Febrero-2016

Sistemas Electrónicos Para la Automatización

Trabajo Primera Parte 1

López Paneque, Julio José
Mier Muñoz, Gonzalo

Título. Trabajo Parte 1 de la asignatura
Razón. Entrega al profesor de la asignatura
Fecha. Febrero - 2016

Índice

DEFINICIÓN DEL PROYECTO

El objetivo de este trabajo es realizar un robot móvil que sea capaz de "atrapar" un objeto. Dicho robot y objeto deben tener una característica determinante que pueda ser reconocida por un sistema de visión (Ej.: un color muy marcado o un identificador en blanco y negro como los de los códigos QR).

Para el reconocimiento de estas características, habrá un ordenador (portátil o fijo) que tendrá instaladas una serie de funciones de reconocimiento visual, y procesará las imágenes de una cámara Kinect, de la empresa Microsoft (aunque otra cámara estándar también sería válida). Dichas imágenes procesadas permitirán extrapolar las posiciones en el espacio del robot y del objeto a perseguir, que serán enviadas al procesador del mismo.

Además de las posiciones predichas, en el ordenador deberá haber un segundo programa en paralelo que lea las instrucciones pulsadas en un mando de Xbox para PC (en este caso sí debe ser dicho mando, pues se tienen los drivers para éste) y se las envíe al robot.

Para que sean posibles todas estas comunicaciones, el robot tendrá montado una antena WiFi (CC3100 SimpleLink ™ BoosterPack), que tendrá implementada comunicación por TCP, mediante la cual recibirá las órdenes y variables enviadas desde el ordenador.

Con todos estos datos, el robot deberá poder ser controlado tanto por una técnica de control automático discreta (PID incremental) y de tiempo de muestreo fijo, como por el propio usuario con el mando. Tanto el control automático como el manual deberán poderse activar y desactivar a voluntad del usuario (quien tiene el mando).

El robot debe disponer de sistemas para recuperarse de un posible fallo de conexión TCP, y deberá ser capaz de ejecutar todas sus tareas necesarias en paralelo (si se requiere controlar, se debe hacer independientemente de lo que esté pasando en el resto de la placa, a excepción del uso de un recurso compartido. Por ello, los programas (tareas) del mismo deberán implementarse en paralelo, no de manera secuencial.

Una vez se dispone de las variables de actuación necesarias, el robot deberá manejar sus dos motores (izquierdo y derecho) mediante sendos PWMs independientes, cuyo rango de valores permitirá mover al robot como uno diferencial. Estas salidas funcionarán como señal de ENABLE de un driver de motores L293D, para el cual se deberá crear una placa de adaptación. Las salidas del driver irán directamente a los motores del robot.

Adicionalmente, se deberá disponer otra salida PWM de 4 pines para dar opción a implantar un servomotor como rueda delantera, haciendo así un robot de tipo triciclo (aunque al final no se hizo por falta de necesidad).

JUSTIFICACIÓN DEL CONTROLADOR USADO

La necesidad de un controlador avanzado como el usado (TM4C1294) se hace latente en las especificaciones del proyecto a desarrollar:

- Se requiere una funcionalidad multihilo, y un sistema en tiempo real, cosas que se pueden conseguir mediante el uso del sistema TIRTOS, que se puede utilizar en placas de Texas Instruments.
- El consumo del procesador es importante, para poder usar el robot durante el mayor tiempo posible. Por ello, poner un procesador como los usados en ordenadores carece de interés.
- Se requiere que la placa a usar sea robusta, pues es probable que se produzcan errores de montado, funciones que provoquen fallos… Esto hace inviable el uso de una placa dedicada íntegramente al control de motores, pues estas suelen ser más frágiles y menos robustas.
- El uso de boosterpacks y placas personalizadas tiene mucho interés en una plataforma de experimentación como la que se ha desarrollado, y para ello el doble hueco de boosterpacks de que dispone esta placa se torna de mucho interés.

DESARROLLO DEL PROYECTO

DESARROLLO DEL SOFTWARE DEL COCHE

Base del Sistema. TIRTOS y Distribución de las Tareas.

Explicación y descripción

La necesidad de RTOSS en las aplicaciones de hoy día (desde wearables hasta controladores críticos) ha llevado a Texas Instruments a crear un sistema de desarrollo en tiempo real, que destaca por, entre otras, las siguientes características:

- Gestión en paralelo de diferentes tareas.
- Herramientas de acceso a memoria compartida (semáforos y colas de mensajes).
- Herramientas de temporización (funciones periódicas).
- Control de prioridades de ejecución.

Para establecer cuáles de estas herramientas se usan, se debe modificar el archivo *'.cfg'*, que tiene una interfaz cómoda para dicho propósito. Se pueden incluir y configurar todas estas opciones mediante dicha GUI, que se encarga de modificar el archivo *'.xml'* de configuración (en el que se podría escribir todo eso, pero manualmente).

Para este robot, se han usado tres tareas:

- La tarea principal, que inicia el procesador y seguidamente se encarga de la recepción de los nuevos mensajes, y de reestablecer la conexión en caso de que se pierda la comunicación con el emisor.
- La tarea de control, que inicia todos los elementos necesarios para el mismo y acto seguido entra en un bucle de descanso. Se podría haber parado definitivamente, pero se quiso dejar así para posibles modificaciones.
- La tarea de interpretación de datos, que se encarga de guardar en sus respectivas variables los datos nuevos que se hayan recibido en el hilo de recepción. Esta tarea también ejecuta el control directo a través del mando conectado al ordenador emisor.

Además se ha añadido una tarea temporizada (a 10 Hz), que permite realizar un control discreto sobre el sistema. Dicha tarea accede también a variables de memoria compartida, aunque tiene un tiempo máximo de espera, pues no se quiere tener un retraso en el control.

Recepción TCP y Creación de Estación WiFi

Inicialización de la recepción TCP

```
void echoFxn(int port)
{

        int         status;
        int         clientfd;
        int         server;
        int i;
        sockaddr_in localAddr;
        sockaddr_in clientAddr;
        socklen_t   addrlen = sizeof(clientAddr);
        char        buffer[TCPPACKETSIZE];

        // Creation of the socket (TI code. Always the same steps)
        server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (server == -1) {
                System_printf("Error: socket not created.\n");
                goto shutdown;
        }

        memset(&localAddr, 0, sizeof(localAddr));
        localAddr.sin_family = AF_INET;
        localAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        localAddr.sin_port = htons(port);

        status = bind(server, (const sockaddr *)&localAddr, sizeof(localAddr));
        if (status == -1) {
                System_printf("Error: bind failed.\n");
                goto shutdown;
        }

        status = listen(server, 0);
        if (status == -1){
                System_printf("Error: listen failed.\n");
                goto shutdown;
        }

        // BENJIE IMPLEMENTATIONS
        //System_printf("Ready for new communications.\n");      // For debugging
        //System_flush();

        // Accept new socket
        clientfd =  accept(server, (sockaddr *)&clientAddr, &addrlen);
        //System_printf("Socket aceptado.\n"); // For debugging
        //System_flush();

        // Receive first input
        bytesRcvd = recv(clientfd, buffer, TCPPACKETSIZE, 0);


        for(i=0;i<TCPPACKETSIZE;i++)
                datos[i] = buffer[i];

        // Uninterrupt the other tasks (now that data is available for the first time)
        WifiON = 1;
        Semaphore_post(StartSemaphore2);
        Semaphore_post(WifiSemaphore);
        Semaphore_post(ControlSemaphore);
```

Al principio, esta función realiza las necesarias inicializaciones de sockets, de acuerdo a los estándares de C, que se han adaptado a esta placa en la librería “sockets.c” de Texas Instruments.

Posteriormente, se acepta la conexión y se reciben datos por primera vez. Al tener esos datos, se activan (desbloqueando los semáforos) las diferentes tareas (ya que ahora tienen datos con los que poder trabajar). Después de esto, se entra en el bucle de recepción de datos, explicado en el siguiente apartado.

Bucle de recepción

```
while(1){
                // Receive new data
                bytesRcvd = recv(clientfd, buffer, TCPPACKETSIZE, 0);

                if(bytesRcvd >= 0)      // No errors
                {
                        Semaphore_pend(WifiSemaphore, BIOS_WAIT_FOREVER);
                        for(i=0;i<TCPPACKETSIZE;i++)
                                datos[i] = buffer[i];
                        dato_rec = 1;    // So that Kinematics.c task uses the data once again
                        Semaphore_post(WifiSemaphore);
                }
                else{ // If there is an error of reception
                        close(clientfd);
                        // New socket
                        clientfd =  accept(server, (sockaddr *)&clientAddr, &addrlen);
                }

        }
```

En este bucle, la tarea se dedica a recibir un mensaje TCP (función recv), para acto seguido comprobar si la comunicación es correcta (si se corta la conexión se recibe un -1, indicando error) y, en caso de serlo, se entra en la zona reservada de memoria (WifiSemaphore) y se escribe el buffer en la variable compartida de recepción (datos). También se avisa de que hay nueva información (dato_rec = 1), y posteriormente se sale de la zona compartida y se vuelve a recibir.

En caso de que la conexión se pierda, el programa vuelve a aceptar una conexión, de manera que el ordenador pueda volver a conectarse.

Tarea llamada

```
Void tcpEchoTask(UArg arg0, UArg arg1)
{
        void *netIF;

        /* Open WiFi and await a connection */
        netIF = socketsStartUp();

        echoFxn(TCPPORT);

        /* Close the network - don't do this if other tasks are using it */
        socketsShutDown(netIF);
}
```

Esta tarea es la que llama el OS tras la ejecución de main. Una vez llamada, ésta llama a la función "echoFxn", que será la que se ejecute durante el funcionamiento del programa.

Funciones de Control de Motores y Control Automático

Inicialización del PWM

```
void PWM_ControlInit( void )
{
        // Initialize GPIO pins
        GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_7 );
        GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_1 | GPIO_PIN_2);

        // Initialize PWM pins
        SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
        GPIOPinConfigure(GPIO_PG1_M0PWM5);       // Forward motor
        GPIOPinConfigure(GPIO_PK4_M0PWM6);       // Left motor
        GPIOPinConfigure(GPIO_PK5_M0PWM7);       // Right motor

        // Initialized PWMs
        PWMGenConfigure(PWM0_BASE, PWM_GEN_0,PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
        //Firstly up, then down
        PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 120000);
                                //1KHz

        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_5)*M3_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
                                //Activate PF0 as output for PWM (not used)
        //1KHz
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                                //Activate PF1 as output for PWM
        //1KHz
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                                //Activate PF2 as output for PWM

        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_7,GPIO_PIN_7);
}

void PWM_ControlEnable( void )
{
        // This function initializes the PWM interrupts
        PWMGenEnable(PWM0_BASE, PWM_GEN_0);
        IntMasterEnable();
}
```

Las dos funciones puestas sirven para inicializar los tres posibles PWMs a usar, uno de los cuales no se ha utilizado en este proyecto (pues el robot se hizo diferencial, no triciclo). Se encargan de ajustar todos los valores necesarios para el funcionamiento de la salida modulada, y esto lo implementan mediante las funciones definidas en las librerías de TIVA-C (usadas también en la asignatura).

