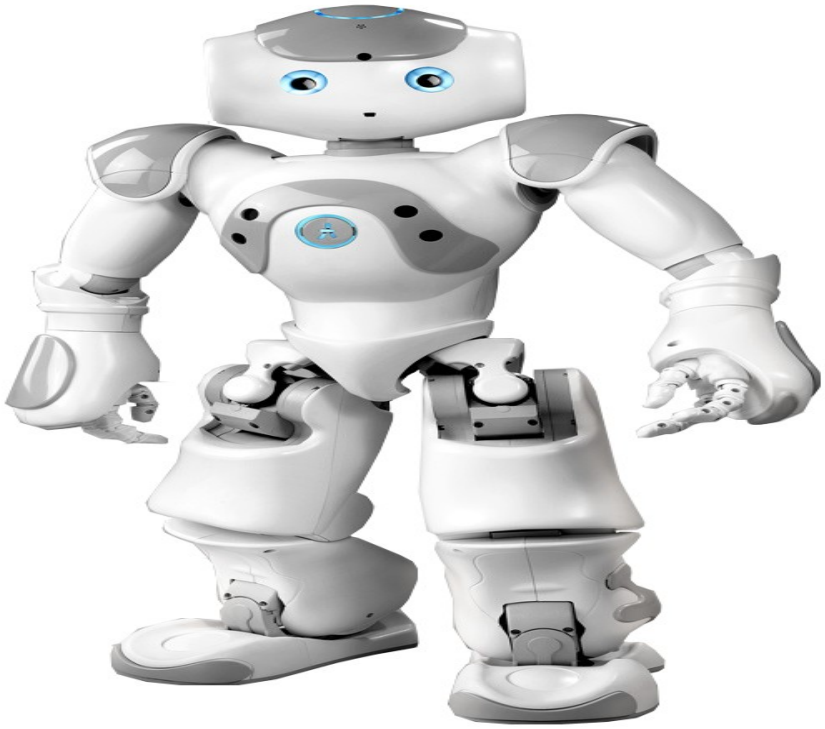
UNIVERSIDAD AUTONOMA DE SAN LUIS POTOSI
Facultad de Ingeniería



Reporte:

Resolución de laberinto y detección de LandMark con Robot Nao utilizando Python SDK en Linux

ALAN IGNACIO SEPULVEDA RODRIGUEZ
alan.ignacio300@gmail.com

Índice de contenido

1. Introducción

Con el paso del tiempo las tecnologías mecánica y electrónica han evolucionado y dejado a un lado los estereotipos de los robots como máquinas que acaban descontrolándose y tomando iniciativa propia, poco a poco se han ido implantando en la sociedad como herramientas que facilitan tareas o resuelven problemas. Esta tecnología ha llegado incluso al campo de la educación, donde se pueden encontrar juguetes que realmente son auténticos robots en miniatura y que ofrecen tal versatilidad que se hace uso de ellos para llevar acabo investigaciones sobre problemas que podrían abordar más tarde robots más grandes.

La automatización para que estos robots realicen tareas en diferentes entornos con los que tienen que interactuar mediante la integración de visión artificial, reconocimiento de formas, planificación de trayectorias atravez de sonares, aún supone un reto en algunos aspectos y ocupa una gran parte de la línea de investigación dentro de la robótica.

Este proyecto engloba el uso de las nuevas tecnologías mecánicas y electrónicas que conforman los robots Nao, con el uso de tecnologías como los sonares ultrasónicos y la visión artificial, con el estudio de la automatización de seguimiento y generación de trayectorias de atravez del sonar para el movimiento autónomo y resolución de laberintos.

2. Estado de la cuestión

2.1 Robot humanoide Nao

El robot NAO es un robot humanoide creado por Aldebaran Robotics, una empresa francesa con sede en París que nació en el año 2005. Aldebaran Robotics fue fundada con la idea de crear robots humanoides que pudieran asistir a las personas. NAO fue creado con el objetivo de ofrecer una plataforma hardware y software que permitiera un avance en las investigaciones en este ámbito, a un precio razonable.

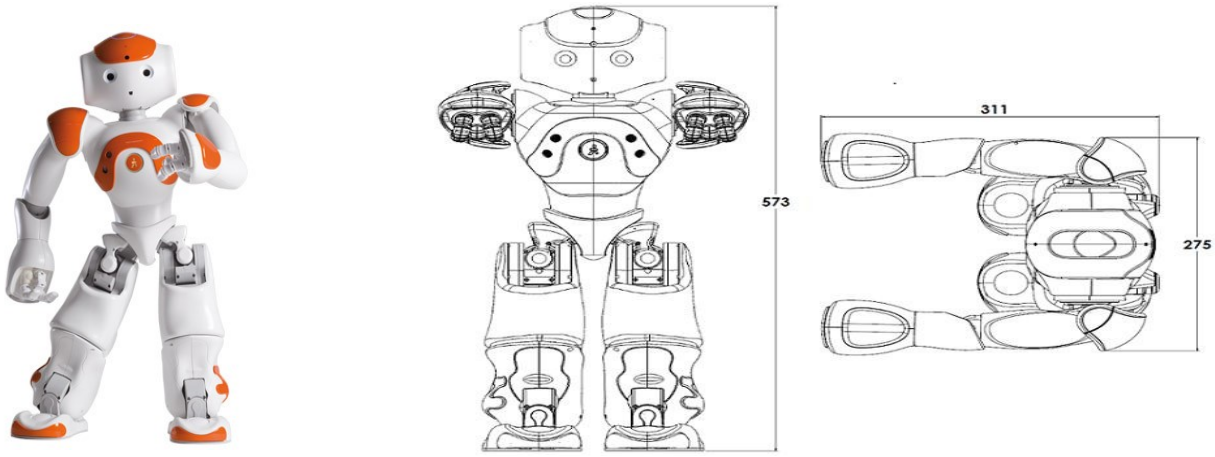


Figura 1.1 Robot Nao.

Nao mide aproximadamente 58 cm de altura y pesa unos 4,8 Kg. Dispone de una batería de ion de litio que le permite una autonomía de unos 90 minutos y funciona con un procesador "x86 AMD GEODE 500MHz CPU" con 256 MB de memoria SDRAM y 2 GB de memoria flash. Cuenta con 26 articulaciones, que se distribuyen de la siguiente manera:

- 2 grados de libertad en la cabeza.
- 5 grados de libertad en cada brazo.
- 2 grados de libertad en cada mano.
- 5 grados de libertad en cada pierna.
- 1 grados de libertad en la pelvis.

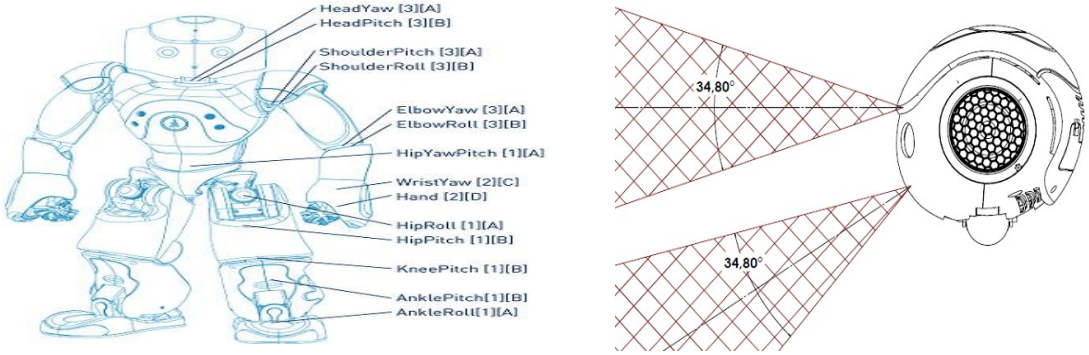


Figura 1.2: (a) articulaciones en detalle del nao y (b) posición de las cámaras

Cuenta con 4 micrófonos en la cabeza, uno a cada lado, uno en la parte de adelante y otro en la parte de atrás; y también dos altavoces uno a cada lado de la cabeza. Para la visión posee dos cámaras una que le permite mirar hacia el frente y otra para ver la parte del mundo que tiene más cercana y que esta inclinada hacia abajo, como se puede ver en la Figura 1.2 Presenta varios sensores de presión en todo el cuerpo: un botón en el pecho, 2 botones tipo bumper en cada pie, 3 sensores táctiles en la cabeza y otros 3 sensores táctiles en cada mano. Cuenta con 8 FSRs (force-sensing resistor), sensores que miden los cambios de resistencia debidos a la fuerza ejercida en un punto, que se encuentran en los pies y son utilizados para que el robot mantenga el equilibrio. Además tiene 2 girómetros de un eje, un acelerómetro de 3 ejes y 2 sonares.

