

```
case MOTOR_BACKWARD:  
    output_bit(PIN_B5, 1);  
    output_bit(PIN_B4, 0);  
    dmr = MOTOR_BACKWARD;  
    break;  
case MOTOR_STOP:  
    output_bit(PIN_B5, 1);  
    output_bit(PIN_B4, 1);  
    dmr = MOTOR_STOP;  
    break;  
}  
break;
```

Rutinas de Servicio para las Interrupciones

para atender el desbordamiento de TIMER0

```
ctrl = 0;  
if (velocity_ctrl == VELOCITY_IDLE_TICS) {  
    hacemos la velocidad para su consulta  
    vels = vels_l_tics;  
    vels = get_timer1();  
    ponemos contadores.  
    vels = 0;  
    vels = 0;  
    velocity_ctrl = 0;  
}
```

```
// ISR para atender a la línea INT/RB0.  
#int_EXT  
EXT_isr() {  
    aux_l_tics++;  
}
```

```
void gobck() {  
    dir_motor(MOTOR_L, MOTOR_BACKWARD);  
    dir_motor(MOTOR_R, MOTOR_BACKWARD);  
}
```

OPTIMUS2003

PFC

mauro silvosa rivera
<http://optimus.meleisland.net>

Este proyecto es un trabajo de investigación realizado en el marco de un curso de posgrado en la Universidad de Chile. El contenido de este documento es de carácter académico y no debe ser utilizado para fines comerciales. Todos los derechos reservados. Última actualización: 2003-10-15.





Diseño y construcción de la base de un robot móvil autónomo gobernada por microcontrolador

Mauro Silvosa Rivera

Tutor: Carlos Vázquez Regueiro





OBJETIVOS:

Diseño y construcción de una base de un robot móvil autónomo, gobernada por microcontrolador:

- Sencilla
- Fácil de implementar
- De bajo coste
- Con buenas prestaciones
- Versátil





CONTENIDO DE LA EXPOSICIÓN:

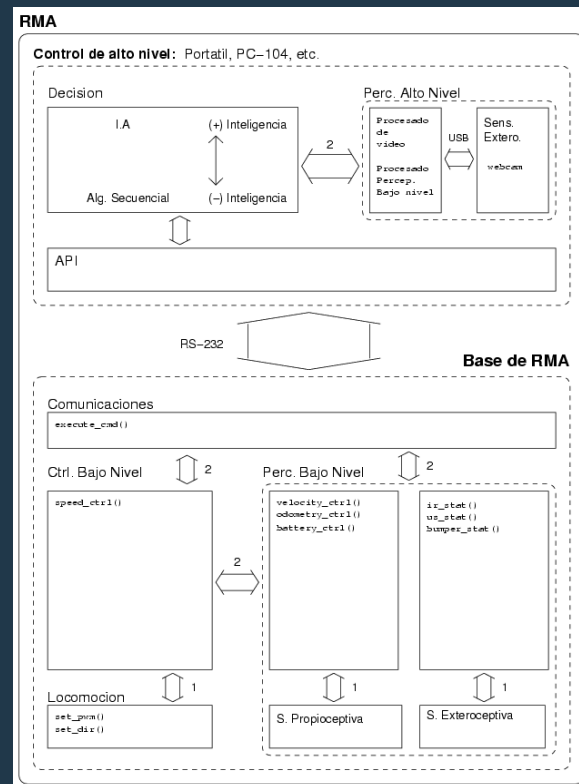
1. Introducción
2. Diseño de la plataforma
3. Diseño del control
4. Diseño de la API
5. Validación experimental
6. Conclusiones y trabajo futuro





1. Introducción

Estructura hardware / software de la base de un robot móvil autónomo.





2. Diseño de la plataforma

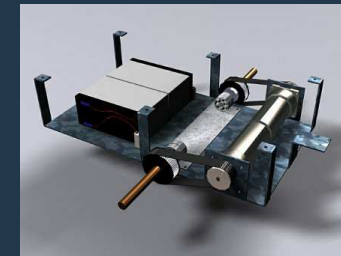
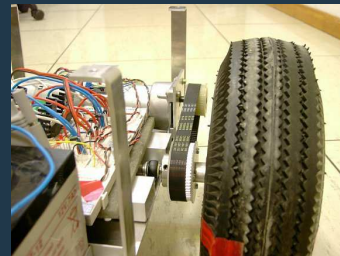
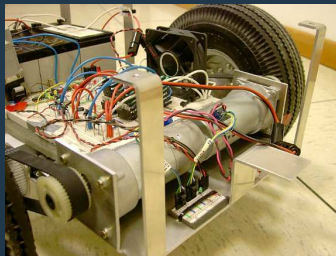
- Etapa mecánica
- Electrónica de potencia y control de motores
- Control por microcontrolador y comunicaciones
- Características finales de la base de robot móvil