Funciones para facilitar el control

```
void _Right_Motor_Speed(float speed)
{
        speed = speed>100?100:speed;     // Treshold
        speed = speed<-100?100:speed;    // Treshold
        if(speed < 10 && speed > -10)    // Too little speed (< 10% DC)
        {
                // Halt motor and PWM
                M2_TOn = 0;
                _Right_Motor_Halt();
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, false);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
        else if(speed > 0)       // Go forward
        {
                M2_TOn = speed/100;
                _Right_Motor_Forward();
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
        else
        {
                M2_TOn = speed/100;
                _Right_Motor_Backwards();
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
}

void _Left_Motor_Speed(float speed)
{
        //speedI = speed;
        speed = speed>100?100:speed;
        speed = speed<-100?100:speed;
        if(speed < 10 && speed > -10)    // Too little speed (< 10% DC)
        {
                M1_TOn = 0;
                _Left_Motor_Halt();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, false);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
        else if(speed > 0)       // Go forward
        {
                M1_TOn = speed/100;
                _Left_Motor_Forward();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
        else
        {
                M1_TOn = speed/100;
                _Left_Motor_Backwards();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
}

void _Left_Motor_Halt(void)
{
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2 | GPIO_PIN_1);    // Both H-Bridge pins
to HIGH
}
```

```
void _Right_Motor_Halt(void)
{
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2 | GPIO_PIN_1);    // Both H-Bridge pins
to HIGH
}

void _Left_Motor_Forward(void)
{
#ifdef LEFT_MOTOR_REVERSE        // Can be easily reversed
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1);
#endif
}

void _Left_Motor_Backwards(void)
{
#ifdef LEFT_MOTOR_REVERSE        // Can be easily reversed
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2);
#endif
}

void _Right_Motor_Forward(void)
{
#ifdef RIGHT_MOTOR_REVERSE       // Can be easily reversed
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1);
#endif
}

void _Right_Motor_Backwards(void)
{
#ifdef RIGHT_MOTOR_REVERSE       // Can be easily reversed
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2);
#endif
}
```

Lo que se persigue al usar estas funciones es facilitar la posterior inclusión en un código más avanzado. Las funciones "_X_Motor_Forwards/Backwards/Halt" controlan los pines de dirección (conectados al puente H) de cada uno de los motores. Se han declarado del tipo "inline", pues ocupan poca memoria y se prefiere evitar llamar cada vez a cada una de estas funciones, que a fin de cuentas sólo son escrituras en el GPIO. También se incluye la posibilidad de dar la vuelta (por software) a los motores, si por lo que sea se requiriese una modificación sin tocar el hardware.

Posteriormente, las funciones "_X_Motor_Speed( )" permiten al programador pedir una velocidad (de -100 a 100) para alguno de los dos motores. Estas funciones discriminan distintos casos dependiendo de la velocidad que se les haya pedido y realizan las operaciones necesarias para darla. Para ello, hacen uso de las funciones que se han dicho anteriormente (para direccionar o parar el motor), e imprimen la velocidad necesaria al PWM correspondiente. Si se tuviese que parar el motor, también se encargan de desactivar el PWM.

Función de control automático

```
#ifdef FINAL_PROYECT
void Control_interrupt(void)
{
        if(Semaphore_pend(ControlSemaphore, 10/*BIOS_NO_WAIT*/)) // Only if data is ready (10ms timeout)
        {
                if(controlON)
                {
                        errorpos = sqrt(pow(ref.cx-robot.cx,2.0)+pow(ref.cy-robot.cy,2.0)); // Error in
position
                        if(errorpos > 7)// Minimun error required
                        {
                                /*
                                 * Control algorithm
                                 * This algorithm consists on two incremental PID algorithms, one for the
orientation and the other
                                 * for the position error. All the constants must be pre-declared. This is a
classical pure prosecution
                                 * control, stated in many books and scientific papers.
                                 */
                                thetaref = atan2((ref.cy-robot.cy),(ref.cx-robot.cx));
                                errorang = thetaref - robot.theta;
                                errorang = errorang>0.0?fmod(errorang,(2*PI)):fmod(errorang,(-2*PI));
                                errorang = errorang>PI?fmod(errorang,-PI):errorang;
                                errorang = errorang<-PI?fmod(errorang,PI):errorang;

                                inc_v = Kp_v * (errorpos - errorpos1) + Kp_v * Ts * errorpos / Ti_v + Kp_v *
Td_v * (errorpos + errorpos2 - 2 * errorpos1) / Ts;
                                inc_alpha = Kp_h * (errorang - errorang1) + Kp_h * Ts * errorang / Ti_h + Kp_h
* Td_h * (errorang + errorang2 - 2 * errorang1) / Ts;

                                inc_M1 = inc_v-inc_alpha;
                                inc_M2 = inc_v+inc_alpha;

                                speedI += inc_M1;
                                speedD += inc_M2;

                                // Speed limiting
                                speedI = speedI>100?100:speedI;
                                speedI = speedI<-100?-100:speedI;
                                speedD = speedD>100?100:speedD;
                                speedD = speedD<-100?-100:speedD;

                                _Right_Motor_Speed((float)(speedD+speedD1+speedD2)/3);    // Filtered signal
                                _Left_Motor_Speed((float)(speedI+speedI1+speedI2)/3);     // Filtered signal

                                // FOR DEBUGGING
                                //System_printf("M1 %d\n",(int)(float)(speedD+speedD1+speedD2)/3);
                                //System_printf("M2 %d\n",(int)(float)(speedI+speedI1+speedI2)/3);
                                //System_flush();

                                // Calculation of future values
                                errorpos2 = errorpos1;
                                errorpos1 = errorpos;
                                errorang2 = errorang1;
                                errorang1 = errorang;
                                speedD2 = speedD1;
                                speedD1 = speedD;
                                speedI2 = speedI1;
                                speedI1 = speedI;
                        }
                        else
                        {
```

```
                                _Right_Motor_Speed((float)0.0);  // If error is to little, the car pauses.
                                _Left_Motor_Speed((float)0.0);
                        }
                }
                Semaphore_post(ControlSemaphore);
        }
}
#endif
```

Esta función, llamada por la placa a 10 Hz, se encarga de realizar un control tipo PID incremental sobre los dos motores del robot, que ejercen un control de tipo persecución pura según se ha aprendido en otras asignaturas. Lo primero que se hace es esperar, como mucho durante 10 milisegundos, a que se pueda acceder a las variables compartidas necesarias. Con ello se evita que el programa se quede colgado, y se da una actuación sólo si el retraso no es muy grande.

En concreto, el semáforo usado es “ControlSemaphore”, que está especialmente definido para esta tarea.

Interpretación de Mensajes TCP y Actualización de las Variables de la Placa

Las estructuras de recepción

```
#ifdef FINAL_PROYECT
struct pos{              // Structure for position values
        double cx;
        double cy;
        double theta;
};
struct marker{   // Low-size marker structure
        int16_t cx;
        int16_t cy;
        int16_t theta;
};
struct msg_struct{       // Low-size message structure
        uint8_t type;
        uint8_t trash;   // Needed because of the way of reading
        struct marker robot_marker;
        struct marker ref_marker;
};
struct pos robot, ref;   // Creates the position structures to use
#endif
```

En los tres posibles casos de recepción (opciones de configuración, órdenes directas e información de visión) se tiene perfectamente definido y contado cómo son los mensajes, tanto en el ordenador como en la placa, de manera que siempre se acceda de forma segura a los datos recibidos. De esta forma, no se requieren de protocolos ni cabeceras innecesarias.

Estas estructuras sólo se definen si el sistema se encuentra en su estado final, dado que si simplemente se están enviando valores a los motores para hacer comprobaciones (PWM_TEST definido y FINAL_PROTECT no definido), el sistema tiene unos mensajes mucho más simples y que ocupan menos.

Las estructuras que se muestran están pensadas para copiar los datos recibidos en el mensaje TCP, de manera que el algoritmo de control los pueda usar posteriormente. En concreto, “msg_struct” se encargará de apuntar al buffer del mensaje, y proveerá información de qué se ha recibido, y “pos” se usará para guardar los valores de posición de robot y referencia. Como curiosidad, por la forma de implementación en C, la estructura obliga a tener elementos de tamaño mínimo 16 bits (y si no lo rellena con ceros), por lo que se usó el valor “trash” para que las diferentes variables se correspondiesen con el espacio teórico que ocupan.

El código de recepción e interpretación

```
        while(1)
        {
                if(dato_rec)
                {
                        //System_printf("Comienzo impresion"); // For debugging
                        //System_flush();
                        Semaphore_pend(WifiSemaphore, BIOS_WAIT_FOREVER);//Here data must be processed

#ifdef PWM_TEST
                        if(datos[0] == '0'){
                                _Right_Motor_Speed((float)(int8_t)datos[1]);
                                vDer = (int8_t)datos[1];}
                        else{
                                _Left_Motor_Speed((float)(int8_t)datos[1]);
                                vIzq = (int8_t)datos[1];}
#else
                        if(datos[0] == 2) // Options change
                        {
                                Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                controlON = datos[1];
                                actualiza_ref = datos[2];
                                mandoON = datos[3];
                                Semaphore_post(ControlSemaphore);
                        }
                        else if(datos[0] == 1) // Direct movement order
                        {
                                if(mandoON){
                                        Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                        _Right_Motor_Speed((float)(int8_t)datos[1]);
                                        _Left_Motor_Speed((float)(int8_t)datos[2]);
                                        Semaphore_post(ControlSemaphore);
                                }
                        }
                        else // Camera info
                        {
                                Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                tcp_msg = *(struct msg_struct *)datos;

                                // Store message parameters
                                robot.cx = ((double)tcp_msg.robot_marker.cx)/10.0;
                                robot.cy = ((double)tcp_msg.robot_marker.cy)/10.0;
                                robot.theta = ((double)tcp_msg.robot_marker.theta)/1000.0;
                                if (tcp_msg.type == 4 && actualiza_ref)
                                {
                                        ref.cx = ((double)tcp_msg.ref_marker.cx)/10.0;
                                        ref.cy = ((double)tcp_msg.ref_marker.cy)/10.0;
                                        ref.theta = ((double)tcp_msg.ref_marker.theta)/1000.0;
                                        actualiza_ref = 0; // Reference is now actualized
                                }
                                Semaphore_post(ControlSemaphore);
                        }
#endif
                        Semaphore_post(WifiSemaphore);
                        dato_rec = 0;
                }
        }
```

Este código espera a que se haya recibido un nuevo dato, para luego hacer lo siguiente:

- Esperar a que se habilite el semáforo de recepción (WifiSemaphore).
- Si se está depurando el PWM (PWM_TEST):
    - Si el mensaje es para el motor derecho ('0'), se escribe el valor del segundo byte en el PWM.
    - En caso contrario, se escribe en el PWM izquierdo.
- Si no se está depurando el PWM:
    - Si se recibe un mensaje de tipo cambio de opciones (valor del byte inicial = 2), se escriben los valores recibidos (que vienen siempre en el mismo orden) en las variables correspondientes.
    - Si se recibe un mensaje de movimiento directo (valor del byte inicial = 1), se llama a las funciones de cambio de velocidad con los valores correspondientes escritos en cada byte siguiente (son valores int8_t, y deberían, aunque no es obligatorio, venir acotados entre -100 y 100).
    - Si se recibe un mensaje de visión (byte inicial = 3 o 4), se apunta al mismo con la estructura tcp_msg, y se discrimina cuál de los dos casos se tiene, ya sea que se ven robot y referencia (caso 4), o sólo robot (caso 3). Si se da el caso 4 y además se busca una nueva referencia (actualiza_ref = 1), el valor de la misma se guarda. El valor de poición del robot siempre se guarda, pues es crítico para el control.
- Por último, se reactiva el semáforo y se indica que el dato se ha recibido.

