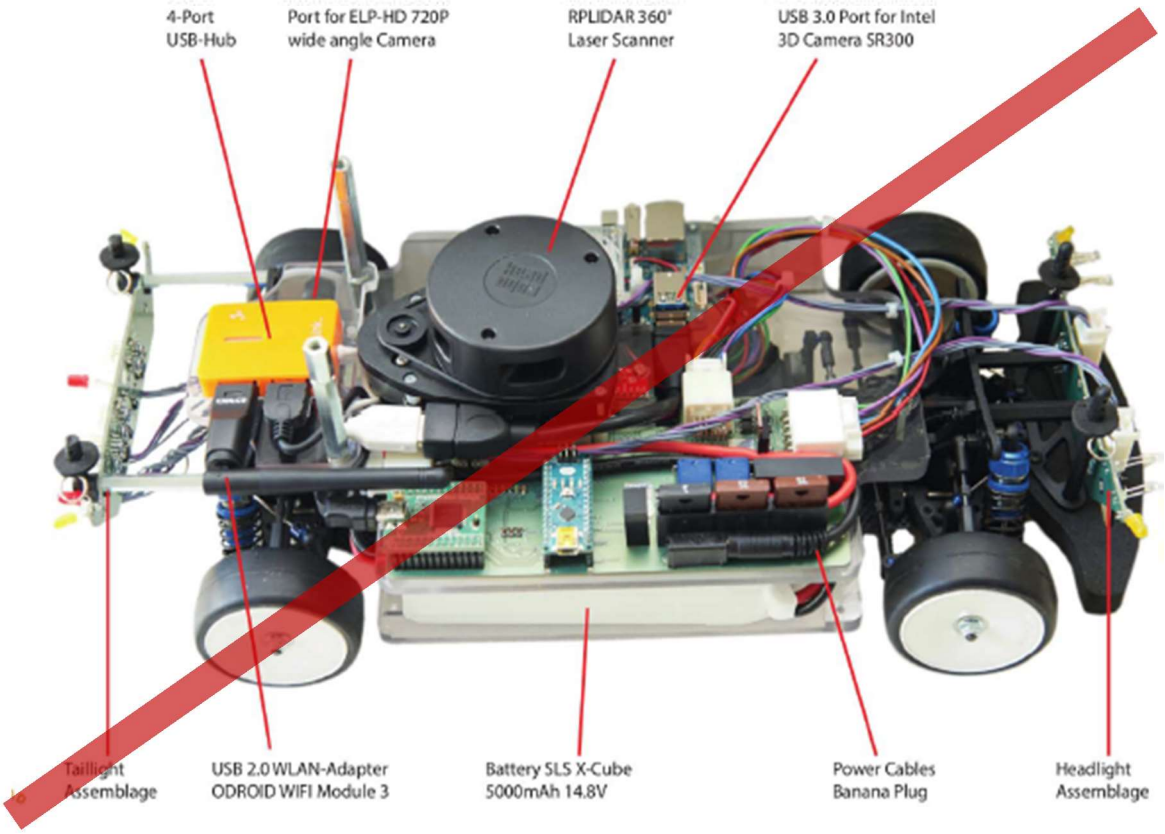
AMPLIACIÓN DE ROBÓTICA
4º GRADO EN ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA
CURSO 2016-2017

SEAT driving challenge



Apellidos	Nombre
Gamero Gómez	Francisco
Pérez Molina	Felipe
Domínguez Bermúdez	Juan Luis
Jiménez León	Mario
Garijo Campos	Alfonso

