UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2017/2018

Trabajo Fin de Grado

SCRATCH4ROBOTS

Autor : Santiago Carrión Vivanco

Tutor : Jose María Cañas Plaza

Memoria del Proyecto

Santiago Carrión Vivanco

11 de octubre de 2018

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Robótica

La palabra Robot se deriva de la palabra de origen checoslovaco robota, que quiere decir siervo. Dicha palabra apareció por primera vez en la literatura en la obra R.U.R (1921)(Robot Universalis de Rossum), de Karel Capek.

Un robot es un sistema electromecánico que utiliza una serie de elementos hardware (actuadores, sensores y procesadores) y cuyo comportamiento viene controlado por un software programable que le da la inteligencia. La robótica se puede ver como la ciencia y la tecnología de los robots, donde se combinan varias disciplinas como la mecánica, la informática, la electrónica y la ingeniería artificial, que hacen posible el diseño hardware y software del robot.

Los robots se componen esencialmente de tres tipos de dispositivos: sensores, procesadores y actuadores. En un robot los sensores son los encargados de recoger la información del entorno. En este grupo se situarían: el láser, el sonar o las cámaras. Estos dispositivos equivaldrían a nuestros sentidos humanos. Por otro lado se encuentran los procesadores, encargados de analizar los datos que le son suministrados por los sensores, también son los encargados de elaborar una respuesta a estos datos y enviar la acción que deba llevarse a cabo a los actuadores, son como nuestro cerebro. Por último los actuadores, principalmente motores eléctricos, se encargan de interactuar con el entorno del mismo modo que lo hacen nuestros músculos.

1.1.1. Historia

A finales del siglo XIX se presentan las primeras máquinas *robots*, pero no será hasta la segunda guerra mundial cuando se realicen los primeros diseños de esta naturaleza.Con el desarrollo de las primeras computadoras digitales se produjo una considerable evolución en este campo. Así, en 1970 una serie de investigadores del Instituto de Investigación de Stanford desarrollaron Shakey (figura 1.1). Su sistema de control creaba una reproducción interna del entorno a partir de los sensores de que disponía y desde ella calculaba su movimiento.

Aunque es un termino relativamente nuevo y cuyo uso extendido es muy reciente, la historia lleva siglos dando pasos para llegar al punto en el que nos encontramos.

Sin duda uno de los mayores impulsos en este mundo se produce con la llegada de la industrialización masiva. A principios de los 60, se instaló en una cadena de General Motors el primer robot industrial, Unimate 1.4, que realizaba tareas en la cadena de producción de los vehículos que podían ser peligrosas para los trabajadores.

En la misma década se creó el robot Shakey, el primero que combinó razonamiento lógico con acción física 1.5. Al igual que en el caso que trata este trabajo, Shakey utilizaba Planificación Automática para determinar sus acciones.

En 1970 se continua con la dinámica de crecimiento del sector, debido esta vez a la evolución del software ocurrida en esta época, y no parará hasta nuestros días, teniendo un crecimiento exponencial con una fuerte correlación con la evolución de software y hardware en general. Desde este momento todas las grandes empresas dotaran sus fábricas de robots industriales.

En la actualidad uno de los mayores logros en el mundo de la robótica se da en 2011, la NASA lanzó al espacio el robot tipo vehículo explorador *Curiosity* 4.4 , que aterrizó en Marte al año siguiente. Su principal cometido es investigar la capacidad pasada y presente del planeta para alojar vida.

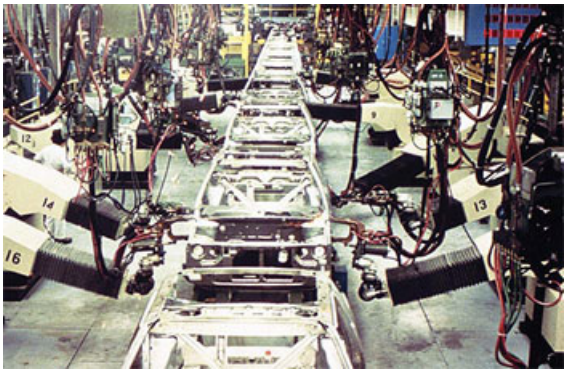

Figura 1.1: Unimate, General Motors

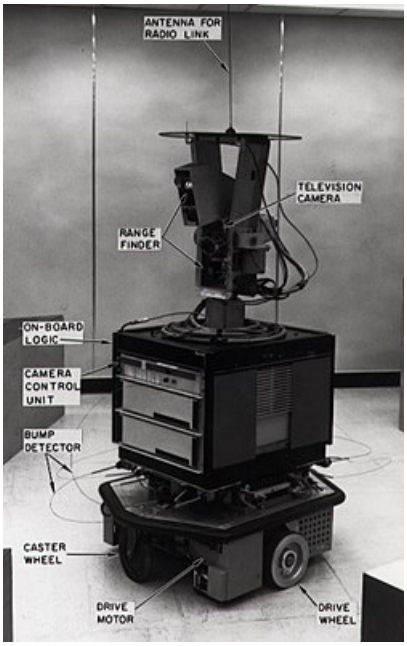

Figura 1.2: shakey en 1972

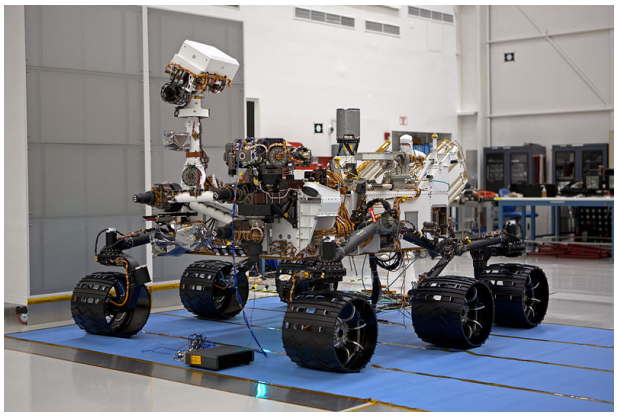

Figura 1.3: El Curiosity en el Laboratorio de Propulsión a Chorro de la NASA

1.1.2. Diferentes aplicaciones

Actualmente la robótica se encuentra en muchos ambitos de nuestra vida cotidiana, algunas aplicaciones son:

- **Robótica educativa**: Es una disciplina que trabaja en la concepción creación e implementación de prototipos robóticos y programas con fines pedagógicos. Con ello se permite al alumno fabricar sus representaciones sobre los fenómenos del mundo, facilitar su adquisición y trasferencia a distintas áreas de conocimiento. A través de la robótica educativa,

los docentes pueden desarrollar de una forma práctica los conceptos teóricos que suelen ser abstractos y confusos, además despierta el interés del alumno por esos temas y relaciona al niño con el mundo tecnológico en el que se mueve. El empleo de un ambiente de aprendizaje basado en la robótica educativa ayuda al desarrollo de nuevas habilidades y conceptos, fortalece el pensamiento lógico, estructurado y formal del alumnado, desarrollando su capacidad para resolver problemas concretos.

Una de las características de este ámbito es la capacidad que posee para mantener la atención del alumno, ya que manipula y experimenta haciendo que se concentre en sus percepciones y observaciones sobre la actividad que realiza. Actualmente existen varios kits de robótica, algunos de ellos son: LEGO MINDSTORMS education, LEGO WeDo, LEGO NXT o Parallax Scribbler. Además, también existe la posibilidad de trabajar con programas con los que controlar y simular diferentes La robótica en Educación Infantil. Realidades y limitaciones robots como son NXT-G Educación, ROBOTC, ROBOLAB o Microsoft Robotics Developer Studio.

- **Medicina**: involucra en si varios ámbitos ya sea en cirugías de alto riesgo, en rehabilitaciones, en ayuda a personas con enfermedades de movilidad o discapacitados, en el almacenamiento de medicamentos y también en lo que se trata en pruebas ficticias y cirugías computarizadas.

- **Militar**: la robótica donde más ha evolucionado es en la creación de vehiculos autónomos, tecnología que tras unos años pasa a a ser de uso civil. Algunas de las creaciones que nos hemos heredados es el UAV, que se conocen comúnmente como dron, es una aeronave que vuela sin tripulación, Capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por un motor de explosión, eléctrico, o de reacción.

- **Robotica aeroespacial**: Uno de los mayores impulsores de la robótica ha sido la conquista del espacio, esta es una parte fundamental de la exploración espacial, debido a la imposibilidad de ser realizada en primera mano por un humano. La idea básica sobre Robots Espaciales consiste en utilizar Inteligencia Artificial para permitir a los robots realizar labores de exploración como si de un humano se tratase, desplazarse en terrenos y ambientes complejos de forma autónoma sin más conocimiento que lo recibido por sus sensores. Se busca no solo el proceso de pensamiento y análisis de los humanos en de-

terminar las características del terreno, sino también la habilidad humana de conducir un vehículo en tiempo real.

- **Industrial**:las aplicaciones de la robótica industrial son cada vez mayores, algunos ejemplos son:

  - Movimiento de material en almacenes: Un robot móvil es capaz de desplazarse por el almacén y recoger las unidades y referencias que incorporan un pedido concreto para un cliente.
  - Procesos de cadenas de montaje:un robot antropomórfico es capaz de realizar movimientos complejos para colocar las distintas piezas que forman un producto final.
  - Empaquetado de producto: La visión es de uso recurrente en el empaquetado correcto de distintos productos genéricos y de alimentación.
  - Procesos de de alta precisión: por ejemplo en la industria aeronáutica se utilizan cabezales de robots multifuncionales para el remachado en el fuselaje de un avión.
  - Seguimiento y verificación de productos: Un robot es capaz de realizar una evaluación detalle del productor final, indicando si en el proceso de fabricación ha habido algún desperfecto o incorreción.