SOFTWARE DEL ORDENADOR

ROS y el Programa Aruco

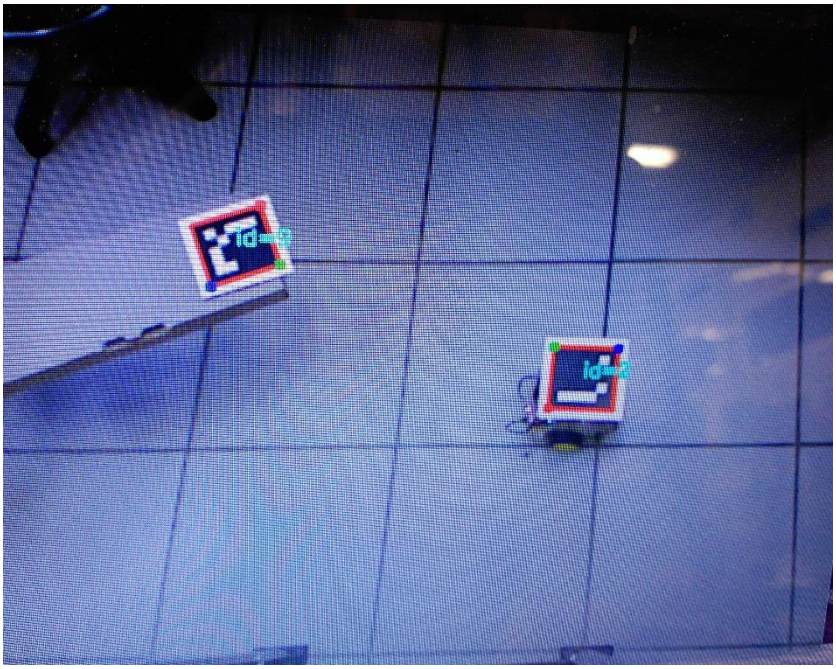
```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>


#include <iostream>
#include <stdio.h>
#include <benjie/MarcadorArray.h>
#include "aruco/aruco.h"


using namespace std;
using namespace cv;
using namespace aruco;
using namespace benjie;

ros::Publisher result_pub;


void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
    aruco::CameraParameters CamParam;
    MarkerDetector MDetector;                                   // Declaracion de detector
    vector<Marker> Markers;                                     // Declaracion de marcadores
    Marcador marcador;
    MarcadorArray marcadores;
    float MarkerSize = -1;

    cv::Mat InImage;                                            // Declaracion de imagen
    cv_bridge::CvImagePtr cv_ptr;

    cv_ptr = cv_bridge::toCvCopy(msg, "bgr8");    // Conversion del mensaje recibido a imagen
    InImage = cv_ptr->image;


    MDetector.detect(InImage,Markers,CamParam,MarkerSize);       // Deteccion de marcadores en la imagen

    for (unsigned int i=0;i<Markers.size();i++) {

        marcador.id = Markers[i].id;
        Point2f center = Markers[i].getCenter(); // Calculo del centro del marcador
        marcador.cx = center.x*10;               // Almacenado de las coordenadas del centro con un decimal
        marcador.cy = center.y*10;


        // Calculo y almacenado de la orientacion del marcador, con 3 decimales
        marcador.alpha = atan2f(Markers[i][2].y - Markers[i][1].y, Markers[i][2].x - Markers[i][1].x)*1000;


        marcadores.marcadorArray.push_back(marcador); // Guardado del marcador en un vector de marcadores

        Markers[i].draw(InImage,Scalar(0,0,255),2); // Dibuja el cuadrado donde se situa el marcador
```

```
    }

        // Dibuja la orientacion del marcador con un cubo
    if (  CamParam.isValid() && MarkerSize!=-1)
        for (unsigned int i=0;i<Markers.size();i++) {
                CvDrawingUtils::draw3dCube(InImage,Markers[i],CamParam);
        }
    // Si hay al menos un marcador, se publica
    if ( 0 < Markers.size() )    result_pub.publish( marcadores );
    imshow( "image", InImage); // Se muestra la imagen por pantalla
    waitKey(2);

}

int main(int argc, char** argv)
{
        ros::init(argc, argv, "robot_detector");
        ros::NodeHandle nh;

        image_transport::ImageTransport it(nh);
        image_transport::Subscriber sub = it.subscribe("/camera/rgb/image_rect", 1, imageCallback); // Recibe
la imagen de la kinect
        result_pub = nh.advertise< MarcadorArray >("/SEPA_node",1); // Publica los marcadores detectados

        ros::spin();

        destroyWindow( "image" );
}
```

En la función *'main'*, el nodo se suscribe a la función *'imageCallback'*, que será ejecutada cada vez que se reciba una nueva imagen, y se publica el resultado de donde se encuentran los marcadores y su orientación. En la función *'imageCallback'*, se recibe la imagen de la cámara y, usando la librería *'Aruco'* para realidad aumentada, se detectan las esquinas de los marcadores. Con los marcadores detectados, se almacena la información en un vector de marcadores, el cual cuenta con 4 datos: el identificador del marcador, el centro en 'x', el centro en 'y' y el ángulo del marcador. Como estos campos son int16, los valores que indican el centro han sido multiplicados por 10 para obtener un decimal (Más no tiene sentido porque al trabajar en pixeles, los decimales de pixeles no influyen realmente), y el ángulo por 100 para obtener 2 cifras significativas. Este cambio deberá ser deshecho posteriormente para el control. Se añade a un vector de marcadores y, si se ha detectado algún marcador se publica.

Además, se muestra la imagen que está siendo tomada y los marcadores hallados.

Este código es el único comentado en español, pues se pasó por alto al escribirlo.


id=3
id=2

Interpretación del Mando y Envío de Mensajes

Inicialización del programa

```
# Intializes everything
def start():
    #Global variables used later
        global pub
        global s
        global Start
        global Select
        global X
        global pulsadoStart
        global pulsadoSelect
        global pulsadoX
        global cambio
        Start = 0
        Select = 0
        X = 0
        pulsadoStart = 0
        pulsadoSelect = 0
        pulsadoX = 0
        cambio = 0

        #Initializes socket
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = "192.168.1.1" # Always the same address
        puerto = 1000
        rospy.loginfo("Conectando\n")
        s.connect((ip,puerto))
        rospy.loginfo("Conectado")

        # Subscribed to joystick inputs on topic "joy"
        rospy.Subscriber("joy", Joy, callback)
        rospy.Subscriber("SLAM_PRO_v3.09", benjie/MarcadorArray , callbackKinect) # Kinect Topic

        # Starts the node
        rospy.init_node('Joy2SEPA')
        rospy.spin()

# Entry point
if __name__ == '__main__':
        start()
```

Este código se encarga de iniciar el socket y la comunicación con otros nodos de ROS, y de asociar las diferentes funciones de respuesta (calback y callbackKinect) a la recepción de mensajes del mando o del nodo de interpretación de la cámara.

Tras iniciar todo esto, el programa entra en un bucle de escucha a los diferentes mensajes.

Lectura (y envío) de los botones del mando

```
# Receives joystick messages (subscribed to Joy topic)
# then converts the joysick inputs into Twist commands
# axis 1 aka left stick vertical controls linear speed
# axis 0 aka left stick horizonal controls angular speed
def callback(data):
		global Start
		global Select
		global X
		global pulsadoStart
		global pulsadoSelect
		global pulsadoX
		global cambio
		# Configuration buttons readings
		if data.buttons[7] == 1 and pulsadoStart == 0: # State machine to detect only new pulsations
			pulsadoStart = 1
			Start = not Start
			cambio = 1
		if data.buttons[7] == 0:
			pulsadoStart = 0
		if data.buttons[6] == 1 and pulsadoSelect == 0: # State machine to detect only new pulsations
			pulsadoSelect = 1
			Select = not Select
			cambio = 1
		if data.buttons[6] == 0:
			pulsadoSelect = 0
		if data.buttons[2] == 1 and pulsadoX == 0:     # State machine to detect only new pulsations
			pulsadoX = 1
			X = not X
			cambio = 1
		if data.buttons[2] == 0:
			pulsadoX = 0
		# Velocity buttons readings
		velF = 100*data.axes[1]
		velD = 100*data.axes[2]
		velm1 = velF - velD
		velm0 = velF + velD # 1 = Izq (send 1). 0 = Der (send 0) ONLY IN PWM TESTING
		# Signal resizing
		if velm1 > 100:
		  velm1 = 100
		if velm1 < -100:
		  velm1 = -100
		if velm0 > 100:
		  velm0 = 100
		if velm0 < -100:
		  velm0 = -100
		if velm1 > -10 and velm1 < 10:
		  velm1 = 0
		if velm0 > -10 and velm0 < 10:
		  velm0 = 0
		#rospy.loginfo("Motor I:"+str(int(velm1)) + " Motor Der:"+str(int(velm0))) # FOR DEBUGGING
		if velm0 < 0:
		  velm0 = 256 - abs(velm0)
		if velm1 < 0:
		  velm1 = 256 - abs(velm1)
		#s.send("0"+chr(int(velm0))) # ONLY FOR PWM TESTING
		#s.send("1"+chr(int(velm1)))

		#rospy.loginfo("Dato I:"+str("0"+int(chr(int(velm0)))) + " Dato D:"+str("0"+int(chr(int(velm1))))) 	# FOR DEBUGGING
		if cambio == 1:
			s.sendall(pack('ccccQQQQQLc',chr(2),chr(Start),chr(Select),chr(X),0,0,0,0,0,0,chr(0))) # Conversion to char and zero-
padding ( options message )
			#rospy.loginfo("Paquete"+str(2)+str(Start)+str(Select)+str(X))  # FOR DEBUGGING
```

```
            else:
                        s.sendall(pack('cccQQQQQLcc',chr(1),chr(int(velm0)),chr(int(velm1)),0,0,0,0,0,0,chr(0),chr(0))) # Conversion to char and
zero-padding ( vel message )
            cambio = 0
```

El objetivo de este programa es realizar una lectura de todos los botones del mando, de manera que varios de ellos sirvan para configuraciones y que los joysticks sirvan para envío de velocidad.

Para los botones (X, start y select), se tienen tres máquinas de estados que varían con los flancos de subida de las pulsaciones, de manera que una vez se ha producido una configuración, no varíe hasta que se pulse de nuevo. Si cualquiera de estos estados cambia, la variable "cambio" se pondrá a 1 y obligará al envío de un mensaje de configuración (descrito en unas líneas).

Para los joysticks, el mando mando remapeará los valores de los mismos (de -1 a 1) a una escala de -100 a 100, y hará nulos aquellos que sean muy pequeños (entre -10 y 10). Posteriormente convertirá a complemento a2 los valores obtenidos, y los pasará a una cadena de caracteres.

En cuanto al envío, el programa se encarga de enviar (función send) una cadena de caracteres (usando la función pack) cuyo valor depende de si se está enviando información de los joysticks o de las configuraciones. El programa enviará ceros suficientes como para que la longitud del mensaje sea la misma que la de un mensaje de visión (para evitar conflictos de tamaño).

