

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 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.

# 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 programados.

## 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.

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.

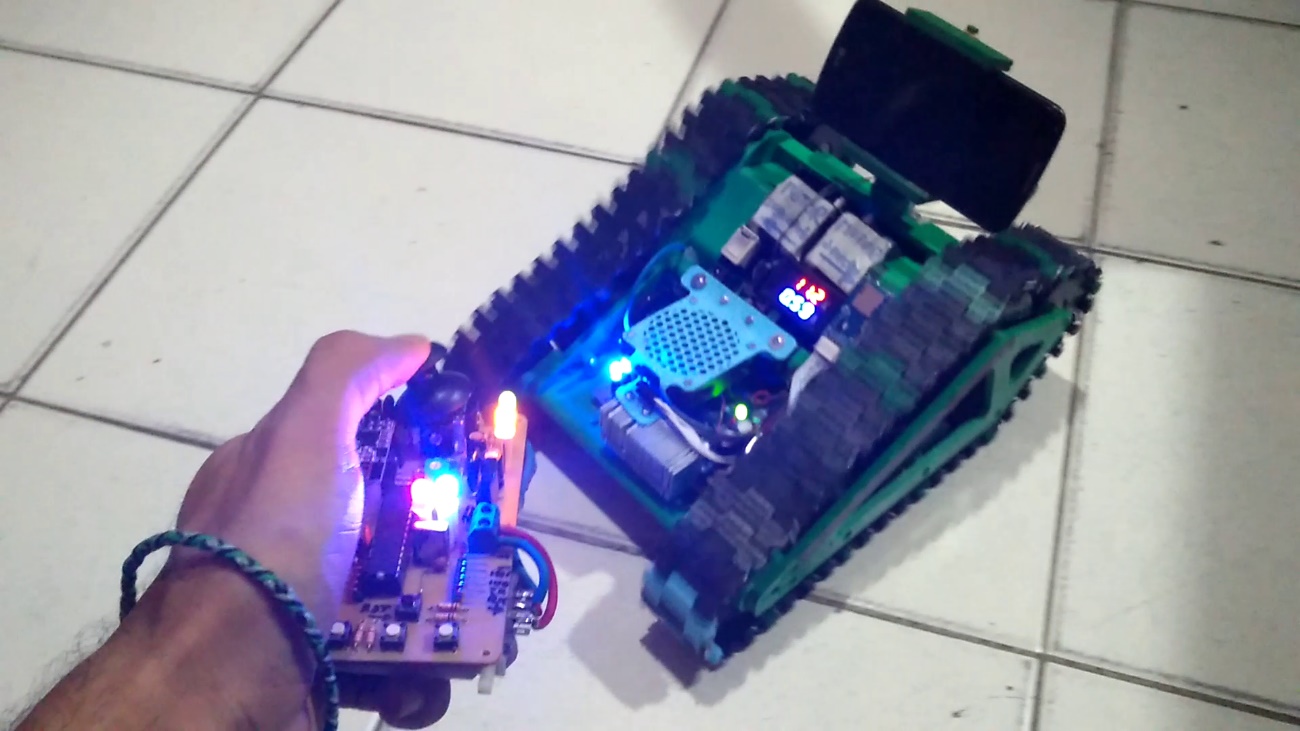


Figura n°1.

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.

## Implementaciones Futuras

Se desean un par de mejoras e integraciones para el proyecto para aumentar la capacidad de realizar tareas propuestas, ya que en el estado actual solamente puede ser controlado mediante radiofrecuencia con la necesidad de un usuario que le maneje. Las cuales se describen más adelante.

* Implementación de una unidad más potente en cuestiones de procesamiento, una Raspberry Pi 3B+ comunicada con la placa de controladora del robot.
* Integración por medio de una APP en el dispositivo Android en frente del robot para darle las prestaciones suficientes de accionar autónomo y/o remoto.
* Desde la APP se comunicará mediante un BOT en Telegram para notificar al o los usuarios correspondientes frente a casos y situaciones determinadas.
* Agregado de sensores de diversa índole, sensor lumínico, sensor de temperatura, sensor de humedad, detector de movimiento y sensor de gas.
* Preservación de datos e informes del robot en servidores locales.

