Capítulo 4. IMPLEMENTACIÓN DEL MÉTODO RUNNING AVERAGE

4.1. Descripción general

En este proyecto, se usa la tecnología LIDAR, en concreto el sensor SICK LD-MRS. Éste proporciona información (gracias a sus propias librerías) sobre la actual nube de puntos, por ejemplo, posición cartesiana (x, y, z), distancia, canal, ángulo horizontal y vertical, etc.

El objetivo es determinar la posición de cada uno de los objetos detectados. Debido a que el LIDAR tiene cuatro canales, se trabajará con ellos de forma independiente, por lo que se determinará la posición del objeto en cada uno de los canales en los que esté presente.

Para ello, se implementará un algoritmo basado en la técnica de sustracción de background, en concreto, en el método de Running Average (detallada en el apartado 3.2).

La *Figura 10* muestra los pasos de este procedimiento, los cuales se detallan a continuación:

El LIDAR detecta, en cada instante de tiempo, una nube de puntos, la cual se almacena y se la denomina imagen de background. Es necesario guardar una copia de este primer background (lo cual se explicará en los siguientes apartados).

Por cada nube de puntos nueva, se compara (uno por uno) el valor de los pixeles actuales con el de background. Si ambos pixeles son parecidos, es decir, no verifican la desigualdad (13), el pixel actual se clasifica como background, actualizándose como en la ecuación (12). En caso contrario, se clasifica como foreground. Para diferenciar los pixeles de background de los de foreground, se guardará en un vector (llamado comparación) un cero si el pixel actual es un pixel de background y un uno si es un pixel de foreground.

Sin embargo, que un pixel sea clasificado como foreground no significa que sea un objeto. Por este motivo, es necesario determinar cuáles de esos pixeles (obtenidos en el paso anterior) son realmente pixeles objeto. Ése es el objetivo principal del vector comparación. Si éste tiene más de un número mínimo de unos seguidos (por ejemplo, 2) o hay menos de un máximo

número de ceros seguidos entre ellos (por ejemplo, 10), entonces esos pixeles pueden ser un posible objeto por lo que se calcula su tamaño. Este tamaño se calcula como la distancia entre el primer y último pixel del posible objeto.

Sólo se detectarán aquellos objetos cuyo tamaño sea mayor que un tamaño mínimo fijado (en este caso, un tamaño adecuado es 20 cm). En caso de serlo, se considerarán esos pixeles como posibles píxeles objeto, pero sólo se almacenarán aquellos que:

- Sean bastante diferentes de los pixeles del primer background almacenado.
- Sean similares entre ellos.

La localización del objeto se calcula como la posición media (media de x, y , z y distancia de los pixeles) de los pixeles objeto almacenados.

Para evitar detectar como objetos aquellos objetos aislados que aparecen y desaparecen rápidamente, es necesario comprobar si este objeto sigue estando en una posición parecida durante un número determinado de veces (por ejemplo, 3). Sólo en ese caso se devolverá la posición del objeto.