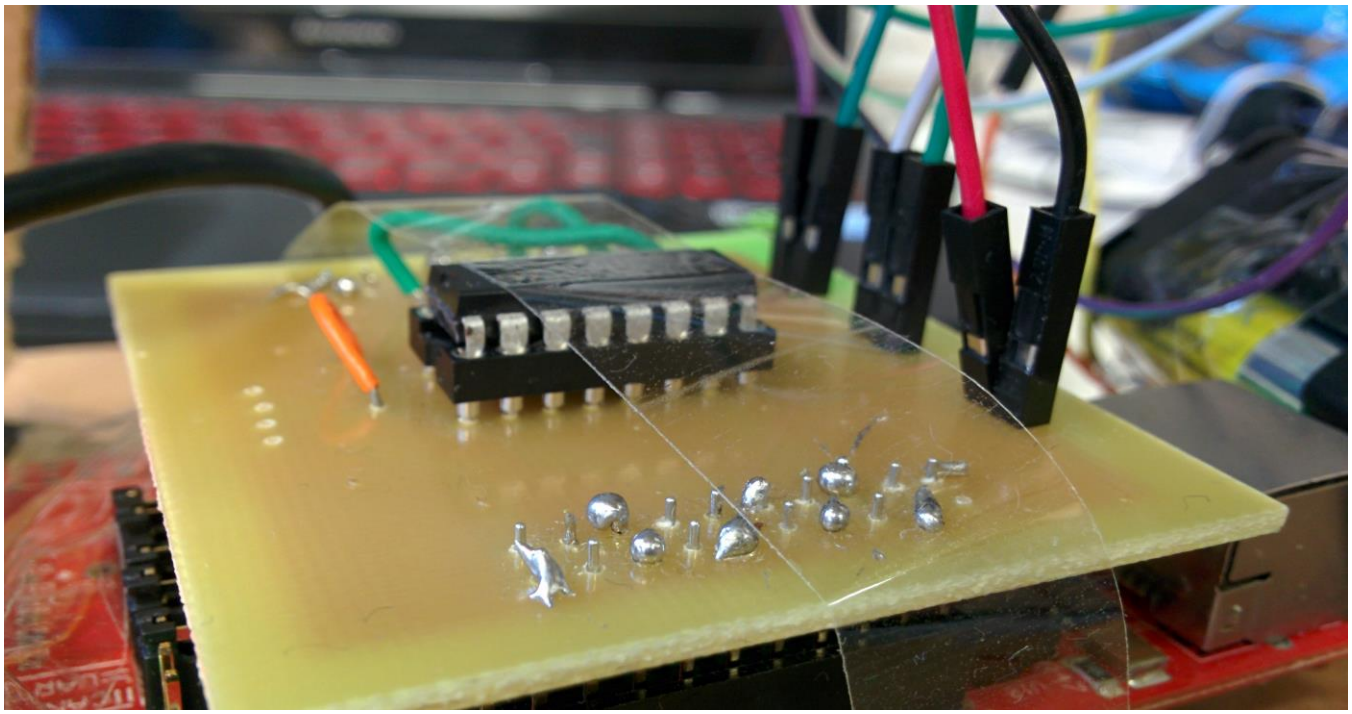
Lectura (y envío) de los mensajes de la cámara

```
def callbackKinect(arr)
    if (len(arr.marcadorarray) == 2): # If we received position of the robot and the reference
      msg = pack('c',chr(2)) # Convert to char
      if (arr.marcadorarray[0].id == 2):   # Robot is the first element
          msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
          msg = msg + pack('ddd',arr.marcadorarray[1].cx,arr.marcadorarray[1].cy,arr.marcadorarray[1].alpha) # Convert to char
       else:                          # Robot is the last element
                          msg = msg + pack('ddd',arr.marcadorarray[1].cx,arr.marcadorarray[1].cy,arr.marcadorarray[1].alpha) # Convert to char
          msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
       s.send(msg) # Send the msg over TCP socket
    elif(arr.marcadorarray[0].id == 2): # If only the robot is received
            msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
            msg = msg + pack('QQQ',0,0,0)   # Convert to char (zero-padding to fill message)
            s.send(msg)
            #rospy.loginfo("Pos: "+str(posrobot)+"Ang: "+str(angrobot)+"Ref: "+str(posdest)) # FOR DEBUGGING
```

Este código se encarga de discriminar el tipo de mensaje a enviar (dependiendo de si se ha recibido o no información de la referencia o sólo del robot) y acto seguido se encarga de enviarlo por TCP (send), en el orden establecido por todos los programas (primero robot, luego referencia). De ser necesario, se rellena con ceros todo el hueco necesario hasta llegar a la longitud del mensaje.

DESARROLLO DEL HARDWARE

Hardware en la Placa: Adaptación del L293D



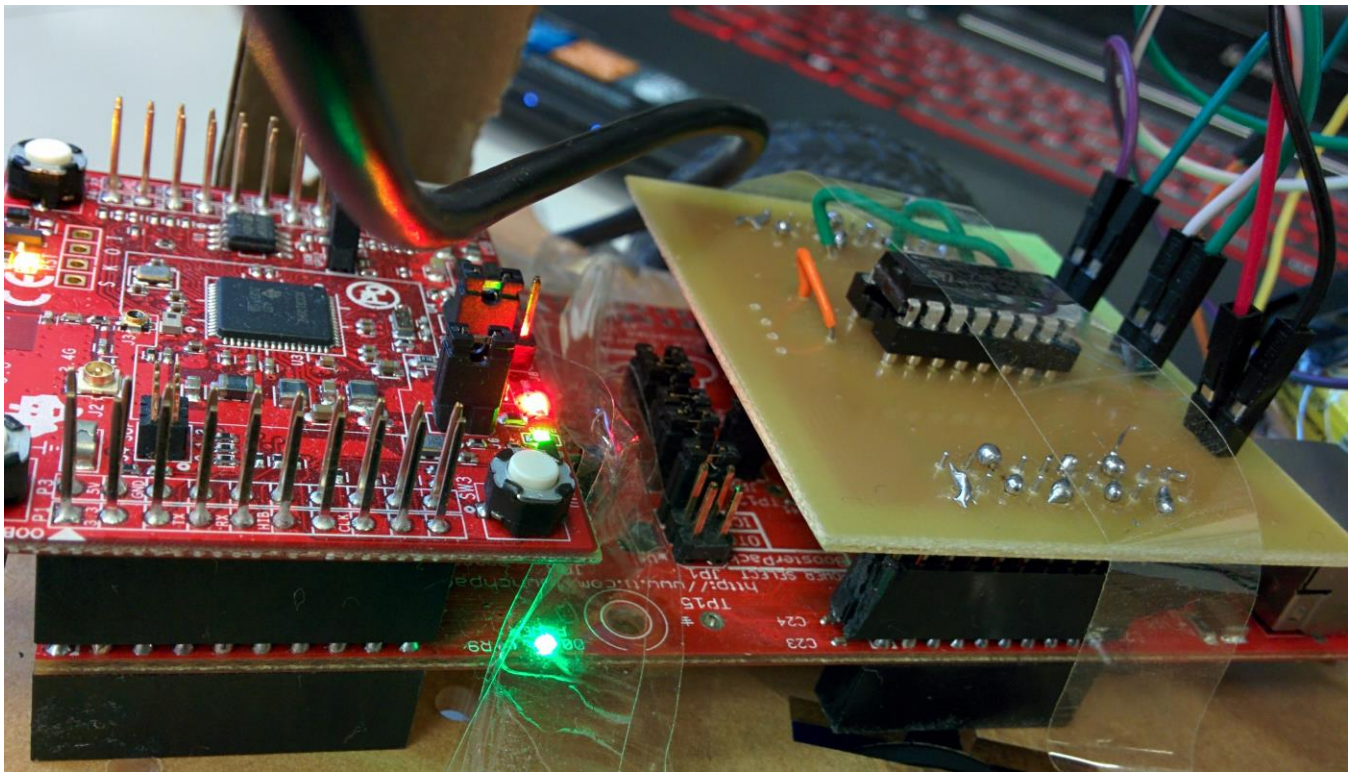
Debido a la necesidad de manejar varios motores al mismo tiempo, con unos requisitos más altos que los que se lograrían con la placa de por sí, y con una alimentación mayor, se ha desarrollado una placa diseñada únicamente para control de motores. El periférico usado es el L293D, y se encuentra conectado a diversos pines de la placa (este detalle se ve mejor en el anexo del plano impreso). Además del L293D, se dejaron otros 4 pines de acceso para un tercer posible PWM alimentado a los 5V de la placa, aunque al final no se usó. En la imagen se ven dichos pines.

La placa debe conectarse obligatoriamente al hueco 2 de boosterpacks de la tm4c1294exp.

La placa se ha hecho en Altium, con la ayuda de un compañero (Valentín) de la otra especialidad que maneja este programa. El principal problema fue ajustar la longitud entre los pines de cada lado, pues no conseguíamos que se correspondieran con nuestras medidas (se consiguió al final por pueba y error). La soldadura final presenta algunos fallos de conexión (hay que apretar un poco la placa) tras el uso continuado (y algunos golpes) de la misma, pero nada que suponga un problema irresoluble.

La placa lleva soldados una serie de cables acabados en hembra para que se puedan conectar los motores y la alimentación. Esta última va en los cables rojo y negro, y cada motor va en uno de los otros dos pares de cables.

Hardware en la Placa: Expansión con el CC3100



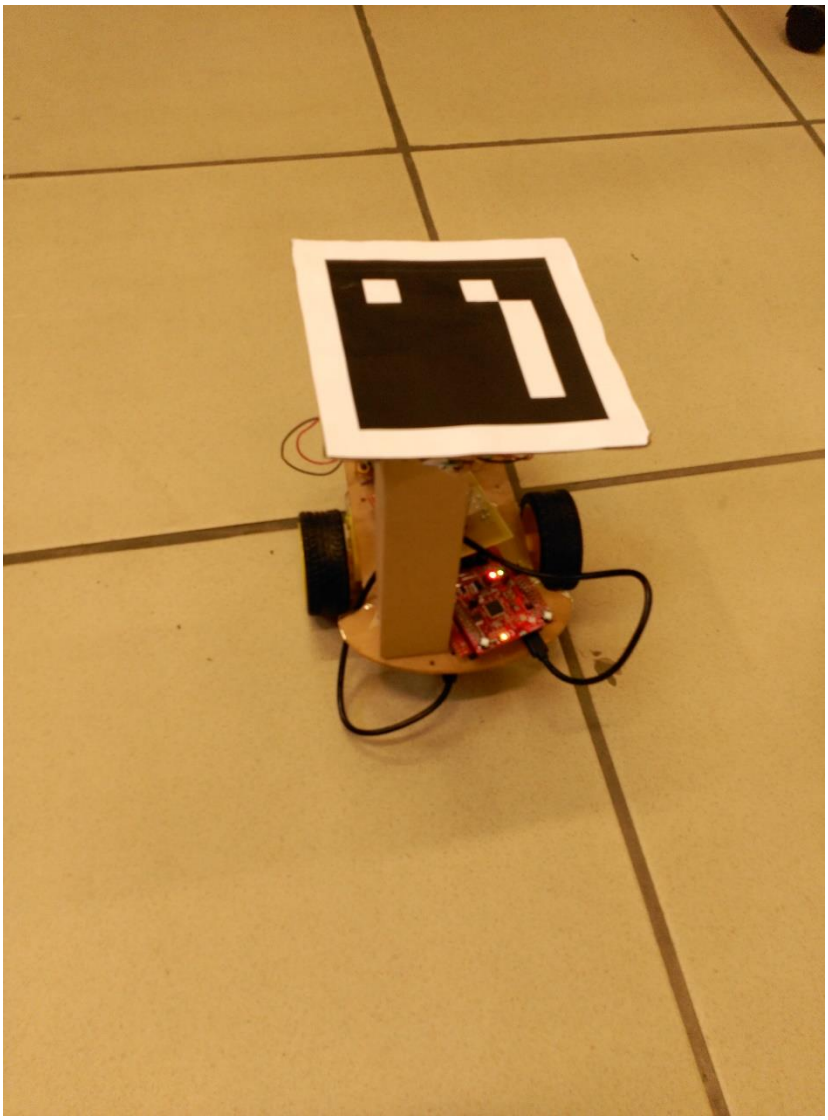
Esta placa de expansión dota al robot de muchos tipos de funcionalidades relacionadas con el Internet. Permite configurar un servidor PHP, albergar sockets, conectarse a una red WiFI o servir como punto de acceso de una red propia…

Para el proyecto, se ha utilizado en concreto la función de punto de acceso y la creación de un socket TCP. Aunque el CC3100 es de por sí una placa y tiene funcionalidades propias, al combinarse con el TIRTOS se consiguen (de forma más o menos sencilla una vez se entienden y editan los tutoriales) funcionalidades web que todo proyecto de robótica tiene hoy día. Permite establecer un nombre y contraseña a la red, activarla o desactivarla (y no sólo manualmente, que se haría mediante una combinación de botones especificada para cada caso en los tutoriales de Texas Instruments),y recibir y enviar información al entorno.