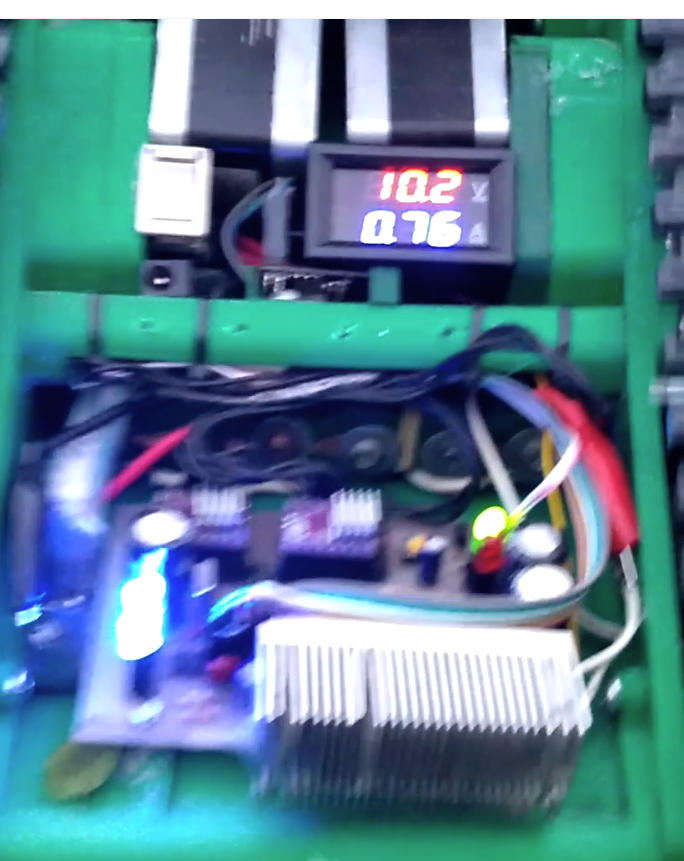


Figura n°2.

# 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>

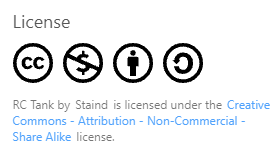


Figura n°3.

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 3,7 v dando en total una diferencia de potencial total de 11,1 v.