- **Automovilismo**: Los coches autónomos actualmente son una relidad, aún queda mucho desarrollo pero ya existen diversos modelos de coche que se comercializan al público, esto es debido a la robustez que se ha conseguido en la lígica que gobierna estos coches. Se trata de uno de los retos cercanos más importantes de la robótica.
  Para alcanzar una conducción realmente automática en una situación urbana con tráfico impredecible son necesarios muchos sistemas de tiempo real, que deben interoperar. Por ejemplo, es necesario un sistema de localización, de percepción del entorno, de planificación y lógicamente, un sistema de control. Además, son necesarios un conjunto de sensores que recojan y proporcionen la información necesaria para poder tomar las decisiones.

- **Domótica**:Es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta

seguridad y confort, además de comunicación directa con los diferentes sistemas y elementos de una casa a través de la red.

Un sistema domótico es capaz de recoger información proveniente de unos sensores o entradas, procesarla y emitir órdenes a unos actuadores o salidas. El sistema puede acceder a redes exteriores de comunicación o información.

Es difícil no parase a pensar en la potencia que tiene esta rama de la ciencia y la tecnología, y no sólo eso, sino también en la que puede alcanzar, ya que esto es sólamente el principio. En todo lo anterior, sale a relucir la utilidad de esta área de desarrollo, que permitirá en el futuro ganar en comodidad, economía e incluso salud. Es importante advertir la importancia de dominar esta disciplina en el futuro cercano, ya que será la llave que abrirá la puerta hacia un mundo más sencillo y seguro a través de la automatización.

1.1.3. Tipos de Robots

Ningún autor se pone de acuerdo en cuántos y cuáles son los tipos de robots y sus características esenciales. La más común son la que a continuación se presentan:

Segun su Estructura:

- **Androides**: Un robot humanoide que se limita a imitar los actos y gestos de un controlador humano, no es visto por el público como un verdadero androide, sino como una simple marioneta animatrónica. El androide siempre ha sido representado como una entidad que imita al ser humano tanto en apariencia, como en capacidad mental e iniciativa

- **Poliarticulados**: En este grupo pueden encontrarse robots de diversas formas y configuraciones, pero su característica en común es la de ser sedentarios. Es decir, que son robots estructurados para moverse en un determinado espacio de trabajo, según uno o más sistemas de coordenadas y con un número limitado de grados de libertad.

- **Móviles**: estos robots cuentan con orugas, ruedas o patas que les permiten desplazarse de acuerdo a la programación a la que fueron sometidos. Estos robots cuentan con sistemas de sensores, que son los que captan la información que dichos robots elaboran. Los móviles son utilizados en instalaciones industriales, en la mayoría de los casos para transportar la mercadería en cadenas de producción así como también en almacenes. Además,

son herramientas muy útiles para investigar zonas muy distantes o difíciles de acceder, es por eso que en se los utiliza para realizar exploraciones espaciales o submarinas.

- **Zoomórficos**: la locomoción de estos robots imita a la de distintos animales y se los puede dividir en caminadores y no caminadores. Estos últimos están aún muy poco desarrollados mientras que los caminadores sí lo están y resultan útiles para la exploración volcánica y espacial.

- **Híbridos**: Este término corresponde a todos aquellos robots que son difíciles de clasificar, ya que corresponden a una combinación de las estructuras anteriormente explicadas. Por ejemplo, un robot híbrido, que esté conformado por la segmentación de articulaciones y ruedas, podría ser considerado tanto móvil, así como zoomórfico.

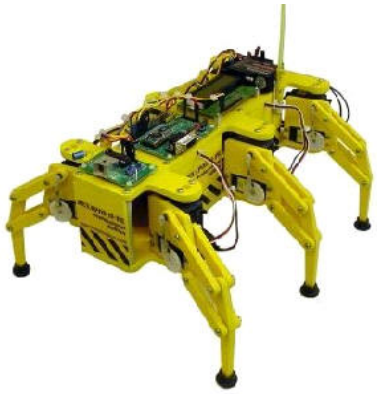

Figura 1.4: Robot zoomórfico

Figura 1.5: Robot poliarticulado

Según su Cronología:

- **1ª Generación. Manipuladores**: Son sistemas mecánicos multifuncionales con un sencillo sistema de control, bien manual, de secuencia fija o de secuencia variable.

- **2ª Generación. Robots de aprendizaje**: Repiten una secuencia de movimientos de movimientos que ha sido ejecutada previamente por un operador humano. El modo de hacerlo es a través de un dispositivo mecánico. El operador realiza los movimientos requeridos mientras el robot le sigue y los memoriza.

- **3ª Generación. Robots con control sensorizado**: El controlador es una computadora que ejecuta las órdenes de un programa y las envía al manipulador para que realice los movimientos necesarios.

- **4ª Generación. Robots inteligentes**: Son similares a los anteriores, pero además poseen sensores que envían información a la computadora de control sobre el estado del proceso. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

1.2. Software para robots

Muchos robots poseen autonomía, la cual proviene del desarrollo de sistemas complejos, aplicaciones e infraestructuras que les dotan de inteligencia autónoma. El desarrollo de software robótico es similar al desarrollo de software en otros ámbitos, donde se parte de ciertos requisitos y se modela un diseño que será creado. Hace años el desarrollo de software robótico se realizaba adoptando soluciones "ad hoc", dotando a cada robot de un diseño específico, y con sensores y actuadores concretos. Esto suponía que no se podía aplicar el software desarrollado a otro robot, por lo que era necesario implementar de nuevo todo el software para un nuevo robot. En la actualidad, existen numerosas plataformas que permiten el desarrollo de aplicaciones robóticas de forma eficiente y genérica. Esto permite reutilizar gran parte de las aplicaciones creadas en otros robots, evitando el coste de realizar todo el proceso de nuevo.

Dotar al robot de cierta inteligencia conlleva desarrollar cierto software, el cual se suele programar apoyándose en herramientas, como los *middleware* robóticos, los simuladores robóticos, o las bibliotecas que facilitan algunos aspectos. A continuación, se exponen algunas de estas herramientas que se emplean en la actualidad.

El software para drones a ido evolucionando con los años llegando a un punto en el que para aislar la complejidad que conlleva el desarrollo de aplicaciones para robots, esto lo consiguen dividiendo el problema con Middlewares.

1.2.1. Middlewares robóticos

Actualmente existen diversos *middlewares* robóticos, con los que somos capaces de gestionar la complejidad y heterogeneidad del hardware y las aplicaciones, promover la integración de tecnlogías en desarrollo, enmascarar los sistemas de percepción complejos, y simplificar el diseño de software.

Mediante el uso de middlewares somos capaces de introducir una capa de abstracción con los drivers y el hardware del robot, disminuyendo de manera importante la complejidad y los conocimientos necesarios para cualquier desarrollo. Además permite paralelizar de forma eficiente el trabajo.

(esquema de capas HW-Drivers-Middleware)

Algunos de los *middlewares* robóticos más destacados son:

- **ros**: Es una plataforma de software libre para el desarrollo de software de robots, que provee servicios estándar de un sistema operativo como la abstracción del hardware, el control de dispositivos de bajo nivel, mecanismos de intercambio de mensajes entre procesos y un conjunto de herramientas utilizadas ampliamente en robótica. La librería está orientada para un sistema UNIX, aunque se está adaptando a otros sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, considerados como "experimentales". En futuros capítulos hablarmemos más en profundidad sobre ros y su funcionamiento.

- Orocos: Es un proyecto de software libre para el control de robots y máquinas. Incluye cuatro bibliotecas C++: Real-Time Toolkit, Kinematics and Dynamics Library, Bayesian Filtering Library y Orocos Component Library. Está orientado a componentes, permitiendo añadir funcionalidades de forma sencilla y sin recompilar todo el código. Incluye paquetes complementarios tales como Filtros de Bayes, Librerías de control Dinámico y Cinemático o Visión.

- Orca: Se trata de una plataforma de software libre para el desarrollo de aplicaciones robóticas. Fundamentalmente orientada al desarrollo de componentes, proporciona los medios para definir y desarrollar los componentes, los cuales pueden unirse para formar sistemas robóticos de distintas complejidades, desde vehículos autónomos hasta redes de sensores distribuidas. Orca permite reutilizar código, de manera que se pueden emplear componentes robóticos ya creados.

- Urbi: Es una plataforma de software multiplataforma de código abierto en C++ utilizada

para desarrollar aplicaciones para robótica y sistemas complejos. Urbi es compatible con ros y se puede emplear en los sistemas operativos Linux, Windows y MAC OS X.

1.2.2. Simuladores robóticos

El diseño de un robot es costoso y caro, lo que implica que muchos componentes necesarios para la construcción de los robots solamente estén disponibles para centros de investigación y corporaciones. Cuando se emplea un robot puede que el código desarrollado falle al probarlo, pudiendo incluso romperse algún robot.

Hoy en día existen numerosos simuladores robóticos, lo que permite a cualquier persona crear, programar y probar infinidad de robots de forma segura y económica. Algunos de los simuladores más empleados son:

- Gazebo: Es un simulador 3D de código abierto distribuido bajo licencia Apache 2.0. Este simulador se ha utilizado en ámbitos de investigación en robótica e Inteligencia Artificial. Es destacado por sus motores de físicas, motor de renderizado avanzado, soporte para *plugins*, un amplio repertorio de robots comerciales, y una extensa gama de sensores y actuadores. Es fácil de integrar con ros.