2.1 Etapa mecánica

- Motores DC y encoders ópticos
- Transmisión
- Ruedas
- Chasis





2.2 Electrónica de potencia y control de motores

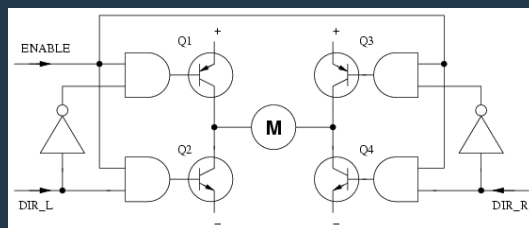
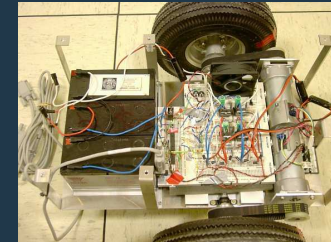
- Baterías

- Motores: 24 V, 7 Ah

- Control y comunicaciones: 12 V, 7 Ah (regulada a 5 V, 1 A)

- Control de motores

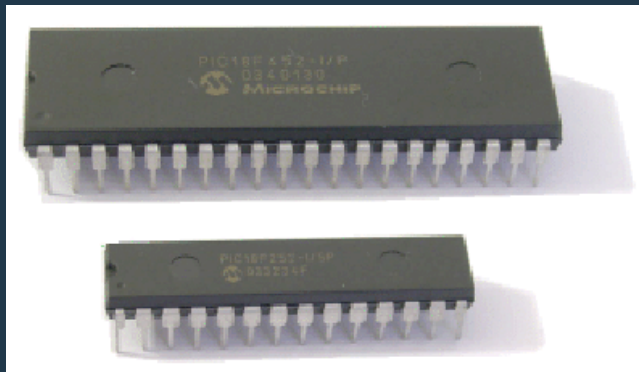
- Mediante puentes-H: Interfaz entre el microcontrolador (poca tensión y bajas corrientes) y los motores (alta tensión y grandes corrientes).





2.3 Control por microcontrolador y comunicaciones (I)

- Motivación: integrar en un único C.I. la electrónica necesaria para el control y la percepción de bajo nivel de la base y la gestión de las comunicaciones
- Comunicaciones: línea serie RS-232 a 115000 bps





2.3 Control por microcontrolador y comunicaciones (II)

Características principales del PIC18F252:

- Módulo de comunicaciones serie RS-232
- 4 contadores: 2 para el uso exclusivo de los encoders
- Múltiples puertos de E/S: control de motores
- 2 módulos PWM: control de velocidad
- Conversores A/D: monitorizar la carga de las baterías
- Potente: hasta 10 MIPS con CPI=1 (CPI=2 en saltos)
- Programable en C

Proporciona los recursos suficientes para ejecutar todas las tareas necesarias para el control de la base de robot móvil





2.4 Características de la base de robot móvil

- Dimensiones: 540 x 520 x 265 mm (Largo x Ancho x Alto)
- Peso: 15 Kg
- Autonomía: 4-7 h
- Diámetro de las ruedas: 260 mm
- Distancia al suelo (vuelo): 85 mm
- Velocidad máxima y mínima: 2,8 m/s - 300 mm/s
- Precio: 495,68 €





3. Diseño del control

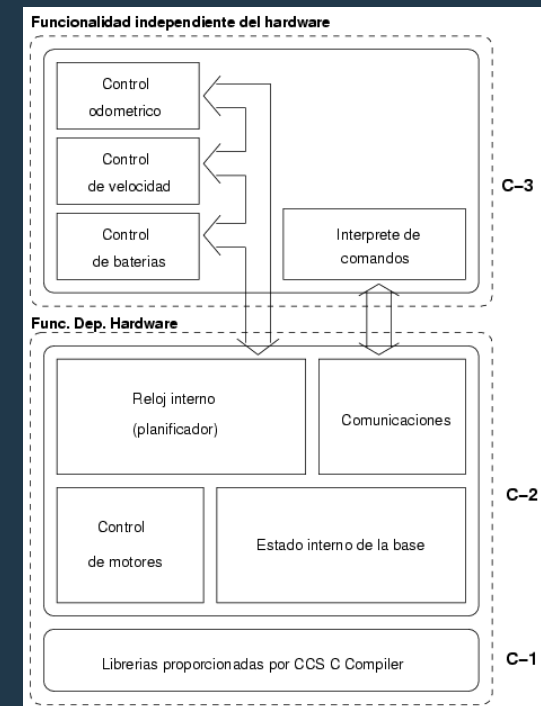
- Estructura de capas del software de control
- Funcionalidad dependiente del hardware
 - Cálculo de la velocidad de las ruedas
 - Cálculo del nivel de carga de las baterías
 - Recepción de datos por la línea serie RS-232
- Funcionalidad independiente del hardware
 - Control de la velocidad
 - Control del nivel de carga de las baterías
 - Cálculo de la odometría
 - Intérprete de comandos
- Planificador de tareas





