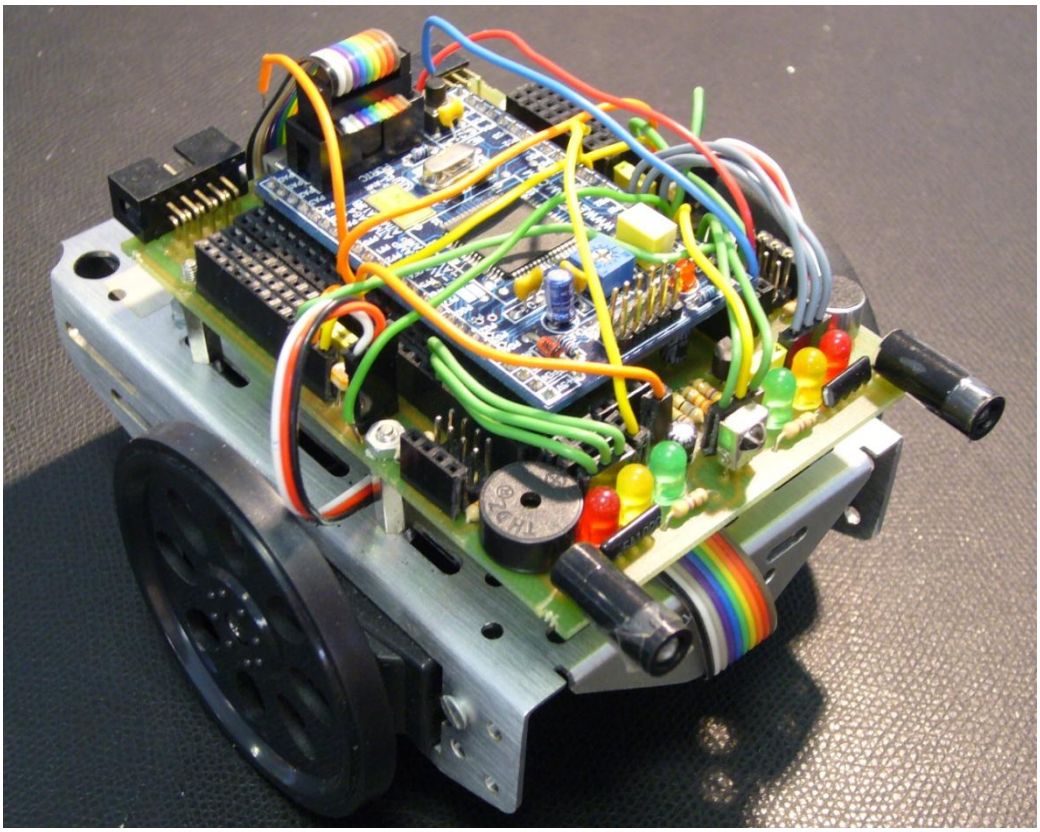
UABot

Projecte d'un robot educatiu



Joan Oliver Malagelada
Ricardo Toledo Morales
Josep Maria Sanz Xicola

Revisions del document:

Versió	Canvis
1.0	Redacció actual document.

Índex

1. Introducció

UABot és una plataforma robòtica dissenyada i programada a la Universitat Autònoma de Barcelona. La plataforma és una eina que compagina hardware/software per el govern d'un robot a través d'un ordinador host. La finalitat del següent document és presentar una explicació clara del hardware així com també de les diferents capes software. Per a tal efecte, el document es desglossa en tres capítols principals.

El primer capítol procura proporcionar una breu explicació de les diferents parts hardware. Junt amb cada explicació, es mostra la configuració i disposició de cada component.

Seguidament, es descriu el desenvolupament software ideat per al control de la plataforma. Es fa un incís teòric sobre sistemes operatius en temps real, en concret sobre el sistema operatiu FreeRTOS, per posteriorment aprofundir en l'explicació del funcionament dels drivers i el mode d'operació del sistema.

El tercer capítol ocupa tota la part referent al computador host. Es descriu l'API escrita en Python que interacciona amb el robot. També es presenta un cas d'ús típic detallat pas per pas per a una fàcil instal·lació i primera posada en marxa.

2. Arquitectura

El present capítol procura fer èmfasis en l'aspecte hardware de la plataforma, tot explicant les diferents parts que la composen.

Es presenta el microcontrolador emprat, descrivint les funcionalitat utilitzades junt amb la seva capacitat de còmput i les seves característiques més interessants.

A continuació s'exposen els diferents sensors/actuador que revesteixen la plataforma i pels quals es determina el comportament del robot ja sigui, o bé adquirint informació de l'entorn, o bé actuant d'una certa manera. Es procura mostrar el connexionat i descriure el seu funcionament.

2.1. Descripció general

Els sistema té com a component principal del microncontrolador ATmega128. La seva tasca és efectuar els càlculs sol·licitats i fer de mitjancer entre l'usuari i els diferents sensor que disposem.

Sota el control del microcontrolador trobem:

- 2 motors de corrent continua dirigits per sendes senyals de control PWM, modulació per amplada de pols.
- Infrarojos laterals amb sortida analògica per a la detecció d'obstacles laterals.
- Infrarojos frontals amb sortida digital per a la detecció d'obstacles frontals.
- Placa amb 4 fotoreflectors amb funció de seguidors de línia.
- 2 LDR per el control de la lluminositat.
- Piezzobuzzer per emetre só a una determinada freqüència .
- Micròfon per a la captació de só.
- 6 leds lluminosos.
- Comunicació sèrie RS232. Wired/Bluetooth.

La **Figura 1** mostra l'esquema general de la plataforma UABot. La **Figura 2** i la **Figura 3** mostren la disposició dels diferents components sobre la placa.

