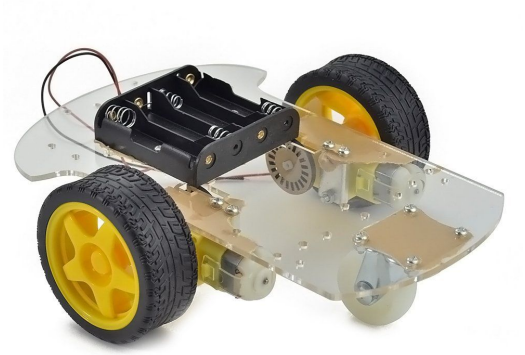
Universidad Simón Bolívar
Departamento de Electrónica y Circuitos
Laboratorio de Proyectos III
Sep - Dic 2014

# Combate de Tanques



Grupo #3:
Danilo Díaz. Carnet #10-10211
Pedro Contreras. Carnet #06-39387

Grupo #8:
Annybell Villarroel. Carnet #10-10768
Jorge Pérez. Carnet #10-10544

Caracas, 27 de noviembre de 2014

# Introducción

La batalla de Kursk (Operación Ciudadela) fue uno de los combates más grandes de la historia, ocurrido entre Julio y Agosto de 1943 durante la Segunda Guerra Mundial . En ella hubo un gran despliegue de tanques de combate (alrededor de 13.000) por parte de Rusia y Alemania. El proyecto es simular este enfrentamiento utilizando dos robots tipo tanque autónomos, que se detecten el uno al otro y que activen un led cuando se encuentren.

Para llevar a cabo este simulacro es necesario utilizar diversos tipos de sensores como infrarrojos, ultrasonido y una cámara USB. Además, se controla el movimiento de los motores utilizando un puente H, en conjunto con dos encoders y se establece comunicación entre la interfaz gráfica y el tanque utilizando un emisor/receptor infrarrojo y un arduino.

A continuación se muestra cómo se realizó la adquisición y emisión de datos y la interfaz gráfica, en conjunto con la implementación de cada uno de los sensores utilizados.

# Objetivos

## Objetivo General

- Desarrollar un robot tipo tanque capaz de seguir emisores infrarrojos, determinar distancias y evitar obstáculos de manera autónoma, utilizando sensores infrarrojos, un ultrasonido, una cámara y la tarjeta de desarrollo DemoQE128

## Objetivos Específicos

- Controlar el movimiento de los motores del tanque utilizando un puente H y encoders.
- Diseñar e implementar el sistema de adquisición para los receptores infrarrojos y el de emisión infrarroja.
- Desarrollar un sistema de comunicación por control remoto utilizando un Arduino
- Obtener información sobre la distancia del tanque enemigo y evitar obstáculos a partir de un sensor de ultrasonido.
- Implementar un sistema de detección de posición utilizando una cámara
- Desarrollar una interfaz gráfica que permita determinar cuándo gana alguno de los tanques

# Interfaz Gráfica

En un principio se planteó desarrollar la interfaz en el lenguaje de programación Python en conjunto con Processing, conectándolos a través de sockets. Esto se implementó efectivamente durante las primeras semanas (entre la 2 y la 4) del trimestre pero, dado ciertos inconvenientes obtenidos en la computadora a usar en la presentación final, se decidió realizar la interfaz completamente processing. En la sección de Anexos (Extracto #1) se puede observar el código implementado en Python como añadido; cabe destacar que en ese lenguaje se implementó un servidor que recibe las coordenadas X y Y del punto detectado con la rutina de la cámara en Processing, lo que se pretendía era realizar el menú y la vista del usuario con este programa.

La interfaz realizada es básicamente una máquina de estados que permite presentar la información al usuario de una manera estructurada. En tal sentido, se tienen en total 6 estados; “WelcomeScreen” el cual es el menú principal en donde se encuentran las instrucciones, información sobre el equipo e inicio, el estado de más información “LearnMore”, el de historia “History”, donde se presenta un breve resumen sobre la Batalla de Kursk, el estado “Battlefield” donde se configura el campo de batalla de los robots para que pueda ser visualizado en pantalla, el estado “Play” donde se presenta la información principal y el estado final “AfterAGame”. El usuario puede cambiar de estado haciendo click con el ratón o presionando alguna tecla, dependiendo del caso.  Se pueden observar el diagrama de estados, de bloques y una captura de pantalla de la interfaz a continuación.


Figura #1: Menú principal de la Interfaz Gráfica

El diagrama de estados de la Interfaz implementada es el siguiente:

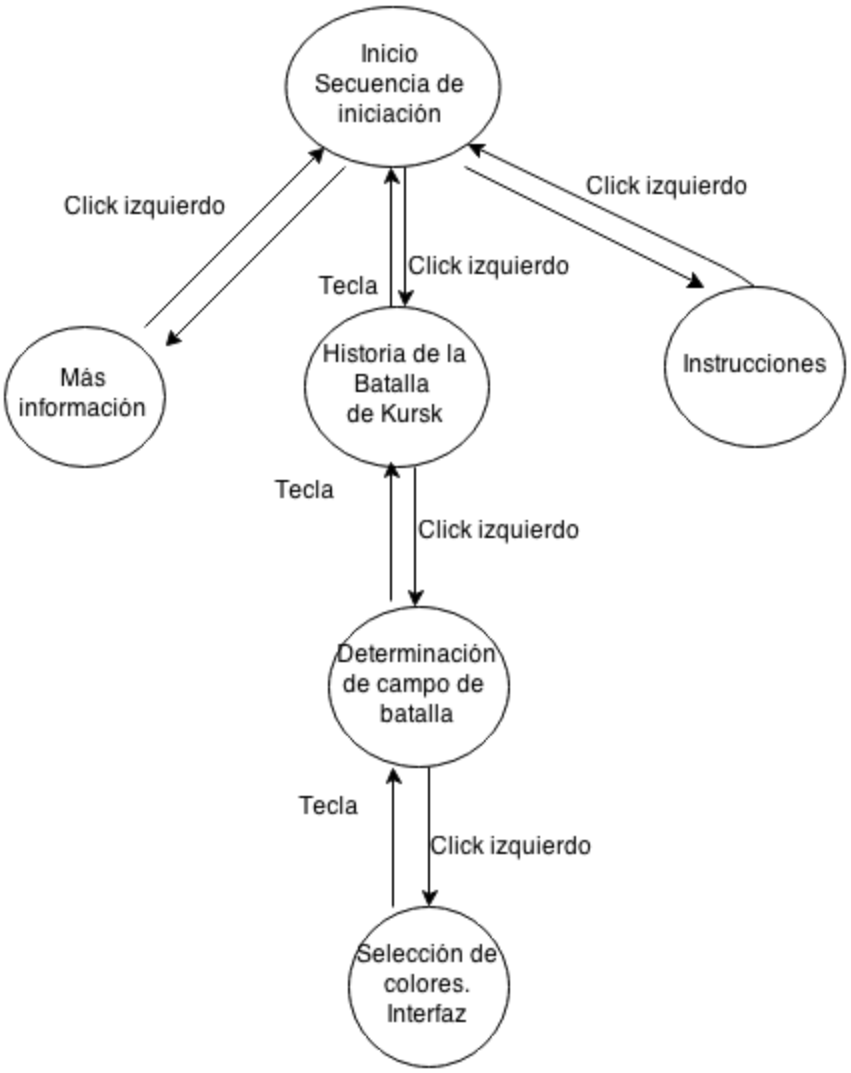



Diagrama #1: Estados del programa implementado en Processing

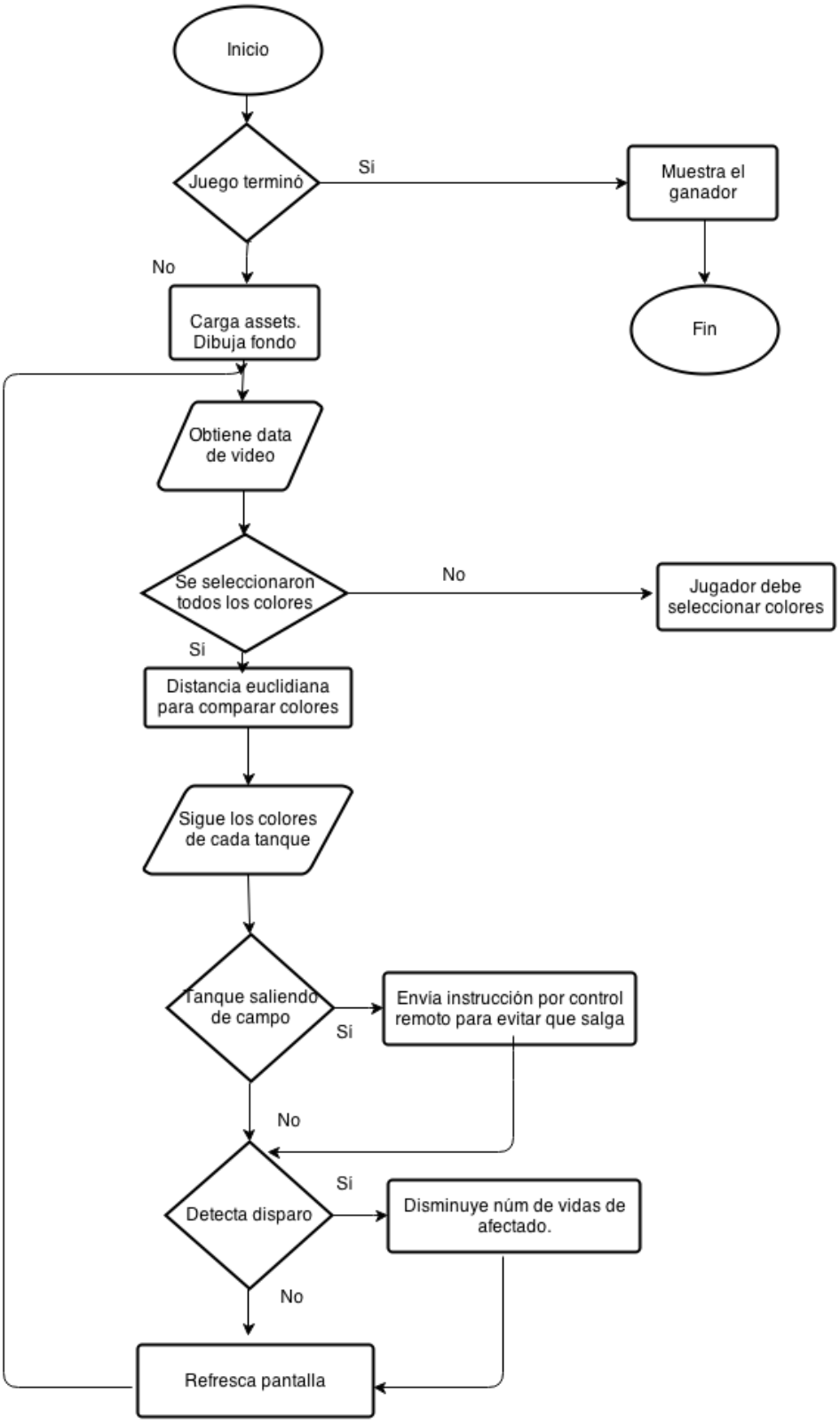



Diagrama #2: Lógica del estado principal del juego

En el estado inicial de la interfaz gráfica se dibuja el menú, en el estado “Battlefield” se dibuja la cámara, se escriben las instrucciones y se determinan los vértices del campo luego de que el usuario hace click sobre ellos, y en el estado principal, esto es el juego en sí, igualmente se vuelve a dibujar la cámara, sobre ella se dibuja el campo de batalla utilizando la instrucción “line” con los vértices obtenidos y se realiza todo el algoritmo de procesamiento de colores luego de que el usuario ha hecho click sobre cada uno de los deseados. En la sección de anexos se pueden ver algunas capturas de pantalla de las secciones mencionadas.

Cada uno de los estados tiene una función handle que maneja lo que debe suceder, la cual se establece en el draw principal (véase Extracto #2 en la sección de Anexos), tiene además una función específica para cuando se presiona algún botón del mouse a partir de la función “moussePressed” y un switch para determinar el estado en el que se encuentra y tiene una función para cuando presiona alguna tecla utilizando “keyPressed”.

La interfaz gráfica no solo le permite al usuario observar lo que sucede en el campo de batalla sino que también, a través del procesamiento de colores, permite controlar aspectos importantes para el robot como determinar si este se está saliendo o no del campo de batalla (razón por la cual se debe configurar este aspecto en un estado previo) y también determinar si se encendió el led de disparo y si el otro tanque está a una distancia minima para diminuir el número de intentos del tanque correspondiente. En la sección de anexos (Extracto #4) se encuentra la rutina utilizada con el ratón del mouse para determinar cada uno de los colores a los cuales se les hará tracking.

Para determinar los colores se revisan todos los píxeles obtenidos de la cámara de video y se mide la distancia euclidiana entre el pixel actual y el deseado (determinado previamente al hacer click con el Mouse), luego si esta distancia está dentro de un rango finito, se guarda la posición actual del pixel y se hace el promedio entre ellas para hallar el centro del color deseado. Se pueden observar los aspectos más relevantes de la rutina implementada en la sección de Anexos, Extracto #5 y Extracto #6. No se incluye la situación para los 4 colores ya que es igual, solo cambian las variables utilizadas.

Como se mencionó anteriormente, la interfaz gráfica le comunica al robot que está a punto de salirse del campo; esto se lográ al enviar comandos predeterminados por el puerto serial, los cuales son recibidos por el arduino, el cual envía por control remoto infrarrojo el comando correspondiente que será interpretado en el MCU. Se puede observar parte de la rutina relacionada con este aspecto en la sección de Anexos Extracto #7

A continuación se coloca una tabla con la serie de comandos enviados por control remoto y las instrucciones correspondientes para cada uno de los tanques.

| Comandos | Descripción |
|---|---|
| **0xFD** | Inicio del juego |
| **0x00** | Dentro del campo Tanque 1 |
| **0x01** | Fuera Pista Derecha Tanque 1 |
| **0x02** | Fuera Pista Izquierda Tanque 1 |
| **0x03** | Fuera Pista Abajo Tanque 1 |
| **0x04** | Fuera Pista Arriba Tanque 1 |
| **0x05** | Fuera Pista Derecha Tanque 2 |
| **0x06** | Fuera Pista Izquierda Tanque 2 |
| **0x07** | Fuera Pista Abajo Tanque 2 |
| **0x08** | Fuera Pista Arriba Tanque 2 |
| **0x09** | Dentro del campo Tanque 2 |

Tabla #1: Comandos para notificación a cada uno de los tanques

## Sistema de detección y emisión de IR. Longitud de onda y modulación.

El sistema de detección infrarrojo se encuentra en la parte delantera del vehículo y consiste en 2 receptores infrarrojos los cuales según la intensidad de la señal que reciba cada uno, deberá centrar el robot hacia el objetivo.

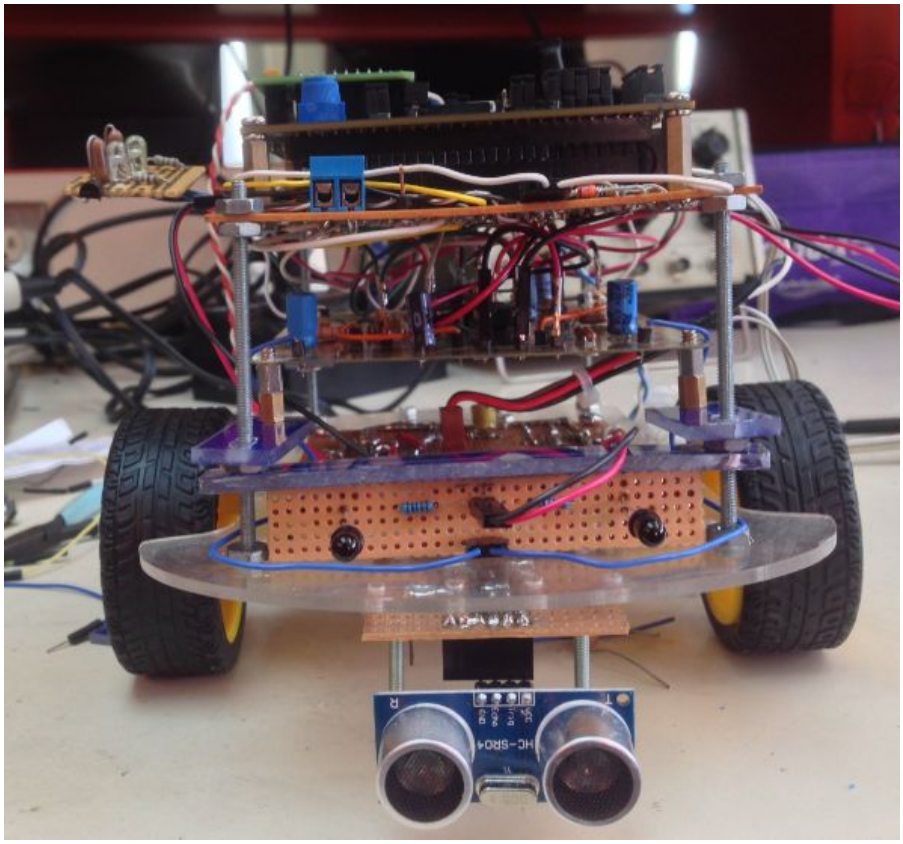



Figura #2: Robot de equipo 3

Mediante el uso de un transistor 2N2222, se procedió a realizar la interfaz entre el MCU y los LEDs que emiten la señal de 1 KHz (equipo 3) o 5 KHz (equipo 8) según el caso. El cambio de frecuencia viene dado por el MCU. A continuación, se puede observar el esquemático del circuito implementado.