- Stage: Simula robots móviles en el plano bidimensional y proporciona diversos tipos de sensores y actuadores. Su finalidad es ayudar a la investigación de sistemas autónomos de múltiples agente, para lo cual proporciona gran cantidad de dispositivos simultáneamente.

- Webots: Es un simulador avanzado de robótica, que permite definir modelos propios, definir la física, escribir controladores para los robots y hacer simulaciones a gran velocidad. Se puede emplear en los sistemas operativos Linux, Windows y MacOS. Los lenguajes de programación que se pueden emplear son C++, C y Java.

1.2.3. Bibliotecas

Las bibliotecas referidas al contexto de software son un conjunto de desarrollos que pueden ser usadas por terceros mediante una serie de interfaces bien definidas, que aportan de una funcionalidad añadida evitando tener que ser desarrollada desde un cero por el desarrollador que

hace uso de ellas. Esto hace que agilicen y ayuden de forma notable cualquier proyecto, ya que únicamente debes centrarte en lo relativo a tu lógica, mientras que las cosas comunes puedes integrarlas en tu código a traves de estas librerías.

Algunas de las bibliotecas utilizadas en robótica son:

- OpenCV: Está dirigida a la visión por computador en tiempo real. Entre las áreas de aplicación de esta biblioteca destacan: segmentación y reconocimiento de objetos, reconocimiento de gestos, seguimiento del movimiento, estructura del movimiento, y robots móviles.

- PCL: Se utiliza para el procesamiento digital de imágenes RGBD mediante el tratamiento de nubes de puntos 3D. Contiene numerosos algoritmos de última generación que incluyen filtrado, estimación de características, reconstrucción de superficies, ajuste de modelos y segmentación entre otros. Para simplificar el desarrollo, PCL se divide en una serie de bibliotecas de código más pequeñas, que se pueden compilar por separado. Es multiplataforma y ha sido compilada con éxito en Linux, Mac OSX, Windows y Android / iOS

- AForge.NET: Es un framework C# de código abierto diseñado para desarrolladores e investigadores en los campos de Visión por Computadora e Inteligencia Artificial. Sus áreas de aplicación son: procesamiento de imágenes, redes neuronales, algoritmos genéticos, lógica difusa, aprendizaje de máquinas, robótica, etc.

1.3. Docencia en robótica

Actualmente la robótica es un mercado al alza, esto hace que aumente de manera importante el número de científicos, ingenieros y técnicos demandados para trabajar e investigar en este sector. Es importante una base sólida en conceptos de programación, procesado de imágenes, calculo, álgebra, electrónica, electricidad etc. Al ser un mundo complejo lleno de retos por resolver se necesita gente preparada, de ahí lo importante que es acercar a los más jovenes desde los colegios e institutos a estas disciplinas.

Es por esto que la docencia en robótica intenta despertar el interés de los estudiantes transformando las asignaturas tradicionales en más atractivas e integradoras, ya que crea entornos de aprendizaje propicios que recrean los problemas del entorno que los rodea.

En el futuro, tener nociones básicas de esta disciplina será clave debido a que cada vez de forma más habitual se implantan robots en diferentes sectores laborales.

La enseñanza de robótica en secundaria se realiza mediante plataformas físicas como los robots LEGO (Mindstorms, RCX, NXT, Evo, WeDo), placas Arduino, los kits de SolidWorks, etc que con una simpleza espectacular son capaces de motivar al alumno ya que obtiene resultados vistosos y a los que le puede dar una aplicación en su vida cotidiana, despertando su interes en esta materia.
Otro sistema introducido en los últimos años en la educación es la programación de robots mediante lenguajes de programación visual, potentes lenguajes, que abstrayendo al alumno de la complejidad de la sintaxis, siendo muy intuitivos y dandole un apoyo visual vistoso hacen que estos softwares sean muy bien aceptados por los estudiantes de menor edad.

1.4. Lenguajes de programación visual

Un lenguaje de programación visual es cualquier lenguaje de programación que permite a los usuarios crear programas manipulando elementos del programa gráficamente en lugar de especificarlos textualmente. Permite la programación con expresiones visuales, arreglos espaciales de texto y símbolos gráficos, utilizados como elementos de sintaxis o notación secundaria. Por ejemplo, muchos se basan en la idea de çajas y flechas", donde las cajas u otros objetos de pantalla se tratan como entidades, conectadas por flechas, líneas o arcos que representan relaciones, mientras que otros se basan en el apilamiento de çajasçon una funcion predefinida, creando así varios flujos de acciones programáticas con una objetivo final.

Estos lenaguajes por regla general se usan en programación dirigida por eventos, La programación dirigida por eventos es un paradigma de programación en el que el flujo del programa está determinado por eventos o mensajes desde otros programas o hilos de ejecución. Las aplicaciones desarrolladas con programación dirigida por eventos implementan un bucle principal

o main loop donde se ejecutan las dos secciones principales de la aplicación: El selector de eventos y el manejador de eventos.

Podemos diferenciar varios niveles dentro de los lenguajes de programación visual:

- **Sintaxis**: en el nivel sintáctico, la explicación del bloque está limitada a estructura del lenguaje. Por ejemplo, una explicación podría revelar que una condición es parte de una declaración *IF* y no debe ser confundido con una acción que se puede ejecutar en *THEN* o *ELSE* parte de una declaración. Sin embargo, este no trata de definir de forma específica qué es lo que define esa condición. Intentan reducir o incluso eliminar por completo los errores sintácticos y ayudan a la creación de programas bien formados. Una analogía sería el corrector ortográfico en procesadores de texto que subraya o incluso corrige automáticamente palabras o gramática individuales.

- **Semántica**: en el nivel de la semántica,se dan explicaciones sobre los bloques, a menudo implementadas a través de funciones de ayuda que describen el significado de un bloque. El usuario obtiene una respuesta semántica en forma de panel de ayuda genérico incluyendo una breve descripción del significado del comando o bloque y la lista de opciones adicionales.

- **Pragmática**: permiten llevar nuestra implementación a un punto de testeo específico, nos ayudan con una serie de módulos propios a crear el entorno necesario para simular el funcionamiento de nuestro programa en ese estado.

1.4.1. Motivación

Las principales motivaciones del uso de estos lenguajes son su **accesibilidad**, ya que no requiere de una infraestructura potente para su funcionamiento. Además el **facil aprendizaje** hace que sea un lenguaje idóneo para aquellos que no tienen ningún conocimiento de informática previo. Y por último el eliminar de la ecuación algo tan simple como los **errores de sintaxis**, nos olvidamos completamente de este tipo de fallos, haciendo el desarrollo más fluido y menos frustrante para los principiantes, algo que puede marcar la diferencia a la hora de encontrar gratificante el desarrollo de aplicaciones con estos lenguajes.

1.4.2. Ejemplos

- **Blocly**: El desarrollo de Blockly comenzó en el verano de 2011, y el primer lanzamiento público fue en Maker Faire en mayo de 2012. Blockly fue originalmente diseñado como un reemplazo para OpenBlocks en App Inventor. [3] Neil Fraser comenzó el proyecto con Quynh Neutron, Ellen Spertus y Mark Friedman como colaboradores. Es un proyecto de Google y es de código abierto bajo la licencia Apache 2.0 Por lo general, se ejecuta en un navegador web y se asemeja visualmente a Scratch. Blockly también se está implementando para Android e iOS aunque no todas las funciones basadas en el navegador web están disponibles para Android / iOS. Utiliza bloques visuales que se unen entre sí para facilitar la escritura de códigos y generar código JavaScript, Python, PHP o Dart. También se puede personalizar para generar código en cualquier lenguaje de computadora textual.

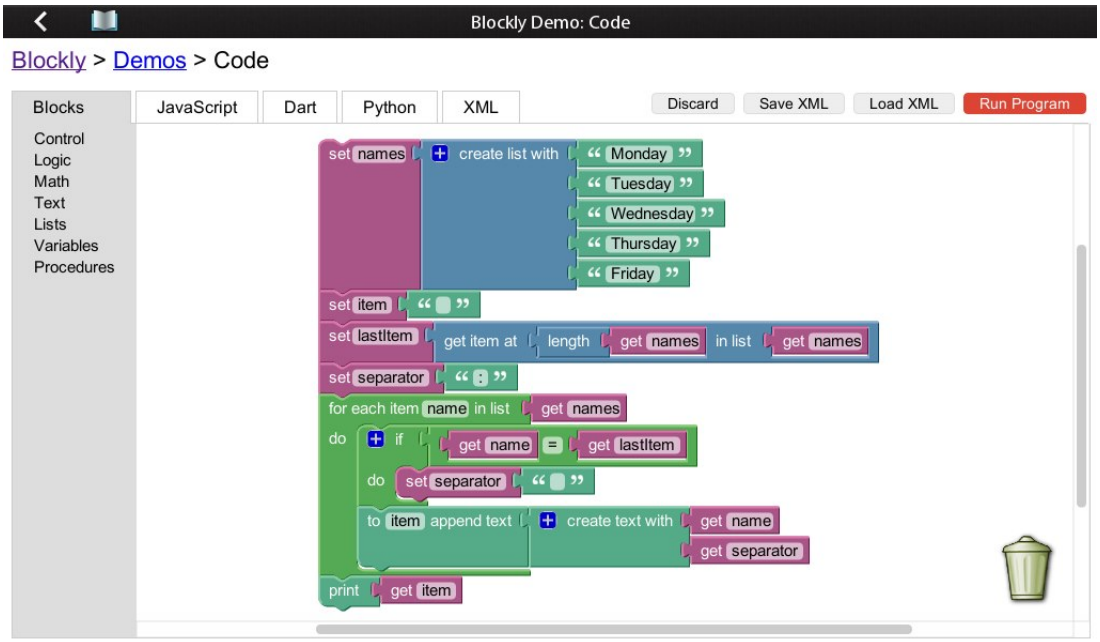

Figura 1.6: Lenguaje de programación visual Blockly

- **Scratch**:es un proyecto del Grupo Lifelong Kindergarten del MIT Media Lab. Es utilizado por estudiantes y docentes de todo el mundo para expresar ideas mediante animaciones, juegos e interacciones facilmente programables con este entorno, hablaremos en capitulos siguientes.