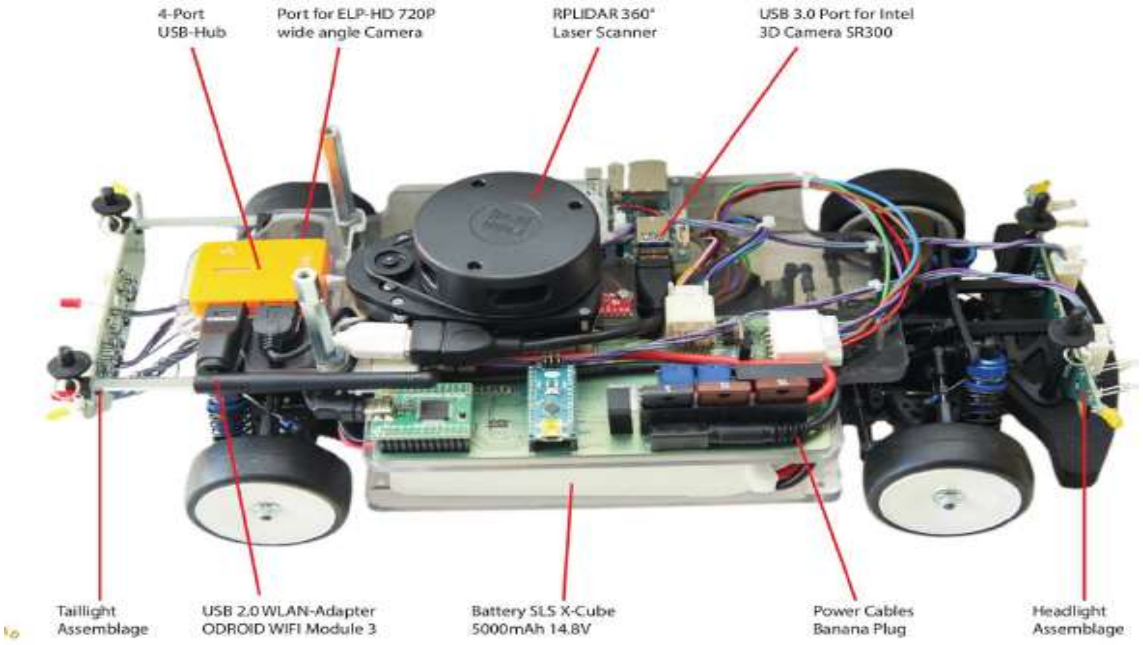
ÍNDICE

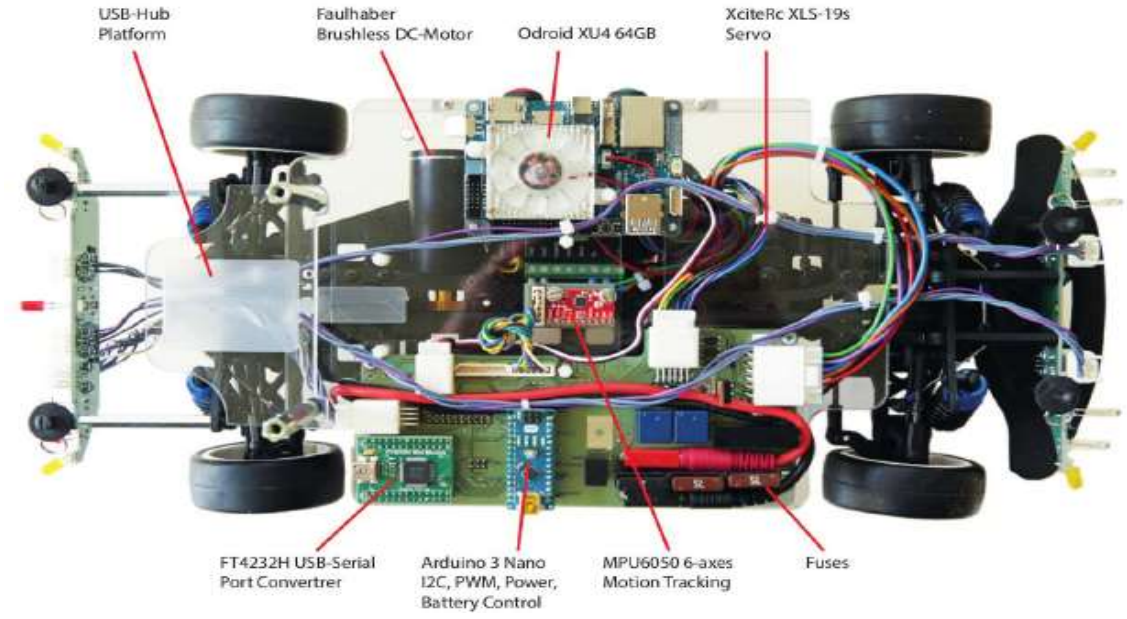
1. Introducción

1.1 Objetivo inicial

El proyecto escogido se corresponde con el desafío: “SEAT driving challenge”. Para llevarlo a cabo se proporcionarán al grupo las funciones implementables en el robot así como su hardware cuando éste esté disponible.

Este es el robot en cuestión:



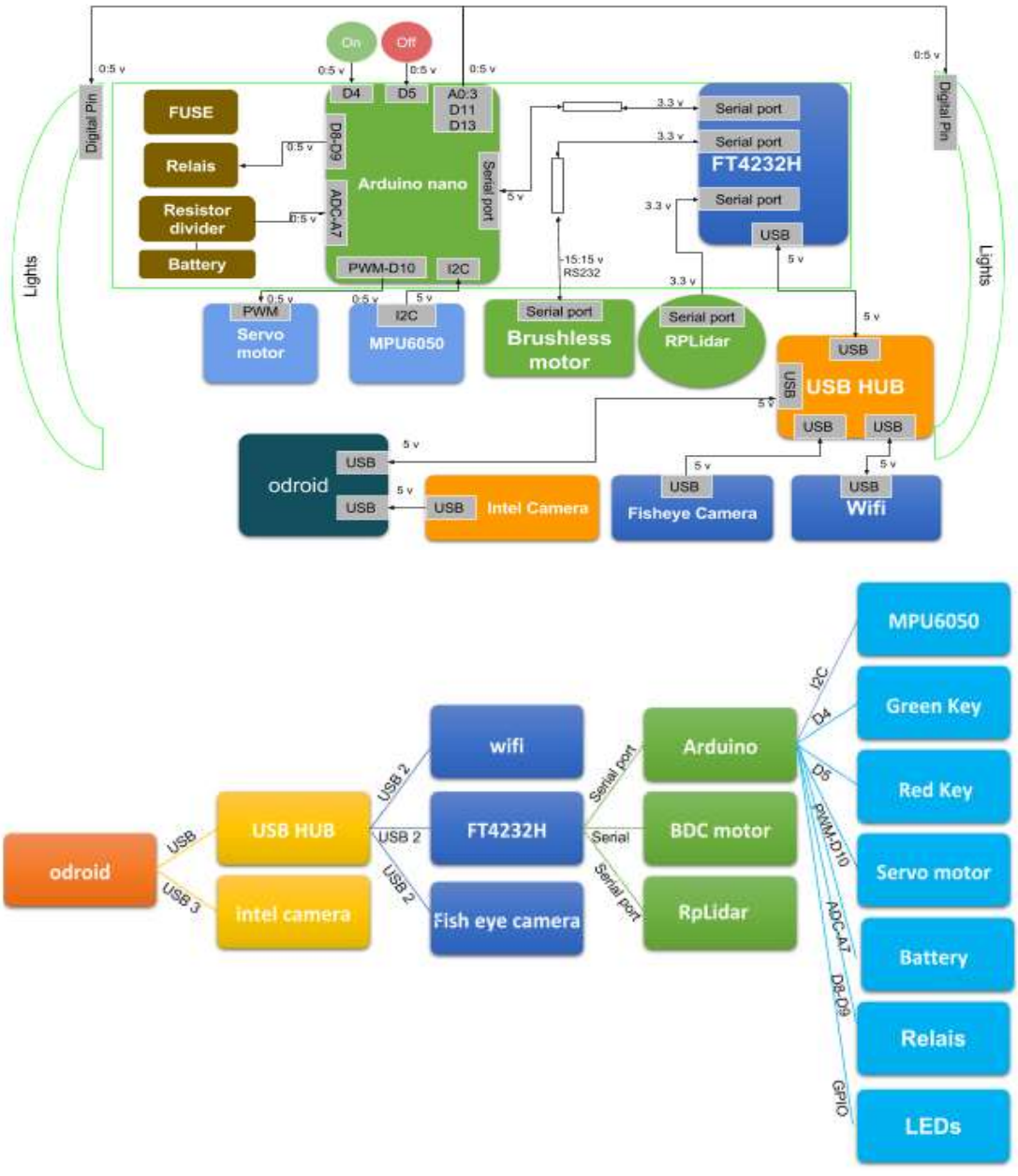


1.1.1 Sensores

Los sensores de los que dispone el coche son, como se ha podido observar en el esquema:

1. Un lídar 360º
2. Una cámara 3D Intel SR300
3. Una cámara ELP-HD 720p con gran angular
4. MPU60 50 6-axes Motion Tracking

1.1.2 Esquema eléctrico



1.1.3 Descripción del reto

ÁREA DEL MAPA

12X12 METROS

CALLES Y CIRCULACIÓN

Carreteras principalmente rectas

Habrá al menos dos curvas de 90º y varias más de diferentes ángulos

Se prevé que haya al menos una intersección

En el entorno habrá obstáculos tanto estáticos como móviles (como por ejemplo otro coche autónomo), y semáforos que se deberán respetar.

Challenge 1

Reconocer y recorrer de manera autónoma el mapa de la prueba. Para ello será necesario el uso de las cámaras y el lídar para reconocer el terreno, así como para geo-localizarse dentro de él.

