UNIVERSIDAD AUTÓNOMA METROPOLITANA

Diplomado Internet de las Cosas de Samsung
Innovation Campus

Proyecto Capstone

SISTEMA DE PREVENCIÓN DE ACCIDENTES PARA VEHÍCULOS PESADOS

Que presenta:
Hernández Moreno Miriam Lizbeth
Sánchez Hernández Luis Ángel
Pérez Murillo José Luis

Asesor(es):
Dra. Vilchis León Paloma Alejandra

Ciudad de México, MEXICO marzo 2021

Contenido

Objetivos

Objetivos generales

- Monitorear con una cámara al operador del vehículo, mediante un detector de somnolencia.
- Monitorear con un sensor ultrasónico la parte frontal del vehículo.

Objetivos específicos

1. Diseñar y construir el sistema de montaje para la cámara y el microcontrolador utilizado en el sistema propuesto.
2. Implementar en un microcontrolador el sistema de control y detector de somnolencia para la interfaz vehículo/conductor.
3. Implementar en un microcontrolador el sistema de control para un sensor ultrasónico.
4. Instalar y configurar un Bróker MQTT accesible desde Internet.
5. Implementar un dashboard que permita la monitorización de "tiempo real" el estado del conductor.
6. Implementar una página web que permita captar video cuando el sensor ultrasónico detecte un nivel corto de proximidad.

Justificación

Como anteriormente se mencionó este proyecto está pensado para disminuir la cantidad de accidentes viales que son causados por vehículos de gran tamaño, centrándose en primer lugar en aquellos que son provocados por el cansancio excesivo en los operadores, debido a las largas distancias que deben recorrer para llegar a sus destinos; como también de los puntos ciegos que este tipo de vehículos poseen.

Es aquí cuando los avances tecnológicos pueden ayudar y mediante el internet de las cosas comunicar el estado actual del conductor y del vehículo hacia una central de monitoreo.

Descripción del proyecto

Ilustración 1 Puntos ciegos del tráiler

Ilustración 2 Modelo tentativo del proyecto

El presente proyecto fue desarrollado en el lenguaje de programación Python y pretende implementarse en vehículos de carga pesada con el fin de alertar a los conductores en caso de altos niveles de cansancio, traducidos en somnolencia, así mismo, eliminar aquellos puntos ciegos que este tipo de vehículos posee como resultado de su gran tamaño.

Consideraciones

Para realizar este proyecto se utilizaron únicamente aquellos dispositivos proporcionados en el Kit de Código IoT por Samsung Innovation Campus, así como, algunos utilizados en la vida cotidiana (bocina y webcam). Por lo que, si bien el proyecto funciona adecuadamente, estamos conscientes que puede ser mejorado haciendo uso de dispositivos especializados para este tipo de tareas, sin embargo, esto significa aumentar el presupuesto y las actualizaciones constantes, por ejemplo, colocar una cámara con visión nocturna que permita al programa de somnolencia operar correctamente con bajos niveles de luz (viajes nocturnos), así mismo, la ESP32CAM cuenta con muy poca calidad de imagen, por lo que puede ser sustituida por una o un conjunto de cámaras IP con mayor calidad.

Desarrollo del proyecto

1.- Se debe armar el siguiente circuito que unirá la ESP32-CAM con el FTDI, para la transmisión del video.