- **Snap!**:desarrollada por la Universidad de California en Berkeley, que sigue la filosofía de

facilidad y sencillez para aprender a programar, Snap se basa en el conocido programa de Scratch, siendo su uso más extendido entre edades más maduras que las de Scratch. Snap está programado en JavaScript. Esto hace que podamos usarlo desde cualquier navegador, ya sea desde un ordenador como desde las tablets. En las versiones actuales de Scratch (1.4 ó 2.0) se requiere el uso del plugin privativo de flash player. Si bien es cierto que a partir de la próxima versión 3.0 estará desarrollado íntegramente en HTML5 para que pueda funcionar en la mayoría de dispositivos móviles y tablets. Puedes acceder a este enlace para más información.

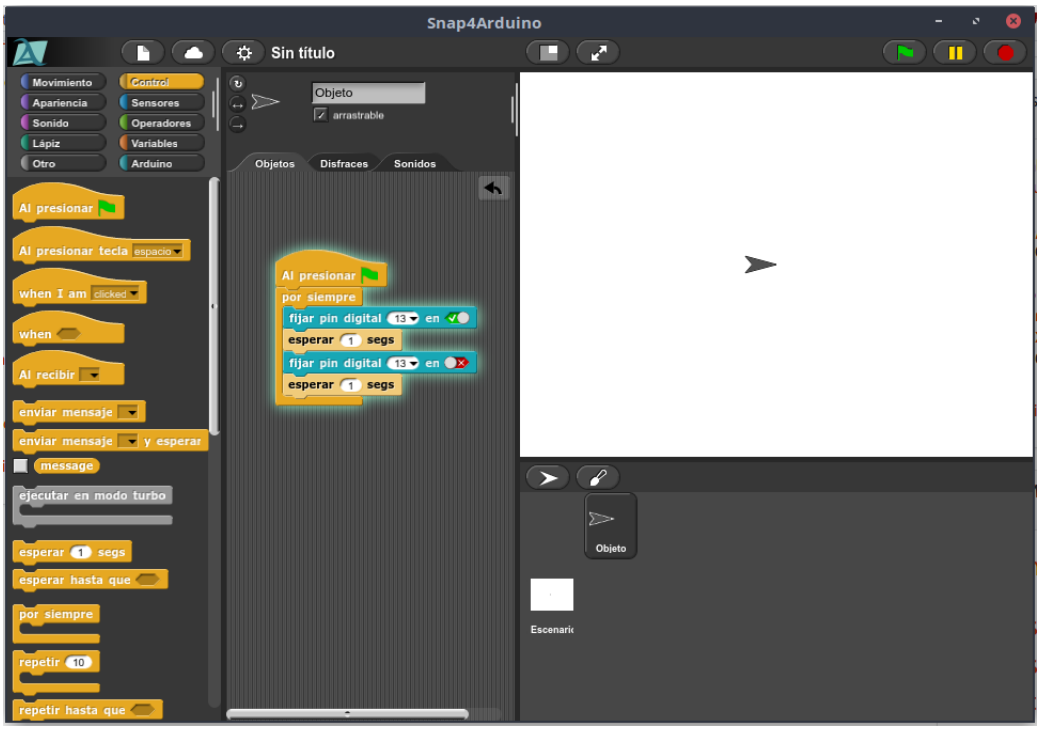

Figura 1.7: Lenguaje de programación visual Snap!

- **Kodu**: originalmente llamado Boku, es un entorno de desarrollo integrado de programación (IDE) de los laboratorios FUSE de Microsoft. Se ejecuta en Xbox 360 y Microsoft Windows XP, Windows Vista, Windows 7, Windows 8 y Windows 10. Fue lanzado en el Xbox Live Marketplace el 30 de junio de 2009. Una versión de Windows está disponible para el público en general para su descarga desde el portal web FUSE de Microsoft.

- **LEGO**:

  TODO

- **VIsualstates**:VisualStates, herramienta integrada en la suit de programación JdeRobot es una herramienta para la programación de comportamientos de robot utilizando máquinas de estados finitos de jerarquía (HFSM - Hierarchichal Finite State Machine). Representa gráficamente el comportamiento del robot en un lienzo compuesto por estados y transiciones. Cuando el autómata está en cierto estado, pasará de un estado a otro según las condiciones establecidas en las transiciones. Esta representación gráfica permite un mayor nivel de abstracción para el usuario, ya que solo tiene que preocuparse por programar las acciones actuales del robot y seleccionar qué componentes puede necesitar de la interfaz del robot.

- **Choregraphe**:Es el software de programación que permite a los usuarios crear y editar movimientos y comportamientos de NAO de manera sencilla.

  Su intuitiva interfaz gráfica, su biblioteca de comportamientos y las funciones de programación hacen posible realizar desde tareas simples como de nivel avanzado.

  Es posible crear comportamientos utilizando la biblioteca de bloques de comportamiento con un simple arrastrar/copiar. primite la programación basada en eventos, secuencial o en paralelo. Además la línea de tiempo permite programar por medio de secuencia lógica.

  Los bloques de comportamiento pre-programadas son fácilmente configurables y con el editor de curvas o por medio de lenguaje Python.

Capítulo 2

Objetivos

Tras haber expuesto las motivaciones y el contexto en el que se engloba este proyecto de fin de grado, en este capítulo daremos una explicación más detallada del problema que se intenta resolver y los pasos dados para conseguirlo.

2.1. Descripción del problema

En la actualidad la robótica está en nuestro alrededor en todo momento, debido a las nuevas tecnologías y al abaratamiento de costes se encuentra en pleno auge. A pesar de que convivimos a diario con ella por lo general sigue siendo un enigma para la mayoría de las personas.

No es menos cierto que existe una enorme complejidad que requiere de un alto conocimiento de las tecnologías implicadas tras la configuración de un robot, y es esta barrera la que queremos eliminar.

Con este proyecto buscamos el acercamiento de la robótica a un público con escasos conocimientos técnicos, simplificando al máximo toda la complejidad que existe tras la programación de un robot, hasta el punto que sea usada por niños para el aprendizaje, haciendo el mundo de la robótica más cercano y más atractivo a ojos de aquellos que serán el futuro de ésta.

Esto lo conseguimos creando un nexo entre un lenguaje de programación visual mediante bloques, para esto hacemos uso de *Scratch*, y la programación de robots en Python. Con *Scratch4Robots* conseguimos que partiendo de un lenguaje fácil e intuitivo, basado en el apilamiento de bloques funcionales de código, la traducción a código Python completamente funcional. Creando unos bloques con funcionalidad dirigida a robots en especifico conseguimos que

esta traducción pueda ser aplicada a robots.

2.2. Requisitos

Para cumplir los objetivos marcados de forma satisfactoria, debemos además satisfacer los siguientes requisitos:

- El desarrollo deberá ser autocontenido en lo que sea posible, esto quiere decir que todas las librerías y dependencias de nuestra herramienta deberán estar contenidas en su interior. La programación se realizará en el lenguaje Python.
- El software desarrollado deberá ser compatible con Ubuntu 16.04, ROS-kinetic y Scratch 2.0 serán los únicos elementos indispensables para el correcto funcionamiento de nuestra herramienta.
- Todos los componentes desarrollados deberán ser compatibles tanto trabajando en entornos simulados, como usando robots reales, esto se consigue usando todas las abstracciones de las que nos provee JdeRobots, totalmente testadas en robots reales.

2.3. Metdología de trabajo

Dada la naturaleza del proyecto, y al igual que en cualquier otro proyecto de software, es necesario el uso de un modelo que defina el ciclo de vida de la aplicación. Para el desarrollo de este proyecto hemos decidido adoptar el modelo de desarrollo en espiral. TODO BIBLIOGRAFIA El modelo en espiral consiste en una serie de ciclos que se repiten en forma de bucle, cada ciclo representa un conjunto de actividades. Las actividades no están prefijadas *a priori*, sino que se eligen en función de las realizadas previamente. Estos ciclos se irán ejecutando hasta que la aplicación sea aceptada y no requiera otro ciclo. Este modelo, definido por Barry Boehm en 1986, se basa en una espiral en la que cada iteración representa un conjunto de actividades.Estas actividades no tienen prioridad, se elegirá en la fase de análisis de riesgos. Así, cada iteración está dividida en las siguientes actividades:

- Determinar objetivos: en esta actividad se definirá el objetivo de la iteración actual. Siguiendo este modelo el objetivo final del proyecto se divide en sub-objetivos.

- Análisis de riesgo: en esta actividad, se lleva a cabo varios estudios con el propósito de conocer las posibles amenazas o eventos no deseados que se puedan producir en el objetivo actual.

