

Design of a platform to test distributed control systems



Miquel Perelló Nieto

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

Directors:

Josep M. Fuertes Armengol

Manel Velasco Garcia

Memòria de Projecte Final de Carrera

Enginyeria Tècnica en Informàtica de Sistemes

Gener 2011

DADES DEL PROJECTE

Títol del Projecte: Design of a platform to test distributed control systems

Nom de l'estudiant: Miquel Perelló Nieto

Titulació: Enginyeria Tècnica en Informàtica de Sistemes

Crèdits: 22,5

Director: Josep M. Fuertes Armengol

Codirector: Manel Velasco Garcia

Departament: ESAII

MEMBRES DEL TRIBUNAL *(nom i signatura)*

1. *Directors:* Josep M. Fuertes Armengol i Manel Velasco Garcia

2. *President:* Pere Marés Martí

3. *Vocal:* Robert Lukas Mario Nieuwenhuis

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Resum

El propòsit d'aquesta memòria és el d'explicar la importància que tenen els Sistemes Distribuïts de Control, i els problemes que sorgeixen al controlar sistemes dinàmics en temps real.

Per tal de mostrar això, des del departament de ESAII s'han dissenyat una sèrie de laboratoris amb els quals els estudiants poden entendre i resoldre els problemes que d'aquest tema se'n deriven.

Amb la realització d'aquest projecte es va procurar donar un pas més amb aquest propòsit i d'aquesta manera es va dissenyar un sistema de monitorització i control sobre les comunicacions entre diferents dispositius que formen llaços de control en el medi compartit d'un bus CAN.

La memòria és el reflex de les tecnologies que es troben en aquests Sistemes i els programes que s'han creat i/o modificat per controlar-los. Entre ells s'en destaca el programa **DCSMonitor**, creat íntegrament en aquest projecte, i dissenyat modularment per tal de servir com a base en qualsevol entorn futur.

Igualment per la part dels dispositius del bus CAN, els quals amb un disseny escalable realitzat en el projecte, poden servir per una infinitud d'objectius nous, i per la realització de projectes enfocats a aquest tipus de sistemes.

Per tant amb la finalització d'aquest projecte, queden obertes moltes possibilitats, i s'espera que tant en l'entorn acadèmic com en l'industrial, es puguin utilitzar les seves idees.

Tot el material emprat en la realització d'aquest projecte pot ser consultat en el repositori <http://code.google.com/p/pfc-platform-test/>

Per la realització de la present memòria ha estat emprat un template de LaTeX el qual es pot trobar en la bibliografia (1)

Dedicat al meu pare, per haver-me deixat llibertat en els estudis,
permetrem decidir equivocar-me i aprendre dels meus errors, i tot i així no
deixar d'ajudar-me en tot moment.

A la meva família, per ensenyar-me cadascun una cosa nova a la seva
manera.

I a la meva mare.

Agraïments

Agraeixo al meu tutor Pep Fuertes que un dia es deixés veure en una conferència sobre grans telescopis, i em captivés des d' aquell instant ja fa algun any. Que el tornés a trobar accidentalment buscant un Projecte Final de Carrera, proposant ampliar el laboratori de Sistemes Distribuïts de Control i que jo acceptés sense saber ven bé que eren, perquè forçosament havia de ser alguna cosa molt interessant. I perquè encara que semblés tenir les coses descontrolades, realment guardés un control absolut de tot.

També haig de donar les gràcies a l'altre tutor Manel Velasco, per poder-me dedicar el temps suficient i just per tirar endavant el projecte; per fer fàcils les coses que semblaven difícils, per ensenyar-me una infinitud de tecnologies desconegudes per mi, per fer-me pensar en algun moment que ell era l'informàtic, per ajudar-me en els moments que estava perdut, i per tranquil·litzar-me quan pensava que no m'en sortiria.

Gracies també al Pau Martí per estar donant suport sense la seva presència física per e-mails i una trucada sorpresa.

Sobretot haig de donar les gracies a la meva noia Virgínia Rodríguez, per obligar-me a aparcar el projecte de tant en tant, i treure'm literalment de casa per descansar. I per comprendre en certs moments que realment de tant en tant s'ha de fer feina.

Gracies també a la Pantera i al Kobu, per fer-me companyia en els moments en els que estava sol en el projecte, i no demanar res a canvi (bé... una mica de pinso i aigua sí...).

*"en definitiva gracies a tothom,
perquè sense ningú res d'això tindria
cap sentit"*

Índex

Índex de figures	vii
Índex de taules	xiii
Índex de codis	xvi
Glossari	xvii
1 Introducció	1
1.1 Estructuració de la memòria	1
1.2 Motivació	4
1.3 Abast	5
1.3.1 Objectius	5
1.4 Planificació	7
2 Tecnologia	15
2.1 Hardware	15
2.1.1 FLEX	16
2.1.2 dsPIC33FJ256MC710	18
2.1.3 MCP2551	18
2.1.4 ICD2/3	19
2.2 Software	20
2.2.1 Eclipse	20
2.2.2 MPLAB® X	21
2.2.3 QtCreator	22
2.2.4 QtLingüist	22
2.2.5 ERIKA Enterprise	22

ÍNDIX

2.3	Protocols de comunicació	23
2.3.1	CAN	23
2.3.2	Serie via RS232	24
2.3.3	Topologia en bus	26
2.4	Conclusions	27
3	Disseny del laboratori	29
3.1	Resum del laboratori actual	29
3.2	Disseny del Nou laboratori	32
3.2.1	Interfície	35
3.2.2	Identificadors dels missatges CAN	38
3.2.2.1	Missatge de control per l' <i>Actuador</i>	42
3.2.2.2	Missatge de l'estat del <i>Sensor</i> per al <i>Controlador</i>	42
3.2.2.3	Missatge de canvi de referència	43
3.2.2.4	Missatge de l'estat del <i>Sensor</i> per al <i>Supervisor</i>	44
3.2.3	Comunicació sèrie	45
3.2.3.1	Senyals de control per monitoritzar	49
3.2.3.2	Senyals de control per consultar llaços	50
3.2.3.3	Senyals de control per generar càrrega	51
3.2.4	Gràfiques en temps real	52
3.2.5	Idiomes	54
3.3	Conclusions	55
4	Implementació del laboratori	57
4.1	DCSMonitor	58
4.1.1	Interfície	59
4.1.2	Comunicació sèrie RS232	62
4.1.2.1	Obrir i tancar dispositiu sèrie	62
4.1.2.2	Enviar petició de llistar llaços de control	66
4.1.2.3	Enviar petició de monitorització	67
4.1.2.4	Enviar percentatge de càrrega	70
4.1.2.5	Recepció de valors monitoritzats	71
4.1.2.6	Recepció de llista de dispositius	76
4.1.3	Generació de gràfiques en temps real	77

4.1.4	Exportar gràfica	79
4.1.5	Mostrant estadístiques	87
4.1.6	Idiomes	89
4.1.6.1	PyQT4	89
4.1.6.2	Pylupdate4	90
4.1.6.3	QtLingüist	91
4.1.6.4	Resultat	91
4.2	dsPIC	93
4.2.1	Tasques en Erika	94
4.2.2	Comunicació serie RS232	96
4.2.2.1	Recepció de senyals de control	96
4.2.2.2	Enviament de dades	99
4.2.3	Comunicació bus CAN	102
4.2.3.1	Creació de les màscares CAN	102
4.2.3.2	Creació dels filtres CAN	103
4.2.3.3	Recepció de missatges CAN	105
4.2.3.4	Pila de llaços de control	108
4.2.3.5	Tractant missatges de control per l' <i>Actuador</i>	110
4.2.3.6	Tractant missatges d'estat del <i>Sensor</i> per al <i>Contro-</i> <i>lador</i>	112
4.2.3.7	Tractant missatges de canvi de referencia	115
4.2.3.8	Tractant missatges d'estat del <i>Sensor</i> per al <i>Supervisor</i>	116
4.2.3.9	Tractant la resta dels missatges	117
4.2.3.10	Monitorització d'un llaç de control	118
4.3	Conclusions	119
5	Anàlisi econòmic	121
5.1	Material	121
5.2	Ma d'obra	122
6	Conclusions	127
6.0.1	Treball futur	129

ÍNDIX

A Guia del laboratori	131
A.1 Preparació de l'entorn de desenvolupament	131
A.1.1 Instal·lació de Mplab	132
A.1.2 Instal·lació d'Eclipse	137
A.1.3 Instal·lació del pluguín <i>RT-Druid</i> per Eclipse	137
A.2 Fer parpellejar un led	142
A.2.1 Entorn Eclipse	143
A.2.2 Entorn MplabX	147
B Guia del Live CD	155
B.1 Requisits mínims	155
B.2 Arrencant el <i>Live CD</i>	156
B.3 Guia pas a pas del Live CD	156
B.3.1 Estructura del <i>Live CD</i>	158
B.3.2 Compilant amb Eclipse el <code>DSPIC_SENSOR</code>	159
B.3.3 Gravant amb MPLAB® X el <code>DSPIC_SENSOR</code>	160
B.3.4 Resum per gravar la resta	167
B.3.5 DCSMonitor	167
B.3.5.1 Connectat a un dispositiu <i>Sensor/Actuador</i>	169
B.3.5.2 Connectat a un dispositiu <i>Monitor</i>	171
B.3.5.3 Exportant gràfica generada	174
B.4 Instal·lant	175
Bibliografia	181

Índex de figures

1.1	Principi del Gantt, esquerre	10
1.2	Principi del Gantt, dreta	11
1.3	Final del Gantt, esquerre	12
1.4	Final del Gantt, dreta	13
2.1	Logotip de la companyia Evidence	16
2.2	FLEX: placa de avaluació de dsPIC de <i>Microchip</i>	17
2.3	Fotografia d'un dsPIC33FJ256MC710	18
2.4	MCP2551	19
2.5	Fotografia d'un programador ICD2	19
2.6	Connector RJ-11	20
2.7	Logotip d'Eclipse	20
2.8	Logotip del pluguín d' <i>Eclipse</i> RT-Druid	21
2.9	Logotip del pluguín d' <i>Eclipse</i> PyDev	21
2.10	Logotip de MPLAB [®] X	21
2.11	Logotip de QtCreator	22
2.12	Logotip de QtLingüíst	22
2.13	Logotip Erika Enterprise	22
2.14	Connector per port serie RS232	25
2.15	Topologia en bus	27
3.1	Microprocessor-based control	30
3.2	Network-based control	30
3.3	Perifèrics per les plaques <i>FLEX</i>	31
3.4	Llaç de control	32

ÍNDIX DE FIGURES

3.5	Entorn del nou laboratori	33
3.6	Diagrama general dels elements del laboratori	34
3.7	Esquema de la interfície del programa DCSMonitor	37
3.8	Codificació de bits dels identificadors CAN per tipus de missatges	40
3.9	Esquema genèric dels identificadors CAN pel nou laboratori.	40
3.10	Diagrama de comunicacions i identificadors CAN	41
3.11	Missatge CAN del <i>Controlador</i> al <i>Actuador</i>	42
3.12	Missatge CAN de l'estat del <i>Sensor</i> per al <i>Controlador</i>	43
3.13	Missatge CAN per actualitzar el valor de referència.	44
3.14	Missatge CAN de l'estat del <i>Sensor</i> per al <i>Supervisor</i>	45
3.15	Trama de supervisió antiga per RS232	45
3.16	Trama de supervisió nova per RS232	46
3.17	Arquitectura bàsica dels missatges enviats per RS232.	47
3.18	Diagrama de comunicacions i missatges RS232.	48
3.19	Missatges de monitorització per RS232.	49
3.20	Missatges per llistar llaços de control per RS232.	51
3.21	Missatges de generació de càrrega al bus CAN per RS232.	52
3.22	Gràfica realitzada amb MATLAB®	53
4.1	Entorn de treball	58
4.2	Interfície final del programa DCSMonitor	61
4.3	Estat de connexió amb el <i>Supervisor</i> via RS232.	65
4.4	Barra lliscant de càrrega en diferents posicions.	71
4.5	Espai de text per valors monitoritzats.	75
4.6	Llista de llaços de control al bus CAN.	77
4.7	Opció de plots per la gràfica.	79
4.8	Finestra per posar nom a la gràfica exportada	80
4.9	Tipus de fitxer als que es pot exportar	81
4.10	Diferents gràfiques generades amb el programa DCSMonitor	86
4.11	Comparació dels valors estadístics.	89
4.12	Utilitzant <i>QtLingüist</i>	91
4.13	Preferències del programa DCSMonitor	92
4.14	Comparació dels labels entre diferents idiomes.	93

5.1	Distribució del preu segons la feina	124
5.2	Percentatge d'hores dedicades al projecte per rol	124
A.1	Pagina oficial MPLABX	133
A.2	Obrint una terminal	133
A.3	Instal·lant MplabX IDE	134
A.4	Termes i condicions MplabX IDE	135
A.5	Directorí d'instal·lació de MplabX	135
A.6	Final instal·lació MplabX IDE	136
A.7	Termes i condicions Compilador MplabX C30	136
A.8	Directorí d'instal·lació compilador C30	137
A.9	Instal·lant nou pluguín en Eclipse	138
A.10	Pantalla per afegir nous pluguíns	138
A.11	Afegint nova adreça de descarrega	139
A.12	Plugins disponibles	140
A.13	Termes i condicions <i>RT-Druid</i>	140
A.14	Confiança en certificat <i>RT-Druid</i>	141
A.15	Reiniciar Eclipse	141
A.16	Path compilador <i>RT-Druid</i>	142
A.17	Nou projecte <i>RT-Druid</i>	143
A.18	Seleccionar tipus de projecte <i>RT-Druid</i>	144
A.19	Posant nom al projecte <i>RT-Druid</i>	144
A.20	Templates <i>RT-Druid</i>	145
A.21	Entorn Eclipse	146
A.22	Desactivar compilació automàtica.	146
A.23	Compilar en Eclipse	147
A.24	Compilació en Eclipse	148
A.25	Nou projecte MplabX	148
A.26	Projecte precompilat MplabX	149
A.27	Carregant fitxer <i>.elf</i> MplabX	149
A.28	Seleccionant dispositiu MplabX	150
A.29	Flex i Mplab ICD3	150
A.30	Programadors MplabX	151

ÍNDIX DE FIGURES

A.31	Posant nom al projecte en MplabX	152
A.32	Treure del directori <i>Debug</i>	152
A.33	Programar microcontrolador des de MplabX	152
A.34	Placa Flex amb programa <i>Blink</i>	153
B.1	Arrencant Live CD	156
B.2	Menu Live CD	157
B.3	Logotip de càrrega d' <i>Ubuntu</i>	157
B.4	Prompt d'usuari i contrasenya	157
B.5	Escriptori del <i>Live CD</i>	158
B.6	Codis comprimits dels programes del laboratori	159
B.7	Indicant espai de treball a <i>Eclipse</i>	160
B.8	Seleccionant RT-Druid project a <i>Eclipse</i>	161
B.9	Seleccionant directori del projecte a <i>Eclipse</i>	161
B.10	Netejant i compilant a <i>Eclipse</i>	162
B.11	Logo carregant MPLAB [®] X	162
B.12	Seleccionant fitxer precompilat MPLAB [®] X	162
B.13	Seleccionant microcontrolador a gravar en MPLAB [®] X	163
B.14	Connexió del programador ICD3 i placa <i>FLEX</i>	163
B.15	Seleccionant programador en MPLAB [®] X	164
B.16	Canviant el directori del projecte a MPLAB [®] X	165
B.17	Directori del projecte ven posat a MPLAB [®] X	165
B.18	Microcontrolador programat correctament a MPLAB [®] X	166
B.19	Director amb el codi de DCSMonitor	168
B.20	Pantalla principal de DCSMonitor	168
B.21	Preferències	169
B.22	Connectant el port serie a DCSMonitor	169
B.23	Programa DCSMonitor connectat a un <i>Sensor/Actuador</i>	170
B.24	Demanant un llistat de dispositius a DCSMonitor	171
B.25	Gràfica amb tots els plots	172
B.26	Gràfica amb plot de referència i segona integral	173
B.27	Barra lliscant per augmentar la càrrega del bus	173
B.28	Posant un nom a la gràfica a exportar	174

B.29 Exemple havent exportat una gràfica	174
B.30 Menú principal LiveCD, instalant	175
B.31 Logotip d' <i>Ubuntu</i> carregant	175
B.32 Preparant particions del disc pel LiveCD	176
B.33 Configurant usuari i contrasenya	177
B.34 Prompt de login	178
B.35 Escriptori del <i>Live CD</i> recent instal·lat	178

ÍNDIX DE FIGURES

Índex de taules

1.1	Estimació de la duració del projecte	7
1.2	Hores de dedicació	9
2.1	Pinatge del connector serie per RS232	25
3.1	Identificadors dels senyals de control enviats al <i>Monitor</i> per RS232.	47
4.1	Màscares per bus CAN	103
4.2	Configuració dels filtres	105
5.1	Preu del material necessari	122
5.2	Preus hora de cada rol	122
5.3	Desglos del preu de la ma d'obra	123
5.4	Hores dedicades per rol	125
A.1	Compatibilitat dels programadors amb MplabX.	151
B.1	Usuari i contrasenya <i>Live CD</i>	157
B.2	Elements de l'escriptori	158
B.3	Elements del directori $\sim$ <i>/workspace/Programs/</i>	159
B.4	Compatibilitat dels programadors amb MplabX.	164

ÍNDIX DE TAULES

Índex de codis

4.1	Fragment del fitxer d'interfície dcsmmainwindow.ui	59
4.2	Creant fitxer d'interfície	60
4.3	Fragment de codi del fitxer dcsmmainwindow.py	60
4.4	Codi per configurar dispositiu sèrie	62
4.5	Disparador de connexió de port serie	62
4.6	Codi per obrir/tancar dispositiu sèrie	63
4.7	Codi per connectar port serie	64
4.8	Disparador de botó llistar dispositius	66
4.9	Codi per demanar una llista de llaços de control	66
4.10	Disparador de botó per monitoritzar	67
4.11	Codi per demanar monitoritzar un llaç de control	68
4.12	Disparador de barra de càrrega	70
4.13	Codi per demanar que saturi el bus CAN	70
4.14	Temporitzador i disparador per rebre dades del port serie	71
4.15	Codi per reactivar l'adquisició de dades	72
4.16	Codi per rebre els valors monitoritzats	73
4.17	Codi per rebre la llista de dispositius	76
4.18	Temporitzador i disparador per dibuixar la gràfica	77
4.19	Codi per actualitzar la gràfica	78
4.20	Disparador per exportar la gràfica	80
4.21	Codi per generar imatge d'una gràfica	81
4.22	Temporitzador i disparador per actualitzar les estadístiques.	87
4.23	Codi per generar i actualitzar les estadístiques.	88
4.24	Exemple de text per traduir	89
4.25	Canviant idioma	90

ÍNDIX DE CODIS

4.26	Crear fitxers per traduccions	90
4.27	Exemple de declaració d'una tasca en el fitxer conf.oil	94
4.28	Codis per interactuar amb tasques	96
4.29	Declaració dels valors dels senyals de control	96
4.30	Interrupció buffer del port sèrie	97
4.31	Alarma de supervisió	100
4.32	Tasca de Supervisió	100
4.33	Funció eCAN1 config	103
4.34	Definicions per la creació de filtres CAN	104
4.35	Funcio activada per una interrupció del bus CAN	106
4.36	Pila d'identificadors	109
4.37	Inicialitzar pila	109
4.38	Comprovar si la pila és plena	109
4.39	Buscar a la pila	110
4.40	Afegir a la pila	110
4.41	Treure de la pila	110
4.42	Captura del missatge de control per l' <i>Actuador</i>	111
4.43	Tasca per capturar el valor de l'actuador	111
4.44	Tasca per aplicar el valor al PWM	112
4.45	Captura del missatge d'estat del Sensor al Controlador	112
4.46	Tasca per capturar el valor d'estat del control	113
4.47	Tasca per fer els calculs del control	113
4.48	Funció per enviar senyal de control per bus CAN	114
4.49	Captura del missatge de canvi de referencia	115
4.50	Captura del missatge d'estat del Sensor al Supervisor	116
4.51	Tasca que emmagatzema els valors d'estat del Sensor	117
4.52	Captura de la resta de missatges	117
4.53	Creant filtres per monitoritzar un llac de control	118
A.1	Instalar Java Runtime Environment en Ubuntu	132
A.2	Comprobant mplab descarregat en Ubuntu	134
A.3	Instalar MPLAB® X en Ubuntu	134
A.4	Instalar compilador mplab 30 en Ubuntu	134
A.5	Instalar Eclipse en Ubuntu	137

Glossari

- .elf** Sigles en Anglès de *Executable and Linkable Format* (Format Enllaçable i Executable); és un format de fitxers per executables, codi obert, biblioteques compartides i bolcat de memòria. Va ser dissenyat per *Unix System Laboratories*, i en principi va ser desenvolupat per plataformes de 32 bits, encara que actualment s'utilitza en varietat de plataformes.
- .qm** Extensió d'un tipus de fitxers que utilitza la llibreria PyQt4 per traduir textos.
- .svg** Sigles en Anglès de *Scalable Vector Graphics* (Gràfics Vectorials Escalables); és una especificació per descriure gràfics vectorials bidimensionals, tant estàtics com dinàmics, en format XML.
- .ts** De l'Anglès *Translation files* (Fitxers de traducció); Tipus d'extensió de fitxers utilitzats per QtLingüist per crear traduccions dels textos d'un programa.
- .ui** Sigles en Anglès de *User Interface* (Interfície d'usuari'); és un format de fitxers creat per la casa QT, i amb informació necessària per muntar la interfície visual d'un programa.
- Actuador** L'actuador en un sistema distribuït de control és l'encarregat de assignar el valor calculat pel controlador a l'entrada del circuit a controlar.
- API** Sigles en Anglès de *Application Programming Interface* (Interfície de Programació d'Aplicacions); és un conjunt de declaracions amb el propòsit de ser usades per un altre programa com una capa d'abstracció.
- CAN** Sigles en Anglès de *Controller Area Network*; És un protocol de comunicacions desenvolupat per la firma alemanya Robert Bosch GmbH, basat en una topologia en bus per la transmissió de missatges en entorns distribuïts.
- Controlador** El controlador en sistema distribuït de control és l'encarregat de calcular el valor a donar al actuador per tal de aconseguir un senyal desitjat.
- DCS** Sigles en Anglès de *Distributed Control System* (Sistema Distribuït de Control); són sistemes de control aplicats generalment en sistemes de fabricació, o qualsevol tipus de sistemes dinàmics, en els quals els elements de control no són centrals sinó que estan distribuïts en el sistema a més cada component del subsistema està connectat als demés mitjançant una xarxa.
- DCSMonitor** Nom del programa creat en aquest projecte per rebre totes les dades monitoritzades presents en el bus CAN. Ve de les sigles en Anglès de *Distributed Control Systems Monitor* (Monitor de Sistemes Distribuïts de Control)

GLOSSARI

- DMA** Sigles en Anglès de *Direct Memory Acces* (Accés Directe a Memòria); és un mètode que permet a certs tipus de circuits integrats accedir a la memòria d'aquests per llegir i escriure independentment de la CPU principal.
- DSP** Sigles en Anglès de *Digital Signal Processor* (Processador de Senyals Digitals); són microprocessadors especialitzats amb una arquitectura optimitzada per el tractament ràpid de senyals digitals.
- dsPIC** Família de microcontroladors fabricats per la casa *Microchip* caracteritzat per comptar amb bus de dades inherent de 16 bits, i per incorporar varies operacions de DSP implementades en hardware.
- dsPIC33FJ256MC710** Model de microcontrolador de la casa *Microchip* caracteritzat per ser de la família dels dsPIC i per comptar amb multitud d'utilitats en sistemes distribuïts.
- IDE** Sigles en Anglès de *Integrated Development Environment* (Entorn de Desenvolupament Integrat); és un programa informàtic compost per un conjunt d'eines de programació en un o varis llenguatges de programació i dedicat a ajudar en les tasques habituals de programació.
- IPS** Sigles en Anglès de *Instructions Per Second* (Instruccions Per Segon); és una mesura de velocitat d'un processador. Indica el nombre d'instruccions que la CPU pot executar en un segon.
- Llaç de Control** En els sistemes distribuïts de control s'anomena d'aquesta manera a un grup de dispositius que formen part del control d'algun proces, en el cas del nostre laboratori aquests grups estan formats per un *Controlador* , un *Sensor* i un *Actuador* .
- Microcontrolador** Es un circuit integrat programable, capaç d'executar les ordres gravades en la seva memòria. Està compost per varis blocs funcionals, els quals compleixen una tasca concreta. Un microcontrolador conté al seu interior les tres unitats funcionals principals de un computador: unitat central de processament, memòria i perifèrics d'entrada i sortida.
- Microprocesador** Els microprocessadors compten amb les funcions d'una CPU (Unitat Central de Procesament) en un o varis circuits integrats.
- Monitor** El monitor es un dispositiu que hem afegit en el nostre laboratori i en el sistema distribuït de control que s'encarrega de monitoritzar el bus del sistema capturant la informació demanada, i poden interferir en el sistema per veure la resposta dels diferents controls.
- Mutex** De l'Anglès *Mutual Exclusion* (Zona d'exclusió mutua); tipus d'algoritmes utilitzats en la programació concurrent per evitar l'ús simultani dels recursos, com variables globals, per fragments de codi coneguts de manera comú com zones crítiques.
- OSEK/VDX** Sigles en Alemà de *Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen* (Sistemes oberts i les seves interfícies per la electrònica en automòbils); és un estandart que especifica el Sistema Operatiu integrat incloent una pila per comunicacions i un protocol per l'administració de xarxes per sistemes empotrats en automòbils.
- PIC** Sigles en Anglès de *Peripheral Interface Controller* (Controlador d'Interfície de Perifèric); són una família de microcontroladors de tipus RISC fabricats per *Microchip Technology Inc.* originalment creats per la divisió de microelectrónica de General Instruments.

PWM	Sigles en Anglès de <i>Pulse-Width Modulation</i> (Modulació per amplària d'impuls); aquesta és una tècnica en la qual es modifica el cicle de treball d'un senyal periòdic (una ona sinusoidal o ona quadrada, per exemple), ja sigui per transmetre informació a través d'un canal de comunicacions o per controlar la quantitat d'energia que s'envia a una càrrega.
RTOS	Sigles en Anglès de <i>Real-Time Operating System</i> (Sistema Operatiu en Temps Real); es diu dels Sistemes Operatius dissenyats per atendre aplicacions en temps real. La principal característica d'aquests SO és el nivell de precisió en els temps d'execució de cada una de les tasques que ha de realitzar.
SAE	Sigles en Anglès de <i>Society of Automotive Engineers</i> (Societat d'Enginyers Automotrius); és l'organització enfocada a la mobilitat dels professionals en l'enginyeria aeroespacial, automoció, i indústries comercials especialitzades en la construcció de vehicles.
Sensor	El sensor en un sistema distribuït de control és l'encarregat de prendre els valors de sortida del sistema, per tal de oferir-li al controlador.
Supervisor	El supervisor en un sistema distribuït de control és el dispositiu que ens dona els valors necessaris del sistema per saber el seu estat.
WBS	Sigles en Anglès de <i>Work Breakdown Structure</i> (Estructura de Descomposició de Treball); en gestió de projectes es una descomposició jeràrquica orientada al entregable, del treball a ser realitzat per l'equip del projecte per complir amb els objectius d'aquest.
XML	De l'Anglès <i>eXtensible Markup Language</i> (Llenguatge de Marques Extensibles); és un metallenguatge d'etiquetes, desenvolupat per el <i>World Wide Web Consortium (W3C)</i> que permet definir la gramàtica de llenguatges específics.

GLOSSARI

1

Introducció

Aquesta memòria pretén reflectir de manera clara i entenedora el procés que s’ha seguit per dur a terme un laboratori de ”Sistemes Distribuïts de Control” (a partir d’ara SDC), així com els problemes que hagin pogut sorgir en el transcurs del projecte.

La realització d’aquest laboratori ha comportat l’adquisició d’una visió profunda del que es pretén ensenyar des del departament de ESAII de la UPC en el tema dels SDC .

Per tant en el transcurs d’aquesta memòria s’anirà endinsant en aquest tipus de sistemes, es coneixeran quines són algunes de les tecnologies base que els conformen, els problemes que en elles esdevenen, i solucions a algunes d’aquestes qüestions.

Amb tot això es podrà tenir una idea més clara del motiu de la realització d’aquest projecte i les portes que a partir d’aquest projecte queden obertes.

A més es poden veure com a apèndix les dues guies realitzades en aquest projecte, una per la preparació de l’entorn de desenvolupament i posterior utilització del laboratori. I una altre per l’ús del LiveCD pel desenvolupament pràctic del laboratori.

1.1 Estructuració de la memòria

En aquesta secció mirarem de donar una visió global de la memòria, perquè sigui fàcil de ser consultada, i d’aquesta manera tenir en tot moment una idea de on es pot trobar qualsevol qüestió. Per tant començar indicant que aquesta memòria ha estat escrita seguint l’ordre cronològic que esdevé al desenvolupar i portar a terme un projecte.

1. INTRODUCCIÓ

D'aquesta manera primerament ens trobem amb el capítol de Introducció (capítol 1), en el qual podem veure resumidament de que tracta el projecte, quines són les raons de la seva elecció i perquè aquest projecte és important (secció Motivació 1), quin era l'objectiu d'aquest projecte amb les limitacions del temps que tenia (secció Abast 1.3), i de quina manera es va organitzar el temps per realitzar-lo, i reorganitzar-lo en el moment que això va caldre (secció Planificació 1.4).

Un cop es té una idea global ben clara dels objectius d'aquest projecte, és el moment d'aprofundir una mica en les tecnologies que en aquest es toquen, per tant tot seguit veurem el capítol Tecnologia (capítol 2). Aquest capítol l'hem dividit en tres parts per tipus de tecnologia.

En la primera part parlarem sobre els dispositius físics (secció Hardware 2.1) en la que explicarem quines plaques s'utilitzen al laboratori per fer els controls (secció FLEX 2.1.1), tot seguit explicarem quin es el nucli d'aquestes plaques (secció dsPIC33FJ256MC710 2.1.2), quin integrat ens permet fer d'intermediari per les comunicacions CAN (secció MCP2551 2.1.3), i amb quin dispositiu podem programar qualsevol microcontrolador de la casa Microchip (secció ICD2/3 2.1.4).

En la segona part parlarem del software que s'ha usat per elaborar el projecte i el laboratori (secció Software 2.2), en ella parlarem del IDE que hem utilitzat per programar tant el codi en python del programa **DCSMonitor** com el codi del Sistema Operatiu en Temps Real Erika (secció *Eclipse* 2.2.1), el programa que hem utilitzat per programar els dispositius (secció MPLAB[®] X 2.2.2), del IDE que hem usat per crear les interfícies del programa **DCSMonitor** (secció QtCreator 2.2.3), de la eina que ens facilita crear els fitxers de traducció del programa **DCSMonitor** (secció QtLingüist 2.2.4), i per ultim el Sistema Operatiu en Temps Real que formen el cervell dels dispositius del bus CAN (secció ERIKA Enterprise 2.2.5).

Per ultim farem una pinzellada a alguns protocols que hem d'entendre per poder ser conscients dels problemes i solucions que atorga cadascun (secció Protocols de comunicació 2.3), així que comencem per el més important en els Sistemes Distribuïts de Control amb el qual tots els dispositius es troben ben comunicats (secció CAN ref-cap:tec:prot:can), tot seguit el que ens permet rebre a l'ordinador tot el que està succeint al bus CAN (secció Serie via RS232 2.3.2), i per finalitzar el tipus de topologia en el que es troben tots els dispositius del laboratori (secció Topologia en bus 2.3.3)

A partir d'aquest punt comença a aparèixer les noves idees que aporta el projecte, en el capítol Disseny del laboratori (capítol 3) primerament s'explicarà que s'ensenya actualment en el laboratori, i quines són les eines que s'utilitzen (secció Resum del laboratori actual ,3.1). Tot seguit amb la secció Disseny del Nou laboratori ,3.2, es plantegen quins són els problemes que es volen resoldre, i com s'aconsegueix en alguns temes del laboratori com són la creació de la interfície (secció 3.2.1), els problemes amb els identificadors que s'han d'assignar als diferents dispositius del bus CAN (secció 3.2.2), la comunicació entre el programa **DCSMonitor** (programa que corre a l'ordinador) i els diferents dispositius mitjançant comunicació serie per RS232 (secció 3.2.3), el disseny que es va pensar pel mostreig de les gràfiques en temps real (secció 3.2.4) i el mètode escollit per poder fer el programa multilingüe i fàcilment ampliable (secció 3.2.5).

Per altre banda tenim el *com* es va portar a terme el sistema dissenyat anteriorment, d'aquesta manera en el capítol Implementació del laboratori (capítol 4) toca el torn d'explicar la metodologia per fer realitat el laboratori plantejat. Primerament ens centrarem en el programa d'ordinador **DCSMonitor** (secció 4.1) començant per la part visual del programa (la interfície en la secció 4.1.1), seguint amb la implementació de la comunicació sèrie (secció 4.1.2). Després una de les parts més importants que és la generació de les gràfiques en temps real (secció 4.1.3); gracies a les quals podem avaluar la bondat dels controls que hi ha al bus CAN, l'exportació d'aquestes gràfiques en diferents formats (secció 4.1.4) i finalment la integració de la multilingüitat, o creació de la interfície multilingüe (secció 4.1.6). La part dels dispositius no és menys important, i per tant té també la seva secció d'implementació (4.2), en ella s'expliquen els mètodes que s'han usat per portar a terme les comunicacions series per RS232 (secció 4.2.2), i tot el relacionat amb les comunicacions CAN (secció 4.2.3).

Al acabar el projecte, ha estat possible comprovar les hores reals que si han dedicat, i per tant fer càlculs dels preus del material i de la ma d'obra. Per tant tot seguit ve el capítol Anàlisi econòmic 5, on es poden veure un desglos del preu del projecte.

Després venen les conclusions ,6, on es deixa constància de quins han estat els objectius aconseguits, possibles ampliacions que pot tindre aquest projecte, i una visió general després d'haver realitzat el projecte.

Com apèndix tenim les dues guies que s'han elaborat per l'execució del laboratori en un sistema Linux (*Ubuntu*) A, i una guia d'us del *Live CD* creat en aquest projecte,

1. INTRODUCCIÓ

per tenir en marxa tot aquest sistema en 6 minuts (més larrancada de l'ordinador) B.

1.2 Motivació

Moltes són les raons per les quals aquest projecte va ser començat i posteriorment tirat endavant. Primerament perquè els sistemes electrònics empotrats són a tot arreu; caixers automàtics, impressores, calculadores, rellotges, equips mèdics, telèfons mòbils... Però què és un sistema si no pot ser consultat en temps real? De que serveix un sistema electrònic aïllat del que no es pot treure cap tipus d'informació a l'instant? Actualment quasi tots els sistemes procuren d'una o altre manera estar en constant comunicació, volem saber en cada moment quin és l'estat d'alguna lectura, si hi ha algun problema en algun aparell que hagi de ser revisat, si el ABS del cotxe funcionarà en el moment que toqui, si el cafè de després de les postres ja és ben calent...

Per tant l'aparició dels sistemes distribuïts es veu forçada a sorgir de les mancances que tenien els sistemes totalment autònoms, i així esdevenir poc a poc una realitat actual. Però aquesta aparició no sorgeix i esdevé una realitat idíl·lica en el moment de néixer, sinó que junt amb ella apareixen nous problemes e inconvenients; la concurrència de tots els sistemes comunicats és un problema greu, i no tots els sistemes són igual de crítics; n'hi ha que només donen lectures periòdiques sense massa importància, però també n'hi ha que salven vides.

Un clar exemple de tot això és el protocol de comunicació CAN, desenvolupat fa només 3 dècades; aquest va començar a l'any 1983 per *Robert Bosch GmbH* i va ser oficialment alliberat al 1986 per la Societat d'Enginyers Automotrius (SAE, Society of Automotive Engineers). Va ser principalment creat per la indústria automòbil, gracies a la qual es van poder posar molts dels sistemes de seguretat que hi ha actualment als cotxes; gracies als quals es salven moltes vides; però que actualment és usat en molts altres sectors, com poden ser equips mèdics, controls industrials, entorns aeroespacials, marítims, etc.

Aquesta és una de les raons per les quals s'ha escollit la realització d'aquest projecte. Perquè són sistemes reals, i que encara no estan perfeccionats; i això és així no perquè no siguin importants, sinó perquè pertanyen a una branca de la tecnologia que encara està en constant evolució.

Per tant, espero que es vegi la importància que aquest projecte porta darrera, i que la realització d'aquest no tanca cap porta, sinó que deixa obert un camí molt llarg per explorar.

1.3 Abast

Aquest projecte pot ser continuat, per tant cal indicar que amb la limitació de temps amb la que conta el projecte (relativa al nombre de crèdits), en algun moment s'ha hagut de posar una fita. Per tant s'exposa en aquest apartat els objectius que s'han proposat aconseguir.

1.3.1 Objectius

Abans d'escollir aquest projecte es van plantejar alguns dels objectius que aquest hauria de complir, però un cop començat es van anar refinant i ampliant fins arribar a generar una idea més específica d'aquests. Per tant tot seguit es mostren els objectius que finalment es van decidir:

1. Adaptar codi existent en el “Laboratori de Sistemes Distribuïts de Control” del màster en “Automàtica i Robòtica” del departament del ESAII per poder-lo realitzar completament amb Software Lliure.
2. Dissenyar un laboratori de Sistemes Distribuïts de Control executable amb Software Lliure. Aquest laboratori ha de poder comprovar el bon funcionament del treball dels diferents alumnes, i per fer aquesta tasca s'han de crear varis programes:
 - (a) Programa per ordinador amb dos modes d'execució (**DCSMonitor**).
 - i. Mode Monitor.
 - S'ha de poder connectar amb una placa Flex monitora via RS232.
 - Ha de poder llistar els diferents llaços de control que hi hagi al bus CAN al que estigui connectat la placa Flex monitora.
 - Ha de ser capaç de rebre tota la informació necessària per avaluar la qualitat dels diferents algoritmes de control que estiguin executant els alumnes.

1. INTRODUCCIÓ

- Capaç de dibuixar les gràfiques en temps real del control que estiguin executant els diferents llaços.
 - Ha de poder generar càrrega al bus CAN per observar els problemes que apareixen quan un bus amb aquest protocol es satura.
- ii. Mode Sensor/Actuador.
- Ha de ser capaç de rebre la informació del *Sensor/Actuador* que envia l'estat de les seves senyals.
 - Ha de poder generar la gràfica en temps real del control que s'està efectuant.
- iii. Opcions compartides.
- Multilingüe.
 - Poder afegir o eliminar de la gràfica alguna de les línies (referència, valor d'entrada, primera i/o segona integral) en temps real, o un cop la imatge fos aturada.
 - Poder generar una imatge amb varies opcions:
 - En varis formats.
 - Autocontinguda amb tota la informació necessària (títol del laboratori, subtítol, llegenda).
 - Amb els plots que es desitgin (referència, valor d'entrada, primera i/o segona integral).
 - Amb tots els textos de la gràfica multilingüe.
- (b) Programa monitor per un microcontrolador.
- Ha de poder guardar quins dispositius hi ha al bus CAN.
 - Ha de ser capaç d'interferir en les comunicacions del bus CAN.
 - Ha de poder capturar l'estat d'un llaç de control.
 - Ha de poder enviar tota aquesta informació al programa d'ordinador mitjançant comunicació sèrie per RS232.
 - Ha de poder rebre ordres del programa d'ordinador mitjançant la comunicació sèrie per RS232.'
- (c) Programa Controlador per un microcontrolador.
- Ha de ser capaç de rebre l'estat del Sensor.

- Ha de calcular el senyal a aplicar al doble integrador tenint en compte la discretització dels temps.
 - Ha d'enviar el senyal d'entrada del doble integrador l'*Actuador*.
- (d) Programa *Sensor/Actuador* per un microcontrolador.
- Ha de ser capaç de llegir l'estat de la primera i segona integral.
 - Ha d'enviar els valors de la primera i segona integral al Controlador.
 - Ha de rebre el senyal a aplicar del controlador.
 - Ha de poder enviar tota aquesta informació al programa d'ordinador mitjançant comunicació sèrie per RS232.
3. Implementar el laboratori mencionat al punt anterior.
 4. Realitzar la documentació necessària per tenir l'entorn de desenvolupament i posata en marxa de tot el sistema multilingüe.
 5. Preparar l'entorn Live CD amb tot el material necessari per realitzar l'execució del laboratori.
 6. Preparar una guia d'ús multilingüe del Live CD.

1.4 Planificació

La planificació del projecte va ser en un principi una aproximació de la feina que s'havia de realitzar, ajustada al temps que oferien el nombre de crèdits a realitzar, però en aquesta estimació no hi havia inconvenients temporals i per tant era un esquema totalment idíl·lic.

crèdits	crèdit/hora	hores	hores/dia	dies	mesos
22,5	20	450	5	90	4.5

Taula 1.1: Estimació de la duració del projecte

D'aquesta manera, al iniciar el projecte el 20 de Juny i havent realitzat els càlculs del temps a dedicar (taula 1.1) es preveia que al voltant de Novembre el projecte ja estaria finalitzat, i es podria procedir a fer la lectura d'aquest. Un cop posada la fita i les hores a dedicar, es va assignar les hores a les diferents tasques que s'esperava realitzar, per d'aquesta manera veure si el projecte s'ajustava correctament a aquestes

1. INTRODUCCIÓ

hores (taula 1.2). I tenint en compte la taula de hores diàries vista anteriorment, ens surten els dies i hores que aquí apareixen. Així doncs es va ajustar la feina al calendari i efectivament tot encaixava, però a finals d'Agost per problemes personals el projecte es va haver d'aturar completament, i això es va allargar un mes i mig. Això va causar una replanificació del diagrama de gantt i l'entrega del projecte es va veure afectada en una demora equivalent a aquest temps (es pot observar en el diagrama de gantt (Principi del gantt 1.1 i 1.2) el període de demora esmentat).

