Fundación Universitaria los Libertadores

Aplicativo para la prevención de choques automovilśticos por medio del filtro de Kalman

Facultad de ingeniería y Ciencias Básicas
Programa: Ingenieria Electrónica

Tutor:Nelson Eduardo Lozano Espinosa
Investigador: Daniel Felipe Ochoa Duran

1 Introducción

El presente trabajo de grado se desarrollo con el objetivo disminuir los choques automovilísticos ocurridos en la vía por medio de un aplicativo que tendrá como objetivo principal alertar ante la amenaza de posibles choques predichos con la ayuda de las imágenes capturadas con la ayuda una cámara.

Esto con el fin de brindar un medio necesario a los conductores para disminuir este tipo de hechos de transito ocurridos en Colombia, en donde solo en 2017 se presentaron 151.171 hechos de transito de los cuales tuvieron como resultado 38.073 lesionados y 6479 fallecidos[1].

Basándonos en este contexto el proyecto consta de tres partes principales: adquisición de imágenes e identificación de actores viales en las mismas, predicción de movimiento de dichos actores viales con ayuda del filtro de kalman y por ultimo la implementación de sistema de alerta con los datos previamente adquiridos.

Este trabajo fue desarrollado a lo largo de un año en lenguaje python con la ayuda de librerías como opencv para el procesamiento de imagenes, YOLO y Tensorflow para la identificación de los actores viales, entre otras. Se trabajo en el sistema operativo windows con sus respectivos complementos para el uso del lenguaje python y teniendo como IDE el software Pycharm.

2 Estado del Arte

2.1 Filtro de Kalman

El filtro de Kalman es esencialmente un conjunto de ecuaciones matemáticas que implementan un estimador de tipo predictor-corrector que es óptimo en el sentido de que minimiza la covarianza de error estimada, cuando se cumplen algunas condiciones presuntas. Desde el momento de su introducción, el filtro Kalman ha sido objeto de una amplia investigación y aplicación, particularmente en el área de navegación autónoma o asistida. Esto es probable debido en gran parte a los avances en computación digital que hicieron práctico el uso del filtro, pero también a la relativa simplicidad y naturaleza robusta del mismo filtro. Raramente existen las condiciones necesarias para la optimización, y sin embargo, el filtro aparentemente funciona bien para muchas aplicaciones a pesar de esta situación.

El filtro de Kalman aborda el problema general de tratar de estimar el estado de un proceso controlado de tiempo discreto que se rige por la ecuación de diferencia estocástica lineal:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{2.1}$$

$$z_k = Hx_k + v_k \tag{2.2}$$

Las variables aleatorias y representan el proceso y el ruido de medición (respectivamente). Se supone que son independientes (el uno del otro), blancos y con distribuciones de probabilidad normal.

$$p(w) \sim N(0, Q) \tag{2.3}$$

$$p(v) \sim N(0, R) \tag{2.4}$$

En la práctica, las matrices de covarianza de ruido de proceso Q y de covarianza de ruido R de medición pueden cambiar con cada paso de tiempo o medición, sin embargo aquí asumimos que son nulos.

La matriz A de n x m, en la ecuación de ecuación de diferencia anterior relaciona el estado en el paso de tiempo anterior k-1 con el estado en el paso actual k, en ausencia de una conducción en función o ruido de proceso. La matriz B de n x l relaciona la entrada de control opcional con el estado x. La matriz H en la ecuación de la ecuación de medición relaciona el estado con la medición.

2.1.1 Origenes Computacionales del Filtro de Kalman

Definimos para que sea nuestra estimación del estado a priori en el paso k dado el conocimiento del proceso antes del paso k, y para ser nuestro estado a posteriori estimación en el paso k medida dada. Entonces podemos definir errores de estimación a priori (Ec. 2.5) y a posteriori (Ec. 2.6) como:

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{2.5}$$

$$e_k \equiv x_k - \hat{x}_k \tag{2.6}$$

La covarianza de error de estimación a priori (Ec. 2.7) y aposteriori (Ec. 2.8) son entonces:

$$P_k^- = E[e_k^- e_k^{-T}] \tag{2.7}$$

$$P_k = E[e_k e_k^T] \tag{2.8}$$

Al derivar las ecuaciones para el filtro de Kalman, comenzamos con el objetivo de encontrar una ecuación que calcule una estimación de estado a posteriori como una combinación lineal de una estimación a priori y una diferencia ponderada entre una medición real y una predicción de medición como se muestra a continuación en ecuación anterior. Alguna justificación para la ecuación anterior se da en "Los orígenes probabilísticos del filtro" que se encuentra a continuación.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{2.9}$$

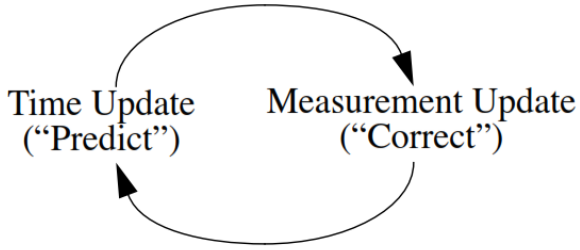
2.1.2 Algoritmo de Filtro Discreto de Kalman

Comenzaremos esta sección con una visión general amplia, cubriendo la operación de "alto nivel" de una forma del filtro Kalman discreto. Después de presentar esta vista de alto nivel, limitaremos el enfoque a las ecuaciones específicas y su uso en esta versión del filtro.

El filtro Kalman estima un proceso utilizando una forma de control de retroalimentación: el filtro estima el estado del proceso en algún momento y luego obtiene retroalimentación en forma de mediciones (ruidosas). Como tal, las ecuaciones para el filtro de Kalman se dividen en dos grupos: ecuaciones de actualización de tiempo y ecuaciones de actualización de medición. Las ecuaciones de actualización de tiempo son responsables de proyectar hacia adelante (en el tiempo) el estado actual y la covarianza de error. Estimaciones para obtener las estimaciones a priori para el siguiente paso de tiempo. Las ecuaciones de actualización de medición son responsables de la retroalimentación, es decir. para incorporar una nueva medición en la estimación a priori para obtener una estimación a posteriori mejorada.

Las ecuaciones de actualización de tiempo también pueden considerarse ecuaciones predictoras, mientras que las ecuaciones de actualización de mediciones pueden considerarse ecuaciones de corrección. De hecho, el algoritmo

de estimación final se asemeja al de un algoritmo predictor-corrector para resolver problemas numéricos como se muestra a continuación en la siguiente figura.



Después de cada par de actualización de tiempo y medición, el proceso se repite con las estimaciones a posteriori anteriores utilizadas para proyectar o predecir las nuevas estimaciones a priori. Esta naturaleza recursiva es una de las características muy atractivas del filtro Kalman: hace que las implementaciones prácticas sean mucho más factibles que por ejemplo una implementación de un filtro Wiener (Brown y Hwang 1996) que está diseñado para operar con todos los datos directamente para cada estimación.

El filtro de Kalman, en cambio, condiciona recursivamente la estimación actual de todas las mediciones pasadas. En donde las ecuaciones que rigen este filtro son las siguientes:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{2.10}$$

$$\hat{P}_k^- = A\hat{P}_{k-1}A^T + Q \tag{2.11}$$

$$K_k = P_k^- h^T (HP_k^- h^T + R)^- 1 \tag{2.12}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.13}$$

$$P_k = (I - K_k H)P_k^- \tag{2.14}$$

Siendo las ecuaciones 2.10,2.11 las ecuaciones de predicción y las ecuaciones 2.12,2.13,2.14 las ecuaciones de actualización.

2.2 Algoritmo YOLO

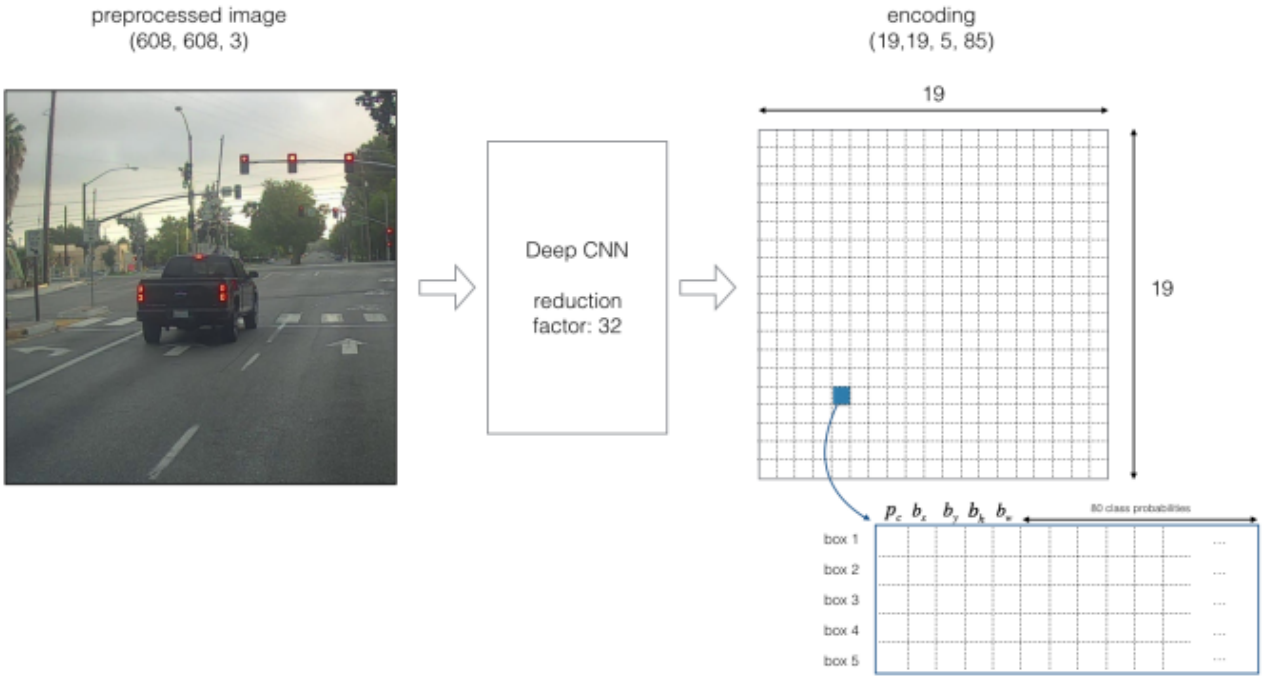
YOLO ("you only look once") es un algoritmo usado comúnmente ya que este alcanza una gran precisión y también teniendo la posibilidad de ejecutarlo en tiempo real. Este algoritmo como indica solo mira una vez a una imagen en el sentido de que solo requiere de un único pase de propagación hacia delante a través de la red para crear predicciones. Después de la supresión no máxima, luego su salida son objetos reconocidos junto con las cajas delimitadoras.