La placa se pudo usar tal y como venía y no hubo que hacer modificación alguna en los jumpers de ninguna de las dos placas.

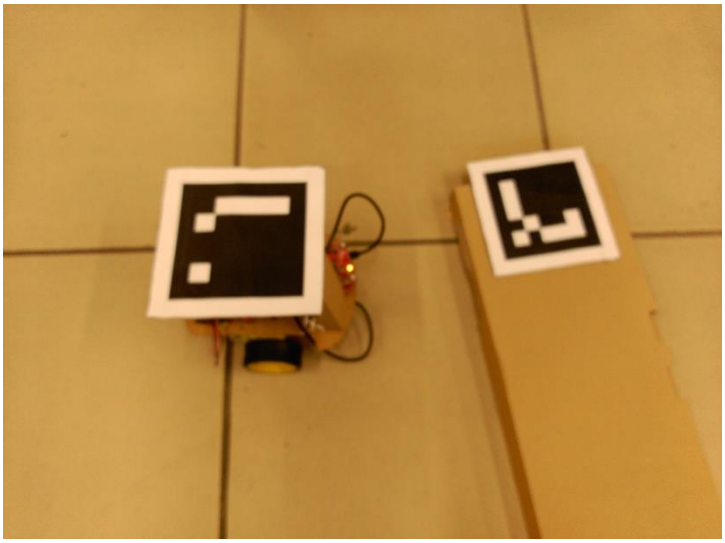
Distribución del Coche y Marcadores

El robot móvil escogido es de modelo diferencial (Dos ruedas motrices) al que se le ha añadido una rueda de castor para añadirle estabilidad. Sobre este diseño base, se sitúa la placa con un adaptador para el manejo de los motores mediante PWM (del cual se explicará más adelante) y con un módulo WiFi, y una batería recargable. Además, todo está cubierto por un marcador.



El marcador es un identificador formado por un borde exterior de color negro y un código de pixeles blanco y negros que se corresponde con otro marcador de referencia con un ángulo

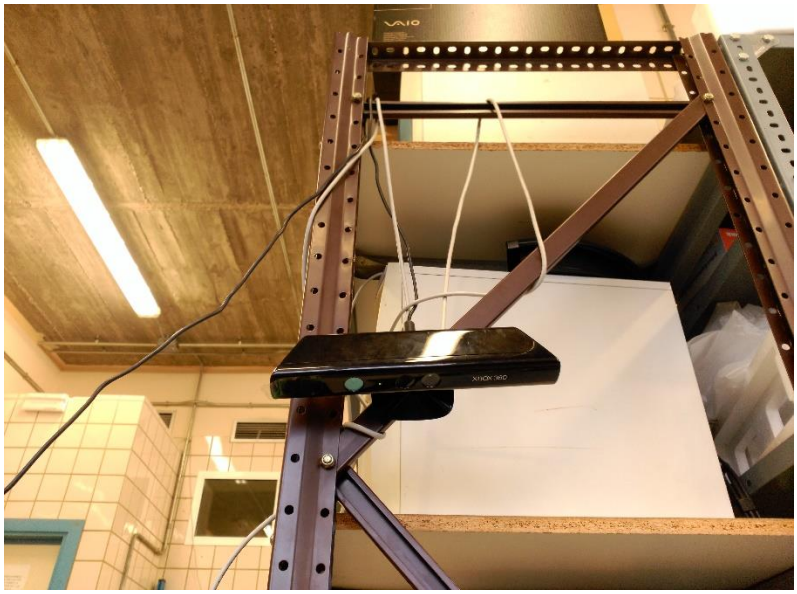
determinado. En el proceso para extraer los marcadores, se hace una separación entre negro y cualquier otro color, se busca el rectángulo negro, se señala sus esquinas y se busca la correspondencia con el identificador correspondiente. Esta tarea se ha simplificado en gran medida gracias a la librería de Aruco, que se apoya sobre OpenCV, y ROS (Robotic Operating System).



Se ha colocado un trozo de cartón bajo el marcador referencia para poder elevarlo y que la cámara tuviese la capacidad de detectarlo. Además, se ha considerado que poner el marcador en el suelo podría provocar que se dañase, aunque si se eleva demasiado oculta el marcador del robot, así que en el entorno de pruebas se dejara el marcador sobre una caja para que el robot lo empuje hasta sacarlo del entorno de trabajo.

Entorno

Para permitir el posicionamiento del robot y del marcador en cada instante, se ha optado por una cámara del tipo Kinect para Xbox 360. Este sensor integra dos cámaras que forman una cámara estéreo, una cámara infrarroja, un sensor laser 3D y micrófonos. Para este uso, únicamente se ha hecho uso de la imagen recibida por una de las cámaras RGB, ya que se aproxima que el eje "z" coincide con el eje "z" del suelo en dirección y con orientación opuestas. Asumiendo esto, se puede realizar un controlador que use como señales de entrada los pixeles del robot y los de la referencia.



MANUAL DE USUARIO

El robot viene acompañado de una batería recargable, la cual se conecta por el puerto micro-USB al microcontrolador. Esta batería alimentará todo el robot, salvo a los motores, a los cuales se deben conectar las pilas que vienen aparte (Se conectan en los cables rojo y negro de la placa personalizada).

Desde el ordenador aparecerá un nuevo punto WiFi, "Benjie". Al intentar realizar la conexión, pedirá una contraseña. Introducir "SEPA_Benjie". A partir de ese momento, el robot podrá recibir las instrucciones que se le manden desde el PC. Este proyecto necesita de una computadora con el framework *Robotic Operating System* (ROS), por lo que se recomienda usar un sistema operativo Unix. Conectar la cámara Kinect y el mando de Xbox por USB al ordenador.

Instalación: Este apartado solo habrá que hacerlo la primera vez que se use el ordenador para esta aplicación para instalar todo lo necesario. En caso de que haya sido instalado anteriormente, saltar este paso:

- Instalar ROS Fuerte en Ubuntu 12.04:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list
.d/ros-latest.list'

wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

sudo apt-get update

sudo apt-get install ros-fuerte-desktop-full
```

- Configurar ROS:

```
echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc

~/.bashrc

sudo apt-get install python-rosinstall python-rosdep

cd

mkdir ~/fuerte_workspace

mkdir ~/fuerte_workspace/sandbox

rosws set ~/fuerte_workspace/sandbox

rosws init ~/fuerte_workspace /opt/ros/fuerte
```

- Añade la carpeta del proyecto *'Benjie'* a la carpeta *'~/fuerte_workspace/sandbox/'.* Para comprobar que se ha realizado correctamente y ha sido indexado, escribe:

```
source ~/fuerte_workspace/setup.bash
```

```
roscd Benjie
```

- Instalar los driver de la cámara y del mando:

```
sudo apt-get install ros-fuerte-openni-camera
sudo apt-get install ros-fuerte-openni-launch
sudo add-apt-repository ppa:grumbel/ppa
sudo apt-get update
sudo apt-get install xboxdrv
```

- Instalar el paquete joy:

```
sudo apt-get install ros-fuerte-joy
```

Para iniciar los programas, abra la consola de comandos. Para iniciar ROS escriba '$ roscore'. Cree 5 pestañas más pulsando Ctrl-T, e inicie el programa que maneja la Kinect '$ roslaunch openni_launch openni.launch', el detector de marcadores '$ rosrun benjie kinect_aruco', el nodo de comunicaciones con el robot 'rosrun sepa_mando mando.py',el controlador de ROS del mando '$ rosrun joy joy_node' y el lector del mando 'sudo xboxdrv --silent'. En el caso que haya algún error en el nodo de comunicaciones, en la pestaña donde ha sido ejecutado pulsar Ctrl-C, reiniciar el robot pulsando el botón de reset y volver a ejecutar el programa.

La cámara hay que colocarla a más de metro y medio del suelo apuntando hacia abajo para obtener el plano cenital de la escena. Hay que hacer pruebas para detectar la región que cubre la cámara y si la luminosidad existente permite una correcta detección de los marcadores.

Si todo ha ido bien, ya se podrá manejar el robot desde el mando. Los comandos desde el mando son:

- **Joystick izquierdo**: Mover hacia delante y atrás.
- **Joystick derecho**: Girar.
- **Botón Start**: Activar/Desactivar el control automático.
- **Botón X**: Activar/Desactivar el control manual.
- **Botón Select**: Tomar nueva referencia.

La referencia se señala con el marcador adicional, aquel que no está colocado sobre el techo del robot. Mueva esa referencia para alterar el objetivo en el cual situar al robot.

Listo. Ha realizado todos los pasos necesarios para poder disfrutar de Benjie.

ANEXO DE PROGRAMAS

PROGRAMAS DEL ORDENADOR

Conexión con la Cámara y Procesado Inicial de Imágenes

Manejo y recepción del puerto TCP

kinect_aruco.cpp

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>


#include <iostream>
#include <stdio.h>
#include <benjie/MarcadorArray.h>
#include "aruco/aruco.h"


using namespace std;
using namespace cv;
using namespace aruco;
using namespace benjie;

ros::Publisher result_pub;


void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
    aruco::CameraParameters CamParam;
    MarkerDetector MDetector;                                   // Declaracion de detector
    vector<Marker> Markers;                                     // Declaracion de marcadores
    Marcador marcador;
    MarcadorArray marcadores;
    float MarkerSize = -1;

    cv::Mat InImage;                                            // Declaracion de imagen
    cv_bridge::CvImagePtr cv_ptr;

    cv_ptr = cv_bridge::toCvCopy(msg, "bgr8");   // Conversion del mensaje recibido a imagen
    InImage = cv_ptr->image;


    MDetector.detect(InImage,Markers,CamParam,MarkerSize);       // Deteccion de marcadores en la imagen

    for (unsigned int i=0;i<Markers.size();i++) {

        marcador.id = Markers[i].id;
```

```
        Point2f center = Markers[i].getCenter(); // Calculo del centro del marcador
        marcador.cx = center.x*10;               // Almacenado de las coordenadas del centro con un decimal
        marcador.cy = center.y*10;


        // Calculo y almacenado de la orientacion del marcador, con 3 decimales
        marcador.alpha = atan2f(Markers[i][2].y - Markers[i][1].y, Markers[i][2].x - Markers[i][1].x)*1000;


        marcadores.marcadorArray.push_back(marcador); // Guardado del marcador en un vector de marcadores

        Markers[i].draw(InImage,Scalar(0,0,255),2); // Dibuja el cuadrado donde se situa el marcador
    }

        // Dibuja la orientacion del marcador con un cubo
    if (  CamParam.isValid() && MarkerSize!=-1)
        for (unsigned int i=0;i<Markers.size();i++) {
                CvDrawingUtils::draw3dCube(InImage,Markers[i],CamParam);
        }
    // Si hay al menos un marcador, se publica
    if ( 0 < Markers.size() )    result_pub.publish( marcadores );
    imshow( "image", InImage); // Se muestra la imagen por pantalla
    waitKey(2);

}

int main(int argc, char** argv)
{
        ros::init(argc, argv, "robot_detector");
        ros::NodeHandle nh;

        image_transport::ImageTransport it(nh);
        image_transport::Subscriber sub = it.subscribe("/camera/rgb/image_rect", 1, imageCallback); // Recibe
la imagen de la kinect
        result_pub = nh.advertise< MarcadorArray >("/SEPA_node",1); // Publica los marcadores detectados

        ros::spin();

        destroyWindow( "image" );

}
```