WBS	Nom	Temps
1	Estudi General	3d 4h
1.1	Buscar informació RTOS	1d 4h
1.2	Mirar Erika RTOS	1d
1.3	Mirar Comunicació CAN bus	1d
2	Planificar Projecte	4h
2.1	Planificació general del projecte	4h
3	Laboratori Actual	28d 4h
3.1	Mirar material necessari	7h
3.2	Preparar i adaptar entorn de treball	2d 4h
3.3	Practica Encendre Led	1d
3.4	Practica Doble Integrador	2d 5h
3.5	Practica Controlador - Actuador	2d 2h
3.6	Recopilar informació per la FAQ	9d 4h
3.7	Recopilar informació sobre els passos seguits	9d 4h
4	Preparar Codis Lliures	6d 2h
4.1	Estudiar la comunicació per RS232 amb PIC	1d
4.2	Mirar comunicació per RS232 amb python	2d 2h
4.3	Mirar llibreries per fer gràfiques amb python	2d 7h
5	Realitzar Ampliació del Laboratori	26d 6h
5.1	Dissenyar primera aproximació ampliació	1d
5.2	Implementació primera aproximació	4d
5.2.1	Implementar part Sampling, Actuation	1d 4h
5.2.2	Implementar part Control, Monitor	1d
5.2.3	Implementar part Receptor (PC)	1d 4h
5.3	Replanificar i reorganitzar	1d 1h
5.4	Redissenyar ampliació	5d 6h
5.4.1	Pensar resultat esperat	1d 5h
5.4.2	Dissenyar parts implicades	2d
5.4.3	Especificar protocols de comunicació	2d
5.5	Implementació	12d 5h

5.5.1	Implementar part Sampling, Actuation	7h
5.5.2	Implementar part Control	1d
5.5.3	Implementar part Monitor	3d 6h
5.5.4	Implementar part Receptor	3d 5h
5.5.4.1	Disseny de la part visual	1d 1h
5.5.4.2	Integració amb comunicació	1d
5.5.4.3	Mostreig de resultats en temps real	1d 4h
5.5.5	Realitzar tests	2d
5.5.6	Depuració de codi	1d 2h
5.6	Documentar nou laboratori	1d 1h
5.7	Traduir al Anglès nou laboratori	1d
6	Preparar Live CD	7d
6.1	Estudi de la creació	2d
6.2	Creació d'entorn	1d 4h
6.3	Implementar Live CD	7h
6.4	Revisar bon funcionament	1d 4h
6.5	Documentar ús del CD	1d
7	Realitzar Guia Laboratori	4d 6h
7.1	Realitzar guia Català	3d 6h
7.1.1	Instal·lació entorn	7h
7.1.1.1	Linux (ubuntu)	7h
7.1.2	Configuració entorn	6h
7.1.2.1	Linux (ubuntu)	6h
7.1.3	Realitzar la guia de seguiment del laboratori	2d
7.2	Traduir guia al Anglès	1d
8	Preparar Informe previ del projecte	3d 4h
9	Entrega Informe previ del projecte	
10	Preparar Memòria	28d
11	Preparar Defensa	3d 4h
12	Reunions periòdiques	78d 3h
13	Defensa del projecte	

Taula 1.2: Hores de dedicació

1. INTRODUCCIÓ

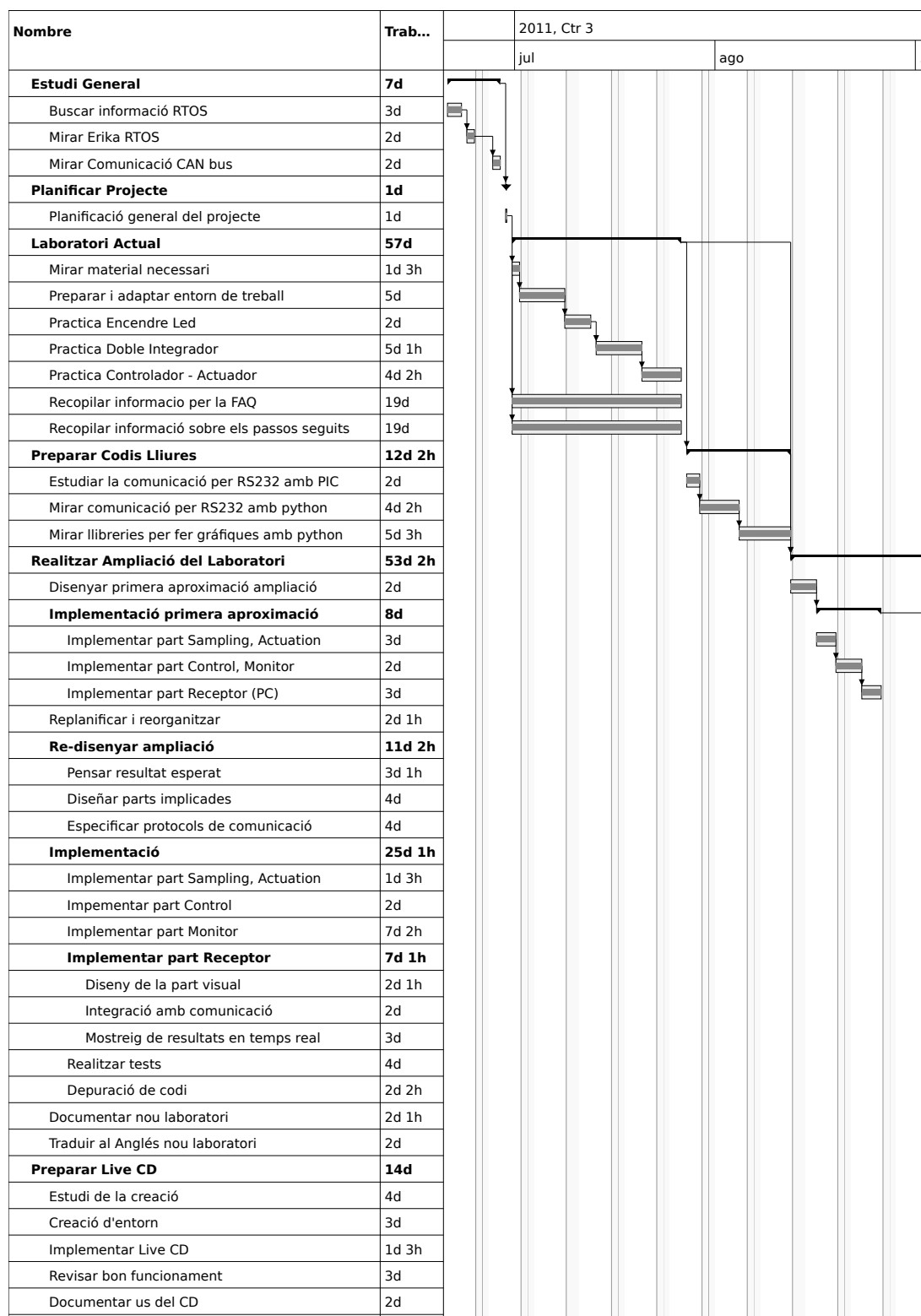


Figura 1.1: Principi del Gantt, esquerre : Juny, Juliol, Agost

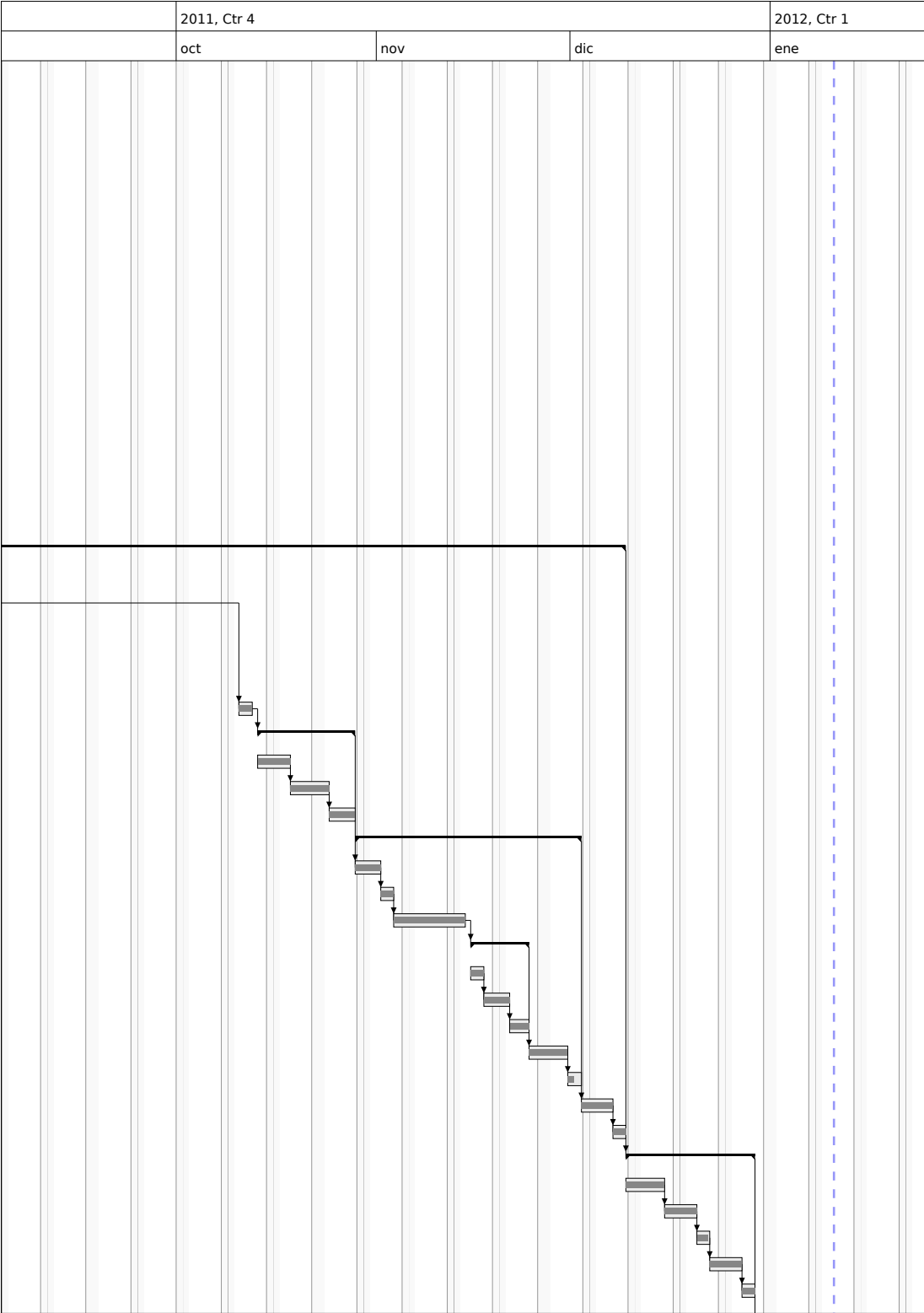


Figura 1.2: Principi del Gantt, dreta : Septembre, Octubre, Novembre, Desembre, Gener

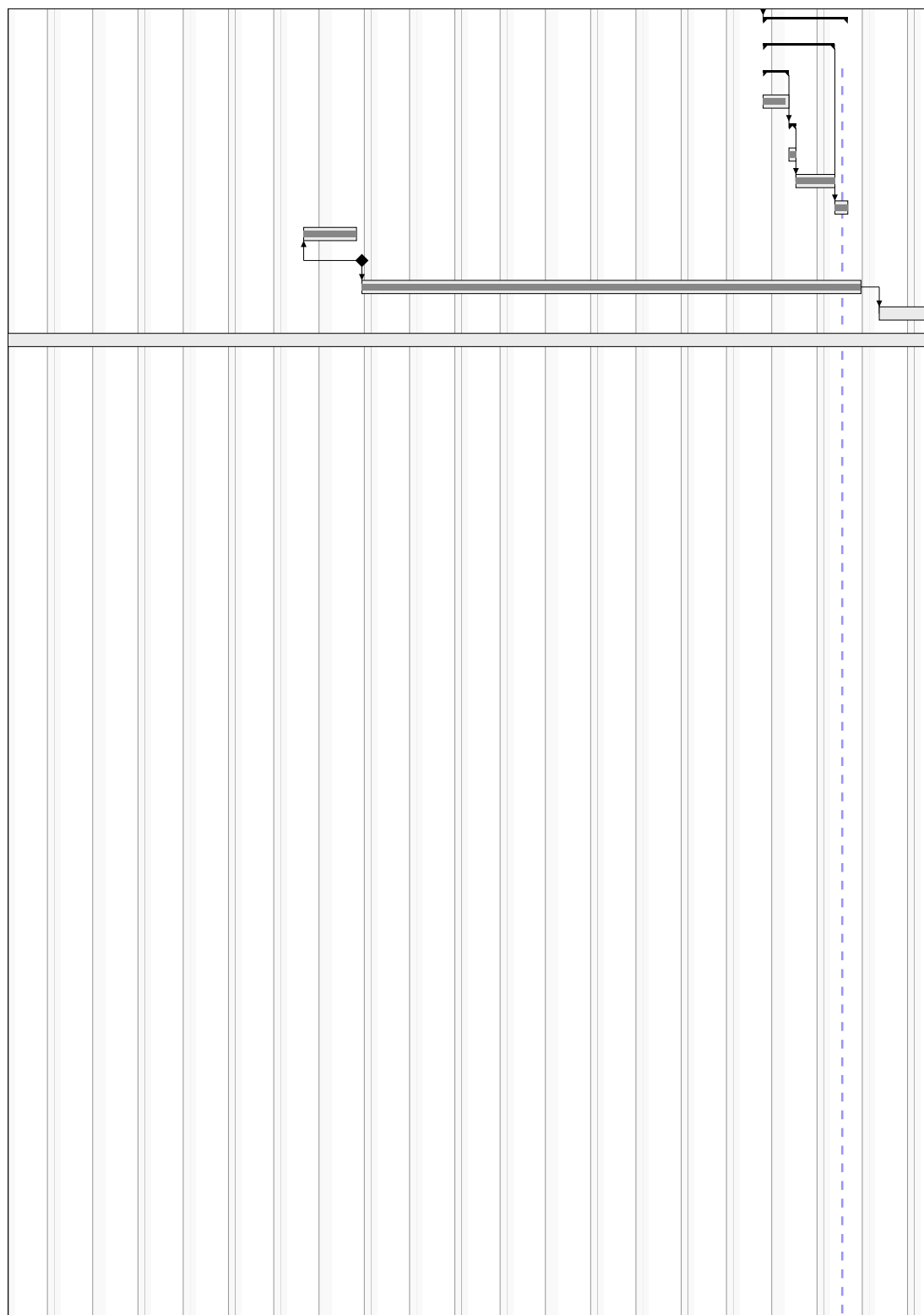


Figura 1.4: Final del Gantt, dreta : Septiembre, Octubre, Noviembre, Diciembre, Gener

1. INTRODUCCIÓ

2

Tecnologia

Una part molt important del projecte, ha estat familiaritzar-se amb una serie de tecnologies, materials i protocols que en certs casos eren totalment desconeguts. Aquestes tecnologies han comportat hores de dedicació i adaptació i per aquesta mateixa raó s'ha creat aquesta secció. Això permetrà que un lector que no conegui alguna d'aquestes tecnologies pugui de manera rapida fer-se una idea del seu significat o propòsit.

En el capítol Tecnologia per tant s'intentarà donar una visió del tot el material que s'ha usat en el laboratori, per a que serveix cada un dels diferents dispositius i quines característiques tenen (tot això en la secció de Hardware 2.1).

Tot seguit s'explicarà quins són els programes que es poden utilitzar per compilar els diferents codis, els programes que ens ajuden a programar els dispositius que formen els llaços de control, els programes que ens han permès crear les interfícies, o traduir-les còmodament en el programa **DCSMonitor** (tot això en la secció Software 2.2).

I perquè es pugui entendre una mica millor tot el projecte, es dona una visió general sobre els diferents protocols que durant la memòria es veuran, i que han estat necessaris per poder realitzar els controls del laboratori (secció de protocols de comunicació 2.3).

Per tant aquesta secció es fa una eina bàsica necessària per tothom que vulgui comprendre una mica tot això, o per algú que vulgui repassar-ho.

2.1 Hardware

Com hem dit en la introducció d'aquest capítol, en aquesta secció es donarà una visió del material que s'ha utilitzat en l'elaboració del laboratori de Sistemes Distribuïts de

2. TECNOLOGIA

Control. Entre aquest material n'hi ha que està ja preparat per el seu ús directe com la placa *FLEX* (secció 2.1.1) i el programador ICD2 (secció 2.1.4). O en canvi necessita ser ensamblat en un protoboard o crear una placa per poder utilitzar-los. En el nostre cas el microcontrolador dsPIC (secció 2.1.2) ja forma part de la placa *FLEX*, i per utilitzar el transceptor (secció 2.1.3), si que s'ha hagut de crear una petita plaqueta que s'explicarà en un altre capítol (secció 3.1).

2.1.1 FLEX



Figura 2.1: Logotip de la companyia Evidence -

FLEX és el nom amb el que han designat a una placa de prototipatge creada per la casa Evidence (logotip 2.1) per treure el màxim partit d'un microcontrolador amb tecnologia dsPIC.

Aquesta placa va néixer amb l'objectiu de desenvolupar aplicacions en temps real, i per aquest motiu compte amb moltes qualitats interessants per el nostre laboratori.

Entre les seves característiques es troben:

- Un disseny de la electrònica robust.
- Una arquitectura modular.
- Disponibilitat d'un gran nombre creixent de guies d'aplicacions.
- Tot el suport del kernel Erika Enterprise, de Evidence.

El cor d'aquesta placa es compon d'un dsPIC33FJ256MC710 (veure secció 2.1.2), el qual ens dona moltes possibilitats a l'hora de crear diferents dispositius.



Figura 2.2: FLEX: placa de avaluació de dsPIC de *Microchip* -

2. TECNOLOGIA

2.1.2 dsPIC33FJ256MC710



Figura 2.3: Fotografia d'un dsPIC33FJ256MC710 -

Aquest és un microcontrolador de la casa *Microchip* Technology (fabricant de microcontroladors, memòries i semiconductors analògics, situat en Chandler, Arizona, EE.UU.), destinat a Sistemes Digitals de Control. Per aquesta raó aquest microcontrolador compta amb una multitud de perifèrics de comunicació, entre els quals es troba el CAN. Això fa que la casa Evidence l'hagi escollit per formar part del nucli de la placa de desenvolupament *FLEX* (explicada en la secció 2.1.1), i que s'hagi adaptat el Sistema Operatiu en Temps Real Erika Enterprise perquè pugui ser instal·lat en ell.

A part del mencionat, aquest microcontrolador compta amb característiques molt interessants, entre d'elles es poden destacar les següents (informació extreta del datasheet (2)):

- Una arquitectura de 16 bits
- CPU a velocitat de 40 MIPS (Mega Instructions Per Second)
- Memòria per programa de 256 KB de tipus Flash
- Memòria RAM de 30,720 Bytes
- 85 pins d'entrada/sortida
- Perifèrics de comunicació UART, SPI, I2C i CAN
- PWM amb resolució màxima de 16 bits
- 8 canals DMA hardware

2.1.3 MCP2551

El semiconductor MCP2551 és un transceptor CAN d'alta velocitat, i un dispositiu tolerant a fallades que serveix d'intermediari entre el controlador CAN i el bus físic. Aquest pot arribar a treballar a velocitats de 1 Mb/s, i en un bus CAN es poden arribar a connectar fins a 112 nodes amb aquest transceptor (datasheet (3)).

Per tant en el nostre laboratori, cada un dels diferents dispositius (*Monitor*, *Sensor/Actuador* i *Controlador*), necessàriament han de comptar amb aquest semiconductor, i s'han creat uns petits circuits per connectar ràpidament en cadascuna de les plaques *FLEX* (veure laboratori actual 3.1).

2.1.4 ICD2/3

Aquest aparell (figura 2.5) és un programador/debugador d'errors, fabricat per la casa *Microchip*, que juntament amb el programa MPLAB® X pot realitzar aquestes funcions. Aquest compta amb una connexió per USB per connectar-lo al ordinador, i un cable amb dos extrems RJ-11 per connectar entre ell i el dispositiu a programar (figura 2.6, connector típicament utilitzat en els cables telefònics a Espanya).

Mentre que fins aquest moment s'utilitzava el programador ICD2 actualment ha sortit al mercat el ICD3. Això en principi no és cap problema ja que l'antic programador segueix funcionant correctament, però així com el programa MPLAB® X en les primeres versions betes estava procurant donar suport al primer, a partir de la versió 0.17 ha deixat de ser així, obligant d'alguna manera a deixar enrere aquest programadors; els quals no són precisament barats.

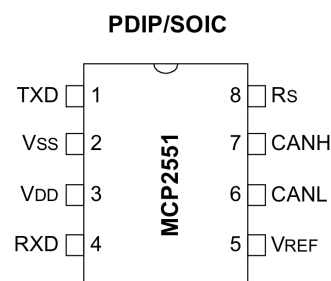


Figura 2.4: MCP2551 -



Figura 2.5: Fotografia d'un programador ICD2



Figura 2.6: Connector RJ-11 -

2.2 Software

Durant el projecte s'han emprat multitud de programes que eren necessaris per poder compilar els codis, gravar els microcontroladors, o preparar l'entorn multilingüe del programa **DCSMonitor** .

En aquesta secció s'explicarà quins són aquests programes, i es farà un resum del seu objectiu.

2.2.1 Eclipse



Figura 2.7: Logotip d'Eclipse -

Aquest software ens ofereix un entorn de desenvolupament integrat (també coneguts com IDE's) multi-plataforma de codi obert. Aquesta plataforma ha estat típicament utilitzada per desenvolupar IDE's. Típicament és usat el IDE de Java *Java Development toolkit* (JDT) el qual ve integrat per defecte amb el propi *Eclipse* .

Entre aquests IDE's desenvolupats per *Eclipse* està el RT-Druid (figura 2.8) creat per la casa Evidence, amb el qual és possible programar de manera facil el Sistema Operatiu en Temps Real Erika, que és el codi base sobre el que treballem en el laboratori per programar les plaques *FLEX* .

A part d'utilitzar aquest IDE per programar els dispositius, també existeix el pluguin PyDev per *Eclipse* , que també ens ofereix un IDE per programar en *Python* , i

gracies al qual hem pogut programar més fàcilment el codi del programa **DCSMonitor**.



Figura 2.8: Logotip del pluguín d'*Eclipse* RT-Druid -



Figura 2.9: Logotip del pluguín d'*Eclipse* PyDev -

2.2.2 MPLAB[®] X

Mplab és un editor IDE gratuït desenvolupat per la casa Microchip i destinat a la programació dels seus productes. Fins aquest moment era exclusivament per Windows, però amb la nova aparició de MPLAB[®] X programat en Java l'han convertit en multiplataforma. Encara que les versions actuals encara estan en beta (això vol dir que algunes de les seves funcionalitats encara poden ser inestables o no estar implementades).



Figura 2.10: Logotip de MPLAB[®] X - Publicitat de la pàgina oficial de Microchip

Encara que sigui un editor IDE, en el nostre laboratori no l'utilitzem per aquesta tasca, ja que això ens ho fa l'entorn comentat anteriorment RT-Druid (veure secció d'*Eclipse* 2.2.1). Per tant té una altra funcionalitat que és la de programar mitjançant el dispositiu ICD2 o altres els microcontroladors; en el nostre cas les plaques *FLEX*.

2. TECNOLOGIA

2.2.3 QtCreator



Figura 2.11: Logotip de QtCreator -

QtCreator és un altre IDE per desenvolupar aplicacions d'escriptori multiplataforma (tant el programa com les interfícies que és capaç de generar poden ser executats en Windows, Linux/X11 i Mac OS). Aquesta eina ens ha sigut molt útil a l'hora de dissenyar tota la part visual del programa **DCSMonitor**, ja que l'entorn que proporciona aquest programa és molt senzill d'utilitzar, i ofereix un gran ventall de possibilitats, com poden ser els botons, les etiquetes, els menús, la finestra de exportar les gràfiques, la integració amb les gràfiques en temps real, etc.

Encara que té moltes característiques interessants nosaltres només l'hem utilitzat per fer l'entorn gràfic en un fitxer de format .ui, per després exportar-lo per Python.

2.2.4 QtLingüist

Qt ens ofereix suport per traduir les aplicacions en altres llenguatges, i gracies a aquest programa podem editar els fitxers de traducció fàcilment, i un cop traduïts tots els textos, el mateix *QtLingüist* ens genera el fitxer adequat perquè el nostre programa **DCSMonitor** pugui canviar l'idioma en qualsevol moment.



Figura 2.12: Logotip de QtLingüist

2.2.5 ERIKA Enterprise

ERIKA Enterprise 2.13 és un Sistema Operatiu en Temps Real de codi obert derivat de OSEK/VDX (veure glosari), creat per la casa Evidence i que està integrat en el plugin RT-Druid d'Eclipse.



Figura 2.13: Logotip Erika Enterprise -

Erika ens ofereix un RTOS d'espai reduït (1-4 Kb Flash) per sistemes embotrats d'un sol nucli o multi-nucli.

Aquest sistema operatiu és usat actualment en més de 20 universitats de tot el món, a més de varies companyies del mercat automobilístic (entre d'elles Magneti Marelli Powertrain i Cobra Automotive Technologies).

2.3 Protocols de comunicació

Les comunicacions en els Sistemes Distribuïts de Control són el pilar mestre sobre el que es recolza el control, per aquesta raó la major part del projecte es basa en aquestes qüestions, i es dissenyen els diferents missatges i identificadors de manera que es pugui maximitzar l'ús dels avantatges que té cada protocol.

En aquesta secció es podrà veure un repàs general sobre el tipus de protocols usats en el laboratori, així com el tipus de connexionat que existeix entre els diferents dispositius que el formen.

2.3.1 CAN

El protocol CAN (de l'Anglès Controller Area Network) va ser dissenyat per permetre la comunicació entre dispositius sense la necessitat d'un host (o amfitrió). Inicialment la seva aparició va esdevenir de la problemàtica que existia en els vehicles automòbils, en els quals el nombre de dispositius era cada cop més gran, fins arribar al punt de necessitar més de 2 km de cable, els quals representaven més de 100 Kg (informació extreta de (4), (5)).

Va ser la firma Alemana *Robert Bosch GmbH*, qui va desenvolupar aquest protocol, basat en una topologia bus per la transmissió de missatges en entorns distribuïts, inicialment (com hem dit anteriorment) per l'entorn automobilístic, però finalment acollit per entorns marins, agrícoles, industrials, etc.

Les característiques més importants de la comunicació CAN són les següents (especificació, (6)):

- Jerarquia de nodes multimaster.
- Tècnica d'accés al medi CSMA/CD+CR (de l'anglès Carrier Sense, Multiple Access/Collision Detection + Collision Resolution).

2. TECNOLOGIA

- Comunicació conduïda per events.
- Broadcast.
- Iniciativa de transmissió a càrrec de la font d'informació.
- El nom del missatge genèricament designa la informació, no el node.
- Resposta a petició remota.
- Detecció i correcció d'error a nivell de missatge.
- Capacitat de detecció d'errors a nivell del medi de comunicació.
- Tolerància a fallades.
- Confirmació.
- Transferència de missatges consistent sobre tot el sistema.
- Codificació de bit.
- Sincronització de bit.
- Sincronització entre nodes.
- Distribució del sistema.
- Velocitat de transmissió.
- Característiques del driver de línia.
- Múltiples proveïdors de xips.

2.3.2 Serie via RS232

En computació la comunicació serie és tota aquella transferència de dades en la que la informació va bit a bit una darrera una altra. Aquest tipus de comunicació és usada en múltiples protocols de comunicació, com poden ser el USB, Ethernet, FireWire o per exemple CAN (vist anteriorment 2.3.1). Però usualment, quan parlem del port serie de l'ordinador, ens referim a la norma RS232 (de l'anglès Recommended Standard 232) (veure (7)).

Aquesta norma determina les seves característiques físiques, la temporització dels senyals, les velocitats de transmissió, i la mida del connector i dels seu pinatge.

Normalment tots els ordinadors de sobretaula solen tenir algun connector destinat a aquest tipus de comunicació, en el format d'un connector DB-9 (originalment DE-9, veure figura 2.14), en canvi els ordinadors portàtils ja no solen tenir aquest tipus de connector, per aquesta raó es necessita utilitzar un convertidor de RS232 a USB.



Figura 2.14: Connector per port serie RS232 - En forma de DE-9

Aquesta comunicació va ser dissenyada per connectar equips terminals de dades amb equips de comunicació de dades, però en ocasions (com en el cas del connexionat entre dos ordinadors) es connecten dues terminals de dades. En aquest cas es sol utilitzar un tipus de connexionat anomena null módem (per la no existència de módem).

El connector DB-9; com el seu nom indica; té 9 pins, i cada un d'ells té definit el seu funcionament, com pot ser el d'enviar dades, o el de rebre dades. S'adjunta una taula amb l'especificació de cada un d'aquests pins (taula tab:tec:prot:rs232).

Número de pin	Nom
1	CD: Detector de transmissió
2	RXD: Recepció de dades
3	TXD: Transmissió de dades
4	DTR: Terminal de dades preparat
5	GND: Senyal de terra
6	DSR: Ajust de dades preparat
7	RTS: Permís per transmetre
8	CTS: Preparat per transmetre
9	RI: Indicador de trucada

Taula 2.1: Pinatge del connector serie per RS232

2. TECNOLOGIA

2.3.3 Topologia en bus

Aquest és un tipus de connexionat entre múltiples dispositius en els quals tots ells estan connectats a un mateix bus central (veure figura 2.15). Això comporta alguns avantatges però també molts inconvenients que es procuren resoldre de diferents maneres.

Per crear aquest tipus de xarxa és necessari finalitzar els cables amb resistències de carrega final. Aquestes resistències són calculades depenent de les senyals que hi circulen, i els metres de cable que formen el bus, i aconseguixen mitigar l'efecte rebot que apareix en un cable tallat (amb la resistència ben calculada a ulls dels dispositius el bus resultaria ser de mida infinita).

- Avantatges
 - Facilitat en la implementació.
 - Creixement del nombre de dispositius fàcil.
 - Simplicitat en el tipus d'arquitectura.
 - Recepció de tota la informació del bus.
 - No hi ha necessitat de redirigir missatges.
- Inconvenients
 - Existeix un límit del nombre de dispositius, depenent de la qualitat del senyal.
 - Pot produir-se degradació del senyal.
 - Complexitat de reconfiguració i aïllament de fallades.
 - Un problema en un canal sol degradar tot el bus.
 - El bon funcionament sol degradar a mesura que el bus creix.
 - El bus ha de ser degudament tancat.
 - Alta pèrdua de transmissions degut a les col·lisions entre missatges.

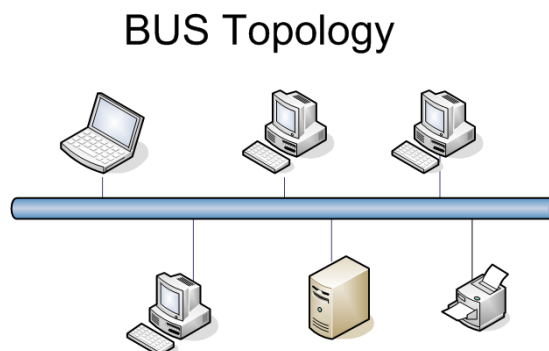


Figura 2.15: Topologia en bus -

2.4 Conclusions

En aquest punt ja ens hem fet una idea de totes les tecnologies que hi ha al voltant d'aquest projecte. Hem vist que les xarxes en bus tenen alguns inconvenients que cal resoldre, quines facilitats ens proporciona utilitzar un bus CAN, com podem rebre els valors d'estat del control mitjançant la comunicació serie via RS232, i quins dispositius s'utilitzen per poder connectar-se a un bus CAN, anomenats transceptor CAN.

També hem pogut conèixer varis programes que ens ajuden en la tasca de programar tots aquests dispositius, com crear interfícies per programes que corrin en varis sistemes operatius, i com generar aquests programes multilingües.

Per tant ja podem abordar la configuració que té el laboratori actual, i podem començar a dissenyar tot el necessari per muntar un laboratori en el que tots els dispositius comparteixin un mateix bus CAN. Tot això ho podrem veure i seguir en el següent capítol.

2. TECNOLOGIA

3

Disseny del laboratori

Aquest capítol comença amb l'explicació de l'entorn i dels objectius que té el laboratori actual de Sistemes de Control Empotrats i en Xarxa. Exposarem per tant el seu propòsit i les limitacions que aquest tenia, i de quina manera hem dissenyat la nova plataforma per poder solventar algunes d'aquestes mancances. Per tant entrarem en la part del disseny del nou laboratori, i veurem les raons que ens han portat a escollir els diferents camins que teniem, així com el disseny de les comunicacions que hi ha entre els diferents dispositius i el nou programa **DCSMonitor** .

3.1 Resum del laboratori actual

L'objectiu d'aquest laboratori és l'anàlisi, el disseny i la implementació d'un sistema de control empotrat i en xarxa (a partir d'ara *NECS* , sigles de l'anglès Networked and Embedded Control Systems) (8, laboratori complet en el document *Networked and Embedded Control Systems (NECS) Double Integrator Control Lab*).

En aquest laboratori es cobreixen varies fases:

- Disseny de controls.
- Anàlisi de sistemes *NECS* multitasques.
- Disseny de sistemes *NECS* multitasques.
- Implementació del sistema *NECS* .

3. DISSENY DEL LABORATORI

La plataforma d'implementació ens permetrà controlar un circuit Doble Integrador (a partir d'ara *DI* de l'anglès Double Integrator) d'un microcontrolador. Com es mostra en les figures 3.1 i 3.2.

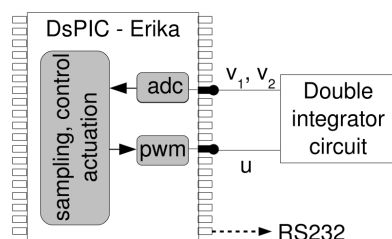


Figura 3.1: Microprocessor-based control -

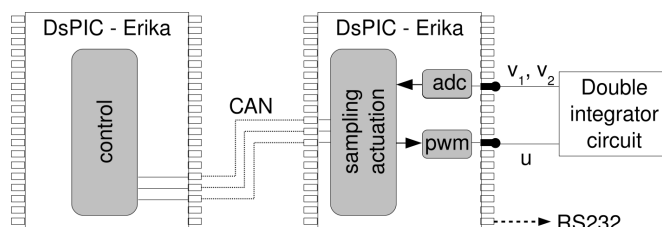


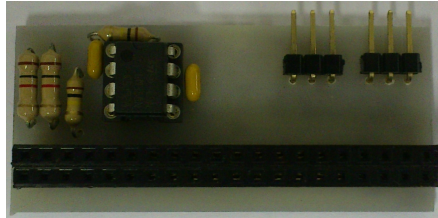
Figura 3.2: Network-based control -

En la primera configuració, figura 3.1, moltes tasques són executades concurrentment al cap davant del Sistema Operatiu en Temps Real Erika. Per aquesta raó les tasques de control del circuit doble integrador han de competir amb la resta per el temps de CPU.

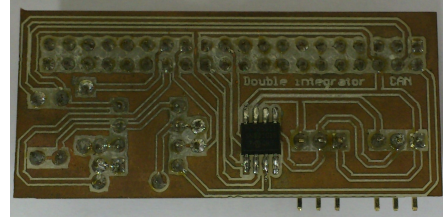
En la segona configuració, figura 3.2, permet tancar el llaç de control en una xarxa, en aquest cas un bus CAN. En aquest escenari, les limitacions dels recursos ve donada per l'ampla de banda de la comunicació. D'aquesta manera connectant varis llaços en una mateixa xarxa ens pot permetre analitzar un sistema distribuït de control més realista.

En les dues configuracions, les plaques estan equipades amb microcontroladors dsPIC33FJ256MC710 de *Microchip*. Les plaques són plaques *FLEX* de Evidence. S'han creat tres plaques amb un circuit doble integrador (figura 3.3a), comunicació CAN (figura 3.3b) i un port RS232 (figura 3.3c) que s'han afegit a aquestes per tal de comunicar-se entre elles, i amb l'ordinador.

La placa *FLEX* que porta el circuit doble integrador es l'encarregada de simular un *Sensor* i un *Actuador*, en els quals el *Sensor* es un conversor analògic digital (ADC)



(a) Doble Integrador



(b) Transceptor CAN



(c) Modul RS232

Figura 3.3: Perifèrics per les plaques *FLEX* .

que llegeix els voltatges de sortida de la primera i segona integral, i l'*Actuador* aplica diferents nivells de voltatge a l'entrada del circuit a través d'un modulador per amplada de polsos (PWM).

L'objectiu del control es que la sortida del doble integrador segueixi un valor de referencia que va variant. Aquest s'aconsegueix aplicant un algoritme de control i aplicant el valor calculat a l'entrada del circuit.

Un cop l'entorn està preparat la configuració queda com a la figura 3.4. En la qual la placa superior actua com a *Controlador* que remotament fa el control a través del bus CAN. En la part inferior tenim el *Sensor/Actuador* que compta amb la placa del doble integrador connectada, i per tal de debugar el control també compta amb la interfície RS232 (en aquesta imatge queda sota de la placa superior) la qual envia periòdicament els valors de l'estat.

3. DISSENY DEL LABORATORI

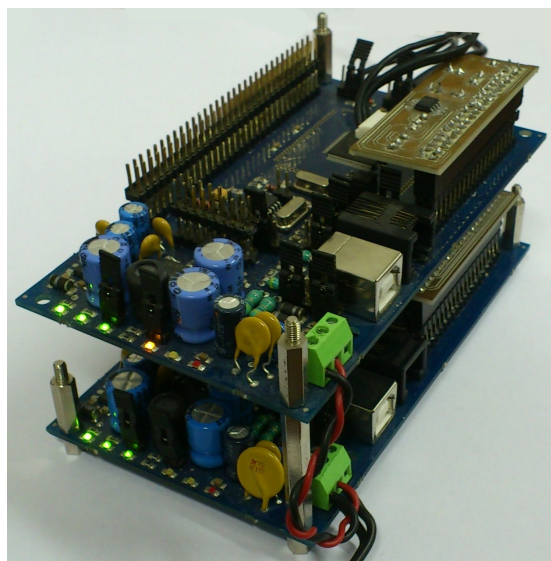


Figura 3.4: Llaç de control -

3.2 Disseny del Nou laboratori

Fins aquest moment els problemes a resoldre han estat pels temps de demora que hi havia entre lectures del *Sensor* , accions de l'*Actuador* i els càlculs que havia de fer el *Controlador* .

En aquest apartat ens interessa que tots els dispositius estiguin en un mateix bus CAN i comprovar el funcionament dels llaços en un entorn compartit, amb diferents tipus de prioritats i amb la possibilitat de saturar el bus en certes ocasions (figura 3.5).

A part d'aquesta connexió general al bus, també s'introduirà en la xarxa un dispositiu que anomenarem *Monitor* , el qual tindrà la opció de monitoritzar tots els paquets de dades d'un cert llaç de control (un grup format per un *Sensor* , un *Actuador* i un *Controlador*).

Tota la informació que el nou dispositiu *Monitor* capturi, podrà ser enviada mitjançant el port sèrie RS232 a un programa d'ordinador (**DCSMonitor**) el qual mostrarà mitjançant unes gràfiques en temps real l'estat del control de cada grup del laboratori.

A més el *Monitor* tindrà la capacitat d'incrementar la carrega del bus en quant a nombre de missatges CAN que hi circulin, dificultant el correcte funcionament dels llaços, i podent comprovar en temps real la resposta d'aquests.

Amb aquesta idea general i tenint en compte les relacions que hi haurà entre tots els elements que formen el laboratori es va dissenyar un esquema general en el que apareixen aquests elements i el tipus de comunicació que hi ha entre ells (figura 3.6).

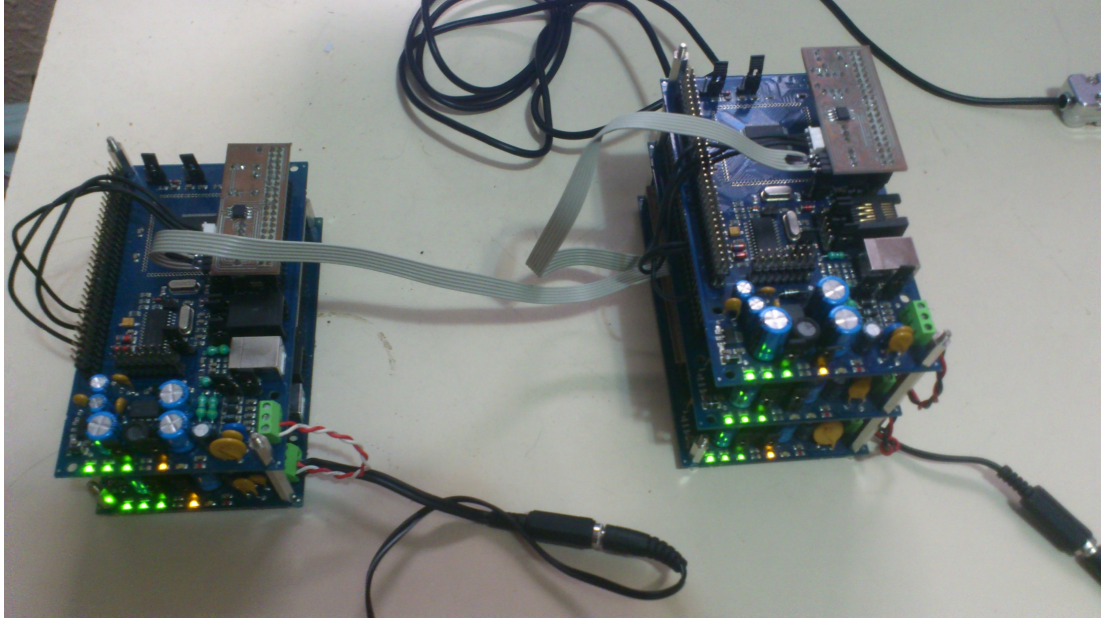


Figura 3.5: Entorn del nou laboratori - A l'esquerra un llaç de control format per un *Controlador* (primer pis) i un *Sensor/Actuador* (segon pis). A la dreta un llaç de control format per un *Controlador* (segon pis) i un *Sensor/Actuador* (tercer pis) i un dispositiu *Monitor* (primer pis).

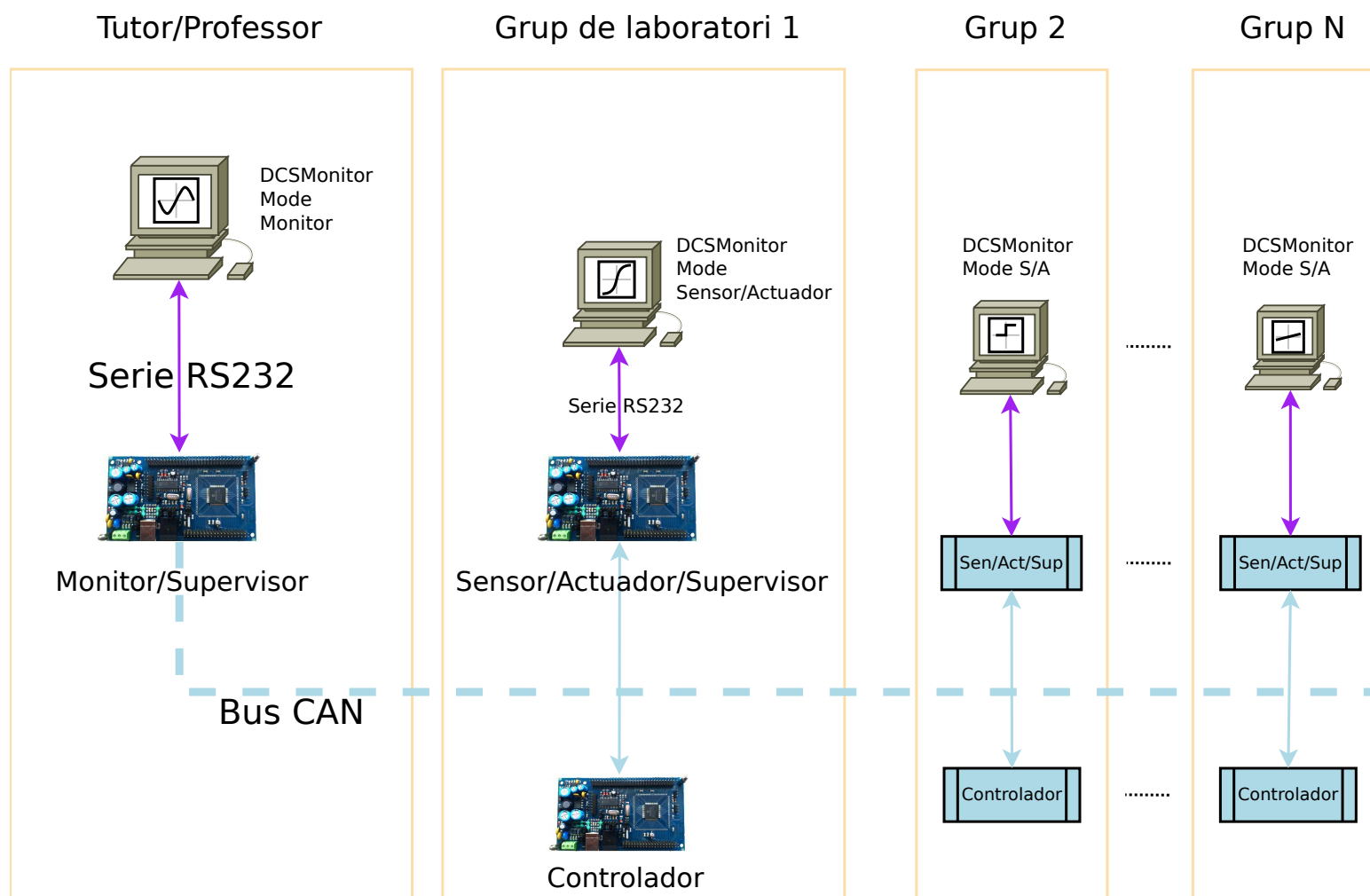


Figura 3.6: Diagrama general dels elements del laboratori - Diagrama de tots els elements que formen part del laboratori, i les diferents comunicacions que els relacionen.

3.2.1 Interfície

Aquest programa es només una eina per visualitzar i comprovar el funcionament dels diferents controls que hi hagin al bus CAN, o el control del llaç al que estigui connectat, però en cap moment ha de ser un programa complicat d'executar, ja que el seu objectiu principal es precisament ajudar en aquesta tasca. Això implicava que el programa fos senzill d'utilitzar, però a l'hora complert.

Primer de tot calia saber quines opcions havien d'aparèixer a primera vista, així que es va fer un llistat d'elements necessaris:

- Llista dels identificadors dels diferents controls al bus CAN
- Estat del port sèrie RS232
- Lectures dels valors del control en format text
- Gràfica del control
- Mostrar/eliminar dades de la gràfica
 - Referència
 - Valor d'entrada
 - Primera integral
 - Segona integral
- Idioma per seleccionar
 - Català
 - Espanyol
 - Anglès
 - Francès
- Mode de comunicació
 - Comunicant amb *Sensor/Actuador*
 - Comunicant amb *Monitor*
- Dades informatives (extres)

3. DISSENY DEL LABORATORI

- Total de llaços de control al bus CAN
 - Imatges per segon mostrades en la gràfica en temps real
 - Nombre de mostres de cada línia de la gràfica
 - Bytes pendents en el buffer d'entrada del port sèrie RS232
 - Algun ítem intermitent que indiqui que les dades s'actualitzen correctament (en temps real)
-
- Botó per connectar al port sèrie
 - Botó per llistar llaços del bus CAN
 - Botó per guardar una imatge de la gràfica
 - Botó per netejar el text
 - Botó per començar i aturar la monitorització

Un cop remarcats aquests elements, només calia identificar aquells que havien de ser accessibles directament, i posicionar-los de tal manera que fos intuïtiu d'utilitzar. Per tant es va treure la selecció d'idioma i de mode d'execució de la pantalla principal i es va posar en les *Preferències* del programa, de la mateixa manera amb el mode d'execució del programa ja que els alumnes sempre faran servir el mode *Sensor/Actuador* i el professor el mode *Monitor* .

Abans de començar a dissenyar la interfície amb algun programa d'edició, es van fer alguns esbossos de com hauria de quedar la organització, i finalment es va decantar pel dibuix de la figura 3.7.

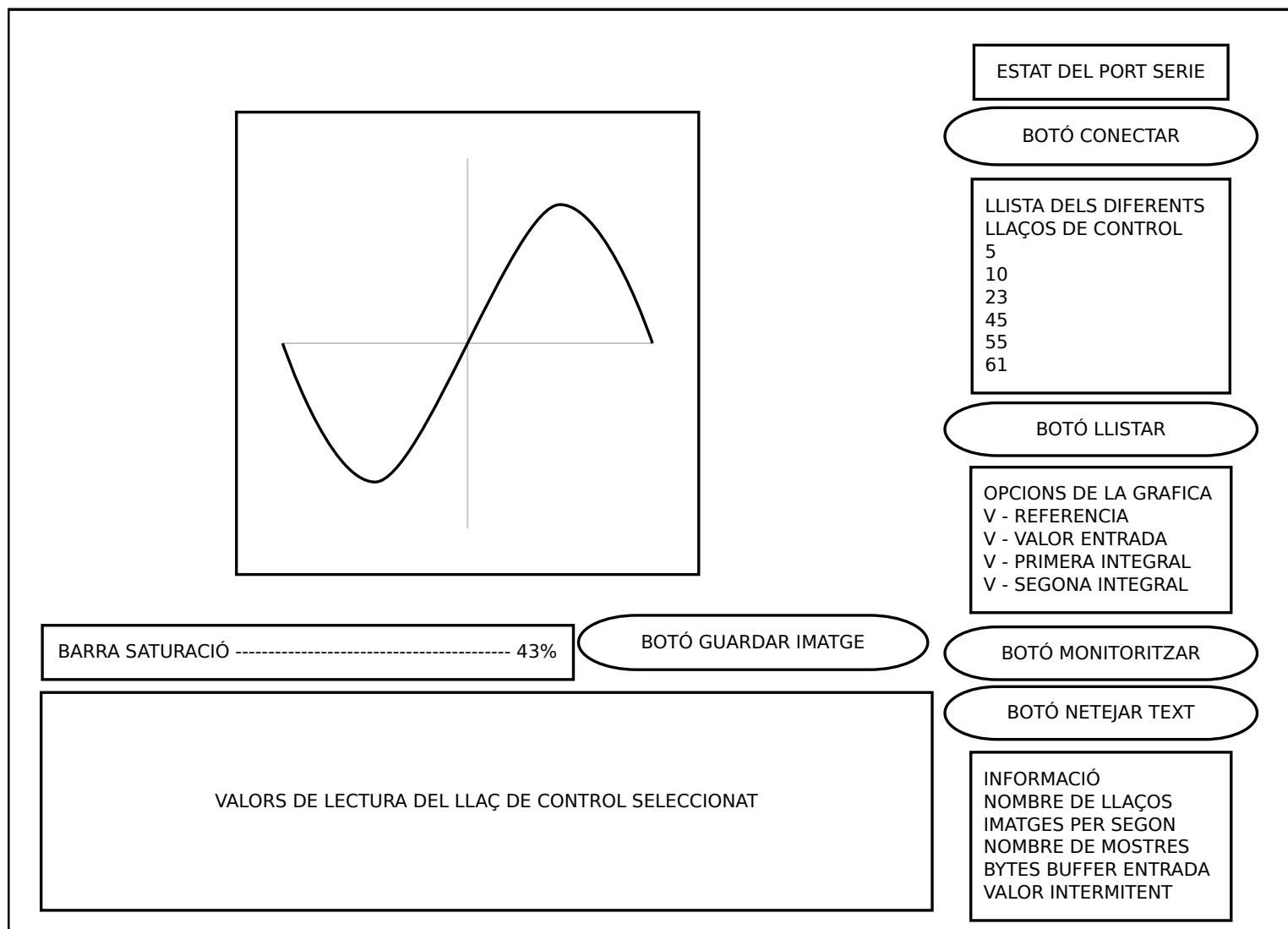


Figura 3.7: Esquema de la interfície del programa DCSMonitor -

3. DISSENY DEL LABORATORI

Ja amb l'esquema ben clar es van provar varis IDE's d'edició d'interfície, i un cop analitzades varies qüestions es va decidir utilitzar *QTCreator* . Aquestes són algunes de les raons que em van fer decantar per aquest IDE:

- Existència de bona documentació de les API's de QT.
- Portabilitat a altres sistemes operatius com Windows.
- Un entorn molt intuïtiu.
- Fàcil integració amb widgets propis.
- Existència d'exemples d'integració de matplotlib amb QT4 (9, Capítol 6 del llibre *Matplotlib for Python Developers*)

3.2.2 Identificadors dels missatges CAN

El primer problema que hem de resoldre al dissenyar un sistema distribuït en el qual hi haurà múltiples dispositius és enviar i rebre els diferents missatges al dispositiu indicat.

En el laboratori anterior existien 3 tipus de missatge diferents (s'han mantingut els noms originals):

1. Sensor to controller message

Aquest missatge l'enviava el *Sensor* i anava destinat al *Controlador* , i portava el primer i el segon valor de sortida del doble integrador.

2. Controller to actuator message

Aquest missatge el generava el *Controlador* i anava destinat al *Actuador* , el seu contingut eren 4 Bytes amb el valor d'entrada del doble integrador.

3. Controller updates reference (supervision)

Aquest missatge generat pel *Controlador* tenia com a destinació el supervisor (*Sensor/Actuador*) i contenia el valor de referència teòric que hauria de seguir el doble integrador.