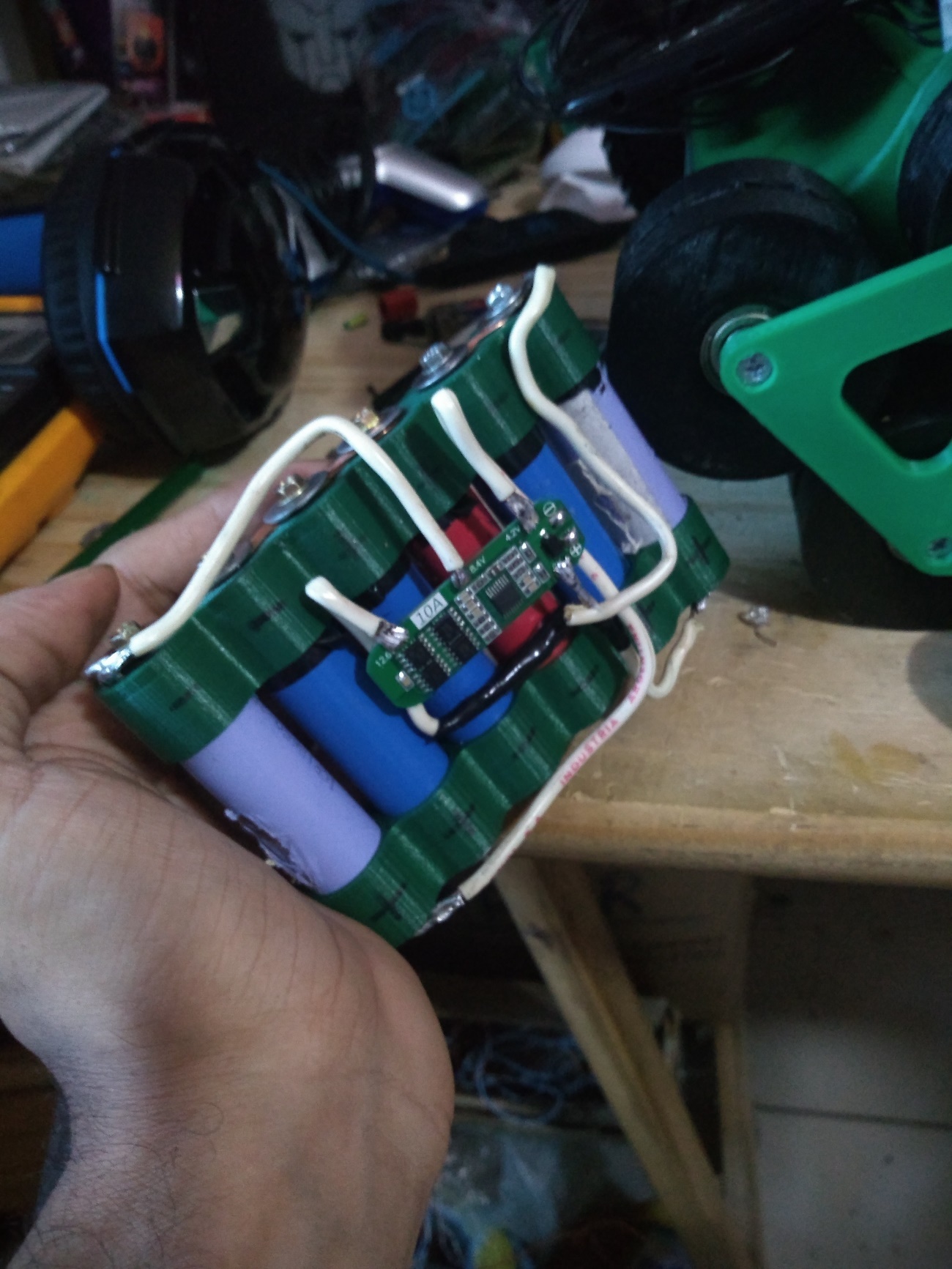


Figura n°5.

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 dos placas de desarrollo electrónicas, las cuales son Raspberry Pi modelo 3 B+ y otra 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. 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.