- Desarrollar y probar: llegados a este punto será necesario la verificación del correcto funcionamiento realizado en la iteración para subsanar los errores y que estos no prosigan en las siguientes iteraciones.

- Planificación: en esta actividad se revisarán las fases anteriores para determinar si se debería continuar.

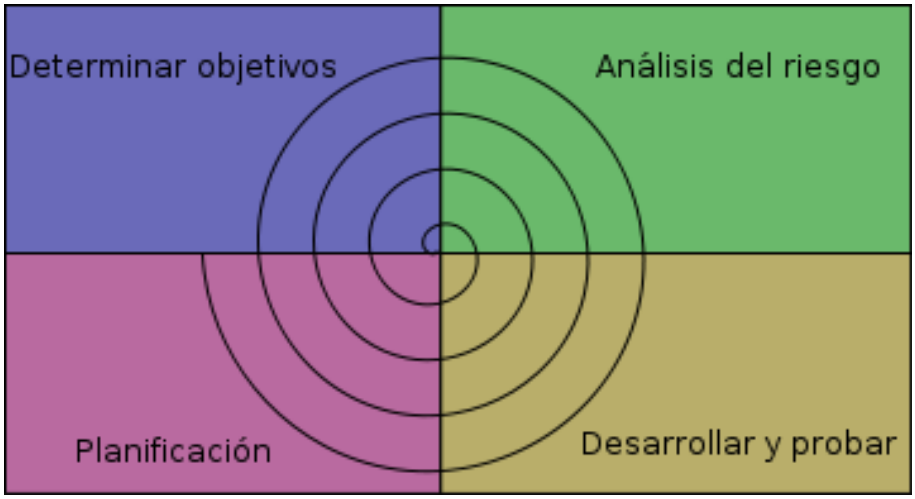

Figura 2.1: Desarrollo en espiral

Para materializar estas actividades e iteraciones mantuvimos reuniones semanales durante todo el desarrollo del trabajo. De este modo, cada semana marcábamos un nuevo subobjetivo. Si el anterior se había completado, planificábamos el siguiente. Si por el contrario, no sé había completado, ahondábamos en el objetivo actual para corregir los errores o volver a planificarlo.

Todo el código fuente generado se mantuvo en un repositorio con un sistema de control de versiones (SVN), *Git* en nuestro caso, accesible desde internet[TODO enlace a git].

De este modo el tutor y cualquier otra persona interesada tiene acceso en cualquier momento al código, además nos apoyamos del sistema de apertura de *issues* en el repositorio remoto *Git* para cada implementación que se necesitaba desarrollar para la mejora del proyecto.

2.4. Plan de trabajo

Simplificaremos la labor a desarrollar en varios puntos:

- **Familiarización con el entorno software**: JdeRobot es la plataforma de desarrollo principal utilizada en la mayoría de proyectos realizados en el departamento de robótica de la URJC. El objetivo principal de esta fase es aprender a utilizar este software, sus componentes y sus drivers para más adelante utilizarlos como parte de nuestro proyecto. Como parte del aprendizaje, se desarrolla una herramienta externa al core principal pero que usa de librerías y recursos de ella.

- **Estudio de KURT**: Librería que nos permite obtener toda la información necesaria de un proyecto Scratch, conocimiento de su API y su funcionamiento interno para saber qué podemos llegar a obtener y como usar esa información para proporcionar una traducción robusta al lenguaje Python.

- **Desarrollo de funcionalidades**: Aumentamos el número de bloques robóticos própios que podremos usar en Scratch, esto es desarrollar la lógica detrás de cada bloque, todo programado en python y la integración de estos bloques con Scratch. Dividimos los bloques entre aptos para drones y robots con ruedas. Estos bloques deben tener una funcionalidad muy específica y funcionar en armonía con el resto, tanto los propios de la aplicación Scratch como los nuevos bloques generados por nosotros propios de aplicaciones robóticas.

- **Facilitar el uso de la herramienta**: Haciendo el uso de la herramienta lo más intuitiva posible, mejorando scripts de lanzamiento y de generación de código. Creando ejemplos autocontenidos para una rápida demostración de la potencia de la herramienta. Generando tutoriales tanto escritos como con vídeo para evitar confusiones.

- **Integración**: La parte sin duda más compleja del desarrollo ya que se busca que nuestra aplicación sea fácilmente instalable en cualquier entorno y con la mayor facilidad posible, únicamente necesitando las dependencias que hemos comentado con anterioridad. Teniendo en cuenta que debe ser fácilmente usada por personas con pocos conocimientos técnicos.

Capítulo 3

Infraestructura

3.1. Scratch 2.0

Scratch es un proyecto del Grupo Lifelong Kindergarten del MIT Media Lab. Es utilizado por estudiantes y docentes de todo el mundo para expresar ideas mediante animaciones, juegos e interacciones facilmente programables con este entorno.

El nombre proviene de la palbra *scratching* que en el mundo de la informática se refiere a reutilizar código, el cual puede ser usado de forma beneficiosa y efectiva para otros propósitos y fácilmente combinado, compartido y adaptado a nuevos escenarios, lo cual es una característica clave de Scratch.

Es la segunda versión principal actual de Scratch, siguiendo a Scratch 1.4. Cuenta con un editor y un sitio web rediseñados, y permite al usuario editar proyectos directamente desde su navegador web, así como en un editor fuera de línea.

Se lanzó oficialmente en 2013 y ha sido completamente reescrito en Adobe Flash además debido a las nuevas características y al diferente lenguaje de programación, los proyectos de Scratch 2.0 se guardan en formato .sb2.

Scratch se define por una interfaz intuitiva y de simple manejo compuesta por tres zonas claramente diferenciadas. En la zona izquierda tendremos un escenario y un selector de *sprites*, en el centro encontramos las diferentes categorías de bloques que podemos utilizar y un listado de los bloques pertenecientes a la categoría seleccionada, y por último tendremos un espacio vacio en la parte derecha en la que crearemos nuestro proyecto, con el simple movimiento de arrastrar el bloque deseado.

Tendremos bloques de las siguientes categorias:

- **Movimiento**: Mueve objetos y cambia ángulos.

- **Eventos**: Contiene manejadores de eventos situado al principio de cada grupo de instrucciones.

- **Apariencia**: Controla el aspecto visual del objeto, añade bocadillos de habla o pensamiento, cambia el fondo, ampliar o reducir.

- **Control**: Sentencian condicionales.

- **Sonido**: Reproduce ficheros de audio y secuencias programables.

- **Sensore**: Los objetos pueden interactuar con el ambiente que ha creado el usuario.

- **Lápiz**: Control del ancho, color e intensidad del lápiz.

- **Operadores** Operadores matemáticos, generador aleatorio de números, operadores booleanos.

- **Datos**: Creación de variables y listas.

- **Mas Bloques**: Dispositivos o bloques externos creados por el usuario.

Al ser una herramienta madura y muy extendida existe una gran comunidad con la que compartir y de la que obtenter proyectos.

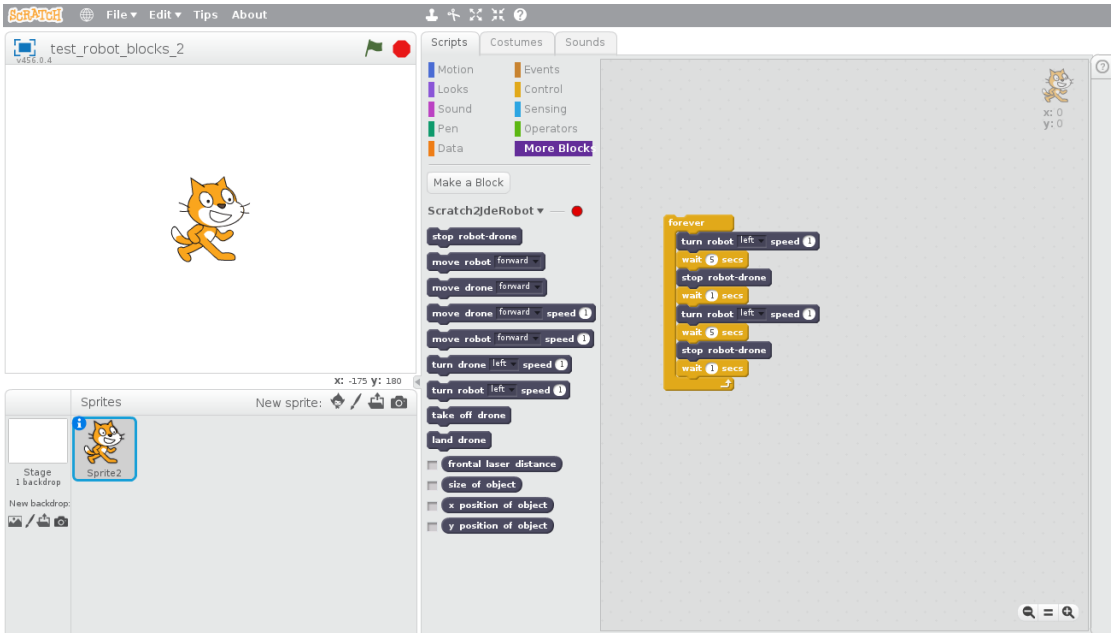

Figura 3.1: Espacio de trabajo de Scratch 2.0

3.2. JdeRobot

Nace de la tesis doctoral de Jose María Cañas en el año 2003 llevando desde ese momento en continua evolución y desarrollo, adaptandose a las tecnologías del momento. Es una suite de desarrollo de software de robótica, domótica y sistemas de visión computerizados cuya última versión, la 5.6, es la usada en este proyecto y permite la integración con ROS Kinetic. Proporciona un entorno distribuido donde las aplicaciones se forman mediante una colección de componentes asíncronos concurrentes, que se conectan mediante el middleware de comunicación ICE o mensajes ROS. Se compone de interfaces, drivers, utilidades y aplicaciones para el desarrollo de cualquier proyecto de robótica En este proyecto nos ayudamos de librerías como pueden ser *comm, Config, JdeRobotTypes* que nos ayudan a agilizar el desarrollo de nuestra herramienta, abstraernos de ciertos niveles de complejidad y hacemos uso de una cantidad de entornos simulados en Gazebo con los que probar el correcto funcionamiento.