3.1 Estructura de capas del software de control

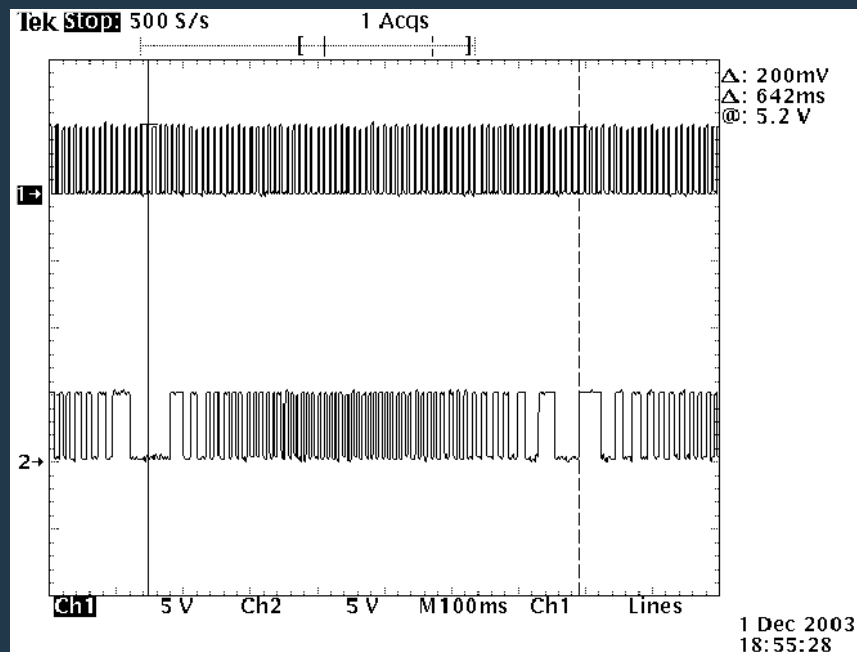
- Ventajas: portabilidad, inteligibilidad, legibilidad, fiabilidad y rapidez de desarrollo.
- Inconvenientes: mayor consumo de memoria.
 - C-1: Funciones librería del compilador. Dependencia directa del hardware.
 - C-2: Funcionalidad dependiente del compilador y del hardware.
 - C-3: Algoritmos de control e intérprete de comandos. Independientes de hardware y compilador.





3.2.1 Cálculo de la velocidad de las ruedas

- Contar los pulsos generados por cada encoder en un intervalo de tiempo regular y almacenar en vector de estado



```
#int_TIMER3
TIMER3_isr() {

    set_timer3(T3_INIT_VALUE);
    time_inc_internal_time();

    // CALCULAMOS LA VELOCIDAD.
    velocity_ctrl++;
    if (velocity_ctrl == VEL_WORKING_CICLE) {

        vel_set_real_tics(MOTOR_LEFT, get_timer0());
        vel_set_real_tics(MOTOR_RIGHT, get_timer1());

        // Reiniciamos contadores.
        set_timer1(0);
        set_timer0(0);
        velocity_ctrl = 0;
    }
}
```





3.2.2 Cálculo del nivel de carga de las baterías

- Leer las entradas de los conversores A/D conectados a las baterías y almacenar en el vector de estado

```
#int_TIMER3
TIMER3_isr() {

    set_timer3(T3_INIT_VALUE);
    time_inc_internal_time();

    // CONTROLAMOS EL NIVEL DE CARGA ACTUAL DE LAS BATERÍAS.
    battery_ctrl++;
    if (battery_ctrl == BATTERY_CTRL_WORKING_CICLE) {

        set_adc_channel(0);
        pwr_set_battery_level(LOGIC_BATT, (unsigned int16) read_adc() * LOGIC_TOVOLTS);

        do_battery_ctrl();
        battery_ctrl = 0;
    }
}
```





3.2.3 Recepción datos por la línea serie RS-232

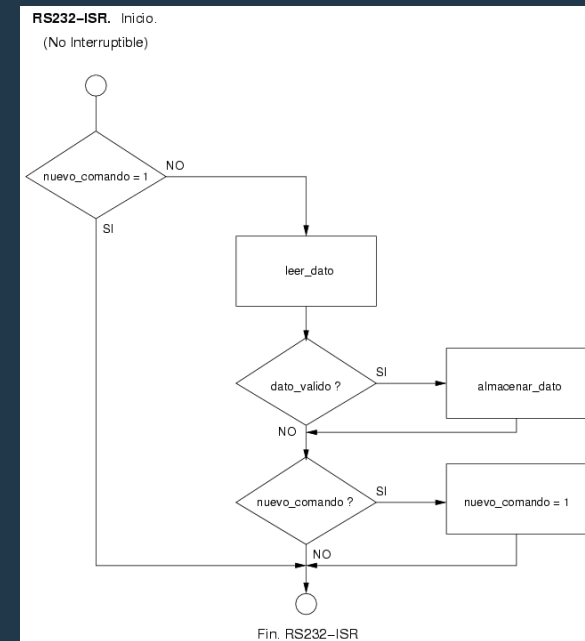
- Con cada interrupción hardware de la línea RS-232 se almacena el dato recibido. Activa un flag cuando recibe un "fin de comando".

```
#int_RDA
RDA_isr() {
    char aux;

    aux = getc();
    if (buffer_current_pos < BUFFER_SIZE && !new_command) {
        if (!(isalnum(aux) || aux==CR || aux==LF || aux==SP || aux=='-' ||
            aux=='_' || aux=='.'))
            return;

        if (aux == CR || aux == LF) {
            buffer[buffer_current_pos] = ' ';
            buffer[buffer_current_pos+1] = '\0';
            new_command = 1;
            buffer_current_pos = 0;
            return;
        }

        buffer[buffer_current_pos] = aux;
        buffer_current_pos++;
    }
}
```