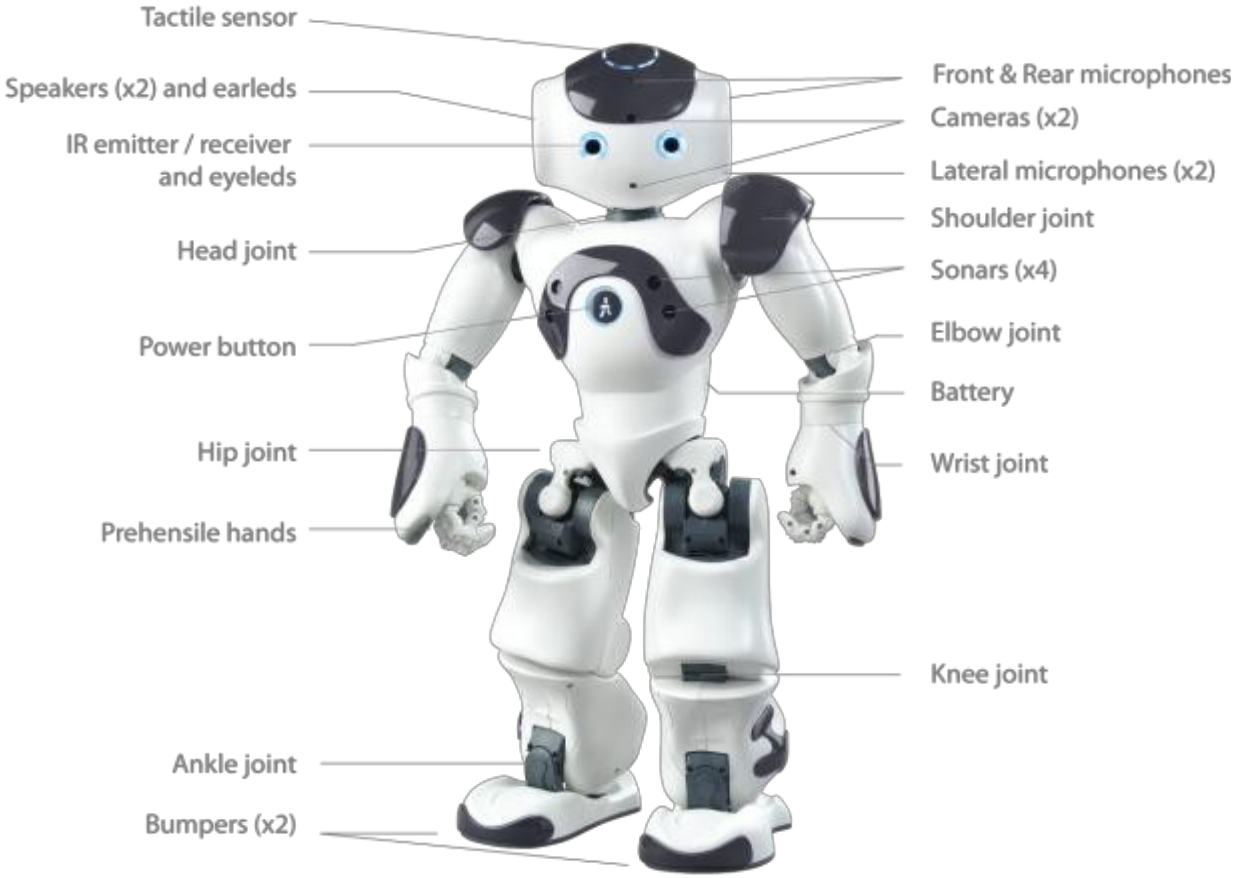


Figura 1.3 Resumen de componentes de Robot Nao

Con estas características el 15 de agosto de 2007, NAO sustituyó al perro Aibot de Sony como robot que competiría en la RoboCup (Robot Soccer World Cup), una competición internacional en la que los robots juegan al fútbol. Desde su nacimiento son muchas las aplicaciones que distintas entidades han desarrollado para el robot NAO: aparte de jugar al futbol, es capaz de reconocer objetos, caras o voces; colaborar con otros robots NAO para cargar objetos; obedecer ordenes; escribir; realizar coreografías en grupo; tocar un xilófono; ayudar en la cocina, y otras muchas cosas. Como se puede ver, a pesar de sus limitaciones, las posibilidades del robot NO son muy grandes.

2.2 Naoqi Framework

El principal componente software del robot NAO es un software embebido llamado NAOqi. Este software proporciona un framework rapido, seguro, confiable, multiplataforma y distribuido, que permite mejorar las funcionalidades del robot. Además, proporciona una comunicación homogénea entre los diferentes módulos ofrece una programacion que se puede realizar en diferentes sistemas operativos (Windows, Mac OS o Linux) y sobre diferentes lenguajes de programación (C++, Python, Urbi o .Net) y También ofrece una introspección, lo que significa que el marco sabe qué funciones están disponibles en los diferentes módulos y donde.

Esta librería define 6 principales módulos que permiten la interacción con los elementos hardware del robot:

- **Core:** Los módulos core o módulos del núcleo se encargan de las funciones primordiales del funcionamiento del robot tales funciones son:

  - Iniciar y detener comportamientos.
  - Gestionar conexión a una red y su configuración.
  - Obtener e insertar datos para cualquier otro equipo.
  - Crear módulos.
  - Leer y guardar la configuración de los archivos de configuración.
  - Manejar los recursos.

- **Motion:** proporciona los métodos que facilitan la realización de movimientos del robot. Además, implementa mecanismos de seguridad en el movimiento, como es manejar las caídas o evitar colisiones.

  Contiene cuatro grupos principales de métodos:

  - La rigidez de las articulaciones (básicamente motor On-Off).
  - La posición conjunta (interpolación, control reactivo).
  - caminar (distancia y control de la velocidad, la posición mundial y así sucesivamente).
  - El efector robot en el espacio cartesiano (cinemática inversa, las restricciones de todo el cuerpo).

- **Audio:** Contiene los elementos software relacionados con el audio del robot.
  Sus principales funciones son:

- Gestionar las entradas y salidas de audio.
- Reproducir archivos de audio en el robot.
- Grabar archivos de audio en el robot.
- Detectar eventos de sonido.
- Localizar los sonidos detectados
- Hacer que el robot entiende lo que dice un ser humano.
- Hacer hablar al robot.

- **Vision:** Contiene los elementos software relacionados con la visión del robot.

- **Sensors:** Contiene los elementos software que sirven para interactuar con los sensores del robot.

  Los modulos de Sensors se dividen en:

  Alto nivel:

  - Parachoques, manos tactiles, cabeza tactil y boton en el pecho.
  - Bateria.
  - Infrarojo.
  - Sonar.
  - Posturas del robot.

  Bajo nivel:

  - Luces led del robot,

- **Trackers:** Los módulos de Tracker le permiten realizar los objetivos de la pista NAO (una bola roja o una cara). El principal objetivo de estos módulos es el de establecer un puente entre la detección de objetivos y el movimiento con el fin de hacer NAO tener en cuenta el objetivo en el medio de la cámara.

  Trae por default 2 trackers:

- ALRedBallDetection.
- ALFaceDetection.

- **DCM:** Significa "Administrador de dispositivos de comunicación".
  - El **DCM** es el módulo de software, parte del sistema **Naoqi**, que está a cargo de la comunicación con todos los dispositivos electrónicos en el robot (tableros, sensores, actuadores ...) excepto el sonido (dentro o fuera) y la cámara. Gestiona la principal línea de comunicación: la conexión USB con el **ChestBoard**. Pero también hay un enlace I2C con los dispositivos de la cabeza del robot. Por lo tanto, el **DCM** es el enlace entre el software "nivel superior" (módulos de otros) y el software de "nivel inferior" (suave en las placas electrónicas).

2.2.1 Características de Naoqi

2.2.1.1 Lenguaje cruzado

El software puede ser desarrollado en C++ y Python. En todos los casos, los métodos de programación son exactamente lo mismo, todas las API existentes pueden ser llamadas indistintamente desde cualquiera de los idiomas soportados:
Si se crea un nuevo módulo de C++, las funciones API de C++ se puede invocar desde cualquier lugar, si se ha definido correctamente La mayoría de las veces se desarrollan sus comportamientos en Python y sus servicios en C++.