3.3. Librería Comm

Librería desarrollada por JdeRobot con versiones tanto en python como en C que nos abstrae del tipo de comunicación utilizada por nuestros componentes. Apoyandose en las librerías ya definidas en JdeRobot para el fácil uso de nodos ROS y Ice crea una capa de abstracción que permite que una aplicación sea capaz de funcionar tanto con *ROS* como con comunicación *Ice* sin necesidad de modificar el código interno de esta. De esta forma se aprovecha el trabajo anterior, se economiza tiempo, y se reduce el código redundante. Su funcionamiento se apoya en el uso de un fichero de configuración necesario para establecer el tipo de comunicación que usaran los sensores y actuadores de nuestro robot. De este fichero obtendremos el tipo de comunicación y toda la información necesaria para poder establecerla. Se trata de un paso más en la reducción de complejidad a la hora de programar algo tan complejo como un robot.

3.4. Kurt

Kurt es una biblioteca de Python que permite la manipulación compleja de proyectos de Scratch (archivos .sb y .sb2) a través de simples comandos de Python. Incluye un compilador y decompilador, que permite que un proyecto se cargue en un conjunto de objetos Python, y un compilador que permite cargar un conjunto de scripts de imágenes o texto en proyectos. Al ser capaz de extraer toda la información contenida en un proyecto scratch, y debido al parecido en la sintaxis de scratch con python nos sirve como principal fuente de recursos a la hora, por ejemplo, de realizar una transcripción de un bloque de scratch a una sentencia python. Se tratará con más profundidad el funcionamiento de esta librería en siguientes capítulos.

3.5. Python

Python es un lenguaje de programación administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License.

Es un lenguaje interpretado, por lo que no se necesita compilar el código fuente para poder ejecutarlo, esto ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad, en ciertos casos, cuando se ejecuta por primera vez un código, se producen unos

bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.

Lenguaje muy popular en los últimos años gracias a la cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero, ademas cuya filosofía hace hincapié en una sintaxis que favorezca un código legible, facilitando su aprendizaje.

Su sintaxis muy visual, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave begin y end. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

Se trata de un lenguaje de propósito general y multiplataforma, Aunque originalmente se desarrolló para Unix actualmente cualquir sistema es compatible con el lenguahe siempre y cuando exista un intérprete programado para él.

Pese a ser un lenguaje multiparadigma, principalmente está orientada a objetos lo que permite en muchos casos crear programas con componentes reutilizables.

Tambien permite programación imperativa y programación funcional por lo que se adapta al estilo del programador y no al reves.

3.6. Gazebo

Es un simulador 3D para robots, es un proyecto de Open Source Robotics Fundation distribuido bajo la licencia Apache 2.0, con un motor de físicas y cinemáticas muy potente, es la principal herramienta en la que se apoya el desarrollador para verificar de forma segura que su implementación cumple con el objetivo determinado. Este tipo de simuladores han conseguido una fuerte evolución del mundo de la robotica ya que nos permite desarrollar componentes para dispositivos y robots complejos de forma barata y rápida. Se trata de una herramienta altamente personalizable y moldeable que admite plugins que permiten una gestión más fina de los recursos de cara a conseguir una simulación más precisa o simplemente perimitir al desarrollador trabajar con mayor comodidad sobre el simulador. Al ser un proyecto open source existe una

comunidad muy activa detras de esta herramienta compartiendo conociemintos y publicando nuevas funcionalidades de esta. Cabe destacar que Gazebo se compone principalmente de un cliente y un servidor. El servidor es el encargado de realizar los calculos y la generación de los datos de los sensores,además de poder ser usado sin necesidad de una interfaz gráfica. El cliente proporciona una interfaz grágica basada en QT que incluye la visualización de la simulación y una serie de controles de multitud de propiedades. Esta configuración permite lanzar multiples clientes sobre un servidor, consiguiendo multiples interfaces de la misma simulación. Destacar que en 2013, este simulador se utilizó en la Virtual Robotics Challenge, un componente del DARPA Robotics Challenge. En este proyecto Gazebo se emplea para probar la solución desarrollada en un entorno controlado para después, pasar a un entorno real.

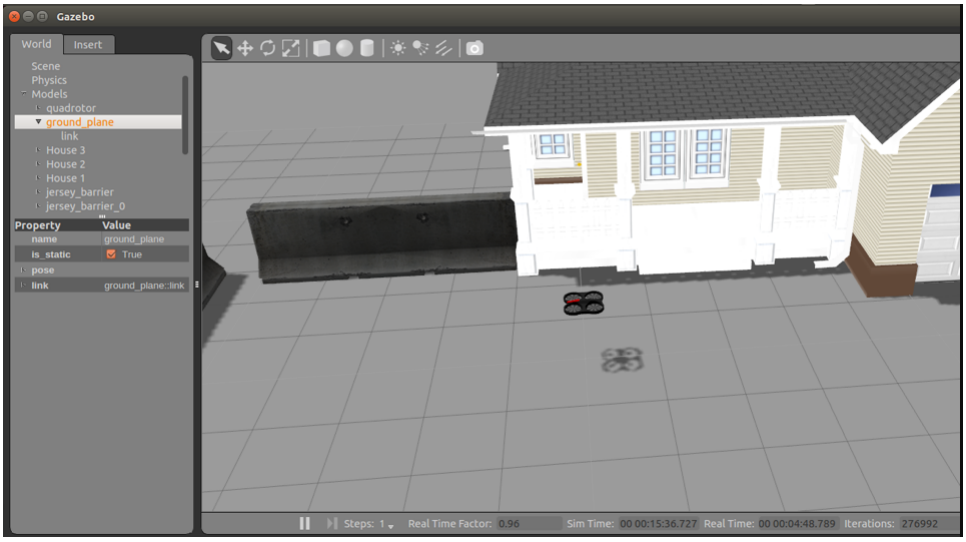

Figura 3.2: Entorno de simulación Gazebo

3.7. ROS

Es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS (Robot Operating System) proporciona bibliotecas y herramientas para ayudar a los desarrolladores de software a crear aplicaciones de robots. Proporciona abstracción de hardware, controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, administración de paquetes y más.

Es completamente de código abierto (BSD) y gratuito para que otros lo usen, cambien y comercialicen. SU objetivo principal es permitir que los desarrolladores de software creen apli-

caciones de robots más capaces de forma rápida y fácil en una plataforma común.

ROS se desarrolló originalmente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford.

Desde 2008, el desarrollo continua primordialmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado

Vamos a describir a desde una perspectiva a muy alto nivel los conceptos básicos en los que se basa su arquitectura.

3.7.1. Maestro ROS

El Maestro permite que todas las demás piezas de software de ROS (Nodos) encuentren y hablen entre sí. Sin el Maestro, los nodos no podrían encontrarse, intercambiar mensajes o invocar servicios. De esta forma, no tenemos que especificar nunca específicamente .[En]viar esta información del sensor a esa computadora en 127.0.0.1. Podemos simplemente decirle al Nodo 1 que envíe mensajes al Nodo 2. El ROS Master proporciona registro y búsqueda de nombres para el resto del gráfico de computación.

BIBLIOGRAfIA

3.7.2. Nodos

Algunos robots llevan varias computadoras, cada una de las cuales controla un subconjunto de los sensores o actuadores del robot. Incluso dentro de una sola computadora, a menudo es una buena idea dividir el software del robot en pequeñas partes independientes que cooperan para lograr el objetivo general. Para esto existen los nodos de ros. Los nodos son procesos que realizan cálculos. ROS está diseñado para ser modular en una escala de grano fino; un sistema de control de robot generalmente comprende muchos nodos. Por ejemplo, un nodo controla un láser, un nodo controla los motores de las ruedas, un nodo proporciona una vista gráfica del sustema y así sucesivamente. Un nodo ROS se escribo con el uso de una biblioteca cliente ROS, como roscpp o rospy, más alante describiremos en detalle el proceso de desarrollo de uno un nodo ros.

3.7.3. Servidor de Parámetros

El servidor de parámetros permite que los datos se almacenen por clave en una ubicación central. Actualmente es parte del Master.

3.7.4. Mensajes

Los nodos se comunican entre sí al pasar mensajes. Un mensaje es simplemente una estructura de datos, que comprende campos tipados. Los tipos primitivos estándar (entero, punto flotante, booleano, etc.) son compatibles, al igual que las matrices de tipos primitivos. Los mensajes pueden incluir estructuras y matrices anidadas arbitrariamente (al igual que las estructuras C).