3.3.1 Control de la velocidad (I)

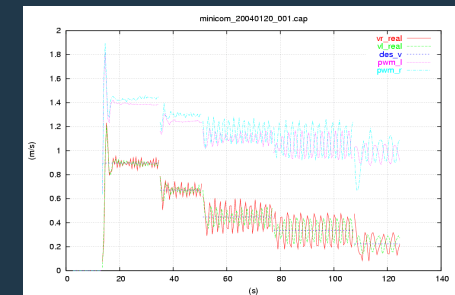
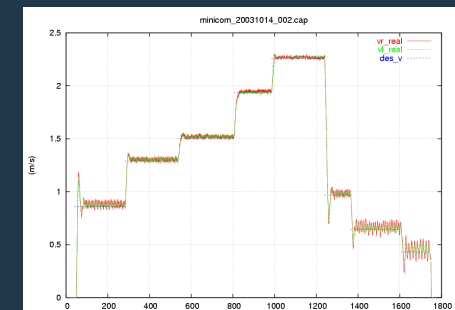
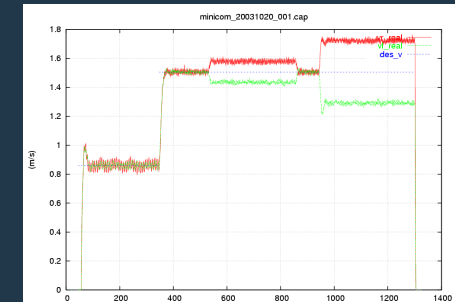
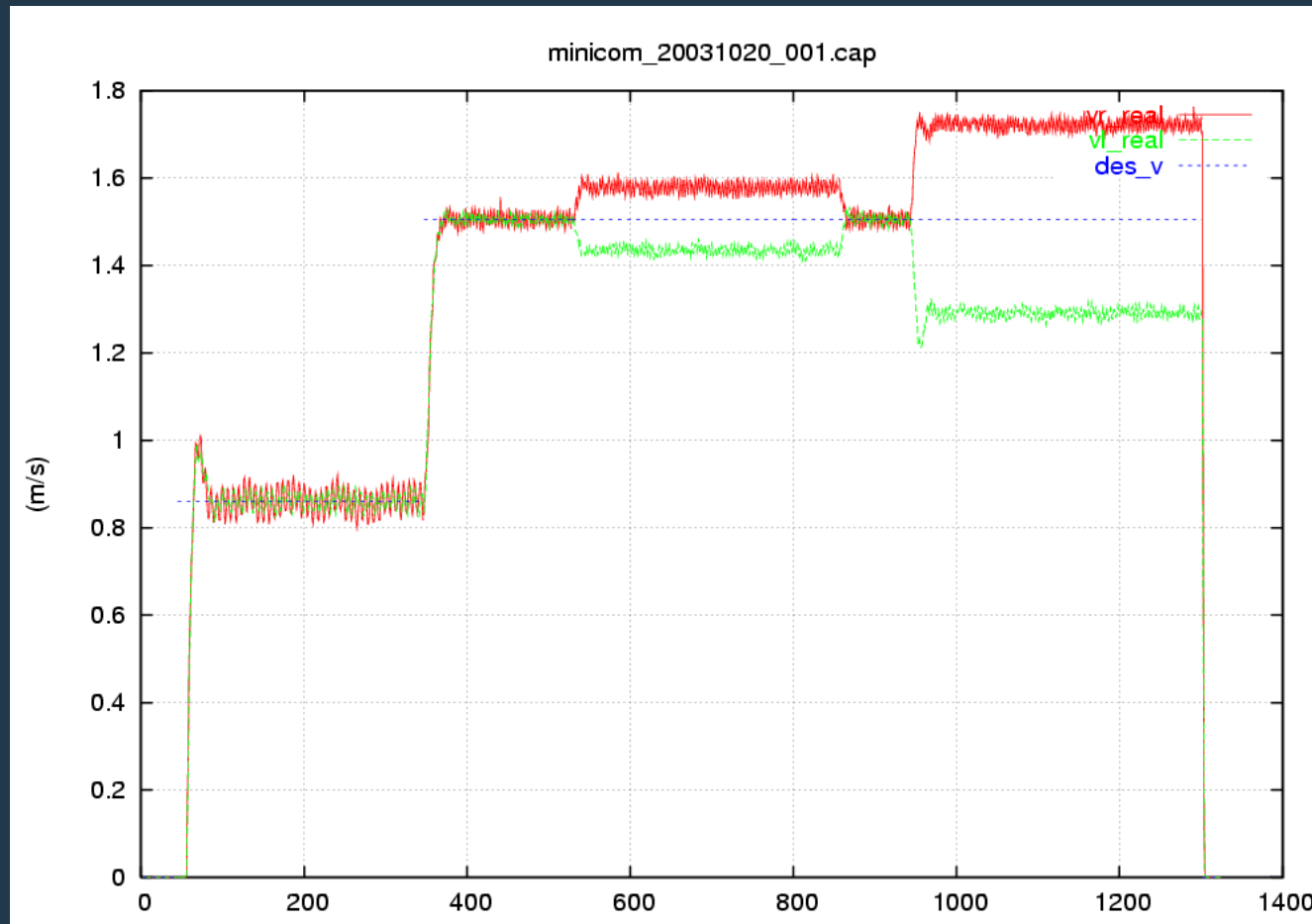
- Modificación del ciclo de trabajo de las señales PWM en función de la diferencia entre la velocidad real y la deseada (control PI).
- Los giros se realizan incluyendo un desplazamiento o "bias".

```
vl_real = vel_get_real_tics(MOTOR_LEFT);  
vr_real = vel_get_real_tics(MOTOR_RIGHT);  
  
bias_error = (bias_error + (vel_get_des_bias() - (vr_real - vl_real))) / _ki;  
  
vl_error = vel_get_des_vel() - vel_get_des_bias()/2 - vl_real - bias_error;  
vr_error = vel_get_des_vel() + vel_get_des_bias()/2 - vr_real + bias_error;  
  
vl_error = vl_error / get_kp(vl_error);  
vr_error = vr_error / get_kp(vr_error);  
  
pwm_set_duty_cicle(MOTOR_LEFT, pwm_get_duty_cicle(MOTOR_LEFT) + vl_error);  
pwm_set_duty_cicle(MOTOR_RIGHT, pwm_get_duty_cicle(MOTOR_RIGHT) + vr_error);
```