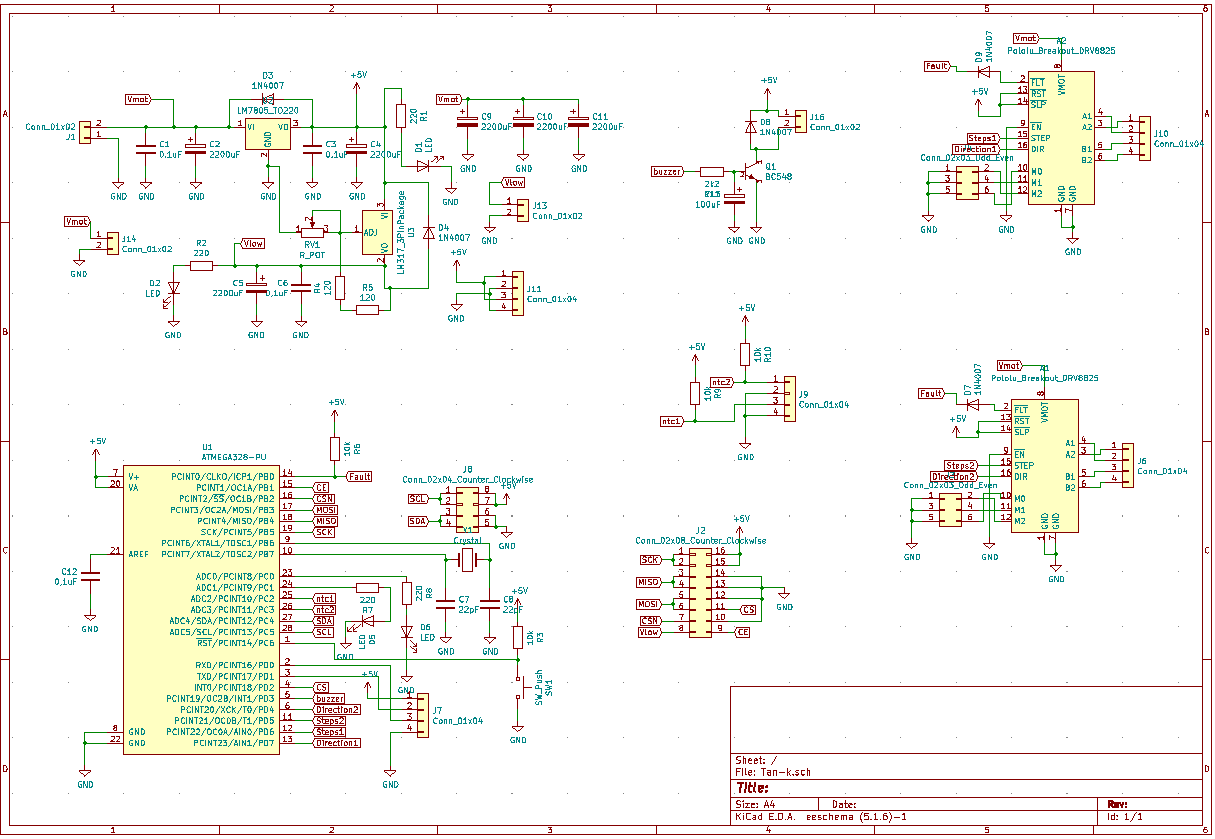


Figura n°6.

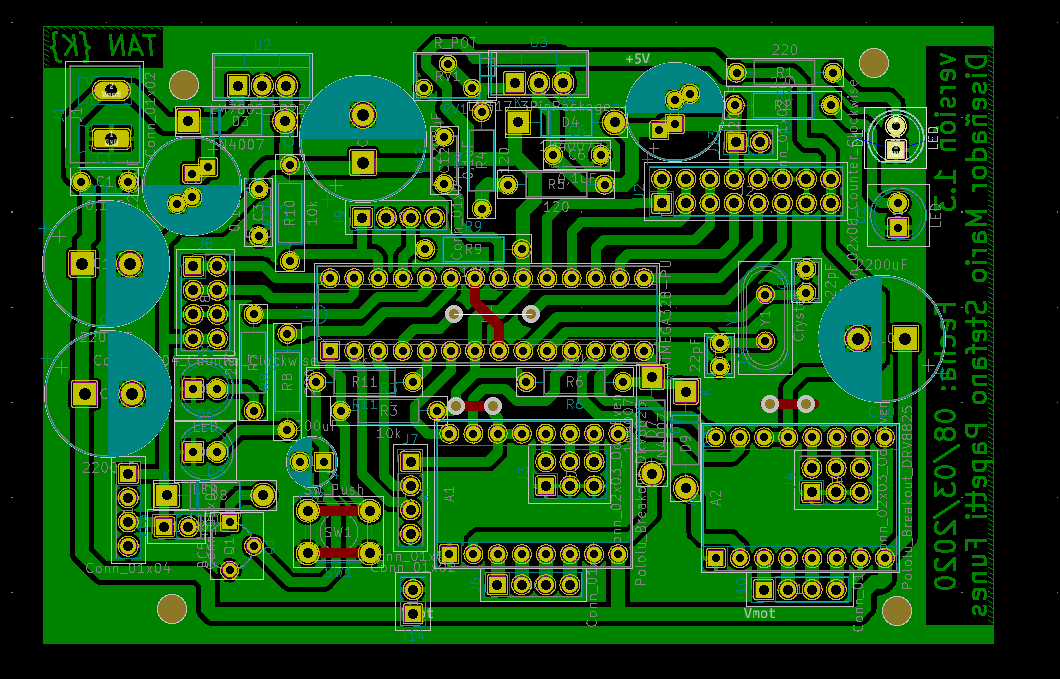


Figura n°1.