2.2.1 Detalles del Modelo

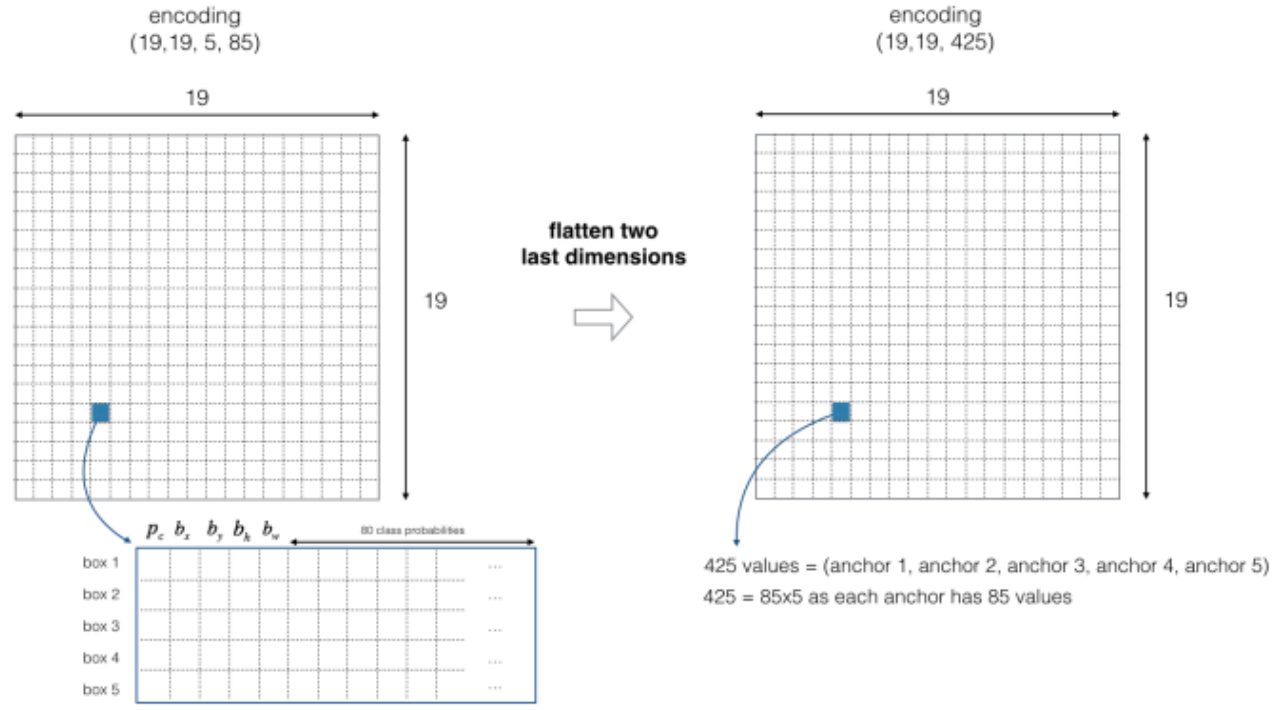
- La entrada es un lote de imágenes de la forma (m, 608, 608, 3).
- La salida es una lista de cajas delimitadoras junto con las clases reconocidas. Cada caja delimitadora está representada por seis números (pc, bx, by, bh, bw, c), Si expandimos c en un vector de 80 dimensiones (Los posibles 80 tipos de objetos que puede detectar), cada caja delimitadora es entonces representada por 85 números.

2.2.2 Codificación

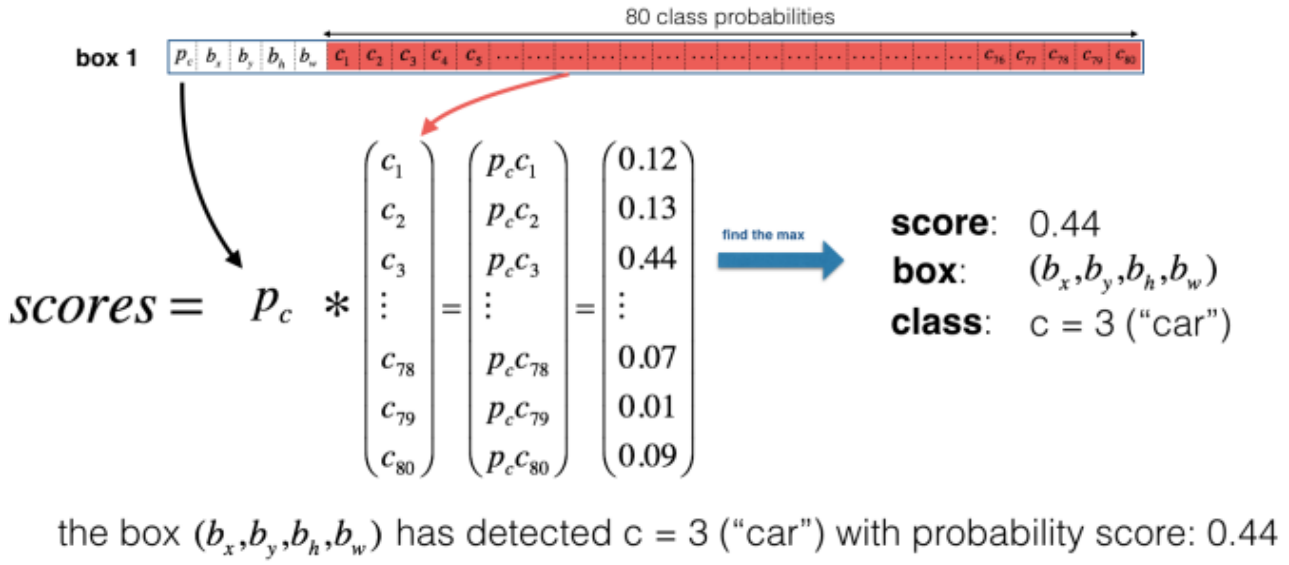
Tendremos una codificacion que tendra como primer paso convertir la imagen (m, 608, 608, 3) a (M, 19, 19, 5, 85) como lo muestra la siguiente imagen



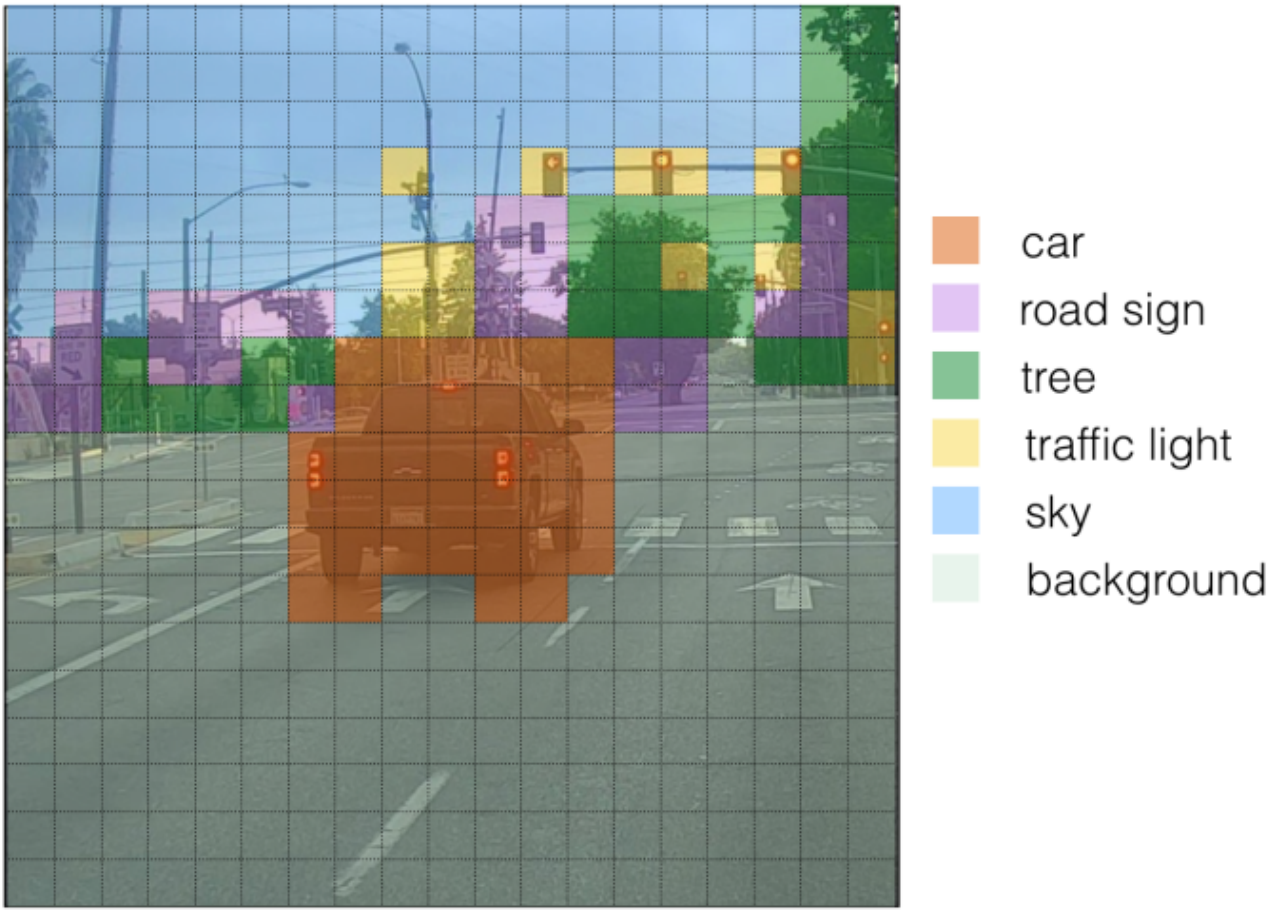
Por simplicidad, aplanaremos las dos últimas dimensiones de la forma codificada (19, 19, 5, 85). Entonces la salida de la CNN profunda es (19, 19, 425). Obteniendo



