_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Desarrollo de un algoritmo de visión artificial para el guiado automático de un robot móvil

Autor: Ignacio Torres Guaita

Tutor: Antonio José Albiol Colomer

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2016-17

Valencia, 3 de julio de 2017

Resumen

El avance tecnológico del sector automovilístico está orientado hacia los coches autómatas, habiendo diversas empresas que experimentan con múltiples sistemas y/o algoritmos novedosos con este objetivo.

En este trabajo se presenta un nuevo algoritmo para la detección y seguimiento de un camino formado por una única línea continua o un carril mediante un robot del tipo uniciclo (GoPiGo). El tratamiento de imagen consiste en el residuo de un doble filtrado morfológico adaptativo a la altura de la imagen donde se aplica, realizando a continuación una segmentación y discriminación de objetos por sus características básicas a partir del cálculo de momentos y de un tracking temporal/espacial. El seguimiento del camino se realiza utilizando el modelo cinemático del uniciclo, habiendo obtenido previamente su estado actual y calculando su velocidad angular como un control *PID* de la diferencia de la orientación del uniciclo respecto a la línea y la orientación deseada. Los resultados experimentales de la ejecución de estos algoritmos sobre el uniciclo demuestran su funcionamiento, incluso en suelos reflectantes o de pintura irregular, coincidiendo perfectamente los resultados teóricos con los prácticos. Además se comentan las limitaciones del uniciclo utilizado, los márgenes de mejora y el alcance de uso del propio algoritmo.

Resum

L'avanç tecnològic del sector automobilístic està orientat cap als cotxes autòmats, havent-hi diverses empreses que experimenten amb múltiples sistemes i/o algoritmes nous amb este objectiu.

En este treball es presenta un nou algoritme per a la detecció i seguiment d'un camí format per una única línia contínua o un carril per mitjà d'un robot del tipus unicicle (GoPiGo). El tractament d'imatge consistix en el residu d'un doble filtrat morfològic adaptatiu a l'altura de la imatge on s'aplica, realitzant a continuació una segmentació i discriminació d'objectes per les seues característiques bàsiques a partir del càlcul de moments i d'un tracking temporal/espacial. El seguiment del camí es realitza utilitzant el model cinemàtic de l'unicicle, havent obtingut prèviament el seu estat actual i calculant la seua velocitat angular com un control PID de la diferència de l'orientació de l'unicicle respecte a la línia i l'orientació desitjada. Els resultats experimentals de l'execució d'estos algoritmes sobre l'unicicle demostren el seu funcionament, inclús en sòls reflectants o amb pintura irregular, coincidint perfectament els resultats teòrics amb els pràctics. A més es comenten les limitacions de l'unicicle utilitzat, els marges de millora i l'abast d'ús del propi algoritme.

Abstract

The technological advance of the automotive sector is oriented towards the automated cars, having several companies that experiment with multiple systems and novel algorithms with this objective.

This work presents a new algorithm for the detection and tracking of a path formed by a single solid line or a lane by a unicycle robot (GoPiGo). The image treatment consists on the residue of a double morphological filter adaptable to the height of the image where it is applied, making a segmentation and a discrimination of objects by their basic characteristics from the calculation of moments and a temporal/spatial tracking. Path tracking is performed using the kinematic model of the unicycle, by obtaining previously its current state and calculating its angular velocity as a PID control of the difference of the orientation of the unicycle respect to the line and the desired orientation. The experimental results of the execution of these algorithms on the unicycle show a correct behavior, even in reflective floors or with irregular paint, matching the theoretical results with the practical ones. In addition, the limitations of the unicycle used, the margins of improvement and the scope of use of the algorithm itself are discussed in this work.

Índice

Capítulo 1. Introducción.

Con la evolución de la tecnología se ha conseguido automatizar procesos muy diferentes y variopintos, facilitando la realización de trabajos o de tareas cotidianas. Desde máquinas que repiten el mismo trabajo en cadenas de montaje en un entorno cerrado y conocido, hasta automóviles que funcionan de manera autómata sin necesidad de un conductor en un entorno completamente libre y desconocido.

En los últimos años se está viendo una importante evolución en el ámbito de los coches autómatas, los cuales son capaces de reconocer el entorno en todos los sentidos: detectar señales de tráfico, detectar cruces entre vías, reconocer peatones y otro tipo de obstáculos que pueden entorpecer la marcha e incluso detectar y realizar un seguimiento de carriles, y todo ello de manera automática.

Varias compañías, como Google [1] y Udacity [2], llevan tiempo trabajando y perfeccionando el desarrollo de este tipo de vehículos, ya que se considera una herramienta útil tanto en el aumento de la comodidad del ciudadano como en la reducción de los accidentes de tráfico. Esto se traduce en una mejora significativa de la seguridad vial que permite el salvamento de vidas.

Por otro lado, es necesario considerar que un error producido por un sistema automatizado generaría daños considerablemente mayores que los de un accidente causado por un error humano. Por esta razón, dichos sistemas deben tener un margen de error nulo, garantizando a los ciudadanos la seguridad que necesitan para confiar en estos nuevos productos.

Otro factor importante a tener en cuenta es la adaptación del entorno a las condiciones requeridas para la correcta ejecución del algoritmo de automatización, lo cual incluye: líneas de carretera bien pintadas, ausencia de obstáculos, señales y vía en buen estado, etc.

Dado que todavía existen muchas mejoras por realizar, trabajos que innoven en el desarrollo de algoritmos para llevar a cabo la aplicación de un programa sin fallos son imprescindibles.

Capítulo 2. Objetivos del trabajo.

El objetivo del presente trabajo es la realización de un algoritmo de visión artificial para el guiado automático de un robot móvil. Para ello el trabajo se divide en la parte del procesamiento de la imagen a la que se le dará más importancia y una parte de robótica utilizada para aplicar el diseño de este algoritmo en tiempo real sobre un uniciclo del fabricante GoPiGo [3].

El algoritmo de procesado de imagen deberá ser capaz de detectar una línea recta en la imagen captada por cámara con el propósito de ajustar a continuación la velocidad del uniciclo para poder seguirla de manera estable. En un principio el seguimiento es realizado sobre líneas rectas, pero se intentará realizar también el seguimiento sobre curvas para observar los resultados que esto desprende.

Como siguiente paso se intentará realizar un segundo algoritmo para la detección y seguimiento de un carril (dos líneas paralelas), de manera que el uniciclo sea capaz de ir por el centro de este. Al igual que el anterior se probará en curvas para observar los resultados experimentales de la ejecución del algoritmo en tiempo real sobre el uniciclo.

Por tanto los objetivos principales de manera secuencial son los siguientes:

- Búsqueda bibliográfica, obtención y definición de un modelo de control a aplicar sobre el uniciclo en cuestión (GoPiGo) para el seguimiento de una trayectoria dada.

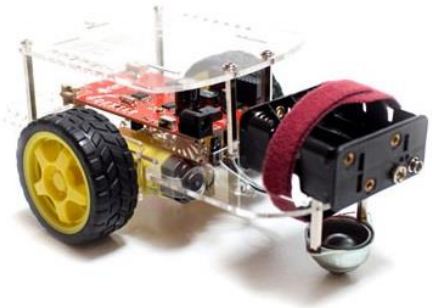


Fig. 1. Robot uniciclo GoPiGo (Dexter Industries).

- Desarrollo de un algoritmo de tratamiento de imagen innovador para la detección de una línea. Esta línea como puede verse en la Fig. 2. se trata de una línea continua sobre una superficie plana, que puede ser tanto blanca como negra y tanto recta como curva.

Fig. 2. (a) Línea recta y negra. (b) Línea curva y blanca.

- Desarrollo de un algoritmo de tratamiento de imagen para la detección de un carril (dos líneas). De la misma manera que el anterior, pero en este caso el uniciclo deberá seguir la trayectoria marcada por el medio del carril, independientemente del ancho de este.

Fig. 3. (a) Carril recto y negro. (b) Carril curvo y blanco.

- Además, se comentará finalmente si este algoritmo puede ser capaz de aplicarse en un entorno más real, es decir, en carreteras convencionales y/o autopistas.

Pero primero debe tenerse en cuenta que las imágenes y pruebas realizadas a lo largo del trabajo se han realizado en varios suelos con características bastante malas para la detección de las líneas:

- Es muy reflectante, esto hace que aparezcan reflejos indeseados que pueden llegar a afectar en los resultados, pero que a pesar de ello se intentarán filtrar.
- La diferencia de nivel de luminancia entre el suelo y la línea no es tan alto como en una carretera normal (blanco-negro) sino que es una diferencia más leve (gris-negro).
- En el caso del primer suelo (Fig. 2.), está formado por pequeños fragmentos de colores oscuros y claros que pueden llegar a entrar en el filtrado. Aunque esto también puede ocurrir en cualquier otro suelo puntualmente, en este ocurre sin cesar.

Capítulo 3. Metodología.

Principalmente las referencias bibliográficas se tratan de fuentes secundarias (tesis, proyectos de fin de estudios y artículos basados en fuentes primarias), aunque también se han utilizado varias fuentes primarias (Artículos de revistas científicas y libros de teoría).

3.1 Esquema temporal de tareas.

Al comienzo de la realización del trabajo se dividieron las tareas de manera temporal en todo el tiempo disponible, de Enero a Junio, de la siguiente manera:

N° Tarea	Inicio	Final	Enero	Febrero - Semana 1	Febrero - Semana 2	Febrero - Semana 3	Febrero - Semana 4	Marzo	Abril	Mayo - Semana 1	Mayo - Semana 2	Mayo - Semana 3	Mayo - Semana 4	Junio - Semana 1	Junio - Semana 2	Junio - Semana 3	Junio - Semana 4
Tarea 1	Enero	Enero															
Tarea 2	Febrero Semana 1	Febrero Semana 2															
Tarea 3	Febrero Semana 3	Febrero Semana 4															
Tarea 4	Marzo	Marzo															
Tarea 5	Abril	Abril															
Tarea 6	Mayo Semana 1	Mayo Semana 2															
Tarea 7	Mayo Semana 3	Mayo Semana 4															
Tarea 8	Junio Semana 1	Junio Semana 3															
Tarea 9	Junio Semana 3	Junio Semana 4															

Tabla 1. Distribución temporal de tareas.

- Actividad 1. Realización del curso de robots móviles de Coursera [4] para la comprensión y elección del modelo de control empleado por el uniciclo en el seguimiento de la línea/carril, además de la búsqueda bibliográfica de contenidos relacionados con el tema.

- Actividad 2. Planteamiento del problema respecto al tratamiento de imagen y búsqueda de la mejor solución.

- Actividad 3. Realización del algoritmo de calibración.

- Actividad 4. Realización del algoritmo de seguimiento de una línea continua.

- Actividad 5. Realización del algoritmo de seguimiento de un carril continuo.

- Actividad 6. Búsqueda de defectos y/o fallos, y pulir programas y funciones implementadas.

- Actividad 7. Experimentación y puesta en marcha de todos los algoritmos sobre el uniciclo GoPiGo.

- Actividad 8. Redacción del trabajo escrito.

- Actividad 9. Preparación de la presentación.

3.2 Recursos y software empleado.

El sistema operativo en el que se ha desarrollado la programación es '*Linux*', se trata de software de código abierto. Este sistema proporciona al desarrollador un amplio abanico de recursos y programas a utilizar [5].

Como lenguaje de programación de los tres algoritmos principales (calibración, una línea y carril) se ha utilizado '*C++*'. Como entorno de desarrollo se utilizó '*Eclipse*', de código abierto y multiplataforma [6] [7].

Fig. 4. Entorno. (a) Linux. (b) C++. (c) Eclipse.

Para el control de las funciones del uniciclo GoPiGo se ha utilizado la librería en el lenguaje '*C*' proporcionada por el mismo fabricante (*Dexter Industries*): '*gopigo.c*' junto su archivo de cabeceras de funciones '*gopigo.h*' [8]. En ella además de las propias funciones se explica la funcionalidad y el uso que debe hacerse de ellas.

Para el tratamiento de imagen se han empleado dos librerías de código abierto: '*OpenCV*' [9] y '*CImg*' [10]. Para ello se ha utilizado la documentación proporcionada por ambas librerías, donde explican y detallan como usar cada una de las funciones que proporcionan.

También se han empleado librerías dadas en la asignatura Tratamiento Digital de Imágenes del tercer curso del grado. Además de estas también se ha empleado la librería '*segment_util.cpp*',

junto con el archivo de cabeceras, útil para la segmentación y cálculo de momentos empleando la librería '*CImg*'.

También se ha utilizado '*Matlab*' en varios apartados con el objetivo de graficar y visualizar diferentes comportamientos, además de realizar el simulador teórico expuesto en los resultados del trabajo [11].

Fig. 5. Librerías para el tratamiento de imagen. (a) CImg. (b) OpenCV.

3.3 El uniciclo GoPiGo.

Conviene resumir brevemente que tipo de sensores tiene el uniciclo utilizado GoPiGo (Fig. 1.), así como explicar el funcionamiento básico de este y de las funciones más relevantes disponibles en la librería dada por el fabricante '*gopigo.c*' hayan sido utilizadas o no en el proyecto, ya que aunque no se hayan llegado a utilizar en el algoritmo realizado, han sido estudiadas y manipuladas.

3.3.1 Sensores y captación del medio.

Como sensores internos (integrados en la estructura mecánica del robot) se dispone de un '*wheel encoders'* en cada rueda, en el GoPiGo hay 18 cuentas (ticks) para cada rotación completa de la rueda, esto nos permitirá obtener medidas aproximadas de la posición y velocidad que tiene el uniciclo.

Como sensores externos (dan información del entorno del robot) en este proyecto se ha utilizado principalmente una cámara web utilizada para conocer el estado del robot en cada momento. Aunque además dispone de otros sensores como el sensor de ultrasonidos, que es capaz de medir la distancia al obstáculo más cercano enfrente suya, pudiendo obtener diferentes medidas de este, y por ejemplo obtener a qué velocidad se aproxima el GoPiGo a un obstáculo realizando la derivada de esta distancia en varios instantes de tiempo.

Dispone de un servo móvil rotatorio al que se le puede adaptar la cámara o el sensor de ultrasonidos, de manera que pueda obtenerse información del entorno en una dirección deseada. Este dispositivo no ha sido utilizado pero se menciona ya que se ha barajado su utilización en el proyecto y se ha manipulado haciendo uso de la librería dada.

También dispone de dos leds, uno a cada lado, que pueden controlarse (apagado/encendido) con las funciones dadas por el fabricante, expuestas junto con las demás en el siguiente apartado.

Fig. 6. Dispositivos del GoPiGo. (a) Cámara. (b) Sensor de ultrasonidos. (c) Servo móvil.

3.3.2 Funciones destacadas de la librería 'gopigo.c'.

Estas funciones conviene leerlas tras haber leído la introducción teórica, en concreto la parte del robot uniciclo. (Nota: en negrita las funciones utilizadas en el alguna parte del trabajo).

`float volt()`: Devuelve el voltaje que se le está aplicando al GoPiGo.

`int stop()`: Para los motores del GoPiGo.

`void pi_sleep(int t)`: "Duerme" '*t*' milisegundos la ejecución del programa.

`int motor1(int direction,int speed)`: controla el motor de la rueda derecha en una dirección ('*0*' para moverse hacia atrás y '*1*' para moverse hacia delante) y a una velocidad en bits dada en valores entre [*0,255*] que tomará el GoPiGo, donde '*0*' significa que no aplica ninguna velocidad y '*255*' es la velocidad máxima aplicable. En cualquiera de las demás funciones de velocidad se utiliza este rango de valores.

`int motor2(int direction,int speed)`: ídem para el motor de la rueda izquierda.

`int fwd()`: mueve hacia delante el GoPiGo en una distancia de entrada dada.

`int bwd()`: ídem hacia detrás.

`int left()`: gira hacia la izquierda encendiendo un motor y apagando el otro, puede utilizarse para rotar una cierta cantidad de grados.

`int right()`: ídem hacia la derecho.

`int increase_speed()`: incrementa la velocidad de ambos motores por igual en un valor dado.

`int decrease_speed()`: ídem decrementando la velocidad.

`int set_right_speed(int speed)`: fija la velocidad del motor derecho del GoPiGo.

`int set_left_speed(int speed)`: ídem para el motor izquierdo.

`void read_motor_speed(unsigned char* speed)`: devuelve la velocidad del motor seleccionado.

`int enc_read(int motor)`: '*wheel encoder*' que devuelve el número total de '*ticks*' que ha contado desde que las ruedas se movieron por primera vez, para el motor seleccionado.

`int us_dist(int pin)`: mide la distancia en centímetros a la que se encuentra el obstáculo situado delante del sensor de ultrasonidos conectado en un pin dado.

`int led_on(int l_id)`: enciende el led izquierdo con '1' o el derecho con '0'.

`int led_off(int l_id)`: ídem apagando el led.

`int servo(int position)`: fija la posición del servo dado el ángulo de apuntamiento.

3.3.3 Funcionamiento básico.

El GoPiGo está formado por dos placas que trabajan en paralelo y comunicándose entre sí conectadas entre ellas por un puerto serie.

- Raspberry Pi: esta placa es el cerebro principal del robot, en esta se ejecutan los algoritmos que realiza el desarrollador. Tiene varios puertos destinados para diferentes usos (cámara, 4 puertos USB, 1 puerto HDMI, 1 puerto Ethernet, 1 lector de tarjetas Micro-SD y una entrada para la alimentación continua). Esta placa también sirve como interfaz para el control de los sensores de ultrasonidos, motores y servo que son controlados por la otra placa, de manera que está manda instrucciones por el puerto serie hacia la otra placa indicando por ejemplo, la velocidad deseada que debe tomar cada uno de los dos motores.

- GoPiGo 2: esta placa es la encargada de aplicar la velocidad a los motores, la activación del sensor de ultrasonidos o el movimiento del servo, dependiendo de las instrucciones recibidas desde la Raspberry Pi. Esta placa es similar a las placas arduino ya que dada una orden, realiza una serie de procesos de escritura determinados sobre los dispositivos que es capaz de controlar. Estos procesos se definen en el *firmware* utilizado por la placa, que en este caso es la versión 16, pudiéndose ver en el archivo '.*ino*' del software proporcionado por el fabricante.

Fig. 7. Placas del GoPiGo. (a) Raspberry-Pi. (b) GoPiGo 2 (Arduino).

Adicionalmente debe comentarse que la computadora del GoPiGo se controlará de manera remota a través de VNC (*Virtual Network Computing*). Esta estructura cliente-servidor nos permite conectarnos a un servidor (GoPiGo) a través de nuestro ordenador (cliente), en el que podrá visualizarse virtualmente la interfaz gráfica (compartición de pantalla) del GoPiGo. Para ello simplemente se conectarán ambas computadoras a una misma red, Wi-Fi en este caso, y mediante la asignación de una contraseña VNC para que el GoPiGo acepte este control remoto por parte del cliente. VNC nos permite conectarnos de un sistema operativo a cualquier otro, en este caso nos hemos conectado desde Windows (cliente) a Ubuntu (servidor). Todo ello nos permitirá modificar cualquier archivo del cliente y ejecutar el programa de seguimiento de línea desde su terminal.

Capítulo 4. Introducción teórica.

4.1 El robot uniciclo.

Este tipo de robot suele ser el más utilizado por los investigadores debido a que su cinemática sencilla permite probar y experimentar con nuevos algoritmos de control (de seguimiento, para esquivar obstáculos, con rutas predeterminadas, etc.). Su estructura es muy sencilla ya que únicamente consta de dos ruedas fijas con control de velocidad independiente (v_L: velocidad lineal de rueda izquierda, y v_R: velocidad lineal de rueda derecha), estas ruedas se sitúan sobre el eje de giro del uniciclo. Además posee una rueda libre sin motor (en algunos artículos llamada rueda "loca") situada en el eje de simetría del robot, proporcionando a este estabilidad y apoyo a la hora de realizar cualquier tipo de giros. Esta rueda libre puede ser tanto una rueda móvil o una pata con una esfera en el extremo de manera que permite el giro en cualquier dirección (este es el caso del robot utilizado, GoPiGo).

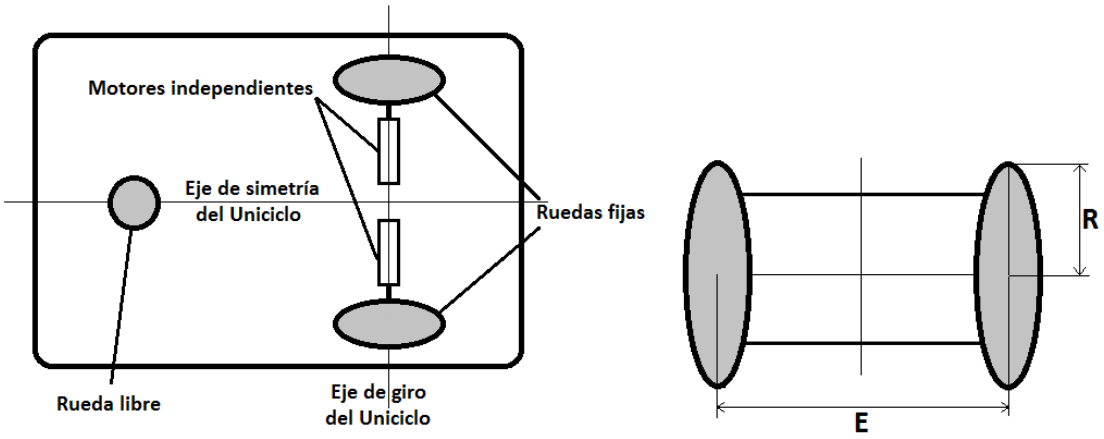


Fig. 8. Esquema básico del uniciclo. (a) Estructura vista en planta. (b) Vista lateral del uniciclo.

4.1.1 Modelo cinemático.

Los modelos cinemáticos para robots móviles se utilizan únicamente cuando el robot en cuestión realiza pruebas con poca velocidad y con poco peso en relación al tamaño de su estructura. En otro caso debería comenzar a tenerse en cuenta las diferentes fuerzas exteriores (modelo dinámico): peso, reacción al suelo, rozamiento, etc.

En este modelo se dimensiona el espacio en el que se moverá el uniciclo utilizando 3 variables: la posición en el eje '*x*', la posición en el eje '*y*' y el ángulo en el que apunta la parte delantera del uniciclo '*θ*'. La posición (*x,y*) coincide con el centro instantáneo de rotación de nuestro uniciclo. De manera que se tendrá un estado con estas tres variables, gráficamente puede observarse en la Fig. 9.

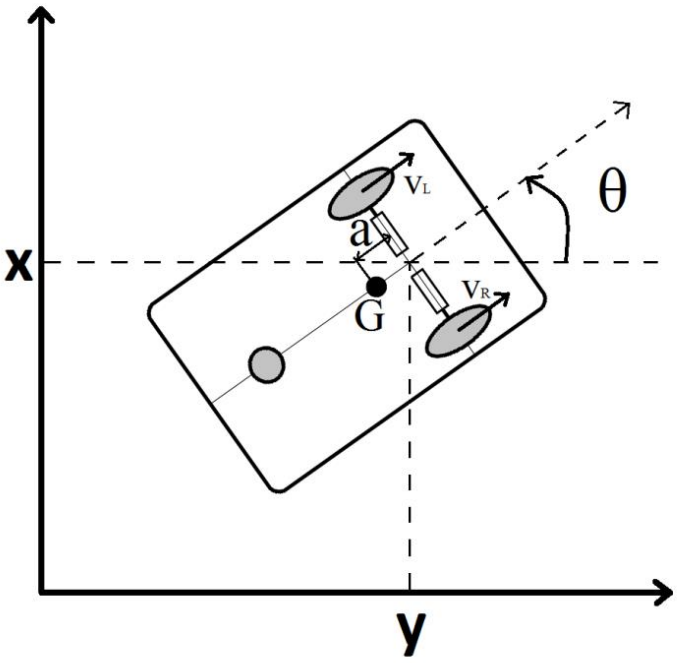


Fig. 9. Estado de uniciclo: (*x,y*) – *θ*, centro de masas '*G*' y velocidad de cada rueda: 'v_L' y 'v_R'.

Principalmente existen dos modelos que modelan la cinemática en los robots uniciclo: cuando el centro de masas del robot coincide con el centro del eje que une las dos ruedas (eje de giro del robot) y cuando este se sitúa en otra posición dentro del eje de simetría del uniciclo. Aunque realmente, el primero es un caso particular del segundo, de manera que se mostrará el segundo caso y al particularizarse dará como resultado el primer caso [12].

Cuando el centro de masas '*G*' (Fig. 9.), se encuentra a una distancia '*a*' del centro de rotación instantáneo el modelo cinemático es el siguiente, donde '*v*' es la velocidad lineal y '*w*' la velocidad angular del uniciclo:

$$\begin{cases} \dot{x} = v \cdot cos(\theta) - a \cdot w \cdot sen(\theta) \\ \dot{y} = v \cdot sen(\theta) + a \cdot w \cdot \cos(\theta) \\ \dot{\theta} = w \end{cases} \quad (1)$$

En el caso de que el centro de masas '*G*' coincida con el centro instantáneo de rotación (IGR), es decir, $a = 0$, el sistema de (1) se simplifica en:

$$\begin{cases} \dot{x} = v \cdot cos(\theta) \\ \dot{y} = v \cdot sen(\theta) \\ \dot{\theta} = w \end{cases} \quad (2)$$

Por otro lado debe conseguirse "mapear" la velocidad lineal 'v' y la velocidad angular 'w' a la velocidad de cada una de las dos ruedas: 'v_L' y 'v_R'. La velocidad lineal y angular dependen de la velocidad de las ruedas de la siguiente manera, donde 'E' es la envergadura del uniciclo (distancia entre ruedas, verse Fig. 8. (b).).

$$v = \frac{v_L + v_R}{2} \qquad w = \frac{v_L - v_R}{E} \tag{3}$$

De manera que la velocidad de cada una de las ruedas, puede obtenerse de (3):

$$v_L = \frac{2 \cdot v + E \cdot w}{2} \qquad v_R = \frac{2 \cdot v - E \cdot w}{2} \tag{4}$$

En resumen, el movimiento general del uniciclo queda modelado por el estado [x, y, θ] y las entradas [v, w] que finalmente se mapearán en [v_L, v_R].

4.1.2 Odometría y 'encoders'.

La odometría es una técnica que tiene como objeto estimar la posición y orientación de un vehículo a partir de la cantidad de vueltas dadas por sus ruedas. En los robots, la manera de saber el número de vueltas dadas, es utilizando '*wheel encoders*'. Los '*wheel encoders*' se tratan de discos ranurados a los que se le aplica un haz de luz constante de manera que gracias a las ranuras, este haz de luz es capaz de pasar periódicamente a través del disco y es captado al otro lado del disco por un foto detector, que cuenta el número de veces que ha "impactado" este haz de luz sobre él. De esta manera, teniendo un '*wheel encoder*' en cada una de las dos ruedas, sabiendo el número de ranuras o '*ticks*' que tiene el disco por cada vuelta y conociendo las dimensiones del robot (radio de sus ruedas y envergadura), podremos conocer el desplazamiento que ha realizado el robot en cuestión y saber dónde se encuentra respecto al comienzo del movimiento realizado [13].

$$\begin{cases} \dot{x} = x_0 + L_C \cdot cos(\theta) \\ \dot{y} = y_0 + L_C \cdot sen(\theta) \\ \dot{\theta} = \theta_0 + \dfrac{L_R - L_L}{E} \end{cases} \qquad donde\ L_C = \frac{L_L + L_R}{2} \tag{5}$$

Donde 'L_C' es el desplazamiento realizado por el centro del robot (IGR), 'L_L' es el desplazamiento de la rueda izquierda y 'L_R' es el desplazamiento de la rueda derecha.

La principal desventaja de la odometría es que no es exacta ya que tiende a la acumulación de errores propios del robot (medidas de sus dimensiones no completamente exactas) y los causados con las aleatoriedad del medio en el que se está moviendo (deslizamiento de las ruedas en medios resbaladizos, pequeños baches o hoyos que producen botes en el robot, etc.). Debe tenerse en cuenta que la odometría supone un contacto continuo de las ruedas en el suelo, ya que de otra forma, se captaría un desplazamiento que realmente no se ha realizado.

Por otro lado, la principal ventaja de la odometría es su simplicidad y su bajo coste. Además de la capacidad de proveer al robot una estimación de su posición si necesidad de receptores externos a este [14].

4.2 Control de robots móviles.

4.2.1 Diferentes tipos de control.

En el mundo de la robótica existen diversos tipos de control sobre los actuadores del robot en cuestión, utilizar uno u otro depende del tipo de aplicación que se esté desarrollando o del objetivo final que desee conseguirse. A continuación se exponen los tipos más frecuentes de control que suelen darse en los robots móviles, de entre los que se escogerá el que más se adapte al algoritmo que se desea desarrollar.

- '*Go-to-goal*': este modelo presenta la manera de ir de una posición origen a una posición destino, siguiendo el camino más corto y/o eficiente, dependiendo del objetivo que se desee.

- '*Go-to-goal*' esquivando obstáculos: este modelo es un modelo abierto en el que normalmente existen muchos caminos diferentes y cualquiera de ellos consigue el objetivo de no colisionar contra ningún obstáculo llegando al destino marcado. Al igual que el anterior se necesita conocer de antemano la situación espacial de los obstáculos y la posición destino para calcular la ruta una vez ha sido detectado el obstáculo en cuestión.

- Seguimiento de camino (*Path Following*): en este caso se desea que el uniciclo converja y siga un camino geométricamente parametrizado. Dependiendo del tipo de parametrización es necesario un número menor o mayor de variables de control. Normalmente en este caso se suele fijar la velocidad lineal del vehículo a un valor constante y el control se realiza sobre la velocidad angular '*w*', aunque si se desea por ejemplo realizar parametrización de curvas es necesario que la velocidad lineal del uniciclo también sea controlable y no sea tomada con un valor fijo constante [15].