Lectura del Mando y Envío de Datos por TCP

Mando.py

```
import rospy
import socket
from struct import *
from sensor_msgs.msg import Joy

# Receives joystick messages (subscribed to Joy topic)
# then converts the joysick inputs into Twist commands
# axis 1 aka left stick vertical controls linear speed
# axis 0 aka left stick horizonal controls angular speed
def callback(data):
		global Start
		global Select
		global X
		global pulsadoStart
		global pulsadoSelect
		global pulsadoX
		global cambio
		# Configuration buttons readings
		if data.buttons[7] == 1 and pulsadoStart == 0: # State machine to detect only new pulsations
			pulsadoStart = 1
			Start = not Start
			cambio = 1
		if data.buttons[7] == 0:
			pulsadoStart = 0
		if data.buttons[6] == 1 and pulsadoSelect == 0: # State machine to detect only new pulsations
			pulsadoSelect = 1
			Select = not Select
			cambio = 1
		if data.buttons[6] == 0:
			pulsadoSelect = 0
		if data.buttons[2] == 1 and pulsadoX == 0:     # State machine to detect only new pulsations
			pulsadoX = 1
			X = not X
			cambio = 1
		if data.buttons[2] == 0:
			pulsadoX = 0
		# Velocity buttons readings
		velF = 100*data.axes[1]
		velD = 100*data.axes[2]
		velm1 = velF - velD
		velm0 = velF + velD # 1 = Izq (send 1). 0 = Der (send 0) ONLY IN PWM TESTING
		# Signal resizing
		if velm1 > 100:
		  velm1 = 100
		if velm1 < -100:
		  velm1 = -100
		if velm0 > 100:
		  velm0 = 100
		if velm0 < -100:
		  velm0 = -100
		if velm1 > -10 and velm1 < 10:
		  velm1 = 0
		if velm0 > -10 and velm0 < 10:
		  velm0 = 0
		#rospy.loginfo("Motor I:"+str(int(velm1)) + " Motor Der:"+str(int(velm0))) # FOR DEBUGGING
		if velm0 < 0:
		  velm0 = 256 - abs(velm0)
		if velm1 < 0:
		  velm1 = 256 - abs(velm1)
```

```
                #s.send("0"+chr(int(velm0))) # ONLY FOR PWM TESTING
                #s.send("1"+chr(int(velm1)))

                #rospy.loginfo("Dato I:"+str("0"+int(chr(int(velm0)))) + " Dato D:"+str("0"+int(chr(int(velm1))))) # FOR DEBUGGING
                if cambio == 1:
                        s.sendall(pack('ccccQQQQQLc',chr(2),chr(Start),chr(Select),chr(X),0,0,0,0,0,0,chr(0))) # Conversion to char and zero-
padding ( options message )
                        #rospy.loginfo("Paquete"+str(2)+str(Start)+str(Select)+str(X))  # FOR DEBUGGING
                else:
                        s.sendall(pack('cccQQQQQLcc',chr(1),chr(int(velm0)),chr(int(velm1)),0,0,0,0,0,0,chr(0),chr(0))) # Conversion to char and
zero-padding ( vel message )
                cambio = 0


def callbackKinect(arr)
        if (len(arr.marcadorarray) == 2): # If we received position of the robot and the reference
          msg = pack('c',chr(2)) # Convert to char
          if (arr.marcadorarray[0].id == 2):   # Robot is the first element
             msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
             msg = msg + pack('ddd',arr.marcadorarray[1].cx,arr.marcadorarray[1].cy,arr.marcadorarray[1].alpha) # Convert to char
           else:                        # Robot is the last element
                        msg = msg + pack('ddd',arr.marcadorarray[1].cx,arr.marcadorarray[1].cy,arr.marcadorarray[1].alpha) # Convert to char
             msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
          s.send(msg) # Send the msg over TCP socket
        elif(arr.marcadorarray[0].id == 2): # If only the robot is received
                msg = msg + pack('ddd',arr.marcadorarray[0].cx,arr.marcadorarray[0].cy,arr.marcadorarray[0].alpha) # Convert to char
                msg = msg + pack('QQQ',0,0,0)   # Convert to char (zero-padding to fill message)
                s.send(msg)
                #rospy.loginfo("Pos: "+str(posrobot)+"Ang: "+str(angrobot)+"Ref: "+str(posdest)) # FOR DEBUGGING

# Intializes everything
def start():
        #Global variables used later
                global pub
                global s
                global Start
                global Select
                global X
                global pulsadoStart
                global pulsadoSelect
                global pulsadoX
                global cambio
                Start = 0
                Select = 0
                X = 0
                pulsadoStart = 0
                pulsadoSelect = 0
                pulsadoX = 0
                cambio = 0

                #Initializes socket
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                ip = "192.168.1.1" # Always the same address
                puerto = 1000
                rospy.loginfo("Conectando\n")
                s.connect((ip,puerto))
                rospy.loginfo("Conectado")

                # Subscribed to joystick inputs on topic "joy"
                rospy.Subscriber("joy", Joy, callback)
                rospy.Subscriber("SLAM_PRO_v3.09", benjie/MarcadorArray , callbackKinect) # Kinect Topic

                # Starts the node
                rospy.init_node('Joy2SEPA')
                rospy.spin()
```

```
# Entry point
if __name__ == '__main__':
        start()
```

PROGRAMAS DE LA PLACA

Manejo y recepción del puerto TCP

tcpEchoCC3100.c

```
/*
 *  ======== tcpEchoCC3100.c ========
 */

#include <string.h>

#include <xdc/std.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/drivers/GPIO.h>

#include <xdc/runtime/Memory.h>
#include <xdc/runtime/Error.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Queue.h>

/* SimpleLink Wi-Fi Host Driver Header files */
#include <simplelink.h>

/* Example/Board Header file */
#include "Board.h"

/* Local Platform Specific Header file */
#include "sockets.h"

/* Kinematics library */
#include "Benji/Kinematics.h"

/* Port number for listening for TCP packets */
#define TCPPORT         1000

/* Spawn Task Priority */
extern const int SPAWN_TASK_PRI;

// User-defined variables
char datos[TCPPACKETSIZE];      // Buffer for receiving
int WifiON = 0;                         // For synchronycing
int     bytesRcvd;                      // Auxiliar variable
uint8_t dato_rec;
/*
 *  ======== echoFxn ========
 *  Echoes TCP messages.
 */
void echoFxn(int port)
{

        int         status;
        int         clientfd;
        int         server;
        int i;
        sockaddr_in localAddr;
```

```
        sockaddr_in clientAddr;
        socklen_t   addrlen = sizeof(clientAddr);
        char        buffer[TCPPACKETSIZE];

        // Creation of the socket (TI code. Always the same steps)
        server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (server == -1) {
                System_printf("Error: socket not created.\n");
                goto shutdown;
        }

        memset(&localAddr, 0, sizeof(localAddr));
        localAddr.sin_family = AF_INET;
        localAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        localAddr.sin_port = htons(port);

        status = bind(server, (const sockaddr *)&localAddr, sizeof(localAddr));
        if (status == -1) {
                System_printf("Error: bind failed.\n");
                goto shutdown;
        }

        status = listen(server, 0);
        if (status == -1){
                System_printf("Error: listen failed.\n");
                goto shutdown;
        }

        // BENJIE IMPLEMENTATIONS
        //System_printf("Ready for new communications.\n");      // For debugging
        //System_flush();

        // Accept new socket
        clientfd =  accept(server, (sockaddr *)&clientAddr, &addrlen);
        //System_printf("Socket aceptado.\n"); // For debugging
        //System_flush();

        // Receive first input
        bytesRcvd = recv(clientfd, buffer, TCPPACKETSIZE, 0);


        for(i=0;i<TCPPACKETSIZE;i++)
                datos[i] = buffer[i];

        // Uninterrupt the other tasks (now that data is available for the first time)
        WifiON = 1;
        Semaphore_post(StartSemaphore2);
        Semaphore_post(WifiSemaphore);
        Semaphore_post(ControlSemaphore);


        while(1){
                // Receive new data
                bytesRcvd = recv(clientfd, buffer, TCPPACKETSIZE, 0);

                if(bytesRcvd >= 0)      // No errors
                {
                        Semaphore_pend(WifiSemaphore, BIOS_WAIT_FOREVER);
                        for(i=0;i<TCPPACKETSIZE;i++)
                                datos[i] = buffer[i];
                        dato_rec = 1;    // So that Kinematics.c task uses the data once again
                        Semaphore_post(WifiSemaphore);
                }
                else{ // If there is an error of reception
                        close(clientfd);
```

```
                        // New socket
                        clientfd =  accept(server, (sockaddr *)&clientAddr, &addrlen);
                }

        }

        // We won´t get here

        /* addrlen is a value-result param, must reset for next accept call */
        addrlen = sizeof(clientAddr);
        close(clientfd);

        shutdown:
        if (server >= 0) {
                close(server);
        }
}

/*
 *  ======== tcpEchoTask ========
 */
Void tcpEchoTask(UArg arg0, UArg arg1)
{
        void *netIF;

        /* Open WiFi and await a connection */
        netIF = socketsStartUp();

        echoFxn(TCPPORT);

        /* Close the network - don't do this if other tasks are using it */
        socketsShutDown(netIF);
}

/*
 *  ======== main ========
 */
int main(void)
{
        /* Call board init functions. */
        Board_initGeneral();
        Board_initGPIO();
        Board_initWiFi();

        /* Turn on user LED */
        GPIO_write(Board_LED0, Board_LED_ON);

        System_printf("Starting the TCP Echo example for the CC3100 \n"
                        "System provider is set to SysMin. Halt the target to view"
                        " any SysMin content in ROV.\n");

        /* SysMin will only print to the console when you call flush or exit */
        System_flush();

        /*
         * The SimpleLink Host Driver requires a mechanism to allow functions to
         * execute in temporary context.  The SpawnTask is created to handle such
         * situations.  This task will remain blocked until the host driver
         * posts a function.  If the SpawnTask priority is higher than other tasks,
         * it will immediately execute the function and return to a blocked state.
         * Otherwise, it will remain ready until it is scheduled.
         */
        VStartSimpleLinkSpawnTask(SPAWN_TASK_PRI);

        /* Start BIOS */
```

```
        BIOS_start();

        return (0);
}
```