Ahora, para cada caja (de cada celda) calculamos el siguiente producto basado en elementos y extraer la probabilidad de la caja que contiene cierta clase.



Para cada una de las cuadriculas de celdas de 19 x 19, se encuentra los máximos puntajes de probabilidad (tomando un máximo tanto en las cinco cajas de ancla como en las diferentes clases). Obteniendo

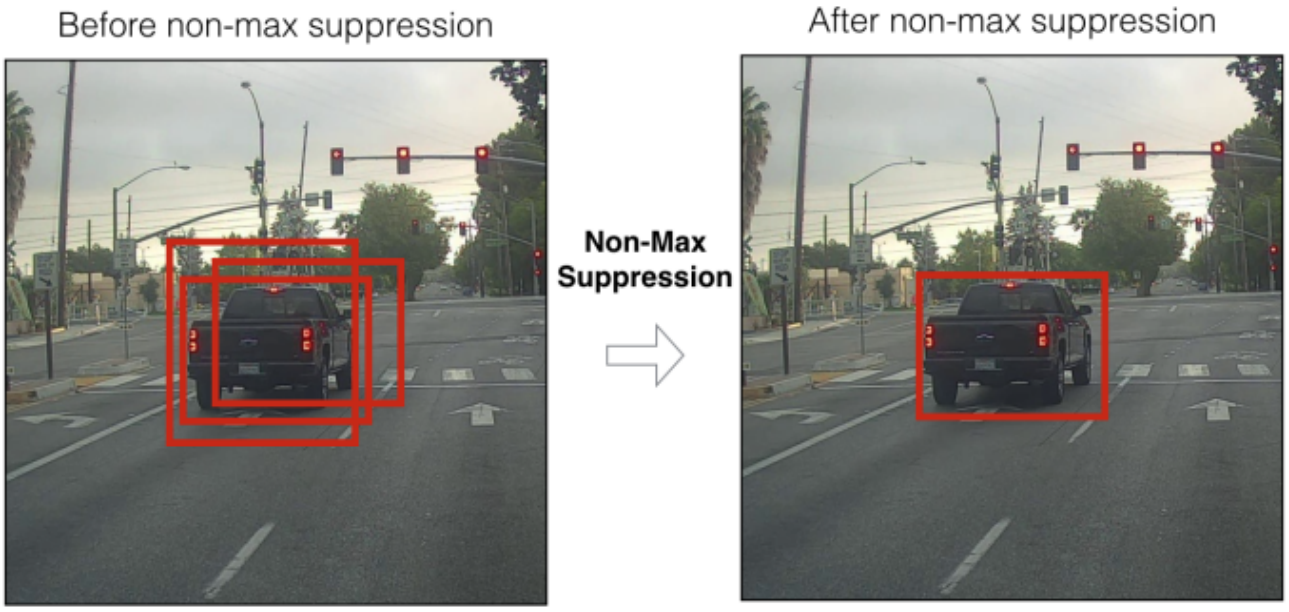


Otra forma de visualizar la salida de YOLO es trazar las cajas delimitadoras que genera. Haciendo esto genera una visualización como esta:

Cada celda nos ofrece cinco cajas. En total, el modelo predice: 19 x 19 x 5 = 1805 cajas solo por mirar una vez a la imagen (un paso hacia delante atreves de la red). Los colores diferentes denotan diferentes clases. En la imagen que se mostraba anteriormente, trazamos solo las cajas que el modelo tiene asignado con una alta probabilidad, pero esto es aún muchas cajas. Sería bueno filtrar la salida del algoritmo a una mucha menor cantidad de objetos detectados. Para entonces, usaremos la supresión no máxima. Específicamente llevamos los siguientes pasos:

- Desechar las cajas con un bajo puntaje (significa que la caja no tiene mucha confianza sobre detectar una clase).
- Seleccione solo una caja cuando varias cajas se sobreponen y detectan el mismo objeto.

Una vez se realiza este filtrado quedaran las cajas con probabilidad más alta de capturar al objeto de manera optima y correcta. Como se ve en la siguiente imagen:



3 Problema

En Colombia tal y como lo establece el Observatorio Nacional de Seguridad Vial solo en 2017 ocurrieron 171.151 hechos de transito en las vías los cuales se clasifican 3 de acuerdo tal y como lo muestra la siguiente gráfica:

Figura 3.1: Cifras de hechos de tránsito según gravedad del hecho[1].

Siendo el 90.37% de los casos ocurridos en zonas urbanas, dejando como saldo un total de 6.479 personas fallecidas y 38.073 personas lesionadas siendo los principales afectados los usuarios de motocicletas y peatones[1]. Y siendo actualmente los principales causantes de estos siniestros de transito: el conductor (37.9%),no mantener distancia de seguridad (10.3%), desobedecer señales de transito(10.1%) y transitar entre vehículos(6.71%)[2]. Y aunque en la actualidad existen sistemas que evitan choques por medición de distancias se dan en automóviles de gamas altas casos como el del toyota prius el cual oscila entre los 23.000 a 25.000 dolares este posee un sistema de radares en el parachoques y en la parrilla delantera[5] sin embargo son características propias de la marca dificultando el acceso de este tipo de sistemas a la población común.

Con el fin de combatir y disminuir los choques ocurridos en Colombia se busca desarrollar un aplicativo el cual advierta al usuario sobre posibles choques con otros actores viales con ayuda de la predicción del movimiento de

estos mismos por medio de la toma,análisis y procesamiento de las imágenes tomadas de la vista del conductor y el Filtro de Kalman.

4 Objetivo Alcanzado

Diseñar y desarrollar un aplicativo (Programa) el cual logre prevenir choques automovilísticos por medio de un sistema que advertirá al usuario un posible choque con la ayuda de las predicciones de movimiento realizadas por medio del filtro de kalman.

