ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

PROYECTO SISTEMAS EMBEBIDOS

Fecha: 23/08/2021

Adrian Steven Narea Jerez
asnarea@espol.edu.ec
Oscar Andres Guapi Mora
oguapi@espol.edu.ec

Table of Contents

Introducción

La seguridad en el área del parqueo en los centros comerciales es muy importante porque las personas ingresan a las instalaciones dejando sus automóviles solos en el parqueadero, muchas veces los usuarios dejan pertenencias de gran valor dentro de sus autos por lo que se debe garantizar el cuidado de su automóvil y lo que se encuentre dentro de este.

Un gran problema sería que exista un robo de vehículo hacia alguno de los usuarios, es por esto que una de las alternativas sería generar un método eficaz que luche contra esta problemática. Este método trata de que en el momento en el que el usuario entre al centro comercial y pase por la garita o entrada, se genere un código QR único que almacene la información del usuario y la placa del automóvil y se imprima un papel con dicho código para entregárselo al usuario. De esta manera al momento que el usuario salga del establecimiento tenga que mostrar el papel con el código QR, luego se procederá a verificar si coinciden los datos con la placa y si esto se cumple, se le permite la salida al usuario.

Si el código no coincide con los datos del usuario se podría tratar de un robo, se emitirá una advertencia y se detendrá al usuario para proceder a investigar las razones por las que no coinciden los datos.

Para nuestro proyecto implementaremos el uso del microprocesador Raspberry pi, la cámara de nuestra computadora, el programa Proteus y ciertas librerías específicas para desarrollar las funcionalidades necesarias para tener éxito con el proyecto.

Nuestra cámara trabajará como la cámara de la garita o entrada al parqueadero del centro comercial. Iniciaremos el programa utilizando la librería “OpenCV” y nuestra cámara se encenderá, tendremos a la mano imágenes de placas de carros que nuestro programa leerá para luego guardarlo en el código QR que se generará mediante la librería “qrcode” y se guardará en nuestra carpeta del proyecto. La acción de “Entrega del papel con el código QR generado” lo trabajaremos de la siguiente manera. Utilizaremos la librería “smtplib” con lenguaje en Python para enviar el código QR con la placa del usuario al correo

electrónico del profesor encargado o a alguno de los integrantes del grupo encargados de presentar el proyecto.

En nuestro simulador Proteus luego de haber generado el código QR utilizaremos un motor que en la vida real sería el encargado de permitirle la entrada al usuario al centro comercial.

Ahora, cuando el usuario quiera dejar el centro comercial se activará nuevamente la cámara y una vez que tengamos el código QR, que anteriormente fue enviado a nuestro correo electrónico, lo mostramos a la cámara. Si el código QR coincide con la placa almacenada se encenderá un led verde y nuestro motor se activará, pero si no coinciden, se encenderá un led rojo y se emitirá una señal de advertencia.

Objetivos Generales

- Simular de manera eficaz el proceso de ingreso del usuario a un centro comercial y la entrega del ticket con el código QR generado para mayor seguridad.
- Utilizar las librerías adecuadas para el trabajo que se realizara dentro de nuestro procesador por medio del programa Proteus.
- Verificar que los datos guardados en nuestra base de datos son los correctos para evitar problemas internos en nuestra simulación.

Objetivos Específicos

- Utilizar el microprocesador Raspberry Pi para el manejo de cada paso de programación y simulación de nuestro proyecto.
- Escoger los elementos adecuados que manejaremos dentro de Proteus que trabajarían de forma eficaz tanto en nuestra simulación como en la vida real.
- Administrar cada bloque de código de programación que cumplen una sola función para que en nuestro “main” solo llamemos estas funciones y asi simplificar el código presentado en el “main”.

Problemática

Nuestra problemática en la vida real trata sobre la seguridad de los autos al momento de ingresar y salir al parqueadero de los centros comerciales o incluso podríamos nombrar las urbanizaciones. En algunos lugares no se toma tanto en cuenta quien entra al establecimiento o urbanización y esto podría generar muchos problemas porque los miembros de seguridad que se encuentran en la garita no están al tanto de la situación dentro del establecimiento en caso de que ocurra algún problema con estas personas que hayan ingresado. Por eso es muy importante tener un informe de toda persona que ingresa a estos lugares, sea una urbanización o un centro comercial.

Otro problema puede ser que a nuestros usuarios que entran con sus automóviles a los lugares ya nombrados anteriormente reciban un robo de sus pertenencias o incluso en las peores situaciones, sea su auto el que pueda haber sido robado. Es por esto que con el informe de ingreso se verifique con todos los usuarios que vayan a salir de los centros comerciales para así evitar cualquier robo de automóviles.

Campo de aplicación a nivel geográfico en la Región o País

Nuestra solución ya está siendo aplicada en los centros comerciales mas importantes del País. Por ejemplo, en Guayaquil tenemos centros comerciales como el “San Marino”, “Mall del sol”, entre otros. Asi mismo en Quito y Cuenca que son nuestras ciudades mas importantes del país. Pero esta solución no está siendo aplicada totalmente en todos los comerciales del país, por ejemplo, en ciudades pequeñas como Milagro, Yaguachi, entre otras, no existe un correcto manejo de la seguridad de los autos, cualquiera persona puede entrar sin ningún control y asi mismo no existe una verificación de los usuarios al momento que dejan el establecimiento.

Elementos Seleccionados

Raspberry Pi4

Es un microprocesador o un ordenador de placa reducida de bajo costo que nos permite desarrollar proyectos usando lenguajes como Scratch o Python, puede hacer tareas desde navegar en internet o grabar y reproducir videos hasta lectura de señales que ingresan por sus puertos. En las Raspberry encontramos componentes como su chip central que es de silicio y está conformado por la CPU (Unidad central de Procesamiento) y el GPU(Procesador de gráficos), una memoria RAM que depende de la gama, módulos WIFI y Bluetooth, 4 puertos USB, puerto Ethernet, conector HDMI para transmisión de audio y video, 40 pines, micro USB para suministrar energía eléctrica, entre otras cosas (Perez, 2021).

Cámara watherproof TTL

Es una cámara con la capacidad de resistir a la intemperie para evitar danos causado por el clima. Es compacta teniendo su carcasa dimensiones de 2” x 2” x 2.5” y un peso de 150 gramos. Cuenta con 30 M pixeles y un ángulo de visión de 60 grados, además tiene un voltaje de funcionamiento de 5V y 75 mA de corriente de consumo sin el uso de leds infrarrojos.

L298

Este driver es un complemento para las Raspberry pi el cual permite controlar 4 motores DC, los cuales pueden funcionar desde 4.5 V a 13.5V, posee diodos internos de protección contra golpes y tiene protección de apagado térmico en caso de una sobrecarga.

Motores DC

Es un motor de alta eficiencia con lo cual no aumenta mucho su temperatura al estar en operación, lo cual es vital al estar en constante movimiento. Tiene 200 pasos por cada vuelta, sus rodamientos son de acero endurecido y tiene un par de torsión máximo de 0.55Nm, además tiene una corriente nominal de 1.68 A/Fase y voltaje de 3.75V.