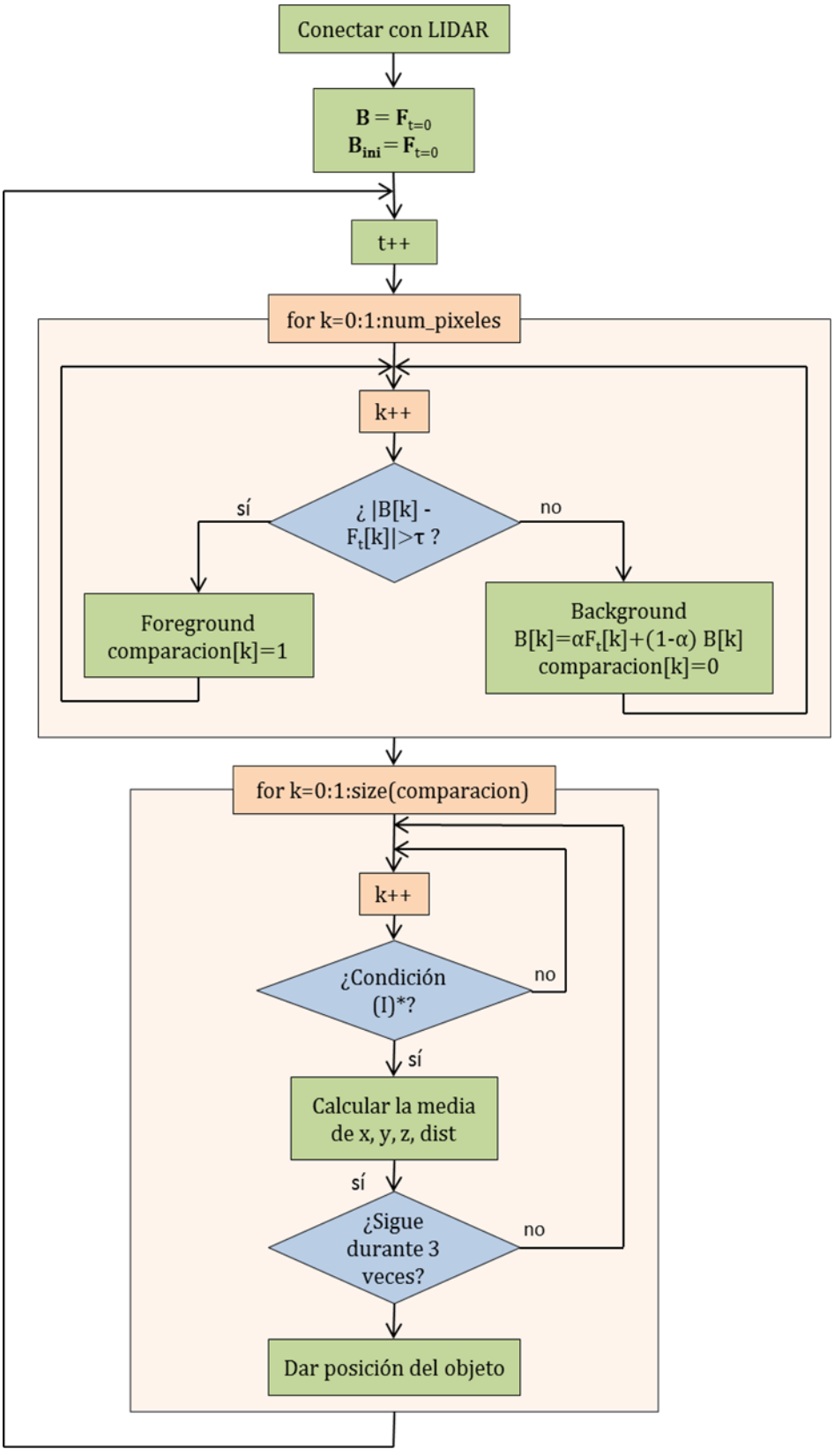

Figura 10: Diagrama del algoritmo

* Condición (I) = (10 ceros seguidos) AND (más de dos unos) AND (tamaño>20cm) AND (pixeles≠B_{ini}) AND (los pixeles son similares)

4.2. Descripción detallada

En la ejecución del algoritmo se han definido las siguientes constantes (pudiéndose cambiar sus valores antes de la ejecución del mismo y según el tipo de objetos que se quieran detectar):

Nombre	Valor	Descripción
TAM_MAX	2000	Número máximo de pixeles de las nubes de puntos
TH	0.4	Umbral utilizado en la ecuación (13)
ALFA	0.05	Radio de actualización utilizado en la ecuación (12)
TAM_MIN	0.2 m	Tamaño mínimo de los objetos a detectar
MAX_CEROS	10	Número máximo de ceros seguidos permitidos en el vector comparación para determinar el fin de los pixeles del objeto
MAX_OBJECTS	10	Número máximo de objetos detectados cada vez y por canal
DIF_PIXEL	0.4	Máxima diferencia entre pixeles de un mismo objeto
REP_OBJ	3	Número de veces seguidas que tiene que estar un objeto para considerarse como objeto y devolver su posición
MIN_PIX_OBJ	2	Número mínimo de pixeles de que está formado un objeto
DIF_OBJ	0.5	Máxima diferencia de distancia entre un mismo objeto en instantes consecutivos

Tabla 1: Constantes del algoritmo

El algoritmo está compuesto de la función principal ($main$) y de una clase denominada $background$ formada a su vez por varias funciones, las cuales se detallan en los siguientes apartados. Su programación en C++ se encuentra detallada el Capítulo 6.

4.2.1. Función principal (MAIN)

En la función principal es donde se produce la conexión con el LIDAR, mediante la dirección IP 192.168.0.1 y utilizando sus librerías específicas. En el caso de que la conexión sea satisfactoria y el escaneo de la nube de puntos actual haya sido detectado correctamente, se llama a la clase de background.

La primera vez se inicializa el background mediante la llamada al constructor $background$. Una vez establecida la primera nube de background y al detectar una nueva se llama a la función $detect$ que se encarga de decidir si hay o no objetos presentes e imprimir su posición en caso de haberlos.

Este algoritmo se repetirá de forma continuada (mediante un bucle while) hasta que se produzca un evento de salida, como son:

- El usuario escribe la combinación Ctrl+C
- Se supera el tiempo de espera

4.2.2. Background

Su formato es:

$$background\ (const\ Scan\ \&scan).$$

Esta función es el constructor que se ejecuta una única vez (la primera). El parámetro de entrada $scan$ contiene toda la información de la nube de puntos detectada por el LIDAR.

El objetivo de esta función es establecer la primera nube de puntos tomada por el LIDAR como background e inicializar y dimensionar variables globales de esta clase.

En esta función se dimensionan los siguientes vectores:

- $num\_objects$ con cuatro elementos a cero. Cada uno de ellos corresponde a un canal y almacena el número de objetos presentes en cada canal.

- $times\_obj\_c0$, $times\_obj\_c1$, $times\_obj\_c2$, $times\_obj\_c3$ son cuatro vectores, cada uno de ellos de dimensión el número máximo de objetos que se puede detectar (MAX_OBJECTS). Estos vectores indican, para cada canal, el número de veces seguidas que se ha detectado el mismo objeto.

- $dist\_obj\_ant\_c0$, $dist\_obj\_ant\_c1$, $dist\_obj\_ant\_c2$, $dist\_obj\_ant\_c3$ son cuatro vectores, cada uno de ellos de dimensión el número máximo de objetos que se puede detectar (MAX_OBJECTS). Almacenan, para cada canal, la distancia media de los objetos detectados en el instante anterior.

El número de puntos detectados por el LIDAR se almacena en una variable llamada $points$. Como este número no es fijo y para evitar problemas de acceso a memoria, se impone que el número de puntos no sea mayor que un máximo establecido (TAM_MAX). En caso de alcanzar este máximo, solo se trabajará con los primeros TAM_MAX píxeles de la nube de puntos detectada.

A continuación se inicializan los vectores correspondientes al background con el tamaño indicado por $points$, mediante la función set_bg y los correspondiente al objeto con tamaño vacío (ya que a priori se desconoce el número de pixeles que formarán los objetos) con la función set_ob.

Por último se recorren los vectores del background almacenando en ellos los puntos detectados por el LIDAR en el instante actual. Esto se consigue con las funciones add_bg y add_bg_ini.

4.2.3. Set_bg

Su formato es:

$$void\ set\_bg\ (const\ unsigned\ int\ n)$$

Esta función inicializa los vectores correspondientes al background con el tamaño indicado por el parámetro de entrada n.

Además de los vectores correspondientes al background (que se irán actualizando a lo largo del programa), se tendrán otros vectores que almacenen toda la información relacionada con la primera nube de puntos guardada como background, llamado background inicial.

