UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un dispositivo de entrada multiusuario basado en punteros láser

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: García Torres, Julián

Tutor: Abad Cerdá, Francisco José

Curso 2015/2016

Resumen

El objetivo de este trabajo final de grado es implementar un sistema para permitir a uno o más usuarios interaccionar con las aplicaciones mediante punteros láser. Varios usuarios podrán usar sendos punteros como dispositivos apuntadores para trabajar sobre una superficie de trabajo. Dicha superficie de trabajo será normalmente la salida de un proyector. Por medio de una webcam, el sistema es capaz de detectar e identificar uno o más punteros láser, basándose en parámetros como el color para distinguirlos entre sí. El tracking de los punteros obtiene su posición, área, color, etc., teniendo en cuenta su historia. Nuestro sistema utiliza el protocolo TUIO para enviar los mensajes con esa información a una aplicación cliente, que será la encargada de implementar la salida gráfica dependiendo de la posición de los punteros. Dicho protocolo es ampliamente utilizado en las aplicaciones multitáctiles.

Palabras clave: Láser, multi-usuario, proyector, webcam, TUIO.

Abstract

The aim of this final degree project is to implement a system to allow one or more users interact with applications using laser pointers. Multiple users can use two separate pointers as pointing devices to work on a work surface. This working surface is normally the output of a projector. By means of a webcam, the system is able to detect and identify one or more laser pointers, based on parameters such as color to distinguish between them. The tracking pointers gets his position, area, color, etc., given its history. Our system uses the TUIO protocol to send messages with that information to a client application, which will be responsible for implementing the graphical output depending on the position of the pointers. This protocol is widely used in multitouch applications.

Keywords : Láser pointer, multi-user, projector, webcam, TUIO.

Tabla de contenidos

1. Introducción

En estos últimos años los dispositivos que más utilizamos se han vuelto táctiles. Ahora tenemos *smartphones*, *tablets*, ordenadores y muchos otros dispositivos que somos capaces de controlar sin necesidad de botones. Esto es gracias a las pantallas táctiles que se han incorporado a ellos, y que poco a poco han ido mejorando para darnos una experiencia cómoda y agradable a los usuarios.

En su constante avance la tecnología nos brinda nuevas formas de interactuar con nuestros dispositivos y aplicaciones. Aparte de los ya mencionados dispositivos táctiles, también podemos interactuar con muchos dispositivos mediante nuestra voz e incluso hay aplicaciones capaces de reconocer nuestros rostros o huellas dactilares para, por ejemplo, desbloquear los móviles, validar que somos nosotros para entrar a gimnasios o salas, etc.

Con el fin de ayudar en el aumento de medios que disponemos para interactuar con nuestras aplicaciones, en este proyecto hemos construido un sistema que permite a uno o más usuarios interaccionar con las aplicaciones mediante uno o varios punteros láser. Por medio de una *webcam*, el sistema es capaz de detectar e identificar uno o más punteros láser basándose en parámetros como el color para distinguirlos entre sí y de otros posibles elementos de la imagen. El tracking de los punteros se realiza teniendo en cuenta su posición, área, color, etc., y también su historia. La aplicación utiliza el protocolo TUIO para enviar los mensajes con las coordenadas y el identificador de cada puntero a una aplicación cliente, que será la encargada de implementar la salida gráfica dependiendo de la posición de los punteros.

Para la realización de este proyecto se estudiaron y analizaron diferentes aplicaciones existentes con funcionalidad similar. En el capítulo siguiente se describen estas aplicaciones.

El principal problema a resolver para la realización de este proyecto ha sido el desarrollo de algoritmos para la detección robusta de los punteros en la imagen. Como se ha comentado, se utilizará una webcam para capturar la zona de trabajo y punteros láser manejados por los usuarios para señalar un punto de dicha zona. Normalmente la zona de trabajo se corresponderá con la pantalla de un proyector, mientras que la imagen captada por la webcam será más grande que dicha zona de trabajo.

El láser será un punto brillante en la imagen capturada, pero dependiendo de las condiciones de luz del entorno, puede ser que haya otros puntos brillantes que se puedan confundir con un puntero. Este proceso es especialmente delicado si se usa un proyector, ya que en ciertos casos producen un brillo característico en el centro de la imagen (ver Figura 1).

Figura 1. Proyector con reflejo

A la hora de determinar cuáles de los puntos brillantes de la imagen se consideran punteros y cuáles no, se utilizará una serie de heurísticas basadas en las características de dichos puntos. Para cada región brillante detectada o *blob*, se tiene en cuenta su área, su color, su forma, etc. Otra característica que se tiene en cuenta a la hora de determinar si un *blob* es un puntero láser o no es su historia previa. Los reflejos generados por la bombilla del proyector o por otras fuentes de luz dentro de la imagen capturada por la webcam son principalmente estáticos, mientras que el puntero siempre estará en movimiento, incluso aunque el usuario pretenda mantenerlo estático. Todas estas características hacen más fácil la tarea de diferenciar entre un láser y un reflejo.

La salida de nuestro sistema es una serie de mensajes basados en el protocolo TUIO, utilizado por muchas aplicaciones con interfaces multitáctiles. Esto nos permitirá utilizar aplicaciones existentes simulando con nuestra aplicación una pantalla táctil, es

decir, se utilizarán los punteros como si fueran contactos de los dedos de los usuarios con dichas pantallas.

Los mensajes TUIO correspondientes a un contacto con la pantalla táctil contienen las posiciones (x,y) de los láseres en coordenadas normalizadas.

Se ha probado el funcionamiento del sistema con varias aplicaciones cliente compatibles con el protocolo TUIO. Como este proyecto puede verse influenciado por la calidad de la cámara web y el entorno, se han probado diversas cámaras y entornos diferentes, describiendo las condiciones ideales para el uso del sistema.

Este sistema puede tener infinidad de utilidades, como por ejemplo juegos multiusuario donde los participantes tengan láseres de distinto color. También se pueden implementar aplicaciones colaborativas donde uno, dos o más usuarios están trabajando simultáneamente en el mismo proyector, haciendo anotaciones, o construyendo un modelo en paralelo. En resumen, en este proyecto implementamos una pantalla multitáctil sin contacto.

El resto de la memoria sigue el siguiente esquema. Primeramente, en el apartado 2, se muestran una serie de aplicaciones similares a la que se ha desarrollado durante el transcurso de esta memoria. En el apartado 3 se hace un análisis de la aplicación, se comenta el flujo del programa y se exponen que características se deben alcanzar. En el apartado 4 se describe detalladamente como funciona la aplicación. Más adelante en el siguiente apartado se estudian los resultados obtenidos y se hacen una serie de pruebas para comprobar la calidad de la aplicación. Seguidamente, en el apartado 6, se expondrán las conclusiones a las que se han llegado en el transcurso de la realización del proyecto. El apartado 7 muestra un pequeño manual de usuario para poder manejar la aplicación. El capítulo 8 contiene una serie de anexos. Por último en el apartado 9 se puede ver la bibliografía.

2. Estado del arte

Antes de empezar a desarrollar este proyecto o cualquier otro de índole similar es importante saber de qué herramientas disponemos y qué aplicaciones existentes nos pueden ayudar, bien para basarnos en ellas, o bien para que realicen una tarea de nuestro trabajo. No se quiere reinventar la rueda, ni desaprovechar el tiempo en construir aplicaciones desde cero. Además de invertir el tiempo en nuestro proyecto de una forma más eficiente, se puede sacar provecho de las librerías disponibles que ofrecen implementaciones robustas, estables y eficientes.

2.1 ReacTIVision

ReacTIVision [1] es un *framework* de código abierto para hacer reconocimiento y seguimiento rápido y robusto de marcadores fiduciales *(ver figura 2)*, los cuales se pueden adherir a objetos físicos. Esta herramienta también nos permite hacer el seguimiento *multitáctil* es decir, los contactos de varios dedos simultáneamente en una pantalla. Fue diseñado principalmente como un conjunto de herramientas para el rápido desarrollo de interfaces basadas en TUI (*Tangible User Interfaces*) y superficies interactivas *multitáctiles*. Este *framework* ha sido desarrollado por Martin Kaltenbrunner y Ross Bencina como el componente subyacente del sensor de la Reactable [2], un sintetizador modular táctil que ha establecido los estándares para aplicaciones *multitáctil.*

Figura 2. Ejemplo de imágenes fiduciales utilizadas por reacTIVision

El funcionamiento es sencillo, se ilumina una superficie/pantalla desde el interior con una luz infrarroja, la cual se refleja en ella y es captada por una cámara. La cámara es la encargada de detectar los marcadores fiduciales y las pulsaciones táctiles en la superficie. Detecta tanto la posición de los objetos como su orientación además del

movimiento de los dedos sobre la pantalla. Al ser una tecnología *multitáctil* varias personas pueden interactuar colaborativamente. Una vez detectados ya bien sean los marcadores como las pulsaciones, este framework encapsula toda la información de cada uno de los elementos y los envía como mensajes TUIO vía UDP al puerto 3333. Los mensajes pueden ser interpretados por cualquier aplicación cliente que sea capaz de entender estos mensajes. La figura 3 muestra el diagrama de su funcionamiento.

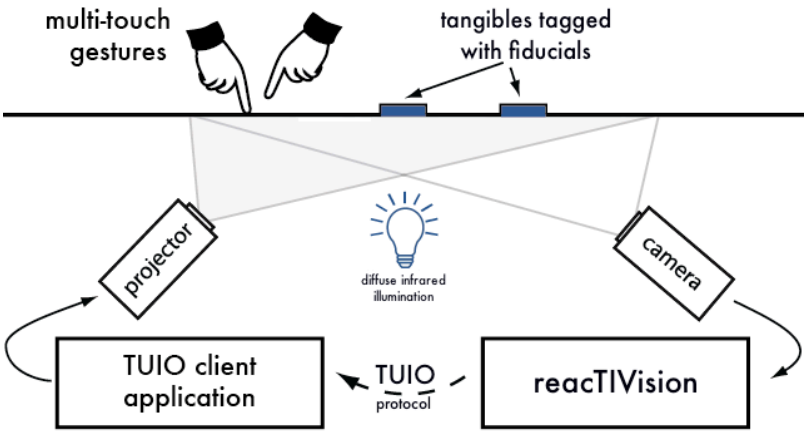


Figura 3. Diagrama de funcionamiento reacTIVision

Fuente: http://reactivision.sourceforge.net

Este sistema es parecido al que se describirá en esta memoria: utiliza una cámara para captar las posiciones de los contactos dactilares y de los marcadores, el protocolo TUIO para el envío de mensajes y un proyector para la salida. Sin embargo, reactivision no tiene en cuenta la detección de los láseres, funcionalidad que se implementará en este proyecto.

2.2 Community Core Vision

Community Core Vision, o CCV [3] es una solución multi-plataforma de código abierto para el seguimiento de *blobs* utilizando visión por computador. CCV puede interactuar con varias cámaras web y dispositivos de vídeo, así como conectarse a varias

aplicaciones compatibles con OSC / XML TUIO y soporta muchas técnicas de iluminación *multitáctil*, incluyendo: FTIR, DI, DSI, y LLP con la expansión prevista para las futuras aplicaciones de visión (módulos personalizados / filtros). Este proyecto está desarrollado y mantenido por la *NUI Group Community* (http://nuigroup.com/go/lite).

De esta aplicación se han estudiado ciertas funcionalidades muy interesantes. Como se puede ver en la Figura 3, los *blobs* detectados tienen un identificador, es decir, en esta aplicación se está utilizando un algoritmo para detectar qué *blobs* de distintas imágenes son en realidad el mismo. Esto es capaz de hacerlo mediante distancias entre blobs y el algoritmo KNN (*K Nearest Neighbors*). Además, se pueden ajustar muchos parámetros como el tamaño mínimo y máximo de un blob, el umbral, el umbral de movimiento, el desenfoque de la imagen, etc.

Figura 4. Interfaz CCV

CCV sigue sin dar una solución completa a nuestro problema original porque sigue sin tener en cuenta los láseres ni tampoco su color, pero nos proporciona unos componentes que sí serán útiles para captar el movimiento a la hora de la detección de los punteros láser. Los pasos son los que se pueden ver en la figura 4 abajo: capturar fondo, desenfoque, paso alto y amplificación.

2.3 Ortholumen

Ortholumen [4] es un sistema basado en láseres para la interacción con pantallas táctiles (figura 5). La luz del láser se proyecta desde arriba sobre una pantalla translúcida horizontal y es rastreada por una webcam que está debajo de la pantalla, mirando hacia arriba. La salida del sistema consiste proyectar de nuevo en la misma pantalla el recorrido del láser. El punto de luz elíptica proyectada por el láser informa al sistema de la posición, la orientación y la dirección del mismo. Por ahora se han probado dos aplicaciones, una de dibujo y un mapa de navegación.

Ortholumen se puede ampliar para realizar un seguimiento de múltiples láseres del mismo o diferentes colores. El sistema emplea sólo componentes de bajo coste, haciendo que sea asequible para usuarios domésticos.

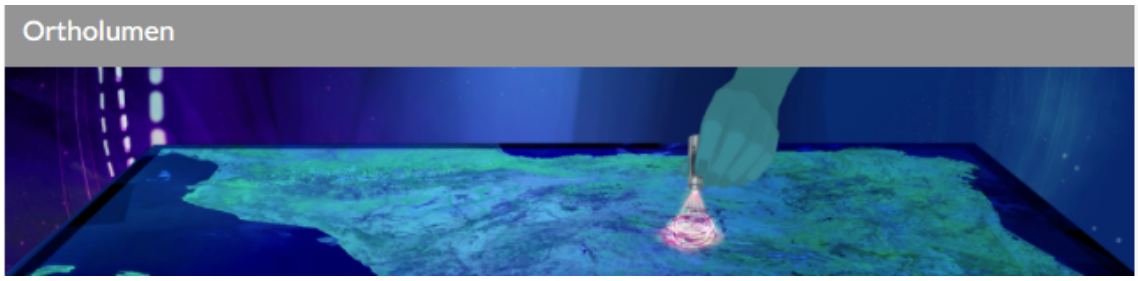


Figura 5. Pantalla táctil manejada por un láser.