Els identificadors per aquests tres missatges eren senzills de proposar ja que tenien els 29 bits de l'identificador per escollir sense cap tipus de restricció, per aquest motiu els missatges de control, estat del sensor i referència podien tenir els identificadors 1,

2, i 3 respectivament. Cada dispositiu conneixia l'identificador del senyal que volia i per tant podia rebre aquests senyals.

En el disseny actual es volia introduir un entorn compartit, per tant hem dividit l'identificador de missatge CAN en subparts, per tal d'afegir noves funcionalitats:

- Diferents nivells de prioritat.
- Diferents identificadors de llaços de control.
- Diferents classes de missatge.
- Diferents subclasses de missatge.

Així que s'ha dissenyat un sistema d'identificadors per tal de complir tots aquests requisits. A més per tal de garantir els temps de resposta dels controladors i el millor funcionament del control s'ha procurat seguir el disseny realitzat a l'article "Schedulability Analysis for CAN-based Networked Control Systems with Dynamic Bandwidth Management"(10).

En la figura 3.8 es pot veure la codificació que es va seguir en l'article en el que es van fer els càlculs per garantir un funcionament òptim del control (s'ha mantingut el text original).

Un cop revisat el funcionament d'aquests identificadors i dels missatges que en el nostre laboratori havíem de generar, es va pensar en la millor reorganització que es podia fer dels nostres missatges, i finalment va quedar el següent esquema genèric per tots els missatges (figura 3.9).

Aquest esquema ens permet adaptar tots els missatges que hi havia fins ara, i a més podem generar nous missatges utilitzant l'identificador de tipus genèric de 3 bits (010) utilitzant els últims dos bits per indicar quin missatge és.

Amb tots aquests punts ben clars es va crear un diagrama dels elements que formaran el laboratori, amb tots els dispositius que poden formar part d'ell, i les diferents interaccions que han de realitzar. En aquest diagrama podrem observar els períodes dels diferents missatges, l'esquema d'identificadors i el tipus de màscares i filtres que seran útils per rebre els missatges. Tot això es pot observar en el diagrama de la figura 3.10.

3. DISSENY DEL LABORATORI

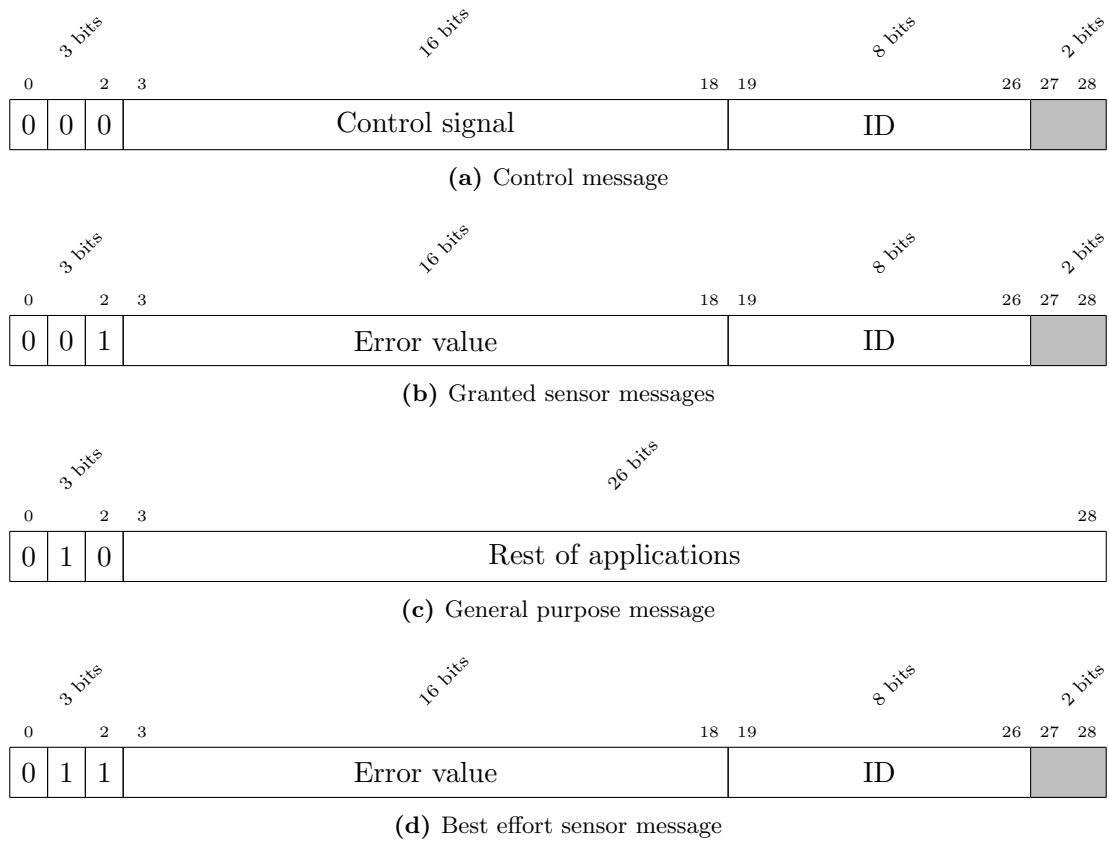
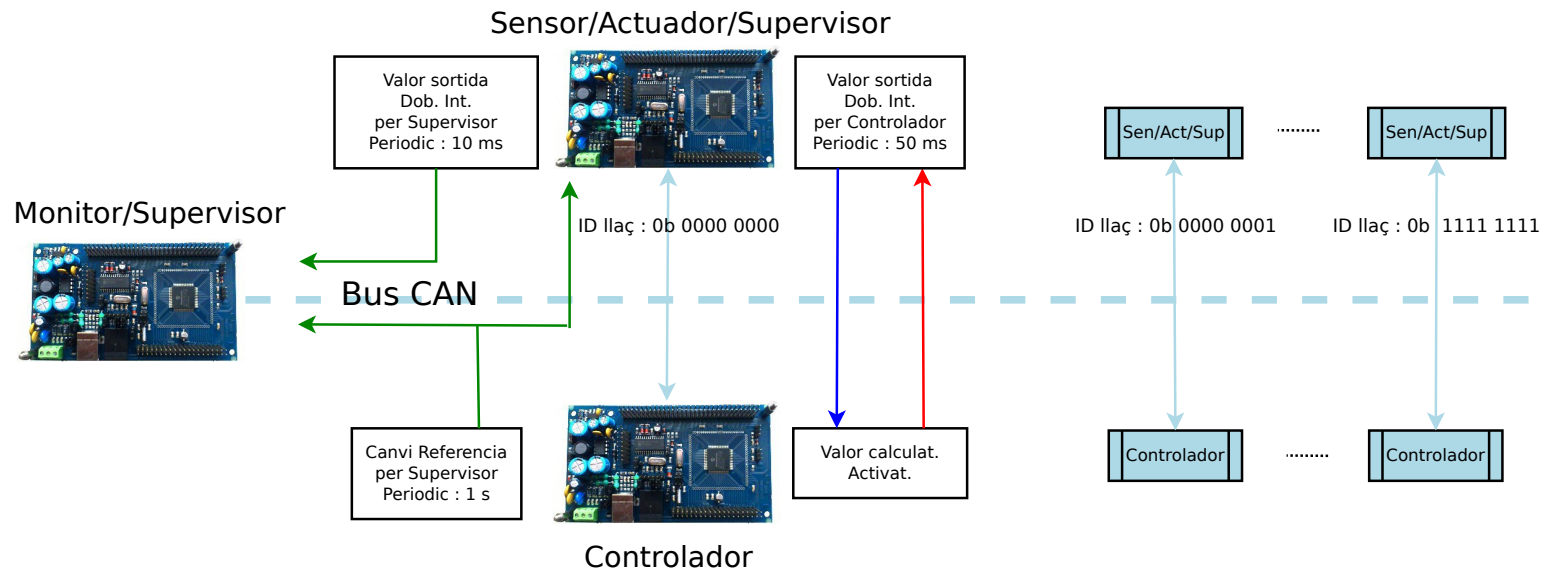


Figura 3.8: Codificació de bits dels identificadors CAN per tipus de missatges - usats en l'article "Schedulability Analysis for CAN-based Networked Control Systems with Dynamic Bandwidth Management"(10) (text original).



Figura 3.9: Esquema genèric dels identificadors CAN pel nou laboratori.



Identificadors comunicació CAN (extended frame de 29 bits)

Classe : 3 bits : ccc
 Prioritat : 16 bits : pppp pppp pppp pppp
 ID llaç : 8 bits : iiii iiii
 Subclasse : 2 bits : ss

Classes
 Missatge de control : 000
 Missatge de sensor garantit : 001
 Missatge de propòsit general : 010

Subclasse
 Missatge de canvi de referencia : 00
 Estat del sensor per supervisor : 01

Esquema general de l'identificador

\ Classe / \ Prioritat / \ ID llaç / \ Subclasse /
 0b ccc pppp pppp pppp pppp iiii iiii ss

Exemples d'identificadors missatges

controlador -> actuador : 0b 0000 00pp pppp pppp pppp pp11 iiii iiss
 sensor -> supervisor : 0b 0000 10pp pppp pppp pppp pp11 iiii ii01

Mascarees identificadors CAN

Tipus de mascara

Mascara 1: Guarda tots els missatges sense importar el filtre
 0b 0000 0000 0000 0000 0000 0000 0000

Mascara 2: Nomes del llaç i classe que indicui el filtre
 0b 0001 1100 0000 0000 0000 0011 1111 1100

Mascara 3: Nomes del llaç, classe i subclasse que indiqui el filtre
 0b 0001 1100 0000 0000 0000 0011 1111 1111

Exemple de filtre i mascara per rebre missatges del llaç 34 de la classe de missatge de propòsit general amb la subclasse de canvi de referencia:

Mascara 3 : 0b 0001 1100 0000 0000 0000 0011 1111 1111
 Filtre : 0b xxx0 10xx xxxx xxxx xxxx xx00 1000 1000

Figura 3.10: Diagrama de comunicacions i identificadors CAN - Diagrama de funcionament del laboratori, disseny dels identificadors i màscarees per bus CAN, i tipus de missatges de comunicació.

3. DISSENY DEL LABORATORI

Tot seguit es veuran un a un tots els missatges que s'han adaptat i creat explicats a fons.

3.2.2.1 Missatge de control per l'Actuador

Aquest tipus de missatge s'envia cada cop que el *Sensor* envia una lectura nova al *Controlador* (missatge de l'estat del *Sensor* per al *Controlador* explicat en l'apartat 3.2.2.2), aquest al rebre els valors de les senyals fa els càlculs oportuns i ha de respondre lo més ràpid possible l'Actuador, que és l'encarregat de donar el valor al doble integrador.

Aquests tipus de missatges han de ser d'alta prioritat i per tant, tenen assignats els primers 3 bits de l'identificador a zero, ja que en els missatges CAN els identificadors amb els primers zeros són els més prioritaris. Així que utilitzant la codificació mencionada anteriorment farem servir el tipus de missatge *control message*, quedant l'estructura de l'identificador i el senyal de control com la figura 3.11.



(a) Identificador CAN (29 bits).



(b) Informació.

Figura 3.11: Missatge CAN del *Controlador* al *Actuador*.

3.2.2.2 Missatge de l'estat del Sensor per al Controlador

Aquest és un missatge que s'envia periòdicament cada 50ms, i és el responsable d'enviar l'estat en el que es troben les dues integrals. Té com a destinació el *Controlador* que posteriorment és el que farà els càlculs dependents d'aquests valors.

En aquests tipus de sistemes aquests missatges, al igual que els del controlador, també són molt importants, per aquest motiu se'ls assigna el tipus 001, ja que és el segon tipus més prioritari. A part dels primers tres bits, els següents 16 bits es poden

utilitzar per codificar l'error que s'està produint, de manera que sigui més prioritari el que tingui el valor més desviat del desitjat.

Així que utilitzant la codificació mencionada anteriorment farem servir el tipus de missatge *granted sensor message*, quedant l'estructura de l'identificador i el senyal de control com la figura 3.12.

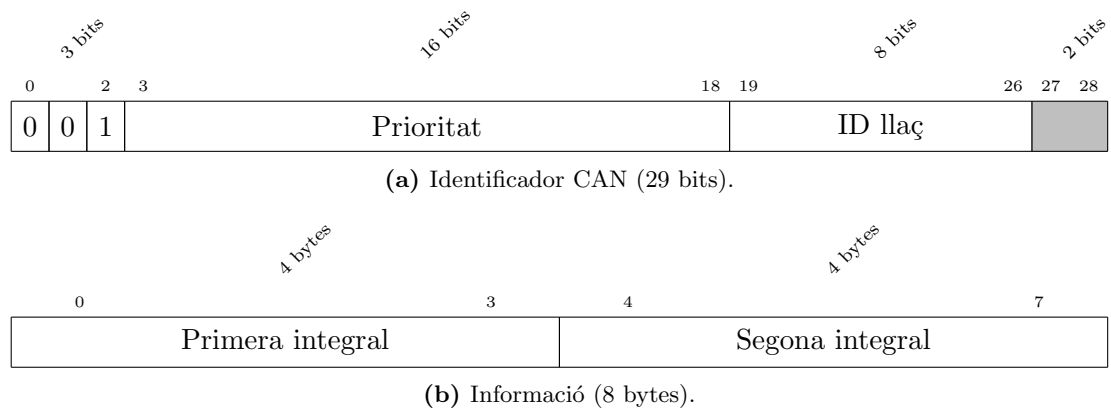


Figura 3.12: Missatge CAN de l'estat del *Sensor* per al *Controlador* .

3.2.2.3 Missatge de canvi de referència

Aquest és un missatge que s'envia periòdicament cada segon, i és el responsable d'enviar el valor de referència que s'intenta aconseguir en cada moment. Aquest valor oscil·la de 0,5 a -0,5 i té com a destinació el *Supervisor* que ha d'enviar aquest valor a l'ordinador per que pugui dibuixar la gràfica correctament.

Fins aquest laboratori el *Supervisor* era el dispositiu *Sensor/Actuador* ja que era l'encarregat de la comunicació per RS232, però en el nou laboratori el microcontrolador que fa de *Monitor* també captura aquest tipus de missatge si des del programa de **DCSMonitor** se li ha indicat. Tot i així el dispositiu *Sensor/Actuador* en el nou laboratori segueix rebent aquesta informació i ja pot seguir enviant informació per RS232 a un altre ordinador.

Com aquest tipus de missatges no té cap tipus d'importància a l'hora de portar a terme el control, utilitzen l'identificador de tipus *general purpose messages*. Així no empitjoren la qualitat del control però no deixen de ser missatges que procuren garantir la seva recepció. A part d'aquest missatge que pretén enviar el valor de referència, hi ha altres missatges que voldran utilitzar el tipus d'identificador que hem mencionat,

3. DISSENY DEL LABORATORI

per aquesta raó utilitzarem els últims dos bits posats a zero per indicar que és d'aquest tipus.

Així que utilitzant la codificació mencionada anteriorment farem servir el tipus de missatge *general purpose messages*, quedant l'estructura de l'identificador i el senyal de control com la figura 3.13.

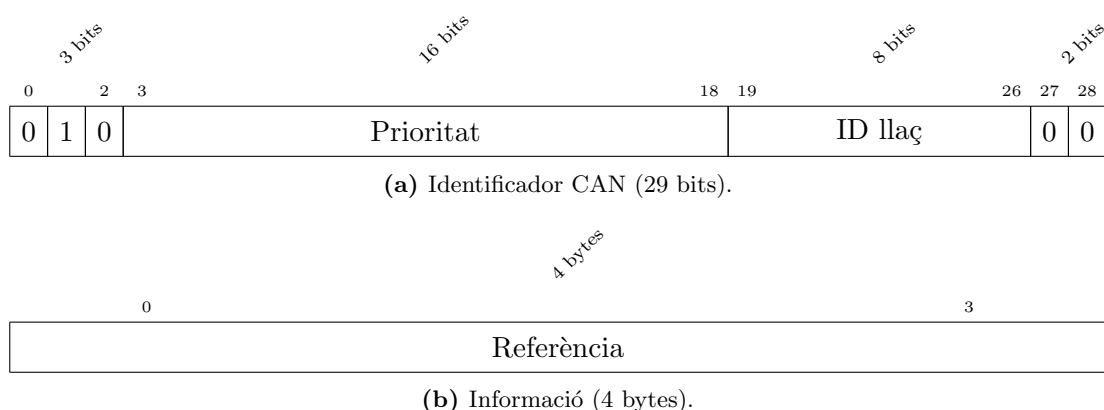


Figura 3.13: Missatge CAN per actualitzar el valor de referència.

3.2.2.4 Missatge de l'estat del *Sensor* per al *Supervisor*

Aquest és un missatge que s'envia periòdicament cada 10ms, envia exactament la mateixa informació que se li envia al *Controlador* en el missatge *Missatge de l'estat del Sensor per al Controlador* (anteriorment explicat en l'apartat 3.2.2.2), però el seu propòsit no és el de calcular el valor de control, sinó purament informatiu.

Com s'ha dit anteriorment en l'anterior laboratori el *Supervisor* era el mateix *Sensor/Actuador* (i encara ho pot seguir sent) i per tant al enviar la informació de l'estat del *Sensor* primer feia la lectura i després enviava els valors. En canvi en el nou laboratori el *Supervisor* és el *Monitor* que es troba en un altre dispositiu, així que periòdicament hem d'enviar aquesta informació, ja que d'aquesta manera es veuran les gràfiques amb els valors reals.

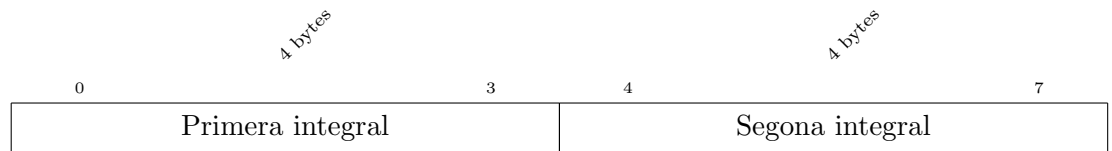
Com aquest tipus de missatges no té cap tipus d'importància a l'hora de portar a terme el control, utilitzen l'identificador de tipus *general purpose messages*. Així no empitjoren la qualitat del control, però no deixen de ser missatges que procuren garantir la seva recepció. A part d'aquest missatge hem vist un altre missatge (*Missatge de canvi de referència* en l'apartat 3.2.2.3) que utilitza el tipus d'identificador

que hem mencionat, per aquesta raó utilitzarem els últims dos bits posats a zero i u respectivament per indicar que és d'aquest tipus.

Així que utilitzant la codificació mencionada anteriorment farem servir el tipus de missatge *general purpose messages*, quedant l'estructura de l'identificador i el senyal de control com la figura 3.14.



(a) Identificador CAN (29 bits).



(b) Informació (8 bytes).

Figura 3.14: Missatge CAN de l'estat del *Sensor* per al *Supervisor*.

3.2.3 Comunicació sèrie

La part de comunicació entre l'ordinador i les plaques Flex s'ha vist modificada, ja que primerament la informació que s'enviava periòdicament a l'ordinador la realitzava el *Sensor/Actuador*, i aquest comptava amb la informació dels valors de sortida de l'integrador de primera ma.

Aquesta comunicació era unidireccional, així que el *Supervisor* enviava periòdicament el seu temps actual, el valor de referència, el valor d'entrada de l'integrador, i els dos valors de sortida d'aquest últim.

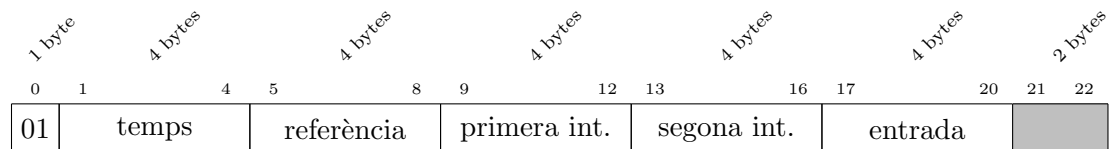


Figura 3.15: Trama de supervisió antiga per RS232 (23 bytes)

Així que en el nou sistema, per tal de complir els requisits de monitoritzar diversos llaços, i amb la intenció de no saturar el microcontrolador amb les dades de tots els

3. DISSENY DEL LABORATORI

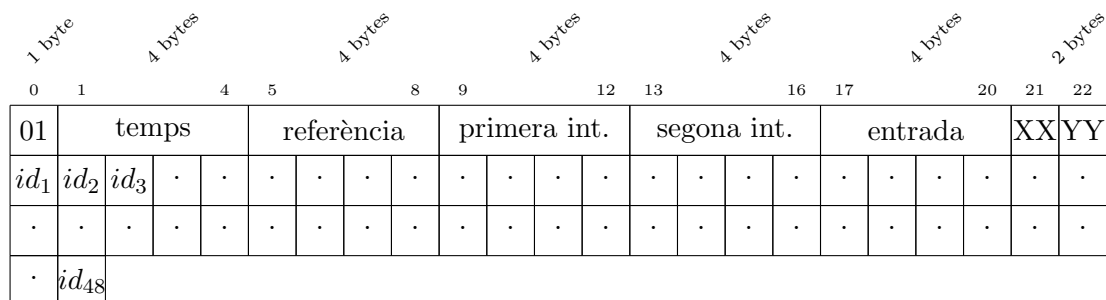


Figura 3.16: Trama de supervisió nova per RS232 (71 bytes). Els primers bytes s'utilitzen exactament igual que en la versió antiga per mantenir la compatibilitat.

dispositius ens veiem forçats a comunicar des de l'ordinador al Monitor de quin llaç volem la informació en cada moment.

A més es vol implementar la capacitat d'enviar diferents instruccions al Monitor, com poden ser saturar el bus fins a cert punt, demanar quants llaços de control existeixen al bus, i possibles ampliacions.

Per tant el sistema de comunicació ha de satisfer les següents necessitats:

- Realitzar una implementació bidireccional.
- Afegir diferents senyals de control.
 - Demanar un llistat dels llaços de control que hi ha al bus CAN.
 - Demanar els valors de *Sensor/Actuador* i *Controlador* d'un llaç.
 - Generar càrrega al bus CAN.
 - Aturar qualsevol de les anteriors accions.

Per complir la bidireccionalitat dels missatges es va estudiar la millor manera de rebre els missatges en el microcontrolador, d'aquesta manera veient com es rebien les interrupcions del bus CAN, es va seguir la mateixa metodologia per rebre una interrupció cada cop que el buffer d'entrada del port sèrie s'omplia.

Tenint en compte que les diferents instruccions que se li demanen al microcontrolador monitor no són crítiques en temps de resposta s'ha optat per crear aquests missatges de 8 bytes més que suficients per complir les especificacions actuals i mantenint un marge per possibles ampliacions (actualment en el pitjor cas s'omplen 2 d'aquests,

quan es demana monitoritzar un llaç de control, al qual ens referim amb l'últim byte de la trama).

Per tant s'ha dissenyat l'estructura bàsica de tots els senyals de control amb la següent arquitectura (figura 3.17):

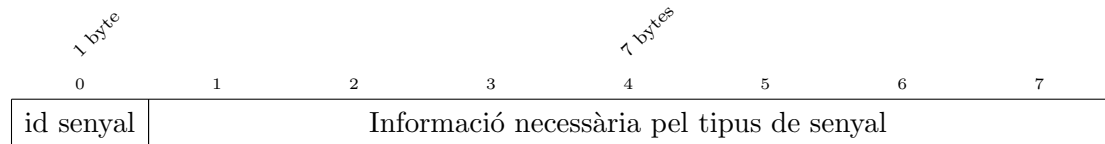


Figura 3.17: Arquitectura bàsica dels missatges enviats per RS232.

On cada identificador té el valor i el significat indicat en la taula 3.1.

Senyal	Valor	Significat
<i>SIGNAL_STOP</i>	0x01	Sumat a qualsevol altre identificador cancel·la l'acció principal.
<i>SIGNAL_MONITOR</i>	0x00	Per començar a monitoritzar un llaç de control.
<i>SIGNAL_PERCENT</i>	0x02	Per començar a generar càrrega al bus CAN.
<i>SIGNAL_DEVICES</i>	0x04	Per començar a llistar tots els llaços que hi ha actualment al bus CAN.

Taula 3.1: Identificadors dels senyals de control enviats al *Monitor* per RS232.

Amb tots aquests punts ben clars es va crear un diagrama dels elements que formaran el laboratori, amb tots els dispositius que poden formar part d'ell, i les diferents interaccions que han de realitzar. En aquest diagrama podrem observar els períodes dels diferents missatges, l'esquema d'identificadors i el tipus de màscares i filtres que seran útils per rebre els missatges. Tot això es pot observar en el diagrama de la figura 3.18.

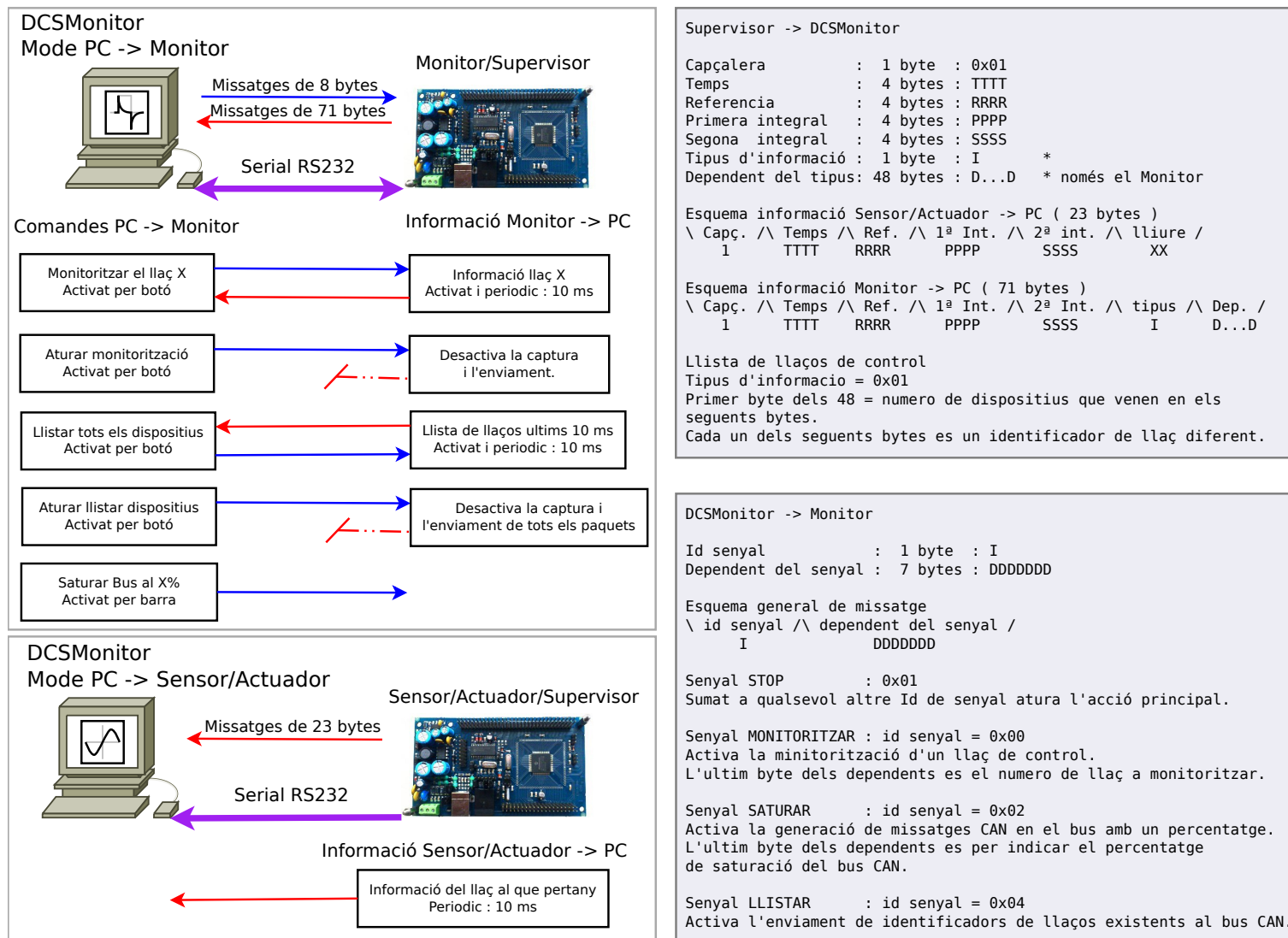


Figura 3.18: Diagrama de comunicacions i missatges RS232. - Diagrama de funcionament del laboratori, amb el disseny dels diferents missatges de comunicació per RS232. A la part superior es pot observar l'entorn entre un dispositiu *Monitor* i el programa en aquest mode, i a la part inferior connectat amb un dispositiu *Sensor/Actuador*.

3.2.3.1 Senyals de control per monitoritzar

Aquest senyal és l'encarregat d'indicar al *Monitor* que volem capturar tota la informació que intercanviï el llaç de control especificat, i que tota aquesta informació ens sigui enviada en temps real per poder dibuixar les gràfiques del control. Aquesta informació serà el valor de referència, el valor d'entrada del doble integrador, i la primera i segona integral (En cas de no indicar-li quin llaç volem monitoritzar per defecte li demanarem el llaç 0, que és el primer llaç disponible).

En el moment que vulguem aturar aquestes captures enviarem al monitor el senyal d'aturada de monitorització amb el qual deixarà de centrar-se en aquest llaç.

Aquestes són les trames dels dos missatges relatius a la monitorització (figura 3.19):

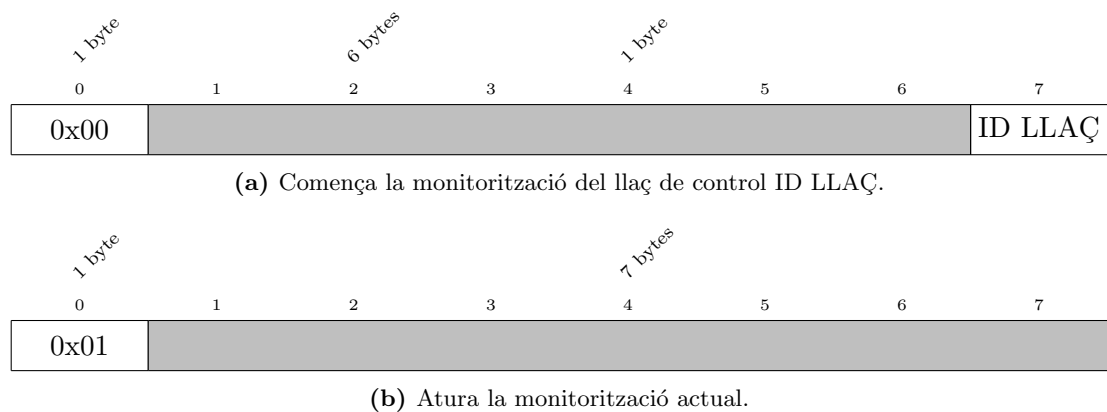


Figura 3.19: Missatges de monitorització per RS232.

Per tant l'esquema natural en la monitorització i parada d'aquesta seria el següent:

1. Seleccionem el llaç de control que volem monitoritzar.
2. Enviem la trama ID_MONITOR amb l'identificador del llaç de control.
3. El microcontrolador monitor rep el senyal i activa els filtres CAN i els seus respectius buffers per capturar totes les trames que el llaç de control intercanviï.
4. El microcontrolador activa una tasca periòdica per enviar totes les lectures que efectui per RS232.
5. En cada missatge CAN que va dirigit a algun dispositiu del llaç el monitor es guarda els valors adequats.

3. DISSENY DEL LABORATORI

6. Periòdicament envia els valors de referència, entrada del doble integrador, sortida de la primera i de la segona integral.
7. El programa monitor de l'ordinador va dibuixant en temps real les lectures que rep (podent filtrar en la gràfica quines variables es volen visualitzar).
8. En el moment que l'usuari vol aturar la monitorització s'envia el senyal d'aturada al microcontrolador.
9. El microcontrolador al rebre el senyal desactiva els filtres CAN que havia activat i atura la tasca periòdica d'enviar informació per RS232.

3.2.3.2 Senyals de control per consultar llaços

Aquest senyal indica al microcontrolador que escolti tots els missatges del bus CAN i que en guardi els identificadors dels diferents llaços de control, d'aquesta manera periòdicament ens enviarà aquests identificadors. Cal remarcar que aquesta tasca pot ser executada encara que s'estigui monitoritzant un llaç de control al mateix temps, ja que s'ha tingut en compte en el disseny de la comunicació per RS232.

Aquest tipus de acció té la possibilitat de ser activada i desactivada, i el motiu es que pot haver molts dispositius al llaç de control, i per tant en el moment que aquesta acció sigui activada, el microcontrolador necessitarà molt de temps per atendre a totes les interrupcions que es generin. A més, el programa **DCSMonitor** ha de controlar en cada moment que no es repeteix cap identificador a la llista, i per tant és més còmut que pot alentir l'ordinador.

El que fa això és activar un filtre CAN sense cap restricció, i li assigna un buffer d'entrada; cada cop que aquest buffer s'omple comprova si és un identificador que ja havia emmagatzemat, si no és així el guarda. Tot seguit torna a deixar lliure el buffer d'entrada.

En el moment que se li envia el senyal de parada es desactiva el filtre i s'allibera el buffer d'entrada.

Aquestes són les trames dels dos missatges relatius a llistar llaços de control (figura 3.20):

Per tant l'esquema natural per llistar els llaços i per parar seria el següent:

1. Enviem la trama ID.DEVICES.

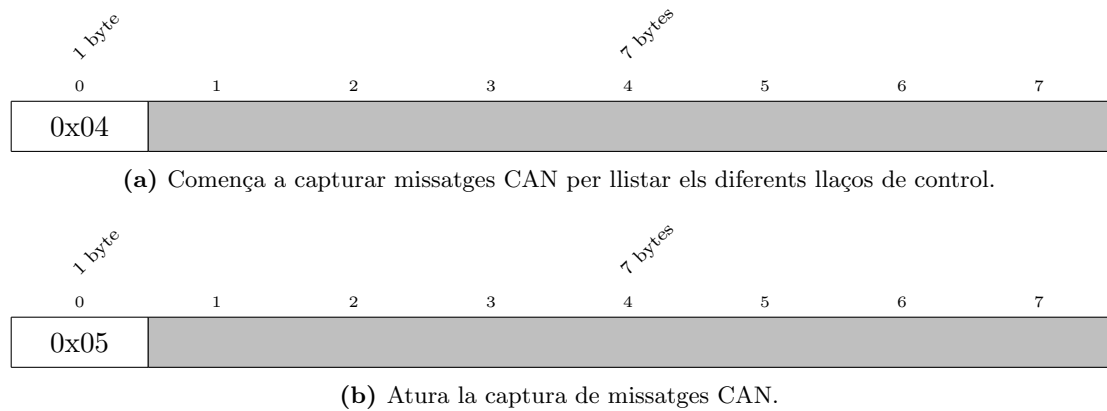


Figura 3.20: Missatges per llistar llaços de control per RS232.

2. El microcontrolador monitor rep el senyal i activa un filtre CAN sense restriccions per capturar tots els missatges que circulin pel bus.
3. El microcontrolador activa una tasca periòdica per enviar els identificadors que vagi capturant.
4. Amb cada missatge CAN que circula es comprova si l'identificador és nou i si és així s'afegeix en una pila.
5. Periòdicament envia els identificadors que hi ha emmagatzemats a la pila per RS232 a l'ordinador i es buida de nou la pila.
6. El programa monitor de l'ordinador comprova si els identificadors són nous i els afegeix a una llista.
7. En el moment que l'usuari té suficients identificadors de llaços de control se li envia al microcontrolador el senyal d'aturada.
8. El microcontrolador al rebre el senyal desactiva el filtre CAN que havia activat i atura la tasca periòdica d'enviar informació per RS232 (només en el cas que no estigui al mateix temps monitoritzant).

3.2.3.3 Senyals de control per generar càrrega

Aquest senyal indica al microcontrolador que comenci a enviar missatges al bus CAN, el nombre de missatges i el nivell de saturació del bus vindrà donat per una barra

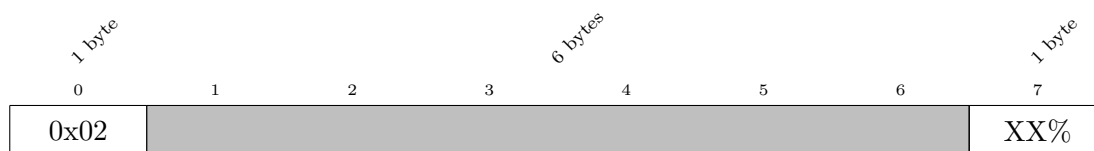
3. DISSENY DEL LABORATORI

desplaçable en el programa **DCSMonitor** . El valor de la barra serà el que s'enviarà al *Monitor* , i aquest activarà més o menys missatges.

La prioritat d'aquests missatges és màxima, per tant pot arribar a col·lapsar totalment el bus, axó provocarà en menor o major mesura que els llaços de control trobin dificultats en controlar els seus integradors i que necessitin millors algorismes de control.

Actualment els missatges generats segueixen períodes de temps variables segons el valor indicat.

Aquestes són les trames dels dos missatges relatius a generar càrrega (figura 3.21):



(a) Comença a generar càrrega amb el valor de percentatge XX.



(b) Atura la generació de càrrega al bus CAN.

Figura 3.21: Missatges de generació de càrrega al bus CAN per RS232.

3.2.4 Gràfiques en temps real

Una de les funcionalitats més importants és la de poder visualitzar en temps real les gràfiques dels controls dels diferents llaços. Anteriorment ja hem explicat quin serà el mètode d'obtenció de les dades necessàries per dibuixar-les (veure capítol 3.2.3.1), però en aquest punt ens plantegem com podem dibuixar en temps real sense que hi hagi problemes de velocitat en l'execució del programa, i mostrar suficients imatges per segon perquè l'ull humà el percebi com moviment.

En el laboratori anterior això s'aconseguia mitjançant el programa matemàtic MATLAB[®] i realitzava unes gràfiques en temps real com a la figura 3.22, aquesta gràfica durava un temps determinat i servia per comprovar que el control funcionava com era desitjat.

Aquest sistema estava bé, però tenia l'inconvenient de necessitar algun tipus de llicència de MATLAB[®] per poder visualitzar el control que havies realitzat. Per tant

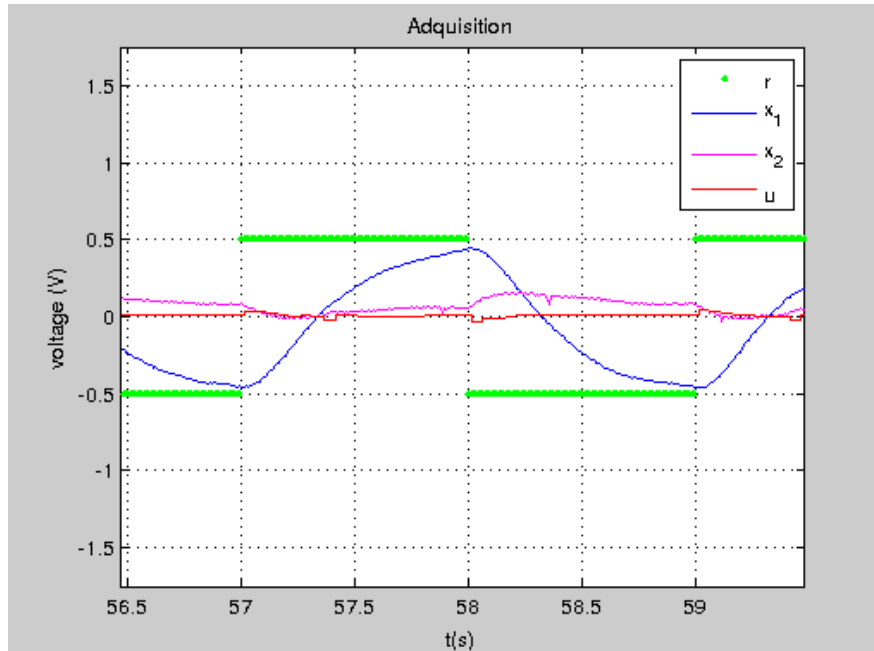


Figura 3.22: Gràfica realitzada amb MATLAB® - Copia de pantalla de la gràfica resultant al rebre les senyals amb MATLAB®

gràcies al programa **DCSMonitor** realitzat en aquest projecte en *Python* tenir aquest programa ja no es un requisit essencial.

A l'hora de realitzar el nou laboratori s'ha tingut en compte la compatibilitat amb l'anterior programa per tant la comunicació sèrie per RS232 amb el dispositiu *Sensor/Actuador* no ha estat modificada i tant es pot seguir utilitzant qualsevol dels dos programes.

Per aquesta raó es va començar a dissenyar aquest nou programa, i ja que s'anava a fer des de zero se li podien afegir algunes funcionalitats relacionades amb les gràfiques que tot seguit es llisten:

- Visualitzar la gràfica en temps real d'un llaç de control.
- Poder visualitzar la gràfica el temps necessari.
- Poder afegir o eliminar de la gràfica alguna de les línies (referència, valor d'entrada, primera i/o segona integral) en temps real, o un cop la imatge fos aturada.
- Poder generar una imatge amb varies opcions:

3. DISSENY DEL LABORATORI

- En varis formats (.eps, .pdf, png, .svg, jpg i molts més).
- Autocontinguda amb tota la informació necessària (títol del laboratori, subtítol, llegenda).
- Es pot generar la imatge només amb les línies que es desitgin (referència, valor d'entrada, primera i/o segona integral).
- Amb tots els textos de la gràfica traduïts a un dels idiomes disponibles (en la secció 3.2.5 s'indiquen quins són en aquest moment).

3.2.5 Idiomes

Un dels requisits que es buscava complir en el nou laboratori era que el programa visual fos multilingüe per tal de poder ser distribuït més fàcilment podent realitzar el laboratori en diferents parts del món.

Per tant es va estudiar la manera de realitzar aquest requisit de forma escalable i que fos ampliable fàcilment. Així que es va trobar una combinació de programes que podien realitzar aquesta tasca d'aquesta manera.

- PyQT4 : Amb aquesta llibreria de *Python* podem crear tot l'entorn visual del programa **DCSMonitor** , i també ens permetrà fer la integració de les gràfiques (referent al idioma veure capítol 4.1.6.1).
- Pylupdate4 : Amb aquest programa podrem generar fitxers entenedibles per QtLingüist per tal de traduir tots els textos que vulguem del programa (veure capítol 4.1.6.2).
- QtLingüist : Amb aquest programa es poden editar fàcilment els fitxers generats amb Pylupdate4, i un cop traduïts es genera un fitxer que la llibreria PyQT4 podrà interpretar en temps d'execució (veure capítol 4.1.6.3)

Gràcies a totes aquestes eines el programa **DCSMonitor** (que s'executa a l'ordinador) serà capaç de mostrar la informació essencial en varis idiomes, i és fàcilment ampliable ja que s'ha realitzat de forma estructurada per tal que seguint algunes instruccions es puguin generar més idiomes.

Actualment està traduït en els següents idiomes, però la llista pot ser fàcilment ampliada:

- Català

- Espanyol
- Anglès
- Francès

3.3 Conclusions

Al llarg d'aquest capítol hem pogut veure més profundament de que tracta el laboratori actual, i en quins punts es quedava curt. Tot seguit hem proposat un entorn en el que tots els llaços de control estiguessin en el mateix bus, i d'aquesta manera hem vist que necessitàvem dissenyar un ordre d'identificadors CAN que solucionessin els conflictes entre dispositius. Així que hem creat identificadors de missatges CAN amb alta prioritat per poder mantenir els controls més estables, i missatges en canvi menys prioritaris, que s'ocuparan exclusivament d'ocupar la resta del bus, sense perjudicar en cap moment el control.

També hem organitzat els diferents senyals de comunicació serie per RS232 per poder demanar l'execució de diferents accions al dispositiu *Monitor*, i quin tipus d'acions volem que aquest sigui capaç de realitzar.

Amb tot això ven clar hem pogut dissenyar la interfície del programa **DCSMonitor**, que ens ha de proporcionar informació de manera fàcil, i poder interactuar amb ell de manera intuïtiva.

Hem pogut veure mètodes per generar les gràfiques del control en temps real, i la seva exportació en diversos formats.

I per finalitzar hem vist que podem realitzar el programa **DCSMonitor** multilingüe amb la utilització de varis programes, i seguint un cert ordre en la obtenció dels fitxers de traducció finals.

Ara que ja hem pogut veure la fase de disseny es l'hora d'implementar tots els sistemes, aquesta part es una de les més llargues, ja que moltes de les eines a utilitzar són noves i fer tasques senzilles poden comportar temps llargs. El que es veurà en la següent secció es el resultat de la implementació de tota la fase de disseny que hem vist. I algunes captures del programa final.

3. DISSENY DEL LABORATORI

4

Implementació del laboratori

En aquest apartat aprofundirem en c  m s'ha dut a terme la programaci   de les diferents parts que conformen el laboratori que hem dissenyat anteriorment (cap  tol 3), i podrem veure d'aquesta manera com podem modificar el codi per ampliar el laboratori amb una infinitud de opcions, modificant o afegint diferents dispositius, o missatges de comunicaci  .

Per implementar el laboratori hem necessitat crear l'entorn necessari per compartir el bus CAN entre varis lla  os de control, i tenir el programa **DCSMonitor** connectat al dispositiu *Monitor* via RS232, per tant existeix una configuraci   b  sica formada per dos lla  os de control monitoritzats per un dispositiu *Monitor* i aquest connectat al programa **DCSMonitor** capa   de modificar el comportament d'aquest ultim. Es pot veure una fotografia de tot l'entorn preparat i configurat per tal de treballar amb aquest sistema, i amb el qual s'ha pogut dur a terme tota la fase de implementaci   (figura 4.1).

Hem dividit el cap  tol en dues seccions principals, la part de programaci   en *Python* del programa **DCSMonitor** (el programa que s'executa en l'ordinador) on podrem veure els passos que s'han seguit per crear la interf  cie visual (secci   4.1.1), com hem pogut obrir el port RS232 i posteriorment enviar i rebre tot tipus de missatges (secci   4.1.2), com hem integrat els plots del lla   de control que estigu  ssim monitoritzant o connectats (secci   4.1.3), com hem fet per exportar una gr  fica generada en diferents formats (secci   4.1.4), com hem realitzat l'actualitzaci   de les estad  stiques en temps real (secci   4.1.5), i finalment quins s  n els passos seguits per implementar els diferents idiomes, i per afegir-ne de nous (secci   4.1.6).

4. IMPLEMENTACIÓ DEL LABORATORI

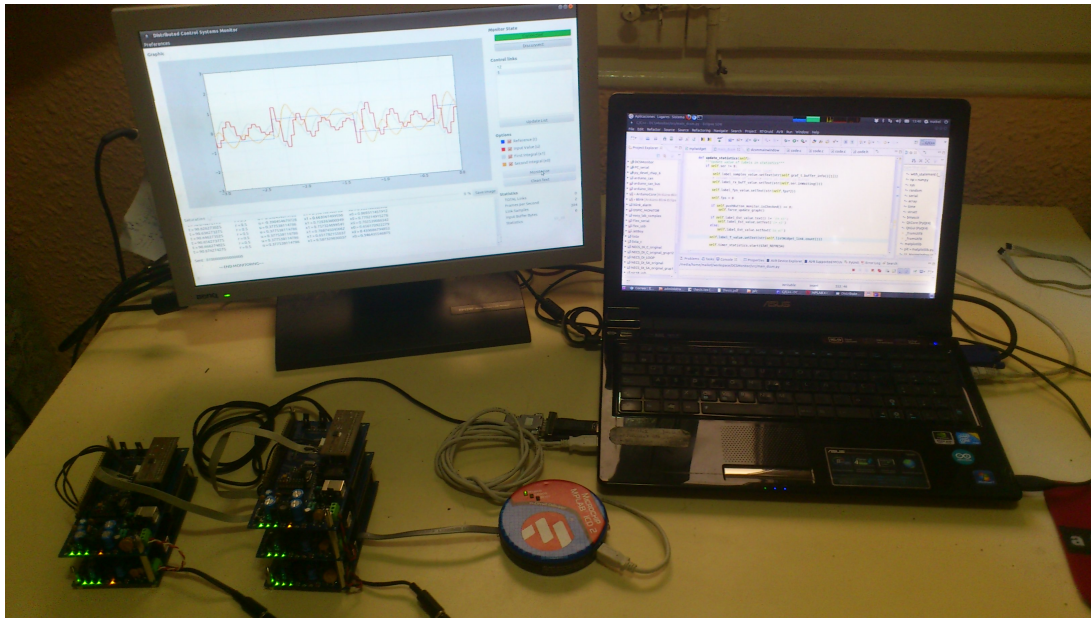


Figura 4.1: Entorn de treball - Tot el material necessari per implementar el laboratori amb dos llargs de control, un programador Mplab ICD2, un convertidor de RS232 a USB, el programa **DCSMonitor** en marxa al monitor esquerra i codi font al monitor de la dreta.