Batería (12V)

Es una batería resistente a impactos y serviría en caso de haber perdido el suministro de energía.

Encapsulado de Plástico

Este es un protector para los componentes del sistema con IP67 para asegurarlo de agentes externos como el polvo o el agua que pueden ser traídos por los vehículos o clima adverso, también protege de los contaminantes que emiten los vehículos.

Descripción de tecnologías

Procesamiento digital de imágenes

El procesamiento de imágenes se remonta a los años 60, pero como esta necesita altos recursos computacionales su desarrollo iba de la mano con el avance de las tecnologías de las computadoras. Las metas del procesamiento de imágenes es que de una imagen se extraiga información que permita entender su contenido, mejorar la calidad de la misma, o conseguir otra imagen que ocupe menos memoria sin perder la calidad (Veintimilla Portilla & Siguencia Carrillo, 2014). El procesamiento de imágenes hace transformaciones a nivel de pixeles o grupos de estos para mejorar las imágenes, se utilizan operaciones como corrección, reducción de ruido, rotación y balance de color. Para ciertas personas el procesamiento de imágenes tiene un sector de investigación a parte de la visión por computadora, pero muchas técnicas de esta son aplicadas en la otra para alcanzar los objetivos planteados. (Miranda, 2018)

Visión por computadora

La visión por computadora cumple un amplio rango de temas, y sus resultados pueden ser usados para estimación de movimiento, seguimiento de objetos o reconstrucción de modelos 3D, permitiéndonos así describir el mundo que nos rodea reconstruyendo sus propiedades a partir de información analizada proveniente de las imágenes (Szeliski, 2010). Para nosotros el reconocimiento de formas, colores y propiedades de algo son cosas relativamente fáciles de hacer, pero implementar esto en sistemas automatizados representa un gran desafío por la información insuficiente que tenemos. A pesar de esto es

un área en constante desarrollo. Según la mayoría de los trabajos el proceso de un sistema de visión por computador es el siguiente:

- Adquisición de imágenes, como una foto digital de las imágenes.
- Procesamiento digital de las imágenes, donde se aplican diversos métodos para que la imagen se vea mejor.
- Segmentación, nos enfocamos en el área de nuestro interés.
- Extracción de características, nos enfocamos en rasgos que nos puedan ayudar a reconocer ciertos elementos del entorno.
- Reconocimiento, fase donde identificamos los objetos distinguiendo unos de otros.

Con este proceso podremos distinguir un objeto a partir de sus características, toda esta información adquirida se podrá usar para cumplir diversos objetivos dependiendo de la finalidad de cada sistema de visión por computador. (Miranda, 2018)

Reconocimiento óptico de caracteres (OCR)

El OCR por las siglas en ingles de Optical Character Recognition es una tecnología la cual nos permite extraer datos como caracteres a partir de archivos PDF o imágenes, y los cuales podremos acceder y editarlos. Esta tecnología funciona buscando formas parecidas a los distintos caracteres mediante la verificación de la imagen pixel por pixel. Los sistemas OCR tienen 4 etapas distintivas las cuales son: adecuación de la imagen, selección de la zona de interés, extraer características de la imagen y reconocimiento del carácter contenido. (Álvarez Durán, 2014)

Soluciones similares

En un trabajo realizado hace una década, Guzmán, Medina & Gualdron (2011) se abordó una solución basada en el procesamiento de una imagen digital para ayudar en la gestión de la congestión vehicular que se encuentra en constante crecimiento. Los autores desarrollaron la investigación para aplicarla específicamente al control de acceso a un parqueadero, entrada la cual tenía luz natural, distancia de los autos controlada y posición de la cámara apropia. En su solución utilizaron dos métodos se segmentación para enfocarse en la zona de la placa, uno era segmentación por detección de bordes y el otro por detección de colores. El trabajo concluyo que los resultados obtenidos fueron satisfactorios con bajo porcentaje de error y muy cercanos a la realidad, además que la etapa del proceso más vulnerable es la de segmentación de caracteres por su influencia de las anomalías físicas presentes en las placas. (Guzman Castillo, Medina Villalobos, & Gualdron Gonzalez, 2011)

Otro trabajo realizado en la ciudad de Cuenca por Mayra Álvarez (2014) donde se desarrolló un sistema de reconocimiento de placas vehiculares el cual se encuentra a la entrada de un parqueadero en una universidad en Cuenca. En este trabajo se compararon distintas librerías para la visión por computadora como OpenCV, LTI y VXL, mediante el tiempo de procesamiento que les tomaban realizar cierta tarea, donde se evidencio que la librería que presentaba el menor tiempo de procesamiento fue OpenCV. El trabajo concluyo que ante la problemática de que ciertas letras el sistema los reconocía como números y viceversa una solución es restringir a que los primeros 3 caracteres sean letras y los demás sean números, esto se logra reemplazando caracteres de forma similar como el 8 con B en caso de estar en zona de letras o la I con el 1 en caso de estar en zona de números. (Álvarez Durán, 2014)

Análisis de nuestra solución

En base a la documentación revisada se estableció que para nuestro proyecto donde debemos realizar visión por computador la mejor opción es utilizar la librería OpenCV por sus bajos tiempos de

procesamiento, además que posee una amplia documentación y ya ha sido adaptada al lenguaje de programación de nuestro interés mediante OpenCV-Python que es la API de Python para OpenCV. También se hará uso del motor de reconocimiento para caracteres tesseract mediante pytesseract por su fácil empleo y poder así reconocer los caracteres de la placa después de haber realizado la segmentación de la zona de la placa en la imagen mediante la segmentación por detección de bordes y filtrando bordes rectangulares a partir de determinada área. Se hará uso de la Raspberry Pi4 ya que este microprocesador cuenta con la capacidad necesaria para realizar el procesamiento de imágenes, además cuenta con los pines suficientes para realizar la captura de la imagen y enviar la señal para el accionamiento de los motores. Para posteriormente verificar posteriormente el dueño del vehículo se le entregara un código QR que tiene datos extraídos de la captura de imagen de la placa, estos datos serán verificados a la salida del parqueo con la lectura del código y una nueva captura del vehículo al salir, los cuales sí coinciden se le enviara la señal al motor para permitirles salir. Esto evitaría que personas que no son los dueños del vehículo fuercen el encendido del mismo y traten de salir del parqueadero para darse a la fuga.

