Apéndice A

Arquitectura Antigua del ROMEO-4R

A. ARQUITECTURA ANTIGUA DEL ROMEO-4R

En el vehículo autónomo ROMEO-4R se implementó un programa propio para su funcionamiento, de forma que todo programador que quisiera modificarlo, añadiendo por ejemplo nuevo código, debería conocer antes el programa completo, teniendo por tanto poca o nula modulabilidad.

En la antigua arquitectura se encontraban una serie de capas software, cada una con una interfaz bien definida, que en su último nivel ofrecía todo un conjunto de métodos o funciones para obtener información de los diferentes sensores y, a la vez, poder actuar sobre la dirección y velocidad del vehículo.

Además de estas funciones, se contaba con un programa completo y único, que dotaba de funcionalidad al vehículo.

En los próximos apartados se dará una visión general de la antigua arquitectura del ROMEO-4R.

A.1. Niveles de abstracción

El software ha sido organizado en diferentes capas o niveles, dependiendo del grado de abstracción, es decir, de la proximidad a la que se encuentra de un dispositivo. Cuanto más próximo está a un dispositivo, desde el punto de vista del software, más directa es la comunicación con el mismo y mayor es la dependencia del software con su implementación electrónica. Por tanto, cuanto más bajo es el nivel de abstracción, mayor es el conocimiento que se debe tener del hardware con el que se trabaja.

En el nivel más bajo (menos abstracto), se encuentra el vehículo en sí, con todo el hardware que lo compone. Para trabajar en esta capa es necesario poseer un gran conocimiento de la electrónica que lleva incorporada el ROMEO-4R.

El siguiente nivel, el de *driver*, constituye el primer nivel de software que se implementa. Para ello es necesario conocer el interfaz que presenta cada dispositivo (sensor o actuador) al PC al que se conecta, es decir, la forma en que se comunica con el mismo. El software implementado en esta capa permite la comunicación a bajo nivel del PC con cada dispositivo, es decir, enviar y recibir información hacia y desde cada dispositivo. Para trabajar en este nivel, sería necesario conocer en profundidad como un dispositivo necesita recibir la información, y como se le va a enviar al PC.

El nivel inmediatamente superior al de driver será el de *dispositivo*, donde cada dispositivo dispondrá de un completo interfaz que incluye todas las funciones y operaciones más importantes que se van a poder llevar a cabo con cada uno de ellos.

El penúltimo nivel será el de *aplicación*, y se encuentra íntimamente relacionado con el uso o aplicación concreta a la que se van a destinar los dispositivos, incluyendo todas las funciones de alto nivel que serán útiles para la aplicación que se desea implementar. Estas funciones dan lugar a un nuevo interfaz, que será el que deba conocer el usuario de alto nivel para poder probar sus algoritmos de control.

Por último, en el mayor nivel de abstracción, tendremos el programa de usuario, donde se ejecutará el algoritmo de control. Aquí, el usuario desarrollará algún tipo de aplicación haciendo uso de la interfaz que se ha diseñado en el nivel de aplicación.

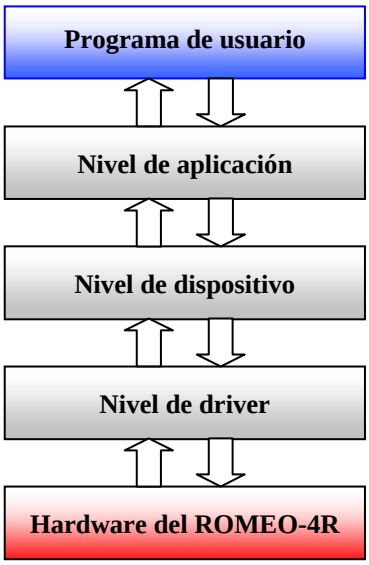

Figura A.1-1: Niveles de abstracción software

A.2. El programa *main*

Una vez conocidos los cimientos en los que se basaba la anterior arquitectura del ROMEO-4R, se presentará una vista generalizada del programa *main* que controlaba el vehículo anteriormente.

Dicho programa se encuentra estructurado en diferentes bloques funcionales, cada uno con una misión bien definida. La idea consiste en que un usuario de alto nivel necesite un conocimiento mínimo del software del ROMEO-4R para poder desarrollar su propia aplicación, estando fuertemente delimitado su campo de acción de forma que se reduzca la posibilidad de que provoque errores en la estructura general del programa.