5 Descripción General del Producto Final

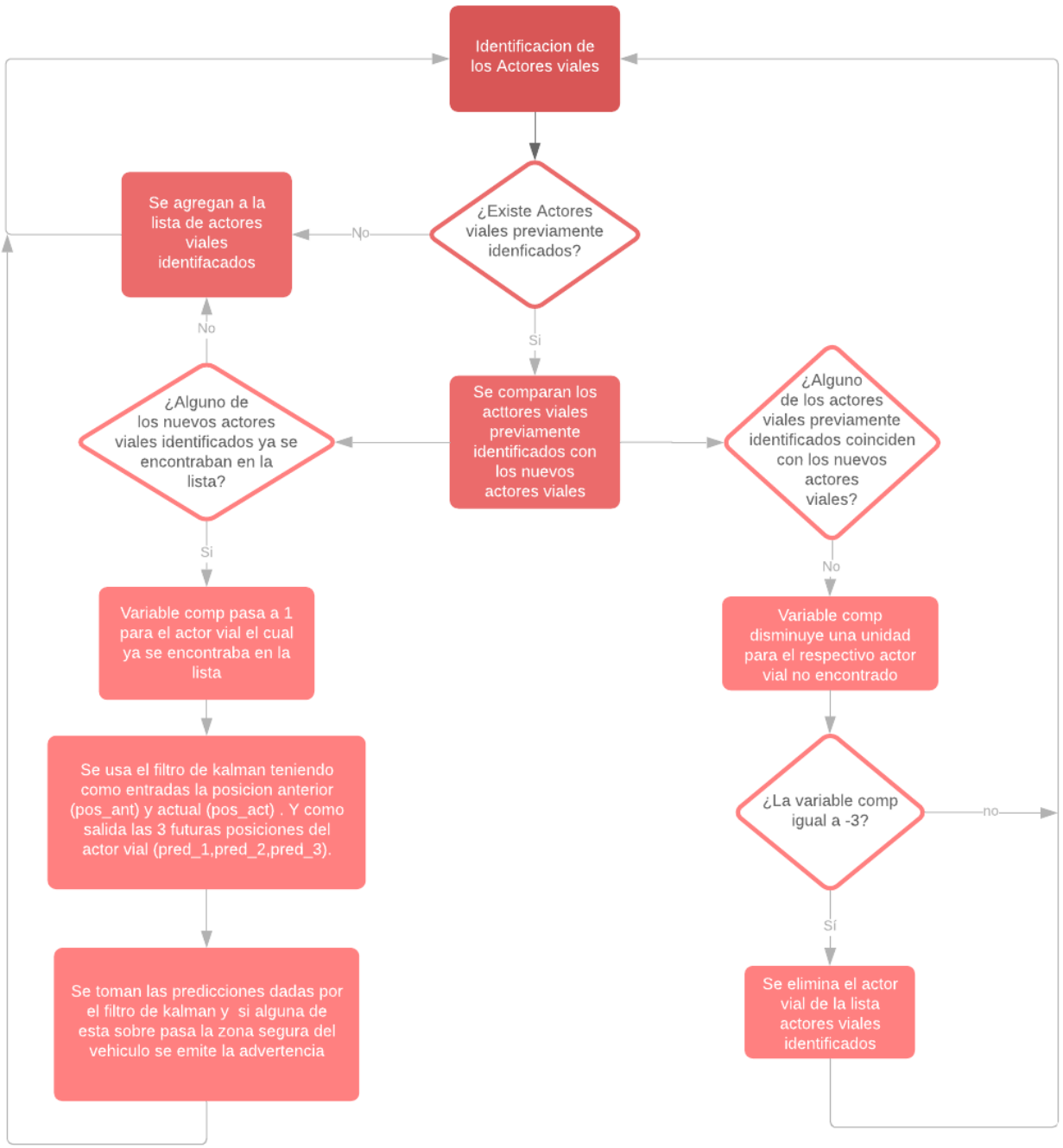


Figura 5.1: Diagrama de Flujo

6 Descripción del Diseño

Teniendo esto en cuenta entraremos a mirar los 4 aspectos fundamentales para el desarrollo del proyecto:

6.1 Reconocimiento Actores viales

Para esta seccion del proyecto reconocieron los distintos actores viales en las imágenes con los siguientes pasos:

6.1.1 Carga de Modelo

En esta sección como primera instancia se inicio sesion que nos permita trabajar con nuestra respectiva red, cargamos un modelo previamente entrenado de nombre "yolo.h5" el cual puede reconocer 80 tipos de objeto, Ademas se cargo el nombre y ancho de cada uno de estos objetos guardados respectivamente en los archivos de texto "coco_classes.txt" y "yolo_anchors.txt". Y por el ultimo se leyó el tamaño de la imagen para su respectivo escalamiento al requerimiento del modelo el cual es de 608x608 tal y como lo describe el siguiente código:

```
sess = K.get_session()
nombre_clases = read_classes("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\
    model_data\\coco_classes.txt")
ancho = read_anchors("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\model_data\\
    yolo_anchors.txt")
modelo_yolo = load_model("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\model_data
    \\yolo.h5")
video = cv2.VideoCapture("VideoFinal.mp4")
f, img = video.read()
image_shape = (float(img.shape[0]), float(img.shape[1]))
```

6.1.2 Conversión de imagen a tensor

Para la realización de este proceso se uso la función yolo_model la cual tiene como entrada una imagen de la forma (m, 608, 608, 3) y como salida un tensor (m, 19, 19, 5, 85) de esta forma la función queda declarada para su posterior uso:

```
salidas_yolo= yolo_head(modelo_yolo.output, ancho, len(nombre_clases))
```

6.1.3 Filtrado

Para realizar el proceso de filtrado en donde pasaran las cajas cuya probabilidad superen un umbral establecido el cual para nuestro casa es de 0.6 y a su vez elimine el solapamiento de cajas se usaron las siguientes funciones

- Filtrado de Cajas: Como su nombre lo indica es la encargada de solamente dejar pasar las cajas que fueran detectadas y que a su vez superen el margen 0.6 en la probabilidad de ser el objeto detectado este procedimiento se realizo con el siguiente código:

```
def filtro_cajas(caja_con, cajas, prob_caja_clases, umbral):
    caja_puntuacion = caja_con * prob_caja_clases
    caja_clases = K.argmax(caja_puntuacion, axis=-1)
    puntuacion_caja_clases = K.max(caja_puntuacion, axis=-1)
    filtro = puntuacion_caja_clases >= umbral
    puntajes = tf.boolean_mask(puntuacion_caja_clases, filtro)
    cajas = tf.boolean_mask(cajas, filtro)
    clases = tf.boolean_mask(caja_clases, filtro)
    return puntajes, cajas, clases
```

- Non max Suppression: Este es el encargado de seleccionar el cuadro correcto entre un conjunto de cuadros que están superpuestos los unos sobre los otros, este procedimiento se realizo con el siguiente código:

```
def non_max_suppression(puntajes, cajas, clases, max_cajas=10, umbral_iou=0.5):
    tensor_max_cajas = K.variable(max_cajas, dtype='int32')
    K.get_session().run(tf.variables_initializer([tensor_max_cajas]))
    indices = tf.image.non_max_suppression(cajas, puntajes, max_cajas, umbral_iou)
    puntajes = K.gather(puntajes, indices)
    cajas = K.gather(cajas, indices)
    clases = K.gather(clases, indices)
    return puntajes, cajas, clases
```

Teniendo estos dos aspectos definidos la funcion evaluar sera la encargada de realizar el filtrado general teniendo en cuenta la salidas dadas por la funcion yolo_head (salidas_yolo), el tamaño de la imagen y el umbral de filtrado tal y como se muestra en el siguiente codigo:

```
def evaluar(salidas, tamano_imagen, umbral=.6):

    caja_con, caja_xy, caja_wh, prob_caja_clases = salidas
    cajas = yolo_boxes_to_corners(caja_xy, caja_wh)
    puntajes, cajas, clases = filtro_cajas(caja_con, cajas, prob_caja_clases, umbral)
    cajas = scale_boxes(cajas, tamano_imagen)
    puntajes, cajas, clases = non_max_suppression(puntajes, cajas, clases)
    return puntajes, cajas, clases
```

6.1.4 Pronostico

Una vez este todo declaro la función pronostico sera la encargada de escalar la imagen de entrada y procesarla para por cada uno de los procesos previamente descritos para obtener como salida los puntajes (probabilidad de que sea un objeto),cajas (localizacion del objeto en la imagen) y clases (tipo de objeto) tal y como lo muestra el siguiente codigo:

```
    imagen_escalada = cv2.resize(imagen, (608, 608), interpolation=cv2.INTER_CUBIC)
    datos_imagen = np.array(imagen_escalada, dtype='float32')
    datos_imagen /= 255.
    datos_imagen = np.expand_dims(datos_imagen, 0)
    puntajes_salida, cajas_salida, clases_salida = sess.run([puntajes, cajas, clases],
                                                feed_dict={modelo_yolo.input: datos_imagen,
    K.learning_phase(): 0})
    colores = generate_colors(nombre_clases)
    draw_boxes(imagen, puntajes_salida, cajas_salida, clases_salida, nombre_clases, colores)
    #Esta linea dibuja las cajas delimitadoras es opcional su uso

    return puntajes_salida, cajas_salida, clases_salida
```

6.2 Comparacion de Actores Viales

La comparación de actores viales es un proceso en el cual se comparan dos imágenes y se determinan si se trata del mismo actor o no. Para realizacion de este procedimiento se usaron dos metodos:

- Como primer aspecto se realiza una correlación entre el histograma de las dos imágenes esta deberá superar un correlación promedio de los tres histogramas (R,G,B) a 0.7 para determinar que las dos imágenes equivalen al mismo actor vial.
- Como actor complementario de el primer factor una vez el actor halla pasado por el filtro de kalman y se tenga una estimacion previa de la ubicación, se comparara la estimacion con la ubicación actual en el caso de que esta ubicación tenga un margen de error menor o igual a 5% se le sumara 0.25 al promedio de correlación de histogramas.

Cabe aclarar que todo este procedimiento tendra como salida un booleano que me determina si las imagenes comparadas equivalen al mismo actor vial. Todo esto se ve reflejado en la siguiente función:

```
def comparacion(img1, img2, estado, prediccion, fila, columna):
    hist1b = cv2.calcHist([img1], [0], None, [256], [0, 256])
    hist1g = cv2.calcHist([img1], [1], None, [256], [0, 256])
    hist1r = cv2.calcHist([img1], [2], None, [256], [0, 256])

    hist2b = cv2.calcHist([img2], [0], None, [256], [0, 256])
    hist2g = cv2.calcHist([img2], [1], None, [256], [0, 256])
    hist2r = cv2.calcHist([img2], [2], None, [256], [0, 256])

    zb = cv2.compareHist(hist1b, hist2b, cv2.HISTCMP_CORREL)
    zg = cv2.compareHist(hist1g, hist2g, cv2.HISTCMP_CORREL)
    zr = cv2.compareHist(hist1r, hist2r, cv2.HISTCMP_CORREL)

    prom = (zb + zr + zg) / 3

    if estado == 2:
        a = prediccion[0][0] / fila
        b = prediccion[0][1] / columna
        prompos = (a + b) / 2
        print('prompos', prompos)
        if prompos >= 0.95 and prompos <= 1.05:
            prom = prom + 0.25
    if prom >= 0.7:
        flag = True
    else:
        flag = False
    return flag
```

6.3 Manejo Lista Actores Viales

El desarrollo de este proyecto tiene como eje estructurado una lista declarada con el nombre lst en donde su tamaño dependerá de los actores viales reconocidos y en donde se guardaran a lo largo del proceso los siguientes parámetros:

$$[[est], [img], [pos\_ant], [pos\_act], [p\_prev], [pred\_1], [pred\_2], [pred\_3], [comp]] \tag{6.1}$$

En donde:

