Proyecto Final
Robot recolector de residuos
Diseño, implementación y construcción física

Guillermo Campelo
Juan Ignacio Goñi
Diego Nul

30 de junio de 2010

Resumen

Palabras clave: *Robot, recolector, residuos, robótica, comportamientos, visión, MR-2FA, L298, FR304, HX5010, GP2D120, BC327, SRF05, CNY70, motor, servo, telémetro, daisy chain, RS232, subsumption, Webots, OpenCV, detección, objetos*

Índice

Índice de cuadros

Índice de figuras

1. Requerimientos

2. Ideas de implementación

2.1. Locomoción

distintos tipos de locomocion que tuvimos en cuenta y porque elegimos este

2.2. Sensado del entorno

distintos tipos de sensores disponibles y porque elegimos estos

2.3. Controlador

distintas formas de diagramar la forma de control, que tipo de controladores necesitamos, en cuales pensamos, con cuales nos quedamos

2.4. Método de recolección

distintos metodos que se nos ocurrieron

3. Actuadores

En nuestro caso, los motores son la principal forma en que el robot puede interactuar activamente con el ambiente que lo rodea. Cada una de las tareas que debíamos realizar requería de actuadores acordes.

Estas cuestiones son las que analizamos en este apartado.

3.1. Motores de contínua

Para la tracción principal de las ruedas necesitabamos motores que tuvieran el torque necesario para mover el robot, pero que pudieramos medir y controlar la velocidad era la principal necesidad. Para esta tarea utilizamos motores de contínua con caja reductora y encoder. Con dos de estos motores logramos poder garantizar una velocidad determinada en las ruedas, controlar de la cantidad de movimiento en forma independiente en cada rueda y entre otras cosas conocer la cantidad de las vueltas dadas por cada una de las ruedas.

3.1.1. Características

Los motores que elegimos son de la marca Ignis[1] modelo *MR-2FA* con características expresadas en la tabla 1, están provistos de una caja reductora, poseen un encoder de 4 estados por vuelta en el eje del motor y un sensor de de efecto de campo para determinar una vuelta en la salida de la caja reductora.

La caja reductora provee una relación de 94 vueltas del motor por cada 1 vuelta del eje de salida de la caja.

En la figura 1 mostramos las dimensiones exteriores del motor.

[1] http://www.ignis.com.ar

Característica	Unidad	Mínimo	Nominal	Máximo
Tensión	V	8	9	12
Corriente	A	0.6	1.2	2.4
Velocidad	RPM	1	60	60
Aceleración	$1/s^2$	0.1	0.1	0.5
Torque	kgf*cm	0	1.2	6.4

Cuadro 1: Características del motor Ignis MR-2FA.

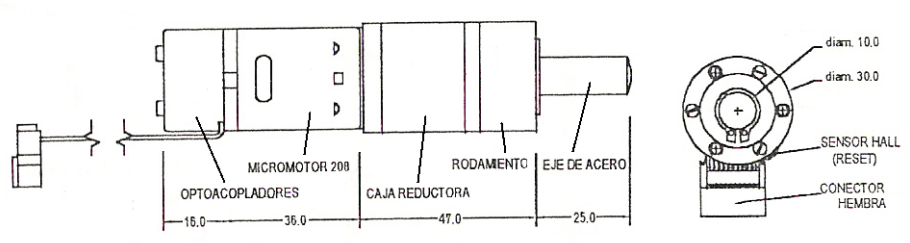

Figura 1: Vista lateral y frontal del motor Ignis MR-2FA.

Una ventaja que encontramos en este modelo es que ya trae el encoder integrado aunque su resolución podría haber sido mayor. Los encoders los explicamos más en detalle en la sección 4.4.

3.1.2. Circuito de control

Para alimentar y poder controlar los motores elegimos el driver *L298* de la marca ST[2]. Internamente tiene dos puentes H puenteables y puede soportar hasta $4A$. Ideal para estos motores u otros que se elijan en el futuro.

La principal función del driver era proveer de la corriente y voltaje necesarios para el funcionamiento del motor, pero la configuración del puente H nos dio la posibilidad de, con una lógica simple, determinar el sentido de la corriente y potencia que recibía el motor.

Este integrado admite el puenteo de la salida aumentando así la corriente que pueda circular por él. Para hacer esto, conectamos las salidas *Out1* y *Out4* por un lado y por el otro las salidas *Out2* y *Out3*. De igual forma los pines de habilitación *EnA* y *EnB*, luego la entrada *In1* con la *In4* por un lado y por el otro la *In2* con la *In3*.

De esta forma, se controla con sólo 3 cables, uno de habilitación y otros 2 de *Input* que determinan la polarización de los transistores internos y por ende, el sentido de giro del motor. En la tabla 2 mostramos la tabla de verdad para los pines de control, donde H es estado algo, L estado bajo y X cualquier estado. En la figura 2 mostramos el diagrama interno del integrado.

Para determinar la potencia que recibiría el motor usamos el módulo de *PWM* del microcontrolador, que explicamos más en detalle en la sección 5.2.2. Variando el ancho del pulso sobre el pin de habilitación del driver determinamos la cantidad de tiempo que el motor recibe tensión, lo cual se traduce en la

[2]http://www.st.com

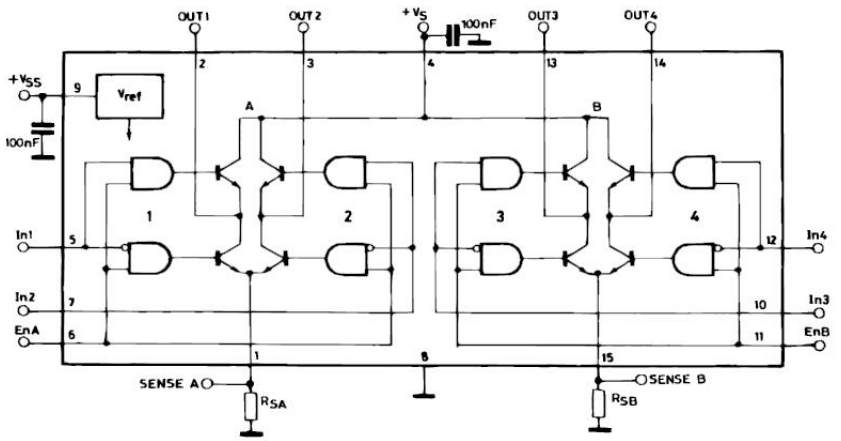

Figura 2: Diagrama interno del driver L298.

Enable	Input 1	Input 2	Función
H	H	L	Sentido horario
H	L	H	Sentido anti-horario
H	L	L	Motor frenado
H	H	H	Motor frenado
L	X	X	Motor libre

Cuadro 2: Tabla de verdad para el control del driver *L298*.

potencia que este tiene para realizar el movimiento. Cuanto mayor es el tiempo en estado alto del pulso, mayor la potencia.

Para contrarrestar la corriente negativa en las salidas del driver usamos los diodos *FR304* que cumplen con las especificaciones del driver con 150ns de tiempo de recuperación y una corriente de $3A$.

El consumo del motor lo medimos mediante los pines de sensado en el driver, conectado a masa por una resistencia y al módulo de *ADC* del microcontrolador, que explicamos en la sección 5.2.2. Conociendo el valor de la resistencia y el valor leido por el módulo de *ADC*, pudimos determinar cuanta corriente que circulaba por la resistencia y por ende la corriente consumida por el motor.

Para controlar la velocidad del motor usamos 1 de los 2 encoders que trae. Conectados como entrada para el módulo de *Timer* del microcontrolador, en configuración de contador, que explicamos más en detalle en la sección 5.2.2. Usando otro *Timer* para tener una base de tiempo fija y con el valor del contador pudimos determinar y controlar la velocidad de las ruedas.

En la sección 7.2 explicamos el desarrollo de la placa controladora de estos motores.

3.1.3. Rutinas de control

Desde el punto de vista del código, tuvimos que desarrollar las rutinas necesarias para el manejo de los motores según las intrucciones del controlador principal.