El tipo de control deseado en el caso a tratar en el presente proyecto es el de seguimiento de camino de manera que se tratará de manera algo más profunda.

4.2.2 Seguimiento de camino (Path Following).

En este modelo de control debe tenerse parametrizado espacialmente el camino que se desea que el uniciclo siga. En este caso se parametrizará el camino como una recta, de manera que a partir de esta se podrá calcular a la distancia '*d*' en la que se encuentra el centro de rotación instantáneo del uniciclo '*IGR*' (que determinará la posición del uniciclo) y el ángulo de orientación 'θ' con el que está orientado el uniciclo respecto a la recta parametrizada. En la Fig. 10. (a). puede observarse el sistema de referencia empleado, donde el uniciclo puede situarse a la derecha o izquierda de la línea a seguir o con una orientación u otra respecto a esta.

En este caso, el estado del uniciclo es más simplificado al propuesto en (2), ya que ahora no tenemos componente espacial '*x*', de manera que el estado en este caso es [d, θ], y la cinemática queda de la siguiente manera:

$$\begin{cases} \dot{d} = \dot{y} = v \cdot sen(\theta) \\ \dot{\theta} = w \end{cases} \quad (6)$$

Para seguir el camino en este método lo que se hace es añadir una constante denominada distancia de observación '*L*', que será la distancia en el eje '*x*', con origen en el centro de rotación instantáneo del uniciclo, para la cual se calculará la orientación que deseamos que tenga el uniciclo 'θ_D'. De manera que sabiendo la orientación deseada y la orientación que realmente tiene en el momento actual, podamos computar un error como la diferencia entre estas dos y mediante un controlador PID de este error '*e*', podamos obtener la velocidad angular necesaria que debe aplicarse sobre el uniciclo. El esquema explicativo puede verse en la Fig. 10. (b).

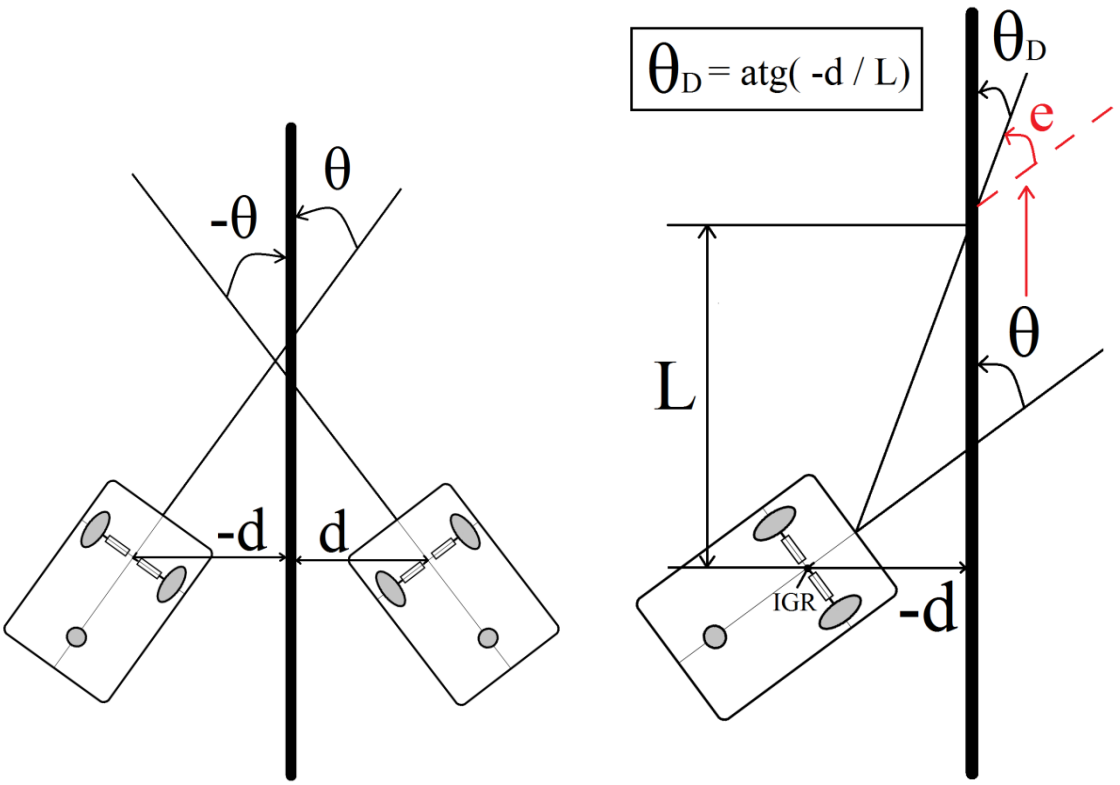


Fig. 10. (a). Sistema de referencia en el estado del uniciclo: *d*– *θ*. (b) Esquema cálculo del ángulo deseado (θ_D).

Por tanto, y aclarando el objetivo más concisamente una vez se ha definido el sistema de referencia en esta pequeña introducción teórica, deberá obtenerse el estado [*d, θ*] del uniciclo a partir de la imagen de captura obtenida a partir de la cámara situada sobre este. Este proceso se clarifica en la Fig. 11. Una vez obtenido su estado se realizará el cálculo de la velocidad angular a aplicar tal y como se explicado en este apartado, y a partir de ella el cálculo de la velocidad a aplicar a cada motor.

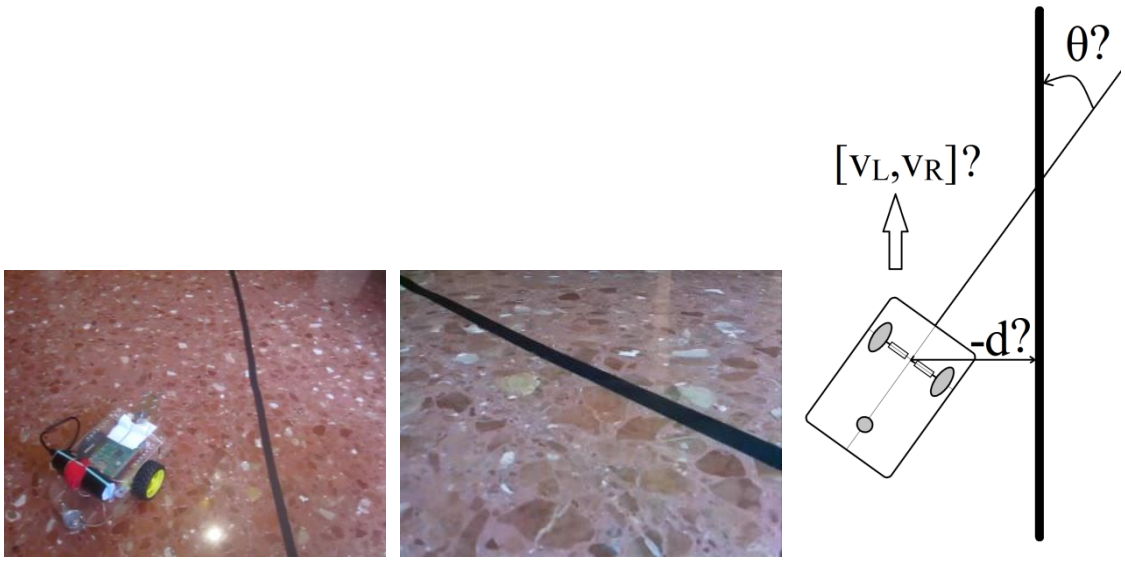


Fig. 11. (a). Foto en 3^{era} persona. (b) Imagen captada. (c) Estado y velocidad del uniciclo a calcular.

Capítulo 5. Desarrollo.

5.1 Flujograma completo del algoritmo desarrollado.

A continuación se exponen dos diagramas de bloques que tienen que ver con los algoritmos realizados, de manera que sirvan de guía en la explicación posterior de cada una de las partes que estos contienen.

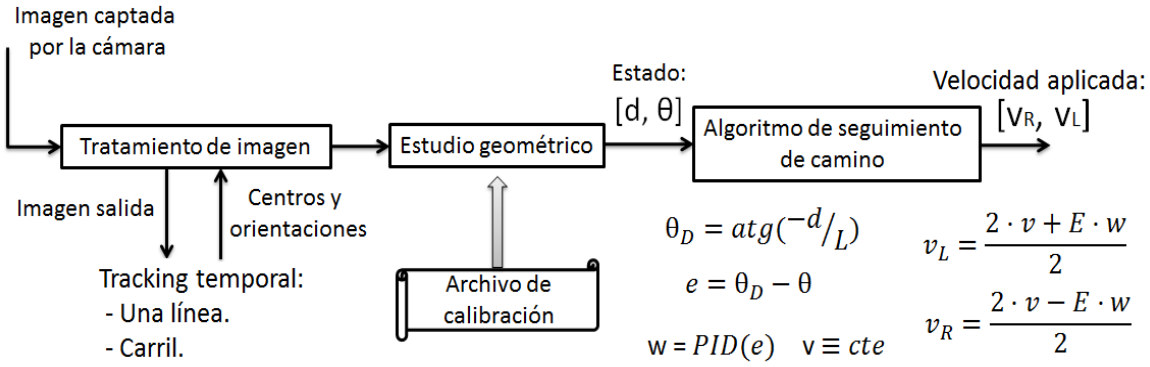


Fig. 12. Diagrama de bloques básico del algoritmo a realizar.

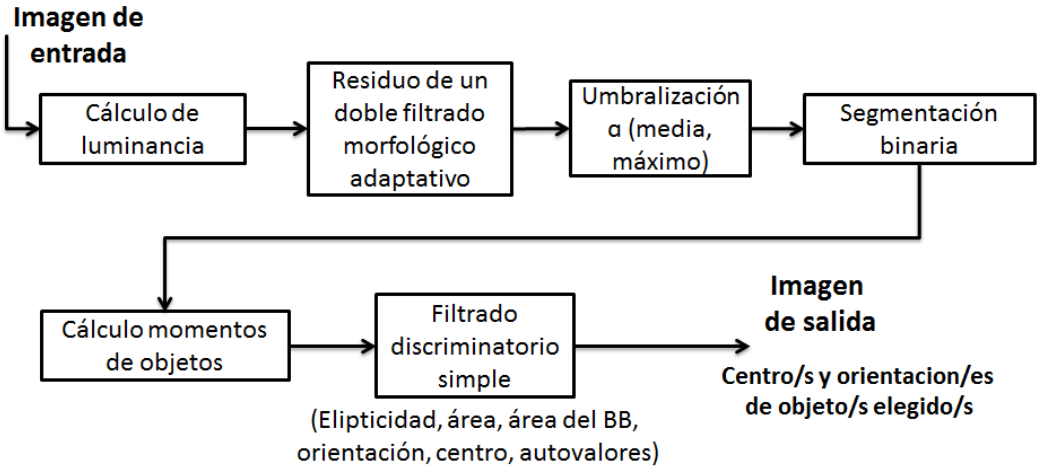


Fig. 13. Diagrama de bloques del tratamiento de imagen realizado.

5.2 Tratamiento de imagen hasta la binarización.

Dos imágenes cualquiera tomadas con la posición en la que se ha decidido colocar la cámara (*5.3.1. Elección de la posición de la cámara*) se pueden ver en la Fig. 14. En estas hay una línea de ancho constante en la realidad en un suelo completamente plano, pudiéndose observar el efecto de la perspectiva perfectamente.

Estas dos imágenes están tomadas con iluminación artificial y natural respectivamente, pudiéndose observar también la notable diferencia iluminativa que existe en cada una de las dos. Además la primera está tomada desde una posición completamente de frente a la recta, mientras que la segunda se ha tomado con el uniciclo separado de ella y además con este orientado de manera que se ve la recta con una angulación diferente.

Para conseguir detectar esta línea se necesita algún tipo de filtro que detecte algo oscuro con un ancho considerable sobre algo claro y que no sea muy sensible a los cambios de iluminación. Lo primero en lo que se pensó fue en utilizar un filtrado Canny, seguido de la Transformada de Hough de búsqueda de rectas [16]. Pero al poco tiempo de experimentación este proceso estaba limitado por un motivo fundamental, y es que el procesador del uniciclo utilizado (GoPiGo) se veía desbordado por el procesamiento que necesitaba este algoritmo. Ya que este algoritmo no sólo necesita aplicar el filtro de Canny además de la Transformada de Hough de la imagen (algo ya costoso de por sí), sino que se necesita realizar una transformación proyectiva previamente sobre la imagen capturada desde la cámara.

Fig. 14. (a) Iluminación artificial, vista de frente. (b) Iluminación natural, vista de lado.

Por tanto se buscó un procesado que aunque posiblemente fuera menos robusto y efectivo que este, el tiempo de ejecución fuera mucho menor de manera que pudiera ejecutarse sin problemas en el GoPiGo.

5.2.1 Desarrollo del algoritmo.

El siguiente algoritmo se basa en el residuo de un doble filtrado que primeramente elimine todo aquello con un ancho inferior al ancho de la línea y posteriormente que elimine la línea en sí, de manera que al realizar el residuo de estas dos imágenes aparezca la línea resaltada sobre el resto de la imagen.

Estos dos filtrados de los cuales se calcula su residuo se tratan de filtrados morfológicos (un cierre en el caso de que se trate de una línea oscura sobre un fondo claro y una apertura en el caso de que se trate de una línea clara sobre un fondo oscuro), que como se sabe de teoría, utilizan un elemento estructurante, en este caso horizontal, realizando una erosión-dilatación en el caso del cierre y una dilatación-erosión en el caso de la apertura, tomando el mínimo (erosión) o máximo (dilatación) de los píxeles que se encuentran debajo de este elemento

estructurante y aplicando este valor al pixel central del propio elemento estructurante en la imagen de salida de dicho filtrado.

Este tipo de operadores se deben de aplicar en una imagen a escala de grises y no en una imagen a color, de manera que primero debemos transformar esta imagen a gris (un único componente de color). Para realizar esta transformación se tienen diversas opciones como tomar una única componente de color (R, G o B), realizar el cálculo de la luminancia (Y), etc. En este caso se toma la luminancia (Y) de la imagen, ya que las carreteras están hechas de manera que las personas puedan diferenciar perfectamente las líneas en el suelo, y la luminancia tiene como objetivo simular este hecho. Este cálculo se realiza de la siguiente manera (7) para cada píxel RGB de la imagen:

$$Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0.114 \cdot B \quad (7)$$

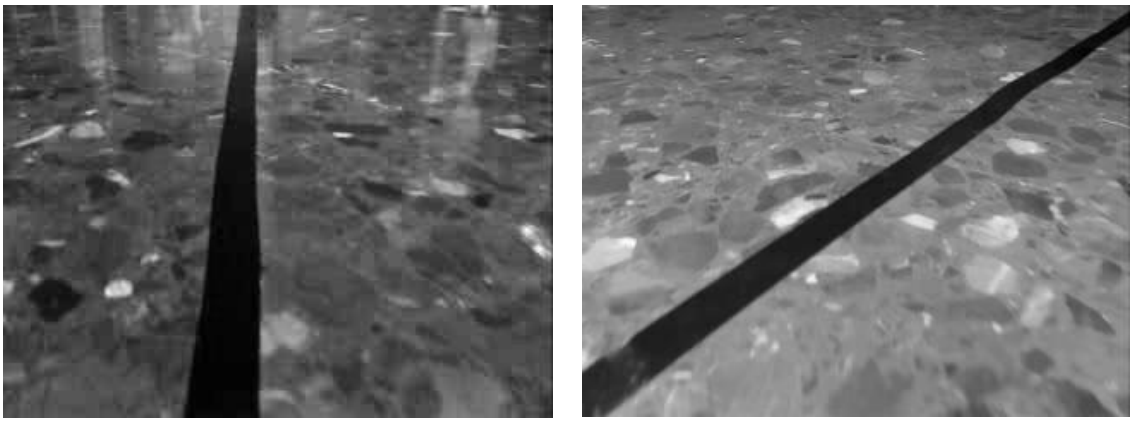


Fig. 15. (a) (b) Cálculo de la luminancia de las imágenes de la Fig. 14.

Observando la Fig. 15., se puede ver que el tamaño aproximado del ancho de la línea en pixeles varía con el eje '*y*' de la imagen, de manera que se hace más ancha cuanto más abajo de la imagen se encuentra la línea (mayor '*y*'). Por tanto, aplicar estos filtros sin variabilidad respecto a la altura de la imagen en la que se encuentra el elemento estructurante, resulta en la eliminación de la línea en el primer filtro, además de eliminar demasiados objetos en el segundo. Este hecho puede verse en la Fig. 16.

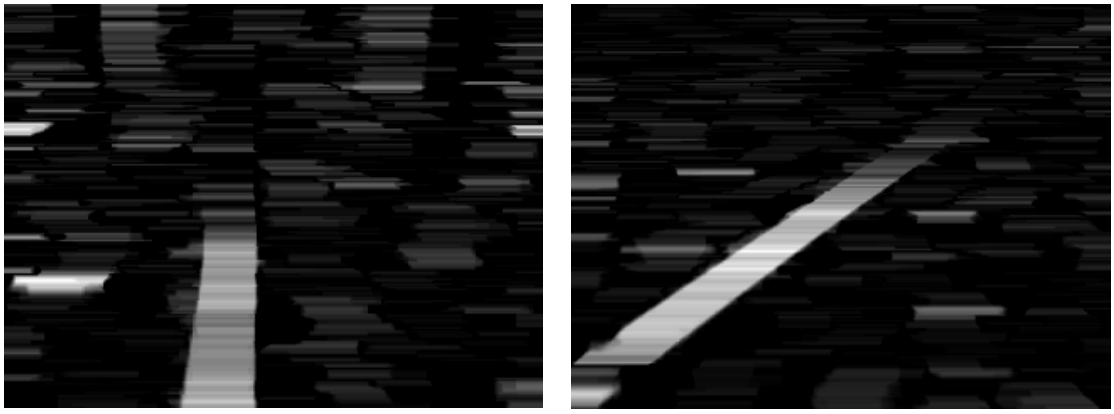


Fig. 16. Aplicación del algoritmo a las imágenes de la Fig. 15. respectivamente con un tamaño de elemento estructurante horizontal fijo.

Se podría pensar que esto se solucionaría no utilizando el primer filtro de eliminación de todo aquello menor al ancho de la línea, y en parte sí, pero aparecería todo aquello que se no se borró con el primer filtrado (Fig. 17.).

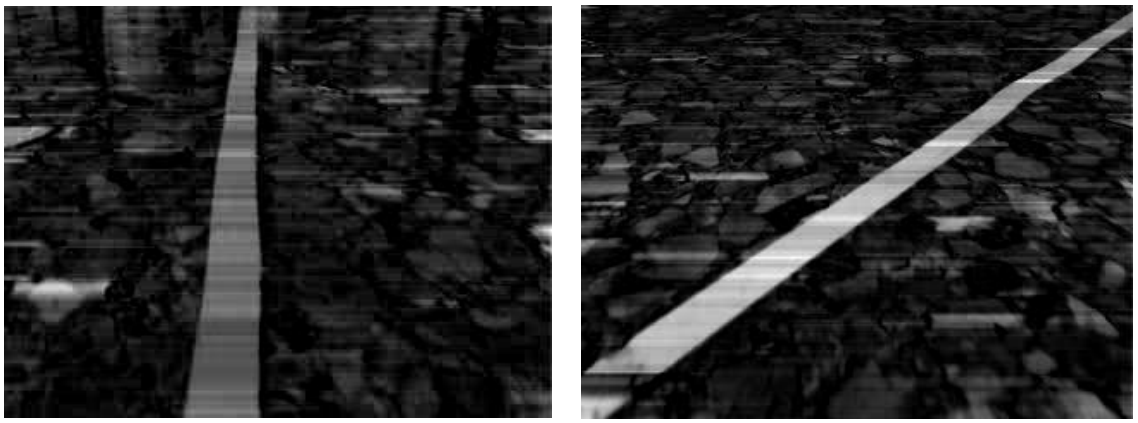


Fig. 17. Residuo de un cierre de tamaño fijo de 60 píxeles para toda la imagen.

De manera que lo ideal sería aplicar un filtrado que se adaptara al ancho de la línea, ya que en la parte superior de la pantalla, el ancho es de aproximadamente 10 píxeles, mientras que en la parte inferior llega a ser de 50 píxeles. Para aplicarlo se puede observar que el ancho de la línea se puede ajustar con una recta respecto a la altura en la que nos encontremos calculando el filtrado. Para ello, en una primera aproximación, se toman valores del ancho de línea/altura con el objetivo de realizar este ajuste en este caso particular de posición de la cámara.

Línea centrada (pixeles)	
Altura imagen	**Ancho línea**
0	-
10	-
20	11
30	13
40	15
50	17
60	18
70	20
80	21
90	23
100	24
110	26
120	27
130	29
140	31
150	32
160	33
170	35

Línea ladeada (pixeles)	
Altura imagen	**Ancho línea**
0	-
10	-
20	12
30	14
40	17
50	19
60	21
70	22
80	24
90	25
100	27
110	29
120	30
130	32
140	33
150	34
160	36
170	38

180	36	180	39
190	37	190	41
200	39	200	43
210	41	210	44
220	-	220	-
230	-	230	-
240	-	240	-

Tabla 2. Relación ancho de línea-altura de imagen para las líneas de las Fig. 13 (a). y (b).

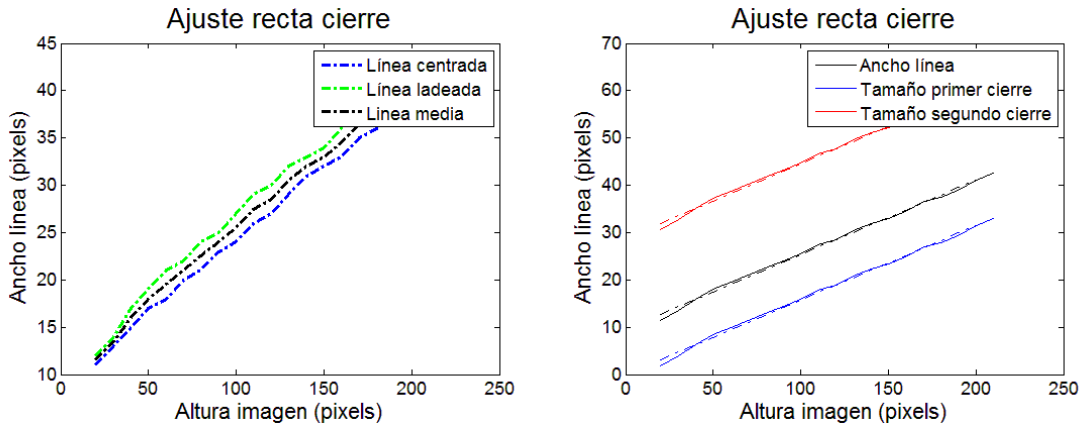


Fig. 18. (a). Variación del ancho de línea/altura de imagen de las líneas de la Fig. 14. junto con la variación media de ambas (negro). (b). Ajuste de recta media de ambas junto con el tamaño de cada uno de los dos elementos estructurantes a aplicar en cada filtrado.

Este ajuste se realizó en Matlab insertando los valores de la Tabla. 2. utilizando la función '*polyfit*' con un ajuste de primer grado (recta). El ajuste se realizó con los valores medios entre las dos líneas (centrada y ladeada). Una vez se tiene la recta ajustada, simplemente se resta y se suma un offset para obtener la recta del primer y segundo cierre respectivamente. Este offset debe escogerse con un margen de seguridad respetable, sobre todo en el segundo filtrado para asegurarnos de que la línea se elimina completamente. Esto puede verse en la Fig. 18. (b). donde puede observarse que el primer filtrado (en azul), no llegará a eliminar la línea (negro), pero si todo lo que quede por debajo de este. Mientras que el segundo filtrado (rojo), eliminará la recta sobradamente pero sin incluir todo aquello que se eliminaba por ejemplo con un filtro fijo de tamaño 60 píxeles. Los coeficientes obtenidos de esta recta son los siguientes:

$$Ancho\ línea\ (píxels) = 0.1574 \cdot Altura\ (píxels) + 9.6026 \quad (8)$$

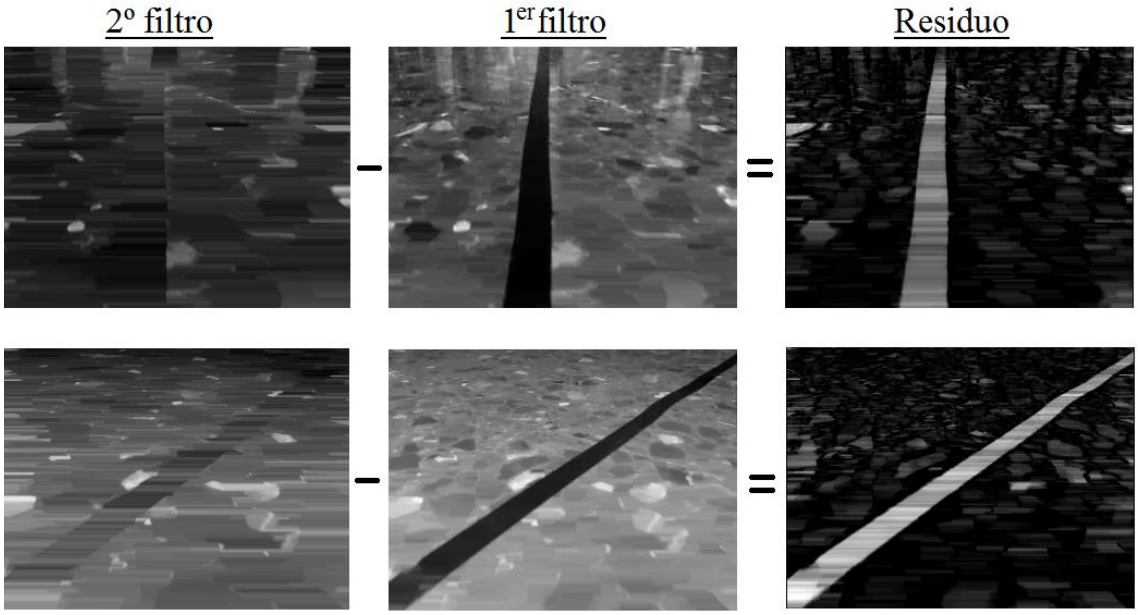


Fig. 19. Aplicación de un filtrado morfológico adaptativo a la altura de las imágenes de la Fig. 15. por pasos.

Comparando los resultados de la Fig. 17. con los resultados de la Fig. 19. podemos ver a simple vista se ha conseguido tener mucho más resaltada la línea respecto al fondo, además de haber muchas menos zonas claras en los resultados de la Fig. 19. y esto nos ayudará notablemente a la hora de la elección del umbral, ya que se tendrá un mayor rango de valores posibles que realicen el umbralizado de la imagen correctamente, aumentando las probabilidades de éxito utilizando este método.

Este método, al utilizar un elemento estructurante variable que se va pasando por toda la imagen de manera secuencial, hace el trabajo que pudiera realizar posteriormente un umbral adaptativo, obteniendo de este la diferencia de valor entre la línea y el fondo, independientemente de si estos valores varían a lo largo de la línea, por ejemplo a causa de reflejos puntuales sobre la línea o a causa de la utilización de diferentes pinturas. Por tanto no tiene sentido realizar un umbralizado adaptativo en esta imagen, de manera que la elección del umbral deberá ser fija. Como los valores de un filtrado a otro varían dependiendo de la iluminación se decide que su valor sea dependiente de la media de valores y/o del máximo valor del residuo resultante. Podrían tomarse varios umbrales fijos con diferentes pesos sobre el valor de la media y/o del máximo con el objetivo de realizar el proceso siguiente a cada una de las imágenes resultantes de los umbralizados, y posteriormente tomando aquella donde se encuentren los resultados más positivos y coherentes. En cambio el procesador del uniciclo utilizado nos va a limitar en el procesado de imágenes impidiéndonos tomar varios umbrales y obligándonos a tomar un único umbral ya que tenemos que trabajar en tiempo real. Un umbral que funciona bastante bien es el siguiente: (9)

$$Umbral = Media \cdot 1.5 + Máximo/15 \quad (9)$$

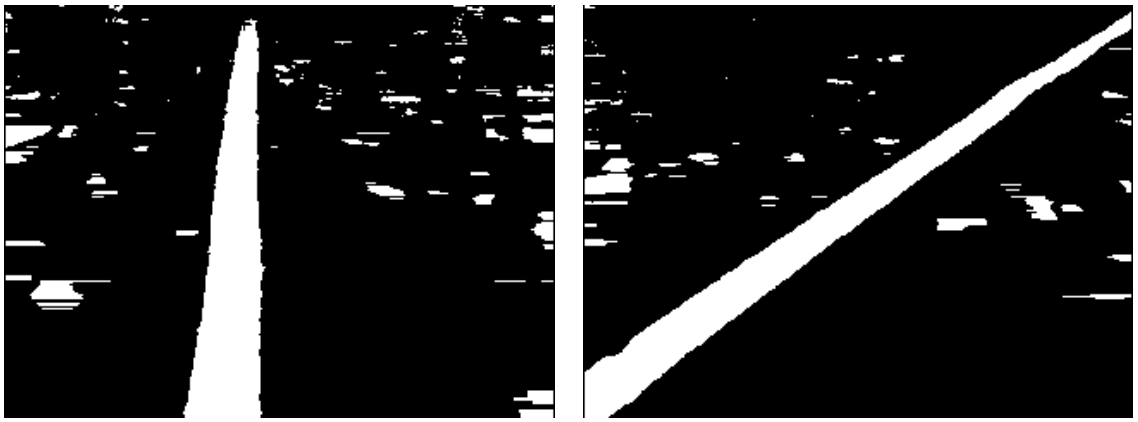


Fig. 20. (a) (b). Líneas de la Fig. 19. umbralizadas con la ecuación (9).