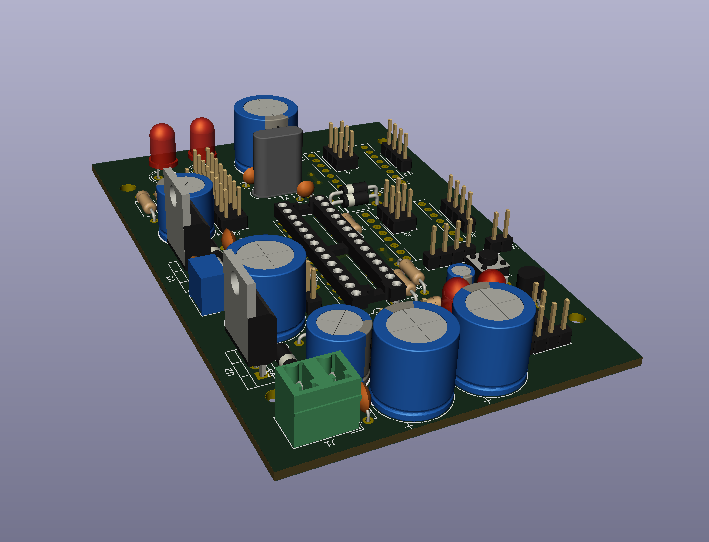


Figura n°1.

## Raspberry Pi 3 B+

La parte de vital importancia en el lado del robot móvil es la que se detallará a continuación, la implementación de una placa de desarrollo con tal potencia, con una basta documentación y una gran comunidad no podría haber sido una mejor opción haber optado por la implementación de la famosa Raspberry Pi 3B+ para este proyecto.

# PROGRAMACIÓN

El enfoque que se le da en el actual estado del proyecto será bajo el concepto de servidor cliente, donde el servidor estará funcionando bajo el hardware que podemos considerar cómo el tanque y el cliente será del lado del usuario solicitante para poder interactuar con el tanque mediante internet dentro de una red local.

El funcionamiento se puede explicar con el siguiente diagrama de clases haciendo distintas distinciones en el funcionamiento del mismo y su interacción entre las clases que lo conforman.

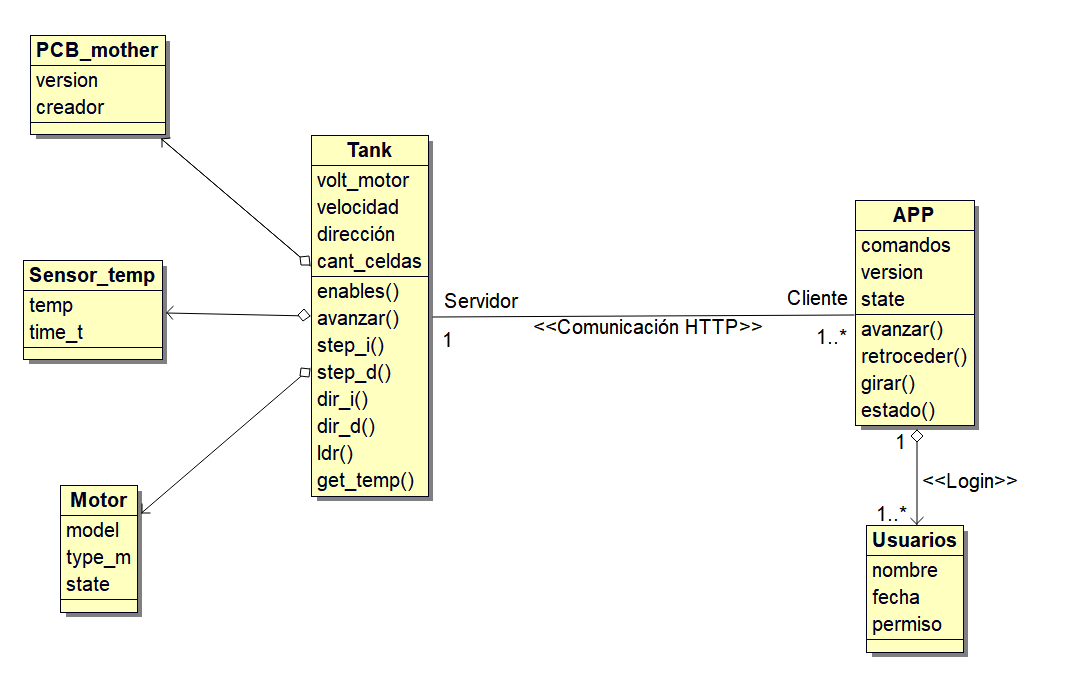


Figura n°x.