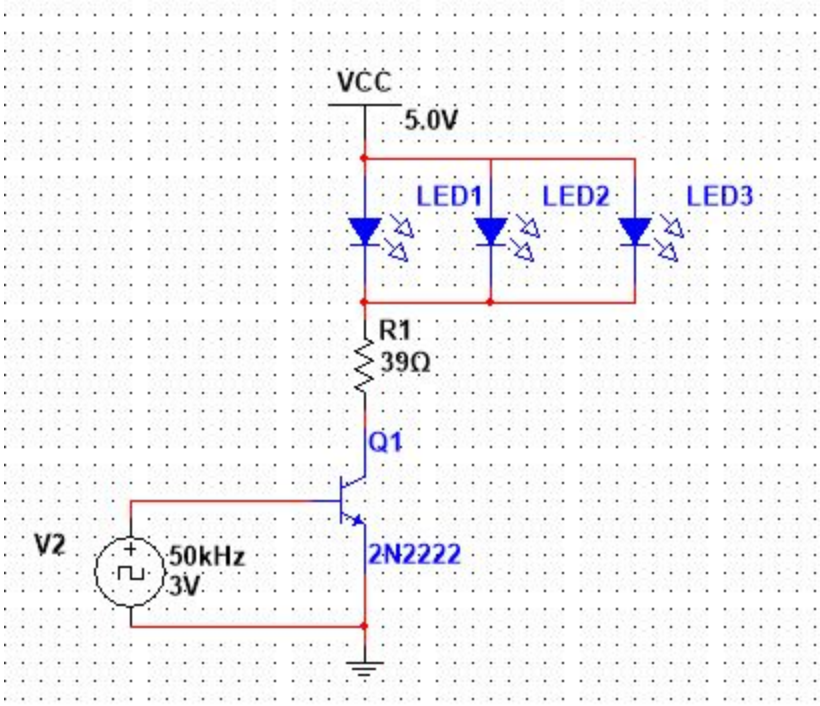
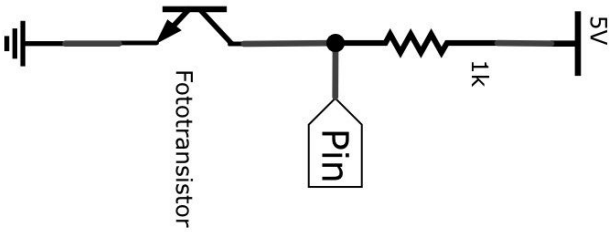



Figura #3: Circuito emisor infrarrojo

A continuación cada receptor está configurado de la siguiente manera




Dado que la amplitud de recepción debe estar normada, se utilizó una etapa de amplificación para este ajuste.

En primer lugar, luego del buffer se implementó un filtro activo pasabanda para seleccionar la frecuencia correspondiente, en este caso 5KHz para un equipo y 1Khz para el otro. Posteriormente se implementó un amplificador y, a la salida de este se realizó una detección de envolvente. Se tomó la decisión utilizar la configuración de “super diodo” o diodo de precisión, para evitar que la señal amplificada se vea distorsionada por el voltaje de caída del diodo.

A continuación, se presentan las figuras correspondientes al diseño de los filtros para la adquisición IR

## ➔ Filtro 5K

A continuación se muestra el esquema del sistema de adquisición utilizado.

