ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE INVESTIGACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA

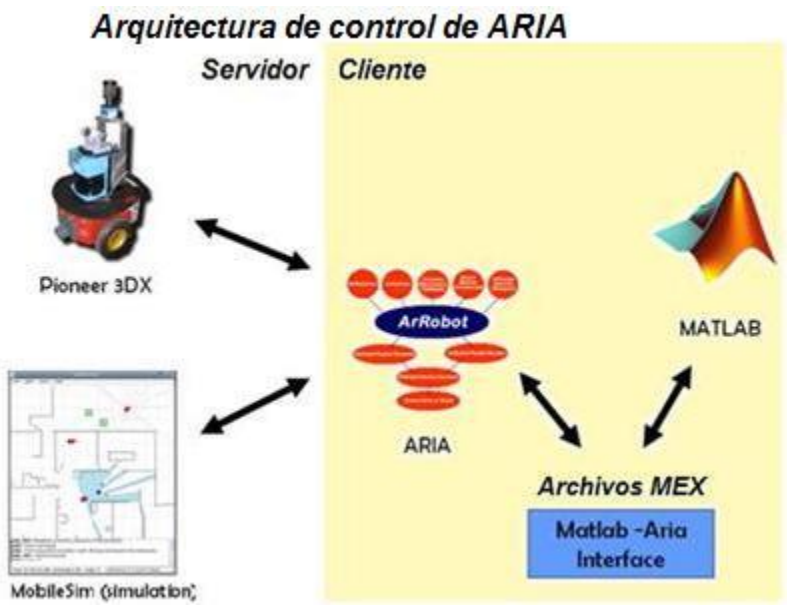
TEMA: CONTROL NEURONAL DEL ROBOT MÓVIL PIONEER P3-DX MEDIANTE UN PERCEPTRÓN MULTICAPA IMPLEMENTADO EN MATLAB

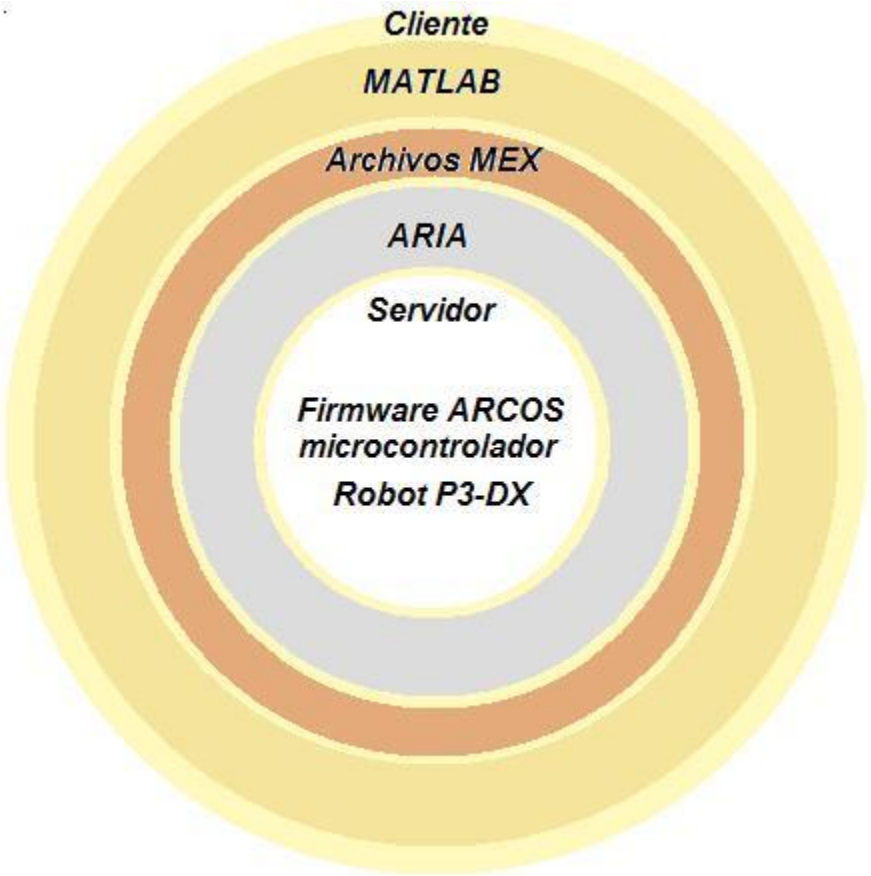
AUTOR: PAÚL SANTILLÁN R.

DIRECTOR: ING. VÍCTOR PROAÑO

SANGOLQUÍ 2015

Esquema Introductorio

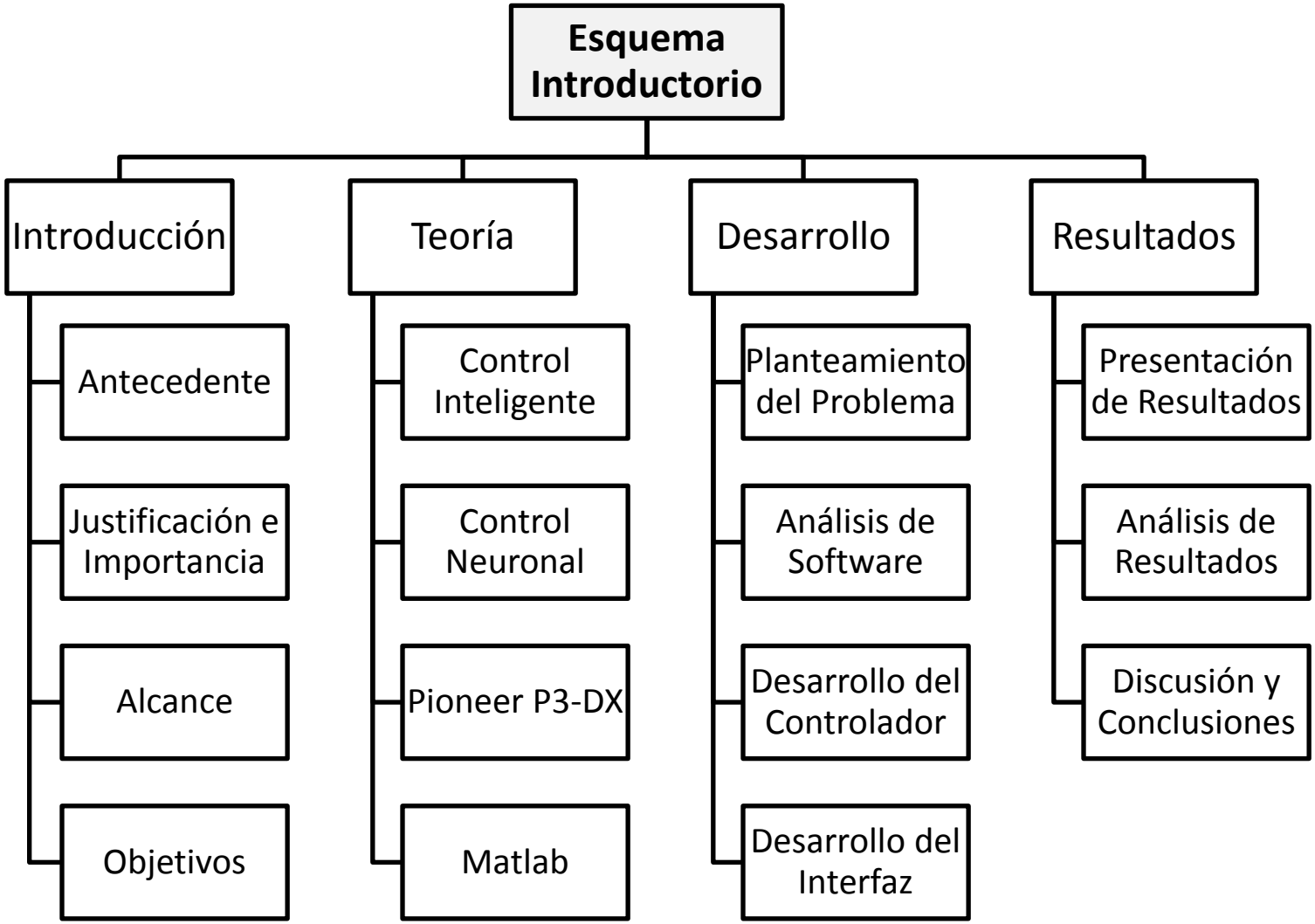




Fuente: (Posada, 2009)

Ref.: Pág. 92

Orden de Presentación



Antecedente

Tres Proyectos, Departamento de Eléctrica y Electrónica, ESPE

- *Desarrollo de Aplicaciones y Documentación de las Plataformas Robóticas Pioneer P3-DX y Pioneer P3-AT*

  Jimena Morales y Daniel Jaramillo, 2010

- *Evolución Artificial y Robótica Autónoma desarrollada en el Robot P3-DX con aproximación basada en comportamientos*

  Marco Flores y Andrés Proaño, 2013

- *Control Remoto por Voz del Robot Pioneer P3-DX*

  Diego Guffanti, 2013

Justificación e Importancia

Aporte al departamento, ámbitos: de Investigación y de Docencia

- Líneas de investigación:
  Automatización y Control → C. Neuronal
  Robótica → Robótica Autónoma

- Proyecto pionero, departamento:
  Control inteligente y Robótica móvil → Grado superior de autonomía

- Conjunción de líneas de investigación:
  Creciente interés, permanente estudio, continuo desarrollo e intensa aplicación → en ingeniería

- Guía de práctica, estudiantes de carrera:
  - Repetible, profundo entendimiento
  - Fundamentada teoría impartida
  - Documentada en detalle

- Beneficios para estudiantes, conocer:
  - Aplicaciones reales: exploración, navegación y vigilancia robótica
  - Temas para futuros trabajos en líneas
  - Interfaz Matlab Aria con Archivos MEX

Beneficiarios: departamento y estudiantes de la carrera

Alcance

Controlador Neuronal, Pioneer P3-DX, Inteligencia y Autonomía

- Demostrar comportamiento inteligente →Control de Trayectoria sobre la plataforma:
  - aplicación tradicional en Redes Neuronales Artificiales
  - base de otras aplicaciones la exploración, la navegación y la vigilancia robótica

- Diseñar Controlador Neuronal → Neural Network Toolbox de Matlab
  disponibilidad de comandos apropiados para creación, entrenamiento y simulación de red.

- Conseguir transferencia de datos C++ (ARIA) - Matlab (Neurocontrolador)→
  Interfaz Matlab ARIA, MEX-Files

- Determinar el algoritmo de entrenamiento rápido → mejores resultados en red:
  esquema comparativo de tres métodos: *'trainbfg'* -*'trainoss'* -*'trainlm*'

Objetivo General

Desarrollar un Controlador Neuronal, capaz de ejecutar un Control de Trayectoria sobre el robot móvil Pioneer P3-DX,

basado en un perceptrón multicapa entrenado mediante método de descenso de gradiente de segundo orden,

empleando las herramientas de Matlab: “Neural Network Toolbox” y “MEX Files”.

