Vehículo Autónomo para la Reconstrucción de Laberintos - Proyecto Digital II

Danny Esteban Portela Robayo
cc: 1000952190

Omar Andrés Cely Villate
cc: 1002697298

Juan Felipe González Pardo
cc: 100596512

Agosto 4 de 2021

Resumen

En el presente documento se muestra el proceso de diseño de un vehículo capaz de recorrer de manera autónoma un laberinto sencillo que contiene un conjunto de figuras geométricas de diferentes colores (minas) en algunas de sus esquinas. Durante el recorrido, el vehículo puede identificar la existencia y color de las minas mediante una cámara que se encuentra en su parte frontal. La información de las minas y el recorrido es enviada a un PC, desde el cuál se hace una reconstrucción gráfica de la trayectoria del vehículo, así como de la ubicación y color de las minas.

Tras el diseño de los módulos de Hardware y Software aquí descritos, pudieron realizarse pruebas preeliminares en las que el reconocimiento de imagen funciona de manera satisfactoria, mientras que la navegación y reconstrucción de la trayectoria funcionan con algunos inconvenientes relacionados al control del comportamiento del vehículo en las rectas y el gran espacio que necesita al dar una curva, dificultando su movilidad en un laberinto estrecho. Se espera que estos inconvenientes queden cubiertos al hacer pruebas del funcionamiento completo del dispositivo para la entrega final, ya que actualmente, se han probado el reconocimiento de la imagen y la navegación de manera independiente.

1. Introducción

Como punto de partida en el diseño del vehículo se utilizaron los objetivos y requerimientos proporcionados en [1]. Posteriormente, con el fin de proporcionar una idea inicial de las funcionalidades del dispositivo, se elaboró el diagrama de bloques de la figura 1, mostrando de manera general sus entradas y salidas.

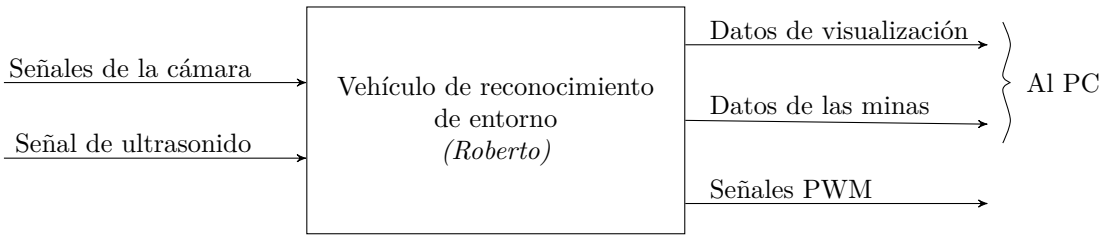


Figura N° 1: Diagrama de entradas y salidas del sistema.

A continuación, se da una breve descripción de cada una de las señales mostradas en la figura 1.

Entradas:

- **Señales de la cámara:** señales de sincronización horizontal y vertical, reloj de pixel y bytes en paralelo de las imágenes captadas por una cámara OV7670. El propósito de esta señal es proporcionar al dispositivo información de su entorno, principalmente relacionada al color y ubicación de las minas que se encuentren en el laberinto.

- **Señal de ultrasonido:** señales de activación y eco de tres sensores de ultrasonido HC-sr04. Estos sensores permiten tener una detección a tiempo real de los obstáculos (paredes) que tenga Vehículo enfrente y a los lados dentro del laberinto, permitiendo su navegación.

Salidas:

- **Datos de visualización:** información sobre el estado actual del vehículo y su distacia a la pared frontal. Se transmite de manera inalámbrica y permite al programa del PC dibujar la trayectoria seguida por el vehículo.
- **Señales PWM:** señales de ciclo útil variable que permiten a los motores del vehículo girar a diferentes velocidades con el fin de tener la movilidad adecuada para recorrer el laberinto.
- **Procesamiento del entorno:** información acerca de la existencia y color de las minas en las esquinas del laberinto. Se transmite actualmente por cable y complementa al programa que reconstruye la trayectoria seguida por el vehículo permitiéndole mostrar las minas que encuentre a su paso.

2. Desarrollo

Nota: el código fuente completo puede encontrarse en el repositorio del proyecto [2].

2.1. Particionamiento HW-SW

En la etapa inicial del diseño se planteó un particionamiento entre Software y Hardware para el cumplimiento de los requerimientos planteados de manera óptima. Los bloques de Hardware se implementaron en la tarjeta Zybo Z7 mediante el lenguaje de descripción de hardware Verilog, y el software se programó dentro de un microcontrolador ESP32 usando la IDE de Arduino. En el diagrama de bloques de la figura 2 pueden verse en detalle las conexiones entre los bloques de Hardware y Software planteados.

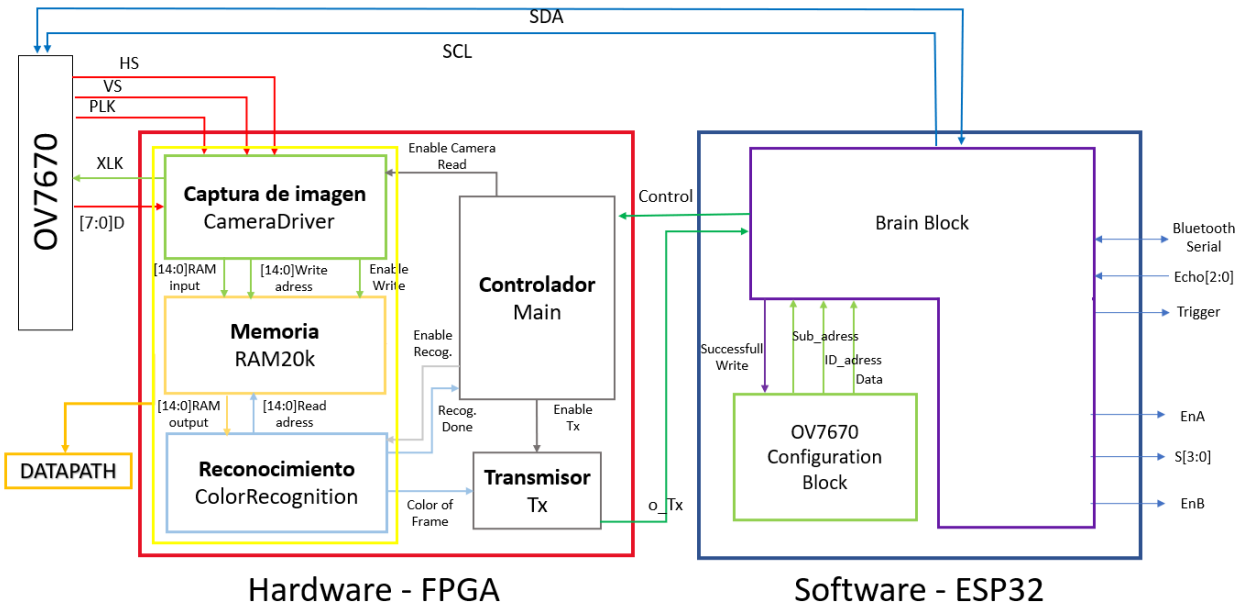


Figura N° 2: Distribución de la separación entre HW y SW

A continuación se describirán de manera detallada los componentes diseñados e implementados en el desarrollo del proyecto.

2.2. Hardware

Dentro del hardware diseñado en la FPGA, se siguió un esquema de Controlador-Datapath, de modo que existen módulos encargados de realizar operaciones específicas (como almacenamiento, captura de imagen, reconocimiento de color y transmisión de datos) y un módulo de control encargado de llevar la máquina de estados del sistema y mantener una sincronización adecuada entre los módulos del datapath.