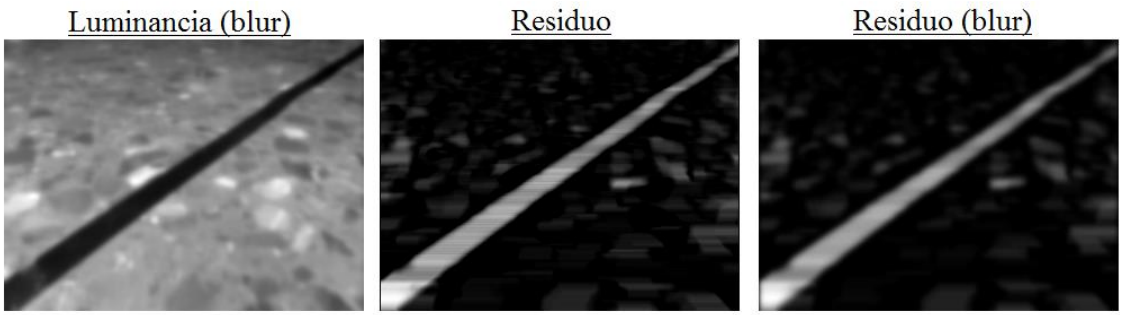
Debe comentarse que la ejecución de este método respecto al método simple no añade absolutamente nada más de retardo al proceso ya que al ser el elemento estructurante utilizado horizontal y el filtrado realizado por líneas de imagen, las funciones de optimización como la recursividad y separabilidad que realizan las funciones de erosión y dilatación de la librería '*CImg*' utilizadas en serie siguen aplicándose. De manera que se ha conseguido reducir el número de elementos indeseados en la imagen con el mismo coste computacional.

Este método, como puede observarse en los resultados obtenidos en la Fig. 20., es capaz de adaptarse a condiciones de luminosidad bastante diversas para el poco procesamiento que se lleva a cabo, detectando perfectamente la línea de la imagen y sin cambiar el ancho que tiene la línea en el resultado umbralizado.

Los coeficientes de la recta ajustada dependerán de la perspectiva de la imagen y por tanto de la posición en la que está colocada la cámara sobre el uniciclo, de manera que este proceso deberá automatizarse, añadiéndose su cálculo en el algoritmo que genera el archivo de automatización en la calibración de cámara (*5.3.2. Archivo de automatización*).

Una vez se tiene la Fig. 20. aún faltaría por realizar la segmentación y el filtrado, pero este proceso varía ligeramente dependiendo del algoritmo ejecutado (seguimiento de una línea o de carril), de manera que este proceso se explicará en cada uno de estos dos apartados.

A pesar de ello puede observarse en estas imágenes que aparece un número demasiado alto de pequeños objetos. Estos aparecen a causa del ruido de alta frecuencia existente en la imagen. Para intentar eliminar el mayor posible de este ruido se aplica un filtro paso bajo (blur-emborronado-suavizado) en la imagen de luminancia y también se aplica posteriormente en el residuo del filtrado morfológico. Este suavizado se realizará levemente ya que no se desea modificar demasiado la imagen real por el posible cambio de resultados en el cálculo de los momentos. Se utilizará para ello la función '*blur*' de la librería '*CImg*', que realiza un filtrado paso bajo Gaussiano, en este caso con un valor *sigma* = 2 se adapta perfectamente a las necesidades descritas.



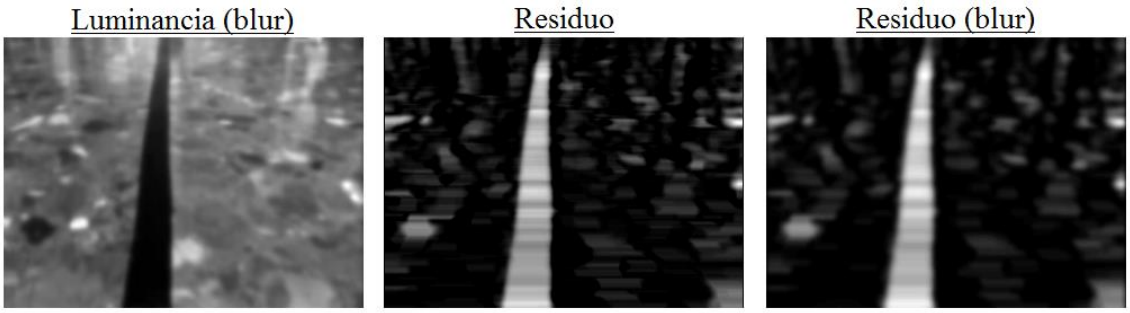


Fig. 21. (a) (b). Proceso de emborronado de las imágenes de luminancia (Fig. 15.).

Fig. 22. (a) (b). Nuevos resultados de la umbralización del residuo suavizado (Fig. 21.).

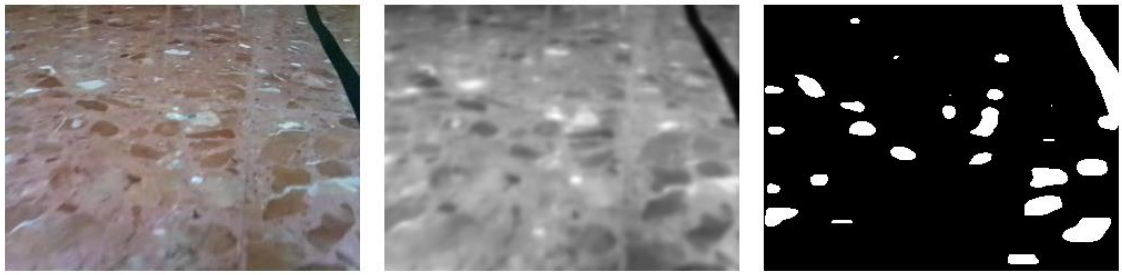


Fig. 23. Aplicación del procesado en un caso más crítico (línea casi esta fuera de la imagen).

En los resultados de la umbralización (Fig. 22.) puede observarse como se han eliminado un número bastante alto de pequeños objetos, esto ayudará no sólo a la detección de la línea sino a optimizar el proceso de segmentación, que tendrá menos objetos que segmentar.

5.2.2 Primeros resultados en curvas.

Aunque el seguimiento de camino únicamente parametrice líneas rectas, es conveniente que se vea el resultado de este procesado de imagen sobre varias imágenes de curvas, ya que aunque sean parametrizadas como líneas rectas y no se tenga en cuenta su curvatura, convendrá que el filtro posterior no las eliminé e intentar realizar un seguimiento de estas aunque no sea tan exacto como en el caso de las rectas.

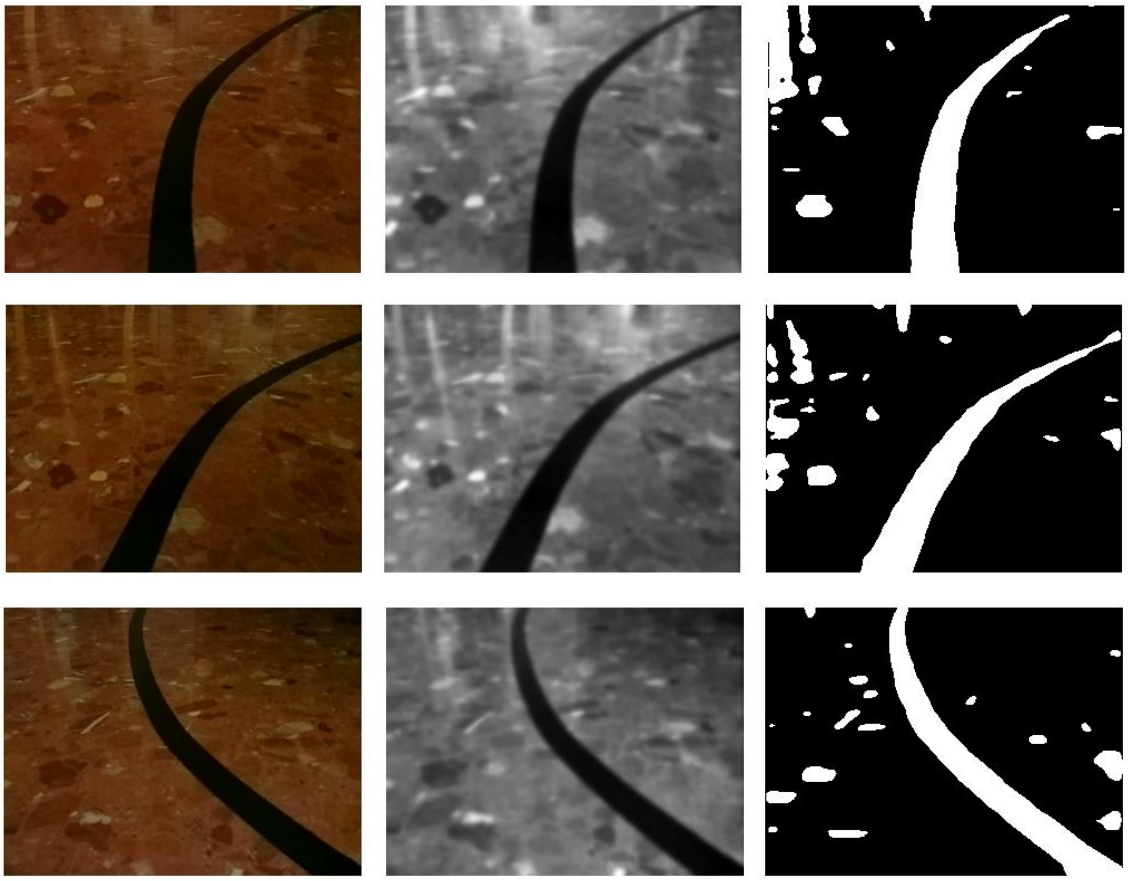


Fig. 24. Aplicación del procesado anterior a tres curvas. (a) (b) (c).

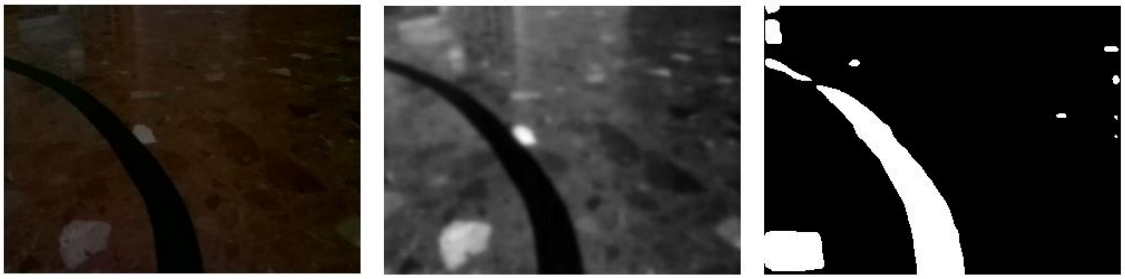


Fig. 25. Aplicación del procesado anterior a una curva cortando la curva. (a) (b) (c).

El único problema de este método ocurre cuando se tienen curvas bastante cerradas a captar (Fig. 25.), cortándola en dos o más trozos en su binarización. Esto ocurre porque el elemento estructurante del filtrado morfológico es horizontal y no tiene en cuenta los cambios de nivel en vertical. Este filtrado no se aplica ya que es imposible de aplicar eficientemente, dado que al realizarse la adaptación a la altura (en horizontal), no puede aplicarse la recursividad y la separabilidad que en el filtrado horizontal si se podía.

5.3 Calibración.

5.3.1 Elección de la posición de la cámara.

Antes de realizar cualquier procesado de imagen o cualquier calibración de cámara debe decidirse en qué posición quiere colocarse teniendo en cuenta el objetivo deseado. Pero antes de ello se debe saber cuál es el ángulo de apertura o visión de la cámara en vertical (aunque de paso también se calculará el horizontal), de manera que con este dato pueda decidirse que inclinación y a que altura se situará la cámara del GoPiGo.

5.3.1.1 Cálculo del ángulo de cobertura de la cámara utilizada.

Para este cálculo se realizó una prueba en la que se pegó un folio DINA4 (210x297 mm) a la pared. Se colocó la cámara sobre una superficie plana de forma que moviendo el folio este quedará centrado y recto en la imagen capturada por la cámara. El tamaño de la imagen capturada por la cámara se configuró en 640x480 píxeles, de manera que conociendo el tamaño real del folio (ancho y largo) y su tamaño en píxeles capturado por la cámara puede realizarse una simple regla de 3 para conocer el tamaño real que abarca la fotografía. En la Fig. 26. puede verse el valor que toma cada uno de los puntos clave para el cálculo en píxeles del ancho y largo de la hoja de papel.

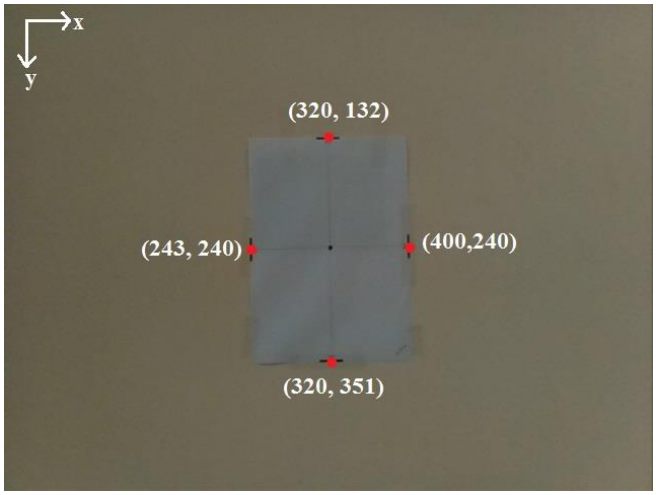


Fig. 26. Imagen captada por la cámara con puntos clave en píxeles.

Alto hoja de papel en píxeles: $351 - 132 = 219\ píxeles$, que equivalen a 29’7 cm, y el alto de la imagen captada es de 480 píxeles lo que equivale a: $(480 \cdot 29'7)/219 = 65'1\ cm$.

De la misma manera para el ancho de la hoja: $400 - 243 = 157\ píxeles$, equivalentes a 21 cm, siendo el ancho captado de 640 píxeles, la equivalencia en cm es: $(640 \cdot 21)/157 = 85'6\ cm$.

Una vez se tiene este valor, para calcular el ángulo de cobertura de la cámara debe medirse a que distancia se ha situado la cámara de la pared, esta medida es de aproximadamente 84 cm como puede verse en la Fig. 27.

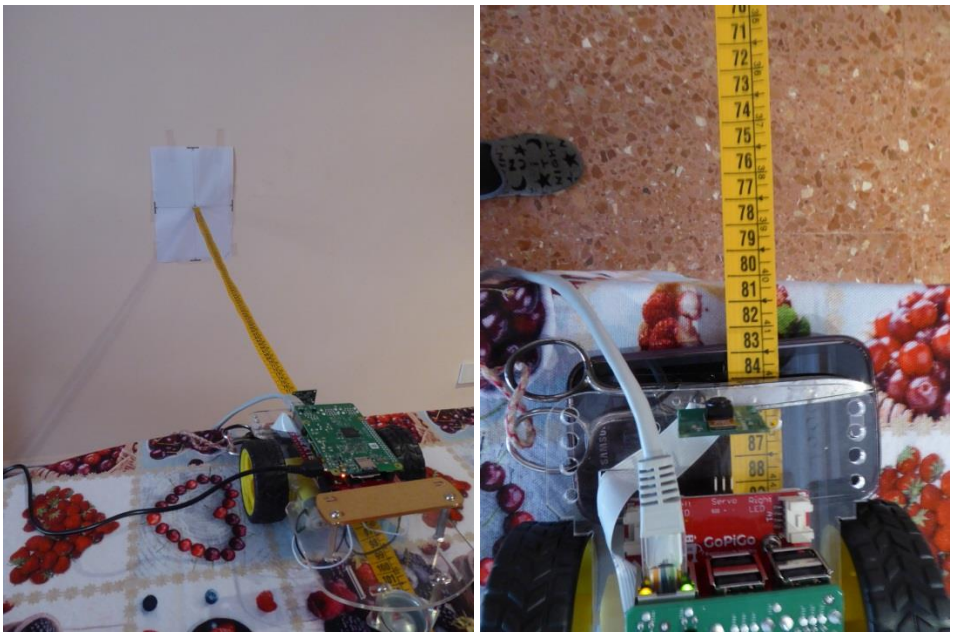


Fig. 27. Distancia desde la pared a la cámara.

Con este dato se puede conseguir el ángulo de apertura de la cámara mediante una sencilla relación trigonométrica. Esto puede verse en el esquema de la Fig. 28. Donde se puede comprobar que la relación ¾ de la cámara se cumple: 42’3/54 ≈ 3/4.

$$\alpha_{vertical} = 2 \cdot atg(\ (\frac{65'1}{2})/84) = 42'3° \qquad (10)$$

$$\alpha_{horizontal} = 2 \cdot atg(\ (\frac{85'6}{2})/84) = 54°$$

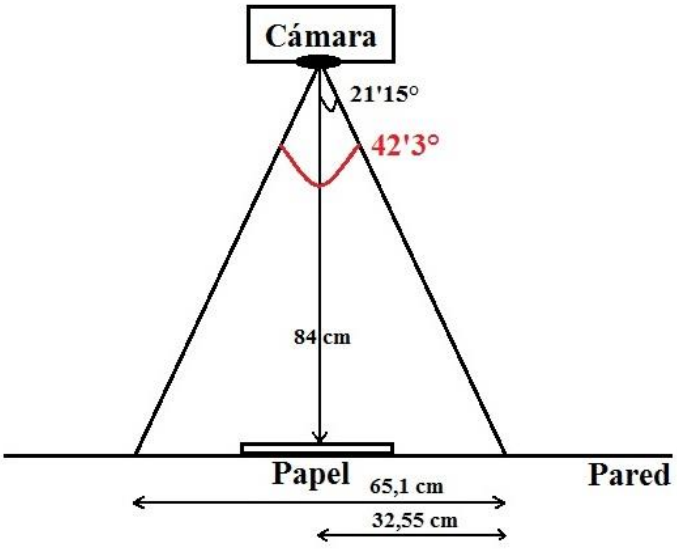


Fig. 28. Esquema de cálculo de ángulo de apertura en vertical.

5.3.1.2 Elección de la inclinación y altura de la cámara.

Si la cámara se coloca sin ningún tipo de inclinación (perpendicular al coche y al suelo), la posible carretera en el caso de que sea plana se situara en la parte mitad inferior de la imagen, de manera que además de malgastar la parte superior de la imagen, se perderá la información que nos pueda dar la parte próxima al coche. Esta suposición nos dice que la inclinación mínima que debe tener la cámara para que no se malgaste visión en la imagen capturada será aquella con la que el haz superior de captación sea paralelo al suelo de manera que se siga mirando al infinito pero ahora teniendo una visión próxima al coche mayor, ya que la parte de visión malgastada se traducirá en una parte aprovechada. Esto puede verse en la Fig. 29. Con esta disposición se tendrá mayor información al mismo coste ya que únicamente tenemos que inclinar la cámara hacia delante en 21'15°.

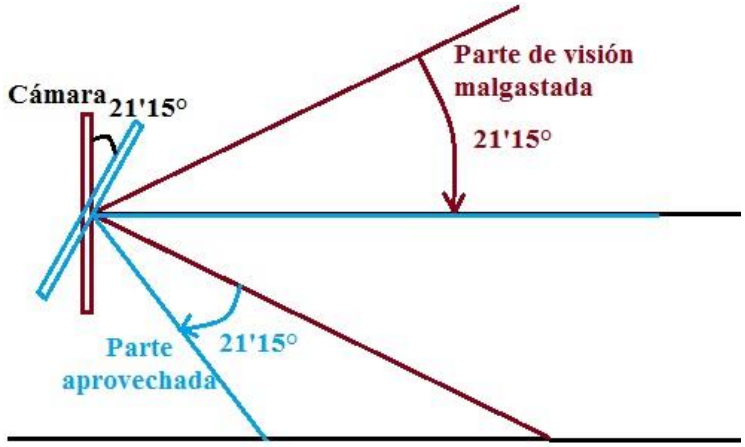


Fig. 29. Parte de visión malgastada a aprovechada.

También debe tenerse en cuenta que a partir de una cierta inclinación, el haz inferior empezará a apuntar hacia detrás del coche, lo cual no tiene sentido, de manera que está será la inclinación máxima de la cámara. Está inclinación se calcula como la diferencia entre 90° y 21'15°, que es el apuntamiento inicial sin inclinación de la cámara, esto son 68'85°. Este comportamiento puede verse en la Fig. 30.

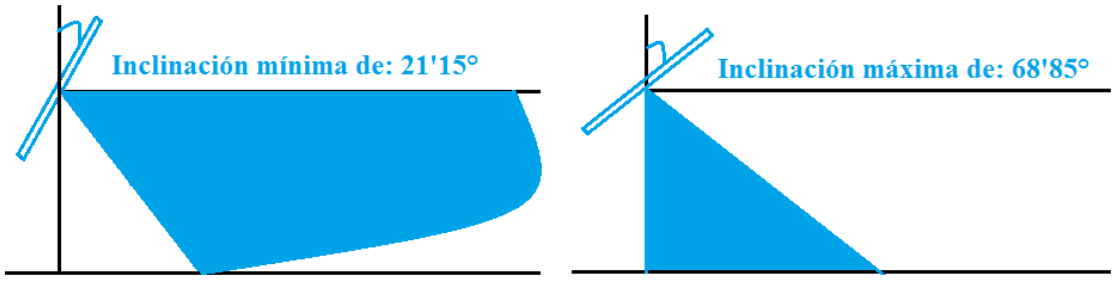


Fig. 30. Inclinación mínima y máxima con la que puede colocarse la cámara.

De manera que el ángulo de inclinación con el que debe posicionarse la cámara debe estar comprendido entre el ángulo máximo y mínimo para que no haya un desaprovechamiento de los recursos de los que disponemos. Para la elección de este ángulo se modelará la variación del margen de captación que tiene la cámara sobre el suelo, para ello se definen dos parámetros: vista perdida y vista total.

La vista perdida (V_P) es aquella comprendida entre la vertical de la cámara y el haz inferior, de manera que cuanto menos inclinado será mayor y cuanto más inclinado tenderá a cero.

La vista total (V_T) es aquella comprendida entre la vertical de la cámara y el haz superior, de manera que cuanto menos inclinado será mayor y conforme vaya inclinándose irá disminuyendo.

Estos dos parámetros dependen tanto de la inclinación de la cámara (α) como de la altura (h). En la Fig. 31. puede observarse la disposición espacial de estos parámetros. Mientras que las ecuaciones que modelan el comportamiento del haz entre el margen de inclinaciones comentados vienen en (8).

El ángulo que encierra la altura (h) y la vista perdida (V_P) lo denominamos α_1, y depende del ángulo de inclinación (α) directamente ya que para una inclinación nula este tiene un valor de 90° - 21'15° = 68'85°. Conforme la inclinación aumenta hasta 68'85°, el ángulo α_1 va disminuyendo de manera que la dependencia entre ambos es: $\alpha_1 = 68'85° - \alpha$.

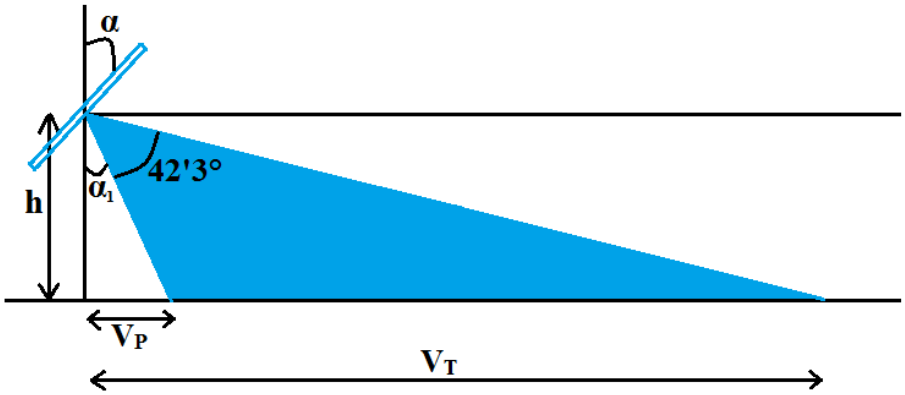


Fig. 31. Parámetros del modelado espacial para una altura (h) e inclinación (α) genérica.

$$V_P = h \cdot tan(\alpha_1) = h \cdot tan(68'85° - \alpha)$$
$$V_T = h \cdot tan(42'3° + \alpha_1) = h \cdot tan(111'15° - \alpha) \quad (11)$$

Aunque la altura (h) sea un parámetro variable se debe tener en cuenta que en el GoPiGo sólo tenemos dos lugares donde colocarla, uno a 5'5 cm y otro a 9'5 cm. De manera que se realiza el cálculo para los ángulos de inclinación comprendidos entre el mínimo y el máximo para estas dos alturas y el resultado puede observarse en la Fig. 32. En esta puede observarse que para una altura mayor, ambos parámetros son mayores, es decir, se tiene mayor vista total pero a cambio se tiene mayor vista perdida.

Tiene que tenerse en cuenta que cuanto más alta este, no sólo se ganará visión a lo lejos, sino también a los laterales, teniendo una visión más amplia y por tanto menor probabilidad de que la línea salga fuera de la imagen. Es por ello que la cámara se situará a la altura mayor posible que es de $h = 9{,}5cm$. También debe tenerse en cuenta que tampoco tiene sentido mirar demasiado lejos, ya que lo que nos interesa como mucho se encuentra a medio metro del coche, por tanto para conseguir una $V_T = 50cm$ ($V_P = 7cm$), se necesita una inclinación de aproximadamente 33°. Este montaje de la cámara sobre el GoPiGo puede verse en la Fig. 33.

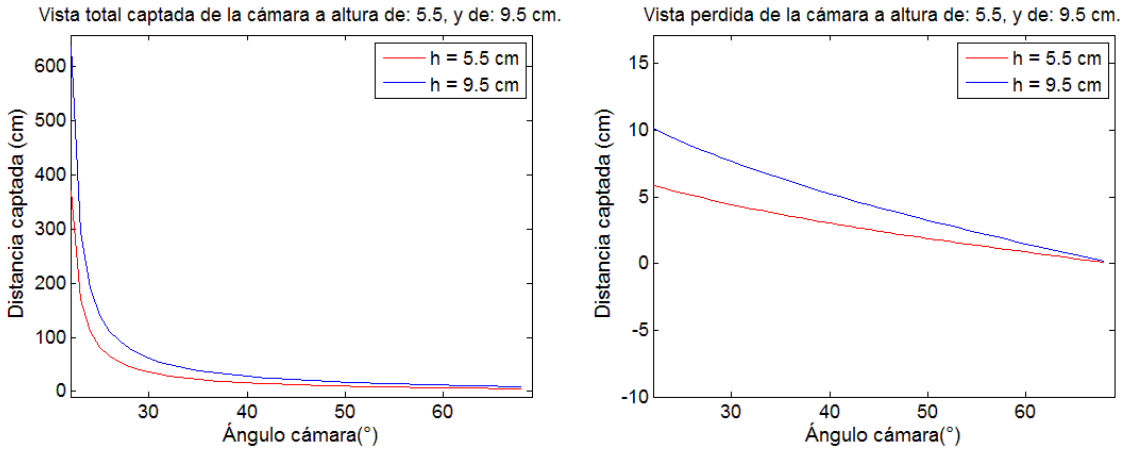


Fig. 32. Variación de V_T y V_P para h = 5'5 cm y 9'5 cm.

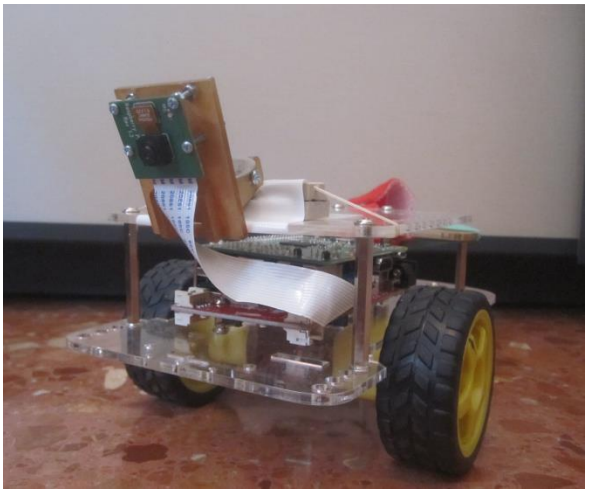


Fig. 33. Montaje final de la cámara sobre el GoPiGo.

5.3.2 Archivo de automatización.

La calibración de la cámara tiene como objetivo principal la obtención de la matriz de homografía que convertirá cualquier punto (coordenada) de la imagen captada en cualquier otro punto (coordenada) de la imagen de proporciones reales, sin ningún tipo de perspectiva y en una unidad de medida conocida. Además también guardara otros parámetros de interés que se comentarán a continuación. De esta manera, tomando el origen de coordenadas en el centro de rotación instantáneo del uniciclo (IGR), podremos calcular en qué estado [*d (cm), θ (°)*] se encuentra respecto a la línea que tendrá que seguir. Básicamente lo que nos permitirá esto será convertir la imagen captada (o algún punto de esta) en una imagen (o algún punto de esta) que esté en unidades métricas conocidas para poder realizar el cálculo de '*w*' de manera correcta y precisa.

Este proceso se realizará en un entorno controlado, de manera que el tratamiento de imagen en este caso no es necesario que sea robusto y pueda adaptarse a cualquier entorno.

La calibración se realizará utilizando una hoja 'calibradora'. Se sabe que la homografía necesita como mínimo de cuatro correspondencias para poder calcularse, pero tomar únicamente cuatro correspondencias puede producir pequeños errores en la calibración, por tanto se utilizarán ocho correspondencias para el cálculo de la matriz de homografía. En la hoja 'calibradora' hay pintados dos cuadrados negros (teniendo un total de ocho esquinas que serán nuestras ocho correspondencias), además de unas marcas de ruedas sobre las cuales deberá colocarse el

uniciclo (Fig. 34.). Conociendo la dimensión del cuadrado (5 cm x 5 cm), colocándolos alineados con el eje de simetría del uniciclo y sabiendo a la distancia que se encuentran en centímetros, podremos confeccionar una 'lista destino' de ocho puntos para el cálculo de la matriz de homografía. Ahora sólo faltará conocer la 'lista origen' de ocho puntos pertenecientes a cada una de las ocho esquinas en la imagen captada por la cámara.

