

ROBOT TANQUE

**Alumno**: Mario Stefano Papetti Funes

**Legajo**: 11807

**Carrera**: Ingeniería en Mecatrónica

Tabla de contenido

[INTRODUCCIÓN 3](#_Toc84780510)

[ESTADO DEL ARTE 3](#_Toc84780511)

[Idea general 3](#_Toc84780512)

[Estado actual 4](#_Toc84780513)

[Implementaciones Futuras 4](#_Toc84780514)

[ASPECTO MECÁNICO 6](#_Toc84780515)

[UNIDAD ALIMENTACIÓN 7](#_Toc84780516)

[UNIDAD DE CONTROL 9](#_Toc84780517)

[PCB ATmega328p 9](#_Toc84780518)

[Raspberry Pi 3 B+ 11](#_Toc84780519)

[PROGRAMACIÓN 11](#_Toc84780520)

[Primera Parte (Placa controladora) 12](#_Toc84780521)

[Segunda Parte (Placa Raspberry PI 3B+) 12](#_Toc84780522)

[Tercera Parte (Solicitante/Cliente) 12](#_Toc84780523)

[Anexos 13](#_Toc84780524)

[Control RF 13](#_Toc84780525)

[Resumen 13](#_Toc84780526)

[PCB 13](#_Toc84780527)

[Programa 15](#_Toc84780528)

[Fotos 17](#_Toc84780529)

[Programas 17](#_Toc84780530)

[Primer Programa (ATmega328p) 17](#_Toc84780531)

[Segundo Programa (Raspberry PI 3 B+) 18](#_Toc84780532)

[Tercer Programa (Cliente) 18](#_Toc84780533)

# INTRODUCCIÓN

El presente informe corresponde en el abordaje de un sistema mecatrónico desde el punto de vista de la programación orientada a objetos, con el fin de poder pulir los conceptos relacionados a la asignatura Programación Orientada a Objetos dictada en el año 2021.

El sistema modelado en su estado del arte se compone por un robot móvil capaz de realizar tareas de forma manual como tomar captura de imágenes, grabar videos, grabar solamente audio, recopilar información útil del medio para su posterior o inmediato análisis y accionamiento, a parte del de desplazarse (las acciones multimedia se realizan mediante la inserción de un dispositivo móvil reciclado que debe de accionarse de forma manual).

# ESTADO DEL ARTE

## Idea general

El robot tanque puede desenvolverse en vario ámbitos cómo vigilancia, seguridad, reconocimientos perimetrales, aplicaciones de machine learning, robot interactivo de entretenimiento. El mismo logra llevar a cabo dichas tareas gracias a la vinculación de diversos dispositivos electrónicos, eléctricos, mecánicos y algoritmos.

## Estado actual

En la instancia actual, el robot se controla mediante una aplicación y un módulo bluetooth, en la versión anterior funcionaba mediante un control de radiofrecuencia dónde solo tiene habilitado los movimientos en el plano (Ver sección de Anexos para mayor detalle de la solución por radiofrecuencia).

El robot está conformado por una estructura impresa en 3D de ABS, internamente funciona con un pack de celdas 18650 y su respectivo módulo de carga para 3 en serie (BMS 3S)

A continuación, se presenta una foto (*figura n°1*) del robot siendo controlado por un control a radiofrecuencia, cómo ya se mencionó anteriormente se deja la presentación de la imagen cómo representativa del robot ya que actualmente ha tenido ligeros cambios físicos y respecto a su control, que actualmente funciona mediante Bluetooth con una aplicación realizada con APP Inventor.

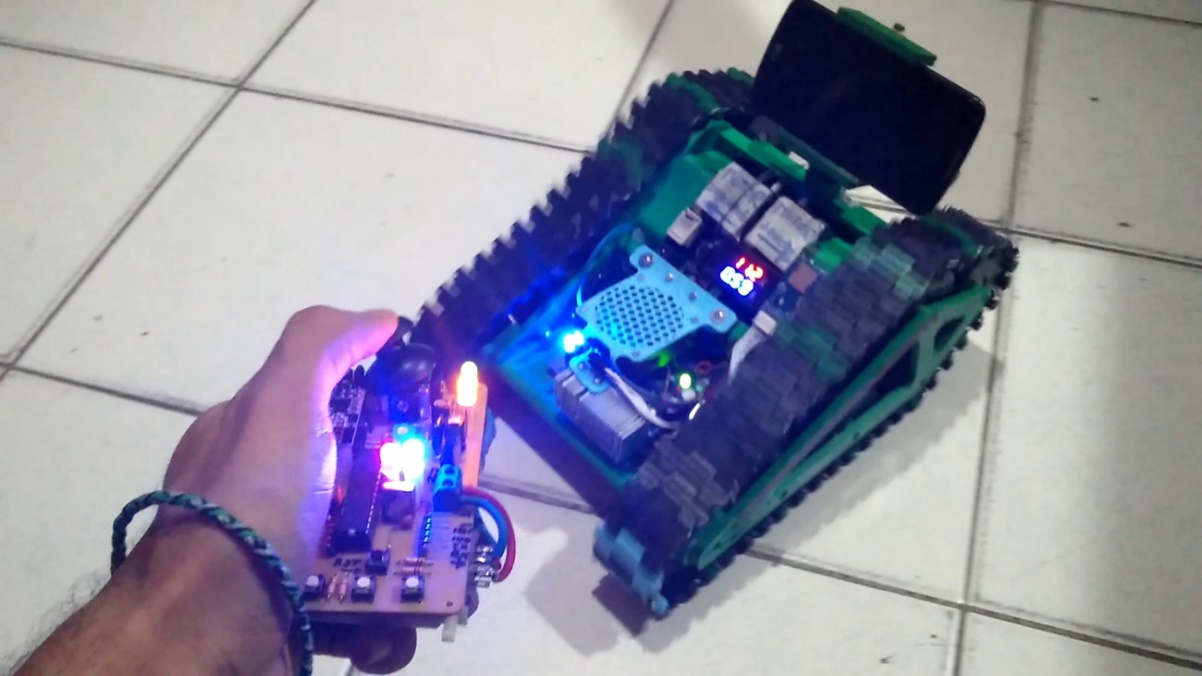


Figura n°1.

Como anteriormente se mencionó, el robot es capaz de realizar movimientos dentro del plano, lo cual consiste en la capacidad de poder recorrer este mismo, sin pendientes pronunciadas mayores a 30° o pi/6.

En este último se aclara que puede desplazarse en dicho plano con 4 movimientos básicos, lo cuales son: avanzar, retroceder, girar sobre su mismo eje en sentido horario y en sentido antihorario.

## Propuesta de proyecto

Se desea realizar dicha comunicación reemplazando el control bluetooth de la aplicación móvil por el control bluetooth mediante la PC cómo interfaz y la capacidad de poder controlar dicho robot bajo la idea de una comunicación servidor-cliente para que otro dispositivo pueda controlarlo de forma remota utilizando el protocolo XML-RPC, montando en la parte del servidor un conjunto de algoritmos en Python que realizarán la comunicación vía Bluetooth con el robot y desplegarían la función de servidor para conectarse con otras aplicaciones o clientes bajo la sintaxis de C++.

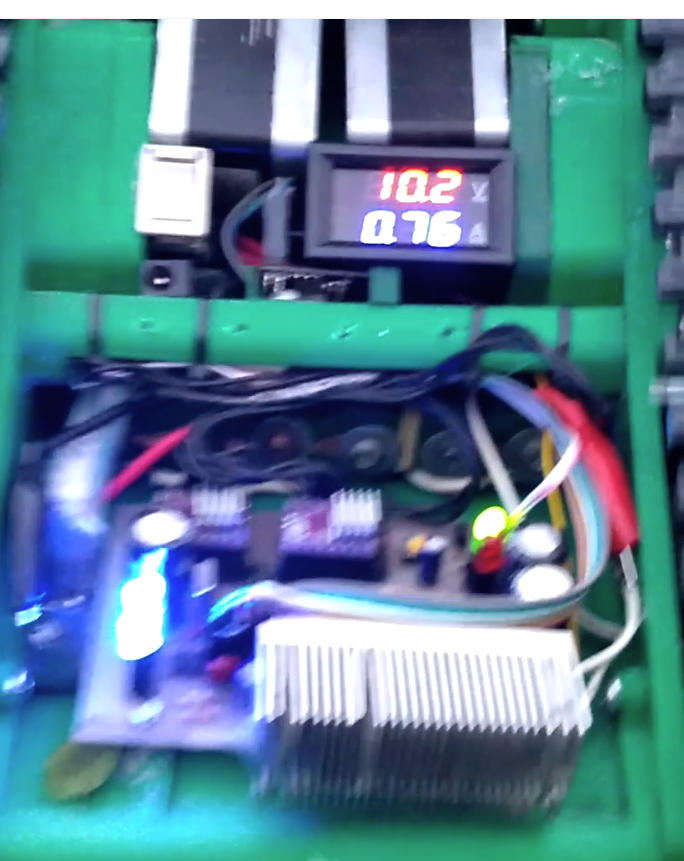


Figura n°2.