Diagrama Esquemático

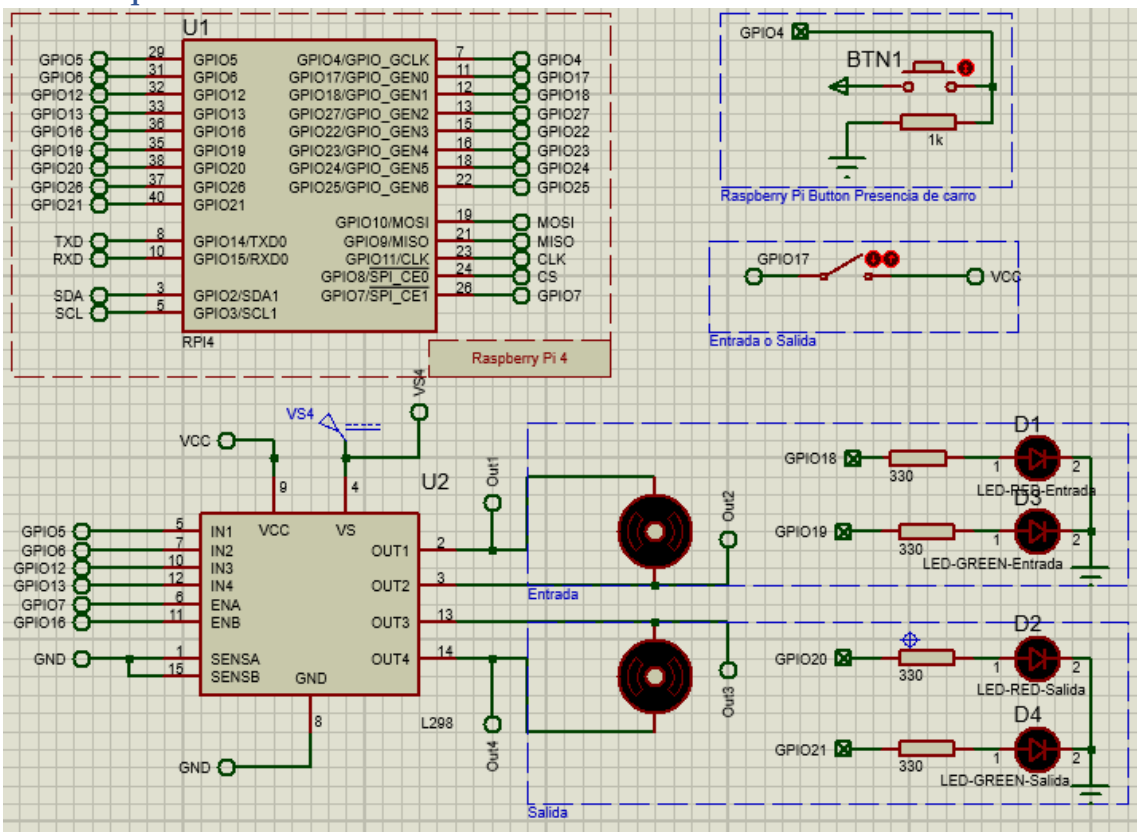

Ilustración 1: Diagrama Esquemático.

Implementación en PCB

A partir del diagrama esquemático realizo una vez que se definió los elementos que tendría nuestra solución, se procedió a realizar la implementación final en PCB, donde se realizó el siguiente diseño en PCB Layoud herramienta con la cual se cuenta en Proteus.

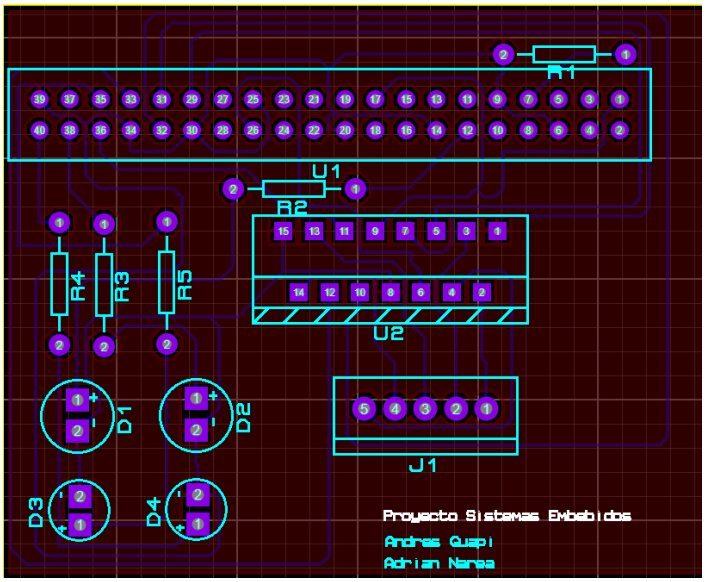

Ilustración 2: Diseño en PCB del proyecto.

El cual cuenta con conexiones para los motores y la alimentación de los mismo. Para una mejor visualización a continuación se presenta la vista 3D del diseño.

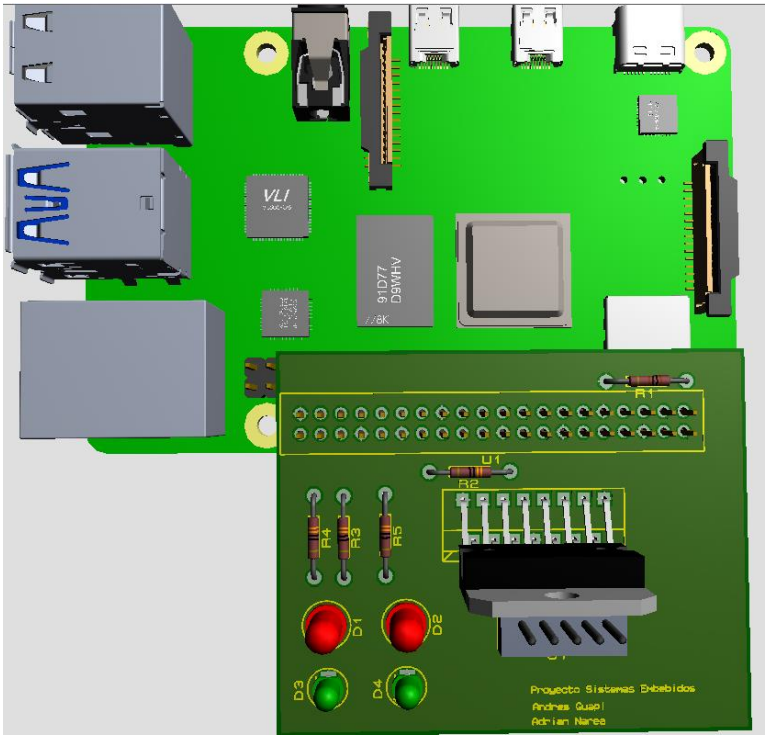

Ilustración 3: Visualización 3D del diseño PCB.

Como vemos en la vista 3D en la parte superior de la PCB tiene la superficie un poco sombreada, esto porque se definió que esa área sea GND para que pueda disipar calor, las conexiones de los elementos están en la parte inferior lo cual se podrá apreciar en la siguiente figura.