Característica	Unidad	Valor
Torque	kg	6,5
Velocidad	segundos/grado	$\frac{0,16}{60}$
Voltaje	V	4,8 a 6
Delay máximo	μs	4
Dimensiones	mm	40x20x38
Peso	g	39

Cuadro 3: Características del servo HX5010.

Configuramos a uno de los timers internos del microcontrolador para que genere una interrupción cada 6,25ms, la cual usamos para realizar chequeos y ejercer control sobre el motor. Verificamos el consumo del motor para evitar sobrecargar el circuito y los motores ante un posible atasco de las ruedas. También actualizamos el acumulado de vueltas realizadas por el motor para la odometría.

Cada 200ms, o sea 32 interrupciones, tomamos la cantidad de cuentas del encoder, correjimos la velocidad de giro del motor ajustando el ancho del pulso generado por el PWM. Luego borramos el contador y esperamos otros 200ms.

3.2. Servo motores

Para el movimiento de las partes del módulo de recolección. una cámara con paneo y giro o un senor de ultrasonido colocado en la parte superior haciendo las veces de radar, pensamos en el uso de servo motores. La principal característica de estos actuadores es que nosotros sólo debemos indicar el ángulo al que queremos que este el eje del motor y este se coloca automáticamente. El ángulo de trabajo va desde 0° a 180° y algunos llegan hasta los 200°.

Aunque no fue implementado el ningún mecanismo que requiriera el uso de estos motores, explicamos en este apartado el trabajo realizado en torno a este tipo de actuadores. En la sección 7.4 explicamos el diseño de las placas que los controlan.

El servo de prueba que utilizamos para el desarrollo es el modelo *HX5010* de la marca Hextronik[3] con caracteristicas que expresamos en la tabla 3

3.2.1. Circuito de control

La alimentación y consumo depende del modelo específico, variando también el torque que posee el servo.

No es necesario el uso de un driver para manejarlos, simplemente con la alimentación y una señal con el ángulo es suficiente. La forma de comunicar el ángulo varía entre los distintos servos y fabricantes. Hay servos analógicos y servos digitales. En los primeros la posición se determina mediante un voltaje que varia según cierto rango y si es digital, se setea mediante el ancho de un pulso que tiene un tiempo mínimo y máximo para mapear los ángulos mínimo y máximo respectivamente.

Dentro del modo de uso, podemos hacer que queden sueltos o que se queden fijos en cierta posición indicando, de forma contínua, el valor del ángulo requerido. La frecuencia a la que se debe setear la posición depende el modelo.

[3] www.hextronik.com/

3.2.2. Rutinas de control

Debido a que pensamos usar servos digitales y por lo menos íbamos a necesitar 3 servos, necesitabamos contar con varios módulos de PWM. Como sólo disponíamos de 1, decidimos realizar la misma función pero por software.

Usamos el timer de 16bits del microcontrolador configurado con el clock interno como medida del tiempo para crear 5 salidas con pulsos que varían el ancho en forma independiente cada una. Definimos un ancho mínimo y máximo, pudiendo configurar pasos intermedios de 1° (aproximadamente 69,4μs).

4. Sensado

En este apartado explicamos detalladamente cada uno de los sensores que utilizamos para realizar tanto las mediciones externas como las internas al robot. Analizamos las ventajas de cada uno, problemas que encontramos y sus soluciones.

En la sección 7.3 explicamos el diseño y construcción de las placas que controlan todos los sensores del robot.

4.1. Telémetros infrarrojos

El principio de funcionamiento de estos sensores es mediante un haz de luz infrarroja que es emitido hacia el objetivo, el cual es reflejado y captado a traves de un lente por un sensor de posición relativa en el interior del sensor. En base a esta medición se calcula la distancia entre el sensor y el objeto reflectivo que se encuentra frente a él.

4.1.1. Características

medidas, tiempo de muestreo, tipo de salida, rangos de distancia, rangos de voltaje, distancia vs voltaje

Los telémetros infrarrojos que elegimos son de la marca Sharp[4], modelo *GP2D120*. En la tabla 4 detallamos los valores característicos del modelo.

Este tipo de sensores tienen un retardo de aproximadamente 43,1ms durante el cual la lectura que se realiza no es confiable y luego las nuevas lecturas se hacen en ventanas de aproximadamente el mismo tiempo. En la figura 3 mostramos el diagrama de tiempos.

Característica	Unidad	Valor
Rango máximo	cm	30
Rango mínimo	cm	4
Tensión para la máxima distancia	V	1.95
Tensión para la mínima distancia	V	2.55
Tensión de alimentación	V	5
Consumo máximo	mA	50

Cuadro 4: Características del sensor de distancia por ultrasonido SRF05.

[4]http://sharp-world.com/products/device

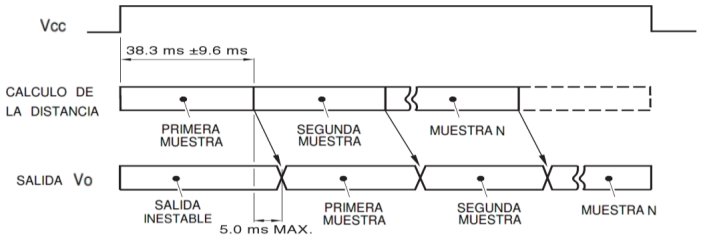

Figura 3: Diagrama de tiempos del sensor GP2D120.

4.1.2. Circuito de control

alimentacion, conmutacion (transistor, estado de habilitacion: 0), conexion con el micro, salida del sensor, modulo ADC, muestreo, capacitor para alimentacion, circuito minimo (diagrama)

No necesitábamos un driver para manejar los telémetros, pero decidimos usar un transistor para poder habilitarlos o no, de otra manera el sensor estaba tomando mediciones continuamente provocando un consumo de batería innecesario. De esta forma sólo se enciende cuando se va a utilizar.

El transistor que utilizamos es un conmutador y amplificador de uso general, el *BC327* y por ser de tipo PNP se exita con un estado bajo, por lo que la lógica de conmitación está negada.

El tipo de salida de este modelo de telémetros es analógica y están conectadas al módulo *ADC* del microcontrolador. En la figura 4 mostramos la tabla de conversión entre voltaje de salida y distancia al objeto, y en la figura 5 mostramos el ángulo de apertura de la zona de detección según la distancia al objetivo.

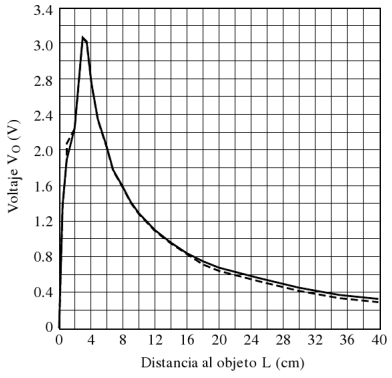

Figura 4: Voltaje de salida según la distancia al objeto del telémetro GP2D120.

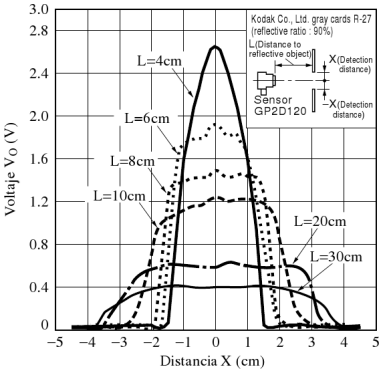

Figura 5: Ángulo de apertura según la distancia del telémetro GP2D120.

4.1.3. Rutinas de control

Controlar los telémetros es relativamente sencillo. Usamos el timer de 16bits del microcontrolador configurado con el clock interno para determinar el tiempo

en el cual debíamos tomar las muestras con el ADC y simplemente realizamos un promedio entre ellas para obtener la distancia al objetivo.

Para hacer esto desarrollamos una pequeña máquina de estados que controla y maneja los tiempos para tomar las muestras que explicamos más en detalle en la sección 7.3.7.

4.2. Sensor de distancia por ultrasonido

Estos sensores de distancia se basan en la velocidad del sonido para calcular la distancia al objetivo. Genera un tren de 8 pulsos ultrasónicos y luego se espera como respuesta, el mismo tren de pulsos que debería haber rebotado contra el objetivo. En base a la diferencia de tiempo entre la emisión del tren de pulsos y la respuesta, se calcula la distancia a la que se encuentra el objetivo.