Se pueden definir 3 programas dentro del esquema de clases, que interactúan entre sí, del lado del servidor habrán 2 de los 3 programas ya que uno se encargará de gestionar todos los movimientos del robot móvil y el otro quien le pasará las órdenes y se comunicará por internet en la red local para poder recibir las solicitudes del tercer programa del lado del cliente.

Desde un punto de vista de clases vemos cómo existe una única clase que llamaremos “Tank” que define al robot móvil, en el sus respectivos atributos y métodos que llevará acabo, las otras 3 clases a la izquierda con clases que se encuentran vinculadas bajo relaciones de agregación para con la clase “Tank” dichas clases poseen atributos. Por otro lado, la clase del cliente la consideramos cómo una “APP” la cual la idea principal es que corra en dispositivos móviles cómo un celular o smartphone, este manejará cierta información bajo la relación de agregación de una clase la cual denominaremos cómo “Usuarios” estos pueden ser muchos o uno, de igual manera que la “APP” respecto al “Tank” puede haber varias “APP” pero solo se comunicaran con un solo “Tank”.

Volviendo a la idea troncal de poseer 3 programas a continuación se dividen en 3 partes donde se describe y explica el contenido documentado de cada una de los programas que integran al sistema en cuestión.

## 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.

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

## Segunda Parte (Placa Raspberry PI 3B+)

La segunda parte radica en la integración de las solicitudes que llegan del lado del cliente y las que debe de enviar al controlador para poder cumplir con las solicitudes. El mismo se ha escrito bajo el lenguaje de Python en el IDE Visual Studio Code en su versión 1.60.2 y cuyos plugins serán agregados en el apartado de Anexos.

El programa se alojará cómo el subtitulo lo indica en la placa Raspberry Pi 3 B+.

## 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

# Anexos

## Control RF

### Resumen

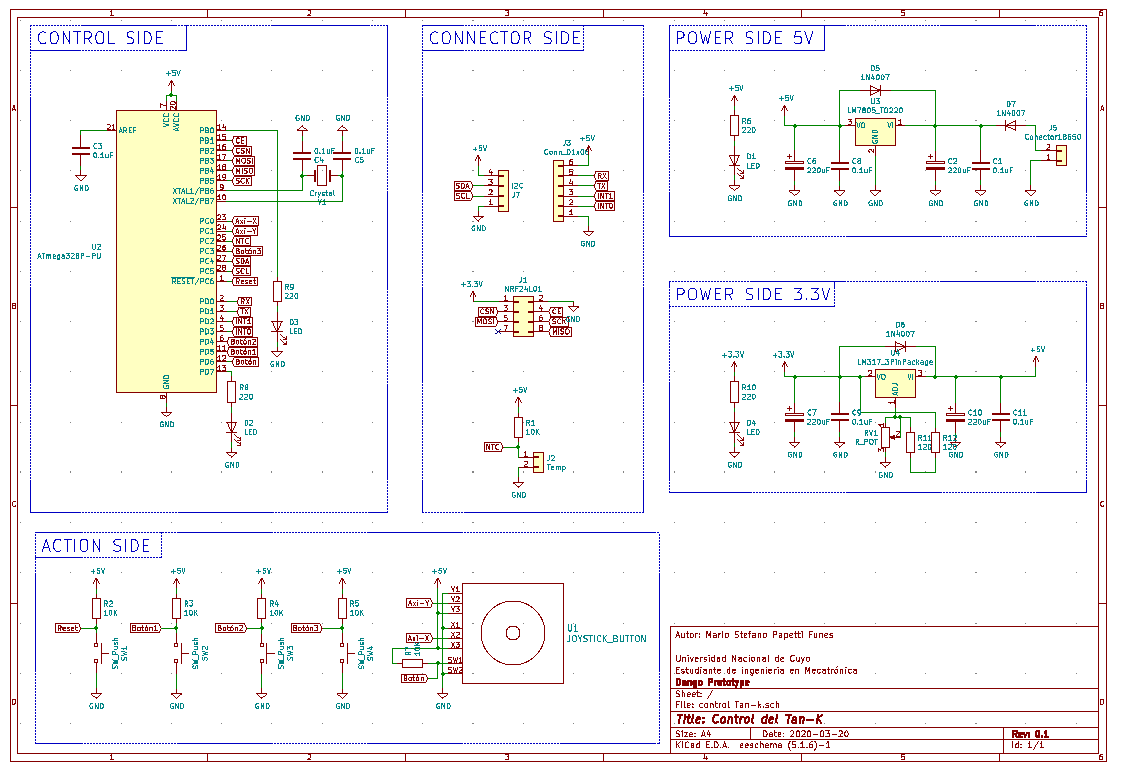
La idea principal era poder verificar la viabilidad mecánica del robot de una forma controlada por el usuario para luego poder llevarlo desde otra perspectiva automatizada, se podría decir que este fue el primer paso dónde se pudieron ver las falencias mecánicas y corrección de las mismas en conjunto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