# ASPECTO MECÁNICO

En la parte donde la mecánica predomina se hace inferencia tanto la transmisión que utiliza cómo la etapa electromecánica que requiere el mismo para sus acciones.

En la siguiente foto (*figura n°4*) se puede observar la estructura del robot que ha sido diseñada por el usuario Staindis sustraído desde la página Thingiverse. A continuación, se deja el link del creador del proyecto original y la imagen (*figura n°3*) representativa bajo la licencia que dicho proyecto se encuentra.

<https://www.thingiverse.com/thing:2414983>

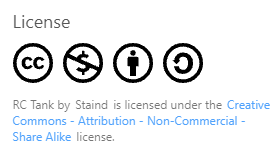


Figura n°3.

En este aspecto se verá la estructura pura del robot móvil, cómo se muestra en la *figura n°4* en donde se observa el robot sin más que la estructura, sacando de lado el celular de la fotografía y demás sutiles componentes que ayudan a la fijación del mismo podemos decir que el tanque está íntegramente formado por piezas impresas en 3D en especial están hechas en material ABS.



Figura n°4.

En cuanto a la dinámica vemos que el mismo es traccionado por dos motores paso a paso que han sido reciclados de alguna impresora en desuso, estos motores poseen en su eje un piñón que engrana a una pequeña caja reductora (ambos impresos en ABS) ya que el modelo estaba pensado para ser usado solamente motores brushless de aeromodelismo, los cuales poseen una elevada velocidad pese a su peso. La pequeña etapa de reducción conecta con un rodillo engranado el cual termina transmitiendo el movimiento del robot a las orugas, la cual está conformada por pequeños eslabones impresos en 3D y unidos con alambre.

# UNIDAD ALIMENTACIÓN

La fuente de energía capaz de darle la potencia que requiere tanto para la etapa mecánica, cómo para poder sostener la unidad de control y las comunicaciones, son las celdas 18650 de Ion-Li.

Para el robot tanque se han implementado un “pack” de 6 celdas conformadas por 3 pares en serie de las mismas, ya que cada celda posee unos 4,2 v dando en total una diferencia de potencial total de 12,6 v.

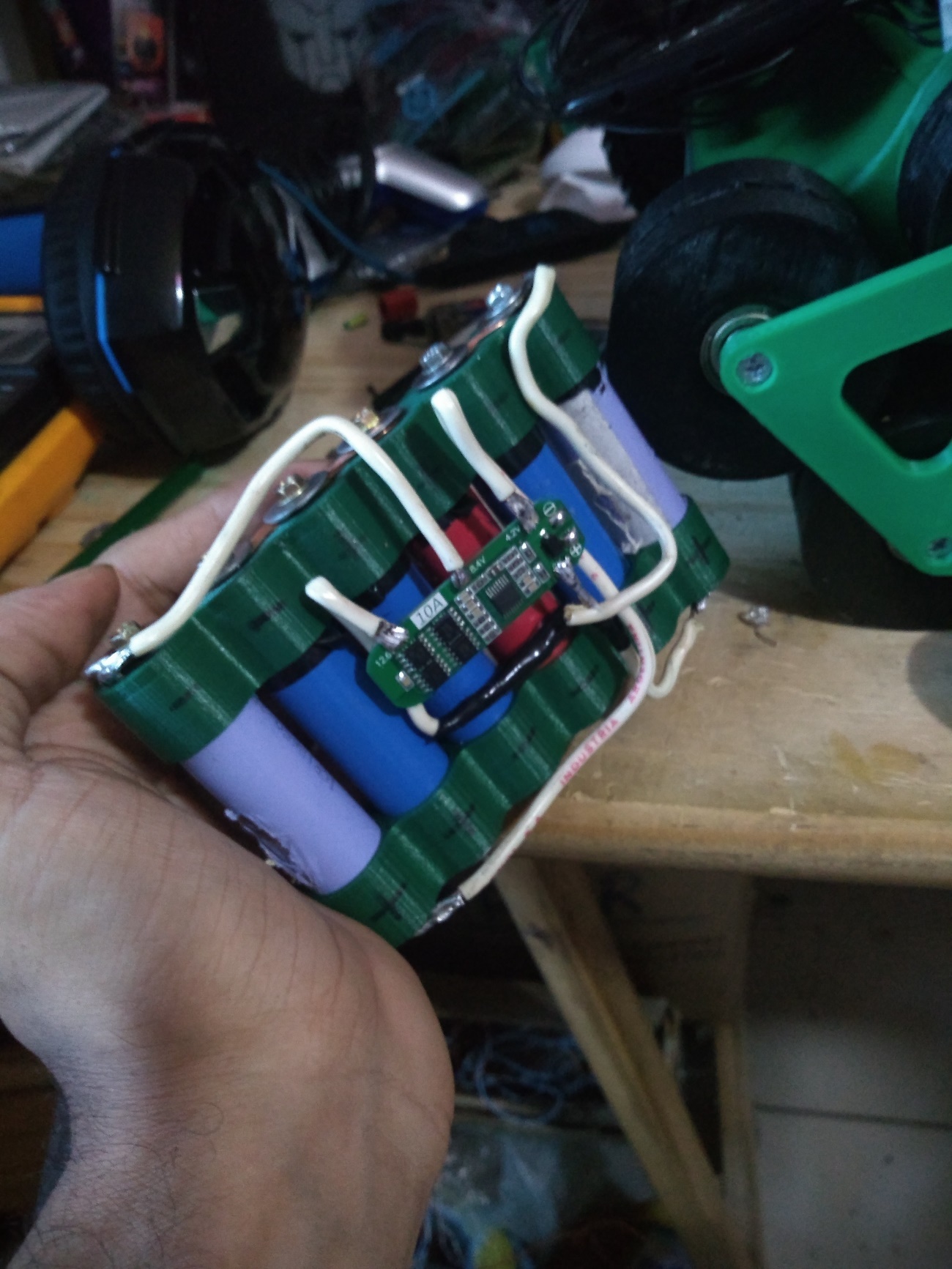


Figura n°5.

En la foto representada en la *figura n°5* se puede observar el módulo que anteriormente denominamos cómo “pack”, cabe rescatar que dichas celdas 18650 Ion-Li son recicladas de diversas baterías donde fallaban por una celda aislada o simplemente por la obsolescencia programada que poseen en las placas que cargan dichas celdas.

# UNIDAD DE CONTROL

La unidad de control del sistema se ve conformada por una placa basada en el modelo de Arduino donde se utiliza cómo eje controlador al microcontrolador ATmega328p, dicha placa ha sido diseñada y fabricada por el presente autor del informe (*figura n°6, n°7 y n°8*). En cada una se detallará tanto las conexiones de todos los periféricos que intervienen en el sistema cómo también aquellos aspectos que conciernen a características electrónicas o de control.

## PCB ATmega328p

La plaqueta electrónica como bien indica su nombre se basó en el microcontrolador ATmega328p para su diseño bajo el software libre de diseño llamado KiCAD.

A continuación, se presenta el esquemático del mismo en la *figura n°6*, el diseño propiamente dicho de la placa en la *figura n°7* y el modelo 3D del mismo en la *figura n°8*.

