



Universidad
Rey Juan Carlos

VISIÓN ARTIFICIAL



PRÁCTICA:

DETECCIÓN DE SEÑALES DE TRÁFICO

Tontong Xu.

Eva María Hoyo De La Cruz.

Antonio Francisco Roldán Martín.

Grado en Ingeniería de Computadores.
Doble Grado Ingeniería Computadores/Informática.

TABLA DE CONTENIDOS:**1. INTRODUCCIÓN:****1.1 MOTIVACIÓN DEL TRABAJO.****1.2 OBJETIVOS DE LA PRÁCTICA.****2 TRABAJO DE LA PARTE CORRESPONDIENTE:****2.1 DESCRIPCIÓN DEL ALGORITMO A IMPLEMENTAR.****2.2 DESARROLLO DE LOS ALGORITMOS.****2.3 ESTADÍSTICAS, PRUEBAS Y RESULTADO.****3 CONCLUSIONES EXTRAÍDAS.****4 COMENTARIOS PERSONALES.****5 REFERENCIAS UTILIZADAS.****6 MATERIAL ADICIONAL DE LA PRÁCTICA.**

1. INTRODUCCIÓN.

No cabe ninguna duda de que el siglo XXI se está convirtiendo en la época dorada en cuanto a avance tecnológico se refiere. Dicho avance provoca que los retos tecnológicos sean cada día más ambiciosos y más factibles de alcanzar.

Un claro ejemplo de ello es la evolución de la telefonía móvil, en donde hace unos años era prácticamente impensable el poder disponer de internet en el mismo, poder enviar y recibir correos, incluso llegar a retransmitir un evento en directo sin necesidad de grandes equipos auxiliares.

La introducción de la electrónica en los automóviles fue un paso muy importante, no solo para la eficiencia y la fiabilidad de las distintas partes del vehículo, sino también para la seguridad vial.

Hasta la década de los 60 los vehículos convencionales estaban formados básicamente por partes mecánicas. A partir de esta década, los fabricantes de automóviles comenzaron a investigar y añadir partes electrónicas que, en un principio y debido a su alto coste, solo se incluían en automóviles de alta gama.

Uno de los principales sistemas implementados en los coches fue el ABS (Sistema antibloqueo de ruedas). Introducido en 1978 por Bosch, este sistema electrónico fue un gran avance en la seguridad, puesto que impedía el bloqueo de las ruedas y con ello disminuía la distancia de frenado y permitía girar el coche durante todo ese tiempo.

Asimismo, el ABS fue la base para crear posteriormente otros sistemas electrónicos como el control de tracción, también por Bosch en 1986, el cual previene el deslizamiento de los neumáticos por pérdida de adherencia al asfalto.

Otro claro ejemplo de dicho avance es la automatización de la conducción, los comúnmente llamados coches autónomos. Los coches autónomos suponen un reto bastante complejo, ya que cualquier sistema que se vaya a implementar tiene que detectar y reconocer perfectamente todas las señales de tráfico, con el principal fin de no poner en peligro la seguridad e integridad de ninguna persona o animal.

Los sistemas de visión por computador han pasado de verse como algo propio a la ciencia ficción a ser una realidad del día a día en los últimos 30 años, gracias en parte a las librerías gratuitas como las OpenCV que han facilitado el acceso a todo el mundo. Como se verá más adelante muchos de los sistemas de ayudas que hay actualmente en los automóviles son gracias a la visión por computador.

1.1 MOTIVACIÓN DEL TRABAJO.

Hace años, cuando se introdujeron los navegadores GPS, como el TomTom, además del avance en cuanto a rutas guiadas, se produjo un avance en la información que se le ofrecía al conductor en cuanto al estado del tráfico, carril por el que circular, velocidad máxima permitida en la vía, etc. Tras estos sistemas llegaron los avisos cuando se superaba dicha velocidad máxima. Pero ¿cómo sabía el navegador cual era la velocidad máxima de la vía con tal precisión? La respuesta es sencilla, por geoposicionamiento.

Sin embargo, con la búsqueda del coche autónomo, el geoposicionamiento ya no es suficiente, el coche ha de ser capaz de entender y adaptarse a las condiciones cambiantes de la vía y así surgieron los sistemas TSR de ayudas a la conducción.