Challenge 2

Ampliación del anterior en la que se introducirán obstáculos estáticos y móviles, los cuales habrá que respetar y evitar sin dejar de llevar a cabo la navegación por el mapa. Para detectarlos de nuevo serán necesarios los sensores mencionados.

Challenge 3

Ampliación del anterior en la que se introducirán semáforos y aparcamientos que se deberán respetar/utilizar.

1.2 Problemas hardware

Desde un principio se dependía del robot móvil en cuestión para aprender a programarlo y comprobar el funcionamiento las distintas soluciones generadas para los retos propuestos.

El robot sería compartido con el grupo de ESIBot, ya que son ellos los que participan en el challenge oficial. Para coordinar nuestro grupo de trabajo con el suyo debía haber una reunión, que se fue retrasando hasta que se dio por perdida la oportunidad de trabajar con la maqueta de SEAT.

Dado que no ha sido posible usar la maqueta se ha tenido que recurrir a generar un robot con los sensores que iban a utilizarse y un mundo físico con carretera y señales de tráfico para poder llevar a cabo simulaciones mediante gazebo, lo cual significó bastante tiempo dedicado a tareas que no se sabían llevar a cabo y que no estaban previstas. Pero esto no fue lo único que lastró la realización del proyecto, también hay que tener en cuenta que al esperar al robot (que teníamos programado que a mediados de abril se sabría cuándo podría trabajarse sobre él), también se perdió mucho tiempo de trabajo que obligó a realizar los objetivos en un periodo de tiempo menor de lo esperado.

1.3 Objetivos alternativos

Los nuevos objetivos son:

- Creación de un robot diferencial
- Incorporación de dos cámaras y un láser al modelo
- Seguimiento de línea (recorrer la carretera), utilizando una cámara situada sobre el robot
- Detección de obstáculos mediante el láser
- Reconocimiento de una señal de stop mediante una cámara frontal y respuesta ante ella

2. Bloques de la solución

2.1 Software utilizado

Todo el proceso de creación del vehículo y el mundo, así como los controladores, algoritmos y nodos relacionados con el funcionamiento del robot se llevará a cabo en un workspace de ROS.

Para llevar a cabo las simulaciones y así comprobar su correcto funcionamiento se utilizara el programa Gazebo.

Se implementarán programas y algoritmos tanto a mano, siguiendo tutoriales, como descargados de repositorios en internet, y en cada caso se especificará como se ha hecho. En varios casos se utilizará OpenCV.

2.2 Diseño del robot

Aunque finalmente se utilizó un robot móvil previamente creado, al que se le hicieron algunas modificaciones y que contaba con los plugins y herramientas necesarias para extraer la información de sus sensores a través de ROS, se ha creído conveniente incluir en la memoria del proyecto los pasos que se siguieron para la creación del modelo del robot inicial (ya que físicamente es muy similar al utilizado) y que, por falta de tiempo no se pudo aprender a poner en funcionamiento. Esto se incluirá en el apartado 2.2.1.

Existen una gran cantidad de vehículos complejos ya creados en Internet y compatibles con Gazebo, sin embargo se decidió que el nuestro debía ser creado completamente por nosotros. Este, sería un sencillo robot formado por un cubo rectangular a modo de chasis, una esfera simulando una rueda “caster”, y dos cilindros de poca altura como ruedas.

2.2.1 Versión inicial

Una vez creado el directorio de trabajo, los pasos junto con el código correspondiente para el modelado del robot, pueden encontrarse detallados en la página web de Gazebo en la sección *tutorials*. A continuación se exponen de manera resumida:

Directorio del modelo

1. Creamos un archivo de configuración para nuestro modelo y copiamos en él el código XML presente en el tutorial.

2. Creamos un archivo .sdf e incluimos lo siguiente:

```
<?xml versión='1.0'?>
<sdf versión='1.4'>
<model name="my_robot">
</model>
</sdf>
```

Estos pasos, expresan los contenidos básicos para el modelo. El archivo .config describe el robot con algunos metadatos adicionales. El archivo .sdf contiene definiciones necesarias para crear instancias de un modelo denominado my_robot.

Construcción de la estructura del modelo

Para empezar de manera sencilla, se comienza creando un modelo *static,* es decir, que no se moverá. Con ello, el robot será creado en un lugar fijo y podremos ensamblar cada uno de sus componentes.

Los siguientes pasos se llevarán a cabo modificando el archivo .sdf creado anteriormente:

1. Hacer que nuestro modelo sea estático añadiendo <*static*>*true*</*static*>.

2. Añadir el chasis rectangular. El código XML que lo implementa se encuentra disponible en la web, simplemente crea una caja de tamaño 0.4 x 0.2 x 0.1 metros con una par de etiquetas que definen su comportamiento, *collision* y *visual*.
3. Una vez se tiene la base, se pueden crear los elementos de locomoción, empezando por una rueda *caster*. Esta, vendrá simulada por una esfera sin fricción y cuyo código nos permite definir su radio o su posición en el vehículo.

4. Ahora, colocar las ruedas. El código utilizado para generar la rueda izquierda puede utilizarse para la derecha simplemente cambiando los valores que definen su posición en el robot. Al ser cilindros, podemos elegir su radio y altura.

5. Una vez montado, se puede hacer el robot dinámico estableciendo el comando <*static*> como falso y añadiéndole un *joint* para cada una de las ruedas laterales. Las dos articulaciones giran alrededor del eje y.

6. Ya estaría el modelo creado. Se puede comprobar su movimiento ejecutando Gazebo y aumentando la fuerza aplicada a cada articulación en la pestaña *Force* a la derecha de la pantalla.

Modificación del chasis

Para dar una visión menos geométrica al vehículo se ha decidido utilizar una malla del modelo *pioneer2dx,* un vehículo disponible en las bases de datos de Gazebo.

Para ello simplemente se busca en el archivo .sdf la estructura *visual* de la caja que hace de chasis y lo sustituimos por lo siguiente:

```
<visual name='visual'?>
    <geometry>
      <mesh>
        <uri>model://pioneer2dx/meses/chassis.dae</uri>
      </mesh>
    </geometry>
  </visual>
```

Una vez incluido, será necesario modificar el tamaño del chasis para hacerlo compatible con el resto de componentes.