Kinematics.h

```
/*
 * Kinematics.h
 *
 *  Created on: 3/11/2015
 *      Author: Julio
 */

#ifndef BENJI_KINEMATICS_H_
#define BENJI_KINEMATICS_H_

//#define PWM_TEST              //ONLY DEFINE ONE OF THESE
#define FINAL_PROYECT

#ifndef FINAL_PROYECT
#define TCPPACKETSIZE   2 // Only if messages are only for pwm testing
#define PWM_TEST
#else
#define TCPPACKETSIZE   14      // Classic size for the final proyect
#endif

#ifdef FINAL_PROYECT
struct pos{             // Structure for position values
        double cx;
        double cy;
        double theta;
};
struct marker{  // Low-size marker structure
        int16_t cx;
        int16_t cy;
        int16_t theta;
};
struct msg_struct{      // Low-size message structure
        uint8_t type;
        uint8_t trash;  // Needed because of the way of reading
        struct marker robot_marker;
        struct marker ref_marker;
};
struct pos robot, ref;  // Creates the position structures to use
#endif

/* Function declarations */
Int Kinematics_Module_startup(Int state);
void Kinematics_Module_Execution( void );
extern uint8_t dato_rec;

#endif /* BENJI_KINEMATICS_H_ */
```

Kinematics.c

```
/*
 *  ======= Kinematics ========
 *  Kinematics target-side implementation
 *
 *  Created on: 3/11/2015
 *  Author:     Julio
 */
#include <string.h>
#include <xdc/std.h>
#include <xdc/runtime/Startup.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/drivers/GPIO.h>

#include <xdc/runtime/Memory.h>
#include <xdc/runtime/Error.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Queue.h>

#include "Kinematics.h"
#include "Control.h"
//#include "../tcpEchoCC3100.h"

/* include Kinematics internal implementation definitions */
//#include "package/internal/Kinematics.xdc.h"

// External variables
extern int Motor1, Motor2;
extern char datos[TCPPACKETSIZE];
extern int WifiON;
extern uint8_t controlON;

// Global variables
char datoLocal1, datoLocal2;
volatile uint8_t actualiza_ref;
volatile uint8_t mandoON = 0,actualiza_ref = 0;

// Receiving structure
#ifdef FINAL_PROYECT
struct msg_struct tcp_msg;
#endif
#ifdef PWM_TEST
int8_t vDer, vIzq;       // For debugging the PWM
#endif
/*
 *  ======== Kinematics_Module_startup ========
 */
Int Kinematics_Module_startup(Int state)
{
        return (Startup_DONE);
}

void Kinematics_Module_Execution( void )
{
        //Semaphore_post(ControlSemaphore);
        Semaphore_pend(StartSemaphore1, BIOS_WAIT_FOREVER);
        while(!WifiON);
```

```
        while(1)
        {
                if(dato_rec)
                {
                        //System_printf("Comienzo impresion"); // For debugging
                        //System_flush();
                        Semaphore_pend(WifiSemaphore, BIOS_WAIT_FOREVER);//Here data must be processed

#ifdef PWM_TEST
                        if(datos[0] == '0'){
                                _Right_Motor_Speed((float)(int8_t)datos[1]);
                                vDer = (int8_t)datos[1];}
                        else{
                                _Left_Motor_Speed((float)(int8_t)datos[1]);
                                vIzq = (int8_t)datos[1];}
#else
                        if(datos[0] == 2) // Options change
                        {
                                Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                controlON = datos[1];
                                actualiza_ref = datos[2];
                                mandoON = datos[3];
                                Semaphore_post(ControlSemaphore);
                        }
                        else if(datos[0] == 1) // Direct movement order
                        {
                                if(mandoON){
                                        Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                        _Right_Motor_Speed((float)(int8_t)datos[1]);
                                        _Left_Motor_Speed((float)(int8_t)datos[2]);
                                        Semaphore_post(ControlSemaphore);
                                }
                        }
                        else // Camera info
                        {
                                Semaphore_pend(ControlSemaphore, BIOS_WAIT_FOREVER);
                                tcp_msg = *(struct msg_struct *)datos;

                                // Store message parameters
                                robot.cx = ((double)tcp_msg.robot_marker.cx)/10.0;
                                robot.cy = ((double)tcp_msg.robot_marker.cy)/10.0;
                                robot.theta = ((double)tcp_msg.robot_marker.theta)/1000.0;
                                if (tcp_msg.type == 4 && actualiza_ref)
                                {
                                        ref.cx = ((double)tcp_msg.ref_marker.cx)/10.0;
                                        ref.cy = ((double)tcp_msg.ref_marker.cy)/10.0;
                                        ref.theta = ((double)tcp_msg.ref_marker.theta)/1000.0;
                                        actualiza_ref = 0; // Reference is now actualized
                                }
                                Semaphore_post(ControlSemaphore);
                        }
#endif
                        Semaphore_post(WifiSemaphore);
                        dato_rec = 0;
                }
        }
}

#ifdef Kinematics_Object
/*
 *  ======== Kinematics_Instance_init ========
 *  Kinematics created or constructed instance object initialization
 */
Void Kinematics_Instance_init(Kinematics_Object *obj, const Kinematics_Params *params)
{
```

```
        /* TODO: initialize Kinematics instance state fields */
}
#endif
```

Control.h

```
/*
 * Control.h
 *
 *  Created on: 3/11/2015
 *      Author: Julio
 */

#ifndef BENJI_CONTROL_H_
#define BENJI_CONTROL_H_

#include <stdint.h>
#include <stdbool.h>
#include <inc/hw_memmap.h>
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/tm4c1294ncpdt.h"
#include "driverlib/pwm.h"

/* Definitions for easy reversal of the motors */
#define LEFT_MOTOR_REVERSE
#define RIGHT_MOTOR_REVERSE
/* Function declarations */
Int Control_Module_startup(Int state);
void Control_Module_Execution( void );
void PWM_ControlInit( void );
void PWM_ControlEnable( void );

inline void _Left_Motor_Forward(void);
inline void _Left_Motor_Backwards(void);
inline void _Right_Motor_Forward(void);
inline void _Right_Motor_Backwards(void);
inline void _Left_Motor_Halt(void);
inline void _Right_Motor_Halt(void);

void _Right_Motor_Speed(float speed);
void _Left_Motor_Speed(float speed);

// For syncing
extern int WifiON;

#endif /* BENJI_CONTROL_H_ */
```

Control.c

```
/*
 *  ======= Control ========
 *  Control target-side implementation
 *
 *  Created on: 3/11/2015
 *  Author:     Julio
 */
#include <xdc/std.h>
#include <xdc/runtime/Startup.h>
#include <xdc/cfg/global.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/drivers/GPIO.h>

#include <xdc/runtime/Memory.h>
#include <xdc/runtime/Error.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Queue.h>

#include "Control.h"
#include "Kinematics.h"
#include <math.h>

#define PI (3.141592653589793)

/* include Control internal implementation definitions */
//#include "package/internal/Control.xdc.h"
int Motor1, Motor2;
float M1_TOn = 0, M2_TOn = 0, M3_TOn = 0; //1->Left;2->Right;3->Forward wheel


#ifdef FINAL_PROYECT     // Enables the control function

// Constants and variables for the control algorithm
extern struct pos robot, ref;
double errorpos, errorpos1, errorpos2;
double errorang, errorang1, errorang2;
double inc_alpha, inc_M1, inc_M2, inc_v, thetaref;
const double Kp_v = 0.01, Ti_v = 10, Td_v = 0.1, Ts = 0.1;
const double Kp_h = 1, Ti_h = 11, Td_h = 0.1;
double speedD, speedI, speedD1 = 0, speedI1 = 0, speedD2 = 0, speedI2 = 0;

// Control initializer
volatile uint8_t controlON = 0;
#endif
/*
 *  ======== Control_Module_startup ========
 */
Int Control_Module_startup(Int state)
{
        return (Startup_DONE);
}

void Control_Module_Execution( void )
{
        Semaphore_pend(StartSemaphore2, BIOS_WAIT_FOREVER);
        while(!WifiON);
        PWM_ControlInit();
        PWM_ControlEnable();
        _Left_Motor_Halt();
```

```
        _Right_Motor_Halt();
        Semaphore_post(StartSemaphore1);
        while(1)
        {
                Task_sleep(1000);       // Do nothing
        }
}
#ifdef FINAL_PROYECT
void Control_interrupt(void)
{
        if(Semaphore_pend(ControlSemaphore, 10/*BIOS_NO_WAIT*/)) // Only if data is ready (10ms timeout)
        {
                if(controlON)
                {
                        errorpos = sqrt(pow(ref.cx-robot.cx,2.0)+pow(ref.cy-robot.cy,2.0)); // Error in
position
                        if(errorpos > 7)// Minimun error required
                        {
                                /*
                                 * Control algorithm
                                 * This algorithm consists on two incremental PID algorithms, one for the
orientation and the other
                                 * for the position error. All the constants must be pre-declared. This is a
classical pure prosecution
                                 * control, stated in many books and scientific papers.
                                 */
                                thetaref = atan2((ref.cy-robot.cy),(ref.cx-robot.cx));
                                errorang = thetaref - robot.theta;
                                errorang = errorang>0.0?fmod(errorang,(2*PI)):fmod(errorang,(-2*PI));
                                errorang = errorang>PI?fmod(errorang,-PI):errorang;
                                errorang = errorang<-PI?fmod(errorang,PI):errorang;

                                inc_v = Kp_v * (errorpos - errorpos1) + Kp_v * Ts * errorpos / Ti_v + Kp_v *
Td_v * (errorpos + errorpos2 - 2 * errorpos1) / Ts;
                                inc_alpha = Kp_h * (errorang - errorang1) + Kp_h * Ts * errorang / Ti_h + Kp_h
* Td_h * (errorang + errorang2 - 2 * errorang1) / Ts;

                                inc_M1 = inc_v-inc_alpha;
                                inc_M2 = inc_v+inc_alpha;

                                speedI += inc_M1;
                                speedD += inc_M2;

                                // Speed limiting
                                speedI = speedI>100?100:speedI;
                                speedI = speedI<-100?-100:speedI;
                                speedD = speedD>100?100:speedD;
                                speedD = speedD<-100?-100:speedD;

                                _Right_Motor_Speed((float)(speedD+speedD1+speedD2)/3);    // Filtered signal
                                _Left_Motor_Speed((float)(speedI+speedI1+speedI2)/3);     // Filtered signal

                                // FOR DEBUGGING
                                //System_printf("M1 %d\n",(int)(float)(speedD+speedD1+speedD2)/3);
                                //System_printf("M2 %d\n",(int)(float)(speedI+speedI1+speedI2)/3);
                                //System_flush();

                                // Calculation of future values
                                errorpos2 = errorpos1;
                                errorpos1 = errorpos;
                                errorang2 = errorang1;
                                errorang1 = errorang;
                                speedD2 = speedD1;
                                speedD1 = speedD;
                                speedI2 = speedI1;
```

```
                                speedI1 = speedI;
                        }
                        else
                        {
                                _Right_Motor_Speed((float)0.0);  // If error is to little, the car pauses.
                                _Left_Motor_Speed((float)0.0);
                        }
                }
                Semaphore_post(ControlSemaphore);
        }
}
#endif

void PWM_ControlInit( void )
{
        // Initialize GPIO pins
        GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_7 );
        GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_1 | GPIO_PIN_2);

        // Initialize PWM pins
        SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
        GPIOPinConfigure(GPIO_PG1_M0PWM5);       // Forward motor
        GPIOPinConfigure(GPIO_PK4_M0PWM6);       // Left motor
        GPIOPinConfigure(GPIO_PK5_M0PWM7);       // Right motor

        // Initialized PWMs
        PWMGenConfigure(PWM0_BASE, PWM_GEN_0,PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
        //Firstly up, then down
        PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 120000);
                                //1KHz

        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_5)*M3_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
                                //Activate PF0 as output for PWM (not used)
        //1KHz
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                                //Activate PF1 as output for PWM
        //1KHz
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );    //0% duty
cycle
        PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                                //Activate PF2 as output for PWM

        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_7,GPIO_PIN_7);
}

void PWM_ControlEnable( void )
{
        // This function initializes the PWM interrupts
        PWMGenEnable(PWM0_BASE, PWM_GEN_0);
        IntMasterEnable();
}

void _Right_Motor_Speed(float speed)
{
        speed = speed>100?100:speed;     // Treshold
        speed = speed<-100?100:speed;    // Treshold
        if(speed < 10 && speed > -10)    // Too little speed (< 10% DC)
        {
                // Halt motor and PWM
                M2_TOn = 0;
                _Right_Motor_Halt();
```

```
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, false);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
        else if(speed > 0)      // Go forward
        {
                M2_TOn = speed/100;
                _Right_Motor_Forward();
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
        else
        {
                M2_TOn = speed/100;
                _Right_Motor_Backwards();
                PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_7)*M2_TOn );
        }
}

void _Left_Motor_Speed(float speed)
{
        //speedI = speed;
        speed = speed>100?100:speed;
        speed = speed<-100?100:speed;
        if(speed < 10 && speed > -10)    // Too little speed (< 10% DC)
        {
                M1_TOn = 0;
                _Left_Motor_Halt();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, false);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
        else if(speed > 0)      // Go forward
        {
                M1_TOn = speed/100;
                _Left_Motor_Forward();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
        else
        {
                M1_TOn = speed/100;
                _Left_Motor_Backwards();
                PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, PWMGenPeriodGet(PWM0_BASE, PWM_OUT_6)*M1_TOn );
        }
}

void _Left_Motor_Halt(void)
{
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2 | GPIO_PIN_1);    // Both H-Bridge pins
to HIGH
}

void _Right_Motor_Halt(void)
{
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2 | GPIO_PIN_1);    // Both H-Bridge pins
to HIGH
}

void _Left_Motor_Forward(void)
{
#ifdef LEFT_MOTOR_REVERSE        // Can be easily reversed
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1);
```

```
#endif
}

void _Left_Motor_Backwards(void)
{
#ifdef LEFT_MOTOR_REVERSE        // Can be easily reversed
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTK_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2);
#endif
}

void _Right_Motor_Forward(void)
{
#ifdef RIGHT_MOTOR_REVERSE       // Can be easily reversed
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1);
#endif
}

void _Right_Motor_Backwards(void)
{
#ifdef RIGHT_MOTOR_REVERSE       // Can be easily reversed
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_1); // Needed H-Bridge Pin to HIGH
#else
        GPIOPinWrite(GPIO_PORTM_BASE,GPIO_PIN_2 | GPIO_PIN_1,GPIO_PIN_2);
#endif
}

#ifdef Control_Object
/*
 *  ======== Control_Instance_init ========
 *  Control created or constructed instance object initialization
 */
Void Control_Instance_init(Control_Object *obj, const Control_Params *params)
{
        /* TODO: initialize Control instance state fields */
}
#endif
```