Objetivos Específicos

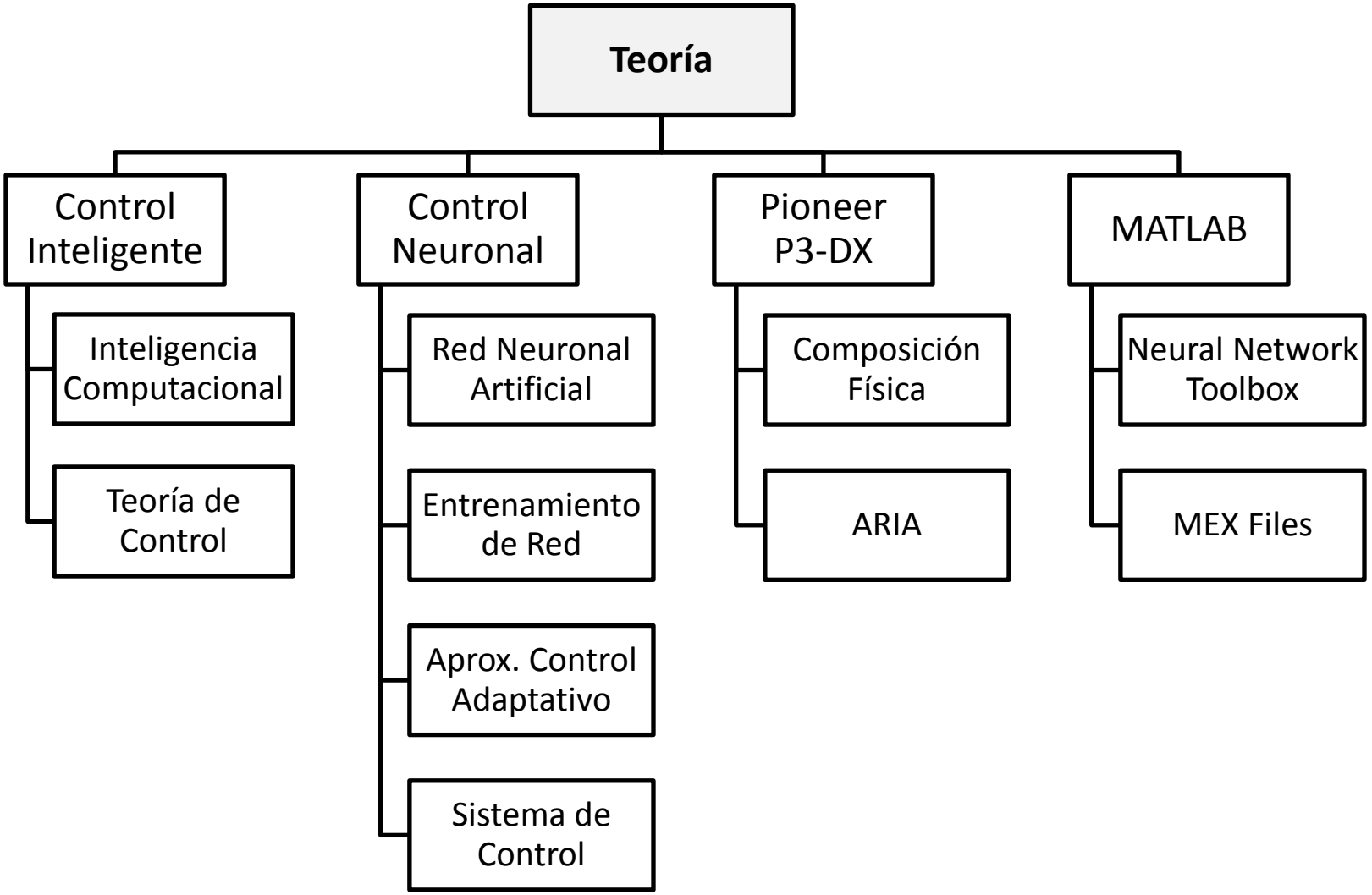
- Diseñar un Controlador Neuronal, usando el esquema de control neuronal directo: basado en un modelo simulado de referencia del entorno , utilizando la herramienta “Neural Network Toolbox”.

- Utilizar la herramienta de Matlab “MEX Files” para crear un Interfaz de Comunicación entre los lenguajes de programación Aria y Matlab, para transferir datos entre los algoritmos de control desarrollados en Matlab y la plataforma robótica móvil Pioneer P3-DX.

Base Teórica

- ¿Por qué un controlador inteligente?
- ¿Por qué un controlador neuronal?
- ¿Por qué el robot móvil Pioneer P3-DX?
- ¿Por qué Matlab?
- ¿Por qué un archivo MEX?

Orden de Presentación



Definición

Conjunto de técnicas y esquemas de control:

basado → en los enfoques de la **inteligencia computacional**

busca → integrar inteligencia en la **teoría de control**

obtener → sistemas y/o máquinas inteligentes.

NSF - National Science Foundation (1992; Handbook of Intelligent Control)

El Control Inteligente debe abarcar tanto la inteligencia como la teoría de control. Debe basarse en una tentativa seria de entender y reproducir el fenómeno que siempre hemos llamado "inteligencia", es decir la capacidad de tipo generalizado, flexible y adaptativo que vemos en el cerebro humano. Además, debe estar firmemente arraigado en la teoría de control a la mayor medida posible.

Ref.: Pág. 5

Definición

“Teoría de las inteligencias múltiples” (Psic. Howard Gardner; 1983)

→ Inteligencia humana: compleja, multifacética, subjetiva y relativa a situaciones y habilidades específicas.

→Dificultades para reproducirla e imitarla en sistemas y/o máquinas: intentos registrados como enfoques de inteligencia computacional.

IEEE CIS - Sociedad de Inteligencia Computacional IEEE Capítulo Chile (2003)

Inteligencia computacional es una colección de **paradigmas computacionales** con **inspiración biológica y lingüística**, en los cuales se incluyen la teoría, el diseño, la aplicación y el desarrollo de redes neuronales, sistemas conexionistas, algoritmos evolutivos, sistemas difusos y sistemas inteligentes híbridos.

Ref.: Pág. 6

Enfoques

	AI Convencional	Redes Neuronales	Lógica Difusa	Métodos Genéticos
1940s	47 Cibernética	43 Modelo Neurona		
1950s	56 AI	57 Perceptrón		
1960s	60 Lenguaje LISP	60s Adaline Madaline	65 Conjuntos difusos	
1970s	75 Sistemas expertos	74 Backpropagation 75 Neocognitrón	74 Controladores difusos	70s Algoritmo Genético
1980s	80s Búsquedas heurísticas	80 Mapa Auto-organizado 82 Red Hopfield 84 Máquina Boltzmann 86 Boom Backpropagation	85 Modelamiento difuso	80s Modelamiento inmune de vida artificial
1990s		Aplicaciones	90 Mo. Neuro-Fuzzy 91 ANFIS 94 CANFIS	90 Programación Genética
2000s			Aplicaciones	Aplicaciones

Fuente: (Cheok, 2002)

Bio-informática
Bio-ingeniería

Criterio unificador

A pesar de que cada uno de los enfoques se fundamenta en un proceso biológico diferente: mental, evolutivo, o lingüístico,

todos tienen como única finalidad → integrar cierto grado de inteligencia a un sistema o una máquina.

Categorías o tipos

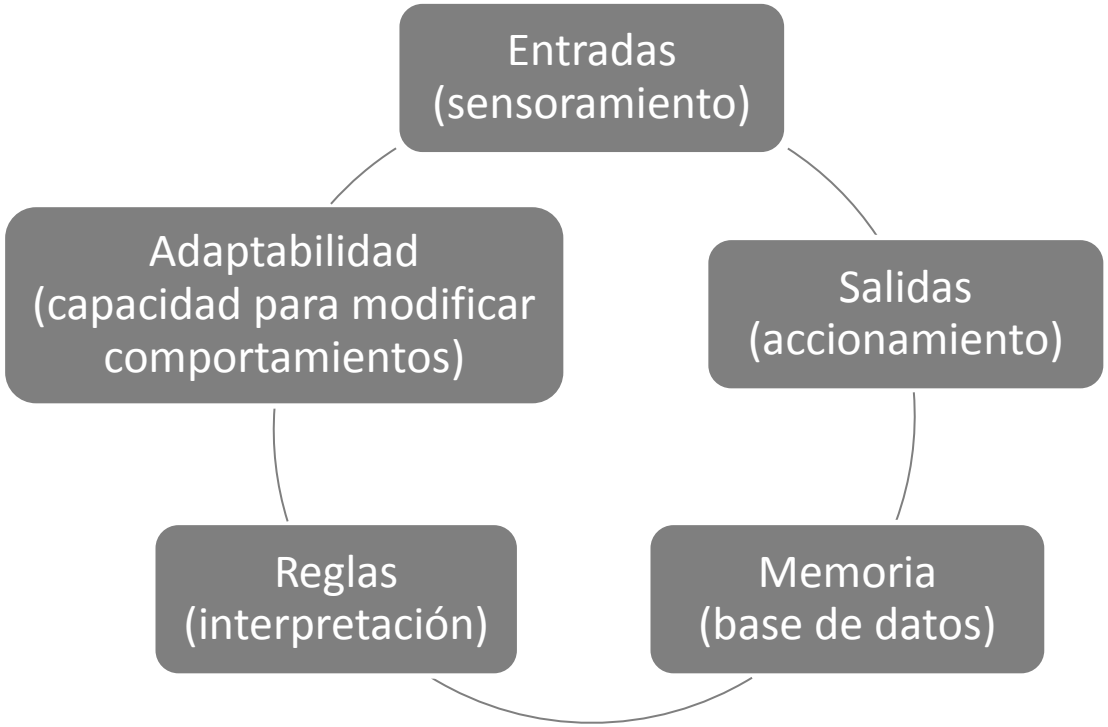
(Stuart Russell & Peter Norvig;2010;Artificial Intelligence: A Modern Approach)

SISTEMA: Que piensa como humano	*Que actúa como humano*	*Que piensa racionalmente*	*Que actúa racionalmente (ideal)*
• Imita pensamiento humano. • Automatiza actividades de procesos: toma de decisiones, resolución de problemas y el aprendizaje. • Ejemplo: redes neuronales artificiales.	• Imita comportamiento humano. • Cómo lograr que las máquinas hagan tareas, que el humano las hace mejor? • Ejemplo: la robótica.	• Cómo pensar racionalmente? • reproduce el pensamiento lógico racional del humano. • Usa cálculos para percibir, razonar y actuar. • Ejemplo: sistemas expertos (*Deep & Deeper Blue*).	• Categoría más compleja. • Cómo actuar racionalmente? • Emula idealmente el comportamiento humano. • Ejemplo: agentes inteligentes.

Elementos esenciales

De un Agente Inteligente

(Prof. K. Cheok ; Universidad de Oakland; 2002)



Ref.: Pág. 8

Desarrollo de Esquemas de CI

Inteligencia Computacional → Teoría de Control

(David White &Donald Sofge ;NSF - National Science Foundation ;1992)

Desarrollados intuitivamente

fundamentados en procesos biológicos y lingüísticos,

Una vez definidos

debieron ser entendidos y evaluados en la más profunda matemática,

Consiguiendo esquemas de control para sistemas de alta complejidad

teoría de control convencional ya era insuficiente.

Convencional vs. Inteligente (1)

Control convencional (clásico o moderno)

- Base fundamental para aplicación

  → Conocimiento de la Dinámica del sistema o proceso a controlar

  → Representada mediante Modelos Matemáticos:

  Función de transferencia o Ecuación de estado;

  → ecuaciones diferenciales o ecuaciones en diferencias

Limitaciones

Ecuaciones solo pueden representar a:

sistemas lineales invariantes en el tiempo (LTI), o

determinados sistemas no lineales.

Convencional vs. Inteligente (2)

¿Qué pasa, si …?

- no se dispone del modelo matemático del proceso, o
- la estructura y los parámetros cambian impredeciblemente en el tiempo aumentando su complejidad

  → sistemas no lineales a un grado tal, que técnicas de control convencional son insuficientes e inaplicables.

Esquemas basados en técnicas introducidas por la teoría de Control Inteligente

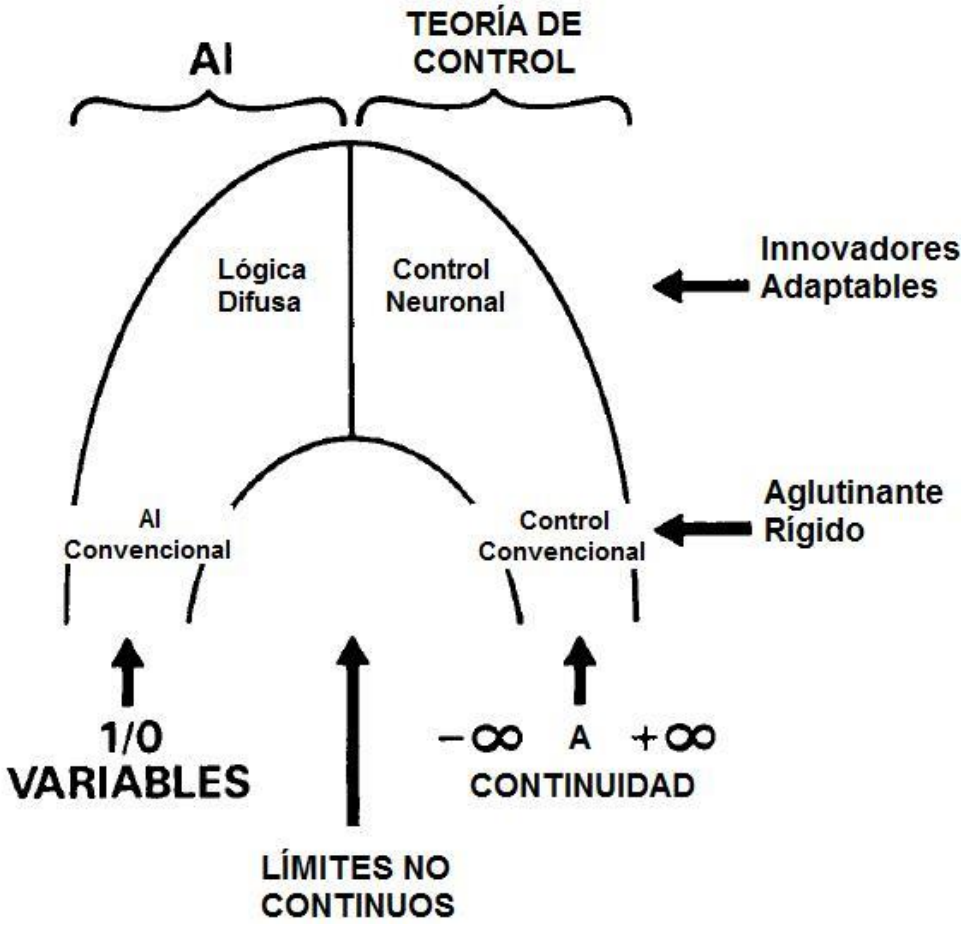
- Características:
  - Capacidad de controlar sistemas lineales, no lineales, estáticos y dinámicos
  - Gran desempeño en operaciones en entornos adversos
  - Capacidad de auto-gobernarse
  - Trabajo con gran cantidad de datos