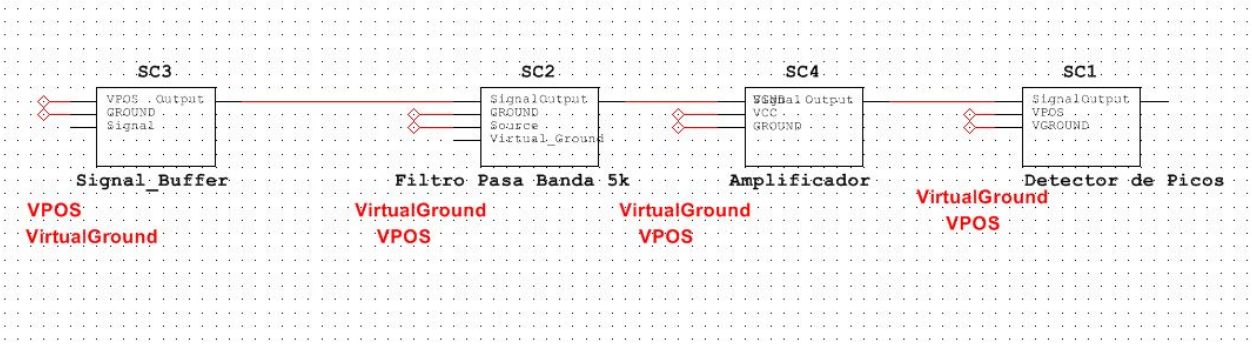



Figura #2: Diagrama de bloques básico del sistema de adquisición

Etapa 1: Buffer

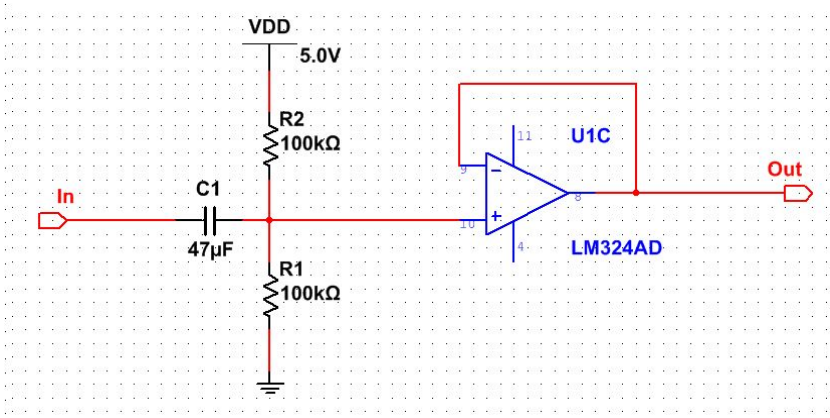



Figura #5: Buffer implementado para la señal recibida del detector infrarrojo

Etapa 2 :Filtraje de la señal

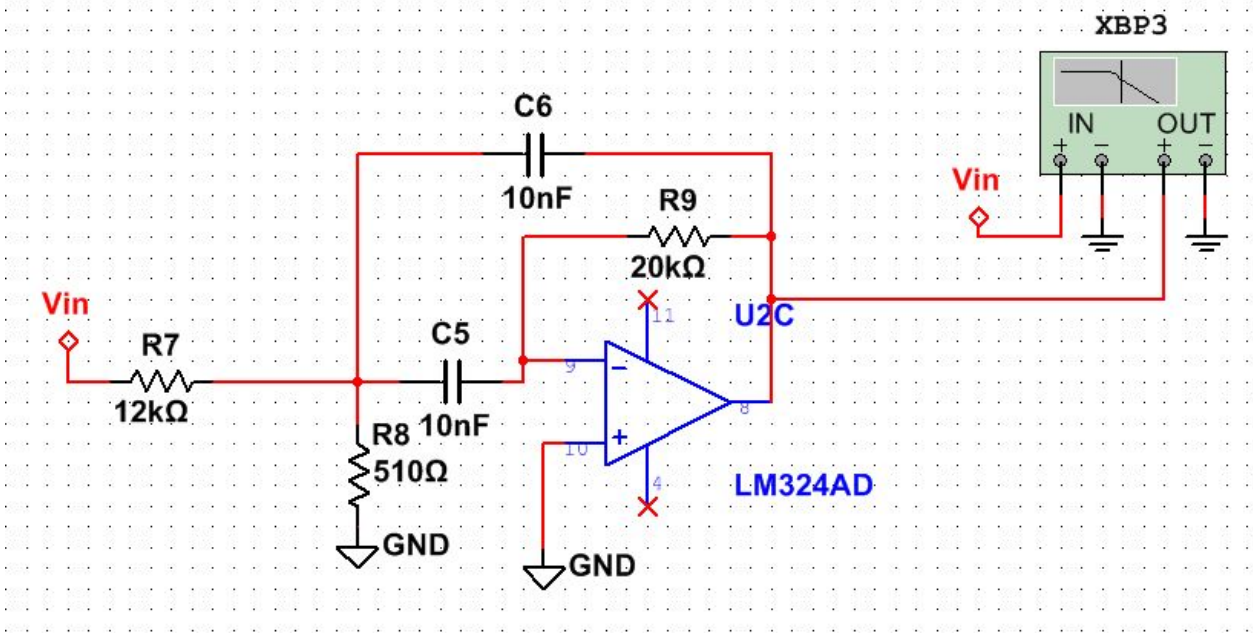



Figura #6 Filtro pasabandas para señal a 5 Khz

Etapa 3: Amplificación

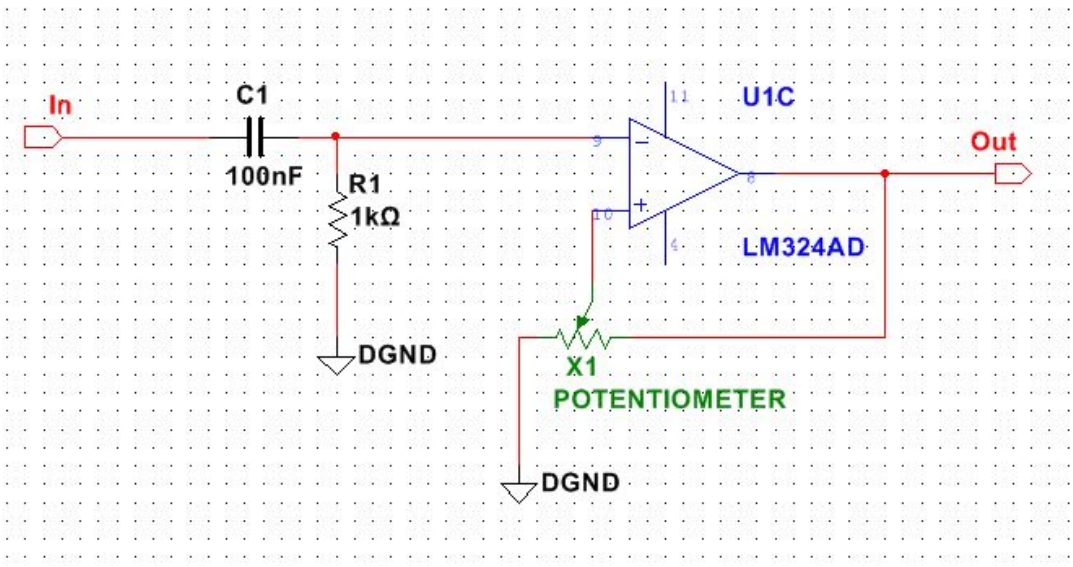



Figura #7 Amplificador para la señal filtrada recibida

Etapa 4: Rectificación + Detector de picos

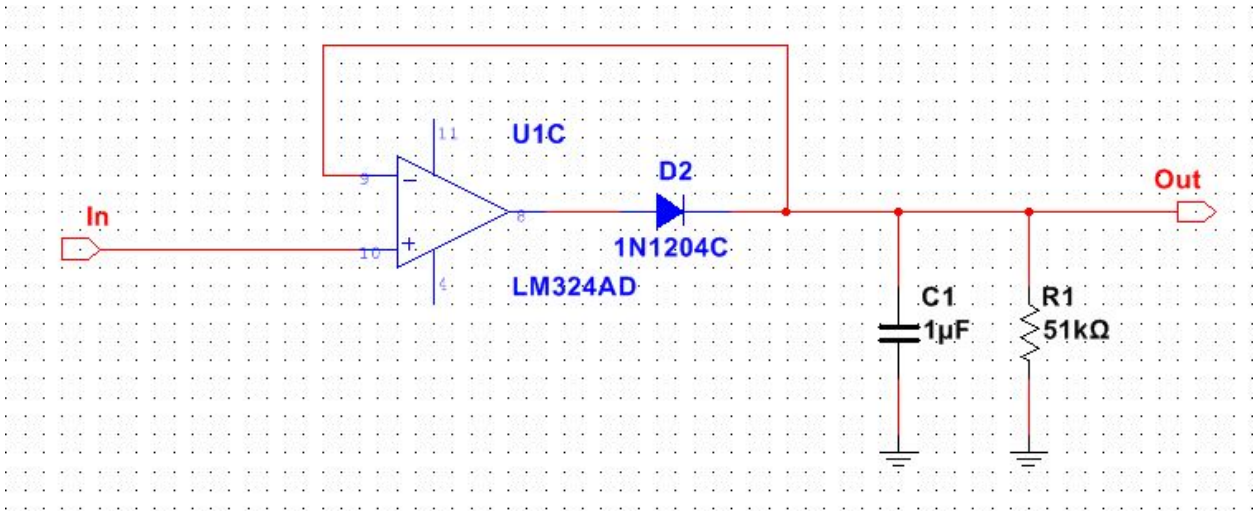



Figura #8 Detector de picos implementado

Todo esto fue probado, implementado mediante PCB (Printed Circuit Board) dada la cantidad de componentes a soldar.

Dando una tarjeta como la que se muestra a continuación

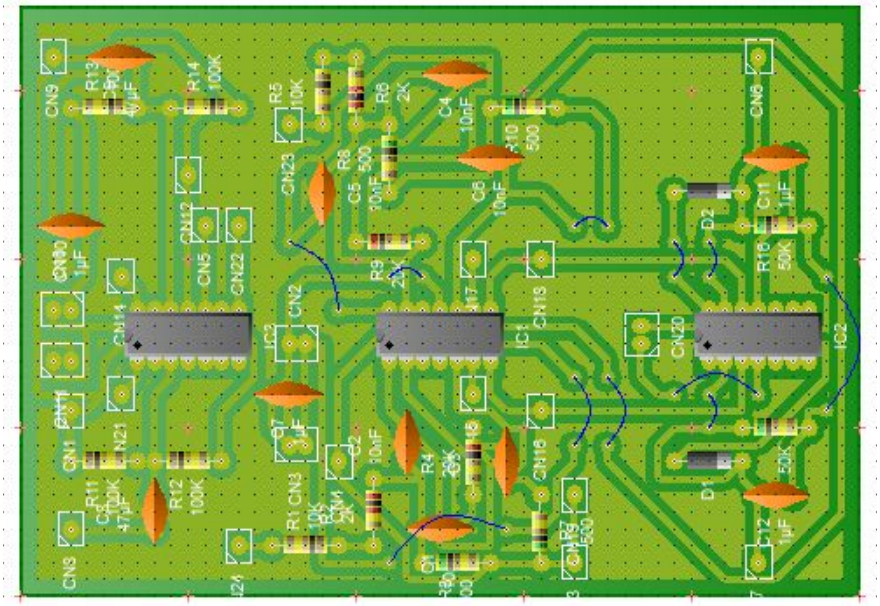


Figura #9 PCB Adquisición

## ➢ Filtro 1K

A continuación se muestra el esquema del sistema de adquisición utilizado.