Vídeo: https://www.youtube.com/watch?v=gMowayRV0VY

En este sistema se están utilizando punteros láser pero se están proyectando directamente sobre una superficie táctil, y además la distancia es muy pequeña. En el proyecto que se presenta en esta memoria los punteros se lanzan contra un proyector y a una mayor distancia por lo que será más complicada la detección de su posición y del color.

2.4 Seguimiento complejo utilizando el filtro de Kalman

El filtro del Kalman es una herramienta matemática que permite calcular y predecir la posición de un objeto. El filtro trabaja en dos etapas: la de predicción y la de actualización. La etapa de predicción permite predecir la posición del objeto mediante su historia, la velocidad de movimiento del mismo y las ecuaciones que identifican su movimiento. La predicción se corrige cada vez que una medida del estado del objeto está disponible mediante el paso de actualización. Lamentablemente las medidas no

son perfectas y suelen haber errores, por lo que los pasos de predicción y de actualización se ponderan utilizando la información que reciben de los errores de medición y predicción.

Como el filtro de Kalman es un algoritmo recursivo, este pude ejecutarse en tiempo real usando únicamente las mediciones de entrada actuales, el estado calculado previamente y su matriz de incertidumbre, y no requiere de ninguna información pasada adicional.

No hemos utilizado el filtro de Kalman en este proyecto por la dificultad de la selección de los parámetros del algoritmo y porque típicamente introduce un retardo en la interacción (a cambio de suavizar un posible movimiento errático y predecir la posición del objetivo cuando no es visible en la imagen).

3. Análisis

3.1 Descripción general

En este proyecto se estudia y se desarrolla una aplicación que capta punteros láser en el flujo de video capturado por una cámara, los identifica, obtiene sus coordenadas, su color, su área, si está estático o en movimiento y envía mensajes con las posiciones como si fueran pulsaciones táctiles a una aplicación cliente que reaccionará a estos mensajes. Si se combina este sistema con un proyector se podrá controlar una aplicación táctil mediante punteros láser sin tener que levantarnos o estar en contacto con la pantalla.

Uno de los objetivos de este proyecto es una detección robusta de los láseres. Para ello se utilizarán varias características de cada blob para así calcular su posición y poder diferenciar los punteros entre sí. Como se ha descrito anteriormente, el láser es un punto brillante en la imagen (probablemente el más brillante), pero la cantidad de luz que capta la cámara depende del entorno. El principal problema de los entornos con mucha iluminación es que puede llegar a saturar la imagen. Cuando se satura la imagen captada por la cámara hay una pérdida importante de información, ya que cualquier zona del sensor que reciba más de un determinado nivel de iluminación resultará en el mismo valor: 255, típicamente en los tres canales de color de la imagen. La saturación puede hacer que partes de la imagen se vean blancas sin serlo y a la vez perder la capacidad de encontrar el punto más brillante de la misma (ya que a partir de un valor de iluminación, el resultado de la digitalización se trunca a 255). Por ello, en entornos luminosos se hace necesario estudiar métodos para reducir el nivel de luz que llega al sensor. Más adelante se describirán los distintos métodos estudiados para conseguirlo.

Dada la naturaleza dinámica de los punteros láser manejados por los usuarios, se ha optado por implementar un algoritmo de detección de movimiento. Los algoritmos de detección de movimiento se basan en el cálculo de la diferencia entre frames consecutivos. La diferencia entre frames nos muestra los objetos cuya posición ha cambiado, es decir, están en movimiento. Una vez se detecta un objeto se calculan sus características como su posición, el área, el color predominante del objeto y mediante su historia se puede saber si ese objeto está estático o no. Estas características ayudan a tomar la decisión de si el objeto que se ha encontrado es un puntero láser o no.

Una vez detectados los punteros el siguiente paso consiste en enviar sus coordenadas mediante el protocolo TUIO. Esto permitirá utilizar nuestra aplicación con cualquier otra que entienda estos mensajes.

Descripción del algoritmo

La figura 6 muestra el diagrama de flujo del algoritmo implementado.

Uno de los primeros pasos es reducir la cantidad de luz que capta la cámara web. Esto ayudará enormemente a la hora de reducir los reflejos y brillos indeseados en la imagen y también dará una ayuda extra para detectar el color del láser.

Una vez iniciado el programa se obtiene la imagen a partir de la webcam, de esta imagen guardamos una copia como fondo o *background.*

El siguiente paso es convertir la imagen a escala de grises, donde los puntos más brillantes serán blancos, los intermedios grises y los más oscuros negros.

Para diferenciar las áreas de la imagen más brillantes se utilizará un algoritmo de binarización, que convierte la imagen en escalas de grises a una imagen binaria. Para ello, estos algoritmos utilizan un umbral, que determina el punto de corte a partir del cual un pixel acaba siendo 0 ó 1. Para seleccionar dicho umbral se realiza un histograma de la imagen, de esta forma se calculan cuantos pixeles hay en un determinado nivel de iluminación, es decir, se sabe cual es el valor del/los pixel/es más luminosos. A partir de esta información se puede binarizar la imagen en función de ese valor más alto menos un valor para tener un pequeño margen.

El siguiente paso es detectar los objetos más brillantes en movimiento haciendo una substracción del fondo obtenido anteriormente. El problema aquí es que los láseres pueden ser pequeños, por lo que se tiene que agrandar el objeto en la imagen. Para ello se va a realizar un desenfoque de la imagen de la diferencia con el fin de aumentar el área del objeto en movimiento.

Una vez conseguida la imagen con los posibles objetos que podrían ser láseres, se procede a hacer el seguimiento de los mismos. Para ello se detectan los contornos en la imagen binaria, para calcular el área y las coordenadas (x, y) del objeto. En esta parte se hacen las comprobaciones para definir si es un puntero láser o no. También se hace el seguimiento para ver si el láser ya estaba en la imagen anterior y saber si es el mismo, es decir, obtener su historia. Así mismo en esta parte también se van a detectar los punteros láser de color rojo y los punteros de color verde en esa misma imagen, ya que en este proyecto se utilizarán láseres de dicho color. Esto no significa que solo se

pueda tener dos punteros en pantalla, se pueden tener tantos como se quiera y cada uno tendrá su identificador.

Después de localizar los láseres, se les asigna un identificador numérico: puntero 0, puntero 1, puntero 2, etc., se empaquetan los datos de cada láser y se envían como mensaje a una aplicación cliente por UDP, mediante el protocolo TUIO, que estará a la espera de recibir los mensajes en un puerto (normalmente el 3333).

Este proceso se repite mientras se ejecuta la aplicación para cada *frame* capturado por la cámara.

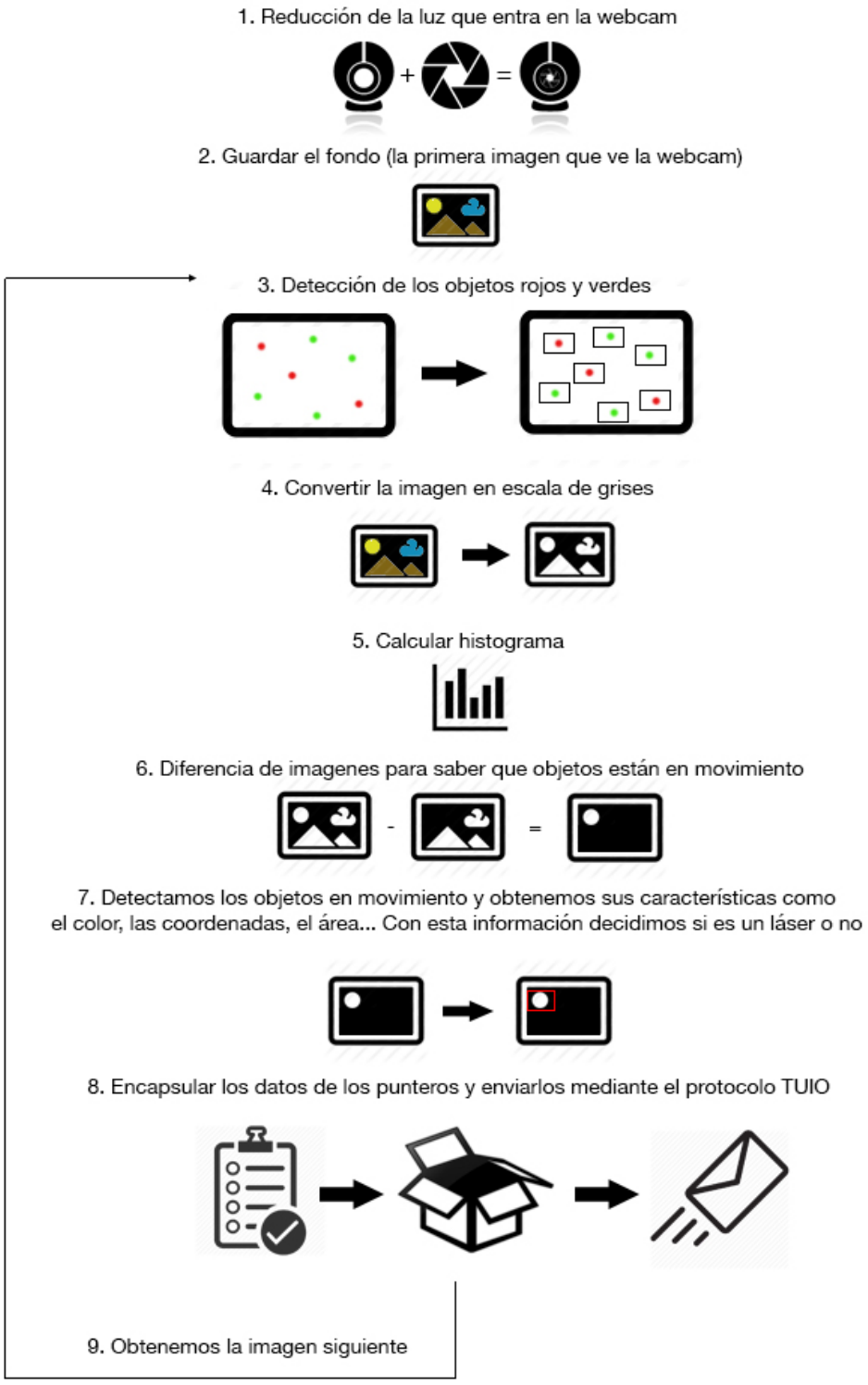


Figura 6. Diagrama de flujo de la aplicación

Más adelante en el apartado 4 "Diseño" se explicará con mucho más detalle cada uno de los pasos descritos.

3.2 Funcionalidades

Como se comentó en el capítulo 1, la captura se ve afectada por el entorno. El uso del sistema en habitaciones con mucha iluminación o en habitaciones más oscuras puede afectar a la tasa de detección de los punteros. Por ello la aplicación dispone de una serie de utilidades o ajustes para mejorar la experiencia del usuario para adaptarse a su entorno. Se han definido una serie de teclas mediante las que el usuario puede:

- definir el umbral de iluminación mediante el cálculo del histograma de la imagen capturada por la cámara
- capturar el fondo de forma manual,
- habilitar o deshabilitar el seguimiento de los láseres manualmente,
- cambiar el modo de detección de modo *background* a diferencia. En caso de que la imagen del proyector fuera estática al poner el modo *background* de detección ahorramos tiempo de cómputo, por lo que hay una mayor fluidez.
- definir el tamaño máximo del área que se puede considerar como un puntero láser en tiempo de ejecución para descartar objetos más grandes en movimiento
- definir la zona de trabajo mediante sus cuatro esquinas. Esto permite ajustar las coordenadas generadas por el sistema a las del proyector.

4. Diseño

Para la realización de este dispositivo de entrada multiusuario basado en punteros láser hemos utilizado el lenguaje C++ y dos librerías: OpenCV [5] y TUIO [7]. OpenCV con sus clases y funciones es muy útil para el procesado de las imágenes que se capturan con la cámara, además de que permite ajustar ciertos aspectos de la cámara. La librería TUIO permite enviar mensajes OSC como hacen las aplicaciones de reactivision (apartado 2.1) y CCV (apartado 2.2). Estos mensajes son los que mandan las coordenadas de los láseres a la aplicación cliente por UDP la cual los recibe como pulsaciones táctiles. Además se ha implementado un nuevo tipo de objeto Blob el cual se definen sus características en el Anexo A.

El primer paso a tener en cuenta es que nuestro proyecto depende del entorno, es decir, funciona mejor en un entorno con menos luz que en un entorno muy iluminado. Para reducir el impacto del entorno sobre la aplicación y que se vea lo menos afectada posible, se han estudiado técnicas para la reducción de la luz que capta la *webcam* mientras estamos utilizando la aplicación.

Las cámaras son capaces de captar X fotogramas/frames por segundo. Cada *frame* es una matriz de ancho x alto, donde cada elemento de la matriz es un pixel con un valor para cada uno de sus tres canales de color. La velocidad en que la cámara es capaz de captar estos *frames* se conoce como FPS (*Frames Per Second*). Cuantos más fotogramas sea capaz de captar la cámara más fluido resultará el movimiento capturado.

Como ya se ha mencionado en el capítulo anterior, uno de los problemas que hay que resolver para realizar una detección robusta de los punteros láser es el de la saturación.

4.1 Reducir la cantidad de luz que entra en la webcam

La saturación

El sensor CCD es un dispositivo de carga acoplada (o en inglés *Charge-coupled device*, CCD) que tiene diminutas células fotoeléctricas que registran la imagen. La salida de cada sensor es aproximadamente lineal con la cantidad de luz que le llega.

La resolución de la imagen depende del número de células fotoeléctricas de este sensor. Este número se expresa en píxeles.

La saturación y la aparición de brillos son fenómenos que ocurren en todos los CCD y que afectan a las características cuantitativas y cualitativas de la imagen. Si imaginamos cada pixel de forma individual como un pozo de electrones, cuando este pozo se llena y no caben más electrones se produce la saturación. La cantidad de carga que se puede acumular en un solo píxel está determinada en gran medida por su área. Sin embargo, debido a la naturaleza del potencial del pozo hay menos probabilidad de atrapar un electrón dentro de un pozo que se está acercando a la saturación.