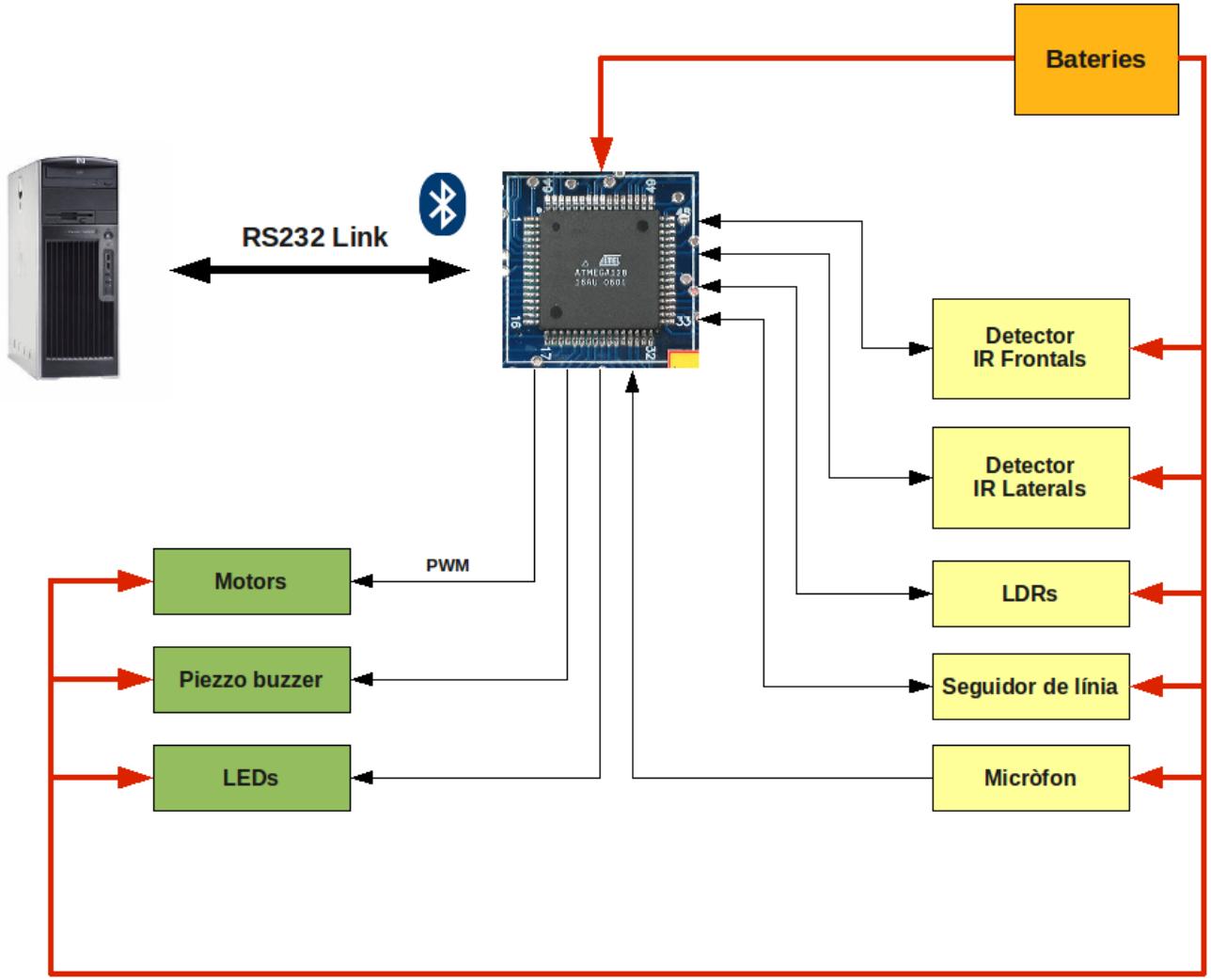


Figura 1: Diagrama general de l'arquitectura UABot

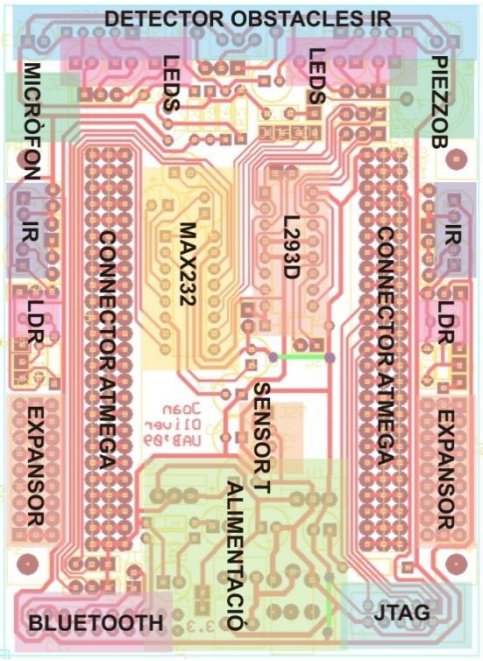


Figura 2: Layout PCB amb la disposició dels diferents components

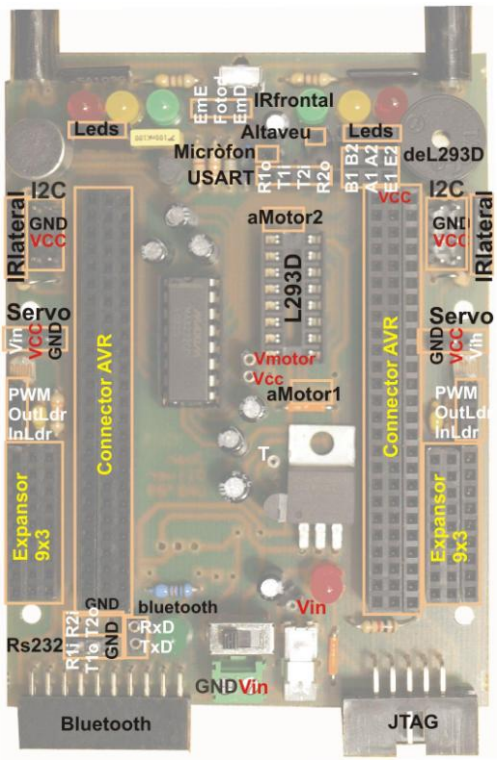


Figura 3: Disposició dels diferents components

2.2. MCU Atmega128

L'integrat que governa el robot és el microcontrolador ATmega128 de la casa Atmel. És un processador de 8-bits RISC (conjunt d'instruccions reduït, conjunt de registres ampli) amb arquitectura Harvard (espai d'instruccions i dades físicament separats), capaç d'arribar a un throughtput de 16 MIPS a una freqüència de treball de 16MHz (una instrucció per cicle de rellotge).

Addicionalment incorpora una esquema d'interrupcions vectoritzades amb prioritat que permet dur una gestió dels perifèrics dirigida per interrupcions.

Disposa de 128Kbytes de memòria Flash programable on s'emmagatzema el programa del MCU, 4 Kbytes de memòria SRAM per dades i 4Kbytes més de memòria EEPROM per emmagatzemar constants o dades persistent.

La gama de perifèrics que caracteritza el microcontrolador és variada:

- 53 Ports programables d'entrada i sortida (pins multiplexats amb altres funcions).
- 2 USARTS d'interfície per la comunicació sèrie RS232.
- 2 Timers de 16 bits i 2 timers més de 8 bits, amb possibilitat de generar fins a 8 senyals PWM.
- Unitat ADC de 8 canals amb conversió de fins a 10 bits.
- Interfície Two-Wire Serial i SPI.
- Watchdog Timer
- ...

El microcontrolador s'alimenta a una tensió de 5V. Suplementàriament, disposa de 8 modes d'estalvi energètic dels quals és pot passar de la suspensió del consum de certes parts del sistema a la hibernació total de microcontrolador.

La **Figura 4** mostra l'organització interna de la CPU del microcontrolador ATmega128

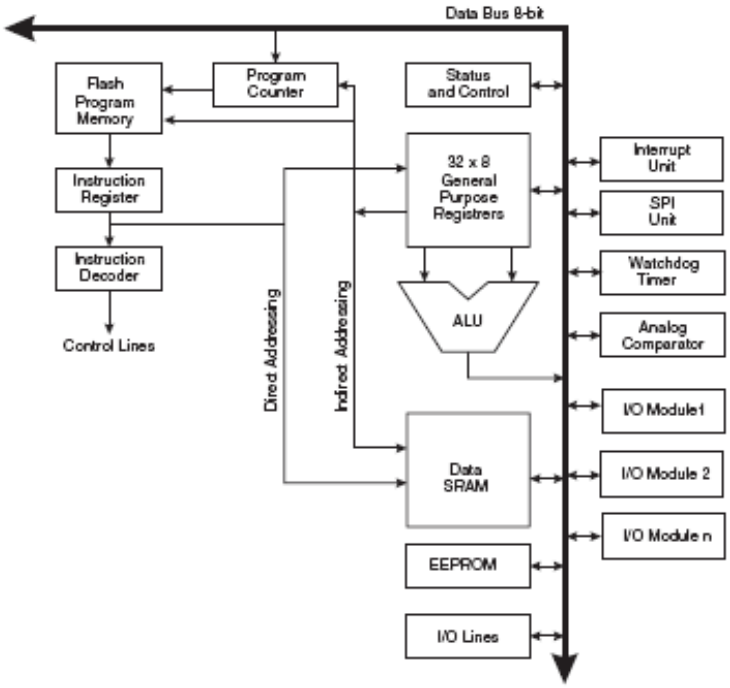


Figura 4: Organització de la CPU del MCU ATmega128

2.3. Detector d'obstacles frontal

El circuit detector d'obstacle frontals està format per dos emissors IR (ref. SFH415) i un detector IR (ref. GP1UX301QS) amb filtre a 38KHz, capaç de detectar obstacles a partir dels 35 cm de distància. A la sortida s'hi poden acoblar dues capacitats per reduir el soroll. La **Figura 5** mostra l'esquema elèctric del sistema detector d'obstacles:

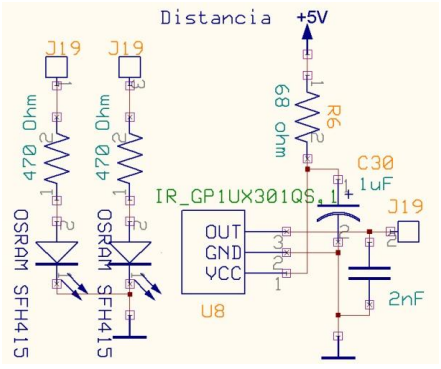


Figura 5: Esquema del sistema detector

El mecanisme de funcionament és el següent:

1. S'envien un seguit de polsos de període 26µs (38.4kHz) a través de l'emissor esquerre. El nombre de polsos pot variar de 15 a 25. En el nostre cas enviem 20 polsos.
2. Esperem un temps d'esperar per a la recepció. Aquest temps sempre ha de ser superior als 0.5ms. En el nostre cas, esperem un temps prudencial de 850µs.
3. Reproduïm el pas 1 però ara per l'emissor dret.
4. Finalment de nou, esperem el temps prudencial del punt 2 per a la recepció del senyal.