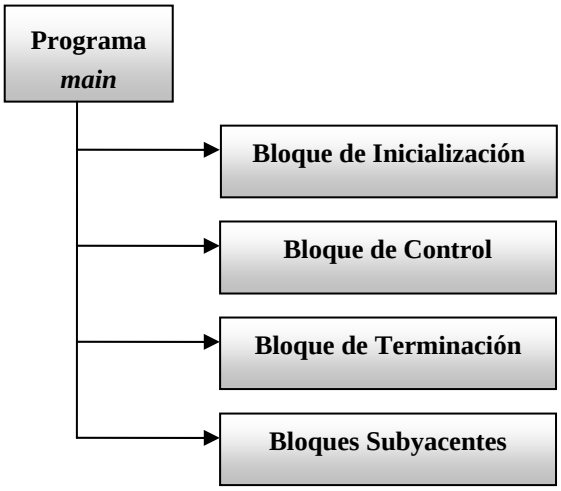

Figura A.2-1: Diagrama de bloques del programa completo

Los bloques funcionales, antes mencionados, son los siguientes:

- ***Bloque de inicialización:*** se encarga de llevar a cabo toda la parte de configuración inicial del programa, entre los que se incluyen ficheros de cabecera, definición de constantes, declaración y definición de funciones, variables locales y globales tanto del programa como del algoritmo de control, etc.