En la saturación, los píxeles pierden su capacidad para dar cabida a más carga adicional. Esta carga adicional luego se extiende a los píxeles vecinos, haciendo que se recojan valores erróneos o también saturados. Esta propagación de la carga a los píxeles adyacentes se conoce como *blooming* y aparece como una raya blanca o blob en la imagen. Debido a que diversos CCDs contienen diferentes arquitecturas, la saturación y el blooming pueden ser definidos y controlados de muchas maneras, pero es conveniente evitar que se produzca en primer lugar.

Una primera aproximación

Para implementar la captura y el procesamiento de las imágenes se utilizará la librería OpenCV [5][6]. Esta librería ofrece funciones para trabajar con las cámaras, aunque dependiendo de la cámara que se está utilizando o del sistema operativo algunas de estas funciones no son soportadas. Estas funciones nos permiten controlar algunos parámetros de la cámara como el ancho y el alto de la imagen, el contraste, la saturación, el brillo, el tiempo de exposición, etc.

Con la librería de OpenCV, capturar una serie de fotogramas con una cámara es tan sencillo como: primero crear un objeto VideoCapture y utilizar las funciones SET y GET que nos permiten obtener y modificar los parámetros de nuestra cámara. Por ejemplo, algunos de estos parámetros son: Modificar el ancho y el alto de la cámara, el brillo, el contraste, el tiempo de exposición, los FPS, etc. La figura 7 muestra un ejemplo.

```cpp
#include <stdio.h>
#include <opencv2/opencv.hpp> //incluimos la librería de opencv

using namespace cv;
using namespace std;

int main (int argc, const char * argv[]) {

    VideoCapture Camara; //creamos el objeto cámara

    Camara.open(0); //la abrimos

    //devuelve el brillo de la cámara
    Camara.get(CV_CAP_PROP_BRIGHTNESS);
    //asigna el valor 50 al brillo de la cámara
    Camara.set(CV_CAP_PROP_BRIGHTNESS, 50);


}
```

Figura 7. Set y get del parámetro de brillo de la cámara con OpenCv

No todas las cámaras ni sistemas operativos soportan todos los parámetros. En nuestro caso, alguna de las cámaras probadas sólo permitían utilizar estas funciones para cambiar el ancho y el alto de la imagen.

Una mejor práctica

Como se ha comentado es común que el láser aparezca en la imagen como un conjunto de pixeles saturados. Por ello, una primera aproximación para detectar los punteros consiste en localizar el punto más brillante de la imagen. Esto podría ser una solución si el entorno en el que se intenta localizar el láser tiene poca luz. Sin embargo, en una situación con mucha luz la cámara captará también reflejos o brillos los cuales también saturarán la imagen.

Los brillos son una molestia a la hora de detectar el puntero láser y sería deseable evitarlos todo lo posible. A continuación se describen algunas de las técnicas estudiadas.

4.1.1 Reducir el tiempo de exposición de la cámara.

Si se reduce el tiempo de exposición del sensor CCD a la luz, este captará menos luz y será más complicado que se sature (por lo tanto, las imágenes resultantes serán más oscuras). OpenCV tiene la siguiente función para ajustar el tiempo de exposición: *objetoCamara*.set(CV_CAP_PROP_EXPOSURE, *value*);

El rango de valores permitidos para el parámetro value de esta función es de 0 a 100. También se puede establecer el valor -1 para poner el valor por defecto de la cámara.

4.1.2 Reducir el tamaño del diafragma

El diafragma es un dispositivo que provee al objetivo la capacidad de regular la cantidad de luz que entra a la cámara. Suele ser un disco o sistema de aletas colocados en el objetivo de la cámara que limita la cantidad de luz que llega hacia el sensor. Generalmente suele ser ajustable. Las variaciones de tamaño del diafragma se denominan aperturas, y se especifican mediante el número f, que es la relación entre la longitud focal y el diámetro de abertura efectivo. Podemos ver un ejemplo en la figura 8.

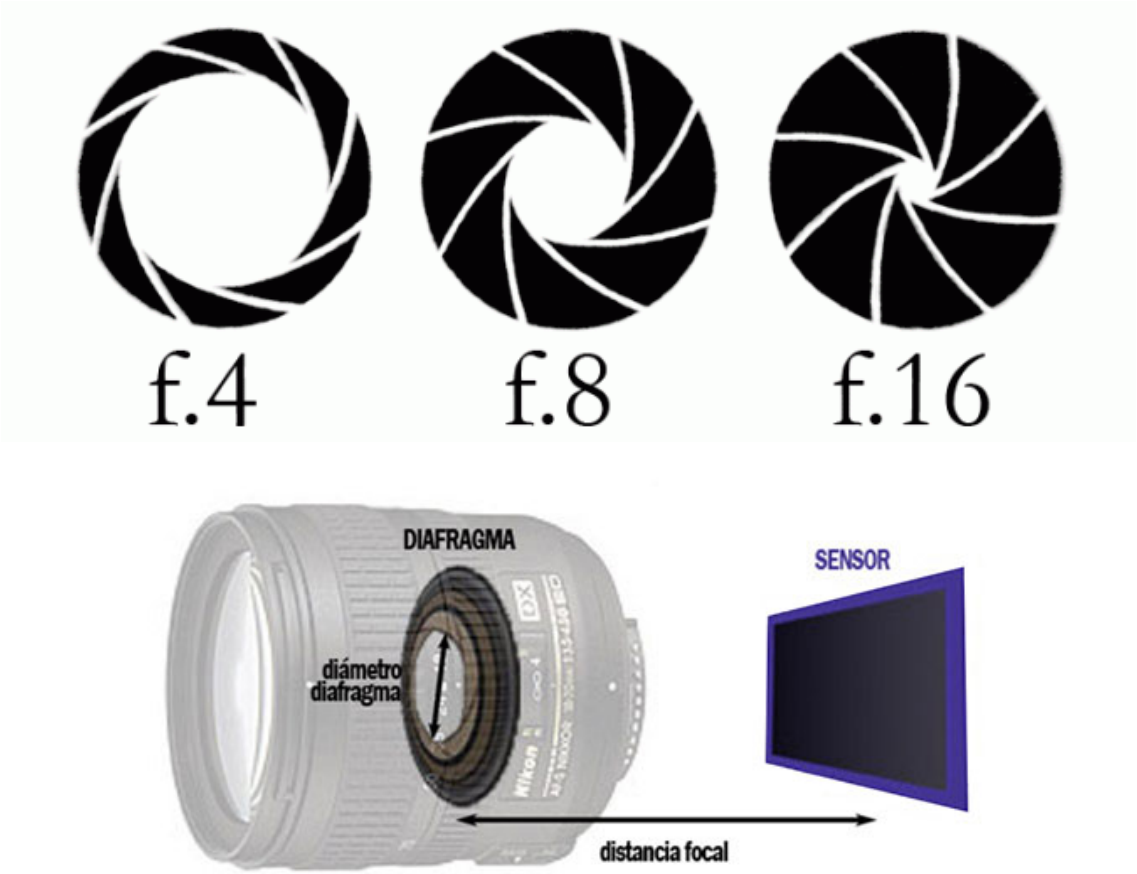


Figura 8. Tamaños de diafragma y diagrama de una cámara

Normalmente las cámaras usb de consumo (webcams), no permiten ajustar el tamaño del diafragma (de hecho, la mayoría ni tiene). Así que tampoco es una opción válida para nuestro proyecto, aunque es una técnica a tener en cuenta si se dispone de un mejor equipo.

4.1.3 Utilizar filtros

Los filtros son un medio que sólo permite el paso a través de él de luz con ciertas propiedades, suprimiendo o atenuando la luz restante.

Se pueden utilizar infinidad de filtros, pero para el caso que nos ocupa que es detectar varios láseres nos hemos decantado por los filtros de *densidad neutra*. Estos filtros son capaces de reducir la cantidad de luz que alcanza el objetivo. También se pueden llamar filtros neutrales puesto que no afectan a la calidad cromática de las imágenes, filtrando todos los colores por igual. El propósito de los filtros de densidad neutra o ND estándar es permitir al fotógrafo una gran flexibilidad para cambiar la apertura o el tiempo de exposición permitiendo más control, particularmente en circunstancias extremas.

Podemos utilizar un filtro de densidad neutra o podemos utilizar dos filtros polarizados. Un filtro polarizador es un material con transmitancia selectiva a una determinada dirección de oscilación del campo eléctrico de una onda electromagnética como la luz. Por lo general se trata de una película polimérica a base de Iodo estirada y emparedada entre dos vidrios. Estos últimos reducen la luz que los traspasa en cierta dirección. Se puede construir un filtro de densidad neutra con dos filtros polarizados [8]. Si colocamos dos filtros polarizados uno encima del otro podemos regular la cantidad de luz que entra en la cámara rotando uno de ellos. Así se consigue un filtro de densidad neutra regulable.

En las siguientes imágenes se muestra una de las primeras pruebas que se hicieron.

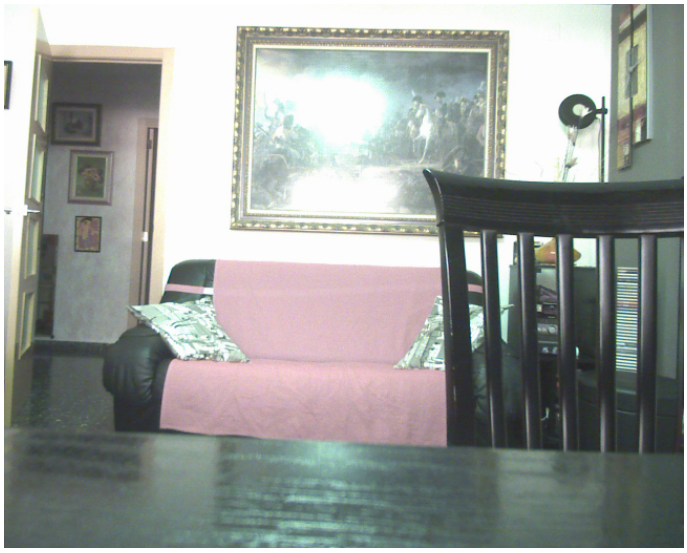


Figura 9. Imagen tomada con la webcam sin filtros. Resolución de 1280x1024. Hay entre 15000 y 17000 píxeles saturados

Ahora se coloca un filtro polarizado en la webcam.

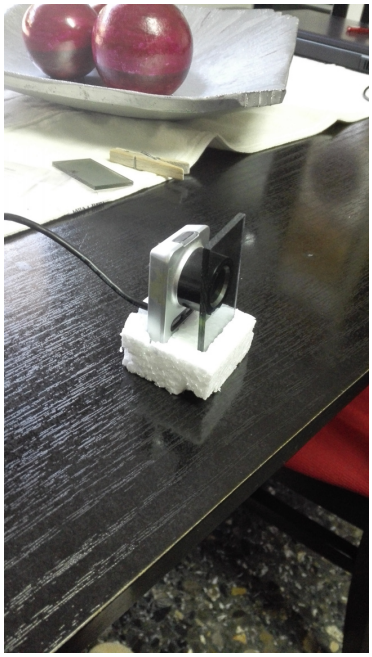


Figura 10. Webcam con un filtro polarizado delante de la lente

Y se vuelve a tomar la imagen calculando los pixeles saturados.

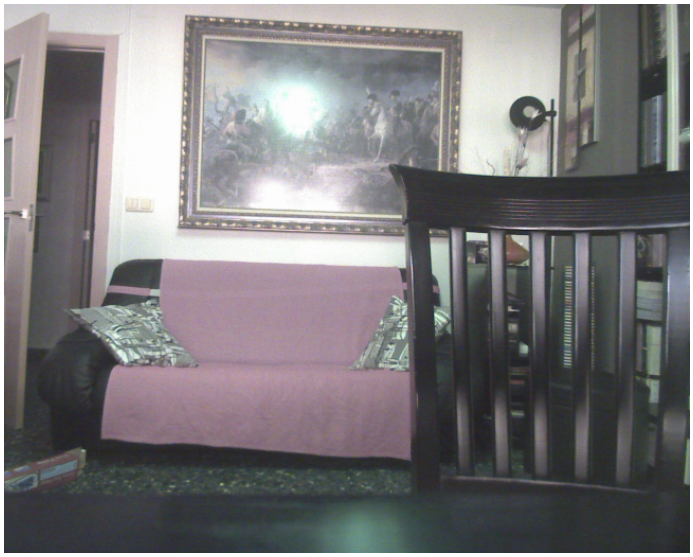


Figura 11. Imagen tomada con la webcam con un filtro polarizado. Hay entre 2000 y 3000 píxeles saturados

Como se puede apreciar, la imagen se vuelve más oscura y lo que es más importante, el número de pixeles saturados ha descendido.

Para este ejemplo en particular, la imagen tiene una resolución de 1280x1024, es decir, 1.310.720 pixeles. Sin utilizar filtros, habían 17.000 pixeles saturados, aproximadamente un 1,3% de la imagen. Con un solo filtro se ha reducido la saturación a 3.000 pixeles, aproximadamente un 0,23% del total.

Se pone el segundo filtro y se rota para reducir la luz que entra por la lente.

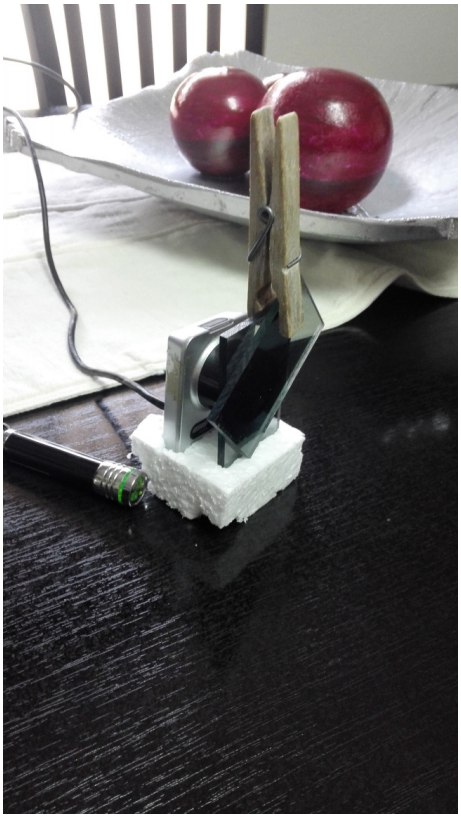


Figura 12. Webcam con 2 filtros polarizados, uno ligeramente rotado para conseguir un filtro de densidad neutra regulable

Veamos ahora qué ha pasado con la imagen.