Fichero Modificado de Configuración de la Placa

SL_common.h

```
/*
 * sl_config.h - get time sample application
 *
 * Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
 */
#ifndef __SL_CONFIG_H__
#define __SL_CONFIG_H__

//****************************************************************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
//****************************************************************************
#ifdef __cplusplus
extern "C" {
#endif

/**/
#define LOOP_FOREVER() \
            {\
                while(1); \
            }

#define ASSERT_ON_ERROR(error_code) \
            {\
                /* Handling the error-codes is specific to the application */ \
                if (error_code < 0) return error_code; \
                /* else, continue w/ execution */ \
            }

#define pal_Memset(x,y,z)   memset((void *)x,y,z)
#define pal_Memcpy(x,y,z)   memcpy((void *)x, (const void *)y, z)
#define pal_Memcmp(x,y,z)   memcmp((const void *)x, (const void *)y, z)
#define pal_Strlen(x)       strlen((const char *)x)
#define pal_Strcmp(x,y)     strcmp((const char *)x, (const char *)y)
#define pal_Strcpy(x,y)     strcpy((char *)x, (const char *)y)
#define pal_Strstr(x,y)     strstr((const char *)x, (const char *)y)
#define pal_Strncmp(x,y,z)  strncmp((const char *)x, (const char *)y, z)
#define pal_Strcat(x,y)     strcat((char *)x, (const char *)y)

/*
 * Values for below macros shall be modified per the access-point's (AP) properties
 * SimpleLink device will connect to following AP when the application is executed
 */
#define SSID_NAME       "T-Mobile Broadband86"         /* Access point name to connect to. */
#define SEC_TYPE        SL_SEC_TYPE_WPA_WPA2    /* Security type of the Access piont */
#define PASSKEY         "43406086"                 /* Password in case of secure AP */
#define PASSKEY_LEN     pal_Strlen(PASSKEY)  /* Password length in case of secure AP */

/* Configuration of the device when it comes up in AP mode */
#define SSID_AP_MODE       "Benji"       /* SSID of the CC3100 in AP mode */
#define PASSWORD_AP_MODE   "SEPA_Benji"                 /* Password of CC3100 AP */
#define SEC_TYPE_AP_MODE   SL_SEC_TYPE_WPA    /* Can take SL_SEC_TYPE_WEP or
                                               * SL_SEC_TYPE_WPA as well */

/*
 * Values for below macros shall be modified based on current time
 */
```

```
#define DATE        24      /* Current Date */
#define MONTH       7       /* Month */
#define YEAR        2014    /* Current year */
#define HOUR        17      /* Time - hours */
#define MINUTE      30      /* Time - minutes */
#define SECOND      0       /* Time - seconds */

#define SUCCESS             0

/* Status bits - These are used to set/reset the corresponding bits in a 'status_variable' */
typedef enum{
    STATUS_BIT_CONNECTION =  0, /* If this bit is:
                                 *      1 in a 'status_variable', the device is connected to the AP
                                 *      0 in a 'status_variable', the device is not connected to the AP
                                 */

    STATUS_BIT_STA_CONNECTED,    /* If this bit is:
                                  *      1 in a 'status_variable', client is connected to device
                                  *      0 in a 'status_variable', client is not connected to device
                                  */

    STATUS_BIT_IP_ACQUIRED,       /* If this bit is:
                                   *      1 in a 'status_variable', the device has acquired an IP
                                   *      0 in a 'status_variable', the device has not acquired an IP
                                   */

    STATUS_BIT_IP_LEASED,           /* If this bit is:
                                     *      1 in a 'status_variable', the device has leased an IP
                                     *      0 in a 'status_variable', the device has not leased an IP
                                     */

    STATUS_BIT_CONNECTION_FAILED,   /* If this bit is:
                                     *      1 in a 'status_variable', failed to connect to device
                                     *      0 in a 'status_variable'
                                     */

    STATUS_BIT_P2P_NEG_REQ_RECEIVED,/* If this bit is:
                                     *      1 in a 'status_variable', connection requested by remote wifi-direct
device
                                     *      0 in a 'status_variable',
                                     */
    STATUS_BIT_SMARTCONFIG_DONE,    /* If this bit is:
                                     *      1 in a 'status_variable', smartconfig completed
                                     *      0 in a 'status_variable', smartconfig event couldn't complete
                                     */

    STATUS_BIT_SMARTCONFIG_STOPPED  /* If this bit is:
                                     *      1 in a 'status_variable', smartconfig process stopped
                                     *      0 in a 'status_variable', smartconfig process running
                                     */

}e_StatusBits;

#define SET_STATUS_BIT(status_variable, bit)    status_variable |= ((unsigned long)1<<(bit))
#define CLR_STATUS_BIT(status_variable, bit)    status_variable &= ~((unsigned long)1<<(bit))
#define GET_STATUS_BIT(status_variable, bit)    (0 != (status_variable & ((unsigned long)1<<(bit))))

#define IS_CONNECTED(status_variable)             GET_STATUS_BIT(status_variable, \
                                                            STATUS_BIT_CONNECTION)
#define IS_STA_CONNECTED(status_variable)         GET_STATUS_BIT(status_variable, \
                                                            STATUS_BIT_STA_CONNECTED)
#define IS_IP_ACQUIRED(status_variable)           GET_STATUS_BIT(status_variable, \
                                                            STATUS_BIT_IP_ACQUIRED)
#define IS_IP_LEASED(status_variable)             GET_STATUS_BIT(status_variable, \
                                                            STATUS_BIT_IP_LEASED)
```

```
#define IS_CONNECTION_FAILED(status_variable)     GET_STATUS_BIT(status_variable, \
                                                        STATUS_BIT_CONNECTION_FAILED)
#define IS_P2P_NEG_REQ_RECEIVED(status_variable)  GET_STATUS_BIT(status_variable, \
                                                        STATUS_BIT_P2P_NEG_REQ_RECEIVED)
#define IS_SMARTCONFIG_DONE(status_variable)      GET_STATUS_BIT(status_variable, \
                                                        STATUS_BIT_SMARTCONFIG_DONE)
#define IS_SMARTCONFIG_STOPPED(status_variable)   GET_STATUS_BIT(status_variable, \
                                                        STATUS_BIT_SMARTCONFIG_STOPPED)




#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /*__SL_CONFIG_H__*/
```

ANEXO DE PLANOS

PLANO DEL BOOSTERPACK DE CONTROL DE MOTORES

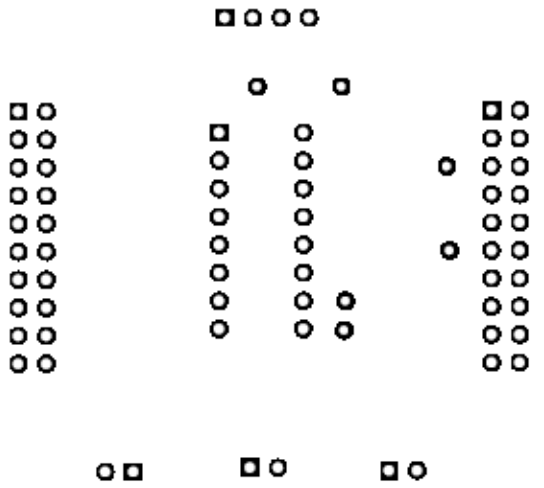


Ilustración 1. Parte Superior (mirando hacia arriba) de la placa. En ésta se colocan los componentes.

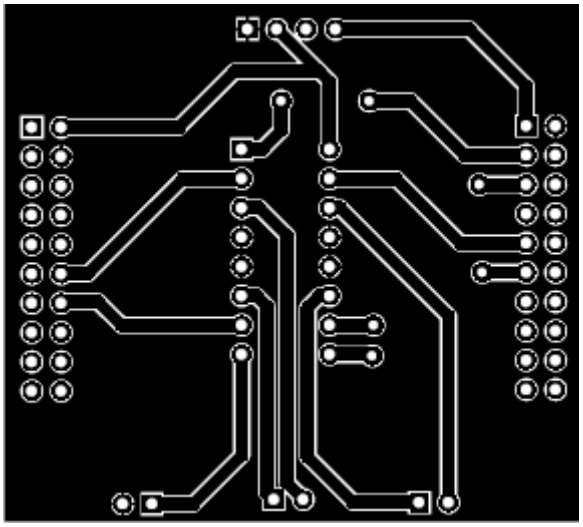


Ilustración 2. Parte inferior (de cara al procesador) de la placa. En ésta se sueldan la mayoría de componentes.

Nota: En el proyecto final sólo se imprimió la parte inferior. Esto dificultó (pero no hizo imposible) la soldadura, ya que los pines se tuvieron que soldar en el poco hueco que dejan.