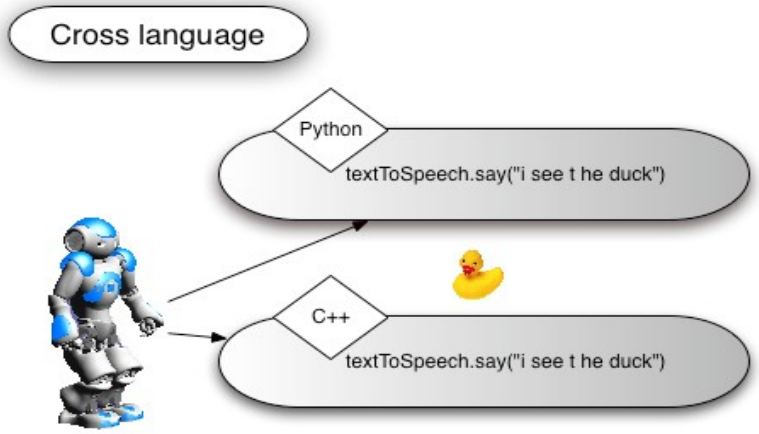


Figura 1.4Lenguaje cruzado

2.2.1.2 Introspección

La introspección es el fundamento de las API del robot, las capacidades, el seguimiento y la acción sobre las funciones monitoreadas. El robot sabe todas las funciones API que tiene disponibles.

La descarga de una biblioteca eliminará automáticamente las funciones de la API correspondientes. Una función definida en un módulo se puede añadir en la API con un BIND_METHOD (definido en almodule.h).

El API se muestra en un navegador web. Sólo se tiene que escribir en la url, el robot y el puerto 9559. por ejemplo: http: //127.0.01: 9559. El Robot mostrará sus módulos de la lista, la lista de método, los parámetros del método, descripciones y ejemplos. El navegador también muestra métodos paralelos que pueden ser monitoreados, hicieron que esperar, se detuvo.

2.2.1.3 Àrbol Distribuido y la comunicación

Una aplicación en tiempo real puede ser sólo un ejecutable independiente o árbol de robot, árbol de procesos, árbol de módulos. Sea cual sea su elección, los métodos de llamada son siempre los mismos. Conectar un ejecutable a otro robot con la dirección IP y el puerto, y todos los métodos de la API de otros ejecutables están disponibles en exactamente la misma manera que con un método local. Naoqi hace la elección entre la llamada directa rápida (LPC) y la llamada remota (RPC).

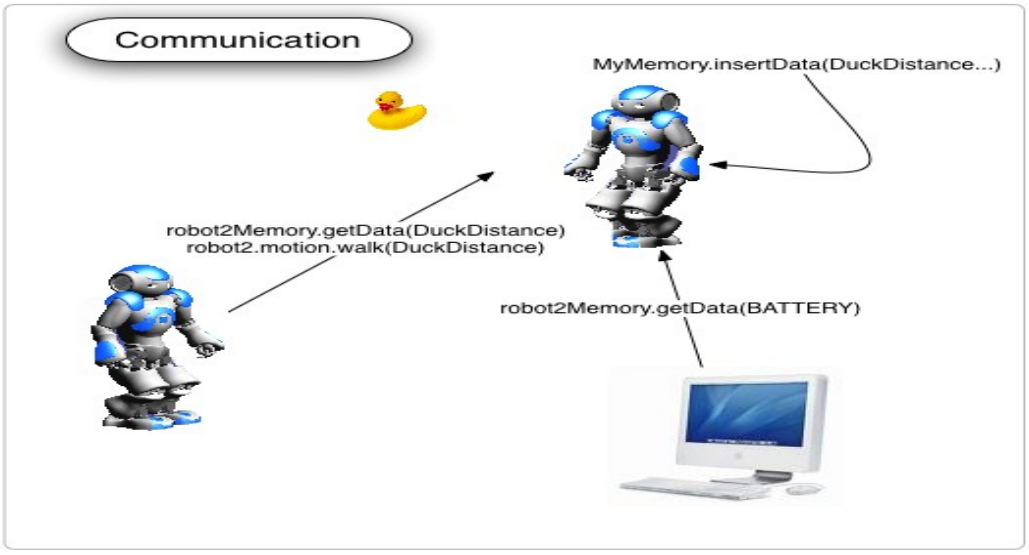


Figura 1.5 Árbol distribuido y comunicación

2.2.2 Métodos de programación de NAO.

Naoqi ofrece tres métodos de programación de llamadas:

- Paralela
- Secuencial
- Por evento

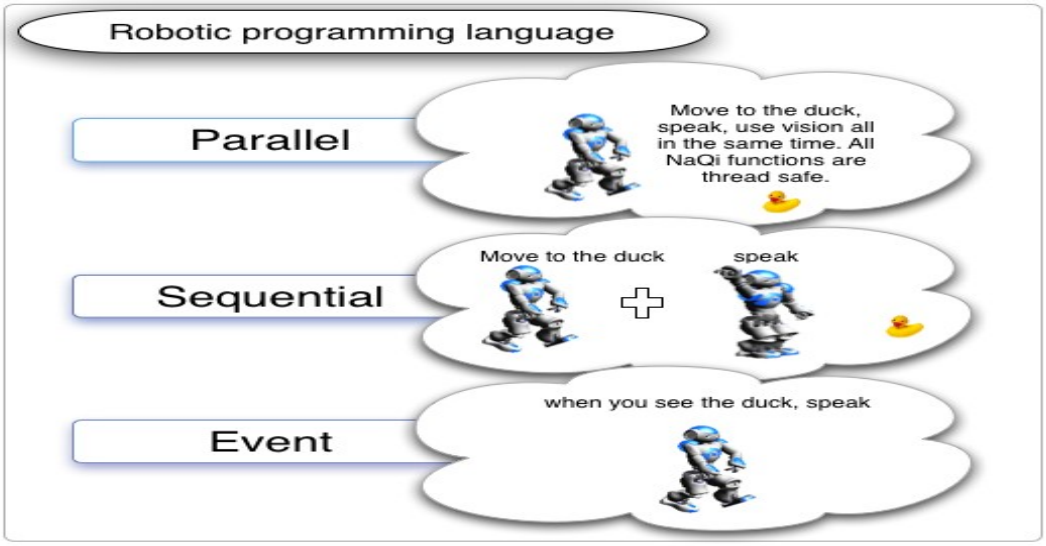


Figura 1.6 Métodos de programación robótica

2.2.3 El proceso de Naoqi

El Naoqi ejecutable que se ejecuta en el robot es un broker. Cuando se inicia, se carga un archivo de preferencias denominado autoload.ini que define qué bibliotecas se deben cargar. Cada biblioteca contiene uno o más módulos que utilizan el agente para anunciar sus métodos.

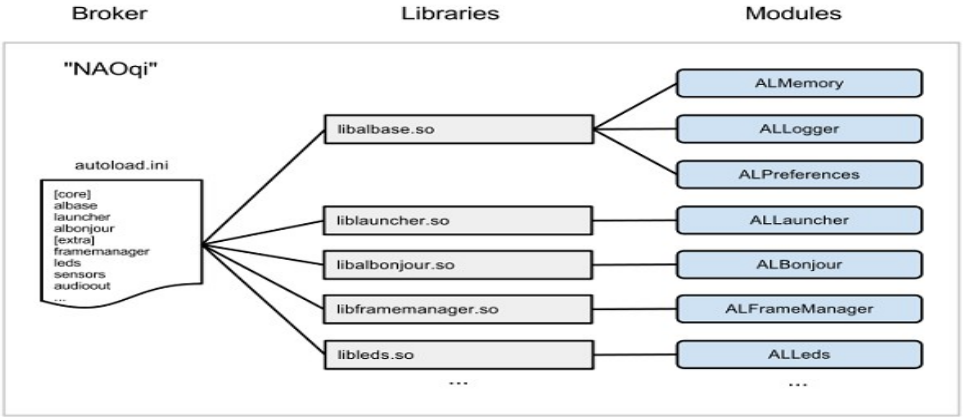


Figura 1.7 El proceso de Naoqi

2.2.3.1 Broker

Un broker es un ejecutable y un servidor que puede atender comandos remotos en un IP (el de la maquina en que esta corriendo) y un puerto, es decir por red. En practica, para implementar una funcionalidad en el robot, tienes que pasar por un broker. Dos casos son posibles:

- o el generador de módulos generar a un ejecutable y este se conectar a al robot (en este caso, este ejecutable es un broker secundario).

- o generar a una librera para cargarla en el broker principal, es decir el programa Naoqi, que es el que provee las funcionalidades básicas en el robot (adquisición de datos por sensores, actuación. . .).