3.3.1 Control de la velocidad (y II)





3.3.2 Cálculo de la odometría (I)

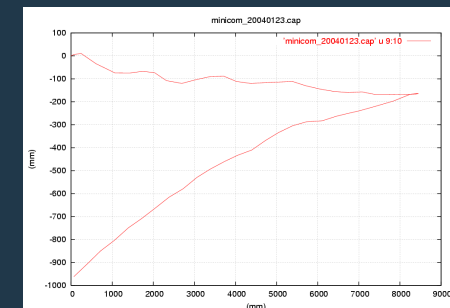
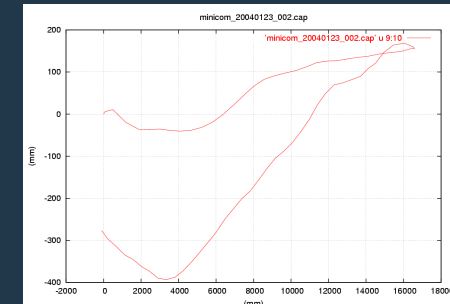
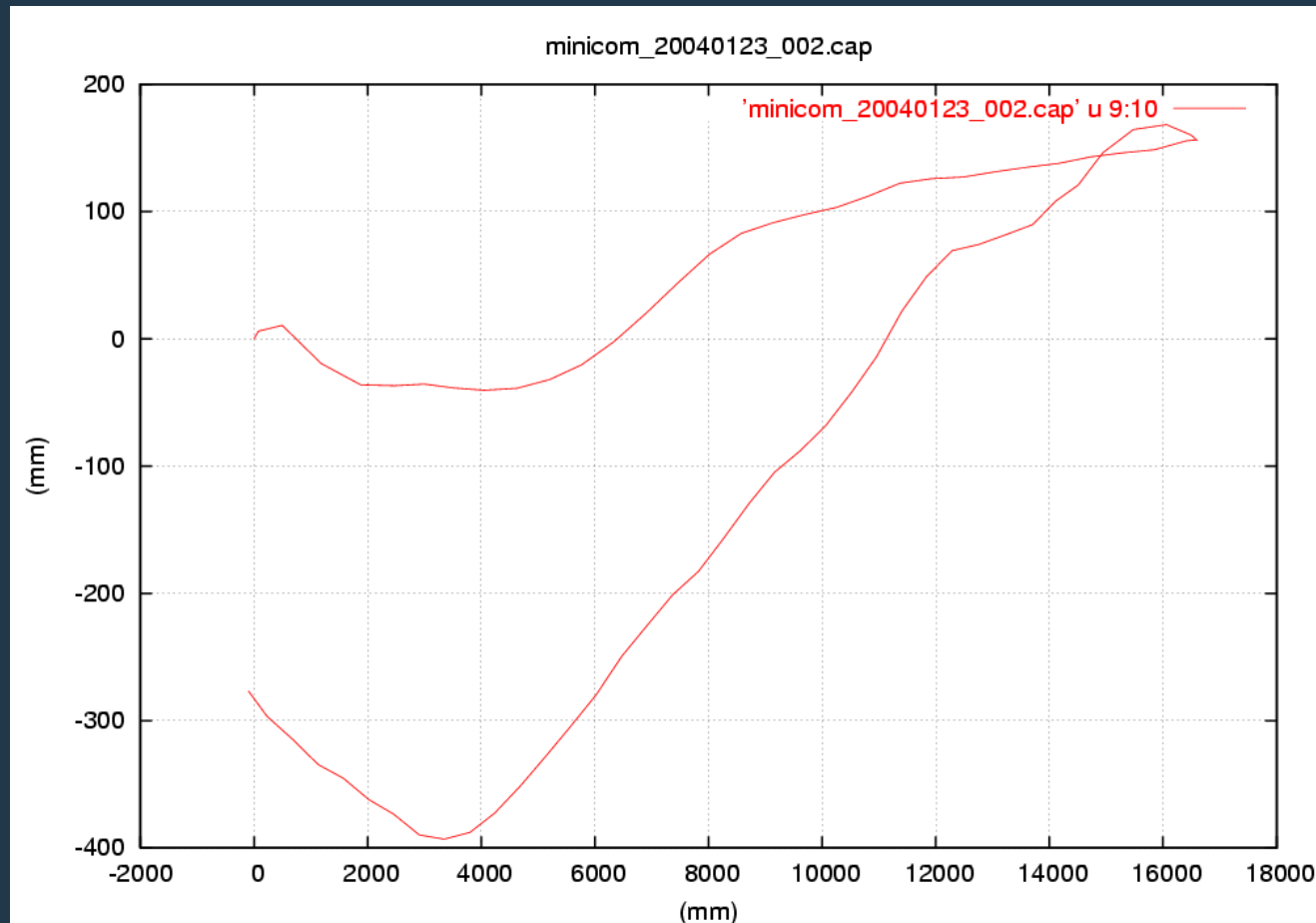
- Con cada actualización de velocidad se recalcula la posición de la base, en función del desplazamiento lineal de cada rueda (Dl y Dr)

```
t1 = (float) vel_get_real_tics(MOTOR_LEFT) * ODOMETRY_F;  
tr = (float) vel_get_real_tics(MOTOR_RIGHT) * ODOMETRY_F;  
tc = (tr + t1) / 2;  
dth = (tr - t1) / ODOMETRY_B;  
  
odo_set_th(odo_get_th() + dth);  
while (odo_get_th() >= DOS_PI) odo_set_th(odo_get_th() - DOS_PI);  
  
odo_set_x(odo_get_x() + tc * cos(odo_get_th()));  
odo_set_y(odo_get_y() + tc * sin(odo_get_th()));
```