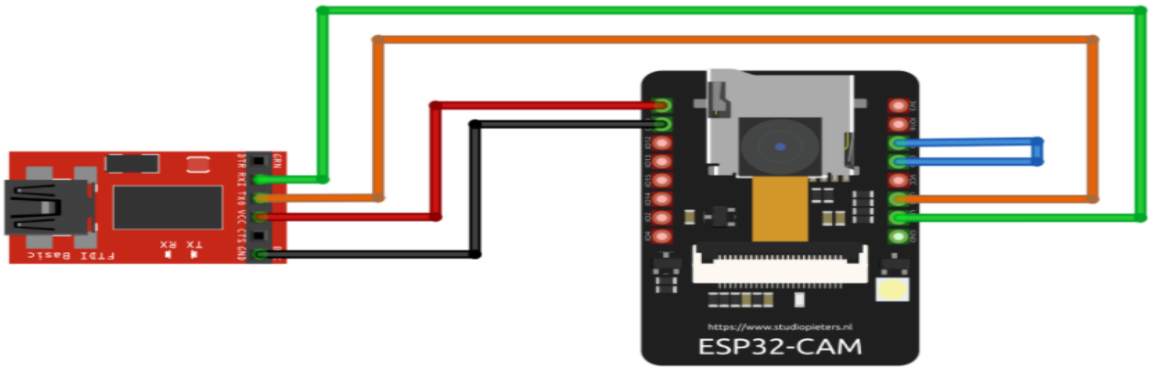

Ilustración 3 Diagrama de conexión esp32-cam con FTDI.

2.- El programa que debe ir cargado a la ESP32-CAM se encuentra en el IDE de arduino, donde, se deben de cambiar el “SSID” y “PASSWORD” que correspondan a la red wifi que se quiera conectar. Cuando se conecte a la red la ESP32 se debe tomar nota de la dirección IP ya que se necesitará.

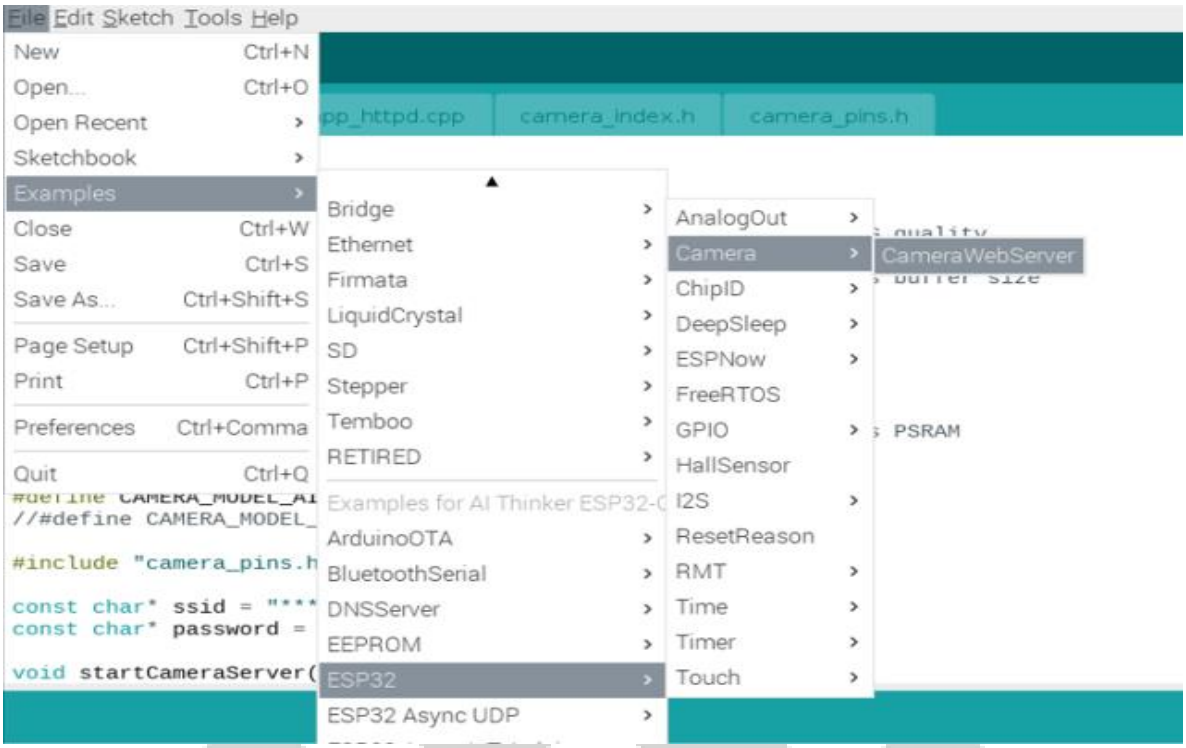

Ilustración 4 Ruta para localizar el programa

3.- Se debe armar el siguiente circuito que incluye al sensor ultrasónico HC-SR04 y además este estará conectado a la Raspberry pi 4.