En este ultimo caso, se tiene que agregar en autoload.ini (del robot) el nombre de la librería.
Hay que entender bien las implicaciones de cada una de esas dos elecciones en cuanto al modulo: en el primer caso, el modulo esta encapsulado en un ejecutable diferente del ejecutable que corre las operaciones criticas en el robot. Eso significa que en caso de que haya un gran error en el código, el ejecutable podrá fallar sin que los sistemas de control críticos del robot fallen también. En el segundo caso, el código compilado esta usado por el mismo ejecutable Naoqi : en caso de problemas graves (error de segmentación), el robot puede caer. Sin embargo, en el segundo caso, ya que el código esta en el mismo ejecutable que los modulos-core del robot, se puede acceder mucho mas rápido a datos del robot (por ejemplo, la imagen de la cámara), ya que la memoria es compartida entre módulos del mismo ejecutable.
En resumen, para el modulo, se tienen siempre dos opciones:

1. correrlo en un broker separado (modo “remote”): seguro pero un poco menos eficiente;

2. correrlo en el broker principal (modo “local”): arriesgado pero muy eficiente.

En la figura 1.2 se ilustra un simple ejemplo de un modulo creado por el usuario, llamado “myModule”. myModule corre como un broker separado (modo “remote”) llamado “myBroker”que se comunica con el broker principal sobre 127.0.0.1:9559.

Observacion: Un punto importante que observar es que, en este caso particular, “Main Broker” y sus módulos están corriendo a bordo del robot, igualmente a “myBroker”(ya que la IP especificada corresponde a la maquina huésped). Pero, hubiéramos podido muy bien correr el “myBroker” en una maquina remota, especificando a este programa una IP correspondiendo a la dirección del robot en la red. El desempeño es en este caso limitado por las propiedades de la red, pero permite por ejemplo correr procesamientos tal vez mas pesados, en la maquina local mas poderosa que la CPU del robot.

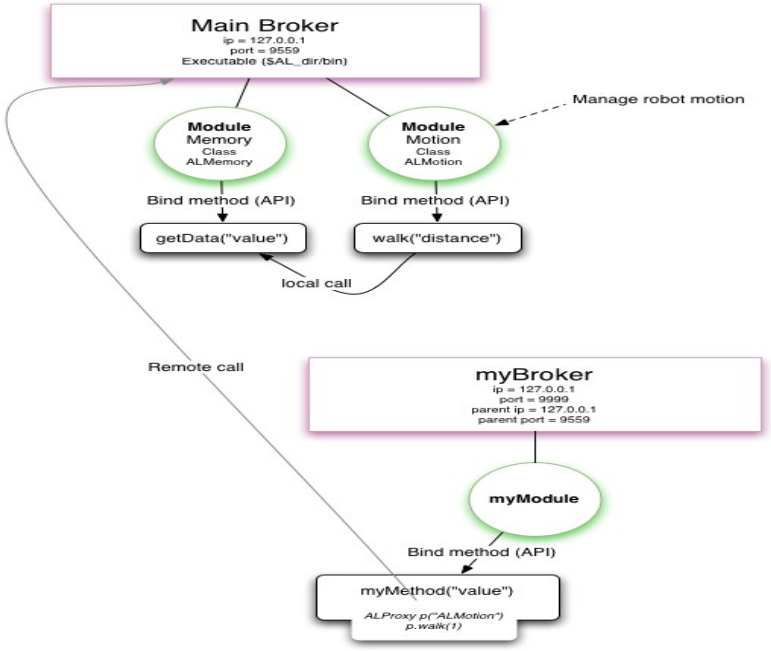


Figura 1.8 Ejemplo de funcionamiento de un broker

2.2.3.2 Proxy

Un proxy es un objeto que se comportará como el módulo que representa.
Por ejemplo, si crea un proxy para el módulo ALMotion, obtendrá un objeto que contiene todos los métodos ALMotion. Para crear un proxy para un módulo, (y por tanto llamar a los métodos de un módulo) se tienen dos opciones:

- Sólo tiene que utilizar el nombre del módulo. En este caso, el código que se está ejecutando y el módulo al que desea conectarse debe estar en el mismo corredor. Esto se llama un local de llamadas.

- Utilice el nombre del módulo, y la IP y el puerto de un proxy. En este caso, el módulo debe estar en el corredor correspondiente.

Ejemplo:

```
from naoqi import ALProxy

def Hablar(robotIP):
        try:
                robotSay = ALProxy("ALTextToSpeech", robotIP, Puerto)
                robotSay.setLanguage("Spanish")
        except Exception, e:
                print ("Error:")
```

```
        print ("?", e)

robotSay.say("Hola Mundo")
```

2.2.3.3 Modulos

Normalmente, cada módulo es una clase dentro de una biblioteca. Cuando la biblioteca se carga desde el autoload.ini , será una instancia de la clase del módulo automáticamente.
En el constructor de una clase que se deriva de ALModule , puede "obligar" métodos. Esto anuncia sus nombres y firmas de los métodos para el proxy, para que estén disponibles para los demás.

Un módulo puede ser local o remoto.

- Si es remoto, se compila como un archivo ejecutable, y se puede ejecutar fuera del robot. Los módulos remotos son más fáciles de usar y se pueden depurar fácilmente desde el exterior, pero son menos eficientes en términos de velocidad y uso de memoria.

- Si es local, se compila como una biblioteca, y sólo se puede utilizar en el robot. Sin embargo, es más eficiente que un módulo remoto.

2.2.3.4 Memoria

ALMemory es la memoria del robot. Todos los módulos se pueden leer o escribir datos, suscribirse a eventos con el fin de ser llamado cuando se plantean los acontecimientos.

Tenga en cuenta que ALMemory no es una herramienta de sincronización en tiempo real. Límite suscribirse en DCM / hora o de movimiento variable de tiempo / sincro o real.

ALMemory es una matriz de ALValue de acceso Variable es seguro para subprocesos. Utilizamos leer / escribir las secciones críticas para evitar el mal rendimiento cuando se lee la memoria.

ALMemory contiene tres tipos de datos y ofrece tres diferentes APIs.

- Principalmente datos de los sensores y las articulaciones
- Evento
- Micro-evento

2.3 Instalación de Python SDK en Linux

1. Descargar e instalar python 2.7

   Puede encontrar el instalador aquí: http://python.org/download/

2. Descargar *Naoqi para Python.* pynaoqi-python-2.7-Naoqi-xx-linux32.tar.gz

   Puede descargar la última versión desde el sitio web Aldebaran Comunidad .

3. Establezca la variable de environnement PYTHONPATH a / ruta / al / python-sdk

   Ejemplo:

```
export PYTHONPATH = ${ PYTHONPATH } : / ruta / ala/ carpeta/pynaoqi
```

3. Analisis y diseño

En este apartado se realiza una descripción del funcionamiento de la aplicación y del proceso de desarrollo que se ha seguido para su implementación. Para ello, tras hablar brevemente de la metodología utilizada, se presentará el análisis y diseño.

3.2. Análisis

A continuación se definirá el proceso de análisis que se ha realizado para construir el algoritmo. El objetivo del análisis es llevar a cabo una especificación profunda y detallada de que es lo que debe realizar el sistema y como lo debe hacer, para servir de base durante el posterior diseño de la aplicación.

3.2.1 Alcance

En este apartado se definirá de manera completa y concisa lo que debe ser capaz de hacer el algoritmo a desarrollar y lo que se debe quedar fuera de su funcionalidad.

Se necesita crear un algoritmo que tenga la capacidad de guiar a un robot NAO dentro de un laberinto evadiendo obstáculos y planificando trayectorias atravez de los sonares, detectando y decodificando LandMarks y encontrar la forma mas rápida de salir del mismo.

3.2.2. Diagrama de flujo

Basándose en las necesidades presentadas anteriormente se creo el siguiente diagrama de flujo donde explica el funcionamiento del algoritmo para guiar al robot atravez del laberinto.