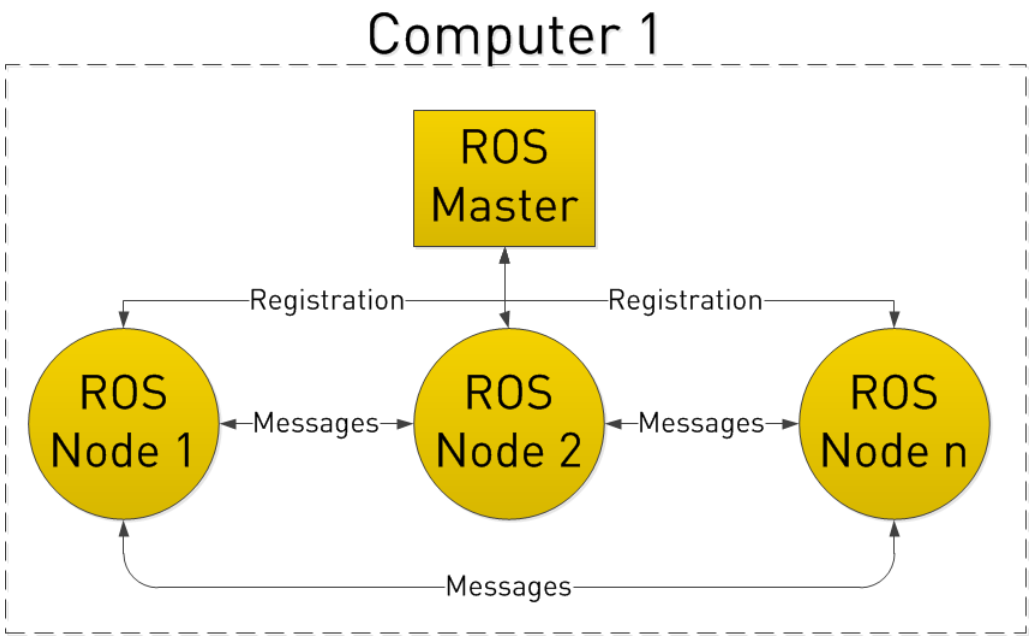

Figura 3.3: Arquitectura básica ROS maestro-nodo

3.7.5. Topics o Temas

Los mensajes se enrutan a través de un sistema de transporte con semántica de publicación suscripción. Un nodo envía un mensaje publicándolo en un tema determinado. El tema es un nombre que se usa para identificar el contenido del mensaje. Un nodo que está interesado en cierto tipo de datos se suscribirá al tema apropiado. Puede haber varios editores y suscriptores

simultáneos para un único tema, y un solo nodo puede publicar y / o suscribirse a múltiples temas. En general, los editores y los suscriptores no conocen la existencia de los demás. La idea es desacoplar la producción de información de su consumo. Lógicamente, uno puede pensar en un tema como un bus de mensajes fuertemente tipado. Cada bus tiene un nombre, y cualquiera puede conectarse al bus para enviar o recibir mensajes, siempre que sean del tipo correcto.

3.7.6. Servicios

El modelo de publicación / suscripción es un paradigma de comunicación muy flexible, pero su transporte unidireccional de muchos a muchos no es apropiado para las interacciones de solicitud / respuesta, que a menudo se requieren en un sistema distribuido. La solicitud / respuesta se realiza a través de servicios, que están definidos por un par de estructuras de mensaje: una para la solicitud y otra para la respuesta. Un nodo proveedor ofrece un servicio con un nombre y un cliente usa el servicio enviando el mensaje de solicitud y esperando la respuesta. Las bibliotecas cliente de ROS generalmente presentan esta interacción al programador como si fuera una llamada de procedimiento remoto.

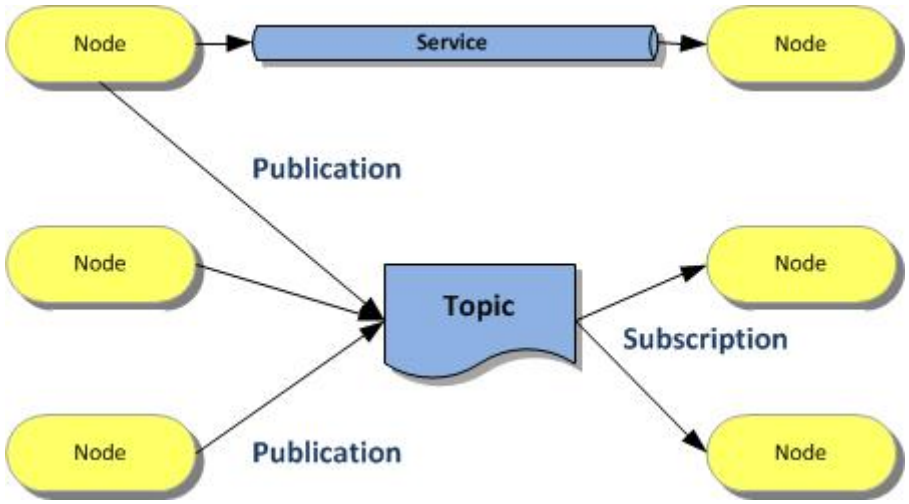

Figura 3.4: Arquitectura básica ROS publicación-suscripción

Capítulo 4

Desarrollo

4.1. Diseño

Diseño QUE BUSCAMOS La finalidad de nuestra aplicación es poder generar scripts completamente funcionales, que sean capaz de manejar con éxito un robot de características complejas, a partir de un proyecto del lenguaje de programación visual Scratch 2.0

DIVISION DEL PROBLEMA

PIEZAS PRINCIPALES

–extensión en scratch

–traducción de scratch

–aplicación sobre el robot real

ESQUEMA DE LAS PIEZAS EN CONJUNTO

EXPLICACION DEL FUNCIONAMIENTO EN CONJUNTO

–esto se comunica con lo otro, traduce de esta forma(cosas del script de traducción), el python traducido se ejecuta, estableciendo la comunicación con el simuladior o el robot real de tal forma.

4.2. kurt

Kurt es una biblioteca de Python que permite la manipulación compleja de proyectos Scratch (archivos .sb o .sb2) a través de simples comandos de Python. Incluye un decompilador, que permite que un proyecto se cargue en un conjunto de objetos de Python, y un compilador que

permite el empaquetamiento de un conjunto de scripts de imágenes / texto en proyectos scratch.

Vamos a estudiar más a fondo como trabaja con esta librería.

4.2.1. Traduccion con kurt

La clase principal de kurt almacena el contenido de un archivo de proyecto Scratch. Los contenidos incluyen variables globales y listas, el escenario y los sprites, cada uno con sus propios scripts, sonidos, variables y listas.

```
# load the scratch project
p = kurt.Project.load(open_path + sys.argv[1])

# show the blocks included
for scriptable in p.sprites + [p.stage]:
        for script in scriptable.scripts:
                # exclude definition scripts
                if "define" not in script.blocks[0].stringify():
                        s = script
print("Stringify:")
sentences = []
for b in s.blocks:
        print(b.stringify())
```

traduccion con kurt

4.3. Desarrollo de bloques

Scratch facilita el uso de bloques personalizados mediante extensiones externas, estas extensiones externas a la aplicación se definen mediante el uso de ficheros JSON, aunque por convención, en Scratch tentrán extensión .s2e .

Este tipo de ficheros se creó para la comunicación mediante HTTP de bloques con aplicaciones auxiliares, por ejemplo algún tipo de hardware. Un *AppHelper* se ejecuta en segundo plano, lista para ser utilizada por los proyectos de Scratch que usan esa extensión. Cada extensión tiene un número de puerto único. Scratch busca la aplicación de ayuda en el número de puerto dado en la máquina local. Se comunica con el *AppHelper* utilizando el protocolo HTTP, la aplicación envía comandos al *AppHelper* y este envía los valores del sensor y la información de estado a Scratch a través de solicitudes HTTP GET. Dado que el protocolo es HTTP estándar, cualquier navegador se puede usar para probar y depurar aplicaciones de ayuda.

Pero nostros no vamos a utilzar esta funcionalidad, únicamente nos ayudamos de éste documento .s2e para definir nuestra extensión y pueda ser usada desde el IDE offline de scratch.

El objeto JSON en el archivo de descripción de la extensión incluye el nombre de la extensión, el puerto TCP / IP utilizado para comunicarse con el *AppHelper* y una lista de bloques de Scratch.

```
{
  "extensionName": "Extension Example",
  "extensionPort": 12345,
  "blockSpecs": [
    [" ", "beep", "playBeep"],
        [" ", "set beep volume to %n", "setVolume", 5],
        ["r", "beep volume", "volume"],
  ]
}
```

El nombre de esta extensión es .[Ej]emplo de extensiónz se conecta a su aplicación auxiliar en el puerto 12345. El El campo "blockSpecs"describe los bloques de extensión que aparecerán en el apartado "Más bloques.[en] la aplicación de Scratch. En en este caso, hay tres bloques: (1) un bloque de comandos que reproduce un pitido; (2) un bloque de comando que establece el volumen del pitido; y (3) un bloque que devuelve un valor, que informa del volumen de un pitido.

Cada bloque se describe mediante una matriz con los siguientes campos:

- **Tipo de bloque**:
  - ’ ’ - bloque de comandos
  - ’w’ - bloque de comandos que esperan
  - ’r’ - bloque que retorna un valor
  - ’b’ - bloque que retorna un booleano
- **Formato de bloque**

  El formato de bloque es una cadena que describe las etiquetas y ranuras de parámetros que aparecen en el bloque. Las ranuras de parámetros están indicadas por una palabra que comienza con ’ %’ y puede ser una de:
  - %n - parámetro de número

- %s - parámetro de cadena
- %b - parámetro booleano

- **Operación o nombre de variable remota**: El campo de operación en una especificación de bloque se usa de dos maneras. Para bloques de comandos, se envía a la aplicación auxiliar, junto con cualquier valor de parámetro, para invocar una operación. O para retornar bloques, es el nombre de una variable de sensor. Los valores de la variable del sensor se guardan en un diccionario. La ejecución de un bloque simplemente devuelve el valor reportado más recientemente para esa variable de sensor.