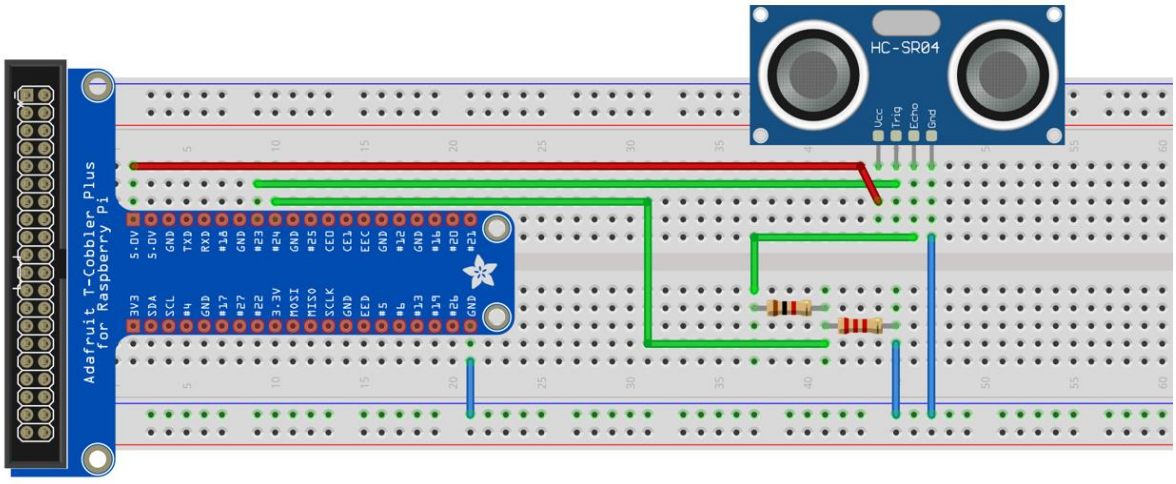

Ilustración 5 Diagrama de conexión del sensor ultrasónico.

4.- Se deben cargar los siguientes programas a la Raspberry pi 4.

El siguiente programa estará a cargo de la medición de la distancia, el cual está programado para operar a 1m.

```
import RPi.GPIO as GPIO
import numpy as np
import time
import cv2

#Configuraciones para leer el sensor ultrasónico
GPIO.setmode(GPIO.BCM)
GPIO_TRIGGER = 23
GPIO_ECHO    = 24
GPIO.setup(GPIO_TRIGGER,GPIO.OUT)
GPIO.setup(GPIO_ECHO,GPIO.IN)
GPIO.output(GPIO_TRIGGER, False)

def CalcDistancia():
        GPIO.output(GPIO_TRIGGER,True)
        time.sleep(0.00001)
        GPIO.output(GPIO_TRIGGER,False)
        start = time.time()
        while GPIO.input(GPIO_ECHO)==0:
                start = time.time()
        while GPIO.input(GPIO_ECHO)==1:
                stop = time.time()
        elapsed = stop-start
        distance = (elapsed * 34300)/2
        return distance

print("Inicia la toma de datos")

try:
        while True:
                print("acerque el objeto para medir la distancia")
                distance=CalcDistancia()
                print( ' distanciacalculada ' + str(distance) )
                time.sleep(2)
                # si se alcanza cierta distancia sonar alarma e iniciar transmicion de la ESP32
                if(distance <= 100): #Un metro de distancia
                #Cambiar por la IP y puerto(81 por default)
                cap = cv2.VideoCapture('http://**.**.**.**:**/stream')
                        print("Hay un objeto demaciado cerca")
                        while(True):
                                ret, frame = cap.read()
                                cv2.imshow('ESP32CAM',frame)
                                distance=CalcDistancia()
                                print( ' entro ' + str(distance) )
                                time.sleep(0.1)
```

```
                    if cv2.waitKey(1) & 0xFF == ord('q') or distance>100:
                        #cv2.destroyWindow('ESP32CAM')
                        cap.release()
                        cv2.destroyAllWindows()
                        break

except KeyboardInterrupt:
        print('\n' + 'termina la captura de datos.' + '\n')
        GPIO.cleanup()
```