Incremento Complejidad Controlador

(Cotero Ochoa; 2005; Control Clásico y Control Inteligente)

Control retroalimentado determinístico	• Plantas representadas con modelos lineales simples • Aproximaciones al comportamiento real
Estimadores de Estado	• Incremento en complejidad de la planta
Filtros de Kalman	• Incremento de la señal de ruido (blanco aditivo) • Estado oculto, no medible, en sistema dinámico lineal
Técnicas de Control Óptimo	• En tiempo mínimo o energía mínima, optimización • **Estocástico:** características cuantificables estocásticas • **Robusto y/o Adaptativo:** variaciones en parámetros
Técnicas Control Inteligente	• **Auto-organizado:** aprendizaje no supervisado • **Autónomo inteligente:** toma de decisiones autónomas

Convencional vs. Inteligente (3)



Fuente: (National Science Foundation NFS: White, David A.; Sofge, Donald A., 1992)

Ref.: Pág. 9

Definición

Deducción de definición de Control Inteligente

(NSF - National Science Foundation ;1992; Handbook of Intelligent Control)

> Es el uso de sistemas de control, capaces de aprender con el tiempo como alcanzar objetivos (u optimizar) en entornos no lineales, ruidosos y complejos, cuya dinámica en última instancia debe ser aprendida en tiempo real. Este tipo de control no se puede lograr mediante simples mejoras incrementales sobre los enfoques existentes.

Controladores Neuronales o Neuro-controladores

(Cotero Ochoa; 2005; Control Clásico y Control Inteligente)

- Sistemas usados en control inteligente → diseñados con redes neuronales artificiales
- Esquema de diseño → problema de optimización numérica no lineal.

Ref.: Pág. 10

Redes Neuronales Artificiales

Características:

- Imitan procesos del pensamiento humano
- Capacidad de :
  - aprender de ejemplos
  - realizar tareas complejas (percepción y reconocimiento de patrones)
- Tolerancia a fallos (cierta medida)
- Uso en:
  - modelos lineales y no lineales
  - sistemas multivariables estáticos y/o dinámicos

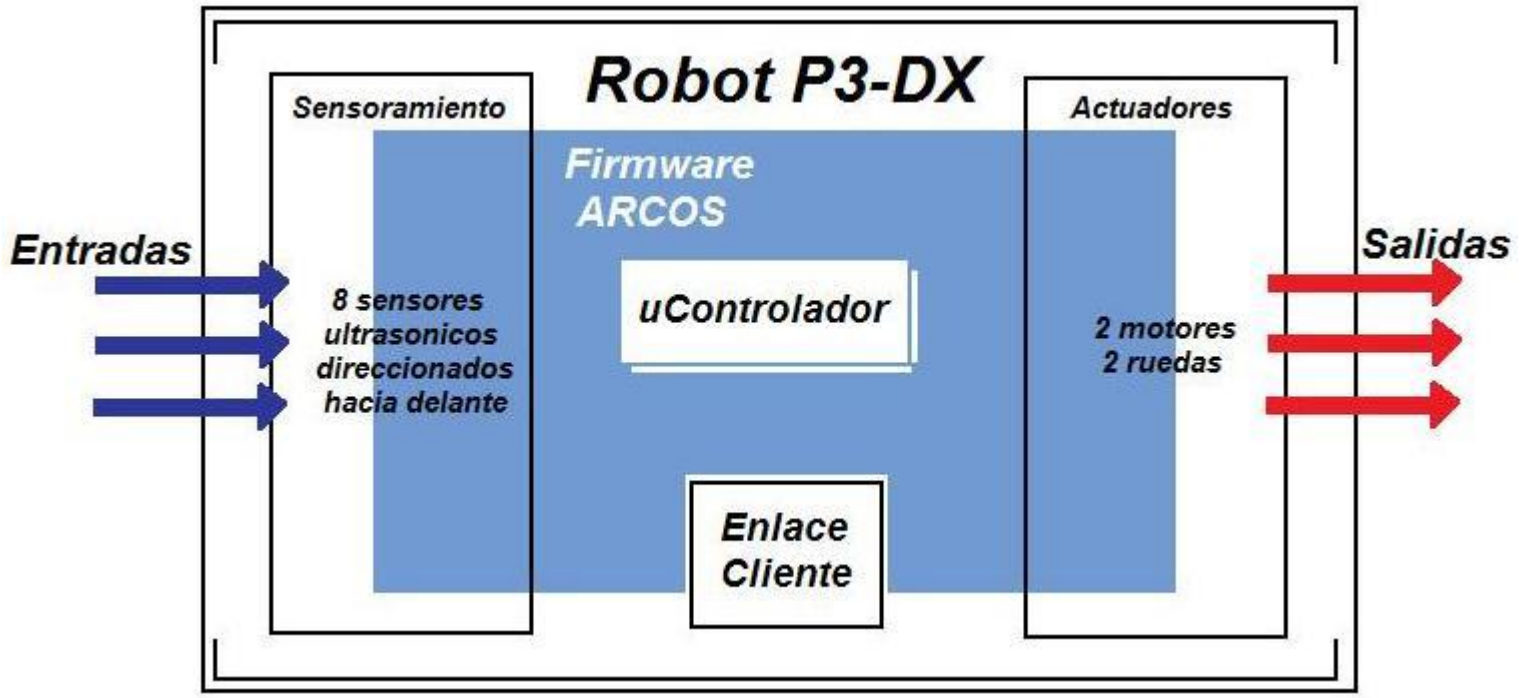
(Robert Babuska; 2001; Fuzzy and Neural Control)

- Procesamiento paralelo → Regla de los 100 pasos

(David Kriesel; 2007; A Brief Introduction to Neural Networks)