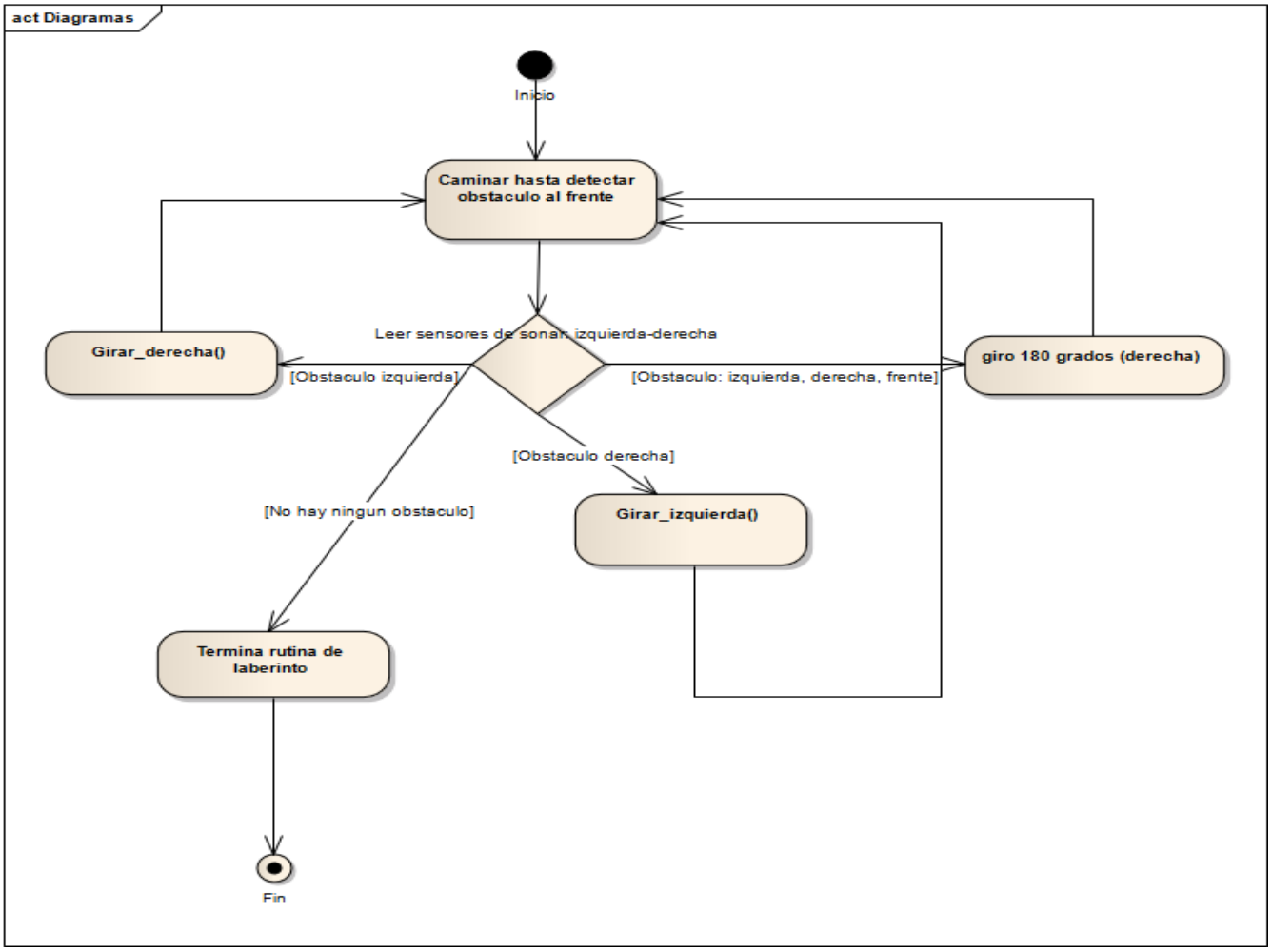


Figura 1.9 diagrama de flujo de algoritmo para resolver laberinto.

3.3. Diseño

Para lograr que el robot logre moverse dentro del laberinto usando la información de los sonares, es necesario conocer los limites maximos y minimos del sonar, tambien es necesario conocer el movimiento que efectuar el robot nao al caminar y la forma en que el robot detecta y codifica las LandMark. A continuación se explicara el proceso de realización de cada unos de los elementos que se tomaron en cuenta para realizar el algoritmo.

Para mayor detalle ver experimento 2.

3.3.1 Caminar

Para caminar se utilizo el modulo **ALNavigation** pues este modulo es un primer intento de hacer que el robot vaya con seguridad a una diferente postura (por ejemplo, ubicación + orientación). El robot todavía no puede evitar los obstáculos, pero es capaz de moverse con cautela, deteniéndose tan pronto como un obstáculo entra en su zona de seguridad.

Mientras se mueve hacia adelante, el robot intenta detectar obstáculos delante de él, utilizando sus parachoques y sonares.

Tan pronto como un obstáculo entra en su área de seguridad, el robot se detiene.

A continuacion se muestra el procedimiento utilizado para caminar.

```
def iniciar(robotIP):

    try:
        motionProxy = ALProxy("ALMotion", robotIP, 9559)
    except Exception, e:
        print "Error al crear  ALMotion"
        print "Error mientras: ", e

    try:
        postureProxy = ALProxy("ALRobotPosture", robotIP, 9559)
    except Exception, e:
        print "No se pudo crear proxy ALRobotPosture"
        print "Error mientras: ", e

    # Enviar la pose a ejecutar
    #----------------------------------------------------------#
    postureProxy.goToPosture("StandInit", 0.4)

def Caminar(robotIp):
            iniciar(robotIp)
            navigationProxy = ALProxy("ALNavigation", robotIp, 9559)

            navigationProxy.moveTo(2.0, 0.0, 0.0)

            navigationProxy.setSecurityDistance(0.3)
```

Donde el procedimiento iniciar() es el que se encarga de establecer la posicion StandInit del robot la cual es primordial tener antes de realizar cualquier movimiento puesto que es la posición que da mayor equilibrio y eficiencia al robot. Y el procedimiento caminar es el que se encarga del movimiento del robot donde el valor de 0.3 de la linea navigationProxy.setSecurityDistance(0.3) indica la distancia maxima en metros que se permitira tener un objeto enfrente del robot para que este se detenga.

3.3.2 Obtener valores de sonar.

Para obtener los valores del sonar es necesario conectarse al modulo sonar a travez de un proxy. El siguiente paso es subscribir los sonares a una aplicacion para iniciar el nivel hardware y empezar la adquisición de datos. Luego de adquirir los datos es necesario obtenerlos de la memoria creando un modulo de memoria a travez de un proxy, indicar la ruta donde se encuentran los valores del sonar indicado y almacenarlo en una variable.

```
def SonarDerecha(robotIP):

# Connectar al modulo Sonar.
        sonarProxy = ALProxy("ALSonar", robotIP, 9559)

#Subscribe los sonares para correr a nivel hardware y ampezar la adquisicion de datos.
        sonarProxy.subscribe("myApplication")

# crear modulo de memoria
        memoryProxy = ALProxy("ALMemory", robotIP, 9559)

# Obtener el valor del sonar derecho (distancia en metros al primer objetivo).
        right = memoryProxy.getData("Device/SubDeviceList/US/Right/Sensor/Value")

        return right
pass
```

Para ver mas detalles sobre la implementacion de los sonares en el algoritmo ver experimento 1 y 2.

3.3.3 Detección de LandMark

Existe una codificación propia de Aldebaran. Son códigos redondos capaces de almacenar nada más que números. A pesar de ser propios para el robot, encontramos que la aplicación de lectura aún no está muy bien implementada y hay muchos problemas a la hora de leer el código desde distintos ángulos puesto que afecta tanto la dirección de la cámara del robot y fenómenos externos como son la luminosidad.

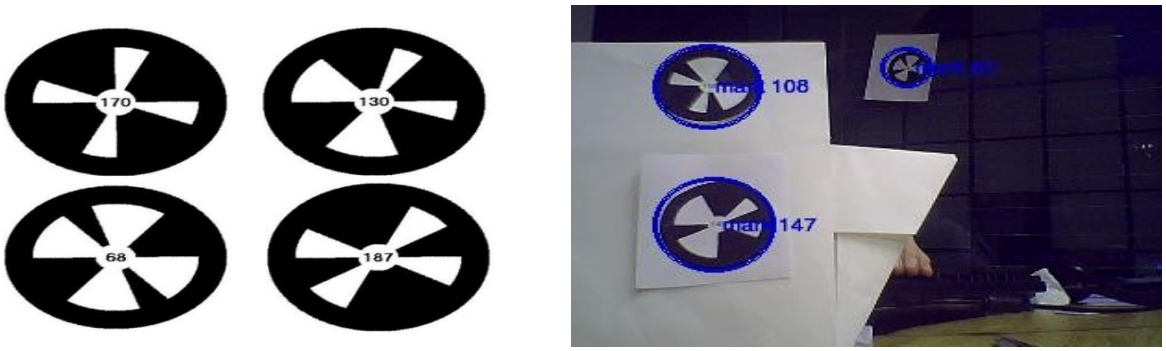


Figura 1.9 LandMarks y ejemplo de detección.