4.2.1. Características

El sensor de distancia por ultrasonido que elegimos es el modelo *SRF05* de la marca Devantech Ltd[5]. Esta versión mejorada del modelo *SRF04*, aumenta el rango de detección y mejora el modo de control y lectura de los datos, permitiendo hacerlo mediante un único pin.

La distancia medida mediante el tren de pulsos es codificada linealmente en el ancho de un pulso que varía de $100\mu s$ a $25ms$. Si dentro del rango de detección no se encuentra ningún objeto, el pulso tendrá un ancho de $30ms$.

En la tabla 5 detallamos las caracterśticas del sensor *SRF05* y en la figura 6 mostramos el haz ultrasónico del sensor.

Característica	Unidad	Valor
Tensión de alimentación	V	5
Corriente	mA	4
Frecuencia de trabajo	KHz	40
Rango máximo	cm	400
Rango mínimo	cm	1.7
Duración mínima del pulso de disparo	μs	10
Duración del pulso eco de salida	μs	100 - 25000
Tiempo mínimo de espera entre mediciones	ms	50
Dimensiones	mm	43x23x40

Cuadro 5: Características del sensor SRF05.

4.2.2. Circuito de control

No necesitamos de un driver para manejar al sensor ya que lo conectamos directo a los $5v$ de la placa. El pin de *Mode* lo dejamos en estado bajo para indicar que debe funcionar bajo el nuevo modo y no en compatibilidad con el *SRF04*.

En el pin de *TRIGGER* sólo generamos un pulso de al menos $10\mu s$ para desencadenar en la lectura de la distancia al objetivo. El sensor nos asegura que

[5]http://www.robot-electronics.co.uk/

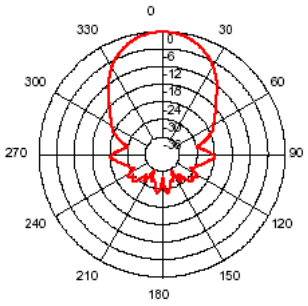

Figura 6: Haz ultrasónico del sensor SRF05.

no generará el pulso de respuesta hasta pasados los $700\mu s$ desde pasado el pulso de trigger. En la figura 7 mostramos el diagrama de tiempos.

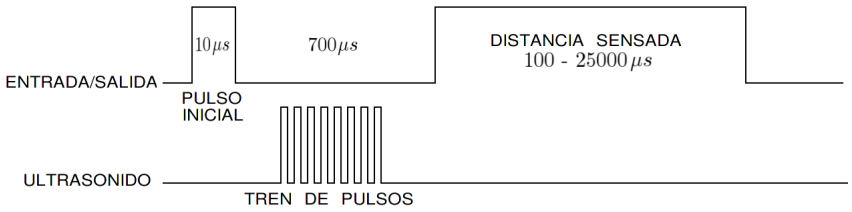

Figura 7: Diagrama de tiempos del sensor SRF05.

Para evitar que el rebote de otros sensores o sensados anteriores influya en la lectura, se debe esperar un mínimo de $50ms$ antes de generar otro medición.

4.2.3. Rutinas de control

Usamos uno de los pines con interrupción externa en el que conectamos el pin de *TRIGGER* y el timer de 16bits del microcontrolador configurado con el clock interno.

Para realizar la medición, generamos un pulso de $15\mu s$ para asegurarnos el disparo del sensor y cambiamos el modo del pin a entrada con interrupción ante un flanco ascendente. Cuando salta la interrupción significa que comienza al pulso con la distancia codificada en su ancho, por lo que tomamos una muestra del timer y configuramos al pin para que genere ahora una interrupción ante un flanco descendente. Cuando salte la próxima vez la interrupción, será porque termino el pulso con la medición, por lo que sólo debemos hacer la resta entre el valor actual del timer y la muestra que tomamos al principio para conocer la distancia a la que se encuentra el objetivo.

Un tiempo obtenido mayor a los $25ms$ indica que no se detectó ningún objeto dentro del rango del sensor.

4.3. Sensor reflectivo de piso

Estos sensores opticos reflectivos emiten luz infrarroja y captan el nivel de luz reflejada sobre la superfície a sensar como muestramos en la figura 9. En

la figura 8 mostramos las dimensiones del sensor. La intensidad de luz captada depende de la distancia al objetivo y del color y nivel de reflectividad de la superficie. Es por esto que usamos estos sensores para identificar una línea en el piso.

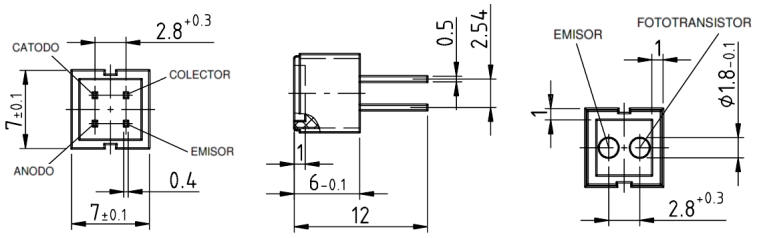

Figura 8: Medidas en milímetros del sensor CNY70.

4.3.1. Características

Los sensores que elegimos para son del modelo *CNY70* de la marca Vishay Semiconductor[6].

El rango efectivo de sensado ronda los $3mm$ de distancia aunque, con un incremento en la corriente que circula por el emisor se puede llegar a una distancia mayor a la recomendada por el fabricante y que nos permita un uso más acorde al proyecto. El emisor soporta un pulso de hasta $3A$ por un tiempo menor o igual a $10\mu s$.

En la figura 10 mostramos la corriente que circula por el colector del fototransistor en base a la distancia al objeto medido.

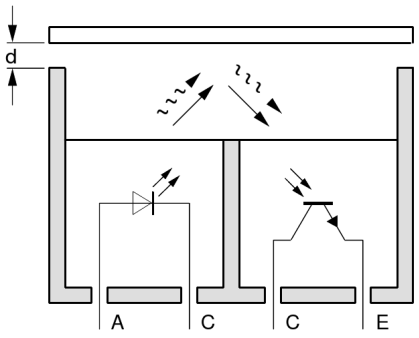

Figura 9: Principio de funcionamiento reflectivo del sensor CNY70.

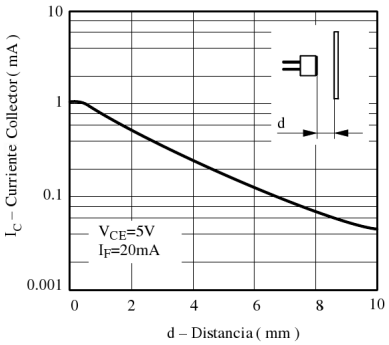

Figura 10: Corriente en el colector según la distancia del sensor CNY70.

[6]http://www.vishay.com/

4.3.2. Circuito de control

De igual forma que en los telémetros utilizamos un transistor como habilitación de la tensión de alimentación en el sensor. Esto nos dio la posibilidad de encenderlo y apagarlo a la hora de tomar las muestras de la luz reflejada por piso o la línea. Nuevamente la lógica de habilitación es invertida por tratarse de un transistor *BC327*.

Agragamos una resistencia para limitar la corriente que circulaba por el emisor y otra como pull-up en el emisor del fototransistor que a su vez, conectamos al módulo de *ADC* del microcontrolador para efectuar las mediciones.

4.3.3. Rutinas de control

El código que desarrollamos para obtener las muestras de estos sensores es sencillo, simplemente debemos habilitar el transistor que alimenta al sensor con un estado lógico bajo, tomar al menos 4 muestras y promediarlas para tener un valor adecuado del nivel de luz reflejado por la superficie. Deshabilitamos el sensor y enviamos el valor.

Debido al circuito que armamos con un nivel alto de reflexión leemos un valor bajo en el conversor analógico digital y con un nivel bajo de luz, un valor alto.

4.4. Encoders

Los encoders son sensores que convierten una posición lineal o angular en señal eléctrica o pulsos. Pueden determinar una posición de forma absoluta o simplemente informar que hubo un movimiento. El método de sensado y la resolución del ángulo de giro que detectan varía según el modelo.