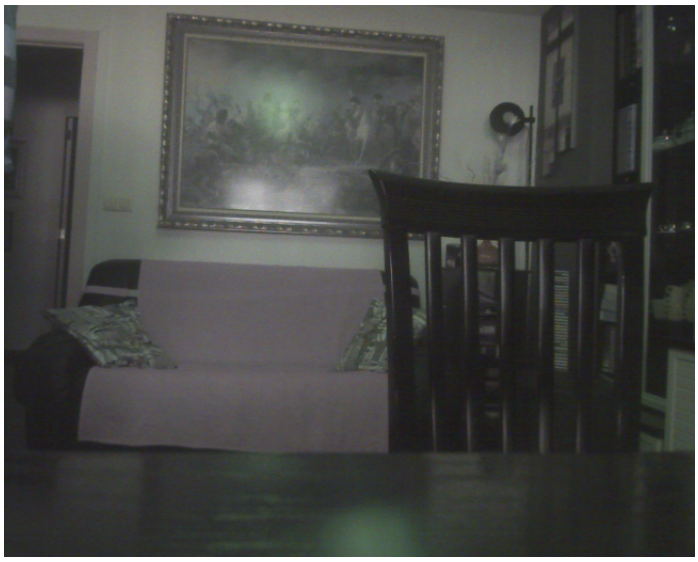


Figura 13. Imagen tomada con la webcam con dos filtros polarizados rotados. No hay píxeles saturados.

Con dos filtros se ha conseguido quitar toda la saturación de la imagen que captábamos con la webcam. Esto podría ser también problemático si hemos reducido demasiado la luz que entra por la lente, ya que no podríamos ver la luz del láser. En este caso, el láser se sigue apreciando como podemos ver en la figura 14.

Figura 14. Imagen tomada con la webcam y dos filtros polarizados rotados y apuntando con el láser

En toda esta discusión estamos asumiendo que el láser es la luz más intensa captada con la cámara. En caso de que la cámara esté captando una región más luminosa que el láser, se deberán utilizar otras técnicas para su detección, como se describe a continuación.

4.2 Detección y tracking de los láseres

Una vez reducida la cantidad de luz captada por la cámara, se procede a capturar las imágenes y a procesarlas una a una. La aplicación funciona a modo de bucle. Una vez iniciada, se van captando imágenes y se van a procesar una por una hasta que el usuario termine el procesamiento pulsando la tecla Esc. Para cada imagen capturada el procesado es el siguiente:

Binarización

Antes de poder binarizar una imagen hay que convertir la imagen original en una imagen en escala de grises, una vez conseguida esa imagen hay que convertirla en una imagen en blanco y negro. OpenCV tiene funciones para binarizar los valores de una imagen de grises utilizando un umbral. Dicho umbral es el valor que determina que todos los píxeles con valor mayores o iguales se convertirán a blanco, y todos los píxeles con valores menores a dicho umbral se convertirán en negro. Se puede ver un ejemplo en la figura 15.

Figura 15. Ejemplo de binarización

Pero, ¿cuál es el valor umbral idóneo? No se puede usar un umbral estático ya que el valor adecuado dependerá de la luz del entorno y por tanto del lugar donde se utilice la aplicación. Tampoco es viable cambiar el valor umbral a mano para cada entorno, ya que se deberían hacer pruebas para ver si todo funciona correctamente. Lo que se necesita es hacer una binarización adaptativa al entorno.

En nuestro caso se va a utilizar el histograma de la imagen (en escala de grises) que se quiere binarizar. El histograma va a revelar la cantidad de pixeles que hay en la imagen y va a decir cuántos de ellos tienen luminosidad 0, 1, 2, 3, ..., 255. Esta información se guardará en un vector de 256 posiciones donde cada posición representa la luminosidad del pixel y el contenido será la cantidad de pixeles que hay en la imagen con esa luminosidad.

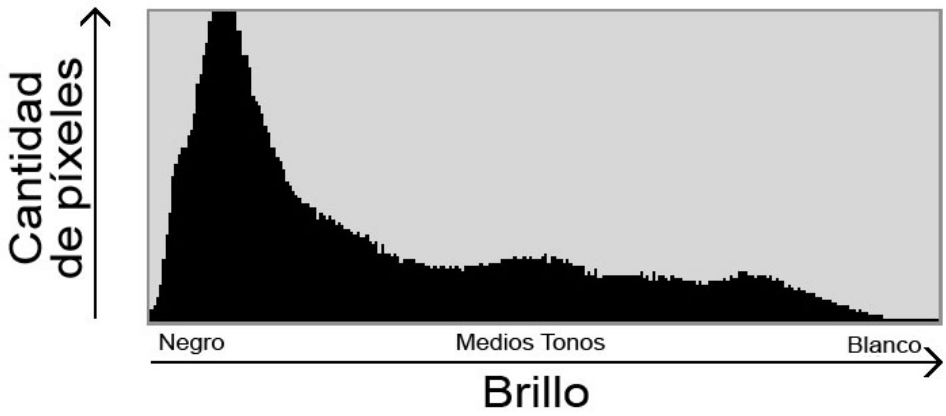


Figura 16. Ejemplo de histograma

Con el histograma se aprecia rápidamente cual es el valor de luminosidad más alto de la imagen y por lo tanto se puede definir el umbral de binarización un poco más bajo. Este histograma y proceso de binarización se puede estar haciendo constantemente para ver si hay cambios de luz bruscos en el entorno que puedan afectar al funcionamiento del proyecto y adaptar el la binarización automáticamente. También se puede adaptar el código, como se ha hecho en este caso, para que al pulsar la letra H se calcule el histograma. Así no se carga tanto el programa y solo lo se actualiza si el cambio de luz es muy grande y afecta al funcionamiento.

Más arriba, como se podía observar en la figura 15 derecha, el punto blanco que aparece es el del láser en la figura 15 izquierda. Este caso, es perfecto ya que el láser es el punto más brillante de la imagen, y no es ningún problema detectarlo. Pero, ¿qué pasaría si no se han podido eliminar los brillos y hay más regiones de la imagen saturadas? En ese caso hay que aplicar un proceso para decidir si un blob es un puntero láser o no teniendo en cuenta otras características del mismo, como la forma y el tamaño. Se puede ver una técnica para reconocimiento de blobs circulares más adelante en el Anexo B.

Eliminando falsos positivos

Incluso utilizando las técnicas de reducción de luz que entra por la cámara y utilizando la binarización adaptativa, algunas veces, en entornos con mucha luz, puede que hayan falsos positivos, como pueden ser los reflejos de las luces.

Para resolver este problema se va a recurrir a una técnica para captar el movimiento. Este trabajo se basa en la premisa de que todo objeto que esté estático no interesa, ya que aunque el usuario quisiera dejar el láser en una posición estática, siempre habrá un ligero movimiento. ¿Cómo se logra que los objetos estáticos no intervengan y así

no tenerlos en cuenta? haciendo una diferencia de imágenes. Así pues la técnica consiste en restarle a cada frame, el frame anterior. Esto generará una imagen en negro si la cámara no capta ningún cambio, es decir, no hay ningún objeto en movimiento (ver figura 17).

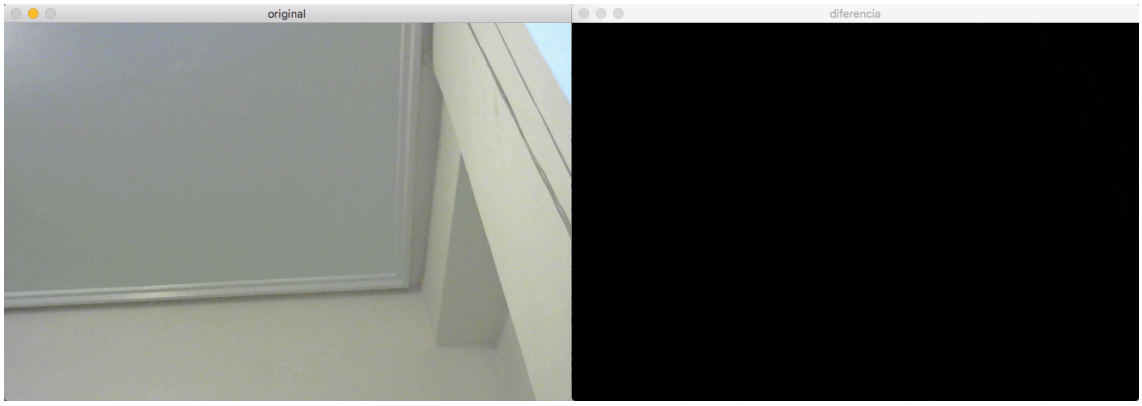


Figura 17. Diferencia de imágenes sin movimiento

Cuando la cámara captura un objeto en movimiento se obtiene la misma imagen en negro en aquellos píxeles estáticos, pero ahora, los píxeles donde ha cambiado la imagen (se ha movido algo) están en blanco. Ver figura 18.

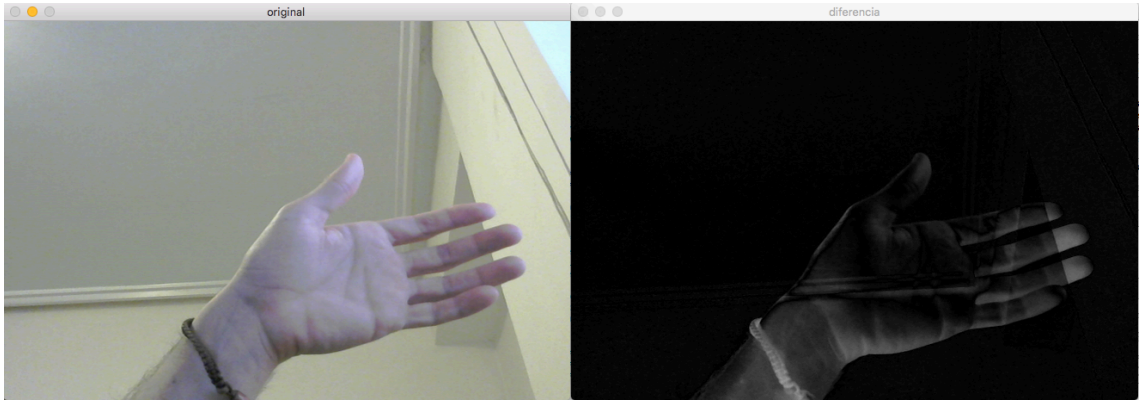


Figura 18. Diferencia de imágenes con movimiento

Se puede utilizar esta misma técnica para captar el movimiento del láser. Pero el láser es un punto muy pequeño y no se puede apreciar bien (ver figura 19). Esto también provoca un problema. ¿Qué sucede si el láser está en una posición estática? Sigue habiendo movimiento, pero a veces se pierde por su tamaño y es deseable que eso no suceda (ver figura 20). La solución pasa por utilizar un filtro de desenfoque para resaltar los objetos en movimiento (ver figura 21). Así se solucionan dos problemas: uno, la pérdida puntual del láser por ser pequeño y dos, cuando el láser está estático en la

imagen ahora se puede ver y localizar sin problemas ya que nadie tiene un pulso tan firme como para mantenerlo completamente inmóvil.

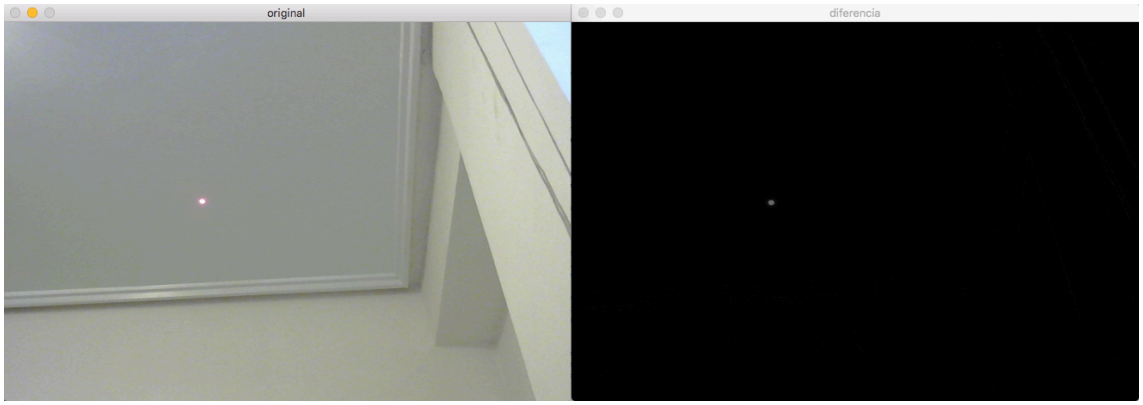


Figura 19. Diferencia de imágenes con el láser en movimiento

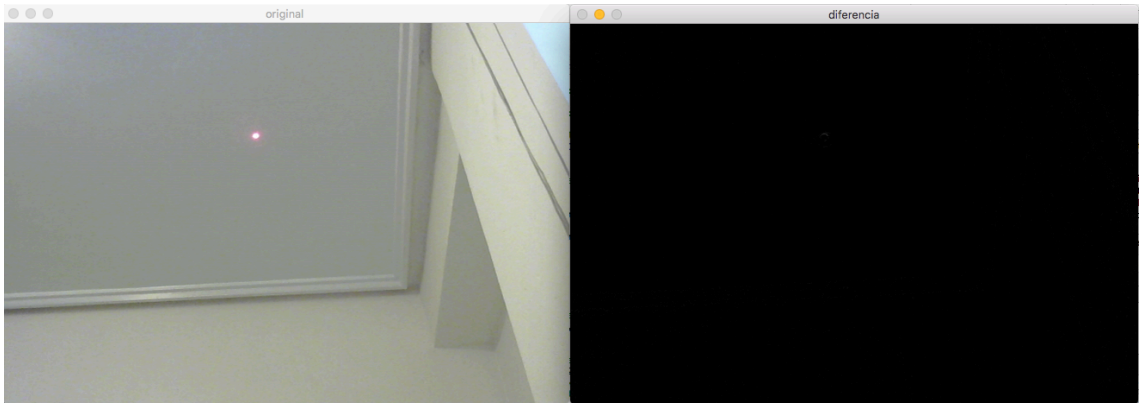


Figura 20. Diferencia de imágenes con el láser estático.