- est: Es una variable de tipo entera la cual puede tomar dos valores:
  - 0: Este valor me indica que el elemento de la lista aun no ha pasado por el filtro de kalman por lo tanto variables como $p\_prev, pred\_1, pred\_2, pred\_3$ aun no han sido calculadas.
  - 1: Este valor me indica que el elemento de la lista aun no ha pasado por el filtro de kalman por lo tanto variables como $p\_prev, pred\_1, pred\_2, pred\_3$ sin embargo ya ha sido encontrado en dos frames y tiene los requerimientos necesarios para pasar por el filtro de kalman.
  - 2: Este valor me indica que el elemento ya ha pasado por el filtro de kalman al menos una vez y existen estimaciones de las variables como $p\_prev, pred\_1, pred\_2, pred\_3$.
- img: Esta variable contiene la imagen del actor vial reconocido.
- pos_ant: Este vector guarda la posición anterior en la imagen en donde se encontraba el actor vial.
- pos_act: Este vector guarda la posición actual en la imagen en donde se encuentra el actor vial.
- p_prev: Este vector guarda la covariancia calculada por el filtro de kalman entre la posición anterior y la posición actual.
- pred_1,pred_2,pred_3: Son las posiciones futuras del actor vial calculadas con ayuda del filtro de kalman.
- comp: Es una variable de tipo entera la cual puede tomar los siguientes valores:
  - 0: El actor vial aun no ha sido comparado.
  - 1: El actor vial ha sido comparado y ha sido encontrado.
  - -1: El actor vial ha sido comparado y no ha sido encontrado durante un frame.
  - -2: El actor vial ha sido comparado y no ha sido encontrado durante dos frames.
  - -3: El actor vial ha sido comparado y no ha sido encontrado durante tres frames.

Declarada de la siguiente forma:

```
#lst = [['est'], ['img'], ['pos ant'], ['pos act'], ['p prev'], ['pred 1'], ['pred 2'], ['pred
    3'], ['comp']]
lst = [0]
```

Teniendo en cuenta estableceremos las 3 situaciones principales que suceden en el manejo de esta lista:

6.3.1 Añadir Elementos

Para añadir elementos a la lista ocurre básicamente en dos ocasiones:

- Cuando existen actores viales reconocidos y la lista se encuentra vacía.
- Cuando existen actores viales reconocidos y ninguno de ellos ha sido encontrado (Pasado la prueba de comparación) en los elementos ya existentes en la lista.

Al momento de añadir un elemento en la lista se establecen los siguientes parámetros:

- est = 0.
- img guarda el fragmento de imagen detectado
- pos_act guarda las coordenadas del centro de masa de la imagen detectada.

6.4 Eliminar elementos

Este proceso únicamente ocurre cuando después de 3 frames de haber sido detectado un actor vial este no ha sido encontrado nuevamente en la imagen.

6.5 Elementos comunes

Este es el proceso que ocurre cuando un actor vial identificado ya pertenecía previamente a la lista. Este proceso se divide en dos partes principales

6.5.1

Las variables cambian de la siguiente forma

- est = 1
- img guarda la version más actualizada (ultima reconocida) de la imagen del actor vial
- pos_ant = pos_act
- pos_act guarda las ultima coordenadas del centro de masa de la imagen detectada.
- comp = 1

6.5.2

Una vez el elemento tiene estos elementos declarados entrara al filtro de kalman donde:

- p_prev,pred_1,pred_3,pred_2 toman el respectivo valor que les asigne el filtro de kalman
- est = 2
- comp = 0

Todo esto se ve reflejado en el siguiente codigo:

```
    if (len(lst) <= 1):
        if (puntajes_salida.shape[0] > 0):
            for i in range(0, len(puntajes_salida)):
                a = int(cajas_salida[i][0])
                b = int(cajas_salida[i][2])
                c = int(cajas_salida[i][1])
                d = int(cajas_salida[i][3])
                imgaux = img[a:b, c:d]
                fil = a + int((b - a) / 2)
                col = c + int((d - c) / 2)
                lst.append([0, imgaux, 0, [[fil, col], [0, 0]], 0, 0, 0, 0, 0])
    else:
        if (puntajes_salida.shape[0] > 0):
            for i in range(0, len(puntajes_salida)):
                a = int(cajas_salida[i][0])
                b = int(cajas_salida[i][2])
                c = int(cajas_salida[i][1])
                d = int(cajas_salida[i][3])
                if (a < 0):
                    a = 0
                if (b < 0):
                    b = 0
                if (c < 0):
                    c = 0
                if (d < 0):
                    d = 0
                imgaux = img[a:b, c:d]
                fil = a + int((b - a) / 2)
                col = c + int((d - c) / 2)
                for j in range(1, len(lst)):
                    f = False
                    if lst[j][8] == 0:
                        f = comparacion(imgaux, lst[j][1], lst[j][0], lst[j][5], fil, col)
                        if f == True:
                            if lst[j][0] == 0:
                                lst[j][0] = 1
                            lst[j][2] = [[lst[j][3][0][0], lst[j][3][0][1]], [0, 0], [0, 0],
    [0, 0]]
                            lst[j][3] = [[fil, col], [0, 0]]
                            lst[j][8] = 1
                            break
                if f == False:
                    lst.append([0, imgaux, 0, [[fil, col], [0, 0]], 0, 0, 0, 0, 0])
        for j in range(1, len(lst)):
```

```
            if lst[j][8] <= 0:
                lst[j][8] = lst[j][8] - 1
                if lst[j][0] == 2:
                    lst[j][2] = [[lst[j][3][0][0], lst[j][3][0][1]], [0, 0], [0, 0], [0, 0]]
                    lst[j][3] = [[lst[j][5][0][0], lst[j][5][0][1]], [0, 0]]
                    lst[j][5] = lst[j][6]
                    lst[j][6] = lst[j][7]

            if lst[j][8] == 1:
                estado, x_act, p_prev, x_pred_1, x_pred_2, x_pred_3 = filtro_kalman(lst[j][0],
    lst[j][2], lst[j][3],lst[j][4])
                lst[j][0] = estado
                lst[j][4] = p_prev
                lst[j][5] = x_pred_1
                lst[j][6] = x_pred_2
                lst[j][7] = x_pred_3
                lst[j][8] = 0
        cont = 0
        for j in range(1, len(lst)):
            j = j - cont
            print(lst[j][8], lst[j][0])
            #Graficar puntos estimados del filtro de kalman
            if lst[j][0] == 2:
                img = cv2.circle(img, ((int)(lst[j][3][0][1]), (int)(lst[j][3][0][0])), 10,
    (0, 255, 0), -1)
                img = cv2.circle(img, ((int)(lst[j][5][0][1]), (int)(lst[j][5][0][0])), 10,
    (0, 0, 255), -1)
                img = cv2.circle(img, ((int)(lst[j][6][0][1]), (int)(lst[j][6][0][0])), 10,
    (0, 0, 255), -1)
                img = cv2.circle(img, ((int)(lst[j][7][0][1]), (int)(lst[j][7][0][0])), 10,
    (0, 0, 255), -1)

                #Simulacion Aviso
                if (lst[j][3][0][0] >= (image_shape[0] - porsec) or lst[j][5][0][0] >= (
    image_shape[0] - porsec) or
                        lst[j][6][0][0] >= (image_shape[0] - porsec) or lst[j][7][0][0] >= (
    image_shape[0] - porsec)):
                    cv2.imshow('Estado', R)
                else:
                    cv2.imshow('Estado', G)

            if lst[j][8] <= -3:
                lst.pop(j)
                cont = cont + 1
```

6.6 Filtro de Kalman

Este filtro consta de dos partes principales basadas en sus respectivas ecuaciones, las cuales son:

6.6.1 Predicción

```
def prediccion(x_prev, p_prev, a, b, u, q):
    x_pred = np.dot(a, x_prev) + np.dot(b, u)
    p_pred = np.dot(a, np.dot(p_prev, a.transpose())) + q
    return x_pred, p_pred
```

6.7 Actualización

```
    v = y_act - np.dot(h, x_pred)
    s = np.dot(h, np.dot(p_pred, h.transpose())) + r
    k = np.dot(p_pred, np.dot(h.transpose(), np.linalg.inv(s)))
    x = x_pred + np.dot(k, v)
    p = p_pred - np.dot(k, np.dot(s, k.transpose()))
    return x, p
```

6.8 Declaración

Cabe aclarar que los ejes principales de este filtro son "x" y "x_pred" los cuales son nuestras variables de posicion y "p" y "p_pred" sus respectivas correlaciones. Teniendo esto en cuenta es necesario de minimo dos momentos de tiempo distintos para la predicciones de futuras posiciones teniendo en cuenta que se necesitara una "p_prev" para el caso de la primera prediccion se le asignara una correlacion auxiliar que sera remplazada una vez se realice la fase de actualización del filtro. Esta sera usada como "p_prev" para futuras ocasiones teniendo como control la variable est (variables almacenadas en la lista de actores viales) para mayor precision de la predicción.