4.4.1. Características

Los motores *MR-2FA* tienen encoders de cuadratura conectados al eje, previo a la caja reductora. Estos encoders son de tipo fotoeléctricos, estan dispuestos a 135° uno del otro y marcan 4 estados por cada vuelta del motor. Estan colocados así para poder conocer el sentido de giro midiendo la secuencia de estados de cada encoder. Adicionalmente el motor cuenta con un sensor de efecto Hall el cual nos permite detectar una revolución completa en el eje de salida de la caja reductora.

Nosotros como conocemos el sentido de giro del motor, pues lo determinamos con el puente H, necesitamos sólo uno de los dos encoders para conocer y controlar la velocidad a la que gira el rotor del motor.

A la tensión máxima los motores superan a las 320 cuentas por segundo, el mínimo y máximo recomendables son, para que podamos mantener un giro constante, 60 y 300 cuentas por segundo respectivamente. Estos cálculos son usando uno de los dos fototransistores del encoder.

Dependiendo el tamaño de las ruedas será la velocidad de final del robot. La relacioón de caja de $94:1$ del motor.

4.4.2. Circuito de control

Conectamos una resistencia pull-up a $5V$ para la salida de sensado de los fototransistores y del sensor de efecto Hall. También incluimos un switch doble inversor para elegir cuál de los dos encoders usar. El punto común del switch está conectado al pin de entrada de clock externo de uno de los Timers del microcontrolador.

La alimentación de los encoders es directa y permanecen encendidos en todo momento.

4.4.3. Rutinas de control

Como explicamos en la sección 3.1.3, en cada interrupción del actualizamos el histórico de cuentas del motor adicionando o restando el último valor del contador.

También comparamos contra las cuentas esperadas por intervalo de tiempo que fueron determinadas desde el controlador principal para poder determinar si debemos incrementar o disminuir la potencia del motor y por ende la velocidad de las ruedas.

4.5. Sensado de la bateria

El sensado del nivel de tensión en la batería lo hacemos mediante un divisor de tensión entre los polos de la batería. La salida del divisor es sensada de igual forma que los otros sensores, mediante el módulo de ADC del microcontrolador. En la figura 11 mostramos el diagrama del divisor de tensión.

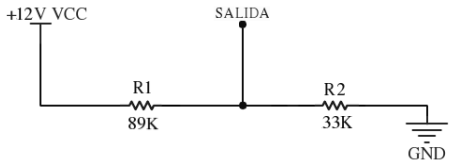

Figura 11: Divisor de tensión para el sensado de la batería.

En la tabla 6 mostramos las posibles tensiones en la batería y la tensión de salida en el divisor. También incluimos el valor aproximado para el ADC con tensión de referencia a $5V$ que leería la salida del divisor. El rango de voltajes que analizamos es tienendo en cuenta la posibilidad de efectuar mediciones durante la carga de la batería y sabiendo que con una tensión menor a $5V$ la lógica del prototipo que construimos comenzaría a fallar.

La conexión es simple, los cables de entrada del divisor los conectamos a los polos de la batería y la salida al ADC. Luego sólo debemos realizar las lecturas en el ADC y promediarlas para poder realizar los cálculos de la tensión en la batería.

4.6. Consumo del motor

El consumo de los motores de corriente contínua lo medimos leyendo el pin de sensado que se encuentra en el puente H que alimenta al motor. Lo que medimos con el módulo de ADC del microcontrolador es la tensión en este

Batería (V)	Salida (V)	Valor en el ADC
16	4.327	886
15	4.057	831
14	3.786	776
13	3.516	720
12	3.245	665
11	2.975	609
10	2.704	554
9	2.434	499
8	2.163	443
7	1.893	388
6	1.623	332
5	1.352	277

Cuadro 6: Tensión de la batería y la tensión de salida en el divisor.

pin. Esta depende de la caída de tensión en la resistencia conectada a masa y de la corriente que circula por el motor. Conociendo el valor de la resistencia podemos calcular el consumo del motor. En la tabla 7 comparamos el consumo en el motor, el voltaje sensado y la lectura en el ADC.

Tensión (V)	Consumo (A)	Valor en el ADC
0	0	0
0,09	0,19	18
0,18	0,38	36
0,27	0,57	55
0,36	0,76	73
0,45	0,95	92
0,54	1,14	110
0,63	1,34	129
0,72	1,53	147
0,81	1,72	165
0,90	1,91	184
0,99	2,10	202
1,08	2,29	221
1,17	2,48	239

Cuadro 7: Tabla comparativa para el consumo del motor.

4.6.1. Pulsador u otro dispositivo disparador

Ademas de los sensores que describimos, pensamos que nos podrían hacer falta pulsadores para controlar funciones simples por ejemplo saber cuando una parte de algún mecanismo llega a cierto punto o detectar finales de carrera de un servo o sin fin. También pueden ser otro tipo de sensores que generen un cambio de estado en el pin de sensado que se conecta al microcontrolador.

Agregamos la posibilidad de usarlos en las distintas placas como explicamos

en detalle en las secciones 7.3 y 7.4.

4.6.2. Rutinas de control

La lectura en el estado de los pulsadores puede ser bajo demanda con solo leer el estado del pin en el que estan conectados o puede ser ante una interrupción por cambio de estado. Estas cuestiones las tuvimos en cuenta a la hora de diseñar las placas.

5. Controladores

Todas las funciones del robot las debíamos controlar mediante algún tipo de dispositivo. Decidimos utilizar una netbook y microcontroladores para esta tarea. Estas cuestiones son las que explicamos en este capítulo.

5.1. Netbook

Elegimos como controlador principal la netbook *EeePC 1005-HA* de la marca Asus[7]. Como características principales cuenta con un procesador Intel[8] Atom N280 1.66 GHz, 1GB DDR-2, un disco de 250GB, una pantalla de 10 pulgadas y una batería de 48Wh que nos da una autonomía de aproximadamente 8 horas. Cuenta con una cámara integrada de 0.3MP, placa de red inalámbrica y ethernet, placa de sonido y 3 puertos USB. Pesa aproximadamente 1.27Kg y sus dimensiones son 26,2 x 17,8 x 3,7 centímetros.

Usamos Ubuntu[9] como sistema operativo y programamos tanto la lógica de comportamientos como la captura y análisis de imágenes en C/C++.

5.2. Microcontrolador

Para el control de la velocidad de los motores, lectura de los encoders y los sensores de distancia usamos un microcontrolador. Nuestro diseño contempló la existancia de varios módulos con una o pocas funciones simples que se comunicaran entre ellos y con el controlador principal, en nuestro prototipo la netbook. La razón por la cual lo armamos así fue para simplificar cada placa controladora a nivel software y hardware.

Explicamos la comunicación entre los distintos controladores en la sección 6.

5.2.1. Características

El microcontrolador que elegimos para realizar las tareas de control, configucarión y comunicación a bajo nivel es el *PIC16F88* de Microchip[10]. Cuenta con una arquitectura de memoria del tipo Hardvard, con una memoria *FLASH* para 4096 instrucciones de programa, una memoria *RAM* de 368 bytes y una memoria *EEPROM* de 256 bytes. Tiene un set de instrucciones básicas reducido y todas con el mismo tiempo de ejecución. En este apartado nombramos algunos

[7] www.asus.com
[8] www.intel.com
[9] http://www.ubuntu.com/
[10] http://www.microchip.com/

de los principales periféricos incluidos en el microcontrolador y la utilidad dentro del proyecto que encontramos para ellos. Utilizamos con un cristal externo de 20MHz como clock.

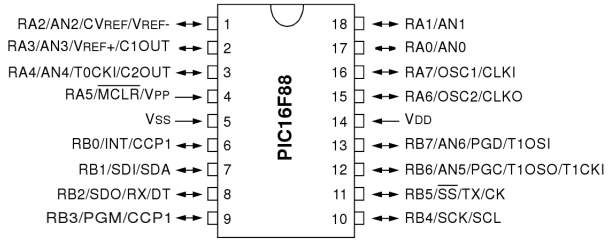

Figura 12: Diagrama del microcontrolador PIC16F88.