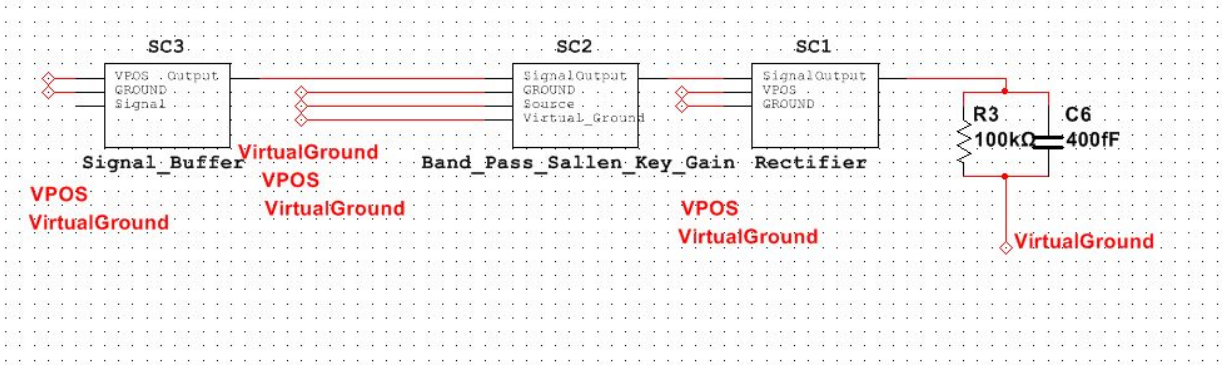



Figura #10: Diagrama de bloques básico del sistema de adquisición 1K

Etapa 1: Buffer

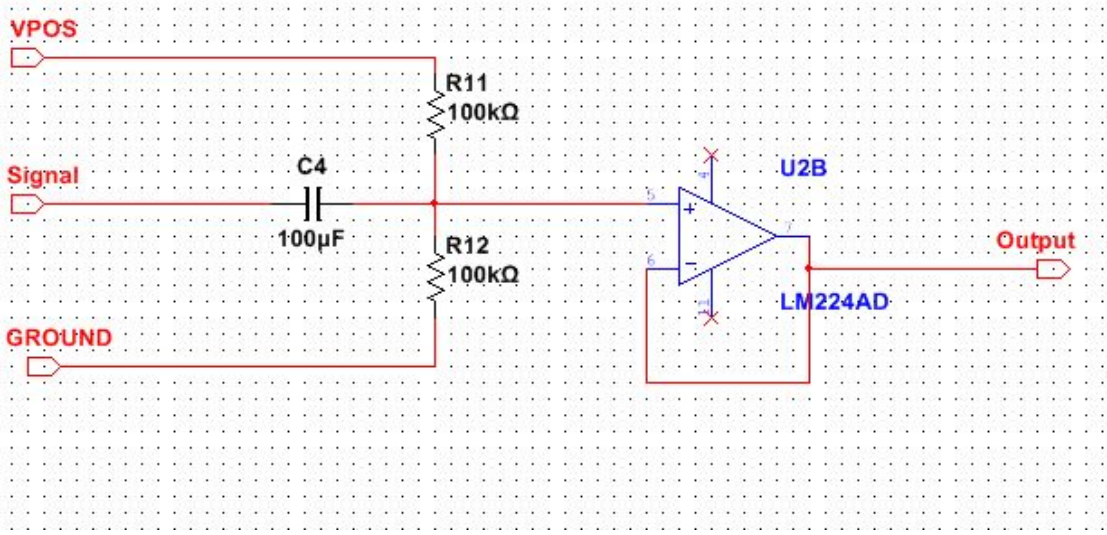



Figura #11: Buffer implementado para la señal recibida del detector infrarrojo

Etapa 2 :Filtraje y amplificación de la señal

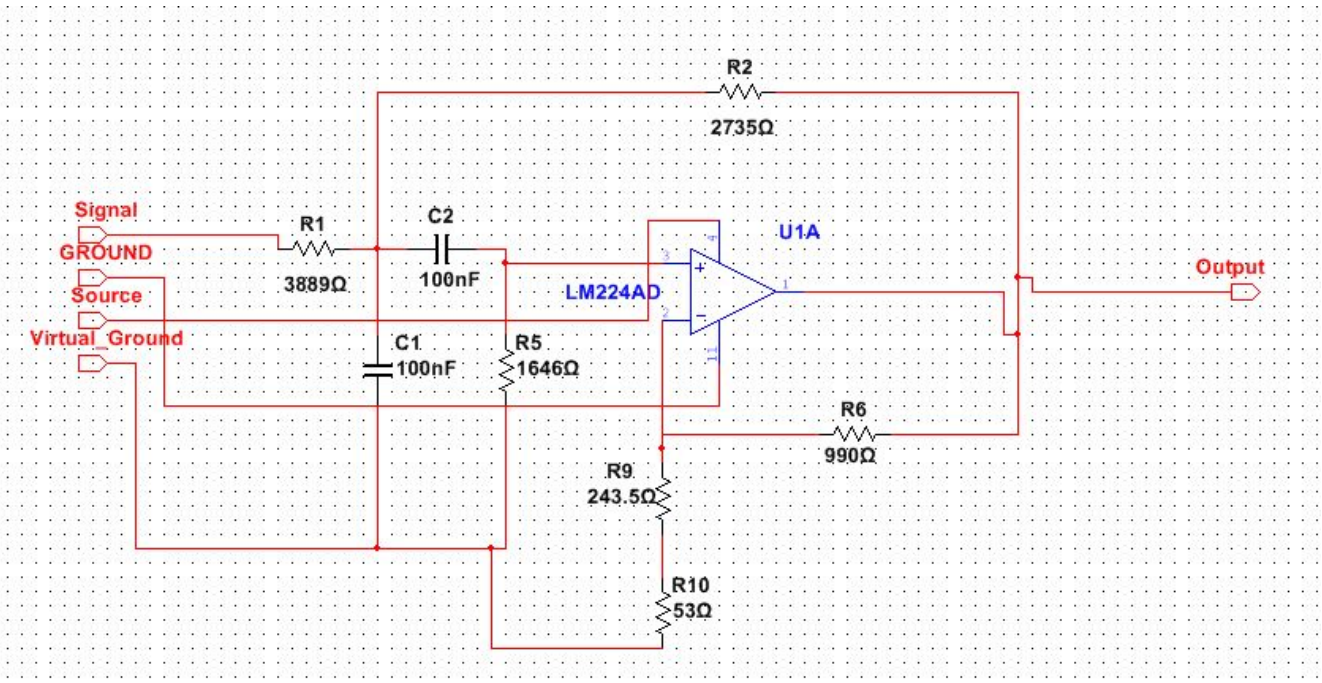



Figura #12 Filtro pasa-bandas con ganancia para señal a 1 Khz

Etapa 3:Rectificación de la señal filtrada

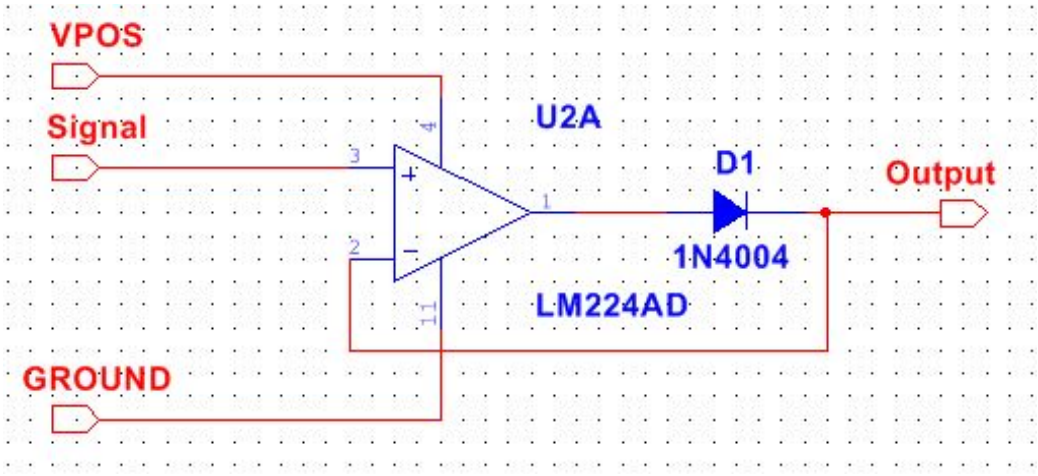



Figura #13 Rectificador

Etapa 4:Detector de Picos

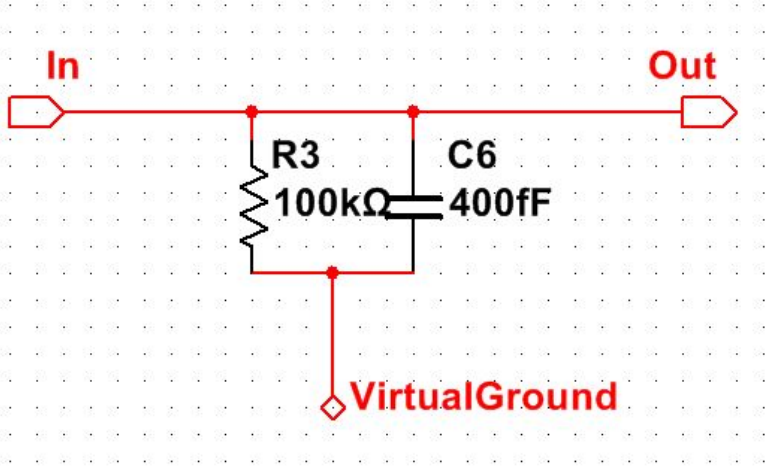
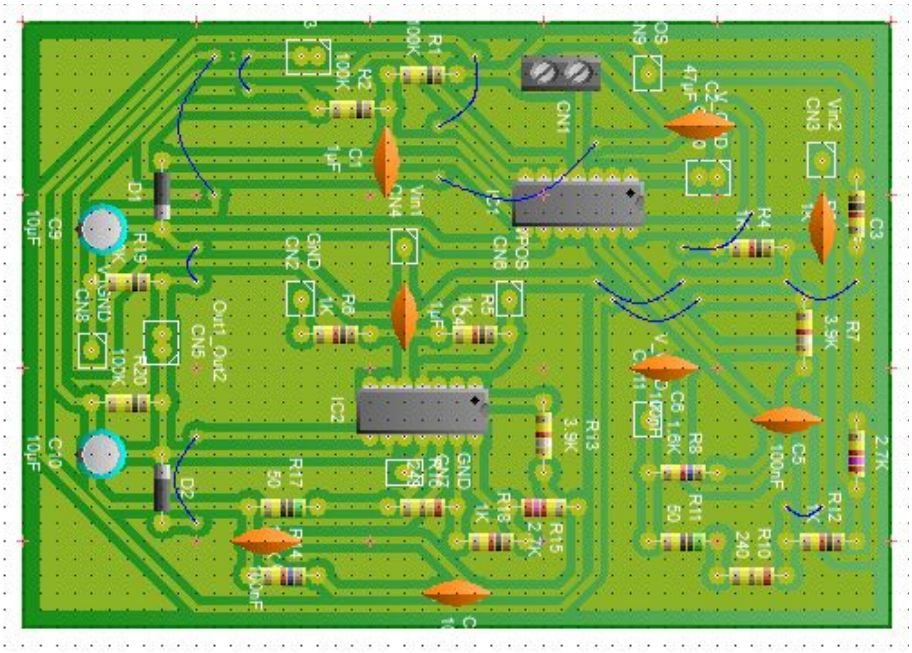



Figura #14 Detector de picos implementado señal 1Khz

Este filtro también fue implementado mediante PCB (Printed Circuit Board) dada la cantidad de componentes a soldar.

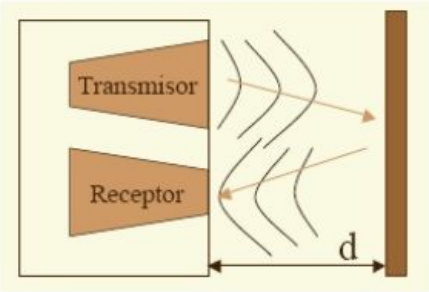
Dando una tarjeta como la que se muestra a continuación



## Sensor de distancia por ultrasonido.

El sensor que se utilizó fue el HC-SR04, que se colocó en la parte delantera del tanque y a partir de él se indica la distancia a la que se encuentra un obstáculo para poder evitarlo.

El funcionamiento básico de los ultrasonidos como medidores de distancia se muestra en el siguiente esquema, donde se tiene un receptor que emite un pulso de ultrasonido que rebota sobre un determinado objeto y la reflexión de ese pulso es detectada por un receptor de ultrasonidos.




La mayoría de los sensores de ultrasonido de bajo coste se basan en la emisión de un pulso de ultrasonido cuyo lóbulo, o campo de acción, es de forma cónica. Midiendo el tiempo que transcurre entre la emisión del sonido y la percepción del eco se puede establecer la distancia a la que se encuentra el obstáculo que ha producido la reflexión de la onda sonora, mediante la fórmula:

$$d = \frac{1}{2} V \cdot t$$

donde V es la velocidad del sonido en el aire y t es el tiempo transcurrido entre la emisión y recepción del pulso.

Para la medición del pulso generado, se realizó en el MCU una medición de tiempo mediante timmers, utilizando el bean Capture de Processor Expert.

Cabe destacar que este sensor trabaja con un voltaje de 5v, por lo que se debe hacer una conversión de 5v a 3v para el procesamiento del pulso en el MCU. Para ello, se colocó un sencillo divisor de voltaje y se agregó un diodo zener como protección.

## Sistema de manejo de los motores.

Para controlar los motores se utilizó el MCU y un puente H. El puente H tiene dos terminales que según estén polarizados hacen que el motor gire para un lado o para el otro,

esto será determinado por la información que reciba el microcontrolador del sistema de detección IR. Cuando el motor gira hace que también gire el encoder. El encoder manda un pulso cada vez que el haz entre el par fototransistor/fotorreceptor se interrumpe y se vuelve a permitir su paso. Este pulso se utilizó como una entrada de interrupción para el conteo y actuación del mismo.

