



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2022-2023

Trabajo Fin de Grado

Extensión de la herramienta VisualCircuit a ROS2
para programar aplicaciones robóticas.[2cm]

Autor: David Tapiador de Vera

Tutor: Jose María Cañas Plaza

Agradecimientos

Después de tanto sufrir, por fin llega el momento de terminar. Ha sido un camino duro, incluyendo noches sin dormir y mucho estrés acumulado, pero por fin se acaba.

Tengo que agradecer a mis padres y hermana por ayudarme día a día a mejorar y superar los momentos malos.

A mis amigos por obligarme a salir incluso cuando menos ánimos tenía y ayudarme a desconectar de todo.

Y sobretodo tengo que agradecer a mi apoyo fundamental. Al pilar de mi vida y al que me hace seguir adelante día tras día, mi perrete Koby. Él sí que me obliga a salir aunque esté lloviendo, nevando, con las calles congeladas, con una pandemia... Simplemente gracias.

*Vida antes que muerte,
fuerza antes que debilidad,
viaje antes que destino.*

Brandon Sanderson

Resumen

Escribe aquí el resumen del trabajo. Un primer párrafo para dar contexto sobre la temática que rodea al trabajo.

Un segundo párrafo concretando el contexto del problema abordado.

En el tercer párrafo, comenta cómo has resuelto la problemática descrita en el anterior párrafo.

Por último, en este cuarto párrafo, describe cómo han ido los experimentos.

Acrónimos

ROS *Robot Operating System*

IMU *Inertial Measurement Unit*

LIDAR *Laser Imaging Detection and Ranging*

ODE *Open Dynamics Engine*

USB *Universal Serial Bus*

VFF *Vector Force Field*

URDF *Unified Robotics Description Format*

RVIZ *ROS Visualization*

RGBD *Red Green Blue - Depth*

FPS *Frames Per Second*

PID *Proportional-Integral-Derivative controller*

Índice general

1. Introducción	1
1.1. La primera sección	1
1.2. Segunda sección	1
1.2.1. Números	2
1.2.2. Listas	2
2. Objetivos y metodología de Trabajo	4
2.1. Descripción del problema	4
2.2. Requisitos	4
2.3. Metodología	4
2.4. Plan de trabajo	4
3. Herramientas y plataforma de desarrollo	5
3.1. Lenguaje de programación	5
3.2. ROS2 (Robot Operating System 2)	5
3.3. Gazebo	6
3.4. Turtlebot2	7
3.4.1. Base Kobuki	7
3.4.2. Cuerpo Turtlebot2	8
3.4.3. Turtlebot2 simulado	9
3.5. RVIZ2	10
3.6. Sensores	11
3.6.1. Cámara ASUS Xtion Pro	11
3.6.2. RPLIDAR A2	11
3.7. VisualCircuit	12
4. Desarrollo de bloques driver	14
4.1. Introducción a VisualCircuit	14
4.2. Snippets	15

4.3. Verbatim	15
4.4. Ecuaciones	16
4.5. Tablas o cuadros	16
5. Sigue personas	18
5.1. Conclusiones	18
5.2. Corrector ortográfico	19
6. Máquina de estados (VFF)	20
6.1. Conclusiones	20
6.2. Corrector ortográfico	21
7. Conclusiones	22
7.1. Conclusiones	22
7.2. Corrector ortográfico	23
Bibliografía	24

Índice de figuras

1.1. Robot aspirador Roomba de iRobot.	2
3.1. Comunicación del nodo Master con los nodos Intermedios y con distintos sensores y actuadores.	6
3.2. Simulador Gazebo.	7
3.3. Kobuki base	8
3.4. Turtlebot2	8
3.5. Turtlebot2 simulado	9
3.6. RVIZ2 Vs mundo gazebo	10
3.7. Cámara ASUS-XTION	11
3.8. RPLIDAR A2	11
3.9. VisualCircuit	12
3.10. Ejemplo VisualCircuit	13
4.1. Creando un bloque en VisualCircuit	15

Listado de códigos

3.1. Hola mundo en python	5
4.1. Función para buscar elementos 3D en la imagen	15
4.2. Cómo usar un Slider	16

Listado de ecuaciones

4.1. Ejemplo de ecuación con fracciones	16
4.2. Ejemplo de ecuación con array y letras y símbolos especiales	16

Índice de cuadros

4.1. Parámetros intrínsecos de la cámara	17
--	----

Capítulo 1

Introducción

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este primer capítulo, el de introducción, se trata de dar un contexto amplio y atractivo del trabajo. Comienza hablando de un contexto general y acaba hablando del contexto más específico en el que se enmarca el proyecto. Es el capítulo idóneo para incluir todas las referencias bibliográficas que hayan tratado este tema; suponen un fuerte respaldo al trabajo.