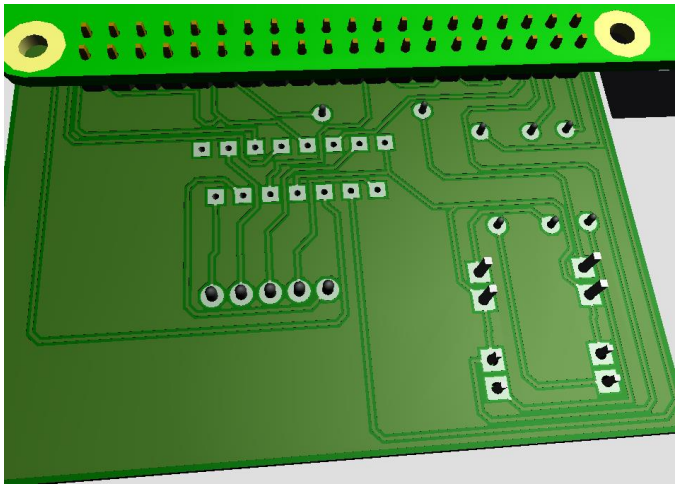

Ilustración 4: Parte inferior de la vista 3D de al PCB.

Simulación

Inicialmente se verificará el estado del switch “Entrada o Salida” el cual si está cerrado quiere decir que el sistema embebido se comportará como si está a la entrada de un parqueadero. A continuación, el sistema espera a que se presione el botón para tomar una foto y guardarla con el nombre “capCarro.jpg”.

Ilustración 5: Captura del carro en la entrada del parqueadero.

Después, se procede a leer la imagen previamente guardada con ayuda de la librería OpenCV para extraer la información. Pero antes se la procesa para mejorar los resultados, como pasarla de RGB a escalas de grises.

Ilustración 6: Procesado de la imagen a escala de grises para mejorar los resultados.

A continuación, se aplica un detector de bordes los cuales se engruesan un poco para a partir de esa imagen extraer todos los contornos y verificar los que tengan 4 vértices (porque la placa es un rectángulo) y el área mayor a 9000 para filtrar más los datos.

Ilustración 7: Se aplica detección de bordes para poder encontrar el área de nuestro interés.

Conociendo más o menos la relación entre el ancho y la altura (aspecto ratio) que debe cumplir la placa, validamos que los contornos recorridos se aproximen a esta relación para establecer el texto de la placa.

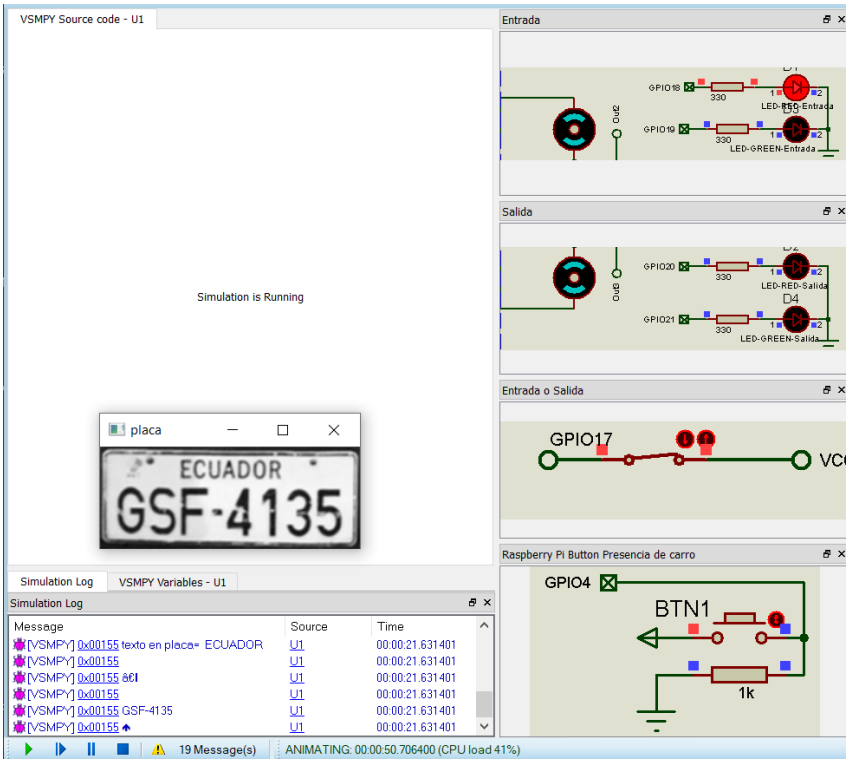

Ilustración 8: Simulación del sistema cuando encontró el área de interés y aplico la búsqueda de caracteres.

Este texto se lo modifica para extraer la parte de interés, a partir de este texto se genera el código QR para enviarlo al usuario a través de correo, por esta razón en el Simulation Log vemos el texto modificado además de los mensajes “Listo para enviar” y “Email enviado” para saber que se logró enviar el email.

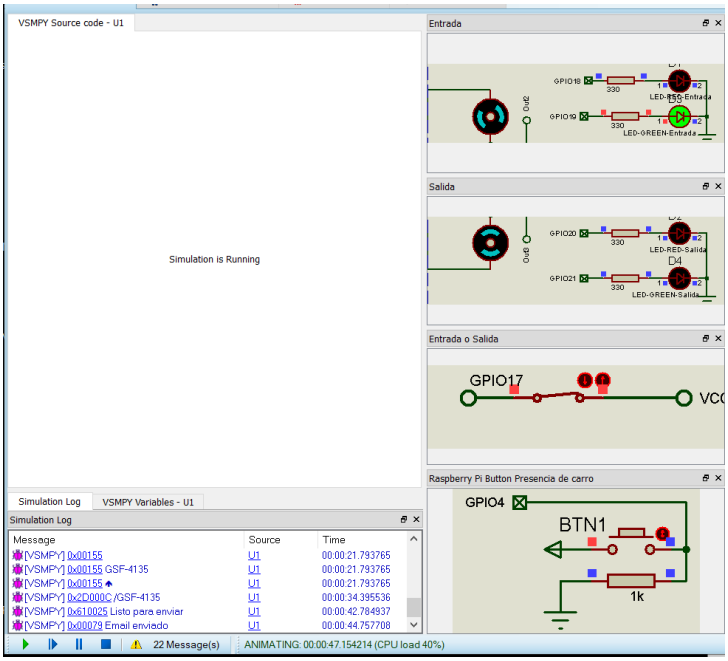

Ilustración 9: simulación del sistema cuando modifico el texto extraído y se quedó con la parte de interés.

También se activó el motor y el led verde que le indicaría al usuario que puede ingresar al parqueadero, pasado los 10 seg el motor se mueve en sentido contrario hacia su posición inicial y se apaga el led verde para que se encienda el rojo.

