
            this->pubPan.publish(this->msgPan);
            this->pubTilt.publish(this->msgTilt);
        }

        private: void QueueThread()
        {
            static const double timeout = 0.01;
            while (this->rosNode->ok())
            {
                this->rosQueue.callAvailable(ros::WallDuration(timeout));
            }
        }

        /// \brief Pointer to the model.
        private: physics::ModelPtr model;

        /// \brief Pointer to the joint.
        private: physics::JointPtr joint;

        /// \brief A node use for ROS transport
        private: std::unique_ptr<ros::NodeHandle> rosNode;

        /// \brief A ROS subscriber
        private: ros::Subscriber rosSub;

        //public: event::ConnectionPtr updateConnection;

        /// \brief A ROS callback queue that helps process messages
        private: ros::CallbackQueue rosQueue;

        /// \brief A thread the keeps running the rosQueue
        private: std::thread rosQueueThread;

    };

    // Tell Gazebo about this plugin, so that Gazebo can call Load on this
plugin.
    GZ_REGISTER_MODEL_PLUGIN(Encoders)
}
#endif
```