3.3.2 Cálculo de la odometría (y II)

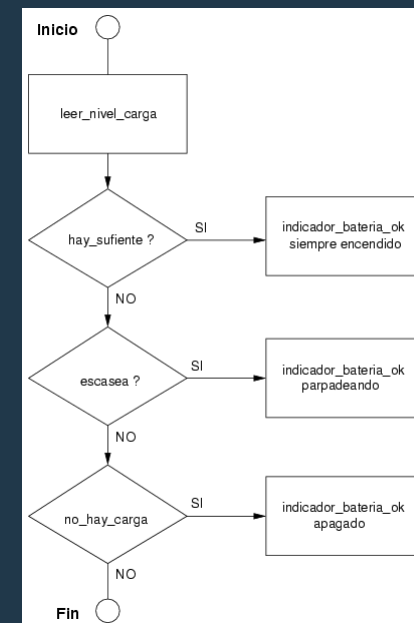




3.3.3 Control del nivel de carga de las baterías

- Monitorización regular del valor de tensión de las baterías. Al bajar de un cierto nivel un led comienza a parpadear. Al agotarse, la base se detiene y entra en modo de bajo consumo (sleep).

```
if (pwr_get_battery_level(LOGIC_BATT) > BLINK_START)
    pwr_set_battery_led(LOGIC_BATT, BATT_LED_ON);
else {
    if (pwr_get_battery_level(LOGIC_BATT) > LOW_BATTERY &&
        pwr_get_battery_level(LOGIC_BATT) < BLINK_START) {
        if (!blink_led) {
            pwr_set_battery_led(LOGIC_BATT, BATT_LED_ON);
            blink_led = 1;
        }
    }
    else {
        pwr_set_battery_led(LOGIC_BATT, BATT_LED_OFF);
        blink_led = 0;
    }
}
else {
    pwr_set_battery_led(LOGIC_BATT, BATT_LED_OFF);
    printf("# ERR. Optimus Panic! Out of battery");
    sleep();
}
}
```



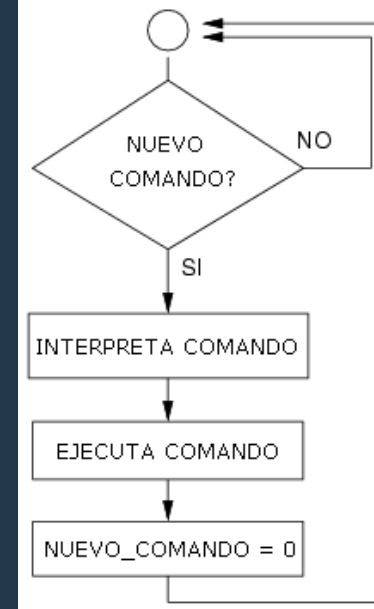


3.3.4 Intérprete de comandos

- Se ejecuta un comando si:
 1. La CPU no está ocupada en tareas prioritarias de control.
 2. Se ha recibido previamente la señal de "fin de comando".

```
void execute_cmd(char *cmd_line) {  
    ...  
    ptr = my_strtok(cmd_line, ' ');  
    switch (identify_cmd(ptr)) {  
        case COD_STAT:  
            _get_status();  
            break;  
        case COD_SETVEL:  
            vel_set_desired(atol(my_strtok(0, ' ')),  
                           atol(my_strtok(0, ' ')));  
            printf(STR_OK);  
            break;  
        ...  
    }  
    ...  
}
```

Programa principal. Inicio.