En la actualidad multitud de marcas de coches luchan por ser punteras en la introducción de sistemas TSR y para ello utilizan procesamiento de imagen sobre la vía por la que se circula. Un gran número de estos sistemas ya están comercializados como software interno del coche o como software externo o adicional y que se vende por separado, al igual que lo hacían aquellos primeros navegadores GPS. Otra ayuda de visión artificial se trata de que, al cambiar de carril de manera involuntaria, llegando el volante a rectificar tu ruta, si antes del cambio de carril no has puesto el intermitente.

La motivación de esta práctica no es competir con un estado del arte ya desarrollado y afianzado en el mercado, si no lograr una primera aproximación a un algoritmo capaz de detectar un gran abanico de señales verticales de tráfico, centrándose, más concretamente, en las señales de Europa.

Dicha aproximación no busca centrarse en un modelo de cámara concreto, una posición fija dentro del coche o sólo señales de velocidad. Si no que busca una solución general a un problema que ya ha sido resuelto por algunos fabricantes de automóviles de forma concreta, la detección y posterior reconocimiento de las señales viales verticales que pueden encontrarse en la carretera.

1.2 **OBJETIVOS DE LA PRÁCTICA.**

El objetivo principal de esta práctica consiste en el diseño y desarrollo de varios sistemas de detección de señales de tráfico, para comprobar hasta qué punto puede ser fiable un sistema basado únicamente en procesamiento de imágenes. Para ello se utilizará el lenguaje Python, unido a la librería OpenCV, bajo el entorno de trabajo de PyCharm.

Para realizar esta práctica hemos dado los primeros pasos en el modo de uso de la visión artificial, muy importante en la tecnología actual, concretándolo en un sistema que además permite ayudar en la seguridad de las personas.

Asimismo, con esta práctica hemos avanzado en el lenguaje Python, un lenguaje que es cada vez más usado en el mundo hasta haber superado en la actualidad a muchos otros lenguajes conocidos internacionalmente. Entender Python y poder darle una utilidad práctica nos sirve de referencia para posibles futuros proyectos en los que sea necesario este lenguaje.

Los objetivos parciales para llegar a alcanzar dicha solución son los siguientes:

Diseño de los diferentes algoritmos propios a desarrollar:

Partiendo de lo aprendido en la teoría de la asignatura, diseñar el algoritmo de forma que pueda ser aplicado lo más genéricamente posible, esto es: variando las condiciones de luz, posición, velocidad, modelo de cámara, etc.

División y desarrollo del algoritmo en diferentes etapas:

Implementación del algoritmo diseñado en el objetivo anterior.

El algoritmo será desarrollado por fases y cada fase se irá probando y revisando de nuevo a medida que se vayan completando nuevas fases.

Evaluación de los resultados del algoritmo:

Análisis de resultados tras realización de pruebas con el objetivo de averiguar el rendimiento real de los algoritmos desarrollados.

2. TRABAJO DE LA PARTE CORRESPONDIENTE:

2.1 DESCRIPCIÓN DE LOS ALGORITMOS A IMPLEMENTAR.

Se han implementado 3 algoritmos que pasamos a detallar a continuación:

ALGORITMO DETECCION-MSER.

Con este algoritmo hemos realizado la búsqueda pedida en la práctica, lo más fiel a la misma que nos ha sido posible.

En este algoritmo se usan las siguientes clases:

- ✓ *signalMask.py:*
- ✓ *procesadoImagen.py:*
- ✓ *guardarSalida.py:*

Por último, el algoritmo principal está en la clase *deteccionMSER.py*.

En resumen, los pasos seguidos en este algoritmo son:

- Se pasa la imagen a yuv.
- Se saca el histograma ecualizado.
- Se saca el resultado de los valores anteriores.
- Se saca la imagen a hsv.
- Por último, se toma el algoritmo común para finalizar.

ALGORITMO DETECCIÓN ALTERNATIVA.

Con este algoritmo pretendíamos no usar MSER para la búsqueda de las señales de tráfico, debido a que como parte de la práctica se pedía que se buscara con MSER, por tanto, se ha realizado otra detección alternativa (sin MSER en este caso).

En este algoritmo se usan las siguientes clases:

- ✓ *signalMask.py:*
- ✓ *procesadoImagen.py:*
- ✓ *guardarSalida.py:*

Por último, el algoritmo principal está en la clase *deteccionAlternativa.py*.