1.1. La primera sección

En los textos puedes poner palabras en *cursiva*, para aquellas expresiones en sentido *figurado*, palabras como *robota*, que está fuera del diccionario castellano, o bien para resaltar palabras de una colección: (a) es la primera letra del abecedario, (b) es la segunda, etc.

Al poner las dos líneas del anterior párrafo, este aparecerá separado del anterior. Si no las pongo, los párrafos aparecerán pegados. Sigue el criterio que consideres más oportuno.

1.2. Segunda sección

No olvides incluir imágenes y referenciarlas, como la Figura UwU 1.1.

Ni tampoco olvides de poner las URLs como notas al pie. Por ejemplo, si hablo de la Robocup¹.

¹<http://www.robocup.org>



Figura 1.1: Robot aspirador Roomba de iRobot.

1.2.1. Números

En lugar de tener secciones interminables, como la Sección 1.1, divídelas en subsecciones.

Para hablar de números, mételos en el entorno *math* de L^AT_EX, por ejemplo, $1,5Kg$. También puedes usar el símbolo del Euro como aquí: 1.500€ .

1.2.2. Listas

Cuando describas una colección, usa `itemize` para ítems o `enumerate` para enumerados. Por ejemplo:

- *Entorno de simulación.* Hemos usado dos entornos de simulación: uno en 3D y otro en 2D.
 - *Entornos reales.* Dentro del campus, hemos realizado experimentos en Biblioteca y en el edificio de Gestión.
1. Primer elemento de la colección.
 2. Segundo elemento de la colección.

Referencias bibliográficas Cita, sobre todo en este capítulo, referencias bibliográficas que respalden tu argumento. Para citarlas basta con poner la instrucción `\cite` con el identificador de la cita. Por ejemplo: libros como [12], artículos como [11], URLs como [10], tesis como [8], congresos como [9], u otros trabajos fin de grado como [6].

Las referencias, con todo su contenido, están recogidas en el fichero `bibliografia.bib`. El contenido de estas referencias está en formato BibTeX. Este formato se puede obtener en muchas ocasiones directamente, desde plataformas como [Google Scholar](#) u otros repositorios de recursos científicos.

Existen numerosos estilos para reflejar una referencia bibliográfica. El estilo establecido por defecto en este documento es APA, que es uno de los estilos más comunes, pero lo puedes modificar en el archivo `memoria.tex`; concretamente, cambiando el campo `apalike` a otro en la instrucción `\bibliographystyle{apalike}`.

Y, para terminar este capítulo, resume brevemente qué vas a contar en los siguientes.

Capítulo 2

Objetivos y metodología de Trabajo

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo lo ideal es explicar cuáles han sido los objetivos que te has fijado conseguir con tu trabajo, qué requisitos ha de respetar el resultado final, y cómo lo has llevado a cabo; esto es, cuál ha sido tu plan de trabajo.

2.1. Descripción del problema

Cuenta aquí el objetivo u objetivos generales y, a continuación, concrétalos mediante objetivos específicos.

2.2. Requisitos

Describe los requisitos que ha de cumplir tu trabajo.

2.3. Metodología

Qué paradigma de desarrollo software has seguido para alcanzar tus objetivos.

2.4. Plan de trabajo

Qué agenda has seguido. Si has ido manteniendo reuniones semanales, cumplimentando objetivos parciales, si has ido afinando poco a poco un producto final completo, etc.

Capítulo 3

Herramientas y plataforma de desarrollo

El desarrollo de nuevo contenido para la plataforma de VisualCircuit, ha necesitado usar distintas herramientas, como por ejemplo programación, ROS2, gazebo..., por lo que voy a hacer una pequeña descripción de cada una, así como el uso que se le ha dado dentro del proyecto.

3.1. Lenguaje de programación

Python es un lenguaje interpretado de alto nivel. Este lenguaje busca facilitar la legibilidad del código, convirtiéndolo en uno de los más comunes a día de hoy. Es un lenguaje de programación multiparadigma, ya que soporta tanto programación orientada a objetos, como programación imperativa y funcional.

```
print("Hello World")
```

Código 3.1: Hola mundo en python

Dentro del TFG se usará para la programación dentro de la plataforma VisualCircuit (3.7).