En resumen, se inicializan los siguientes vectores:

BACKGROUND	BACKGROUND INICIAL
x_bg	x_bg_ini
y_bg	y_bg_ini
z_bg	z_bg_ini
dist_bg	dist_bg_ini
hAngle_bg_deg	hAngle_bg_deg_ini
channel_bg	channel_bg_ini

Tabla 2: Vectores inicializados en la función set_bg

4.2.4. Add_bg

Su formato es:

$$void\ add\_bg\ (const\ size\_t\ i, const\ float\ \_x, const\ float\ \_y, const\ float\ \_z, const\ float\ d, const\ float\ ha, const\ unsigned\ int\ c)$$

Esta función almacena la nube de puntos inicial en los vectores correspondientes al background ($x\_bg, y\_bg, z\_bg, dist\_bg, hAngle\_bg\_deg, channel\_bg$). Para ello, se le pasa como argumentos de entrada:

Parámetro	Descripción
i	Posición del elemento a almacenar
_x	Posición en el eje x a almacenar (metros)
_y	Posición en el eje y a almacenar (metros)
_z	Posición en el eje z a almacenar (metros)
d	Distancia respecto al LIDAR a almacenar (metros)
hA	Ángulo horizontal a almacenar (grados)
c	Canal a almacenar

Tabla 3: Parámetros de entrada de add_bg

Para la posición dada por i, está función asigna cada parámetro de entrada a su vector correspondiente de background. El ángulo horizontal (hA) se da en grados pero se almacena en radianes, por lo que se aplica la transformación:

$$ang(rad) = ang(°)\,\frac{\pi}{180°} \tag{27}$$

4.2.5. Add_bg_ini

Su formato es:

$$void\ add\_bg\_ini\ (const\ size\_t\ i, const\ float\ \_x, const\ float\ \_y, const\ float\ \_z, const\ float\ d, const\ float\ ha, const\ unsigned\ int\ c)$$

Esta función almacena la nube de puntos inicial en los vectores correspondientes al background ($x_bg_ini, y_bg_ini, z_bg_ini, dist_bg_ini, hAngle_bg_deg_ini, channel_bg_ini$) Para ello, se le pasa como argumentos de entrada los mismos que en la *Tabla 3*.

Para la posición dada por i, está función asigna cada parámetro de entrada a su vector correspondiente de background inicial. El ángulo horizontal (hA) se da en grados pero se almacena en radianes, por lo que se aplica la transformación (27).

4.2.6. Set_ob

Su formato es:

$$void\ set\_ob\ (\ )$$

es decir, no necesita parámetros de entrada.

Esta función inicializa como vectores vacíos a los vectores correspondientes al objeto, es decir:

OBJETO	OBJETO AUXILIAR
x_obj	x_obj_aux
y_obj	y_obj_aux
z_obj	z_obj_aux
dist_obj	dist_obj_aux
Tamaño_obj	hAngle_obj_deg_aux

Tabla 4: Vectores inicializados en la función set_ob

Los vectores objeto almacenan las posiciones medias de cada uno de los objetos detectados, mientras que los vectores auxiliares van almacenando la nube de puntos que compone cada uno de los objetos.

4.2.7. Detect

Su formato es:

$$void\ detect\ (const\ Scan\ \&scan)$$

El parámetro de entrada $scan$ contiene toda la información de la nube de puntos detectada por el LIDAR.

El primer paso consiste en comparar el background con la nube de puntos detectada en cada instante. Al tener estas nubes distinto número de puntos no se pueden comparar los elementos de la posición i en ambas nubes porque es posible que pertenezcan a pixeles con distinto canal y ángulo horizontal. La solución es tener dos índices ii e i, los cuales recorren la nube de puntos actual y el background respectivamente y otros dos índices, $ultii$ e $indexi$ que indiquen cual ha sido el último elemento procesado de la nube actual de puntos.

Inicialmente, estos índices están a cero, ya que el primer elemento a comparar está en la posición 0 de ambas nubes. A continuación, se comprueba si estos pixeles tienen el mismo canal y ángulo horizontal ya que sólo se pueden comparar en ese caso. Si es así, se inicializa una variable (booleana) llamada bg a false. Esta variable se resetea cada vez que se cambia de canal y ángulo horizontal e indica si los pixeles hay que tomarlos como background (en este caso, bg vale true) o como foreground (si bg vale false).

Esquemáticamente, lo que se hace es comparar los pixeles (que tienen mismo canal y ángulo horizontal) en el orden en que aparece en la *Figura 11*, es decir, el primer pixel del background con todos los del mismo canal y ángulo horizontal de la nube actual; si el siguiente pixel del background tiene el mismo canal y ángulo se compara también con los pixeles de la nube actual y así hasta que se encuentre un pixel con distinto canal y ángulo horizontal que el que se estaba procesando.

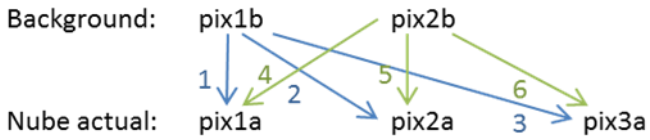

Figura 11: Comparación de pixeles con mismo canal y ángulo horizontal

Para afirmar que los pixeles con un determinado canal y ángulo horizontal corresponden a un objeto, es necesario que todos esos pixeles cumplan la desigualdad:

$$\left|dist_{B_t} - dist_{F_t}\right| > \tau \tag{28}$$

En el momento en que algún pixel no cumpla dicha desigualdad, se concluye que son pixeles de background, es decir, se cambia el valor de bg a true y se añade un cero al vector comparación (sólo uno por canal y ángulo). Como cabe la posibilidad de que algún pixel para el canal y ángulo procesado cumpla la desigualdad (28) pero no se considere objeto porque algún pixel con su mismo ángulo y canal no la haya cumplido, sólo actualizarán el background (según la ecuación (12)) aquellos pixeles que no cumplan la desigualdad (28).