En resumen, los pasos seguidos en este algoritmo son:

- Se baja el brillo
- Se filtra la imagen en hsv por tonos rojos y se saca el kernel.
- Se cierra la imagen y se sacan los bordes de esta.
- Por último, se pasa el resultado por el algoritmo común.

ALGORITMO DETECCIÓN ALTERNATIVA-MSER.

Con este último algoritmo hemos querido juntar las partes de los dos algoritmos anteriores que más nos han llamado la atención, con la finalidad de mejorar la búsqueda lo máximo posible.

En este algoritmo se usan las siguientes clases:

- ✓ *signalMask.py:*
- ✓ *procesadoImagen.py:*
- ✓ *guardarSalida.py:*

Por último, el algoritmo principal está en la clase *deteccionAlternativaconMSER.py*.

En resumen, los pasos seguidos en este algoritmo son:

- Se baja el brillo.
- Se filtra la imagen en hsv por tonos rojos.
- Se saca el kernel.
- Se cierra la imagen y se sacan los bordes de esta.
- Se pasa por MSER.
- Finalmente se pasa el resultado por el algoritmo común.

ALGORITMO COMÚN.

Estos algoritmos son aquellos que hemos usado para todas las partes, ya que nos ha parecido fundamental, una vez llegado a un punto, que las imágenes sean recortadas y procesadas individualmente.

Comparadas con las máscaras de las señales, se procesa a que señal se parece más o incluso si no puede ser considerada una señal de tráfico.

Luego es almacenada la imagen con los recuadros de las señales detectadas en la carpeta de destino correspondiente, (*resultado_imgs*), y si se ejecuta los distintos algoritmos, se irá sobrescribiendo el contenido de la misma.

Almacena en el fichero de texto la imagen de origen, los puntos que rodean la imagen, la señal con la que se corresponde y el peso de esa señal.

2.2 DESARROLLO DE LOS ALGORITMOS.

ALGORITMO COMÚN.

El algoritmo común se compone de las 3 funciones comentadas anteriormente:

✓ **signalMask.py:**

Formada por las funciones →

❖ **correlarMascara**

(utilizada en la función *recorteCorrelarSignals*)

❖ **calcularsennalcorrecta**

(utilizada en la función *correlarMascara*)

Y por las máscaras de las señales →

❖ **mascaraStop**

❖ **mascaraProhibido**

❖ **mascaraPeligro**

❖ **macaraPeligro45**

❖ **arraymaskceda**

✓ **guardarSalida.py:**

Formada por las funciones →

❖ **guardarcarpetaasyfichero**

❖ **guardarimagencarpeta**

Métodos de OpenCV utilizados en esta función:

➤ *cv2.imwrite*

✓ **procesadolmagen.py:**

Formada por las funciones →

❖ **mascararojo**

Métodos de OpenCV utilizados en esta función:

➤ *cv2.inRange*

➤ *cv2.add*

➤ *cv2.blur*

➤ *cv2.threshold*

❖ **filtradororoDifuminarNucleoCerradoCanny**

(utiizada en los 3 algoritmos, que a su vez hace uso de la función "mascararojo")

Métodos de OpenCV utilizados en esta función:

➤ *cv2.getStructuringElement*

➤ *cv2.morphologyEx*

➤ *cv2.Canny*

❖ **hacedordeMSER***(utilizada en "DETECCION MSER")**(utilizada en "DETECCION ALTERNATIVA-MSER")*

Métodos de OpenCV utilizados en esta función:

- *cv2.MSER_create*
- *mser.detectRegions*
- *cv2.fillPoly*

❖ **recorteCorrelarSignals***(utilizada en los 3 algoritmos)*

Métodos de OpenCV utilizados en esta función:

- *cv2.minAreaRect*

❖ **guardados***(utilizada en los 3 algoritmos llamada desde recorteCorrelarSignals, que a su vez hace uso de la clase "guardarSalida")*

Métodos de OpenCV utilizados en esta función:

- *cv2.resize*
- *cv2.drawContours*
- *cv2.rectangle*

ALGORITMO DETECCION-MSER.