3.2. ROS2 (Robot Operating System 2)

ROS¹ o Robot Operating System es un *middleware*² formado por un conjunto de herramientas y librerías de software libre empleadas para el desarrollo de aplicaciones robóticas. Su objetivo es ofrecer una plataforma estándar para todas las ramas de la robótica.

¹ROS: <http://wiki.ros.org/es>

²Middleware: software que se sitúa entre las aplicaciones y el sistema operativo

ROS se basa en una arquitectura *cliente-servidor* centralizado que, mediante suscriptores y publicadores, permite enviar información, ya sean medidas de sensores, cambios de estado, decisiones usando árboles de decisión, órdenes a los actuadores, etc.

Para comunicarse con los servidores (o como se llaman en ROS, *topics*) se usan nodos. Estos nodos pueden contar con varios publicadores y suscriptores simultáneos. Cada *topic* se define con un tipo de mensaje, que será el único que se pueda enviar y recibir a través de él. Estos tipos de mensajes pueden ser mensajes simples como una cadena de caracteres o tipos compuestos con otros tipos, permitiéndonos crear topics adecuados a las necesidades de cada proyecto.

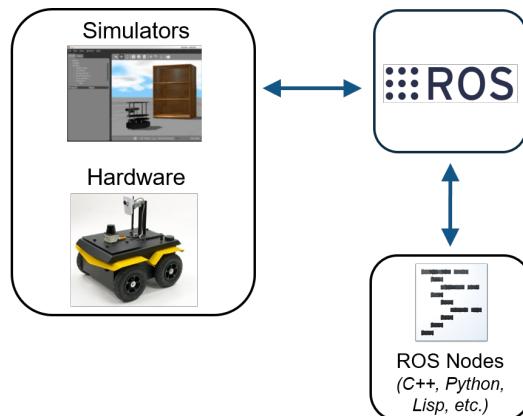


Figura 3.1: Comunicación del nodo Master con los nodos Intermedios y con distintos sensores y actuadores. Imagen obtenida de [2]

ROS2 lo usaremos para obtener información del robot turtlebot2 (3.4), tanto real como simulado, como por ejemplo su posición en el entorno simulado o las últimas medidas de sus sensores, y para comandarle instrucciones (velocidades a sus motores)

3.3. Gazebo

Gazebo³ es un simulador 3D de código abierto orientado a la robótica que permite fusionar escenarios realistas con robots simulados, ofreciendo un entorno seguro para probar algoritmos. Éste utiliza el motor de físicas ODE⁴, aunque se puede configurar con otros motores, como Bullet⁵ o DART⁶.

³Gazebo: <https://classic.gazebosim.org/>

⁴Open Dynamics Engine: <https://www.ode.org/>

⁵Bullet: <https://pybullet.org/wordpress/>

⁶DART: <https://dartsim.github.io/>

Al estar orientado a la robótica, permite integrar fácilmente modelos de robots reales con sensores (incluso simulando sus ruidos) y enviar a través de los distintos topics de ROS o ROS2 (3.2) alguna información directa del simulador, como la posición, medidas de los sensores simulados o incluso información de objetos no programables (del entorno).

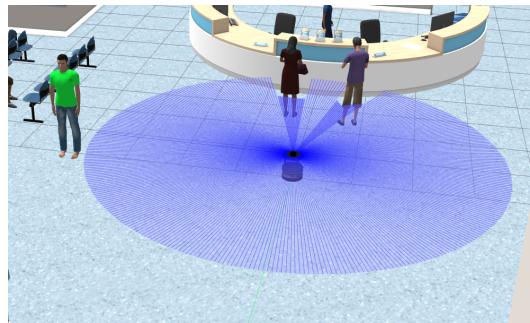


Figura 3.2: Ejemplo de ejecución en gazebo.

A lo largo de todo el proyecto, usaremos gazebo para simular el turtlebot2 (3.4) al igual que los distintos entornos que iremos usando para probar los programas.

3.4. Turtlebot2

La URJC de Fuenlabrada, en sus laboratorios de robótica, cuenta con varios robots Turtlebot2⁷ a disposición de los alumnos del grado. Estos son perfectos para la enseñanza e investigación en robótica, por su sencilla introducción a temas como ROS o el uso de sensores. Los Turtlebot2 están formados por dos partes principales: una base Kobuki y una estructura superior.

3.4.1. Base Kobuki

La base del Turtlebot2 se llama *Kobuki*. En apariencia, es similar a un robot de limpieza como podrían ser los Roomba. En cuanto al hardware, lleva integrados tres bumpers (sensores de contacto), odometría, sensor de caída y varios giroscopios. Tiene una velocidad lineal máxima de 0.7 m/s y angular de 180 grados/s. Su batería le permite una autonomía de entre 3 y 7 horas. Cuenta con varios puertos, entre ellos un USB para poder conectar nuestro portatil y ejecutar los distintos algoritmos.