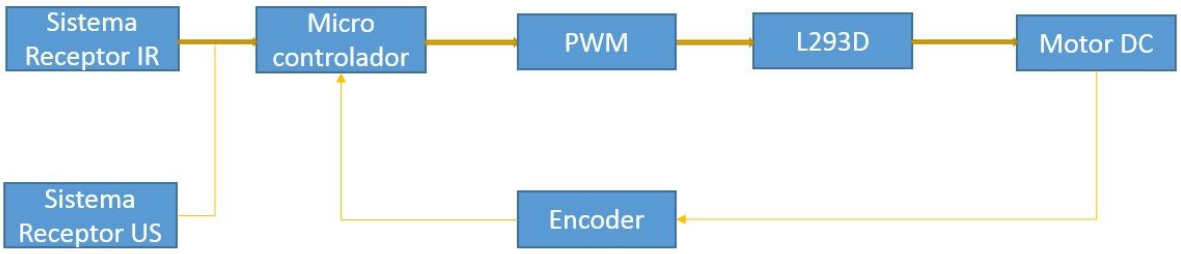



Diagrama #3 Diagrama de bloques del sistema

## Encoder

Los Encoders son sensores que generan señales digitales en respuesta al movimiento. Están disponibles en dos tipos, uno que responde a la rotación, y el otro al movimiento lineal. Cuando son usados en conjunto con dispositivos mecánicos tales como engranajes, ruedas de medición o flechas de motores, estos pueden ser utilizados para medir movimientos lineales, velocidad y posición.

Se utilizaron principalmente para realizar giros precisos con el motor, mediante el conteo de los pasos a medida que se realizaban los movimientos. Esto permitía prescindir de timmers u otros recursos del MCU, además de su precisión a la hora de evitar obstáculos.

## Puente H

En el circuito de abajo vemos un Puente H de transistores, nombre que surge, obviamente, de la posición de los transistores, en una distribución que recuerda la letra H. Esta configuración es una de las más utilizadas en el control de motores de CC, cuando es necesario que se pueda invertir el sentido de giro del motor.

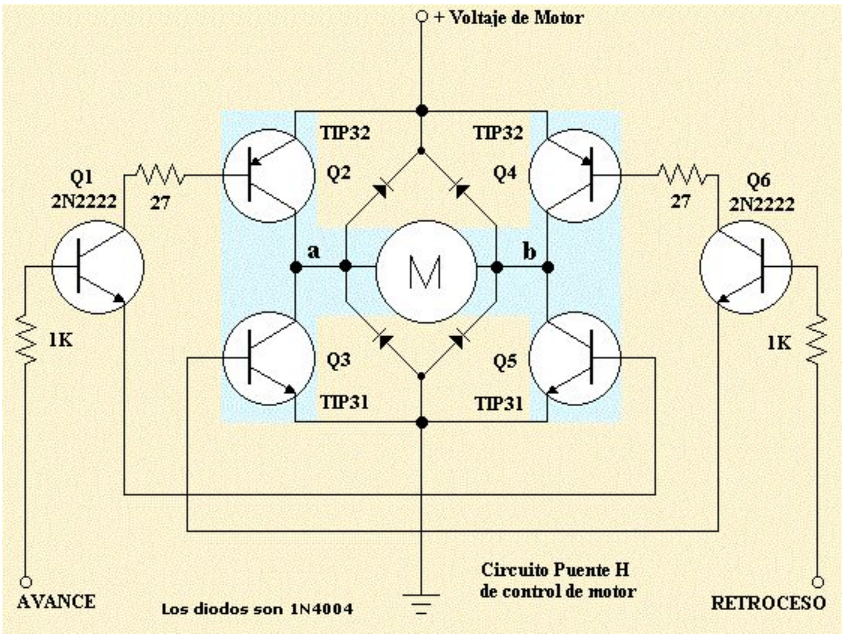



Figura #10 Esquema de puente H

Funcionamiento:

Aplicando una señal positiva en la entrada marcada AVANCE se hace conducir al transistor Q1. La corriente de Q1 circula por las bases, de Q2 y Q5, haciendo que el terminal a del motor reciba un positivo y el terminal b el negativo (tierra).

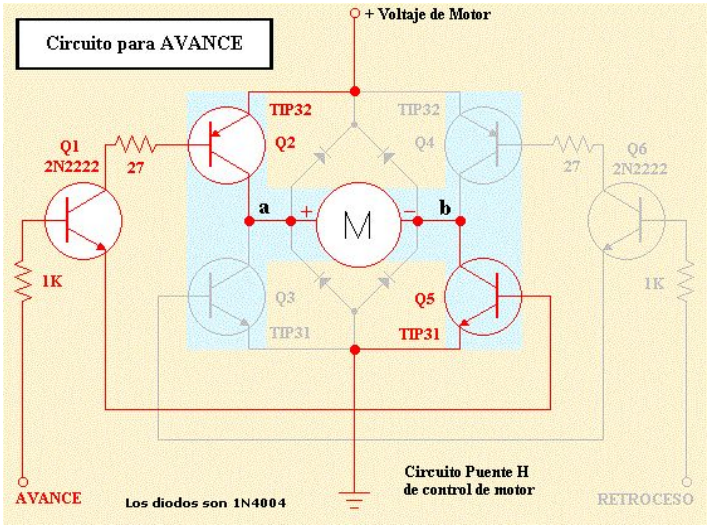



Figura #11 Esquema de puente H

Si en cambio se aplica señal en la entrada RETROCESO, se hace conducir al transistor Q6, que cierra su corriente por las bases, de Q4 y Q3. En este caso se aplica el positivo al terminal b del motor y el negativo (tierra) al terminal a del motor.

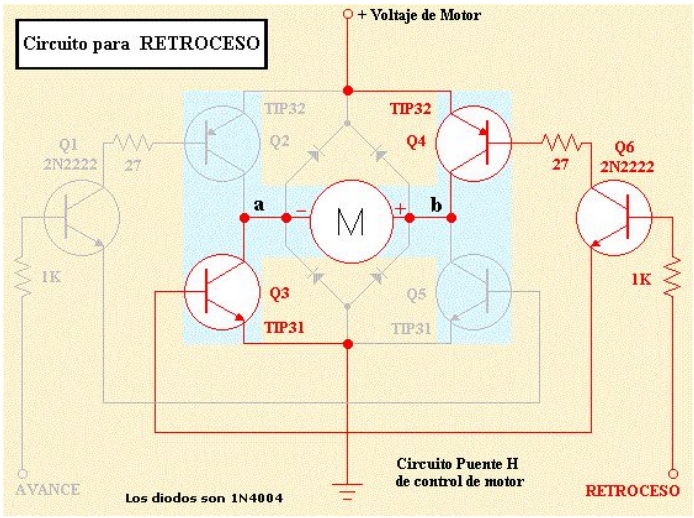



Figura #12 Esquema de puente H

Entonces para la implementación se usó el integrado L293D el cual posee 2 puentes H..

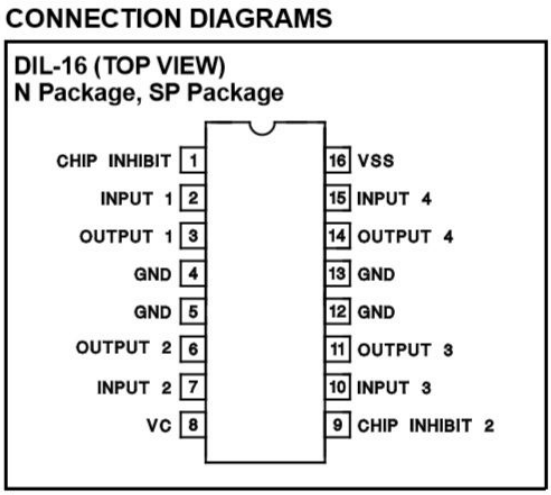
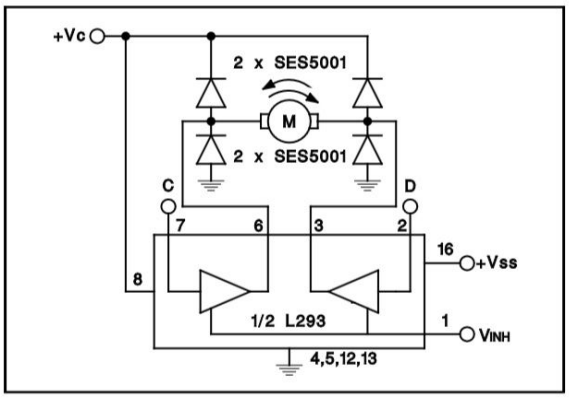



Figura #13 Integrado L293D

Y el cual para tener la configuración deseada se conecta de la siguiente manera al motor DC.




**Bidirectional DC Motor Control**

| INPUTS | | FUNCTION |
|---|---|---|
| VINH = H | C = H; D = L | Turn Right |
| | C = L; D = H | Turn Left |
| | C = D | Fast Motor Stop |
| VINH = L | C = X; D = X | Free Running Motor Stop |

*L = Low  H = High  X = Don't Care*

Figura #14 Conexión del L293D

Esto nos permitió trabajar el motor tanto en directo como inverso cambiando los voltajes de las entradas C y D dependiendo de la dirección e intensidad de la señal que recibían los sensores infrarrojos, así como los datos recibidos del sensor de posición Ultrasonido. De esta forma, el tanque puede girar para buscar al enemigo o pararse cuando lo encontró

## Circuito Modulador por Ancho de Pulso (PWM)

La modulación por ancho de pulsos de una señal o fuente de energía, es una técnica en la que se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones, o para controlar la energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el periodo. Expresado matemáticamente:

$$D = \frac{t_1}{T}$$

Donde:

- **D es el ciclo de trabajo.**
- $t_1$ es el tiempo en que la función es positiva (ancho del pulso).
- T es el periodo de la función.

El PWM fue generado por el MCU con un periodo de 20 ms. Mediante el cambio del duty cycle se pudo controlar la cantidad de energía transferida a los motores.

**MCU**

El MCU tuvo que realizar muchas tareas en paralelo, es por esto que es de suma importancia tener en cuenta al principio los recursos disponibles, las estrategias posibles para resolver los problemas y el tiempo que lleva cada una.