En el caso de que todos los pixeles con un determinado canal y ángulo cumplan la desigualdad (28), se clasificarán dichos pixeles como objeto, añadiendo un uno (solo uno por canal y ángulo) al vector comparación y manteniendo el valor de bg a false. Además, es necesario guardar el índice de uno de estos pixeles (tanto de la nube actual como de background), para acceder a su valor de x, y, z y distancia más adelante.

El índice $indexi$ se pone a cero cuando se empieza a comparar un pixel del background, es decir, se pondría a cero al comparar $pix1b - pix1a$ y $pix2b - pix1a$ de la *Figura* ***11***. El valor

de $indexi$ va aumentando conforme se compara cada uno de los pixeles de background con todos los de la nube actual que tengan el mismo canal y ángulo, esto es, al comparar $pix1b - pix2a$ el valor sería $indexi = 1$, al comparar $pix1b - pix3a$ valdría $indexi = 2$ y así sucesivamente.

En resumen, para el caso de la *Figura 11*, los valores del índice $indexi$ son:

Pixeles a comparar	Valor de indexi
$\mathrm{pix1b} - \mathrm{pix1a}$	$\mathrm{indexi} = 0$
$\mathrm{pix1b} - \mathrm{pix2a}$	$\mathrm{indexi} = 1$
$\mathrm{pix1b} - \mathrm{pix3a}$	$\mathrm{indexi} = 2$
$\mathrm{pix2b} - \mathrm{pix1a}$	$\mathrm{indexi} = 0$
$\mathrm{pix2b} - \mathrm{pix2a}$	$\mathrm{indexi} = 1$
$\mathrm{pix2b} - \mathrm{pix3a}$	$\mathrm{indexi} = 2$

Tabla 5: Actualización del índice indexi

Al comparar los últimos pixeles con un mismo canal y ángulo, el valor del índice $indexi$ se añade el índice que recorre la nube de puntos actual, es decir, a ii. De esta manera, se tendrá situado el índice ii en el último pixel procesado de la nube actual de puntos.

El índice $ultii$ debe apuntar al próximo pixel de la nube de puntos actual que se tenga que procesar (es decir, al siguiente elemento apuntado por ii). Por ello, su valor es una unidad más que el índice ii. Esto es necesario ya que al comparar el último pixel con el mismo canal y ángulo ($\mathrm{pix2b} - \mathrm{pix3a}$ para el caso de la figura 3), el algoritmo se sale del for que recorre los pixeles de la nube de puntos actual (del cual se encarga el índice ii). Al continuar dentro del for que recorre el background (del cual se encarga el índice i), se prosigue con el siguiente elemento apuntado por i y se vuelve a entrar en el bucle for que recorre la nube de puntos actual. Como no interesa que empiece a recorrer esta nube desde el principio, se utiliza el índice $ultii$ y por tanto, empieza a recorrerse desde el primer pixel que no se ha procesado aún.

Una vez terminada esta rutina, se obtienen rellenas las siguientes variables:

Nombre	Tipo	Descripción
Comparación	Vector de unos y ceros	Si el elemento es cero indica que el pixel (de la nube de puntos actual) en esa posición es background. Si es uno, indica que el pixel es foreground
Index_ob	Vector de enteros sin signo	Contiene el índice respecto a la nube de puntos actual de los pixeles foreground
Index_bg	Vector de enteros sin signo	Contiene el índice respecto al background de los pixeles foreground

Tabla 6: Variables al terminar la comparación de pixeles

Aunque en la *Tabla 6* solo aparezca un vector de cada tipo, en realidad son cuatro (uno por canal), puesto que la detección se hace de forma independiente al canal.

Una vez terminada la parte de comparación de los pixeles actuales con los de background, se recorre el vector comparación de cada canal para determinar si los pixeles clasificados como foreground (y que están a uno en dicho vector) son realmente objetos. La metodología empleada es la siguiente:

Se utilizan los contadores $count\_1$ y $count\_0$. El contador $count\_0$ solo cuenta el número de ceros seguidos en el caso de que se haya encontrado un uno anteriormente, es decir, si $count\_1 > 0$. El contador $count\_1$ aumenta una unidad cada vez que se encuentra con un uno en el vector comparación. Cuando se encuentra un elemento a uno en el vector comparación, se cuentan los unos que hay con una separación de menos MAX_CEROS ceros seguidos. Cuando se encuentran MAX_CEROS ceros seguidos, se da por terminado el objeto y se estudian los pixeles que lo forman para determinar si pertenecen realmente a un objeto. Como puede haber más de un objeto, es necesario almacenar en una variable llamada ind_ini el número de unos (de otros objetos) que hay antes del primero uno del objeto a estudiar. Este dato será necesario en la función $check\_obj\_retirado$ como se verá en el apartado 4.2.10. Para determinar el valor de ind_ini es necesaria otra variable, llamada j, que lleva la cuenta del número de unos anteriores a la posición actual.

Aunque a priori se haya establecido que a partir de MAX_CEROS ceros seguidos estamos ante otro objeto, puede que haya dos objetos tan cercanos que los ceros entre ellos sean menos que MAX_CEROS. Para ello, se utilizan las variables $count\_1\_2$ y ind_ini_2, cuya funcionalidad es la misma que $count\_1$ y ind_ini y cuya actualización se verán en el apartado 4.2.10.

Las condiciones que deben cumplir estos pixeles para afirmar que pertenecen a un objeto son:
a) El número de unos ($count\_1$) sea mayor que MIN_PIX_OBJ, es decir, que el objeto esté formado por más de MIN_PIX_OBJ pixeles.
b) El tamaño del objeto sea mayor que TAM_MIN (el tamaño lo calcula la función $calc\_tamaño$).
c) Los pixeles se diferencien de los del background inicial y los pixeles del objeto sean similares entre ellos (de estas dos últimas condiciones se encarga la función $check\_obj\_retirado$).
d) El objeto aparezca en una posición parecida durante REP_OBJ veces.
Las funciones presentes en estas condiciones se verán en los apartados siguientes.