⁷Turtlebot2: <https://www.turtlebot.com/turtlebot2/>

Algunos de los paquetes de ROS2 que instalaremos para poder usarlo son los drivers del kobuki para ROS2-Humble⁸ de IntelligentRoboticsLabs, compañeros de la URJC. Siguiendo las instrucciones de instalación que se encuentran en dicho repositorio de github, accedemos a varios paquetes básicos para el uso de kobuki, como *kobuki_ros* o *kobuki_node*, entre otros.



Figura 3.3: Base kobuki. Imagen obtenida de [5]

3.4.2. Cuerpo Turtlebot2

El cuerpo del turtlebot2 (también conocido como *TurtleBot Structure*) está formado por una serie de plataformas y tubos que se atornillan a la base kobuki y permiten fijar nuevos sensores, como podrían ser una cámara o un láser, o actuadores como brazos robóticos. También ofrece un sitio cómodo para poder colocar el portátil encima del robot y así poder conectarlo a la base mediante USB.



Figura 3.4: Turtlebot2. Imagen obtenida de [4]

⁸Drivers kobuki ROS2-Humble: <https://github.com/IntelligentRoboticsLabs/Robots/tree/humble/kobuki>

3.4.3. Turtlebot2 simulado

Para algunas partes del proyecto, como el desarrollo de drivers para ROS2 (4) o el VFF usando máquinas de estados (6), hemos usado el simulador para probar y desarrollar los algoritmos. Para esto, he tenido que usar un modelo del turtlebot2 que cuenta con los mismos sensores (cámara, RPLIDAR, bumper, etc) que el real, así como los mismos topics.

Para integrar el modelo del robot en el simulador, necesitamos su representación en URDF⁹, una forma estandarizada de crear los modelos de los robots incluyendo sus sensores y actuadores, partes móviles etc.

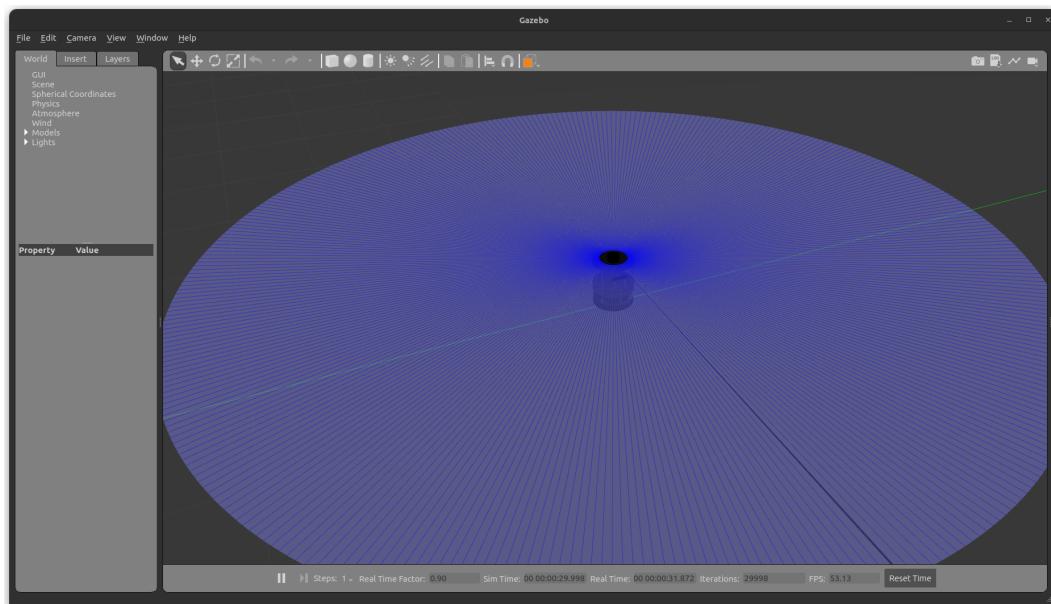


Figura 3.5: Turtlebot2 en gazebo.

⁹URDF: Unified Robot Description Format

3.5. RVIZ2

RVIZ2 es una herramienta de visualización 3D para robots, el ambiente y las medidas de los sensores de éstos.