I la part de programació en C en el Sistema Operatiu en Temps Real Erika Enterprise que corre en els diferents dispositius del control (microcontroladors dsPIC33FJ256MC710, en la placa *FLEX*), on es podrà veure com implementar noves tasques en el SO Erika (secció 4.2.1), com hem implementat la recepció i enviament dels diferents missatges per RS232 (secció 4.1.2), i el més important en els sistemes distribuïts la comunicació mitjançant el bus CAN (secció 4.2.3); la creació de màscares i filtres, la recepció dels diferents missatges, l'enviament d'aquests, i la creació de una pila per mantenir un llistat de tots els llargs de control presents al bus.

4.1 DCSMonitor

Com hem vist anteriorment, el programa **DCSMonitor** és l'encarregat de mostrar-nos en temps real el funcionament dels diferents controls als que estem connectats via RS232. Gràcies a ell podem monitoritzar un bus CAN (si estem connectats a un dispositiu *Monitor*), o podem comprovar que el control que estem executant sigui correcte (connectats directament a un dispositiu *Sensor/Actuador*). En aquest apartat

s'explica com s'han programat totes les parts implicades del laboratori que formen part del programa **DCSMonitor** , començant per l'aspecte visual, passant per les comunicacions i el mostreig dels plots, i acabant finalment per la integració amb varis idiomes.

Cal indicar que per crear les gràfiques en temps real s'ha pres com a exemple el codi de Yassine Benabbas amb Llicencia Creative Commons present en la bibliografia (11, Créer un graphe dynamique avec PyPlot).

4.1.1 Interficie

Pel disseny de la part visual del programa es va utilitzar la eina *QTCreator* , el qual és un IDE per realitzar interfícies de programes arrossegant i enganxant les diferents parts. Això ens permet guanyar temps en petits programes, i en programes grans és quasi impossible prescindir d'aquesta ajuda.

Un cop hem realitzat la organització de tots els botons i etiquetes que formen el programa, li indiquem a *QTCreator* que ens generi el codi font d'aquest en format *.ui* (es posa un fragment de codi com a exemple, 4.1, ja que el codi complert ocupa unes 700 línies, més o menys 25 pàgines)

Codi XML 4.1: Fragment del fitxer d'interficie dcsmmainwindow.ui

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow" name="MainWindow">
5     <property name="geometry">
6       <rect>
7         <x>0</x>
8         <y>0</y>
9         <width>752</width>
10        <height>790</height>
11      </rect>
12    </property>
13    <property name="windowTitle">
14      <string>Distributed Control Systems Monitor</string>
15    </property>
16    ...
```

4. IMPLEMENTACIÓ DEL LABORATORI

Un cop hem generat aquest fitxer és hora de traduir-ho a *Python* , per tal de que la llibreria que utilitzem *PyQT4* pugui generar l'interficie. Per fer aixó ens ajudem del programa *pyuic4* que amb la comanda 4.2 genera un fitxer com el del fragment 4.3

Codi Bash 4.2: Creant fitxer d'interficie

```
pyuic4 dcsmmainwindow.ui > dcsmmainwindow.py
```

Codi Python 4.3: Fragment de codi del fitxer dcsmmainwindow.py

```
1  try:
2      _fromUtf8 = QtCore.QString.fromUtf8
3  except AttributeError:
4      _fromUtf8 = lambda s: s
5
6  class Ui_MainWindow(object):
7      def setupUi(self, MainWindow):
8          MainWindow.setObjectName(_fromUtf8("MainWindow"))
9          MainWindow.resize(752, 790)
10         self.centralwidget = QtGui.QWidget(MainWindow)
```

Finalment es pot veure a la figura 4.2 el resultat d'aquests passos, amb una captura de la pantalla de l'execució.

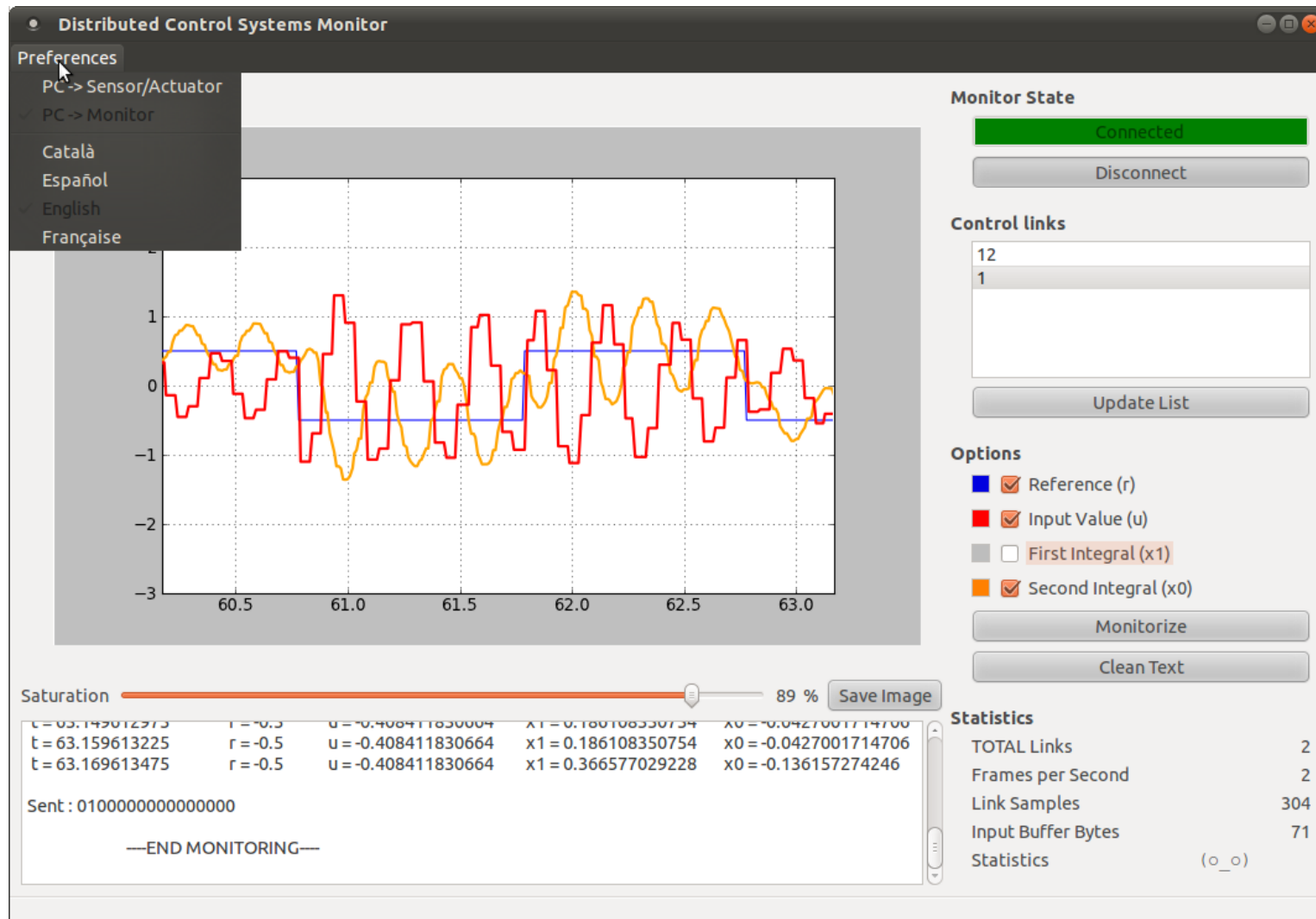


Figura 4.2: Interfície final del programa DCSMonitor - Captura de pantalla del programa DCSMonitor mostrant el control amb el bus CAN saturat al 89%, sense el plot de la primera integral i en anglès.

4. IMPLEMENTACIÓ DEL LABORATORI

4.1.2 Comunicació sèrie RS232

Ja hem vist en la secció de disseny dels tipus de missatge serie 3.2.3 els diferents missatges que han de enviar-se entre el programa **DCSMonitor** i els dispositius, ara veurem com hem implementat aquest tipus de comunicacions per la part de l'ordinador.

En aquest apartat indicarem com hem fet per configurar el port serie amb *PySerial*, els mètodes que hem utilitzat per enviar els senyals al dispositiu connectat, i com hem rebut els valors necessaris per poder mostrar les gràfiques o llistar els diferents llaços del bus CAN.

4.1.2.1 Obrir i tancar dispositiu sèrie

Per utilitzar aquest tipus de comunicació primer de tot hem de decidir alguns dels paràmetres que porta associats el tipus de comunicació serie. Com anteriorment en el laboratori ja es van ajustar aquests paràmetres el que farem és adaptar-nos a ells i configurar el dispositiu tal i com estava funcionant.

Per fer això primer de tot importem el paquet serial de *Python*, i tot seguit assignem els valors als diferents paràmetres (codi 4.4).

Codi Python 4.4: Codi per configurar dispositiu sèrie

```
1 # for communicate with pic
2 import serial
3
4 PORT = '/dev/ttyUSB0'
5 BAUDRATE = 115200
6 BYTESIZE = serial.EIGHTBITS
7 PARITY = serial.PARITY_NONE
8 STOPBITS = serial.STOPBITS_ONE
9 TIMEOUT = 10
```

Amb tots els paràmetres ben instanciats, ja podem esperar el senyal de connexió per activar aquest dispositiu. El metode que s'utilitza és mitjançant un botó a la interfície del programa, la qual està enllaçada amb un senyal. En el moment en el que el botó sigui premut, s'activarà la funció que li hem indicat (codi 4.5).

Codi Python 4.5: Disparador de connexió de port serie

```
1 QtCore.QObject.connect(self.pushButton_connect, QtCore.  
    SIGNAL("clicked()"), self.clicked_connect)
```

La funció amb la que està enllaçada el botó comprovarà si s'està activant o s'està desactivant. Si es prem per activar el dispositiu, provarà d'activar el port serie cridant a la funció *connect_serial* (codi 4.7, explicat més endavant), si això te èxit habilitarà tots els botons que estiguin permesos segons el mode d'execució en el que es trobi el programa (PC- > *Sensor/Actuador* o PC- > *Monitor*) (figura 4.3b), en cas contrari marcarà en vermell que no s'ha pogut connectar (figura 4.3d).

En el cas en que s'estigui desconnectant el dispositiu serie, aturarà tots els events periòdics que estiguin activats (recepció de dades, dibuix de la gràfica i calcul d'estadístiques), tanca el dispositiu serie, i finalment desactivarà tots els botons que no calen usar.

Codi Python 4.6: Codi per obrir/tancar dispositiu sèrie

```
1 def clicked_connect(self):  
2     if not self.pushButton_connect.isChecked():  
3         self.timer_graph.stop()  
4         self.timer_data.stop()  
5         self.timer_statistics.stop()  
6         self.ser.close()  
7         self.ser = 0  
8         self.lineEdit_state.setPalette(QtGui.QPalette(QtGui.  
            QColor("gray")))  
9         self.lineEdit_state.setText(QtGui.QApplication.  
            translate("MainWindow", "Disconnected", None,  
            QtGui.QApplication.UnicodeUTF8))  
10        self.pushButton_connect.setText(QtGui.QApplication.  
            translate("MainWindow", "Connect", None, QtGui.  
            QApplication.UnicodeUTF8))  
11        self.pushButton_monitor.setEnabled(0)  
12        self.pushButton_monitor.setChecked(0)  
13        self.pushButton_reload.setEnabled(0)  
14        self.slider_saturation.setEnabled(0)  
15  
16    else:
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
17     if self.connect_serial():
18         self.lineEdit_state.setPalette(QtGui.QPalette(
19             QtGui.QColor("green")))
20         self.lineEdit_state.setText(QtGui.QApplication.
21             translate("MainWindow", "Connected", None,
22             QtGui.QApplication.UnicodeUTF8))
23         self.pushButton_connect.setText(QtGui.
24             QApplication.translate("MainWindow", "
25             Disconnect", None, QtGui.QApplication.
26             UnicodeUTF8))
27         self.pushButton_monitor.setEnabled(1)
28         self.slider_saturation.setEnabled(1)
29         if self.actionPC_Monitor.isChecked():
30             self.pushButton_reload.setEnabled(1)
31     else:
32         self.lineEdit_state.setPalette(QtGui.QPalette(
33             QtGui.QColor("red")))
34         self.lineEdit_state.setText(QtGui.QApplication.
35             translate("MainWindow", "Error connecting",
36             None, QtGui.QApplication.UnicodeUTF8))
37         self.pushButton_connect.setChecked(0)
```

La funció *connect_serial* és l'encarregada de connectar el dispositiu serie, primer de tot comprova si el dispositiu ja havia estat engegat algun cop. Si no ho havia estat li assigna tots els paràmetres anteriorment instanciats, i posteriorment intenta connectar. En cas de error la funció retornarà 0, en cas contrari retornarà 1. En cas que el dispositiu ja hagués estat obert anteriorment, la configuració ja està instanciada, per tant només provarem d'obrir-la de nou. Igual que abans retornarà zero o u segons hagi fracassat o connectat amb èxit (codi 4.7).

Codi Python 4.7: Codi per connectar port serie

```
1 def connect_serial(self):
2     """Connects to the serial device, if not return 1
3     """
4     if self.ser == 0:
5         try:
```

```

5         self.ser = serial.Serial(PORT, BAUDRATE,
                                   BYTESIZE, PARITY, STOPBITS, TIMEOUT)
6         #self.ser = serial.Serial('/dev/ttyUSB0',
                                   115200, timeout=10)
7     except:
8         self.textBrowser.append(QtGui.
                                   QApplication.translate("MainWindow", "
                                   \n\t---Failed to connect to the
                                   device---\n", None, QtGui.
                                   QApplication.UnicodeUTF8))
9         return 0
10    # this code shouldn't be never executed
11    else:
12        try:
13            self.ser.open()
14        except:
15            self.textBrowser.append(QtGui.
                                    QApplication.translate("MainWindow", "
                                    \n\t---Failed to connect to the
                                    device---\n", None, QtGui.
                                    QApplication.UnicodeUTF8))
16            return 0
17    return 1

```

Tot seguit es pot veure els canvis que s'efectuen durant la connexió del port serie, en la primera imatge abans de connectar al dispositiu `/dev/ttyUSB0`, després un cop ja s'ha establert la connexió, un cop hem desconnectat correctament el port, i la ultima imatge quan hi ha algun problema de connexió (figura 4.3).

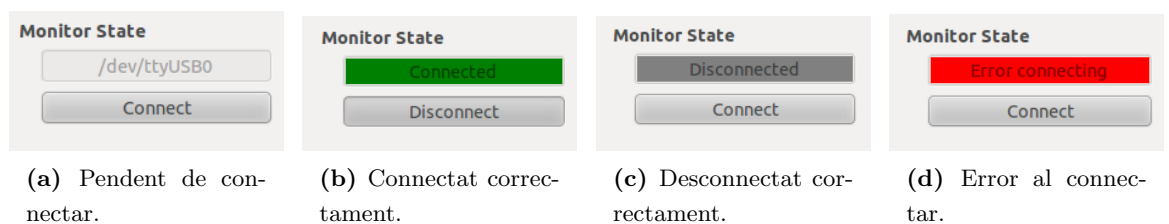


Figura 4.3: Estat de connexió amb el *Supervisor* via RS232.

4. IMPLEMENTACIÓ DEL LABORATORI

4.1.2.2 Enviar petició de llistar llaços de control

Al connectar el programa **DCSMonitor** amb un dispositiu *Monitor* el que ens interessa primer de tot és veure una llista dels diferents llaços de control existents al bus CAN. Aquesta acció només es pot iniciar quan el port serie esta degudament connectat i si el programa s'està executant en mode PC- >Monitor, per aquest motiu el botó que permet fer aquesta crida romandrà si no es compleixen aquestes dues premisses.

Per activar-ho s'ha creat un disparador que enllaça el botó *Llistar* amb la funció *toogled_reload* (codi del disparador 4.8).

Codi Python 4.8: Disparador de botó llistar dispositius

```
1 QtCore.QObject.connect(self.pushButton_reload, QtCore.  
    SIGNAL(_fromUtf8("clicked()")), self.toggled_reload)
```

Primer de tot al entrar a aquesta funció (codi 4.9) comprova que estem correctament connectats al port serie, si no és així atura aquest procés. Si en canvi tot és correcte, comprova si estem aturant o estem iniciant la recepció d'identificadors, així que segons el que estem realitzant codificarà un o un altre missatge per després enviar-lo pel port serie (les codificacions és poden veure en la figura 3.21 del capítol de disseny)

En el cas de començar a demanar els identificadors, s'activa una alarma periòdica la qual està enllaçada a una funció que captura els missatges que es reben al port serie. Aquesta recepció es pot veure en la secció 4.1.2.6. En el cas de aturar aquesta petició, comprovarà si s'està monitoritzant, en cas negatiu aturarà la recepció de dades.

Codi Python 4.9: Codi per demanar una llista de llaços de control

```
1 def toggled_reload(self):  
2     """Stops or Start to recive devices id's"""  
3     if not self.connect_serial():  
4         self.pushButton_reload.setChecked(0)  
5         return  
6  
7     if self.pushButton_reload.isChecked():  
8         self.listWidget_link.clear()  
9         word = struct.pack("BBBBBBB", ID_DEVICES  
10            ,0,0,0,0,0,0,0)  
10         self.timer_data.start(DATA_TIME)
```

```
11
12     else:
13         word = struct.pack("BBBBBBBB", ID_DEVICES +
14                               ID_STOP, 0, 0, 0, 0, 0, 0, 0)
15         if (not self.pushButton_monitor.isChecked()):
16             self.timer_data.stop()
17
18     self.textBrowser.append(QtGui.QApplication.translate(
19         "MainWindow", "Sent : ", None, QtGui.QApplication.
20         UnicodeUTF8)+binascii.hexlify(word)+"\n")
21
22     self.ser.write(word)
```

4.1.2.3 Enviar petició de monitorització

Aquesta és la acció principal sobre la que es basa el programa, en el moment que volem veure la gràfica en temps real d'un llaç de control hem d'enviar aquesta petició al *Monitor* . Però també podem estar connectats a un dispositiu *Sensor/Actuador* , però en aquest cas ; com ja hem explicat en seccions anteriors (secció 3.2); aquesta petició es perdrà, però posarem en marxa tota la part de recepció de dades.

Primer de tot creem el disparador que detectarà que s'ha premut el botó per monitoritzar (codi 4.10), aquest cridarà la funció *toogled_monitor* (codi 4.11).

Codi Python 4.10: Disparador de botó per monitoritzar

```
1 QtCore.QObject.connect(self.pushButton_monitor, QtCore.
    SIGNAL("clicked()"), self.toggled_monitor)
```

Aquesta serà la funció detectarà si el que volem és aturar o començar la monitorització, en cas d'iniciar-la comprovarà si hi ha un identificador de llaç de control de la llista seleccionat. Si hi és formarem el missatge amb aquest identificador, en cas contrari monitoritzarem el llaç de control numero u. Tot seguit enviarem aquest senyal per el port serie, netejarem el buffer d'entrada per no confondre algun senyal residual anterior amb un de nou. Netegem la gràfica i activem una funció (*get_periodical_data*, veure codi 4.15) que rep periòdicament informació per el port serie i la emmagatzema en arrays (els quals ens permetran dibuixar la gràfica), i activa una altre tasca periòdica

4. IMPLEMENTACIÓ DEL LABORATORI

que en cas de existir dades noves en els arrays tornarà a dibuixar la gràfica (secció de generació de gràfiques en temps real 4.1.3). En el cas que estem parant la monitorització aturarem la tasca periòdica que dibuixava la gràfica, i si no s'estan rebent identificadors també aturarem la recepció de dades.

Codi Python 4.11: Codi per demanar monitoritzar un llaç de control

```
1 def toggled_monitor(self):
2     """Starts or Stops reciving data"""
3     if self.pushButton_monitor.isChecked():
4
5         if not self.connect_serial():
6             self.pushButton_monitor.setChecked(0)
7             return
8
9         self.textBrowser.insertPlainText(QtGui.QApplication.
            translate("MainWindow", "\n\t----NEW MONITORING
            ----\n", None, QtGui.QApplication.UnicodeUTF8))
10
11         id_link = self.listWidget_link.selectedItems()
12         if len(id_link):
13             id_link = int(id_link[0].text())
14         else:
15             id_link = 1
16
17         self.textBrowser.insertPlainText(QtGui.QApplication.
            translate("MainWindow", "link to monitor : ", None
            , QtGui.QApplication.UnicodeUTF8)+str(id_link)+"\n
            ")
18
19         word = struct.pack("BBBBBBBB", ID_MONITOR
            ,0,0,0,0,0,0,id_link&0xFF)
20
21         self.textBrowser.append(QtGui.QApplication.translate(
            "MainWindow", "Sent : ", None, QtGui.QApplication.
            UnicodeUTF8)+binascii.hexlify(word)+"\n")
22
```

```
23     self.ser.write(word)
24
25     # clear data of the serial port
26     self.ser.flushInput()
27     # clear graph
28     self.clean_graph()
29     # start timer data reception
30     self.get_periodical_data()
31     # start timer graphic refresh
32     self.update_graph()
33
34     self.pushButton_save.setEnabled(False)
35 else:
36     # send Monitor
37     word = struct.pack("BBBBBBBB", ID_MONITOR+ID_STOP
38                          ,0,0,0,0,0,0,0)
39
40     self.textBrowser.append(QtGui.QApplication.translate(
41         "MainWindow", "Sent : ", None, QtGui.QApplication.
42         UnicodeUTF8)+binascii.hexlify(word)+"\n")
43
44     self.ser.write(word)
45
46     self.ser.flushInput()
47
48     # reload number of waiting chars in serial rx buffer
49     self.label_rx_buff_value.setText(str(self.ser.
50         inWaiting()))
51
52     self.textBrowser.insertPlainText(QtGui.QApplication.
53         translate("MainWindow", "\n\t----END MONITORING
54         ----\n", None, QtGui.QApplication.UnicodeUTF8))
55
56     self.pushButton_save.setEnabled(True)
57
58     if (not self.pushButton_reload.isChecked()):
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
53         self.timer_data.stop()
54         self.timer_graph.stop()
```

4.1.2.4 Enviar percentatge de càrrega

Igual que en els casos anteriors per enviar aquest missatge al dispositiu *Monitor* ho farem a través d'un disparador (codi 4.12). Però així com en els anteriors casos s'havia de prémer un botó per activar-ho, en aquest cas hi ha una barra que en canviar de posició és la que agafarà el seu valor (veure figura fig:comparacio:barra) i cridarà a la funció *percent_changed* (codi de la funció 4.13). La barra que activa aquest enviament només està habilitada en el cas que el programa **DCSMonitor** estigui en mode PC- >Monitor.

Codi Python 4.12: Disparador de barra de càrrega

```
1  QtCore.QObject.connect(self.slider_saturation, QtCore.
    SIGNAL(_fromUtf8("valueChanged(int)")), self.
    percent_changed)
```

Aquesta funció comprova abans de res si estem degudament connectats al port serie, si no ho estem s'atura, en cas positiu prepara el missatge de la forma vista en la secció de disseny 3.2.3.3, i li envia pel port serie. Aquesta funció no ha d'esperar cap tipus de dades per tant no activa cap tasca periòdica.

Codi Python 4.13: Codi per demanar que saturi el bus CAN

```
1  def percent_changed(self, num):
2      """Change de label percent value and send this
        information to the Monitor Flex"""
3      self.label_percent_value.setNum(num)
4
5      if not self.connect_serial():
6          return
7
8      if num != 0:
9          word = struct.pack("BBBBBBBB", ID_PERCENT
                                ,0,0,0,0,0,0,num)
```

```

10     else:
11         word = struct.pack("BBBBBBBB", ID_PERCENT+ID_STOP
12                               ,0,0,0,0,0,0,0)
13
14     self.textBrowser.append(QtGui.QApplication.translate(
15         "MainWindow", "Sent : ", None, QtGui.QApplication.
16         UnicodeUTF8)+binascii.hexlify(word)+"\n")
17
18     self.ser.write(word)

```

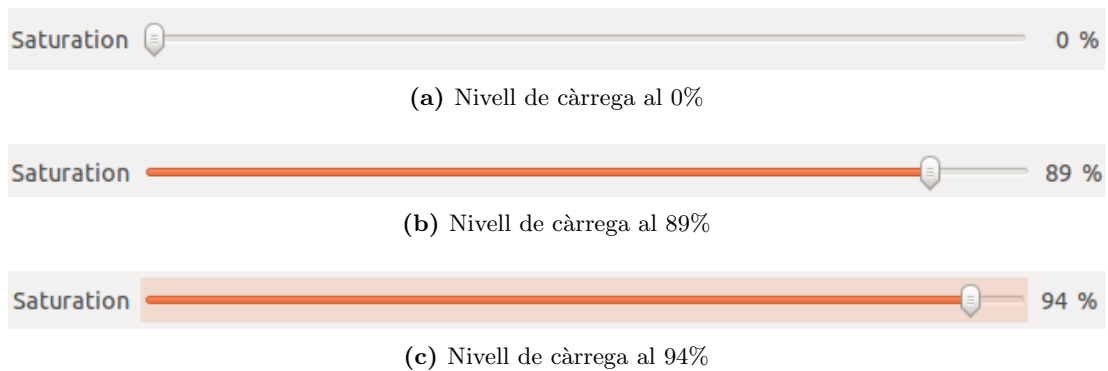


Figura 4.4: Barra lliscant de càrrega en diferents posicions.

4.1.2.5 Recepció de valors monitoritzats

Per poder realitzar la recepció de moltes dades pel port serie, es fa necessari deixar algun proces escoltant contínuament al port serie, però no podem permetre que el nostre programa es dediqui exclusivament a aquesta tasca, ja que tenim una interfície que hem d'atendre, gràfiques a dibuixar, i botons, que poden ser activats o desactivats. Per aquesta raó primer de tot creem un temporitzador, que permeti fer aquesta recepció de manera intermitent, deixant temps per atendre les altres tasques. Per fer això creem aquest temporitzador, i li assignem un disparador, que activarà la funció *get_periodical_data* cada cop que passi el temps que li indiquem (codi 4.14).

Codi Python 4.14: Temporitzador i disparador per rebre dades del port serie

```

1 timer_data = QtCore.QTimer()
2

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
3 # Periodical tasks
4 QtCore.QObject.connect(self.timer_data, QtCore.SIGNAL("
    timeout()"), self.get_periodical_data)
```

Peró aquesta declaració que hem creat no s'activa per si sola, sinó que hem d'engegar el temporitzador quan el necessitem, per tant cada cop que necessitem rebre informació de un llaç de control, o que volem una llista dels diferents llaços que hi ha al bus CAN al clicar el botó adequat (i després de realitzar codi que hem explicat en anteriors seccions) acabarà cridant a la funció *get_periodical_data* (codi 4.15).

Aquesta funció és l'encarregada de comprovar que la temporització funciona correctament, reactivant-la o parant-la en el moment oportú. El que fa primer de tot es comprovar que un dels botons que indiquen rebre informació estigui premut (aquests botons es queden premuts fins que no es torna a clicar per segon cop), en cas contrari ha d'aturar aquesta periodicitat i per tant ja no activa el temporitzador. Un cop sap que realment es vol rebre informació, comprova que s'hagi dibuixat ja la informació que hi havia pendent, si ja s'ha dibuixat torna a rebre dades cridant a la funció *recv_data* (codi de la funció 4.16). Això és així per crear una zona d'exclusió mútua (també anomenada Mutex), que evitarà que intentem dibuixar la gràfica de dades que estan a mig omplir. I finalment reactiva el temporitzador per tornar a executar la mateixa funció.

Codi Python 4.15: Codi per reactivar l'adquisició de dades

```
1 def get_periodical_data(self):
2     """Updates the data"""
3     if not self.pushButton_monitor.isChecked() and not
4         self.pushButton_reload.isChecked():
5         return
6
7     if self.updated_data == 0 :
8         # receive new serial data
9         self.recv_data()
10
11     # activate the periodical reception of data
12     self.timer_data.start(DATA_TIME)
```

Finalment la funció encarregada en rebre les dades pel port serie és la funció *receive_data* (codi 4.16). Aquesta funció procura agafar tota la informació que hi hagi pendent al buffer d'entrada del port serie. Tenint en compte que el dispositiu *Monitor* i el *Sensor/Actuador* envien aquesta informació cada 10 milisegons, i nosaltres fem lectures com a mínim cada 5 milisegons, tenim temps per fer la lectura i seguir executant altres codis. Primer de tot busca el byte de capçalera; aquest byte és un 0x01 que està al principi de tots els missatges que el dispositiu ens envia. Un cop ha trobat aquest byte sap que te les dades alineades, i pot prosseguir a llegir el numero de bytes que li correspongui (segons estigui en mode PC- >*Sensor/Actuador* o PC- >*Monitor* llegirà 23 o 71 bytes respectivament). Ja amb el numero de bytes llegits, comprova si el botó de monitorització esta premut, per tal de agafar les dades per dibuixar la gràfica o passar a rebre la llista de dispositius (el codi que falta està en la secció 4.1.2.6, codi 4.17).

En la majoria de casos aquest botó si que està premut, i per tant agafa els bytes en l'ordre correcte per formar totes les variables necessàries, prepara el text per mostrar per pantalla i l'escriu en la part inferior del programa (figura 4.5). Després afegeix als arrays el nou valor, i en treu un si ha arribat al màxim. Després modifica els valors que apareixen en les estadístiques i habilita la zona d'exclusió.

Codi Python 4.16: Codi per rebre els valors monitoritzats

```
1 def receive_data(self):
2     """Get messages from the serial port"""
3     # read all available data
4     while self.ser.inWaiting() > self.INPUT_DATA_SIZE+1:
5         data = array.array('c')
6         # search the header
7         data.append(self.ser.read(1))
8         while data[0] != chr(1):
9             data[0] = self.ser.read(1)
10
11        # wait for all available data
12        while self.ser.inWaiting() < (self.
13            INPUT_DATA_SIZE-1):
14            time.sleep(0.03);
```

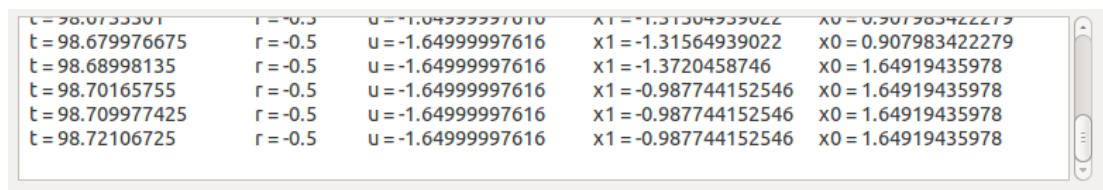
4. IMPLEMENTACIÓ DEL LABORATORI

```
15         # recives data
16         data = self.ser.read(self.INPUT_DATA_SIZE-1)
17
18         # prove if you want graphical data
19         if self.pushButton_monitor.isChecked():
20             # decodes the data
21             t = struct.unpack('I', data[3]+data[2]+data
22                               [1]+data[0])
23             r = struct.unpack('f', data[4]+data[5]+data
24                               [6]+data[7])
25             x0 = struct.unpack('f', data[8]+data[9]+data
26                                [10]+data[11])
27             x1 = struct.unpack('f', data[12]+data[13]+
28                                data[14]+data[15])
29             u = struct.unpack('f', data[16]+data[17]+
30                                data[18]+data[19])
31
32             self.time = t[0]*25e-9
33
34             # prepare the string output
35             aux_str = " t = "+str(self.time)+"\t"
36             aux_str += " r = "+str(r[0])+"\t"
37             aux_str += " u = "+str(u[0])+"\t"
38             aux_str += " x1 = "+str(x1[0])+"\t"
39             aux_str += " x0 = "+str(x0[0])+"\n"
40             # print string output
41             self.textBrowser.insertPlainText(aux_str)
42
43             # append data to the arrays
44             self.graf_t.append(self.time)
45             self.graf_r.append(r[0])
46             self.graf_x0.append(x0[0])
47             self.graf_x1.append(x1[0])
48             self.graf_u.append(u[0])
```

```

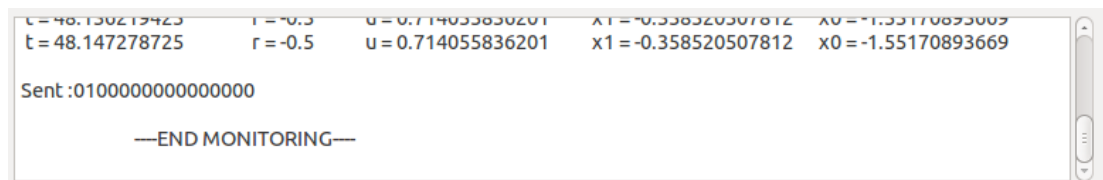
45         # remove one value if the arrays have maximum
           length
46     if self.graf_t.buffer_info()[1] >=
        NUM_SAMPLES:
47         self.graf_t.pop(0)
48         self.graf_r.pop(0)
49         self.graf_x0.pop(0)
50         self.graf_x1.pop(0)
51         self.graf_u.pop(0)
52
53         # reload number of samples lavel
54     self.label_samples_value.setText(str(self.
        graf_t.buffer_info()[1]))
55     # reload number of waiting chars in serial rx
        buffer
56     self.label_rx_buff_value.setText(str(self.ser
        .inWaiting()))
57
58     # reload mutex area
59     self.updated_data = 1

```



t=98.679976675	r=-0.5	u=-1.64999997616	x1=-1.31564939022	x0=0.907983422279
t=98.68998135	r=-0.5	u=-1.64999997616	x1=-1.3720458746	x0=1.64919435978
t=98.70165755	r=-0.5	u=-1.64999997616	x1=-0.987744152546	x0=1.64919435978
t=98.709977425	r=-0.5	u=-1.64999997616	x1=-0.987744152546	x0=1.64919435978
t=98.72106725	r=-0.5	u=-1.64999997616	x1=-0.987744152546	x0=1.64919435978

(a) Rebut els valors d'una monitorització.



Sent:0100000000000000	—END MONITORING—
-----------------------	------------------

(b) Havent finalitzat una monitorització.

Figura 4.5: Espai de text per valors monitoritzats.

4. IMPLEMENTACIÓ DEL LABORATORI

4.1.2.6 Recepció de llista de dispositius

Per la recepció dels identificadors dels llaços de control com hem vist en l'apartat anterior 4.1.2.2 s'activa la mateixa funció periòdica per rebre informació, per tant el codi que segueix és la continuació de la funció vista anteriorment *receive_data* (codi 4.16). Per tant seguint l'execució d'aquesta funció abans mencionada, comprovem si el byte 20 de les dades que ens arriben està a valor dos. En aquest cas significa que en el següent byte ve el nombre de identificadors diferents de mida un byte que el segueixen. Així que comprova un a un si ja es trobaven a la llista, i si no és així els afegeix.

Codi Python 4.17: Codi per rebre la llista de dispositius

```
1  ....
2  # prove if there are available id's
3  if (self.actionPC_Monitor.isChecked() and data[20] == chr
    (2)):
4      # if it is true, looks how much id's
5      i = struct.unpack('B', data[21])
6
7      if i[0] < STACK_SIZE:
8          for z in range(i[0]):
9              new_device = struct.unpack('B', data[z+22])
10             new_string = str(new_device[0])
11
12             llista = self.listWidget_link.findItems(
13                 new_string, QtCore.Qt.MatchExactly)
14             if len(llista) == 0:
15                 self.listWidget_link.addItem(new_string)
```

Es poden veure com a exemple algunes captures de pantalla de la llista abans de començar a demanar-li els identificadors (figura 4.6a), actualitzant la llista en un bus amb dos llaços de control (figura 4.6b), i la ultima imatge havent demanat la llista en un bus on existien varis llaços de control (figura 4.6c).

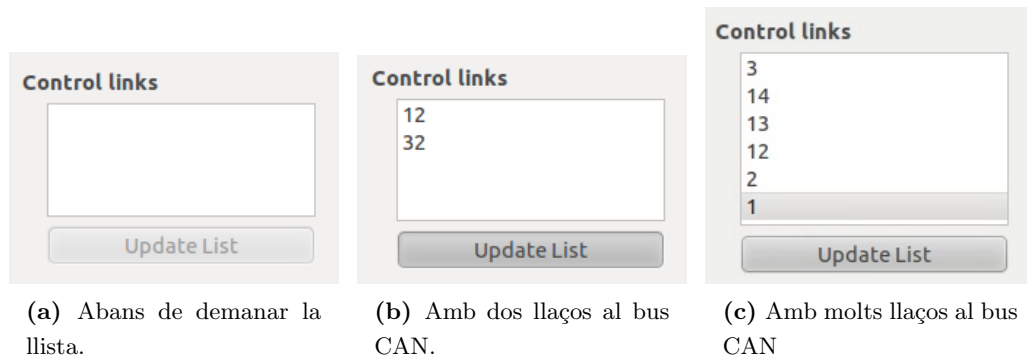


Figura 4.6: Llista de llaços de control al bus CAN.

4.1.3 Generació de gràfiques en temps real

Generar aquestes gràfiques és un dels propòsits més importants del programa, i per tant ha de ser prou ràpida per no saturar el programa però al mateix temps donar una sensació visual de moviment. Aquesta generació està associada a un temporitzador específic, enllaçat a un disparador que activa la funció *update_graph* (codi 4.19). Aquest temporitzador s'activa només quan es prem el botó de monitorització, vist en la secció 4.1.2.3.

Codi Python 4.18: Temporitzador i disparador per dibuixar la gràfica

```
1 timer_graph = QtCore.QTimer()
2
3 # Periodical tasks
4 QtCore.QObject.connect(self.timer_graph, QtCore.SIGNAL("
    timeout()"), self.update_graph)
```

Un cop s'ha activat el temporitzador que crida aquesta funció, primer de tot comprova que hi hagi dades noves per dibuixar i que no s'estiguin modificant en aquest precís instant (això s'ha vist en la secció de recepció de dades 4.1.2.5). Si les dades que hi ha han estat actualitzades dibuixa primer les línies verticals i horitzontals que hi ha darrera dels plots. I després comprova un a un si hi ha marcada la opció de dibuixar els diferents plots (referència, primera i/o segona integral i valor d'entrada; figura 4.7); segons quins estiguin activats, els crea i dibuixa o no. Finalment prova de pintar tot el que ha generat, actualitza la variable *frames per second* per indicar que s'ha fet un

4. IMPLEMENTACIÓ DEL LABORATORI

nou dibuix i es marca la variable per la zona d'exclusió mútua perquè es puguin seguir afegint dades als arrays. Finalment comprova que encara es vulgui seguir dibuixant la gràfica i es reactiva la temporització.

Codi Python 4.19: Codi per actualitzar la gràfica

```
1 def update_graph(self):
2     """Updates the graph"""
3     # looks for new data
4     if self.updated_data == 1:
5
6         self.mplWidget.canvas.ax.set_xbound(self.time-
7             XLIM, self.time)
8         self.mplWidget.canvas.draw()
9
10        try:
11            #Draw the lines
12            if self.checkBox_R.isChecked():
13                self.referenceLine.set_data(self.graf_t,
14                    self.graf_r)
15                self.mplWidget.canvas.ax.draw_artist(self
16                    .referenceLine)
17            if self.checkBox_x0.isChecked():
18                self.x0Line.set_data(self.graf_t, self.
19                    graf_x0)
20                self.mplWidget.canvas.ax.draw_artist(self
21                    .x0Line)
22            if self.checkBox_U.isChecked():
23                self.uLine.set_data(self.graf_t, self.
24                    graf_u)
25                self.mplWidget.canvas.ax.draw_artist(self
26                    .uLine)
27            if self.checkBox_x1.isChecked():
28                self.x1Line.set_data(self.graf_t, self.
29                    graf_x1)
30                self.mplWidget.canvas.ax.draw_artist(self
31                    .x1Line)
```

```
23         except AssertionError:
24             pass
25
26         try:
27             self.mplWidget.canvas.blit(self.mplWidget.
28                                         canvas.ax.bbox)
29         except AttributeError:
30             pass
31
32         self.fps = self.fps+1
33
34         self.updated_data = 0
35
36     if self.pushButton_monitor.isChecked() == 1:
37         # activate the periodical update graph
38         self.timer_graph.start(GRAPH_REFRESH)
```

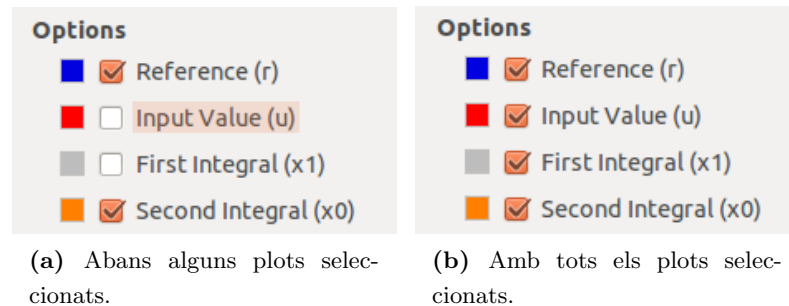


Figura 4.7: Opció de plots per la gràfica.

4.1.4 Exportar gràfica

Exportar les gràfiques que s'estiguin monitoritzant en temps real és molt important, ja que gràcies a això es pot guardar una captura de cada grup del laboratori, o generar imatges vectorials fàcilment introduïbles en articles o llibres (formats com .eps, .svg, .ps o .pdf). Per poder generar aquestes imatges s'espera que es premi un botó, que està enllaçat mitjançant un disparador (codi 4.20) a la funció que genera la imatge

4. IMPLEMENTACIÓ DEL LABORATORI

save_image (codi 4.21).

Codi Python 4.20: Disparador per exportar la gràfica

```
1 QtCore.QObject.connect(self.pushButton_save, QtCore.  
    SIGNAL(_fromUtf8("clicked()")), self.save_image)
```

Un cop han activat aquesta funció primerament s'obra una finestra nova on es pot introduir el nom del nou fitxer a crear (figura 4.8). A part de poder introduir el nom, podem llistar tots els fitxers dels diferents tipus que el programa és capaç de exportar, que hi hagin en el directori que estem observant. Per aquest motiu en la creació de la finestra apareixen descripcions d'aquests tipus de fitxers (figura 4.9).

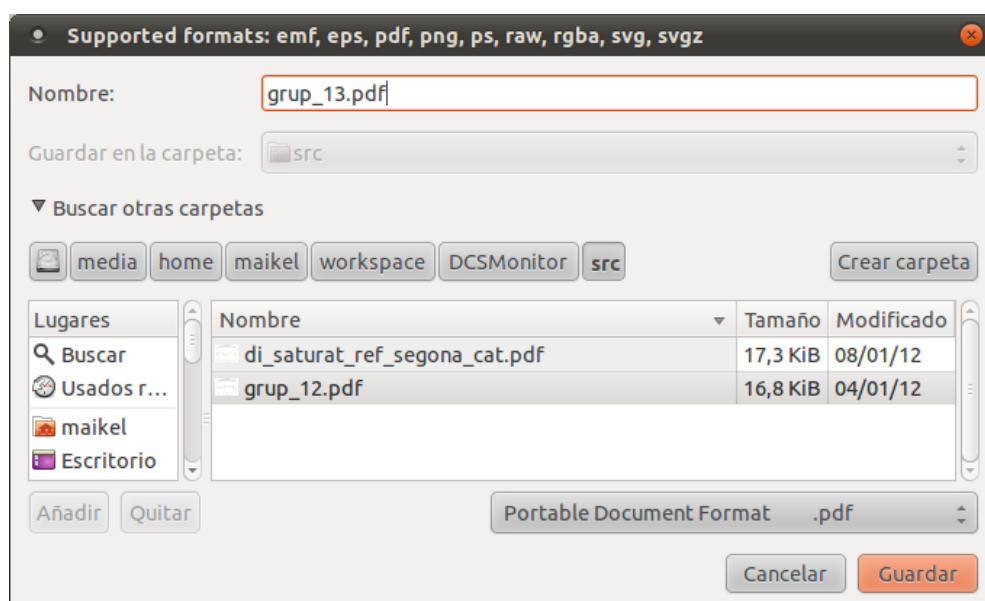


Figura 4.8: Finestra per posar nom a la gràfica exportada -

Un cop l'usuari ha introduït el nom i clicat a acceptar començarem a generar una gràfica nova des de zero. Comprovarem un a un tots els plots que hem de dibuixar, afegint a la gràfica els que calguin. Tot seguit indicarem quin rang de valors volem dibuixar, i posarem títol, subtítol, i noms als diferents eixos (tots aquests textos són traduïts a l'idioma que hi hagi seleccionat en el programa). Després assigna algunes de les mides del dibuix, i finalment intenta guardar la imatge amb el nom i format que li hagim indicat anteriorment. Si per alguna raó fallés en aquesta tasca, procuraria

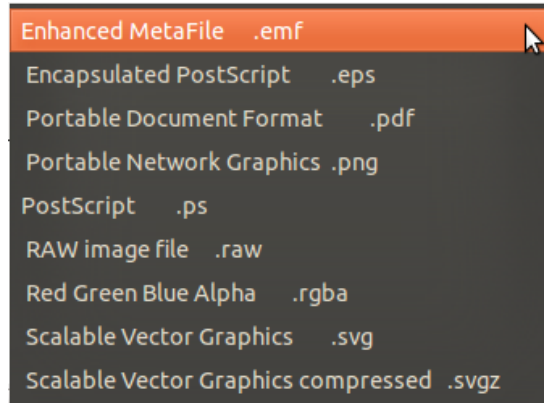


Figura 4.9: Tipus de fitxer als que es pot exportar -

guardar-lo en format *.svg*. Al acabar de generar la imatge aquesta gràfica nova es neteja per alliberar la memòria.

Un cop finalitzat el programa, el resultat de la captura de imatges és el que es pot observar en la figura 4.10, en la que apareix una gràfica generada en anglès, amb totes les línies, i una altre en català només amb les línies de referència i de sortida del doble integrador.

Codi Python 4.21: Codi per generar imatge d'una gràfica

```

1  def save_image(self):
2      """Exports the graph into an image"""
3      # open the dialog and get the selected file
4      file_one = QtGui.QFileDialog.getSaveFileName(
5          self,
6          QtGui.QApplication.translate(
7              "MainWindow",
8              "Supported formats:",
9              None,
10             QtGui.QApplication.UnicodeUTF8)+
11             " emf, eps, pdf, png, ps, raw, rgba, svg,
12             svgz",
13             "",
14             "Enhanced MetaFile \t.emf (*.emf);; "+
15             "Encapsulated PostScript \t.eps (*.eps);; "+
16             "Portable Document Format \t.pdf (*.pdf);; "+