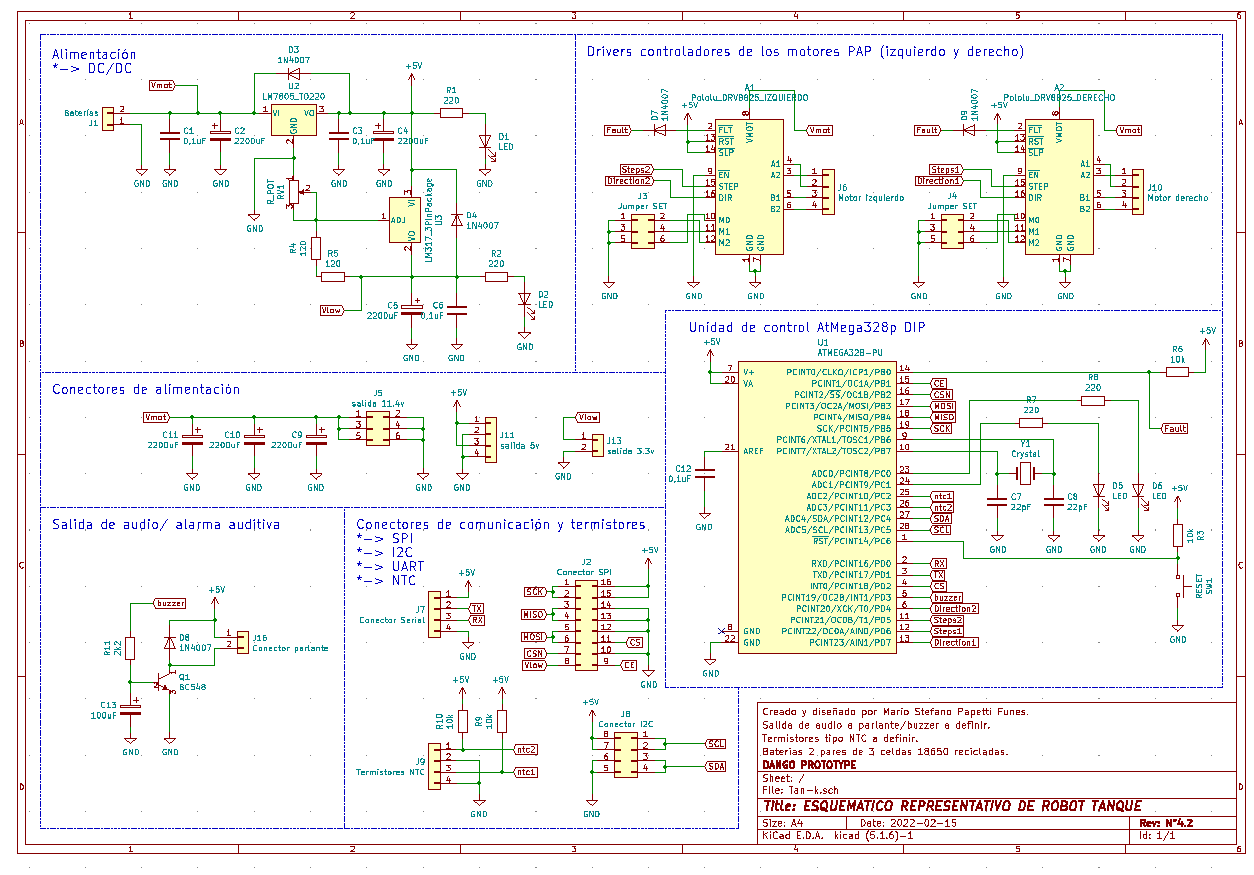


Figura n°6.

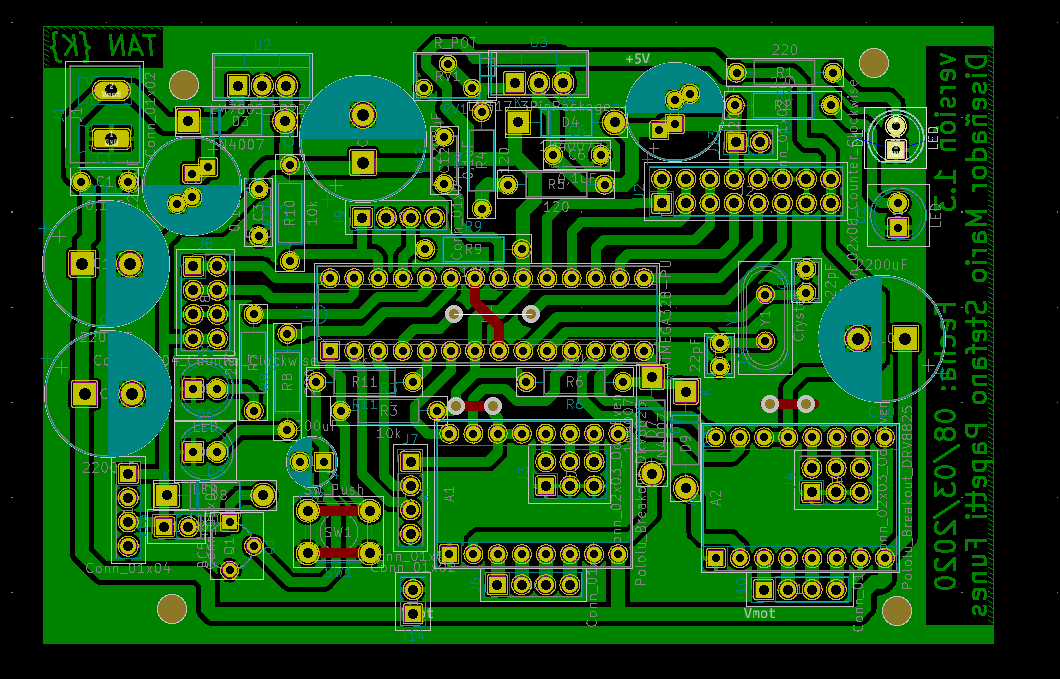


Figura n°7.

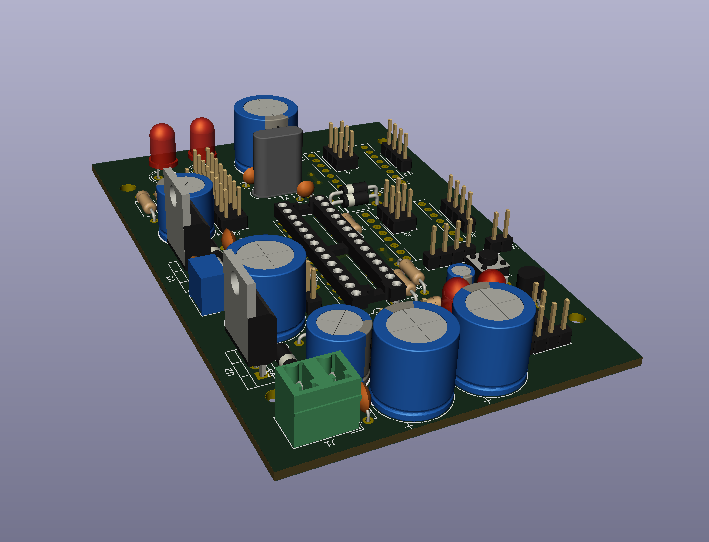


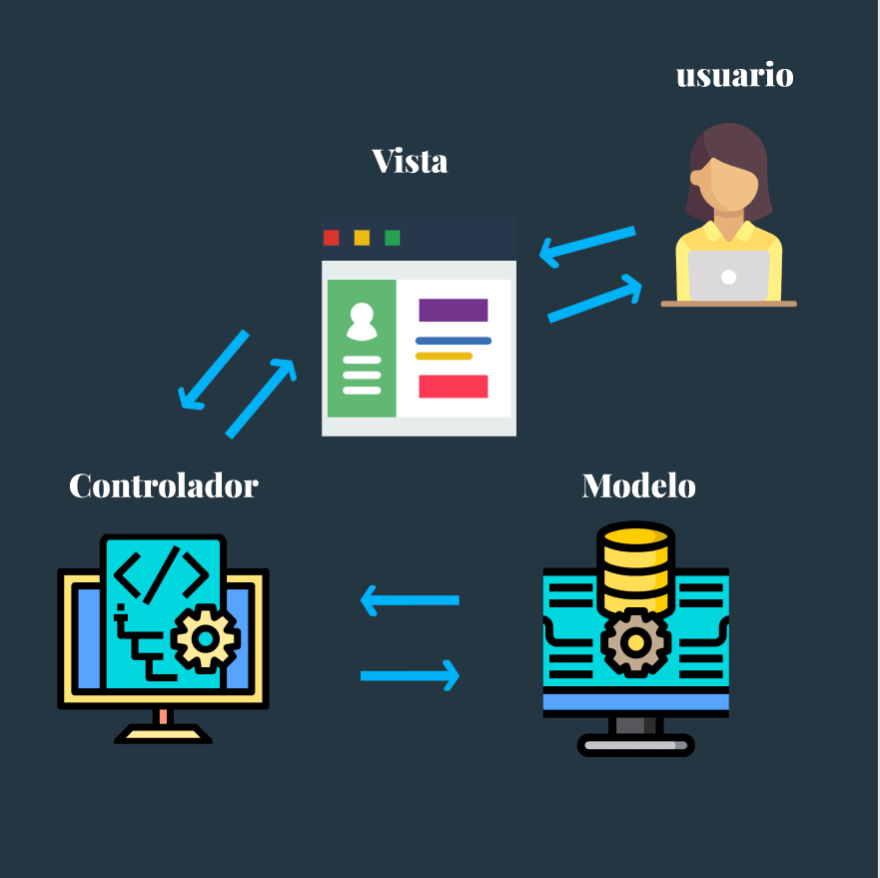
Figura n°8.

# PROGRAMACIÓN

El enfoque que se le da en el actual estado del proyecto será bajo el concepto de servidor cliente, donde el servidor (*en Python*) estará funcionando desde una computadora portátil y el cliente (*en C++*) en otro dispositivo del mismo tipo, ambos estarán funcionando bajo la misma red que en este caso será la red doméstica inalámbrica del presente autor, cabe aclarar que tanto el servidor cómo el cliente también podría funcionar bajo el mismo dispositivo y por medio de la comunicación local o *LocalHost* (127.0.0.1).

El código y archivos que conlleva el presente proyecto se encuentran en el siguiente enlace de GitHub: [**https://github.com/Mario-dango/poo\_libre**](https://github.com/Mario-dango/poo_libre)

Se aplican en los siguientes sub ítems modelado UML para el entendimiento del proyecto, tanto por el lado del servidor como el del cliente. Además, se aplicó el método de diseño de software del estilo modelo vista controlador (MVC) que consta de ciertos pasos para poder crear un sistema más ordenado y prolijo, brindando así su posibilidad en escalar en futuras actualizaciones.



Si bien en el modelo usualmente se realiza con integración con base de datos relacionales o documentales, para el presente proyecto proponer dicha implementación sería interesante aunque queda fuera del alcance del mismo propuesto por la asignatura.