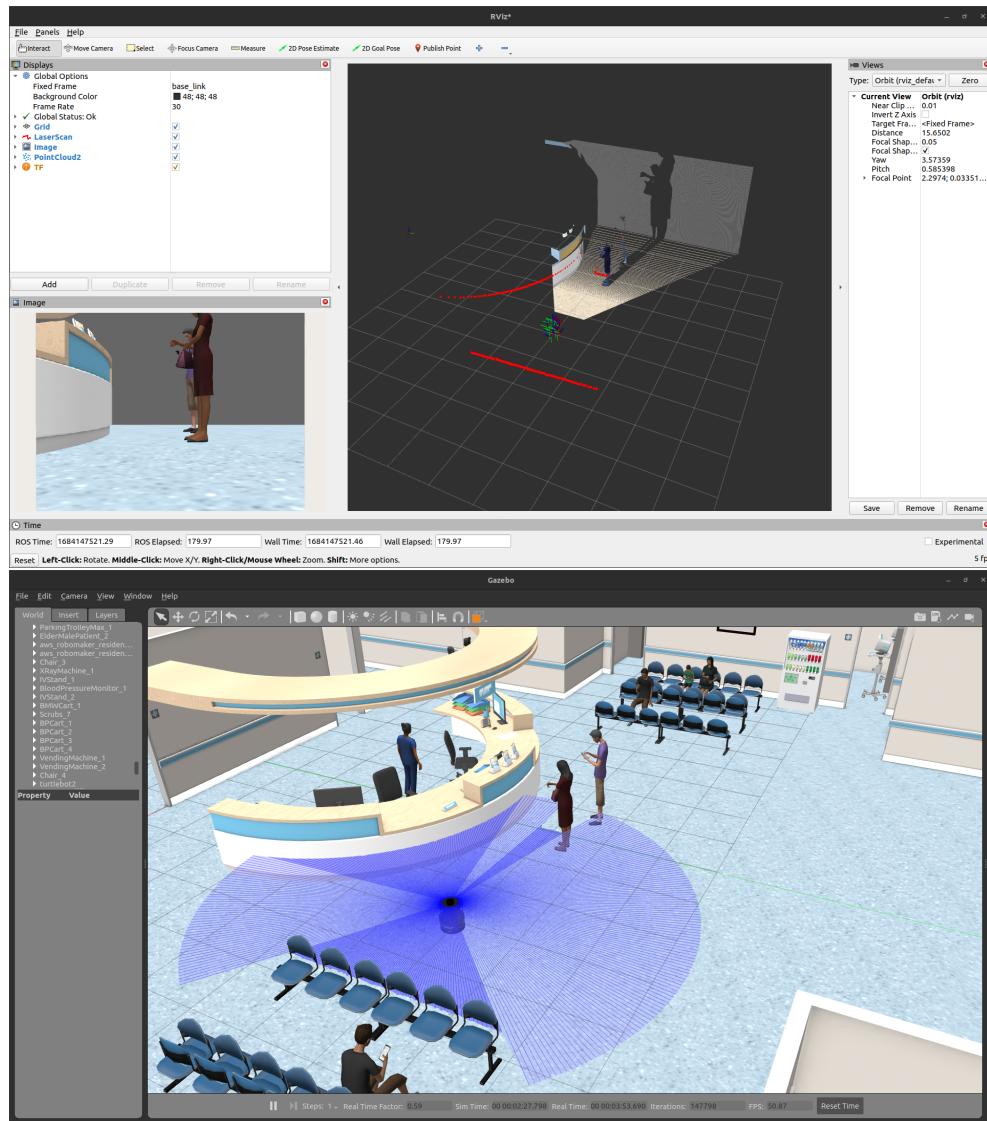


Figura 3.6: Ejemplo de RVIZ2 frente al mundo gazebo real.

RVIZ2 se usará bastante durante el proyecto tanto para depurar como para observar las medidas de los distintos sensores a tiempo real.

3.6. Sensores

Como hemos mencionado anteriormente, un robot se compone, a grandes rasgos, de sensores y actuadores. Para este proyecto se han usado varios de ellos, por lo que aquí hay una pequeña introducción a cada uno.

3.6.1. Cámara ASUS Xtion Pro

La cámara *ASUS Xtion* es una cámara RGB-D¹⁰, que ofrece tanto imagen como una nube de puntos con la distancia medida para cada pixel de la imagen. Esta cámara ofrece una imagen de 720p, con una frecuencia de 60fps. En la parte de profundidad, es capaz de captar desde 0.8m hasta 3.5 con un ángulo efectivo de 70°. Se conecta mediante USB directamente al ordenador.

En el proyecto, como debemos usarla con ROS2, usaremos el paquete creado por un usuario de internet¹¹.