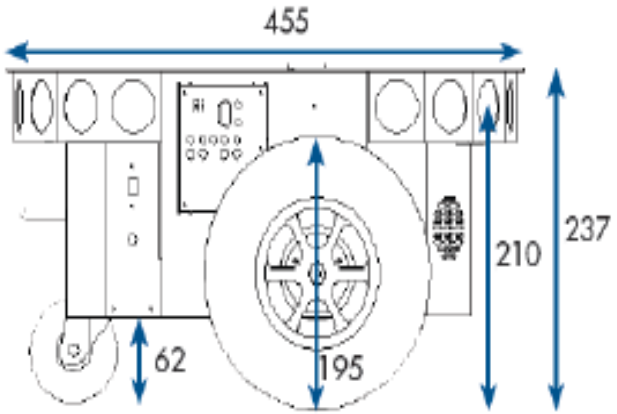
Ref.: Pág. 11

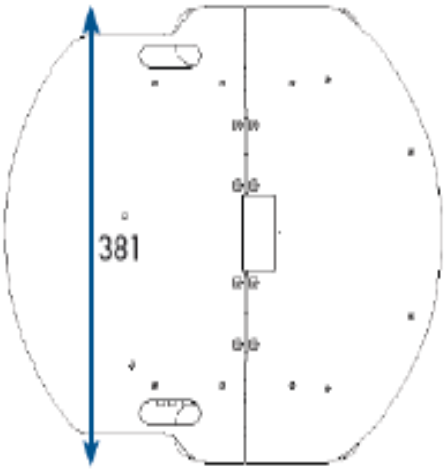
Definición de Plataforma

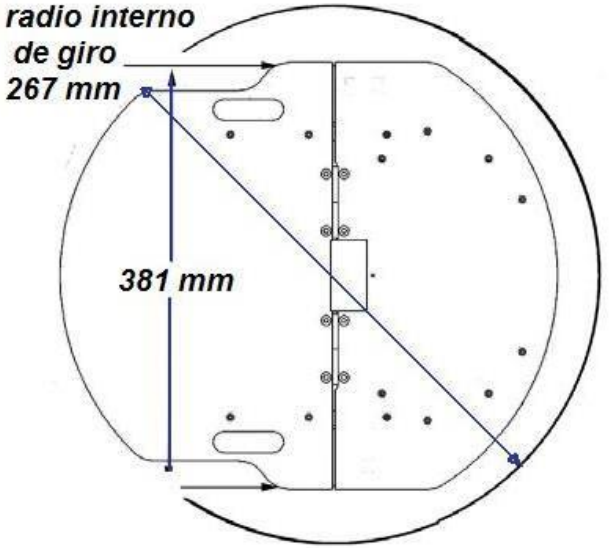


Ref.: Pág. 16

Dimensiones

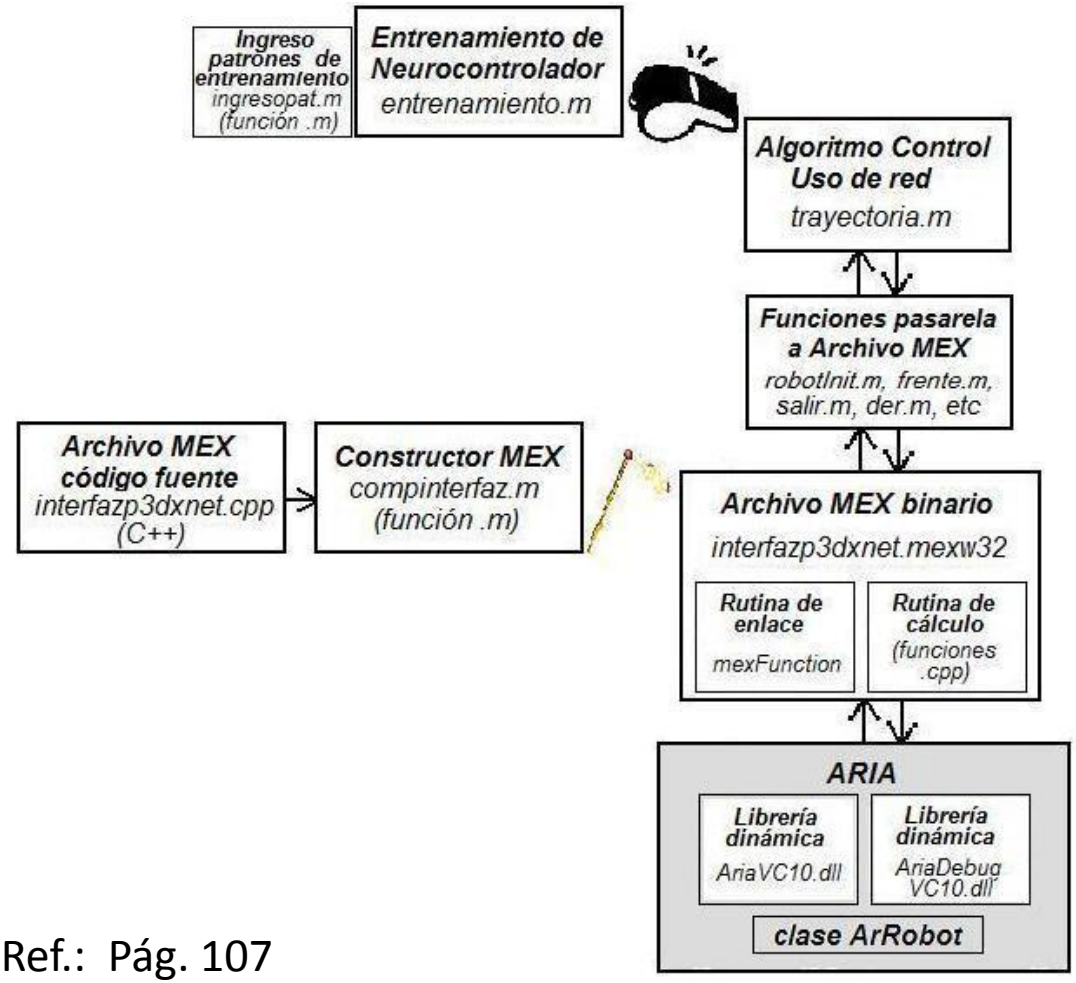






Ref.: Pág. 16

Desarrollo



Planteamiento Problema

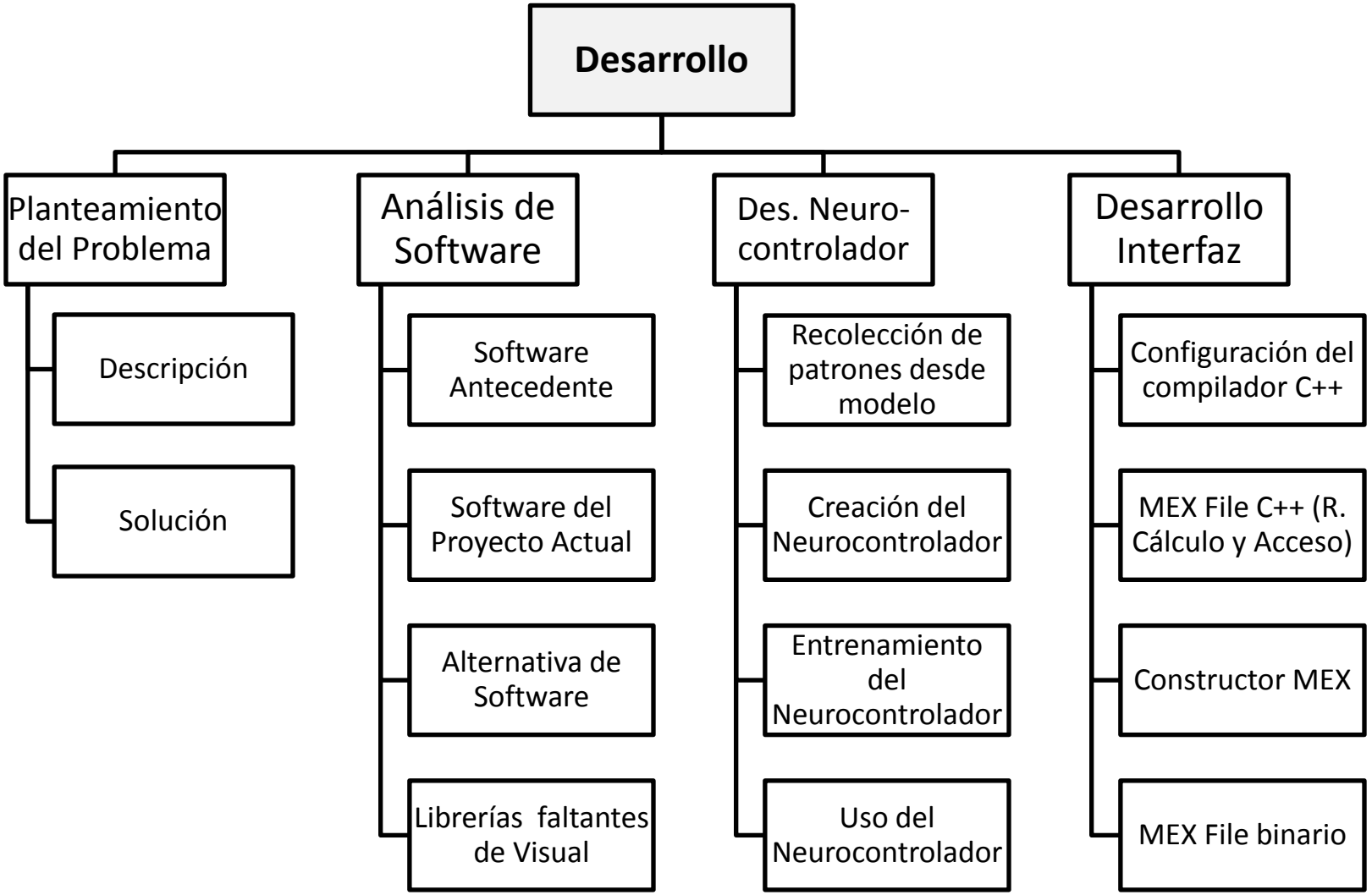
Análisis de Software

Desarrollo de Neurocontrolador

Desarrollo de Interfaz

Ref.: Pág. 107

Orden de Presentación



Descripción

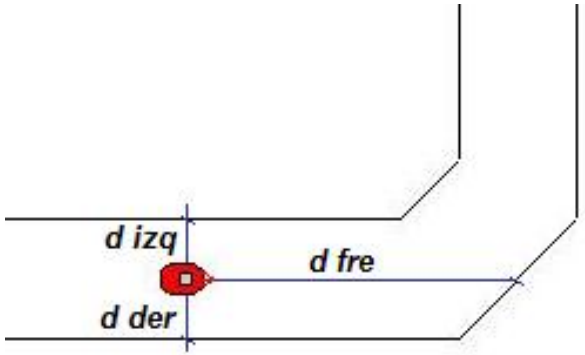
Controlar la trayectoria de la plataforma robótica móvil Pioneer P3-DX, cuando ésta se mueve dentro de una pista para evitar que colisione con las paredes. La plataforma no tiene una posición inicial definida dentro de la pista, ni tampoco un sentido de giro definido.

Inspirado en la aplicación introductoria que David Kriesel publicó en su libro "A Brief Introduction to Neural Network"

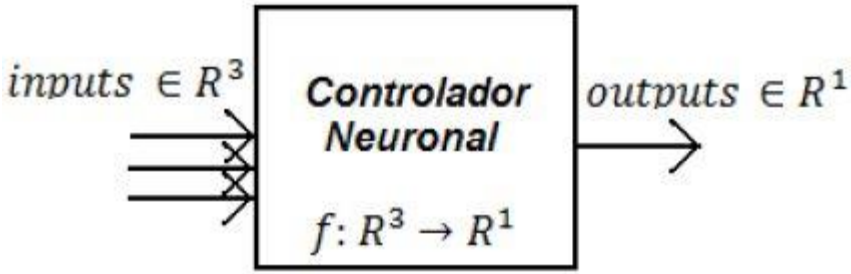
Ref.: Pág. 131

Solución

Distancia de sensoramiento



Aproximación de función con Neurocontrolador



Ref.: Pág. 132

Software Antecedente

Proyecto antecedente:

"*Control remoto por voz del robot Pioneer P3-DX*"

- Sistema operativo: Windows XP, arquitectura de 32 bits
- Microsoft Visual Studio .NET 2003 (7.1), que contiene a Visual C++
- Matlab R2009a (7.8)
- ARIA 2.7.3
- MobileSim (versión sin especificar)

Ref.: Pág. 93

Software Actual y Alternativa

Software del proyecto actual

Sistema Operativo	Windows 7 - SP2 Arquitectura de 32 bits
Microsoft Visual Studio 2010 Ultimate	(10.0), contiene a Visual C++ 2010
Matlab R2014a	(Versión 8.3)
ARIA 2.7.6	Liberado el 25 de julio del 2014
MobileSim-0.7.2-1	Liberado el 11 de junio del 2013

Alternativa para el software actual

Sistema Operativo	Windows Vista Home Basic - SP2 Arquitectura de 32 bits
Microsoft Visual Studio 2010 Ultimate	(10.0), contiene a Visual C++ 2010
Matlab R2011a	(Versión 7.12.0)
ARIA 2.7.6	Liberado el 25 de julio del 2014
MobileSim-0.7.2-1	Liberado el 11 de junio del 2013

Ref.: Pág. 95

Microsoft Visual Studio 2010 Ultimate

Sistemas Operativos Compatibles

Versión	32 bits	64 bits
Windows XP, excepto Starter Edition	Si	N/D
Windows Vista con Service Pack 2 (SP2), excepto Starter Edition	Si	Si
Windows 7	Si	Si
Windows Server 2003 SP2	Si	Si
Windows Server 2003 R2	Si	Si
Windows Server 2008 SP2	Si	Si
Windows Server 2008 R2	N/D	Si

Fuente: (Microsoft-Support-Visual-Studio, 2010)

Ref.: Pág. 96

Matlab R2014a

Compiladores soportados por esta versión de Matlab

Compiler	MATLAB *For MEX-file compilation and external usage of MATLAB Engine and MAT-file APIs*	MATLAB Compiler *For C and C++ shared libraries*	MATLAB Builder EX *For all features*	MATLAB Builder NE *For all features*	MATLAB Builder JA *For all features*	MATLAB Coder *For all features*	SimBiology *For accelerated computation*	Fixed-Point Designer *For accelerated computation*
lcc-win32 v2.4.1 *Included with MATLAB*	✓					✓ 5	✓	✓
Microsoft Windows SDK 7.1 *Available at no charge; requires .NET Framework 4.0*	✓	✓	✓	✓ 3		✓ 5	✓	✓
Microsoft Visual C++ 2013 Professional	✓	✓	✓	✓ 3		✓	✓	✓
Microsoft Visual C++ 2012 Professional	✓	✓	✓	✓ 3		✓	✓	✓
Microsoft Visual C++ 2010 Professional SP1	✓	✓	✓	✓ 3		✓	✓	✓
Microsoft Visual C++ 2008 Professional SP1 [1]	✓	✓	✓	✓ 3		✓	✓	✓
Intel C++ Composer XE 2013 [2]	✓							

Fuente: (MathWorks-Support, 2015)

Ref.: Pág. 99

Compatibilidades de Matlab

Con los SO y con MV C++ 2010

Matlab	Corre en Windows Vista SP2, Windows 7 y Windows 7 SP1	Compatible con Visual C++ 2010 Professional y Professional SP1
R2009b (7.9)	Sí	No
R2010a (7.10)	Sí	No
R2010b (7.11)	Sí	Sí
R2011a (7.12)	Sí	Sí
R2011b (7.13)	Sí	Sí
R2012a (7.14)	Sí	Sí
R2012b (8.0)	Sí	Sí
R2013a (8.1)	Vista SP2 & 7 SP1	SP1
R2013b (8.2)	Vista SP2 & 7 SP1	SP1
R2014a (8.3)	Vista SP2 & 7 SP1	SP1
R2014b (8.4)	Vista SP2 & 7 SP1	SP1
R2015a (8.5)	Vista SP2 & 7 SP1	SP1

Ref.: Pág. 101

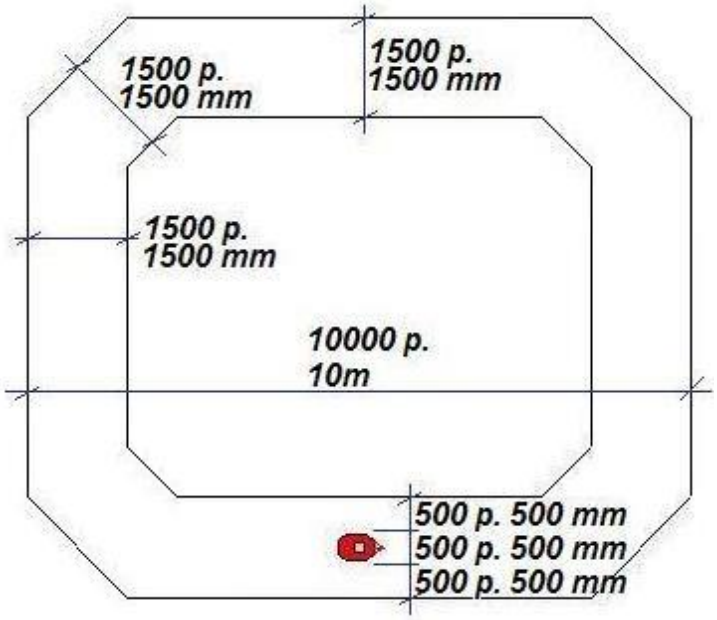
Descripción

- Neurocontrolador desarrollado:
  - Con red neuronal multicapa alimentada hacia delante 'feedforwardnet',
  - Empleando un sistema de control de tipo adaptativo directo basado en un modelo del entorno, simulado en MobileSim.

Inspirado en el proceso de diseño propuesto por Beale, Hagan y Demuth en la guía de usuario del Neural Network Toolbox

Ref.: Pág. 133

Modelo Simulado



Simulación: Pista en recta, pista en giro de 45° y Pioneer P3-DX

Ref.: Pág. 133

Recolección de patrones desde el modelo

- Beale, Hagan y Demuth advierten que:

  generalmente es difícil incorporar conocimiento previo en una red neuronal, por lo tanto la red puede ser solo tan precisa como los datos que son usados en su entrenamiento. La red no es capaz de extrapolar con presión más allá del rango de entradas, por eso es importante que los patrones de entrenamiento cubran completamente dicho rango.

  Según David Kriesel:
  si los patrones han sido correctamente seleccionados, la red neuronal generaliza desde los ejemplos y encuentra una regla universal para evitar choques con las paredes (comportamiento inteligente).