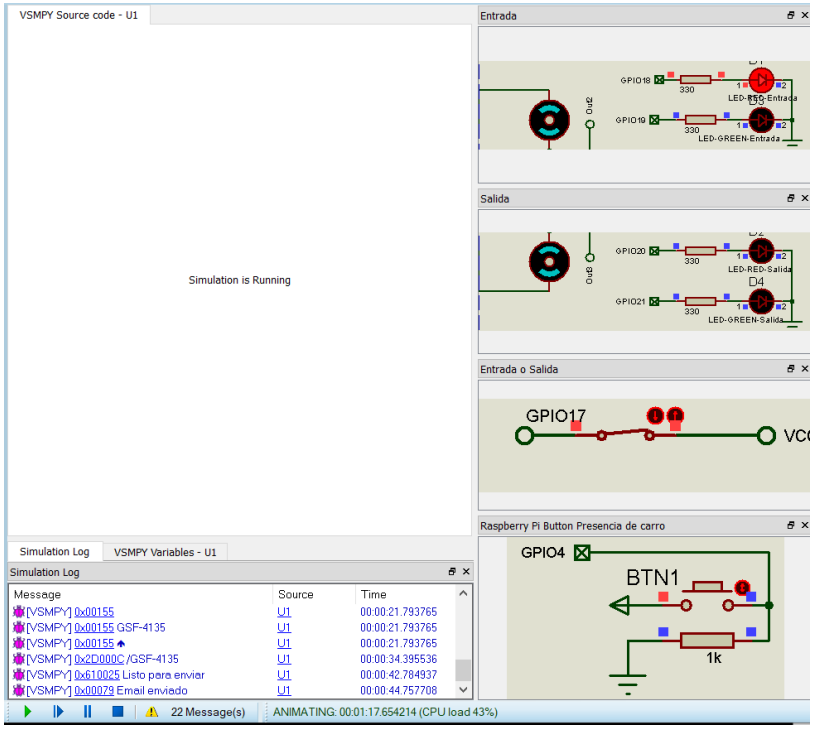

Ilustración 10: simulación del sistema cuando el motor1 volvió a su posición inicial y se vuelve a encender el led rojo.

En el correo tenemos acceso al código QR que fue enviado, y en Firebase tenemos el registro de la placa con True lo cual indica que se encuentra en el parqueadero.

Ilustración 11: Código QR enviado al usuario y registro de la placa en la base de datos.

En el caso que el sistema este en modo salida, inicialmente se procede a hacer lo mismo, que es tomar una captura del vehículo, se extrae la información de la placa y tratar esta información como vemos en Simulator Log en la siguiente imagen. El sistema se pondrá a la espera de escanear el código QR.

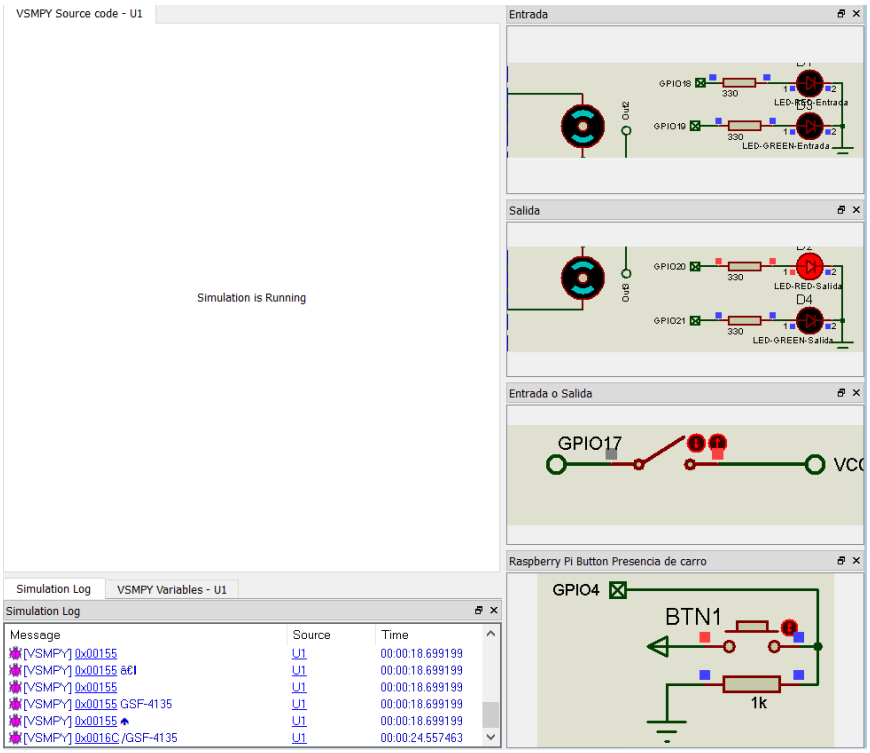

Ilustración 12: Simulación del sistema en la salida del parqueadero.

Se procede a escanear el código QR para verificar si el ticket coincide con la placa al momento de salir, si se cumple en el "Simulation Log" se imprime "Son iguales" y se enciende tanto el motor 2 como el led verde2, además se verifico que la placa se encuentre en la base de datos.

Ilustración 13: Captura del Código QR que se envió al usuario.

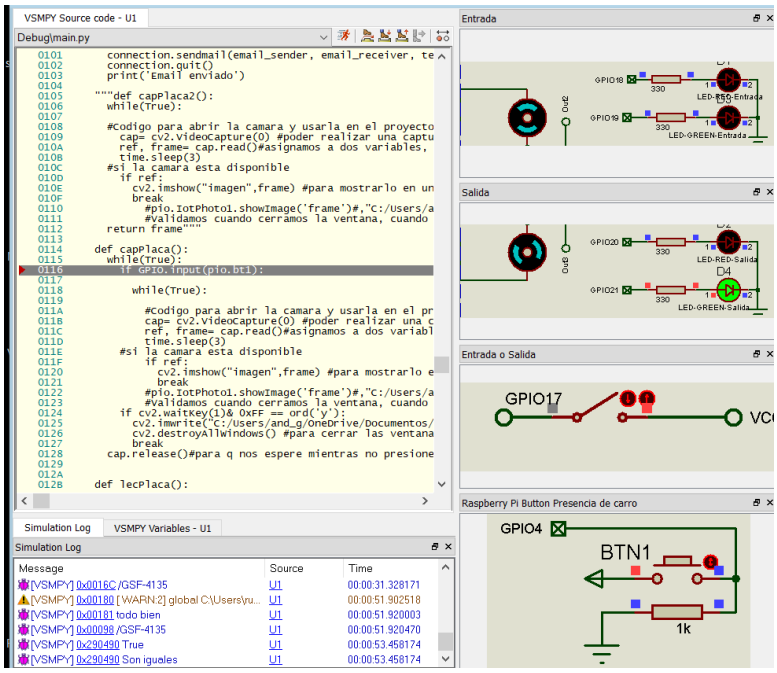

Ilustración 14: Simulación cuando el texto de la placa y del código QR enviado coinciden.

Pasado los 10 seg el motor vuelve a la normalidad y se vuelve a encender solo el led rojo y finalmente el sistema se pone a la espera del siguiente usuario.