La senyal de sortida del receptor és una senyal digital que es manté constant a 1 si no es detecta cap obstacle. Per altra banda, si és present un obstacle detectable a una certa distància, la senyal baixa a 0 durant un temps proporcional a la distància de detecció. A una distància menor, abans cau la senyal de sortida del receptor. La **Figura 6** mostra una captura de l'oscil·loscopi:

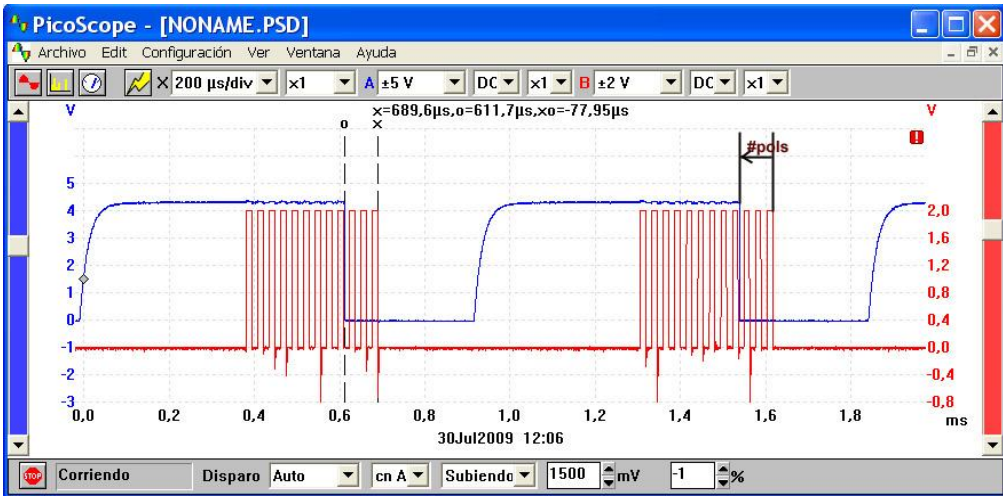


Figura 6: En vermell, seqüència de polsos (12 polsos) més espera. En blau, la resposta

Podem definir tres estats característics en la sortida del detector d'obstacles:

- Estat de no obstacle. La sortida és constant, sempre a 5V.
- Estat d'obstacle a distància. Proporciona un pols d'amplada variable en funció de la distància de detecció.
- Estat d'obstacle a màxima proximitat. Un pols de període constant a la sortida.

Existeix un **llindar de proximitat** on, a distàncies inferiors a aquest llindar, el receptor ja no és capaç de proporcionar una senyal proporcional a la distància de l'obstacle, entregant a la sortida un pols amb període constant.

Cal esmentar unes matisacions. Totes les mostres obtingudes per dispositius infrarojos estan supeditades a la lluminositat que incideix en el receptor. Per tant, la mostra obtinguda per el receptor frontal estarà condicionada per la llum ambient, sent possible que, per un obstacle a la mateixa distància però variant la lluminositat, obtinguem una senyal de sortida diferent.

També és important la naturalesa del material que suposa l'obstacle. Objectes de color blanc lluent reflecteixen molt millor els senyals IR que no pas objectes de color fosc mat, ja que aquests últims absorbeixen la radiació infraroja, sent necessari una distància més propera per a ser detectats.

Per altra banda, en el cas concret del receptor IR frontal, aquest no és immune a un cert grau d'error. Durant la detecció d'un obstacle a distància, la senyal de sortida pot oscil·lar en un rang de períodes diferents. A mesura que ens aproximem a l'obstacle, la sortida convergeix a oscil·lar menys. A màxima proximitat, la sortida s'estabilitza, sent rarament oscil·lant.

Distància	**Valor** (mean l., mean r.)
< 30 cm	126 – 132
30 cm	130 – 140
40 cm	145 – 145
50 cm	170 – 157,
60 cm	196 – 200,
70 cm	237 – 192,
80 cm	255 – 253,
> 80 cm	255 – 255,

Taula 1: Relació distancia - valor

2.4. Detector d'obstacles laterals

La detecció d'obstacles laterals es fa amb dos parelles d'emissor/receptor d'infrarojos (ref. emissor TSF5210, ref. receptor BFV10NF), una parella situada a cada lateral del robot.

El funcionament d'aquest detector lateral és més simple que el frontal. Senzillament el receptor respon a freqüències IR que s'emeten prop de la seva posició. Només cal alimentar el diode emissor i observar la resposta del receptor.

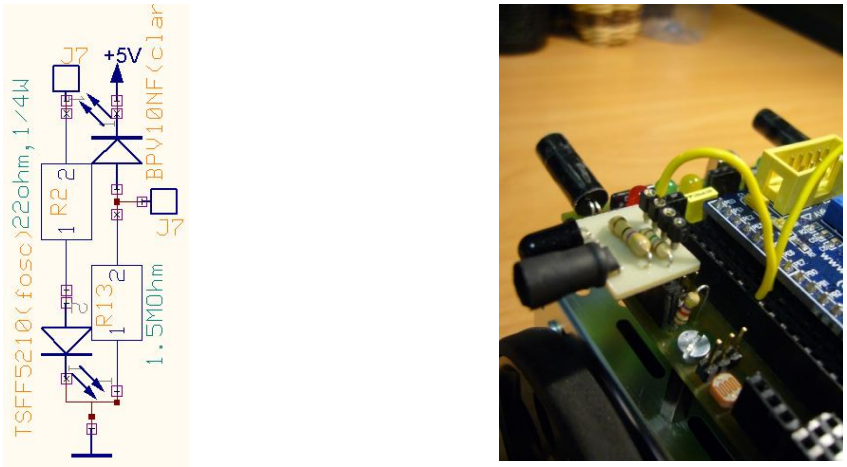


Figura 7: Esquema elèctric i imatge d'un dels dos detectors d'obstacles laterals

La sortida del receptor és una senyal analògica que varia de nivell en funció de la proximitat de l'objecte. A màxima proximitat, obtenim un nivell de tensió màxim a la sortida. A mesura que ens allunyem, la tensió de sortida descendirà de manera proporcional al distanciament. Finalment, a distàncies superiors als 35cm, la senyal de sortida romandrà constant, propera a un valor de tensió baix.

És important fer esment que al igual que el sistema de detecció frontal, la lluminositat ambient i el material de l'obstacle són factors que condicionen l'obtenció de la mostra. Si bé, és important remarcar que en el cas dels detectors laterals, la sortida és estable a qualsevol distància de l'obstacle (existeix una petita oscil·lació inherent en procés d'obtenció de la mostra).

La Taula 2 recull diferents mostres obtingudes en funció de la seva distància amb lluminositat ambient sempre constant de 155 (valor obtingut amb les fotodetector)

Distància	Valor
< 4 cm	255
5 cm	215
6 cm	165
7 cm	132
8 cm	111
9 cm	95
10 cm	85
11 cm	76
12 cm	70
13 cm	64
14 cm	60
15 cm	57
16 cm	54
17 cm	50
18 cm	47
19 cm	45 (No obstacle)

Taula 2: Relació distancia – valor amb valor de llum de 155.

2.5. Fotodetectors

El robot té dues cèl·lules fotoconductores (LDR, ref. VT935G) disposades una a cada lateral. El seu ús permeten obtenir una dada numèrica de les condicions lumíniques de l'ambient.

El circuit elèctric realitzat es basa en un divisor de voltatge / circuit RC que permet detectar canvis sobtats de llum. La construcció del circuit RC té com a màxim una constant $\tau = 155\mu s$.

L'obtenció d'una mostra d'un dels fotodetectors es pot realitzar alimentant la respectiva entrada i observant la sortida. Un pols de durada 0.5ms a l'entrada és suficient per obtenir una sortida a un nivell satisfactori. Posteriorment cal convertir la sortida analògica a digital emprant l'unitat ADC del microcontrolador.

La **Figura 8** apareix l'esquema del sistema fotodetector així com una imatge de una de les fotocel·lules.

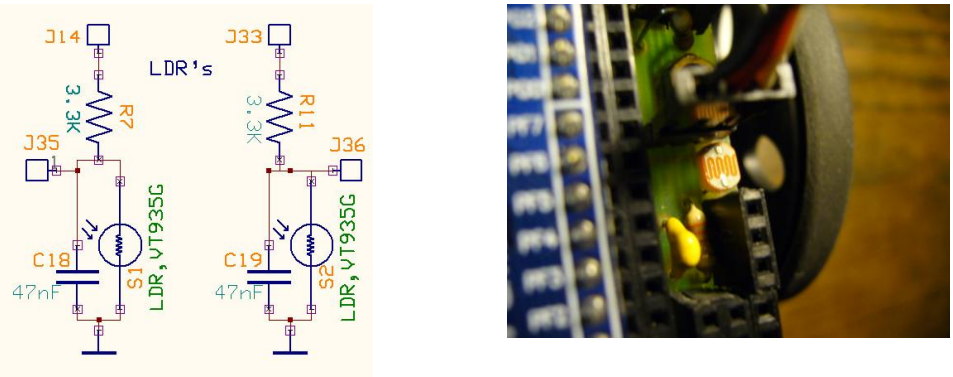


Figura 8: Esquema elèctric del sistema fotodetector

2.6. Seguidor de línia

El dispositiu seguidor de línia és una placa formada per 4 fotoreflectors (ref. APDS9104) alineats d'una determinada manera per a la correcte detecció d'una línia. La disposició lineal dels fotoreflectors permet detectar una transició de color de blanc a negre o viceversa.