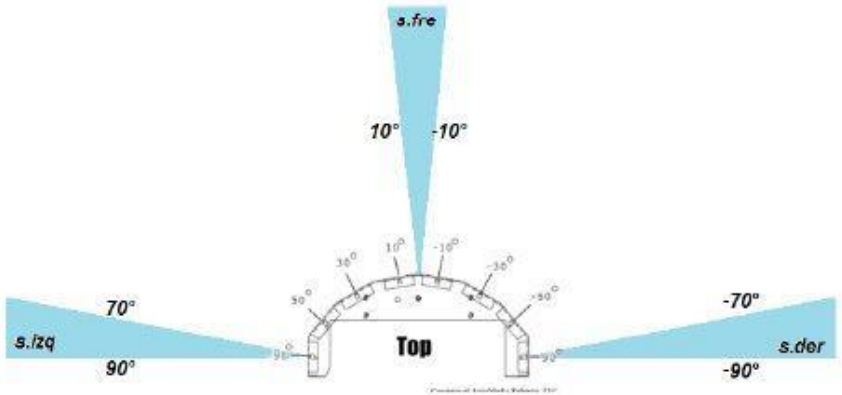
Ref.: Pág. 134

Composición de los Patrones

Entradas del Neurocontrolador

Alcance [0, +4,75]

Entradas *Inputs*	Correspondencia	Características Región barrida
in1	Lectura del sensor izquierdo	De 90° a 70°
in2	Lectura del sensor frontal	De 10° a -10°
in3	Lectura del sensor derecho	De -70° a -90°



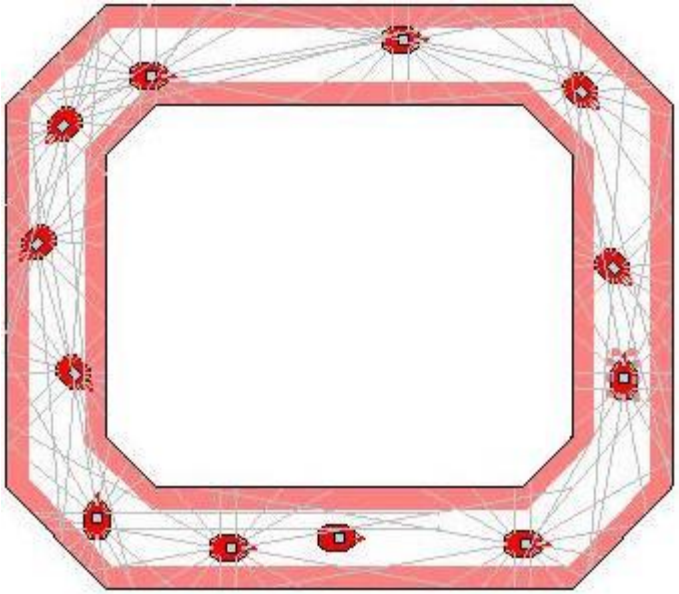
Fuente: (MobileRobots - ActivMedia Robotics, 2006)

Targets del Neurocontrolador

Acciones	Referencia *Targets*	Movimiento
Movimiento 1: ***Giro izquierda y avance***	-1	Robot gira a la izquierda 30° y avanza 200 milímetros
Movimiento 2: ***Solo avance***	0	Robot avanza 350 milímetros
Movimiento 3: ***Giro derecha y avance***	1	Robot gira a la derecha 30° y avanza 200 milímetros

Ref.: Pág. 135

Lógica de adquisición de patrones



Tres regiones de la pista: dos de alarma y una central.

Ref.: Pág. 136

Código de Matlab

- **Ingreso de patrones** → ingresopat.m
- **Creación del neurocontrolador** → entrenamiento.m

Cálculo de número de neuronas en la capa escondida (Aprox. Barron)

$$E = \mathcal{O}\left(\frac{1}{h}\right)$$

$$E = \mathcal{O}\left(\frac{1}{36}\right) \cong 0.02777$$

- **Entrenamiento del neurocontrolador** → entrenamiento.m
- **Uso del neurocontrolador** → trayectoria.m

Ref.: Pág. 138

Descripción

Pasos	Descripción	Programa
1	Tener instalado un compilador C++ soportado por Matlab	Microsoft Visual Studio C++ 2010
2	Escribir la rutina de cálculo en el archivo MEX C++	*interfazp3dxnet.cpp* (primera parte)
3	Escribir la rutina de enlace/acceso en el archivo MEX C++	*interfazp3dxnet.cpp* (segunda parte)
4	Escribir la función constructor MEX	*compinterfaz.m*
5	Compilar archivo MEX binario y usarlo como cualquier función integrada Matlab	*Interfazp3dxnet.mexw32* (Funciones pasarela: *robotInit.m*, *salir.m*, etc)

Procedimiento establecido en el manual de Interfaces Externas del Matlab R2015a – capítulo “Intro a archivos MEX”.

Ref.: Pág. 108

Configuración del compilador C++

- **Identificación del Compilador**

>>mex.getCompilerConfigurations('C' , 'Selected')

- **Cambio de compilador precargado**

>> mex -setup

Ref.: Pág. 109

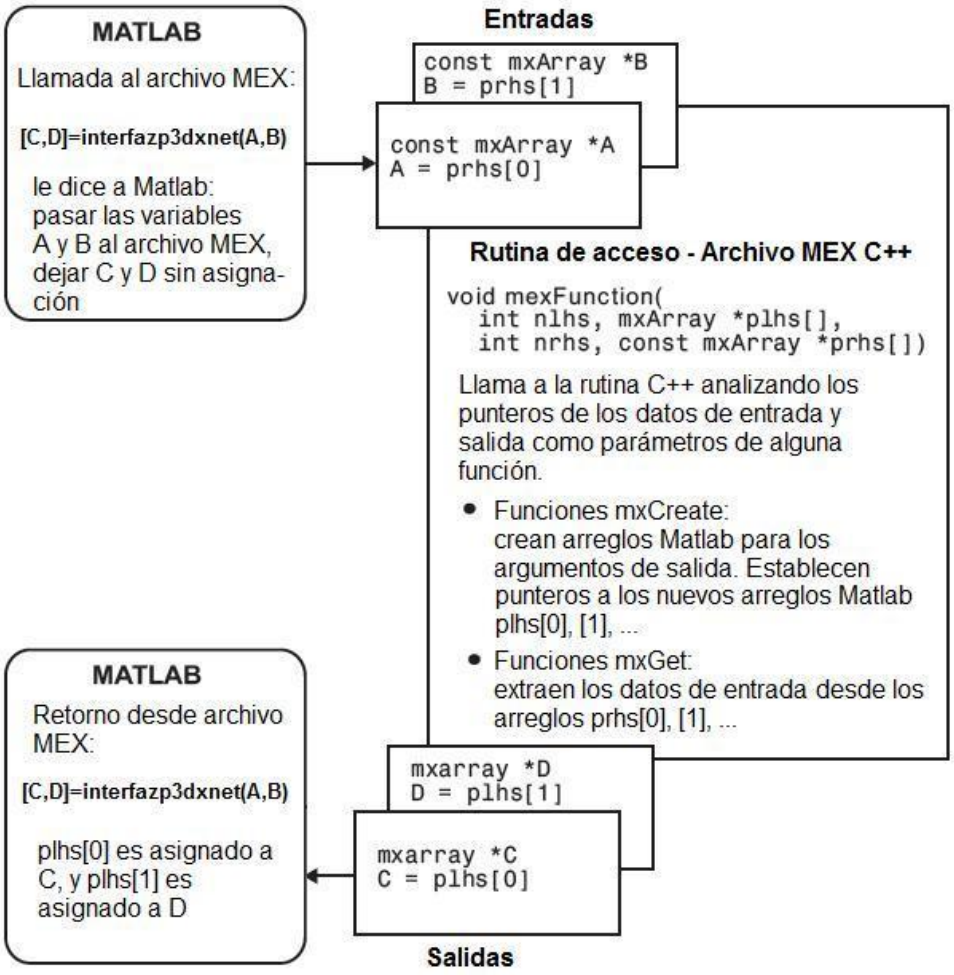
MEX File C++

- **Rutina de Cálculo:**
  - Librerías de referencia y Contenedores
  - Funciones de Configuración
  - Funciones de Sensoramiento
  - Funciones de Actuación

- **Rutina de Acceso/Enlace:**
  - Creación de la rutina, mexFunction()
  - Verificación de parámetros MEX – File de entrada
  - Selector de Funciones

Ref.: Pág. 110

Flujo de datos en Rutina de Enlace



Ref.: Pág. 85

Flujo a través de una función receptora

trayectoria.m

```
fprintf('\nPulsar
pause()
robotInit() % ///
move(400)
```

move.m

```
function move(distance)
error(nargchk(1, 1, nargin))
interfazp3dxnet(2,distance);
```

interfazp3dxnet.cpp — *Rutina de enlace*

```
void mexFunction( int nlhs, mxArray *plhs[],
          int nrhs, const mxArray*prhs[] )
   int myRobotFunction;
   double *firstArg, *secondArg;
   myRobotFunction = mxGetScalar(prhs[0]);
   if (nrhs >= 2){
       firstArg  = mxGetPr(prhs[1]);}
   switch(myRobotFunction){
      case 2:
       robot.move(*firstArg);
       break;
```

Ref.: Pág. 116

Flujo función de recepción y respuesta

trayectoria.m

```
infr=sonarFrente();
infr=infr*0.001;
inde=sonarDer();
```

sonarFrente.m

```
function [sensorArray] = sonarFrente()
error(nargchk(0, 0, nargin))
sensorArray = interfazp3dxnet(11);
```

interfazp3dxnet.cpp — Rutina de enlace

```
void mexFunction( int nlhs, mxArray *plhs[],
          int nrhs, const mxArray*prhs[] )
    double *output;
    int myRobotFunction;
    double *firstArg, *secondArg;
    myRobotFunction = mxGetScalar(prhs[0]);
    switch(myRobotFunction){
    case 11:
        plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
        output = mxGetPr (plhs[0]);
        mySonarFrente(output);
        break;
```

interfazp3dxnet.cpp — Rutina de cálculo

```
void mySonarFrente(double output[]){
    output[0]=robot.checkRangeDevicesCurrentPolar(-10,10)-robot.getRobotRadius();
}
```

Ref.: Pág. 117

Constructor MEX

- **Actualización de rutas de acceso (*path*)**
  - *Archivos Visual Studio enlazados*
  - *Archivos ARIA enlazados*

Tipo	Archivos	Descripción
De inclusión	*Aria.h, ArRobot.h, ArRangeDevice,…* (todos los archivos en carpeta include ARIA)	Integrados para que el binario pueda acceder a través de los identificadores a cualquier función de cualquier clase de la librería ARIA.
Librerías	*AriaVC10.lib* *AriaDebugVC10.lib*	Integrados para señalizar la conexión entre Aria y el compilador C++ del Microsoft Visual C++ 2010

- **Compilación combinada**
  - *Función 'mex()'*

Ref.: Pág. 118

Funciones del proyecto (1)

¿Cuántas funciones son de control?,

¿Cuál es la diferencia entre las funciones primitivas y compuestas?,

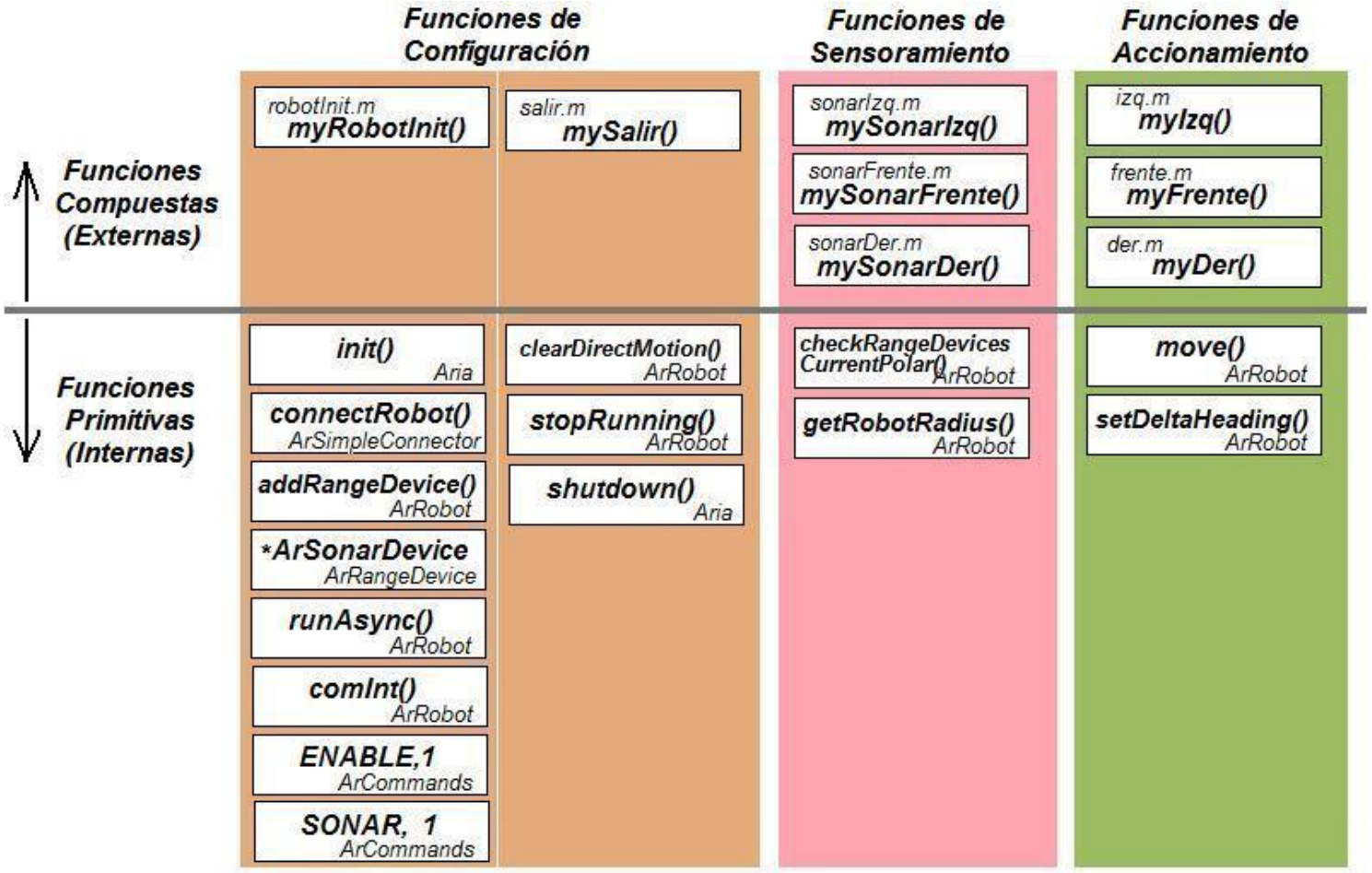
¿Por qué hay funciones pasarela?