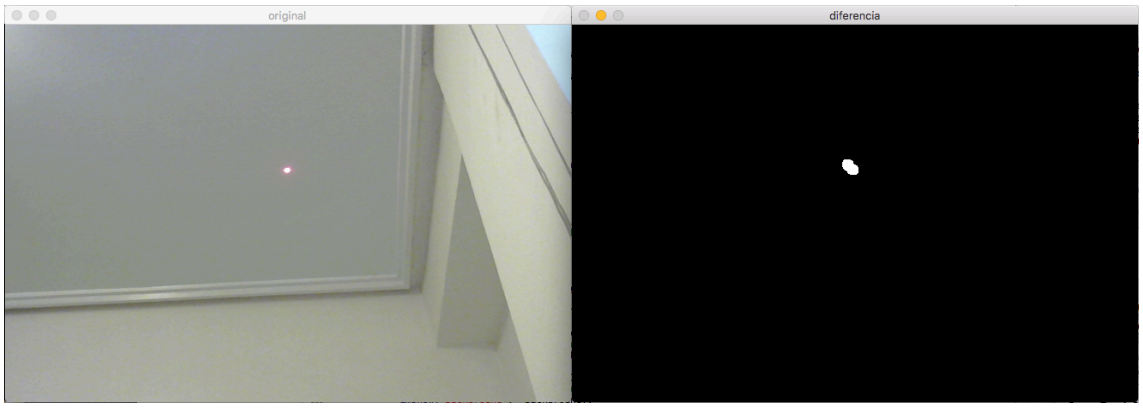


Figura 21. Diferencia de imágenes y el filtro de desenfoque con el láser en movimiento.

Con la ayuda de las técnicas que se han visto hasta el momento y con esta nueva para captar el movimiento ahora se puede localizar el láser con precisión. En el siguiente apartado se discutirá cómo localizar el puntero y obtener sus coordenadas (x, y).

Para reducir la cantidad de operaciones y conseguir una mayor fluidez en la ejecución lo que se ha hecho es que en primera instancia, al iniciar la aplicación, se captura el fondo (la primera imagen que capta la cámara) y los frames siguientes se van restando sobre el fondo. Una vez iniciada la aplicación se puede mover la cámara, colocarla en la posición que se quiera y volver a capturar el fondo con la tecla B. También se ha implementado una funcionalidad que aborta el tracking de los punteros cuando detecta más del número que se le especifique a la aplicación. Por ejemplo, si se especifica que como máximo habrá seis punteros, si se detectan siete la aplicación detendrá el tracking. Esta funcionalidad se activa cuando hay una situación anómala como una mala detección, un movimiento brusco de la cámara, etc.

Ahora se dispone una imagen binaria que tiene en blanco solo los movimientos de objetos brillantes. Esta imagen es la se va a utilizar para encontrar las coordenadas de los láseres y sacar sus características.

Procesado de la imagen binaria

Esta función encuentra los láseres y obtiene sus características como el color, el área y sus coordenadas.

Para obtener el color de cada objeto se calculan los contornos de los objetos en movimiento de la imagen binaria. Para cada objeto se obtiene uno a uno los píxeles que forman parte de él y se extrae de cada uno de ellos cual es el color predominante (rojo o verde). Una vez analizados los píxeles que forman parte del objeto solo hay que decir cual es el color que predomina para ese objeto.

El proceso de filtrado de píxeles por color es muy dependiente de la cantidad de luz del entorno. Es más fácil distinguir el color de los láseres en entornos oscuros que en entornos muy luminosos. Si estamos en un entorno con mucha luz, probablemente no se distinga con claridad el color del láser o la imagen esté tan sobreexpuesta que no se distinga ni el puntero. Pero para ello tenemos las técnicas que hemos comentando en el apartado 4.1 que nos ayudarán a reducir el impacto negativo que tiene la luz en nuestra aplicación.

Otra técnica que se puede utilizar para ayudar a detectar el color del láser es colocar un papel translúcido a modo de difusor en el cabezal del mismo con el fin de

desenfocar suavemente el puntero. Esto hace que el punto de luz sea de un área mayor y que el color sea más visible a la cámara. Se puede ver un láser con el filtro en la figura 22 y un ejemplo de uso en la figura 23.

Figura 22. Puntero láser con un trozo de papel transparente del mismo color

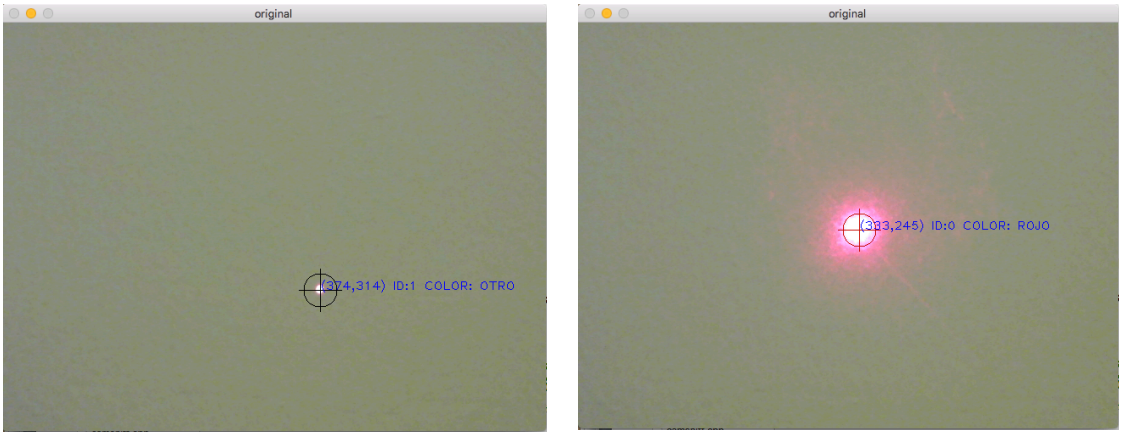


Figura 23. A la izquierda el puntero sin el difusor, a la derecha el puntero con el papel translúcido difusor.

Este proceso también ese el encargado de realizar el seguimiento o tracking de los punteros para obtener su historia, asignarles un identificador y saber si son blobs estáticos o no. Su funcionamiento es el siguiente:

El primer paso consiste extraer los blobs de la imagen y asignarles un identificador. Para ello, en extraen los contornos de la imagen binaria utilizando la función *findContours* de OpenCV.

Se ha establecido un límite que para el procesamiento de un frame en caso de que haya un número de blobs mayor que 6. Se ha puesto esta restricción, que parece bastante razonable, por si en el transcurso de la ejecución hubiera un movimiento brusco de la cámara (no debería) o un cambio importante en la iluminación del entorno. Funciona como una salida de emergencia cuando algo va mal al detectar el movimiento.

Cada objeto encontrado definirá un nuevo Blob, que tendrá un identificador asignado por orden de inserción. Cada vez que se procesa una imagen binaria para detectar los punteros, los objetos que se encuentran se guardan en un vector de blobs. Estos blobs se comparan con los blobs que se encontraron en el frame anterior. Esta comparación sirve para saber si alguno de los blobs que se han detectado en el frame actual se corresponde con alguno anterior. Para determinar si un blob es el mismo a otro blob se utilizan sus características, sobre todo sus coordenadas (x, y). Si se encuentra una coincidencia, se actualiza su identificador, para que siempre sea el mismo, y sus características. La función que se utiliza para hacer la comparación de los blobs es la misma que utiliza CCV [3] apartado 2.2. Es con esta misma función con la que comparamos los blobs con los objetos encontrados de color verde o rojo y decidimos de qué color son.

La función de tracking para decidir si un blob es igual a otro también marca los blobs que no son iguales a ningún otro con un identificador -1 para poder eliminarlos y así no tenerlos en cuenta a la hora de procesarlos.

Si la función encuentra un nuevo blob que no aparece en la historia se asume que es un blob nuevo y se añade al vector de punteros activos. Del mismo modo si se encuentra un blob que no se está moviendo (esta información la podemos sacar de sus coordenadas y la historia del blob) se marca como que no es un láser y tampoco se tiene en cuenta.

Para facilitar la depuración del sistema se ha implementado la posibilidad de mostrar la localización de los láseres pintando en la ventana original una mirilla del color del láser encima de los punteros encontrados. También muestra las coordenadas y su identificador.

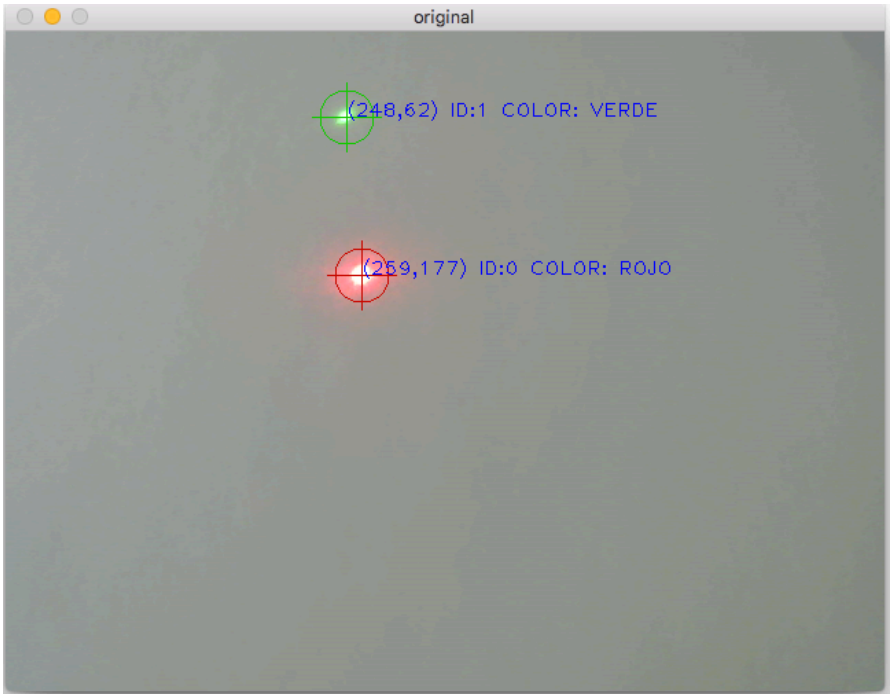


Figura 24. Muestra del resultado de la detección de los láseres rojo y verde con sus identificadores y sus coordenadas

4.3 Envío de mensajes OSC a través del protocolo TUIO

La clase TuioServer es el componente central del protocolo TUIO. Con el fin de codificar y enviar mensajes TUIO se debe crear una instancia TuioServer. Este objeto TuioServer es el encargado de enviar el mensaje a través de OSC mediante UDP a la dirección IP y el puerto que se ha configurado al crear el objeto:

```
TUIO::TuioServer *tuioServer;

tuioServer = new TUIO::TuioServer(ADDRESS, PORT);
```

Cada mensaje enviado debe estar creado siguiendo una norma. Primero se pasa el identificador del paquete, este será un identificador de sesión con el tiempo que ha transcurrido desde que se inició la sesión TUIO. Después se añaden, actualizan o se borran objetos, cursores, puntos, etc. y al final del mensaje se hace un *commit* de los cambios.

Un mensaje TUIO se construye de la siguiente forma:

```
server->initFrame(TuioTime::getSessionTime());

TuioObject *tobj = server->addTuioObject(xpos,ypos, angle);
```

```
TuioCursor *tcur = server->addTuioCursor(xpos,ypos);
server->commitFrame();
```

Cada blob que se ha determinado que se corresponde con un puntero láser creará un objeto TuioCursor. Del mismo modo cuando se actualicen los blobs (por ejemplo, al moverse de un frame a otro), también se actualizarán los objetos TuioCursor. Y al igual que se eliminan blobs que ya no existen, también se eliminarán los objetos TuioCursor cuando sea necesario. Se puede ver una tabla con algunas de las funciones básicas del TuioServer, además de una explicación más profunda en el anexo C. En la figura 25 se puede ver el resultado de enviar los mensajes por el protocolo TUIO de la figura 24 a una aplicación cliente instalada en un navegador web que muestra por pantalla un círculo por cada uno de los contactos detectados.