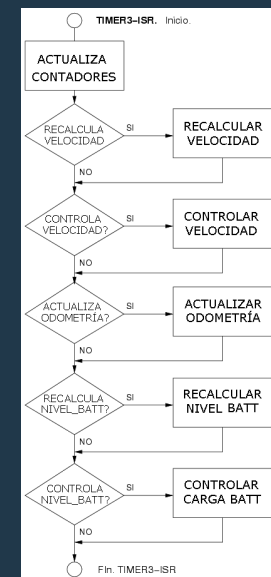


3.4 Planificador de tareas

- Implementación:

1. Configurar la interrupción por desbordamiento de TIMER3, para que interrumpa a la CPU cada 100 ms.
2. En la rutina de servicio de la interrupción se determina qué tareas se deben ejecutar y se ejecutan.

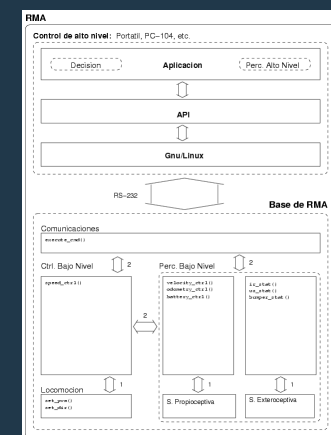
```
#int_TIMER3
TIMER3_isr() {
    ...
    velocity_ctrl++;
    if (velocity_ctrl == VEL_WORKING_CICLE) { // Recalcula velocidad
        check_velocity();
        velocity_ctrl = 0;
    }
    speed_ctrl++;
    if (speed_ctrl == SPEED_CTRL_WORKING_CICLE) { // Controla velocidad
        do_speed_control(_do_stop());
        speed_ctrl = 0;
    }
    odo_ctrl++;
    if (odo_ctrl == ODOMETRY_CTRL_WORKING_CICLE) { // Actualiza odometría
        do_odometry_ctrl();
        odo_ctrl = 0;
    }
    ...
}
```





4. Diseño de la API

- Motivación: proporcionar al programador del control y percepción de alto nivel una interfaz software que le permita aprovechar toda la funcionalidad de la base al tiempo que le oculta los detalles de implementación del control y el protocolo de comunicaciones.
- Ventajas: las derivadas del diseño por capas: legibilidad, fiabilidad, portabilidad e independencia (del hardware).
- Tipos y listado de las funciones más importantes.





4.1 Listado de las funciones más importantes

- 3 tipos: para ocultar comunicaciones, control y estado de la base.

Comunicaciones	Estado	Control
<code>int connect()</code> <code>int disconnect()</code>	<code>int read_status()</code> <code>int get_internal_time()</code> <code>int get_left_vel()</code> <code>float get_pos_x()</code> <code>float get_logic_batt()</code>	<code>void set_pose()</code> <code>void set_time()</code> <code>void set_vel()</code> <code>void_stop()</code> <code>void_free_stop()</code>