¿Qué significa funciones de configuración, de sensoramiento y de actuación?

¿Son las mismas funciones, que las de las clases de Aria?

Ref.: Pág. 118

Funciones del proyecto (2)



Ref.: Pág. 122

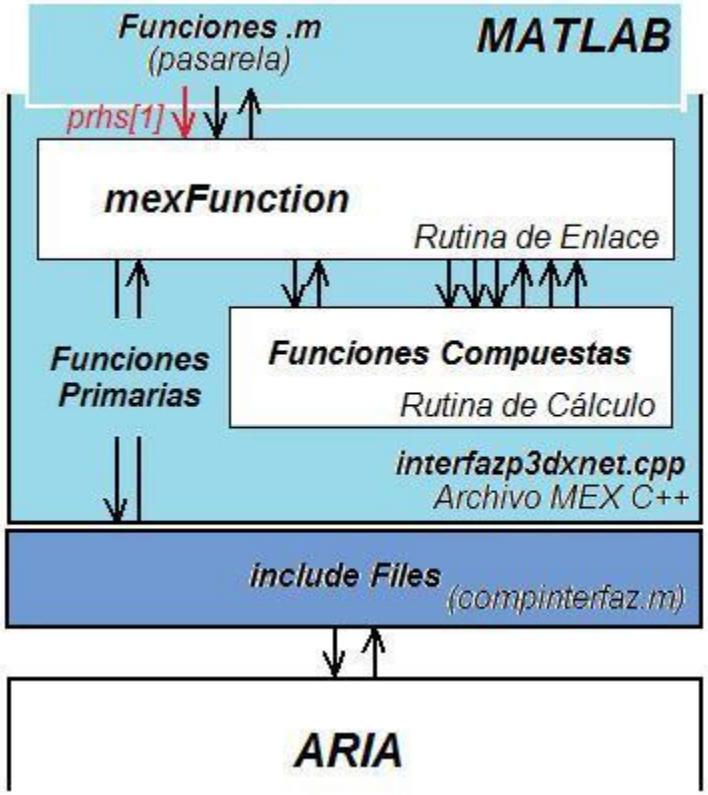
Funciones para el control del robot

Funciones Compuestas

	Matlab		Archivo MEX
	trayectoria.m	***funciones .m*** *(respectivas)*	***interfazp3dxnet.cpp***
FC	robotInit()	interfazp3dxnet(0)	myRobotInit()
FC	salir()	interfazp3dxnet(1)	mySalir()
FS	sonarIzq()	[sensorArray]=interfazp3dxnet(10)	mySonarIzq(output[])
FS	sonarFrente()	[sensorArray]=interfazp3dxnet(11)	mySonarFrente(output)
FS	sonarDer()	[sensorArray]=interfazp3dxnet(12)	mySonarDer(output[])
FA	izq()	interfazp3dxnet(20)	myIzq()
FA	frente()	interfazp3dxnet(21)	myFrente()
FA	der()	interfazp3dxnet(22)	myDer()
Fal	move(dis)	interfazp3dxnet(2,dis)	robot.move()

Ref.: Pág. 123

Funciones Primarias o Primitivas



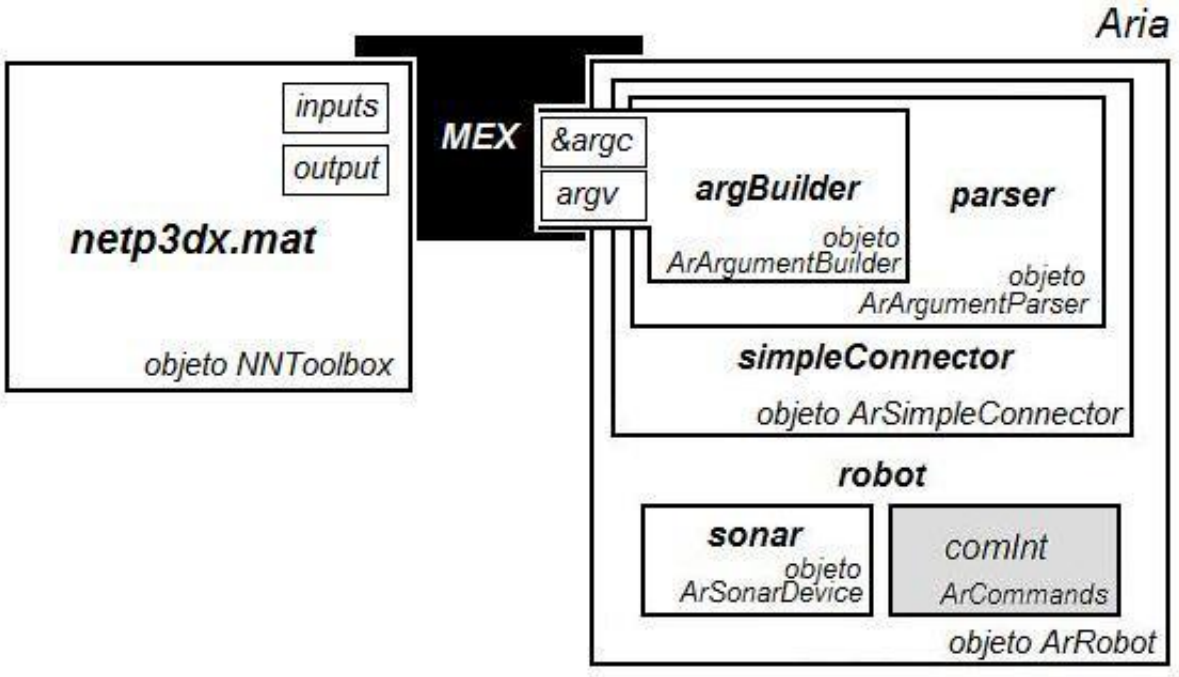
Ref.: Pág. 123

Funciones Alternas

	Matlab		Archivo MEX
	trayectoria.m	***funciones .m*** *(respectivas)*	***interfazp3dxnet.cpp***
Fal	move(dis)	interfazp3dxnet(2,dis)	robot.move(dis)
Fal	setDelta Heading(dgr)	interfazp3dxnet(3,dgr)	robot.setDeltaHeading(dgr)
Fal	setHeading (dgr)	interfazp3dxnet(4,gra)	robot.setHeading(gra)
Fal	setVel(mms)	interfazp3dxnet(5,mms)	robot.setVel(mms)
Fal	setRotVel(gs)	interfazp3dxnet(6,gs)	robot.setRotVel(gs)
Fal	setVel(mms)	[vel]=interfazp3dxnet(7)	robot.setVel(mms)
Fal	setRotVel(gs)	[rVel]=interfazp3dxnet(8)	robot.setRotVel(gs)
Ca	gIzq()	interfazp3dxnet(23)	myGIzq() → robot.setDeltaHeading(30)
Ca	gDer()	interfazp3dxnet(24)	myGDer() → robot.setDeltaHeading(-30)

Ref.: Pág. 126

Objetos del Proyecto



Ref.: Pág. 127

Arranques

En ventana de comandos de Matlab y en ventana de Simulación de MobileSim

Arranque de la Aplicación

Requisitos:

- Carpeta actual de Matlab → bin del proyecto
- MobileSim→ mapa cargado y devolución de puerto abierto

Invocación: >>trayectoria

Retorno:

- Matlab →plantilla de presentación y orden de inicio de control
- MobileSim → región de barrido de SONAR

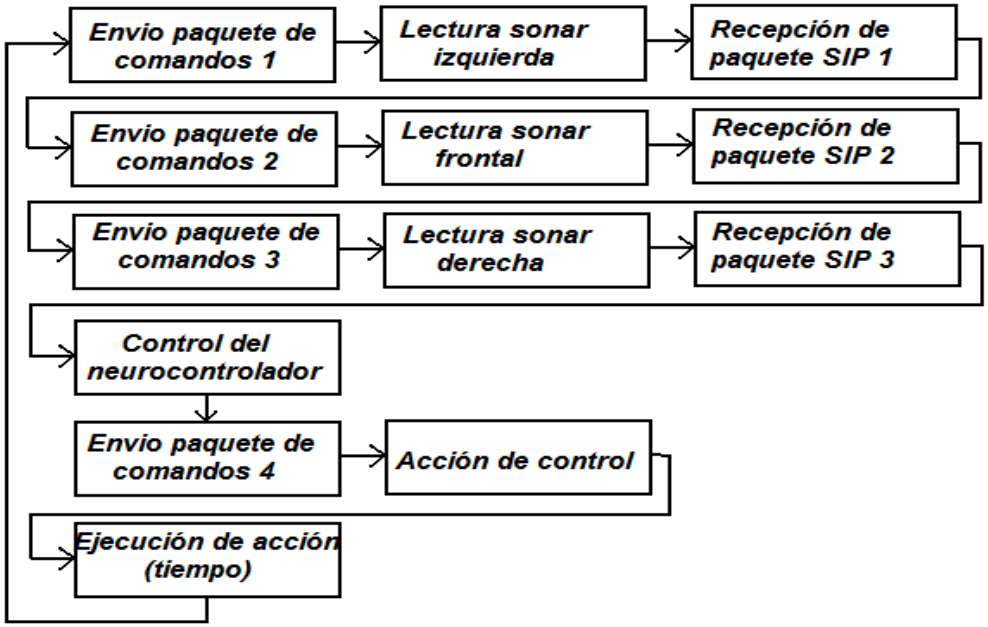
Arranque modo manual

Configuración manual de posición → funciones de control y alternas

Ref.: Pág. 145

Aplicación en Ejecución

Ciclo de ejecución retroalimentado:



Retornos en ventana de comando de Matlab:

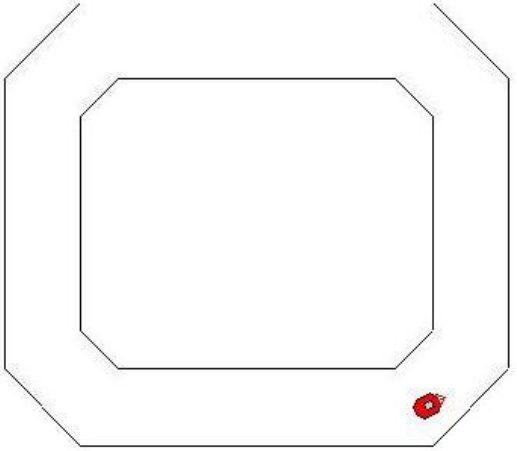
- *inputs* → lectura derecha – frontal – izquierda actualizada del SONAR
- *output* → salida generada y leyenda

Salidas de Emergencia

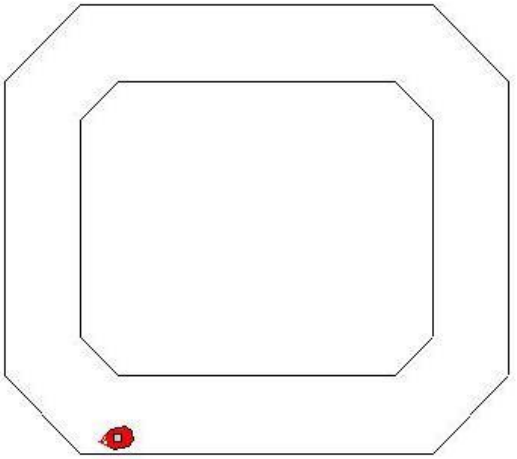
Por errores:

Tipo de salida	Criterio
Mal entrenamiento	Cuando el neurocontrolador genera una salida no acotada, a pesar de que las entradas están dentro del rango de entrenamiento
Choque inminente	Cuando una o varias lecturas del sonar muestran que el robot está demasiado cerca de una pared y un choque es inminente. Este tipo de salida es también producto de un mal entrenamiento, puntualmente de una mala generalización.

```
(10) Lectura actual del SONAR

inputs =

    0.7973
    0.9742
    0.3679


output =

    -2

Entrenamiento deficiente...
Control detenido!
==============================
```



```
(10) Lectura actual del SONAR

inputs =

    0.0403
    0.5462
    1.3175

Robot a punto de chocar!...
Detención de emergencia
==============================
```



Criterios de diseño

- Fundamentados en referencias bibliográficas
- Pruebas de rendimiento → Modificaciones para nuevos entrenamientos

Del diseño del neurocontrolador	Del diseño del interfaz de datos MEX	Del diseño completo
• Elección del perceptrón • Elección del número de neuronas ocultas • Elección del método de entrenamiento	• Frecuencia del ciclo de control	• Elección del número de patrones de entrenamiento

Ref.: Pág. 149

Análisis de Resultados

Modificaciones para nuevos entrenamientos

	Modificación	*Resultado Esperado*
1.	*>>net=init()*	Una inicialización y/o reinicialización de los pesos iniciales y bias, antes del entrenamiento, produce resultados distintos.
2.	*net.layers{1}.dimensions*	Una mayor cantidad de neuronas en la capa escondida, produce mejoras en la aproximación de la función.*
3.	Patrones de entrenamiento	Un incremento de datos de ejemplo produce mejoras en el entrenamiento.
4.	Inputs	Se debe incrementar el número de elementos de entrada, si aún se dispone de información relevante.
5.	*net.trainFcn*	Si las modificaciones anteriores no generan mejoras significativas, se recomienda también intentar con un algoritmo de entrenamiento distinto acorde al tamaño de la red.