- **(opcional) cero o más valores de parámetros predeterminados**

- **Menus desplegables**: Los bloques que definimos pueden hacer uso de parámetros de menú, los cuales definiremos de dos formas:
  - %m.menuName - parámetro de menú (no editable), proporciona un sencillo espacio para los parámetros del menú desplegable.
  - %d.menuName - parámetro de número editable con menú, proporciona una ranura de parámetro numérico con un menú auxiliar.

Con todo esto podemos entender la definición de alguno de nuestros bloques como podemos en el siguiente extracto de código.

```
{
  "extensionName": "Scratch4Robots",
  "extensionPort": 12345,
  "blockSpecs": [
    ["", "stop robot—drone", "stop"],
    ["", "move robot %m.robotDirections speed %n", "robot/move/speed", "forward", 1],
    ["r", "color detection %m.color", "camera/all","red"],
    ["r", "frontal laser distance", "laser/frontal"],
  ],
  "menus": {
    "robotDirections": ["forward", "back"],
    "color": ["red", "blue"]
  }
}
```

4.3.1. Bloques genericos

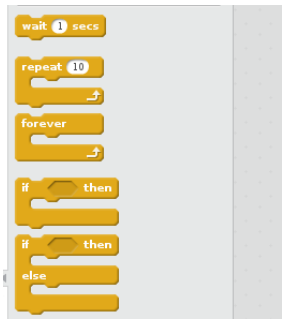

Figura 4.1: Bloques de control

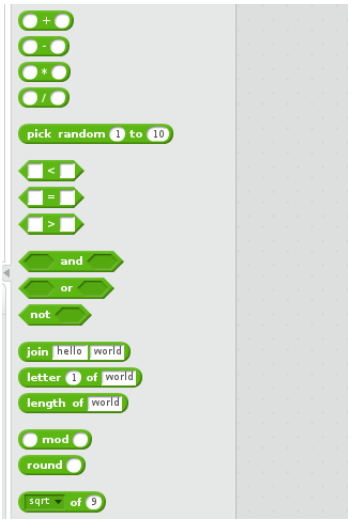

Figura 4.2: Bloques matemáticos

Figura 4.3: Bloques de listas

- **Bloques de operaciones matemáticas**
  - **sqrt of** (): Realiza la operación raiz cuadrada de un número dado
  - **sin of** (): Realiza la operación seno de un número dado
  - **cos of** (): Realiza la operación coseno de un número dado
  - **tan of** (): Realiza la operación tangente de un número dado
  - **asin of** (): Realiza la operación arcoseno de un número dado
  - **acos of** (): Realiza la operación arcocoseno de un número dado
  - **atan of**(): Realiza la operación arcotangente de un número dado
  - **log of** (): Realiza la operación logaritmo de un número dado
  - **ln of** (): Realiza la operación logaritno neperiano de un número dado
  - **abs of** (): Devuelve el valor absoluto de un número
  - **mod of** (): Devuelve el módulo de un número dado
- **Bloques de operaciones lógicas**
  - **And**

- **Or**
- **NOT**
- **Mayor que**
- **Menor que**

- **Bloques de control**

  - **Wait () secs**: Pausa la ejecución el tiempo especificado, equivalente a la sentencia *time.sleep()* de python.
  - **Forever**: Bucle infinito, equivalente a *while(True)* en lenguaje python.
  - **If () then**: Comprueba la condición para que si la condición es verdadera, los bloques dentro de ella se ejecuten.
  - **If () Then, Else**: Comprueba la condición para que si la condición es verdadera, los bloques dentro de la primera condición se activen y si la condición es falsa, los bloques dentro de la segunda condición se activarán.
  - **Repeat ()**: Un ciclo que repite la cantidad de veces especificada, sería la equivalencia al bucle *for* en python.

- **Miscelaneos**

  - **say ()**: Imprime lo que le añadas como argumento, equivalente *print*
  - **Set () to ()**: Utilizado para dar valor a una variable en concreto

- **Bloques de listas**

  - **Insert () at () of ()**: Inserta elemento en la posición seleccionada de la lista indicada.
  - **Item () of ()**: Devuelve el elemento almacenado en la posición indicada de la lista.
  - **Add () to ()**: Inserta en la listaInsert.
  - **Delete () of ()**: Elimina el elemento en una posición determinada de la lista.

4.3.2. Bloques de drone

- **Bloques perceptivos**
  - **Pose3D**: Obtiene el valor de la posición 3D del robot
  - **Color detection**: Haciendo uso de una cámara en el robot, detecta objetos de un determinado color, devolviendo su posición en la imagen.
- **Bloques de movimiento**
  - **stop robot**: Pone a su valor inicial todas las velocidades del robot
  - **drone take off** Makes the drone take off
  - **drone land** Makes the drone land
  - **drone move direction**: (forward, back, up, down, left, right), speed: velocity (integer) Gives the drone a speed in the indicated direction
  - **drone turn direction**: (left, right), speed: velocity (integer) Gives the drone a turning speed in the indicated direction

4.3.3. robots con ruedas

- **Bloques perceptivos**
  - **Pose3D** :Obtiene el valor de la posición 3D del robot.
  - **Color detection**: Haciendo uso de una cámara en el robot, detecta objetos de un determinado color, devolviendo su posición en la imagen.
  - **Frontal distance**: Obtiene la medida promedio de los datos del láser frontal.
- **Bloques de movimiento**
  - **robot move ()**: Da una velocidad en la dirección indicada
  - **robot turn** () direction: (left, right); speed: (integer) Gives a turning speed in the indicated direction
  - **robot move () to position ()** direction: (forward, back, left, right); meter: (integer) Move robot the indicated meters in one direction

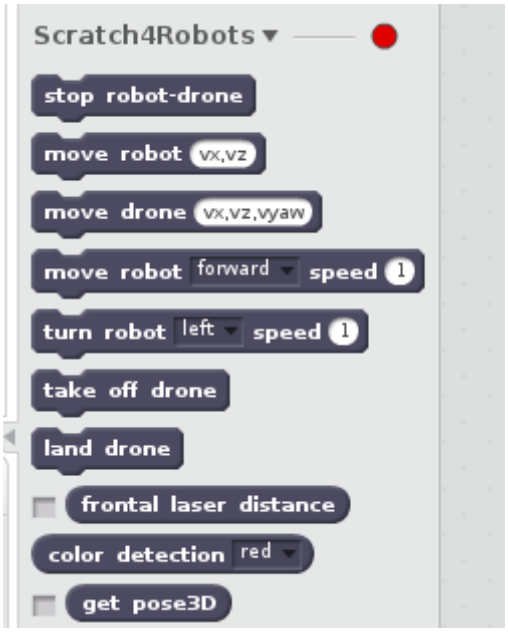

Figura 4.4: Bloques propios de la extensión Scratch4Robots

EXPLICAR LÓGICA DE LOS BLOQUES EN PYTHON

– explicar los perceptivos

– explicar el problema con los de movimiento y las listas, solución aportada

Capítulo 5

Integración

integración Fases:

se recorre un gran camino desde la idea inicial de la herramienta hasta su ejecución final, tanto en tiempo como en desarrollos.

Vamos a desglosar los diferenetes tramos por los que hemos pasado hasta cerrar una versión standalone y estable de la herramienta. 0-Estado inicial Partimos de una versión parcialmente independiente de la herramienta ya propuesta por Raúl Perula desarrollada para Gsoc[enlace git-bibliografia]. Como ya hemos hablado anteriormente, podría dividirse la herramienta en dos módulos, el módulo de traducción y generación de código, y la aplicación de este código sobre robótica. En esta fase nos encontramos con la parte de traducción y generación funcionando de forma relativamente aislada del resto, su ejecución y funcionamiento es totalmente aislada a cualquier dependencia externa ya que se empaqueta en un paquete ROS, con todo lo necesario para su ejecución. La dependencia la encontramos a la hora de ejecutar el código generado sobre el robot real o simulado. Este código está fuertemente acoplado a la plataforma JdeRobot, necesitando de su instalación completa ya que usa muchas de sus librerías.

5.1. Integración con JdeRobot

En esta iteracion buscamos la total integracion de la herramienta en JdeRobot. Trabajando directamente con el equipo de desarrollo de JdeRobot agregamos las librerías externas al framework e introducimos la herramienta.

5.2. Dependencias de la herramienta

Dependencias

5.3. Creación de paquete ROS Independiente

Una vez introducida en la suit de programación buscamos todas y cada una de las dependencias, se refactoriza el código para depender de las menos librerías posibles de JdeRobot y facilitar la futura migración de la herramienta a una versión complatemente independiente.

–EXPLICAR COMO FUNCIONA UN PAQUETE ROS

–SUBID A REPOS OFICIALES PASANDO CALIDAD DE CÓDIGO

–CREACIÓN DE PAQUETES PIP

— QUE ES UN PAQUETE PIP

5.4. Documentación de la herramienta

–documentacion en ros

–documentacion en git

–documenatacion en pagina d jderobot

–apoyo de videotutoriales

Capítulo 6

Experimentos

6.1. Persecución entre drones

persecucion

6.2. Sigue lineas con Kobuki

siguelineas

6.3. Evitar obstáculos con Kobuki

evitar obstaculos

Capítulo 7

Conclusiones

7.1. Conclusiones

conclusiones

7.2. Trabajos Futuros

trabajos futuros