Los pasos seguidos en este algoritmo son:

- Se mejora la imagen.
- Se pasa a niveles de gris.
- Se pasa la imagen por la función MSER.
- Finalmente se pasa el resultado por el algoritmo común (recorteCorrelarSignals)

Métodos de OpenCV utilizados en este algoritmo:

- *cv2.imread*
- *cv2.cvtColor*
- *cv2.equalizeHist*
- cv2.findContours*

Además, se hace uso de las siguientes funciones, cuyos métodos de OpenCV se describen en sus apartados correspondientes:

- ✓ ***filtradororojoDifuminarNucleoCerradoCanny(imagenhsv)***
- ✓ ***hacedordeMSER(cannybordes)***
- ✓ ***recorteCorrelarSignals(contornos, imagen.copy(), imagen, "MSER", strinentradaimg)***

ALGORITMO DETECCIÓN ALTERNATIVA.

Los pasos seguidos en este algoritmo son:

- Se filtra la imagen en hsv por tonos rojos.
- Se saca el kernel.
- Se cierra la imagen y se sacan los bordes de ésta.
- Finalmente se pasa el resultado por el algoritmo común (recorteCorrelarSignals)

Métodos de OpenCV utilizados en este algoritmo:

- *cv2.imread*
- *cv2.addWeighted*
- *cv2.cvtColor*
- *cv2.findContours*

Asimismo, usan de las siguientes funciones, cuyos métodos de OpenCV se describen en sus apartados correspondientes:

- ✓ ***filtradorojoDifuminarNucleoCerradoCanny(imagenhsv)***
- ✓ ***recorteCorrelarSignals(contornos, imagen.copy(), imagen, "Alternativa", strinentradaimg)***

ALGORITMO DETECCIÓN ALTERNATIVA MSER.

Los pasos seguidos en este algoritmo son:

- Se hace un suavizado de la imagen.
- Se filtra la imagen en hsv por tonos rojos.
- Se saca el kernel.
- Se cierra la imagen y se sacan los bordes de ésta.
- Se pasa la imagen por la función MSER y...
- Finalmente se pasa el resultado por el algoritmo común (recorteCorrelarSignals)

Métodos de OpenCV utilizados en este algoritmo:

- *cv2.imread*
- *cv2.addWeighted*
- *cv2.cvtColor*
- *cv2.findContours*

Igualmente, utilizan de las siguientes funciones, cuyos métodos de OpenCV se describen en sus apartados correspondientes:

- ✓ ***filtradorojoDifuminarNucleoCerradoCanny(imagenhsv)***
- ✓ ***hacedordeMSER(cannybordes)***

✓ *recorteCorrelarSignals(contornos, imagen.copy(), imagen, "MSER", strinentradaimg)*

2.3 ESTADÍSTICAS, PRUEBAS Y RESULTADO.

Ejemplos de pantallas correspondientes a la ejecución del programa: .







Las siguientes imágenes son bastante oscuras, aun así, nuestro algoritmo consigue detectar las señales:



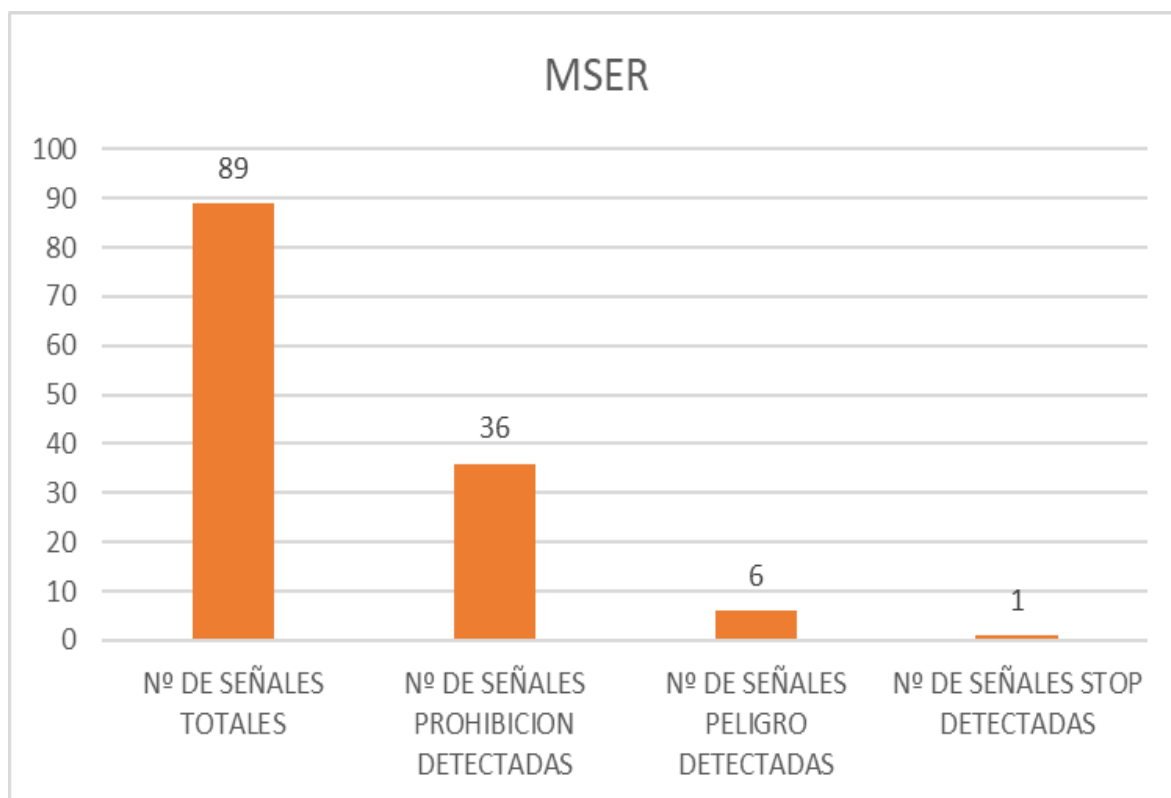




A continuación, se incluye unas estadísticas y su gráfica correspondiente al resultado de la ejecución del programa en nuestros distintos algoritmos sobre la muestra de test.