2.2.1. Descripción de los módulos del datapath

- **Memoria**

  El módulo *RAM20k.v* contiene la descripción de una SRAM dual con capacidad para 20000 palabras de 8 bits. Dado que el propósito de este módulo es almacenar la información de un frame captado por la cámara, su dirección de escritura, datos de entrada y señal de activación están conectados directamente al módulo *CameraDriver.v*. Por otra parte, sus datos sirven para alimentar el módulo de reconocimiento de minas, así que la dirección de lectura y el bus de salida de datos se encuentran conectados al módulo *ColorRecognition.v*

  El tamaño de la RAM se estimó a partir del redondeo a la decena de millar inmediatamente superior al tamaño de un frame captado por la cámara en formato RGB555 con resolución de 79×72 ($79 \times 72 \times 2\,\text{Bytes} = 11\,376\,\text{Bytes}$). El espacio restante en la RAM tenía como propósito ser utilizado en una hipotética implementación de reconocimiento de formas.

- **Captura de imagen**

  El módulo de captura de imagen (descrito en *ReadImage.v*) es el encargado de propoporcionar el reloj de operación a la cámara, recibir las señales que esta proporciona y preparar los bytes de cada frame para ser almacenados en la memoria.

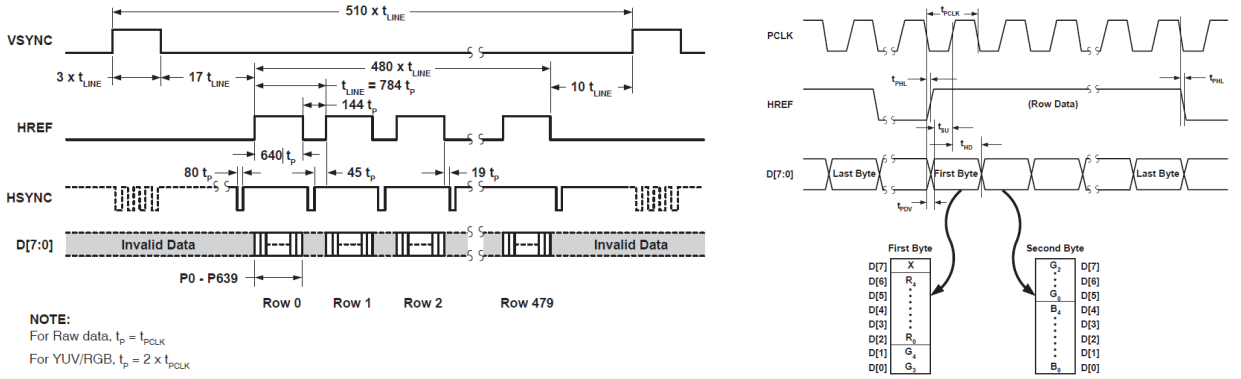
  Para la generación del reloj de entrada de la cámara se tuvo en cuenta el rango frecuencias proporcionado en [3], que va entre los 10 MHz y los 48 MHz. Dado que el reloj en el puerto K17 de la tarjeta Zybo Z7 tiene una frecuencia de 125 MHz, bastó con hacerle una división de 10 a su frecuencia para generar la señal *o_XLK* dentro del rango de operación, como puede verse en el código 1:

```
21      always @(posedge i_Clk) begin
22        PLK_Current_Value <=  i_PLK;
23        PLK_Previous_Value <=  PLK_Current_Value;
24        if (s_Clock_Count <4) begin
25          s_Clock_Count <= s_Clock_Count +1;
26        end
27        else begin
28          s_Clock_Count <=0;
29          s_Clock_Value <= ~s_Clock_Value;
30        end
```

Código 1: Fragmento de *ReadImage.v* en que se genera el reloj de entrada de la cámara.

Para la obtención y preparación de los bytes correspondientes al frame actual se partió de los diagrama de tiempos para las salidas de la cámara de la figura 3. Como puede verse allí, el comienzo de un frame viene marcado por un flanco de bajada en *i_VS* (VSYNC), demarcando el comienzo de cada línea con un flanco de subida en *i_HS* (HREF) y el envío de cada byte con un flanco de subida de

i_PLK(PCLK). De esta manera el código 2 almacena en memoria la información del bus de entrada *i_D* cada vez que haya un flanco de subida en *i_PLK*, se cumplan las condiciones de envío de una fila, y exista autorización del módulo de control para capturar una imagen (Autorización que se da mediante la entrada *i_EnableCameraRead*).



(a) Diagrama de tiempos de frame VGA.

(b) Diagrama de tiempos de salida RGB555

Figura N° 3: Diagramas de tiempos de las señales de salida de la cámara [3]

```
34      if (i_VS == 1'b0) begin
35        if((i_EnableCameraRead == 1'b1) && (i_HS == 1'b1)) begin
36          if(PLK_Posedge == 1'b1) begin
37            o_RAM_Write_Enable <= 1'b1;
38          end
39          else begin
40            o_RAM_Write_Enable <= 1'b0;
41          end
42
43          if(PLK_Negedge == 1'b1) begin
44            o_RAM_Adress <= o_RAM_Adress+1;
45          end
46          else begin
47            o_RAM_Adress <= o_RAM_Adress;
48          end
49        end
50        else begin
51          o_RAM_Adress <= o_RAM_Adress;
52          o_RAM_Write_Enable <= 1'b0;
53        end
54      end
55      else begin
56        o_RAM_Write_Enable <= 1'b0;
57        o_RAM_Adress <= 0;
58      end
59    end
60    always @(negedge i_Clk) begin
61        o_to_RAM <= i_D;
62    end
```

Código 2: Fragmento de *ReadImage.v* en que se capturan los datos provinientes de la cámara

Dado que la resolución de la cámara es bien conocida, se almacenan todos los datos de un frame de manera consecutiva sin ningún tipo de separador de filas. Al captarse un frame nuevo, éste sobreescribe la información del frame anterior al almacenarse en las mismas direcciones de memoria.

Antes de poder desarrollar el módulo de reconocimiento de imagen, hizo falta hacer algunas pruebas para verificar que las imágenes de la cámara se estaba recibiendo, procesando y almacenando de manera correcta. Por esta razón, se escribieron y probaron múltiples veces los archivos *Config.ino* y

CameraCapture.ino, que buscaban transmitir la información de un frame almacenada en la FPGA y transmitirla por medio de la ESP32 hasta un PC, donde finalmente se haría la visualización de la imagen.

Durante las pruebas de visualización se presentaron múltiples inconvenientes por causa de una incorrecta configuración de los registros de la cámara. Finalmente pudo obtenerse una imagen de manera exitosa al seguir la configuración QQCIF de la guía de implementación [5]. No obstante, la resolución obtenida resultó ser de 79×72, la cuál dista ligeramente del QQCIF esperado (88×72). En la figura 4 puede verse una de las primeras fotografías capturadas y mostradas exitosamente con la OV7670. Dicha fotografía muestra el frente de la estructura del vehículo.

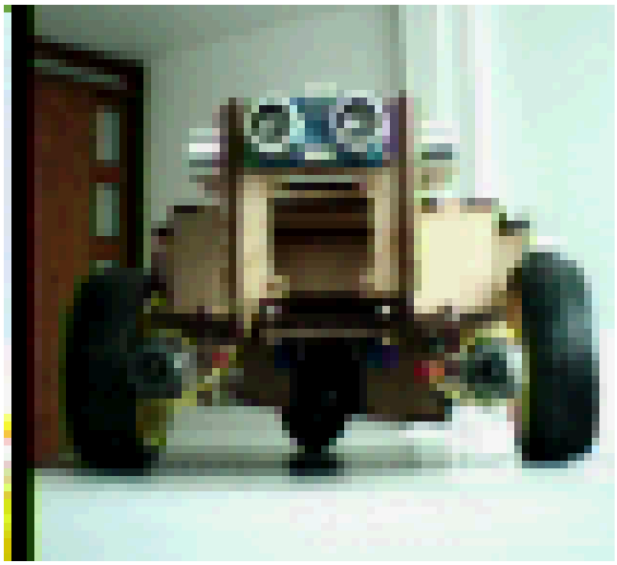


Figura N° 4: Fotografía de la estructura del vehículo tomada con la OV7670.