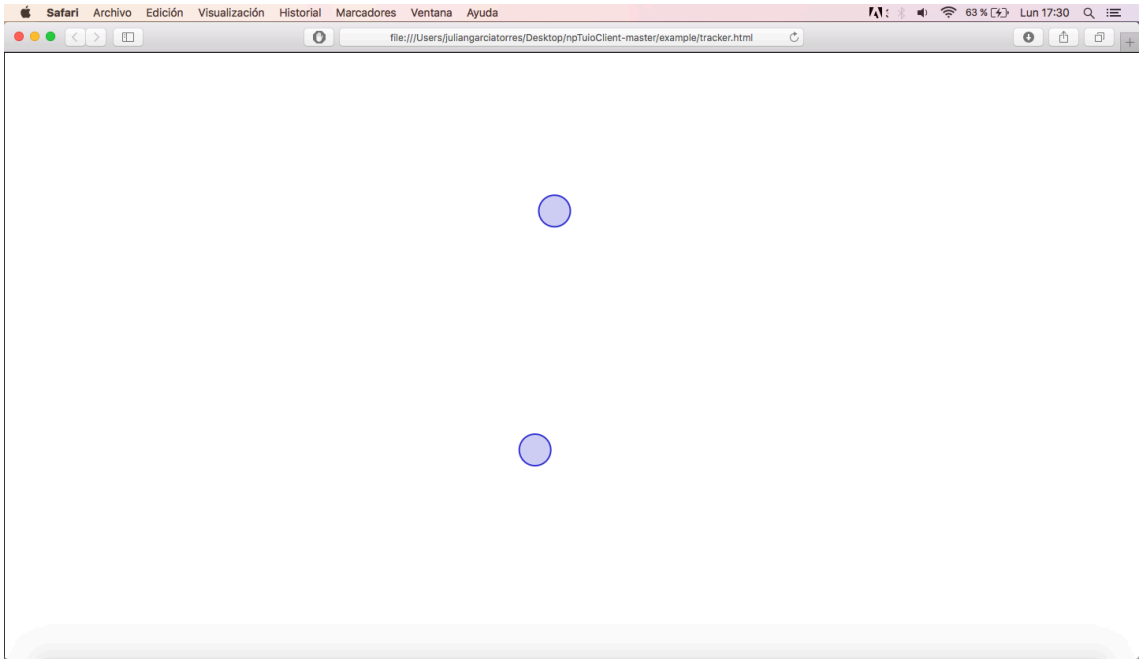


Figura 25. Muestra de que la aplicación envía mensajes que entienda una aplicación TUIO

4.4 Ajuste de coordenadas de la zona de trabajo

El sistema trabaja con varios sistemas de coordenadas distintos: el sistema de coordenadas de la cámara (definido por la resolución de las imágenes), el sistema de coordenadas de la zona de trabajo (normalmente una zona trapezoidal dentro de la imagen captada por la cámara), y el sistema de coordenadas normalizado, que va desde cero a uno en dirección vertical y horizontal, en el que trabajan las aplicaciones multitáctiles.

La aplicación cliente se proyecta en un rectángulo donde la esquina superior izquierda es la coordenada (0, 0) y la esquina inferior derecha es la coordenada (1, 1). Los ejes del sistema de coordenadas de la webcam tienen la misma dirección, pero las unidades son píxeles. Para el funcionamiento más óptimo las esquinas de la aplicación cliente y las de las imágenes que son capturadas con la webcam deberían coincidir. Si no coinciden estaremos teniendo un margen extra para utilizar la aplicación (en caso de que la visión de la cámara sea más amplia que el rectángulo proyectado) o un margen más reducido para utilizarla (en caso de que la visión de la cámara sea más pequeña que la proyección). La Figura 26 muestra varios ejemplos de posibles configuraciones:

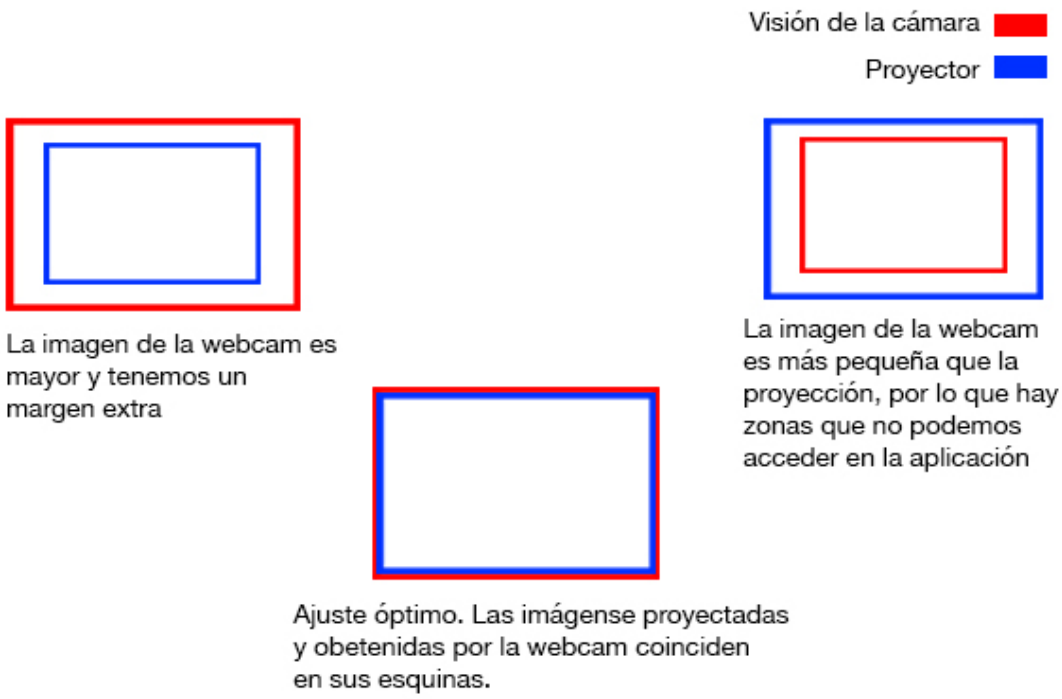


Figura 26. Muestra los diferentes tipos de ajuste que nos podemos encontrar al utilizar la aplicación

La peor situación ocurre cuando se acerca demasiado la cámara a la imagen proyectada, puesto que hay una zona del espacio de trabajo que no se puede capturar. Si se pudiera obtener un ajuste perfecto entre la zona de trabajo y la imagen capturada por la cámara, se conseguiría la mejor precisión en la captura de la posición de los punteros.

Como en muchos casos esto será imposible, se ha optado por implementar una homografía que cambia del sistema de coordenadas de la zona de trabajo al sistema de coordenadas normalizado. Una homografía es una transformación proyectiva que determina una correspondencia entre dos figuras geométricas planas, de forma que a cada uno de los puntos y las rectas de una de ellas le corresponden, respectivamente, un punto y una recta de la otra. Se ha implementado una función que permite marcar con el láser la posición de la coordenada (0, 0) o esquina superior izquierda, la posición de la coordenada (1,0) o esquina superior derecha, la posición de la coordenada (1, 1) o esquina inferior derecha y la posición de la coordenada (0,1) o esquina inferior izquierda. Así se puede enmarcar la aplicación y que las coordenadas sean exactas respecto a la aplicación proyectada. La figura 27 muestra un ejemplo de un marco creado con esta función.

Figura 27. Marco de ajuste para las coordenadas

En la figura 28 se puede ver una prueba realizada con la aplicación TuioSmoke y el ajuste de las coordenadas.

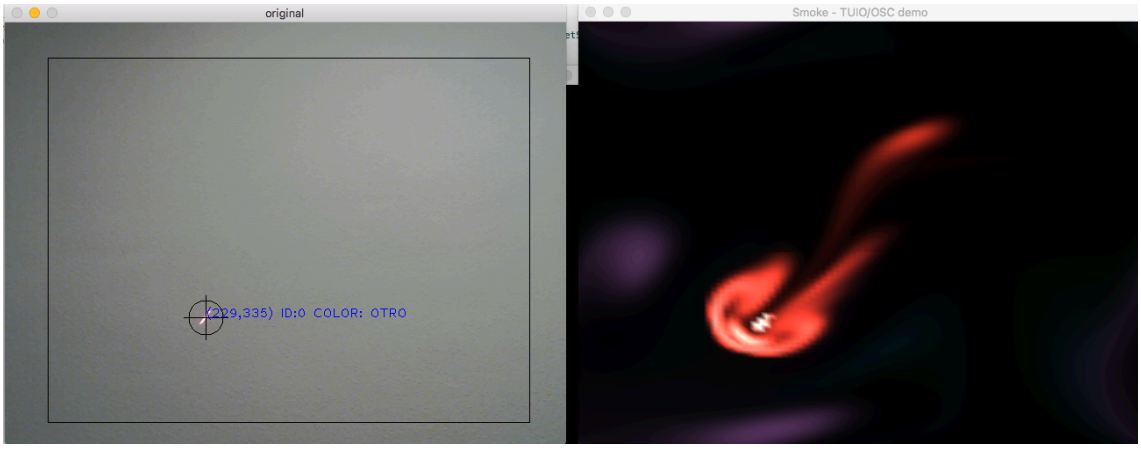


Figura 28. TuioSmoke con ajuste de coordenadas

Gracias a la homografía se pueden ajustar las coordenadas incluso si no tienen una forma rectangular. De este modo se puede corregir la distorsión perspectiva de la cámara que ocurre al no estar completamente perpendicular al proyector. En la figura 29 se puede ver que si el láser está fuera de los marcos que se han establecido para el uso de la aplicación, no tiene efecto en ella. Pero, si por el contrario está dentro del marco, la aplicación lo interpreta como un contacto. Ver figura 30.

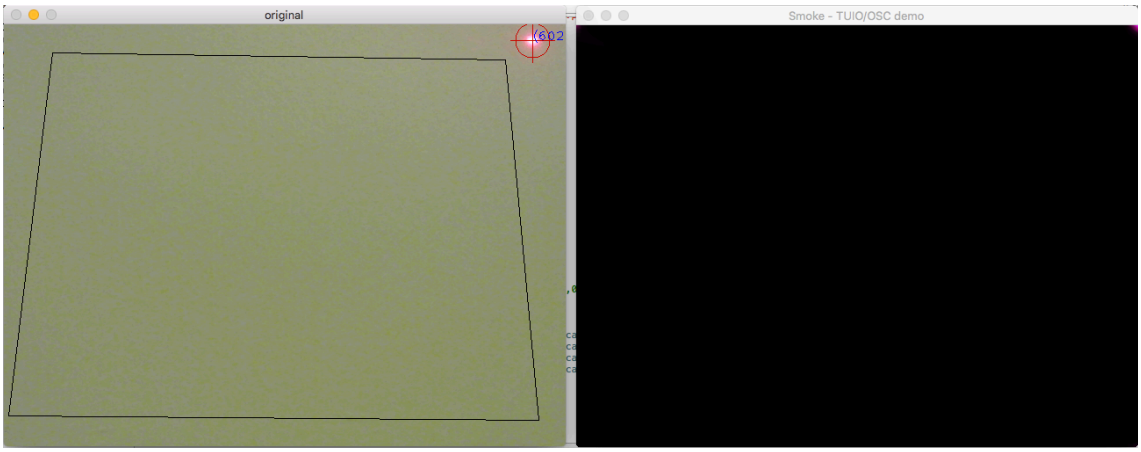


Figura 29. Muestra un puntero fuera del área que se ha seleccionado para utilizar la aplicación

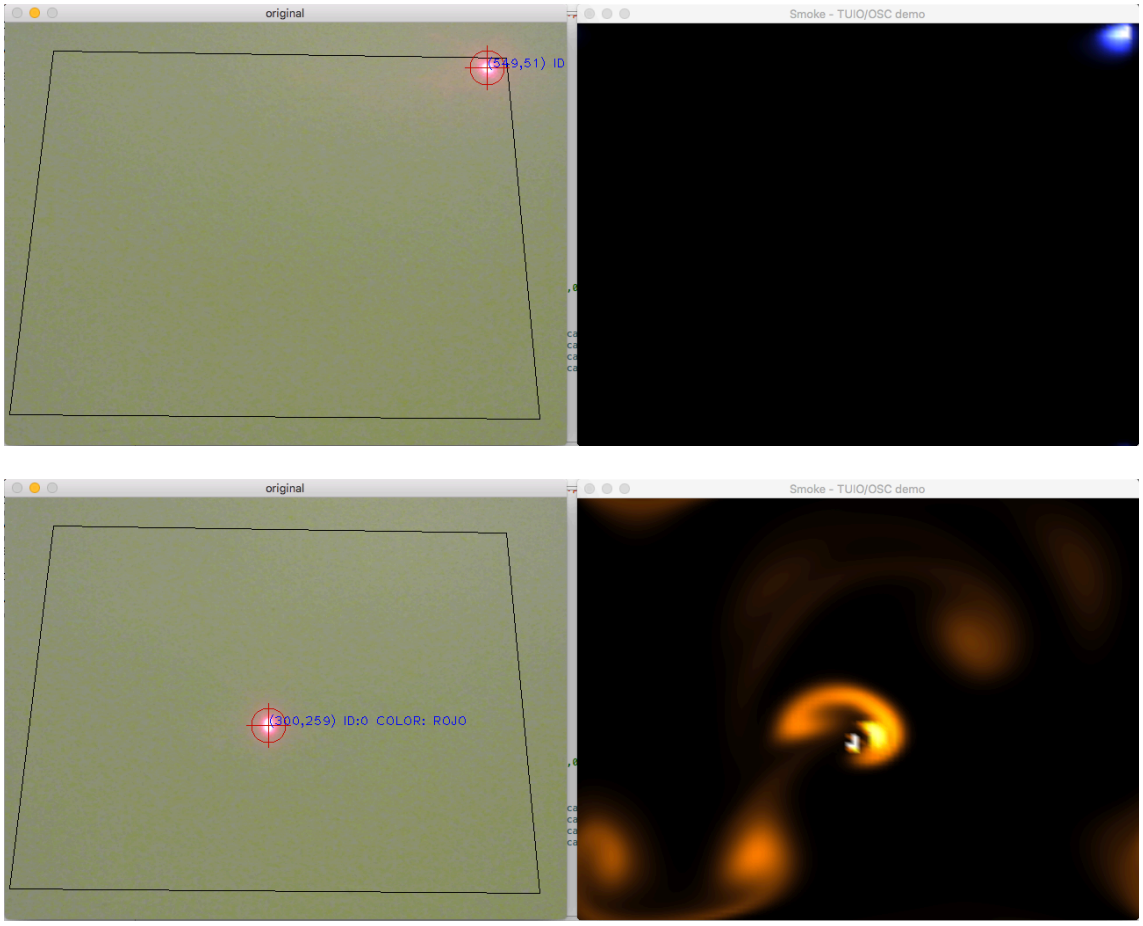


Figura 30. Puntero dentro del área habilitada para utilizar la aplicación

5. Resultados

Con el fin de comprobar que la aplicación funciona como se espera y se obtienen unos resultados fiables, en el apartado 5.1 se muestran unos ejemplos de ejecución paso a paso ilustrando con imágenes el proceso.