Siendo este el flujo básico del filtro predicción, actualización y posteriormente 3 predicciones futuras basadas en los datos dados por la actualización. Tal y como lo muestra el siguiente codigo:

```
def filtro_kalman(estado, x_prev, x_act, p_prev):
    if estado == 1:
        p_prev = np.diag((0.01, 0.01, 0.01, 0.01))
    dt = 100
    a = np.array([[1, 0, dt, 0], [0, 1, 0, dt], [0, 0, 1, 0], [0, 0, 0, 1]])
    size = 4
    q = np.eye(size)
    b = np.eye(size)
    u = np.zeros((size, 1))

    h = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
    r = np.eye(2)
    x_pred, p_pred = prediccion(x_prev, p_prev, a, b, u, q)
    x_prev, p_prev = actulizacion(x_pred, p_pred, x_act, r, h)
    x_pred_1, p_pred_1 = prediccion(x_prev, p_prev, a, b, u, q)
    x_pred_2, p_pred_2 = prediccion(x_pred_1, p_pred_1, a, b, u, q)
    x_pred_3, p_pred_3 = prediccion(x_pred_2, p_pred_2, a, b, u, q)
    estado = 2
    return estado, x_act, p_prev, x_pred_1, x_pred_2, x_pred_3
```

6.9 Gráfica y Advertencia

Esta es la parte final del proyecto en donde se vera reflejado la predicción de las posiciones de los actores viales y si estos exceden o entran a la zona segura del vehiculo.

6.9.1 Zona Segura

Esta equivale a un porcentaje de la imagen en su parte inferior para nuestro casi equivalente al 25% y se calcula y demarca con respecto al tamaño de la imagen como se ve en el siguiente codigo:

```
porsec = image_shape[0] * 0.25
    img = cv2.line(img, (0,int(image_shape[0]-porsec)), (int(image_shape[1]),int(image_shape
    [0]-porsec)), (0, 0, 255), 4)
```

6.9.2 Advertencia y Visualización

Esta es una parte muy simple del código donde se evalúa si las posiciones (actual y predicciones) superan la zona segura sirviendo como notificación una imagen que cambiara de color Verde o Rojo de acuerdo al estado que se encuentre (Verde = Normal, Rojo=Advertencia) cabe aclarar que este sistema es únicamente ilustrativo y en el caso de que el proyecto se llegue a implementar deberá ser cambiado por un sistema auditivo que advierta al conductor. Ademas a su vez este sistema proyectara por puntos las predicción del Filtro siendo el punto verde la posición actual y los puntos rojos las predicciones. Este desarrollo se ve de la siguiente manera:

```
R = cv2.imread('R.png')
G = cv2.imread('G.png')
flagf = False
if lst[j][0] == 2:
    img = cv2.circle(img, ((int)(lst[j][3][0][1]), (int)(lst[j][3][0][0])), 10, (0, 255, 0),
    -1)
    img = cv2.circle(img, ((int)(lst[j][5][0][1]), (int)(lst[j][5][0][0])), 10, (0, 0, 255),
    -1)
    img = cv2.circle(img, ((int)(lst[j][6][0][1]), (int)(lst[j][6][0][0])), 10, (0, 0, 255),
    -1)
    img = cv2.circle(img, ((int)(lst[j][7][0][1]), (int)(lst[j][7][0][0])), 10, (0, 0, 255),
    -1)
    # Simulacion Aviso
    if (lst[j][3][0][0] >= (image_shape[0] - porsec) or lst[j][5][0][0] >= (image_shape[0] -
    porsec) or
            lst[j][6][0][0] >= (image_shape[0] - porsec) or lst[j][7][0][0] >= (image_shape[0]
     - porsec)):
            flagf = True

if flagf == True:
    cv2.imshow('Estado', R)
else:
    cv2.imshow('Estado', G)
```

6.10 Codigo Final

Cabe aclarar que el proceso se base en la captura, identificación de actores, comparación, predicción (filtro kalman) y advertencia y este proceso se repetirá frame a frame como se muestra en el siguiente codigo:

```
import numpy as np
import cv2
import tensorflow as tf
from keras import backend as K
from keras.models import load_model
from Proyecto.yolo_utils import read_classes, read_anchors, generate_colors, draw_boxes,
    scale_boxes
from Proyecto.yad2k.models.keras_yolo import yolo_head, yolo_boxes_to_corners

def filtro_cajas(caja_con, cajas, prob_caja_clases, umbral):
    caja_puntuacion = caja_con * prob_caja_clases
    caja_clases = K.argmax(caja_puntuacion, axis=-1)
    puntuacion_caja_clases = K.max(caja_puntuacion, axis=-1)
    filtro = puntuacion_caja_clases >= umbral
    puntajes = tf.boolean_mask(puntuacion_caja_clases, filtro)
    cajas = tf.boolean_mask(cajas, filtro)
    clases = tf.boolean_mask(caja_clases, filtro)
    return puntajes, cajas, clases

def non_max_suppression(puntajes, cajas, clases, max_cajas=10, umbral_iou=0.5):
    tensor_max_cajas = K.variable(max_cajas, dtype='int32')
    K.get_session().run(tf.variables_initializer([tensor_max_cajas]))
    indices = tf.image.non_max_suppression(cajas, puntajes, max_cajas, umbral_iou)
    puntajes = K.gather(puntajes, indices)
    cajas = K.gather(cajas, indices)
    clases = K.gather(clases, indices)
    return puntajes, cajas, clases

def evaluar(salidas, tamano_imagen, umbral=.6):
    caja_con, caja_xy, caja_wh, prob_caja_clases = salidas
    cajas = yolo_boxes_to_corners(caja_xy, caja_wh)
    puntajes, cajas, clases = filtro_cajas(caja_con, cajas, prob_caja_clases, umbral)
    cajas = scale_boxes(cajas, tamano_imagen)
    puntajes, cajas, clases = non_max_suppression(puntajes, cajas, clases)
    return puntajes, cajas, clases

def pronosticar(sess, imagen):
    imagen_escalada = cv2.resize(imagen, (608, 608), interpolation=cv2.INTER_CUBIC)
    datos_imagen = np.array(imagen_escalada, dtype='float32')
    datos_imagen /= 255.
    datos_imagen = np.expand_dims(datos_imagen, 0)
    puntajes_salida, cajas_salida, clases_salida = sess.run([puntajes, cajas, clases],
                                                            feed_dict={modelo_yolo.input:
    datos_imagen,                                                                    K.
    learning_phase(): 0})
    colores = generate_colors(nombre_clases)
    draw_boxes(imagen, puntajes_salida, cajas_salida, clases_salida, nombre_clases, colores)
    return puntajes_salida, cajas_salida, clases_salida

def comparacion(img1, img2, estado, prediccion, fila, columna):
    hist1b = cv2.calcHist([img1], [0], None, [256], [0, 256])
    hist1g = cv2.calcHist([img1], [1], None, [256], [0, 256])
    hist1r = cv2.calcHist([img1], [2], None, [256], [0, 256])

    hist2b = cv2.calcHist([img2], [0], None, [256], [0, 256])
    hist2g = cv2.calcHist([img2], [1], None, [256], [0, 256])
    hist2r = cv2.calcHist([img2], [2], None, [256], [0, 256])
    zb = cv2.compareHist(hist1b, hist2b, cv2.HISTCMP_CORREL)
    zg = cv2.compareHist(hist1g, hist2g, cv2.HISTCMP_CORREL)
    zr = cv2.compareHist(hist1r, hist2r, cv2.HISTCMP_CORREL)

    prom = (zb + zr + zg) / 3

    if estado == 2:
        a = prediccion[0][0] / fila
        b = prediccion[0][1] / columna
        prompos = (a + b) / 2
        print('prompos', prompos)
        if prompos >= 0.95 and prompos <= 1.05:
            prom = prom + 0.25

    if prom >= 0.7:
```

```
        flag = True
    else:
        flag = False
    return flag

def prediccion(x_prev, p_prev, a, b, u, q):
    x_pred = np.dot(a, x_prev) + np.dot(b, u)
    p_pred = np.dot(a, np.dot(p_prev, a.transpose())) + q
    return x_pred, p_pred

def actulizacion(x_pred, p_pred, y_act, r, h):
    v = y_act - np.dot(h, x_pred)
    s = np.dot(h, np.dot(p_pred, h.transpose())) + r
    k = np.dot(p_pred, np.dot(h.transpose(), np.linalg.inv(s)))
    x = x_pred + np.dot(k, v)
    p = p_pred - np.dot(k, np.dot(s, k.transpose()))
    return x, p

def filtro_kalman(estado, x_prev, x_act, p_prev):
    if estado == 1:
        p_prev = np.diag((0.01, 0.01, 0.01, 0.01))
    dt = 100
    a = np.array([[1, 0, dt, 0], [0, 1, 0, dt], [0, 0, 1, 0], [0, 0, 0, 1]])
    size = 4
    q = np.eye(size)
    b = np.eye(size)
    u = np.zeros((size, 1))

    h = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
    r = np.eye(2)
    x_pred, p_pred = prediccion(x_prev, p_prev, a, b, u, q)
    x_prev, p_prev = actulizacion(x_pred, p_pred, x_act, r, h)
    x_pred_1, p_pred_1 = prediccion(x_prev, p_prev, a, b, u, q)
    x_pred_2, p_pred_2 = prediccion(x_pred_1, p_pred_1, a, b, u, q)
    x_pred_3, p_pred_3 = prediccion(x_pred_2, p_pred_2, a, b, u, q)
    estado = 2
    return estado, x_act, p_prev, x_pred_1, x_pred_2, x_pred_3

sess = K.get_session()
nombre_clases = read_classes("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\
    model_data\\coco_classes.txt")
ancho = read_anchors("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\model_data\\
    yolo_anchors.txt")
modelo_yolo = load_model("C:\\Users\\Danie\\Documents\\Proyectos\\Tesis\\Proyecto\\model_data
    \\yolo.h5")
video = cv2.VideoCapture("VideoFinal.mp4")
f, img = video.read()
image_shape = (float(img.shape[0]), float(img.shape[1]))
porsec = image_shape[0] * 0.25
R = cv2.imread('R.png')
G = cv2.imread('G.png')
salidas_yolo = yolo_head(modelo_yolo.output, ancho, len(nombre_clases))
puntajes, cajas, clases = evaluar(salidas_yolo, image_shape)
#lst = [['est'], ['img'], ['pos ant'], ['pos act'], ['p prev'], ['pred 1'], ['pred 2'], ['pred
     3'], ['comp']]
lst = [0]
while (True):
    flagf = False
    puntajes_salida, cajas_salida, clases_salida = pronosticar(sess, img)
    if (len(lst) <= 1):
        if (puntajes_salida.shape[0] > 0):
            for i in range(0, len(puntajes_salida)):
                a = int(cajas_salida[i][0])
                b = int(cajas_salida[i][2])
                c = int(cajas_salida[i][1])
                d = int(cajas_salida[i][3])
                imgaux = img[a:b, c:d]
                fil = a + int((b - a) / 2)
                col = c + int((d - c) / 2)
                lst.append([0, imgaux, 0, [[fil, col], [0, 0]], 0, 0, 0, 0, 0])
    else:
        if (puntajes_salida.shape[0] > 0):
            for i in range(0, len(puntajes_salida)):
                a = int(cajas_salida[i][0])
                b = int(cajas_salida[i][2])
                c = int(cajas_salida[i][1])
```

```
                d = int(cajas_salida[i][3])
                if (a < 0):
                    a = 0
                if (b < 0):
                    b = 0
                if (c < 0):
                    c = 0
                if (d < 0):
                    d = 0
                imgaux = img[a:b, c:d]
                fil = a + int((b - a) / 2)
                col = c + int((d - c) / 2)
                for j in range(1, len(lst)):
                    f = False
                    if lst[j][8] == 0:
                        f = comparacion(imgaux, lst[j][1], lst[j][0], lst[j][5], fil, col)
                        if f == True:
                            if lst[j][0] == 0:
                                lst[j][0] = 1
                            lst[j][2] = [[lst[j][3][0][0], lst[j][3][0][1]], [0, 0], [0, 0],
[0, 0]]
                            lst[j][3] = [[fil, col], [0, 0]]
                            lst[j][8] = 1
                            break
                if f == False:
                    lst.append([0, imgaux, 0, [[fil, col], [0, 0]], 0, 0, 0, 0, 0])
        for j in range(1, len(lst)):
            if lst[j][8] <= 0:
                lst[j][8] = lst[j][8] - 1
                if lst[j][0] == 2:
                    lst[j][2] = [[lst[j][3][0][0], lst[j][3][0][1]], [0, 0], [0, 0], [0, 0]]
                    lst[j][3] = [[lst[j][5][0][0], lst[j][5][0][1]], [0, 0]]
                    lst[j][5] = lst[j][6]
                    lst[j][6] = lst[j][7]
            if lst[j][8] == 1:
                estado, x_act, p_prev, x_pred_1, x_pred_2, x_pred_3 = filtro_kalman(lst[j][0],
 lst[j][2], lst[j][3],
                   lst[j][4])
                lst[j][0] = estado
                lst[j][4] = p_prev
                lst[j][5] = x_pred_1
                lst[j][6] = x_pred_2
                lst[j][7] = x_pred_3
                lst[j][8] = 0
        cont = 0
        for j in range(1, len(lst)):
            j = j - cont
            print(lst[j][8], lst[j][0])
            # Graficar puntos estimados del filtro de kalman
            if lst[j][0] == 2:
                img = cv2.circle(img, ((int)(lst[j][3][0][1]), (int)(lst[j][3][0][0])), 10,
(0, 255, 0), -1)
                img = cv2.circle(img, ((int)(lst[j][5][0][1]), (int)(lst[j][5][0][0])), 10,
(0, 0, 255), -1)
                img = cv2.circle(img, ((int)(lst[j][6][0][1]), (int)(lst[j][6][0][0])), 10,
(0, 0, 255), -1)
                img = cv2.circle(img, ((int)(lst[j][7][0][1]), (int)(lst[j][7][0][0])), 10,
(0, 0, 255), -1)
                # Simulacion Aviso
                if lst[j][3][0][0] >= (image_shape[0] - porsec) or lst[j][5][0][0] >= (
image_shape[0] - porsec) or \
                        lst[j][6][0][0] >= (image_shape[0] - porsec) or lst[j][7][0][0] >= (
image_shape[0] - porsec or flagf == True):
                    flagf = True
            if lst[j][8] <= -3:
                lst.pop(j)
                cont = cont + 1
    img = cv2.line(img, (0, int(image_shape[0] - porsec)), (int(image_shape[1]), int(
image_shape[0] - porsec)),
                   (0, 0, 255), 4)
    if flagf == True:
        cv2.imshow('Estado', R)
    else:
        cv2.imshow('Estado', G)
    cv2.imshow("Salida Final", img)
    f, img = video.read()
```