- **Reconocimiento**
  Como bien su nombre lo indica, este bloque (como representación del módulo *ColorRecognition.v*) es el encargado de analizar la información captada por la cámara y determinar que color posee la mina que haya detectado.
  Para lograr el reconocimiento de color de la imagen es necesario extraer la información del frame almacenado en la RAM20K. Con ello, a partir de la comparación de los componentes de RGB por cada pixel, es posible diferenciar entre *Rojo*, *Verde* y *Azul*. Por esto las entradas y salidas del módulo se plantean como:

```
1 module ColorRecognition( o_color, o_RAM_adress,o_done,i_enable,i_RAMinfo,i_BytesPerFrame, i_clk);
2 input [7:0]i_RAMinfo;
3 input [0:0]i_enable, i_clk;
4 input [14:0]i_BytesPerFrame;
5 output reg[7:0] o_color = 8'b11110000;
6 output reg[14:0] o_RAM_adress = 0;
7 output reg[0:0]o_done = 1'b0;
```

Código 3: Entradas y salidas del módulo de reconocimiento de color

Como entradas se tienen entonces: i_enable que corresponde la señal de control del módulo, i_RAMinfo que será la información proveniente de la RAM, i_BytesPerFrame como la cantidad de bytes que contiene 1 frame y finalmente el reloj de la fpga i_clk. Por otro lado, para el caso de las salidas es necesario

que una de estas sea la dirección de la RAM de la cual se quiere leer la información, ya que deberá cambiarse constantemente para analizar la totalidad del frame; dicha salida será o_RAM_adress. También se requiere de una variable que indique al bloque de control que el reconocimiento ha sido finalizado, de modo que se añade la salida o_done. Por último se tiene la salida o_color, un registro de 8 bits que almacenará el resultado del reconocimiento para posteriormente ser enviado a través del módulo Tx hacia la ESP32.

Detallando un poco más el procedimiento, al momento de realizar el reconocimiento se lleva a cabo el análisis por cada píxel, lo cual implica que se deben revisar 2 direcciones de la RAM dado el formato RGB555 (en cuanto a que los el primer byte contendrá la información del rojo y parte del verde, mientras que el segundo tendrá lo restante del componente verde y el azul). El análisis implica comparar cual de las componentes de color es la mayor para cada pixel, a la cual se le añadirá un punto a su contador total. Esto quiere decir que el objetivo es comparar que componente de color tiene la mayor cantidad de píxeles, y con esto concluir cual es el que predomina en la imagen captada; a continuación se presenta la sección de código de verilog que realiza esta acción:

```
if(r_totalWhite > 3000)begin
    o_color <= 8'b00000100;
end else begin
    if((r_totalRed > r_totalGreen) & (r_totalRed > r_totalBlue))begin
        o_color <= 8'b00000001;
    end
    if(((r_totalGreen-72) > r_totalRed) & ((r_totalGreen-72)> r_totalBlue))begin
        o_color <= 8'b00000010;
    end
    if((r_totalBlue > r_totalGreen) & (r_totalBlue > r_totalRed))begin
        o_color <= 8'b00000011;
    end
end
```

Código 4: Condiciones para la identificación de color

Es importante añadir la componente blanca que se almacena en el registro r_totalWhite, dado que por el funcionamiento general del dispositivo es necesario que este sea capaz de distinguir entre la presencia o no de una mina. Debido a esto, el registro aumenta en 1 (cuenta como un pixel blanco) cuando ninguna de las componentes es mayor a las otras dos, es decir si dos o más componentes de color tienen el mismo valor.
La condición para que una imagen sea captada como vacía es que la cantidad de píxeles sea mayor a 3000, lo que indica que aproximadamente un 60 % del frame es blanco.
Si se desea conocer a profundidad el funcionamiento y el código completo, este se encuentra disponible en el repositorio GitHub del proyecto [2].

Apuntes del funcionamiento: Con base en las pruebas realizadas al funcionamiento de este módulo, se lograron determinar ciertas consideraciones que tienen que ser tenidas en cuenta al momento de la implementación, las cuales son:

- Dada la configuración de la cámara OV7670, es necesario un punto de referencia para el contraste al momento de tomar una imagen o si no los colores resultaran incorrectos. Es decir que una imagen debe tener buena cantidad de píxeles blancos dentro del encuadre (que las minas posean un fondo blanco).
- Así mismo la iluminación debe ser homogenea sobre toda la imágen, mientras que el objeto a capturar no puede estar a menos de 3 cm o más de 14 (dependiendo del tamaño de la figura) ya que puede resultar en cambios del color o una detección falsa.
- Es necesario esperar entre el cambio de un frame y la toma del siguiente. Esto implica que al momento de que se posicione el vehículo tiene que esperar aproximadamente 2 segundos antes de

tomar la foto a analizar. Esto se debe a que si se toma una imagen en movimiento se generan aberraciones cromáticas con píxeles de color verde, lo cual produce al final un error en el análisis.

- **UART**

  También conocido como *comunicación serial*, sus siglas en inglés se refieren *Universal asynchronous receiver-transmitter*. Es un protocolo o interfaz de comunicación que consiste en el envío o recepción de bytes de manera serial mediante dos conexiones (una para envío y otra para recepción) sin incluir un reloj, por ello es que se le denomina asíncrono.
  Dado que la comunicación no posee una señal de reloj de referencia, es necesario que el receptor esté constantemente buscando por un bit de iniciación. Si el receptor no está realizando el muestreo a la velocidad correcta resultara en captar la información errónea; por lo tanto tanto el emisor como el receptor tienen que coincidir en el **baud rate**, el cual es la velocidad de transmisión de la información, referida en bits por segundo.
  El principio de funcionamiento en la FPGA al momento de recibir se basa en un muestreo constante de la señal recibida que se mantiene en 1, y al presentarse un flanco de bajada se detecta el inicio de la transmisión. Aquí se debe esperar medio ciclo de duración de bit, para así asegurar que se mide en medio del bit de inicio; por lo cual en adelante se medirá cada periodo de bit (definido por el **Baud rate**). En la figura 5 se presenta la forma que posee el flujo de bits en serie que se enviarían en un Byte mediante el protocola UART.

Figura N° 5: Flujo de bits mediante comunicación serial [7]

Entrando en la implementación para el proyecto primordialmente en el lenguaje de descripción de hardware Verilog, es necesario generar un módulo tanto para la transmisión (Tx) como para la recepción (RX). Dadas las implicaciones del proyecto se consideró la opción de un planteamiento completo de dichos módulos, sin embargo resultó más conveniente emplear elementos ya disponibles en la red para la implementación. Por esto, a continuación se detallarán ciertas secciones del código empleado, que fue extraído de la fuente [7] y en donde se puede encontrar más información al respecto.