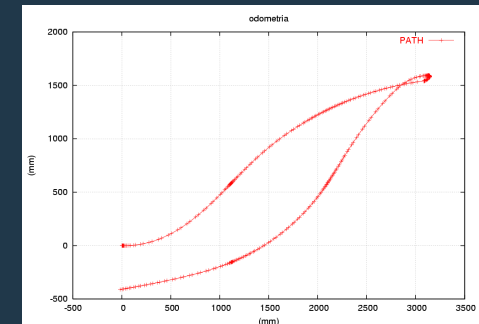
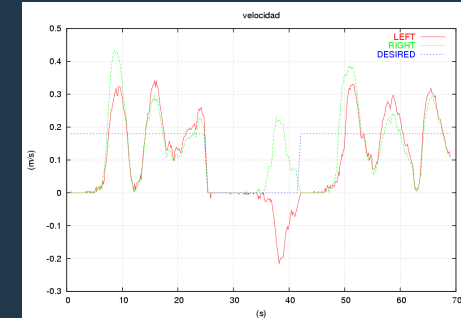
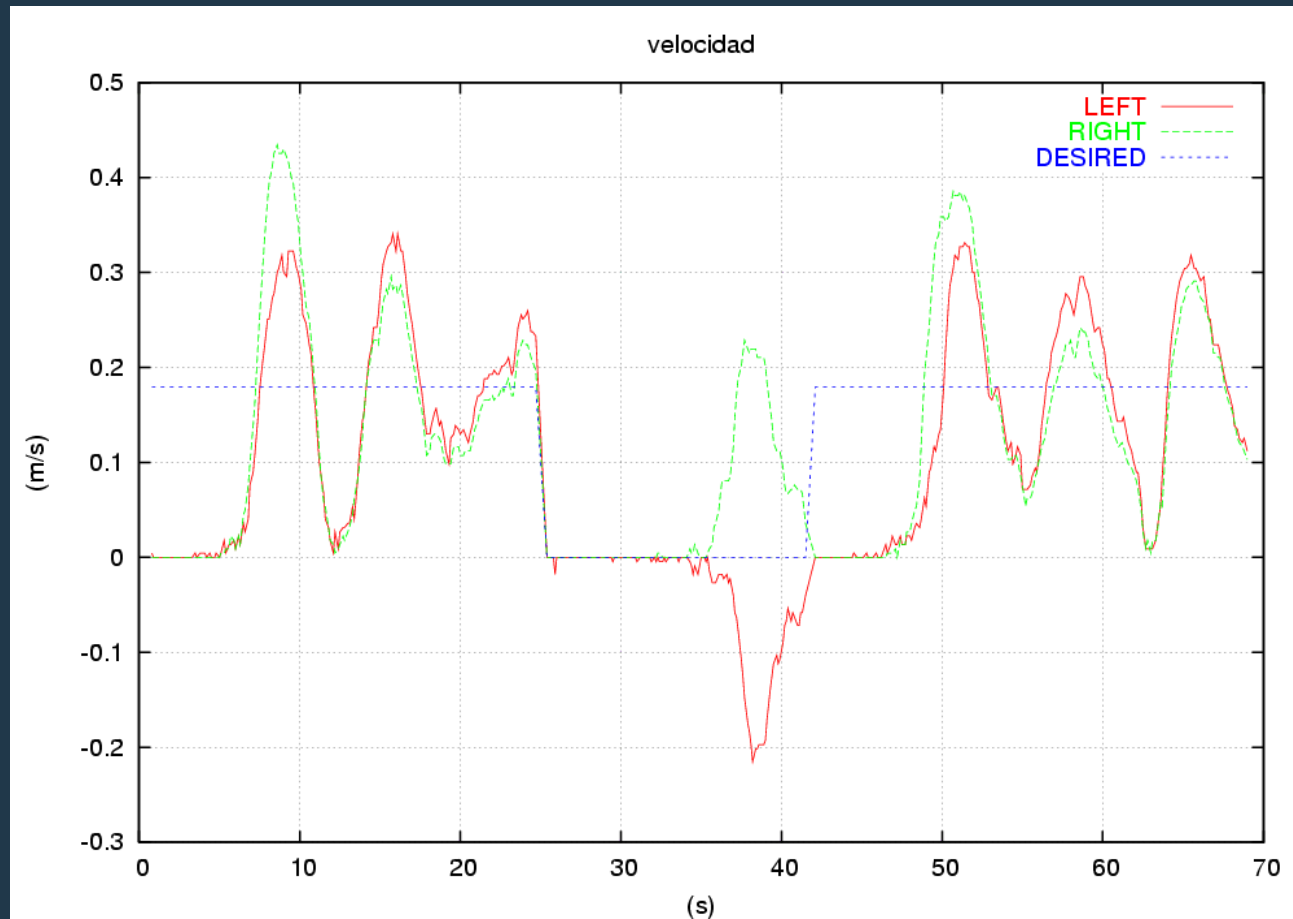
5. Validación experimental

- Vídeo ejemplo
- Resultados
- Discusión





5.1 Resultados





5.2 Discusión

- A la vista de los resultados podemos concluir que:
 1. El control no consigue mantener estable la dirección de la base
 2. Tampoco consigue mantener velocidades pequeñas constantes
 3. El tiempo de respuesta es muy grande
- Las razones de estos problemas son fundamentalmente mecánicas: diseño del chasis, tamaño de las ruedas, transmisión, rodamientos, motores, precisión encoders, ...
- Subsanaos estos errores de diseño hay motivos suficientes para pensar que el control implementado es estable





6. Conclusiones (I)

- Se ha diseñado y construido una base de robot móvil de bajo coste a partir de sus componentes básicos: motores, chasis, ...
- La base implementa todas las funciones de bajo nivel de un robot móvil autónomo comercial: percepción, control y comunicaciones:
 - odometría, carga de las baterías, ...
 - control en velocidad lineal y angular
 - comunicaciones vía puerto serie
- Para simplificar la electrónica el control se ha basado en un microcontrolador (μC): el PIC18F252
- Los recursos de este μC nos permiten controlar todos los elementos necesarios de la base





6. Conclusiones (y II)

- La programación del microcontrolador se ha estructurado en capas y se ha codificado en lenguaje C:
 - independencia del hardware
 - facilita la migración a otros microcontroladores
- Todas las tareas de control (odometría, control de velocidad, ...) se logran ejecutar en tiempo real
- También se ha desarrollado una sencilla interfaz para las comunicaciones y el control de la base
- La odometría implementada posee una buena precisión aunque el control de la velocidad es algo deficiente





6. Trabajo futuro

- Mejoras mecánicas: desplazar el centro de gravedad del vehículo hacia el eje motriz, aumentar factores de reducción, ...
- Trabajar con encoders de cuadratura para aumentar la precisión del control
- Mejora del control usando la velocidad media y no la instantánea a velocidades bajas
- Proporcionar la capacidad de añadir sensores a la base
- Definir una arquitectura de múltiples μC 's (bus I2C) dedicados a las distintas tareas de control





<http://optimus.meleeisland.net>