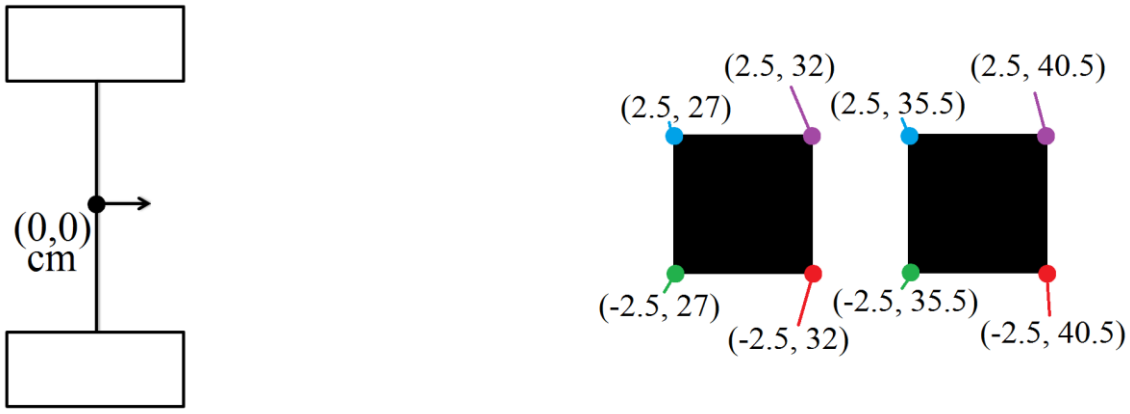


Fig. 34. Hoja calibradora utilizada con las coordenadas especificadas.

Fig. 35. Imagen captada desde la cámara de la hoja calibradora.

El algoritmo realizado es el encargado de obtener casi-automáticamente las coordenadas pertenecientes a las ocho esquinas en la imagen origen captada por la cámara del uniciclo. Este programa es capaz de obtener las imágenes a través de la cámara en tiempo real o aplicar el proceso para una imagen de entrada que se tenga guardada.

Los únicos parámetros que deben insertarse son: un valor algo menor al ancho mínimo en pixeles del cuadrado más alejado y un valor algo mayor al ancho máximo en pixeles que llega a tener el cuadrado más cercano. Cada uno de estos dos valores se utilizará para hacer un proceso similar a lo explicado (*5.2. Tratamiento de imagen hasta la binarización*), es decir, aplicar dos cierres de diferente tamaño, en el que uno eliminará todo aquello de tamaño inferior a los cuadrados, mientras que el siguiente eliminará los propios cuadrados. A continuación se calculará su residuo, obteniendo la imagen deseada. En el caso de ser una imagen de entrada, se aplicará lo mismo que se aplica para cada una de las imágenes en el caso de que tome directamente de la cámara.

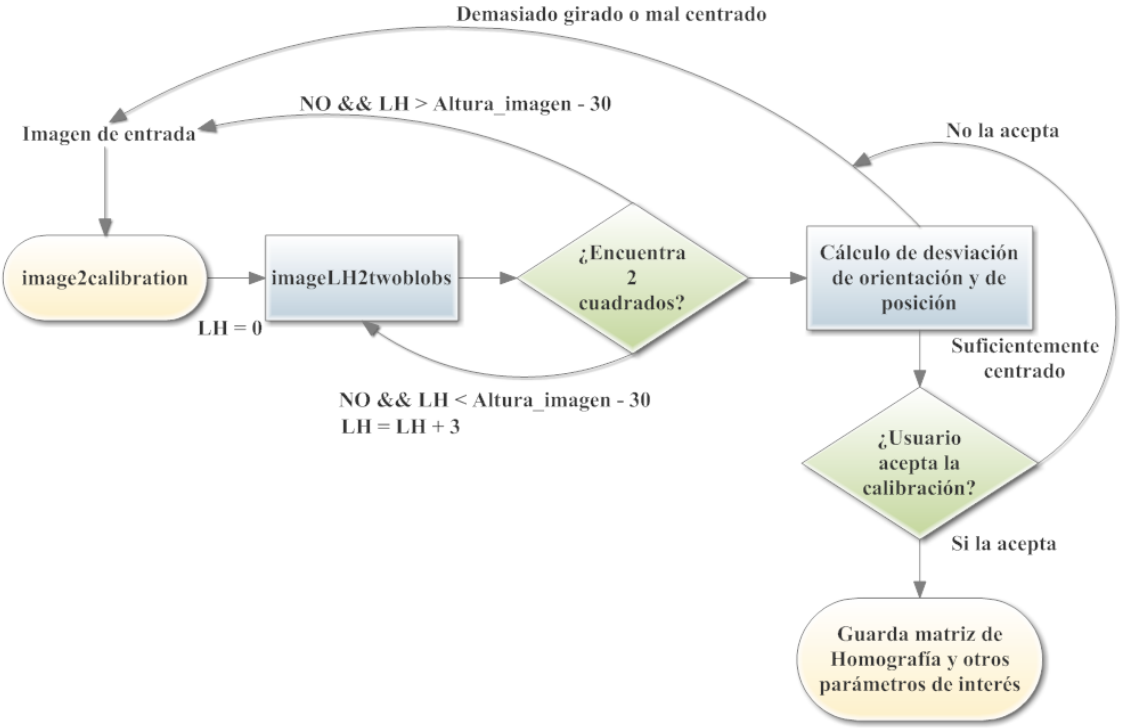


Fig. 36. Flujograma del algoritmo de calibración que genera el archivo de automatización.

Como puede observarse en la Fig. 35. los cuadrados de calibración deben encontrarse sobre una imagen blanca, ya que su búsqueda se realizará de 3 en 3 píxeles hacia abajo (*LH*), de manera que si no se encuentra en la imagen de entrada original, se recortará 3 píxeles por la parte superior y se volverá a buscar utilizando la función '*findContours*' de '*OpenCV*', etc. Este proceso se repite hasta conseguir llegar a la zona del papel blanco, donde únicamente existen dos objetos grandes, que son los cuadrados y una vez encontrados se guardarán utilizando una lista de listas de puntos de sus contornos mediante la librería '*OpenCV*' (Fig. 37.).

En este punto se obtendrá el número de esquinas que tiene cada cuadrado a partir de la función '*approxPolyDP*' de '*OpenCV*', que a partir del contorno de cada objeto que en principio es un cuadrado, calculará las esquinas que tiene a partir de una desviación dada '*epsilon*' que determinará a partir de qué diferencia de píxeles se detectará una esquina o no. Si cada uno tiene cuatro esquinas significa que realmente es un cuadrado y se procederá con el cálculo, sino se comenzará de nuevo el algoritmo para la siguiente imagen captada.

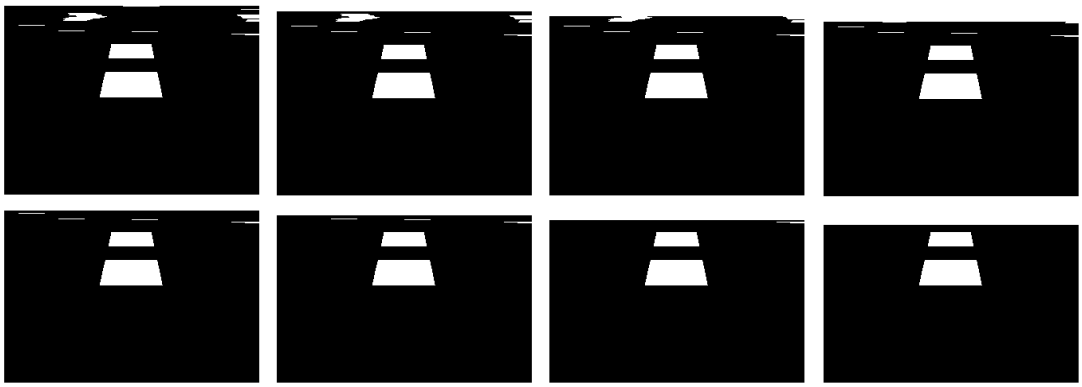


Fig. 37. Búsqueda secuencial de 3 en 3 píxeles de los dos cuadrados para cada imagen.

Una vez se han encontrado los dos cuadrados de cuatro esquinas se calculará el centro de los cuadrados como la media de las cuatro esquinas detectadas y no del propio contorno ya que quiere medirse el grado de desviación en orientación y posición del uniciclo sobre la hoja calibradora o en un peor caso de la cámara sobre el uniciclo (aunque esta se situó perfectamente en el centro del uniciclo con una orientación perfectamente centrada). Estas coordenadas son tomadas en el eje 'x', sin tener en cuenta la desviación en el eje 'y' ya que no es de interés.

$$Desviación\ posición = Centro\ cuadrado\ 1 - Centro\ imagen$$
$$Desviación\ orientación = Centro\ cuadrado\ 1 - Centro\ cuadrado\ 2 \quad (12)$$

Estos valores deben ser menores a 3 píxeles en el caso de que la imagen tenga un ancho de 320 píxeles, es decir, se permite aproximadamente un 1% de error en la medida. A pesar de ello, una vez reordenados los puntos para que coincidan con la 'lista destino' confeccionada manualmente, se muestra al usuario el error de orientación y de posición y este decide si aceptar o no la medida realizada. Esto se hace con el objetivo de realizar una calibración lo más perfecta posible.

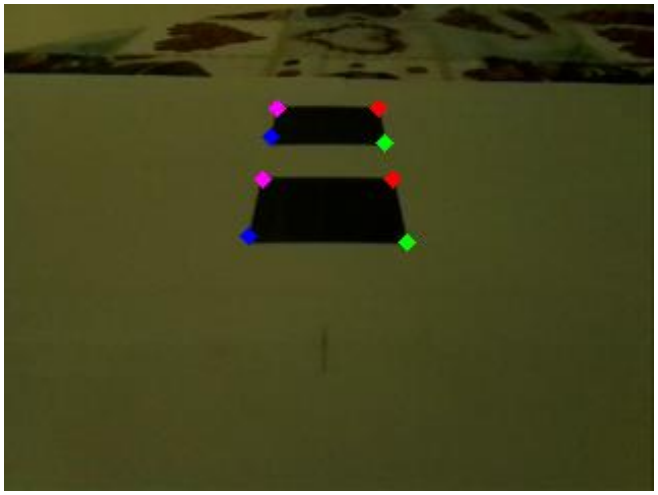


Fig. 38. Imagen captada desde la cámara de la hoja calibradora con las esquinas detectadas y ya ordenadas, el orden correcto de cada cuadrado es: 1. Inferior izquierda (azul). 2. Inferior derecha (verde). 3. Superior izquierda (rojo). 4. Superior derecha (morado).

Una vez se tienen las listas de puntos origen y destino, podría computarse la matriz de homografía en este punto del programa si lo que únicamente se necesitará es la conversión de unidades de medida de píxeles a centímetros, pero previamente a esto se realiza una variación en la lista de puntos destino que se confeccionó manualmente con el objetivo de poder visualizar correctamente el resultado de la transformación, para asegurarnos de que es correcta y para poder detectar y depurar posibles errores en los algoritmos de seguimiento de una línea y de carril.

Esta variación se trata de una translación y una rotación. La translación nos permitirá poder observar completamente todo lo transformado, ya que antes, al tener coordenadas negativas (la mitad de la imagen destino), estas aparecían fuera de la imagen y no podrían visualizarse. Mientras que la rotación nos servirá para poder ver en sentido natural (hacia arriba), lo que ve el uniciclo. Además se aplicará un factor de escalado que multiplicará cada una de las coordenadas de la lista destino, para poder visualizarse con un mayor tamaño (Fig. 39.). Estos valores nuevos de origen de coordenadas y constante de proporcionalidad deberán guardarse para el futuro cálculo del estado del uniciclo [d, θ].

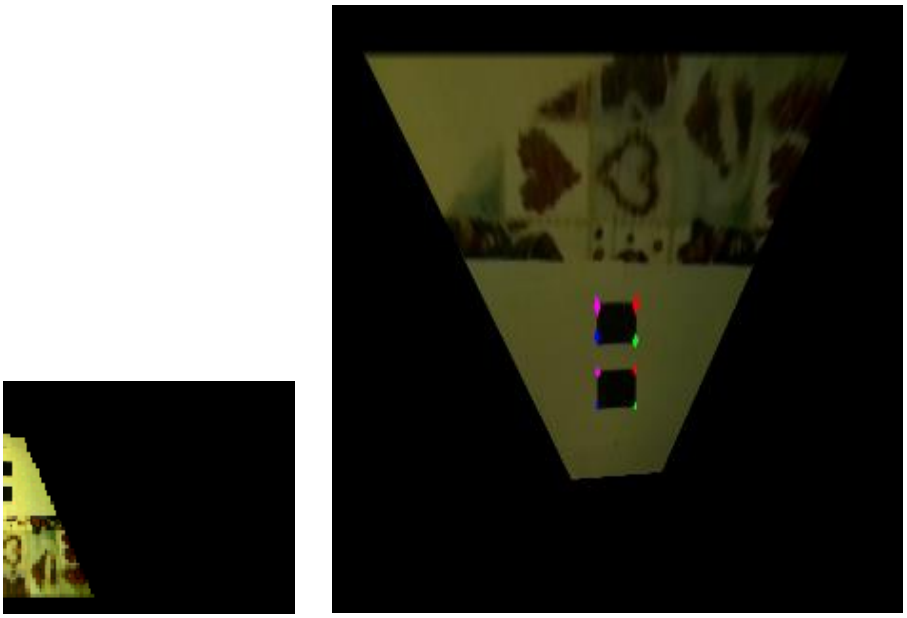


Fig. 39. Comparación de la visualización antes y después de la translación, rotación y escalado.

El último cálculo que debe automatizarse, como se comentó (*5.2. Tratamiento de imagen hasta la binarización*), es el ajuste de la recta entre ancho de línea y altura a la que se encuentra (en píxeles) para realizar el filtrado morfológico explicado en la detección de líneas. Para este ajuste tenemos cuatro puntos de ajuste: las dos esquinas inferiores del cuadrado inferior, las dos esquinas superiores del cuadrado inferior, las dos esquinas inferiores del cuadrado superior y las dos esquinas superiores del cuadrado superior. Para cada par de estos puntos se calculará el ancho (distancia en el eje '*x*') entre ellos y la altura a la que se encuentran en la imagen (valor media de '*y*' de ambos puntos), ajustando una recta mediante la función '*polyfit*' (en C++) obtenida de una libraría de terceros [19]. Así se obtendrán los coeficientes de la recta que también deberán guardarse en el archivo de automatización. Estos coeficientes equivalen en este caso a una línea de ancho 5 cm, que es la longitud del lado del cuadrado, de manera que también se deberá guardar este valor en el archivo de automatización, que permitirá tener una referencia de a que ancho fueron tomados los coeficientes de la recta, escalándolos en función del ancho de la línea que quiera detectarse.

También conviene guardar el tamaño de la imagen de calibración, ya que nos permitirá independizar el tamaño de la imagen durante su procesado, pudiendo escalar y re-escalar los puntos a transformar que se obtendrán.

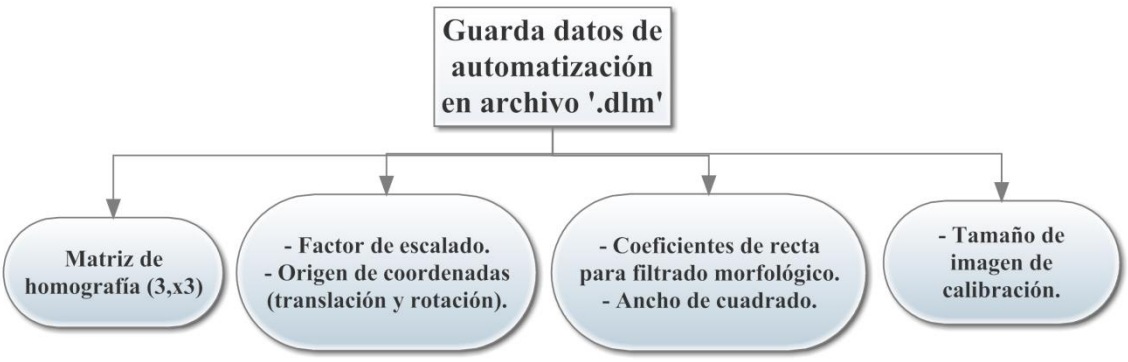


Fig. 40. Datos guardados por el programa de calibración.

5.4 Filtrado de objetos por sus características básicas.

Una vez se tiene la imagen binarizada (Fig. 22. y Fig. 23. (c).), deberá realizarse el proceso de segmentado y de un filtrado discriminatorio de todo aquello que no se adapte a las características de una línea. En el tratamiento y procesado de imágenes este proceso suele estar dividido en dos etapas claramente diferenciadas.

La primera de ellas filtra los objetos de manera más simple y rudimentaria a partir del cálculo de sus momentos (área, Bounding Box, centro de masas, autovalores, orientación, etc.), de manera que conociendo las características básicas del objeto que deseamos filtrar se realice un filtro dependiendo del valor de estos momentos. Esta etapa es la menos costosa computacionalmente y es por ello que suele ser exprimida al máximo por los desarrolladores de aplicaciones.

La segunda etapa, una vez se ha realizado el filtrado más simple, se realiza un filtrado mucho más complejo y costoso que aplicado directamente sin haber pasado por el primer filtro sería inviable debido al alto número de objetos en la imagen, que se traduce en una cantidad demasiado elevada de operaciones. Este filtrado es mucho más específico de la aplicación que desea realizarse y está abierto a cualquier idea feliz por parte del desarrollador. Normalmente un buen filtrado de este tipo puede ser incluso más útil para otras aplicaciones diferentes que las que para las que se está realmente realizando (detector de bordes de Canny, detector de esquinas 'fuertes' de Harris, la transformada de Hough para rectas o círculos, cálculo de la convexidad y búsqueda de los defectos de convexidad, etc.). Estos métodos son bastante complejos y muy costosos, y como se verá a continuación, el procesador del uniciclo es bastante lento y no podrán aplicarse.

Por tanto nos conformaremos con un primer filtrado a partir del cálculo de los momentos más básicos de la caracterización de objetos. Dependiendo del algoritmo (una línea o carril) se realizará un filtrado u otro.

En ambos algoritmos se realiza el segmentado y el cálculo de momentos (centros, Bounding Box, orientación, autovalores) de la misma manera, de manera que cada uno de los dos algoritmos se explicará a partir de este punto y hasta el cálculo del estado [d, θ]. A partir de la determinación del estado en el que se encuentra el uniciclo, también ejecutarán el mismo código.

Observando la líneas de Fig. 22. y Fig. 23. (líneas rectas), sus características básicas son las siguientes:

- Son objetos muy alargados (elípticos).
- Son objetos con un área mayor a los demás objetos.
- Son objetos con un área de Bounding Box mayor a los demás objetos.

Observando la líneas de Fig. 24. (líneas curvas), la única diferencia entre las líneas rectas es que no tienen una elipticidad tan diferenciable, pudiendo ser nada elípticos (*elipticidad = 1*), si el Bounding Box que encierra a la línea curva es cuadrado completamente.

A partir de estos tres parámetros se ha obtenido un parámetro que los engloba dándole diferentes pesos a cada uno. Esta combinación es la siguiente, y tiene en cuenta principalmente el peso del área del Bounding Box (para no filtrar las curvas) y el valor de la elipticidad (para no filtrar las rectas). Este valor se utilizará para realizar un primer filtrado y para saber cuán grande es el objeto detectado en la imagen. El resultado de este cálculo para las líneas de las Fig.s anteriormente binarizadas puede verse en la Tabla. 4.

$$Área\_elipticidad = 7 \cdot BB\_área + Área + 150 \cdot Elipticidad \quad (13)$$

Para observar el resultado de la aplicación de esta ecuación sobre las imágenes anteriores se ha realizado un pequeño programa que realiza la segmentación de ellas, pintando de color gris el

objeto con mayor '*área_elipticidad*', y de rojo, azul y amarillo los siguientes tres valores más grandes.

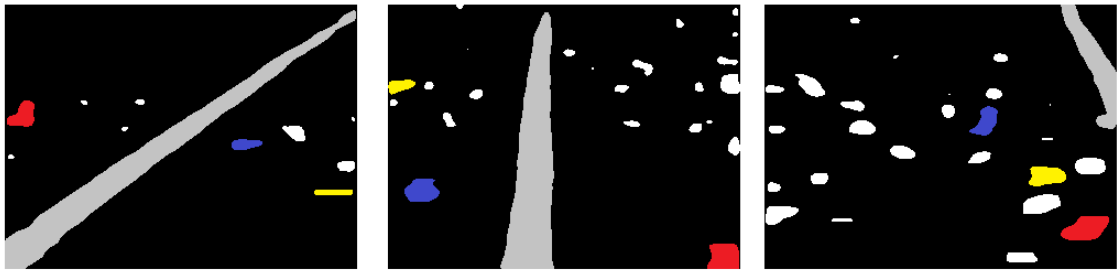


Fig. 41. (a) (b) (c). Segmentación de imágenes en Fig. 22. y Fig. 23. con colores distintivos.

Fig. 42. (a) (b) (c). Segmentación de imágenes en Fig. 24. con colores distintivos.

Línea (gris)	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Fig. 41. (a)	15,98	7 190	74 412	530 472	0,096
Fig. 41. (b)	6,55	6 651	11 136	85 585.6	0,597
Fig. 41. (c)	5,13	1 947	5 600	41 916,8	0,347
Fig. 42. (a)	4,26	6 722	32 802	236 976	0,204
Fig. 42. (b)	5,64	6 726	50 328	359 868	0,133
Fig. 42. (c)	6,38	7 625	49 712	356 567	0,153

Tabla 3. Objetos con mayor '*área_elipticidad*' en cada una de las seis imágenes anteriores.

Objetos indeseados	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Rojo	1,48	405	552	4 491,32	0,733
Azul	3,03	216	243	2 371,7	0,889
Amarillo	7, 08	136	172	2 187,4	0,790

Tabla 4. Objetos indeseados más grandes en Fig. 41. (a).

Objetos indeseados	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Rojo	1,14	600	616	5 084,2	0,974
Azul	1,4	528	620	5 078,04	0,851
Amarillo	2,37	223	312	2 762,58	0,714

Tabla 5. Objetos indeseados más grandes en Fig. 41. (b).

Objetos indeseados	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Rojo	2,13	706	903	7 346,86	0,781
Azul	1,79	401	650	5 220,38	0,617
Amarillo	1,94	479	561	4 697,82	0,853

Tabla 6. Objetos indeseados más grandes en Fig. 41. (c).

Objetos indeseados	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Rojo	4,13	1 699	2 349	18 762,3	0,72
Azul	4,49	367	588	5 156,57	0,62
Amarillo	1,51	532	608	5 015,3	0,87

Tabla 7. Objetos indeseados más grandes en Fig. 42. (a)

Objetos indeseados	Elipticidad	Área	Área del BB	Área_elipticidad	Área/Área_BB
Rojo	3,51	706	1344	10 641,5	0,525
Azul	1,38	575	1008	7 839,22	0,570
Amarillo	2,04	257	288	2 580,07	0,892

Tabla 8. Objetos indeseados más grandes en Fig. 42. (b)

En estas tablas puede observarse que en el caso más extremo (Fig. 41. (c).), el valor de '*área_elipticidad*' de la línea correcta tiene un valor de 40 mil, mientras que en el caso más extremo de objeto indeseado (Fig. 42. (a).) que en este caso fue producido por un reflejo en el suelo, tiene un valor de 18 mil. Por tanto un valor bueno a partir el cuál el objeto es filtrado por este parámetro es 25 mil.

Además de este filtro se realizan filtros muy básicos con el objetivo de filtrar objetos enanos, poco elípticos o cuyo centro de masas se encuentre demasiado pegado a los bordes laterales de la imagen donde es imposible que el centro de masas de la línea llegue a estar.

Por tanto estos filtros quedan resumidos en lo siguiente:

```
if(areas[x] > (img.height()*img.width())/750   &&   elipticidad > 1.7   &&    area_el
ipticidad_actual > 25000    &&   autovalor_min > 2   &&    autovalor_max > 15      &&
    centros(x,1) > (1*img.height())/32    &&    centros(x,1) < (31*img.height())/32)
{   /// Puede ser una línea, pasa primer filtro /// }
```

Otros filtros discriminatorios:

Una vez se ha pasado este primer filtro, es posible que algún objeto que realmente no es una línea haya sido aceptado hasta este momento. Este tipo de filtro suele saltarlo dos tipos de objetos:

- Objetos cuyo tamaño es muy poco elíptico para lo grande que es su proporción '*Área/Área_BB*': estos objetos pueden aparecer a causa de encontrar algún reflejo o mancha en el suelo lo suficientemente grande como para pasar el filtro anterior, aunque estos objetos suelen ser poco elípticos (Fig. 25. (c), objeto abajo a la izquierda). El único objeto que realmente es una línea cuya proporción '*Área/Área_BB*' es cercano a 1, es cuando el uniciclo se sitúa justo encima de la línea y completamente bien orientado hacia esta (Fig. 41. (b).), pero para compensarlo tiene una elipticidad bastante

grande. También debe tenerse en cuenta el tamaño de '*área_elipticidad*', aceptándolo sólo si supera un umbral superior al anterior. De manera que para solucionar este problema se utilizan tres filtros en cascada basados en que cuanta mayor proporción '*Área/Área_BB*', se les pedirá cada vez mayor '*área_elipticidad*' pero menor elipticidad, es decir, objetos que para tener una proporción '*Área/Área_BB*', son poco elípticos y con poco valor '*área_elipticidad*'.

```
-   if(areas(x)/bbox_areas(x) > 0.6 && elipticidad < 3.8 && area_elipticidad_actua
    l < 50000){
-       std::cout<<"Filtro objeto malo 1. \n";
-       continue;
-   }
-   if(areas(x)/bbox_areas(x) > 0.4 && elipticidad < 4 && area_elipticidad_actual
    < 35000){
-       std::cout<<"Filtro objeto malo 2. \n";
-       continue;
-   }
-   if(areas(x)/bbox_areas(x) > 0.3 && elipticidad < 5 && area_elipticidad_actual
    < 27500){
-       std::cout<<"Filtro objeto malo 3. \n";
-       continue;
-   }
```

- Objetos que son demasiado inclinados: para que la línea sea filtrada adecuadamente tiene que tener una inclinación no demasiado grande, ya que de ser así el filtrado morfológico horizontal no hubiera sido capaz de filtrarla correctamente. De manera que aquellos objetos demasiado inclinados no se tienen en cuenta (antes de realizar esta comprobación debe mapearse la orientación del objeto devuelta por la función de cálculo de momentos al sistema de referencia utilizado en el cálculo de 'θ'). En este caso se ha escogido un valor tope de ± 72°, de manera que si la línea tiene un valor de orientación mayor a este valor, el uniciclo se perderá.

```
-   if(orientacion(x) > 72 || orientacion(x) < -72){
-       std::cout<<"Filtro: objeto demasiado inclinado. \n";
-       continue;
-   }
```

5.4.1 Seguimiento de una línea.

El proceso anterior es compartido por ambos algoritmos pero en el seguimiento de una única línea, sólo tendremos que buscar el objeto más parecido a una línea dentro de la imagen. Esto se consigue guardando los dos objetos con mayor '*área_elipticidad*' de todos y comparando estos dos valores. Si el mayor valor es superior al doble del segundo valor mayor, entonces se aceptará la línea como el objeto de mayor '*área_elipticidad*' de estos dos, sino no se aceptará nada y se volverá a comenzar de nuevo el tratamiento para el siguiente fotograma captado.

5.4.2 Seguimiento de carril.

En el seguimiento de carril, se guardarán todas las líneas aceptadas y que no han sido discriminadas por el filtro anterior, siendo el proceso de tracking temporal/espacial explicado a continuación el encargado de escoger que líneas o línea será aceptada y de cómo se calculará el camino a seguir dependiendo de haber encontrado una o dos líneas.

5.5 Tracking temporal/espacial para el seguimiento correcto del camino.

El tracking temporal se trata del seguimiento temporal/espacial de la línea entre un instante anterior y el actual. Se realiza con el objetivo de emparejar objetos, de esta manera se consigue asegurar que la línea detectada es correcta o que un objeto detectado como línea realmente no lo es pudiendo filtrar este error que consiguió atravesar el filtro de características anterior.

En el caso de seguimiento de una única línea se realiza un tracking temporal muy simple ya que únicamente se está emparejando un objeto.

En cambio en el caso de un carril (dos líneas), el emparejamiento es algo más complejo ya que debe emparejarse cualquier número de líneas aceptadas por el filtro de características de objetos, con un carril o en el caso de que no estén las dos líneas en la imagen, con la línea izquierda o derecha.

Por ello el tracking temporal/espacial es diferente en cada uno de estos dos casos. Además debe tenerse en cuenta en que momento deben pararse los motores en el caso de que no se encuentre línea, esto también se solucionará en este apartado.

5.5.1 Seguimiento de una línea.

Tras el filtro basado en las características de objetos realizado en el paso anterior, pueden ocurrir tres cosas:

- Ninguna línea fue detectada, entonces capta el siguiente fotograma.
- Una línea fue detectada, esta línea es a la que se le aplica el tracking.
- Dos o más líneas fueron detectadas, se toman las dos líneas con mayor '*área_elipticidad*' y si este valor es dos veces mayor que la segunda con mayor '*área_elipticidad*' es tomada como línea a seguir. Sino, es demasiado arriesgado seguirla ya que no hay una gran diferencia de valores y sigue buscando en el siguiente fotograma.