- **Modulo de Recepción RX**:

```
1  module Rx (
2     input        i_Clock,
3     input        i_Rx_Serial,
4     output       o_Rx_DV,
5     output [7:0] o_Rx_Byte
6    );
7    parameter CLKS_PER_BIT = 1085;
8    parameter s_IDLE         = 3'b000; // State (Rest)
9    parameter s_RX_START_BIT = 3'b001; // State (Star bit)
10   parameter s_RX_DATA_BITS = 3'b010; // State (Bits' Middle)
11   parameter s_RX_STOP_BIT  = 3'b011; // State (Stop bit)
12   parameter s_CLEANUP      = 3'b100; // State (Clean up)
13   reg           r_Rx_Data_R = 1'b1; // Auxiliar
14   reg           r_Rx_Data   = 1'b1; // Bit recived sets in 1
15   reg [10:0]     r_Clock_Count = 0; //
16   reg [2:0]     r_Bit_Index   = 0; // 8 bits total (Which bit is going)
17   reg [7:0]     r_Rx_Byte     = 0; // Array where we save th bits before send it to o_RX_Byte
18   reg           r_Rx_DV       = 0;
19   reg [2:0]     r_SM_Main     = 0; // Main State Machine
```

Código 5: Entradas y salidas del módulo RX

Sin entrar excesivamente en detalle, en el anterior código se muestra la parte inicial del modulo RX, el cual consiste en dos entradas y dos salidas. Las entradas siendo el reloj a emplear (en este caso particular el de 125 MHz de la FPGA zybo-7), y la información proveniente de la conexión física i_Rx_Serial. En tanto las salidas, estas serán el registro donde se almacene el byte de llegada (o_Rx_Byte) y un indicador de finalización (o_Rx_DV).
Este módulo fue planteado como una máquina de estados finitos con 5 estados, que corresponden a:

-s_IDLE: Estado por defecto cuando no se está recibiendo información.
-s_RX_START_BIT: Estado cuando se recibe el bit de iniciación.
-s_RX_DATA_BITS: Estado en el que lee la información entrante al modulo.
-s_RX_STOP_BI: Estado en el que lee el bit de finalización.
-s_CLEANUP: Estado final para limpiar los registros.

Además de los estados, se tiene el parámetro de CLKS_PER_BIT, el cual corresponde a la cantidad de ciclos de reloj que toma un bit en ser leído. Este valor será el que determine la velocidad en baudios de la comunicación, y se determina a partir de la velocidad del reloj principal; por lo que si se tiene que el transmisor posee una velocidad de 115200 baudios como es el caso:

$$\text{CLKS\_PER\_BIT} = \frac{125 \times 10^6}{115200} = 1085{,}069$$

Se emplean demás registros para poder hacer el tratamiento de los bits dentro de los ciclos *always*. Si se desea conocer el código completo de verilog, este se puede encontrar en repositorio de *Github* del proyecto, en la sección CameraModules.

- **Modulo de transmisión TX**

```
1   module Tx(i_Clock, i_Tx_DV, i_Tx_Byte, o_Tx_Active, o_Tx_Serial, o_Tx_Done);
2
3   input       i_Clock;
4   input       i_Tx_DV;
5   input [7:0] i_Tx_Byte;
6   output      o_Tx_Active;
7   output reg  o_Tx_Serial;
8   output      o_Tx_Done;
9
10  parameter s_IDLE         = 3'b000;
11  parameter CLKS_PER_BIT   = 1085;
12  parameter s_TX_START_BIT = 3'b001;
13  parameter s_TX_DATA_BITS = 3'b010;
14  parameter s_TX_STOP_BIT  = 3'b011;
15  parameter s_CLEANUP      = 3'b100;
16
17  reg [2:0]    r_SM_Main     = 0;
18  reg [10:0]    r_Clock_Count = 0;
19  reg [2:0]    r_Bit_Index   = 0;
20  reg [7:0]    r_Tx_Data     = 0;
21  reg          r_Tx_Done     = 0;
22  reg          r_Tx_Active   = 0;
```

Código 6: Entradas y salidas del módulo TX

De igual manera que con el módulo RX, no se entrará muy en detalle del funcionamiento del modulo de transmisión, pero se mencionaran ciertos parámetros importantes.
Para la transmisión se posen 3 entradas al módulo, siento nuevamente el reloj de la FPGA, una variable de control i_Tx_DV que es la que habilitará el envío de la información entrante por i_TX_Byte. Las salidas por su parte corresponden a un indicador de que el modulo está activo (o_Tx_Active), la salída física de la fpga que enviará los bytes de información que corresponde a o_Tx_Serial, y finalmente el indicador o_Tx_Done para cuando el módulo ha enviado el byte de información.
Este bloque de igual forma se planteó como una maquina de estados finita cuyos pasos son:

-s_IDLE: Estado por defecto cuando no se está transmitiendo información.
-s_TX_START_BIT: Estado que realiza el flanco de bajada para indicar el bit de inicio.
-s_RX_DATA_BITS: Estado en el que se envía bit por bit a través de la salida o_Tx_Serial.
-s_RX_STOP_BI: Estado en el se escribe el bit de finalización.
-s_CLEANUP: Estado final para volver al estado inicial y dejar la salida serial en un 1 constante.

Así mismo, está presente la variable CLKS_PER_BIT que indicará la velocidad de transmisión de la información, que se determina de manera idéntica a como se realizó en el módulo **RX**.
Si se desea conocer el planteamiento completo del módulo es posible remitirse una vez más a la sección de CameraModules en el repositorio de *Github*, exactamente en el archivo llamado **Tx**.

2.2.2. Control

Como elemento fundamental del bloque de HW se encuentra el módulo de control (equivalente al módulo *main.v* en verilog), el cual es el encargado de modificar las variables de control que activan o desactivan los demás módulos, a partir de las condiciones necesarias. Para el correcto funcionamiento de este fue necesario implementar una máquina de estados finitos, en donde cada estado realizará una función, desde la carga de la información de un frame hasta el envío del reconocimiento de color a la ESP32.
Con el propósito de un mejor entendimiento de la áquina de estados, se generó el diagrama de FSM que se encuentra en la figura 6.

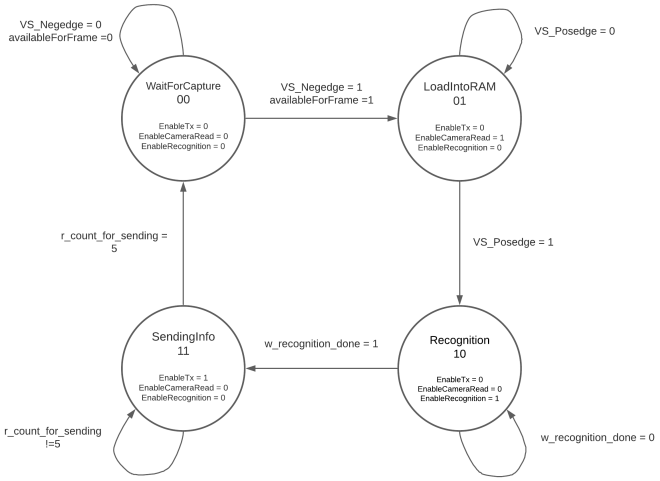


Figura N° 6: Diagrama de máquina de estados finitos para el módulo de Control

Por lo tanto, a partir de la figura 6 se nota como el control del hardware se realiza mediante 4 estados, los cuales poseen las siguientes funciones:

WaitForCapture: Es el estado por defecto de la FSM, en el cual se espera por la señal de activación para la toma de imagen *availableForFrame*, que proviene directamente de la ESP32. Además también espera por un flanco negativo de la señal *VS* proveniente de la cámara, que indicaría el inicio de la toma de un frame.

LoadIntoRAM: Al recibir la señal de activación y la presencia de un flanco negativo de VS, este esta-

do la carga de información de la cámara a la RAM20k. Al detectar un flanco positivo VS se indíca la finalización del la toma de la imagen por lo que se pasa al siguiente estado,

Recognition: Ya cargada la información de un frame en la RAM, el estado activa la señal de control para el módulo de ColorRecognition. Este proceso se realiza hasta que el propio bloque genera la señal de control w_recognition_done, que indica el final del proceso de análisis de color y permite el paso de estado.

SendingInfo: Teniendo la información obtenida del reconocimiento, esta se carga al módulo de transmisión Tx, mediante el cual será enviado a la ESP32. Para que esto sea posible, el último estado del módulo de Main activa la señal de control para que el envío se pueda realizar (haciendo $EnableTx = 1$).