El siguiente programa se encargará de monitorear los ojos del conductor, para determinar si existe somnolencia.

```
from paho.mqtt import client as mqtt_client
from scipy.spatial import distance
from imutils import face_utils
from datetime import datetime
import numpy as np
import pygame #For playing sound
import time
import dlib
import cv2
import random
import getpass
import os

#Variables para publicar en MQTT
broker = 'broker.hivemq.com'
port = 1883
topic = "somnolencia/******" # Reemplazar asteriscos por el nombre del conductor
estado = 0 # 0 Despierto - 1 Dormido
#generar ID de cliente con prefijo pub al azar
client_id = f'python-mqtt-{random.randint(0, 100)}'

#Inicializar Pygame y cargar música
pygame.mixer.init()
pygame.mixer.music.load('audio/alarma.mp3')

#Umbral mínimo de la relación de aspecto de los ojos por debajo del cual se activa la alarma
EYE_ASPECT_RATIO_THRESHOLD = 0.3#Fotogramas consecutivos mínimos en los que la proporción de
ojos está por debajo del umbral para que se active la alarma
EYE_ASPECT_RATIO_CONSEC_FRAMES = 50
#Número de recuentos de fotogramas consecutivos por debajo del valor de umbral
COUNTER = 0
#Cargue la cascada de caras que se usará para dibujar un rectángulo alrededor de las caras detectadas
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

#Función para conectar al broker MQTT
def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
```

```
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)
    client = mqtt_client.Client(client_id)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

#Función para publicar en el broker MQTT, se utiliza un topic conocido para poder leerlo por fuera
def publish(client, topic, counter):
    usr_name = getpass.getuser()
    #formato_Mensaje: ' nombre;counter;fecha '
    msg = usr_name + ";"
    msg += str(counter) + ";"
    msg += str(datetime.now())
    print(msg)
    result = client.publish(topic, msg)
    status = result[0]
    if status == 0:
        print(f"Send `{msg}` to topic `{topic}`")
    else:
        print(f"Failed to send message to topic {topic}")
def enviar_mensaje_central(topic, counter):
    client = connect_mqtt()
    client.loop_start()
    publish(client, topic, counter)
    client.loop_stop(True)

#Esta función calcula y devuelve la relación de aspecto de los ojos
def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    ear = (A+B) / (2*C)
    return ear

#Cargue el detector y el predictor de rostros, utiliza el archivo predictor de forma dlib
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('../shape_predictor_68_face_landmarks.dat')

#Extraiga índices de puntos de referencia faciales para el ojo izquierdo y derecho
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS['left_eye']
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS['right_eye']

#Inicie la captura de video de la cámara web
video_capture = cv2.VideoCapture(0)
#Dar algo de tiempo para que la cámara se inicialice
time.sleep(2)
os.system("lxterminal -e 'python3 distancia.py'")
while(True):
    #Leer cada cuadro, gírarlo y convertir a escala de grises
    ret, frame = video_capture.read()
```

```
  frame = cv2.flip(frame,1)
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

  #Detecta puntos faciales a través de la función de detector.
  faces = detector(gray, 0)
  #Detectar rostros a través de haarcascade_frontalface_default.xml
  face_rectangle = face_cascade.detectMultiScale(gray, 1.3, 5)
  # Dibuja un rectángulo alrededor de cada cara detectada
  for (x,y,w,h) in face_rectangle:
     cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
 #Detectar puntos faciales
  for face in faces:

     shape = predictor(gray, face)
     shape = face_utils.shape_to_np(shape)

     #Obtenga una matriz de coordenadas de ojo izquierdo y ojo derecho
     leftEye = shape[lStart:lEnd]
     rightEye = shape[rStart:rEnd]

     #Calcular la relación de aspecto de ambos ojos
     leftEyeAspectRatio = eye_aspect_ratio(leftEye)
     rightEyeAspectRatio = eye_aspect_ratio(rightEye)
     eyeAspectRatio = (leftEyeAspectRatio + rightEyeAspectRatio) / 2

     #Eliminar las discrepancias de contorno convexo y dibuje la forma de los ojos alrededor de los ojos
     leftEyeHull = cv2.convexHull(leftEye)
     rightEyeHull = cv2.convexHull(rightEye)
     cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
     cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

     #Detectar si la relación de aspecto de los ojos es inferior al umbral
     if(eyeAspectRatio < EYE_ASPECT_RATIO_THRESHOLD):
        COUNTER += 1
        #Si el número de fotogramas es mayor que el umbral de fotogramas,
        if COUNTER >= EYE_ASPECT_RATIO_CONSEC_FRAMES:
           estado = 1
           enviar_mensaje_central(topic, estado);
           pygame.mixer.music.play(-1)
           cv2.putText(frame, "Somnolencia, peligro", (150,200), cv2.FONT_HERSHEY_SIMPLEX, 1.5,
(0,0,255), 2)
     else:
        pygame.mixer.music.stop()
        COUNTER = 0
        if(estado == 1):
           estado = 0
           enviar_mensaje_central(topic, estado);

  #Mostrar transmisión de video
  cv2.imshow('Video', frame)
  if(cv2.waitKey(1) & 0xFF == ord('q')):
     break
```

```
#Finalmente, cuando termine la captura de video, suelte la captura de video y destruya Todas las ventanas
video_capture.release()
cv2.destroyAllWindows()
```