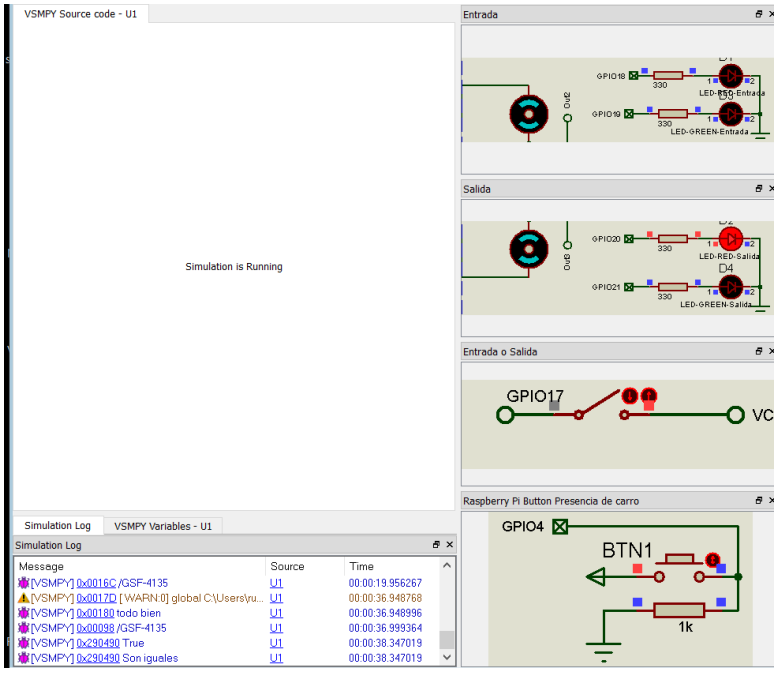

Ilustración 15: Simulación cuando el motor 2 vuelve a su posición inicial y se vuelven a prender el led rojo2.

En caso de no coincidir la placa con el ticket se lanza una alerta mediante el parpadeo del led rojo cada medio segundo.

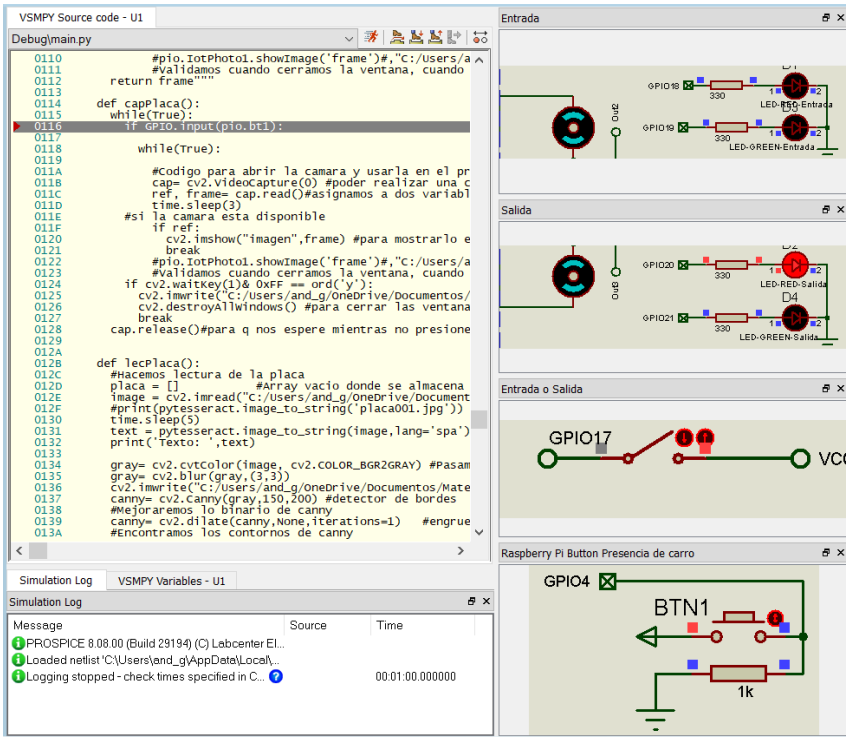

Ilustración 16: Simulación de la alerta al no coincidir los textos del Código QR y de la placa.

Análisis de Resultados

En base a los resultados obtenidos por la simulación podemos decir que nuestro sistema embebido cumple con el registro de la placa que tienen los vehículos que desean entrar al parqueadero, esto mediante la adquisición de una imagen la cual será procesada con ayuda de la librería OpenCV y así extraer la parte de nuestro interés como podemos apreciar en la Ilustración 8 para posteriormente aplicar el motor de búsqueda de caracteres Tesseract, el resultado también lo podemos apreciar en la Ilustración 8 en la parte de "Simulation Log". Como los caracteres no todos los caracteres extraídos de la placa son de nuestro interés como por ejemplo caracteres que pueden salir por un área sucia de la placa se modifico el texto buscando el "-" que presentan todas las placas mediante la función find() de Python y cortando la cadena de caracteres desde 3 puestos antes hasta el final para obtener la parte de nuestro interés como podemos apreciar en la antepenúltima línea del "Simulation Log" de la Ilustración 9. A partir del texto se pudo generar el Código QR el cual se hizo de manera rápida y sencilla usando la función make() de la librería qrcode. En la Ilustración 13 a pesar de tener condiciones de poca luz se obtuvieron resultados satisfactorios al hacer la lectura del código para comprobar la placa del usuario, esto tal vez porque el código cuenta con áreas bien delimitadas. Finalmente, con el uso de interrupciones por tiempo se pudo enviar señales a

los motores para que se muevan y pasado un tiempo vuelvan a su posición original con lo cual se simula el movimiento de la barra para la entrada del vehículo del usuario y posteriormente la vuelta a su lugar.

Conclusiones y Recomendaciones

Conclusiones

Recomendaciones

d

Bibliografía

Álvarez Durán, M. (2014). *Analisis, diseno e implementacion de un sistema de control de ingreso de vehiculos basado en vision artificial y reconocimiento de placas en el parqueadero de la Universidad Politecnica Salesiana-Sede Cuenca.* Cuenca: Universidad Politecnica Salesiana-Sede Cuenca (Bachelor's thesis).

Guzman Castillo, P., Medina Villalobos, J., & Gualdron Gonzalez, O. (2011). Reconocimiento automatico de placas de automoviles: automatizacion de parqueaderos. *Energia y Computacion, V XL*.

Miranda, J. C. (2018). *Clasificacion Automatica de naranjas por tamano y por defectos utilizando tecnicas de vision por computadora.* San Lorenzo: Universidad Nacionall de Asuncion.

Perez, I. (2021). Raspberry Pi. *Vida Cientifica, 9*(17), 40-41.

Szeliski, R. (2010). *Computer vision: algorithms and applications.* Springer Science & Business Media.

Veintimilla Portilla, D., & Siguencia Carrillo, Y. (2014). *Diseno de un sistema inteligente de parqueo vehicular mediante videograbacion e implementacion de un prototipo de prueba para la FIEE.* Quito: Bachelor's thesis, Escuela Politecnica Nacional.