Adición de sensores

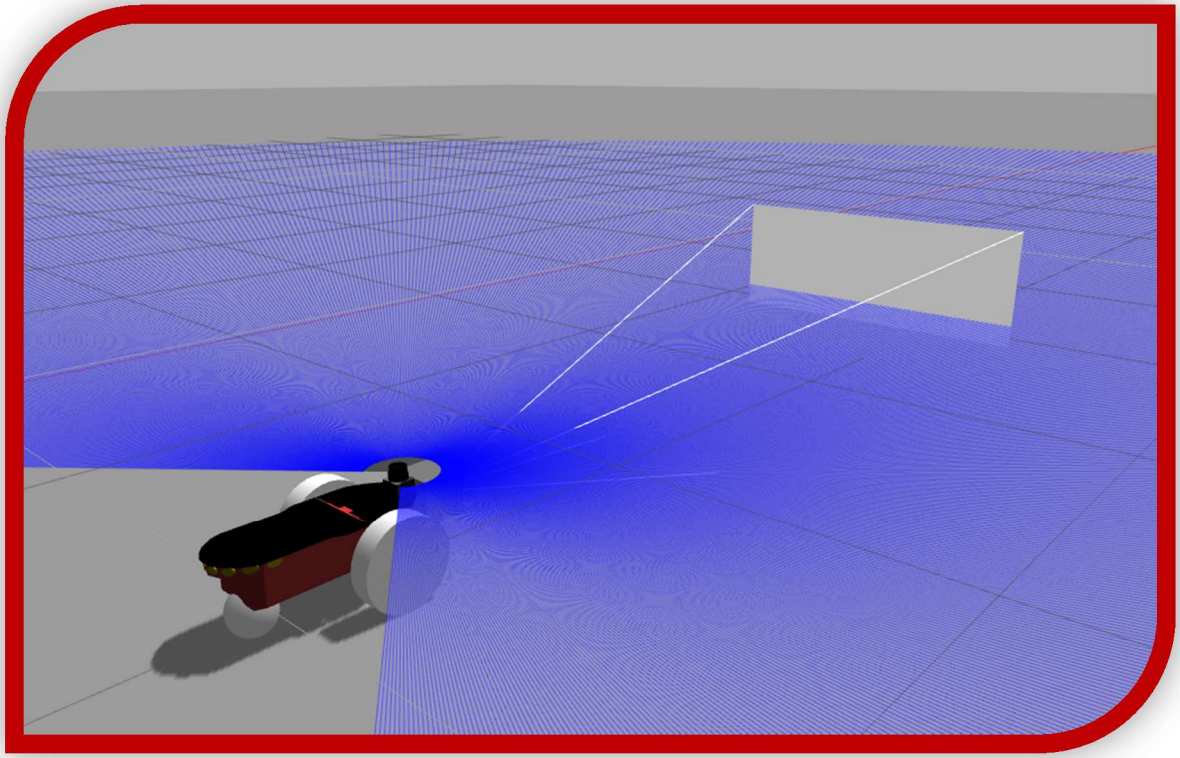
Una vez que tenemos nuestro vehículo creado con un chasis renovado, es hora de colocarle los sensores. Para nuestro proyecto necesitamos un sensor láser para poder detectar obstáculos y dos cámaras, una apuntando hacia el suelo para que el robot reciba información acerca de la línea que ha de seguir y otra apuntando hacia el frente para el reconocimiento de las señales.

1. Sensor láser: el sensor que utilizaremos se trata del modelo Hokuyo y añadirlo a nuestro robot es tan sencillo como definirlo en nuestro archivo .sdf añadiendo las siguientes líneas.

```
<include>
  <uri>model://hokuyo</uri>
  <pose>0.2 0.2 0 0 0</pose>
</include>
<joint name="hokuyo_joint" type="revolute">
  <child> hokuyo::link</child>
  <parent>chassis</parent>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <upper>0</upper>
      <lower>0</lower>
    </limit>
  </axis>
</joint>
```

2. Cámaras: incluir un módulo que actúe a modo de cámara es tan sencillo como lo hecho anteriormente con el láser y basta con incluir las características de la cámara en lugar de las del láser.

Como resultado, el modelo del robot que creamos como primera opción queda de la siguiente manera:



2.2.1 Versión definitiva

Mediante otro tutorial se consiguió crear un vehículo con la misma estructura que el descrito en el apartado anterior pero más sencillo de manejar a través de ROS. Para crearlo fueron necesarios 4 archivos:

- **macros.xacro:**
  Donde se definen los momentos de inercia y la dinámica de las distintas partes del modelo

- **materials.xacro:**
  Donde se definen los colores en formato RGBA que se utilizarán

- **mybot.xacro**

  Donde se definen los bloques físicos que conforman la figura del robot, se sitúan las uniones (joints), y se asocian las funciones de los sensores/actuadores a los bloques.

  - Por ejemplo, para crear una rueda será necesario crear un bloque visual
  - Luego definir su joint con el chasis
  - Y luego asignar el bloque al plugin de un sensor/actuador, cuyo funcionamiento vendrá definido en otro archivo.

- **mybot.gazebo:**

  Aquí se definen los plugins necesarios junto con sus características para controlar los actuadores y leer los sensores a la vez que se enlazan con los definidos en el archivo .xacro recientemente explicado

Estructura del robot:

El robot dispondrá, al igual que el anterior de un chasis con una rueda de castor en la parte posterior y de dos ruedas con tracción en la parte delantera que serán las que caracterizarán al robot móvil como un vehículo diferencial.

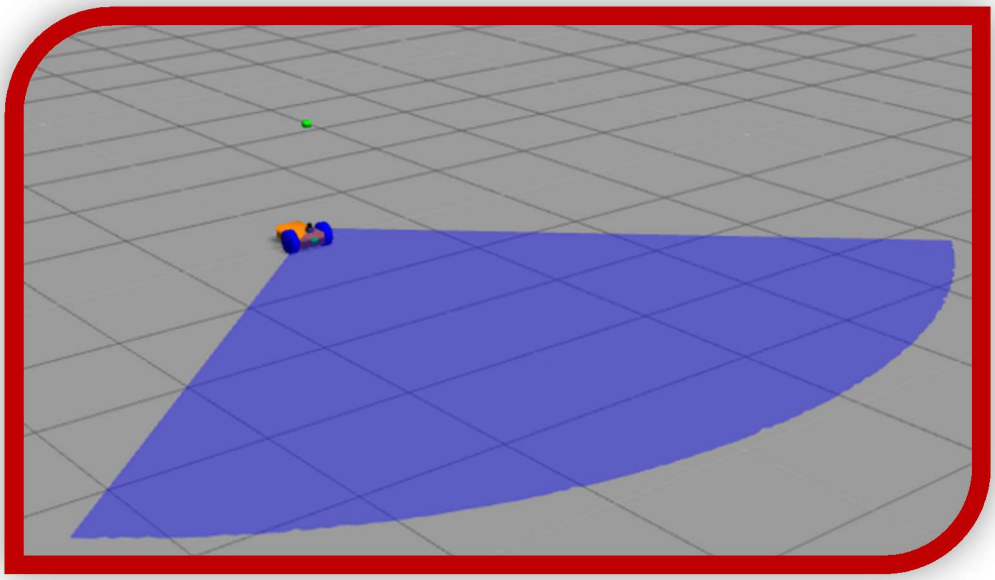
Sobre el chasis se acomodarán 3 sensores:

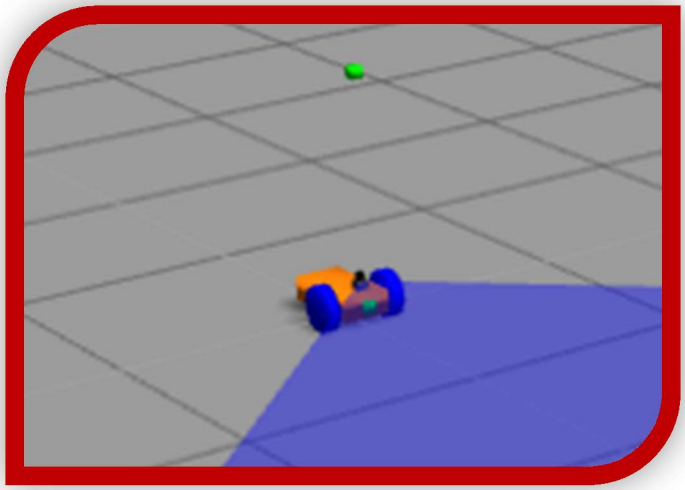
- Una cámara frontal, en el parachoques, que se encargará de detectar la señal de tráfico → Camera 2 ←
- Una cámara volante, situada un metro por encima del robot, cuya imagen se utilizará para detectar y seguir la línea de la carretera → Camera 1←. El joint de esta cámara no solo deberá estar situado donde se menciona, sino que el ángulo de visión debe ser 90º superior en el eje Y.

- Un emisor de láser que barrerá 90 grados del plano paralelo al suelo utilizando un láser por cada medio grado y detectando para cada uno la distancia a la que se encuentra el objeto más cercano, abarcando una distancia de entre 0.1 y 5 metros (definido en .gazebo)

```
<scan>
  <horizontal>
    <samples>180</samples>
    <resolution>1</resolution>
    <min_angle>-0.78</min_angle>
    <max_angle>0.78</max_angle>
  </horizontal>
</scan>
<range>
  <min>0.10</min>
  <max>5.0</max>
  <resolution>0.01</resolution>
</range>
```

Visualización del robot:



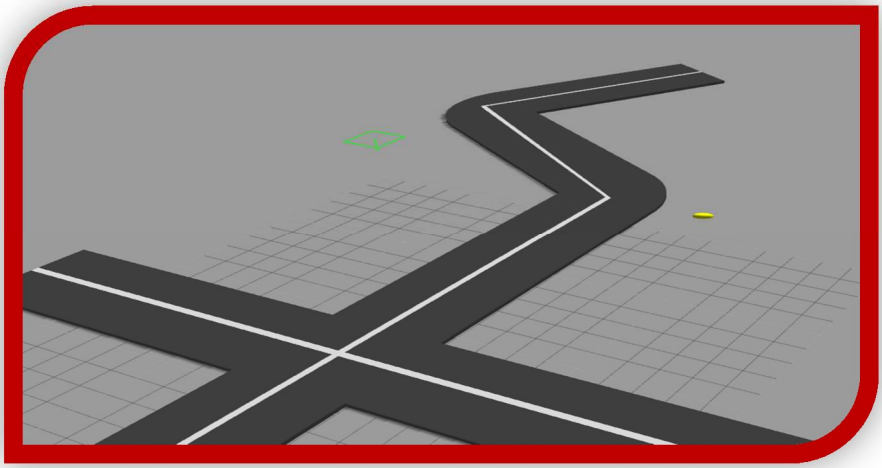


2.3 Diseño del mundo

Para conseguir una simulación válida hay que tener un escenario acorde con las pruebas que se quieren realizar. En este caso es necesaria una carretera que pueda seguir nuestro robot, y que incluya curvas. También será necesario equipar esta carretera con una señal de tráfico, más concretamente con la señal de Stop. El resto de obstáculos fijos se añadirían manualmente durante la simulación.

2.3.1 Carretera:

La creación de la carretera la llevamos a cabo con sketchup, un programa de Trimble para diseño 3d muy sencillo. La carretera deberá tener una línea blanca en el centro de la calzada que el robot pueda seguir fácilmente.



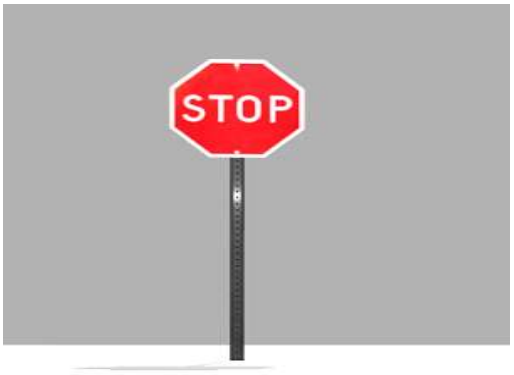
Como vemos en la carretera tiene un cruce. De cara a posibles mejoras en posteriores avances, la idea es que sea capaz de detectar cuando se encuentra en un cruce y decidir si que camino debe seguir.

Asimismo se han incluido curvas de diferentes ángulos para demostrar que el robot es capaz de seguirlas.

2.3.2 STOP:

Para la señal de Stop, la primera intención fue desarrollar las señales en programas de modelado en 3D, como Catia o sketchup, al igual que la carretera, y después exportar estos archivos a nuestra simulación. Tras varios intentos no se obtuvo el resultado deseado, así que se buscó un modelo de señal de stop en gazebo, que funcionó muy bien, aunque no puede posicionarse sobre el modelo de sketchup de la carretera, porque por razones desconocidas se bambolea, sino que debe posicionarse sobre el "suelo" del entorno de simulación.

1. Señal de Stop modelada en Catia



2. Señal de Stop (Gazebo)

2.4 Algoritmos de visión

Una vez las cámaras funcionan correctamente, es el momento de resolver los dos problemas visión planteados:

- Reconocimiento de señales de tráfico. En este caso debido al tiempo, nos centraremos en reconocer la señal de STOP.
- Reconocimiento de la línea de la carretera para su seguimiento.