5.1 Ejemplos de ejecución

Para demostrar que el proyecto funciona se han escogido dos aplicaciones multitáctiles que funcionan con el protocolo TUIO. Al igual que se han escogido estas dos aplicaciones, se podría haber escogido cualquier aplicación multitáctil que funcione con el protocolo TUIO. La primera es una aplicación que se instala en el navegador y funciona como un tracker. Cada vez que se detecta un contacto se genera un círculo azul, que se mueve siguiendo el contacto hasta que se deja de detectar. La aplicación ya se ha visto antes en la figura 25. A continuación se muestra su ejecución utilizando el proyector.

Para empezar, se apunta la cámara al proyector, se selecciona el marco donde se utilizará la aplicación (redefinición de coordenadas), se ajusta el histograma y se empieza el tracking de los punteros. Más adelante en el apartado 7 (Manual de usuario) se explica con más detalle cómo utilizar la aplicación. En la imagen figura 31 se puede ver una captura de la aplicación haciendo el tracking del láser.

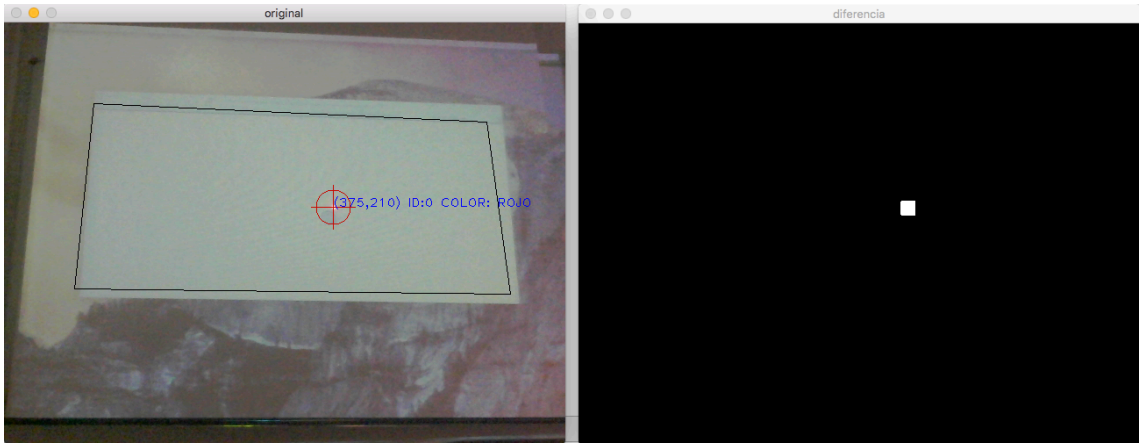


Figura 31. Aplicación que realiza el tracking de los punteros

En la figura 32 se puede ver el funcionamiento completo. Se puede ver el proyector, donde se está reproduciendo la aplicación cliente y se ve el ordenador, con la aplicación que se ha desarrollado a lo largo de la memoria, que es la encargada de localizar el puntero en el proyector y enviar los mensajes a través del protocolo TUIO a la aplicación cliente.

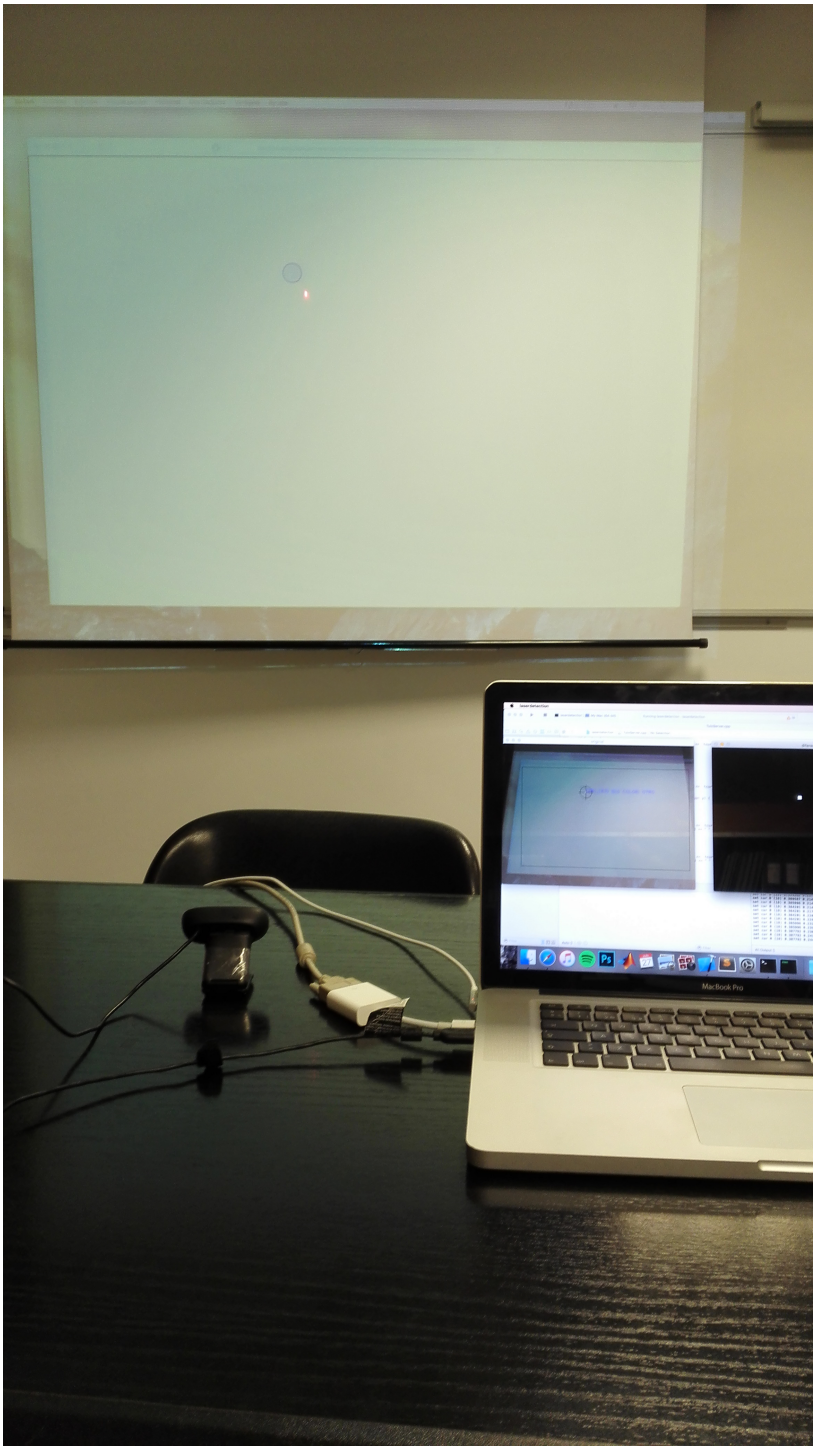


Figura 32. Funcionamiento de la aplicación ejecutándola junto a una aplicación de tracking cliente

La segunda aplicación que se ha escogido es un poco más visual, se trata de una aplicación que al contacto con la pantalla genera un "humo" de colores por cada pulsación, la aplicación se llama TuioSmoke. En la figura 33 se puede ver el tracking del puntero utilizando el ordenador.

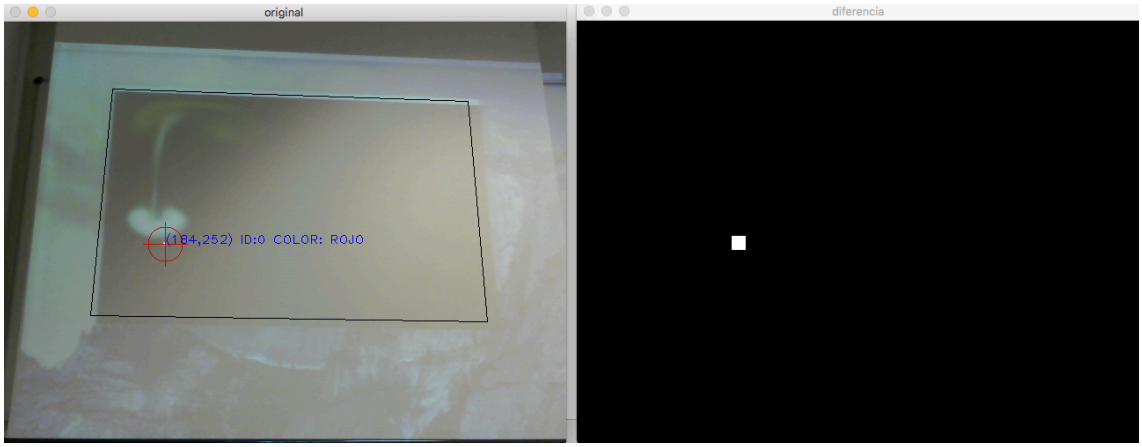


Figura 33. Tracking del puntero en la proyección de la aplicación TuioSmoke

A continuación en la figura 34 muestra la aplicación ejecutándose y el tracking en el ordenador.

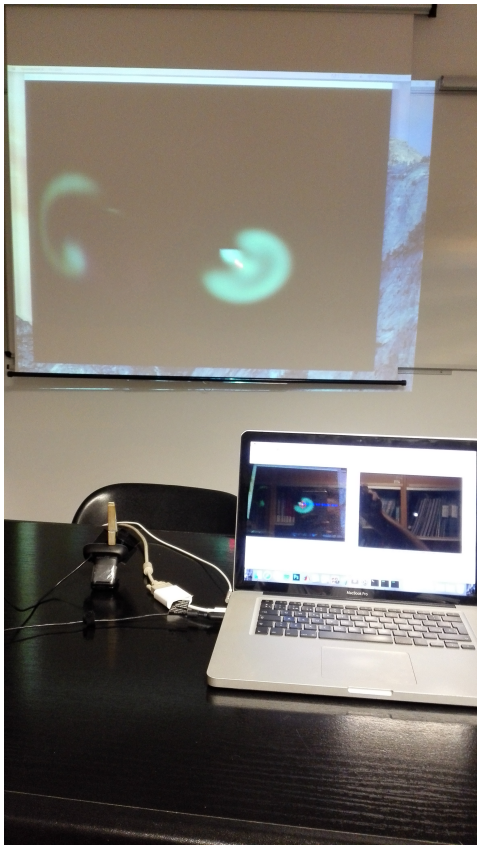


Figura 34. Ejecución de TuioSmoke en el proyector

5.2 Evaluación del sistema

Para evaluar la calidad de los resultados obtenidos se ha diseñado un banco de pruebas. La estrategia de las pruebas se basa en recopilar una serie de vídeos con distintas escenas en las que se podría utilizar el sistema final. Por ejemplo, se han hecho pruebas con proyectores en habitaciones con mucha luz, con proyectores y con las luces apagadas, sin proyector y con mucha luz, sin proyector y con poca luz, con fondos dinámicos, con fondos estáticos, utilizando filtros con las cámaras y sin filtros. También se han utilizado distintas cámaras. A continuación se describen varias de las escenas capturadas:

En una habitación iluminada, con
1. un fondo blanco,
2. un fondo negro,
3. la aplicación del tracker web (imagen 22), y
4. la aplicación TuioSmoke (imagen 24).

En la misma habitación, con las luces que estaban sobre el proyector apagadas y utilizando los filtros polarizados, y con
5. un fondo negro,
6. un fondo negro (láser rojo y verde),
7. la aplicación del tracker web,
8. la aplicación TuioSmoke, y
9. la aplicación TuioSmoke (láser rojo y verde).

En la misma habitación, con las luces que estaban sobre el proyector apagadas y sin utilizar los filtros polarizados.
10. sobre fondo negro, y
11. sobre fondo blanco.

Estas escenas reales se han capturado con varias cámaras, y se han almacenado los fotogramas del vídeo con un formato de compresión sin pérdidas.

Por otro lado, también se han diseñado tests sintéticos, en los que se han generado varias escenas en las que se ha simulado un puntero del color deseado, con un movimiento predefinido. Para cada escena el resultado se ha guardado en un archivo de texto en el cual cada línea nos indica el número de fotograma, la/s coordenadas del láser (según encuentre uno o varios) y el color. Es decir, tenemos un fichero de texto en el cual cada línea tiene el formato:

```
Frame 298: (0.964516, 0.717391)  color: ROJO (0.964516, 0.500000)
color: VERDE.
```

Se han realizado un total de cinco tests sintéticos. El primer test ha sido la simulación de un puntero verde en movimiento sobre un fondo negro (ver figura 35). El segundo simula un puntero rojo sobre un fondo negro (ver figura 36). El tercero y el cuarto

simulan 2 láseres verdes y dos láseres rojos respectivamente (ver figuras 37 y 38 respectivamente). En todas ellas las pruebas han salido perfectas un 100% de detección y un 100% de reconocimiento del color.