Anexos

Codigo de Programacion en Python

main.py | codigo.txt

```python
#Importamos las librerias necesarias
import cv2
import pytesseract
import camera
import time
import pio
import resource
import Controls
import RPi.GPIO as GPIO
import qrcode
import threading #para las interrupciones
from wiringpi import Serial
#Para correo
import smtplib
#Para firebase
from firebase import firebase

from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email import encoders
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract'
#Creamos la variables de firebase (creo instancia)
firebase = firebase.FirebaseApplication('https://psistemasembebidos-default-rtdb.firebaseio.com/', None)
```

```python
def peripheral_setup () :
# Peripheral Constructors, los que se van a ejecutar una unica vez

  #Seteamos la forma de manejar los pines de la raspberry tal cual salen en los terminal labels
  GPIO.setmode(GPIO.BCM)
  pio.bt1= 4                          #poniendo el boton como global con el pio para llamarlo en otras funciones
  pio.modo= 17                        #Si es 1 es Entrada, si es 0 es Salida
  pio.M1P1= 5      #Para el giro del motor 1, si P1 es False y P2 es True el motor va en sentido antihorario
                   #en cambio si es P1 True y P2 False el motor va en sentido horario
  pio.M1P2=6
  pio.M1PWM=7                         #velocidad para el motor1 (entrada)
  pio.ledRojo1= 18                    #led rojo en la entrada
  pio.ledVerde1=19                    #led verde entrada

  pio.M2P1= 12     #Para el giro del motor 2, si P1 es False y P2 es True el motor va en sentido antihorario
                   #en cambio si es P1 True y P2 False el motor va en sentido horario
  pio.M2P2=13
  pio.M2PWM=16                        #velocidad para el motor2 (salida)
  pio.ledRojo2= 20
  pio.ledVerde2=21
  alerta= False                       #variables para entrar a la alerta

  GPIO.setup(pio.bt1,GPIO.IN, GPIO.PUD_DOWN)#configuramos el boton como entrada y con la conexion de pull do
  GPIO.setup(pio.modo, GPIO.IN)  #Configuramos el modo como entrada
  GPIO.setup(pio.M1PWM, GPIO.OUT)
  GPIO.setup(pio.M1P1, GPIO.OUT)
  GPIO.setup(pio.M1P2, GPIO.OUT)
  GPIO.setup(pio.M2PWM, GPIO.OUT)
  GPIO.setup(pio.M2P1, GPIO.OUT)
  GPIO.setup(pio.M2P2, GPIO.OUT)
  GPIO.setup(pio.ledRojo1, GPIO.OUT)
  GPIO.setup(pio.ledVerde1, GPIO.OUT)
  GPIO.setup(pio.ledRojo2, GPIO.OUT)
  GPIO.setup(pio.ledVerde2, GPIO.OUT)
  #Creamos las instancias de PWM (canal,frecuencia)
  pio.servo1= GPIO.PWM(pio.M1PWM,50)
  pio.servo2= GPIO.PWM(pio.M2PWM,50)
  pio.servo1.start(0)
  pio.servo2.start(0)
```

```
def peripheral_loop () :
  #Si el Switch esta en alto entonces es la entrada
  if GPIO.input(pio.modo):
    capPlaca()
    GPIO.output(pio.ledRojo1,True)       #Encendemos el led rojo de la entrada
    texto= lecPlaca()
    #Extraccion de la parte de interes
    texte= texto.find("-")  #para determinar la ubicacion de la parte de interes se busca el '-'(LLL-1111), empieza en 0 a contar
    texto1= texto[texte-3:]              #cortamos la parte de interes
    texto1= '/' + texto1.strip()         #concatenamos y borramos posibles espacios existentes
    print(texto1)
    entCarro(texto1)                     #funcion que pone la nueva placa en la base de datos
    generacionQR(texto1)
    envioEmail()
    GPIO.output(pio.ledRojo1,False)      #Apagamos el led rojo
    GPIO.output(pio.ledVerde1,True)      #Encendemos el led verde
    GPIO.output(pio.M1P1,False)
    GPIO.output(pio.M1P2,True)

    pio.servo1.ChangeDutyCycle(10)      # Para cambiar el ciclode trabajo de 0 a 10
    threading.Timer(1.35,int1).start()
```

```
#Modo salida del parqueo
  else:
    capPlaca()                           #capturamos la placa
    global alerta                        #variables para lanzar la alerta
    alerta= False
    GPIO.output(pio.ledRojo2,True)       #prendemos el led rojo de la salida
    textoSalida= lecPlaca()
    #Extraccion de la parte de interes
    texts= textoSalida.find("-")  #para determinar la ubicacion de la parte de interes se busca el '-'(LLL-1111), empieza en 0 a contar
    texto2= textoSalida[texts-3:]        #cortamos la parte de interes
    texto2= '/' + texto2.strip()         #concatenamos y borramos posibles espacios existentes
    print(texto2)
    textQR= lecQR()
    #Si el texto de la placa coincide con el del codigo QR y la placa se encuentra en la base de datos movemos el motor
    #sacarlo de la funcion a ver si funciona
    placaf= firebase.get("/Carro", texto2)  #si la placa no esta ref es iguala None
    print(placaf)
    if((texto2 == textQR) and (placaf != None)):
      print('Son iguales')
      GPIO.output(pio.ledRojo2,False)
      GPIO.output(pio.ledVerde2,True)
      GPIO.output(pio.M2P1,False)
      GPIO.output(pio.M2P2,True)
      firebase.put("/Carro", texto2, False) #Actualizamos la base de datos
      pio.servo2.ChangeDutyCycle(10)     # Para cambiar el ciclo de trabajo de 0 a 10
      threading.Timer(1.35,int2).start()
    else:
      alerta= True
      threading.Timer(0.5,alerta1).start()
      print('Entramos a la alerta1')

# Main function
def main () :
# Setup
 peripheral_setup()                      #llamamos a los perifericos
# Infinite loop
 while 1 :
  peripheral_loop()
  pass
# Command line execution
if __name__ == '__main__' :
  main()
```

```
#Funciones para las interrupciones por tiempo
def int1():
  pio.servo1.ChangeDutyCycle(0)
  if(GPIO.input(pio.M1P1)==False and GPIO.input(pio.M1P2)==True):
    #Invertimos el giro
    GPIO.output(pio.M1P1,True)
    GPIO.output(pio.M1P2,False)
    threading.Timer(10,int11).start()        #q se active solo al inicio
  else:
    GPIO.output(pio.ledVerde1,False)    #Apagamos el led verde
    GPIO.output(pio.ledRojo1,True)

def int11():
  pio.servo1.ChangeDutyCycle(10)
  threading.Timer(1.5,int1).start()

def int2():
  pio.servo2.ChangeDutyCycle(0)
  if((GPIO.input(pio.M2P1))==False and (GPIO.input(pio.M2P2))==True):
    #Invertimos el giro
    GPIO.output(pio.M2P1,True)
    GPIO.output(pio.M2P2,False)
    threading.Timer(10,int21).start()        #q se active solo al inicio
  else:
    GPIO.output(pio.ledVerde2,False)    #Apagamos el led verde
    GPIO.output(pio.ledRojo2,True)

def int21():
  pio.servo2.ChangeDutyCycle(10)
  threading.Timer(1.5,int2).start()

def alerta1():
  GPIO.output(pio.ledRojo2,False)         #apagamos el led rojo de la salida
  #Si se pulsa el boton bt1 salimos de la alerta
  if(alerta):
    threading.Timer(0.5,alerta12).start()
  else:
    #Si alerta esta en False, salimos del threading alerta
    GPIO.output(pio.ledRojo2,True)

def alerta12():
  GPIO.output(pio.ledRojo2,True)         #Prendemos el led verde
  #Si se pulsa el boton bt1 salimos de la alerta