## Primera Parte (Placa controladora)

La primera parte cómo el nombre lo indica es aquella en donde el programa estará en el corazón del robot móvil, esta parte estará en la placa controladora en el microcontrolador ATmega328p. La misma se desarrolla bajo el lenguaje C++ modificado con el IDE de Arduino en la versión 1.8.12.

En esta parte se establece la comunicación con el módulo bluetooth HC-06 para enlazar una comunicación vía bluetooth con el servidor, este recibirá cómo parámetros los correspondientes al movimiento básico y la habilitación de los motores, los mismos se detallan a continuación:

* Al recibir el dato b‘A’ el robot habilita los motores.
* Al recibir el dato b‘B’ el robot deshabilita los motores.

A partir de ahora en adelante se consideran que los casos siguientes los motores se encuentran habilitados.

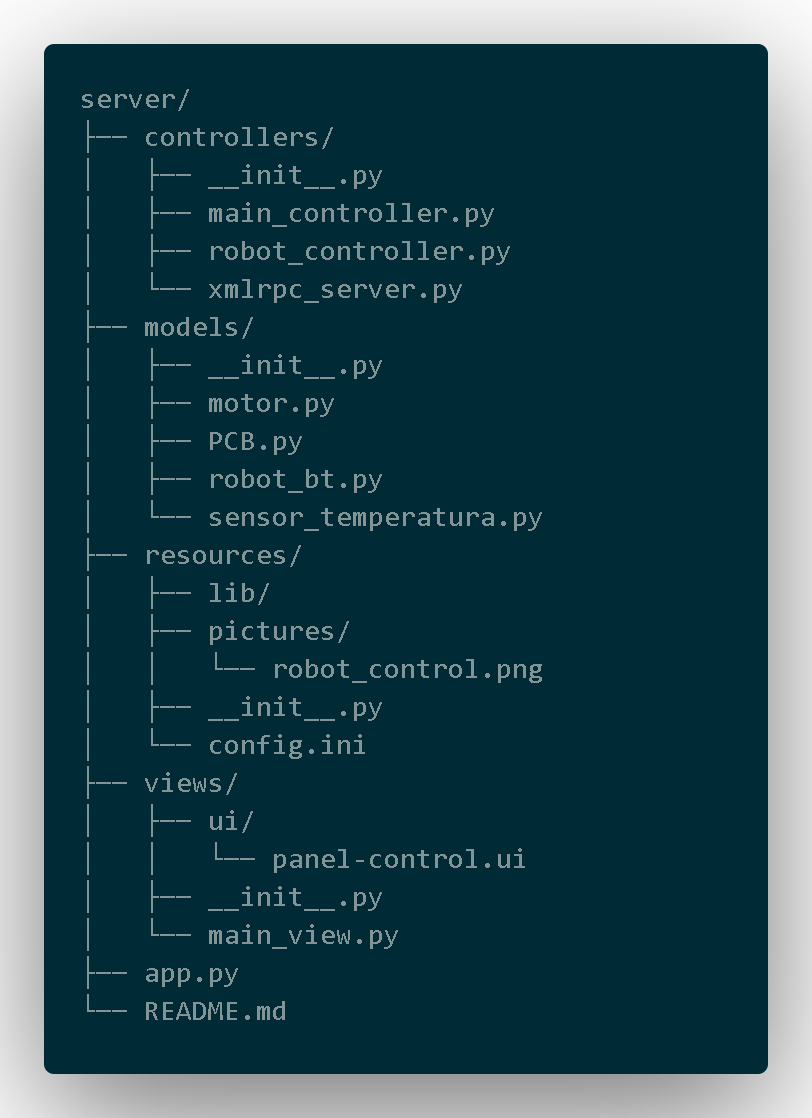
* Al recibir el dato b’8’ el robot comienza a avanzar hacia adelante.
* Al recibir el dato b’2’ el robot comienza a avanzar hacia atrás.
* Al recibir el dato b’6’ el robot comienza a girar hacia la derecha.
* Al recibir el dato b’4’ el robot comienza a girar hacia la izquierda.
* Al recibir el dato b’5’ el robot detiene el movimiento de los motores.

Actualmente no posee la configuración para admitir lectura de sensores, la idea es poder agregarle más adelante sensores cómo proximidad, corriente y temperatura.

El código correspondiente a la primera parte se encuentra en la sección de anexo.

## Segunda Parte (Servidor)

La segunda parte radica en la integración de las solicitudes que llegan del lado del cliente y las que debe de enviar al robot para poder cumplir con lo solicitado. En este lado se encuentra el servidor armado por distintos archivos, aplicando el diseño de modelo vista controlador. En la siguiente figura se puede ver su distribución de carpetas y archivos.



A continuación, se presenta el siguiente diagrama de clases que da la relación entre las clases que se encuentran del lado del servidor

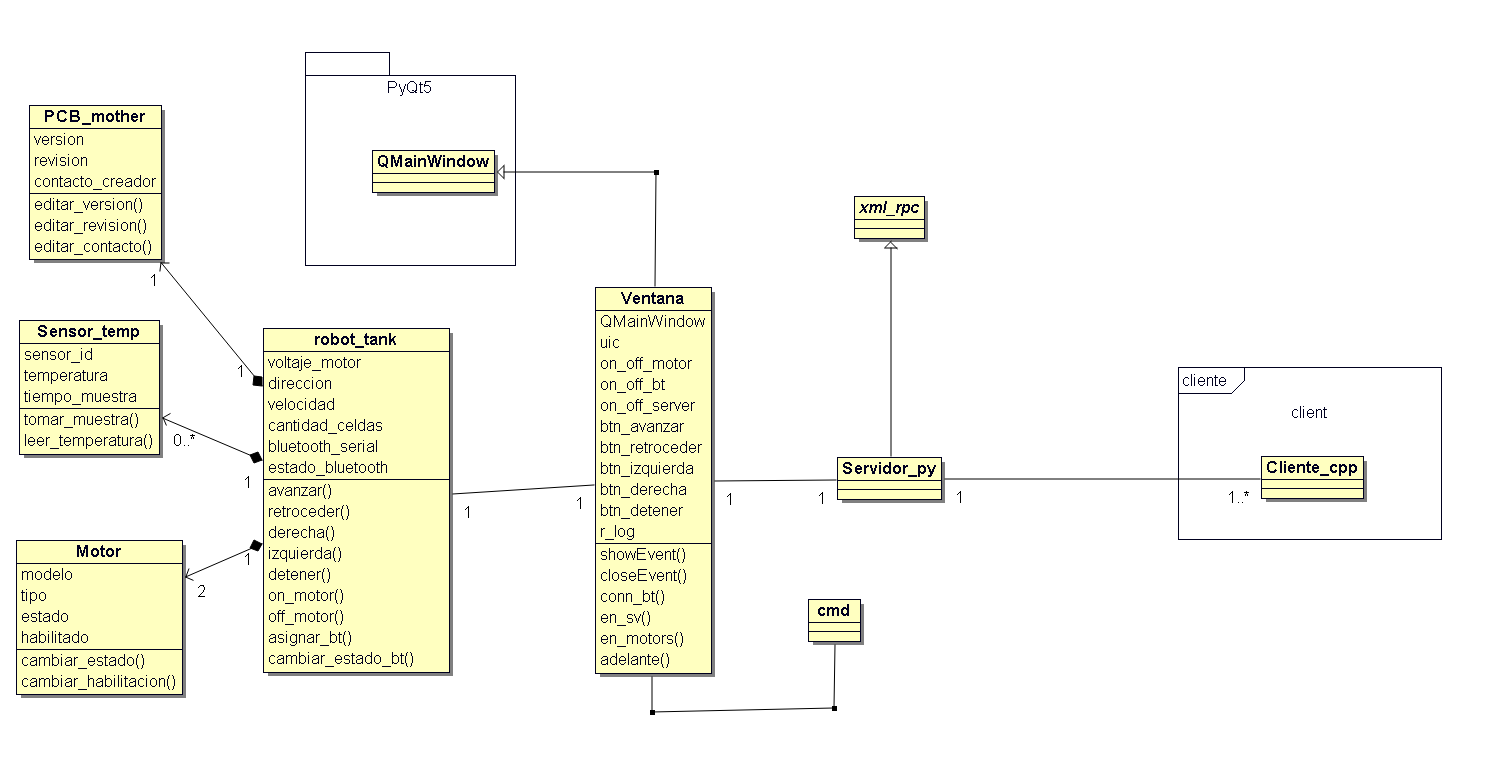


Figura n°9.

Cómo se puede observar de la figura N°9 del diagrama de clases se tiene una clase llamada robot\_tank la cual tendrá relación de composición a las clases motor, sensor\_temp y PCB\_mother, ya que las mismas no existirían si el objeto robot no existe o es destruido. Se tiene en cuenta que esta clase es quien se comunicará con el robot físico, por medio de bluetooth enviando las consignas correspondientes. También se puede observar cómo está clase interactúa con la clase ventana. Por último, se dejan estás cuatro clases en el mismo archivo llamado robot\_bt.py

Para la clase ventana se toma en cuenta que el objeto que la misma instancia es la interfaz visual que el usuario en la parte del servidor con la cual podrá interactuar directamente con el robot e iniciar el servicio XML-RPC, la misma está basada de PyQt5 y hereda funcionalidades de la clase QMainWindow para el funcionamiento del mismo.