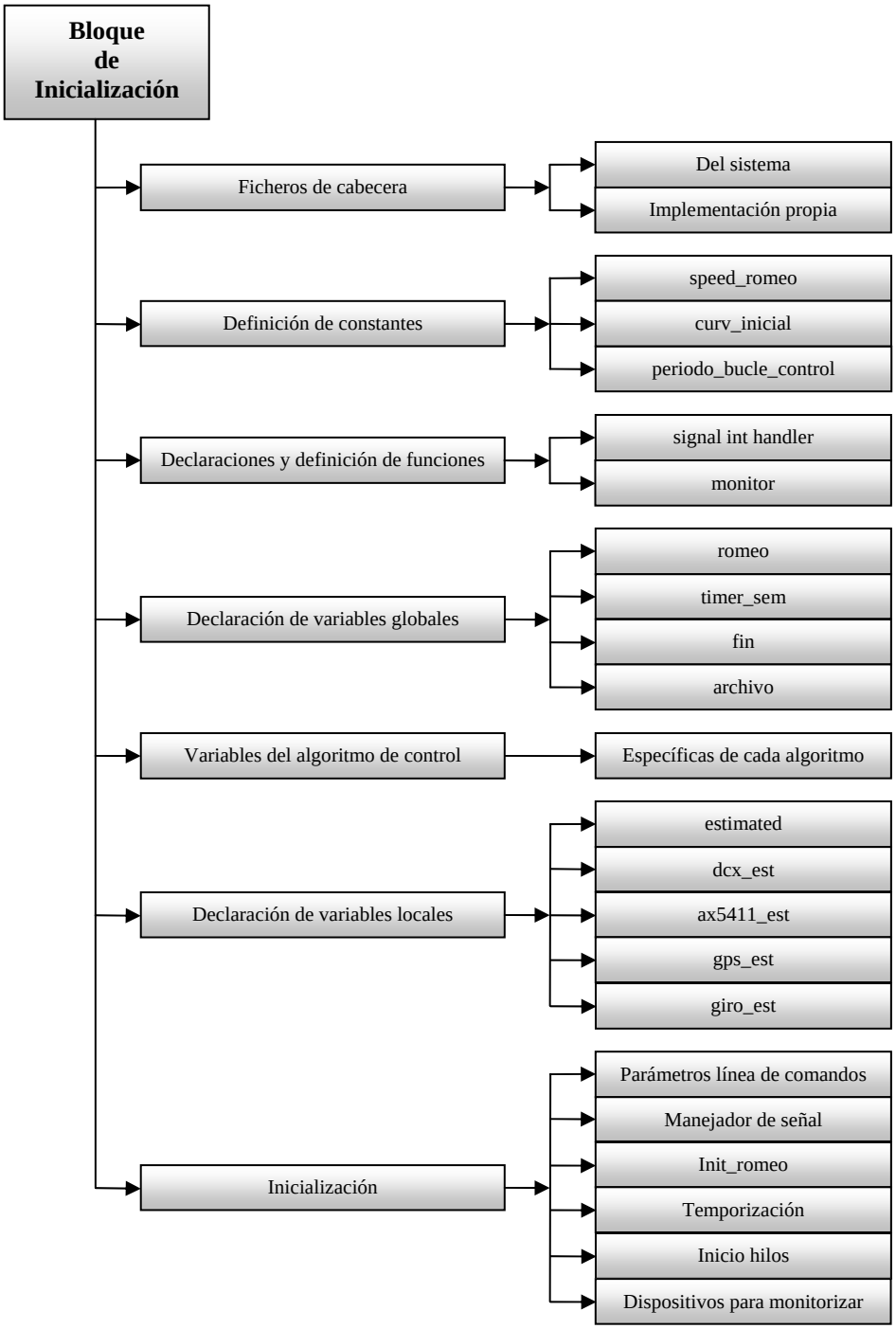

Figura A.2-2: Bloque de inicialización

- ***Bloque de control:*** se encarga de toda la actividad del control automático. Su parte principal será un algoritmo de control, implementado por algún usuario, que será el que tome el control del vehículo para llevar acabo alguna aplicación concreta. Exceptuando una primera parte, la de inicialización del algoritmo, el resto se encuentra inmerso en un bucle que se ejecuta con una periocidad bien temporizada. Este bucle, durante cada iteración, comprobará la existencia de nuevos datos de los dispositivos, procesará la información, y generará nuevas consignas de control, que serán aplicadas al final del mismo. Por último, existe un proceso de monitorización de datos.

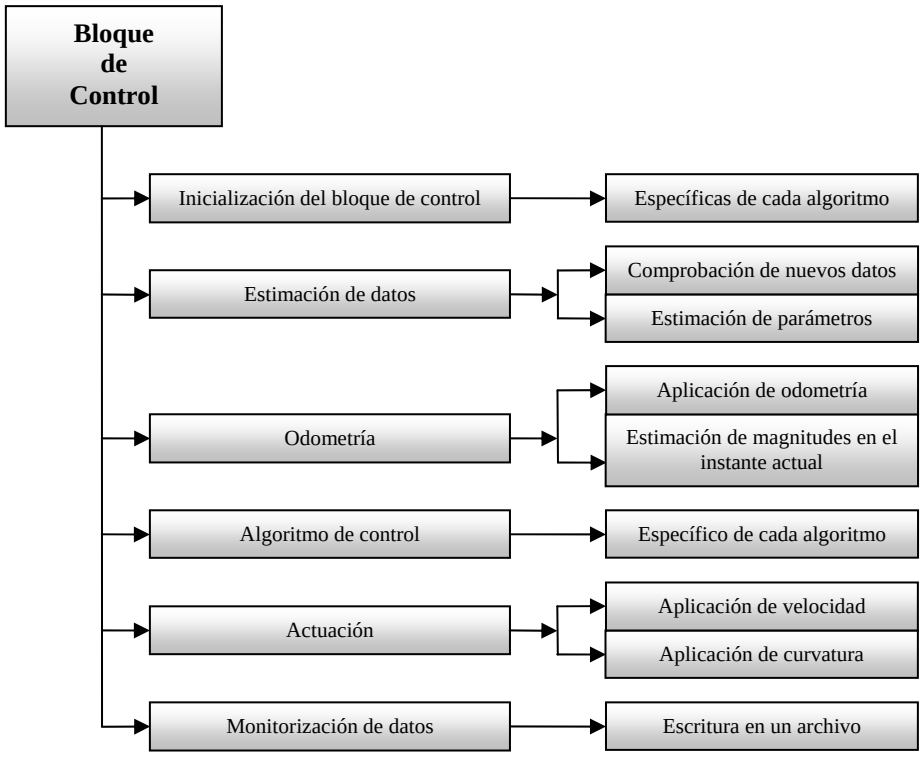

Figura A.2-3: Bloque de control

- ***Bloque de terminación:*** se encarga de llevar al sistema a un estado seguro para finalizar satisfactoriamente la ejecución del programa. Este bloque se ejecutará cuando el algoritmo de control lo indique, o cuando el usuario lo desee.