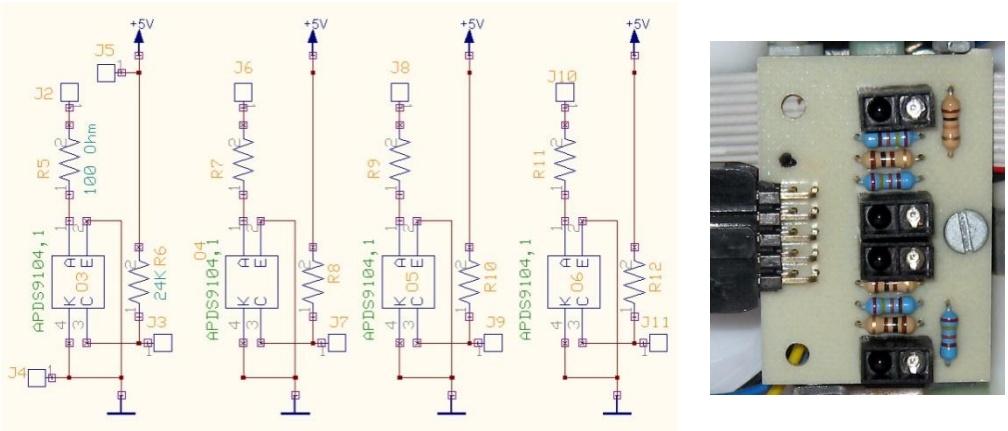


Figura 9: Esquema elèctric i disposició dels fotoreflectors

El funcionament dels fotoreflectors és simple: després de l'aplicació d'un pols en un dels diodes IR emissors, s'espera una resposta en el corresponent fototransistor.

El color negre absorbeix la radiació infraroja, motiu pel qual, si l'emissió incideix sobre material obscur, el fototransistor no captarà cap emissió. Contràriament, si la senyal

incideix sobre material blanc lluent, l'emissió infraroja serà reflectida, captant-se en el fototransistor.

2.7. Piezzo buzzer i micròfon

El piezzo buzzer és un altaveu petit que respon a impulsos de freqüències audibles. El seu funcionament és molt simple: utilitzant la línia d'entrada del dispositiu podem transmetre una senyal digital polsant a una freqüència coneguda. La freqüència de vibració de l'altaveu piezzo buzzer serà la mateixa que la subministrada a l'entrada.

La placa incorpora un micròfon electret amb circuiteria de preamplificació que permet mostrejar so. A la sortida del preamplificador és possible posar-hi un condensador de desacoblament de 100nF. El funcionament és molt simple: senzillament la sortida del preamplificador s'envia a qualsevol dels pins analògics del microcontrolador per convertir el seu valor en digital.

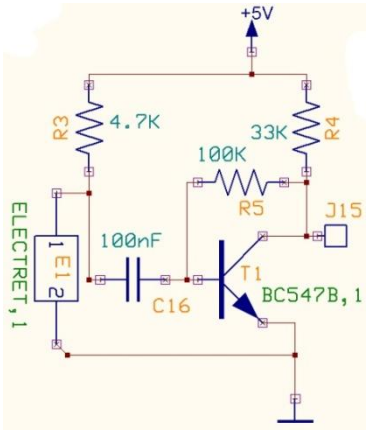


Figura 10: Micròfon i circuiteria de preamplificació

2.8. Comunicació RS232

La comunicació sèrie RS232 és una de les més comunes per a la transmissió de dades. No obstant, el protocol RS232 té definits uns nivells de tensió diferents als nivells TTL habitualment utilitzats. Per aquest motiu, la placa incorpora l'integrat MAX232 que efectua la translació de MCU a RS232 i de RS232 a MCU.

El dispositiu MAX232 permet fins a dues translacions simultànies de dues comunicacions sèrie RS232 diferents.

Així doncs, amb l'adaptació dels nivells TTL, podem utilitzar, indistintament, les dues USARTs que disposem per a la comunicació sèrie.

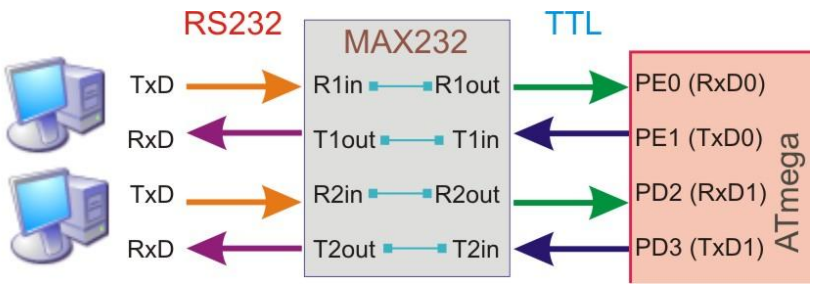


Figura 11: MAX232. Translació de TTL a RS232 i viceversa

2.9. Bluetooth

La comunicació pot ser mitjançant un cable sèrie connectat a un port de comunicació de l'estació host. No obstant, això limita molt la mobilitat del robot. Precisament per aquest motiu, la placa incorpora un connector d'expansió per poder connectar un mòdul Bluetooth per a la comunicació sense fils.

Aquest mòdul Bluetooth emula la comunicació sèrie, enviant les dades seguint el l'estandard de comunicació Bluetooth. De cara al microcontrolador, la comunicació es totalment transparent, no sent necessari haver de configurar cap opció addicional. De la banda de l'estació host, es requereix un dispositiu usb detector de bluetooth que emuli un port sèrie (port sèrie virtual).

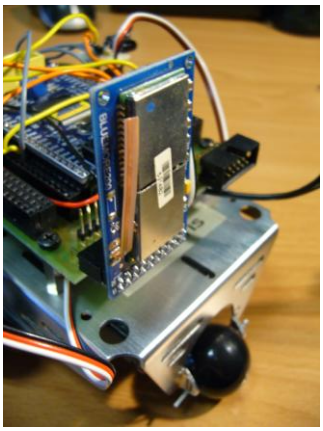


Figura 12: Mòdul bluetooth

2.10. Motors

La plataforma UABot està equipada amb dos servomotors de corrent continua. Ambdós són independents l'un de l'altre, corresponent un, a la roda de l'esquerra i l'altre, a la roda de la dreta. La seva independència permet controlar el sentit i la velocitat de gir de cada roda de forma individual.

El control d'un motor s'efectua a través d'una senyal de control d'amplada de pols PWM que determina el sentit de gir i la velocitat.

El període de la senyal PWM és sempre constant als 20ms (50 Hz). A l'inicií del període, la senyal s'inicialitza a nivell alt. Després d'un cert temps determinat, cau a zero. L'amplada del semiperíode alt determina la configuració de gir del servomotor.

Un semiperíode d'amplada 1.5ms manté el motor en repòs. Semiperíode d'amplada superior a 1.5ms gira vers un sentit. Semiperíode inferior a 1.5ms gira vers el sentit contrari.

Respecte la velocitat, un increment lineal en el període representa un increment lineal de la velocitat de gir. Òbviament, existeix un llindar on superat aquest, la velocitat de gir és manté constant encara que continuem incrementant/decrementant l'amplada del pols. El llindar superior el trobem entorn un semiperíode d'amplada 1.63ms; el llindar inferior ronda un semiperíode d'amplada 1.37ms. Més enllà d'aquest valors, la velocitat romandrà constant.

2.11. Leds

La placa disposa de 6 led lluminosos, tres de color vermell i 3 de color verd. Cada led esta connectat a una resistència de polarització per a reduir la intensitat que travessa per cada led, assegurant la seva vida útil.

2.12. Bateries

L'alimentació de la plataforma UABot s'aconsegueix amb quatre bateries convencionals AA de 1.5V connectades en sèrie.

La tensió d'alimentació és manté estable a 4.9V gràcies a la incorporació d'un regulador de tensió que procura una tensió constant d'alimentació.

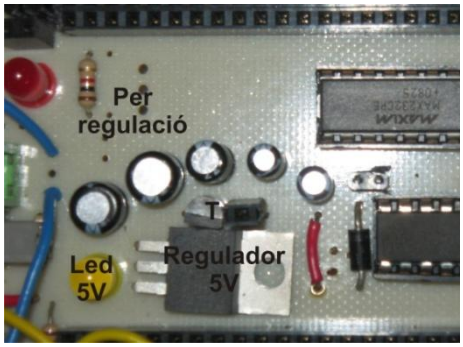


Figura 13: Regulador de tensió

2.13. Altres components

La placa on es munten els diferents components que hem anat enunciant, incorpora de manera addicional els següents components:

- Integrat LD293 per el control de dos motors de corrent continua.
- Sensor de temperatura.

Tot i ser presents i completament funcionals, la plataforma UABot no els fa servir.

3. Capa Software

El present capítol procura explicar el concepte teòric de sistema operatiu en temps real. Introdueix de forma breu el sistema operatiu FreeRTOS, incorporat en la plataforma UABot.