En la siguiente figura (*figura N°10*) se puede observar la interfaz del servidor.

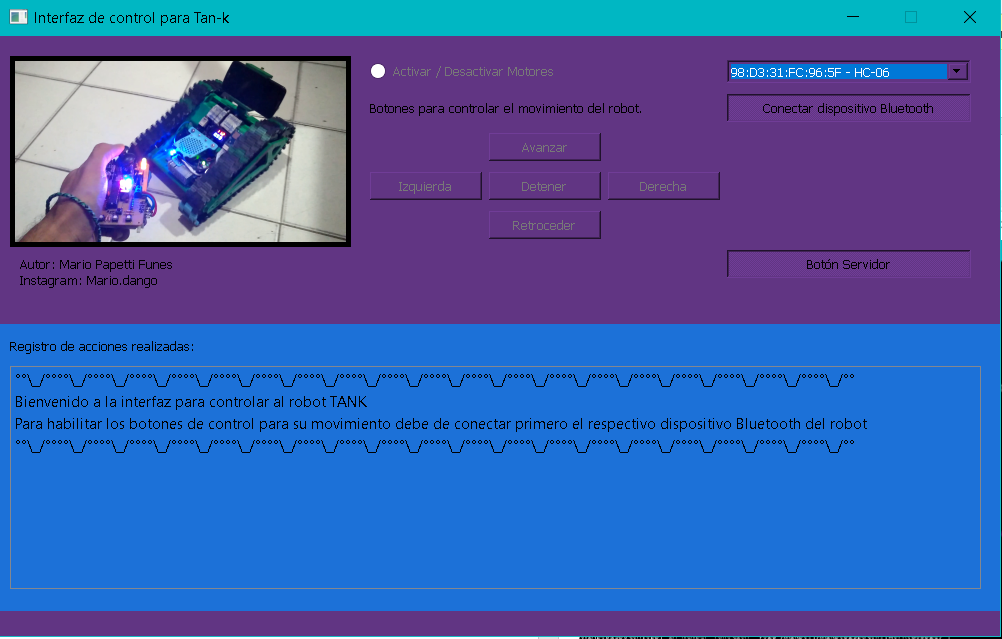


Figura n°10.

A continuación, se presenta en la *figura N°11* marcado los ítems que conforman la interfaz que interactuará con el usuario, el robot y los diversos clientes:

.

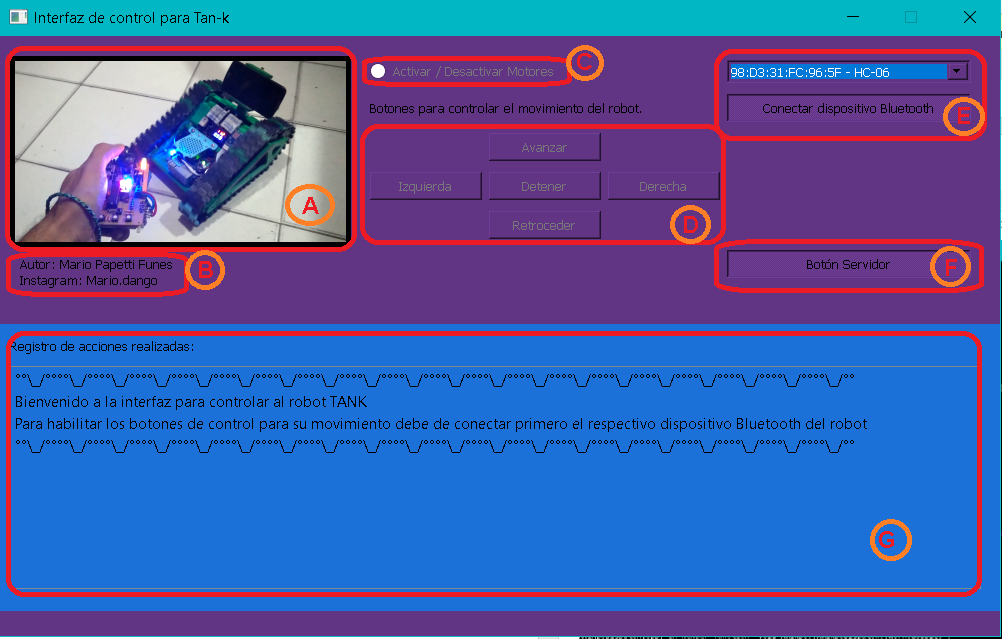


Figura n°11.

A continuación, se detallan los ítems que se visualizan en orden alfabético:

1. Corresponde a una etiqueta representativa del robot a controlar.
2. Otra etiqueta que posee algunos datos del desarrollador.
3. Botón para habilitar y deshabilitar los motores del robot.
4. Botones para el control direccional del robot, tales son: Avanzar, Retroceder, Derecha, Izquierda y Detener, los mismos al igual que el botón de habilitación de motores estarán inhabilitados hasta tanto y en cuanto no se establezca la conexión con el robot.
5. Apartado con menú desplegable para la selección de la dirección MAC asociada al módulo bluetooth del robot. También el botón correspondiente para poder realizar la conexión con el dispositivo.
6. Botón para iniciar el servidor XML-RPC con el cual podrá establecer comunicación con los distintos clientes que lo requieran.
7. Ventana de registro de acciones, en donde cada acción tanto por parte del servidor cómo del cliente se verán reflejadas en la misma (conexión exitosa por bluetooth, movimiento del robot, on/off del XML-RPC, etc.).

## Tercera Parte (Solicitante/Cliente)

La tercera y última parte del sistema será la encargada de poder generar las solicitudes que requiere el servidor para poder satisfacer las necesidades del o los usuarios que lo soliciten, para ello también se ha modelado un diagrama de clases correspondiente a la vista desde la perspectiva del cliente como se puede observar en la figura N°12.

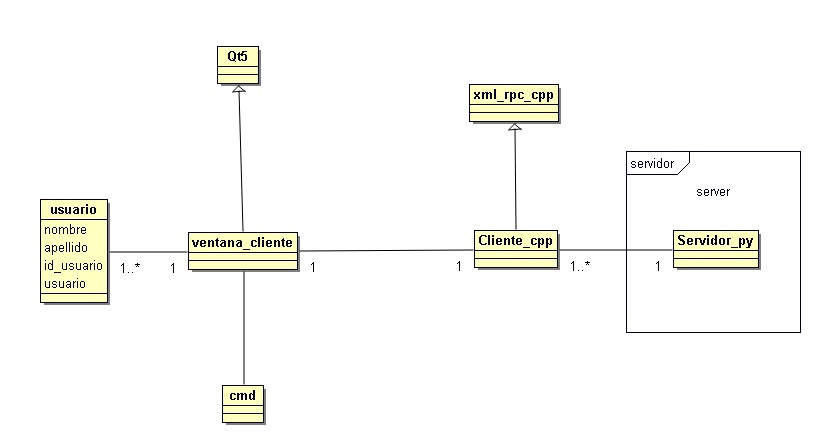


Figura n°12.

(FALTA COMPLETAR)

Actualmente se encuentra en desarrollo la parte gráfica del lado del cliente por medio de Qt5 similar al del servidor con unos ligeros cambios de vital importancia, además del agregado de registro e ingreso de usuario para el control del robot. En la siguiente figura N°13 se puede observar las distintas ventanas que el cliente conlleva.