El último paso antes de determinar el estado [d, θ] en el que se encuentra el uniciclo se trata de realizar un pequeño seguimiento de la línea. Este seguimiento únicamente se realiza cuando en algún momento deja de detectarse la línea, confiando en que si se está detectando fotograma tras fotograma correctamente significa que realmente se está siguiendo la línea. De manera que existen tres posibles 'modos' que determinan que ocurrió en el fotograma anterior:

- Modo -1: el uniciclo nunca ha visto una línea. Comienza estando en este modo, si en el primer fotograma no detecta línea deja de ejecutar el programa ya que no puede conocer su estado, y si la detecta pasa al modo 1 sin ningún tipo de restricción especial. Una vez se encontró línea no vuelve al modo -1.

- Modo 0: el uniciclo viene de estar perdido. Se realiza un pequeño seguimiento en el que únicamente se aceptará la línea detectada si su centro se encuentra cercano en el eje 'x' al centro de la línea detectada en el fotograma anterior. Este seguimiento se realiza principalmente cuando la línea desaparece de la imagen debido a que el uniciclo está demasiado inclinado respecto a la línea, de manera que no acepte ningún objeto cuyo centro no vuelva a aparecer por ese mismo lado de la imagen. La distancia se calcula en el eje 'x' ya que la línea puede desaparecer por la parte inferior pero volver a aparecer por la parte superior, y esta diferencia de altura (eje 'y') no se quiere tener en cuenta (Fig. 43.). Si el objeto detectado como línea tiene un valor bastante grande de

'*área_elipticidad*' significa que sí es la línea realmente y por algún motivo (sucesión de no detecciones, demasiado retardo, etc.) ha realizado un salto bastante grande en el eje '*x*' y en ese caso sí que se aceptará como línea. Esto permitirá no acumular errores a lo largo de la ejecución del programa.

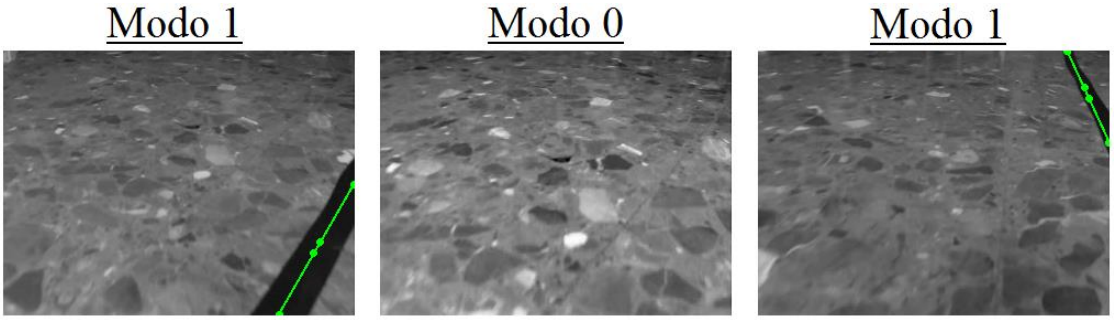


Fig. 43. Línea desaparece y aparece por el mismo lado de la pantalla (línea y puntos verdes significan la detección del camino a seguir).

- Modo 1: no está perdido. Significa que en el fotograma anterior encontró línea, de manera que esté donde esté la línea encontrada en este fotograma la acepta como correcta.
- Modo de transición entre el modo -1 o 0 y el modo 1, en el que aunque se sigue la línea encontrada no se reinicia el contador de línea encontrada. De esta manera se consigue continuar perdido en el caso que aparezcan objetos fugaces a causa de reflejos u otros errores, y parando en el momento que se supere el tiempo de máximo tiempo perdido. Esto se explicara más detalladamente en el apartado de '*5.5.3. Parada de motores*'.

5.5.2 Seguimiento de carril.

El algoritmo de doble línea, al dejar pasar todos aquellos objetos parecidos a una línea y no filtrarlos con un proceso más complejo tal y como se comentó antes, nos deja en una situación difícil ya que es posible detectar objetos indeseados y parecidos a líneas que realmente no lo son. Para ello se realiza este tracking temporal/espacial que es bastante más complejo que el del seguimiento de una única línea, aunque sigue sin ser exacto y puede dar lugar a errores en el caso de que haya demasiado retardo entre la toma de fotogramas que pueda llegar a producir un movimiento demasiado grande para poder realizar el tracking correctamente.

En este caso se tienen varias combinaciones de casos a tratar, y es que hay varios casos diferentes ya que se puede detectar ninguna, una, dos o muchas líneas en el fotograma actual; viniendo de un fotograma anterior en el que no se detectó ninguna línea, hubo una (izquierda, derecha o desconocida) o hubo dos líneas. El número de líneas aceptadas en el fotograma anterior se le llama modo al igual que en el tracking de seguimiento de una única línea y hay 4 modos diferentes:

- Modo -1: aún no ha encontrado ninguna línea en la ejecución del programa, es decir, es el primer fotograma en el que se realiza el tratamiento de imagen y búsqueda de línea.

- Modo 0: no ha encontrado ninguna línea en el instante anterior. Es posible que la última línea vista sea la izquierda, la derecha, desconocida o incluso ambas líneas (carril), de manera que debe considerarse cada uno de los casos en el caso de que vuelva a detectar una posible línea.

- Modo 1: encontró una línea en el fotograma anterior sea la derecha, izquierda o desconocida, habiendo guardado su centro de masas y cuál de ellas se trata para poder realizar un tratamiento diferente en cada caso.

- Modo 2: encontró las dos líneas (carril) en el fotograma anterior, guardando sus dos centros de masas de cada una de las dos líneas (izquierda y derecha).

Para cada uno de los casos se emparejará una línea con otra anterior siguiendo unas pautas similares y unos límites previamente fijados.

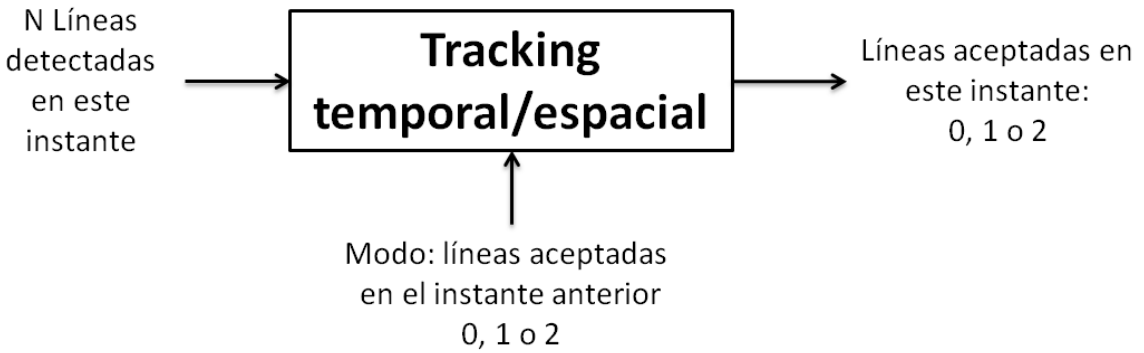


Fig. 44. Esquema básico del tracking temporal/espacial para el carril.

Al igual que antes lo que se utiliza para comparar con respecto a la posición anterior es el centro de masas de las líneas detectadas ya que utilizar también la orientación de las líneas es demasiado arriesgado ya que en el fotograma actual puede aparecer un objeto indeseado de orientación similar a la línea detectada anteriormente, pero que se encuentra muy separado de ella. En cambio con los centros esto es mucha más difícil que ocurra ya que entre instantes seguidos, el centro de masas de la línea no suele desplazarse demasiado y es muy difícil que además aparezca un objeto indeseado más cerca de la línea anterior que la propia línea algo desplazada.

Esta distancia entre el centro de la línea detectada en el fotograma anterior y línea o líneas detectadas en el fotograma actual se realizarán sobre las coordenadas de estos puntos transformados a vista planta teniendo en cuenta el factor de escalado de la imagen transformada destino con el objetivo de obtener la distancia de estos dos puntos en centímetros. El valor utilizado como límite en los casos que utilizan este filtro se toma de 4 centímetros ya que un salto demasiado grande indicará que no se trata de la línea correcta (*distancia_centros < 4cm).*

Por otro lado es posible que el salto de distancia de un punto al siguiente sea demasiado grande, por ejemplo, esto puede verse en la Fig. 43. donde la línea desaparece en una situación muy inferior (como mucho a 10 centímetros del coche) y vuelve a aparecen en una distancia lejana (aproximadamente a unos 30 centímetros del coche). Por tanto utilizando el filtro anterior se filtraría este caso, de manera que además debe tenerse en cuenta el movimiento realizado en el eje '*x*'. Este movimiento en este mismo caso es de pocos píxeles de manera que la línea podrá ser aceptada y realizar el seguimiento que toque respecto a ella. El ancho máximo de movimiento tomado donde se utiliza este filtrado es un desplazamiento no mayor a la octava parte del ancho de la imagen captada (*Δx < ancho_imagen/8*). De manera que si una línea cumple uno de estos dos requisitos será emparejada con la anterior y por tanto aceptada.

Además en el caso de que haya una única línea deberá decidirse si se trata de la línea izquierda, la línea derecha o si no se sabe qué línea es. En el caso de que se sepa que línea es se utilizará

un offset del valor de la mitad de la envergadura del uniciclo (5,5 cm) que se restará (línea izquierda) o se sumará (línea derecha) a la distancia entre la línea y el uniciclo para seguir la línea por dentro del carril. En el caso de que no se sepa que línea es se seguirá como si se tratará del seguimiento de una única línea hasta que encuentre el carril por primera vez. La orientación del camino a seguir será la de la línea detectada en cualquiera de los tres casos (Fig. 45.).

(Nota: en las imágenes se han utilizado tres colores para diferenciar los diferentes objetos: verde significa el camino a seguir, azul la línea o líneas detectadas, y dos puntos rojos, un objeto que se detectó como línea por el filtrado de características de objetos pero se discriminó como línea gracias al tracking temporal/espacial de carril).

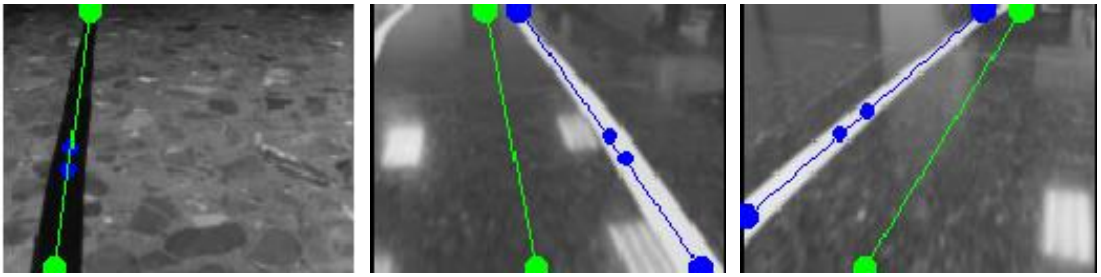


Fig. 45. Modo carril. (a) Línea desconocida. (b) Línea derecha. (c) Línea izquierda.

En el caso de que sea un carril lo que se haya detectado, el camino a seguir se calcula como la media de las distancias y orientaciones de las dos líneas que forman el carril.

$$Distancia = \frac{Dist.izq. + Dist.der.}{2} \qquad Orientación = \frac{Orient.izq. + Orient.der.}{2} \qquad (14)$$

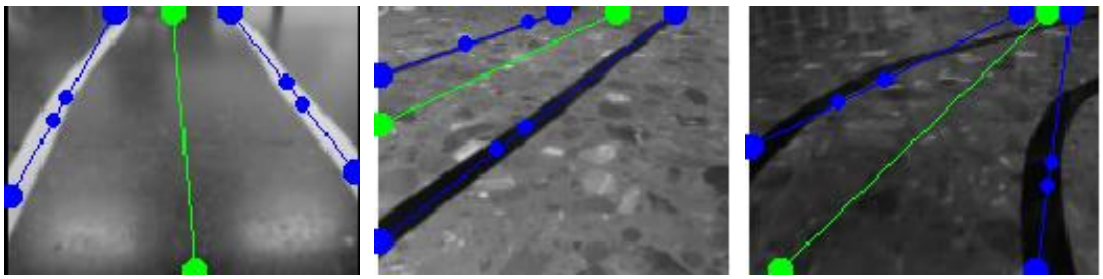


Fig. 46. Detección de varios carriles.

A continuación se explicará brevemente y se mostrará algunos resultados de algunas de las combinaciones posibles entre el modo (líneas aceptadas en instante anterior) y las líneas detectadas en el instante actual.

5.5.2.1 Detección de ninguna línea en el instante actual.

En este caso no se realizará ninguna modificación sobre la velocidad aplicada anteriormente a los motores. Además se irá acumulando el tiempo en el que el robot se encuentra perdido sin encontrar ninguna línea para realizar la parada de motores en el caso que se supere el tiempo máximo en el que el uniciclo puede estar perdido (explicado en el siguiente apartado).

5.5.2.2 Detección de 1 línea en el instante actual.

Se ha detectado una línea en el instante actual de la que se conoce su centro de masas, orientación, distancia al origen y su valor de '*área_elipticidad*'.

- Modo -1: en este caso no ha encontrado ninguna línea de manera que la línea es aceptada pero no se sabe cuál es, si la izquierda o la derecha. Por tanto se seguirá sin realizar la suma o resta del offset comentando anteriormente, tal y como si fuera el seguimiento de una única línea. (Fig. 45. (a)).

- Modo 0: en este caso hay 4 posibilidades y es lo que se haya visto por última vez antes de haberse perdido: que la última línea vista sea la derecha, que la última línea vista sea la izquierda, que la última línea vista no se sabía cuál era o que en el instante anterior se detectó el carril (línea izquierda y derecha).

En el caso de que fue una línea lo que se detectó por última vez tras haberse perdido se procede de la misma manera en los 3 casos, si la distancia es menor a 4 cm o el cambio de en el eje '*x*' es menor a una octava parte del ancho de la imagen, se acepta, sino se sigue estando perdido. Esto ocurre principalmente cuando una línea desaparece y vuelve a aparecer por un lado de la imagen aunque en el caso del carril no es tan fácil que esto ocurra como en el caso del seguimiento de una única línea (Fig. 47.). De manera que la línea emparejada sigue siendo la misma que antes (izquierda, derecha o desconocida).

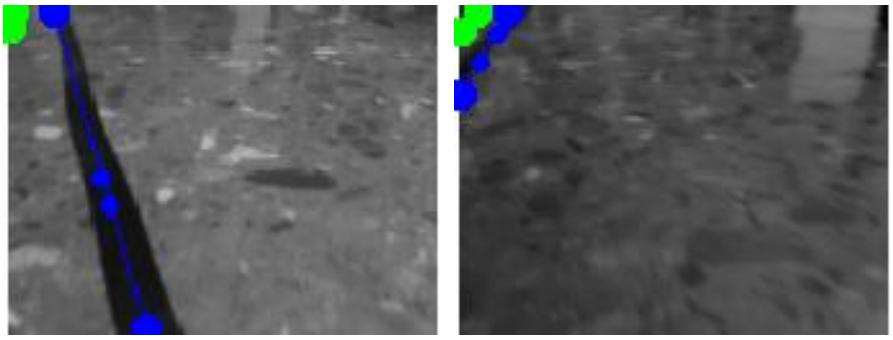


Fig. 47. Línea derecha desaparece y vuelve a aparecer.

En el caso de que lo último visto sean las dos líneas se intenta emparejar la línea detectada con una de estas últimas dos utilizando el mismo criterio que en el caso anterior para cada uno de los centros. Si no se empareja por ningún criterio se sigue perdido.

- Modo 1: en este caso es muy simple, se intenta emparejar la línea detectada con la línea aceptada en el fotograma anterior de la misma manera que antes (*distancia_centros* < *4cm* || Δx < *ancho_imagen/8*). Si se empareja sigue en el modo 1 y si no pasa al modo 0, es decir, se pierde.

- Modo 2: en este caso se intenta emparejar de la misma manera la línea detectada con una de las dos líneas aceptadas en el fotograma anterior. Si se empareja pasa al modo 1 y si no pasa al modo 0.

5.5.2.3 Detección de 2 líneas en el instante actual.

Se han detectado dos posibles líneas en el instante actual y se emparejaran con líneas anteriores dependiendo de lo que se haya detectado en el instante anterior. Este caso es más complejo al anterior ya que deben barajarse todas las posibilidades, que las dos líneas sean malas, que una sea mala o que las dos sean buenas, detectando así el carril.

Para poder aceptar las dos líneas (carril), en cualquier caso, deben darse varías condiciones mínimas fijas que dependen de la geometría instantánea de las dos líneas, es decir, no dependen de la variación temporal pero sí de la espacial. Estas son las siguientes:

- Diferencia entre las orientaciones de ambas líneas: esta diferencia alcanza su valor máximo en las curvas más pronunciadas y aun así no es un valor demasiado grande. Viendo el sistema de referencia se puede ver que la orientación varía entre ±90° y se han filtrado a ±72° con el filtrado de características de objetos, de manera que la diferencia máxima posible es de 144°. Observando la diferencia máxima de orientaciones que suele haber en los casos más extremos de curvas esta no llega a superar ni siquiera los 50°, por tanto este será nuestro primer filtro.

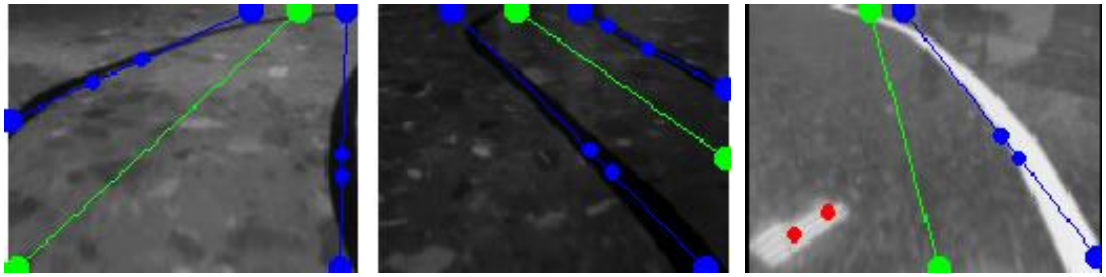


Fig. 48. Diferencia de orientaciones. (a) Aceptada: 11°. (b) Aceptada: 7°. (c) No aceptada: 77°.

- Ancho del carril: dadas las distancias de cada una de las dos líneas al centro de coordenadas (IGR del uniciclo), puede obtenerse el ancho del carril en ese instante como la diferencia de estas distancias. De manera que si se obtiene el ancho de carril en centímetros y se sabe que la envergadura del uniciclo es de 11 centímetros, puede tomarse un rango de valores alrededor de este a partir de los cuales se despreciará una de las dos líneas por dar un valor incorrecto de ancho de carril. En las rectas este ancho de carril permanece prácticamente constante pero en las curvas llega a variar bastante debido a la no parametrización de las curvas. Estos valores límite de ancho carril se fija entre 3 y 35 cm, de manera que un valor por encima o debajo de este rango hará que las dos líneas no se acepten como carril. Este rango es bastante grande ya que se desea aceptar el carril aunque su ancho sea bastante variable. Un ancho de carril menor de 3 cm suele ocurrir con líneas cortadas y un ancho mayor de 35 cm suele ocurrir cuando los objetos están muy separados y apuntan en direcciones opuestas. (Fig. 49.)

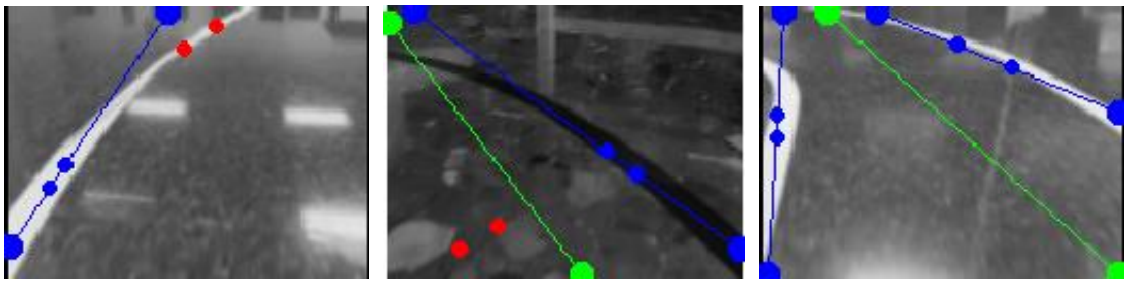


Fig. 49. (a) No aceptado: Ancho carril menor de 3 cm (línea cortada). (b) No aceptado: Ancho carril mayor de 35 cm (objeto molesto). (c) Aceptado: Ancho carril de 20 cm (entre 3 y 35 cm).

- Posición en el eje '*x*' de la línea izquierda y derecha: es imposible que la línea derecha este más a la izquierda en la imagen que la línea izquierda, o viceversa, de manera que si esto ocurre ya no se aceptarán las dos líneas. Para saber qué línea es la izquierda y cuál es la derecha se comparan las distancias al centro de coordenadas (IGR del uniciclo), y conociendo el sistema de referencia se sabe que la distancia de la línea izquierda siempre será mayor que la de la línea

derecha. De esta manera se filtraran objetos mal orientados que se encuentran o a la derecha de la línea derecha o la izquierda de la línea izquierda.

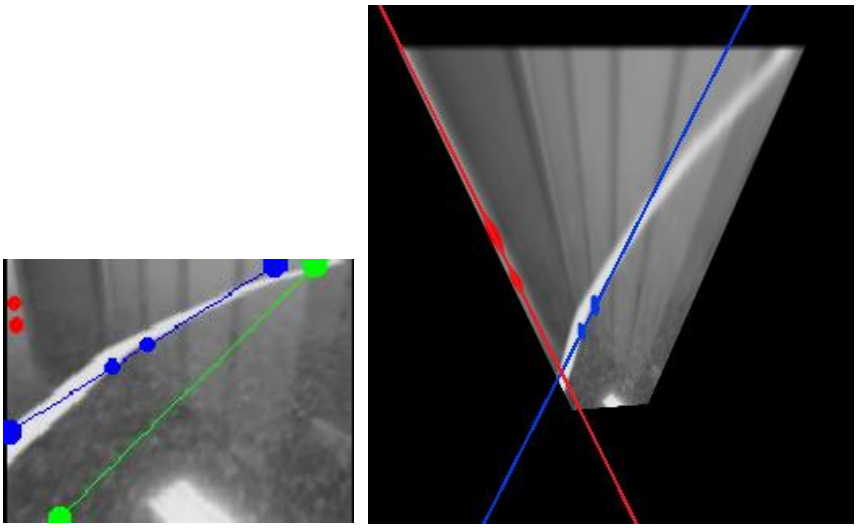


Fig. 50. Situación imposible: línea derecha (roja) está a la izquierda de la línea izquierda (azul).

Una vez se han descrito las condiciones mínimas para aceptar un carril se pasa a explicar el emparejamiento que es similar al anterior pero además teniendo estas condiciones mínimas en cuenta para poder entrar al modo 2.

- Modo -1: si se cumplen las condiciones mínimas de carril comentadas se acepta el carril, y si no se para la ejecución del programa ya que en el primer fotograma no es capaz de encontrar ninguna línea. En el caso de que una de las líneas tenga un valor de '*área_elipticidad*' elevado (más de 50 mil), se acepta como una línea desconocida y se pasa al modo 1.

- Modo 0: en este caso se realiza una especie de "cortocircuito" de manera que si se detectan dos líneas con un '*área_elipticidad*' muy elevado (100 mil), además de las condiciones mínimas, se acepta el carril, ignorando cuál fue el estado anterior tras haberse perdido. Esto es útil cuando ha habido un salto bastante grande a causa de un retardo demasiado grande de manera que es capaz de reengancharse al seguimiento del carril independientemente de cuales fueran las líneas aceptadas antes de haberse perdido.

Si los valores '*área_elipticidad*' no son tan elevados se pasa al emparejamiento temporal de las líneas, que depende de cuál fue la última o últimas líneas detectadas antes de perderse.

En el caso de que hubiera detectado una línea antes de perderse se realiza el emparejamiento anteriormente explicado (*distancia_centros* < *4cm* || Δx < *ancho_imagen/8*), y únicamente intenta emparejar una línea. En el caso de que no se empareje con ninguna de las dos detectadas se pasa al modo 0. Si se consigue emparejar con una de las dos líneas se calculan las condiciones mínimas de carril y en el caso que se cumplan pasa al modo 2, sino únicamente se acepta la línea aceptada en primera instancia y se pasa al modo 1.

En el caso de que hubiera detectado un carril realiza el mismo procedimiento, pero en este caso se puede seguir en el modo 2, o pasar al modo 1 o 0.

- Modo 1: se realiza el emparejamiento de una de las líneas actual con la línea aceptada en el fotograma anterior. Si ninguna se empareja se pasa al modo 0 (en el que en el caso de que el carril sea correcto se aplicará el 'cortocircuito' y el carril será aceptado en el siguiente fotograma). Si se empareja una de las dos con (*distancia_centros* < *4cm* || Δx < *ancho_imagen*

/8) se intenta calculan las condiciones mínimas de carril con la otra línea, si se cumplen pasa al modo 2, sino se continúa estando en el modo 1.

- Modo 2: este es el estado más complejo de todos hasta ahora ya que de las dos líneas detectadas pueden ser erróneas las dos, sólo una o ninguna, de manera que se debe ser capaz de realizar un emparejamiento doble y además cumplirse las condiciones mínimas de carril. Si sólo se consigue emparejar una, se tiene en cuenta a cuál de las dos anteriores corresponde (izquierda o derecha, ya que una vez se encuentra el carril no se vuelve a tener una única línea como desconocida) y se actúa en consecuencia pasando al modo 1. Si no se consigue emparejar nada se pasa al modo 0.

5.5.2.4 Detección de 'N' líneas en el instante actual.

Debido a la naturaleza del filtro de objetos que precede al tracking, es posible que hayan llegado hasta aquí más de dos líneas, de manera que se realizará un emparejamiento por cercanía relativa y no absoluta entre líneas, es decir, en el caso del modo 2, se tomaran los dos objetos más cercanos a estas dos líneas. Como esto no es del todo correcto se harán pasar estos dos objetos detectados como los más cercanos a las líneas anteriores por el filtrado realizado en el caso de cuando son detectadas 2 líneas. Así se será igual de selecto y preciso que en ese caso. Esto se realizará de la misma forma para cuando el modo sea 1 y 0.

Por otro lado en el caso de que el modo sea -1, es decir, que se vean más de dos posibles líneas en el primer fotograma y aún no se disponga de información temporal, se realizará la decisión por el valor de '*área_elipticidad*'. Para aceptar dos líneas, las dos líneas de mayor valor tendrán que superar los 50 mil además de ser el doble de grandes que el tercer valor de '*área_elipticidad*' más grande. En el caso de que no se cumpla se intentará al menos aceptar una línea de la misma manera pero respecto al objeto con el segundo valor mayor. Si no, no llegará a ejecutarse el programa y será necesario saber que objeto está interfiriendo en la detección de la línea.

Esto será útil para discriminar cualquier tipo de objetos, desde líneas pintadas dentro del carril, como otras líneas que se encuentren fuera del carril.

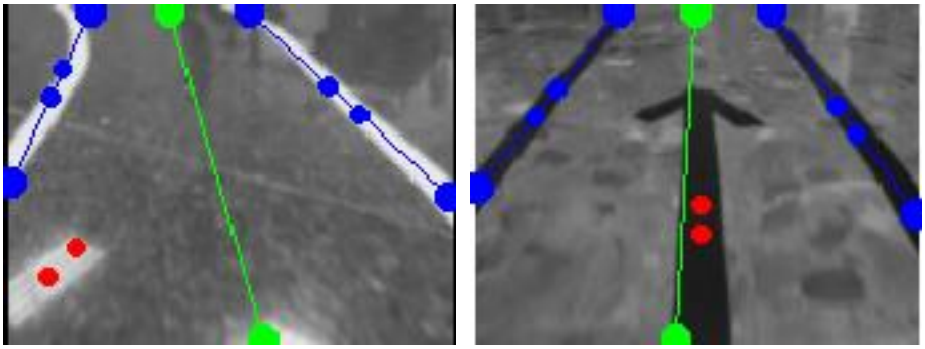


Fig. 51. Decisión temporal correcta con 3 objetos detectados como líneas.

5.5.3 Parada de movimiento al no encontrar un camino a seguir.

Un tema muy importante acerca del movimiento del robot es decidir en qué momento debe pararse el robot en el caso de que ya no tenga ningún camino que seguir, es decir, o se haya perdido o el camino haya terminado. Este cálculo se realiza principalmente para que el uniciclo pare automáticamente en el caso de que se pierda, de manera que no haya que pararlo

manualmente. Esto es ejecutado exactamente igual en ambos algoritmos (seguimiento de una única línea y de carril).

Para decidir en qué momento debe pararse se ha utilizado una variable dependiente de la velocidad lineal constante que lleva el uniciclo en el transcurso del seguimiento que indica el tiempo máximo sin ver línea, de manera que una vez se haya superado este tiempo, el robot simplemente pare sus motores y termine la ejecución del programa. Este tiempo es dependiente de la velocidad lineal ya que si por ejemplo fijáramos un valor de tiempo y aumentáramos la velocidad al doble, el uniciclo se pararía en el doble de espacio en este último caso. Esto no es lo que queremos de manera que buscamos un tiempo que se ajuste bien a una velocidad lineal determinada y para independizar este valor hacemos que dependa de ella de manera inversamente proporcional. Así conseguimos indicar al uniciclo, en cuanto espacio recorrido sin ver línea debe pararse, y no en cuanto tiempo.

Para una velocidad de 7 cm/s, un tiempo de 5 segundos sin ver línea se adapta bien a este caso (recorre 35 cm estando perdido), ya que tampoco se debe escoger un valor demasiado bajo porque en las curvas hay momentos en los que es posible dejar de ver la línea y no se quiere que el uniciclo se sienta perdido en estos casos. Entonces el tiempo sin ver línea se toma como:

$$tiempo\ máximo\ modo\ 0\ (s) = \frac{35\ cm}{v\ (\frac{cm}{s})} \qquad (15)$$

De manera que se realizará un sumatorio del tiempo que se está perdido y una vez encuentre la línea de nuevo se reiniciará este valor. En el caso de que este sumatorio de tiempo perdido supere al tiempo máximo establecido, se pararán los motores del uniciclo y la ejecución del programa.