7 Manejo y Operatividad del Producto Final

7.1 Caracteristicas Tecnicas

Para el manejo de este proyecto es recomendable que el computador cumpla con los siguientes requerimientos:

- Memoria RAM 8GB
- Tarjeta de Video Nvidia
- Procesador Intel Core i5 o Superior
- Sistema Operativo compatible con Python

Como factor principal para mejorar el rendimiento del proceso la Tarjeta de Video (Con su respectivo soporte para este tipo de procesos CUDA) es fundamental debido a que tensorflow trabaja con este tipo de tarjetas sus desarrollos agilizando el proceso.

7.1.1 Interface

Para el manejo de este proyecto es necesario una IDE compatible con Python 3.7 se recomienda Pycharm (IDE trabajada) con la siguiente lista de librerias:

Package	Version
Click	7.0
Keras	2.2.4
Keras-Applications	1.0.8
Keras-Preprocessing	1.1.0
Markdown	3.1.1
Pillow	6.2.1
PyYAML	5.1.2
Werkzeug	0.16.0
absl-py	0.8.1
astor	0.8.0
cffi	1.13.2
colorama	0.4.1
enum34	1.1.6
gast	0.2.2
google-pasta	0.1.8
grpcio	1.25.0
h5py	2.10.0
numpy	1.17.3
opencv-python	4.1.1.26
opt-einsum	3.1.0
pip	19.0.3
plaidbench	0.6.4
plaidml	0.6.4
plaidml-keras	0.6.4
protobuf	3.10.0
pycparser	2.19
pynput	1.4.5
scipy	1.3.2
setuptools	41.6.0
six	1.13.0
tensorboard	1.15.0
tensorflow	1.15.0
tensorflow-estimator	1.15.1
termcolor	1.1.0
wheel	0.33.6
wrapt	1.11.2

Figura 7.1: Librerias

Se debe manejar un version inferior de tensorflow a la 2.0 (Actual) debido a que la función getsession() vital para el proyecto fue removida de en la ultima actualización.

7.2 Ejecución

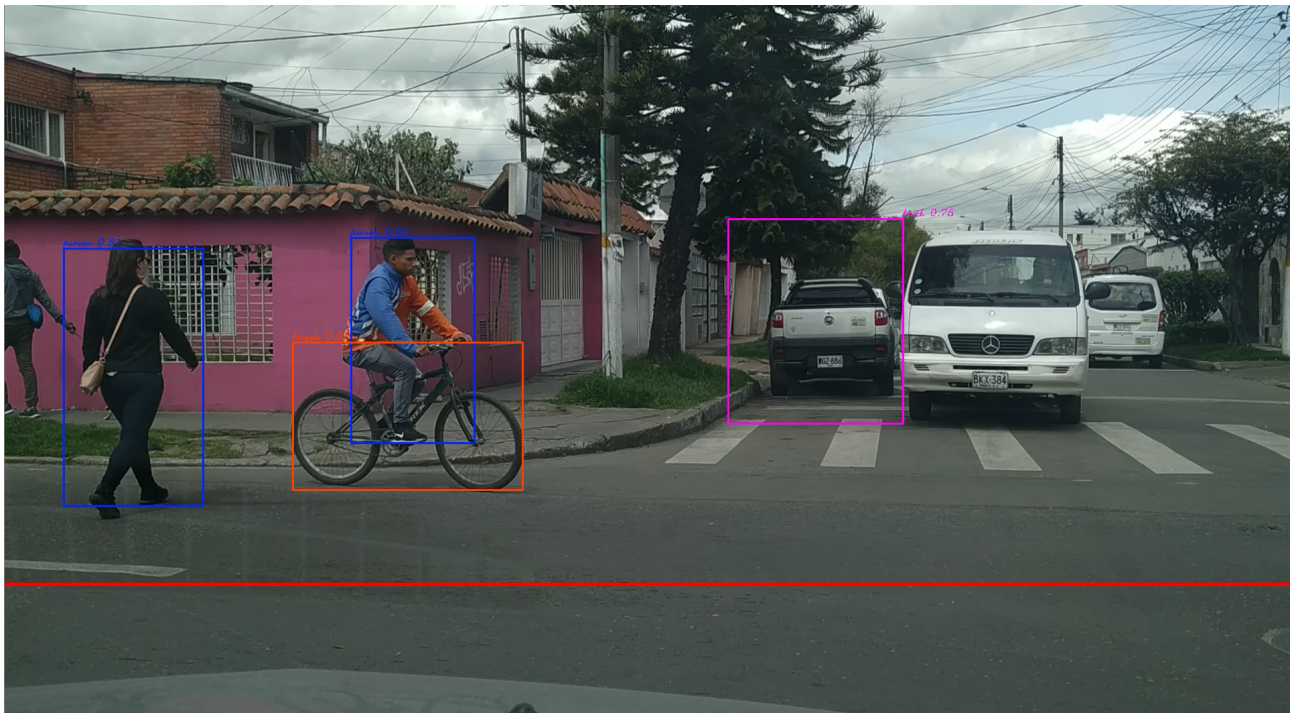
Para la ejecución de este archivo sera necesario cumplir las anteriores especificaciones ademas de tener los archivos complementos de yad2k para la librería yolo. Teniendo esto en cuenta el script se podrá ejecutar normalmente teniendo que especificar únicamente el vídeo o dispositivo que tomara el mismo.