Limitaciones y Funcionamiento

Iluminación : El hito de de detección ha sido probado en condiciones de iluminación de oficinas - es decir, en 100 a 500 lux. Como la propia detección se basa en diferencias de contraste, que en realidad debería comportarse bien mientras las marcas en sus imágenes de entrada están razonablemente bien contrastados. Si usted siente que la detección no está funcionando bien, asegúrese de que la ganancia automática de la cámara se activa - a través de la interfaz de Monitor - o tratar de ajustar manualmente el contraste de la cámara.

Campo de medidas para las marcas detectadas:

Mínimo: ~ 0.035 rad = 2 grados. Corresponde a ~ 14 píxeles QVGA en una imagen

Máximo: ~ 0,40 rad = 23 grados. Corresponde a ~ 160 píxeles QVGA una imagen en

Inclinación: +/- 60 grados (0 ° corresponde a la marca frente a la cámara)

Rotación en plano de la imagen : invariante.

3.3.4 Algoritmo de detección de LanDMarks en laberinto

Este algoritmo esta conformado por los siguientes elementos :

- Un ciclo infinito para obtener los datos en tiempo real.
- Un modulo ALLandMarkDetection, que es donde se encuentran as funciones para detectar las LandMark.
- Un modulo ALTextToSpeech, este es utilizado para que el robot hable y diga el código de las landmark detectadas.
- Un modulo ALMemory, para extraer los valores de las landmark detectadas, puesto que estas al momento de ser encontradas se almacenan en la memoria y necesitamos extraerlas.
- Un procedimiento que se encarga de almacenar las landmark detectadas y almacenarlas en un arreglo para luego comparar la información del arreglo con las landmark detectadas, esto se hace para evitar que el robot diga una LandMark que ya fue detectada anteriormente.

```
def LandMarkDetection(robotIP):
	while 1:
		try:
			landMarkProxy = ALProxy("ALLandMarkDetection", robotIP, 9559)
		except Exception, e:
			print "Error creando ALLandMarkDetection:"
			print str(e)
		try:
			sayProxy = ALProxy("ALTextToSpeech", robotIP, 9559)
			sayProxy.setLanguage("Spanish")
		except Exception, e:
			print "Error creando ALLandMarkDetection:"
			print str(e)

		period = 500
		landMarkProxy.subscribe("Test_LandMark", period, 0.0 )
		memValue = "LandmarkDetected"

		try:
			memoryProxy = ALProxy("ALMemory", robotIP, 9559)
		except Exception, e:
			print "Error creando memory Proxy:"
			print str(e)

		for i in range(0, 20):
			time.sleep(0.1)
			val = memoryProxy.getData(memValue)

		if(val and isinstance(val, list) and len(val) >= 2):

			timeStamp = val[0]
			markInfoArray = val[1]

			try:
				for markInfo in markInfoArray:
					markShapeInfo = markInfo[0]

					markExtraInfo = markInfo[1]
					a = (markExtraInfo[0])
					print a
					b = land.count(a)
					if b == 0:
						land.append(a)
						sayProxy.say("Landmark : %d" % (markExtraInfo[0]))
					else:
						print ("landmark deplicada")

			except Exception, e:
				sayProxy.say(val)
		else:
			print ("")
			#sayProxy.say("landmark no detectadas")
		landMarkProxy.unsubscribe("Test_LandMark")
```

Para ver mas detalles sobre la implementación de la detección de LandMarks en el algoritmo ver experimento 4 y 5.

3.3.5 Estrategia de laberinto

La estrategia que se tomo para el laberinto es bastante sencilla esta compuesta por 2 procesos independientes uno encargado de establecer la trayectoria a seguir tomando en cuenta la información tomada por los sonares y otro proceso encargado de detectar, procesar y decir el código de cada LandMark encontrada.

La estrategia esta compuesta de la siguiente forma:

El robot camina hasta encontrar una pared a 30 cm de distancia. El robot se detiene y lee los datos de los sonares y los procesa de la siguiente manera.

- Si el valor del sonar izquierdo es menor significa que hay un obstáculo ala izquierda, entonces giramos a la derecha y vuelvo a caminar hasta encontrar un nuevo obstáculo.
- Si el valor del sonar derecho es menor significa que hay un obstáculo ala derecha, entonces giramos a la izquierda y vuelvo a caminar hasta encontrar un nuevo obstáculo.
- Si el valor del sonar izquierdo y el valor del sonar derecho son iguales pero estos son diferentes ala longitud máxima del sonar giro 2 veces ala derecha y vuelvo a caminar hasta encontrar un nuevo obstáculo.
- Si el valor del sonar izquierdo y derecho son igual a la longitud máxima alcanzada por los sonares, significa que hemos finalizado el laberinto.

Ver figura 1.9 en la sección de análisis y experimento 3 y 5 pare ver mas detalles.

```
def main(robotIP):
        while 1:
                # primero lo inicio caminado el robot caminara hasta encontrarse con obstaculo a 30 cm de distancia hacia el frente
                Caminar(robotIP)
                #Cuando detecte el obstaculo lo mando ala postura StandInit pra que relie el movimiento hacia los lados con mayor eqilibrio
                iniciar(robotIP)
                #obtengo el valor de los sonares
                l = SonarIzquierda(robotIP)
                r = SonarDerecha(robotIP)
                print l
                print r
                #tomo la desicion hacia cual lado girar de acuerdo a la lectura de los sonares
                if l<r:
                        Giro_derecha(robotIP)
                elif r<l:
                        Giro_izquierda(robotIP)
                elif r == l and l < 2.5 and r < 2.5:
                        Giro_derecha(robotIP)
                        Giro_derecha(robotIP)
                        Caminar(robotIP)
                        iniciar(robotIP)
                elif r == 2.53999996185 and l == 2.53999996185:
```

iniciar(robotIP)
Say(robotIP)
return 1

4. Evaluación

En este apartado se detallará como se llevó a cabo la evaluación del proyecto. Para realizarla, se contó con la participación de 2 usuarios que no tenían ningún tipo de relación con el proyecto y que tuvieron que dar su punto de vista en cuanto a la mejora y optimización de la aplicación efectuando varios experimentos . Estos experimentos consistían en una serie de tareas de dificultad variable que tenían como objetivo poner a prueba las capacidades de la aplicación .
Los objetivos de la evaluación son:

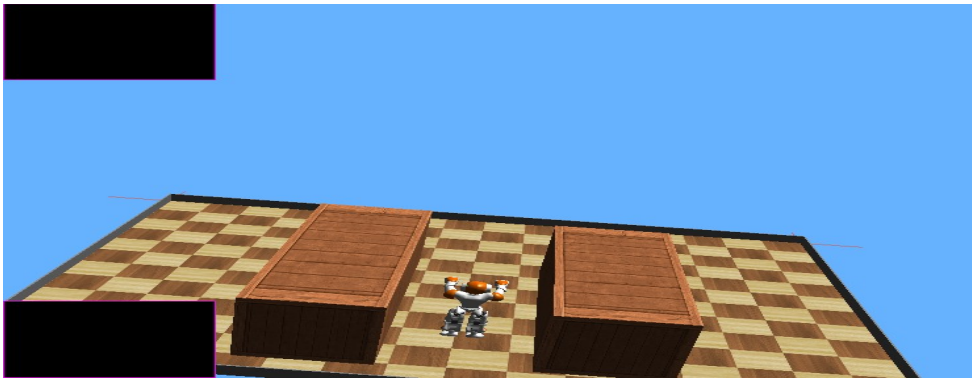
- Comprobar la capacidad de la aplicación para ser utilizado en la realización de laberintos sencillos.
- Observar hasta que punto el sistema no necesita un aprendizaje previo para su manejo.
- Conocer las sensaciones y opiniones que tienen los usuarios durante la utilización de la aplicación.
- Recoger sugerencias de los usuarios para que puedan ser aplicadas en mejora de la aplicación.

4.1 Experimentos

A continuación se describirán los cuatro experimentos que fueron llevados a cabo.