2.4.1 Reconocimiento señal STOP

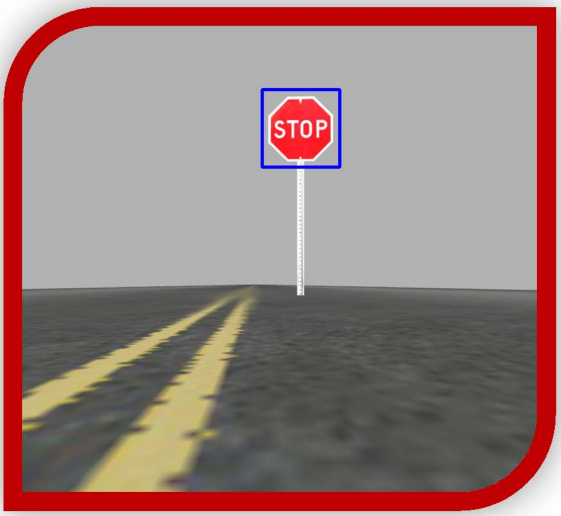
Cascada de Haar:

Para abordar este problema, la primera solución intentada fue implementar una cascada de Haar.

Los clasificadores haar, definen regiones rectangulares sobre una imagen en escala de grises (imagen integral) y al estar formada por un numero finito de rectángulos, se puede obtener un valor escalar que consiste en sumar los pixeles de cada rectángulo, en base a una serie de clasificadores en cascada. Cada clasificador determina si la subregión se trata del objeto buscado o no. A diferencia de otros algoritmos, este solo invierte capacidad de procesamiento a las subregiones que posiblemente representen el objeto buscado.

Se usó una cascada ya entrenada para identificar señales de STOP. Se consiguió el archivo .xml en un repositorio de github ("https://github.com/sherrardTr4129/Haar-Cascade-Classifiers/blob/master/frontal_stop_sign_cascade.xml") y se probó en nuestro entorno de simulación.

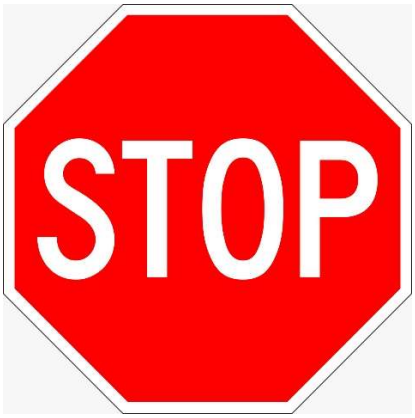
El clasificador funciona bastante bien, aunque no es todo lo fiable que requerimos, y se optó por buscar otra solución.

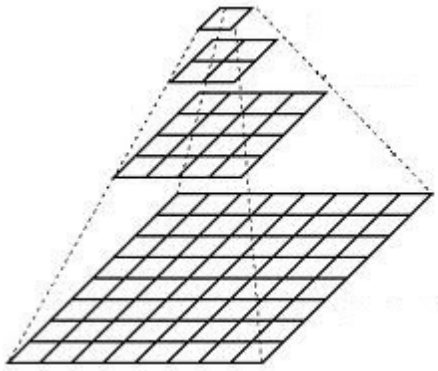


Búsqueda por prototipo

Se volvió a recurrir a un repositorio github para tomar otro algoritmo, en este caso basado en la búsqueda de un prototipo en la imagen ("https://github.com/mbasilyan/Stop-Sign-Detection")

Este código toma un "prototipo" de la señal de STOP (stopPrototype.png) y crea una pirámide de downsampling. Luego va calculando el error cuadrático medio en cada zona de cada nivel de la pirámide creada. La zona con menos error cuadrático medio será la señal de Stop.



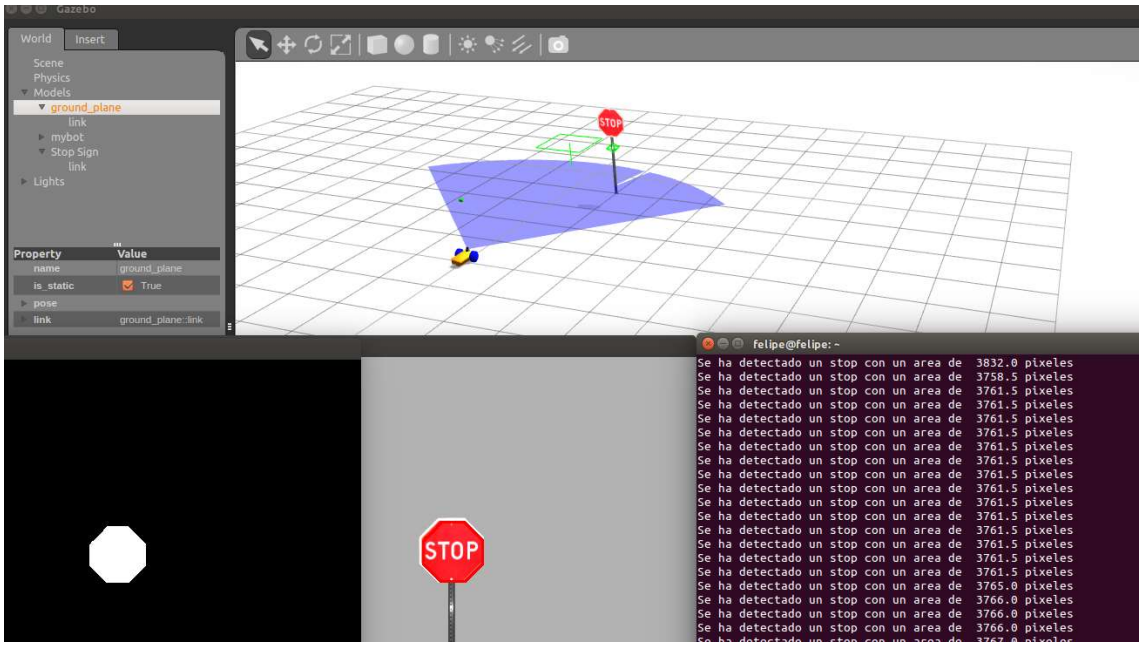


Los resultados obtenidos con este método son bastante buenos, el problema es que es muy ineficiente computacionalmente, necesitando en torno a unos 20s para cada iteración, así que tras las primeras pruebas, no se llegó ni a implementar en la simulación debido a que no es apto para un sistema en "tiempo real".

Forma y color

Tras dos intentos fallidos recurriendo a algoritmos externos, se decidió crear un programa basado en la búsqueda de un hexágono rojo.

Para ello se transformó la imagen a HSV, lo que facilita el reconocimiento de color y se creó una máscara seleccionando los puntos rojos de la imagen, mediante umbrales. Una vez se tuvo la máscara, se filtró el objetivo primero comprobando si tiene un área suficientemente grande. Si el área es suficientemente grande, mediante las funciones de la librería OpenCV findContours y approxPolyDP, se comprueba si es un hexágono, y si es así, el programa supondrá que un hexágono rojo es la señal de STOP.



El reconocimiento resultó preciso, fiable y rápido, así que será el método aplicado.