Uno de los aspectos más importantes son los timmers, ya que para este proyecto se utilizaron todos los temporizadores disponibles, dada la necesidad de emitir ondas de luz infrarroja, ver la velocidad con la que se desplazaban las ruedas, la generación de PWMs para los motores e interrupciones periódicas para control de estado. Por ello es de suma importancia tener definido de antemano los recursos necesarios para realizar la tarea asignada.

Como tareas básicas, el MCU se encargó de:

- Controlar la velocidad de los motores.
- Medir la amplitud que devuelven los receptores.
- Recibir comandos serial por infrarrojo.
- Emitir señal infrarroja para los tanques enemigos.
- Medir distancias mediante el ultrasonido.
- Ejecutar algoritmo para resolver la tarea asignada.

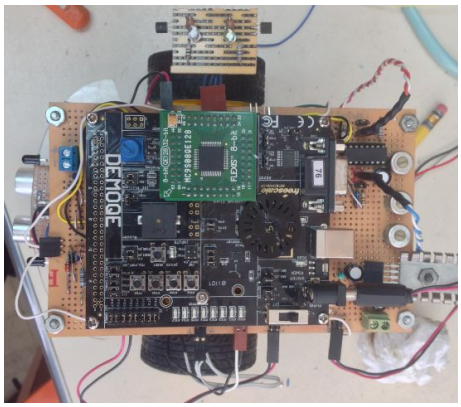



Figura #15 MCU y tarjeta de desarrollo utilizados.

Se utilizó el siguiente algoritmo como estrategia.

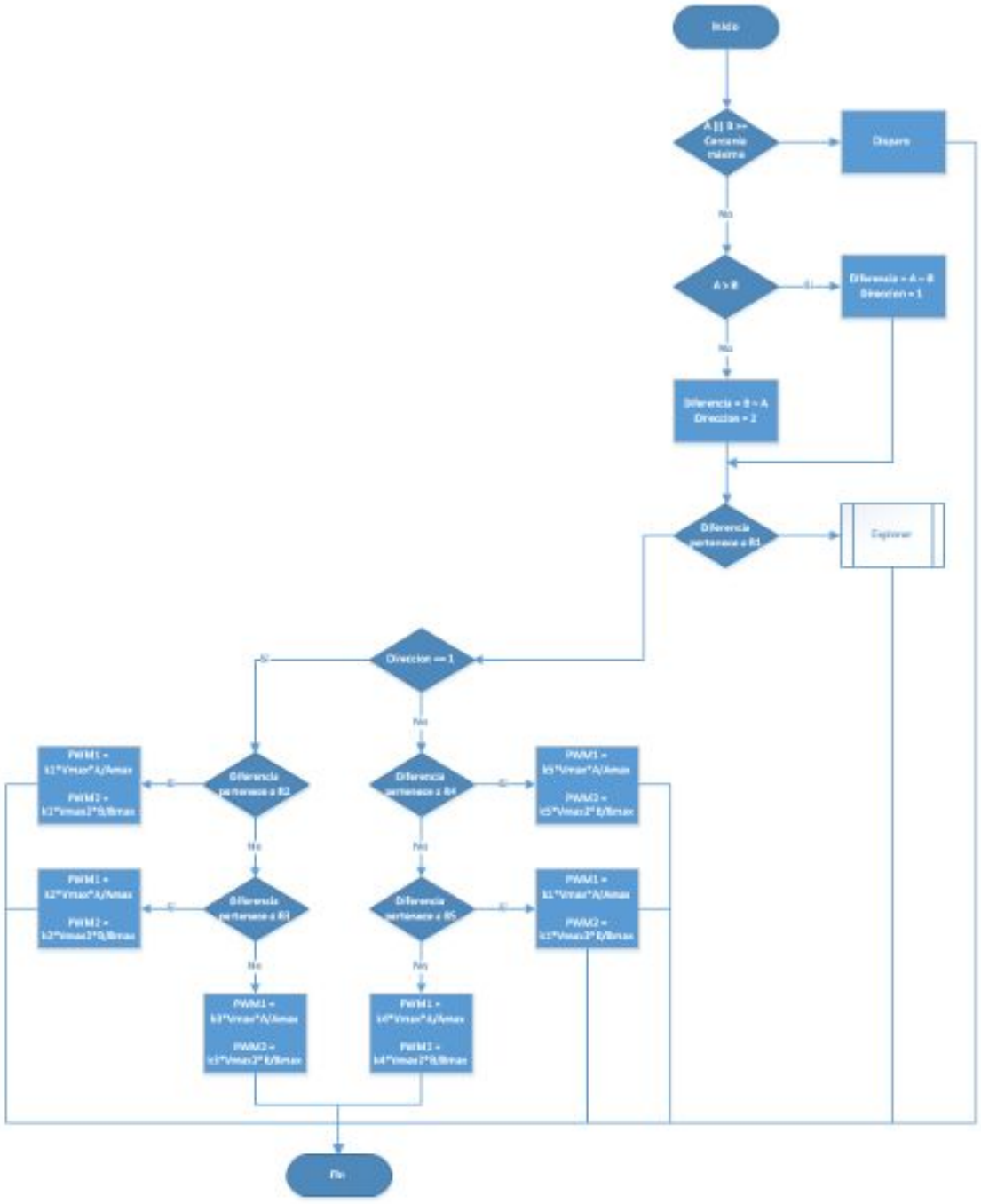



Diagrama #4: Diagrama de flujo del algoritmo implementado en el MCU

# Conclusiones y recomendaciones

Después de haber realizado este proyecto, se pudo llegar a las siguientes conclusiones y recomendaciones.

- Es importante tomar en cuenta el ancho de banda de los OPAMPs en la vida real, ya que los diseños realizados puede que no funcionen correctamente.
- Se recomienda asegurar los cables, ya que el mal accionar del robot causa estragos en la circuitería.
- Se recomienda realizar un análisis mecánico sencillo del robot antes de empezar a hacer circuitos, ya que esto permite una distribución sencilla, fácil de usar y segura para los circuitos y sus conexiones.
- Se debe tomar en cuenta los mAh de las baterías disponibles, para realizar las pruebas de manera apropiada.

# Anexos

```
while True: #Start (ports already opened)
    data = conn.recv(1) #Coord X data received from socket
    data2 = conn.recv(1) #Coord Y data received from socket
    if not data: break
    if not data2: break
    data="000"+data #Constant added to have 4 bits and use struct.unpack
    data2="000"+data2
    size=len(data) #Size measured to verify the length of the data
    size2=len(data2)
    if size==4: #If Coord X data has 4 bits
        myint= struct.unpack("!i",data) #Then use struct.unpack to transform string to int
        int1 = myint[0] #Int obtained from data received
        print("Los X")
        CoordenadaX=int1-808464384 #Int obtained minus the contant added to complete the 4
        print(CoordenadaX)    #Bits needed to use struct.unpack
    if size2==4: #Same
        myint2= struct.unpack("!i",data2)
        int3 = myint2[0]
        print("los Y")
        CoordenadaY=int3-808464384
        print(CoordenadaY)

    if size==4 and size2 == 4: #If both of them have 4 bits
        SCREEN.fill([0,0,0])   #Draw a square on the (X,Y) position obtained
        Carrito = pygame.draw.rect(SCREEN, (0, 0, 255), (CoordenadaX,CoordenadaY,100,100))
        pygame.display.flip() #Refresh the screen

    for event in pygame.event.get(): #Exit

      if event.type == pygame.QUIT:

          sys.exit()

conn.close() #Close com
```

Extracto #1: Código implementado en Python (servidor) para la primera versión de la interfaz

```
void draw () {
  background(0);
  tint(255,255,255,150);
  image(Background,0,0);
  noTint();

  switch (state) {
  case stateWelcomeScreen:
    handleStateShowWelcomeScreen();
    break;
  case statePlay:
    handleStatePlay();
    break;
  case stateLearnMore:
    handleStateLearnMore();
    break;
  case stateAfterAGame:
    handleStateAfterAGame();
    break;
  case stateHistory:
    handleStateHistory();
    break;
  case stateBattlefield:
    handleStateBattlefield();
    break;
  default:
    // error
    println("Estado indefinido, error: "
      + state
      + ".");
    exit();
    break;
```

Extracto #2: Manejo de cada estado en el Draw Principal del programa

```
void mousePressedStateBattlefield(){
  //User clicks on the vertices of the battlefield so we can determine if the robot is getting out
  //User clicks any key to continue to the actual game to select the colors of the robots
  BattlefieldClickCont++;
  switch(BattlefieldClickCont){
   case 1:
    Vertice1_X=mouseX;
    Vertice1_Y=mouseY;
    print("Vertice1 está en " + Vertice1_X+","+Vertice1_Y +"\n");
    break;
  case 2:
    Vertice2_X=mouseX;
    Vertice2_Y=mouseY;
    print("Vertice2 está en " + Vertice2_X+","+Vertice2_Y+"\n");
    break;
  case 3:
    Vertice3_X=mouseX;
    Vertice3_Y=mouseY;
    print("Vertice3 está en " + Vertice3_X+","+Vertice3_Y+"\n");
    break;
  case 4:
    Vertice4_X=mouseX;
    Vertice4_Y=mouseY;
    print("Vertice4 está en " + Vertice4_X+","+Vertice4_Y+"\n");
    break;
  case 5:
    state=statePlay;
    StartColorDetect=1;
    break;
  default:
    println ("Error en Input Mouse "
      + currentGame);
```