El microcontrolador tiene 2 puertos de 8 entradas y salidas cada uno de tipo TTL y CMOS. Como mostramos en la figura 12 cada pin se encuentra multiplexado con uno o más periféricos internos.

5.2.2. Módulos internos

Internamente el microcontrolador tiene una serie de periféricos que proveen de funciones extras y que utilizamos para lograr cumplir con las necesidades de nuestro proyecto. En la figura 13 mostramos los distintos módulos internos del microcontrolador.

Cuenta con 3 timers, 2 de 8bits (*TIMER0* y *TIMER2*) y 1 de 16bits (*TIMER1*). Podemos configurarlos para que tomen al clock del microcontrolador o que tomen una fuente externa de clock. También podemos aplicarles demultiplicadores que generan un clock de menor frecuencia al que se usa como entrada. Pueden ser etapas previas o posteriores al timer y nos dan gran flexibilidad de uso.

El *TIMER0* lo utilizamos para hacer control del tiempo en nuestros códigos. Conectados con el clock principal y configurados para que generen una interrupción al hacer overflow, obtenemos una buena medida del paso del tiempo.

El *TIMER1* configurado como fuente externa a la salida del encoder, lo usamos como contador de pasos para medir la velocidad del motor. También lo usamos como medición del tiempo para hacer las lecturas de los sensores. Elegimos a este timer ya que al ser de 16bits posee un mayor rango de valores y

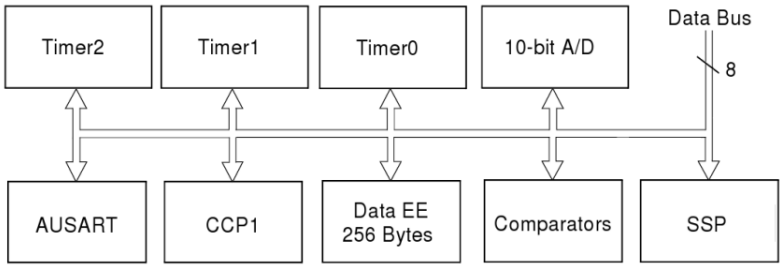

Figura 13: Módulos internos del microcontrolador PIC16F88.

por lo tanto, podíamos medir un mayor lapso de tiempo o cuentas del encoder en cada caso.

El *TIMER2* lo usamos en conjunto con el módulo de *PWM* para determinar el ancho del pulso que habilita al puente H que provee de energía a los motores.

El módulo conversor analógico digital nos dio la posibilidad de medir tensiones analógicas como por ejemplo las salidas de los sensores, tensión en la bateria o el consumo de los motores. Tiene 7 canales o pines distintos y podemos configurarlo para que genere un valor de 8 o 10bits. También podemos determinar si se debe usar el valor de *Vcc* y *GND* como referencia o podemos proveer de forma externa de los voltajes de referencia para generar un rango de voltajes diferente y aumentar o disminuir así la resolución del conversor.

El módulo de *PWM* nos provee la posibilidad de generar pulsos contínuos de un ancho determinado. En nuestro proyecto lo utilizamos como habilitación del puente H que alimenta a los motores variando así la potencia y por lo tanto, la velocidad final de las ruedas. Se utiliza en conjunto con el *TIMER2* seteando el prescaler y postscaler para determinar el ancho del estado alto y del estado bajo de los pulsos.

Gracias al módulo de *AUSART* podemos realizar la comuinicación entre los distintos microcontroladores por hardware. Este periférico nos provee una comunicación sincrónica o asincrónica dependiendo de la configuración. Creamos la red de *Daisy Chain* sobre el protocolo de RS-232.

El microcontrolador dispone de otros periféricos como un comparador *CCP* y comunicación sincrónica *SPI*. Utilizamos los pines de uso general para realizar otras funciones como habilitación, conmutación o las señales de *PWM* por software para determinar la posición de los servo motores, por ejemplo.

5.2.3. Programación del firmware

El firmware de cada microcontrolador lo escribimos en C y usamos la IDE de programación *Microchip MPLAB*[11] con el compilador *CCS PCM V4.023*[12]. Usamos el programador *ICD2* de la empresa *Microchip* para descargar código compilado al microcontrolador. El cual nos dio la posibilidad de debuguear el código en más de una oportunidad.

6. Comunicación

La comunicación interna entre los controladores de cada periférico y el controlador principal, donde concentramos la lógica de los comportamientos era vital. Para poder tomar las decisiones adecuadas la telemetría debía tener toda la información requerida, la frecuencia suficiente para que las reacciones sean lo más dinámicas y rápidas posibles y poseer la menor cantidad de errores para evitar retransmisiones.

En esta sección analizamos todo lo referente al medio de transmisión, protocolo y comandos que forman parte de la comunicación interna del proyecto.

[11] http://www.microchip.com/
[12] http://www.ccsinfo.com/

6.1. Conectividad entre módulos

Para establecer el canal de comunicaciones decidimos emplear una configuración basada en el método *Daisy-Chain*[13] entre los controladores. Creando un anillo donde cada nodo de la cadena se comunica con su vecino retransmitiendo cada paquete hacia adenlante hasta su destinatario como mostramos en la figura 14.

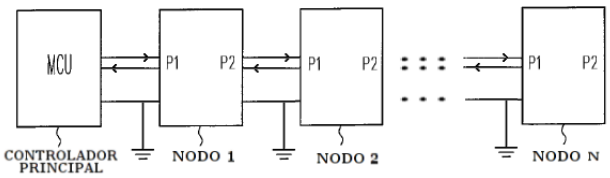

Figura 14: Diagrama general del método daisy chain

La configuración que elegimos para realizar la comunicación fue una velocidad de 115200 baudios, 8 bits, con 1 bit de parada, sin bit de paridad y sin control de flujo. Logramos una gran velocidad de respuesta a los comandos de esta forma.

En las tablas 8 y 9 especificamos el conexionado entre las placas y contra el controlador principal.

Función	Conector RJ11	Conector RJ11	Función
Serial RX	2	2	Serial TX
Serial TX	3	3	Serial RX
No conectado	4	4	No conectado
GND	5	5	GND

Cuadro 8: Conexionado entre placas en modo Link

Función	Conector RJ11	Conector DB9	Función
Serial RX	2	3	Serial TX
Serial TX	3	2	Serial RX
No conectado	4	4	Shield
GND	5	5	GND

Cuadro 9: Conexionado entre placa y la PC

Como terminación de la cadena, debemos o colocar una ficha nula que interconecte *TX* con *RX* o cambiar el switch de configuración de *LINK* a *LAST*.

6.2. Protocolo de comunicación

El protocolo de comuncación está formado por paquetes que tienen un formato específico y representan un pedido de información o comando que debe ser ejecutado en el destino.

[13]Patente US20090316836A1

El paquete consta de un header común con datos que identifican al emisor y receptor del paquete, el comando a enviar y posibles datos extras que sean requeridos. Todos los paquetes tienen una respuesta obligatoria de confirmación de recepción. Cuando el paquete requiere una respuesta con datos, la confirmación va acompañada de la información requerida. En el cuadro 10 mostramos la estructura interna de un paquete típico.

LARGO	DESTINO	ORIGEN	COMANDO	DATO	CRC

Cuadro 10: Formato y header del paquete de datos

Tanto los paquetes de envío de datos como los de respuesta tienen el mismo formato y comparten el valor en el campo de comando.

El campo *LARGO* indica el tamaño en bytes del paquete que viene detras, consta de 1 byte. Necesario porque el campo *DATO* es de longitud variable, si la longitud del campo *DATO* es cero, es 0x04.

El campo *DESTINO* identifica al destinatario del paquete y consta de 1 byte. Los 4 bits más significativos indican el grupo y los 4 bits menos significativos, el número de ID de la placa de destino. Si el ID de placa es F, entonces el paquete es broadcast a todos los IDs del grupo indicado y si el valor es 0xFF, el paquete es broadcast a todas las placas de todos los grupos.

De igual forma *ORIGEN* determina el emisor del paquete para la respuesta y es de 1 byte. Los 4 bits más significativos indican el grupo y los 4 bits menos significativos, el número de ID de la placa de origen. Los valores permitidos son del 0 al E, ya que F indica broadcast y no es válida una respuesta broadcast.