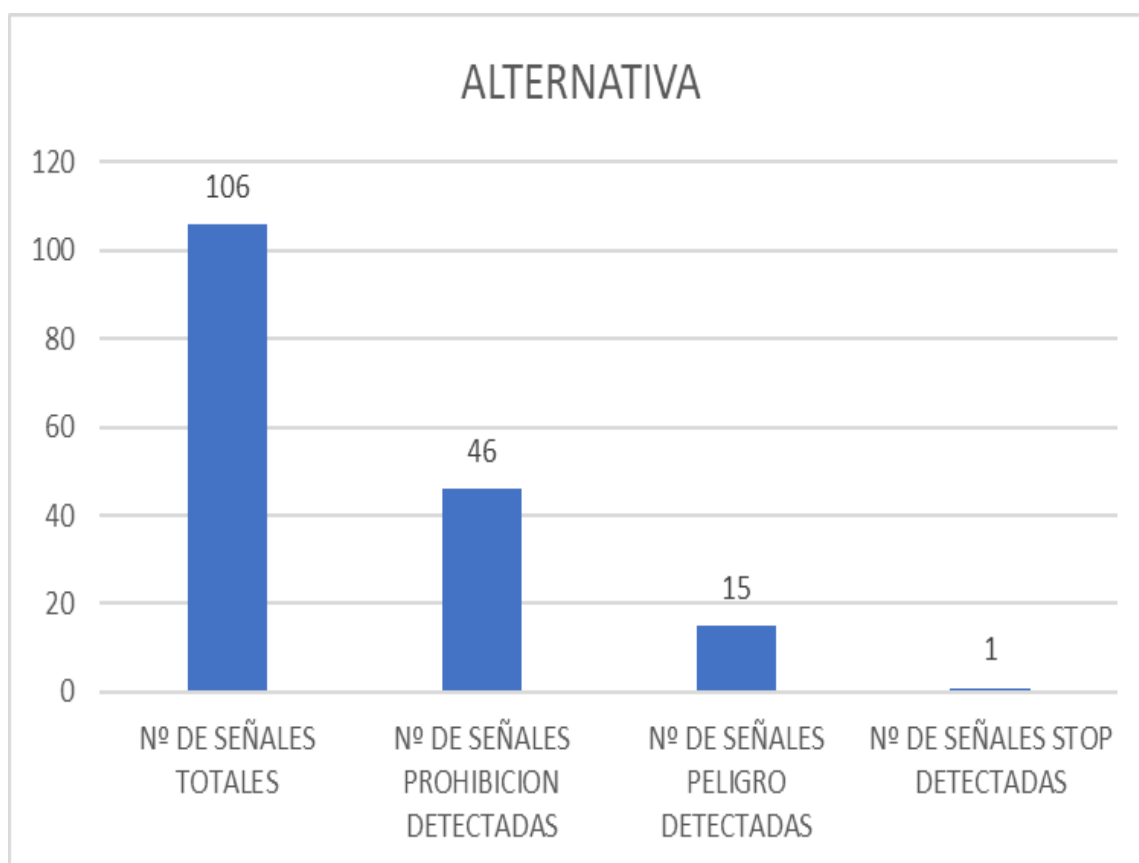
ALGORITMO DETECCION-MSER

Nº de IMÁGENES TOTALES	152
Nº DE SEÑALES TOTALES	89
Nº DE SEÑALES PROHIBICION DETECTADAS	36
Nº DE SEÑALES PELIGRO DETECTADAS	6
Nº DE SEÑALES STOP DETECTADAS	1



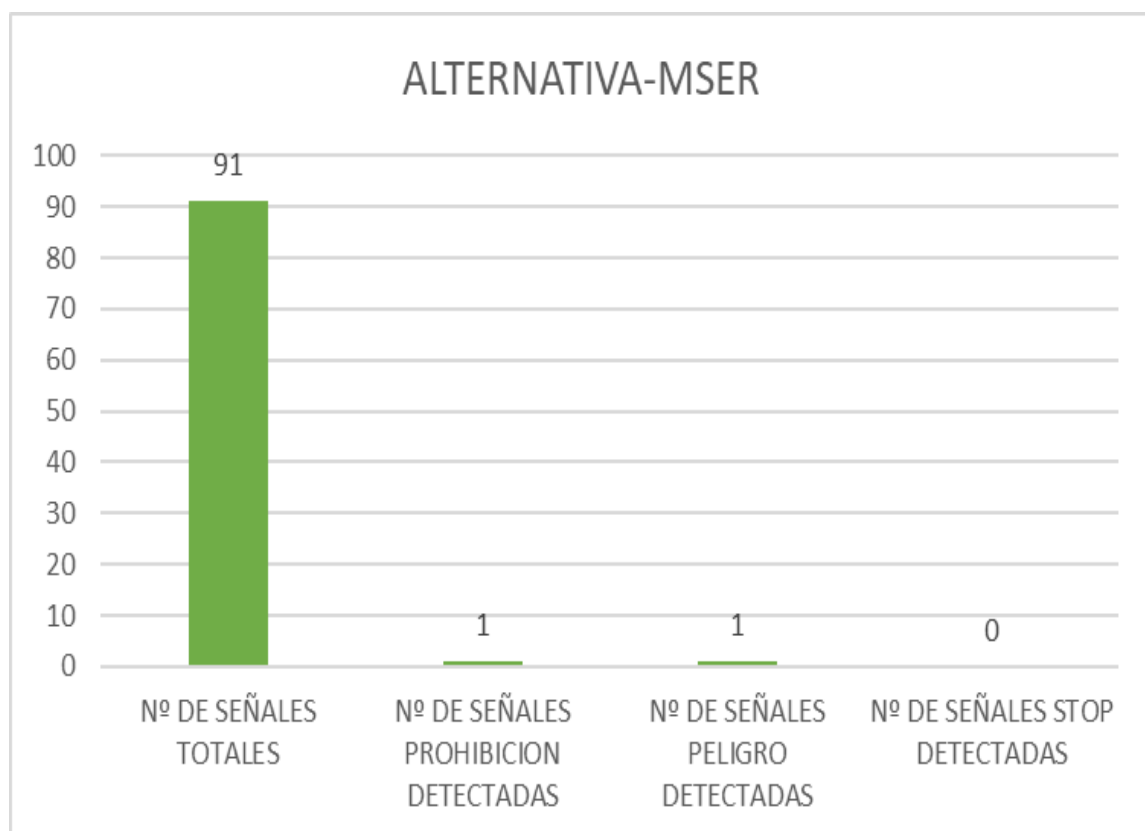
ALGORITMO DETECCIÓN ALTERNATIVA.

Nº de IMÁGENES TOTALES	182
Nº DE SEÑALES TOTALES	106
Nº DE SEÑALES PROHIBICION DETECTADAS	46
Nº DE SEÑALES PELIGRO DETECTADAS	15
Nº DE SEÑALES STOP DETECTADAS	1



ALGORITMO DETECCIÓN ALTERNATIVA MSER.