De forma resumida, la función $check\_obj\_retirado$ elimina aquellos pixeles que no sean parecidos al anterior o que sean similares al background inicial, por lo que el número de pixeles que forman el objeto puede variar al ejecutar dicha función. Si devuelve el valor true es que ningún pixel del objeto es válido (ya sea porque es parecido al background inicial o porque son muy distintos entre ellos) y se asumirá que el objeto ha sido retirado.

En caso de que el valor de esta función sea false, se vuelve a comprobar que el número de pixeles objeto es mayor que MIN_PIX_OBJ y que su tamaño es mayor que TAM_MIN (ahora el tamaño se calcula con la función $calc\_tamaño\_fin$). Es necesario volver a hacer esta comprobación porque $check\_obj\_retirado$ puede cambiar el número de pixeles que forman el objeto.

Si cumple estas condiciones (a,b,c y las del párrafo anterior), se calcula la media de la posición del objeto con la función $mean$ (apartado 4.2.16) y se almacena en los vectores x_obj, y_obj, z_obj, $dist\_obj$. Una vez aquí, se aumenta en uno la variable que cuenta el número de objetos ($num\_objects$) y otra variable ($times\_obj$) que cuenta el número de veces seguidas que aparece ese objeto.

A continuación, hay que comprobar que la posición en la que aparece ese objeto es similar a la del instante anterior. Si no fuera así, es probable que fueran pixeles aislados que se han salido de la media del background, por lo que no deben considerarse objetos y el valor de la variable $times\_obj$ se impone a 1 (ya que es la primera vez que se obtiene esa nueva posición del objeto). En el caso de que la posición fuera similar a la obtenida en el instante anterior y que fuera la REP_OBJ que sale, entonces se puede afirmar que es un objeto. En caso contrario, habría que borrar el último valor de los vectores x_obj, y_obj, z_obj, $dist\_obj$, ya que es una posición no valida porque el objeto se ha desechado. Esta es la finalidad de la variable booleana $imprimir\_ob$. Si $imprimir\_ob = 0$ indica que la posición calculada debe ser borrada (mediante la función $delete\_calc\_ob$ detallada en el apartado 4.2.18); si su valor es 1 indica que el objeto es válido y por tanto su posición también.

Este algoritmo tiene determinado el número máximo de objetos que puede detectar mediante la constante MAX_OBJECTS (cuyo valor puede cambiarse antes de empezar la ejecución del mismo). Si en algún momento se alcanza el número máximo de objetos, esto es, $num\_objects \geq MAX\_OBJECTS$, se dejarían de detectar los demás objetos, pero no se bloquearía el algoritmo.

Una vez que se ha terminado de recorrer todo el vector comparación, se imprimen las posiciones de todos los objetos que se han detectado y cumplido todas las condiciones (en caso de que los hubiera). Para ello se utiliza la función $print\_ob$.

Cabe la posibilidad de que haya dos objetos muy cercanos en el eje x (pero cuya distancia al LIDAR sea bastante diferente), tanto que su separación sea menor de MAX_CEROS y por tanto se detectaría como un único objeto. Tal y como se ha definido hasta ahora la función $check\_obj\_retirado$, los pixeles correspondientes al segundo objeto se eliminarían ya que son muy diferentes al primero. Por este motivo, cuando un pixel se diferencia del anterior se guarda el número de unos procesados hasta ese momento en ind_ini_2 y el número de unos restantes por procesar en $count\_1\_2$. Una vez procesado el primer objeto tal y como se ha descrito anteriormente, se actualizan los valores ind_ini y $count\_1$ como ind_ini_2 y $count\_1\_2$ respectivamente, se pone count_1_2 a cero y se vuelve a ejecutar desde la llamada a $check\_obj\_retirado$. Esto se consigue con un bucle while, el cual se repite mientras que el número de unos pendientes por procesar ($count\_1$) sea mayor que MIN_PIX_OBJ.

Al igual que para la primera parte del algoritmo, este procedimiento hay que repetirlo para cada canal de forma independiente.

Por último, hay que actualizar el número de veces seguidas que aparece un objeto a cero (es decir, poner a cero todos los elementos de $times\_obj$) en los siguientes casos:

- No hay ningún objeto ($num\_objects == 0$).

- El número de objetos detectados en el instante anterior es menor que el número de objetos detectados en el instante actual.
- Algún objeto ha alcanzado REP_OBJ (es decir, $times\_obj \geq REP\_OBJ$).

De esta forma, si hay más de un objeto, sus posiciones se imprimirán en el mismo instante de tiempo (ya que sus respectivos $times\_obj$ alcanzarán REP_OBJ en el mismo instante) y se podrá controlar cuando hay más de un objeto a la vez.

4.2.8. Print_ob

Su formato es:

$$void\ print\_ob\ (const\ unsigned\ int\ c)$$

Esta función recibe como parámetro de entrada el canal donde se encuentran cada uno de los objetos que se va a imprimir (ya que la detección se realiza de forma independiente para cada canal). Inicialmente, se imprime el número de objetos que se han detectado, es decir, el tamaño de cualquiera de los vectores objeto (columna izquierda de la *Tabla 4*) así como el canal en que se ha detectado.

Como los vectores objeto tienen almacenado la posición y tamaño de cada uno de los objetos detectados, hay que recorrer cada uno de sus elementos con un for (ya que cada elemento es un objeto distinto), imprimiendo la posición (x_obj, y_obj, z_obj), distancia ($dist\_obj$) y tamaño ($tamaño\_obj$) de cada uno de ellos.

Tal y como se verá en el apartado 5.3, también se imprimirán dichas coordenadas expresadas en el sistema de referencia geodésico.

4.2.9. Keep_object

Su formato es:

$$void\ keep\_object\ (const\ float\ \_x, const\ float\ \_y, const\ float\ \_z,\\ constfloat\ \_d, constfloat\ \_ha)$$

Esta función almacena los elementos dados por los parámetros de entrada en los vectores correspondientes a los vectores auxiliares del objeto, es decir, va almacenando los pixeles que componen cada uno de los objetos.