* En caso de sobre-entrenamiento (rendimiento de entrenamiento bueno y específicamente rendimiento de prueba malo) se recomienda disminuir el número de neuronas.

Ref.: Pág. 80

Elección del Perceptrón (1)

‘feedforwardnet’ vs. ‘fitnet’ vs. ‘newff’

- 3 redes alimentadas hacia delante → aproximación de una función (*data-fiting)*

Criterios constantes:

- Número de capas: 1 oculta – 1 de salida (Criterio Aprox. De Barron)
- Número de neuronas en capa oculta: 36 (Criterio Aprox. De Barron)
- Funciones de transferencia: *‘tansig’ / ‘purelin’* (Cybenko & Beale, Hagan, y Demuth)
- Función de entrenamiento: *„trainlm*“ (Beale, Hagan, y Demuth)
- Función de división de patrones: *„dividetrain*“
- Número de patrones de entrenamiento: 50

Ref.: Pág. 149

Elección del Perceptrón (2)

'feedforwardnet' vs. 'fitnet' vs. 'newff'

Resultados:

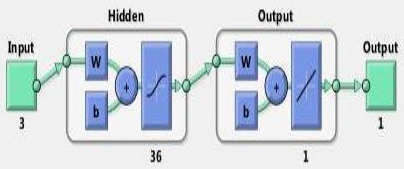
Red	Prueba	Rendimiento (mse)	Épocas	Parámetro (corta entrenamiento)
'feedforwardnet'	1	1.53 e -16	151	Valor mínimo de gradiente *min_grad* encontrado
	2	2.17 e -13	353	Valor mínimo de gradiente *min_grad* encontrado
	3	4.78 e- 19	255	Valor mínimo de gradiente *min_grad* encontrado
'fitnet'	1	1.10 6 -7	229	Valor mínimo de gradiente *min_grad* encontrado
	2	9.66 e -13	205	Valor mínimo de gradiente *min_grad encontrado*
	3	2.24 e -8	170	Valor mínimo de gradiente *min_grad* encontrado
'newff'	1	1.13 e -15	379	Valor mínimo de gradiente *min_grad* encontrado
	2	2.37 e -7	356	Valor mínimo de gradiente *min_grad* encontrado
	3	2.41 e -13	244	Valor mínimo de gradiente *min_grad* encontrado

Elección del perceptrón (3)

‘feedforwardnet’ vs. ‘fitnet’ vs. ‘newff’

Entrenamiento 'feedforwardnet' (prueba 3)

Neural Network



Algorithms

Data Division:	Training Only (dividetrain)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Derivative:	Default (defaultderiv)

Progress

Epoch:	0	255 iterations	1000
Time:		0:00:06	
Performance:	5.47	4.78e-19	0.00
Gradient:	14.8	2.83e-09	1.00e-05
Mu:	0.00100	1.00e-08	1.00e+10
Validation Checks:	0	0	6

Entrenamiento 'fitnet' (prueba 2)

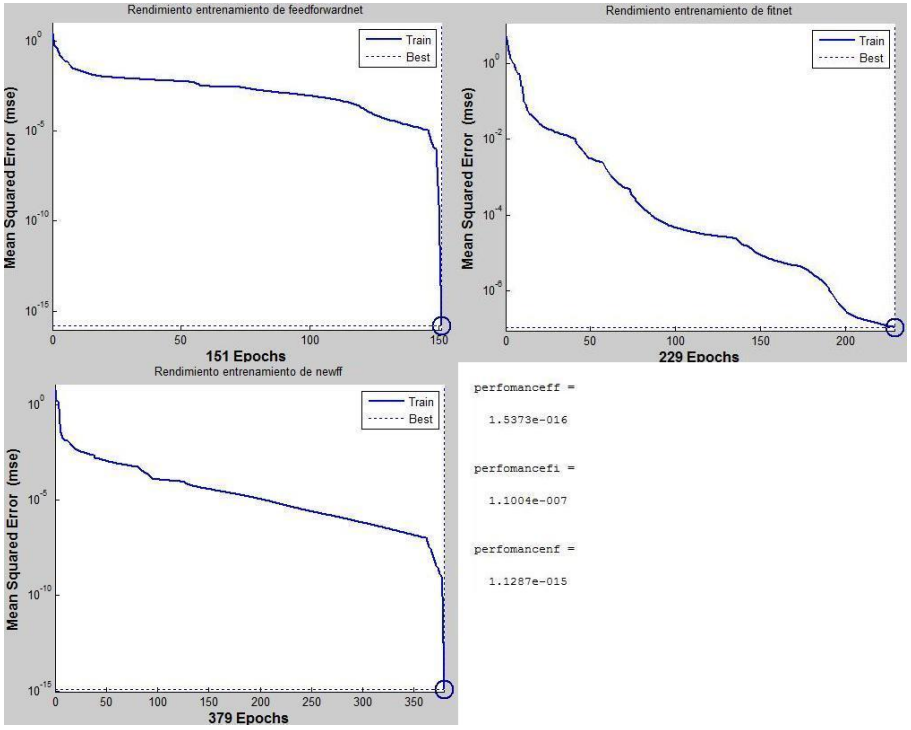
Progress

Epoch:	0	205 iterations	1000
Time:		0:00:04	
Performance:	3.32	9.66e-13	0.00
Gradient:	11.4	5.57e-06	1.00e-05
Mu:	0.00100	1.00e-07	1.00e+10
Validation Checks:	0	0	6

Entrenamiento 'newff' (prueba 1)

Progress

Epoch:	0	379 iterations	1000
Time:		0:00:08	
Performance:	5.87	1.13e-15	0.00
Gradient:	13.4	2.44e-07	1.00e-05
Mu:	0.00100	1.00e-07	1.00e+10
Validation Checks:	0	0	6




```
perfomanceff =

  1.5373e-016

perfomancefi =

  1.1004e-007

perfomancenf =

  1.1287e-015
```

Número de neuronas ocultas (1)

8 vs. 36 vs. 108

- Error cuadrático medio (*mse*) aproximado por criterio de Barron
- Red sobre-entrenada → poco espacio a generalización

Criterios constantes:

- *Tipo de red: 'feedforwardnet'* *(Probado)*
- Número de capas: 1 oculta – 1 de salida (Criterio Aprox. De Barron)
- Funciones de transferencia: *'tansig' / 'purelin'* (Cybenko & Beale, Hagan, y Demuth)
- Función de entrenamiento: *„trainlm"* (Beale, Hagan, y Demuth)
- Función de división de patrones: *„dividetrain"*
- Número de patrones de entrenamiento: 50

Ref.: Pág. 149

Número de neuronas ocultas (2)

8 vs. 36 vs. 108

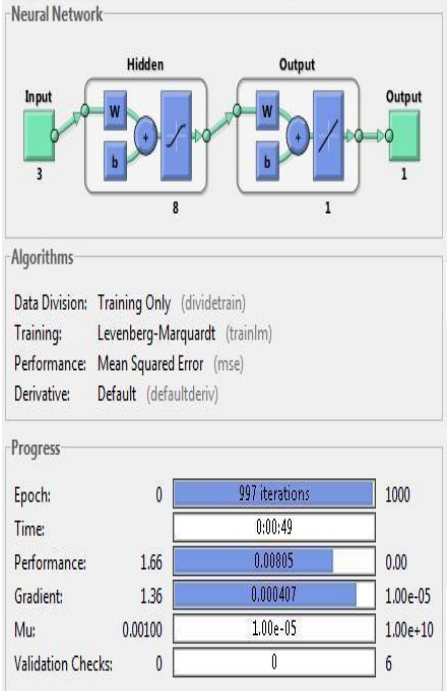
Resultados:

Num.	Prueba	Rendimiento (mse)	Épocas	Parámetro (que corta el entrenamiento)
8 ocultas	1	0.0081	>1000	exceso de épocas
	2	0.0029	>1000	exceso de épocas
	3	9.01 e-4	>1000	exceso de épocas
36 ocultas	1	4.19 e -13	269	Valor mínimo de gradiente *min_grad* encontrado
	2	4.56 e -16	187	Valor mínimo de gradiente *min_grad encontrado*
	3	2.05 e-8	360	Valor mínimo de gradiente *min_grad* encontrado
108 ocultas	1	7.92 e-14	144	Valor mínimo de gradiente *min_grad* encontrado * Cada época se demora más
	2	5.17 e-13	39	Valor mínimo de gradiente *min_grad* encontrado * Cada época se demora más
	3	6.45 e-14	55	Valor mínimo de gradiente *min_grad* encontrado * Cada época se demora más

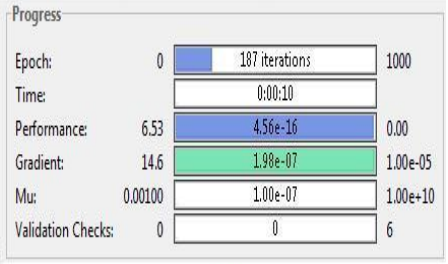
Número de neuronas ocultas (3)

8 vs. 36 vs. 108

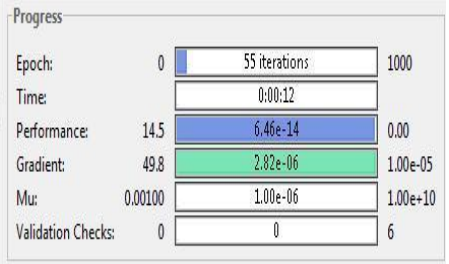
Entrenamiento 8 neuronas (prueba 1)



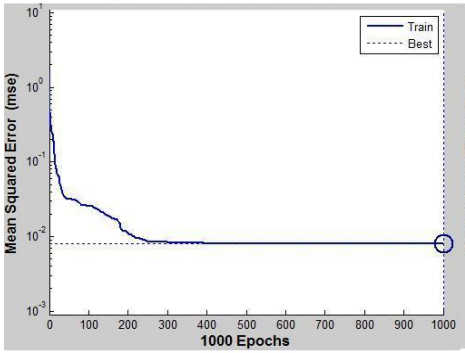
Entrenamiento 36 neuronas (prueba 2)



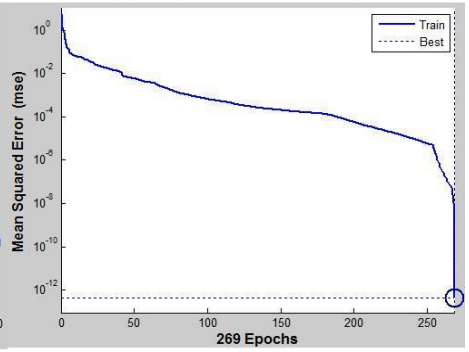
Entrenamiento 108 neuronas (prueba 3)



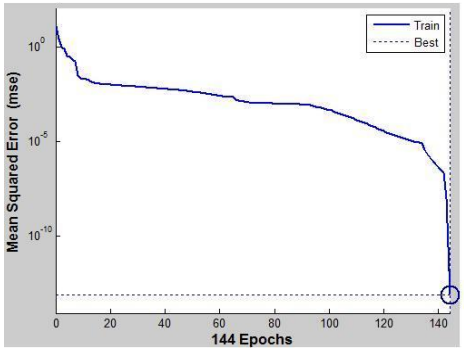
Entrenamiento 8 neuronas (prueba 1)