- ***Bloques subyacentes:*** son hilos que se ejecutan simultáneamente con el bucle de control, y realizan diversas tareas en paralelo con el resto del programa. Estas tareas serán, principalmente, la de controlar cada uno de los dispositivos, recibiendo y procesando la información que se recibe de los mismos. Una vez procesada, deben encargarse de poner la información a disposición del bucle principal de control, avisando convenientemente de la aparición de nuevos datos.

A.3. Código del programa *main*

La estructura básica del código del programa principal (*main*), sobre el que se basaba la arquitectura antigua del ROMEO-4R, donde se especifican los bloques estudiados en el apartado anterior, en el cual el usuario debe implementar su aplicación, es el siguiente:

```
/*****************************************************************************
*  MAIN.CPP
*
* Función main del programa para el control del vehículo autónomo ROMEO-4R
*
* AUTOR: Rafael Martín de Agar Tirado
*
* ULTIMA MODIFICACION: 03-II-2005
*
*****************************************************************************/

/*****************************************************************************
*          A R C H I V O S   D E   C A B E C E R A
*****************************************************************************/

/************************
* D E L   S I S T E M A *
************************/

// Entrada y salida básica
#include <iostream.h>
#include <stdio.h>

// Para atof()
#include <stdlib.h>

// Para manipulación de archivos
#include <fstream.h>
#include <iomanip.h>

// Para fork()
#include <unistd.h>

// Para bzero()
#include <string.h>

// Para floor()
#include <math.h>

// Para manejo de señales
#include <signal.h>

// Para funciones de tiempo (ponerle el nombre al archivo de LOGS)
#include <time.h>

/************
* M I S  .H *
************/

// Definición de la clase Romeo
#include "romeo.hpp"

// Para los hilos
#include "./hilos/hilo_gps.hpp"
#include "./hilos/hilo_dcx.hpp"
#include "./hilos/hilo_ax5411.hpp"
#include "./hilos/hilo_gyro.hpp"
#include "./hilos/hilo_laser.hpp"
#include "./hilos/hilo_estimacion.hpp"
#include "./hilos/hilo_teleop.hpp"

// Para el temporizador del bucle principal
#include "./timers/timers.hpp"
#include "./timers/ipc.h"
```

```
// Para funciones implementadas fuera de ninguna clase y que se van a
// usar en el main
#include "./funciones_main.hpp"
#include "./tipos_romeo.h"

/*****************
* APLICACION  .H *
*****************/

/*****************************************************************************
*         F I N    A R C H I V O S    D E    C A B E C E R A
*****************************************************************************/

/*****************************************************************************
*           D E F I N I C I O N    D E    C O N S T A N T E S
*****************************************************************************/

// Tamaño del mensaje que se envía funcionando como cliente-servidor
//#define TAM_MENSAJE_CLIENTE 80

/*****************************************************************************
*        F I N    D E F I N I C I O N    D E    C O N S T A N T E S
*****************************************************************************/

/*****************************************************************************
*                V A R I A B L E S     G L O B A L E S
*****************************************************************************/

// El objeto "romeo" debe ser global para permitir el acceso al mismo desde
// los hilos
Romeo romeo;

// Semáforo del timer del bucle principal: debe ser global para que el hilo
// del timer pueda modificar su valor
int timer_sem;

// Variable que marca la salida del bucle de control: debe ser global para
// que pueda ser modificada por la función manejadora de la señal SIGINT
int fin = 0;

// Variable que indica si se desea realizar una simulación o una prueba real
int simulate = 0;

/*****************************************************************************
*            F I N    V A R I A B L E S     G L O B A L E S
*****************************************************************************/

/*****************************************************************************
*              A L G O R I T M O     D E     C O N T R O L
*****************************************************************************/



/*****************************************************************************
*           F I N    A L G O R I T M O     D E     C O N T R O L
*****************************************************************************/

/*****************************************************************************
*                           M A I N
*****************************************************************************/

main(int argc, char *argv[])
{

/*****************************************************************************