  threading.Timer(0.5,alerta1).start()

#Funcion para el envio del codigo
def envioEmail():
  print('Listo para enviar')
  email_sender= 'and5gua97@gmail.com' #correo desde donde se envia
  email_receiver= 'andres.guapi22@gmail.com' #correo a quien se le envia
  subject= 'Proyecto Embebidos'
  msg= MIMEMultipart()
  msg['From']= email_sender
  msg['To']= email_receiver
  msg['Subject']= subject
  body= 'Hola, este correo fue enviado desde python, prueba codigo QR'
  msg.attach(MIMEText(body, 'plain'))
  #Parte del archivo
  filename= 'C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/codigoQR.jpg'
  attachment= open(filename, 'rb')
  part= MIMEBase('application', 'octet_stream')
  part.set_payload((attachment).read())
  encoders.encode_base64(part)
  part.add_header('Content-Disposition', 'attachment; filename= '+filename)
  msg.attach(part)

  text= msg.as_string()
  connection= smtplib.SMTP('smtp.gmail.com', 587)
  connection.starttls()
  connection.login(email_sender, 'sistemasEmbebidosP101')
  connection.sendmail(email_sender, email_receiver, text)
  connection.quit()
  print('Email enviado')

def capPlaca():
  while(True):
    if GPIO.input(pio.bt1):

      while(True):
        #Codigo para abrir la camara y usarla en el proyecto
        cap= cv2.VideoCapture(0)                    #poder realizar una capturas de un video en tiempo real
        ref, frame= cap.read()#asignamos a dos variables, uno es verdad si la camara esta disponible y la otra es la captura
        time.sleep(3)
        #si la camara esta disponible
        if ref:
          cv2.imshow("imagen",frame)            #para mostrarlo en una ventana nueva
          break
        #Validamos cuando cerramos la ventana, cuando el usuario digita "y" se guarda la imagen
    if cv2.waitKey(1)& 0xFF == ord('y'):
      cv2.imwrite("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/capCarro.jpg",frame)
      cv2.destroyAllWindows() #para cerrar las ventanas
      break
  cap.release()                                     #para q nos espere mientras no presione,finalizamos la camara
```

```
def lecPlaca():
  #Hacemos lectura de la placa
  placa = []                                          #Array vacio donde se almacena la placa detectada
  image = cv2.imread("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/capCarro.jpg")
  time.sleep(5)
  text = pytesseract.image_to_string(image,lang='spa')#leemos los caracteres poniendo la variable donde esta la imagen
  print('Texto: ',text)

  gray= cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #Pasamos de BGR a escalas de grises
  gray= cv2.blur(gray,(3,3))
  cv2.imwrite("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/placagray.jpg",gray)
  canny= cv2.Canny(gray,150,200)                      #detector de bordes
  #Mejoraremos lo binario de canny
  canny= cv2.dilate(canny,None,iterations=1)          #engruesa las areas blancas
  #Encontramos los contornos de canny
  cnts,_ = cv2.findContours(canny,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

  #Placas ecuador ancho=404mm,alto=154mm
  #aspect ratio placa = ancho/alto= 404/154= 2,62
  #Desechamos los contornos no deseados, nos basamos en su area y tratamos de encontrar la forma rectangular de la placa
  for c in cnts:
    area= cv2.contourArea(c)                          #sabremos el area de un contorno
    x,y,w,h= cv2.boundingRect(c)          #nos ayuda a detectar un rectangulo en la imagen,nos ayuda con el aspect ratio del contorno
    epsilon= 0.09*cv2.arcLength(c,True) #parametro necesario para aproxPoly, 9% se determino despues de experimentacion
    approx= cv2.approxPolyDP(c,epsilon,True)      #para determinar los vertices del contorno

    if (len(approx)==4 and area >9000):
      print("Area: ",area)
      #solo se muestan los contornos mayores a 9000 y los contornos con 4 vertices
      aspect_ratio= float(w)/h                        #Agregamos el aspect ratio para ser mas precisos
      #2.4 porque anadimos un margen de error
      if aspect_ratio>2.1:
        placa= gray[y:y+h,x:x+w]
        texto= pytesseract.image_to_string(placa, config='--psm 11')#extraemos el texto especificando el modo de segmentacion de pagina

        print('texto en placa= ',texto)
        cv2.imshow('placa',placa) #mostramos
        cv2.moveWindow('placa',780,10) #movemos la imagen a cierta posicion
        cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),3)
        cv2.putText(image,texto,(x-20,y-10),1,2.2,(0,255,0),2)
  if(texto == None):
    texto= text                                       #Decimos que el texto a exportar sea el leido inicialmente para evitar errores por valor nulo
  cv2.imwrite("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/placacanny.jpg",canny)
  cv2.imwrite("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/placaimage.jpg",image)
  cv2.imshow('Imagen',image)                          #para visualizar
  cv2.imshow('Canny',canny)                           #visualizamos despues de detectar bordes
  cv2.moveWindow('Image',45,10)
  cv2.waitKey(0)                                      #el proceso siga hasta presionar alguna letra
  cv2.destroyAllWindows()
  return texto

#Generamos el codigo QR
def generacionQR(texto):
  images= qrcode.make(texto)
  #guardamos en una carpeta nuestro codigo de la placa
  images.save('C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/codigoQR.jpg')

#Leemos el codigo QR
def lecQR():
  while(True):
    if GPIO.input(pio.bt1):
      while(True):
        #Codigo para abrir la camara y usarla en el proyecto
        cap= cv2.VideoCapture(0)                      #poder realizar una capturas de un video en tiempo real
        ref, frame= cap.read()#asignamos a dos variables, uno es verdad si la camara esta disponible y la otra es la captura
        time.sleep(3)
        #si la camara esta disponible
        if ref:
          cv2.imshow("imagen",frame) #para mostrarlo en una ventana nueva
          break
    if cv2.waitKey(1)& 0xFF == ord('y'):
      cv2.imwrite("C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/QRsalida.jpg",frame)
      cv2.destroyAllWindows() #para cerrar las ventanas
      break
  cap.release()                                       #para q nos espere mientras no presione,finalizamos la camara
  print("todo bien")
  d= cv2.QRCodeDetector()                             #d es un arreglo de 3 variables
  #usamos una funcion de openCV para leer la imagen previamente creada
  val, points, straingh= d.detectAndDecode(cv2.imread('C:/Users/and_g/OneDrive/Documentos/Materias/Sistemas Embebidos/Practico sist embebidos/Proyecto/Proyectov5/QRsalida.jpg'))#
  print(val)
  return val

def entCarro(texto):
  firebase.put("/Carro", texto, True)
```