Per acabar, s'explica amb detall les diferents parts que composen l'esquema software del microcontrolador: desglossament en tasques i drivers diversos.

El present capítol és fonamentalment teòric. El lector pot prescindir dels conceptes que s'exposen a continuació. Tot i així, és recomanable que el lector tingui una mínima idea de com funciona la plataforma a nivell software.

3.1. Sistema Operatiu en Temps Real (RTOS)

Hi ha molta i variada literatura per a definir amb exactitud que és un sistema operatiu. Totes les definicions són sinònimes però no obstant, sovint trobem punts concrets que difereixen. És difícil sintetitzar un concepte tant genèric con pot ser un sistema operatiu. Un sistema operatiu pot ser des de un petit programa que gestiona un sistema encastat, com un enorme sistema que controla tota una xarxa de computadors interconnectats. És inevitable divergir en alguns aspectes de la definició quan una mateixa idea és aplicable en un ventall tant ampli i heterogeni. Simplement, proporcionar una definició exacte és impossible. No obstant, sí podem proporcionar un definició de sistema operatiu fitada al nostre context, els sistemes encastats.

Donada la nostra situació, un sistema operatiu té dues funcions principals: gestionar els recursos dels que es disposa i dur una planificació dels processos/tasques a executar. Així doncs, una possible definició seria la següent: el sistema operatiu és un programa pare, sempre resident en memòria, que s'executa cíclicament cada cert període de temps per atorgar o revocar recursos i decidir qui o que pot emprar els recursos computacionals de la CPU.

La idea darrere un sistema operatiu és la de posseir una eina software que donades unes condicions o paràmetres de configuració, faci una execució eficient intentant treure el màxim rendiment en base a aquests prerequisits inicials.

Podem trobar tres punts comuns a tots els sistemes operatius:

- Control i gestió de l'execució de la CPU
- Escalabilitat
- Màxim rendiment respectant les restriccions del sistema.

Considerem ara els **sistemes operatius en temps real**. A més d'incloure els punts anteriors, han de complir un requeriment temporal. Un RTOS ha de ser capaç de, produïda una eventualitat, proporcionar una resposta dins un espai de temps límit. Així doncs, un RTOS és un sistema determinista el qual assegura una resposta dins unes restriccions temporals.

Aquest requeriment temporal significa que, a diferencia d'un sistema convencional on s'intenta maximitzar rendiment promig de la CPU, un sistema operatiu en temps real sovint compromet l'eficiència per afavorir la demanda d'una resposta. *No és una màxima dels sistemes en temps real aconseguir un rendiment elevat. Per contra, s'exigeix un temps de resposta fitat.*

3.2. FreeRTOS

El sistema operatiu FreeRTOS és un projecte open source amb llicència GNU. FreeRTOS proporciona el *kernel* d'un sistema operatiu amb temps real (*Real Time Operative System*, **RTOS**) per a dispositius *embedded* junt amb una API que implementa tota una sèrie de recursos per aconseguir sincronització, exclusió mútua i comunicació entre tasques (*Inter-Task Communication*, **ITC**).

El seu disseny està pensat per ser portable i altament escalable. Quasi la totalitat del seu codi està escrit en C, permeten tenir un codi transversal a l'arquitectura. Exemple d'aquesta portabilitat és l'actual suport per a 23 arquitectures diferents.

Part de la feina del *kernel* del FreeRTOS és planificar tasques. Una tasca és una instància d'un programa en curs. Associada a una tasca tenim informació de l'estat de la CPU així com d'altres recursos. Per tant, les tasques són entitats que s'executen en la CPU per ordre del *kernel* del sistema.

Normalment, els dispositius embedded per als que FreeRTOS està dirigit, són sistemes

petits amb un sol nucli de processament. Aquesta limitació implica que només una tasca pot estar resident en la CPU, fent ús del seu còmput, trobant-se la resta en un estat d'espera. En conseqüència, parlarem d'**execució concurrent**, en el sentit de **multiprogramació**, quan diverses tasques competeixen per l'execució de la CPU però només una s'executa. La compartició del temps de còmput entre tasques permet simular un comportament d'execució pseudoparal·lel. Aquesta forma d'execució implementa un paral·lelisme fictici amb la intensió de fer un ús extensiu de la única CPU que disposem.

Per altra banda, existeix el paral·lelisme real, que no tractarem per estar fora del àmbit d'aquest document però que sí és d'interès esmentar. El paral·lelisme real es produeix quan dues o més tasques/processos s'executen a la vegada en un o varis nuclis de computació. FreeRTOS no seria la millor opció per a la gestió d'aquests dispositiu, existint d'altres sistemes operatius preparats per explotar dites característiques.

Cal entendre les tasques com entitats que requereixen temps de còmput. Aquest temps de còmput és gestionat mitjançant el *kernel* del sistema operatiu, com ja s'ha ben dit. De fet, el mateix sistema operatiu pot entendres com una tasca de màxima prioritat que s'executa cíclicament cada cert període de temps per assignar *time-slice* (temps de processament) a una certa tasca concreta.

El *kernel* de FreeRTOS implementa un planificador **preemtiu**. Una política preemtiva pot suspendre la tasca en curs per assignar temps de CPU a una altra tasca amb més prioritat. Aquesta política de planificació és àmpliament acceptada per una gran majoria de sistemes operatius, no necessariament de temps real.

Els principals beneficis de la planificació preemtiva són:

- Ús més racional de la CPU.
- Augment eficient de l'ús de còmput de la CPU.
- Implementació de la multiprogramació.

Per contra, té dos petits desavantatge, ambdós relacionats. La planificació preemtiva requereix una gestió periòdica per part del sistema operatiu. Això comporta que, una tasca en execució pugui ser interrompuda per el requeriment de CPU del kernel.

Un temps de *quantum* (*time slice* o porció de temps que s'assigna) massa petit pot derivar en una situació de rendiment pobre, ocasionada per la continua irrupció del sistema operatiu. La conseqüència directe es que s'inverteix més temps de computació en la gestió del sistema, sent el kernel qui monopolitza la CPU, que no pas efectuant càlculs productius.

Per altra banda, un *quantum* massa llarg pot degenerar en situacions de CPU ociosa, existint tasques suspeses en espera. Per tant, és important mantenir un *quantum* adequat que equilibri el rendiment real de la CPU.

Relacionat amb la gestió del *kernel*, a cada interrupció d'aquest es produeix un canvi de context o *context switching*. El canvi de context és una operació destinada a preservar la integritat de les dades i el flux d'execució d'una tasca. El pilar de la concurrència, multiprogramació de la CPU, s'aconsegueix gràcies a l'operació de canvi de context. No obstant els grans beneficis que comporta una operació de *context switch*, aquesta implica un petit *overhead* en la CPU perdut en la gestió del manteniment de les tasques.

Una operació de canvi de context ineficient o amb freqüència molt alta pot restar prestacions de velocitat i rendiment.

Part d'aquest problemàtica recau en el disseny del sistema operatiu, havent de confiar amb el correcte disseny del *kernel* del FreeRTOS.

L'altra part recau sobre el programador que decideix adoptar FreeRTOS en el seu sistema *embedded*. FreeRTOS és altament configurable, permeten modificar valors com ara el *quantum* o quantitat de memòria reservada per tasca. Modificar aquests valors a consciència pot millorar el rendiment però, inversament, introduir valors de configuració inadequats pot manifestar-se en un rendiment pèssim.

3.3. API del FreeRTOS

El projecte FreeRTOS proporciona a més d'un *kernel* RTOS per a dispositius *embedded*, un complet conjunt de funcions escrites en C per cobrir les diferents necessitats: des de la creació de tasques fins a mecanismes de comunicació i exclusió mútua. De fet,

l'API que acompanya al *kernel* són una sèrie de funcions que interactuen amb el nucli del sistema operatiu.

Enunciem els diferents mecanismes disponibles:

- Creació i destrucció de tasques.
- Creació de co-rutines (tasques lleugeres)
- Control de les tasques.
- Implementació de cues amb accés atòmic per a la intercomunicació de tasques.
- Semàfors binaris i semàfors n-aris per a la sincronització.
- Mutexs per a l'exclusió mútua.
- Control directe del *kernel*.

La disposició d'aquests mecanismes ens permeten augmentar el grau de multiprogramació de la CPU ja que ajuden al kernel a fer una planificació més intel·ligent.

3.4. Esquema general del sistema UABot

La idea d'incorporar un sistema operatiu per al dispositiu microcontrolador ens brinda l'oportunitat de dissenyar un sistema encarat a execució de tasques.

Cada tasca té un codi propi d'execució, representant un mòdul independent a la resta de les tasques. Llavors, el desenvolupament és relativament net, podent tractar cada tasca com una funció, sent el cos de la funció el flux d'execució de cada tasca concreta.