*                 V A R I A B L E S    L O C A L E S
*****************************************************************************/

// Es una buena práctica de programación declarar como externas aquellas
// variables no definidas dentro de una función
extern Romeo romeo;
extern int timer_sem;
extern int fin;

// variable para ir leyendo las medidas del laser
LASER_DATA laser_data;
bzero(&laser_data,sizeof(LASER_DATA));
```

```
// Estructura para almacenar la información leída del fichero de
// configuración del main
struct conf_struct conf_info;

// Manejo de señales
struct sigaction sa;

// Estructuras de datos para el bucle de control
// Estimación de los parámetros de interés
ESTIMATED estimated;
bzero(&estimated, sizeof(ESTIMATED));

// Algunas variables generales necesarias
double time_act = 0;
double ref_speed = 0, ref_curv = 0;

// Variables relacionadas con la conexión
//Socket conexion;
// Almacena la información recibida de la conexión
//char cadena[TAM_MENSAJE_CLIENTE];

// Para llevar la cuenta de iteraciones
long iteracion = 0;

/***************************************************************************
*              F I N   V A R I A B L E S   L O C A L E S
***************************************************************************/

/***************************************************************************
*                    I N I C I A L I Z A C I O N
***************************************************************************/

// Cargamos el fichero de configuración (main.conf)
load_conf_file(&conf_info,"romeo.conf");

// Copiamos la tabla de 2 dimensiones donde se encuentran los grupos de
// activacion de los sonares en la variable "cadena_sonares" dentro de
// la clase Romeo.
romeo.copy_cadena_sonares(conf_info);

// Leemos del fichero de configuración si se trata de simulación o no
simulate = conf_info.simulation;

// Inicialización del Romeo
if(romeo.init_romeo() < 0)
  {
    cerr << "main: error inicializando Romeo" << endl;
    exit(ERROR);
  }

cout << "main: romeo inicializado!" << endl;

// Centramos el volante.Esto debemos hacerlo antes de iniciar el hilo del timer
// para evitar que incremente el semaforo del timer indiscriminadamente.
//romeo.centrar_volante_modo_posicion();

// Le damos al vehículo la curvatura inicial y la posición inicial
if (!simulate)
  {
    //Inicialazación de la posición y orientación por línea de comandos
    if((argc>1)&&(!romeo.setup.gps_use))
    {
      romeo.set_initial_position(atof(argv[1]),atof(argv[2]),atof(argv[3]));
    }
    else
    {
     romeo.set_initial_position();
    }
    ref_curv = conf_info.init_curv;
    romeo.set_curv(ref_curv);
    // Y esperamos 2 segundos para que dé tiempo a moverse el volante
    sleep(2);
    // Aquí ponemos el vehículo a la velocidad deseada
    ref_speed = conf_info.init_speed;
    romeo.set_speed(ref_speed);
  }
```

```
// Inicialización del temporizador del bucle de control
setup_timer(conf_info.control_period);

// Configuración de la señal SIGINT para que nos saque del bucle de control
config_int_signal();

// Iniciamos los hilos necesarios en función de los dispositivos que queramos
// usar

iniciar_hilo_dcx();
iniciar_hilo_ax5411();

if (conf_info.gps_use)
    iniciar_hilo_gps();
if (conf_info.gyro_use)
    iniciar_hilo_gyro();
if (conf_info.laser_use)
    iniciar_hilo_laser();

iniciar_hilo_estimacion();

// Hilo de teleoperación
if (romeo.get_teleop_mode() > 0)
    iniciar_hilo_teleop();

/*******************************
* Inicialización del algoritmo *
*******************************/



/*****************************
* Fin inicial. del algoritmo *
*****************************/

/*****************************************************************************
*                F I N    I N I C I A L I Z A C I O N
*****************************************************************************/

/*****************************************************************************
*                 B U C L E   D E   C O N T R O L
*****************************************************************************/

cout << "main: entrando en bucle de control" << endl;

while(!fin)
{
    iteracion++;

    // Nos esperamos hasta que expire el temporizador
    semaphore_p_restart(timer_sem);

    // Cuando comenzamos la iteración, calculamos el instante actual
    time_act = romeo.get_relative_time();


/*****************************************************************************
     B L O Q U E   D E    E S T I M A C I O N    D E    D A T O S
*****************************************************************************/

    if(romeo.get_teleop_mode() != 1)
    {
    // Volcamos en "estimated" los datos de la última estimación
    estimated = romeo.get_estimated_status(0);

/*****************************************************************************
  F I N   B L O Q U E   D E    E S T I M A C I O N    D E    D A T O S