Para el correcto funcionamiento de este módulo, se requiere un correcto instanciamiento de los módulos del **Datapath**, realizando las conexciones adecuadas. A continuación se presenta esta sección dentro del código de Verilog:

```
RAM20k RAM (w_RAM_Output,
            w_Write_Adress,
            w_Read_Adress,
            w_RAM_Input,
            1'b1,
            w_Enable_Write,
            Clk);

ReadImage Image (o_XLK,
                 w_RAM_Input,
                 w_Write_Adress,
                 w_Enable_Write,
                 i_D,
                 i_PLK,
                 Clk,
                 i_VS,
                 i_HS,
                 r_EnableCameraRead);

Tx Transmiter(Clk,
              r_Enable_Tx,
              w_colorOfFrame,
              o_led[0],
              o_Tx,
              o_led[1]);

ColorRecognition Color(w_colorOfFrame,
                       w_Read_Adress,
                       w_recognition_done,
                       r_EnableRecognition,
                       w_RAM_Output,
                       BytesPerFrame,
                       Clk);
```

Código 7: Instanciamiento de módulos del Datapath en el módulo de control

En el anterior código se muestran las instancias de los 4 módulos explicados anteriormente del Datapath, desde la memoria RAM, la carga de imagen, el reconocimiento y finalmente la transmisión.
Se observa por ejemplo, como la entrada de datos de la RAM (w_RAM_input), así como la dirección de escritura (w_Write_Adress) vienen directamente del módulo ReadImage, ya que este se encarga de tomar los datos de los píxeles captados por la cámara y son enviados a la RAM para ser almacenados. Por otro lado, se tiene que la dirección de lectura de la memoria (w_Read_Adress) es una salida controlada por el módulo de ColorRecognition, dado que aquí es donde esta cambia para solicitar a la RAM la lectura de ciertas direcciones (w_RAM_Output).Así mismo, la salida del módulo de reconocimiento w_ColorOfFrame, es la entrada al módulo de transmisión ya que es la que será enviada al dispositivo de software.

De manera identica a como se ha desarrollado con los anteriores módulos, es posiblñe revisar y conocer

el contenido completo de los códigos de verilog a través del repositorio de GitHub del proyecto [2]

2.3. Software

Siguiendo el estándar de diseño elegido (Particionamiento HW/SW), se describe la parte programable mediante el uso de una ESP32. Sus funciones principal son: la lectura de las distancias por parte de los sensores de ultrasonido HC-SR04, generación de las señales PWM, control de trayectoria y comunicación Bluetooth con el PC, para la visualización del camino.

Se construye un programa ejecutable en Arduino IDE para el cumplimiento de las tareas requeridas. Teniendo en cuenta el siguiente diagrama de flujo:

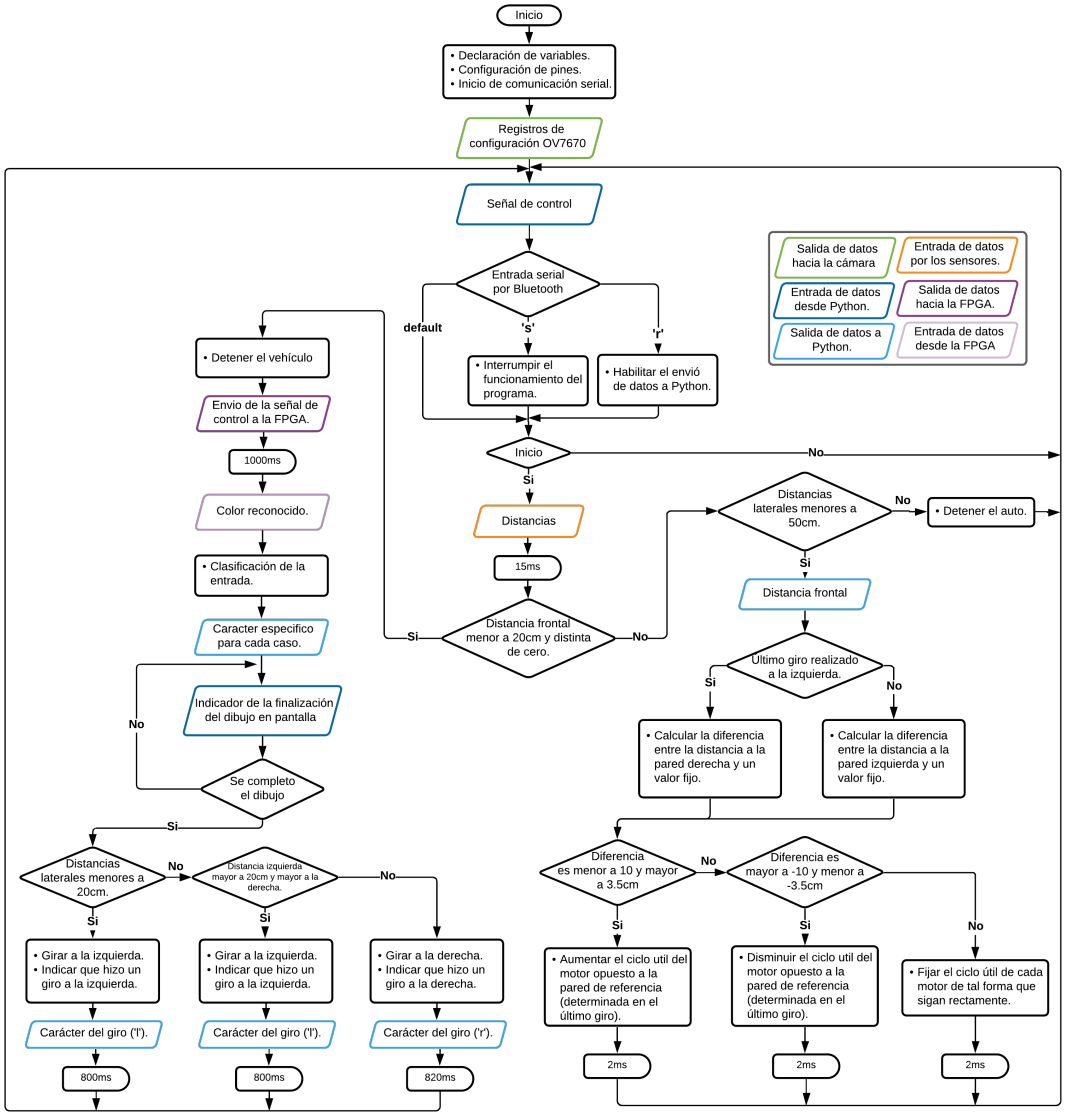


Figura N° 7: Diagrama de flujo - ESP32

De manera general, cuando se ejecuta el programa, se realiza la escritura de los registros para la configuración de la cámara utilizada. Después, se espera la señal de control para el inicio del vehículo.

El código escrito sigue una estructura como la mostrada en la Figura 1, dividiendo el programa en dos archivos de extensión .ino, con tareas especificas para el funcionamiento conjunto con los periféricos.

2.3.1. Brain Block

En este archivo se declaran las variables globales y el funcionamiento de los pines. Además se escriben los registros de la cámara (con la ayuda de la archivo *OV7670ConfigurationBlock.ino*, que se explica más adelante), se inicia la comunicación serial (Bluetooth y UART) y se procesan los datos de los sensores.

Uno de los objetivos del proyecto es permitirle al usuario el inicio y detención del dispositivo de manera remota desde el computador. Para esto, se hace uso de ciertos caracteres para indicar dichas tareas, que se envían por Bluetooth desde el PC. Además, permitir el monitoreo del movimiento y el reconocimiento de las minas. Por lo que se planteó un sistema de caracteres que se asociaran a una tarea especificas.

Salidas	
Carácter	**Función**
'l'	Indicar que el vehículo realizó un giro a la izquierda.
'r'	Indicar que el vehículo realizó un giro a la derecha.
'b'	Indicar que el vehículo detectó una mina color azul.
'g'	Indicar que el vehículo detectó una mina color verde.
'x'	Indicar que el vehículo detectó una mina color rojo.
'n'	Indicar que el vehículo no detectó mina.
default	Indicar que el vehículo avanza, por lo que se recibe la distancia frontal captada.