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
16         "Portable Network Graphics \t.png (*.png);;" +
17         "PostScript \t.ps (*.ps);;" +
18         "RAW image file \t.raw (*.raw);;" +
19         "Red Green Blue Alpha \t.rgba (*.rgba);;" +
20         "Scalable Vector Graphics \t.svg (*.svg);;" +
21         "Scalable Vector Graphics compressed \t.svgz (*.
            svgz)")
22 # if a file is selected
23 if file_one:
24     # update the QLineEdit text with the selected
        filename
25     self.textBrowser.append(
26         QtGui.QApplication.translate(
27             "MainWindow",
28             "Saving to ",
29             None,
30             QtGui.QApplication.UnicodeUTF8
31             )+file_one)
32
33     plt.clf()
34     # adding different plots if required
35     if self.checkBox_R.isChecked():
36         lines = plt.plot(self.graf_t, self.graf_r)
37         plt.setp(lines[0],
38                 lw=1,
39                 label=str(
40                     QtGui.QApplication.translate(
41                         "MainWindow",
42                         "Reference",
43                         None,
44                         QtGui.QApplication.
45                             UnicodeUTF8)),
46                 color = 'blue')
47
48     if self.checkBox_U.isChecked():
49         lines = plt.plot(self.graf_t, self.graf_u)
```

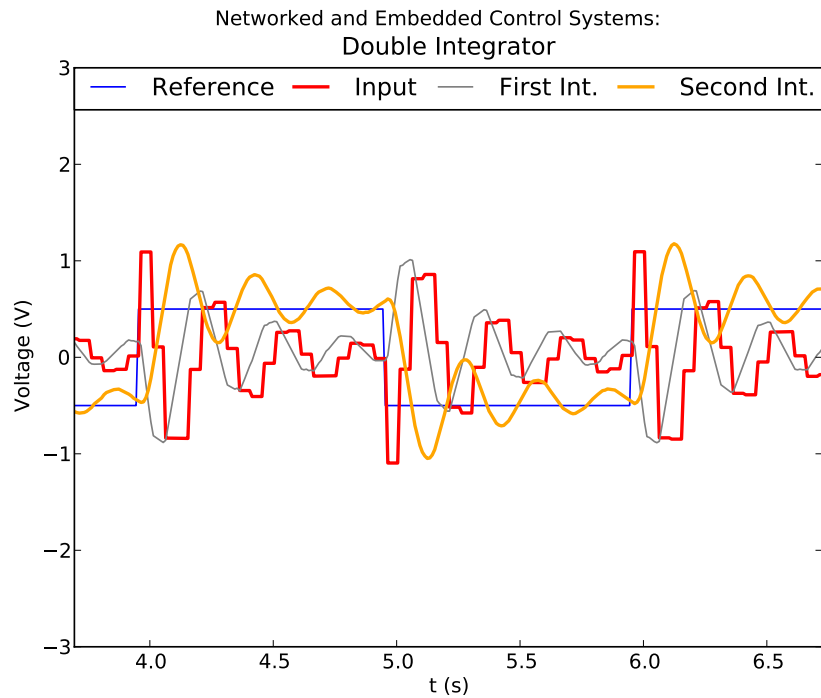
```
49         plt.setp(lines[0],
50                 lw=2,
51                 label=str(
52                     QtGui.QApplication.translate(
53                         "MainWindow",
54                         "Input",
55                         None,
56                         QtGui.QApplication.
57                             UnicodeUTF8)),
58                 color='red')
59
60     if self.checkBox_x1.isChecked():
61         lines = plt.plot(self.graf_t, self.graf_x1)
62         plt.setp(lines[0],
63                 lw=1,
64                 label=str(
65                     QtGui.QApplication.translate(
66                         "MainWindow",
67                         "First Int.",
68                         None,
69                         QtGui.QApplication.
70                             UnicodeUTF8)),
71                 color='grey')
72
73     if self.checkBox_x0.isChecked():
74         lines = plt.plot(self.graf_t, self.graf_x0)
75         plt.setp(lines[0],
76                 lw=2,
77                 label=str(
78                     QtGui.QApplication.translate(
79                         "MainWindow",
80                         "Second Int.",
81                         None,
82                         QtGui.QApplication.
83                             UnicodeUTF8)),
84                 color='orange')
```

4. IMPLEMENTACIÓ DEL LABORATORI

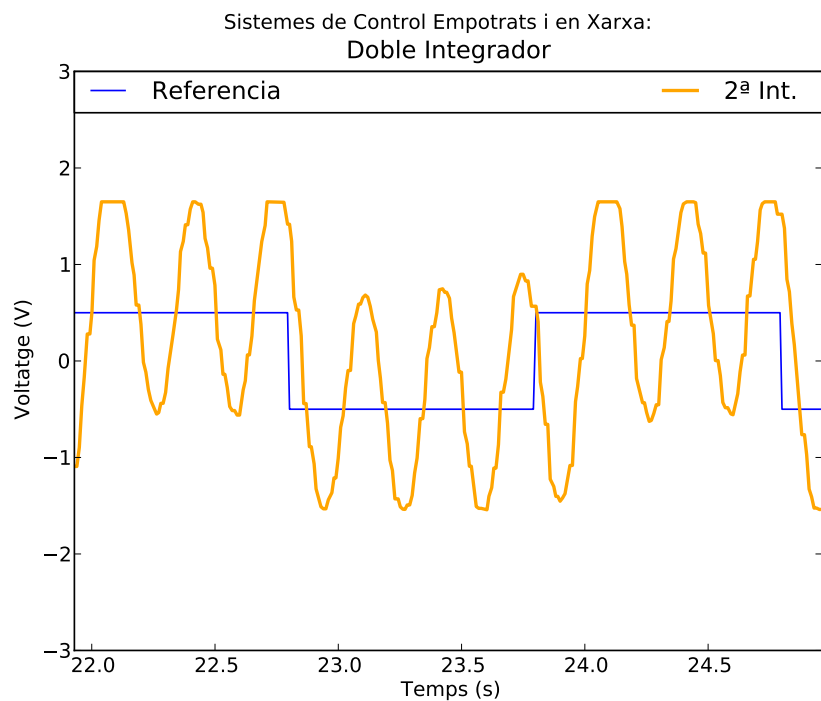
```
82
83     plt.axis([self.graf_t[0], self.graf_t[-1], -3,
84              3])
85
86     plt.title(str(
87         QtGui.QApplication.translate(
88             "MainWindow",
89             "Double Integrator",
90             None,
91             QtGui.QApplication.UnicodeUTF8)))
92
93     plt.suptitle(str(
94         QtGui.QApplication.translate(
95             "MainWindow",
96             "Networked and Embedded Control Systems:",
97             None,
98             QtGui.QApplication.UnicodeUTF8)))
99
100    plt.xlabel(str(
101        QtGui.QApplication.translate(
102            "MainWindow",
103            "t (s)",
104            None,
105            QtGui.QApplication.UnicodeUTF8)))
106
107    plt.ylabel(str(
108        QtGui.QApplication.translate(
109            "MainWindow",
110            "Voltage (V)",
111            None,
112            QtGui.QApplication.UnicodeUTF8)))
113
114    plt.legend(bbox_to_anchor=(0, 1, 1, 0), loc=1,
115              ncol=6, mode="expand", borderaxespad=0.)
```

```
115         filename = str(file_one)# + '.svg'
116     try:
117         plt.savefig(filename, dpi=300)
118     except ValueError as e:
119         print(e)
120         self.textBrowser.append(str(e))
121         self.textBrowser.append(
122             QtGui.QApplication.translate(
123                 "MainWindow",
124                 "Trying to save as .svg",
125                 None,
126                 QtGui.QApplication.UnicodeUTF8))
127
128         filename = str(file_one) + '.svg'
129         plt.savefig(filename, dpi=300)
130         pass
131     except ImportError as e:
132         print(e)
133         self.textBrowser.append(str(e))
134         self.textBrowser.append(
135             QtGui.QApplication.translate(
136                 "MainWindow",
137                 "Trying to save as .svg",
138                 None,
139                 QtGui.QApplication.UnicodeUTF8))
140
141         filename = str(file_one) + '.svg'
142         plt.savefig(filename, dpi=300)
143         pass
144     except:
145         self.textBrowser.append("Error al guardar la
146                                 imatge")
147
148 plt.clf()
```

4. IMPLEMENTACIÓ DEL LABORATORI



(a) Amb textos en anglès, totes els plots i en format PDF.



(b) Amb el bus CAN carregat de missatges, amb textos en català, només amb plots de referència i segona integral i en format PDF.

Figura 4.10: Diferents gràfiques generades amb el programa **DCSMonitor** del control d'un doble integrador.

4.1.5 Mostrant estadístiques

Com en la part de disseny es va explicar, periòdicament anirem creant estadístiques per saber que tot funciona correctament. Els valors que ens interessa saber en cada instant són:

- Nombre de llaços de control que hi ha connectats al bus CAN.
- Imatges per segon que es mostren en la gràfica en temps real.
- Nombre de mostres per plot dibuixat.
- Nombre de bytes esperant al buffer d'entrada del port serie.
- Marca canviant per saber que les estadístiques s'estan actualitzant periòdicament.

Per fer que aquestes estadístiques s'actualitzin periòdicament, com hem vist en altres seccions primer de tot hem de crear un temporitzador, i tot seguit li assignarem un disparador que activarà la funció *update_statistics* (codi del disparador, i activació de les estadístiques 4.22).

Codi Python 4.22: Temporitzador i disparador per actualitzar les estadístiques.

```
1 timer_statistics = QtCore.QTimer()
2
3 # Periodical tasks
4 QtCore.QObject.connect(self.timer_statistics, QtCore.
    SIGNAL("timeout()"), self.update_statistics)
5
6 self.timer_statistics.start(STAT_REFRESH)
```

El que fa la funció *update_statistics* (codi 4.23) és primer de tot comprovar que el port serie estigui degudament connectat, si no és així no modifica cap valor estadístic, en cas contrari comença a actualitzant el nombre de mostres de tots els plots, el nombre de bytes pendents al port serie, el nombre d'imatges per segon que estan apareixent a la gràfica (si aquesta està aturada, força un dibuix cada cop, per si hi ha algun problema), actualitza el valor canviant per indicar que aquesta funció està funcionant correctament, actualitza el nombre de llaços de control que hi ha al bus CAN i finalment torna a reactivar la funció al cap de mig segon.

4. IMPLEMENTACIÓ DEL LABORATORI

Codi Python 4.23: Codi per generar i actualitzar les estadístiques.

```
1 def update_statistics(self):
2     """Update value of labels in statistics"""
3     if self.ser != 0:
4         # reload number of samples lavel
5         self.label_samples_value.setText(str(self.graf_t.
6             buffer_info()[1]))
7         # reload number of waiting chars in serial rx
8         buffer
9         self.label_rx_buff_value.setText(str(self.ser.
10             inWaiting()))
11
12         self.label_fps_value.setText(str(self.fps*2))
13
14         self.fps = 0
15
16         if self.pushButton_monitor.isChecked() == 0:
17             self.force_update_graph()
18
19         if self.label_Est_value.text() != '(>_<)':
20             self.label_Est_value.setText('(>_<)')
21         else:
22             self.label_Est_value.setText('(o_o)')
23
24         self.label_T_value.setText(str(self.
25             listWidget_link.count()))
26
27         self.timer_statistics.start(STAT_REFRESH)
```

Tot seguit es poden veure diferents captures del canvi que es produeix en les estadístiques, primer es veu quan encara no està connectat al port serie, després al connectar però no tractar les dades que ens envia el dispositiu, després un cop estem tractant les dades rebudes i dibuixant la gràfica en temps real, i per ultim un cop hem aturat la monitorització del llaç de control (figura 4.11).

Statistics		Statistics		Statistics		Statistics	
TOTAL Links	0	TOTAL Links	0	TOTAL Links	2	TOTAL Links	2
Frames per Second	0	Frames per Second	2	Frames per Second	26	Frames per Second	2
Link Samples	0	Link Samples	0	Link Samples	304	Link Samples	304
Input Buffer Bytes	0	Input Buffer Bytes	3996	Input Buffer Bytes	21	Input Buffer Bytes	4047
Statistics		Statistics	(o_o)	Statistics	(>_<)	Statistics	(>_<)
(a) Pendent de començar.		(b) Connectat al dispositiu.		(c) Monitoritzant.		(d) Monitorització aturada.	

Figura 4.11: Comparació dels valors estadístics.

4.1.6 Idiomes

Un dels requisits que es buscava complir en el nou laboratori era que el programa visual fos multilingüe per tal de poder ser distribuït més fàcilment podent realitzar el laboratori en diferents països.

Així que el programa **DCSMonitor** (que s'executa a l'ordinador) és capaç de mostrar tota la informació en varis idiomes, i és fàcilment ampliable ja que s'ha realitzat de forma estructurada per tal que seguint algunes instruccions es puguin generar més idiomes.

Com vam dir a l'apartat de disseny 3.2.5, això ho hem aconseguit utilitzant una combinació de diferents eines amb les quals una rere l'altre ens generen el resultat desitjat. Tot seguit s'expliquen les diferents eines utilitzades, i com s'implementa cadascuna.

4.1.6.1 PyQT4

Aquesta llibreria és l'encarregada de deixar-ho tot preparat perquè els textos del programa siguin traduïts. Així que primerament, dintre de tot el codi del programa, es pot observar que tots els textos traduïbles estan envoltats amb la mateixa funció *QtGui.QApplication.translate* (veure exemple en el codi 4.24). Aquesta funció prepara tot l'entorn necessari per agafar tots aquests textos, i posteriorment mitjançant uns fitxers de traducció traduir en temps d'execució. En aquest exemple li indiquem que la paraula "Sent : " volem que pugui ser traduïda, per tant cada cop que cridem aquesta funció amb aquesta estructura ens retornarà la paraula en l'idioma que tinguem carregat.

Codi Python 4.24: Exemple de text per traduir

```
1 QtGui.QApplication.translate(
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
2     "MainWindow",
3     "Sent : ",
4     None,
5     QtGui.QApplication.UnicodeUTF8)
```

Un cop tenim tot el codi amb els textos a traduir envoltats de la funció anterior, generarem les traduccions en fitxers *.qm* amb els dos programes que explicarem més endavant (*Pylupdate4* 4.1.6.2 i *QtLingüist* 4.1.6.3), i un cop generats només caldrà carregar executar el codi 4.25. Per tant cada cop que des de les preferències es seleccioni un idioma només carregarem el fitxer *.qm* que desitgem.

Codi Python 4.25: Canviant idioma

```
1 QtGui.qApp.removeTranslator(self.translator)
2 self.translator.load('dcsmmainwindow_es.qm')
3 QtGui.qApp.installTranslator(self.translator)
4 self.retranslateUi(self)
```

4.1.6.2 Pylupdate4

Un cop tenim tot el codi del programa **DCSMonitor** preparat amb els textos a traduir de la forma que s'ha comentat en la secció 4.1.6.1, és hora d'utilitzar aquesta eina. Aquest programa recorre els diferents codis que li indiquem, buscant aquells textos preparats per traduir. Un cop trobats genera un fitxer de traduccions seguint el format *.ts*. Aquesta acció s'ha de executar per cada idioma a traduir que vulguem per tant es posa el codi per generar els quatre fitxers de traducció que actualment tenim (codi 4.26). Els fitxers generats poden ser editats amb varis programes, i posteriorment es generarà un nou fitxer amb extensió *.qm*.

Codi Bash 4.26: Crear fitxers per traduccions

```
pylupdate4 dcsmmainwindow.py main_dcs.py -ts
dcsmonitor_en.ts
pylupdate4 dcsmmainwindow.py main_dcs.py -ts
dcsmonitor_es.ts
pylupdate4 dcsmmainwindow.py main_dcs.py -ts
dcsmonitor_ca.ts
```

```
pylupdate4 dcsmainwindow.py main_dcsm.py -ts
dcsmonitor_fr.ts
```

4.1.6.3 QtLingüist

Gracies a aquest programa podem editar un a un els diferents fitxers generats anteriorment (secció 4.1.6.2), i un cop hem traduït totes les paraules li podem demanar que ens generi el fitxer de traduccions, que és capaç de carregar la llibreria *PyQT4* com hem dit en la secció 4.1.6.1. En la figura següent (figura 4.12) es pot veure l'entorn d'edició d'un fitxer, en la part central les paraules marcades en verd ja traduïdes i la paraula seleccionada que apareix per traduir una mica més abaix.

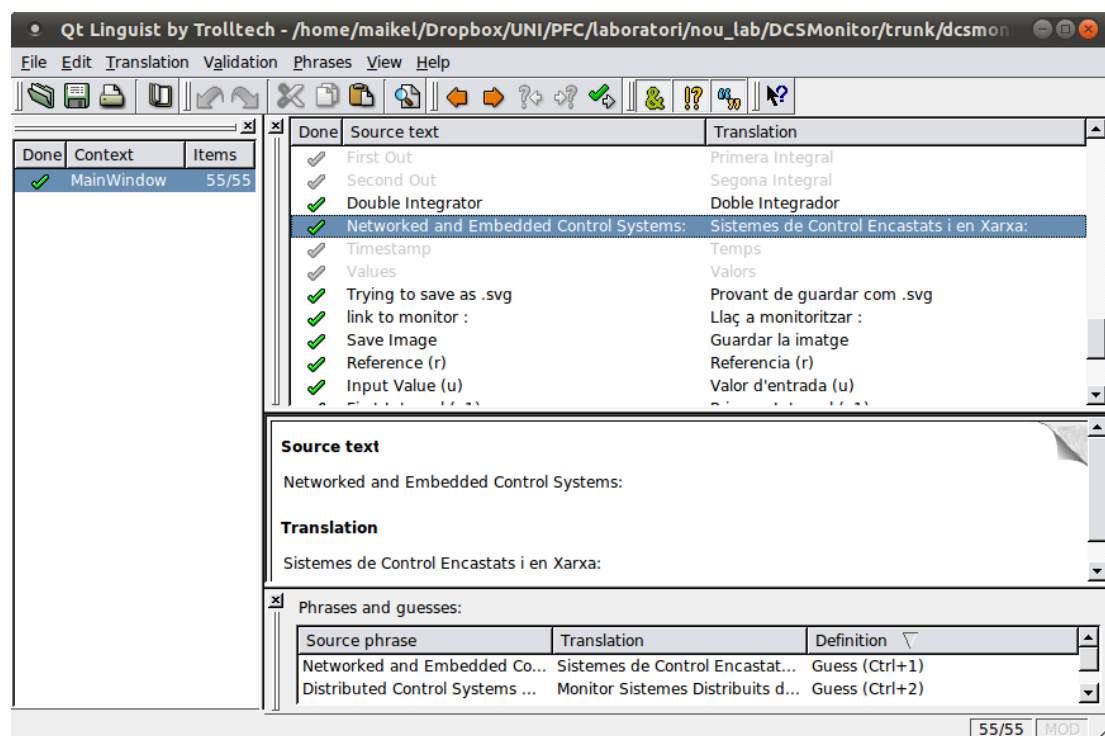


Figura 4.12: Utilitzant *QtLingüist* - Programa que ens permet editar un fitxer de traduccions *.ts*, i generar el fitxer *.qm*

4.1.6.4 Resultat

Un cop realitzats tots els passos indicats en aquesta secció, tindrem els quatre fitxers de traducció següents:

4. IMPLEMENTACIÓ DEL LABORATORI

- dcsmonitor_en.qm
- dcsmonitor_es.qm
- dcsmonitor_ca.qm
- dcsmonitor_fr.qm

I ja podem canviar l'idioma anant a les preferències del programa (Figura 4.13) i seleccionant un dels idiomes llistats.

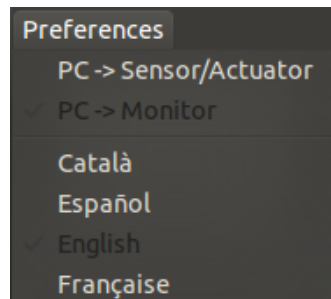


Figura 4.13: Preferències del programa DCSMonitor - Des de les preferències podem seleccionar l'idioma

Tot seguit es poden observar alguns dels canvis que s'efectuen al canviar l'idioma (Figura 4.14)

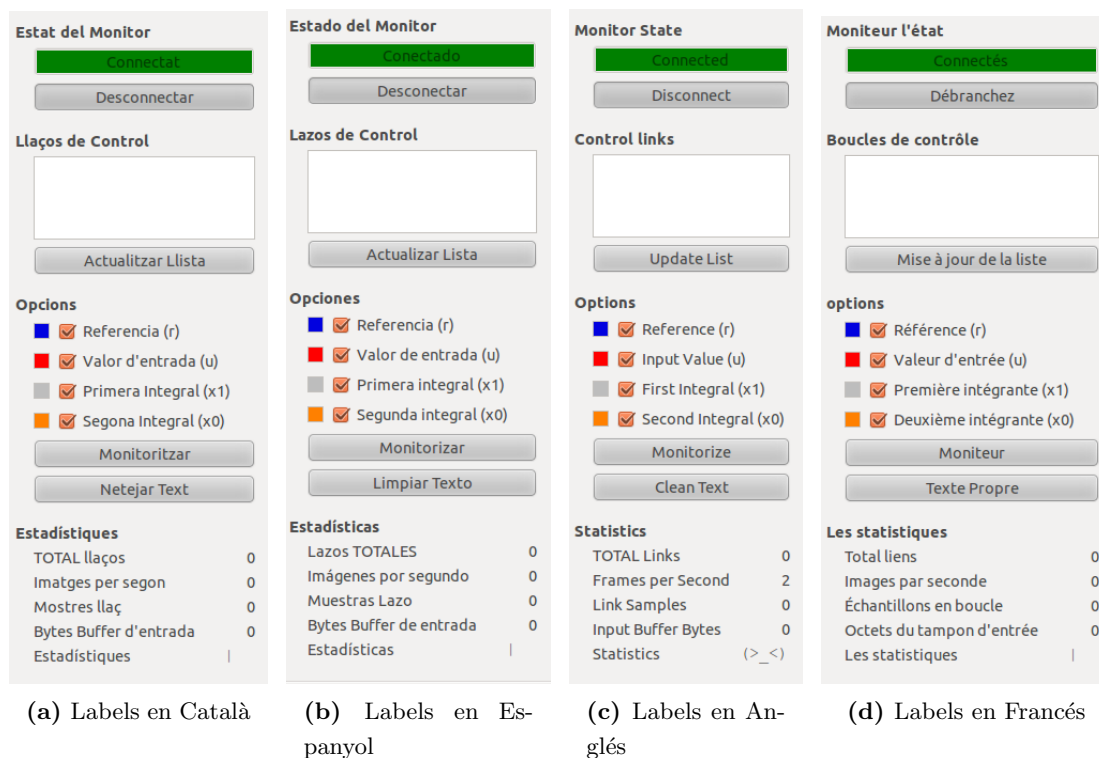


Figura 4.14: Comparació dels labels entre diferents idiomes.

4.2 dsPIC

Aquesta secció pretén explicar les parts del codi més importants dels microcontroladors dsPIC. Tot el sistema està basat en el RTOS Erika Enterprise, així que la majoria de les funcions són tasques que són activades, o es realitzen periòdicament mitjançant alarmes.

Aquesta manera de programar permet la concurrència entre diferents tasques, el que comporta una major atenció als diferents tipus de accions que el microcontrolador ha de dur a terme.

En els microcontroladors que utilitzem aquesta concurrència es en el mateix nucli, per tant tot s'executa en serie, però aquest sistema operatiu garanteix l'execució de totes les tasques.

Tot seguit es poden veure les diferents parts més importants d'aquests programes.

4. IMPLEMENTACIÓ DEL LABORATORI

4.2.1 Tasques en Erika

El RTOS Erika ens ofereix una serie d'eines per generar d'alguna manera tasques que s'executin periòdicament. En el nostre laboratori ens aprofitem d'aquesta facilitat i moltes de les funcions que fem servir en els diferents dispositius estan realitzades d'aquesta manera.

Per tal d'aconseguir això primerament s'han de declarar les tasques que el nostre Sistema Operatiu podrà executar. Per això existeix un fitxer anomenat *conf.oil* en el qual hem d'avisar a RT-Druid de la creació de les tasques que necessitem, a les quals se'ls pot assignar una prioritat diferent i temps màxim d'atenció entre d'altres opcions. També es necessari indicar-li quin dispositiu volem programar, amb quin programador, quins codis formaran part del SO, i altres qüestions que podem configurar.

Tot seguit es posa a mode d'exemple el fitxer *conf.oil* necessari per programar un dispositiu que només té la tasca de supervisió (aquesta tasca enviava informació d'estat a l'ordinador via RS232).

En el codi 4.27, es pot veure la declaració del nostre sistema indicant-li el micro-controlador que usarem, els fitxers que formaran part del codi, i finalment la declaració de la tasca *TaskSupervision* amb una prioritat de valor dos (valor més alt indica major prioritat), li donem també el màxim de temps per ser executada i que volem que la administri completament el Sistema Operatiu.

Més abaix creem una alarma per poder executar aquesta tasca de manera periòdica en cas de necessitar-ho.

Codi C 4.27: Exemple de declaració d'una tasca en el fitxer *conf.oil*

```
1 CPU mySystem {
2
3     OS myOs {
4         EE_OPT = "DEBUG";
5
6         CPU_DATA = PIC30 {
7             APP_SRC = "setup.c";
8             APP_SRC = "uart_dma.c";
9             APP_SRC = "e_can1.c";
10            APP_SRC = "code.c";
11            MULTI_STACK = FALSE;
```

```
12     ICD2 = TRUE;
13 };
14
15     MCU_DATA = PIC30 {
16         MODEL = PIC33FJ256MC710;
17     };
18
19     BOARD_DATA = EE_FLEX {
20         USELEDS = TRUE;
21     };
22
23
24     KERNEL_TYPE = EDF { NESTED_IRQ = TRUE; TICK_TIME = "
25         25ns"; };
26 };
27
28     TASK TaskSupervision {
29         REL_DEADLINE = "0.1s";
30         PRIORITY = 2;
31         STACK = SHARED;
32         SCHEDULE = FULL;
33     };
34
35     COUNTER myCounter;
36
37     ALARM AlarmSupervision {
38         COUNTER = "myCounter";
39         ACTION = ACTIVATETASK { TASK = "TaskSupervision"; };
40     };
41 };
```

Un cop tenim ben definit l'entorn del nostre Sistema Operatiu ja només cal utilitzar les tasques dintre del programa (codis d'exemple 4.28). Per poder posar en marxa una tasca RT-Druid ens dona la funció *ActivateTask(nom de la tasca)*, amb la qual només s'executarà un cop.

4. IMPLEMENTACIÓ DEL LABORATORI

En el cas de voler activat aquesta tasca periòdicament podem activar una alarma amb la funció *SetRelAlarm(AlarmType AlarmID, TickType increment, TickType cycle)* en la qual li indiquem el nom de l'alarma a executar, el temps d'espera per executar-la i el temps entre cicles d'execució. Finalment si en algun moment volguessin aturar aquesta alarma cíclica només hauríem de cridar la funció *CancelAlarm(AlarmType AlarmID)*.

Codi C 4.28: Codis per interactuar amb tasques

```
1 // Active one execution of a task
2 ActivateTask(TaskSupervision);
3
4 //Data is sent to the PC every 10ms
5 SetRelAlarm(AlarmSupervision, 1000, 10);
6
7 // Stops the periodicity of a task
8 CancelAlarm(AlarmSupervision);
```

4.2.2 Comunicació serie RS232

En aquest apartat explicarem com s'ha implementat la comunicació serie per RS232 que es va dissenyar en la secció 3.2.3. Recordem que els dos dispositius capaços d'interaccionar mitjançant aquesta tecnologia són el *Monitor* i el *Sensor/Actuador*, per tant s'explicaran els codis d'aquests relacionats amb la comunicació serie.

4.2.2.1 Recepció de senyals de control

Com hem explicat en la secció anterior 3.2.3, el dispositiu *Monitor* ha de ser capaç de rebre diferents instruccions del programa **DCSMonitor** per tal de poder posar en marxa certes accions.

Aquestes instruccions estan numerades i les dues parts de la comunicació han de conèixer-la, per tant primer de tot en el codi del *Monitor* cal declarar aquests valors:

Codi C 4.29: Declaració dels valors dels senyals de control

```
1 #define SIGNAL_STOP      1
2 #define SIGNAL_MONITOR   0
3 #define SIGNAL_PERCENT   2
4 #define SIGNAL_DEVICES   4
```

Per tal de rebre els missatges per el port sèrie RS232 es va observar el mode de funcionament de la recepció dels missatges CAN i es va utilitzar la mateixa metodologia, creant una funció que s'executés cada cop que el port sèrie s'omplís amb 8 bytes. Aquesta funció seria la encarregada de comprovar el primer dels 8 bytes del missatge, i realitza una o altre acció depenent de la taula que s'ha mostrat anteriorment 3.1.

Tot seguit es pot veure la funció que rep la interrupció del buffer d'entrada del port sèrie:

Codi C 4.30: Interrupció buffer del port sèrie

```

1  // Input serial buffer interrupt
2  ISR2(_DMA5Interrupt) //NOTE: Disabled when IEC3bits.
    DMA5IE = 0;
3  {
4      float *p_k0=(float *)&InBufferA[0];
5      unsigned long id_link;
6
7      /*Code to update received data*/
8      k_updated[0]= *(p_k0);
9      k_updated[1]= *(p_k0+1);
10
11     switch (InBufferA[0])
12     {
13     case SIGNAL_MONITOR:
14
15         id_link = (unsigned long) InBufferA[7]<<2;
16         // Generating id's for a control plant
17         ID_TO_ACTUATOR      = CONTROL_MESSAGE      |
            GENERAL_PRIORITY | id_link;
18         ID_FROM_SENSOR      = GRANTED_SENSOR_MESSAGE |
            GENERAL_PRIORITY | id_link;
19         ID_REFERENCE        = GENERAL_PURPOSE_MESSAGE |
            GENERAL_PRIORITY | id_link | REFERENCE_MESSAGE;
20         ID_SUPERVISOR       = GENERAL_PURPOSE_MESSAGE |
            GENERAL_PRIORITY | id_link | SUPERVISOR_MESSAGE;
21

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
22     // Activating filters for different message id's
23     //ecan1WriteRxAcptFilter(filter, id, exide, buffer,
        masc)
24     ecan1WriteRxAcptFilter(0x0, ID_FROM_SENSOR ,0x1,0x1,0
        x2);
25     ecan1WriteRxAcptFilter(0x1, ID_TO_ACTUATOR ,0x1,0x2,0
        x2);
26     ecan1WriteRxAcptFilter(0x2, ID_REFERENCE ,0x1,0x3,0
        x2);
27     ecan1WriteRxAcptFilter(0x3, ID_SUPERVISOR ,0x1,0x4,0
        x2);
28     CancelAlarm(AlarmSupervision);
29     // Starts sending supervision messages
30     SetRelAlarm(AlarmSupervision, 1000, 10);
31     monitoring = 1;
32     break;
33
34 case SIGNAL_PERCENT:
35     // Starts to generate useless CAN messages
36     CancelAlarm(AlarmCANUseless);
37     SetRelAlarm(AlarmCANUseless, 1000, 101-(InBufferA[7]))
        ;
38     break;
39
40 case SIGNAL_MONITOR+SIGNAL_STOP:
41     // Stops the reception of control messages
42     ecan1DisableRXFilter(0x0);
43     ecan1DisableRXFilter(0x1);
44     ecan1DisableRXFilter(0x2);
45     ecan1DisableRXFilter(0x3);
46     // If is not listing devices, stops to send
        supervisions
47     if (!listing) CancelAlarm(AlarmSupervision);
48     LATBbits.LATB14 = 0;
49     monitoring = 0;
50     break;
```

```
51
52     case SIGNAL_PERCENT+SIGNAL_STOP:
53         // Stops the generation of useless messages
54         CancelAlarm(AlarmCANUseless);
55         break;
56
57     case SIGNAL_DEVICES:
58         // Activate filter 4 to put all CAN messages into
59         // buffer 5
60         ecan1WriteRxAcptFilter(0x4,0x00000000,0x1,0x5,0x1);
61         SetRelAlarm(AlarmSupervision, 1000, 10);
62         listing = 1;
63         break;
64
65     case SIGNAL_DEVICES+SIGNAL_STOP:
66         // Deactivate filter 4
67         ecan1DisableRXFilter(0x4);
68         // If is not monitoring devices, stops to send
69         // supervisions
70         if (!monitoring) CancelAlarm(AlarmSupervision);
71         LATBbits.LATB14 = 0;
72         listing = 0;
73         break;
74
75     default:
76         break;
77 }
78
79 IFS3bits.DMA5IF = 0; // Clear the DMA1 Interrupt Flag
80 }
```

4.2.2.2 Enviament de dades

Tant el dispositiu *Monitor* com el dispositiu *Sensor/Actuador* compten amb la informació periòdica d'algun llaç de control. Aquesta informació es va emmagatzemant temporalment per enviar-les cada 10 ms al programa **DCSMonitor** .

4. IMPLEMENTACIÓ DEL LABORATORI

Per realitzar aquesta tasca periòdicament el *Sensor/Actuador* activa una alarma al iniciar l'execució, el *Monitor* en canvi, espera a que el programa **DCSMonitor** li indiqui (codi 4.31)

Codi C 4.31: Alarma de supervisió

```
1 //Data is sent to the PC every 10ms
2 SetRelAlarm(AlarmSupervision, 1000, 10);
```

La informació que envien els diferents dispositius varia, així el *Sensor/Actuador* envia només 23 bytes, amb els valors del temps, referència, entrada i primera i segona integral. En canvi el *Monitor* envia 71 bytes, els quals porten la mateixa informació que hem dit, però poden incloure'n més. En aquest cas afegeixen els diferents identificadors de llac de control, existents al bus CAN.

En el codi següent es pot veure la part comuna a tots dos codis, i una part que és única del dispositiu *Monitor*, en la que comprova si hi ha identificadors a la pila, per cada identificador que veu incrementa el valor del byte 22, i guarda en les següents posicions els identificadors que va trobant. Al final posa el byte 21 a 2, per indicar-li al programa *Monitor* que li està enviant identificadors (codi 4.32).

Codi C 4.32: Tasca de Supervisió

```
1 TASK(TaskSupervision)
2 {
3     //Send_buffer_to_pc();
4
5     static unsigned char *p_r = (unsigned char *)&r;
6     static unsigned char *p_x0= (unsigned char *)&x[0];
7     static unsigned char *p_x1= (unsigned char *)&x[1];
8     static unsigned char *p_u = (unsigned char *)&u;
9
10    //static unsigned long sys_time=0;
11
12    LATBbits.LATB10 = 1; //To get time with the
        oscilloscope
13
14    sys_time=GetTime(); //Get system time (EDF Scheduler)
15
```

```

16 //Read_State();           //Before sending state, read it
17
18 OutBuffer[0]=0x01;//header;
19 OutBuffer[1]=(unsigned char)(sys_time>>24);//4th byte
    of unsigned long
20 OutBuffer[2]=(unsigned char)(sys_time>>16);//3rd byte
    of unsigned long
21 OutBuffer[3]=(unsigned char)(sys_time>>8); //2nd byte
    of unsigned long
22 OutBuffer[4]=(unsigned char)sys_time;      //1st byte
    of unsigned long
23 OutBuffer[5]=*p_r;      //4th byte of float (32bits)
24 OutBuffer[6]=*(p_r+1); //3rd byte of float (32bits)
25 OutBuffer[7]=*(p_r+2); //2nd byte of float (32bits)
26 OutBuffer[8]=*(p_r+3); //1st byte of float (32bits)
27 OutBuffer[9]=*p_x0;
28 OutBuffer[10]=*(p_x0+1);
29 OutBuffer[11]=*(p_x0+2);
30 OutBuffer[12]=*(p_x0+3);
31 OutBuffer[13]=*p_x1;
32 OutBuffer[14]=*(p_x1+1);
33 OutBuffer[15]=*(p_x1+2);
34 OutBuffer[16]=*(p_x1+3);
35 OutBuffer[17]=*p_u;
36 OutBuffer[18]=*(p_u+1);
37 OutBuffer[19]=*(p_u+2);
38 OutBuffer[20]=*(p_u+3);
39
40
41 // This is for Monitor device
42
43 OutBuffer[22]=0x00;//number of devices;
44 unsigned char id;
45 int i = 23;
46 while (get_device(&id))
47 {

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
48     OutBuffer[i++]=(unsigned char)id;    //1st byte of
        unsigned long
49     OutBuffer[22]++;
50 }
51 if (OutBuffer[22] != 0x00)
52     OutBuffer[21]=0x02;
53 else
54     OutBuffer[21]=0x00;
55
56 // End extra part of Monitor device
57
58 //Force sending data
59 DMA4CONbits.CHEN = 1;           // Re-enable DMA4 Channel
60 DMA4REQbits.FORCE = 1;         // Manual mode: Kick-start
        the first transfer
61
62 LATBbits.LATB10 = 0; //To get time with the
        oscilloscope
63 }
```

4.2.3 Comunicació bus CAN

Com hem vist a la secció de disseny 3.2.2, existeix varis tipus de missatges que poden coexistir en el bus CAN, i a més d'això aquest tipus de missatges poden pertànyer a diferents llaços de control (fins un màxim de 256).

Administrar tots aquests missatges no és una feina fàcil, per això es va haver de crear diferents regles de filtratge per facilitar aquesta tasca.

Tot seguit en aquesta secció es podrà veure com s'ha implementat aquest tipus de mascare, els diferents filtres, i l'atenció als diferents missatges que circulen per el bus CAN.

4.2.3.1 Creació de les màscares CAN

Perquè el dispositiu *Monitor* sigui capaç de capturar els diferents missatges del bus CAN, hem de crear unes màscares per poder aplicar als filtres que posteriorment creem.

Aquestes màscares tenen com a objectiu, indicar quins dels bits del filtre són els que han de coincidir amb l'identificador del missatge CAN que circula per el bus, i d'aquesta manera capturar només aquestes coincidències.

Com aquest dispositiu rebrà els missatges en dos modes diferents, hem creat dos màscares noves que són les que utilitzarà en aquest laboratori, i hem deixat una mascara genèrica que fins ara era necessària, i que segons com evolucioni el programa podria ser-nos d'utilitat.

Num	Bits	Utilitat
0	0x1FFFFFFF	Només deixa passar els identificadors que coincideixi totalment amb el filtre
1	0x00000000	Deixa passar tots els missatges, sense tenir en compte el filtre
2	0x1C0003FF	Deixa passar els missatges en els que coincideixi la classe, l'id del llaç, i la subclasse amb els del filtre

Taula 4.1: Màscares per bus CAN

El codi que crea aquestes màscares es troba inclòs en la funció de configuració del dispositiu CAN (Funció 4.33), el qual crida a la funció *ecan1Initialize()* la qual configura el dispositiu CAN, i crea dos buffers, un d'entrada i un de sortida mitjançant DMA's.

Codi C 4.33: Funció eCAN1 config

```

1 void eCAN1_config(void)
2 {
3     //Initialize enhanced CAN bus number 1
4     ecan1Initialize();
5
6     //ecan1WriteRxAcptMask(num masc, bit masc, mide, exide)
7     ecan1WriteRxAcptMask(0x0,0x1FFFFFFF, 0,0x1);
8     ecan1WriteRxAcptMask(0x1,0x00000000, 0,0x1);
9     ecan1WriteRxAcptMask(0x2,0x1C0003FF, 0,0x1);
10 }
```

4.2.3.2 Creació dels filtres CAN

Els filtres CAN ens serveixen primerament per descartar els missatges CAN que no siguin valuosos per el nostre objectiu, i per altra banda els utilitzarem per separar en diferents buffers provisionals els missatges que ens siguin d'interès.

4. IMPLEMENTACIÓ DEL LABORATORI

D'aquesta manera el dispositiu *Actuador* estarà interessat només en els missatges de control, per tant aquest dispositiu filtrarà aquest tipus de missatges a un buffer en concret, el qual al estar ple activarà una interrupció. El mateix passarà amb els altres dispositius, per tant hem definit dins del codi els diferents elements que formen l'identificador, per crear els filtres més fàcilment.

Codi C 4.34: Definicions per la creació de filtres CAN

```
1 // DEFINITIONS TO CREATE CAN ID'S
2 // id CAN : 0b 000c ccpp pppp pppp pppp ppri iiii iiss
3 // ccc = CLASS
4 // pppp pppp pppp pppp = PRIORITY
5 // iiii iiii = ID_PLANT
6 // ss = SUBCLASS
7
8 // MESSAGE SUBCLASS
9 #define REFERENCE_MESSAGE (0)
10 #define SUPERVISOR_MESSAGE (1)
11
12 // MESSAGE ID
13 #define ID_PLANT (1<<2)
14
15 // MESSAGE PRIORITY
16 #define GENERAL_PRIORITY (0xFFFF<<10)
17
18 // MESSAGE CLASS
19 #define CONTROL_MESSAGE ((unsigned long)0<<26)
20 #define GRANTED_SENSOR_MESSAGE ((unsigned long)1<<26)
21 #define GENERAL_PURPOSE_MESSAGE ((unsigned long)2<<26)
22 #define BEST_EFFORT_SENSOR_MESSAGE ((unsigned long)
    3<<26)
23
24 unsigned long ID_TO_ACTUATOR = CONTROL_MESSAGE |
    GENERAL_PRIORITY | ID_PLANT;
25 unsigned long ID_FROM_SENSOR = GRANTED_SENSOR_MESSAGE |
    GENERAL_PRIORITY | ID_PLANT;
26 unsigned long ID_REFERENCE = GENERAL_PURPOSE_MESSAGE
```

```

        | GENERAL_PRIORITY | ID_PLANT | REFERENCE_MESSAGE;
27 unsigned long ID_SUPERVISOR    = GENERAL_PURPOSE_MESSAGE
        | GENERAL_PRIORITY | ID_PLANT | SUPERVISOR_MESSAGE;

```

4.2.3.3 Recepció de missatges CAN

Un cop creades les diferents màscares i filtres dels missatges CAN, és hora de crear una funció que capturi les interrupcions generades per el bus CAN.

La implementació CAN amb la que comptem ens permet definir fins a 16 filtres diferents, i assignar-los a 16 buffers. Aquests buffers posen a 1 un bit en el moment en que estàn plens, per tant aprofitem aquest comportament per definir cadascun dels diferents missatges a un sol filtre i un sol buffer, d'aquesta manera en el moment que un dels buffers estigui ple, sabrem quin tipus de missatge és sense haver de mirar l'identificador. Però existeix una excepció; en el cas de llistar tots els dispositius que existeixen al bus CAN. En aquest cas existeixen 2^{29} missatges diferents, i per tant hem de reunir-los d'alguna manera. Així doncs, tenint en compte que aquesta captura no és crítica pel control de cap llaç, crearem un filtre sense restriccions cap a un buffer. En el moment en que aquest buffer estigui plé, comprovarem l'identificador i el guardarem en una pila (més endavant explicarem més detalladament aquest funcionament).

En la taula 4.2 es pot veure els diferents filtres que s'han de crear, i a quin buffer estan assignats. De totes maneres, la activació o desactivació o canvi d'aquests filtres es pot realitzar en temps d'execució, pel que fa que per exemple el *Monitor* els activi i desactivi a conveniència, mentre que els dispositius *Controlador* i *Sensor/Actuador* no els cal modificar.

Num	Id	Extended	Buffer	maskara
0	ID_FROM_SENSOR	sí	1	0x1C0003FF
1	ID_TO_ACTUATOR	sí	2	0x1C0003FF
2	ID_REFERENCE	sí	3	0x1C0003FF
3	ID_SUPERVISOR	sí	4	0x1C0003FF
4	indiferent	sí	5	0x00000000

Taula 4.2: Configuració dels filtres

Tenint clar a quin buffer es redireccionarà cada missatge CAN, s'ha implementat la funció d'atenció a aquestes interrupcions tenint en compte el buffer activat; com es pot veure en el codi de la funció 4.35.

4. IMPLEMENTACIÓ DEL LABORATORI

Codi C 4.35: Funcio activada per una interrupció del bus CAN

```
1  /* CAN bus 1 Interrupt, ISR2 type */
2  ISR2(_C1Interrupt)
3  {
4      IFS2bits.C1IF = 0; // clear interrupt flag
5
6      // Transmission interrupt (nothing to be done but clear
7          flag)
8      if(C1INTFbits.TBIF)
9      {
10         C1INTFbits.TBIF = 0;
11     }
12
13     /*Reception interrupt, different code for different
14         filtered id's */
15     if(C1INTFbits.RBIF)
16     {
17         LATBbits.LATB14 ^= 1; //Toggle orange led
18         /* Filter 0(ID_FROM_SENSOR):
19             * Sensor to controller message */
20         if(C1RXFUL1bits.RXFUL1==1)
21         {
22             /* Tells rxECAN1 the buffer to pass from DMA to
23                 RAM */
24             rx_ecan1message1.buffer=1;
25
26             C1RXFUL1bits.RXFUL1=0;
27             rxECAN1(&rx_ecan1message1);
28             C1INTFbits.RBIF = 0;
29
30             /* Custom code to address Sensor message */
31             ActivateTask(TaskControllerMonitor);
32         }
33     }
34
35     /* Filter 1(ID_TO_ACTUATOR):
36         * Controller to Actuator message */
```

```

33     if(C1RXFUL1bits.RXFUL2==1)
34     {
35         /*Tells rxECAN1 the buffer to pass from DMA to RAM
36            */
37         rx_ecan1message2.buffer=2;
38
39         C1RXFUL1bits.RXFUL2=0;
40         rxECAN1(&rx_ecan1message2);
41         C1INTFbits.RBIF = 0;
42
43         /* Custom code to capture values from controller */
44         ActivateTask(TaskActuatorMonitor);
45     }
46
47     /* Filter 2(ID_REFERENCE):
48        * Controller updated reference (supervision) */
49     if(C1RXFUL1bits.RXFUL3==1)
50     {
51         /*Tells rxECAN1 the buffer to pass from DMA to RAM
52            */
53         rx_ecan1message3.buffer=3;
54
55         C1RXFUL1bits.RXFUL3=0;
56         rxECAN1(&rx_ecan1message3);
57         C1INTFbits.RBIF = 0;
58
59         /* Custom code to address the Controller message
60            * which updates the reference change */
61         r=*(float *)(&rx_ecan1message3.data[0]);
62     }
63
64     //Filter 3(ID_SUPERVISOR): ID_SUPERVISOR
65     if(C1RXFUL1bits.RXFUL4==1)
66     {
67         /*Tells rxECAN1 the buffer to pass from DMA to RAM
68            */

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
66     rx_ecan1message4.buffer=4;
67
68     C1RXFUL1bits.RXFUL4=0;
69     rxECAN1(&rx_ecan1message4);
70     C1INTFbits.RBIF = 0;
71
72     /* Custom code to address Sensor message */
73     ActivateTask(TaskSensor_supervision);
74 }
75
76 //Filter 4(ALL): all messages
77 if(C1RXFUL1bits.RXFUL5==1)
78 {
79     /*Tells rxECAN1 the buffer to pass from DMA to RAM
80        */
81     rx_ecan1message5.buffer=5;
82
83     C1RXFUL1bits.RXFUL5=0;
84     rxECAN1(&rx_ecan1message5);
85     C1INTFbits.RBIF = 0;
86
87     char id_link = (char)(rx_ecan1message5.id>>2)& 0xFF
88         ;
89
90     /* Custom code to add an id to a list*/
91     if (!search_device(id_link))
92         add_device(id_link);
93 }
```

4.2.3.4 Pila de llaços de control

Quan el *Monitor* necessita llistar els diferents llaços de control del bus CAN primer ha de capturar aquests identificadors per més tard enviar-li periòdicament al programa **DCSMonitor** . Per dur a terme aquesta tasca de manera modular s'ha creat un tipus

d'estructura que anomenarem pila (codi 4.36). Aquesta pila és estàtica i té una mida de 30 identificadors, però per poder modificar tot això sense haver de reimplementar el codi del *Monitor* s'han creat funcions tot un seguit de funcions bàsiques per tractar amb els valors continguts en ella.

Codi C 4.36: Pila d'identificadors

```
1 #define STACK_SIZE 30
2
3 struct stack{
4     unsigned char ids[STACK_SIZE];
5     int num;
6     int max;
7 }stack_ids;
```

Tot seguit es posen els codis de tractament de la pila:

- Inicialitzar 4.37
- Comprovar si és plena 4.38
- Buscar identificador 4.39
- Afegir identificador 4.40
- Treure identificador 4.41

Codi C 4.37: Inicialitzar pila