Por otro lado debe considerarse los momentos en los que no hay línea y de manera fugaz y puntual a causa de reflejos u otros artefactos en la imagen se detecta una línea, pero a continuación sigue sin detectarse línea. En este caso no se querrá reiniciar el sumatorio del tiempo que indica cuánto tiempo lleva el uniciclo perdido, para ello se fija un valor de tiempo de transición de manera que si durante este tiempo de transición se ha detectado línea en todos los fotogramas, sí reinicie la cuenta de tiempo del uniciclo perdido. El tiempo de transición escogido es de 0,5 segundos, aunque podría escogerse cualquier otro si no es ni muy pequeño ni muy grande.

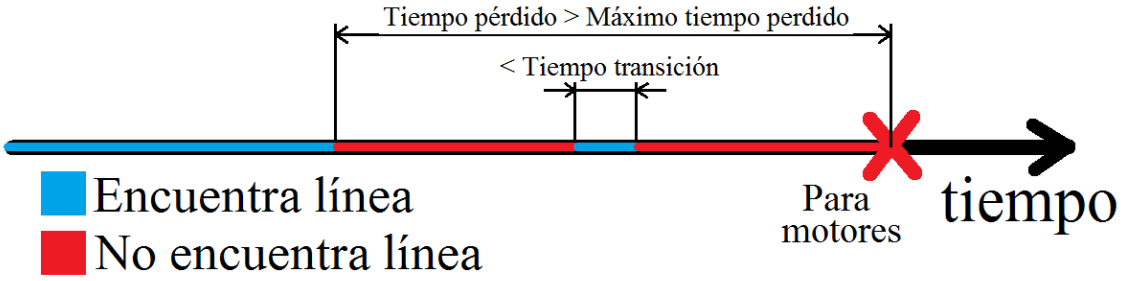


Fig. 52. Esquema explicativo de parada de motores con el tiempo de transición.

Así queda solucionada la parada del movimiento en la ejecución del programa de seguimiento de línea o de carril.

5.6 Estudio geométrico de la posición relativa del uniciclo.

Este proceso, una vez se conoce el punto y la orientación que parametrizan el camino (línea) que desea seguirse, se calcula de la misma manera en ambos algoritmos (una línea y en carril para las líneas encontradas). Este proceso se realiza una vez se ha realizado el filtrado de líneas y ya teniendo el punto y la orientación que parametrizan la línea escogida. Este proceso se explicará de manera secuencial y está ligado con el siguiente apartado '*5.7. Aplicación del modelo cinemático del uniciclo en el GoPiGo*'.

5.6.1 Orientación y punto a dos puntos.

Esta operación es un cálculo geométrico simple, se trata de obtener un segundo punto de la recta a partir del primero y de la orientación del objeto. Para ello se toma una distancia fija 'D_y' (en este caso se tomó $D_y = 10$ *píxeles*) para calcular el segundo punto de la recta. Esto se realiza con el objetivo poder tener dos puntos de la recta en las coordenadas normales de captación de la imagen, para a continuación poder transformar estos dos puntos haciendo uso de la matriz de homografía y poder realizar el cálculo del estado del robot explicado en el siguiente apartado.

$$Pto2 = (\, y_2 = y_1 - D_y \,, \qquad x_2 = x_1 + D_y \cdot tan(\theta) \,) \,; \qquad con \quad D_y \equiv cte \tag{16}$$

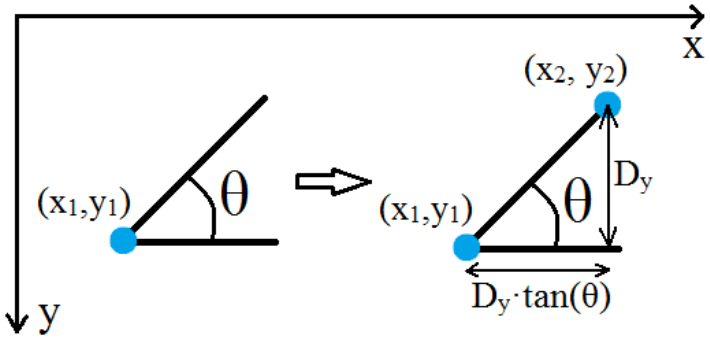


Fig. 53. Esquema explicativo para el cálculo de un segundo punto de la línea recta: (x_2, y_2).

5.6.2 Dos puntos a estado [d, θ].

Previo a este cálculo se normalizan los dos puntos obtenidos de la función anterior tal y como se explica en un apartado posterior (*5.8.2.1. Normalización de puntos*). Una vez se tienen los dos puntos normalizados respecto al tamaño de imagen con el que fue tomada la calibración, se realiza el cálculo del estado [*d, θ*].

Lo primero que se realiza es la transformación de los dos puntos mediante la matriz de homografía proporcionada por el archivo de automatización. De manera que en este momento ya se tienen las coordenadas de los dos puntos transformados (en centímetros, dimensiones reales).

A partir de estos se realiza un cálculo geométrico de la distancia a la que se sitúa el uniciclo de esta línea y de la orientación que tiene respecto a esta, respecto al sistema de referencia (*4.2.2. Seguimiento de camino*. Fig. 10. (a).).

La distancia a la línea detectada (recta 1) se calcula respecto a la perpendicular (recta 2) a esta que pasa por el origen de coordenadas (punto de rotación instantánea del uniciclo, IGR) y debe ponderarse por el factor de escalado con el que fue tomada la matriz de homografía, y que por tanto, esta distancia de como resultado centímetros reales. Estos cálculos se exponen en las siguientes ecuaciones:

$$m_1 = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \rightarrow m_2 = \frac{-1}{m_1} \rightarrow \begin{cases} Recta\ 1: & y - y_1 = m_1 \cdot (x - x_1) \\ Recta\ 2: & y - y_0 = m_2 \cdot (x - x_0) \end{cases} \quad (17)$$

$$Punto\ de\ corte: (x_C = \frac{m_1 \cdot x_1 - y_1 - m_2 \cdot x_0 + y_0}{m_1 - m_2}, y_C = m_2 \cdot (x - x_0) + y_0) \quad (18)$$

$$\text{Estado del uniciclo } [d, \theta] = \begin{cases} Distancia = \sqrt{(x_C - x_0)^2 + (y_C - y_0)^2} \\ Orientación = acos(\frac{y_2 - y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}) \end{cases} \quad (19)$$

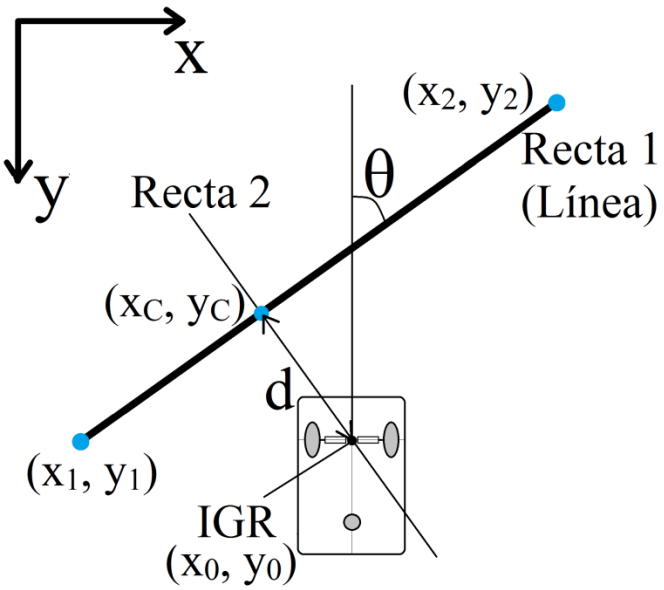


Fig. 54. Esquema explicativo para el cálculo del estado [*d, θ*] a partir de dos puntos de la línea y el origen de coordenadas.

5.7 Aplicación del modelo cinemático del uniciclo en el GoPiGo.

5.7.1 Estado [d, θ] a velocidad angular 'w'.

Esta función realiza el cálculo de '*w*' tal y como se explicó (*5.1. Flujograma completo del algoritmo desarrollado*. Fig. 12.): calculando el ángulo deseado 'θ_D', a continuación el error respecto de la orientación actual 'θ', y realizando un control *PID* (que acabará únicamente siendo proporcional), sobre el error para el cálculo de '*w*'. Esto por tanto devuelve el valor de la velocidad angular en '*radianes/ segundo*'.

5.7.2 Velocidad angular 'w' a velocidad aplicada a los motores del uniciclo [v_L, v_R].

Obtener la velocidad en '*centímetros/segundo*' que debe aplicarse por separado a cada motor se realiza de forma directa a partir de la velocidad angular tal y como se explicó (*4.1.1. Modelo cinemático*).

Una vez hemos calculado la velocidad necesaria en '*centímetros/segundo*' a aplicar en cada una de las ruedas del uniciclo se tiene un pequeño problema, ya que la librería de nuestro uniciclo, el GoPiGo, permite modificar la velocidad de cada una de las ruedas dando valores de 0 a 255 bits, mediante la función '*motor1*' y '*motor2*', y no aplica ningún tipo de conversión de '*centímetros/segundo*' a esta unidad.

Una primera hipótesis de partida que tomamos con el objetivo de mapear valores de '*centímetros/segundo*' a '*bits*' es que probablemente el mapeo de valores se podrá ajustar con una recta, esperando que el aumento de '*centímetros/segundo*' se traduzca a un aumento lineal y proporcional de '*bits*'. Para esta primera hipótesis también tendremos en cuenta que el GoPiGo puede ser alimentado con diferente voltaje, teniendo un voltaje diferente si está conectado por ejemplo mediante el enchufe de toma de tierra (4.5 Voltios) o directamente a la batería (9-12 Voltios).

Para observar este comportamiento se ha realizado un pequeño programa con el objetivo de experimentar e intentar conseguir una relación de velocidades. Este programa utiliza una función perteneciente a la librería '*gopigo.c*' llamada '*enc_read*'. Esta función, como se explicó, es un '*wheel encoder*' que devuelve el número total de '*ticks*' que ha contado desde que las ruedas se movieron por primera vez.

Para ello, se necesita mapear de '*centímetros/segundo*' a '*vueltas/segundo*', esta conversión es muy sencilla ya que conociendo el diámetro de la rueda (6'5 cm) puede calcularse el perímetro de la rueda, que será la distancia recorrida para cada vuelta, y con ello la conversión:

$$\frac{v\ (^{cm}/_{s})}{2 \cdot \pi \cdot R\ (^{cm}/_{vuelta})} = \frac{v\ (^{cm}/_{s})}{20{,}42\ \ (^{cm}/_{vuelta})} = \frac{v}{20{,}42}\ \ (^{vueltas}/_{s}) \qquad (20)$$

Con la función '*enc_read*' se trata de saber cuántos '*ticks*' se han capturado en un intervalo de tiempo medido, y de esta manera y sabiendo que el número de '*ticks*' en una vuelta completa en el GoPiGo es de 18, podremos saber el número de '*vueltas/segundo*' que ha dado para una determinada velocidad en '*bits*', pudiendo mapear valores. Realizando este cálculo para un rango de velocidades entre 0 y 255 bits:

$$\frac{\Delta tick\ (ticks)}{\Delta t\ (s)} : 18\ ^{ticks}/_{vuelta} = \frac{(tick_{final} - tick_{inicial})(ticks)}{\Delta t\ (s)} : 18(^{ticks}/_{vuelta}) = \frac{(tick_{final} - tick_{inicial})}{\Delta t \cdot 18}\ (^{vueltas}/_{s}) \qquad (21)$$

Además esta función guardara el voltaje medio del GoPiGo (media de voltaje al comienzo y final de la ejecución del programa). De esta manera podremos comparar los valores de velocidad en '*vueltas/segundo*' cuando hay diferentes voltajes.

Las tablas de valores que se guardaron ejecutando este programa dos veces para dos voltajes diferentes (cable de alimentación y con batería de pilas) es la siguiente:

Voltaje: 4,394 Voltios	
Vel. (bits)	**Vel. (vueltas/s)**
100	0,32

Voltaje: 9,277 Voltios	
Vel. (bits)	**Vel. (vueltas/s)**
100	0,73

110	0,36	110	0,82
120	0,41	120	0,90
130	0,46	130	0,97
140	0,49	140	1,05
150	0,52	150	1,12
160	0,56	160	1,20
170	0,61	170	1,27
180	0,64	180	1,38
190	0,68	190	1,47
200	0,73	200	1,55
210	0,79	210	1,67
220	0,82	220	1,76
230	0,88	230	1,84
240	0,93	240	1,93
250	0,97	250	2,02

Tabla 9. Valores de mapeo de velocidades resultante de la ejecución del programa.

Para observar la gráfica resultante del mapeo de estos valores se utiliza Matlab, además nos permitirá ajustar estos valores a una recta (polinomio de 1er grado) mediante la función '*polyfit*'.

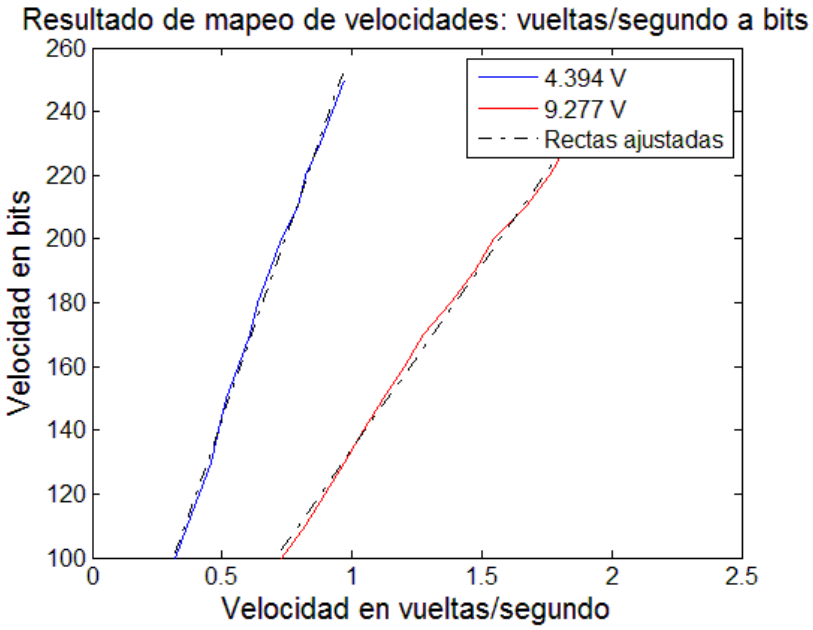


Fig. 55. Gráfica del mapeo de velocidades con dos voltajes diferentes: 4'4 y 9'3 Voltios aproximadamente, ajustados con rectas.

En la Fig. 55. se puede observar cómo se cumple la hipótesis de partida, donde el mapeo entre valores de velocidad puede ajustarse mediante una recta perfectamente, además se puede observar que se obtiene una mayor velocidad en '*vueltas/segundo*' para el mismo número de '*bits*' de entrada en la función '*motor1*' o '*motor2*' para un voltaje superior.

Para modelar la dependencia de la velocidad con el voltaje, podemos observar que el cociente entre los valores de velocidad en '*vueltas/segundo*' entre los voltajes 9'277 y 4'394 es aproximadamente constante e igual al cociente entre voltajes aplicados (9'277/4'394 = 2'111) y puede verse en la siguiente Tabla. 10.

	Vel. (vueltas/s)		
Vel. (bits)	**9,277 V**	**4,394 V**	**$Vel._{9'277}$ / $Vel._{4'394}$**
100	0,73	0,32	2,286
110	0,82	0,36	2,250
120	0,90	0,41	2,186
130	0,97	0,46	2,133
140	1,05	0,49	2,156
150	1,12	0,52	2,177
160	1,20	0,56	2,135
170	1,27	0,61	2,100
180	1,38	0,64	2,167
190	1,47	0,68	2,156
200	1,55	0,73	2,125
210	1,67	0,79	2,114
220	1,76	0,82	2,149
230	1,84	0,88	2,086
240	1,93	0,93	2,082
250	2,02	0,97	2,078

Tabla 10. Cociente entre valores de velocidad de los dos voltajes aplicados.

De esta manera podemos realizar un mapeo de velocidades de '*vueltas/segundo*' a '*bits*', teniendo en cuenta el voltaje aplicado. Para ello deberá tomarse una recta de referencia junto al voltaje de esta. En este caso se toma la recta de referencia la de voltaje 9'277 y con la función '*polyfit*' de Matlab obtenemos sus coeficientes:

$$Vel\ (bits) = 115{,}7985 \cdot Vel\ (^{vueltas}/_{segundo})_{9{,}277Voltios} + 18{,}1987 \qquad (22)$$

Y la conversión final utilizada en el algoritmo realizado para un voltaje cualquiera aplicado al GoPiGo será la siguiente:

5.8 Diferentes automatizaciones previas a la búsqueda y seguimiento del camino.

A continuación se expone todo aquello que realizan los dos algoritmos realizados (seguimiento de línea y seguimiento de carril) previo a la ejecución del bucle de búsqueda de líneas. En este caso ambos realizarán lo mismo y lo único que cambia uno respecto a otro es la función esqueleto del algoritmo.

Ambos programas permiten el guardado de estadísticas acerca de los parámetros generales de su ejecución que afectan a todos los fotogramas por igual (velocidad lineal constante 'v', distancia de observación 'L', constantes PID, velocidad angular de saturación 'w_{max}', ángulo deseado de saturación '$angulo_{max}$', ancho de línea, coeficientes de la recta ajustada al filtrado morfológico) y del procesado que realiza en tiempo real, tanto del procesado de imagen (media, máximo, umbral), como de la parte robótica (ángulo deseado, velocidad angular aplicada 'w', velocidad aplicadas a cada rueda [v_L, v_R]), del estado [d, θ], indicando si encontró línea y del tiempo que ha tardado en procesar cada fotograma. Esto es útil para poder graficar temporalmente el movimiento que el uniciclo ha realizado.

5.8.1 Independizar del color de la línea.

Hay dos casos posibles, cuando la línea es oscura y el fondo claro, y cuando la línea es clara y el fondo oscuro. Lo explicado en el tratamiento de imagen hasta la binarización se ha realizado utilizando el primer caso, pero la única diferencia con el segundo es que en este debe realizarse una apertura en vez de un cierre en el filtrado morfológico. Un cierre se trata de la realización de una dilatación y a continuación una erosión, mientras que la apertura es el proceso contrario. De manera que se ajusta perfectamente a las necesidades que requiere este caso, ya que hace todo lo contrario al cierre.

Para independizar el proceso del color de la línea y sea capaz de seguir tanto líneas blancas (claras) como negras (oscuras), debe realizarse un cálculo previo al bucle de búsqueda y detección de líneas, y de aplicación de velocidades. Este procesado previo consiste en tomar la primera imagen (sea de vídeo o cámara) y realizar el filtrado morfológico de un cierre y una apertura en paralelo. A estos dos resultados se les aplicará el filtrado de características explicado anteriormente, realizando el sumatorio de todos los valores '*área_elipticidad*' de los objetos que fueron detectados como líneas. Una vez realizado se tendrán dos valores, uno perteneciente al cierre y otro a la apertura, de manera que el mayor de los dos dictará si se trata de una línea negra o blanca. Este cálculo podría realizarse en tiempo real pero retardaría demasiado el proceso, por tanto se decide que se compute al comienzo, de manera que para cada ejecución, el uniciclo sólo será capaz de seguir o líneas negras o blancas.

Este cálculo se realiza de la misma manera para el algoritmo de carril, ya que realiza el sumatorio de los valores '*área_elipticidad*' de los objetos detectados como líneas.

En la Fig. 56. se muestra el proceso que sigue este cálculo. Primero, a partir de la imagen original, calcula la imagen luminancia, y esta se filtra utilizando el residuo entre cierres y aperturas en paralelo. A continuación se segmentan y se filtran utilizando el proceso explicado de '*área_elipticidad*', en este punto se tienen todos aquellos objetos que fueron detectados como líneas. Como puede verse, en los tres primeros casos hay líneas negras, de manera que se filtra correctamente aplicando el cierre, y en los tres últimos casos hay líneas blancas, de manera que se filtra correctamente aplicando la apertura. Y la decisión de aplicar o cierres o aperturas será lo que se devolverá a la función principal.

En el único caso donde no se realiza correctamente es en la Fig. 56. (e). En esta puede verse como aun habiendo una línea blanca, se ha detectado un objeto como línea negra en la

aplicación del cierre. A pesar de ello, como el objeto detectado por la apertura tiene una '*área_elipticidad*' mayor entonces se devolverá que el filtro a realizar es la apertura.

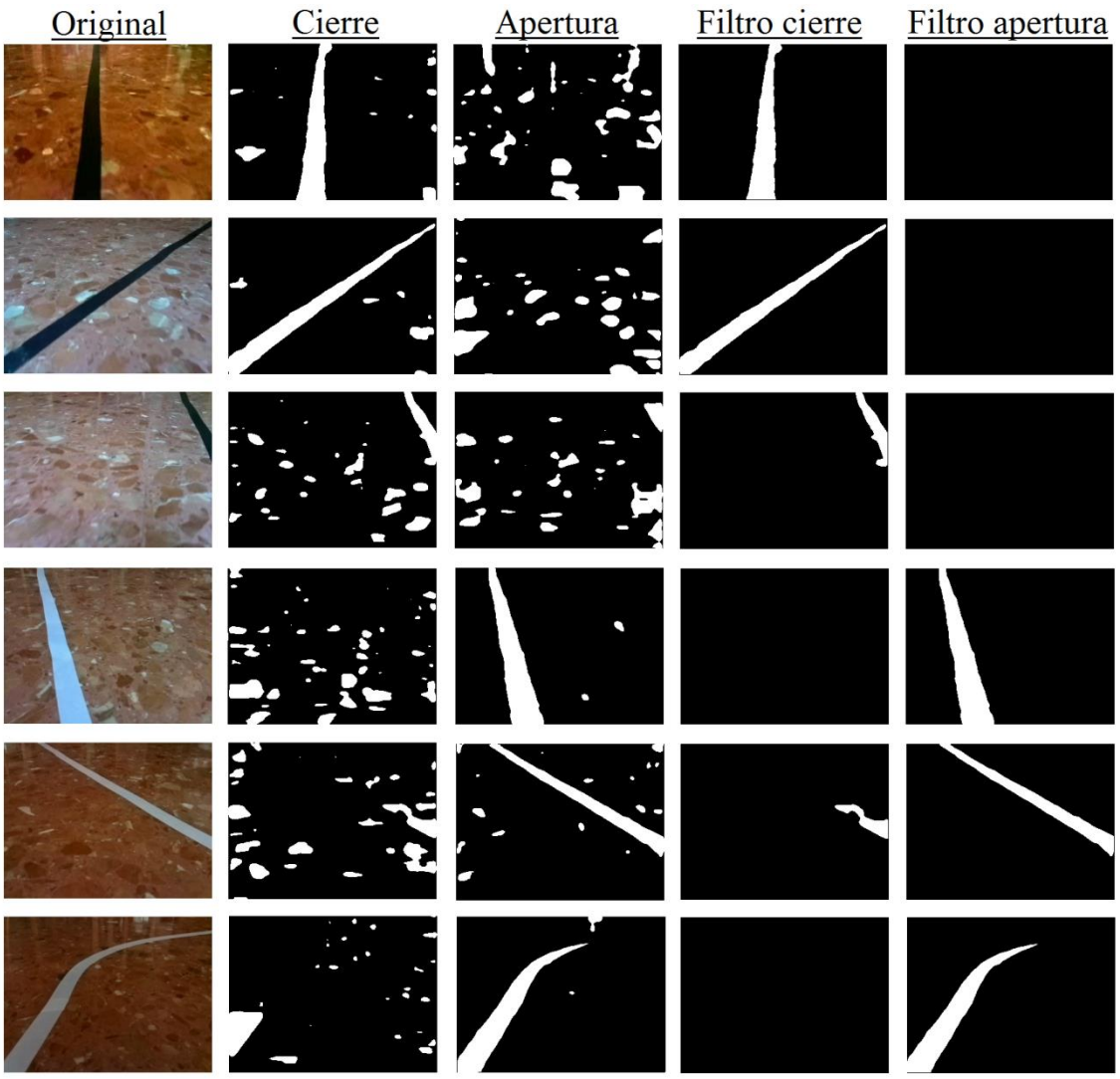


Fig. 56. Comparación entre el uso de cierre y apertura en líneas negras y blancas (a) (b) (c) (d) (e) (f).

5.8.2 Independizar del tamaño de la imagen captada y del ancho de la línea.

Estos cálculos buscan independizar el proceso de captación y detección de líneas del tamaño de la imagen captada, de manera que pueda tomarse, por ejemplo, una imagen la mitad de pequeña y siga funcionando correctamente, con el objetivo de realizar menos cálculos en el tratamiento de imagen y que haya un menor retardo. El procesado de imagen (cálculo de luminancia, filtrado paso bajo, filtrado morfológico, residuo, binarizado y segmentado) se realizará cuatro veces más rápido cada vez que se tome una imagen la mitad de grande en las dos dimensiones (ancho y alto).

Lo primero que debe calcularse son los factores de reducción respecto a la imagen de calibración en el eje '*x*' e '*y*' por separado, y también el factor de reducción total de la imagen en ambos ejes. Estos factores serán utilizados para la ponderación de varios procesos que se exponen a continuación.

```
-   factor_x = ancho_imagen / ancho_imagen_calibracion;
-   factor_y = alto_imagen / alto_imagen_calibracion;
```

```
-   factor_size = (ancho_imagen_calibracion*alto_imagen_calibracion) / (ancho_imag
    en * alto_imagen);
```

5.8.2.1 Normalización de puntos.

Principalmente los factores de reducción en los ejes '*x*' e '*y*' se utilizan para normalizar los dos puntos que parametrizan la línea encontrada al tamaño con el que se realizó la homografía y de esta forma calcular correctamente los puntos transformados. Si se quisiera transformar la imagen entera y no se dispone de una matriz de homografía para cada tamaño posible de imagen captada, entonces antes de transformarla debería realizarse un cambio de tamaño '*resize*' para que tuviera el mismo tamaño con el que fue tomada la imagen calibradora.

```
-   punto1_x = punto1_x/factor_x;
-   punto1_y = punto1_y/factor_y;
-   punto2_x = punto2_x/factor_x;
-   punto2_y = punto2_y/factor_y;
```

5.8.2.2 Coeficientes de la recta ajustada para el filtrado morfológico.

El programa tiene como dato de entrada introducido por el usuario el ancho de la línea que se desea seguir, aunque al tomar unos márgenes de seguridad bastante grandes en el filtrado morfológico, si se toma un tamaño de línea más o menos normal (1,5 ~ 2 cm), cualquier ancho de línea podrá ser filtrado.

Los coeficientes de la recta ajustada para el filtrado morfológico en la calibración fueron calculados con un tamaño de imagen y con un ancho en específico, de manera que deben independizarse doblemente.

- Independizar del cambio de tamaño en cada eje. Cuando la imagen se hace menos ancha (eje '*x*'), la recta se hace menos ancha proporcionalmente, de manera que debe multiplicar el factor del eje '*x*' por cada coeficiente. Y cuando la imagen se hace menos alta (eje '*y*'), el comienzo y el salto que debe dar por cada píxel de altura más debe ser mayor, por tanto se divide el factor del eje '*y*' por cada coeficiente.

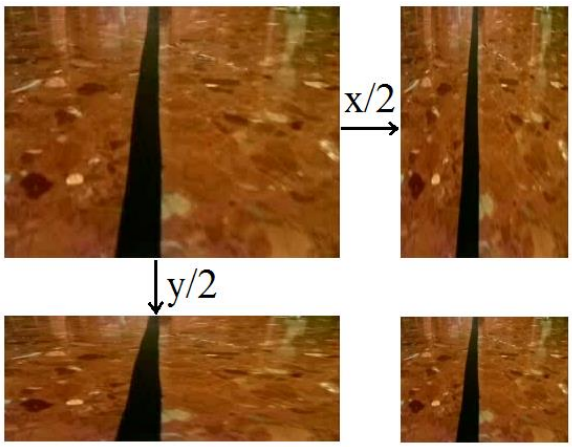


Fig. 57. Ejemplo de reducción de tamaño de la imagen a la mitad en cada eje junto a los coeficientes de la recta ajustada para el filtrado morfológico.

```
-   coeficientes_size(2) = (coeficientes_size(2) * factor_x) / factor_y;
-   coeficientes_size(1) = (coeficientes_size(1) * factor_x) / factor_y;
```

- Independizar del ancho de línea tomado en la imagen de calibración. En la imagen de calibración los coeficientes de la recta ajustada se calcularon mediante un ancho en

específico (en este caso el lado de los cuadrados que era de 5 cm), pero se necesita ajustarlos a un ancho de línea típico (1,5 ~ 2 cm). Esto simplemente se realizará normalizando proporcionalmente en ambos coeficientes de la siguiente manera:

$$Coeficiente = Coeficiente \cdot \frac{ancho\_linea\_actual}{ancho\_linea\_calibración} \quad (23)$$

```
- coeficientes_size(2) = (coeficientes_size(2) * ancho_linea_actual) / coeficien
  tes_size(0);
- coeficientes_size(1) = (coeficientes_size(1) * ancho_linea_actual) / coeficien
  tes_size(0);
```

5.8.2.3 Filtrado paso bajo 'blur'.

El filtrado paso bajo se trata de un filtro local que se va deslizando a lo largo de los píxeles de la imagen, dando un valor a cada píxel de salida dependiente de el mismo y de sus vecinos (máscara de convolución). De manera que, por ejemplo, al tener una imagen la mitad de tamaño que con la que se realizó la decisión de tomar una '*sigma = 2*' en este filtrado paso bajo (Gaussiano), al variar el tamaño de la imagen pero no de la máscara, el valor del píxel de salida dependerá de píxeles el doble de alejados que con el tamaño normal. Por tanto, el parámetro '*sigma*' con el que se realiza el filtrado debe independizarse del tamaño de la imagen también. Esto se realiza de la siguiente manera para todos los filtrados paso bajo del código:

```
- img.blur(2 / factor_size);
```

5.8.2.4 Valor 'área_elipticidad'.

Este valor se calcula tal y como dice (13). Pero este valor depende tanto del área del objeto, como del área de su Bounding Box, valores que varían dependiendo del tamaño de la imagen.