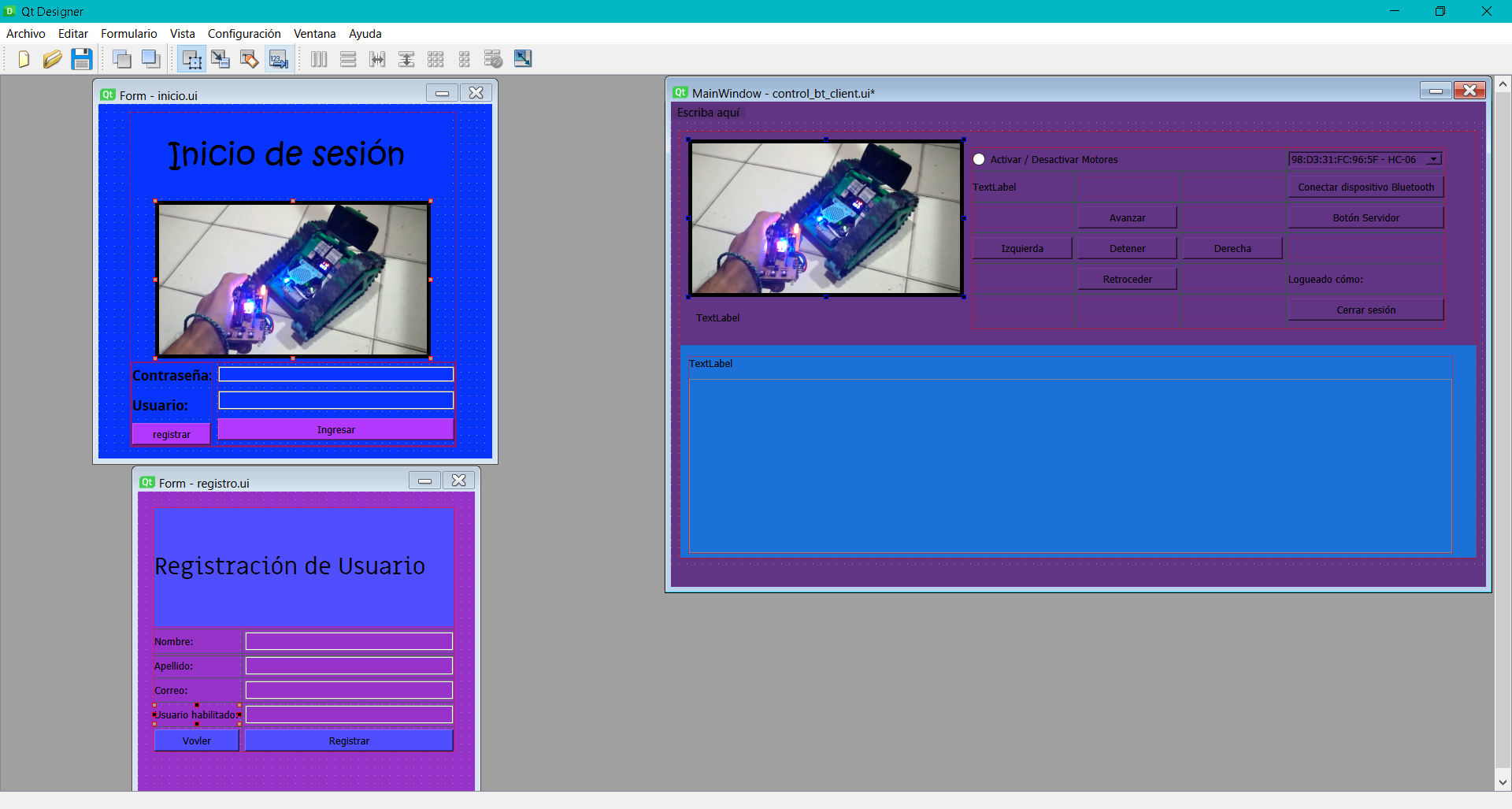


Figura n°13.

Actualmente se encuentra un error en la compilación por parte de las librerías de funcionalidad con XML-RPC, las mismas se han obtenido desde la siguiente fuente: <http://sourceforge.net/projects/xmlrpcpp/>

En la siguiente figura se puede observar el error que larga al intentar ejecutar la siguiente línea de comando:

g++ -Wall cliente\_basico.cpp XmlRpcClient.cpp XmlRpcUtil.cpp XmlRpcValue.cpp XmlRpcSocket.cpp XmlRpcSource.cpp XmlRpcDispatch.cpp -o cliente

La cual corresponde a la compilación de las dependencias sin la integración de Qt5.

# Anexos

## Control RF

### Resumen

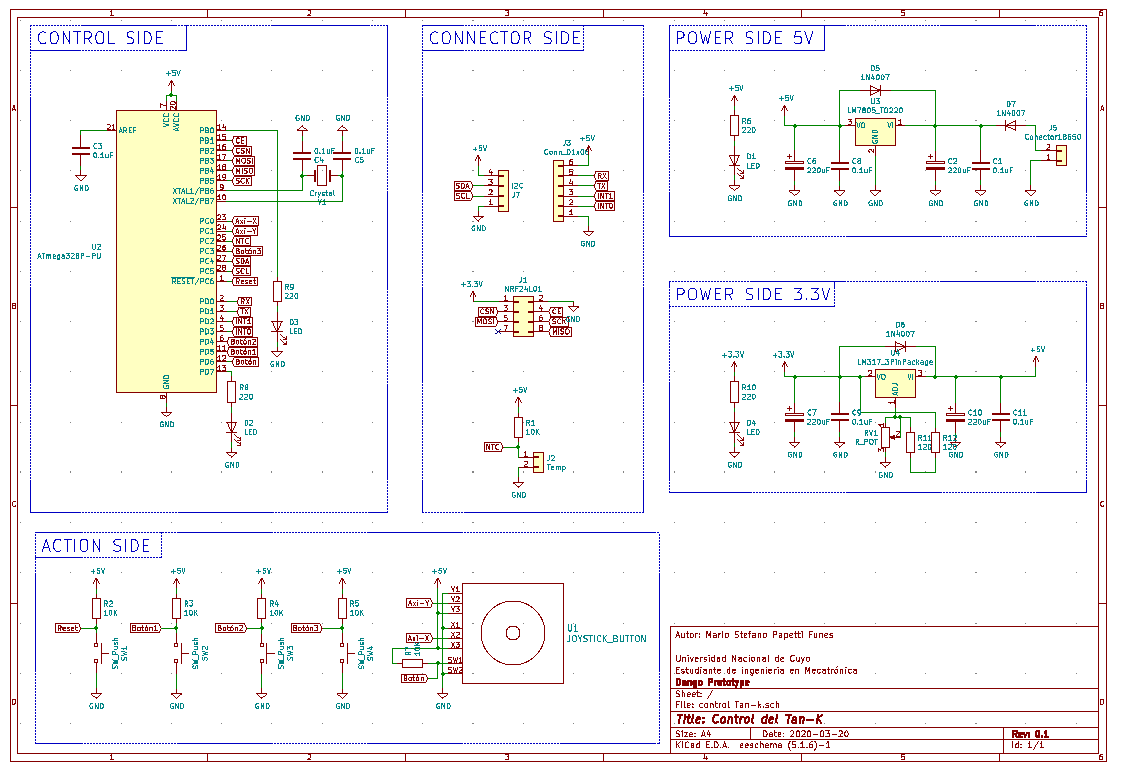
La idea principal era poder verificar la viabilidad mecánica del robot de una forma controlada por el usuario para luego poder llevarlo desde otra perspectiva automatizada, se podría decir que este fue el primer paso dónde se pudieron ver las falencias mecánicas y corrección de las mismas en conjunto con el hardware de control que posee.

A continuación, se presenta tanto algunas fotos representativas del mismo cómo también se detallará el esquema del control.

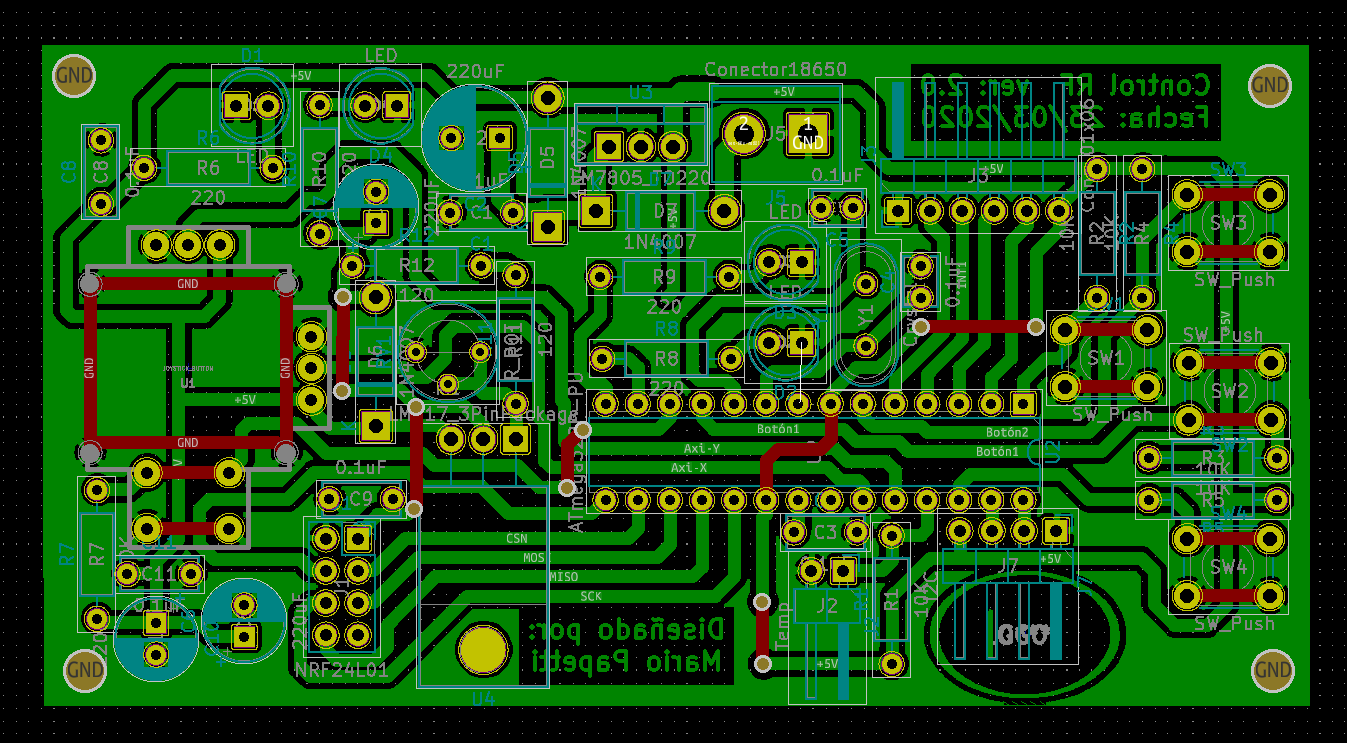
### PCB

La placa ha sido diseñada con el programa de software libre llamado KiCAD en el cual a continuación se presentan 3 representaciones graficas del control, las cuales son:

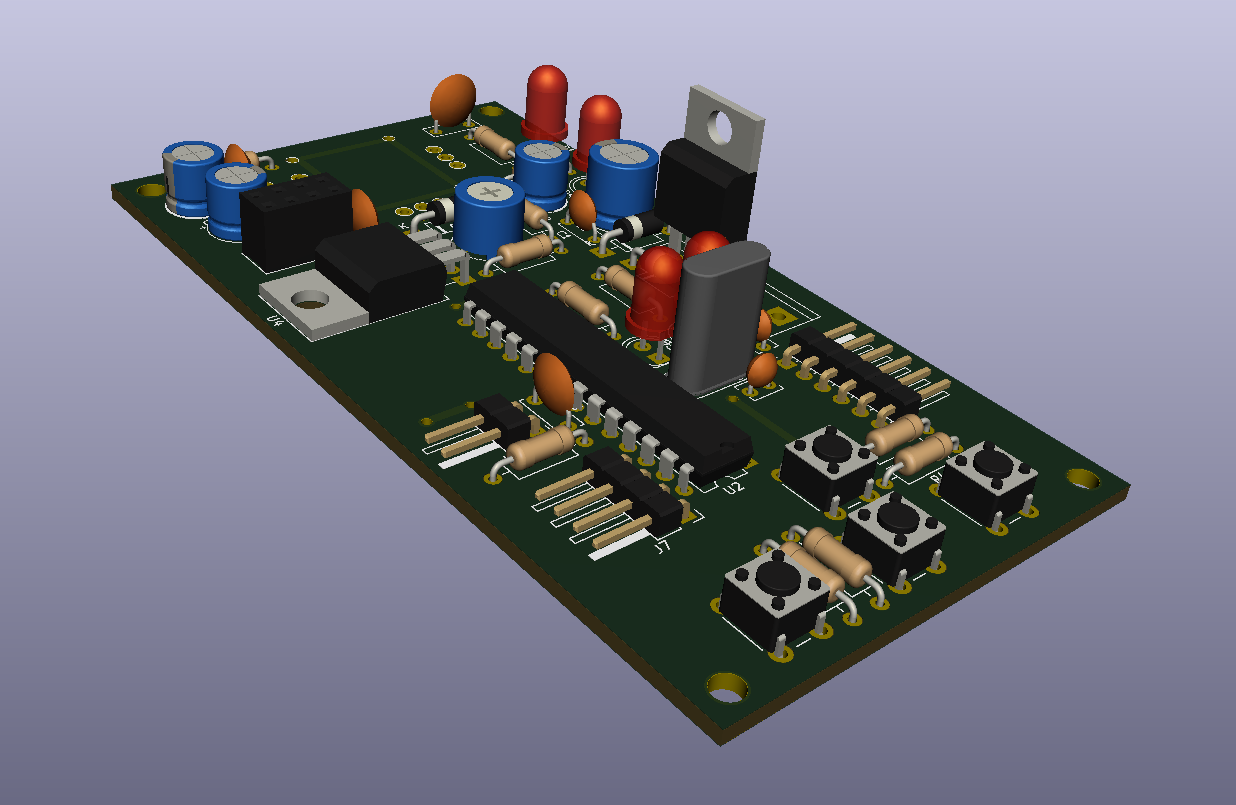
* Esquema electrónico
* Diseño del PCB
* Representación 3D



A continuación, se presenta una imagen constructiva de la disposición de cada elemento representativo del esquema anteriormente presentado.



En la siguiente imagen se presenta el modelo 3D en conjunto con el motor grafico de FreeCAD, otro software libre. En otra versión del informe posiblemente se agregue las modificaciones correspondientes en la asociación de los modelos 3D cómo por ejemplo en el analógico, aunque al estar descontinuada esta versión no se le harán cambios.



### Programa

Se agrega en esta sección el programa el cuál en su última versión quedo en el microcontrolador, el mismo fue diseñado bajo el IDE de Arduino.

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

//Declaremos los pines CE y el CSN

#define CE\_PIN 9

#define CSN\_PIN 10

//Variable con la dirección del canal por donde se va a transmitir

byte direccion[5] ={'t','a','n','-','k'};

//creamos el objeto radio (NRF24L01)

RF24 radio(CE\_PIN, CSN\_PIN);

#define SW\_pin  6

#define Y\_pin  A0

#define X\_pin  A1

#define boton1  5             //Botón

#define boton2  4             //Botón

#define boton3  A3           //La entrada analogica la tomo como digital

#define ntc  A2           //Sensor de temperatura para las 18650.

#define ledOK  8          //Led indicador de emparejamiento del NRF24L01

#define alerta 7           //Led indicador de alerta.

float datosValores[7];

bool flagmed = false;

bool flagIZQ = false;

bool flagSTOP = false;

bool flagDER = false;

void setup()

{

  //inicializamos el NRF24L01

  radio.begin();

  //inicializamos el puerto serie

  Serial.begin(9600);

//Abrimos un canal de escritura

 radio.openWritingPipe(direccion);

  //Configurando el Joystick0

  pinMode(SW\_pin, INPUT);

  pinMode(boton1, INPUT);

  pinMode(boton2, INPUT);

  pinMode(boton3, INPUT);

  pinMode(ledOK, OUTPUT);

  pinMode(alerta, OUTPUT);

  digitalWrite(alerta, false);

  digitalWrite(ledOK, false);

}

void loop(){

 if (!digitalRead(boton1)){

   if (!flagDER){

     flagDER = true;

     datosValores[3]= 1;

   }else{

     flagDER = false;

     datosValores[3]= 0;

   }

 }

 if (!digitalRead(boton2)){

   if (!flagSTOP){

     flagSTOP = true;

     datosValores[4]= 1;

   }else{

     flagSTOP = false;

     datosValores[4]= 0;

   }

 }

 if (!digitalRead(boton3)){

   if (!flagIZQ){

     flagIZQ = true;

     datosValores[5]= 1;

   }else{

     flagIZQ = false;

     datosValores[5]= 0;

   }

 }

 if (!digitalRead(SW\_pin)){

   if (!flagmed){

     flagmed = true;

     datosValores[2]= 1;

   }else{

     flagmed = false;

     datosValores[2]= 0;

   }

 }

 //cargamos los datosValores en la variable datosValores[]

 datosValores[0]= analogRead(X\_pin);

 datosValores[1]= analogRead(Y\_pin);

 //datosValores[2]= digitalRead(SW\_pin);

 //datosValores[3]= digitalRead(boton1);

 //datosValores[4]= digitalRead(boton2);

 //datosValores[5]= digitalRead(boton3);

 datosValores[6]= analogRead(ntc);

 datosValores[0] = (-1)\*round(datosValores[0]/100 - 5);

 datosValores[1] = (-1)\*round(datosValores[1]/100 - 5);

 //enviamos los datos

 bool ok = radio.write(datosValores, sizeof(datosValores));

  //reportamos por el puerto serial los datos enviados

  if(ok)

  {

     digitalWrite(8, true);

     digitalWrite(7, false);

     Serial.println("Se ha establecido la comunicación.");

  }

  else

  {

     digitalWrite(8, false);

     digitalWrite(7, true);

     Serial.println("no se ha podido enviar");

  }

}

### Fotos

Algunas imágenes del control remoto en su funcionamiento.

## Programas

### Primer Programa (ATmega328p)

El siguiente fragmento de código fue escrito bajo el IDE de Arduino en su versión 1.8.16. El mismo se comenzó a escribir el 10/10/2021 hasta su última versión actualmente, pasando por las versiones:

* Versión 1.0 con fecha del sábado 10 de octubre de 2021.
* Versión actual: 1.0 (Abierto a modificaciones).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Titulo

 \* Autor: Mario Stefano Papetti Funes

 \* Versión: 1.0

 \* Fecha: 10/10/2021

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* El programa recibe argumentos tales como:

 \* [A; R; D; I; S] Correspondientes al movimiento, vienen en conjunto con la potencia.

 \*    Ejemplo: "A:30" -> El tanque avanzará adelante con velocidad 30% nominal.

 \*             "S" -> El tanque se detendrá.

 \* [T] correspondiente a petición de lectura a los termistores (NTC1 y NTC2).

 \*    Ejemplo: "T:1" -> devuelve resistencia y temperatura actual de la NTC número 1.

 \* [H] Corresponde a la petición de información del firmware.