4.2.10. Check_obj_retirado

Su formato es:

$$bool\ check\_obj\_retirado\ (const\ unsigned\ int\ c, const\ Scan :: PointList\&\ nube\_act,\\ const\ unsigned\ int\ i, const\ unsigned\ int\ size, unsigned\ int\&\ ind\_ob\_cerc,\\ unsigned\ int\&\ count1\_ob\_cerc)$$

El parámetro de salida viene especificado en la *Tabla 9*.

Los parámetros de entrada son los siguientes:

Parámetro	Descripción
c	Canal al que pertenece el objeto a analizar
nube_act	Nube de puntos actual detectada por el LIDAR
i	Número de unos anteriores al objeto que se está procesando
size	Número de unos que forman el objeto que se está procesando
ind_ob_cerc	Número de unos anteriores al objeto cercano del que se está procesando
count1_ob_cerc	Número de unos que forman el objeto cercano del que se está procesando

Tabla 7: Parámetros de entrada de check_obj_retirado

Los parámetros ind_ob_cerc y $count1\_ob\_cerc$ sólo se utilizan en el caso de que haya dos objetos tan cercanos (menos de MAX_CEROS huecos de separación entre ellos) que se han detectado como uno solo y tienen la misma funcionalidad que i y $size$. En caso de que no haya ningún objeto cercano al que se está procesando, ind_ob_cerc y $count1\_ob\_cerc$ tendrán valor cero.

Además de los parámetros anteriores, es necesario incluir algunas variables nuevas:

Variables	Tipo	Descripción
count	entero sin signo	Lleva la cuenta del número de pixeles que no son realmente objetos (es decir, no han pasado alguna de las condiciones de esta función). Inicialmente valdrá cero y su valor máximo viene determinado por $size$.
obj_retirado	booleana	Variable que inicialmente está a cero y que toma valor uno cuando el objeto ha sido retirado, es decir, el valor de los pixeles objeto es muy parecido al background inicial
ult_pixel_ob	entero	Inicialmente vale -1 y cambia su valor cuando se determina que un pixel es verdaderamente un pixel objeto y se guarda en los vectores correspondientes, es decir, se llama a la función $keep\_object$

Tabla 8: Variables de la función check_obj_retirado

Esta función analiza una parte del vector $comparacion$ (determinada en la función $detect$). Por ello, es necesario indicarle como parámetro de entrada cuál es la posición del primer uno que tiene que analizar (i) y el número de unos a analizar ($size$), tal y como aparece en la figura:

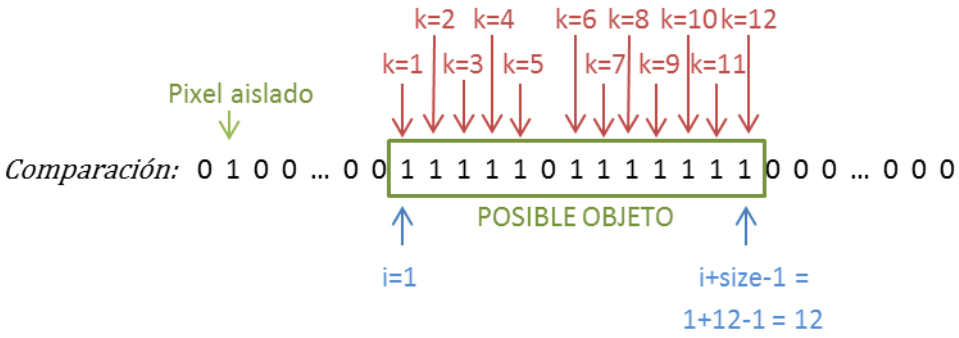

Figura 12: Índices necesarios para los objetos en check_obj_retirado

Cuando se rellena el vector $comparacion$ (en la función $detect$), solamente se guardan los índices de los elementos puestos a uno, por lo que al recorrer el posible objeto, no hay que tener en cuenta las posiciones de los ceros. Por eso $size$ únicamente cuenta el número de unos y el for de la función $check\_obj\_retirado$ sólo recorre los unos, como se muestra en la *Figura 12* con el índice k en rojo.

El objetivo de esta función es comprobar que los píxeles de un objeto sean realmente píxeles objeto y sólo en ese caso llamar a la función $keep\_object$ para que los almacene en los vectores auxiliares del objeto.

Las comprobaciones que tiene que hacer son las siguientes:

- Que los pixeles objeto no se parezcan a los píxeles del background inicial porque eso significaría que el objeto ha estado ahí pero actualmente ya ha sido retirado. En tal caso (objeto retirado), los pixeles detectados como objeto, en realidad serían pixeles de background por lo que habría que actualizar el background mediante (12) y aumentar en una unidad el contador $count$, ya que el pixel analizado no cumple con una de las condiciones.

- Que los píxeles objeto sean parecidos entre ellos. En caso contrario, podría significar que hay más de un objeto, estando estos tan cercanos (menos de MAX_CEROS huecos de separación entre ellos) que se ha detectado como uno solo; o bien, simplemente existe algún pixel aislado que se sale de la media y que no hay que tenerlo en cuenta. La condición para que dos pixeles sean parecidos es que su diferencia sea menor que un umbral establecido, en concreto, DIF_PIXEL. Cuando un pixel difiere del anterior que ha sido considerado objeto (es decir, el último almacenado por la función $keep\_object$), se aumenta el contador $count$ (ya que el pixel analizado no cumple con una de las condiciones). Además, si es el primer pixel que difiere hay que almacenar su posición en ind_ob_cerc y el número de unos que quedan por analizar en $count1\_ob\_cerc$, lo cual nos servirá para volver a entrar en esta función y determinar si hay un objeto muy cercano al anterior.