Entradas	
Carácter	**Función**
'r'	Indicar que el usuario oprimió la tecla 'r' en el teclado, lo que permite iniciar las tareas del vehículo.
's'	Indicar que el usuario oprimió la tecla 's' en el teclado, lo que permite detener las tareas del vehículo y el reinicio del camino.

Tabla N° 1: Caracteres involucrados en la comunicación vía Bluetooth serial, acompañado de su función dentro del código.

Al iniciar el código se espera la señal de control ('r') que proviene del PC. Una vez captada esta señal de control, la ESP32 empieza a enviar datos y ejecutar tareas de control, dependiendo los datos de entrada de los sensores HC-SR04. Al tener la información de las distancias se clasifican las condiciones para la ejecución de algunos procesos.

De manera general se explica cada condicional construida y las tareas que se realizan cuando estas se cumplen.

- **Encuentra una pared**

Cuando esto sucede, el dispositivo se detiene y envía la señal de control a la FPGA para que guarde un frame y procese la imagen. Una vez identificado el color, la FPGA envía un número que representa a cada uno. Una vez llega este número, se asocia a un carácter para enviarlo vía Bluetooth al PC y se espera la confirmación del dibujo en pantalla.

Al terminar el proceso anterior, ejecuta el giro pertinente a las distancias medidas y envía el carácter asociado al movimiento que hizo. Un caso especial que se tuvo en cuenta en el código es cuando se encuentra una pared y las distancias laterales son grandes, en este caso se prioriza el giro a la izquierda.

- **Avance**

Cuando las distancia laterales son menores a 50 cm el vehículo avanza, envía la distancia frontal captada al PC y ejecuta las correcciones pertinentes dependiendo a las distancias laterales medidas. Para eso, se prioriza las correcciones a un pared de referencia, que al iniciar el vehículo es la izquierda y va cambiando dependiendo el giro que realizó. Por ejemplo: cuando ejecuta un giro a izquierda, prioriza la derecha y hace las próximas correcciones con respecto a esa pared y viceversa.

La distancia ideal a la pared quedó fijada en 12 cm, por lo que en cada ciclo se determina la diferencia entre este y la distancia medida. Si el resultado es negativo, indica que el vehículo se leja de la pared, por lo que aumenta el ciclo útil del motor opuesto para acércalo. Por otro lado, si el resultado es positivo el vehículo se acerca y se aumenta la velocidad del motor junto a la pared.

- **Detenerse**

Cuando las distancias laterales son muy grandes el vehículo se detiene, indicando que ya salió del laberinto.

Función "readUltrasonic"

El sensor de ultrasonido HC-SR04 cuenta con una precisión de ±3 mm [9], por lo que es necesario la construcción de una función capaz de retornar la distancia con un mayor número de cifras significativas. La función fue elaborada partiendo de la documentación [8]. Sin embargo, se preciso el valor utilizado como velocidad del sonido.

```
72 double readUltrasonic(int trig, int echo){
73   digitalWrite(trig, LOW);
74   delayMicroseconds(2);
75   digitalWrite(trig, HIGH);
76   delayMicroseconds(10);
77   digitalWrite(trig, LOW);
78   return pulseIn(echo, HIGH)* 0.01716;
79  }
```

Código 8: Fragmento de *DecisionBlock.ino* en que se define la función *readUltrasonic*.

2.3.2. OV7670 Configuration Block

La cámara OV7670, al igual que todos los chips de cámara de OmniVision, utiliza el protocolo de comunicación SCCB (Serial Camera Control Bus) para hacer operaciones de lectura y escritura en sus registros de configuración. El protocolo SCCB puede implementarse con 2 o 3 cables dependiendo de la cantidad y tipo de los dispositivos conectados; además tiene variaciones según se soporten o no entradas lógicas de triestado. En el caso concreo de la cámara empleada, se usó el protocolo de dos cables sin triestado descrito

en [5] para configurar desde una tarjeta ESP32 la resolución, orientación y colores de la cámara.

Si bien SCCB de dos cables es compatible con el estándar I2C, se elaboró un archivo específico (*SCCB_OV7670.ino*) para entender correctamente el protocolo de comunicación y hacer corrección de errores de manera más efectiva. Las funciones presentes en dicho archivo fueron elaboradas partiendo de la documentación de [4] y el código de [6].

Dentro de *SCCB_OV7670.ino* pueden encontrarse dos funciones directamente ejecutables: *readRegister* y *writeRegister*. A partir de la llamada de estas funciones en archivos externos es posible configurar los registros deseados de la cámara y hacer una lectura para verificar que la información que tienen almacenada es la correcta. En los códigos 9 y 10 puede verse la definición, huella y tipo de retorno de las funciones *writeRegister* y *readRegister*.

```
2  byte readRegister(byte IdAdress, byte SubAdress, int SCL, int SDA, int Delay){
3      byte Message[2] = {IdAdress-1,SubAdress};
4      ////////////////Two Phases write Transmission//////////////////
5      startTransmission(SCL, SDA, Delay);
6      for (int phase = 1; phase <3; phase++){
7            if (!writeByte(Message[phase - 1], SCL, SDA, Delay)){
8              }
9       }
10      stopTransmission(SCL,SDA,Delay);
11      ////////////////Two Phases read Transmission//////////////////
12      startTransmission(SCL, SDA, Delay);
13      writeByte(IdAdress, SCL, SDA, Delay);
14      byte ReadedData = readByte(SCL, SDA, Delay);
15      stopTransmission(SCL, SDA, Delay);
16      return ReadedData;
17    }
```

Código 9: Fragmento de *SCCB_OV7670.ino* en que se define la función *readRegister*.

```
22 bool writeRegister(byte IdAdress, byte SubAdress, byte InputValue, int SCL, int SDA, int
       Delay){
23      bool writingWasPerfect = true;
24      byte Message[3] = {IdAdress,SubAdress, InputValue};
25      startTransmission(SCL, SDA, Delay);
26      for (int phase = 1; phase <4; phase++){
27           if (!writeByte(Message[phase - 1], SCL, SDA, Delay)){
28               writingWasPerfect = false;
29             }
30       }
31       stopTransmission(SCL,SDA,Delay);
32       return writingWasPerfect;
33    }
```

Código 10: Fragmento de *SCCB_OV7670.ino* en que se define la función *writeRegister*.

Usando la función *writeRegister* mostrada previamente, se siguió la guía de implementación en [5] para Configurar la cámara a una resolución de 79x72 en RGB555. Adicionalmente, se configuró la matriz de conversión de color para obtener colores más vivos y, se invirtió la imagen vertical y horizontalmente para facilitar su visualización. El bloque y su implementación pueden encontrarse en el Repositorio del Proyecto [2].

2.4. Visualización del camino y control

Para la construcción del laberinto se hace uso de Python, ya que cuenta con distintas librerías que facilitan el cumplimiento de los objetivos de manera sencilla.

Se elaboró el diagrama de flujo 8, que describe de manera general las tareas necesarias para el cumplimiento de los objetivos planteados en el proyecto.