Figura 3.7: Cámara ASUS-XTION. Imagen obtenida de [1]

3.6.2. RPLIDAR A2

Se trata de un láser de 360° con un rango de medida desde 0.2m hasta 16m y una frecuencia de muestreo que se puede ajustar desde 5Hz hasta 15Hz. Usando los drivers mencionados en el apartado del Turtlebot2 (3.4.1) encontraremos un paquete para poder activar y usar este sensor con ROS2.



Figura 3.8: Sensor RPLIDAR A2. Imagen obtenida de [3]

¹⁰**RGB-D**: RedGreenBlue-Depth, hace referencia las cámaras que captan la imagen y las distancias de cada pixel.

¹¹**Drivers ASUS-Xtion ROS2**: https://github.com/mgonzs13/ros2_asus_xtion

3.7. VisualCircuit

VisualCircuit¹² es un editor visual online basado en programación por bloques de código orientado al desarrollo de aplicaciones robóticas. Está desarrollado sobre IceStudio¹³.

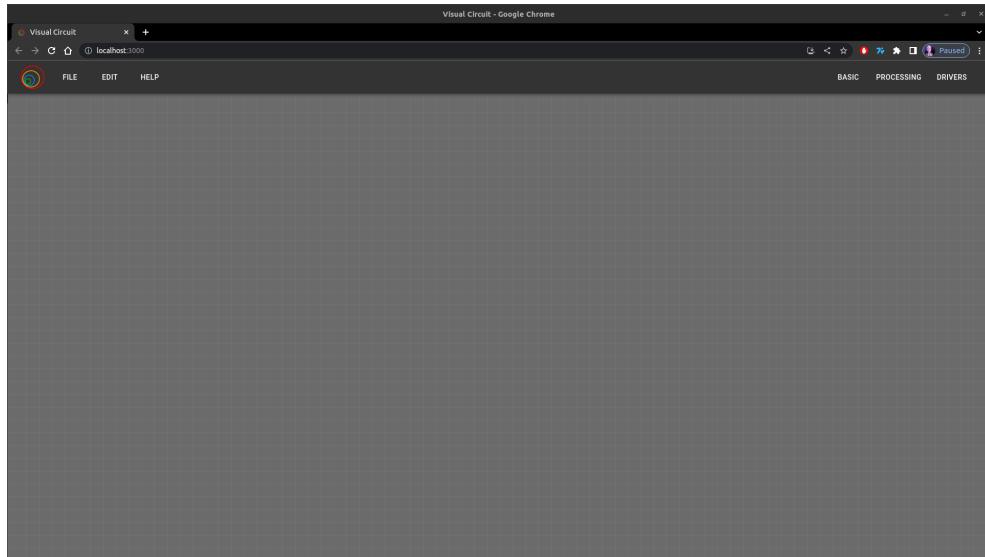


Figura 3.9: Página de VisualCircuit.

Los bloques que se pueden usar están divididos en varias pestañas: *basics*, *processing* y *drivers*.

- *Basic*: bloques simples, como inputs y outputs, bloques para definir parámetros y constantes, o bloques para insertar nuestro código.
- *Processing*:
 - *Control*: bloques de control (PID).
 - *OpenCV*: bloques relacionados con OpenCV¹⁴ y edición de imagen (filtros de color, detección de contornos, erosión, etc).
 - *TensorFlow*: un bloque para detección de objetos.

¹²VisualCircuit Docs: <https://jderobot.github.io/VisualCircuit/>

¹³IceStudio Project: <https://github.com/FPGAwars/icestudio>

¹⁴OpenCV: <https://opencv.org/>

- *Drivers*: drivers que conectan con los sensores y actuadores

- *Control*: motordriver (ROS) y teleoperador.
- *OpenCV*: lector de imágenes desde archivos y desde cámaras, pantalla para mostrar las imágenes.
- *ROS-Sensors*: cámara, odometría e IMU¹⁵ usando ROS.

Ésta será la herramienta principal en torno a la que girará este proyecto, tanto buscando añadir funcionalidades como en desarrollar aplicaciones con ella.