5.- Se debe instalar Node-red e importar el siguiente Flow (instalando las librerías que requiera).

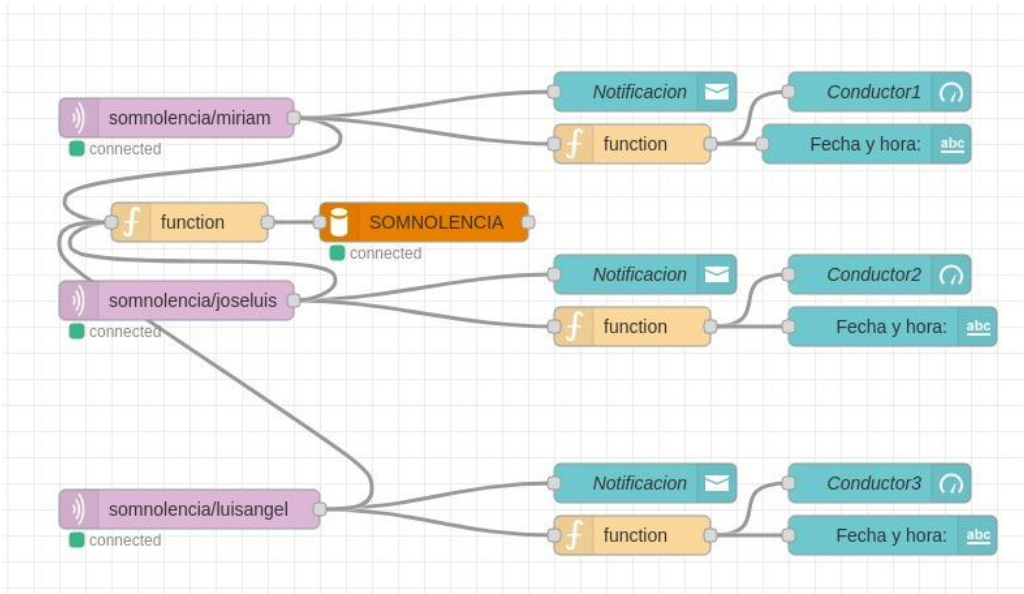

Ilustración 6 Flow para generar dashboard de node-red

6.- Se debe conectar una cámara web a la Raspberry pi 4.