```
1 void init_devices_list()
2 {
3     stack_ids.num = 0;
4     stack_ids.max = STACK_SIZE;
5 }
```

Codi C 4.38: Comprovar si la pila és plena

```
1 int stack_full()
2 {
3     return (stack_ids.num == stack_ids.max);
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
4 }
```

Codi C 4.39: Buscar a la pila

```
1 int search_device(unsigned char id)
2 {
3     int i;
4     for (i = 0; i < stack_ids.num; i++)
5         if (stack_ids.ids[i] == id) return 1;
6     return 0;
7 }
```

Codi C 4.40: Afegir a la pila

```
1 int add_device(unsigned char id)
2 {
3     if (stack_ids.num == stack_ids.max) return 0;
4     stack_ids.ids[stack_ids.num] = id;
5     stack_ids.num++;
6     return 1;
7 }
```

Codi C 4.41: Treure de la pila

```
1 int get_device(unsigned char * id)
2 {
3     if (stack_ids.num == 0) return 0;
4     *id = stack_ids.ids[stack_ids.num-1];
5     stack_ids.num--;
6     return 1;
7 }
```

4.2.3.5 Tractant missatges de control per l'Actuador

Aquests missatges com s'ha explicat en la secció 3.2.2.1, porten amb ells el valor a aplicar a l'entrada del doble integrador. Les dades del missatge són emmagatzemades

en el buffer 2, per tant primer de tot es comprova que sigui un missatge d'aquest tipus (codi 4.42).

Aquest codi desactiva els flags del buffer que s'han activat al omplir-se, i crida a una tasca.

Codi C 4.42: Captura del missatge de control per l'*Actuador*

```
1  /* Filter 1(ID_TO_ACTUATOR):
2   * Controller to Actuator message */
3  if(C1RXFUL1bits.RXFUL2==1)
4  {
5      /*Tells rxECAN1 the buffer to pass from DMA to RAM */
6      rx_ecan1message2.buffer=2;
7
8      C1RXFUL1bits.RXFUL2=0;
9      rxECAN1(&rx_ecan1message2);
10     C1INTFbits.RBIF = 0;
11
12     /* Custom code to capture values from controller */
13     ActivateTask(TaskActuatorMonitor);
14 }
```

En el cas del *Monitor* només necessita aquesta informació per poder enviar-la al programa **DCSMonitor** perquè pugui generar les gràfiques correctament, per tant la tasca que activa només guarda aquests valors per posteriorment ser enviats (codi 4.43) per RS232.

Codi C 4.43: Tasca per capturar el valor de l'actuador

```
1  /* ActuatorMonitor Task */
2  static float *p_u = (float *)&rx_ecan1message2.data[0];
3  static unsigned long sys_time=0;
4
5  TASK(TaskActuatorMonitor)
6  {
7      //sys_time=GetTime(); //Get system time (EDF Scheduler
8      )
9      u=(*p_u);
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
9    x0=(p_x0); //Get state x[0] from rx_ecan1_message1
      [0]..[3] data field
10    x1=(p_x1); //Get state x[1] from rx_ecan1_message1
      [4]..[7] data field
11 }
```

En canvi en el cas de l'*Actuador* aquest valor l'ha d'introduir al PWM destinat a l'entrada del doble integrador (codi 4.44)

Codi C 4.44: Tasca per aplicar el valor al PWM

```
1  /* Actuator Task */
2  static float *p_u = (float *)&rx_ecan1message2.data[0];
3
4  TASK(TaskActuator)
5  {
6      u=(p_u);
7
8      PDC1=((*(p_u))/v_max)*0x7fff+0x3FFF;
9  }
```

4.2.3.6 Tractant missatges d'estat del *Sensor* per al *Controlador*

Els missatges d'estat contenen els valors de sortida de la primera i segona integral (capítol 3.2.2.2). Com hem vist anteriorment (taula 4.2) aquests missatges estan dirigits al buffer 1, per tant al activar-se la interrupció i comprovar que aquest buffer està ple activarem la tasca Controller en el cas del dispositiu *Controlador*, i la tasca ControllerMonitor en el cas del *Monitor*.

Aquí es pot veure el codi que detecta aquest missatge i activa la tasca del *Monitor* (codi 4.45):

Codi C 4.45: Captura del missatge d'estat del Sensor al Controlador

```
1  /* Filter 0(ID_FROM_SENSOR):
2   * Sensor to controller message */
3  if(C1RXFUL1bits.RXFUL1==1)
4  {
5      /* Tells rxECAN1 the buffer to pass from DMA to RAM */
```

```

6   rx_ecan1message1.buffer=1;
7
8   C1RXFUL1bits.RXFUL1=0;
9   rxECAN1(&rx_ecan1message1);
10  C1INTFbits.RBIF = 0;
11
12  /* Custom code to address Sensor message */
13  ActivateTask(TaskControllerMonitor);
14  }

```

Per tant, el dispositiu *Monitor* només captura els valors de l'estat del doble integrador (codi 4.46).

Codi C 4.46: Tasca per capturar el valor d'estat del control

```

1  static float *p_x0 = (float *)&rx_ecan1message1.data[0];
2  static float *p_x1 = (float *)&rx_ecan1message1.data[4];
3  static float x0=0;
4  static float x1=0;
5  TASK(TaskControllerMonitor)
6  {
7    x0=*(p_x0); //Get state x[0] from rx_ecan1_message1
               [0]..[3] data field
8    x1=*(p_x1); //Get state x[1] from rx_ecan1_message1
               [4]..[7] data field
9    x[0] = x0;
10   x[1] = x1;
11  }

```

Mentre que el dispositiu *Controlador* utilitza aquests valors per fer els calculs del control (codi 4.47); puntualitzem que en aquest cas el control no és bo. Un cop ha determinat el valor d'entrada per el doble integrador, activa una funció per enviar aquest valor en un missatge CAN (funció 4.48).

Codi C 4.47: Tasca per fer els calculs del control

```

1  static float Nu =0.0;
2  static float Nx[2] ={1.0,0.0};

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
3 static float k[2]={0.9834 , -0.8931};
4 static float x_hat[2]={0,0};
5 static float u_ss=0;
6
7 static float *p_x0 = (float *)&rx_ecan1message1.data[0];
8 static float *p_x1 = (float *)&rx_ecan1message1.data[4];
9 static float x0=0;
10 static float x1=0;
11 TASK(TaskController)
12 {
13     x0=*(p_x0); //Get state x[0] from rx_ecan1_message1
14                 [0]..[3] data field
15     x1=*(p_x1); //Get state x[1] from rx_ecan1_message1
16                 [4]..[7] data field
17
18     x_hat[0]=x0-r*Nx[0]; //Nx, Nu matrices needed for
19                         regulation purposes
20     x_hat[1]=x1-r*Nx[1];
21     u_ss=r*Nu;
22
23     u=-k[0]*x_hat[0]-k[1]*x_hat[1]+u_ss;
24
25     /* Check for saturation */
26     if (u>v_max/2) u=v_max/2;
27     if (u<-v_max/2) u=-v_max/2;
28
29     Send_Controller2Actuator_message(&u); //identifier=
30                                         ID_PLANT+1
31 }
```

Aquesta funció segueix el patró explicat en l'apartat 3.2.2.1, el qual només conté el valor a introduir al doble integrador.

Codi C 4.48: Funció per enviar senyal de control per bus CAN

```
1 static unsigned char *p_data= NULL;
2 void Send_Controller2Actuator_message(float *data)
3 {
```

```
4   p_data=(unsigned char *)data;
5   C1CTRL1bits.ABAT = 1;
6   while(C1TR01CONbits.TXREQ0){};
7
8   tx_ecan1message.buffer=0; //Buffer number
9   tx_ecan1message.frame_type=1; //0->Std Id, 1->Ext Id
10
11  tx_ecan1message.id=ID_TO_ACTUATOR; //Identifier;
12  tx_ecan1message.message_type=0; //0->Normal, 1->Remote
    Transmit
13  tx_ecan1message.data_length=4; //Length of data (0 to 8
    bytes)
14  tx_ecan1message.data[0]= *p_data;
15  tx_ecan1message.data[1]=*(p_data+1);
16  tx_ecan1message.data[2]=*(p_data+2);
17  tx_ecan1message.data[3]=*(p_data+3);
18
19  ecan1SendMessage(&tx_ecan1message);
20
21  while(C1TR01CONbits.TXREQ0){};
22 }
```

4.2.3.7 Tractant missatges de canvi de referencia

Els missatges de canvi de referencia són únicament indicatius per poder dibuixar la gràfica correctament (explicat en la secció 3.2.2.3). Estan referenciats al buffer 3, i només porten el valor de referencia actual, per tant tant el *Sensor/Actuador* com el *Monitor* executen el mateix codi, que detecta el tipus de missatge, i guarda el valor de referencia (codi 4.49).

Codi C 4.49: Captura del missatge de canvi de referencia

```
1  /* Filter 2(ID_REFERENCE):
2   * Controller updated reference (supervision) */
3  if(C1RXFUL1bits.RXFUL3==1)
4  {
5      /*Tells rxECAN1 the buffer to pass from DMA to RAM */
```

4. IMPLEMENTACIÓ DEL LABORATORI

```
6    rx_ecan1message3.buffer=3;
7
8    C1RXFUL1bits.RXFUL3=0;
9    rxECAN1(&rx_ecan1message3);
10   C1INTFbits.RBIF = 0;
11
12   /* Custom code to address the Controller message
13    * which updates the reference change */
14   r=*(float *)(&rx_ecan1message3.data[0]);
15 }
```

4.2.3.8 Tractant missatges d'estat del *Sensor* per al *Supervisor*

Aquest missatge té exactament el mateix format que l'anterior *missatge d'estat del Sensor al Controlador* però només té com a destinació el dispositiu *Monitor* , per tant farà exactament el mateix que el codi vist anteriorment, però es veu activat per un filtre diferent, i es guarda en el buffer 4 (codi 4.50).

Codi C 4.50: Captura del missatge d'estat del Sensor al Supervisor

```
1 //Filter 3(ID_SUPERVISOR): ID_SUPERVISOR
2 if(C1RXFUL1bits.RXFUL4==1)
3 {
4     /*Tells rxECAN1 the buffer to pass from DMA to RAM */
5     rx_ecan1message4.buffer=4;
6
7     C1RXFUL1bits.RXFUL4=0;
8     rxECAN1(&rx_ecan1message4);
9     C1INTFbits.RBIF = 0;
10
11    /* Custom code to address Sensor message */
12    ActivateTask(TaskSensor_supervision);
13 }
```

I aquest és el codi de la tasca que emmagatzema els valors de sortida del doble integrador (codi 4.51).

Codi C 4.51: Tasca que emmagatzema els valors d'estat del Sensor

```

1 static float *p2_x0 = (float *)&rx_ecan1message4.data[0];
2 static float *p2_x1 = (float *)&rx_ecan1message4.data[4];
3 TASK(TaskSensor_supervision)
4 {
5     x0=*(p2_x0); //Get state x[0] from rx_ecan1_message4
6                 [0]..[3] data field
7     x1=*(p2_x1); //Get state x[1] from rx_ecan1_message4
8                 [4]..[7] data field
9     x[0] = x0;
10    x[1] = x1;
11 }

```

4.2.3.9 Tractant la resta dels missatges

Aquest tipus de tractament només l'efectua el dispositiu *Monitor*, i és activat només quan necessita llistar tots els dispositius que existeixen al bus CAN. El motiu que sigui activat és perquè el nombre de dispositius que pot arribar a haver al bus és molt elevat, per tant ens interessa deixar temps per capturar els missatges del control d'un llaç.

Tots els missatges que no compleixin els altres filtres aniran al buffer 5, i per tant un cop activada la interrupció i comprovat que el buffer activat és aquest agafem l'identificador de llaç de control (que són els 8 bits començant per el tercer de la dreta; explicat en la secció 3.2.2).

Tot seguit abans d'emmagatzemar el seu valor, comprovarem que la pila no estigui plena (la pila esta explicada en la secció 4.2.3.4), i que aquest identificador no hi sigui ja, un cop comprovat l'afegirem (codi 4.52).

Posteriorment amb la tasca periòdica de supervisió, els identificadors apilats seran enviats al programa **DCSMonitor** i la pila tornarà a estar buida.

Codi C 4.52: Captura de la resta de missatges

```

1 //Filter 4(ALL): all messages
2 if(C1RXFUL1bits.RXFUL5==1)
3 {
4     /*Tells rxECAN1 the buffer to pass from DMA to RAM */
5     rx_ecan1message5.buffer=5;

```

4. IMPLEMENTACIÓ DEL LABORATORI

```
6
7   C1RXFUL1bits.RXFUL5=0;
8   rxECAN1(&rx_ecan1message5);
9   C1INTFbits.RBIF = 0;
10
11   char id_link = (char)(rx_ecan1message5.id>>2)& 0xFF;
12
13   /* Custom code to add an id to a list*/
14   if (!stack_full() && !search_device(id_link))
15       add_device(id_link);
16 }
```

4.2.3.10 Monitorització d'un llaç de control

Perquè el dispositiu *Monitor* sigui capaç de rebre tots els missatges CAN d'un llaç determinat, el que fem es generar els quatre identificadors dels missatges que necessitem rebre a partir del numero de llaç de control que ens envia el programa **DCSMonitor** . Com hem vist en la secció anterior (capítol 3.2.3.1) aquest valor ve en l'últim byte del missatge sèrie. Per tant agafem aquest identificador i el desplaçem dos bits a l'esquerra perquè quedi en la posició correcta.

Un cop tenim l'identificador bé, generem mitjançant una OR bit a bit tots els elements que formen l'identificador; els quals ja estan ben posicionats (Codi 4.34). I creem els filtres per rebre els missatges d'aquest llaç.

Codi C 4.53: Creant filtres per monitoritzar un llaç de control

```
1 id_link = (unsigned long) InBufferA[7]<<2;
2 // Generating id's for a control plant
3 ID_TO_ACTUATOR      = CONTROL_MESSAGE      |
   GENERAL_PRIORITY | id_link;
4 ID_FROM_SENSOR      = GRANTED_SENSOR_MESSAGE |
   GENERAL_PRIORITY | id_link;
5 ID_REFERENCE        = GENERAL_PURPOSE_MESSAGE |
   GENERAL_PRIORITY | id_link | REFERENCE_MESSAGE;
6 ID_SUPERVISOR       = GENERAL_PURPOSE_MESSAGE |
   GENERAL_PRIORITY | id_link | SUPERVISOR_MESSAGE;
```

```
7
8 // Activating filters for different message id's
9 //ecan1WriteRxAcptFilter(filter, id, exide, buffer, masc)
10 ecan1WriteRxAcptFilter(0x0, ID_FROM_SENSOR ,0x1,0x1,0x2);
11 ecan1WriteRxAcptFilter(0x1, ID_TO_ACTUATOR ,0x1,0x2,0x2);
12 ecan1WriteRxAcptFilter(0x2, ID_REFERENCE ,0x1,0x3,0x2);
13 ecan1WriteRxAcptFilter(0x3, ID_SUPERVISOR ,0x1,0x4,0x2);
14 CancelAlarm(AlarmSupervision);
15 // Starts sending supervision messages
16 SetRelAlarm(AlarmSupervision, 1000, 10);
17 monitoring = 1;
```

4.3 Conclusions

En aquest punt ja s'ha finalitzat tota la part de programació dels diferents dispositius i del programa d'ordinador **DCSMonitor**.

Primerament hem pogut veure quins són els passos exactes per el disseny d'una interfície mitjançant QT, i quins són els resultats que d'ell s'en deriven.

Hem vist com un programa en llenguatge *Python* pot tractar amb el port serie, com s'obre aquest tipus de dispositiu, i com s'interactua amb ell per rebre i enviar missatges als perifèrics que hi hagi connectats. Després hem vist com a partir dels valors rebuts per el *Monitor* podíem generar les imatges en temps real, activant un parell de tasques periòdiques que ven compenetrades aconseguen crear unes gràfiques que semblen en moviment. I com en qualsevol moment podem obtenir una captura d'aquesta gràfica en varis formats (.pdf, .eps, .png, .emf...). I finalment per la part del programa d'ordinador hem vist com generar els fitxers de traducció per aconseguir varis idiomes en el programa, canviant aquest idioma en temps d'execució.

Per la part dels dispositius del llaç de control hem vist com s'utilitzen les funcions del Sistema Operatiu en Temps Real Erika; com es creen tasques periòdiques, com s'activen de noves i com s'aturen, com es creen mascarees CAN, la creació de filtres i l'assignació de diferents buffers als missatges del bus CAN, que fer amb els diferents missatges que ens arriben, i com generar-ne de nous.

4. IMPLEMENTACIÓ DEL LABORATORI

Ara només queda fer els càlculs del temps que hem dedicat a tot, i realitzar un anàlisi econòmic del conjunt, tot això en el següent capítol.

5

Anàlisi econòmic

En aquest capítol mostrem un desglos del preu que ha comportat aquest projecte, separant la part de materials necessaris per l'execució dels experiments de laboratori, i la part d'hores dedicades a cada una de les fases del projecte segons el tipus de personal necessari per dur la feina.

5.1 Material

Tot el material necessari per dur a terme el projecte, es bàsicament el material que es necessita per elaborar un laboratori de Sistemes Distribuïts de Control com el que hem realitzat, en el que existeixen dos grups de laboratori i un professor. Cada un d'els grups de laboratori necessita per tant dues plaques *FLEX* per la elaboració del control, i el professor en necessita una que faci de *Monitor* . Com realment no existeixen alumnes no necessitàvem dos ordinadors per els grups ja que amb un ordinador connectat al dispositiu *Monitor* ja podíem realitzar totes les lectures.

També era necessari un programador per poder programar els diferents dispositius, els alimentadors (que fent ponts entre les plaques només ens han calgut un parell), un convertidor de RS232 a USB ja que l'ordinador portàtil no comptava amb aquest port, i el material d'oficina que hem anat necessitant per prendre apunts, imprimir material didàctic, o realitzar els *Live CD* necessaris.

A continuació es pot veure desglossat el preu de tot el material mencionat (taula 5.1).

5. ANÀLISI ECONÒMIC

Material	descripció	preu un.	unitats	preu
Placa FLEX	Placa de prototipat de la casa <i>FLEX</i>	119	5	595
Modul CAN	Circuit amb transceptor CAN	8	5	40
Modul RS232	Circuit amb codificador RS232	9	1	9
ICD3	Programador Mplab de la casa <i>Microchip</i>	148	1,00	148
Ordinador portatil	Portatil estandart	399	1	399
Alimentadors	Alimentador 9VDC 500mA	12	2	24
Convertidor RS232	Convertidor de estandart RS232 a USB	9	1	9
Cables varis	(cables per bus CAN, cables alimentació...)	35	1	35
Material oficina	(impresions, dvd's, cd's, fulls...)	50	1	50
TOTAL				1309

Taula 5.1: Desglos del preu de tot el material empleat

5.2 Ma d'obra

Aquest es un dels aspectes més importants a l'hora de fer el calcul de preus del projecte, ja que en aquest cas s'emporta el 90% del pressupost total del projecte.

Per fer aquest calcul s'ha estimat el preu de la ma d'obra de diferents rols que s'han tocat durant el projecte, per exemple l'Enginyer de Software que ha realitzat tots els codis relacionats amb interfícies, programes, disseny d'aquests, etc. L'enginyer Industrial que s'ha encarregat dels diferents protocols de comunicació, les connexions elèctriques i el disseny del laboratori físic. El director de projecte, que ha planificat els temps d'entrega, els canvis del diagrama de Gantt, i ha fet les reunions periòdiques per controlar el bon funcionament de tot. I el becari que ha fet les proves dels sistemes, documentat l'ús del laboratori i de part del *Live CD*.

Rol	Preu
Enginyer de Software	30
Enginyer Industrial	25
Director de Projecte	60
Becari	15

Taula 5.2: Preu de cada rol en Euros/hora

5.2 Ma d'obra

Per tant es posa una taula amb els preus de cada un dels diferents rols (taula 5.2), i tot seguit un desglos dels dies dedicats a cada una de les tasques, el seu equivalent en hores, el percentatge d'elaboració d'aquestes feines per rol, i finalment el preu resultant de fer els calculs (taula 5.3).

Tota la feina indicada en aquesta taula es pot contrastar en el diagrama de Gantt mostrat en el primer capítol (Introducció, 1) en l'apartat de planificació (secció 1.4).

Feina	E.S.	E.I.	P.M.	Bec.	dies	hores	Euros
Estudi General	50	50	0	0	7,00	24,50	673,75
Planificar Projecte	25	25	50	0	1,00	3,50	153,12
Laboratori Actual	35	60	0	10	19,00	66,50	1.795,50
Preparar codis lliures	70	30	0	0	12,40	43,40	1.236,90
Ampliació del laboratori	50	50	0	0	53,40	186,90	5.139,75
Preparar Live CD	80	0	0	20	14,00	49,00	1.323,00
Realitzar Guies	80	0	0	20	9,40	32,90	888,30
Preparar Informe Previ	50	45	5	0	7,00	24,50	716,62
Preparar Memòria	49	49	2	0	15,00	52,50	1.477,88
Preparar Defensa	50	40	5	0	7,00	24,50	686,00
Reunionons periódiques	25	25	50	0	9,00	31,50	1.378,12
TOTAL					154,20	539,70	15.468,95

Taula 5.3: Desglos del preu de la ma d'obra, amb el percentatge de dedicació per rol

Tot seguit es pot observar una gràfica de pastís (figura 5.1) que s'ha creat per veure visualment com s'ha distribuït la inversió de capital en la ma d'obra segons la feina que s'ha realitzat. Com era d'esperar una gran part del preu del projecte recau sobre l'elaboració de l'ampliació del nou laboratori, i la preparació anterior amb el laboratori actual, que ens va servir per entendre els Sistemes Distribuïts de Control.

També és important veure quin es el percentatge d'hores que li ha dedicat cada rol al projecte, i amb la taula 5.4 podem veure que l'Enginyer de Software i l'Enginyer Industrial són els que més hores han dedicat al projecte, donant una expectativa positiva envers el resultat del projecte.

5. ANÀLISI ECONÒMIC

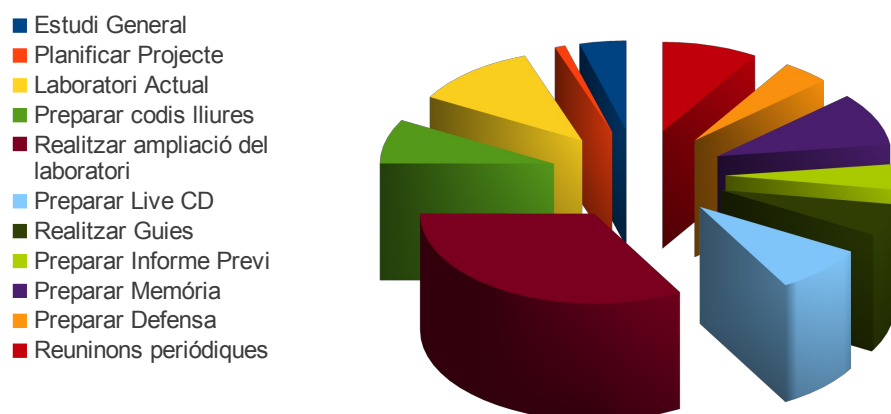


Figura 5.1: Distribució del preu segons la feina -

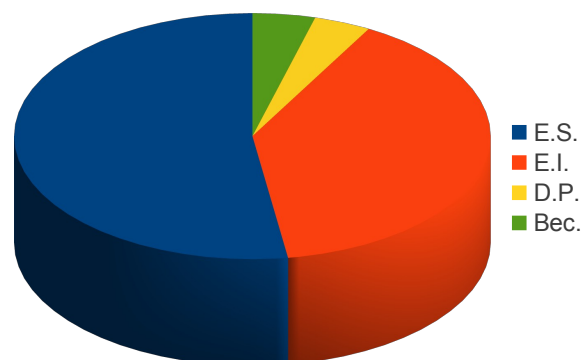


Figura 5.2: Percentatge d'hores dedicades al projecte per rol -

Rol	hores	percentatge
Enginyer de Software	283,85	53%
Enginyer Industrial	213,92	40%
Director de Projecte	21,00	4%
Becari	23,03	4%

Taula 5.4: Hores dedicades al projecte per rol, i els seus percentatges

5. ANÀLISI ECONÒMIC

6

Conclusions

Amb aquest projecte hem pogut aproximar-nos als Sistemes Distribuïts de Control, i veure els problemes que aquests sistemes arrossegueu. Al principi podia ser una mica desconcertant, però un cop ens hi endinsem les coses comencen a agafar un cert sentit i es comencen a veure d'una altra manera.

En el projecte finalment hem pogut complir tots els requisits plantejats des de bon principi.

Hem pogut agafar i entendre tot el codi del laboratori actual (secció 3.1), per d'aquesta manera crear tot l'entorn de codi lliure, el receptor dels missatges que existia anteriorment era un programa per Matlab, el qual necessitava una llicència per ser executat, així que amb la creació del nou programa **DCSMonitor** hem pogut resoldre aquest problema, ja que ara no es necessari comptar amb aquesta llicència (secció 4.1).

Hem dissenyat l'entorn de laboratori d'un Sistema Distribuït de Control real, en el qual tots els dispositius comparteixen realment el mateix bus CAN, i d'aquesta manera els alumnes poden aprendre en un entorn conflictiu real (3.2).

Per dissenyar aquest laboratori hem hagut d'organitzar els diferents identificadors que hi ha permesos al protocol CAN, de manera que existeixi una jerarquia de prioritats entre els missatges CAN, que una part pugui identificar l'identificador del llaç de control, i una altra quin tipus de missatge és (secció 3.2.2). Tot això s'ha dissenyat seguint aquests requisits, i finalment els identificadors tenen una estructura totalment ben definida.

Aquest nou laboratori ha comportat la creació d'un nou programa per ordinador; el qual hem anomenat **DCSMonitor** ; i que hem dissenyat per complir tots els objectius

6. CONCLUSIONS

que es van plantejar.

D'aquesta manera hem estudiat i creat tot el codi necessari perquè aquest programa fos capaç de comunicar-se a través del port serie via RS232 a una placa *FLEX* (secció 4.1.2) per tal de: demanar un llistat de tots els llaços de control existents al bus CAN (secció 4.1.2.2), demanar-li que carregués el bus de manera que els llaços de control es veiessin amb problemes reals (secció 4.1.2.4), demanar-li totes les dades de l'estat d'un llaç de control i d'aquesta manera poder avaluar la bondat del control de cada grup del laboratori (secció 4.1.2.3). I tot això mantenint la compatibilitat del laboratori antic, creant dos modes d'execució, un en el que es totalment compatible amb les plaques *FLEX* de l'antic laboratori i on cada alumne pot avaluar el seu control, i un mode d'execució en el que intervé un dispositiu nou que anomenem *Monitor*, amb el que es poden fer totes les accions anteriorment comentades.

Una peculiaritat de la comunicació serie que s'ha dissenyat es que s'ha realitzat de manera que pugui ser fàcilment ampliable i reutilitzable per tal d'afegir més opcions per al *Monitor* o per el programa **DCSMonitor**.

Amb les dades rebudes d'un llaç de control connectat al programa, o monitoritzat per un dispositiu *Monitor*, hem fet que el programa **DCSMonitor** sigui capaç de generar unes gràfiques en temps real amb els valors del control de: referència, valor d'entrada del doble integrador i valors de la primera i segona integral. A més aquests plots poden ser trets de la gràfica individualment (secció 4.1.3).

Les gràfiques en temps real també poden ser exportades a diferents tipus d'imatge, entre d'elles imatges vectorials, molt interessants per afegir en documents o articles (secció 4.1.4).

Un altre dels requisits del projecte era que el programa fos multilingüe, i ho hem complert creant l'entorn disponible en català, castellà, anglès i francès (secció 4.1.6). A més hem utilitzat una metodologia de programació modular que ens permet fàcilment afegir tots els idiomes que vulguem.

A part del programa d'ordinador **DCSMonitor**, també calia crear un dispositiu que pogués monitoritzar els llaços de control que es trobessin al bus CAN (dispositiu que hem anomenat *Monitor*). Per tal s'ha dissenyat el sistema de recepció de missatges per RS232 del programa **DCSMonitor**, activant interrupcions i comprovant quin tipus de acció es volia portar a terme, i posant en marxa diferents tasques dependents d'això (secció 4.2.2.1).

Els dispositius que ja existien en el laboratori actual s'han mantingut totalment compatibles, i s'els ha afegit el necessari per poder compartir el bus CAN sense conflictes, creant l'estructura d'identificadors de manera modular, i havent de incloure únicament el numero de grup al que pertanyen (veure codi 4.34).

Sobre les guies, s'ha preparat una guia de laboratori per posar l'entorn de laboratori necessari per poder executar les diferents pràctiques que des de la universitat s'imparteixen (apèndix A).

També s'ha creat un *Live CD* amb un Linux *Ubuntu*, amb tot el codi necessari preparat per ser arrencat i provat (el pes de tot el sistema amb *Eclipse* i *MPLAB®* X inclosos han fet que el pes total del sistema sigui de 1,4GB el que ha comportat que el *Live CD* s'hagi de gravar en un DVD o en un llapis de memòria). Gracies a aquest DVD només logejar-nos podem programar el llaç de control, i començar a fer les lectures. I la guia d'aquest CD que ens explica pas a pas com poder fer una instal·lació de l'entorn en un ordinador o en una màquina virtual, i després com utilitzar-lo tant en l'entorn instal·lat com des de el *Live CD* (apèndix B).

Per tant un cop finalitzat han estat complerts tots els requisits proposats en un principi, quedant constància de manera desglossada al llarg de tota la memòria.

Cal destacar que l'entrega d'aquest projecte no tanca aquest laboratori, ja que el disseny que s'ha realitzat durant el projecte ha estat encaminat cap al doble integrador, però s'ha programat l'entorn, i s'han pensat tots els identificadors i missatges de manera que pugui ser ampliat per atendre altre tipus de controls, realitzar diferents accions des de l'ordinador, deixant espai suficient perquè la comunicació entre el programa **DCSMonitor** i el dispositiu *Monitor* tinguin una gran varietat de senyals diferents i es pugui modificar el laboratori de varies maneres.

6.0.1 Treball futur

En el transcurs del projecte han anat apareixen diverses funcions que podien ampliar el projecte però quedaven fora de l'abast d'aquest, així que posteriorment a l'entrega d'aquest projecte encara es poden realitzar aquestes ampliacions i/o millores:

1. Ampliar el nombre d'idiomes en el que està traduït el programa per l'ordinador.
2. Traduir les guies a altres idiomes.

6. CONCLUSIONS

3. Que el programa **DCSMonitor** sigui capaç de mostrar les gràfiques de més d'un llaç de control al mateix temps.
4. Que el dispositiu *Monitor* del microcontrolador sigui capaç d'estimar el percentatge de saturació del bus CAN, i d'aquesta manera poder enviar aquesta informació al programa **DCSMonitor** .
5. Crear el codi necessari perquè els missatges CAN no provoquin interrupcions, sinó que en tot moment es faci un pooling controlat.
6. Mirar de programar el microcontrolador amb una eina més senzilla i menys pesada que el MPLAB® X , el qual actualment només s'utilitza per aquest fi.
7. Reduir la mida de la imatge del laboratori perquè càpiga en un CD.
8. Utilitzar les tres màscares CAN disponibles en els microcontroladors.
9. En el programa **DCSMonitor** separar en diferents processos l'adquisició de dades del port serie i el dibuix de la gràfica.
10. Organitzar de manera eficient els textos per traduir del programa **DCSMonitor**

Apèndix A

Guia del laboratori

A.1 Preparació de l'entorn de desenvolupament

Per tal de programar les plaques Flex per la realització del laboratori es necessita tenir l'entorn de desenvolupament preparat amb els següents programes:

- Eclipse (EE_160) : entorn de programació, reanomenat Erika Enterprise (EE) *RT-Druid* .
- Mplab (mplabx_ide_beta_7) : entorn per descarregar els binaris al dsPIC.
- Matlab (matlab 7.8) : entorn per la comunicació entre el PC i el dsPIC.
- Python (Python 2.7) : interpret de programes en python.
 - PySerial : modul per comunicació serie.
 - Matplotlib : modul per la interpretació de les dades rebudes del dsPIC.

Tots aquests programes són de lliure distribució (excepte el Matlab), i aquests es poden aconseguir de les seves pàgines oficials:

- Eclipse
<http://erika.tuxfamily.org/erika-for-multiple-devices.html>
- Mplabx
http://ww1.microchip.com/downloads/mplab/X_Beta/installer.html

A. GUIA DEL LABORATORI

- Python

<http://www.python.org/download/>

- PySerial

<http://pypi.python.org/pypi/pyserial>

- Matplotlib

<http://matplotlib.sourceforge.net/users/installing.html>

- PyQt4

<http://www.riverbankcomputing.co.uk/software/pyqt/download>

Llibries Python `sudo apt-get install gnuplot` `sudo apt-get install python-matplotlib`
`sudo apt-get install python-scitools` `sudo apt-get install python-qt4`

A.1.1 Instal·lació de Mplab

Per poder instal·lar MPLAB[®] X es necessari tenir instal·lat Java, en molts casos ja ve preinstal·lat (com en el nostre cas si treballem amb Ubuntu), però en cas de necessitar-lo instal·lar visiteu la seva pagina oficial des de on es pot descarregar: <http://www.java.com/es/>

En Ubuntu el podeu instal·lar directament des dels repositoris amb la comanda:

```
Codi Bash A.1: Instalar Java Runtime Environment en Ubuntu
```

```
sudo apt-get install default-jre
```

Un cop tingueu instal·lat l'entorn de Java, accediu a la pagina de Mplab X de Microchip (Figura A.1) i seleccioneu:

- MPLAB IDE X Beta
- MPLAB C30 Lite Compiler for dsPIC DSCs and PIC24 MCUs

http://ww1.microchip.com/downloads/mplab/X_Beta/installer.html

Un cop descarregats obrirem una terminal per instal·lar-los (Figura A.2):

Accedim al directori on s'hagin descarregat els fitxers i comprovem que efectivament s'han descarregat:

A.1 Preparació de l'entorn de desenvolupament

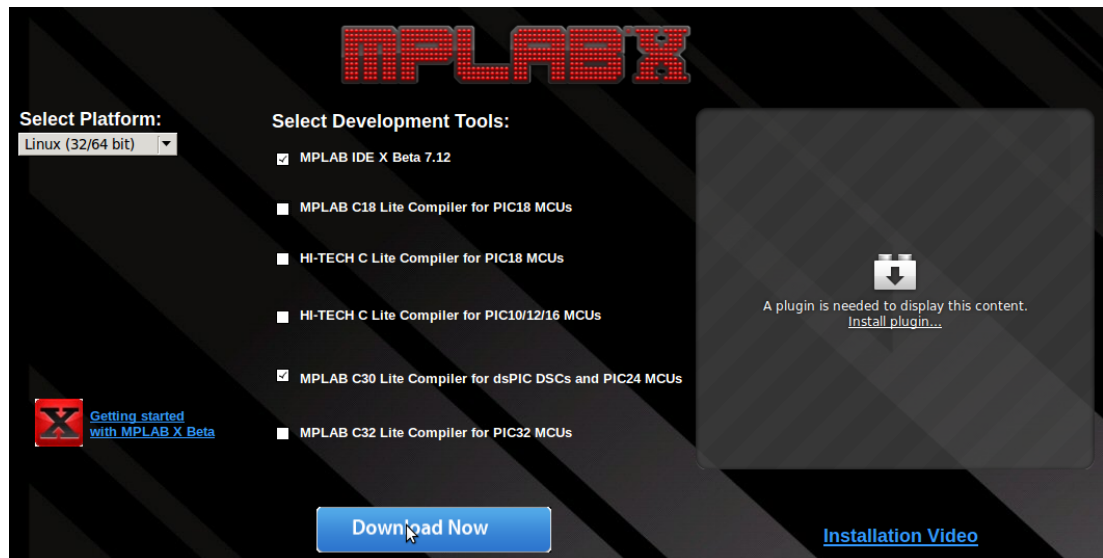


Figura A.1: Pagina oficial MPLABX - En aquesta pàgina podem descarregar la ultima versió per Windows o Linux.

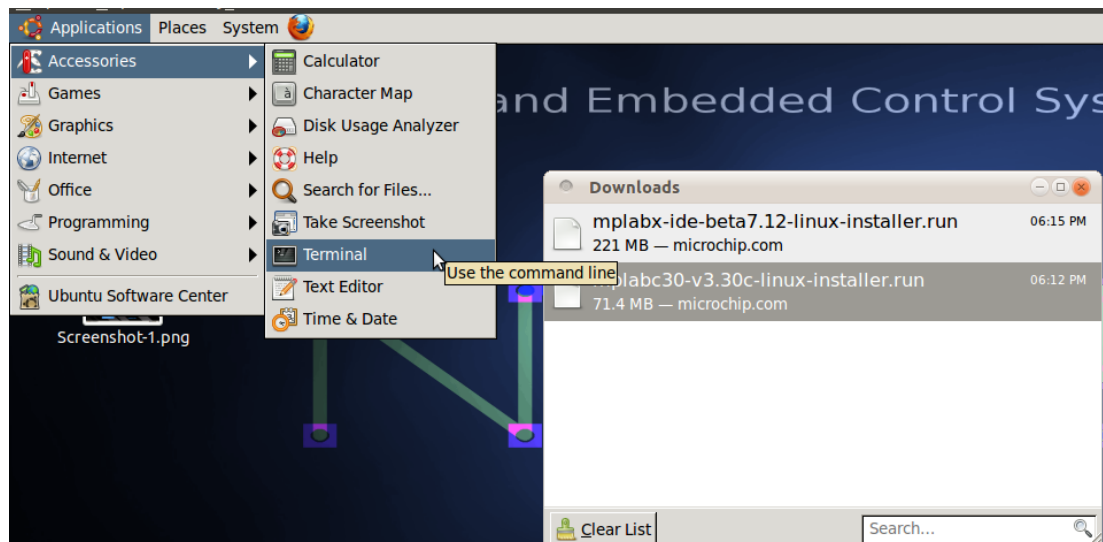


Figura A.2: Obrint una terminal - Per obrir una terminal anem a Aplicacions ->Accessoris ->Terminal

A. GUIA DEL LABORATORI

Codi Bash A.2: Comprobant mplab descarregat en Ubuntu

```
cd ~/Downloads/  
ls
```

Donem permís d'execució a mplabx-ide e instal·lem MplabX IDE(Figura A.3):

Codi Bash A.3: Instalar MPLAB® X en Ubuntu

```
chmod u+x mplabx-ide-beta7.12-linux-installer.run  
sudo ./mplabx-ide-beta7.12-linux-installer.run
```

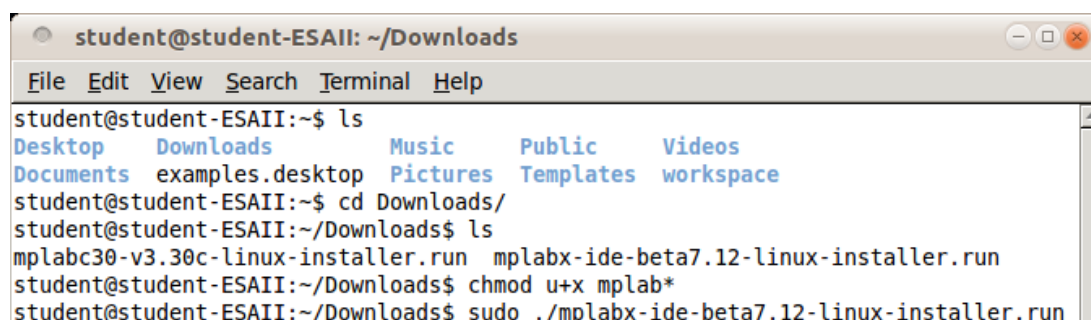


Figura A.3: Instalant MplabX IDE - Canviem els permisos d'execució i executem l'instal·lador.

Tot seguit només haurem d'acceptar totes les condicions d'us (Figura A.4) i indicar-li el directori d'instal·lació (Figura A.5), el qual podem deixar per defecte a '/opt/microchip/mplabx'.

Ens dirà que hem de reiniciar perquè els canvis tinguin efecte (Figura A.6). Però de totes maneres no reiniciarem fins que hàgim instal·lat el següent paquet.

Un cop hem acabat d'instal·lar l'IDE de MPLAB® X procedim a instal·lar el paquet per programar els dsPIC:

Codi Bash A.4: Instalar compilador mplab 30 en Ubuntu

```
chmod u+x mplabc30-v3.30c-linux-installer.run  
sudo ./mplabc30-v3.30c-linux-installer.run
```

I igual que en el cas anterior només haurem de acceptar les condicions (Figura A.7) i deixar el directori d'instal·lació per defecte '/opt/microchip/mplabc30/v3.30c'.

A.1 Preparació de l'entorn de desenvolupament

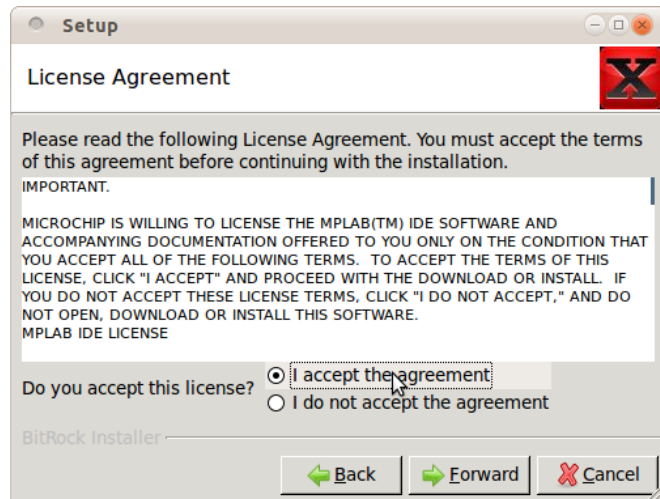


Figura A.4: Termes i condicions MplabX IDE - Acceptem els termes i les condicions d'us de MplabX.

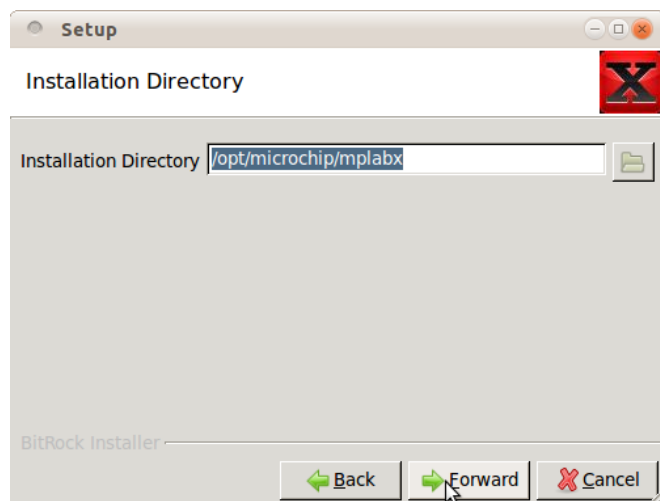


Figura A.5: Directori d'instal·lació de MplabX - Podem deixar per defecte aquest directori.

A. GUIA DEL LABORATORI

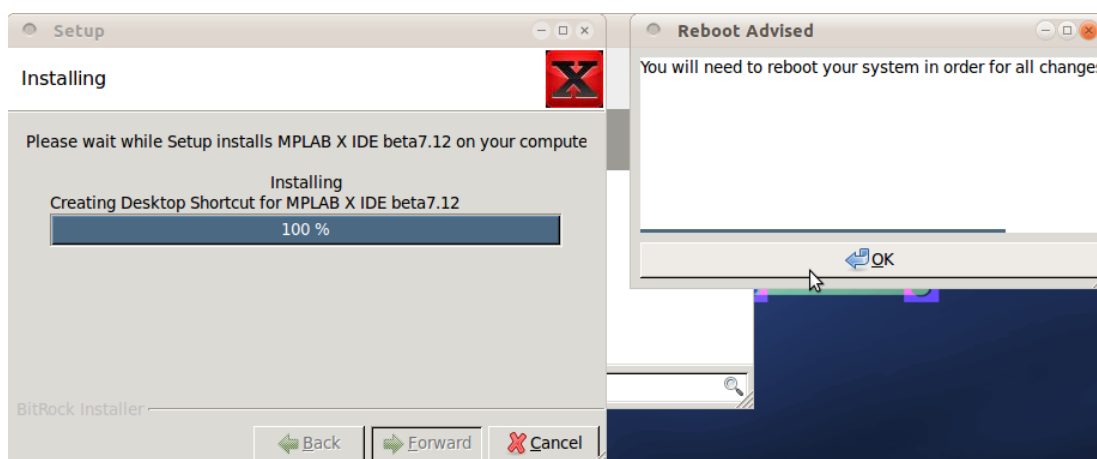


Figura A.6: Final instal·lació MplabX IDE - Ens indica que es necessari reiniciar el sistema.

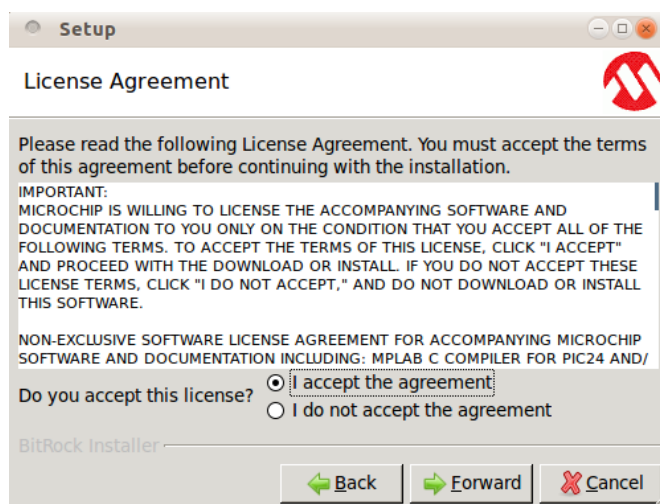


Figura A.7: Termes i condicions Compilador MplabX C30 - Acceptem els termes d'us del compilador

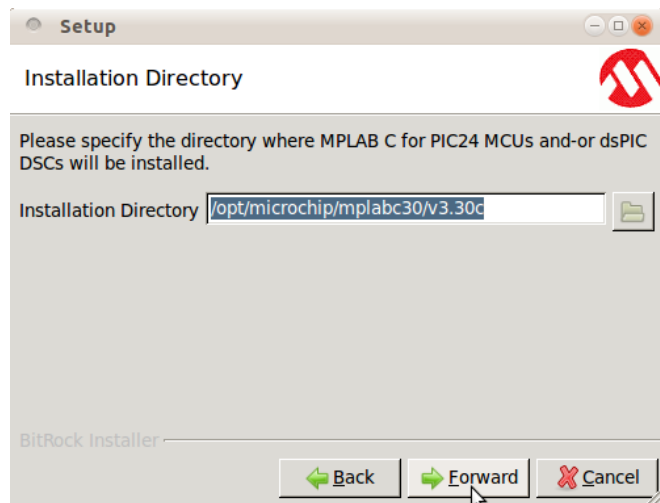


Figura A.8: Directori d'instal·lació compilador C30 - Si en el cas anterior no hem canviat el directori deixem-lo per defecte.

A.1.2 Instal·lació d'Eclipse

En aquest cas instal·lem seguint les instruccions que ens indica a la seva pàgina oficial:

<http://www.eclipse.org/>

En linux el trobem en els repositoris. Així que en Ubuntu faríem el següent :

Codi Bash A.5: Instalar Eclipse en Ubuntu

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install eclipse
```

A.1.3 Instal·lació del pluguín *RT-Druid* per Eclipse

Un cop tinguem Eclipse instal·lat hauríem d'afegir a aquest el paquet corresponent a *RT-Druid* , per fer això obrirem l'Eclipse i farem els següents passos:

Anem a:

- Help ->Install new Software...

Afegirm una entrada nova (Figura A.10) clicant a:

- Add...