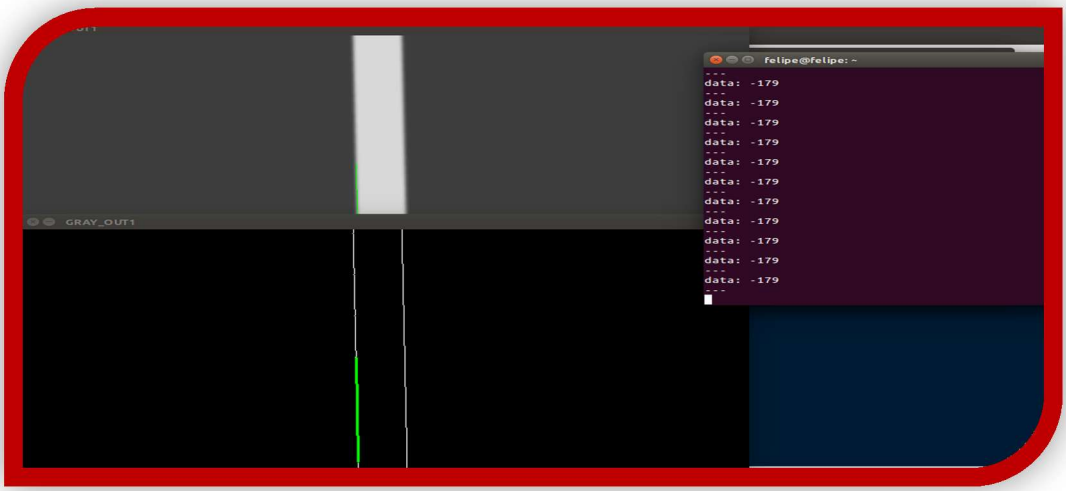
2.4.2 Seguimiento línea

Seguimiento ángulo

El primer método para realizar el seguimiento de la imagen es controlar el giro del robot directamente con el ángulo que tiene la línea.

Para ello, se aplica la transformada de Hough, obteniendo las líneas de nuestra imagen, se realiza la media de los ángulos de las líneas obtenidas y así se obtiene el ángulo a seguir.

Se aplicará una pequeña zona muerta alrededor de 0 para que no esté constantemente girando.



Una vez realizado, se observó que el método no es correcto ya que puede ir perfectamente en paralelo a la línea. Además, la transformada de Hough no da de forma continua todas las líneas de la imagen, haciendo que, sobre todo en los cambios de ángulo, el robot no tenga un comportamiento bueno.

Seguimiento de punto

Para poder aplicar el algoritmo de persecución pura, se intentó buscar un punto objetivo en la línea.

Para ello, trabajando de nuevo en HSV, se creó una máscara con los puntos blancos mediante umbrales. Con la librería OpenCV, se obtienen los momentos de la máscara. Mediante las siguientes relaciones se hallan las coordenadas del centro de masa:

$$c_x = \frac{m_{10}}{m_{00}}$$

$$c_y = \frac{m_{01}}{m_{00}}$$

Este será el punto el punto a seguir, mediante un control p con la diferencia en el eje x se controlará el ángulo de giro. Se realizará el seguimiento a velocidad fija.

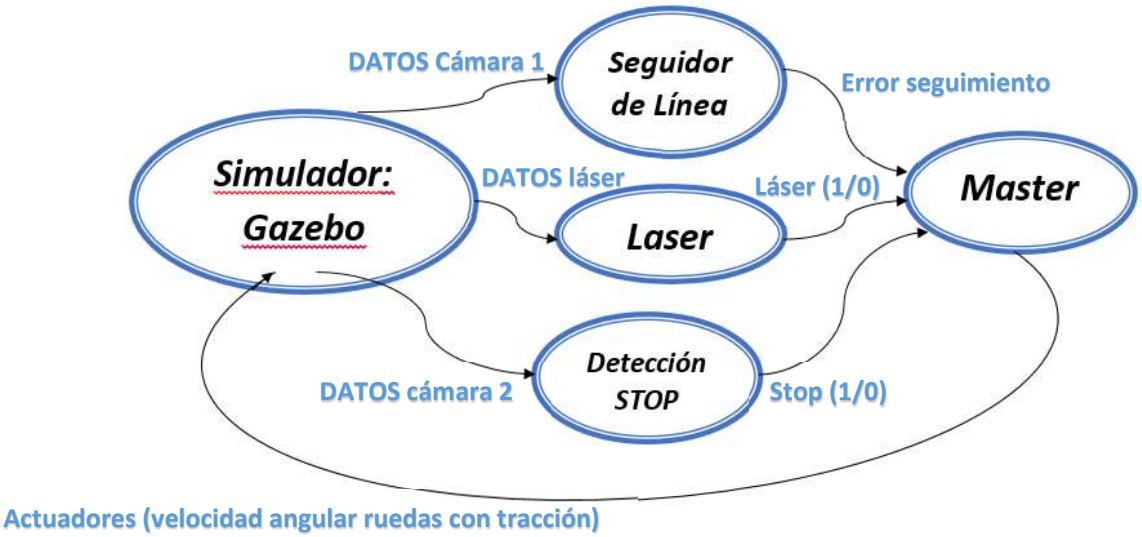
2.5 Estructura de la solución

La solución se estructurará de la siguiente manera:

- Adquisición de datos mediante sensores
- Procesamiento de dichos datos en algoritmos de seguimiento de línea, reconocimiento de señal de STOP y datos del láser
- Nodo maestro que decidirá los valores de los actuadores en función de la información recibida y que por tanto será responsable del comportamiento del vehículo.

El diagrama de bloques junto con la explicación de los mismos se mostrará a continuación.

2.5.1 Comunicación entre nodos



Descripción Nodos:

- ·**Simulador Gazebo:** Es el entorno de simulación donde se encuentra tanto nuestro robot con todos sus dispositivos (cámaras, laser, etc), como el mundo creado en apartados anteriores. Aquí se llevará a cabo la obtención de datos mediante los sensores, y se enviarán a los bloques intermedios.

- **Nodos intermedios:**

  - **Seguidor de Línea:** Es el nodo encargado de leer desde la primera cámara el camino que debemos seguir, reconocerlo, y calcular el error que se comete en la dirección actual respecto a la de la línea. Tal como se explicó en su sub-apartado, esto se hará emplazando un punto en el centro de masas de la línea y restándole sus coordenadas al ancho total en píxeles de la imagen. Dicho valor será nuestro error de seguimiento, y será lo que se transmita al nodo maestro, re-escalado, ya que a la salida original del programa, el error se transmitía sobredimensionado (nº de píxeles).