8 Evidencias y Pruebas de Funcionamiento

En esta sección se centrara en tres partes principales mostrando evidencias de la captura los actores viales, la comparación de actores viales y por ultimo, los resultados del filtro de kalman y visualización del sistema de advertencia.

8.1 Identificación de Actores viales

Tal y como se estableció en el documento los actores viales se identificación en una caja la cual indicara el tipo de actor y el porcentaje de que lo sea tal y como se muestra en la siguiente imagen:



Cabe aclarar que aunque la eficiencia del algoritmo Yolo es alto existe la posibilidad de que omita alguno de los actores viales, sin embargo este puede llegar a identificar actores sobre puestos como en el caso del ciclista donde identifica tanto al usuario como a la bicicleta tal y como se ve en la siguiente imagen:

8.2 Comparación

Para el proceso de comparación se toman los dos fragmentos de imagen obteniendo los siguientes resultados

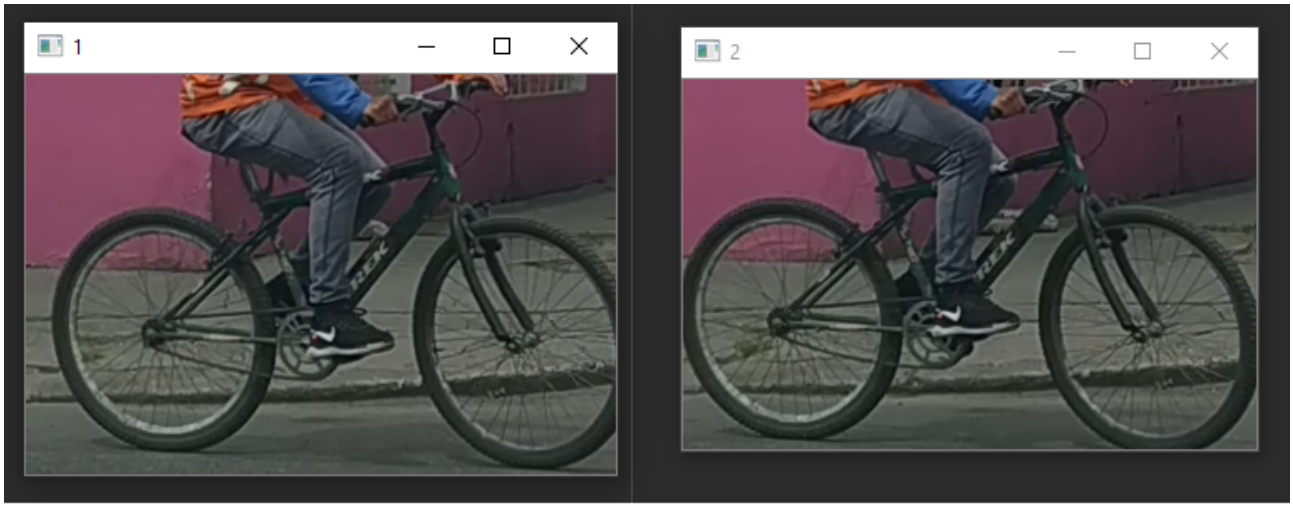


Figura 8.1: Resultado equivalencia = 0.91 , Bandera de Equivalencia = True

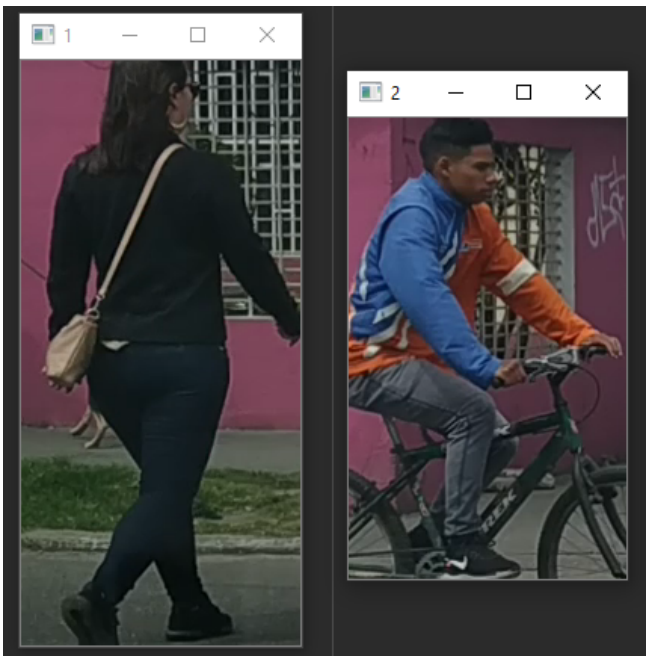


Figura 8.2: Resultado equivalencia = 0.51 , Bandera de Equivalencia = False

8.3 Sistema de Advertencia

Y por ultimo como muestra del Filtro de Kalman y el sistema de advertencia:

Figura 8.3: Zona Segura = 25%, Estado = Normal

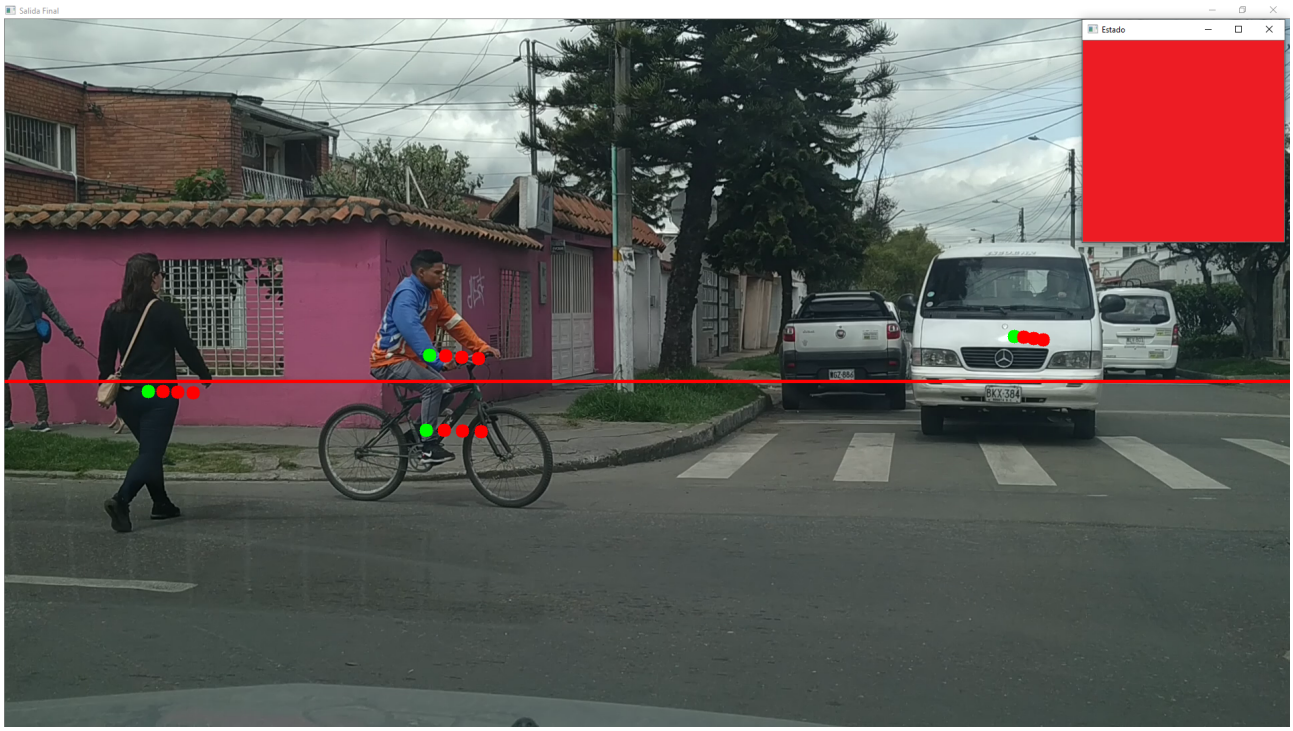


Figura 8.4: Zona Segura = 50%, Estado = Advertencia

En donde los puntos rojos son las predicciones del filtro de Kalman, la linea roja el limite de la zona segura y el cuadro en la parte superior el sistema que simula advertencia.

Referencias

[1] Agencia Nacional de Seguridad Vial y Observatorio Nacional de Seguridad Vial, *FALLECIDOS, LESIONADOS Y ACCIDENTES EN HECHOS DE TRANSITO - COLOMBIA 2016-2017P.* Serie: BTE 2018001; febrero, 2018.

[2] Agencia Nacional de Seguridad Vial y Observatorio Nacional de Seguridad Vial, *https://ansv.gov.co/observatorio/?op=Contenidos&sec=59&page=54* Noviembre, 2018.

[3] *https://quantdare.com/filtro-kalman/* Marzo, 2014.

[4] *https://la.mathworks.com/discovery/filtros-kalman.html.*

[5] *https://www.toyota.com/espanol/prius/features/mpg/1221/1223/1224* Enero, 2019.

[6] *https://sandipanweb.wordpress.com/2018/03/11/autonomous-driving-car-detection-with-yolo-in-python/*

[7] G. Welch and G. Bishop. *An Introduction to the Kalman Filter* University of North Carolina, Department of Computer Science, TR 95-041. 1995.