Extracto #3: Manejo del input del mouse para determinar los vértices del campo de batalla

```
void mousePressedStatePlay() {
  switch (currentGame) {
  case chooseGame1: //Camera
    //Track Color Routine for each color
    ColorsToTrack++;
    switch(ColorsToTrack){
     case 1:
       clicked=1;
       int loc = mouseX + mouseY*video.width;
       trackColor = video.pixels[loc];
       break;
     case 2:
       clicked2=1;
       int loc2 = mouseX + mouseY*video.width;
       trackColor2 = video.pixels[loc2];
       break;
     case 3:
       clicked3=1;
       int loc3 = mouseX + mouseY*video.width;
       trackColor3 = video.pixels[loc3];
       break;
     case 4:
       clicked4=1;
       int loc4 = mouseX + mouseY*video.width;
       trackColor4 = video.pixels[loc4];
       ColorsToTrack=0;
       break;
     default:
        println ("Error en Input Mouse "
        + currentGame);
        exit();
        break;
```

Extracto #4: Rutina del input del mouse para determinar cada color para el tracking

```
for (int x = 0; x < video.width; x ++ ) {
  for (int y = 0; y < video.height; y ++ ) {
    int loc = x + y*video.width;
    // What is current color
    currentColor = video.pixels[loc];
    float r1 = red(currentColor);
    float g1 = green(currentColor);
    float b1 = blue(currentColor);
    float r2 = red(trackColor);
    float g2 = green(trackColor);
    float b2 = blue(trackColor);
    float r3 = red(trackColor2);
    float g3 = green(trackColor2);
    float b3 = blue(trackColor2);
    float r4 = red(trackColor3);
    float g4 = green(trackColor3);
    float b4 = blue(trackColor3);
    float r5 = red(trackColor4);
    float g5 = green(trackColor4);
    float b5 = blue(trackColor4);

    float d = dist(r1, g1, b1, r2, g2, b2); // Distancia euclidiana del color actual con el color Track seleccionado
    float d2= dist(r1, g1, b1, r3, g3, b3);
    float d3= dist(r1, g1, b1, r4, g4, b4);
    float d4= dist(r1, g1, b1, r5, g5, b5);
```

Extracto #5: Distancia euclidiana entre el color actual y los deseados

```
    if (d < worldRecord) { //Current color similar to track color
      cont++;              //then the location will be saved along
      worldRecord = d;     //with the euclidian distance between them

      closestXOld=closestX;
      closestYOld=closestY;
      closestX = closestX+x; //Color Average. It will be divided by cont
      closestY = closestY+y; //This happens for all the colors to be tracked

    }
    if (d2 < worldRecord2) { //Theres a d3 and a d4 too
      cont2++;
      worldRecord2 = d2;

      closestXOld2=closestX2;
      closestYOld2=closestY2;
      closestX2 =closestX2+ x;
      closestY2 =closestY2+y;

    }
```

Extracto #6: Se guarda la posición del color actual y se hace el promedio

```
//Robot Team 1 Should move to the left to get back in
if(((closestX)/cont-SpriteSpace>RightX)||((closestX2)/cont2-SpriteSpace>RightX)){
  println("Robot Equipo 1 Se sale por la derecha del campo");

 contador1=contador1+1; //Using counters allows to send the
 contador2=0;           //Instruction only once
 contador3=0;           //There are as many counters as instructions to be sent
 contador4=0;
 contador5=0;
 contador6=0;
 contador7=0;
 contador8=0;
 contador9=0;
 if(contador1==1)
  myPort.write(0x01);

}
```

Extracto #7: Se envía por puerto serial la instrucción correspondiente para notificar al tanque

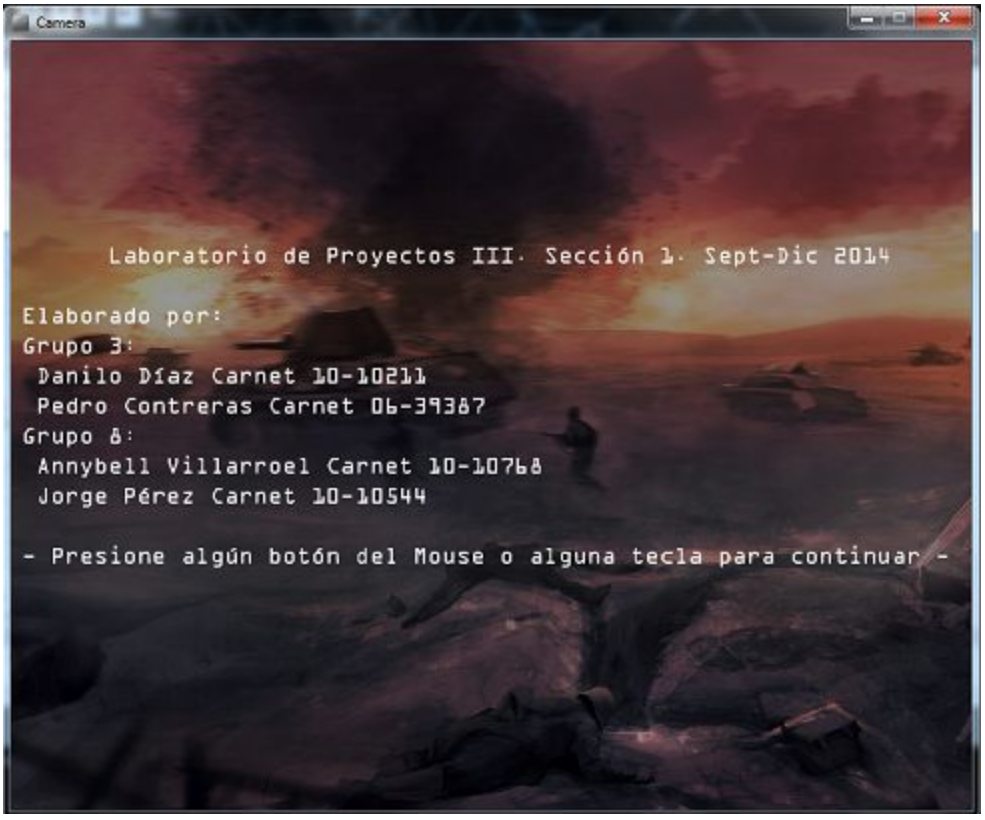



Figura #16: Estado de “Más Información” de la interfaz gráfica

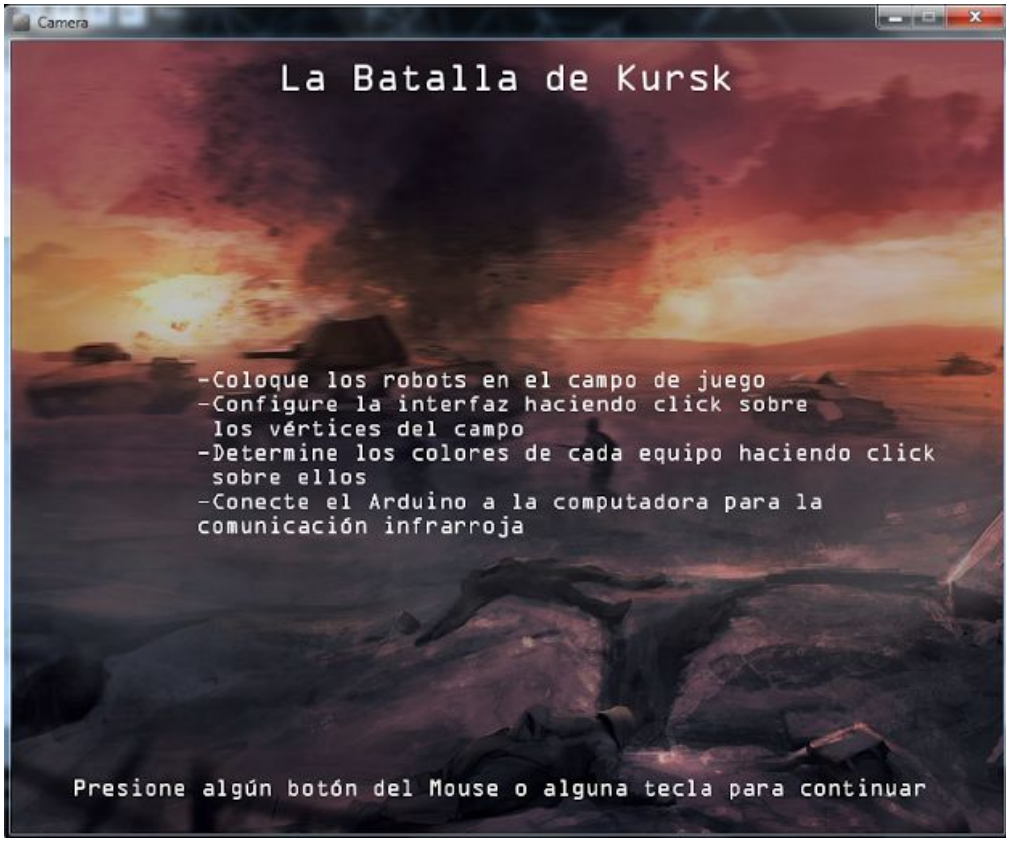


Figura #16: Estado de Instrucciones de la interfaz gráfica


Figura 17: Estado de Historia de la interfaz gráfica

# Referencias


- http://www.daycounter.com/Calculators/Standard-Resistor-Value-Calculator.phtml
- http://en.wikipedia.org/wiki/Sallen%E2%80%93Key_topology
- http://bkargado.blogspot.com/2013/09/todosobrehc-sr04.html
- http://users.ece.utexas.edu/~valvano/Datasheets/HCSR04b.pdf
- http://tinkerprojects.blogspot.com/2010/09/ir-sensor-for-arduino.html
- http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html
- https://learn.adafruit.com/ir-sensor