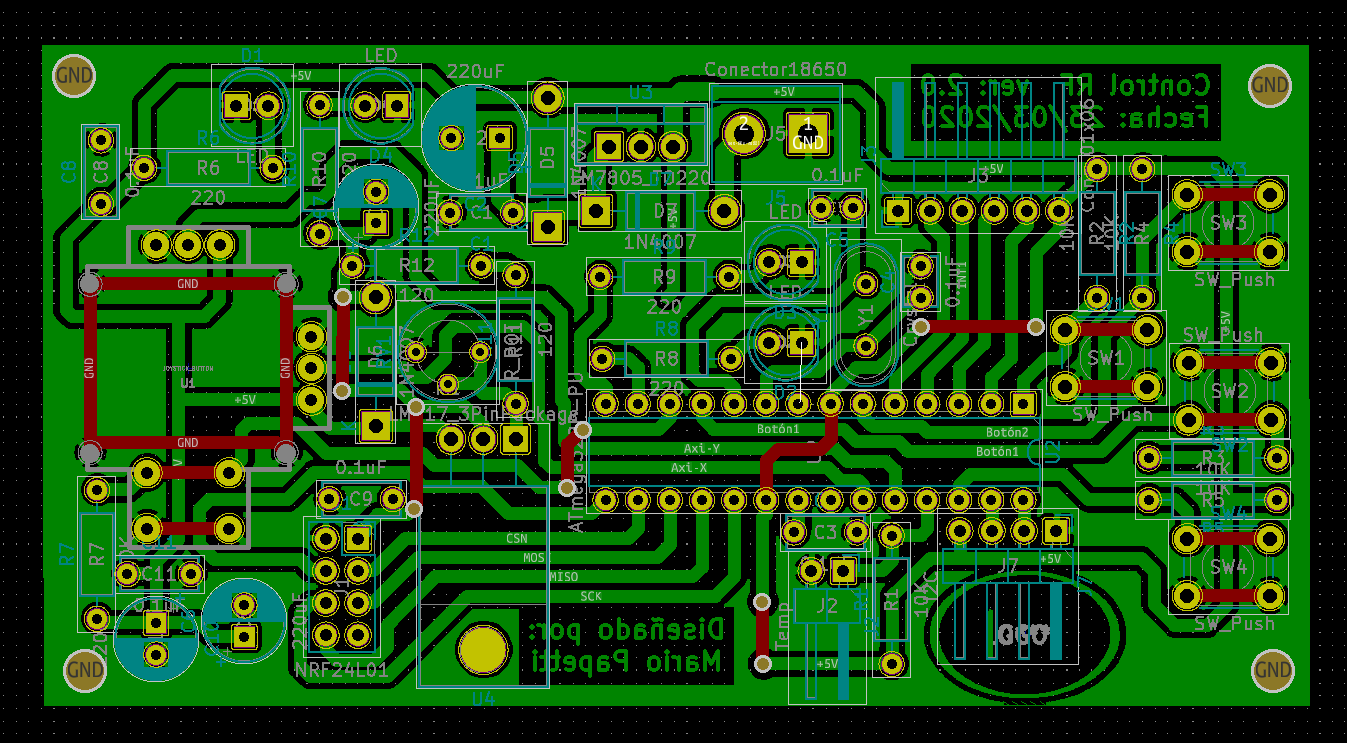
### PCB

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:

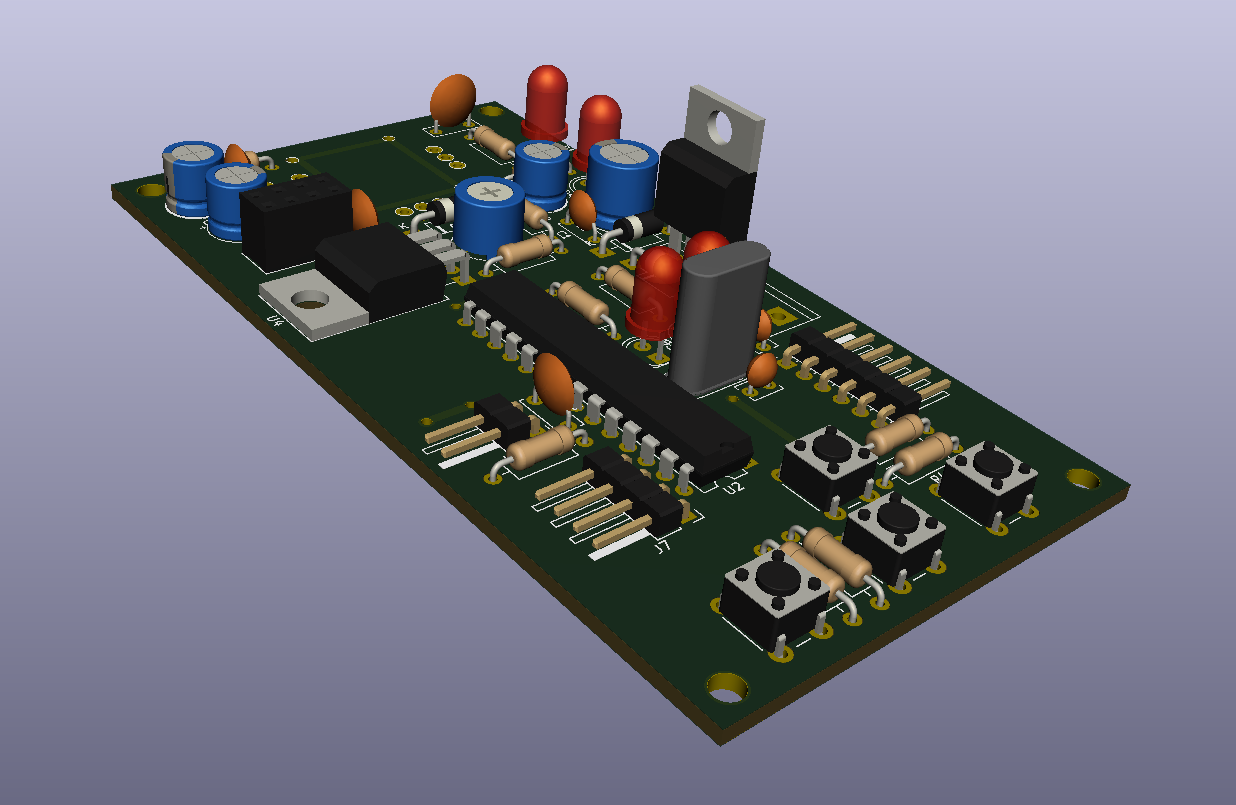
* Esquema electrónico
* Diseño del PCB
* Representación 3D



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;

}

### Segundo Programa (Raspberry PI 3 B+)

#

#

### Tercer Programa (Cliente)

#

#