Sólo aquellos pixeles que hayan verificado ambas condiciones serán considerados como pixeles que verdaderamente pertenecen a un objeto y añadidos a los vectores auxiliares del objeto mediante la función $keep\_object$. En este caso, también se guardará (en ult_pixel_ob)

el índice del último pixel añadido, ya que éste servirá para comparar dicho pixel con el actual y ver si difieren entre ellos (es decir, para comprobar la condición segunda).

En el caso de que el contador $count$ alcance su valor máximo ($size$), el objeto se dará por retirado ya que ningún pixel de ese objeto realmente pertenece a un objeto (no se ha llamado ninguna vez a la función $keep\_object$). Por ello, se cambia el valor de la variable $obj\_retirado$ a uno, indicando que el objeto ya no está. Esta variable es el parámetro devuelto por la función, es decir, el parámetro de salida, que indica lo siguiente:

Parámetro	Valor	Significado
obj_retirado	0	Algún pixel objeto ha sido considerado como realmente un pixel objeto y por ello se ha llamado a la función $keep\_object$ para que lo almacene. En resumen, el LIDAR está detectando un objeto.
	1	Ningún pixel objeto ha sido considerado como tal, es decir, no se ha llamado nunca a la función $keep\_object$ porque ningún pixel ha cumplido las condiciones para ello. En resumen, el LIDAR no está detectando un objeto.

Tabla 9: Parámetro de salida de check_obj_retirado

Al igual que se hizo en el apartado 4.2.7, este procedimiento hay que repetirlo para canal de forma independiente.

4.2.11. Calc_tamaño

Su formato es:

$$float\ calc\_tamaño\ (const\ unsigned\ int\ c, const\ Scan :: PointList\&\ nube\_act, const\ unsigned\ int\ i, const\ unsigned\ int\ size)$$

Los parámetros de entrada son los siguientes:

Parámetro	Descripción
c	Canal al que pertenece el objeto
nube_act	Nube de puntos actual detectada por el LIDAR
i	Número de unos anteriores al objeto actual
size	Número de unos que forman el objeto actual

Tabla 10: Parámetros de entrada de calc_tamaño

Como parámetro de salida se devuelve el tamaño del objeto (en metros) calculado por esta función, mediante la variable tam.

Esta función calcula el tamaño del objeto determinado por los índices i y $size$ (cuyo significado ya se vio en la *Figura 12*). Para ello, únicamente se necesita el valor del primer y último pixel

de la nube de puntos del objeto, determinada entre i y $size$, y calcular la hipotenusa entre ellos. Gráficamente:

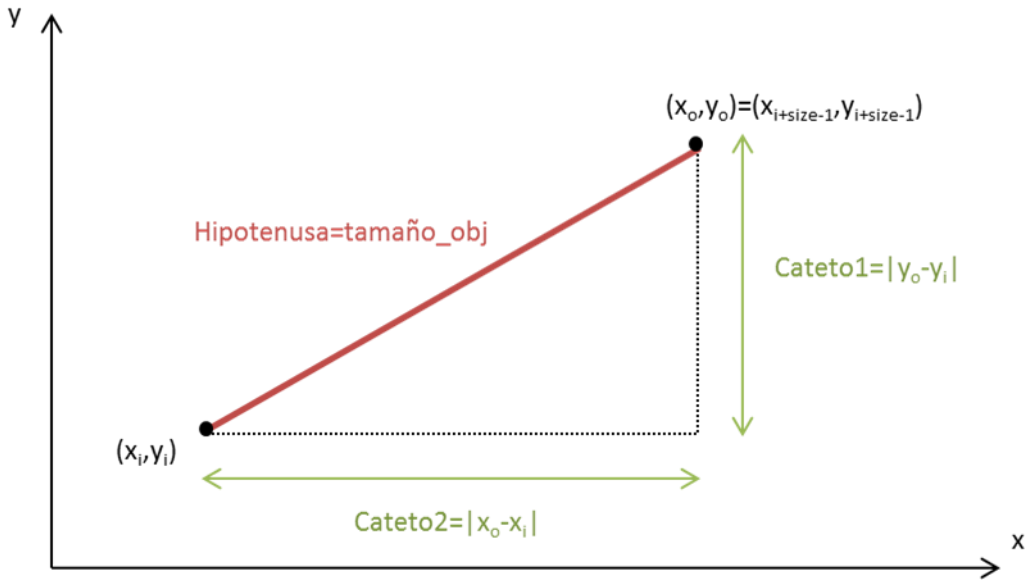

Figura 13: Forma gráfica de calcular el tamaño del objeto

Tal y como se vio en la *Figura 12* el primer y último pixel de un objeto vienen determinados por los índices i y $i + size - 1$, por tanto los valores de la nube de puntos actual ($nube\_act$) en estos índices determinará los puntos claves dibujados en la *Figura 13*.

Una vez que se obtienen estos valores, se calcula la hipotenusa mediante la función $calc\_hip$ que se verá en el apartado 4.2.13, que devuelve la medida de la hipotenusa, es decir, el tamaño del objeto.

El tamaño devuelto por esta función no es exactamente el que buscamos, ya que se ejecuta antes de la función $check\_obj\_retirado$ que puede modificar el número de pixeles de los que está formado el objeto y por consiguiente su tamaño. Para ello, existe otra función ($calc\_tamaño\_fin$), cuyo objetivo es el mismo que esta función pero se ejecuta después de $check\_obj\_retirado$, devolviendo el tamaño real del objeto.