Nº de IMÁGENES TOTALES	170
Nº DE SEÑALES TOTALES	91
Nº DE SEÑALES PROHIBICION DETECTADAS	1
Nº DE SEÑALES PELIGRO DETECTADAS	1
Nº DE SEÑALES STOP DETECTADAS	0



3. CONCLUSIONES EXTRAÍDAS.

Esta práctica tenía como propósito la implementación de una primera aproximación a un algoritmo de detección de señales viales dejando a un lado la IA (Inteligencia Artificial) y el aprendizaje automático.

Como conclusiones tras el análisis de las prestaciones del sistema desarrollado podemos destacar principalmente los dos siguientes puntos:

1. El sistema no funciona muy bien en casos con imágenes de una dificultad elevada, los resultados son suficientemente buenos, con lo que no se podría implementar en un sistema real de conducción autónoma, ya que se debería de exigir que un sistema de este calibre tenga un 100% de detección y reconocimiento de las señales de tráfico, ya que está en juego la seguridad vial.

2. Existe una clara dependencia entre los resultados y las imágenes utilizadas, ya que prácticamente todas las imágenes tienen unas condiciones de detección muy desfavorables. La mala iluminación es una de estas condiciones. Unido a esto hay que tener en cuenta que no todas las señales están perfectamente cuidadas, hay señales muy deterioradas y otras muchas señales parcialmente ocultas entre la naturaleza. También hay imágenes en donde las señales de tráfico están “demasiado” lejanas como para que puedan ser correctamente detectadas.

Además, el diseño del algoritmo permite, con una pequeña modificación, abarcar aún más señales, ya que únicamente han de añadirse colores a la búsqueda.

Por tanto, con un coste muy bajo se podría instalar en cualquier vehículo un sistema de detección de señales de tráfico con una alta tasa de éxito. Esta tecnología permitiría no sólo evitar multas de tráfico sino disminuir el número de accidentes de tráfico.

En vista de los resultados de las estadísticas nuestra elección de algoritmo estaría entre detecciónMSER o detecciónAlternativa, ya que son los que mejores resultados dan. El hecho de haber realizado la práctica de tres formas distintas nos ha dado un amplio conocimiento en el reconocimiento y modificación de las imágenes.

4. COMENTARIOS PERSONALES.

La realización de esta práctica nos ha supuesto un reto al que enfrentarse y muchas horas de frustración delante de la pantalla buscando porque el programa no compilaba.




Por otra parte, nos ha servido para adquirir muchos conocimientos sobre visión por ordenador, sistemas inteligentes en los vehículos.... y también y muy importante como se lleva a cabo una práctica de estas características, las distintas fases que se van siguiendo (planificación, desarrollo, pruebas...) y la forma en la que se trabaja en ella.

La realización de esta práctica ha sido para nosotros un reto que nos ha servido para adquirir muchos conocimientos sobre sobre visión por ordenador y sistemas inteligentes en los vehículos. Nos ha supuesto también muchas horas de frustración frente la pantalla buscando soluciones cuando el programa no compilaba. Pero hemos aprendido como se lleva a cabo las distintas fases de su evolución (planificación, desarrollo, pruebas...) y la forma en la que se trabaja en él, y todo esto es muy importante.

5. REFERENCIAS UTILIZADAS.

- ✓ PYTHON. The Python tutorial. <https://docs.python.org/3/tutorial/>
- ✓ ESCUELA DE PYTHON. PyCharm: uno de los mejores IDE para Python. <https://www.escuelapython.com/pycharm-uno-de-los-mejores-ide-para-python/>
- ✓ WIKIPEDIA. <https://www.wikipedia.org/>
- ✓ OpenCV, «Camera Calibration and 3D Reconstruction».
- ✓ Detector de Regiones de Interés MSER:
<http://stackoverflow.com/questions/17647500/exact-meaning-of-the-parameters-given-toinitialize-mser-in-opencv-2-4-x>
- ✓ Ejemplo de uso de MSER:
<https://github.com/opencv/opencv/blob/master/samples/python/mser.py>

6. MATERIAL ADICIONAL DE LA PRÁCTICA.

-  Código fuente de la práctica.
-  Ejecutable de la práctica.
-  Memoria explicativa