Actualment, la plataforma UABot implementa tres tasques amb diferents nivells de prioritat (veure **Figura 14**).

Cada una de les tasques s'encarrega d'una parcel·la de gestió concreta, fragmentant el codi de control del robot en parts independents però interconnectades per el pas d'informació mútua.

Enumerem les tasques que s'implementen:

- Tasca Comunicació
- Tasca Control dels Motors

- Tasca Control dels Sensors

3.4.1. Tasca Comunicació

S'encarrega de capturar les comandes enviades per l'usuari i expedir l'ordre a una de les altres dues tasques.

Després de la tasca que representa el *kernel* del sistema operatiu, la tasca de comunicació és la que té més prioritat. Per la seva naturalesa, exigeix un tractament ràpid de les dades, ja que de ella, en depenen les altres dues tasques

Algorisme d'acció de la Tasca Comunicació és el següent:

1. Rep una comanda de la cua de la comunicació entrant.
2. Efectua una petit processat de la dada rebuda per identificar comanda i tasca.
3. Envia la comanda (l'ordre de l'usuari) a la tasca escaient a través d'una cua de comunicació dedicada per a cada tasca.
4. Es suspèn fins a l'espera d'una nova comanda per processar.

3.4.2. Tasca Motors

Tasca que governa el control dels motors. Rep la comanda, prèviament tractada en la Tasca Comunicació, i configura els motors en base els arguments subministrats per l'usuari.

La seva prioritat és moderada, sent més prioritària que la Tasca Sensors però menys que la Tasca Comunicació.

L'usuari envia junt amb l'opcode de la comanda motors, una parell de dades numèriques. Aquestes dades són les emprades per a la configuració de sendes rodes.

Algorisme d'acció de la Tasca Motors:

1. Tasca Motors rep l'ordre de configurar motors. Junt amb l'ordre, l'acompanyen les dades de configuració.
2. Tasca Motors efectua un processat previ i modifica els registres de configuració de motors pertinents.
3. Tasca Motors es suspèn fins a nova ordre.

3.4.3. Tasca Sensors

És la tasca amb menys prioritat de les tres esmentades. Es responsabilitza de totes les operacions relacionades amb els sensors.

De manera similar a la Tasca Motors, rep l'ordre enviada per l'usuari i filtrada per Tasca Comunicació. L'ordre identifica de forma unívoca un sensor. S'obté la mostra del sensor requerit i s'envia a la cua de sortida, cap a l'usuari.

Algorisme d'acció de la Tasca Sensors:

1. Tasca Sensors rep ordre relacionada amb una operació de sensors.
2. Tasca Sensors determina quina operació de sensors és sol·licitada.
3. Tasca Sensors efectua el procés d'obtenció de la mostra sol·licitada.
4. Tasca Sensors envia la mostra al usuari a través de la cua de dades de sortida.
5. Tasca Sensors es suspèn fins a nova ordre.

3.5. Esquema de comunicació

Les tres tasques introduïdes anteriorment són independents l'una de les altres. Cadascuna viu en el seu espai de memòria reservat a tal fi, intentant preservar, fins a certa mesura, aquesta independència també en les dades que es manipulen (existeixen unes poques dades globals accessibles a totes les tasques).

Tot i les avantatges d'aquest aïllament entre tasques, en ocasions és necessari l'ús de mecanismes de sincronització i comunicació entre tasques. Una de les formes més usuals per a la sincronització i pas de missatges és a través de cues.

Les cues són eines software emprades per a emmagatzemament i sincronització. L'API de FreeRTOS implementa l'ús de cues genèriques (qualsevol tipus de dada), homogènies (totes les dades de la cua són del mateix tipus) i amb accés atòmic (mecanisme de pas de dades segur).

En el cas de la plataforma UABot, s'implementen les cues en un sentit unidireccional similars a les *pipes* que s'utilitzen en entorns Unix. No és una limitació de la API de FreeRTOS, si bé, és una restricció de disseny adoptada (segueix un plantejament productor/consumidor).

Per interconnectar les tasques hem emprat les següents cues.

- Cua de dades entrants.
- Cua de dades sortints.
- Cua de Motors.
- Cua de Sensors.

3.5.1. Cua de dades entrants

Cua per on passen totes les dades d'entrada, independentment de la naturalesa del destinatari. La cua està orientada a bytes.

Aquesta cua és alimentada per el servei d'interrupció de la USART (ISR USART). Quan la USART obté un byte de l'enllaç de comunicació, dispara una interrupció. El cos de la rutina s'encarrega d'agafar la dada i inserir-la a la cua.

A l'altre extrem de la cua, trobem com a consumidor la Tasca Comunicació, esperant dades per processar i distribuir-les a les altres dues tasques.

3.5.2. Cua de dades sortints

Cua amb funció inversa a l'anterior. Canalitzant totes les dades de sortida de les tres tasques (productors) per que siguin rebudes per una tasca d'enviament de dades (consumidor) encarregada exclusivament per a tal efecte.

La tasca d'enviament de dades, no esmentada anteriorment, és una tasca molt pròxima al servei d'interrupció de la USART. Hem considerat tractar la tasca d'enviament de dades i el ISR USART com tota una unitat.

Per ser una cua amb tres productors comuns orientada a bytes (atomicitat només en la inclusió d'un byte), s'ha implementat un esquema d'exclusió mútua que garanteix l'enviament d'un bloc de bytes íntegre.

3.5.3. Cua de Motors

Cua alimentada per la Tasca de Comunicació. Els elements d'aquesta cua són paquets de configuració dels motors.

Escoltant a l'extrem oposat trobem la Tasca Motors preparada per consumir paquets de la cua i configurar els motors.

3.5.4. Cua de Sensors

Cua amb funció anàloga a la Cua Motors. La Tasca de Comunicació s'encarrega d'inserir peticions de consulta de sensors a la Cua de Sensors.

A l'altra banda, Tasca Sensors està a l'espera de processar ordres.

3.6. Drivers

A continuació exposem els diferents algorismes emprats per a l'ús dels recursos que disposa la plataforma.

Els algorismes que a continuació s'exposen són meres descripcions formals. La seva implementació pot variar en alguns punts específics.

3.6.1. Motors

El control dels motors es gestiona mitjançant dues senyals PWM amb període 20 ms, una per motor. Les senyals són generades amb un dels timers (**TIMER1**) que disposa el microcontrolador ATmega128.

```
proc Motors():
    Mpkg_t m;

    Configurar_Motors(DEFAULT_LEFT, DEFAULT_RIGHT);

    mentre( m = Cua_Motors() )    //Bucle infinit
      Configurar_Motors(m.left, m.right)
    fi mentre
fi
```

Observem que la gestió dels motors radica simplement en iniciar una configuració per defecte i posteriorment, esperar noves configuracions des de la Cua Motors.

`Configurar_Motors()` és un procediment molt simple que accedeix als registres del timer per modificar la sortida de la senyal PWM (canviar el semiperíode alt).

Per cada recepció, es reconfiguren els motors amb la nova configuració subministrada `m` i torna a l'espera de noves recepcions (bucle infinit).

3.6.2. Sensors

L'algorisme seguit per a la gestió dels sensors és un procediment que està constantment escoltant la cua de sensors. Quan es rebuda una operació, decideix quin procés de control cridar.

```
proc Sensor():

    Spkg_t s;
    Proc_t vec[] = {
      get_irf,
      get_irl,
      get_ldr,
      get_lfw,
      …
    };

    Configurar_Sensors();

    mentre( s = Cua_Sensors() )  //Bucle infinit
    {
      Op_t op;

      op = parseSensor(s);
      vec[op](s.args);
    }

fi
```

Utilitzem un vector de rutines per decidir quina crida efectuar. Amb aquesta tècnica obtenim un temps d'accés constant i no atorga prioritat sobre cap cas.

Fotoresistors (LDR)

Per la captació del nivell lluminositat es requereix una entrada digital que, a nivell alt, alimenti la fotoresistència. La sortida del LDR és una sortida analògica que ha de ser convertida emprant el conversor ADC.

Tenim dues cèl·lules fotoresistives, una a cada lateral. L'ordre de captació d'una mostra significa l'obtenció del valor de lluminositat del LDR esquerre i del LDR dret.

La nomenclatura de les senyals és la següent:

- **LDR_IN:** comuna a ambdós LDRs. Alimentem alhora els dos LDRs amb una mateixa senyal.
- **LDR_OUT_LEFT:** Sortida analògica del LDR esquerre.

- **LDR_OUT_RIGHT:** Sortida analògica del LDR dret.

L'algorisme seguit per l'obtenció de les mostres:

```
proc get_ldr(Byte n(0..1)):

    initADC()
    set(LDR_IN);
    wait(1ms);

    n[0] = getADC(LDR_OUT_LEFT);
    n[1] = getADC(LDR_OUT_RIGHT);

    clr(LDR_IN);
    endADC();

fi
```

És necessari l'espera d'un temps de càrrega doncs, com hem vist, les fotoresistències formen junt amb un condensador, un circuit RC.