A. GUIA DEL LABORATORI

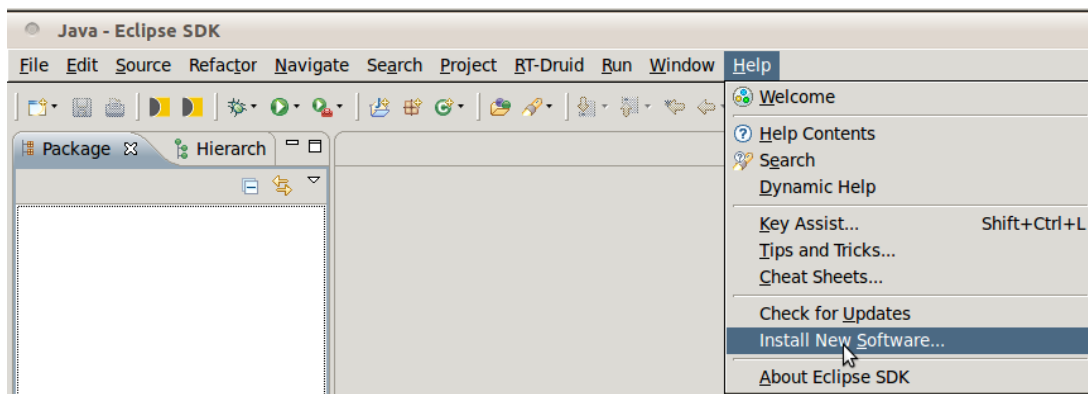


Figura A.9: Instalant nou plugin en Eclipse - Per instal·lar un plugin nou anem a Help ->Install new Software...

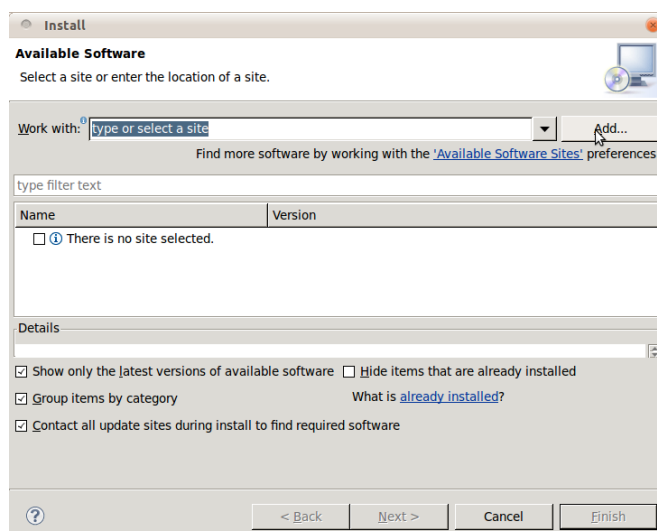


Figura A.10: Pantalla per afegir nous plugins - Des de aquesta pantalla podem instal·lar nous plugins a Eclipse.

A.1 Preparació de l'entorn de desenvolupament

En la finestra que apareix (Figura A.11) ompliu els camps amb les següents dades:

- Name -> *RT-Druid*
- Location -> `http://download.tuxfamily.org/erika/webdownload/rtdruid_160_nb/`

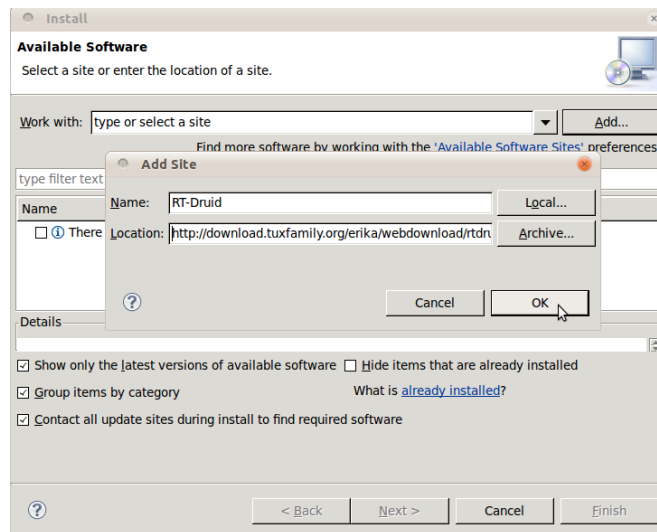


Figura A.11: Afegint nova adreça de descarrega - Aquí podem afegir una adreça on existeixin diferents pluguins.

Tot seguit haurien d'aparèixer els paquets que ha trobat en l'adreça que li hem indicat (Figura A.12), els marquem tots i prosseguim.

Tot seguit ens apareixerà un desglos de tots els paquets que s'instal·laran, i si seguim ens demanarà que acceptem els termes i condicions d'us (Figura A.13)

En un punt de la instal·lació aquesta s'aturarà i darrera la finestra principal apareixerà un avís referent a la confiança del lloc de descarrega d'un dels paquets, fixeuvos-hi i accepteu (Figura A.14).

Finalment ens avisarà que hauríem de reiniciar eclipse perquè els canvis tinguin efecte, així que digueu-li que sí volem reiniciar (Figura A.15).

Per tal que el mateix Eclipse sigui capaç de compilar el codi i generar el fitxer .elf necessari per Mplab, li hem d'indicar la ruta d'on tenim instal·lat el compilador. Això li podem indicar accedint a:

- Window -> Preferences

A. GUIA DEL LABORATORI

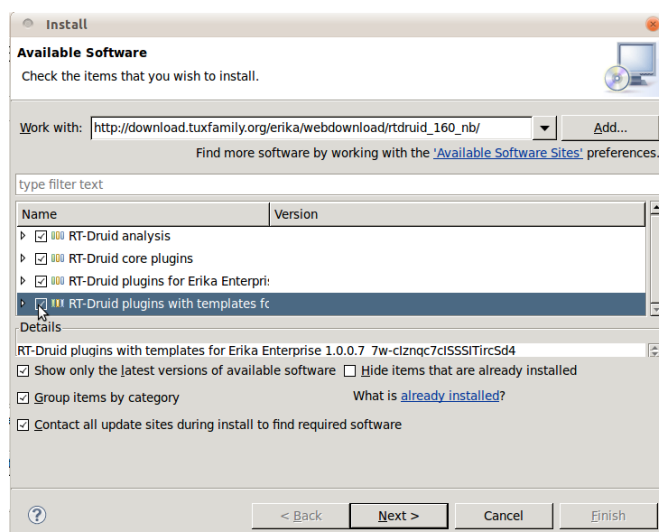


Figura A.12: Plugins disponibles - Aquí surt una llista de tots els plugins disponibles a l'adreça indicada.

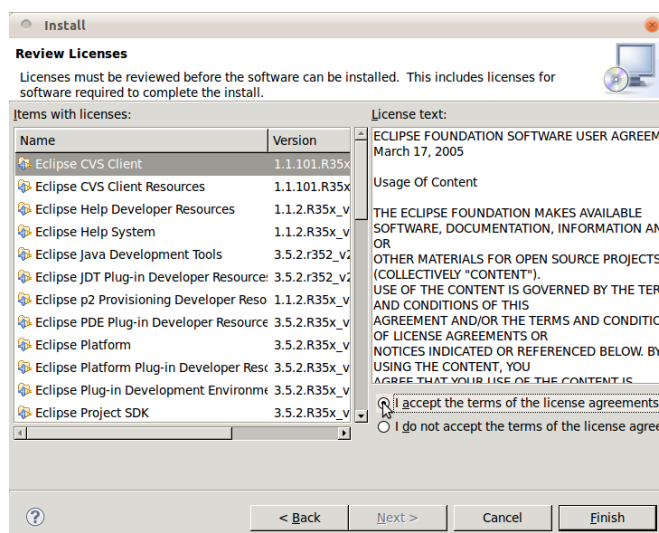


Figura A.13: Termes i condicions *RT-Druid* - S'han d'acceptar els termes i les condicions d'us del software de *RT-Druid*.

A.1 Preparació de l'entorn de desenvolupament

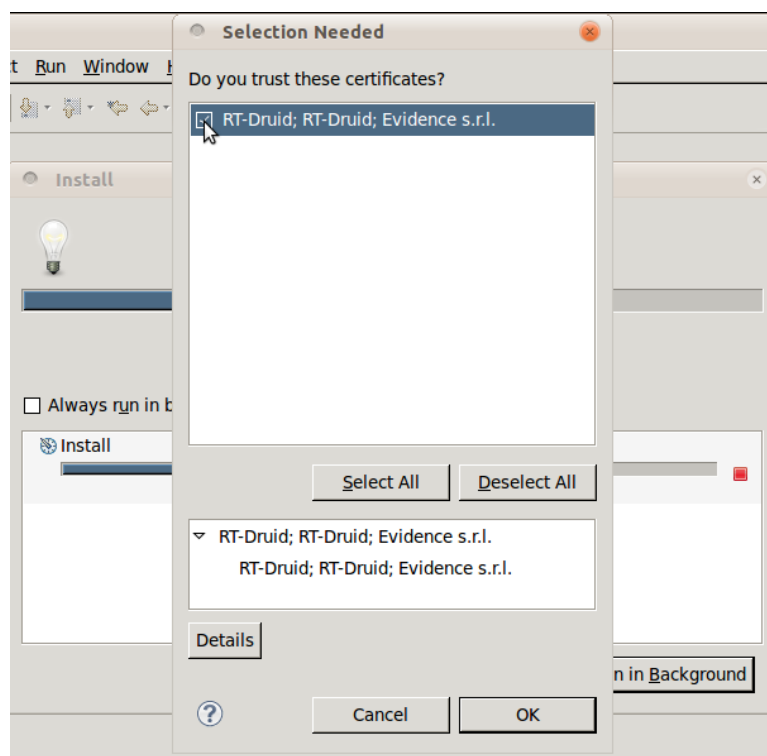


Figura A.14: Confiança en certificat *RT-Druid* - Acceptar la confiança del certificat de *RT-Druid* .

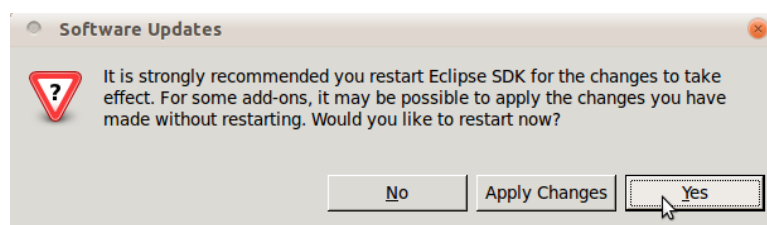


Figura A.15: Reiniciar Eclipse - Indicar que reiniciï Eclipse.

A. GUIA DEL LABORATORI

Dintre de les preferències naveguem fins a :

- *RT-Druid* -> Oil -> dsPic

I comprovem que la ruta es correcta (Figura A.16) (vigileu que la versió de mplab pot ser diferent, però de totes maneres en una instal·lació per defecte hauria de ser de la forma:

- Gcc path /opt/microchip/mplabc30/X.XXy
- Asm path /opt/microchip/mplabx/asm30

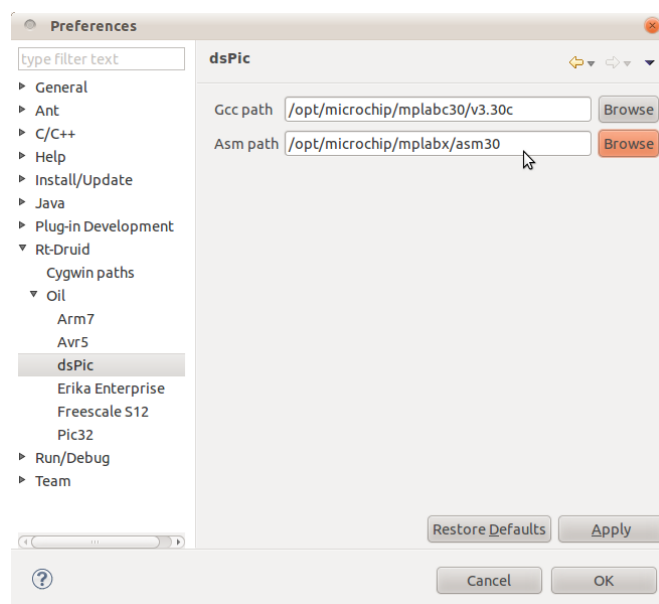


Figura A.16: Path compilador *RT-Druid* - Introduïm el path dels compiladors.

Amb aquests passos tindrem l'entorn necessari per programar els microcontroladors amb el sistema operatiu en temps real Erika.

A.2 Fer parpellejar un led

Aquests són els passos necessaris per crear un nou projecte a partir d'una plantilla, compilar-la i gravar-la en el microcontrolador.

A.2.1 Entorn Eclipse

Primer de tot haurem d'obrir el programa Eclipse, que és un IDE de programació, en aquest cas porta un plugin *RT-Druid* de Evidence que ens permetrà programar el RTOS Erika.

Un cop obert el programa crearem un nou projecte (Figura A.17)

- File ->New ->Project...

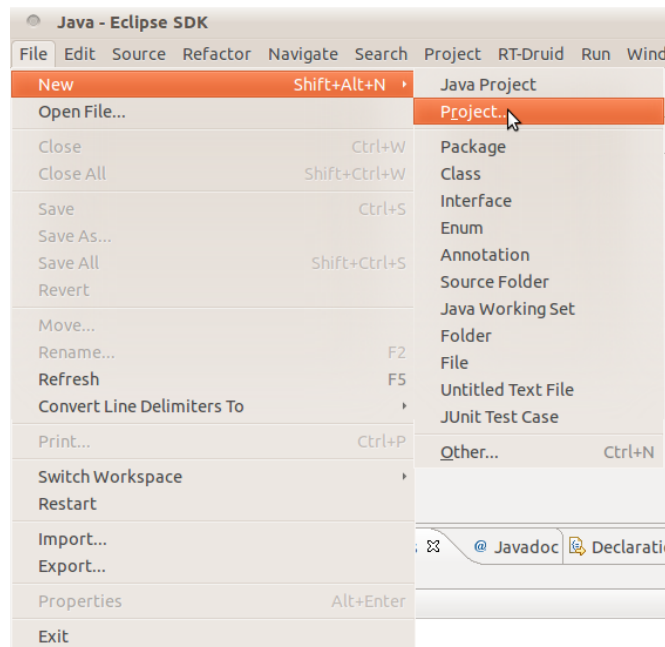


Figura A.17: Nou projecte *RT-Druid* - Per crear un nou projecte *RT-Druid* anar a New, Project...

Seleccionarem el tipus de projecte que apareix a la llista (Figura A.18):

- Evidence ->*RT-Druid* Oil and C/C++ Project

Ara li posarem un nom de projecte (Figura A.19) i deixarem les altres opcions per defecte (si volem podem canviar el directori on volem crear-lo, però en principi volem tenir tots els projectes junts al nostre *workspace*).

En aquest punt es quan podem seleccionar una plantilla per començar el projecte (Figura A.20), així no haurem de programar les coses bàsiques. Per tant fem el següent:

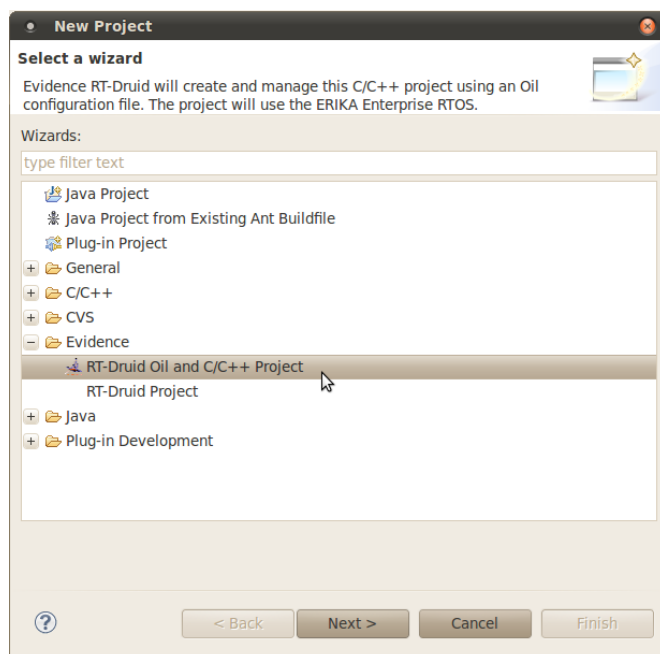


Figura A.18: Seleccionar tipus de projecte *RT-Druid* -

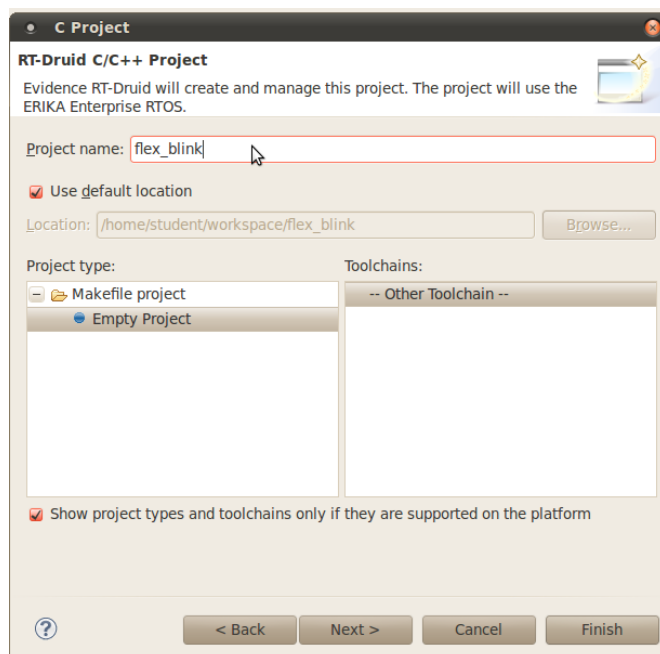


Figura A.19: Posant nom al projecte *RT-Druid* - Deixem les opcions per defecte

Marquem la caixaeta *Create a project using one of these templates* i seleccionem *pic30* (en el nostre cas, ja que estem programant un dsPIC33XXX) i aquí dintre seleccionem un dels *templates*, en aquest cas:

- *pic30* ->FLEX ->EDF: Periodic task with period

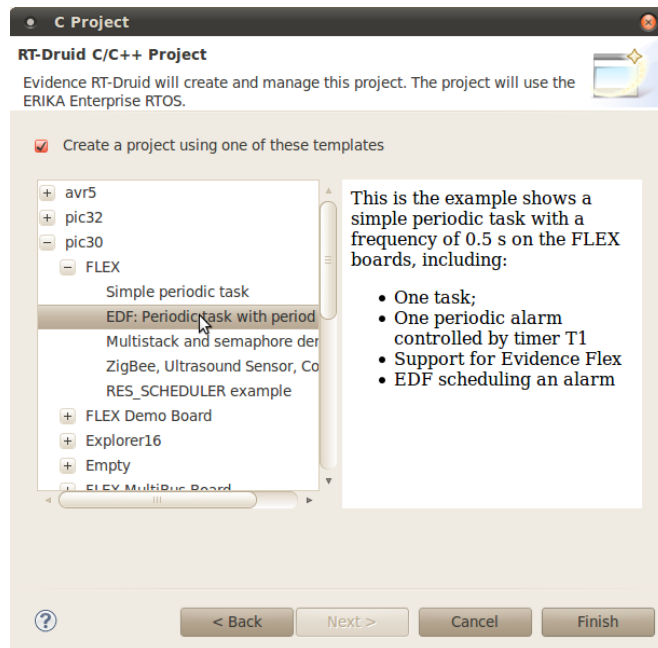


Figura A.20: Templates *RT-Druid* - En aquest apartat podem seleccionar entre alguns *templates* predisenyats els quals només cal compilar i gravar en la placa Flex

Un cop li donem a *Finish* ja tindrem carregat l'entorn per programar.

En cas de tenir activada la opció *Build Automatically* es compilarà per primer cop sense haver de realitzar cap acció, però de totes maneres ens interessa desactivar-ho. Per tant mirem que *Build Automatically* estigui desmarcat (Figura A.22):

- Project ->Build Automatically

Amb tot això ja podríem modificar el codi i programar el que sigui necessari. Un cop tinguem el codi desitjat per compilar-lo hauríem **primer de tot Guardar el projecte** (si no guardem estarem compilant una versió anterior), y un cop guardat anem a:

A. GUIA DEL LABORATORI

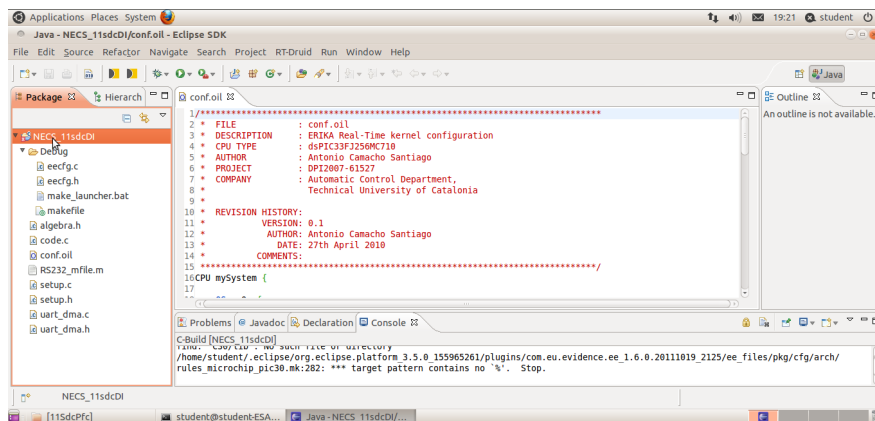


Figura A.21: Entorn Eclipse - Entorn amb projecte carregat preparat per compilar.

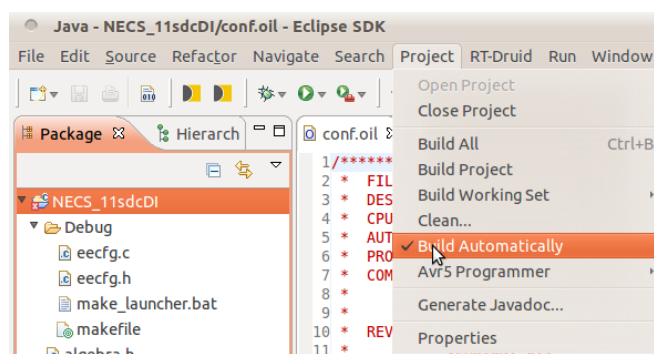


Figura A.22: Desactivar compilació automàtica. -

- Project ->Clean

Fent el clean ens assegurem que el codi que es compila sigui realment la versió que acabem de programar, i podem deixar les opcions que hi ha per defecte (Figura A.23) (que netejaran tots els projectes que tinguem oberts i els compilaran), o seleccionar *Clean projects selected below* i *Build only the selected projects* per tal de netejar i compilar només el projecte que seleccionem.

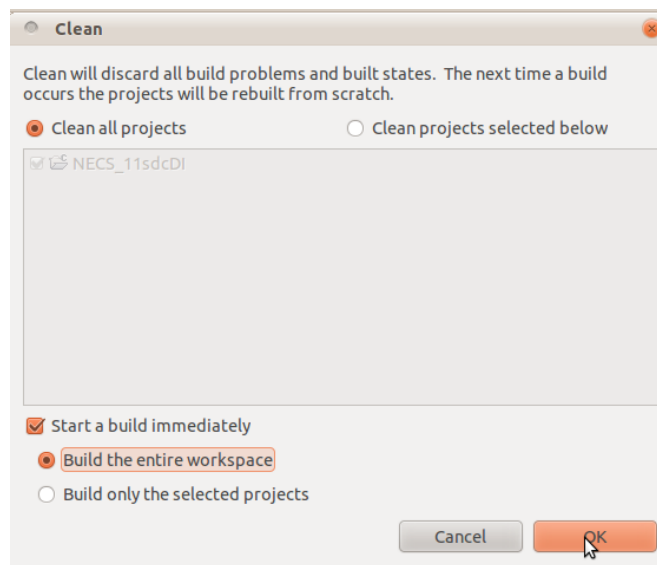


Figura A.23: Compilar en Eclipse - Per tal de compilar primer es recomanable netejar l'entorn.

Un cop acabi de compilar ho indicarà a la Consola inferior (Figura A.24) amb el missatge *Compilation terminated successfully!* això haurà creat un fitxer nou anomenat *pic30.elf*, que serà el que més endavant utilitzarem el MPLAB[®] X per gravar al microcontrolador.

Podria ser que al compilar no trobés la ruta al compilador, comproheu que hagueu indicat el path correcte en el pas d'instal·lació del pluguín *RT-Druid* de la secció anterior A.1.3, (Figura A.16).

A.2.2 Entorn MplabX

Un cop hem generat el fitxer *.elf* procedim a obrir MPLAB[®] X . Amb aquest programa podrem finalment grabar el microcontrolador.

A. GUIA DEL LABORATORI

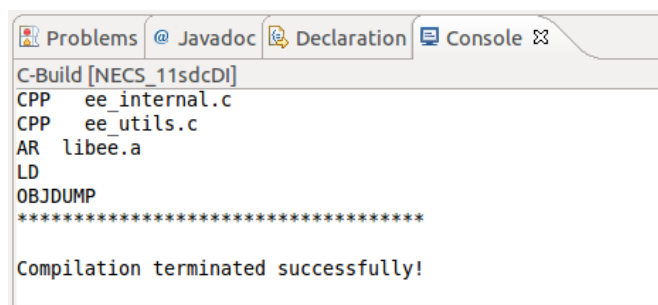


Figura A.24: Compilació en Eclipse - Missatge de compilació satisfactoria

Així que obrim el programa i anem a crear un nou projecte (Figura A.25):

- File ->New Project...

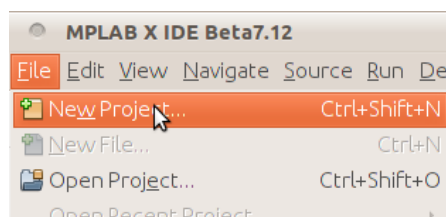


Figura A.25: Nou projecte MplabX -

Un cop seleccionat, ens preguntarà quin tipus de projecte volem crear. En el nostre cas com ja hem precompilat en Eclipse li indiquem que utilitzi el nostre fitxer *.elf* (Figura A.26).

- Categories : Microchip Embedded
- Projects: Prebuilt (Hex, Loadable image) Project

A la següent secció ens demanarà que li indiquem el path del fitxer, així que anem fins a ell i el seleccionem (Figura A.27). Si hem seguit la guia fins aquest punt hauria de ser en el path següent:

- *home/student/workspace/flexlink/Debug/pic30.elf*

(Això depèn d'on hàgiu creat el vostre *workspace* i el nom que li hàgiu donat al projecte)

A.2 Fer parpellejar un led

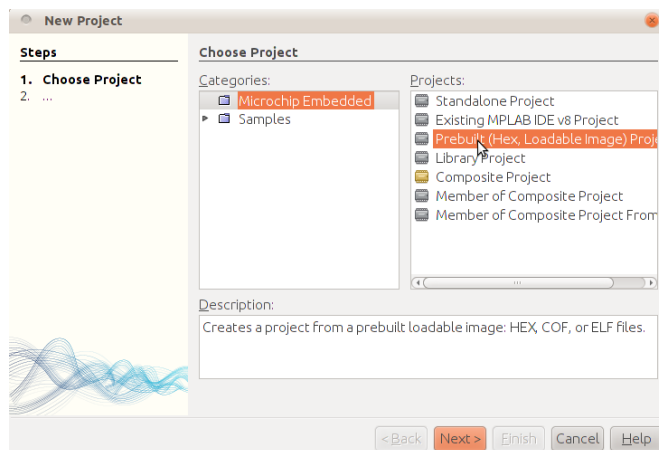


Figura A.26: Projecte precompilat MplabX -

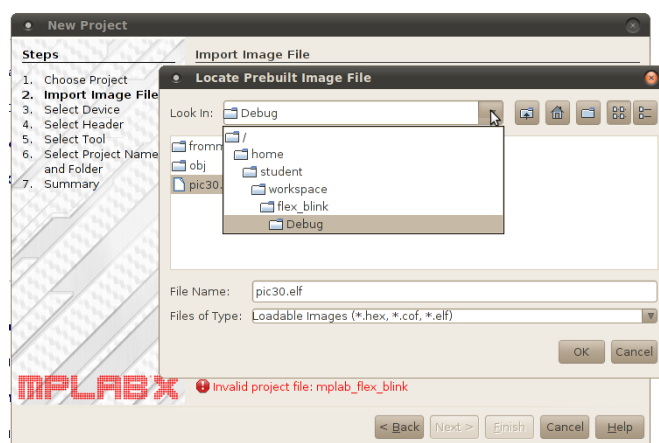


Figura A.27: Carregant fitxer .elf MplabX - El fitxer .elf ja esta precompilat.

A. GUIA DEL LABORATORI

Aquí seleccionem quin microcontrolador ens disposem a programar (Figura A.28), en el nostre cas es:

- Family: DSPIC33
- Device: dsPIC33FJ256MC710

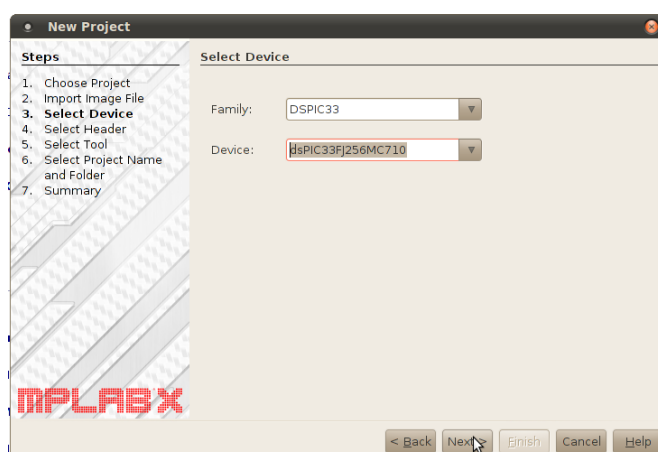


Figura A.28: Selecció de dispositiu MplabX - Aquí apareix una llista de microcontroladors per ser programats

Ara s'ha d'indicar quin és el programador que es vol utilitzar, en el nostre cas programarem les plaques Flex amb el ICD3 (Figura A.30). La connexió de la placa Flex i el programador hauria de estar ara mateix com la fotografia (Figura A.29):

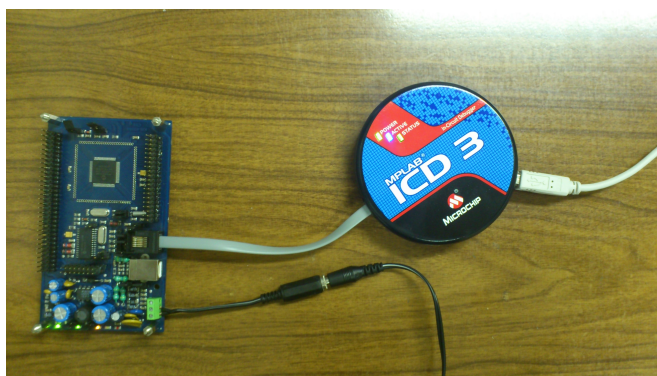


Figura A.29: Flex i Mplab ICD3 - Correctament connectats per ser programats.

En el cas que es triï una altra eina per programar-los s'ha de tenir en compte els colors en els que apareixen els diferents programadors:

Taula A.1: Compatibilitat dels programadors amb MplabX.

Color	Significat
Verd	Indica que aquest programador ha estat totalment testejat i certificat per l'ús general. Per tant és possible seleccionar-lo.
Groc	Indica que el programador ha estat minimament provat i només hauria de ser utilitzat per primeres proves. En aquest cas també ens permeten seleccionar-lo.
Vermell	Indica que el programador no és actualment suportat. En aquest cas no podrem seleccionar-lo.

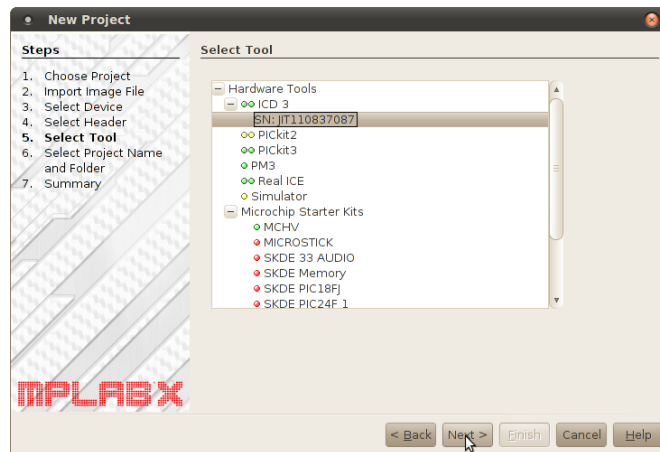


Figura A.30: Programadors MplabX - Els programadors apareixen de diferent color segons la compatibilitat amb MPLAB® X veure en la taula A.1

Ara només queda posar-li un nom al projecte (Figura A.31), i *sobretot vigilar de canviar el directori del projecte* i treure'l del directori *Debug* (Figura A.32). Si el deixesiu en aquest directori al fer qualsevol *Clean* al programa Eclipse podrieu tenir problemes en el projecte creat al MPLAB® X .

Finalment apareixeran els detalls del projecte que estem a punt de crear. Acceptem i ens deixarà el projecte en l'entorn de treball. A l'hora de programar la placa, clicarem al botó *Make and Program Device* a dalt a la dreta (Figura A.33).

Un cop programat hauria de començar a parpellejar el primer led taronja començant per l'esquerra (Figura A.34) (després dels tres leds verds que indiquen alimentació):

A. GUIA DEL LABORATORI

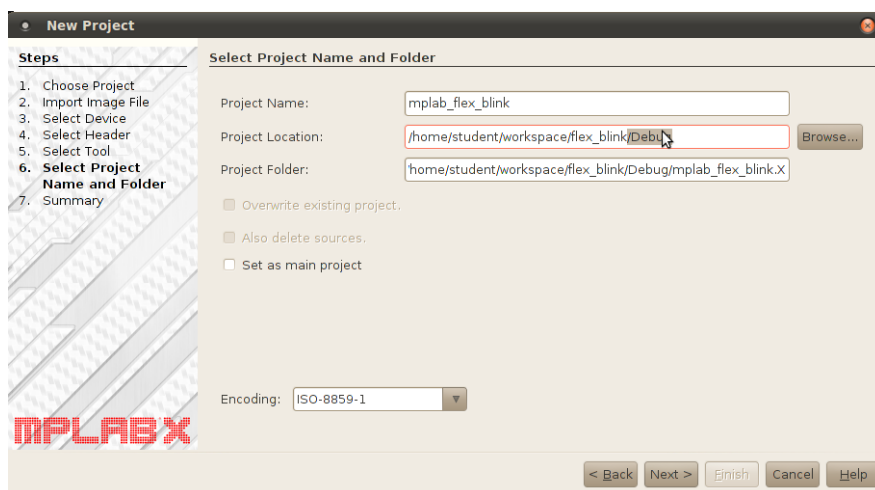


Figura A.31: Posant nom al projecte en MplabX -

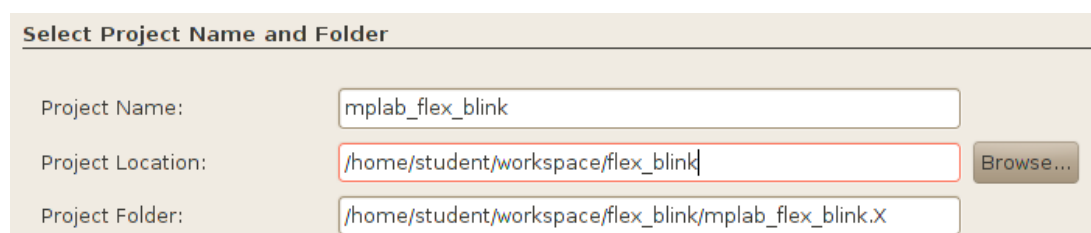


Figura A.32: Treure del directori *Debug* - S'ha de treure el projecte del directori *Debug* si no volem que MplabX tingui problemes amb els seus fitxers

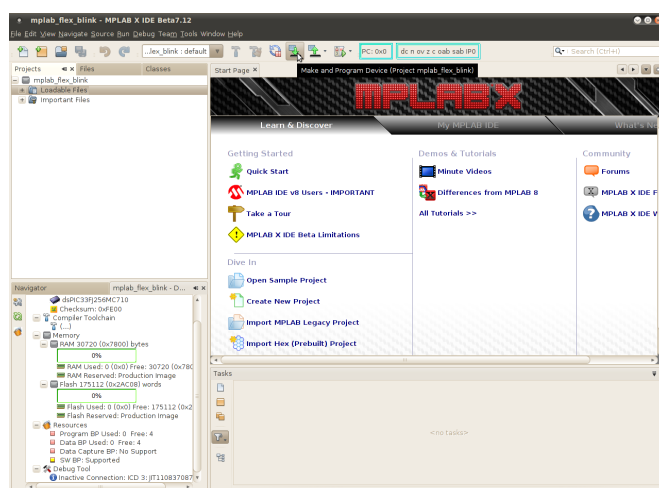


Figura A.33: Programar microcontrolador des de MplabX - Per programar el microcontrolador clicar el botó que apareix a dalt a la imatge

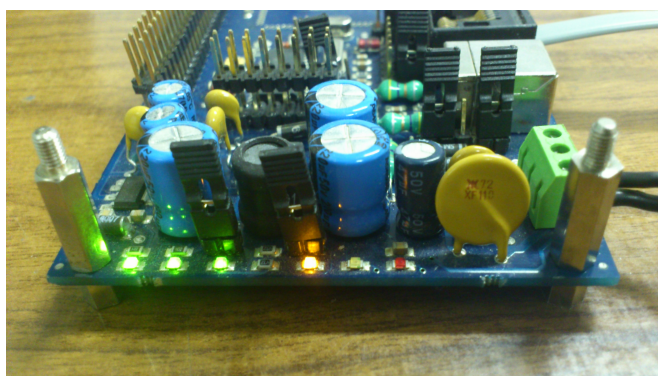


Figura A.34: Placa Flex amb programa *Blink* - Amb aquest programa carregat el primer led taronja parpelleja cada mig segon.

Apèndix B

Guia del Live CD

El laboratori que s'ha dissenyat pot ser executat sense haver de preparar tot un entorn de compilació i muntatge gracies a aquest Live CD.

Tot l'entorn està preparat perquè es pugui engegar el Live CD des de l'arranc del ordinador i ja es pugui treballar en ell.

Cal dir que la velocitat de resposta d'aquest entorn es algo inferior a la que podria donar un sistema instal·lat, però aquest sistema ens dona una portabilitat i una velocitat en la preparació del entorn.

De totes maneres el mateix Live CD compta amb l'opció de fer una instal·lació en el disc, compartint-lo amb un altre sistema operatiu si es necessari i podent arrancar des de qualsevol dels instal·lats.

També estan disponibles una serie de guies d'us del Live CD i del programa **DC-SMonitor** en format vídeo a la pàgina del Projecte de Final de Carrera "Design of a platform to test distributed control systems" (<http://code.google.com/p/pfc-platform-test/>) en l'apartat de vídeos (<http://code.google.com/p/pfc-platform-test/wiki/VideoExamples>).

B.1 Requisits mínims

Per tal d'iniciar el Live CD del laboratori, el nostre ordinador hauria de tenir els següents requisits mínims:

- Procesador: Intel x86 o compatible, amb $\geq 200\text{MHz}$
- Memoria RAM: 256 MB.

B. GUIA DEL LIVE CD

- Unitat lectora de DVD.
- BIOS: ha de ser capaç d'arrencar des de DVD.
- Targeta gràfica: estàndard, compatible amb SVGA.

B.2 Arrencant el *Live CD*

Abans de tot per arrencar un *Live CD* s'ha de configurar en la BIOS que abans d'arrencar des de el disc dur, provi d'arrencar desde el CD/DVD. Aixó en molts ordinadors s'aconsegueix apretant la tecla *F2*, o *Sup* varis cops només arrencar l'ordinador (aixó pot variar segons la màquina). Un cop estém en la BIOS normalment sol existir un apartat de *Boot* en el que podem escollir el primer lloc des de on arrencar, per tant primer de tot seleccionar com a primera opció la unitat de CD/DVD, i deixar com a segona opció el disc dur intern.

Un cop hem configurat això podem introduir el *Live CD* al lector i reiniciar l'ordinador guardant els canvis.

Quan arrenqui el *Live CD* apareixerà el següent text (figura B.1), cal esperar una mica abans de que aparegui el següent menú.



Figura B.1: Arrencant Live CD -

En aquest menú es pot arrencar el Live CD directament (primera opció), instal·lar la imatge al disc (tercera opció)..

Si volem arrencar desde el disc seguim a la secció Guia pas a pas del Live CD B.3.

Si el que volem es instal·lar el sistema en el nostre ordinador seleccionem la tercera opció i seguim els passos de la secció Instal·lant B.4

B.3 Guia pas a pas del Live CD

Al escollir arrancar el *Live CD* apareixerà el logotip d'*Ubuntu* carregant.

I al final el prompt d'usuari i contrasenya.

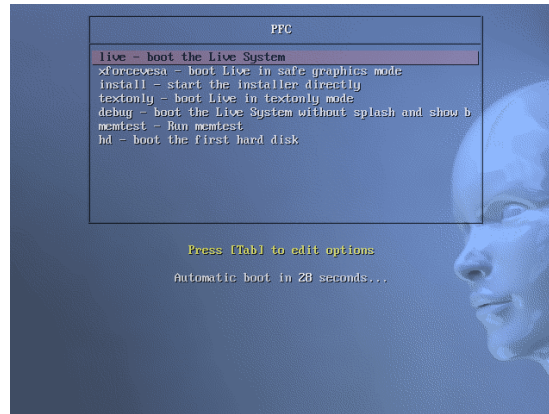


Figura B.2: Menu Live CD -

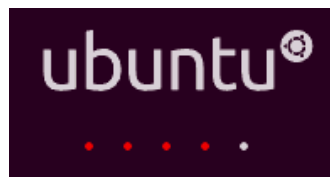


Figura B.3: Logotip de càrrega d'*Ubuntu* -

Usuari	student
Contrasenya	student

Taula B.1: Usuari i contrasenya *Live CD*

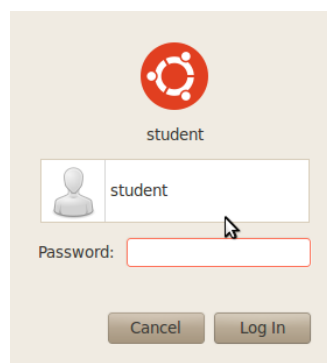


Figura B.4: Prompt d'usuari i contrasenya -

B. GUIA DEL LIVE CD

B.3.1 Estructura del *Live CD*

Un cop logejats, estarem en un escriptori típic d'*Ubuntu* (figura B.5), amb els següents elements d'interès:

MPLAB X IDE beta7.12	Accés directe al programa MPLAB [®] X , el farem servir per programar la placa <i>FLEX</i>
Eclipse	Accés directe al programa <i>Eclipse</i> , el farem servir per compilar els programes per la placa <i>FLEX</i>
Programs	Carpeta amb un recull del codi font de tots els programes utilitzats al laboratori

Taula B.2: Elements de l'escriptori

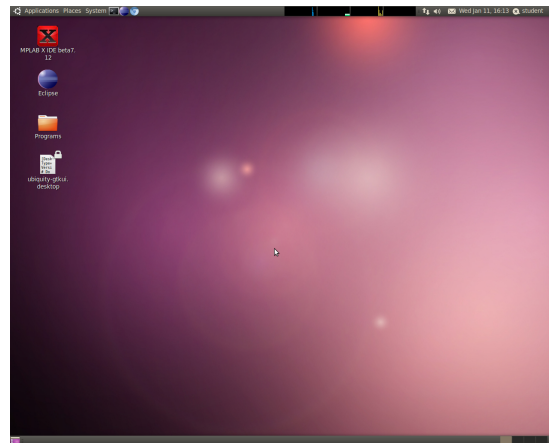


Figura B.5: Escriptori del *Live CD* -

Ara si entrem al directori `/home/student/workspace/Programs/` ens trobarem amb tots els codis necessaris per executar el laboratori comprimits en .zip (figura ??).

DCSMonitor	Aquest és el programa que s'executa a l'ordinador per veure el control que s'està realitzant. Necessita comunicar-se via RS232 amb una placa <i>FLEX</i> amb el programa DSPIC_SENSOR instal·lat.
DSPIC_SENSOR	Aquest és el programa per la placa <i>FLEX</i> que actua sobre el doble integrador, representa un dispositiu <i>Sensor/Actuador</i> . Ja que obté els valors de sortida del doble integrador (feina de <i>Sensor</i>), i també assigna el valor d'entrada (feina d' <i>Actuador</i>).
DSPIC_CONTROLLER	Aquest és el programa per la placa <i>FLEX</i> que fa els càlculs del control per el doble integrador, aquests càlculs els envia pel bus CAN cap al <i>Sensor/Actuador</i> .
DSPIC_MONITOR	Aquest és el programa per la placa <i>FLEX</i> que monitoritza tot el bus CAN.

Taula B.3: Elements del directori $\sim /workspace/Programs/$

Descomprimir els que ens calguin per començar a compilar i executar els diferents codis.

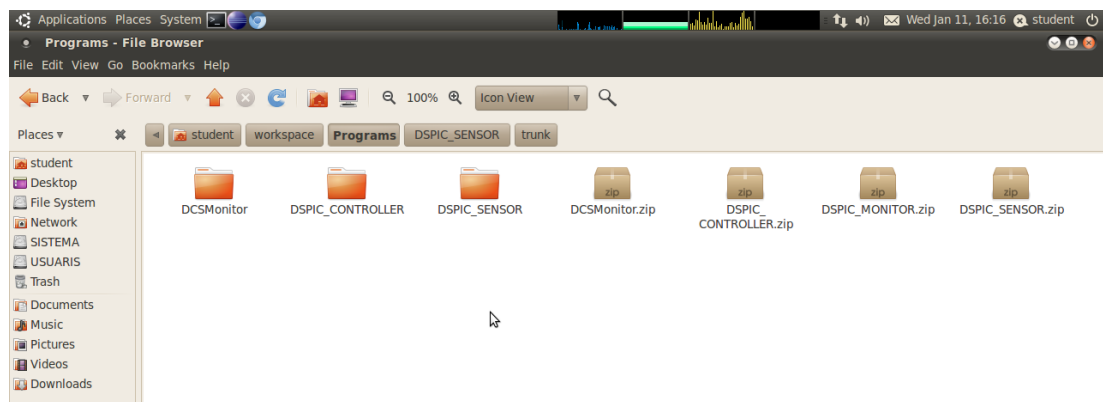


Figura B.6: Codis comprimits dels programes del laboratori -

B.3.2 Compilant amb Eclipse el DSPIC_SENSOR

Un cop tenim l'estructura dels directoris que hi ha ben clara, procedim a obrir el programa de l'escriptori *Eclipse* . Al obrir-lo per primer cop, ens preguntarà quin es el directori per defecte dels nostres projectes, així que deixeu per defecte el que ens apareix (Podeu marcar la casella inferior esquerra perquè no torni a aparèixer el missatge) (figura B.7).

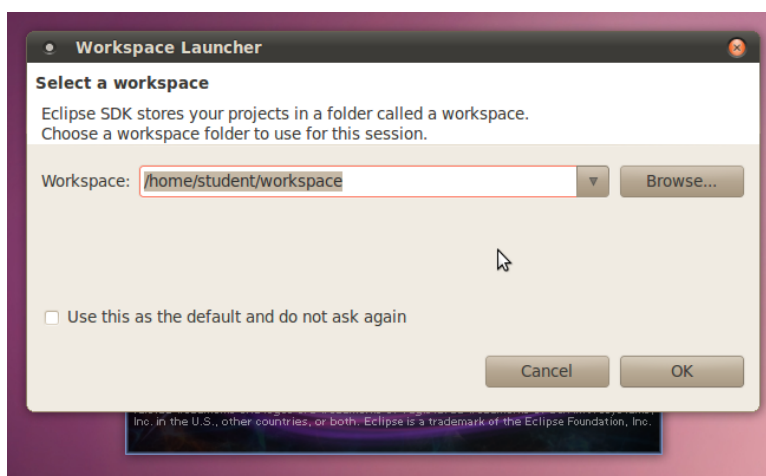


Figura B.7: Indicant espai de treball a *Eclipse* -

Ja en la pantalla principal d'*Eclipse*, anem a crear un nou projecte *RT-Druid*, per fer això anem a *File* → *New* → *NewProject*, i a la finestra que apareix, seleccionem *Evidence* → *RT - DruidOilandC/C++Project* (figura B.8).