Por ejemplo si tenemos una imagen original y una imagen re-escalada a la mitad en las dos dimensiones, el valor del área del objeto y del área del Bounding Box disminuirá a la cuarta parte. Para solucionarlo, en el programa se insertan límites de valor de '*área_elipticidad*' fijos, de manera que en el caso de que la imagen sea la mitad en ambas dimensiones, se continúa teniendo el mismo valor de '*área_elipticidad*' que antes. Para ello se multiplica por el factor de escalado. Por tanto la ecuación final utilizada es la siguiente:

$$Área\_elipticidad = (7 \cdot BB\_área + área) \cdot factor\_size + 150 \cdot elipticidad \quad (24)$$

También debe observarse que el valor de la elipticidad no cambiará ya que los autovalores mayor y menor cambiarán de la misma manera al cambiar el tamaño de la imagen, por tanto no afectarán al cálculo de la elipticidad. Este valor se calcula de la siguiente manera, siendo 'λ_M' el autovalor mayor y 'λ_m' el autovalor menor:

$$Elipticidad = \sqrt{\lambda_M / \lambda_m} \quad (25)$$

Capítulo 6. Resultados.

A continuación se muestran los resultados de la ejecución de los algoritmos de seguimiento de una única línea y del seguimiento de carril. Estos resultados se muestran tanto en una sucesión de imágenes en este documento, cómo en varias pruebas colgadas en la plataforma *Youtube*, en los que se tiene una doble pantalla: grabación en 3^{era} persona y grabación de lo que ve la cámara del GoPiGo. Este modo de visualización ayudará a la observar perfectamente el funcionamiento de estos algoritmos.

Pero antes de ello se mostrarán los resultados teóricos de la ejecución del modelo cinemático del uniciclo en un simulador realizado en *Matlab* con el objetivo de poder comparar resultados teórico-prácticos.

6.1 Simulador del modelo cinemático planteado.

Para poder saber si los resultados de la ejecución de cualquiera de los dos algoritmos son correctos, se ha realizado un simulador simple de la cinemática del uniciclo. Este, dado un estado inicial *[d, θ]*, y a partir la ecuación cinemática que modela el movimiento del uniciclo (6), realiza un bucle que va modificando y actualizando su estado.

Tal y como se explicó anteriormente, la velocidad angular 'w' se calcula realizando un control *PID* (proporcional-integral-derivativo) del error 'e', calculado como la diferencia entre el ángulo deseado 'θ_D' y el ángulo de orientación del uniciclo 'θ' respecto a la línea. Este simulador se ha realizado con el objetivo de observar que papel tienen estas constantes *PID* en el cálculo de 'w', además del valor de la distancia de observación 'L' necesario para el cálculo del ángulo deseado 'θ_D'.

Este simulador también se ha computado para poder realizar una elección de antemano de estas constantes además de para ver qué tipo de movimiento realizará el uniciclo en el seguimiento de la trayectoria. Además, este tiene la opción de añadir un sesgo y una desviación típica al estado, para que la medida no sea completamente ideal.

A continuación se muestran varias gráficas del movimiento obtenido, para un estado inicial *[d, θ]* = [10 cm, 20°]. En este estado inicial, el uniciclo se encuentra a la derecha de la línea y con una orientación contraria a la línea.

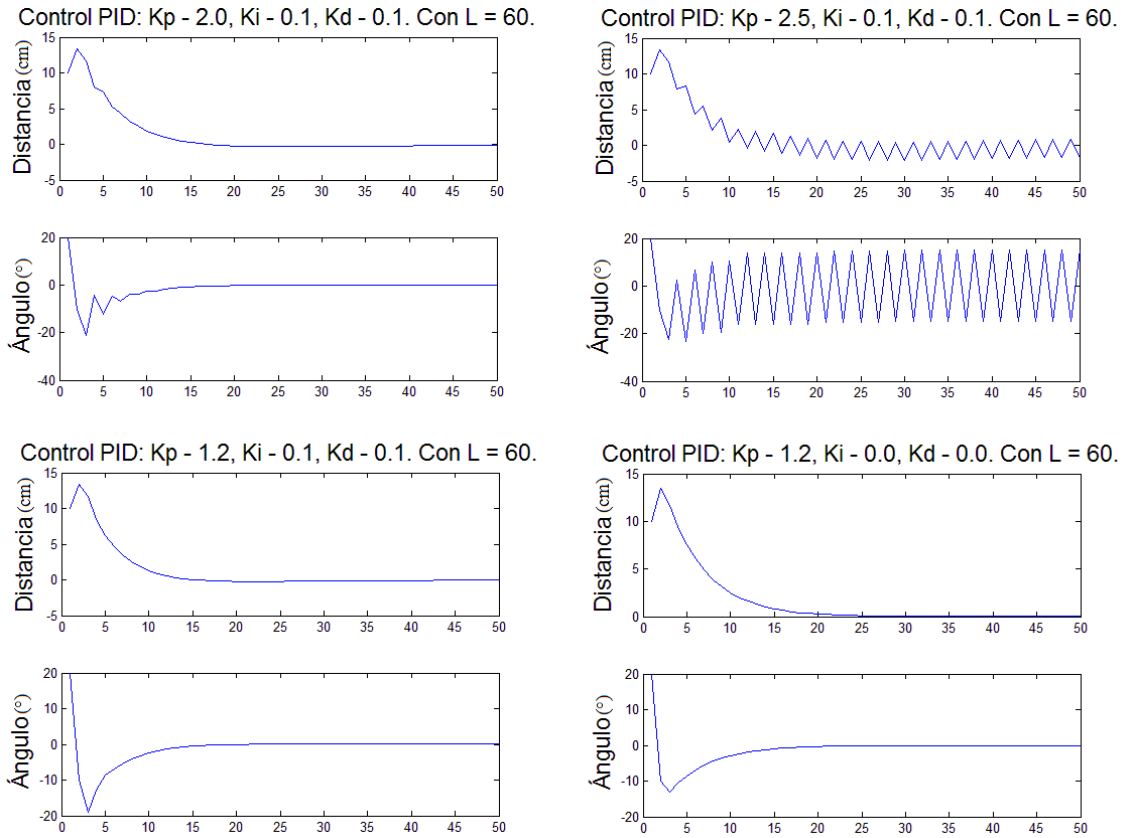


Fig. 58. (a) (b) (c) (d). Variación de parámetros K_P principalmente, para K_I y K_D y L fijas.

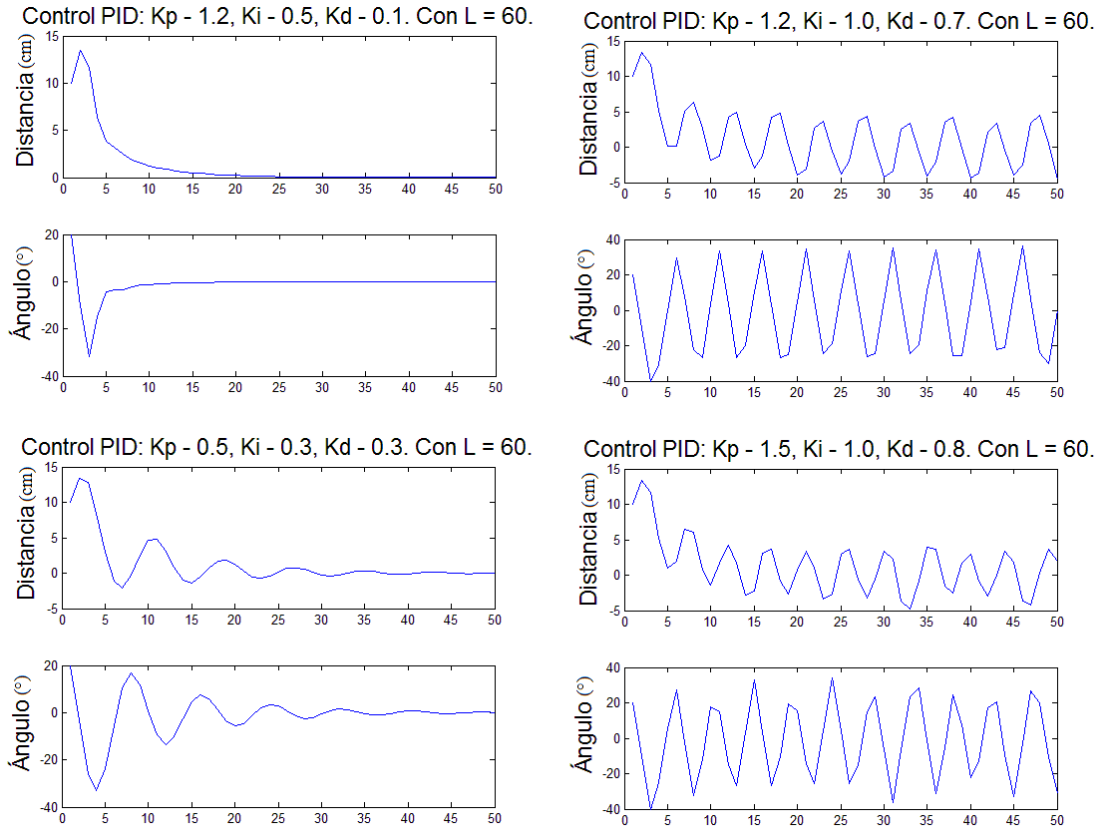


Fig. 59. (a) (b) (c) (d). Variación de parámetros K_P, K_I y K_D para L fija de 60cm.

En la Fig. 58 (b). puede observarse que escoger una constante de proporcionalidad 'K_P' demasiado elevada (>2.5) induce a una inestabilidad total del uniciclo. Comparando la evolución de la orientación en las Fig. 58. (a). y (d). (K_P = 2 y K_P = 1.2) puede observarse que en esta primera existen maniobras demasiado bruscas, mientras que en esta última la transición es bastante suave. Por otro lado comparando las Fig. 58. (c). y (d). para ver el peso que tienen las constantes integral y derivativa, puede observarse que incluirlas no sólo no ofrece mejora alguna, sino que consiguen empeorar la evolución de la orientación ya que en el primero llega a alcanzar -20°, mientras que el segundo llega a valer sólo -10°. En el caso real no podemos permitirnos una variación tan grande de la orientación ya que de esta manera la línea puede llegar a caer fuera de la imagen y en este caso no podríamos actualizar el estado. Por tanto, el mejor comportamiento es utilizando únicamente un control proporcional, es decir: '*w = P(error)*'.

Esto puede observarse en las Fig. 59. donde se puede ver que variando ligeramente las constantes integral y derivativa hace que el uniciclo se descontrole demasiado. En la realidad, en el caso que se origine un estado *[d, θ]* con ambos valores positivos o ambos valores negativos al mismo tiempo, el uniciclo se perderá más fácilmente ya que se situará a un lado de la línea y orientado hacia el lado donde no se encuentra la línea, donde no alcanza la visión de la cámara.

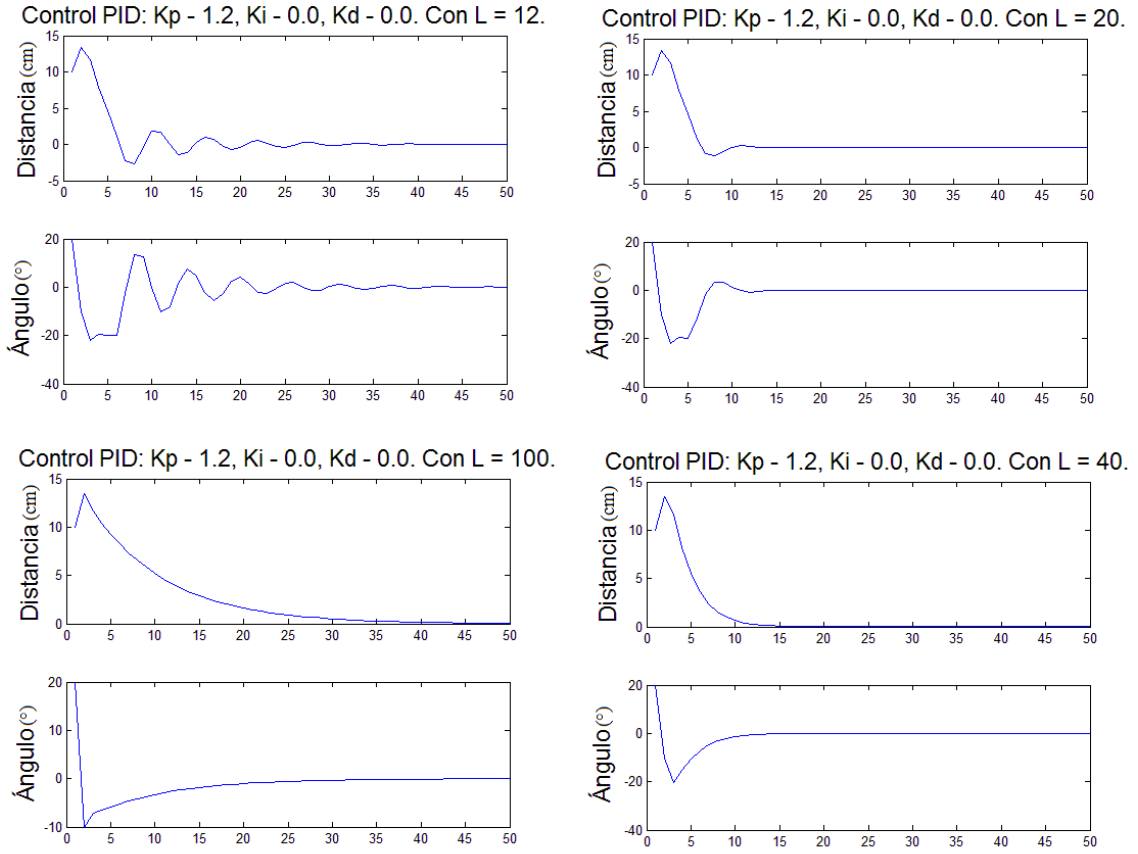


Fig. 60. (a) (b) (c) (d). Variación de parámetro *L*, con control proporcional de K_P = 1,2.

En la Fig. 60. puede observarse como varían los resultados cuando se varía la distancia de observación '*L*'. Cuando esta es muy pequeña, al estar observando demasiado cerca, el cálculo del ángulo deseado puede llegar incluso a ser menor que la propia orientación, haciendo que se descontrole y nunca llegue a converger. Cuando tiene un valor lo suficientemente grande como para conseguir converger, lo hace con varias oscilaciones alrededor de la línea (*L = 12*). Por el contrario, cuando esta es muy grande (*L = 100*), al observar demasiado lejos, la transición se realiza muy suavemente, tardando demasiado en converger.

En este punto ya se puede ver en qué rangos de valores el uniciclo puede llegar a funcionar. Se realizará un control puramente proporcional del error ($K_I = 0$, $K_D = 0$), con el valor $K_P = [0.9 \sim 1.3]$ y la distancia de observación $L = [30 \sim 60]$. En concreto se tomará: $K_P = 1.1$ y $L = 40$.

En este caso un control puramente proporcional es posible que llegue a funcionar, pero en el caso de vehículos mucho más pesados, debería tenerse en cuenta que un control *PID* puede llegar a ser más efectivo, además de tener en cuenta la dinámica del uniciclo que en este caso se está despreciando.

Por último se mostrará el comportamiento del uniciclo en el caso de añadir sesgo y desviación típica en dos casos diferentes: cuando los valores del estado inicial $[d, \theta]$ son ambos positivos/negativos y cuando tienen el signo cambiado uno respecto al otro.

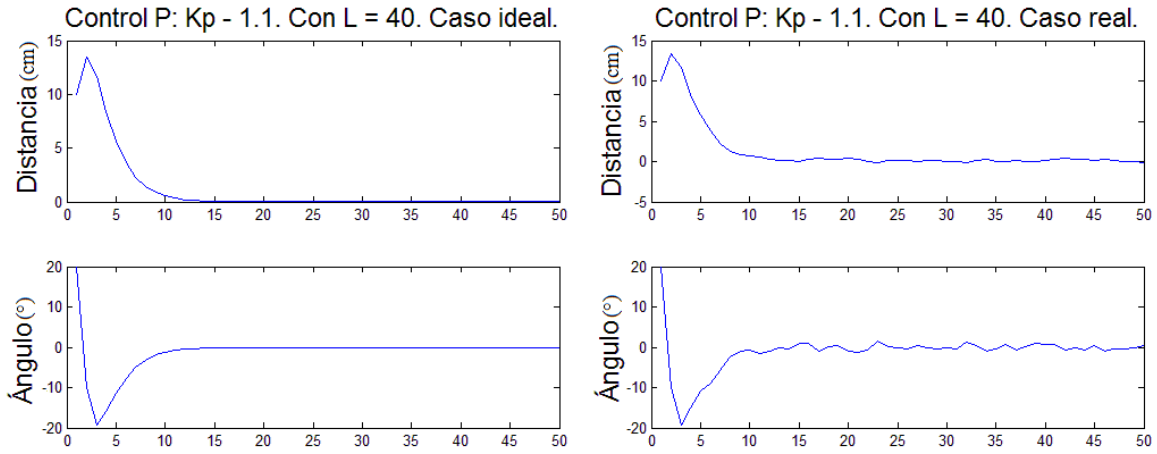


Fig. 61. (a) (b) (c) (d). Control proporcional: K_P = 1,1. L = 40. Estado inicial: $[d, \theta]$ = [10 cm, 20°]. En el caso ideal y en caso real (con sesgo y desviación típica).

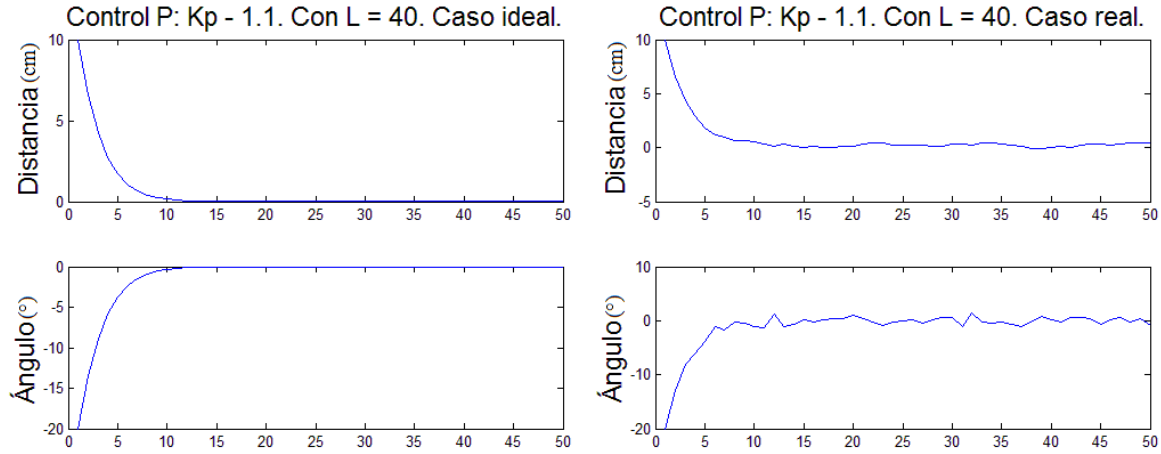


Fig. 62. (a) (b) (c) (d). Control proporcional: K_P = 1,1. L = 40. Estado inicial: $[d, \theta]$ = [10 cm, -20°]. En el caso ideal y en caso real (con sesgo y desviación típica).

En el caso de que en el estado inicial ambos tengan el mismo signo (Fig. 61.), significará que el uniciclo se encuentra a un lado de la línea, orientado hacia el lado donde no está la línea. De manera que en este caso realiza un movimiento estroboscópico de estabilización sobre la línea. En este el uniciclo llega a aumentar la distancia a la que se sitúa respecto a la línea en el instante inicial hasta que consigue cambiar el signo de su orientación y en ese momento se encuentra orientado hacia la línea. El movimiento a partir de aquí será el mismo que si directamente en el

estado inicial los parámetros tienen el signo cambiado (Fig. 62.), en el que directamente convergerá sobre la línea sin cambiar su orientación.

También puede observarse al comparar los casos ideales con los reales, que en estos primeros el uniciclo no llega a pasar al otro lado de la línea, manteniéndose en un lado de la línea a lo largo de todo el movimiento aunque este se prolongue hasta el infinito. En cambio, en los casos reales pueden verse las pequeñas oscilaciones que realiza alrededor de la línea debido a que en este caso se añade una desviación típica que simula el retardo que habrá entre la captación del fotograma (captación del estado del uniciclo) y el momento en el que se actúa sobre la velocidad de las ruedas (v_L, v_R).

6.2 Seguimiento de línea.

6.2.1 En líneas rectas.

Como se explicó anteriormente, el algoritmo desarrollado esta hecho principalmente para ser aplicado a una línea recta ya que se parametriza como tal (centro de masas y orientación del objeto), de manera que la comparación con el comportamiento teórico explicado en el anterior apartado (*6.1. Simulador del modelo cinemático planteado*), debe realizarse utilizando el seguimiento de una línea recta.

Tal y como puede verse en las Fig. 61. y Fig. 62., y como se explicó, existen dos casos principales, cuando ambos parámetros [d, θ] del estado tienen el mismo signo (Fig. 61.) y cuando tienen el signo cambiado (Fig. 62.). Además estas dos simulaciones se realizaron utilizando un controlador proporcional del error con: K_P = 1,1 y L = 40, de manera que el experimento práctico se insertaran estos mismos valores.

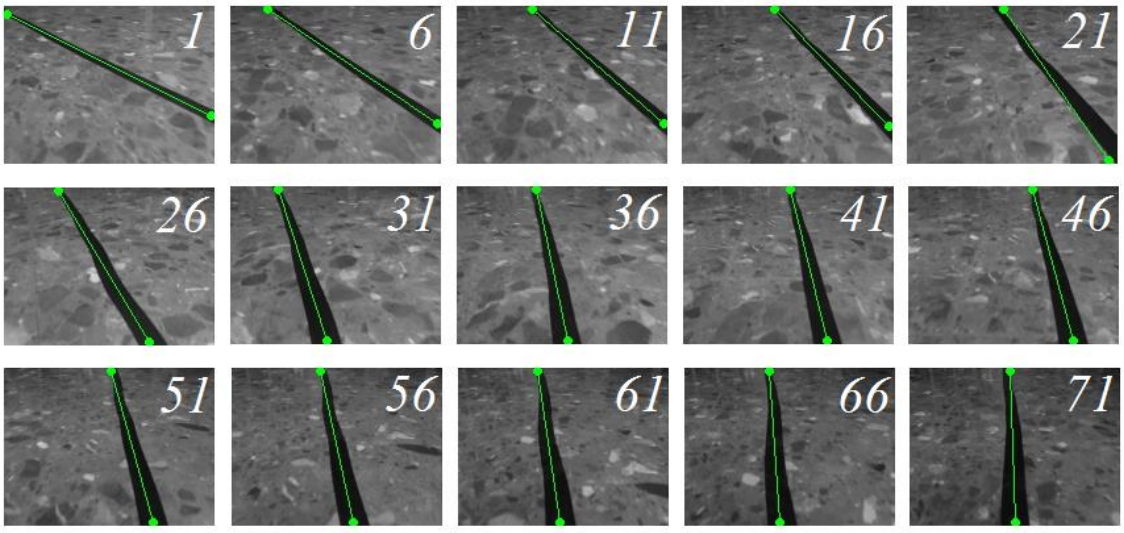


Fig. 63. Secuencia de 15 imágenes capturadas (1 de cada 5) pertenecientes al movimiento mostrado en la Fig. 64. Imágenes mostradas para cada fotograma: luminancia con línea pintada en verde (detecta la línea).

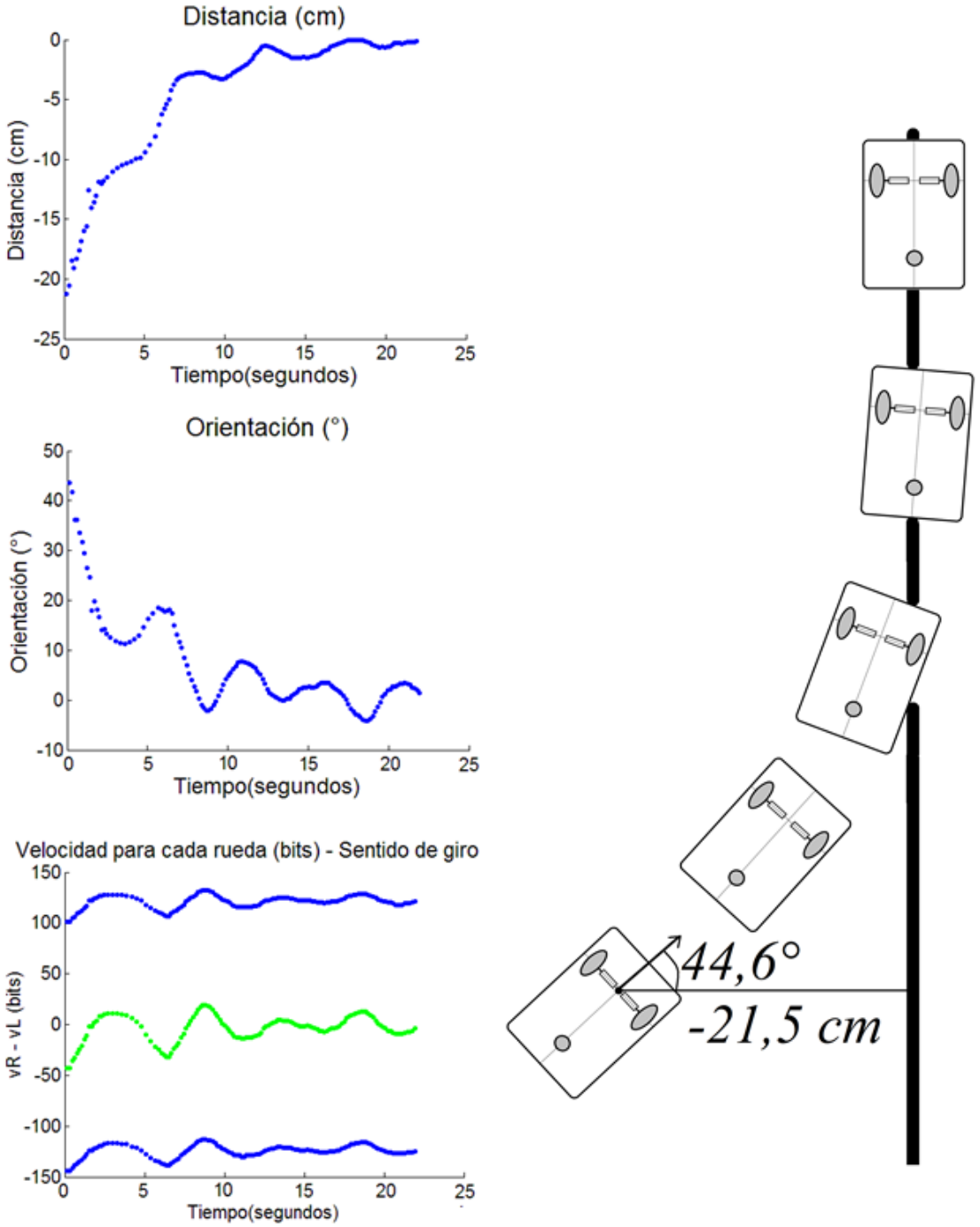


Fig. 64. Resultado de la ejecución: desarrollo del estado [*d (cm), θ (°)*] y de aplicación de velocidades 'v_L' (+) y 'v_R' (-) en el tiempo (ms). Estado inicial: [*-21,5 cm, 44,6°*]. K_P = 1,1 y L = 40. Velocidad lineal: v = 7 *cm/s*.