El campo *COMANDO* informa al destinatario la tarea a realizar o determina un pedido de información que puede o no tener parámetros. Ocupa 1 byte.

El campo *DATO* contiene los parámetros o datos extras que puedan ser necesarios para el comando enviado. En el caso que el comando no los requiera, el campo debe ser nulo y el campo *LARGO* será 0x04.

El control de errores lo realizamos mediante un checksum calculado, haciendo un *XOR* con cada byte del contenido del paquete y colocandolo en el campo *CRC*. Cuando un paquete se encuentre con errores o esté mal formado, el destinarario debería pedir la retrasmisión. De igual forma, creimos necesaria la creación de un control adicional por medio de una lista de paquetes no confirmados mantenida por el controlador principal para evitar la pérdida de comandos.

6.2.1. Comandos comunes

Creamos comandos que son comunes a cualquier placa sea cual sea su grupo. La intención de esto fue poder tener un espacio de comunicación común para poder intercambiar comandos de inicialización, identificación de controladores, versionado o algúna otra necesidad genérica que se desarrolle en un futuro y deba ser incluida en el protocolo. El controlador principal debería ser quien envía este tipo de comandos, pero no creamos restricciones al respecto.

En la tabla 11 listamos y explicamos brevemente los comandos comunes del protocolo.

Comando: *INIT*. Sincroniza el inicio de todas las placas en la cadena.	
0x01	Vacío
Respuesta	
0x81	Descripción de la placa en texto plano.
Comando: *RESET*. Pide el reset de la tarjeta.	
0x02	Vacío
Respuesta	
0x82	Descripción de la placa en texto plano.
Comando: *PING*. Envia un ping a la placa.	
0x03	Vacío
Respuesta	
0x83	Vacío
Comando: *ERROR*. Informa que ha habido un error.	
0x04	Código de error.
Respuesta	
0x00	Único comando sin respuesta directa. Paquete original.

Cuadro 11: Comandos comunes a todos los controladores.

6.2.2. Comandos específicos

Cada grupo de placas tiene comandos propios y específicos dependiendo de la función que deban desempeñar en el sistema. Existen grupos con comandos predefinidos cada uno se trata en las secciones como se detalla en el listado. Los comandos específicos para cada grupo deben ser desde 0x40 hasta el valor 0x7E.

El *MAIN CONTROLLER* no posee por ahora ningún comando escpecífico pero le reservamos el espacio en el protocolo. En las tablas 12 y 13 enumeramos los comandos para el controlador *DC MOTOR*. El controlador *SERVO MOTOR* responde a los comandos de las tablas y En la tabla 16 listamos los comandos del controlador *DISTANCE SENSOR*. De igual forma, aunque no esten implementados reservamos y especificamos los comandos para *BATTERY CONTROLLER* y *TRASH BIN* en las tablas 17 y 18 respectivamente.

6.2.3. Estadísticas

analisis de paquetes por segundo, bytes de datos vs bytes de header, retransmisiones, etc

7. Placas controladoras

porque tuvimos que disear nuestras propias placas, cosas que tuvimos en cuenta y decisiones tomadas, codigos fuente a los apendices

Las funciones y requerimientos de nuestro proyecto tenían partes que eran comunes a otros trabajos en robótica o quizás de electrónica general. Pero tenían peculiaridades y aspectos que no pudimos satisfacer con placas prefabricadas. Por ejemplo, un protocolo de comunicación unificado entre todos los módulos era, de entrada, una barrera que acotaba las posibilidades. Y mantener distin-

SET DIRECTION. Seteo del sentido de giro del motor.	
0x40	Sentido de giro.
Respuesta	
0xC0	Vacío.
SET DC SPEED. Seteo de la velocidad del motor.	
0x01	Sentido de giro y velocidad en cuentas por segundos.
Respuesta	
0xC1	Vacío.
SET ENCODER. Seteo de las cuentas históricas del encoder.	
0x42	Valor para setear en el histórico del encoder.
Respuesta	
0xC2	Vacío.
GET ENCODER. Obtener las cuentas históricas del encoder.	
0x43	Vacío.
Respuesta	
0xC3	Valor histórico del encoder.
RESET ENCODER. Resetear las cuentas históricas a cero.	
0x44	Vacío.
Respuesta	
0xC4	Vacío.
SET ENCODER TO STOP. Seteo de cuantas cuentas para detenerse.	
0x45	Cuentas del encoder restantes para que el motor se detenga.
Respuesta	
0xC5	Vacío.
GET ENCODER TO STOP. Obtener la cuentas restantes hasta detenerse.	
0x46	Vacío.
Respuesta	
0xC6	Cuentas del encoder restantes para que el motor se detenga.
DONT STOP. Deshabilita el conteo de cuentas para frenar.	
0x47	Vacío.
Respuesta	
0xC7	Vacío.
MOTOR CONSUMPTION. Consulta sobre el consumo actual del motor.	
0x48	Vacío.
Respuesta	
0xC8	Consumo promedio del último segundo.

Cuadro 12: Comandos específicos al *DC MOTOR* parte A.

MOTOR STRESS ALARM. Alarma sobre un consumo extremo en el motor.	
0x49	Consumo ante el cuál sono la alarma.
Respuesta	
0xC9	Vacío.
MOTOR SHUT DOWN ALARM. Alarma, el motor ha sido apagado.	
0x4A	Consumo ante el que sono la alarma.
Respuesta	
0xCA	Vacío.
GET DC SPEED. Obtiene la velocidad en cuentas del encoder por segundo.	
0x4B	Vacío.
Respuesta	
0xCB	Sentido y velocidad de giro del motor.

Cuadro 13: Comandos específicos al *DC MOTOR* parte B.

tos modos y protocolos dentro del proyecto generaba una carga de lógica que podiamos evitar.

Analizando las posibilidades y entendiendo que una de las principales razones de nuestro trabajo era generar las bases de conocimiento y el punto de partida a futuro de un proyecto más grande, decidimos producir nuestra propia versión de placas controladoras.

Aunque iban a tener un uso bien definido dentro de nuestro trabajo, también creímos necesario que tuvieran un nivel de generalidada suficiente como para poder utilizarlas en otros proyectos con mínimos cambios, adelantando así mucho trabajo a futuro.

La modularización fue un factor importante y muy presente durante todo el desarrollo. En este apartado explicamos el diseño, desarrollo, programación y especificaciones de las distintas placas que creamos durante nuestro proyecto.

7.1. Placa genérica

Durante el diseño nos surgió la necesidad de establecer un módulo común de comunicación y una base de prueba para los testeos con los distintos sensores y periféricos que utilizaríamos en el robot. Es por esto que luego de pruebas aisladas, creamos una placa para estas cuestiones. Nos ayudó y aceleró en gran medida el diseño de las placas definitivas y nos dió la posibilidad de diseñar futuras actualizaciones a nuestro proyecto.

7.1.1. Características principales

Las características principales eran proveer de una interfaz común entre todas las placas para la comunicación y exportar todos los pines y periféricos del microcontrolador que elegimos para placas con las conexiones a los sensores o motores. Todo esto nos servía realizar pruebas de concepto para la conexión con los sensores, valores de componente pasivos ideales, comportamiento e interacción entre placas, para optimizar el código del firmware o crear nuevas expansiones.