Despres desmarquem la casella *Use default location*, i anem a buscar el codi del *Sensor* al workspace : `/home/student/workspace/Programs/DSPIC_SENSOR/trunk` I tot seguit li posem un nom al projecte (figura B.9).

Un cop ens ha creat el nou projecte, el compilem amb clicant a *Project* → *Clean*, amb aquesta opció netejarem el directori d'antigues compilacions (això es útil per estar segurs que el que compila es tot nou) i marcarem la opció *Start build immediately* perquè un cop netejat compili el codi (figura B.10).

Començarà a compilar i un cop acabat a la Consola (finestra de text de la part inferior del programa) ha d'aparèixer el següent missatge:

Compilation terminated successfully!

B.3.3 Gravant amb MPLAB® X el DSPIC_SENSOR

Un cop compilat el programa, obrim MPLAB® X (figura B.11)

I creem un nou projecte important el .elf que ens ha generat *Eclipse* :

File → *ImportHex(Prebuilt)Project*

I introduïm el directori on ens ha generat el fitxer .elf (figura B.12):

`/home/student/workspace/Programs/DSPIC_SENSOR/trunk/Debug/pic30.elf`

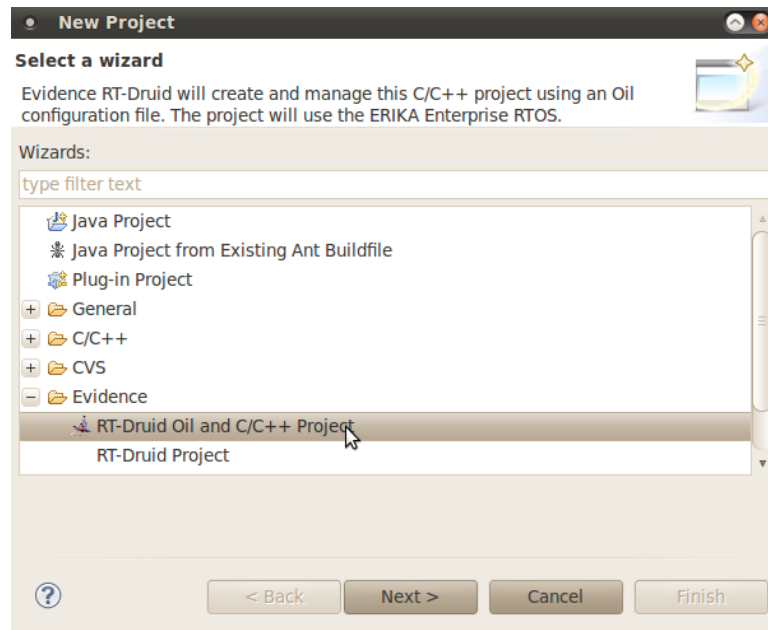


Figura B.8: Seleccionant RT-Druid project a *Eclipse* -

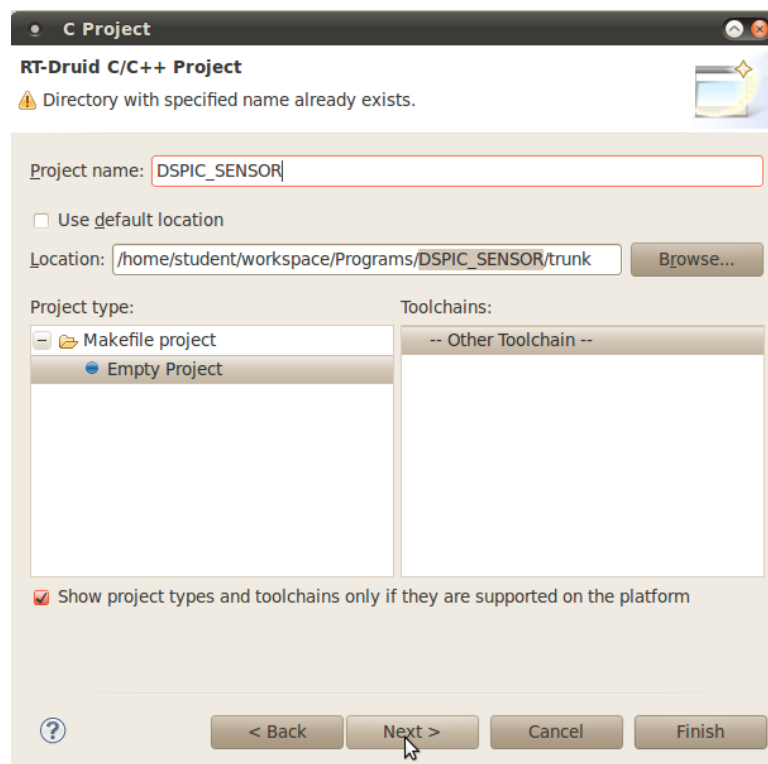


Figura B.9: Seleccionant directori del projecte a *Eclipse* -

B. GUIA DEL LIVE CD

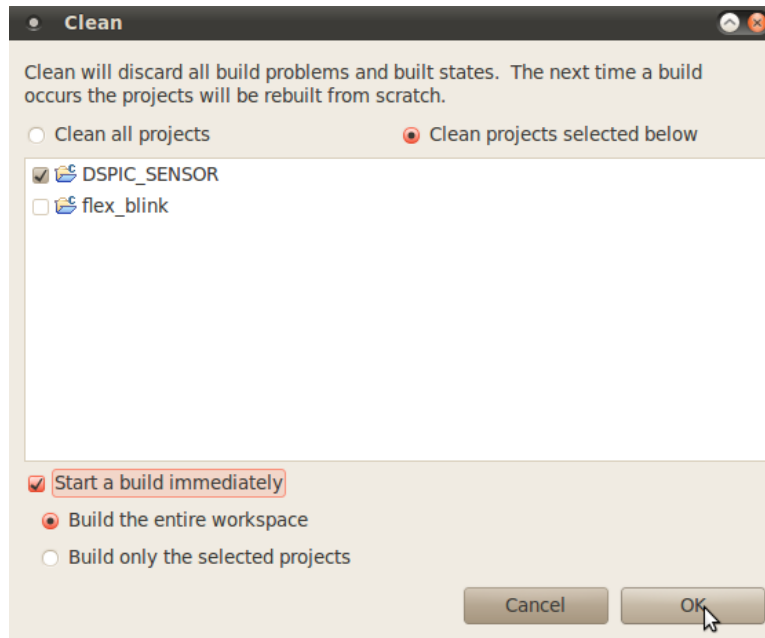


Figura B.10: Netejant i compilant a *Eclipse* -



Figura B.11: Logo carregant MPLAB® X -

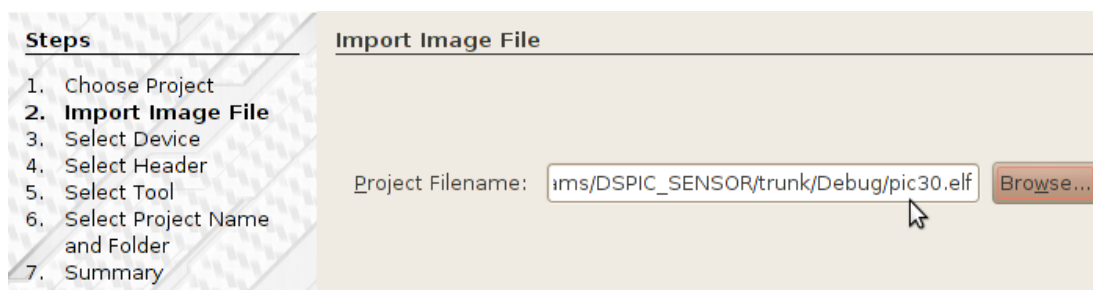


Figura B.12: Seleccionant fitxer precompilat MPLAB® X -

Aquí seleccionem quin microcontrolador ens disposem a programar (Figura B.13), en el nostre cas es:

- Family: DSPIC33
- Device: dsPIC33FJ256MC710



Figura B.13: Selecció de microcontrolador a gravar en MPLAB® X -

Ara s'ha d'indicar quin és el programador que es vol utilitzar, en el nostre cas programarem les plaques Flex amb el ICD3 (Figura B.15). La connexió de la placa Flex i el programador hauria de estar ara mateix com la fotografia (Figura B.14):

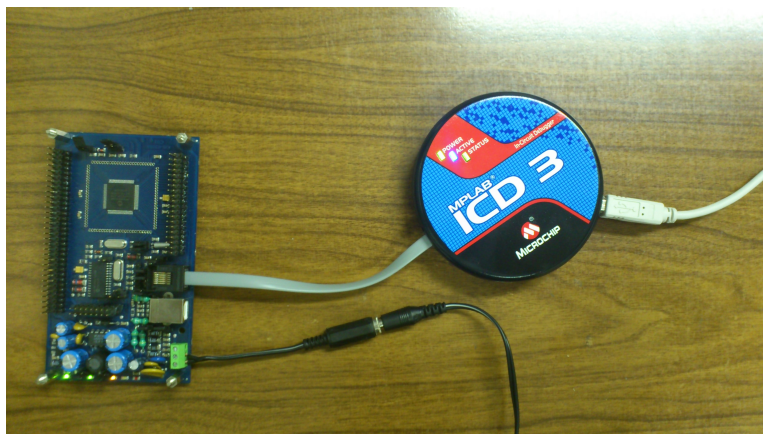


Figura B.14: Connexió del programador ICD3 i placa *FLEX* -

En el cas que es triï una altra eina per programar-los s'ha de tenir en compte els colors en els que apareixen els diferents programadors, per saber que significa cada color dels colors consulteu a la taula B.4.

B. GUIA DEL LIVE CD

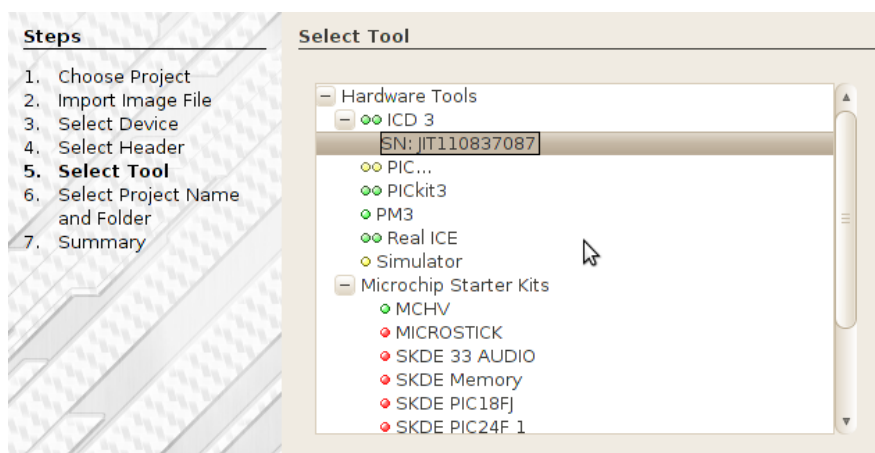


Figura B.15: Seleccionant programador en MPLAB® X -

Taula B.4: Compatibilitat dels programadors amb MplabX.

Color	Significat
Verd	Indica que aquest programador ha estat totalment testejat i certificat per l'us general. Per tant és possible seleccionar-lo.
Groc	Indica que el programador ha estat minimament provat i només hauria de ser utilitzat per primeres proves. En aquest cas també ens permeten seleccionar-lo.
Vermell	Indica que el programador no és actualment suportat. En aquest cas no podrem seleccionar-lo.

Ara només queda posar-li un nom al projecte (Figura B.16), i *sobretot vigilar de canviar el directori del projecte* i treure'l del directori *Debug* (Figura B.17). Si el deixessiu en aquest directori al fer qualsevol *Clean* al programa Eclipse podríeu tenir problemes en el projecte creat al MPLAB[®] X.

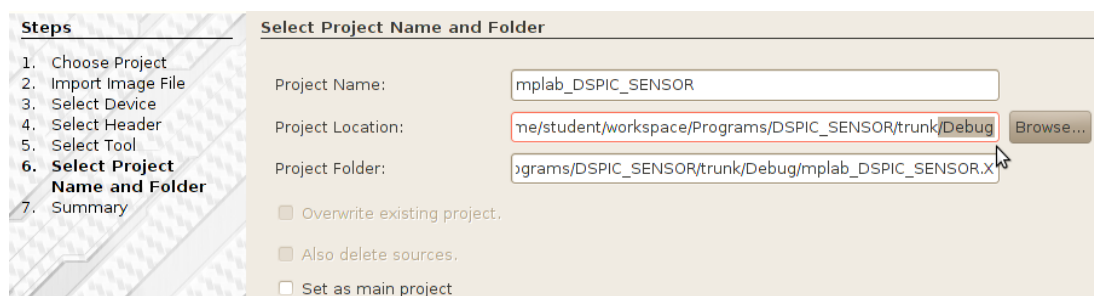


Figura B.16: Canviant el directori del projecte a MPLAB[®] X - S'ha de treure del directori */Debug*

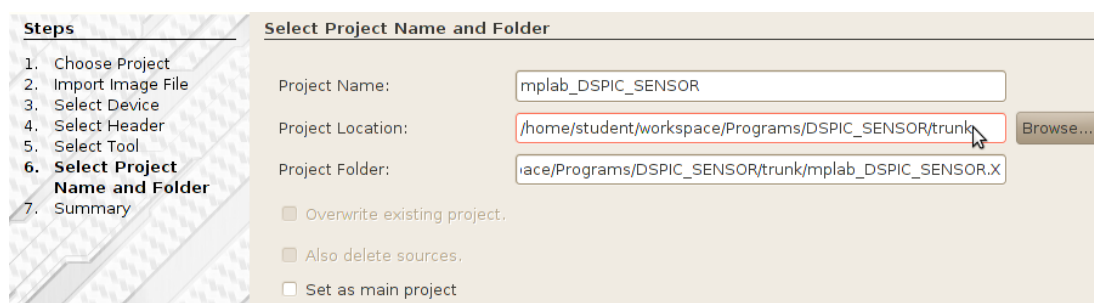


Figura B.17: Directori del projecte ven posat a MPLAB[®] X -

Finalment apareixeran els detalls del projecte que estem a punt de crear. Acceptem i ens deixarà el projecte en l'entorn de treball. A l'hora de programar la placa, clicarem al botó *Make and Program Device* a dalt a la dreta, i si tot funciona hauria d'apareixer a la part inferior del programa el següent text (Figura B.18).

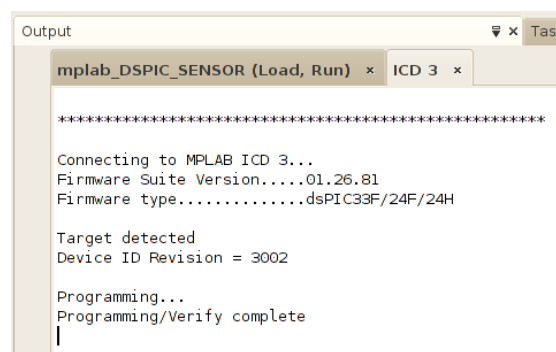


Figura B.18: Microcontrolador programat correctament a MPLAB® X -

B.3.4 Resum per gravar la resta

Els passos per compilar i gravar la resta de programes que ens interessin (DSPIC_CONTROLLER i/o DSPIC_MONITOR) són exactament els mateixos que hem fet per programar el DSPIC_SENSOR. Es possa aquesta secció per fer un resum de tots els passos a seguir a mode de recordatori:

1. Descomprimim els codis de la carpeta */home/student/workspace/Programs/*
2. Obrim el programa *Eclipse*
 - (a) Creem un nou projecte RT-Druid
 - (b) Seleccionem el directori on es troba el projecte, i li posem un nom.
 - (c) Creem el projecte
 - (d) Natejem l'entorn i compilem
3. Obrim el programa MPLAB® X
 - (a) Creem un nou projecte amb el codi precompilat
 - (b) Busquem el fitxer .elf (dintre del directori del projecte creat amb Eclipse i dintre de Debug)
 - (c) Seleccionem el microcontrolador DSPIC33 i dsPIC33FJ256MC710
 - (d) Endollem el programador ICD3 al ordinador per USB
 - (e) Seleccionem dintre del programador ICD3 el numero de serie que ens aparegui.
 - (f) Posem un nom al projecte i **eliminem la part Debug** de la localitzacio
 - (g) Acceptem i programem.

B.3.5 DCSMonitor

El programa **DCSMonitor** ens permet comprobar si el control dels dispositius s'està executant correctament. Per poder executar el programa aneu al directori */home/student/Desktop/Programs/DCSMonitor* i executeu el programa *main_dcsmon.py* (figura B.19)

B. GUIA DEL LIVE CD

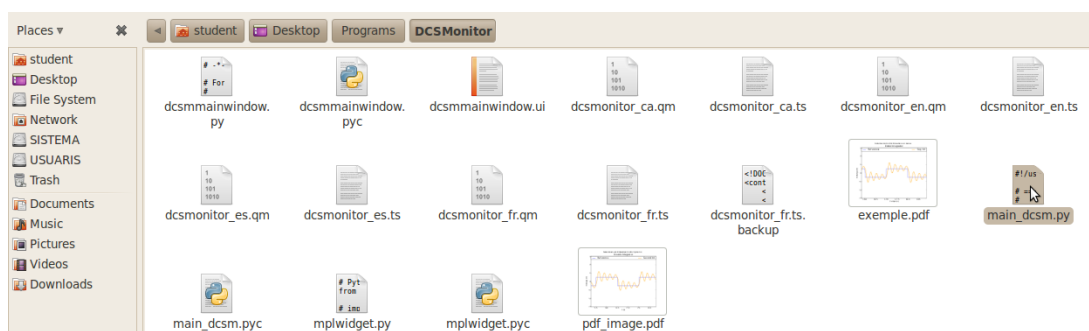


Figura B.19: Directori amb el codi de DCSMonitor -

Os demanarà si voleu executar-lo en una terminal o directament executar-lo. Agafeu la opció que preferiu.

Un cop obert tindreu davant un programa amb l'aspecte del de la figura B.20. En aquest programa podreu veure el control dels vostres dispositius en la part central esquerra en una gràfica. A la part dreta tindreu el botó per connectar el port serie. Just a sota els llaços de control que hi hagi connectats (en cas que tingueu un dispositiu *Monitor* connectat al bus CAN), sota d'aixó podeu seleccionar quins són els plots que voleu que apareguin en la gràfica, després un botó per començar a rebre les dades del llaç de control, un botó per netejar el text, i valors estadístics.

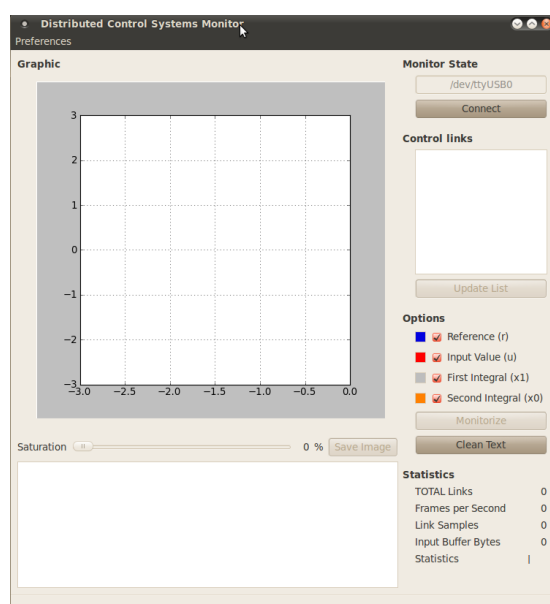


Figura B.20: Pantalla principal de DCSMonitor -

Abans de res podeu anar a les Preferències del programa per poder escollir l'idioma i si connectareu el programa amb un dispositiu *Monitor* , o amb un dispositiu *Sensor/Actuador* (figura B.21).

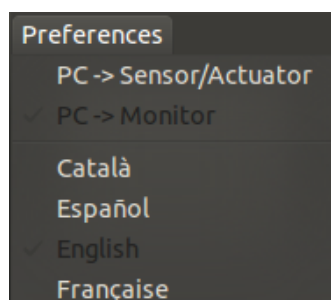


Figura B.21: Preferències -

Un cop hem seleccionat l'idioma i a quin dispositiu estem connectats mitjançant el port RS232, pode clicar en el botó *Connectar*, i si tot va bé hauria de pintar-se de verd i indicar que està connectat com a la figura B.22.

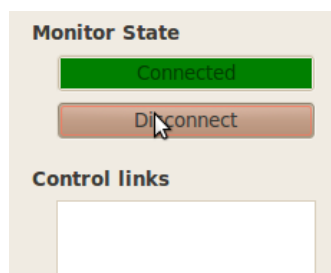


Figura B.22: Connectant el port serie a DCSMonitor -

Tot seguit si esteu connectats a un dispositiu *Monitor* salteu a la secció Connectat a un dispositiu *Monitor* B.3.5.2.

Si en canvi esteu connectats a un dispositiu *Sensor/Actuador* seguiu en la següent secció Connectat a un dispositiu *Sensor/Actuador* B.3.5.1.

B.3.5.1 Connectat a un dispositiu *Sensor/Actuador*

En aquest mode d'execució només està habilitada la opció de monitoritzar, netejar el text i exportar la gràfica per tant si cliqueu en *Monitoritzar* i teniu els dispositius correctament programats i connectats hauria de començar a aparèixer text en la part inferior de l'espai de la gràfica, indicant les lectures de l'estat del control, i també hauria

B. GUIA DEL LIVE CD

d'aparèixer una gràfica en temps real, tal com surt en la figura B.23. Si volem exportar la gràfica podem fer-ho un cop hem deixat de monitoritzar (veure en la secció B.3.5.3).

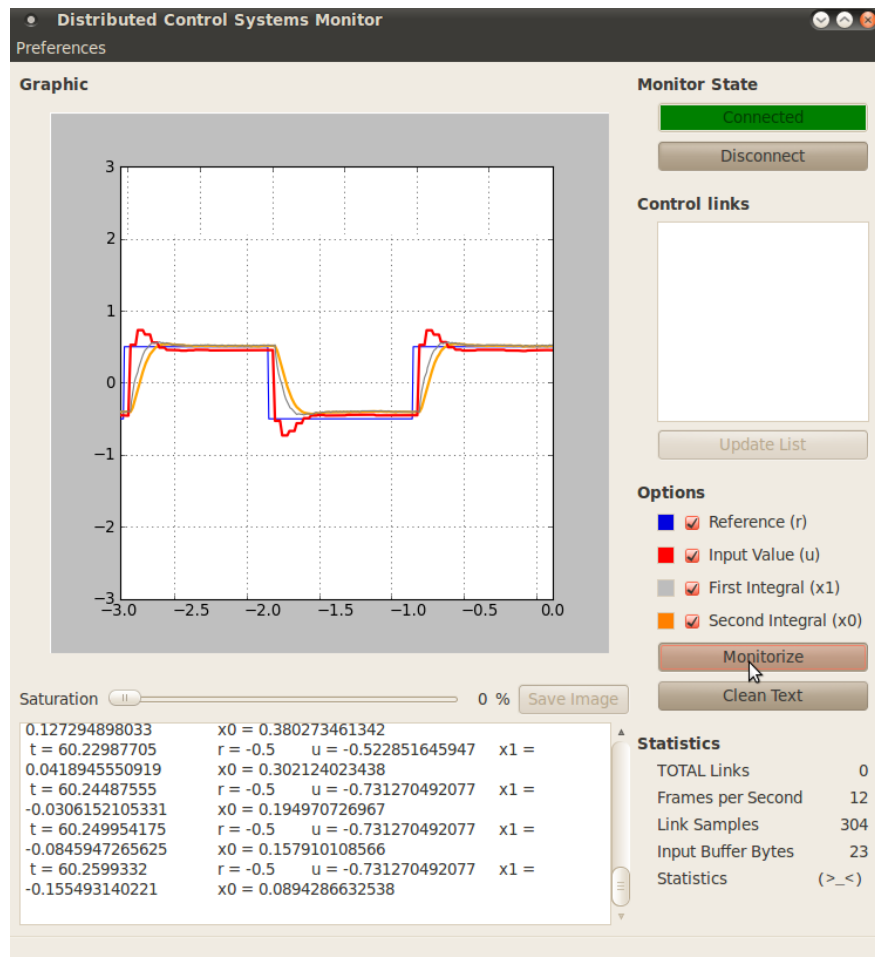


Figura B.23: Programa DCSMonitor connectat a un *Sensor/Actuador* - Al monitoritzar ha de dibuixar una gràfica en temps real i escriure els valors de l'estat

B.3.5.2 Connectat a un dispositiu *Monitor*

En el mode d'execució del *Monitor* estan totes les opcions habilitades. Primer de tot si cliquem en el botó *Llistar Dispositius* ens apareixerà una llista de tots els llaços de control que hi hagi connectats al bus CAN (exemple en la imatge B.24).



Figura B.24: Demanant un llistat de dispositius a DCSMonitor -

Un cop tenim la llista dels llaços de control, podem deixar d'actualitzar-la tornant a clicar el botó, per tal de baixar la carrega de l'ordinador i del dispositiu *Monitor*.

Despres seleccionem un llaç de control de la llista i donem al botó *Monitoritzar* per començar a rebre el seu estat. Hauria d'aparèixer una gràfica en moviment i els diferents valors de lectura en la part inferior (figura B.25). I si volem que no es mostri algun dels plots de la gràfica només hem de desseleccionar-lo a la llista que apareix en les *Opcions* (veure figura B.26).

Una altra opció que tenim al estar connectats al *Monitor* es enviar-li un senyal per que comenci a generar carrega al bus CAN, per fer això només hem de desplaçar la barra *Saturació* (figura B.27) fins al punt que ens interressi i d'aquesta manera el dispositiu *Monitor* passarà a enviar missatges amb màxima prioritat al bus CAN.

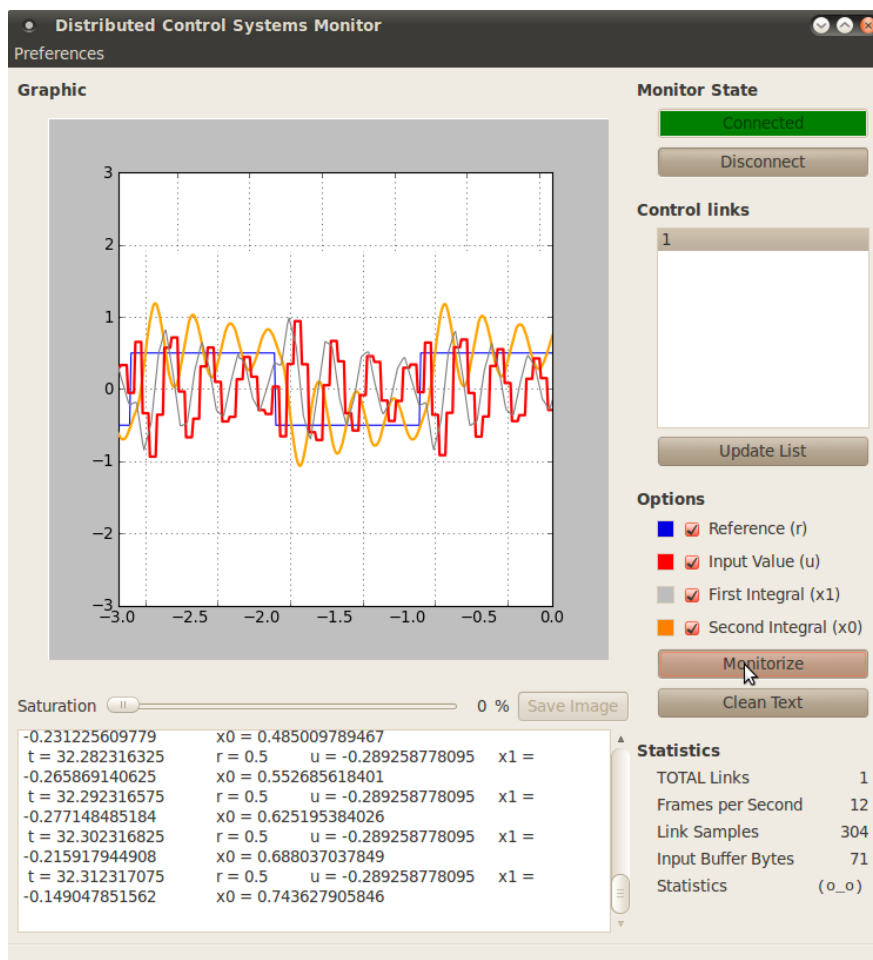


Figura B.25: Gràfica amb tots els plots -

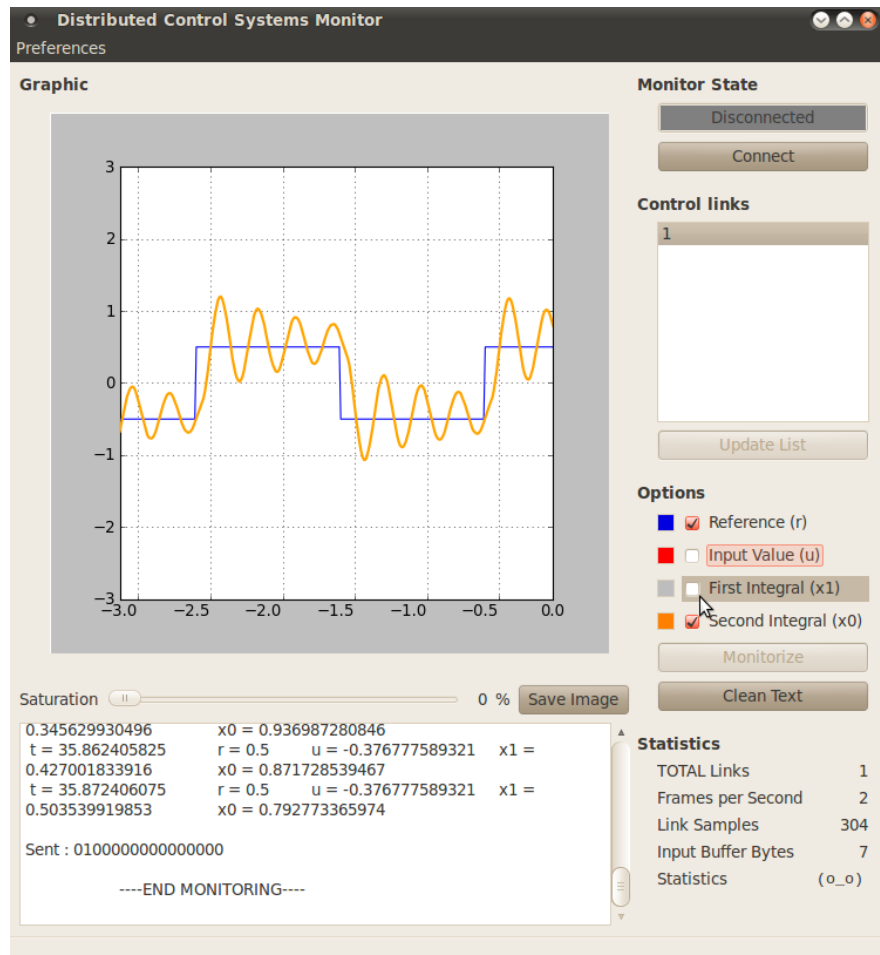


Figura B.26: Gràfica amb plot de referència i segona integral -

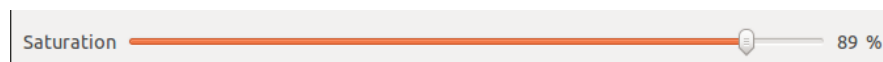


Figura B.27: Barra lliscant per augmentar la carrega del bus -

B. GUIA DEL LIVE CD

B.3.5.3 Exportant gràfica generada

En qualsevol dels modes d'execució del programa podem capturar un dels moments de la gràfica, per fer això només hem d'aturar la monitorització en l'instant que ho desitgem, seleccionem els plots que vulguem que apareguin i li donem al botó *Guardar imatge* just a la dreta de la barra de carrega. Un cop haguem clicat apareixerà una finestra nova (figura B.28) on hem d'indicar-li el lloc on guardar la imatge i el nom amb l'extensió suportada que preferim (.pdf, .eps, .ps, .png, entre d'altres) (exemple d'imatge creada en l'escriptori en la figura B.29).

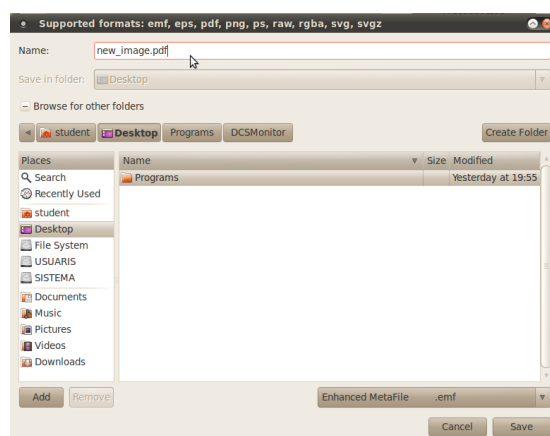


Figura B.28: Posant un nom a la gràfica a exportar -

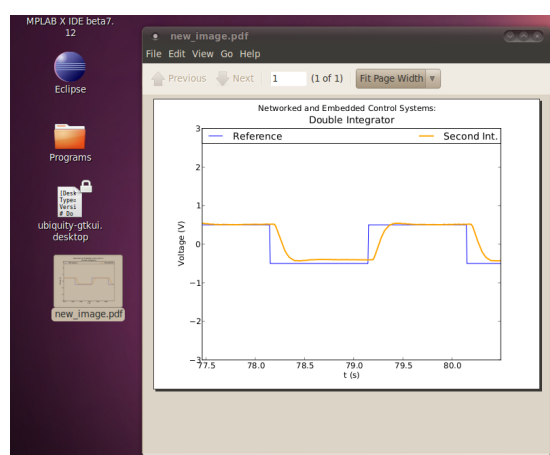


Figura B.29: Exemple havent exportat una gràfica -

B.4 Instal·lant

Com hem indicat el sistema pot ser instal·lat a l'ordinador, compartint l'espai amb un altres sistema operatiu, o instalant-lo completament sol a l'ordinador. Fer una instal·lació d'un sistema operatiu malament podria comportar esborrar informació important del vostre ordinador, per tant feu això només si esteu segurs del que féu.

També existeix la opció de crear una maquina virtual (com pot ser amb VirtualBox o VMWare) i d'aquesta manera treballar desde el nostre sistema operatiu actual, i corrent una maquina virtual amb el laboratori.

Tot seguit es podrà veure una guia pas a pas de com instal·lar aquest live CD.

En el menú principal del CD (figura B.30), seleccionar la tercera opció *Install*.

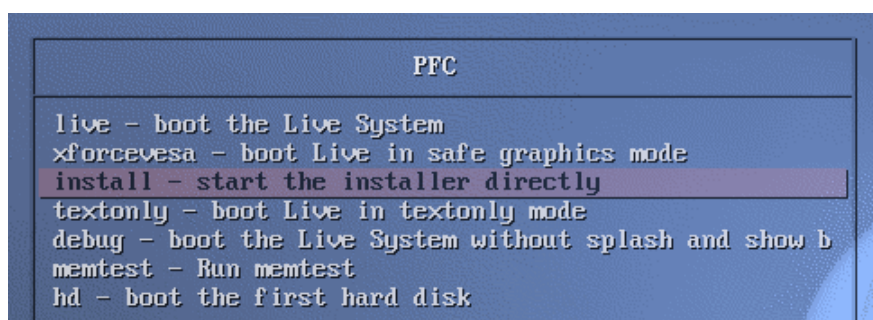


Figura B.30: Menú principal LiveCD, instalant -

Un cop hem seleccionat instal·lar apareixerà el logotip d'*Ubuntu* (figura B.31) i començarà a carregar l'entorn d'instal·lació.



Figura B.31: Logotip d'*Ubuntu* carregant -

Un cop l'entorn hagi carregat, el primer que ens sortirà serà que seleccionem l'idioma de la instal·lació, així que seleccioneu l'idioma desitjat i cliqueu endavant.

Després os farà escollir la vostra franja horària per poder sincronitzar el rellotge, i tot seguit el tipus de teclat que tingueu.

Fins aquí no hi ha cap problema, ara ve quan hem d'escollir on instal·lar el *Live CD*.

B. GUIA DEL LIVE CD

En aquesta pantalla ens demana com volem instal·lar el sistema operatiu del *Live CD* (figura B.32).

Aquí tenim varies opcions :

- Instalar el *Live CD* adjacent al sistema operatiu que tinguem instalat (opció recomenada).
- Instalar el *Live CD* esborrant l'actual sistema operatiu.
- Instalar el *Live CD* especificant les particions manualment (avançat).

Si no volem complicar-nos seleccionem la primera opció, i escollim l'espai que volem assignar a cada un dels sistemes operatius que hi ha a l'ordinador.

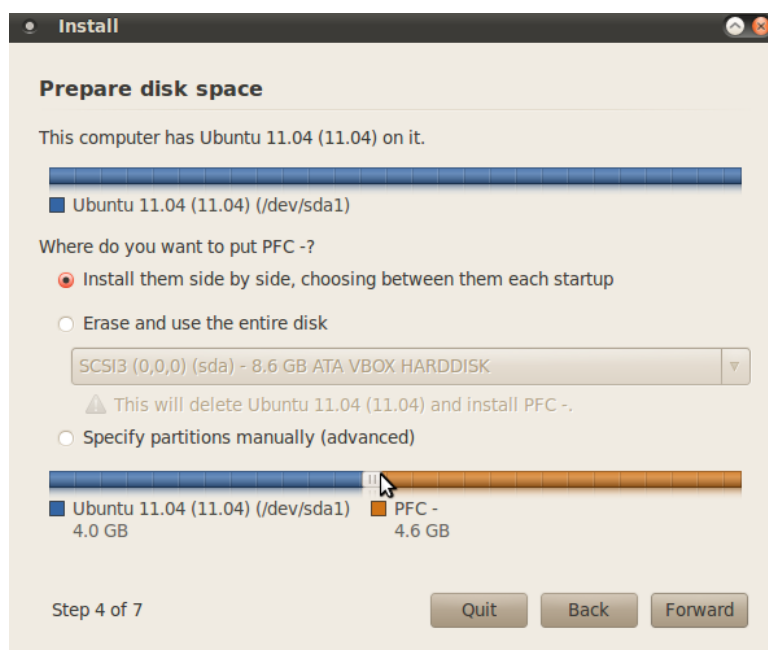


Figura B.32: Preparant particions del disc pel LiveCD -

Quan esteu acceptant os saltarà un avís indicant que aquests canvis s'efectuarà al disc dur i per tant aquest pas podrà no ser reversible, per tant accepteu si sabeu el que esteu fent.

Ara tocarà esperar una estona mentre fa les particions que li hàgim indicat. Un cop particionat el disc ens demanara que li indiquem un usuari, contrasenya i nom de la màquina (figura B.33). Aquest *Live CD* ha estat creat perquè només crei l'usuari

i contrasenyes predefinits *student* i *student*, per tant posar aquestes dades, ja que de totes maneres al arrencar des del disc dur intern l'usuari i contrasenya seran *student*.

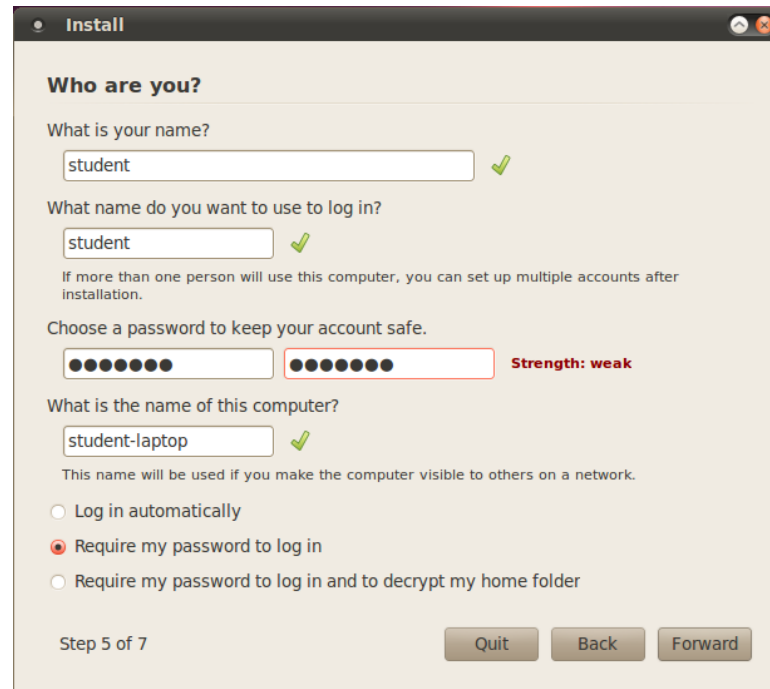


Figura B.33: Configurant usuari i contrasenya - Deixar com a usuari i contrasenya la paraula *student*

Un cop totes les opcions estiguin omplertes continuarem, i apareixerà un resum de tot el que es desitja realitzar. Acceptar en aquest punt i començarà la instal·lació (això pot trigar varis minuts segons la màquina).

Quan la barra arribi al 100% apareixerà un avis d'instal·lació completada, i demanara que es reiniciï l'ordinador, així que doneu-li a *Reiniciar Ara*, i quan os ho demani treieu el disc i doneu-li a l'intro.

Quanengegueu de nou l'ordinador seleccioneu el sistema operatiu instal·lat (ubuntu) i després de carregar apareixerà el prompt per introduir usuari i contrasenya (figura B.34). Així que introduïu l'usuari i contrasenya *student*, i ja tindreu l'entorn del laboratori instal·lat i preparat per provar els diferents codis (escriptori del *Live CD* un cop instal·lat B.35).

Si s'instal·la en una màquina virtual de VirtualBox podem instal·lar un paquet anomenat *Guest Additions*. Aquest paquet ens permet una interacció més amigable amb

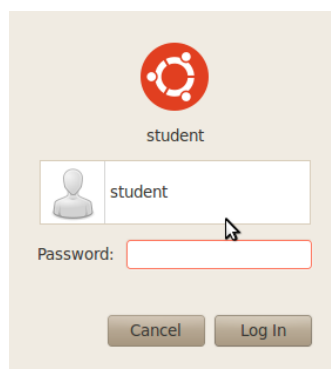


Figura B.34: Prompt de login -



Figura B.35: Escritori del *Live CD* recent instal·lat -

la màquina virtual. Per instal·lar això hem d'engegar primerament la màquina virtual, i un cop estem en l'escriptori seleccionar a la finestra de la part superior *Dispositivos* – > *InstalarGuestAdditions*, amb això apareixerà un CD nou a l'escriptori de la màquina virtual, així que obriu-lo i seguiu les seves instruccions.

Si la màquina amfitrió resulta ser una màquina *Ubuntu*, pot ser que després d'instal·lar els Guest Additions la màquina virtual no canvia la resolució al re-dimensionar la finestra. Això és un error conegut per aquesta versió d'*Ubuntu* ja que no reconeix el XSERVER. Per solucionar aquest problema s'ha de fer el següent en la màquina virtual:

1. `sudo apt-get update`
2. `sudo apt-get install build-essential linux-headers-$(uname -r)`
3. `sudo apt-get install virtualbox-ose-guest-x11`

B. GUIA DEL LIVE CD

Bibliografia

- [1] TILL KORTEN AND HARISH BHANDERI. **LaTeX template for PhD thesis**, 2010. v
- [2] MICROCHIP TECHNOLOGY INC. **dsPIC33FJXXXMCX06A/X08A/X10A Data Sheet**, 2011. 18
- [3] PACKAGE TYPES AND BLOCK DIAGRAM. **High-Speed CAN Transceiver MCP2551**. *Technology*, pages 1–24, 2010. 19
- [4] PAU MARTI AND JOSEP M. FUERTES. **CAN (Controller Area Network)**. In *Comunicacions Industrials*, pages 155–175. ESAII EUPVG UPC, Vilanova i la Geltrú, 2000. 23
- [5] IH KASCHEL. **Análisis de la capa física del bus de campo can**. *cabierta.uchile.cl*, pages 1–6, 2003. 23
- [6] BOSCH. *CAN Specification version 2.0*. Bosch, 1991. 23
- [7] WIKIPEDIA. **RS-232**, 2012. 24
- [8] MANEL VELASCO AND PAU MARTÍ. **Networked and Embedded Control Systems (NECS): Double Integrator Control Lab**. Technical report, Technical University of Catalonia, Barcelona, Spain, 2011. 29
- [9] SANDRO TOSI. **Embedding Matplotlib in Qt 4**. In *Matplotlib for Python Developers*, number 6, chapter 6, page 308. PACKT, 2009. 38
- [10] MANEL VELASCO, JOSEP M FUERTES, PAU MARTÍ, JOSÉ YÉPEZ, AND RICARD VILLÀ. **Schedulability Analysis for CAN-based Networked Control Systems with Dynamic Bandwidth Management**. Technical report, UPC, Barcelona, Spain, 2009. 39, 40
- [11] YASSINE BENABBAS. **Créer un graphe dynamique avec PyPlot**, 2011. 59
- [12] PAU MARTÍ, MANEL VELASCO, JOSEP M FUERTES, ANTONIO CAMACHO, AND GIORGIO BUTTAZZO. **Design of an Embedded Control System Laboratory Experiment**. 2010.
- [13] STAND-ALONE C A N CONTROLLER AND WITH SPI. **MCP2515 - Stand-alone CAN Controller With SPI Interface**. *Writing*, pages 1–88, 2010.
- [14] PAU MARTÍ, MANEL VELASCO, JOSEP GUARDIA, FREDERIC PÉREZ, JOSÉ YÉPEZ, JORDI AYZA, RICARD VILLÀ, AND JOSEP M FUERTES. **Control Performance Optimization in Networked Embedded Systems**.
- [15] MANEL VELASCO, CAMILO LOZOYA, AND JORDI AYZA. **Distributed Control Systems (DCS) Group: Activity Report**. Technical report, Technical University of Catalonia, Barcelona, 2006.
- [16] NOKIA CORPORATION. **Qt Reference Documentation**, 2008.

BIBLIOGRAFÍA

- [17] UBUNTU. **Live CD de Ubuntu**, 2008.
- [18] UNIVERSITY OF CAMBRIDGE. **PhD/MPhil Thesis - a LaTeX Template**, 2011.
- [19] PAU MARTÍ, ROSA CASTAÑÉ, MANEL VELASCO, JOSEP GUARDIA, AND JOSEP M FUERTES. **A CAN Application Profile for Control Optimization in Networked Embedded Systems**. 2006.
- [20] EVIDENCE. *RT-Druid Code generator Plugin reference manual*. Evidence, Pisa - Italy, 1.5.0 edition, 2011.

Declaracions

Per la present declaro que he elaborat aquest document sense l'assistència prohibida de tercers i sense fer ús diferent de les ajudes especificades; les nocions assumides directament o indirectament d'altres fonts han estat identificades com a tals. Aquest document no ha estat presentat de manera idèntica o similar a qualsevol altra junta examinadora Catalana o estrangera.

El Projecte Final de Carrera ha estat produït des del 20 de Juny del 2011 fins al 17 de Gener de 2012 sota la supervisió de Josep M. Fuertes Armengol i Manel Velasco Garcia en la Facultat de Matemàtiques i Estadística de la Universitat Politècnica de Catalunya.

Barcelona,