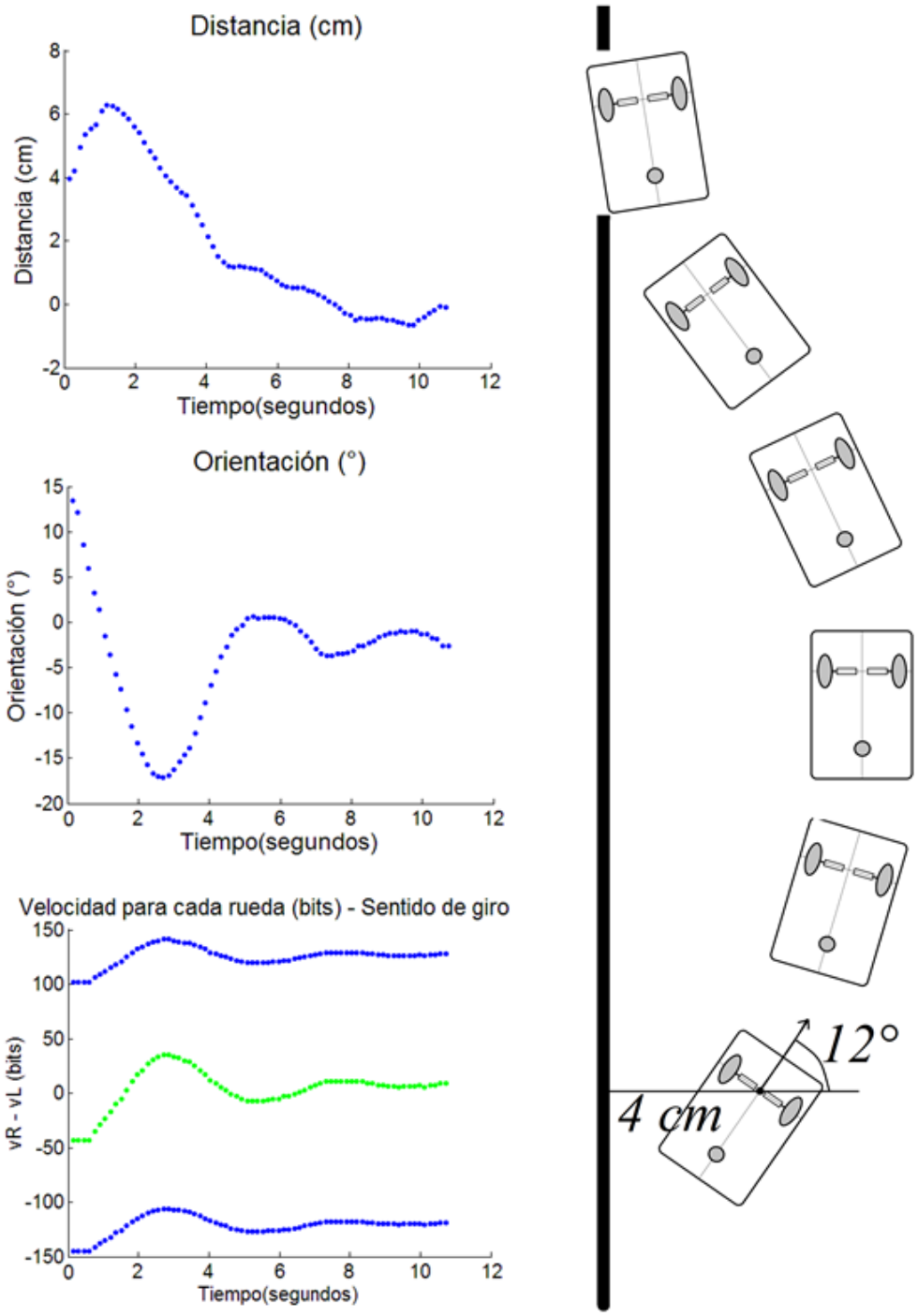


Fig. 65. Resultado de la ejecución: desarrollo del estado [*d (cm), θ (°)*] y de aplicación de velocidades 'v_L' (+) y 'v_R' (-) en el tiempo (ms). Estado inicial: [*4 cm, 12°*]. K_P = 1,1 y L = 40. Velocidad lineal: v = *7 cm/s*.

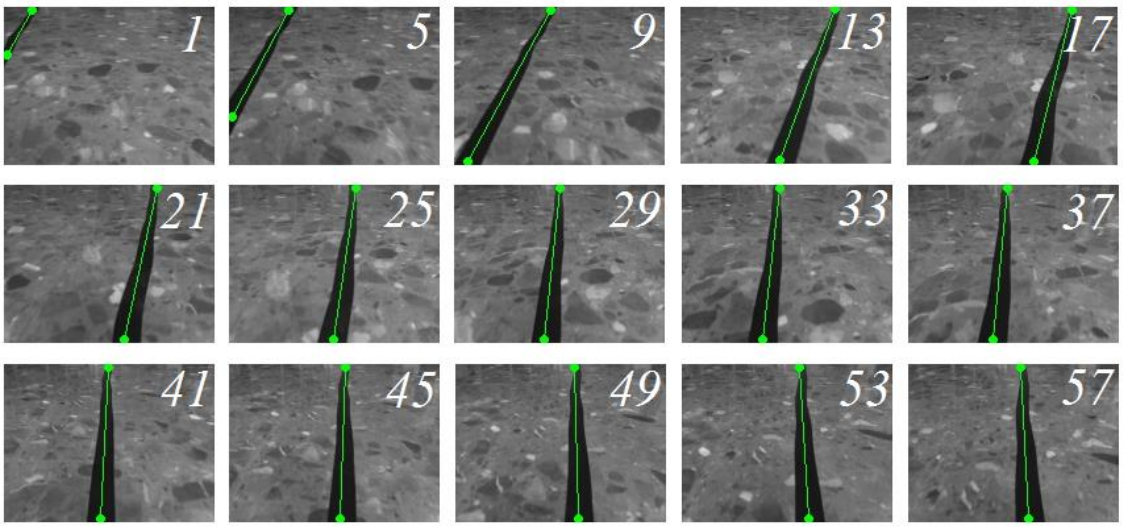


Fig. 66. Secuencia de 15 imágenes capturadas (1 de cada 4) pertenecientes al movimiento mostrado en la Fig. 65. Imágenes mostradas para cada fotograma: luminancia con línea pintada en verde (detecta la línea).

Como podemos ver en la Fig. 64., el desarrollo del estado es tal y como se predijo utilizando el simulador (Fig. 62.). Se trata del caso cuando los dos parámetros del estado tienen signo cambiado en el instante inicial, en este caso el uniciclo como se comentó antes, se sitúa a un lado de la línea y orientado hacia ella, de manera que su movimiento acaba convergiendo idealmente al estado [0 cm, 0°]. Observando que en diez segundos aproximadamente, a una velocidad de 7 cm/s y empezando a una distancia de -21,5 cm, ya ha convergido al estado ideal.

En la Fig. 65. puede verse también que realiza el mismo movimiento predicho por el simulador, este movimiento estroboscópico hace que comience a desplazarse alejándose de la línea al mismo momento que comienza a cambiar su orientación para apuntar hacia la línea. Una vez consigue cambiar su orientación de signo alcanza la máxima distancia de separación de la línea, realizando a partir de aquí el mismo movimiento que el de la Fig. 64. Debe tenerse en cuenta que en este caso no se ha podido separar más el uniciclo ya que como puede verse en el primer fotograma de la Fig. 66. este se encuentra al límite de no ver la línea y por tanto no poder conocer su estado inicial, aun así se han obtenido unos resultados vistosos.

Por otro lado, y en cuanto al tratamiento de imagen, puede verse en las Fig. 63. y Fig. 66. que la línea recta se ha detectado siempre perfectamente a lo largo de la secuencia de imágenes.

- Vídeos en *Youtube* de estos dos tipos de movimientos:

Movimiento 1 (convergencia, Fig. 64. Estado inicial: [d_{ini} *(cm),* θ_{ini} *(°)*] = *[-19 cm, 47'5°]*): https://youtu.be/EREqnito-5w

Movimiento 2 (estroboscópico, Fig. 65. Estado inicial: [d_{ini} *(cm),* θ_{ini} *(°)*] = *[-3 cm, -14°]*): https://youtu.be/5vcXOnCtd0o

6.2.2 En curvas.

En la sucesión de imágenes Fig. 67. y Fig. 68. puede verse el seguimiento de una curva negra y dos curvas blancas respectivamente. Puede verse como estas curvas se detectan perfectamente al igual que las líneas rectas anteriores. El problema principal de las curvas por tanto no es el tratamiento de imagen, sino la parte de robótica del seguimiento y es debido principalmente a:

- Las curvas no son parametrizadas como tal, sino como líneas rectas, por tanto se aproxima a ellas como de una recta se tratara.
- Al realizar el cálculo del centro de masas y la orientación del objeto en la imagen captada directamente por la cámara, no se tiene en cuenta de la misma manera la parte baja de la imagen, a la que se le da más peso, que a la parte alta de la imagen. Para solucionar este problema simplemente debería transformarse la imagen binarizada utilizando la matriz de homografía calculada y realizar el cálculo de momentos y el filtro en esta. De esta manera, al verse la curva sin ningún tipo de perspectiva, el centro de masas y la orientación serían estrictamente correctos. Este proceso no se realiza debido al coste computacional que supone, pero sería una mejora a realizar si se tuviera un procesador más potente.

En las curvas no tiene sentido graficar los resultados del desarrollo del estado [d, θ], ya que como se dijo, las curvas no son parametrizadas como tal, sino como rectas. De manera que simplemente se mostrará varias sucesiones de imágenes, además de varios videos subidos a *Youtube*, captando curvas y demostrando que aunque el uniciclo no las siga perfectamente como en el caso de las líneas rectas, puede detectarlas y seguirlas de manera aproximada sin perderse en ningún caso.

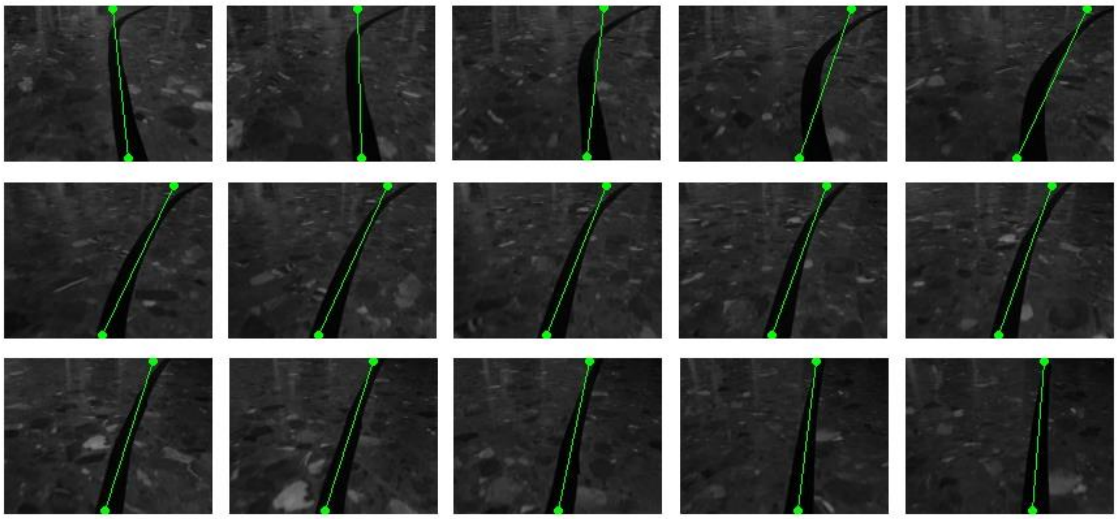


Fig. 67. Secuencia de 15 imágenes capturadas (1 de cada 5) del seguimiento de una curva de color negro.

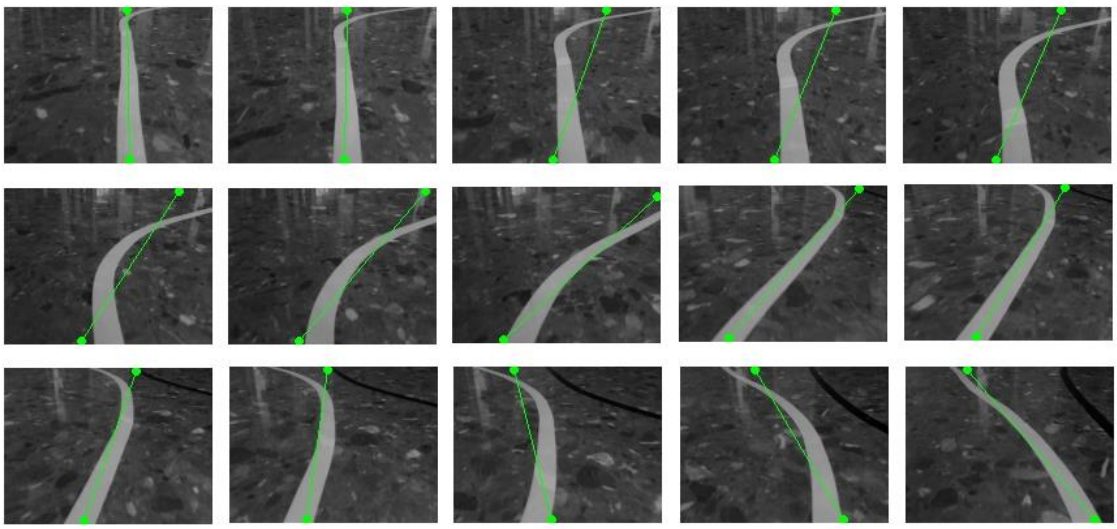


Fig. 68. Secuencia de 15 imágenes capturadas (1 de cada 5) del seguimiento de dos curvas de color blanco.

- Vídeos colgados en *Youtube* del seguimiento de una línea:

Vídeo 1, recorrido 1 (v = 11 cm/s, tamaño imagen 134x100): https://youtu.be/H53av_bQZNI

Vídeo 2, recorrido 1 (v = 12,5 cm/s, tamaño imagen 123x92): https://youtu.be/s8sIV-tnoaQ

Vídeo 3, recorrido 2 (v = 11 cm/s, tamaño imagen 134x100): https://youtu.be/i9K8U7Hr4eo

Vídeo 4, recorrido 2 (v = 13 cm/s, tamaño imagen 123x92): https://youtu.be/pmL4iJXsUFs

En estos vídeos puede observarse la facilidad que tiene el uniciclo de perderse en el seguimiento de una única línea (*6.3. Seguimiento de carril.* Fig. 71.) cuando se aumenta la velocidad del uniciclo (Vídeo 4, en el que llega a perderse dos veces). De manera que para que no se pierda, deberá bajarse la velocidad lineal, aunque esto hará que el uniciclo tome las curvas por dentro de la línea, no perdiéndose de vista en la imagen en ningún momento (Video 3, donde no se pierde en ningún momento). Todo esto se soluciona y es explicado de manera más detallada en el seguimiento de carril (siguiente apartado).

6.3 Seguimiento de carril.

A continuación se expondrán los resultados en el seguimiento de un carril continuo de color tanto negro como blanco sobre una superficie plana. En este caso, al igual que en el caso anterior, no tiene sentido graficarse los resultado, ya que la convergencia al camino seguido será de la misma manera que en el caso anterior pero en este caso sobre el centro del carril. El seguimiento se realizará tanto sobre líneas oscuras como claras y en diferentes suelos, además de realizar el seguimiento en curvas.

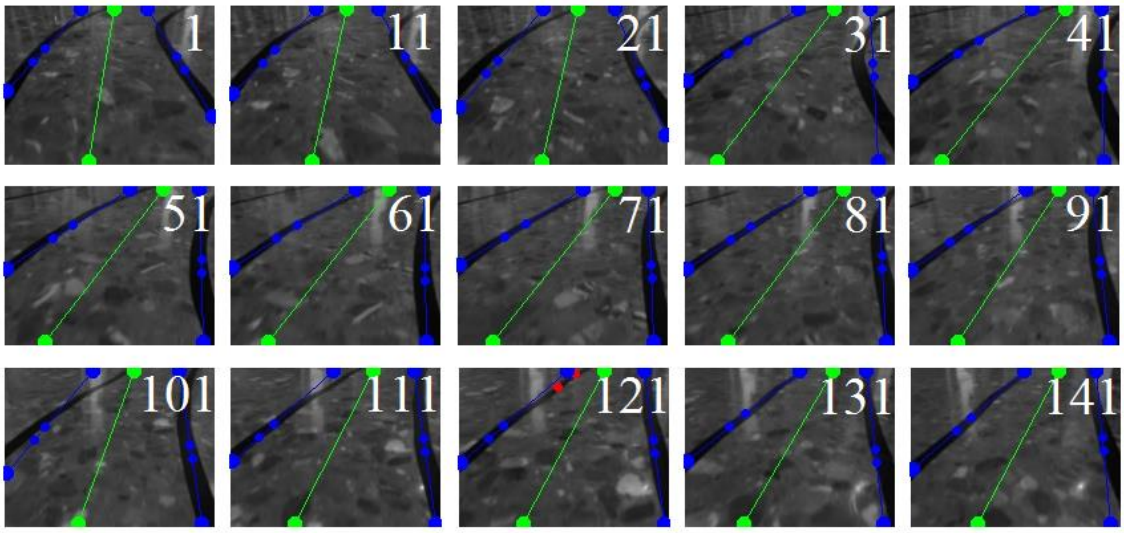


Fig. 69. Seguimiento de carril de líneas oscuras en una curva, mostrando 1 de cada 10 fotogramas.

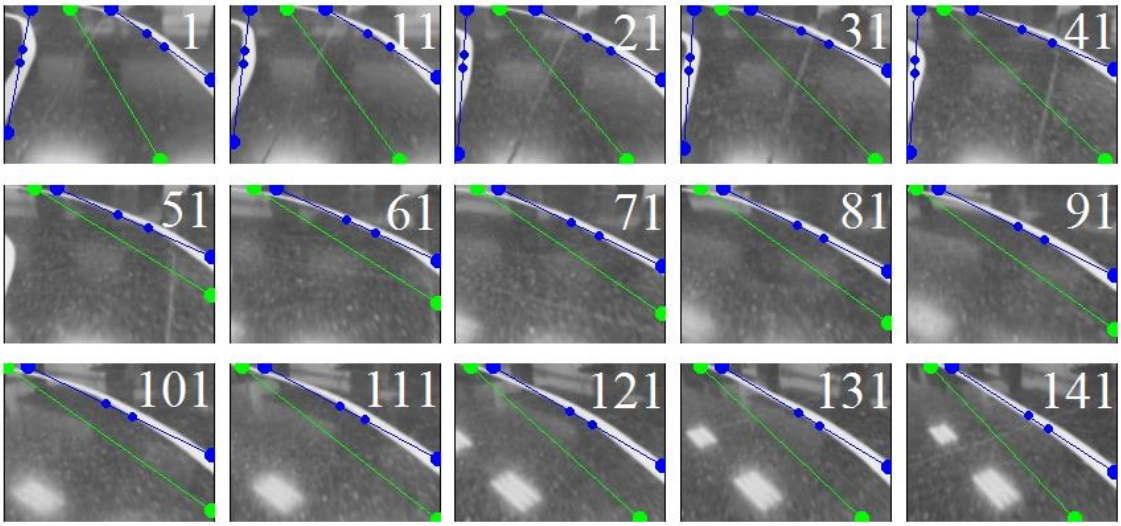


Fig. 70. Seguimiento de carril de líneas blancas en una curva, mostrando 1 de cada 10 fotogramas.

Al contrario que en el seguimiento de una única línea en este caso es mucho más complicado perderse del carril a seguir ya que el uniciclo al seguir el camino a un lado de la línea no llega a perder de vista en ningún momento la línea (Fig. 71.). Esto proporciona mejores resultados en cuanto a que el uniciclo no realiza cambios bruscos de velocidad en los motores, siguiendo la línea de manera más fluida y uniforme. Estos cambios bruscos de velocidad y la comparativa entre el seguimiento de una única línea y dos líneas puede verse en los vídeos de seguimiento colgados en la plataforma *Youtube*.

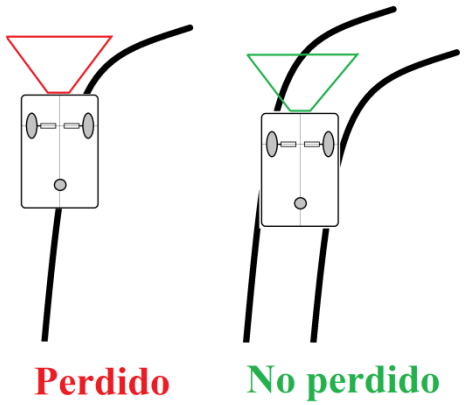


Fig. 71. Comparación de pérdida en el seguimiento de una única y en el seguimiento de carril.

- Vídeos colgados en *Youtube* del seguimiento de carril:

Vídeo 1, recorrido 1 (v = 13,5 cm/s, tamaño imagen 134x100): https://youtu.be/09GnPlJb13w

Video 2, recorrido 1 (v = 15 cm/s, tamaño imagen 123x92): https://youtu.be/AUxUKrjCONQ

Video 3, recorrido 2(v = 16 cm/s, tamaño imagen 123x92): https://youtu.be/9OiqVY_MzPY

Video 4, recorrido 3 (v = 13,5 cm/s, tamaño imagen 170x120): https://youtu.be/8r9K7y7m1QE

Capítulo 7. Conclusiones y propuesta de trabajo futuro.

7.1 Conclusiones.

Los objetivos marcados al comienzo del proyecto eran los siguientes:

- Obtención y uso de un modelo de control para el uniciclo para el seguimiento de una trayectoria dada.

- Conseguir un algoritmo de tratamiento de imagen para la detección de una línea continua sobre una superficie plana.

- Conseguir un algoritmo de tratamiento de imagen para la detección de un carril continuo (dos líneas paralelas) sobre una superficie plana.

El algoritmo de tratamiento de imagen hasta la binarización, el residuo de un doble filtrado morfológico adaptativo a la altura de la imagen, ha conseguido resaltar la línea sobre el fondo de la imagen con un coste computacional muy bajo, de manera que en este sentido el algoritmo puede ser utilizado y añadido en el futuro a algoritmos más robustos como la detección de líneas utilizando el filtro de Canny y la Transformada de Hough, obteniendo resultados independientes de la iluminación en la imagen ya que el filtrado morfológico se adapta a cualquier iluminación.

El problema del tratamiento de imagen es tras la binarización, ya que la aparición de reflejos demasiado grandes en el suelo con una forma demasiado similar a por ejemplo, líneas discontinuas, pueden llegar a no ser filtradas, por esta razón se necesitaría un filtro más complejo que determine si se trata o no de una línea además de por su forma geométrica. Este problema aparece principalmente cuando se realiza el seguimiento de una línea oscura sobre fondo claro ya que estos al ser de color claro pueden llegar a cortar la línea por la mitad, en cambio con una línea clara sobre fondo oscuro, los reflejos simplemente se adhieren a la línea como parte de esta, sin modificar prácticamente el resultado (Fig. 72.).

Fig. 72. Imagen con reflejos adheridos a la línea blanca. (a) Imagen captada. (b) Imagen binarizada.

Por otro lado, en cuanto al modelo cinemático planteado para el seguimiento de trayectoria en el uniciclo se ha utilizado uno de los métodos más simples al mismo tiempo que efectivo planteados físicamente en diferentes artículos [13] [14] [15]. En este sentido no se ha intentado innovar, lo único que se quería era utilizar un método simple de seguimiento, con el objetivo de llevar a la práctica el tratamiento de imagen realizado. A pesar de ello, el modelo cinemático planteado, ha sido aplicado con éxito sobre el GoPiGo (Dexter Industries), habiendo sido capaz de utilizar correctamente las librerías que el fabricante proporcionaba.

7.2 Trabajo futuro.

Existen varias partes que podrían añadirse al algoritmo, principalmente en el tratamiento de imagen, con el objetivo de poder ser aplicado a casos más reales en carreteras convencionales o autopistas.

- El primer paso sería utilizar un tratamiento más robusto junto al tratamiento de imagen planteado, que sirviera como filtro discriminatorio de las líneas que realmente no son líneas a seguir sino objetos situados en el entorno de manera aleatoria. Para ello podría utilizarse, como se comentó y como plantean varios artículos [18] [20], el uso del filtro de Canny y la Transformada de Hough para la parametrización de rectas, en lugar del cálculo de momentos.

- El segundo aspecto mejorable sobre la detección de líneas continuas es poder parametrizar la curvatura de las curvas, de manera que pudiera saberse exactamente cuál es la velocidad angular necesaria en cada instante para que la curva sea tomada correctamente de manera exacta, ya que parametrizar curvas como rectas no es aplicable en un caso real.

- El siguiente paso en cuanto al tratamiento de imagen sería ser capaz de seguir un carril formado, por ejemplo y típicamente, por una línea continua a la derecha y una línea discontinua a la izquierda. Siendo capaz de emparejar las líneas discontinuas por un lado y de parametrizarlas como si se trataran de una línea continua. Una manera de realizarlo viene propuesta en [20], utilizando una doble Transformada de Hough en cascada para la detección de líneas discontinuas.

Por otro lado y como se comentó anteriormente, las limitaciones del mecanismo de control son bastante limitadas y sólo pueden utilizarse para uniciclos, de manera que si se desea aplicar a otro tipo de robot debería volverse a estudiar y a obtener un mecanismo de control diferente específico para dicho robot. Además, en un caso real no podría despreciarse el modelo dinámico del robot tal y como se ha hecho en este trabajo, ya que en este caso esta parte si tendría un peso a tener en cuenta en el resultado final debido a fuerzas de rozamiento o de reacción al peso con el suelo mucho más significantes.

En cuanto al mecanismo de control del uniciclo podría mejorarse una vez se ha conseguido parametrizar las curvas utilizando la velocidad lineal como variable de control, además de la velocidad angular. De esta manera podría controlarse mejor el uniciclo en las curvas, adaptando la velocidad lineal a estas [15].

Así quedan definidos varios aspectos de mejora del algoritmo desarrollado con tres enfoques distintos, que dependerán de los objetivos del trabajo futuro:

- Mejora del tratamiento de imagen para la detección de líneas y/o carril.
- Obtención de un modelo cinemático y dinámico para un robot diferente al uniciclo.
- Mejora del modelo de control del robot móvil uniciclo.

7.3 Valoración personal.

El algoritmo se ha realizado con el objetivo principal de aplicar un tratamiento de imagen al mundo real, a través de un robot del tipo uniciclo. Por tanto, la parte más importante y destacada del trabajo es esta, el tratamiento de imagen previo a la binarización. Ya que una vez llegados a este punto, si se desea aplicar a un caso más real, sería necesario contar con un procesador más potente para poder realizar una selección de líneas más compleja en la que cualquier objeto con características básicas parecidas a las de una línea no sea capaz de burlar el filtrado discriminatorio simple realizado en este algoritmo. Aunque a pesar de ello, con el filtrado de bajo coste computacional realizado, se es capaz de realizar un seguimiento exitoso pero no tan robusto como debería ser para ser aplicado a un caso real.

Por otro lado, el retardo desprendido del tratamiento de imagen de una imagen de tamaño 320x240 píxeles era aproximadamente de 200 milisegundos (5 fotogramas cada segundo), algo inviable en la práctica ya que este tiempo entre la captación y aplicación de la velocidad deseada no es aplicada en el instante que debería, pudiendo causar fallos y movimientos bruscos en este sentido. Para solucionarlo, en este caso, se ha tomado la imagen mitad en las dos dimensiones (160x120), consiguiendo que el retardo fuera 4 veces menor (50 milisegundos, 20 fotogramas por segundo). De esta manera y bajando la velocidad lineal del uniciclo (realizando así un muestreo con mayor frecuencia), si se ha podido ser capaz de aplicar adecuadamente el modelo de control del uniciclo y ver como realmente funciona el seguimiento de la línea en el mundo real, comportándose exactamente igual al predicho por el simulador teórico. Pero esto deja casi sin espacio de cálculo al uso de procesos más complejos comentados anteriormente, por ello se plantea la necesidad de un procesador (CPU) más potente que el que posee el GoPiGo.

A pesar de todo ello, se ha conseguido un algoritmo exitoso y útil que unifica diferentes campos: el tratamiento de imagen, la geometría y la robótica (en este caso, los robots uniciclo).

Capítulo 8. Referencias bibliográficas.

[1] IEEE SPECTRUM. ERICO GUIZZO (POSTED 18 OCT 2011). <http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works> [Consultado: 5 de febrero de 2017].

[2] UDACITY WEB PAGE. *Self-Driving Car Engineer*. <https://www.udacity.com/drive> [Consultado: 5 de febrero de 2017].

[3] GOPIGO. DEXTER INDUSTRIES. <https://www.dexterindustries.com/shop/gopigo-kit/> [Consulta: 3 de junio de 2017].

[4] CONTROL OF MOBILE ROBOTS, DR. MAGNUS EGERSTEDT. Coursera. <https://www.coursera.org/learn/mobile-robot>

[5] LINUX. UBUNTU. <https://www.ubuntu.com/>

[6] ECLIPSE. <https://eclipse.org/>

[7] ANDERSON, T. (BASED ON AN EARLIER VERSION BY WAYNE, C). (1995). *A Quick Introduction to C++*. Online: <http://homes.cs.washington.edu/~tom/c++example/c++.pdf>.

[8] LIBRERÍA Y DOCUMENTACIÓN GOPIGO. <https://github.com/DexterInd/GoPiGo/tree/master/Software/C>.

[9] LIBRARÍA Y DOCUMENTACIÓN OPENCV. <http://docs.opencv.org/>.

[10] LIBRARÍA Y DOCUMENTACIÓN CIMG. <http://cimg.eu/reference/>.

[11] DOCUMENTACIÓN MATLAB. <https://es.mathworks.com/help/matlab/>.

[12] LA VALLE S. M. (2006). *Planning Algorithms: 13.1.2 Kinematics for Wheled Systems*. Cambridge University Press.

[13] BAMBINO, I (2008). *Una introducción a los robots móviles*. Trabajo. pp. 31-33. <http://aadeca.org/pdf/CP_monografias/monografia_robot_movil.pdf> [Online].

[14] BORENSTEIN. J, EVERETT H. R, AND FENG L. (1996). *Where Am I?*. EEUU: Michigan.

[15] CARONA R, AGUIAR A. P. Y GASPAR J (2008). "Control of unicycle type robots: Tracking, Path Following and Point Stabilization" en Proc. de IV Jornadas de Ingeniería Electrónica, de Telecomunicaciones y de Computadores, Universidad Técnica de Lisboa, pp. 180-185.

[16] RUSS J.C. (2011). *The image processing handbook.* Raleigh, North Carolina. (6ª edición).

[17] GONZALEZ R. C. (2002). *Digital Image Processing*. Upper Saddle River, New Jersey. (2ª edición).

[18] MEDINA BALLESTER, A. LEAL MESEGUER, X. (2010). *Detección de Líneas y Sistema de Estabilidad de Carril basado en cámara frontal*. Universidad de las Islas Baleares, Visón por

computador.<http://ibdigital.uib.es/greenstone/collect/enginy/index/assoc/Enginy_2/010v02p0/33.dir/Enginy_2010v02p033.pdf > [Online].

[19] DOMIN. N. Github. Función 'polyfit.c' (Third Party). <https://github.com/natedomin/polyfit> [Online].

[20] ZHANWEI WU, BIN KONG AND FET ZHENG. "Detection and Extraction of Discontinuous Lines". International Conference on Robotics and Biomimetics December 17 - 20, 2006, Kunming, China.