SET POSITION. Determina la posición del servo motor indicado.	
0x40	*ID* del servo y la posición del eje.
Respuesta	
0xC0	Vacío.
SET ALL POSITIONS. Setea las posiciones de cada uno de los servomotores.	
0x01	La posición para cada uno de los 5 servos.
Respuesta	
0xC1	Vacío.
GET POSITION. Obtiene la última posición del servomotor indicado.	
0x42	*ID* del servo del que se quiere la posición.
Respuesta	
0xC2	*ID* del servo y la posición del eje.
GET ALL POSITIONS. Obtiene todas las últimas posiciones de los servomotor.	
0x43	Vacío.
Respuesta	
0xC3	La posición para cada uno de los 5 servos.
SET SERVO SPEED. Setea la velocidad para el servomotor indicado.	
0x44	*ID* del servo y la velocidad a setear.
Respuesta	
0xC4	Vacío.
SET ALL SPEEDS. Setea la velocidad para cada servomotor.	
0x45	Velocidad a setear para cada uno de los servos.
Respuesta	
0xC5	Vacío.
GET SERVO SPEED. Obtiene la velocidad para el servomotor indicado.	
0x46	*ID* del servo.
Respuesta	
0xC6	*ID* y velocidad del servo.
GET ALL SPEEDS. Obtiene las velocidades de cada uno de los servomotores.	
0x47	Vacío.
Respuesta	
0xC7	Velocidad de cada uno de los servomotores.
FREE SERVO. Deja de aplicar fuerza sobre el servo indicado.	
0x48	*ID* del servo a liberar.
Respuesta	
0xC8	Vacío.
FREE ALL SERVOS. Deja de aplicar fuerza sobre cada uno de los servomotores.	
0x49	Vacío.
Respuesta	
0xC9	Vacío.

Cuadro 14: Comandos específicos al *SERVO MOTOR* parte A.

GET STATUS. Obtiene el estado de cada uno de los switches.	
0x4A	Vacío.
Respuesta	
0xCA	1 byte con el estado de cada switch.
ALARM ON STATE. Establece si se desea recibir alarmas por cambio de estado.	
0x4B	*ID* y tipo de cambio en el switch .
Respuesta	
0xCB	Vacío.
SWITCH ALARM. Alarma ante un cambio de estado programado.	
0x4C	*ID* y estado del switch que provocó el comando.
Respuesta	
0xCC	Vacío.

Cuadro 15: Comandos específicos al *SERVO MOTOR* parte B.

7.1.2. Módulo de comunicación

Como explicamos en la sección 6 para la comunicación nos basamos en el modelo de *daisy chain* sobre una capa de transporte de *RS232*.

Los conectores y configuración son comunes y utilizamos dos tipos de conectores para realizar la interconexión entre las placas. Como mostramos en la figura 15 usamos conectores *RJ11* para generar el lazo entre nodos de la cadena y el conector *DB9* para la conexión con la PC. En las tablas 8 y 9 explicamos, respectivamente, el conexionado para cada caso.

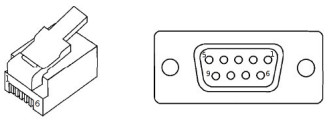

Figura 15: Conectores RJ11 (6P4C) y DB9.

Para determinar si la placa es un nodo más dentro de la cadena o es la terminación, colocamos una llave de dos posiciones que setea la configuración. En la posición *LINK* la placa actúa como nodo intermedio y en la posición *LAST* como terminación.

Es de vital importancia colocar en forma la correcta esta llave porque podemos dejar sin conexión al resto de las placas o mismo perder todos los paquetes transmitidos al no cerrar la cadena en el final de la misma.

7.1.3. Alimentación de la placa

La alimentación principal de la placa es 7 a 20 voltios, con la posibilidad de alimentarla directamente con 5 voltios por uno de los pines del conector. En la figura 16 mostramos la bornera y en el cuadro 19 el pinout.

La regulación interna de voltaje realiza por medio de un regulador 7805 corriente máxima de 1A.

ON DISTANCE SENSOR. Enciende el sensor de distancia indicado.	
0x40	ID del sensor a encender.
Respuesta	
0xC0	Vacío.
OFF DISTANCE SENSOR. Apaga el sensor de distancia indicado..	
0x01	ID del sensor a apagar.
Respuesta	
0xC1	Vacío.
SET DISTANCE SENSORS MASK. Habilita o no los sensores para lecturas.	
0x42	1 byte con cada ID del sensor a habilitar o deshabilitar.
Respuesta	
0xC2	Vacío.
GET DISTANCE SENSORS MASK. Obtiene la máscara de lectura.	
0x43	Vacío.
Respuesta	
0xC3	1 byte con cada ID del sensor.
GET VALUE. Obtiene el valor promedio de la entrada de los sensores indicados.	
0x44	1 byte con cada ID del sensor.
Respuesta	
0xC4	1 byte con cada ID y los valores de lectura promedio de cada sensor.
GET ONE VALUE. Obtiene el valor de la entrada del sensor indicado.	
0x45	1 byte con cada ID del sensor.
Respuesta	
0xC5	1 byte con cada ID y los valores de lectura de cada sensor.
ALARM ON STATE. Setea el tipo de cambio de estado del switch para la alarma.	
0x46	Tipo de cambio de estado.
Respuesta	
0xC6	Vacío.
SWITCH ALARM. Alarma informando que fue satisfecha la condición.	
0x47	Tipo de cambio y estado actual del switch.
Respuesta	
0xC7	Vacío.

Cuadro 16: Comandos específicos al *DISTANCE SENSOR*.

ENABLE. Habilita la alimentación del robot mediante la batería.	
0x40	Vacío.
Respuesta	
0xC0	Vacío.
DISABLE. Deshabilita la alimentación del robot mediante la batería.	
0x01	Vacío.
Respuesta	
0xC1	Vacío.
GET BATTERY VALUE. Obtiene el valor de la entrada de la batería.	
0x42	Vacío.
Respuesta	
0xC2	Lectura de la tensión en la batería.
BATTERY FULL ALARM. Mensaje informando que se completado la carga.	
0x43	Vacío.
Respuesta	
0xC3	Vacío.
SET BATTERY EMPTY VALUE. Valor de la batería crítico.	
0x44	Lectura de los voltios de la batería.
Respuesta	
0xC4	Vacío.
BATTERY EMPTY ALARM. El voltaje llegó a un valor crítico.	
0x45	Lectura de los voltios de la batería.
Respuesta	
0xC5	Vacío.
SET FULL BATTERY VALUE. Establece el valor para la carga completa.	
0x46	Lectura de la tensión en la batería.
Respuesta	
0xC6	Vacío.

Cuadro 17: Comandos específicos al *BATTERY CONTROLLER*.

GET TRASH BIN VALUE. Consulta que tan lleno está el cesto de basura.	
0x40	Vacío.
Respuesta	
0xC0	Valor que representa que tan lleno está el cesto interno de basura.
BIN FULL ALARM. El cesto de basura se ha completado y debe ser descargado.	
0x01	Vacío.
Respuesta	
0xC1	Vacío.
SET FULL BIN VALUE. Setea el valor para determinar que el cesto esta lleno.	
0x42	Valor que especifica que lecura se debe tomar como cesto lleno.
Respuesta	
0xC2	Vacío.

Cuadro 18: Comandos específicos al *TRASH BIN*.

Figura 16: Bornera de alimentación.

Pin	Voltaje
1	GND
2	5v
3	7v a 12v

Cuadro 19: Alimentación de la lógica

7.1.4. Configuración

La fila de pines *P2* exporta todos los pines con funciones dentro del microcontrolador, para realizar conexiones con periféricos de prueba o nuevas extensiones. Los headers *P3* y *P4* son jumpers que vinculan los pines *RA1* y *RA4* del microcontrolador los leds 1 y 2 respectivamente.

El header de programación *P1* lo utilizamos para conectar la placa con el programador y debuguear de código *ICD2* como explicamos en en apartado 5.2.3. El switch *S2* lo usamos para asociar los pines del microcontrolador con los canales de clock y data del header de programación (modo *ICD2*) o con los pines del header *P2*.

7.1.5. Esquemático

En la figura 17 mostramos el esquemático del microcontrolador y el conexionado con los headers, el módulo de comunicación y conectores para conformar la cadena *Daisy chain* la figura 18. En la figura 19 mostramos el diagrama de la fuente de alimentación y bornera.