 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Extras o fuentes usadas para la construcción del algoritmo

 \*

 \* Fuente de lectura de NTC

 \*  https://www.geekfactory.mx/tutoriales-arduino/termistor-ntc-con-arduino-como-sensor-de-temperatura/

 \*

 \*/

//Inclusión de librerias para el programa

#include <AccelStepper.h>

//Declaraciones referentes a la naturaleza y origen del firmware

const char autor[] = "Mario Stefano Papetti Funes";

const char vers[] = "v1.0";

const char fecha[] = "10/10/2021";

//Etiquetas definidas para los pines que usan los drivers DRV8825 para controlar los motores

#define pulsosDER 5 //Motor derecho visto desde el frente

#define pulsosIZQ 6 //Motor izquierdo visto desde el frente

#define dirDER 7    //dirección del motor derecho

#define dirIZQ 4    //dirección del motor izquierdo

#define EnDER A5    //Enable motor Derecho

#define EnIZQ A4    //Enable motor Izquierdo

#define DRV8825 1   //Tipo de interfaz del motor

#define Fault 8 //Entrada por si hay algún problema con los DRV8825

//Etiquetas definidas para el uso general de pines

#define NTC1 A2  //Entrada para sensar temperaturas o alguna LDR

#define NTC2 A3  //Entrada para sensar temperaturas o alguna LDR

#define temp\_r 10000  //Resistencia en serie con el termistor para divisor de tensión

#define LedALR A1 //Salida led indicadora de que hay un problema

#define LedOKA A0 //Salida led indicadora de que está todo OKAY

//#define buzzer 9 //Salida del Parlante indicador

//const int speakerPin = 9;

//Cosntantes de nivel, para velocidades de pulsos para los motores

const int velMax = 1000;

const int cteVel = 200;

// variable para almacenar la temperatura y resistencia

float temp1 = 0;    //Temperatura correspondiente al termistor en NTC1

uint32\_t res1 = 0;  //Resistencia actual correspondiente al termistor en NTC1

float temp2 = 0;    //Temperatura correspondiente al termistor en NTC2

uint32\_t res2 = 0;  //Resistencia actual correspondiente al termistor en NTC2

//Creo los objetos motor Izquierdo y motor Derecho para posterior uso

AccelStepper motorIZQ = AccelStepper(DRV8825, pulsosIZQ, dirIZQ); //Creamos la instancia motor Izquierdo

AccelStepper motorDER = AccelStepper(DRV8825, pulsosDER, dirDER); //Creamos la instancia motor Derecho

//Pre defino funciones que se utilizan para movimiento o alerta visual

void alert(int tiempo);

boolean serial\_decode(char\* clase, int valor);

boolean mover(char\* clase, int potencia);

boolean leer\_temp(int ntc);

void setup(){

  /\*

  Audio.speakerPin = 9; //Selecciona la salida de audio: pin 9 (UNO y nano) pines 5,6,11 o 46 (Mega)

  Audio.quality(1); // Mejoramos la calidad de sonido (puede ser 0 o 1)

  Audio.setVolume(5); // Se selecciona el volumen: valor entre 0 y 7. Solo funciona bien si la calidad del audio está a 1.

                      //En caso de que no se oiga nada o se  oiga mal, bajad el volumen o borrad la función

  \*/

   //inicializamos el puerto serie

   Serial.begin(9600);

   while(!Serial){};

  // Defino respectivos pines cómo entradas y salidas

   pinMode(Fault, INPUT);

   pinMode(LedOKA, OUTPUT);

   pinMode(LedALR, OUTPUT);

   pinMode(EnDER, OUTPUT);

   pinMode(EnIZQ, OUTPUT);

   digitalWrite(LedOKA, false);

   digitalWrite(LedALR, false);

  //Inicializo los objetos motores y los detengo

   motorIZQ.setMaxSpeed(velMax); //Definimos la velocidad maxima del motor izq

   motorDER.setMaxSpeed(velMax); //Definimos la velocidad maxima del motor der

   digitalWrite(EnDER, true);

   digitalWrite(EnIZQ, true);

   motorIZQ.setSpeed(0);

   motorDER.setSpeed(0);

}

void loop(){

  //Variables para movimiento

  int poder = 0;

  char modo = ' ';

  // Verifico si se enceuntra disponible la comunicación serial

  if (Serial.available()){

    //Defino algunas variables utiles para le decodificación del mensaje

    char mensaje = Serial.read();

    const char s[2] = ":";

    char \*token;

    int cont = 0;

    /\* get the first token \*/

    token = strtok(mensaje, s);

    /\* walk through other tokens \*/

    while( token != NULL ) {

      if (cont == 1){

        poder = atoi(token);

        printf("%i\n", poder);

        break;

      }else{

        modo = token;

      }

      cont ++;

      token = strtok(NULL, s);

    }

    //Una vez descompuesto el string con el modo y la potencia se procede a actuar

    // bajo verificación en el control del if.

    if (serial\_decode(modo, poder)){

      printf("ok");

      digitalWrite(LedOKA, true);

      digitalWrite(LedALR, false);

    }else{

      printf("Error!");

      alert(150);

    }

  }

}

//Función derivadora del codigo recibido

boolean serial\_decode(char\* codigo, int valor){

  if (codigo == 'T'){

    if ((valor == 1) || (valor == 2)){

      leer\_temp(valor);

    }else {

      Serial.print("Error de parametro en Temperatura.");

      return false;

    }

  }else if (codigo == 'H'){

    //Pasa la información cargada del firmware, lo que es autor, versión y fecha de la última modificación.

    Serial.print("V:");

    Serial.println(vers);

    Serial.print("A:");

    Serial.println(autor);

    Serial.print("F:");

    Serial.println(fecha);

    return true;

  }else{

    return mover(codigo, valor);

  }

}

//Función encargada de lectura de los termistores

boolean leer\_temp(int ntc){

  if (ntc == 1){

    // calcular la resistencia electrica del termistor usando la lectura del ADC

    res1 = get\_res(analogRead(NTC1));

    // luego calcular la temperatura segun dicha resistencia

    temp1 = get\_temp(res1);

    Serial.print("R1:");

    Serial.println(res1);

    Serial.print("T1:");

    Serial.println(temp1, 2);

  }else if (ntc == 2){

    // calcular la resistencia electrica del termistor usando la lectura del ADC

    res2 = get\_res(analogRead(NTC2));

    // luego calcular la temperatura segun dicha resistencia

    temp2 = get\_temp(res2);

    Serial.print("R2:");

    Serial.println(res2);

    Serial.print("T2:");

    Serial.println(temp2, 2);

  }else {

    Serial.println("Error");

    return false;

  }

}

//Función encargada del movimiento del tanque

boolean mover(char\* clase, int potencia){

  //analizando el valor de la variable potencia para determinar que clase de valores aparecen

  if (potencia != 0){

    clase = 'S';

  }else if (potencia > velMax){

    potencia = velMax;

  }else if ((potencia < cteVel) && (potencia > 10)){

    potencia = cteVel;

  }

  if (clase == 'S'){

    //Detenerse

    digitalWrite(EnDER, true);

    digitalWrite(EnIZQ, true);

    motorDER.setSpeed(0);

    motorIZQ.setSpeed(0);

  }

  if (clase == 'A'){

    //Avanzar

    digitalWrite(EnDER, false);

    digitalWrite(EnIZQ, false);

    motorIZQ.setSpeed(velMax);

    motorDER.setSpeed(velMax);

  }else if (clase == 'R'){

    //Retroceder

    digitalWrite(EnDER, false);

    digitalWrite(EnIZQ, false);

    motorIZQ.setSpeed(-velMax);

    motorDER.setSpeed(-velMax);

  }else if (clase == 'I'){

    //Girar a la izquierda

    digitalWrite(EnDER, false);

    digitalWrite(EnIZQ, false);

    motorIZQ.setSpeed(velMax);

    motorDER.setSpeed(0);

  }else if (clase == 'D'){

    //Girar a la derecha

    digitalWrite(EnDER, false);

    digitalWrite(EnIZQ, false);

    motorIZQ.setSpeed(0);

    motorDER.setSpeed(velMax);

  }else {

    //stop

    digitalWrite(EnDER, true);

    digitalWrite(EnIZQ, true);

    motorDER.setSpeed(0);

    motorIZQ.setSpeed(0);

    alert(150);

    return false;

  }

   motorIZQ.runSpeed();

   motorDER.runSpeed();

   return true;

}

//Función para dar destello de alerta en el led ALERTA

void alert(int tiempo){

  digitalWrite(EnDER, true);

  digitalWrite(EnIZQ, true);

  for (int i = 0; i < 2; i++){

    delay(tiempo);

    digitalWrite(LedOKA, false);

    delay(tiempo);

    digitalWrite(LedOKA, true);

  }

  return;

}

//Funciones para calcular la resistencia de los termistores

int32\_t get\_res(uint16\_t adcval)

{

  // calculamos la resistencia del NTC a partir del valor del ADC

  return (temp\_r \* ((1023.0 / adcval) - 1));

}

//Función para el calculo de la temperatura en funcion de la resistencia actual

float get\_temp(int32\_t resistance)

{

  // variable de almacenamiento temporal, evita realizar varias veces el calculo de log

  float temp;

  // calculamos logaritmo natural, se almacena en variable para varios calculos

  temp = log(resistance);

  // resolvemos la ecuacion de STEINHART-HART

  // http://en.wikipedia.org/wiki/Steinhart–Hart\_equation

  temp = 1 / (0.001129148 + (0.000234125 \* temp) + (0.0000000876741 \* temp \* temp \* temp));

  // convertir el resultado de kelvin a centigrados y retornar

  return temp - 273.15;

}