*****************************************************************************/

/*****************************************************************************
*              A L G O R I T M O   D E   C O N T R O L
*****************************************************************************/

    // Imprimir el estado del ROMEO por pantalla
    cout<<"Estado de Romeo en el instante "<<estimated.global_time<<endl;
    cout<<"velocidad= "<<estimated.speed<<endl;
```

```
    cout<<"curvatura= "<<estimated.curv<<endl;
    cout<<"time_speed_curv= "<<estimated.time_speed_curv<<endl;
    cout<<"x = "<<estimated.x<<endl;
    cout<<"y= "<<estimated.y<<endl;
    cout<<"time_x_y= "<<estimated.time_x_y<<endl;
    cout<<"orientacion= "<<estimated.orient<<endl;
    cout<<"derv. orientacio= "<<estimated.dorient<<endl;
    cout<<"time_orient= "<<estimated.time_orient<<endl;
    cout<<"beta_remolque= "<<estimated.beta_remolque<<endl;
    cout<<"time_beta_sonar= "<<estimated.time_beta_sonar<<endl;

   for (int i=0;i<NUM_SONAR;i++)
    {
      cout<<"sonar "<<i<<"= "<<estimated.sonar[i];
    }

   // Faltaría los puntos del láser
  cout<<"time_laser= "<<estimated.time_laser<<endl;

    // Protección ante curvaturas demasiado elevadas. Éstas pueden provocar que
    // se golpee el tope de la dirección, afectando de alguna forma al motor de
    // dirección.
    if(ref_curv>CURV_MAX)
      {
        ref_curv=CURV_MAX;
      }
    if(ref_curv<(-CURV_MAX))
      {
        ref_curv=(-CURV_MAX);
      }


/*****************************************************************************
*          F I N    A L G O R I T M O    D E    C O N T R O L
*****************************************************************************/

/*****************************************************************************
*              B L O Q U E    D E    A C T U A C I O N
*****************************************************************************/

    } // fin if(romeo.get_teleop_mode != 1)

    else
    {
     REFS_TELEOP refs_teleop;
     romeo.receive_refs();
     refs_teleop = romeo.get_refs_teleop(0);

     ref_speed = refs_teleop.speed;
     ref_curv = refs_teleop.curv;
     fin = refs_teleop.end;

     cout << "Referencias rxdas: " << ref_speed << ref_curv << fin << endl;
     }

     // Protección ante curvaturas demasiado elevadas. Éstas pueden provocar que
     // se golpee el tope de la dirección, afectando de alguna forma al motor de
     // dirección.
     if(ref_curv > CURV_MAX)
     ref_curv = CURV_MAX;
    if(ref_curv < -CURV_MAX)
    ref_curv = -CURV_MAX;

    // Aplicamos las consignas
    if (!simulate)
      {
        romeo.set_curv(ref_curv);
        romeo.set_speed(ref_speed);
      }

/*****************************************************************************
        F I N    D E     B L O Q U E    D E    A C T U A C I O N
*****************************************************************************/

/*****************************************************************************
            B L O Q U E    D E    M O N I T O R I Z A C I O N
*****************************************************************************/
```

```
/*****************************************************************************
      F I N   D E   B L O Q U E   D E   M O N I T O R I Z A C I O N
*****************************************************************************/

} // Fin while(!fin)

cout << "main: saliendo del bucle de control" << endl;

/*****************************************************************************
*         F I N   D E L   B U C L E   D E   C O N T R O L
*****************************************************************************/

/*****************************************************************************
*            B L O Q U E   D E   T E R M I N A C I O N
*****************************************************************************/

// Destruimos los hilos iniciados
destruir_hilo_dcx();
destruir_hilo_ax5411();

if (conf_info.gps_use)
    destruir_hilo_gps();
if (conf_info.gyro_use)
    destruir_hilo_gyro();
if (conf_info.laser_use)
    destruir_hilo_laser();

destruir_hilo_estimacion();

// Hilo de teleoperación
if (conf_info.teleop)
destruir_hilo_teleop();

cout << "main: hilos destruidos" << endl;

// Para llegar al estado final adecuado
romeo.end_romeo();

cout << "main: romeo terminado" << endl;
cout << "main: fin del programa" << endl;

/*****************************************************************************
*       F I N   D E    B L O Q U E   D E   T E R M I N A C I O N
*****************************************************************************/

}  // Fin main(...)
```

Para una información más detallada sobre la arquitectura antigua del Romeo-4R y su código asociado, ver el Proyecto Fin de Carrera de Rafael Martín de Agar Tirado "*Control automático de un vehículo autónomo bajo el sistema operativo GNU/Linux. Implementación de drivers y software de control*".