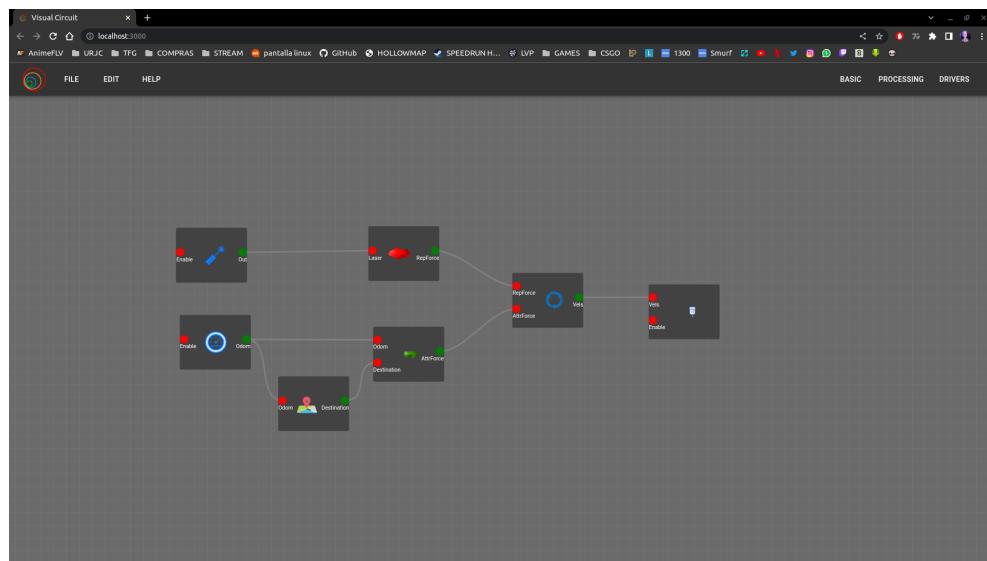


Figura 3.10: Ejemplo de proyecto de VisualCircuit.

¹⁵IMU: Inertial Measurement Unit

Capítulo 4

Desarrollo de bloques driver

Como ya he explicado, VisualCircuit es una plataforma de programación online mediante el uso de bloques, pero para que esté actualizado, hay que ir añadiendo bloques nuevos que ofrezcan esas nuevas funciones y añadirlos a las listas de bloques estándar que ofrece la página.

En este capítulo profundizaremos en el funcionamiento de la plataforma VisualCircuit así como en el proceso seguido para desarrollar nuevos bloques para poder añadir ROS2 a la misma.

4.1. Introducción a VisualCircuit

En VisualCircuit, antes de realizar este proyecto, ya existían bloques dedicados específicamente a la robótica para algunos sensores (cámara, odometría e IMU¹) y para los motores usando ROS melodic², pero al tratarse de una versión antigua, decidimos que ya era momento de actualizar a ROS2 humble³, ya que era la versión estable más moderna hasta el momento.

Para crear nuestro propio bloque, debemos añadir varios bloques prefabricados. Para introducir nuestro código principal usaremos el bloque *code*. Al crearlo, se nos permite definir el número de entradas, salidas y parámetros que tendrá nuestro código.

¹IMU: Inertial Measurement Unit

²ROS melodic: <http://wiki.ros.org/melodic>

³ROS humble: <https://docs.ros.org/en/humble/index.html>

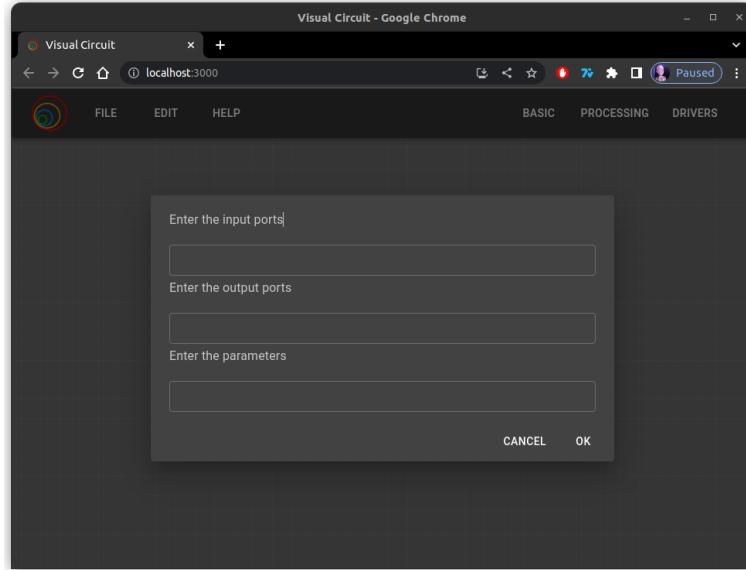


Figura 4.1: Creando un bloque en VisualCircuit.