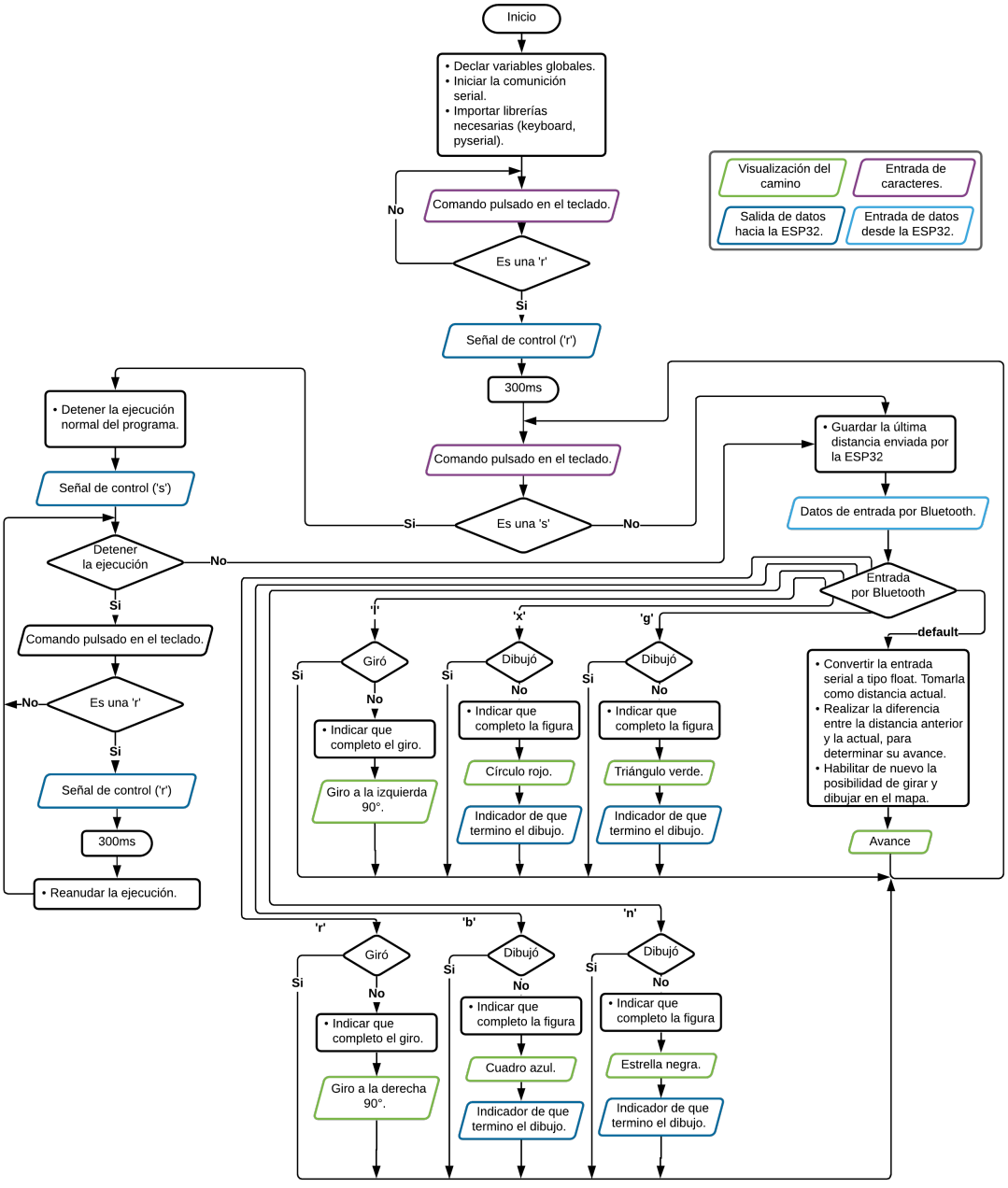


Figura N° 8: Diagrama de flujo - Python

De manera general, el anterior diagrama describe la comunicación serial con la ESP32. Ejecutando las tareas que se asocian a cada carácter propuesto en la tabla 1.

El propósito del programa es lograr la comunicación serial con el microcontrolador de forma remota.

Además, interpretar los datos recibidos y enviar datos devuelta si es el caso. Con el fin de cumplir estás tareas se utilizan las librerías:

- **keyboard:**

  Permite escuchar un evento onClick, con el que se determina que carácter fue oprimido en el teclado.

- **pyserial:**

  Este módulo encapsula el acceso a los puertos seriales. Para recurrir a uno de estos, se declara un objeto, aclarando el puerto donde se encuentra conectado el microcontrolador y la velocidad de transmisión de datos.

- **turtle:**

  Turlte graphics proporciona herramientas para el diseño de figuras primitivas en dos dimensiones. Esta librería fue escogida, ya que cumple con los requerimientos básicos para la visualización del camino de forma sencilla.

El código fue dividido en dos archivos .py, contando con tareas especificas cada uno. Sin embargo, para el entendimiento del funcionamiento en general, solo se requiere explicar el archivo principal *RoadConstruction.py*.

Nota: El archivo secundario (*tools_RoadConstruction.py*) se encuentra disponible en el repositorio del proyecto, en dicho archivo se declaran las funciones utilizadas en el archivo principal.

- **RoadConstruction**

```
1  import keyboard
2  import tools_RoadConstruction
3
4  firstTurn = False
5  firstTime = True
6  lastDistance = 0
7  tools_RoadConstruction.start()
8
9  while True:
10
11     if keyboard.is_pressed("s"):
12         tools_RoadConstruction.ser.write(b's')
13         tools_RoadConstruction.turtle.bye()
14         tools_RoadConstruction.start()
15
16     else:
17
18         (firstTurn, firstTime, lastDistance) =
               tools_RoadConstruction.makeRoad(lastDistance, firstTurn, firstTime)
```

Código 11: Archivo principal *RoadConstruction.py*.

Al inicio del código se utiliza la función *start()*, donde se envía la señal de control ('r') que da inicio al vehículo, permitiendo la ejecución de sus tareas.

Después el programa entra en el ciclo principal, donde se evalúa constantemente si se oprime la tecla 's' para detener la operación del dispositivo y permitir el reinicio del sistema. Si no es el caso, el computador recibe los datos enviados por serial, los filtra y ejecuta las tareas especificadas en la tabla 1 para cada entrada posible. Lo anterior se realiza en la función *makeRoad* que se encuentra en el archivo

secundario.

Nota: para determinar la distancia que avanzó en dispositivo, constantemente se evalúa la diferencia entre la distancia que llegó y la anterior (esto se ejecuta cuando los datos que llegan no son alfabéticos). También es importante aclarar que las variables *firstTurn* y *firstTime* sirven como auxiliares que no permiten que se dibuje dos veces lo mismo si llega el mismo carácter por serial.

3. Integración

En esta sección se discutirán ciertos elementos y detalles que tuvieron cabida al momento de la implementación final, con la integración del software y hardware. Para ello, se tiene que la unión entre estos dos grandes bloques se lleva a cabo mediante comunicación UART y una señal de control, como lo indica la imagen de la figura 9.

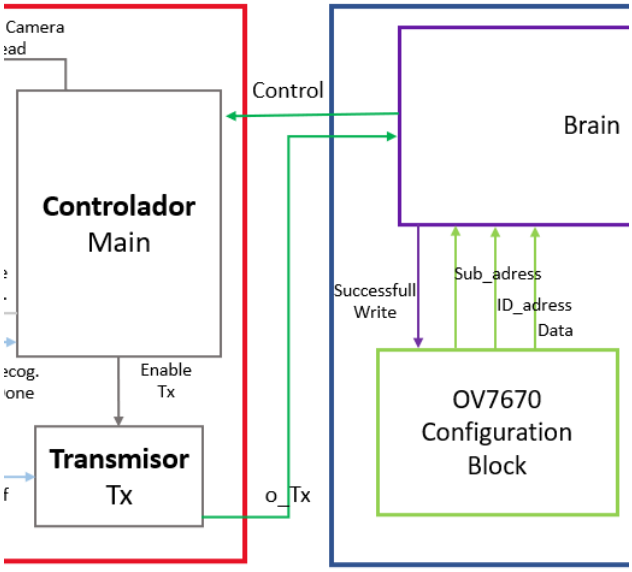


Figura N° 9: Conexión entre Software y Hardware

En esta se observan como únicamente se emplea el cable de transmisión o_Tx, para enviar a través de este la información captada por el hardware respecto al reconocimiento de color. Por otro lado, el cable de control es mediante el cual el software solicida al hardware la captura de imagen para el posterior reconocimiento.
Por lo tanto, en la sección de *BrainBlock* perteneciente al software, que se encarga del movimiento en general del dispositivo, es la que indica cuando tomar una foto para reconocer el color de la figura. Debido a la limitaciones inherentes al funcionamiento de la cámara OV7670, es necesario que el vehículo se detenga 5 segundos antes de tomar captura de una imagen, para así evitar que el movimiento del lente genere aberración cromática. Es decir, dado que la cámara se encuentra en la parte frontal del vehículo y este se detiene antes de los giros, únicamente se podrán reconocer las minas colocadas en las intersecciones. Este hecho produce la configuración del laberinto que se encuentra en la imagen 10.