Entrenamiento 36 neuronas (prueba 1)



Entrenamiento 108 neuronas (prueba 1)




```
perfomance8 =

    0.0081


perfomance36 =

  4.1951e-013


perfomance108 =

  7.9263e-014
```

Elección del método de entrenamiento (1)

‘trainlm’ - ‘trainbfg’ - ‘trainoss’

- 3 métodos de entrenamiento rápido
  → Aproximan el cálculo del gradiente de segundo orden

Criterios constantes:

- *Tipo de red: ‘feedforwardnet’* *(Probado)*
- Número de capas: 1 oculta – 1 de salida (Criterio Aprox. De Barron)
- *Número de neuronas en capa oculta: 36* *(Probado)*
- Funciones de transferencia: *‘tansig’ / ‘purelin’* (Cybenko & Beale, Hagan, y Demuth)
- Función de división de patrones: *„dividetrain“*
- Número de patrones de entrenamiento: 50

Ref.: Pág. 149

Elección del método de entrenamiento (2)

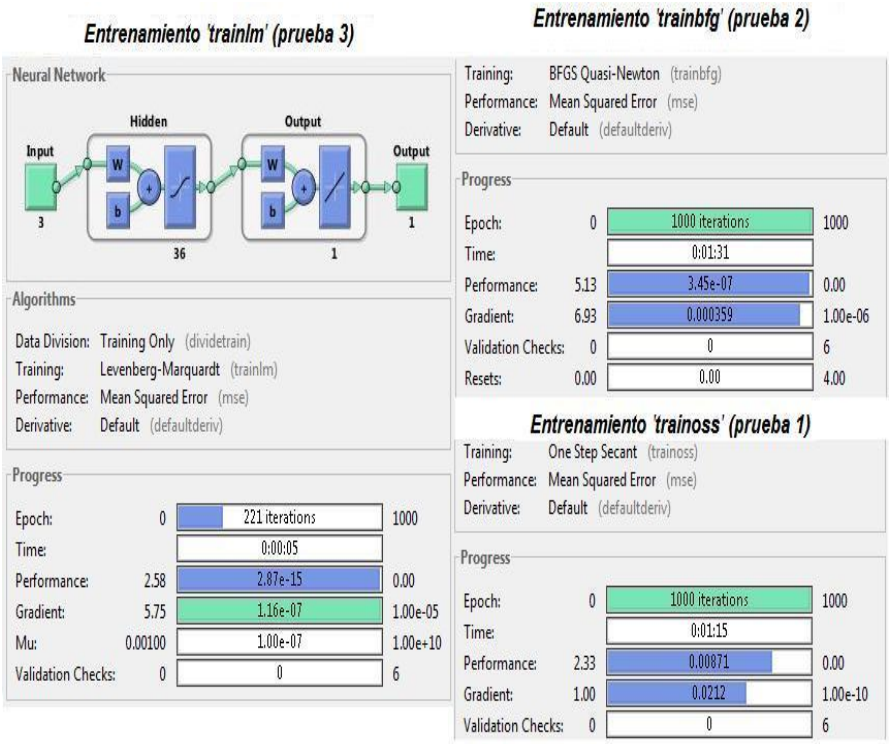
'trainlm' - 'trainbfg' - 'trainoss'

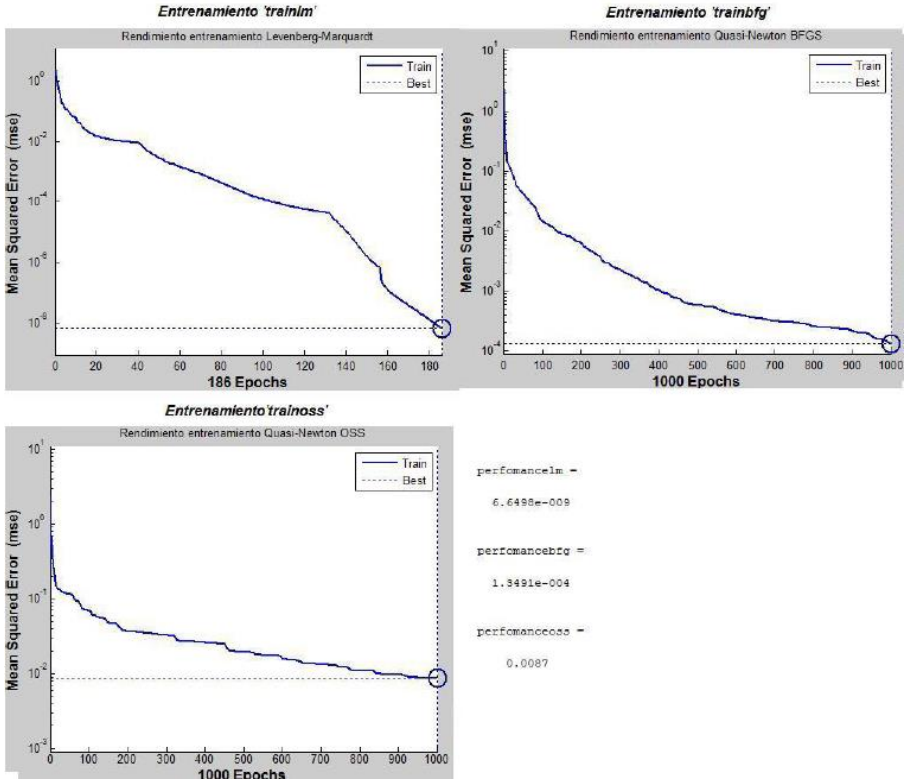
Resultados:

Num.	Prueba	Rendimiento (mse)	Épocas	Parámetro (que corta el entrenamiento)
'trainlm'	1	6.65 e-9	186	Valor mínimo de gradiente *min_grad* encontrado
	2	1.20 e-9	431	Valor mínimo de gradiente *min_grad* encontrado
	3	2.87 e-15	221	Valor mínimo de gradiente *min_grad* encontrado
'trainbfg'	1	1.34 e-4	>1000	exceso de épocas
	2	3.44 e-7	>1000	exceso de épocas
	3	5.89 e-14	755	Valor mínimo de gradiente *min_grad* encontrado
'trainoss'	1	0.0087	>1000	exceso de épocas
	2	0.0096	>1000	exceso de épocas
	3	0.0072	>1000	exceso de épocas

Elección del método de entrenamiento (3)

‘trainlm’ - ‘trainbfg’ - ‘trainoss’



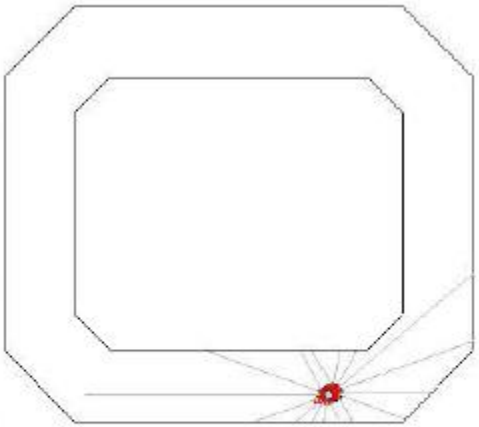


Esquema

Frecuencia de ciclo de control (1)

Velocidad en recta

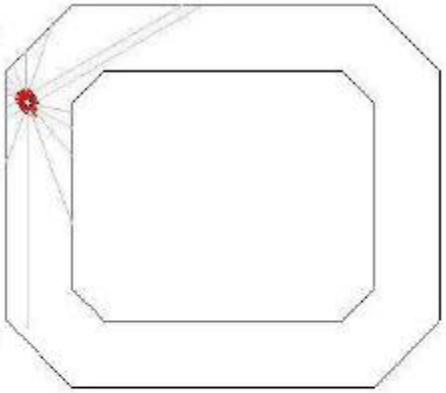
```
>> frente()  >> frente()  >> frente()
>> getVel()  >> getVel()  >> getVel()

ans =        ans =        ans =

   210          181          180

>> getVel()  >> getVel()  >> getVel()

ans =        ans =        ans =

   529          452          432

>> getVel()  >> getVel()  >> getVel()

ans =        ans =        ans =

   290          302          372

>> getVel()  >> getVel()  >> getVel()

ans =        ans =        ans =

     0           29          192
```



$$velPromedio = 264.33\ [mm/seg]$$

Velocidad en giro

```
>> izq()     >> izq()     >> izq()
>> getRotVel()  >> getRotVel()  >> getRotVel()

ans =        ans =        ans =

    27           24           24

>> getRotVel()  >> getRotVel()  >> getRotVel()

ans =        ans =        ans =

    12           13           12

>> der()     >> der()     >> der()
>> getRotVel()  >> getRotVel()  >> getRotVel()

ans =        ans =        ans =

   -27          -24          -27

>> getRotVel()  >> getRotVel()  >> getRotVel()

ans =        ans =        ans =

   -10          -18          -18
```



$$velRotPromedio = 18.83\ [grad/seg]$$

Ref.: Pág. 159

Frecuencia de ciclo de control (2)

Duración de ciclo Movimiento 2

Actividad de ciclo	Duración de actividad
Tiempo máximo de espera del paquete SIP del firmware 0.1 [seg] 3 SIPs esperados	t= 0.3 [seg]
Tiempo necesario para movimiento 2 distancia=350 [mm] velPromedio=264.33[mm/seg]	t= 1.33 [seg]
Tiempo aproximado de envió de paquetes de comando	t= 0.1 [seg]
Tiempo aproximado de un ciclo completo de movimiento 2	T= 1.73 [seg]

Movimientos 1 - 3

Actividad de ciclo	Duración de actividad
Tiempo máximo de espera del paquete SIP del firmware 0.1 [seg] 3 SIPs esperados	t=0.3 [seg]
Tiempo necesario para movimiento 1 – 3 distancia=30 [grad] velPromedio=18.83[grad/seg]	t=1.59 [seg]
Tiempo aproximado de envió de paquetes de comando	t=0.1 [seg]
Tiempo aproximado de un ciclo completo movimientos 1 y 3	T= 1.99 [seg]

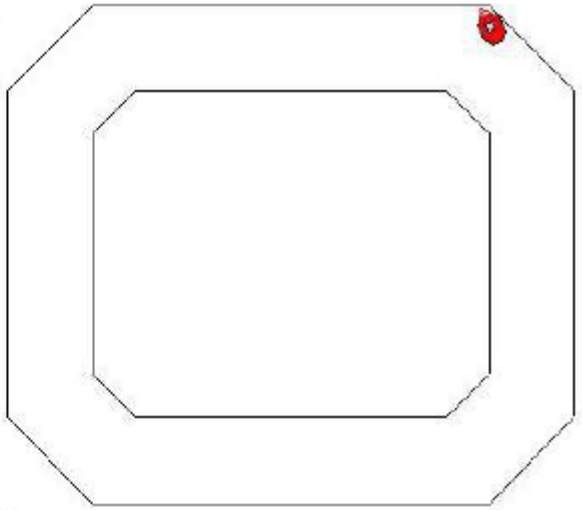
Ref.: Pág. 160

Perdida de datos en ciclo de control

Duración de ciclo	Taza de perdidas	Características
2 [seg]	90%	En 9 de cada 10 ejecuciones del algoritmo de control se pierde datos de sensoramiento, especialmente después del decimo ciclo. (Ejemplo Figura 78)
3 [seg]	<10%	En 1 de cada 10 ejecuciones del algoritmo se pierde datos de sensoramiento, mayormente después del ciclo número 35.

Ejemplo

```
------------------------------

(27) Lectura actual del SONAR

inputs =

    0.6627
    4.7500
    0.5300


output =

     0

Avance al frente
------------------------------
```



Ref.: Pág. 161

Control de Trayectoria

Ciclos ejecutados vs. Números de Patrones

Num. Patrones	Prueba	Ciclos ejecutados	Características
25	1	5	El robot inicia aproximando bien, pero en muy pocos ciclos. Se presentan fallas ya sea por una mala aproximación o también porque las entradas actuales no fueron consideradas en el rango de entrenamiento.
	2	11	
	3	8	
50	1	22	El robot muestra un comportamiento más robusto ante distintas entradas, aunque sigue siendo débil. En algunos casos se presentan fallas por mala aproximación, pero las originadas por entradas no consideradas en entrenamiento se presentan más a menudo.
	2	18	
	3	25	
+90	1	56	El robot consigue darse una vuelta completa en la pista 1, en ambas direcciones. Las fallas por mala aproximación se minimizan, no obstante las de entradas fuera de rango no disminuyen en la misma magnitud.
	2	45	
	3	51	

Ref.: Pág. 162

Agradecimientos