Software y Hardware

Software necesario

Para la correcta ejecución de este programa es necesario contar con las siguientes librerías, la mayoría pueden ser descargadas directamente desde pip, sin embargo, por problemas de compatibilidad algunas otras (como dlib y opencv) deben descargarse de otro modo:

- numpy
- imutils
- getpass
- paho-mqtt
- pygame
- opencv
- dlib

Para descargar opencv y dlib se pueden consultar los siguientes enlaces:

- https://learnopencv.com/install-dlib-on-ubuntu/
- https://pypi.org/project/opencv-contrib-python/

Hardware utilizado

- Raspberry Pi4B
- Webcam GameFactor WG400
- Bocinas BOC-067
- Sensor ultrasónico
- Computadora

Productos

- **Alerta de cansancio:** Monitorear el estado del conductor y dar una alerta cuando este comience a presentar un nivel elevado de somnolencia.
- **Dashboard en Node-Red:** La emisión de la alerta enviara un mensaje a la central de monitoreo indicando el estado actual del conductor identificado por un id.
- **Detector de objetos cercanos:** Control de objetos cercanos al vehículo que iniciara la transmisión-envío de datos por sensor radiofrecuencia

Servicios

- Identificar el estado de alerta del conductor.
- Monitorización en tiempo real del estatus del vehículo.
- Monitorización de los objetos cercanos al área del vehículo.

Conclusiones

Resultados Esperados

- Que la alerta funcione de manera óptima y sea suficiente para poner en alerta al operador del vehículo.
- Que la creación del dashboard envié los mensajes en “tiempo real”, con el fin de conocer el estado del conductor con el fin de intentar evitar un posible accidente.
- Que el sensor ultrasónico ayude a disminuir los puntos ciegos que tiene el vehículo y emita la alerta con eficiencia para evitar un posible accidente.

Limitaciones técnicas

Debido a lo que este proyecto pretende hacer, se puede puede ver limitado debido a algunos factores como: la potencia de la `Raspberry pi 4` que se propone utilizar y que para subir la señal se tiene que conectar a una red wifi, lo que implica una complejidad de diseño y

presupuesto aún mayor. Entonces, si se llegan a mejorar estas limitantes se presentaría un producto con menores perdidas en los datos.

Otra de las limitaciones a considerar es el envío de datos del sensor ultrasónico, dado que, al ser un sensor de mínima capacidad de rango para la detección de objetos cercanos, podría emitir una falsa alarma.

Rol de cada miembro

Hernández Moreno Miriam Lizbeth – **Líder del proyecto**

Pérez Murillo José Luis – **Análisis e implementación de circuitos**

Sánchez Hernández Luis Ángel – **Programador**

A pesar de que se están definiendo roles específicos, cabe destacar que durante la elaboración del proyecto todos los miembros participaron de forma equitativa en todas las actividades para la conclusión de este.

Referencias

- Resumen boletines. (s/f). Instituto Mexicano del Transporte. Recuperado el 7 de marzo de 2022, de https://imt.mx/resumen-boletines.html?IdArticulo=334&IdBoletin=120
- Rosebrock, A. (2017, mayo 8). Drowsiness detection with OpenCV. PyImageSearch. https://pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/
- Soukupová, T. (s/f). Real-time eye blink detection using facial landmarks. Uni-lj.si. Recuperado el 7 de marzo de 2022, de http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf

Nota: Los archivos de audio, código y json (Flow de node-red) necesarios pueden ser encontrados en nuestro repositorio de Github.

https://github.com/DayalizMLHM/Proyecto-Capston