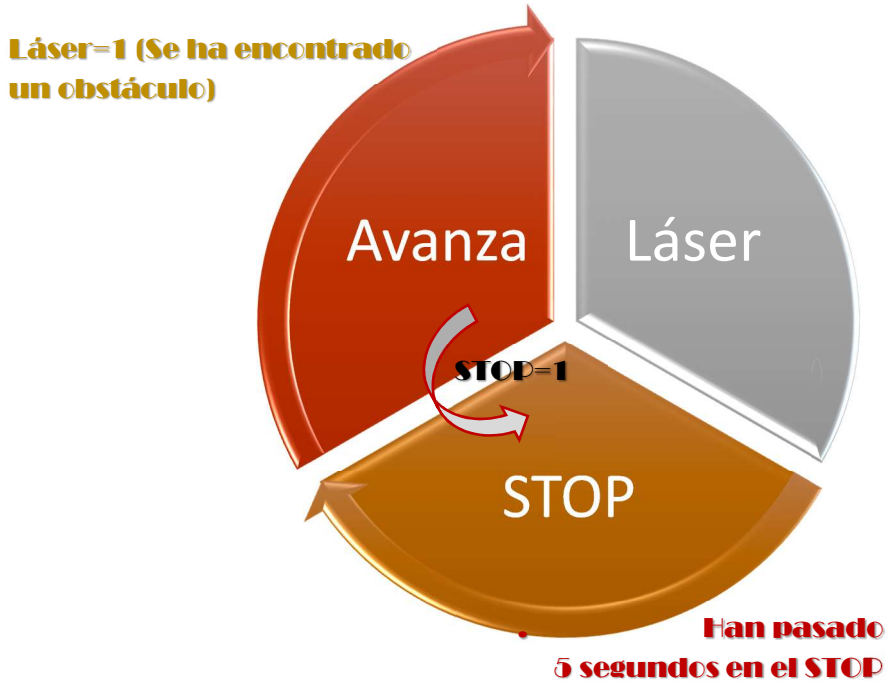
  - **Láser:** En este nodo se mide la distancia a un objeto fijo a través de un láser equipado en el modelo de Gazebo del robot. Se recuerda que el ángulo de barrido del láser es de 90 grados, cubriendo toda el plano frontal del robot, que detecta obstáculos entre 0.1 y 5 metros, y que emite láseres cada medio grado. Desde este nodo se enviará un 1 lógico (variable con formato int8) cuando se detecte un obstáculo a menos de 1 metro de distancia.

  - **Detección STOP:** Gracias a la información proporcionada por la segunda cámara en Gazebo, se reconoce el entorno en busca de alguna señal de STOP, en caso de encontrarse, se enviará un 1 lógico (variable con formato int8) al nodo maestro.

- **Máster:** Es el encargado de recibir la información tanto de Seguidor de Línea, Láser y Detección STOP, y con esta información ser capaz de calcular una buena respuesta para nuestro robot, y mandarle esta información, a través de topic geometry_msgs/twist, a nuestra simulación de Gazebo.

2.5.2 Nodo maestro

El nodo maestro tiene la estructura de una máquina de estados. Ésta cuenta con tres estados fundamentales que se intercomunican como representamos en el siguiente esquema:



De modo que normalmente el programa se ubicará en el estado Avanza, donde el robot se moverá normalmente, mediante el control del coche diferencial, que recibirá la velocidad lineal, determinada por defecto, y cuánto deberá girar para corregir su dirección, error calculado previamente en el nodo de seguimiento de línea.

Hasta que se den las circunstancias para llevar a cabo alguna de las dos transiciones:

- En el momento que se detecte la interrupción Laser, se parará, y se seguirá en esta situación para siempre. Un obstáculo es la manera decidida para indicar que se ha terminado el recorrido. Añadir una nueva transición para que se vuelva al estado avanza si desaparece el obstáculo es muy sencillo, pero para esta simulación se ha preferido dejarlo tal y como se ha explicado.
- En el momento que en el nodo de reconocimiento de señal de tráfico se visualice una, se pasará al estado STOP, en el que se permanecerá durante 5 segundos, tras los cuales se volverá automáticamente al estado de avance.

3. Conclusión

Tras superar lo mejor posible los múltiples problemas y contratiempos con los que fue necesario enfrentarse, se ha conseguido un robot móvil con una serie de sensores que es capaz de seguir una carretera (o línea, para simplificarlo), detectar un tipo de señal de tráfico y parar en caso de encontrar obstáculos.

Sin embargo, todo esto no ha sido lo que más difícil ha resultado al grupo de trabajo. La mayor parte de los problemas y contratiempos ocurridos han sido ocasionados por el absoluto desconocimiento de ROS por parte de 4 de los 5 miembros del grupo, así como por la necesidad de instalación de sistemas operativos LINUX, ROS y Gazebo, junto con todos los paquetes necesarios para su funcionamiento. Que se debieron llevar a cabo a causa de la imposibilidad de cooperar con ESIBot para realizar trabajo sobre el coche de SEAT.

No se han alcanzado todos los objetivos del reto original, pero se han adquirido muchos conocimientos de ROS que serán útiles en el futuro, se han ideado soluciones para los distintos problemas aplicando conocimiento de otras asignaturas de la carrera, como por ejemplo el tratamiento de la imagen para seguir la línea, a partir de la asignatura de percepción, del primer cuatrimestre de cuarto… Y por supuesto se ha continuado aprendiendo a coordinarse y trabajar en grupo para cumplir una serie de requisitos en un tiempo acotado.

3.1 Posibles mejoras

Hay muchas posibles mejoras para el programa, tales como:

- Detección de más señales de tráfico y reacción ante ellas, por ejemplo implementando el programa desarrollado por nuestros compañeros Alejandro Trujillo, Javier Domínguez y Fernando Cristo (red neuronal)
- Aparcamiento automático
- Adición de focos luminosos para evitar tener que iluminar las señales de tráfico
- Toma de decisiones al encontrar cruces (algoritmo búsqueda de rutas)
- Geolocalización y filtros de datos para los sensores (Por ejemplo SLAM)

3.2 Bibliografía

- Para comprender ROS: http://wiki.ros.org/ROS/Tutorials
- Para el filtro haar: http://www.trevorsherrard.com/Haar_training.html
- Clasificador xml: https://github.com/sherrardTr4129/Haar-Cascade-Classifiers
- Enviar un dato en ROS: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

- Usar la usb cam para debug: https://rosstitchernode.wordpress.com/2014/06/05/ros-and-opencv-with-usb_cam/
- Seguidor de linea el de la transformada de Hough: https://github.com/robotictang/eddiebot_line_follower/tree/master/eddiebot_line_follower
- Gazebo: http://gazebosim.org/tutorials
- Creación del robot: http://moorerobots.com/blog/post/1
- Programming robots with ROS (calcular el punto objetivo) , chapter 12: follow-bot: https://books.google.es/books?id=Hnz5CgAAQBAJ&printsec=frontcover&hl=es#v=onepage&q&f=false