4.2.12. Calc_tamaño_fin

Su formato es:

$$float\ calc\_tamaño\_fin\ (\ )$$

Esta función hace lo mismo que $calc\_tamaño$ con la diferencia de que los valores del primer y último pixel no los obtiene de la nube de puntos detectada por el LIDAR sino de los vectores auxiliares del objeto, es decir, de aquellos pixeles que se han comprobado que verdaderamente pertenecen al objeto. Por este motivo, no es necesario ningún parámetro de entrada del tipo de i y $size$ (como lo eran en el apartado 4.2.11), ya que el elemento inicial de un vector es el que está en la posición 0 y el último es el que está en la posición indicada por el tamaño del vector:

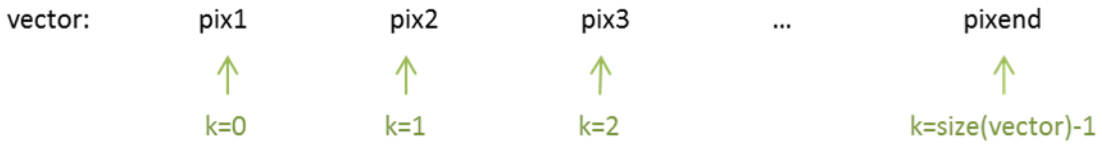

Figura 14: Índices en un vector cualquiera

4.2.13. Calc_hip

Su formato es:

$$float\ calc\_hip\ (float\ x1, float\ x2, float\ y1, float\ y2)$$

Esta función calcula la hipotenusa de un triángulo formado por los vértices (x1,y1), (x2,y2), (x2,y1). Gráficamente:

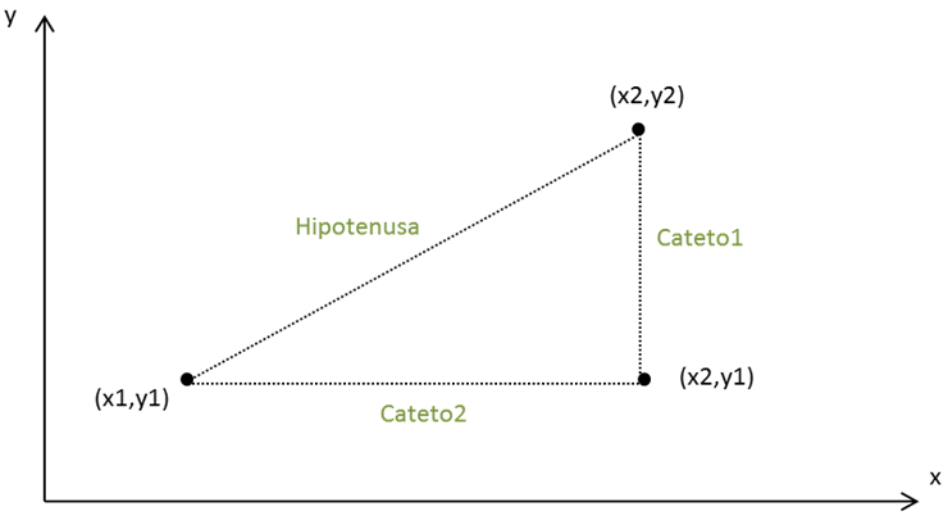

Figura 15: Triángulo

Por el teorema de Pitágoras, la hipotenusa se obtiene mediante la fórmula:

$$h^2 = c1^2 + c2^2 \tag{29}$$

donde h es la hipotenusa, c1 uno de los catetos y c2 el otro de los catetos. En el caso de la *Figura 15*:

$$c1 = |y2 - y1| \tag{30}$$

$$c2 = |x2 - x1| \tag{31}$$

Sustituyendo en (29):

$$h = \sqrt{(y2 - y1)^2 + (x2 - x1)^2} \tag{32}$$

La ecuación (32) es la que se implementa en esta función, devolviendo el valor de la hipotenusa.

4.2.14. Clear_ob_aux

Su formato es:

$$void\ clear\_ob\_aux\ (\ )$$

Esta función limpia los vectores auxiliares del objeto, es decir, x_obj_aux, y_obj_aux, z_obj_aux,$dist\_obj\_aux$ y $hAngle\_obj\_deg\_aux$.

4.2.15. Clear_ob

Su formato es:

$$void\ clear\_ob\ (\ )$$

Esta función limpia los vectores que contienen la posición media de los objetos, es decir, x_obj, y_obj, z_obj,$dist\_obj$ y $tamaño\_obj$.

4.2.16. Mean

Su formato es:

$$void\ mean\ (\ )$$

Esta función calcula la media de los vectores auxiliares del objeto, es decir, la media de los pixeles que componen el objeto y se almacenan en las variables med_x, med_y,med_z y med_d, inicializadas a cero.

Para ello, se calcula inicialmente el tamaño de los vectores y se almacena en una variable llamada tam_obj. A continuación, se recorren los vectores auxiliares, de tal manera que el valor del pixel actual se divide entre el tamaño del objeto (tam_obj) y se suma al valor obtenido hasta el momento de la media (med_x para el caso de x_obj_aux; med_y para el caso de y_obj_aux y así sucesivamente).

Por último, el valor final de la media, se almacena en los vectores que contienen la posición media de cada objeto, es decir, en x_obj, y_obj,z_obj y $dist\_obj$.

4.2.17. Clear_all_possible

Su formato es:

$$void\ clear\_all\_possible\ (\ )$$

Esta función limpia los vectores correspondientes a todos los índices que se necesitan durante la ejecución de $detect$, los vectores de comparación y los correspondientes a los objetos (mediante la llamada a las funciones $clear\_ob$ y $clear\_ob\_aux$).

Esquemáticamente, los vectores borrados son:

Índices		Comparación	Objetos	
index_ob_c0	index_bg_c0	comparacion_c0	x_obj	x_obj_aux
index_ob_c1	index_bg_c1	comparacion_c1	y_obj	y_obj_aux
index_ob_c2	index_bg_c2	comparacion_c2	z_obj	z_obj_aux
index_ob_c3	index_bg_c3	comparacion_c3	dist_obj	dist_obj_aux
			tamaño_obj	hAngle_obj_deg_aux

Tabla 11: Vectores borrados en clear_all_possible

4.2.18. Delete_calc_ob

Su formato es:

$$void\ delete\_calc\_ob\ (\ )$$

Esta función elimina el último elemento almacenado en los vectores que contienen la posición media del objeto detectado, es decir, en x_obj, y_obj,z_obj y $dist\_obj$.