4.2. Snippets

Puede resultar interesante, para clarificar la descripción, mostrar fragmentos de código (o *snippets*) ilustrativos. En el Código 4.1 vemos un ejemplo escrito en C++.

```
void Memory::hypothesizeParallelograms () {
    for(it1 = this->controller->segmentMemory.begin(); it1++) {
        squareFound = false; it2 = it1; it2++;
        while ((it2 != this->controller->segmentMemory.end()) && (!squareFound))
        {
            if (geometry::haveACommonVertex((*it1), (*it2), &square)) {
                dist1 = geometry::distanceBetweenPoints3D ((*it1).start, (*it1).end);
                dist2 = geometry::distanceBetweenPoints3D ((*it2).start, (*it2).end);
            }
        // [...]
    }
}
```

Código 4.1: Función para buscar elementos 3D en la imagen

En el Código 4.2 vemos un ejemplo escrito en Python.

4.3. Verbatim

Para mencionar identificadores usados en el código —como nombres de funciones o variables— en el texto, usa el entorno literal o verbatim `hypothesizeParallelograms()`. También se puede usar este entorno para varias líneas, como se ve a continuación:

```

def mostrarValores():
    print (w1.get(), w2.get())

master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.pack()
Button(master, text='Show', command=mostrarValores).pack()

mainloop()

```

Código 4.2: Cómo usar un Slider

```

void Memory::hypothesizeParallelograms () {
    // add your code here
}

```

4.4. Ecuaciones

Si necesitas insertar alguna ecuación, puedes hacerlo. Al igual que las figuras, no te olvides de referenciarlas. A continuación se exponen algunas ecuaciones de ejemplo: Ecuación 4.1 y Ecuación 4.2.

$$H = 1 - \frac{\sum_{i=0}^N \frac{(\frac{d_{js} + d_{je}}{2})}{N}}{M} \quad (4.1)$$

Ecuación 4.1: Ejemplo de ecuación con fracciones

$$v(\text{entrada}) = \begin{cases} 0 & \text{if } \epsilon_t < 0,1 \\ K_p \cdot (T_t - T) & \text{if } 0,1 \leq \epsilon_t < M_t \\ K_p \cdot M_t & \text{if } M_t < \epsilon_t \end{cases} \quad (4.2)$$

Ecuación 4.2: Ejemplo de ecuación con array y letras y símbolos especiales

4.5. Tablas o cuadros

Si necesitas insertar una tabla, hazlo dignamente usando las propias tablas de L^AT_EX, no usando pantallazos e insertándolas como figuras... En el Cuadro 4.1 vemos un ejemplo.

Parámetros	Valores
Tipo de sensor	Sony IMX219PQ[7] CMOS 8-Mpx
Tamaño del sensor	3.674 x 2.760 mm (1/4"format)
Número de pixels	3280 x 2464 (active pixels)
Tamaño de pixel	1.12 x 1.12 um
Lente	f=3.04 mm, f/2.0
Ángulo de visión	62.2 x 48.8 degrees
Lente SLR equivalente	29 mm

Cuadro 4.1: Parámetros intrínsecos de la cámara

Capítulo 5

Sigue personas

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

5.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

5.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Capítulo 6

Máquina de estados (VFF)

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

6.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

6.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Capítulo 7

Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

7.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

7.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Bibliografía

- [1] I. Amazon.com. Imagen de la cámara asus-xtion., 2011.
- [2] S. Castro. Comunicación entre nodos intermedios, hardware/software y nodo master, 2017.
- [3] R. Components. Imagen del sensor laser rplidar a2., 2011.
- [4] I. Open Source Robotics Foundation. Imagen de un turtlebot2.
- [5] Robosavvy. Base kobuki para turtlebot2, 2022.
- [6] J. Vega. Navegación y autolocalización de un robot guía de visitantes. Master thesis on computer science, Rey Juan Carlos University, September 2008.
- [7] J. Vega. De la tiza al robot. Technical report, June 2015.
- [8] J. Vega. *Educational framework using robots with vision for constructivist teaching Robotics to pre-university students*. Doctoral thesis on computer science and artificial intelligence, University of Alicante, September 2018.
- [9] J. Vega. JdeRobot-Kids framework for teaching robotics and vision algorithms. In *II jornada de investigación doctoral*. University of Alicante, June 2018.
- [10] J. Vega. El profesor Julio Vega, finalista del concurso 'Ciencia en Acción 2019'. URJC, on-line newspaper interview, July 2019.
- [11] J. Vega and J. Cañas. PyBoKids: An innovative python-based educational framework using real and simulated Arduino robots. *Electronics*, 8:899–915, August 2019.
- [12] J. Vega, E. Perdices, and J. Cañas. *Attentive visual memory for robot localization*, pages 408–438. IGI Global, USA, September 2012. Text not available. This book is protected by copyright.