Figura N° 10: Recorrido Que efectúa el vehículo

Finalmente se realizo el montaje final del vehículo, integrando en el chasis la FPGA (hardware) y la ESP32 (software), junto con los sensores de ultrasonido y motores. El resultado es el presentado en la figura 11.

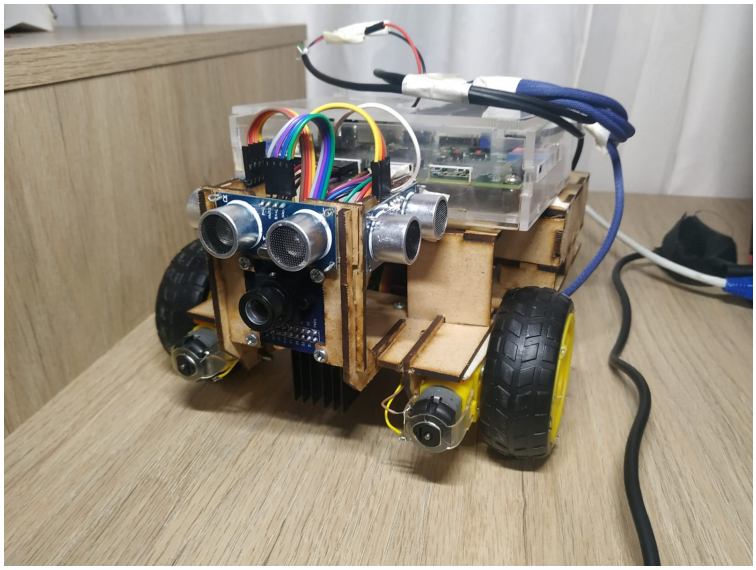


Figura N° 11: Estado final del vehículo

Buscando la practicidad y fácil programación, se emplea el mismo cable de conección al PC del ESP32 como alimentación. Para el caso de la FPGA, es necesario realizar su alimentación directamente con 5 voltios desde la fuente, al tiempo que se conecta un cable USB a micro USB para la programación.

3.1. Resultados y análisis

Al realizar la integración y las pruebas subsecuentes, se observaron ciertos comportamientos que no permitieron el recorrido total del dispositivo dentro del laberinto planteado.
Si bien el planteamiento en general permite el movimiento en un escenario de gran tamaño y amplitud, resulta más difícil un correcto control dentro de espacios pequeños, que fue el caso debido a limitaciones de espacio.
Estos problemas se atribuyen principalmente a la efectividad de los sensores de ultrasonido y motores DC, debido a la variabilidad y poco control que se tiene de ambos dentro de los experimentos.

- **Sensores de Ultrasonido:** Debido al tamaño reducido del laberinto, en especial en relación al tamaño del vehículo, este debe ser capaz de realizar correcciones de manera rápida. Sin embargo, el realizar una actuación constante de los sensores de ultrasonido, genera interferencias entre estos y medidas erróneas, por lo cual, la lógica interpretará una posición diferente y efectuará un movimiento también incorrecto. Cabe aclarar que dichas mediciones ocurren de manera casi aleatoria.

- **Motores DC:** El principal problema encontrado respecto al comportamiento de los motores es de su variabilidad, en donde para una misma señal PWM (que controla su velocidad) puede generar un giro más o menos rápido de forma inconsistente. Además, el peso considerable que tienen los elementos genera que los motores requieran una velocidad mayor, de forma que se saturan mucho más rápido.

- **Posición de las minas:** A partir de lo planteado con los puntos anteriores, se extrae el poco control que existe respecto a la posición resultante del vehículo al momento de una intersección, lo que conlleva a un mal encuadre de la figura por parte de la cámara. Esto genera también problemas en el reconocimiento, en cuanto a que la cámara no solo puede perder contraste (al perder el fondo de la pared) sino que puede directamente no observar una mina. Para sobrellevar esto se buscó colocar múltiples minas en el mismo giro y así tratar de corregir lo máximo posible este problema.

- **Iluminación:** Dadas las condiciones tan específicas que requiere la cámara para la visualización, un horario nocturno desfavorecerá el como se capten las imágenes, por lo que es necesario el apoyo con una fuente de luz externa.

A partir de estas observaciones se consideran las siguientes soluciones para una posterior rectificación:

- Emplear sensores de proximidad más fiables, preferiblemente a partir de láser u ópticos, para mejorar la fiabilidad de la medida y la velocidad.

- El uso de teoría de control permitiría una mejor aproximación al manejo de los motores, por lo que la corrección de trayectoria sería más efectiva.

- Emplear elementos más livianos permitiría una menor velocidad necesaria de movimiento y con ello una saturación más lejana.

Conclusiones y Comentarios

- Se desarrolló un vehículo capaz de recorrer de manera limitada un laberinto y enviar información suficiente para hacer una reconstrucción de su trayectoria y la ubicación de las figuras geométricas dispuestas en algunas esquinas.

- Fue posible desarrollar un sistema que identifica de manera exitosa los colores de las minas que se pongan frente de una cámara OV7670, siempre que estas sean azules, rojas o verdes y se encuentren sobre un fondo blanco. Adicionalmente, el sistema es capaz de decidir si no existe una mina.

- Debido a la baja precisión y fiabilidad de los sensores ultrasónicos utilizados, así como la amplia incertidumbre y variabilidad en el modelo de los motores, la movilidad dentro de un laberinto reducido se vio afectada, llevando a que el vehículo no pudiera navegar a lo largo de muchas intersecciones sin colisionar con las paredes. Para futuras implementaciones se recomienda utilizar sensores de distancia más rápidos y fiables, como lo pueden ser los sensores ópticos.
- Siguiendo el principio de diseño de particionamiento SH-WH, fue posible desarrollar un vehículo de reconocimiento y reconstrucción de laberintos funcional de manera modular.

Referencias

[1] D. Martínez, "Proyecto Final-Electrónica Digital 2", Requerimiento técnico, 2021.

[2] O. Cely, F. González, D. Portela "Roberto2021", *GitHub*, 2021. [En línea]. Disponible en: https://github.com/ocely/Roberto2021 [Accedido: 17-jul-2021].

[3] "OV7670/OV7171 CMOS VGA (640X480) CAMERACHIP™ with OmniPixel® Technology", OmniVision Technologies Inc., Datasheet Preeliminar. jul. 2005.

[4] "Serial Camera Control Bus Functional Specification", OmniVision Technologies Inc., Nota de Aplicación. 101, mar. 2002.

[5] "OV7670/OV7171 CMOS VGA (640X480) CameraChip™ Implementation Guide", OmniVision Technologies Inc., Nota de Aplicación, sep. 2005.

[6] M. Yaseen, "Arduino SCCB Driver for OV7670 Camera Module", *GitHub*, 2016. [En línea]. Disponible en: https://gist.github.com/muhammadyaseen/75490348a4644dcbc70f [Accedido: 30-may-2021].

[7] NANDLAND. UART, Serial Port, RS-232 Interface [En línea]. Disponible en https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html

[8] L. Llamas. Medir distancia con Arduino y sensor HC-SR04 [En línea]. Disponible en: https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/

[9] Naylamp Mechatronics. Sensor Ultrasonido HC-SR04 [En línea]. Disponible en: https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html