Seguidor de línia

La implementació del software seguidor de línia requereix de quatre entrades digitals per excitar els quatre fotoreflectors. Conseqüentment, són necessàries quatre sortides digitals, una per fotoreflector, que representin l'estat del seguidor de línia.

Les senyals implicades són:

- **LFW_LL_IN:** Entrada fotoreflector esquerre.
- **LFW_LL_OUT:** Sortida fotoreflector esquerre.
- **LFW_LC_IN:** Entrada fotoreflector centre-esquerre.
- **LFW_LC_OUT:** Sortida fotoreflector centre-esquerre.
- **LFW_RC_IN:** Entrada fotoreflector centre-dret.
- **LFW_RC_OUT:** Sortida fotoreflector centre-dret.
- **LFW_RR_IN:** Entrada fotoreflector dret.
- **LFW_RR_OUT:** Sortida fotoreflector dret.

L'algorisme que s'implementa per a l'obtenció de l'estat del seguidor de línia és el següent:

```
func Byte get_lfw():

    integer i, r = 0;
    Pin_t fr(0..3) = {
       {LFW_LC_IN, LFW_LC_OUT, 2},
       {LFW_RR_IN, LFW_RR_OUT, 0},
       {LFW_LL_IN, LFW_LL_OUT, 3},
       {LFW_RC_IN, LFW_RC_OUT, 1},
    };

    for(i = 0; i < 4; i++)
       set(fr[i][0]);
       wait(1ms);
       r |= (get(fr[i][1]) << fr[i][2]);
       clr(fr[i][0]);
    fi for;

    return r;
fi
```

És d'interès veure com es realitza la consulta als diferents fotoreflectors. La senyal emesa per una dels emissors es captada per el respectiu receptor però també és una font de soroll per a la resta de receptors. Per tant, és important trobar una disposició de consulta ràpida i que a més, minimitzi el soroll. Aquesta configuració està representada en el vector `fr`. La seqüència de consulta és la següent:

FR centre-esquerre → FR dret → FR esquerre → FR centre-dret

Contemplem també que es requereix un temps d'espera 1ms per obtenir una resposta estable a la sortida del respectiu receptor.

Detector d'obstacles frontals (IR Frontals)

El sistema detector d'obstacles frontals requereix de dues entrades i una sortida digitals:

- **IRF_L_IN:** Entrada emissor esquerre.
- **IRF_R_IN:** Entrada emissor dret.
- **IRF_OUT:** Sortida receptor.

Com ja hem vist, és necessari aplicar a cada entrada una seqüència de polsos determinada per obtenir una senyal de sortida vàlida. Tal patró d'entrada es genera emprant un dels timers disponibles (**TIMER3**).

La sortida del receptor d'infrarojos frontal baixa de nivell en un temps determinat per la proximitat de l'obstacle. S'utilitza el sistema d'interrupcions externes per detectar la baixada de la senyal de sortida.

```
integer *side;

proc get_irf(integer t(0..1)):
    enableInt6();
    initTimer(TIMER3);

    side = &t[0];
    psignal(38KHz, 50%, IRF_L_IN);
    wait(28us * NPULSES);

    psignal_stop();
    wait(WAIT_TIME);

    side = &t[1];
    psignal(38KHz, 50%, IRF_R_IN);
    wait(28us * NPULSES);

    psignal_stop();
    wait(WAIT_TIME);

    endTimer(TIMER3);
    disableInt6();
fi

ISR(INT6):
    *side = getTime();
fi
```

`psignal(38KHz, 50%, IRF_X_IN)` genera una senyal quadrada de freqüència 38kHz i *duty cicle* al 50% a través del pin `IRF_X_IN`.

Comprovem que la descripció de l'algorisme es bastant intuïtiva. Inicialitzem les configuracions pertinents, generem una seqüència de polsos a l'emissor esquerre, esperem un temps, tornem a generar la seqüència però a l'emissor contrari. Finalment, tornem a esperar. En qualsevol dels moments d'espera, es pot disparar la interrupció que captura l'instant de temps on ha caigut el senyal del receptor.

Detector d'obstacles laterals (IR Laterals)

El procés d'obtenció d'una mostra per els detectors laterals és més simple. Els detectors laterals és una parella emissor/receptor d'infrarojos amb una senyal digital d'entrada per governar l'emissor i una senyal analògica de sortida proporcionada per el receptor:

- **IRL_IN:** Entrada digital comuna per ambdós emissors.
- **IRL_L_OUT:** Sortida analògica receptor esquerre.
- **IRL_R_OUT:** Sortida analògica receptor dret.

El fet d'obtenir el valor del detector de l'esquerra i el de la dreta de forma consecutiva, ens permet reutilitzar la mateixa senyal digital d'entrada per estimular ambdós emissors.

L'algorisme seguit és el següent:

```
proc get_irl(Byte n(0..1)):

    initADC()
    set(IRL_IN);

    wait(1ms)
    n[0] = getADC(IRL_OUT_LEFT);
    n[1] = getADC(LDR_OUT_RIGHT);

    clr(LDR_IN);
    endADC();

fi
```

Destacar la necessitat d'un temps d'espera abans d'obtenir les mostres analògiques d'ambdues sortides.

Micròfon

El dispositiu micròfon ens proporciona una sortida analògica en funció del so ambient. Llavors, l'obtenció d'una mostra és tant fàcil com convertir la senyal de sortida analògica del micròfon a digital, emprant l'ADC del microcontrolador.

```
func Byte get_mic():
    Byte sp;

    initADC()
    sp = getADC(MIC_OUT);
    endADC();

    return sp;
fi
```

LEDs

El control dels 6 leds s'efectua mitjançant 6 línies E/S configurades com a sortides del microcontrolador. Cada línia controla un únic led.

```
proc led(Byte set, Byte clr):
    set &= 0x3F;
    clr &= 0x3F;

    PORTIO |= set;
    PORTIO &= not(clr);
fi
```

Observem que el procés és extremadament senzill. Amb els valors de les mascares *set* i *clr* es configuren les sortides que governen els leds.

Piezzo buzzer

La senyal que excita el piezzo buzzer es generada mitjançant un dels timers disponible (**TIMER2**).

```
proc piezzo(Byte cfg, Byte pre):
    TIMER = cfg;
    TIMER_PRE = PRE;
fi
```

La senyal de sortida es configura amb els valors *cfg* i *pre*, proporcionats per l'usuari.

Nota: Tot i estar incloses en l'apartat de Sensors, les rutines dels Leds i Piezzo buzzer s'executen en la tasca motors aprofitant un giny d'optimització.