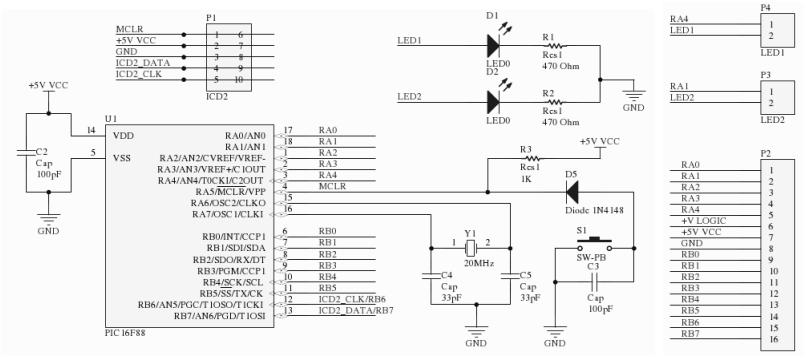

Figura 17: Microcontrolador y headers

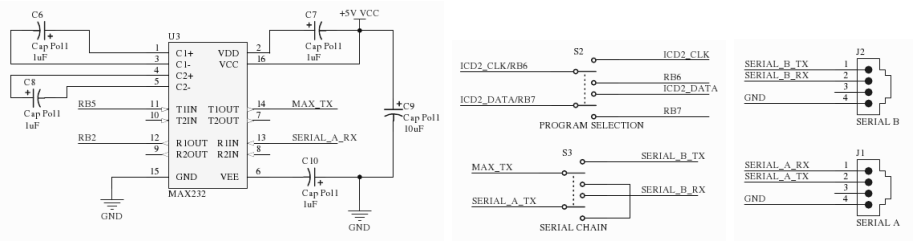

Figura 18: Comunicación, switch de modo y conectores de entrada y salida

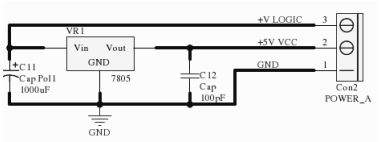

Figura 19: Fuente de alimentación

7.1.6. Circuito

En la figura 20 mostramos la máscara de componentes de la placa y en la figura 21 ambas capas de la placa.

7.1.7. Código básico

Según el uso que le fuimos dando a las placas genéricas fuimos cambiando el código del firmware que le íbamos cargando, pero siempre manteníamos partes básicas que nos ayudaban.

Principalmente la configuración de flags de configuración base del microcontrolador, desde la velocidad del clock para los cálculos de tiempos hasta determinar que pines eran mapeados a las funciones de RX y TX para la comunicación serial.

Para hacer pruebas sobre la comunicación RS232 o del protocolo, armamos maquetas para las funciones que atienden a los comandos específicos y funciones de control del protocolo comunes para todas las placas. También generamos un modelo general del programa, con un loop principal y llamadas incializadoras. De esta sólo nos concentrábamos en la prueba que estábamos realizando.

7.1.8. Posibles extensiones

Son varias las extensiones que se podrían realizar a estas placas pero teniendo en cuenta que debían ser lo más genéricas posibles como para probar nuevas funcionalidades o componentes, creemos que aportan mucho valor en la fase de testeo. Aunque usar componentes de montaje superficial nos permitirían tener una placa mas compacta.

7.2. Placa controladora de motores DC

funcion de una placa controladora de motorDC, porque fue armada, para que sirve, porque hay 2, porque no esta en una sola

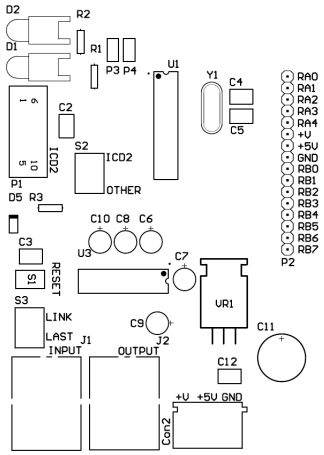

Figura 20: Máscara de componentes.

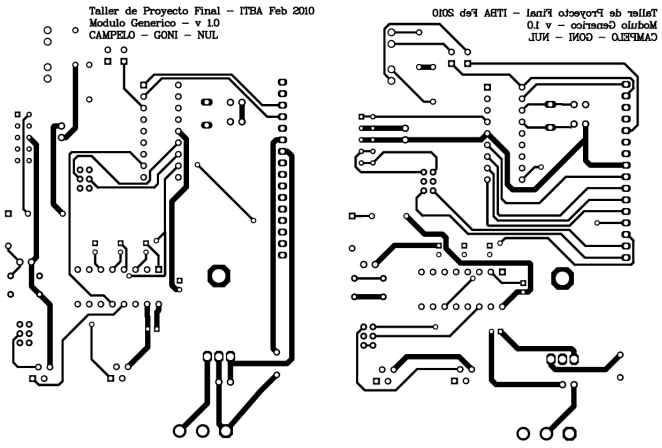

Figura 21: Capas superior e inferior de la placa.

7.2.1. Características principales

principio de funcionamiento, como logra controlar la velocidad, como logra ser parte de la cadena, como logra sensar el consumo, controlar el motor, puente H, diodos, leds, VREF

7.2.2. Módulo de comunicación

se explico en el modulo generico, se agregan los comandos especificos y se puede explicar como se obtiene la informacion para dar las respuestas

7.2.3. Alimentación de la placa

se explico en el modulo generico, tension para la alimentacion para los motores, necesidad de masa unica como referencia, consumo aproximado de los motores

7.2.4. Configuración

configuracion de la placa, leds, comunicacion, header de programacion, switch de seleccion de encoder

7.2.5. Esquemático

esquematicos de la placa

7.2.6. Circuito

circuito de la placa

7.2.7. Código básico

explicacion de lo minimo que deberia tener para ser parte de la cadena de comunicacion, sensado y control de la velocidad de los motores

7.2.8. Posibles extensiones

unificar en una placa el control de mas de un motor, pasar a montaje superficial los componentes, hacerla mas chica

7.3. Placas de sensado

funcion de una placa de sensado, porque fue armada, para que sirve, que tipo de sensores puedo conectar, cuales son las posibles configuraciones, diferencias, sensado de la bateria

7.3.1. Características principales

principio de funcionamiento, como logra tomar las muestras de los sensores, como logra ser parte de la cadena, seteo de los tipos de sensores

7.3.2. Módulo de comunicación

se explico en el modulo generico, se agregan los comandos especificos y se puede explicar como se obtiene la informacion para dar las respuestas

7.3.3. Alimentación de la placa

se explico en el modulo generico

7.3.4. Configuración

configuracion de la placa, comunicacion, header de programacion

7.3.5. Esquemático

esquematicos de la placa

7.3.6. Circuito

circuito de la placa

7.3.7. Código básico

explicacion de lo minimo que deberia tener para ser parte de la cadena de comunicacion y sensado de los distintos perifericos

7.3.8. Posibles extensiones

uso de componentes como resistencias variables para regular la alimentacion de los sensores de piso y resistencias pull-up, pasar a montaje superficial los componentes, hacerla mas chica

7.4. Placa controladora de servo motores

funcion de una placa controladora de servos, porque no fue armada, para que se penso, alguna otra opcion de conexion, pines libres

7.4.1. Características principales

principio de funcionamiento, como logra generar varios pwm por software, como logra ser parte de la cadena

7.4.2. Módulo de comunicación

se explico en el modulo generico, se agregan los comandos especificos y se puede explicar como se obtiene la informacion para dar las respuestas

7.4.3. Alimentación de la placa

se explico en el modulo generico, con modificaciones que permiten que circule una mayor cantidad de corriente para alimentar a los servos.

7.4.4. Configuración

configuracion de la placa, comunicacion, header de programacion

7.4.5. Esquemático

esquematicos de la placa

7.4.6. Circuito

circuito de la placa

7.4.7. Código básico

explicacion de lo minimo que deberia tener para ser parte de la cadena de comunicacion y control de los servos

7.4.8. Posibles extensiones

uso de componentes como resistencias variables para regular la alimentacion de los sensores de piso y resistencias pull-up, pasar a montaje superficial los componentes, hacerla mas chica

8. Armado del prototipo

8.1. Diseño

8.2. Características

con las ruedas de 10cm y teniendo en cuenta que el motor gira a unas 300 cuentas/segundo (usando un solo sensor en el encoder) llegamos a 50 cm/s de velocidad

8.3. Desarme

8.4. Costo y proveedores

A. Primer apéndice Hardware

protocolo de comunicacion.
conexionado y configuracion de la comunicacion.
circuitos de las placas.
codigo fuente de las controladoras.
costo del prototipo.

B. Segundo apéndice Hardware