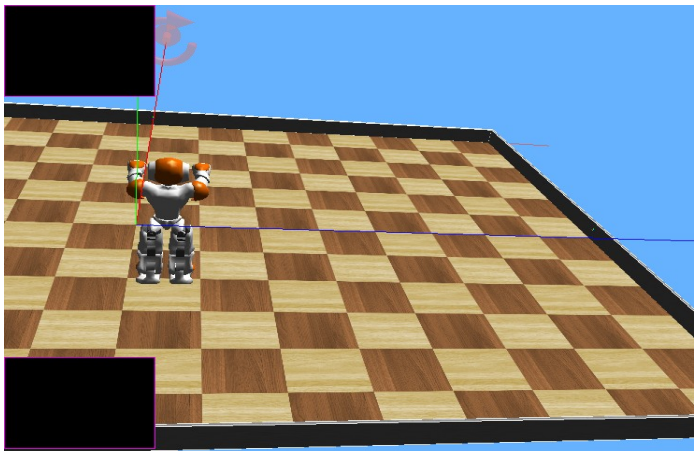
Experimento 1

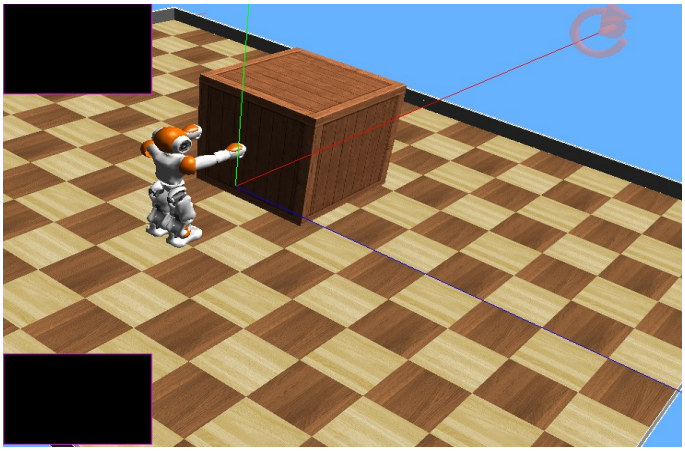
El experimento número 1 consiste en verificar si el movimiento del robot nao al caminar hacia enfrente lo hace en linea recta y por consiguiente se opto por hacer caminar al robot dentro de un túnel para verificar el movimiento del robot nao al caminar.

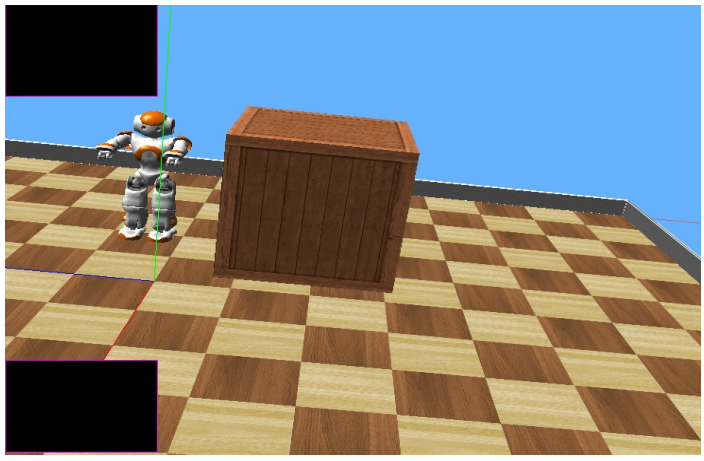


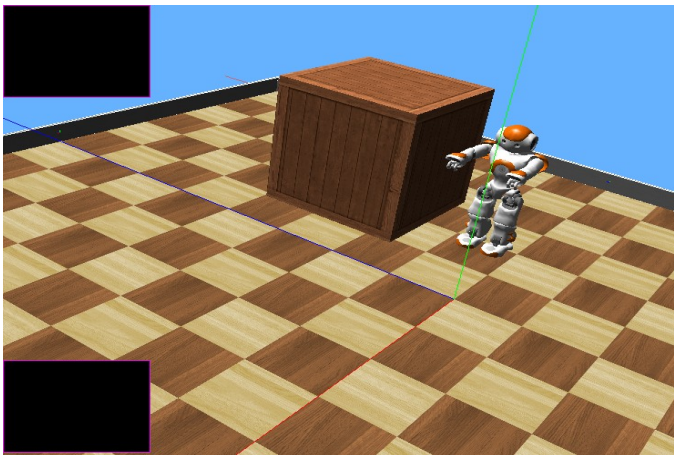
Experimento 2

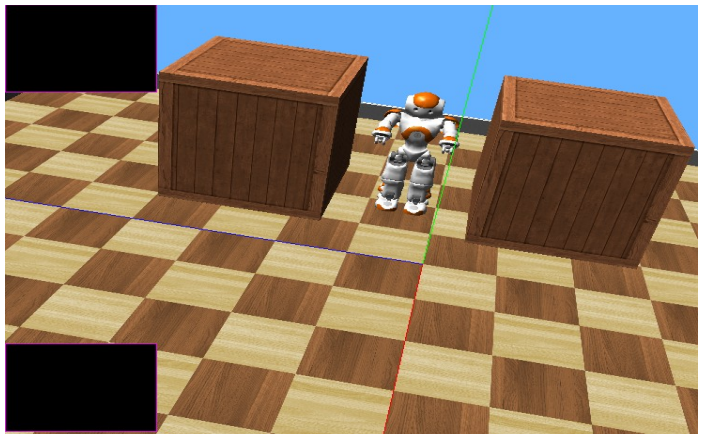
El experimento número 2 consiste en obtener los valores máximos y mínimos del sonar. Como se puede observar, el robot obtiene información de los sonares detectando los objetos que se encuentran enfrente y en sus laterales. Esta información recabada es de mucha utilidad pues encontramos los limites de los sonares y tenemos las bases para guiar al robot dentro de un laberinto contando únicamente con la información que reciba a través de los sonares.

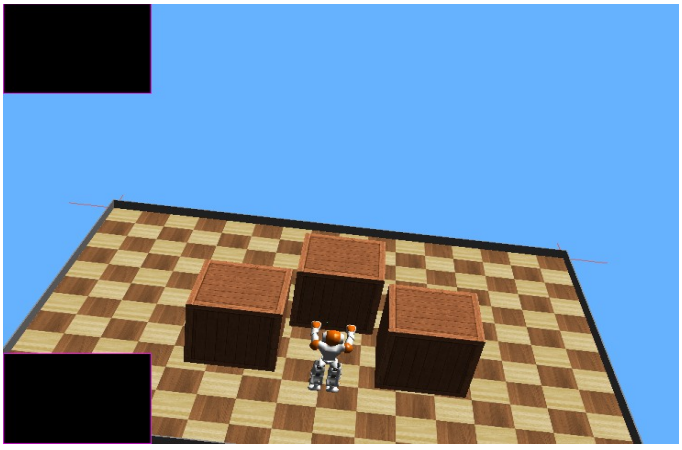












Los resultados fueron los siguientes:

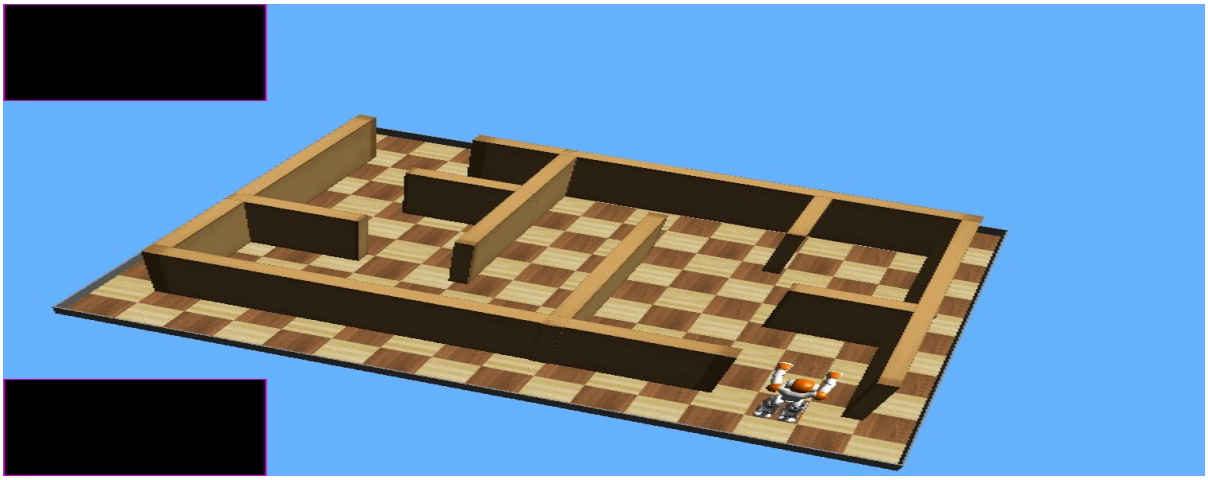
Rango máximo de detección de sonares en metros:

Derecha: 2.5399999618530273
Izquierda: 2.5399999618530273

Para lograr un óptimo desempeño al momento de caminar por el laberinto se ha encontrado que la distancia óptima para que el robot se detenga cuando encuentra un obstáculo enfrente es de 30 cm.