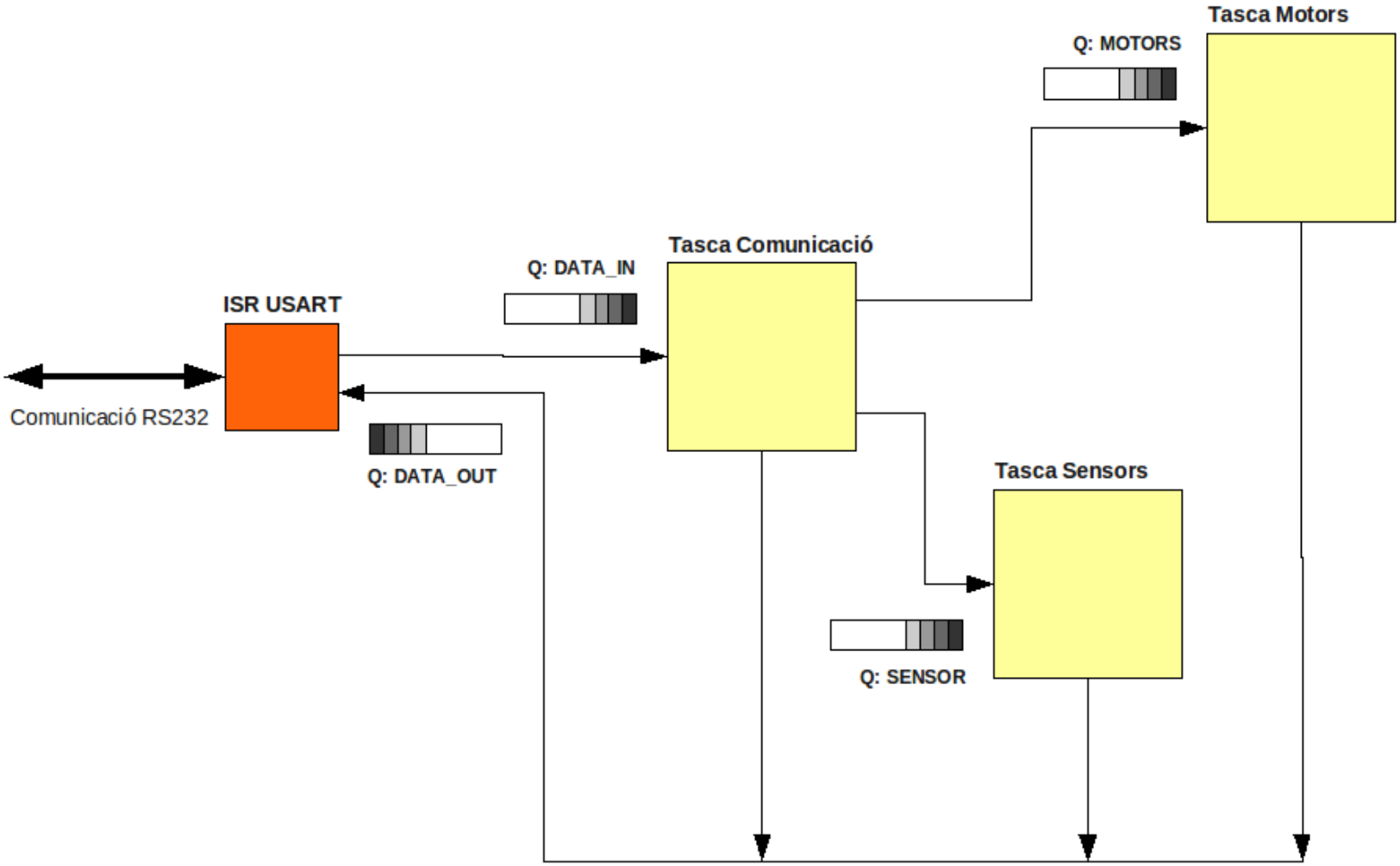


Figura 14: Diagrama de tasques i cues

4. Interfície Host

Aquest capítol final serveix de guia ràpida per a la instal·lació i primeres proves. No és una introducció exhaustiva de com i perquè s'ha d'instal·lar quelcom, ni ho pretén ser. Simplement s'enuncien les passes a seguir per tenir un entorn funcional.

Addicionalment, enunciarem els diferents mètodes dissenyats en Python per a que l'usuari pugui interactuar amb el robot.

4.1. Guia d'instal·lació Windows

Seguirem un itinerari pas per pas, molt esquemàtic per a poder tenir l'entorn ideal per a l'execució de la plataforma UABot.

4.1.1. Instal·lació Python

Per a la programació dels diferents mètodes de comunicació i configuració hem utilitzat la versió 2.6.5 del llenguatge Python. Cal tenir cura de la versió escollida. Una versió inferior pot ser incompatible amb l'actual codi Pyuabot implementat, mentre que una versió superior, tot i conserva la compatibilitat cap enrere, és possible que les dependencies requerides (Pywin32) no estiguin actualitzades per a tal versió.

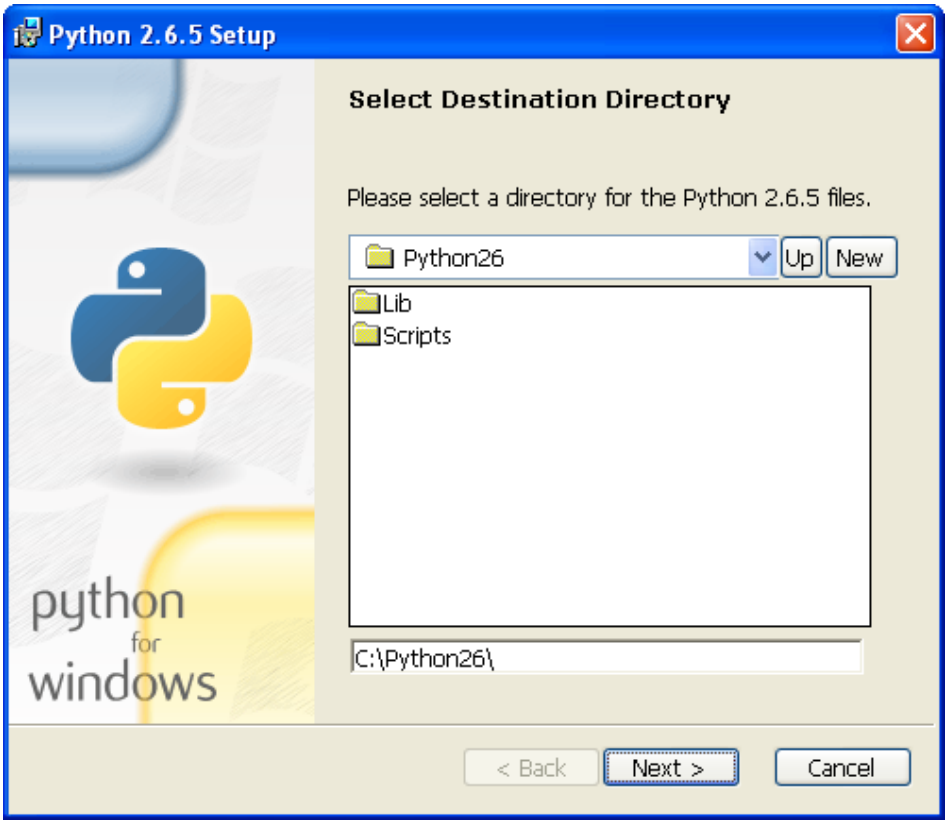
Simplement podem garantir que per la versió Python 2.6.5 les dependències existeixen i el seu funcionament és correcte.

Per a la instal·lació cal seguir les següents passes:

1. Accedir a Internet: www.python.org
2. Descarregar Python: Download > Python 2.6.5 Windows Installer
3. Executar l'instal·lador del paquet descarregat. (Si Windows ens demana la confirmació de l'execució de l'aplicatiu, clickar **Executar**)

4. Apareix **Python 2.6.5 Setup**. Seleccionar opció **Install for all users** i clickar **Next.**

5. Localització on es produirà la instal·lació. Per defecte C:\Python26\



6. Personalització de la instal·lació. Si volem tot per defecte, clickar **Next >**.

7. Esperem un temps fins a la fi de la instal·lació.
8. Si tot va correctament clickem **Finish**.

4.1.2. Instal·lació Pywin32 i Pyserial

Pywin32 (Python for Windows Extension) és una extensió dels serveis de Windows per a Python.

Pyserial és un paquet escrit en Python per configuració i connexió de comunicacions sèries RS232.

Instal·lació Pywin32:

1. Descarreguem l'instal·lador de la pàgina web Pywin32: Pywin32 > Download Now.
2. Clickar sobre l'executable descarregat. Si Windows ens alerta amb un missatge de seguretat, continuem el procés clickant **Executar**.
3. Apareix la pantalla Setup de l'instal·lador. Clickar **Next** (**Següent** si la configuració de l'idioma és el català).
4. Localització destí on s'instal·laran tots els arxius del procés d'instal·lació. Clickar **Next**.
5. Instal·lació preparada per realitzar-se. Clickar **Next** per confirmació.
6. El procés d'instal·lació es du a terme.
7. Finalitzat el procés d'instal·lació, clickar **Finish**.

Instal·lació Pyserial (versió emprada Pyserial 2.4):

1. Descarregar l'instal·lador de la pàgina web Pyserial: Pyserial > Download Page > Pyserial version.
2. Clickar sobre l'executable descarregat. Si Windows ens alerta amb un missatge de seguretat, continuem el procés clickant **Executar**.
3. Apareix la pantalla Setup de l'instal·lador. Clickar **Next.**
4. Destí de la instal·lació. Clickar **Next**.
5. Instal·lació preparada per a realitzar-se. Clickar **Next** per confirmació.
6. El procés d'instal·lació en progrés.
7. Finalització de la instal·lació. Clickar **Finish**.

4.1.3. Instal·lació Pyuabot

Un cop tenim tot l'entorn instal·lat, podem començar a utilitzar la llibreria de mètodes preparada per a la comunicació amb la plataforma UABot.

Pyuabot permet establir la comunicació amb el robot, així com enviar i rebre dades. Posseeix una sèrie de mètodes, a banda d'aquells per gestionar la comunicació, que permeten el control de totes les parts descrites. Són mètodes programats en Python que actuen com a ordres d'alt nivell: la invocació d'un mètode suposa l'enviament d'un ordre seguint un protocol propi de comunicació de la plataforma UABot.

Tot les passes de comunicació són transparents a l'usuari final, qui només ha d'invocar el mètode desitjat i esperar una resposta de retorn.

La instal·lació i utilització del paquet Pyuabot és extremadament simple. Per a la instal·lació només cal descomprimir l'arxiu `pyuabot.zip` en una carpeta determinada.

Per a la seva utilització només és necessari importar el mòdul `uabot.py` des de la consola del Python. Dins aquest mòdul resideix la classe que modelitza l'abstracció del robot en l'estació de treball host.