Experimento 3

El experimento número 3 consiste en la resolución de un circuito formado por una serie de obstáculos. La forma que tendrá dicho circuito será la que aparece representada en la siguiente figura. Como se puede observar, el robot tendrá que caminar, y realizar giros. De esta manera irá sorteando los obstáculos hasta llegar a la zona de meta, contando únicamente con la información que reciba a través de los sonares.



Es difícil guiar al robot por medio de sonares ya que en ocasiones el robot no se detiene de manera paralela al ángulo de la pared.

Se realizo la prueba 10 veces de las cuales:

1. El robot salio 5 veces exitosamente del laberinto con un promedio de tiempo de 10 a 15 minutos.

2. El robot salio 3 veces por la linea de entrada.

3. El robot tropezaba y detenía su funcionamiento.

Experimento 4

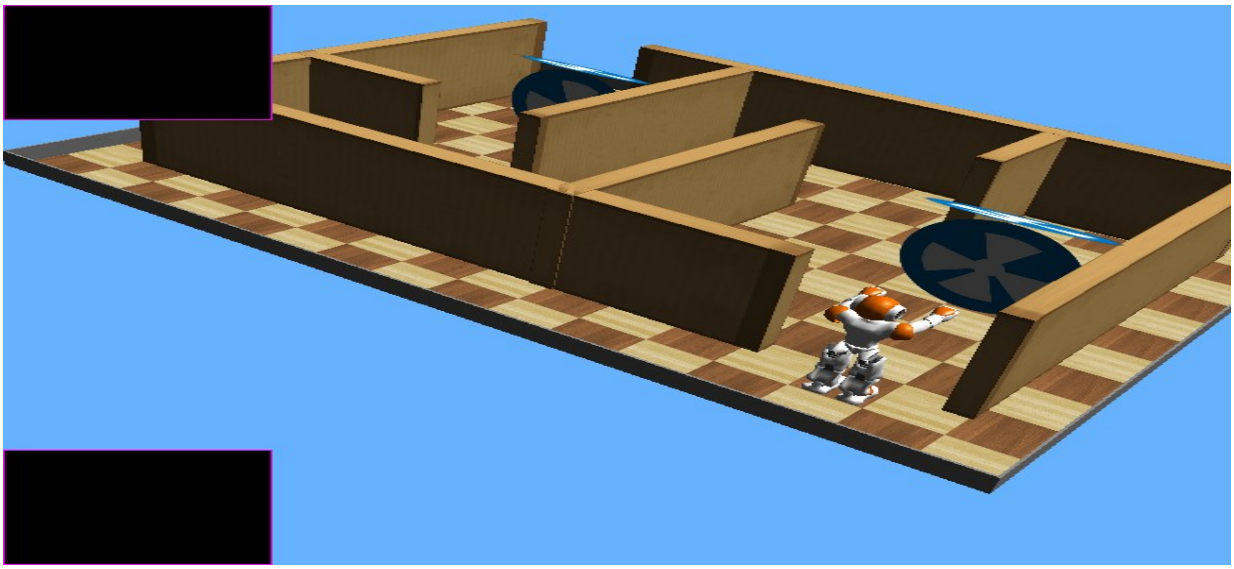
El experimento número 4 consiste en determinar el ángulo y distancia máxima en la que el robot detecta, decodifica y dice el numero que ocultan las LandMark en un espacio abierto.


En base al experimento realizado se pudo obtener la siguiente información.

1. El limite máximo de distancia para la detección de las LandMark es aproximadamente de 2.5 metros.
2. Si se tiene mas de 1 LandMark en un espacio de detección confiable el robot dirá siempre la que se encuentre mas cerca y así sucesivamente.

Experimento 5

El experimento número 5 consiste en la resolución de un circuito formado por una serie de obstáculos y la detección de las LandMark. La forma que tendrá dicho circuito será la que aparece representada en la siguiente figura. Como se puede observar, el robot tendrá que caminar, y realizar giros. De esta manera irá sorteando los obstáculos hasta llegar a la zona de meta, contando únicamente con la información que reciba a través de los sonares y de la cámara del robot puesto que esta ultima esta ligada con el procesamiento de imágenes para detectar y decodificar las LandMark encontradas durante el recorrido del laberinto..



4.2 Problemas encontrados

En este apartado se describirán los principales problemas encontrados y las distintas soluciones aportadas a cada uno de ellos.

- **Existencia de retardos.**

Existen retardos entre el momento de enviar una orden hasta que el robot la ejecuta. Estos no afectan demasiado en lo que se refiere a la operación de y toma de decisiones para el movimiento del robot ya que son especialmente peligrosos a la hora de detener el desplazamiento del robot, ya que si no se realiza la parada con suficiente tiempo, puede provocar choques y caídas. Para descartar que la mayor parte del tiempo de este retardo se estuviera consumiendo en los distintos procesamientos que lleva a cabo el sistema, se opto por utilizar el multiprocesamiento ya que con este se realiza un proceso distinto por cada actividad que realizara el robot.

Por ejemplo: 1 Proceso para caminar y otro proceso que detectara las LandMark.

El resultado fue que el procesamiento apenas necesitaba unas milésimas de segundo.
Ejemplo:

Esta es la parte del algoritmo donde inicio la parte motriz y otro proceso para el procedimiento infinito de la detección de LandMark.

```
def start(robotIp):
        p = Process(target=main, args=(robotIp,))
        f = Process(target=LandMarkDetection, args=(robotIp,))
        p.start()
        f.start()
        p.join()
        f.join()
pass
```

- **Incompatibilidades entre la versión de NAOqi y la versión de python.**

Es muy importante que la versión de NAOqi sea compatible con la versión de python que se esté utilizando, si no, se pueden producir errores que produzcan un mal funcionamiento. A los largo del desarrollo, este problema surgió varias veces al probar el sistema en distintos equipo .Fue muy complicado dar con la solución, debido a la falta de información respecto a este tipo de errores que Aldebaran Robotics proporciona. En la versión final del proyecto se utiliza NAOqi 1.14.5 y Python 2.7.

- **Incompatibilidades entre la versión de NAOqi de los simuladores y la versión de Naoqi del Robot.**

Es muy importante que la versión de NAOqi que usan los simuladores sea la misma que se esté utilizando en el robot, si no, se pueden producir errores que produzcan un mal funcionamiento. El principal problema que se tuvo, fue que los módulos de Naoqi del robot no eran compatibles con los del simulador Webots, por lo que en algunos casos fue indispensable dejar de lado el simulador para trabajar conectado con el robot.

- **Imposibilidad de utilizar las dos cámaras de NAO simultáneamente.**

Poder acceder a las dos cámaras de Nao al mismo tiempo hubiera sido muy útil para ampliar el campo visual que se le presenta al operador. De esta forma, el operador tendría información tanto de la parte más lejana del entorno, como de lo que el robot tiene inmediatamente delante. Sin embargo, debido a la infraestructura interna del NAO, es imposible.

5. Conclusión.

En este apartado se presentan las conclusiones obtenidas tras la finalización del proyecto y la realización de la evaluación. Además, se comentan las posibles mejoras que se podrían realizar en el futuro.

En este trabajo se realizó una investigación y análisis exhaustivo para crear un algoritmo en python capaz de encargarse del funcionamiento de un robot autónomo NAO que fuera capaz de recorrer un laberinto con cualquier tipo de formato, a condición de que los trayectos fueran rectos y con desviaciones a 90 grados en ambos sentidos.

La principal conclusión obtenida tras la realización de este proyecto es que las oportunidades que ofrece el robot son inmensas, pero cabe señalar que existen varios problemas relacionados con guiar el robot con los sonares, puesto que existen técnicas mas avanzadas como procesar imágenes para obtener la posición y obstáculos dentro del entorno donde se encuentra el robot y por ende la determinación de las trayectorias a seguir serian mas precisas.

En definitiva, la tarea de investigar y estudiar los diferentes campos que se proponían se ha visto recompensada y concluida en aplicaciones que cubren con altas expectativas lo estudiado. El laberinto, hace uso de la comunicación de hardware y software para tratar de manera distinta la generación y gestión de trayectorias. Los problemas y complicaciones surgidos durante la realización del proyecto, han derivado en soluciones que han motivado la ambición de seguir.

6. Bibliografía.

http://community.aldebaran.com/doc/