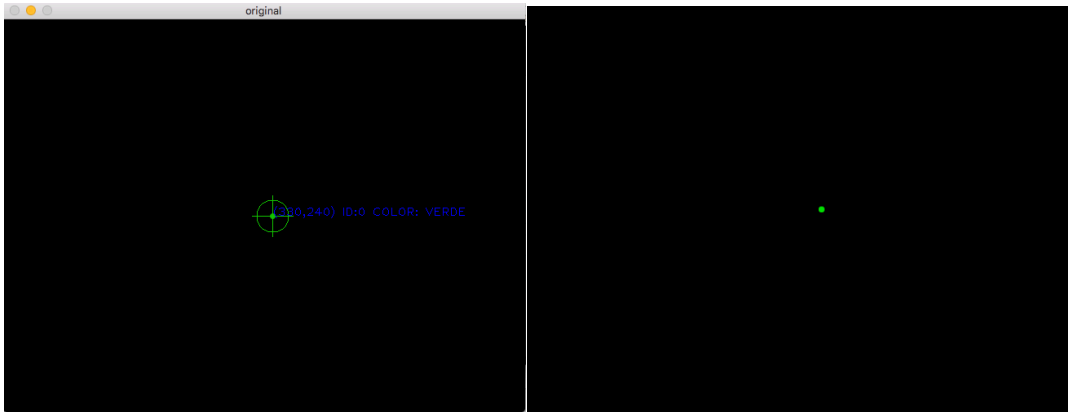


Figura 35. Test sintético 1 y su reconocimiento con el algoritmo

Figura 36. Test sintético 2 y su reconocimiento con el algoritmo

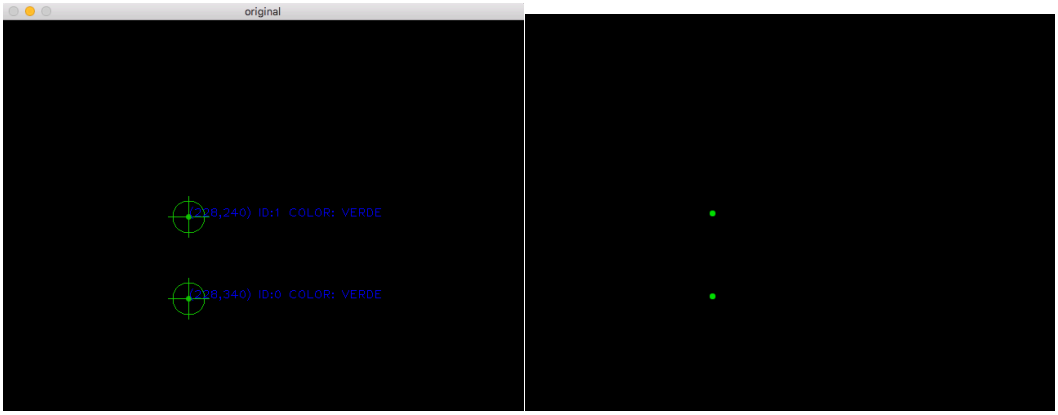


Figura 37. Test sintético 3 y su reconocimiento con el algoritmo

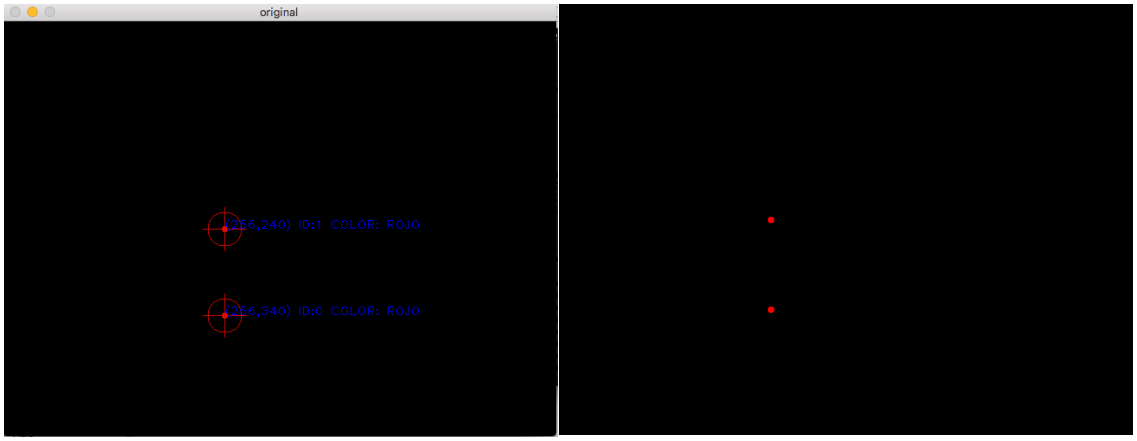


Figura 38. Test sintético 4 y su reconocimiento con el algoritmo

En estas cuatro pruebas sintéticas no ha habido ningún fallo. No se han creado más láseres de los que debería (uno o dos) y ha acertado siempre con el color de cada uno. Sin embargo, cuando hemos llegado a la quinta prueba, un láser de cada color, se ha detectado un error. Ver la figura 39.

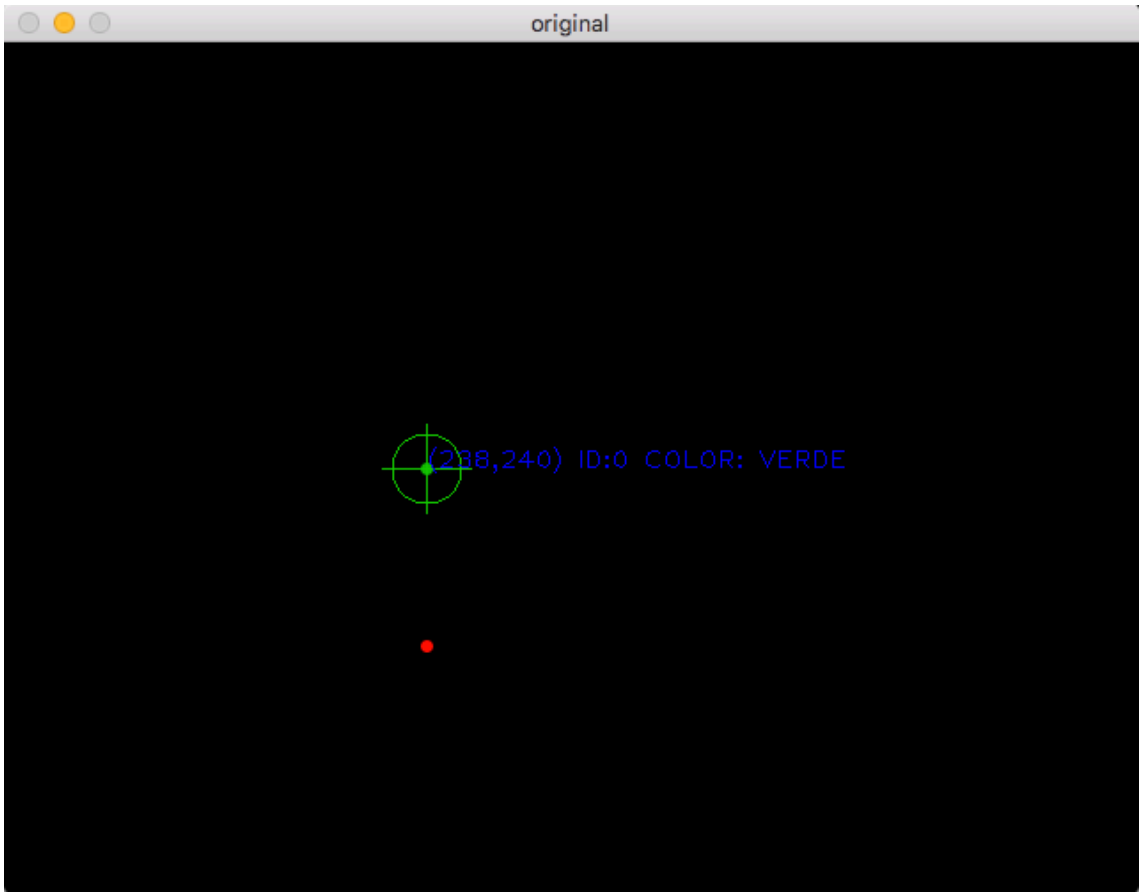


Figura 39. Quinta prueba sintética con error.

¿Qué ha sucedido en esta prueba? ¿Por qué solo detecta el puntero verde si en las anteriores ha detectado los rojos y los verdes sin problemas? La respuesta está en el color, más bien, en la intensidad lumínica del color. Como se ha comentado a lo largo de la memoria, antes de comenzar a utilizar la detección es muy importante calcular el histograma de la imagen. De esta forma sabemos cuál es el valor más brillante de la imagen, a ese valor se le resta un pequeño valor para solo obtener el movimiento de los elementos con brillo superior al calculado con el histograma - X. Pero en esta prueba se está calculando el histograma con los dos puntos proyectados, el verde y el rojo. El color verde es un color mucho más brillante que el rojo por lo que el rojo, al estar mucho más bajo que el verde, nunca llega a ser reconocido. ¿Cuál podría ser una solución? Aumentar el valor que se le resta al histograma o calcular el histograma cuando solo está el puntero rojo en pantalla. Ver figura 40.

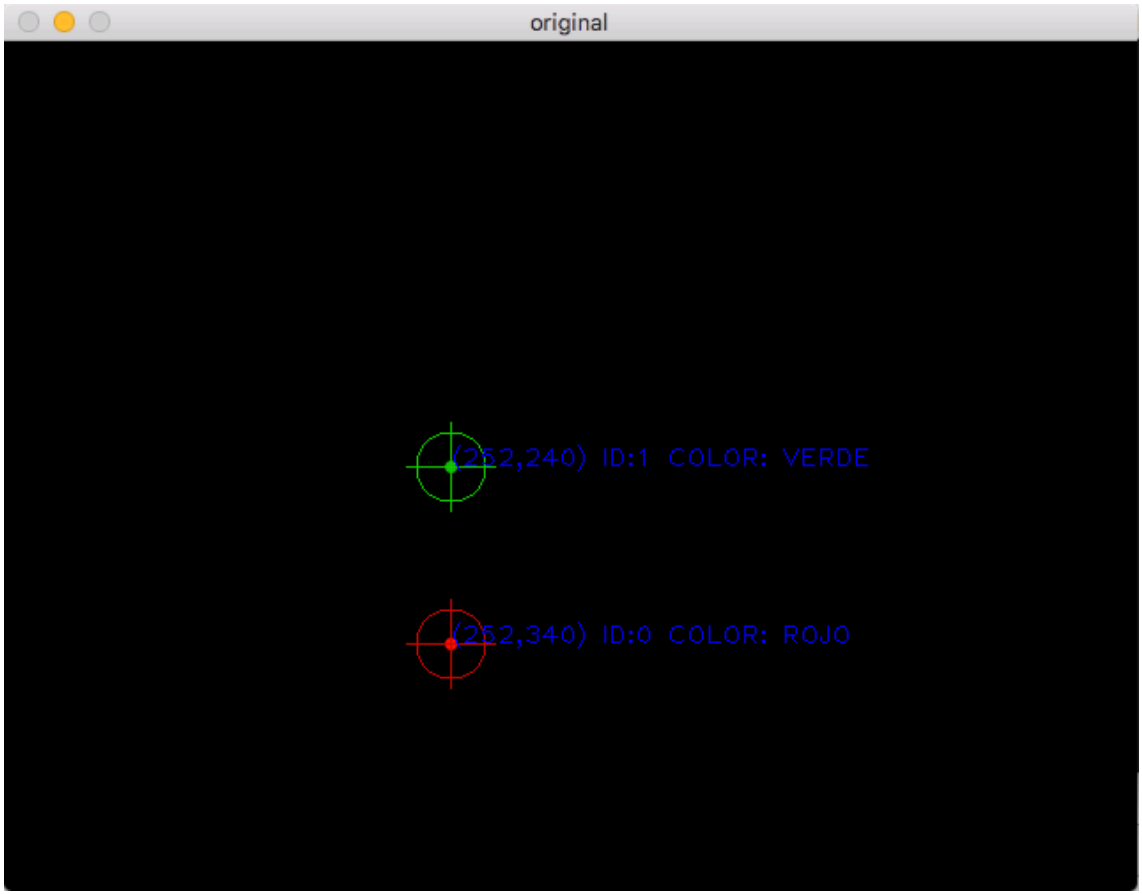


Figura 40. Quinta prueba sintética solucionada.

Después del ajuste, el algoritmo es capaz de reconocer los dos punteros y sus respectivos colores de una forma robusta. Un ejemplo real de la salida del fichero para esta última prueba sería:

[...]

```
Frame 29: (0.096774, 0.717391) color: ROJO (0.096774, 0.500000)  color: VERDE
Frame 30:  (0.100000, 0.717391)  color: ROJO (0.100000, 0.500000)  color: VERDE
Frame 31:  (0.103226, 0.717391)  color: ROJO (0.103226, 0.500000)  color: VERDE
Frame 32:  (0.106452, 0.717391)  color: ROJO (0.106452, 0.500000)  color: VERDE
Frame 33:  (0.109677, 0.717391)  color: ROJO (0.109677, 0.500000)  color: VERDE
Frame 34:  (0.112903, 0.717391)  color: ROJO (0.112903, 0.500000)  color: VERDE
Frame 35:  (0.116129, 0.717391)  color: ROJO (0.116129, 0.500000)  color: VERDE
```

[...]

La utilización del banco de pruebas nos ha permitido probar la evolución del algoritmo utilizando siempre la misma entrada, y hemos podido cuantificar el resultado de una forma fiable. Al ejecutar el algoritmo de detección sobre una escena de prueba se almacena en un fichero de texto las características de cada detección en cada frame (en especial el color y la posición). Dicho fichero de texto se puede comparar con los resultados esperados, que se han generado previamente anotando los vídeos de entrada, o automáticamente en caso de los tests sintéticos. El resultado de la ejecución de cada escena del banco de pruebas es una medida de calidad, que se explicará a continuación.

Si al reproducir la prueba se comprueba que la detección no obtiene los resultados esperados, se puede estudiar la causa de ese comportamiento con precisión, ajustar el algoritmo y volver a ejecutar el banco de pruebas para comprobar si los resultados mejoran.

Como se ha mencionado más arriba, en la prueba de cada escena se almacena tanto el color detectado del láser como su posición. Por ejemplo, si se sabe que en una escena solo hay un láser de color rojo, siempre que el algoritmo encuentre un puntero de otro color en esa escena se puede considerar un error. De esta forma se puede modificar la parte del algoritmo encargada de la detección del color y volver a ejecutar la misma prueba, comprobando si ha mejorado la detección del color.

La fiabilidad de la detección del puntero en cada prueba se ha hecho de la siguiente forma: Cada frame se identifica con su índice (0 hasta N-1, siendo N el número total de frames del vídeo). Se le pasa cada frame al algoritmo, que resultará en la detección de cero o más punteros en dicho frame. La salida del algoritmo será una serie de líneas, donde se indica el índice del frame analizado, las coordenadas del láser, y su color. Esta salida se almacena en un archivo de texto.

Para obtener el resultado esperado (correcto) en el caso de las escenas reales, ha sido necesario anotar las posiciones de los mismos en cada frame. Sin embargo, no se ha hecho de forma manual, sino que se ha utilizado la salida del propio algoritmo, comprobando visualmente que la salida se ajustaba al puntero en el vídeo.

A continuación se discutirán los resultados de varias pruebas.

Entorno iluminado sin filtros sobre fondo negro

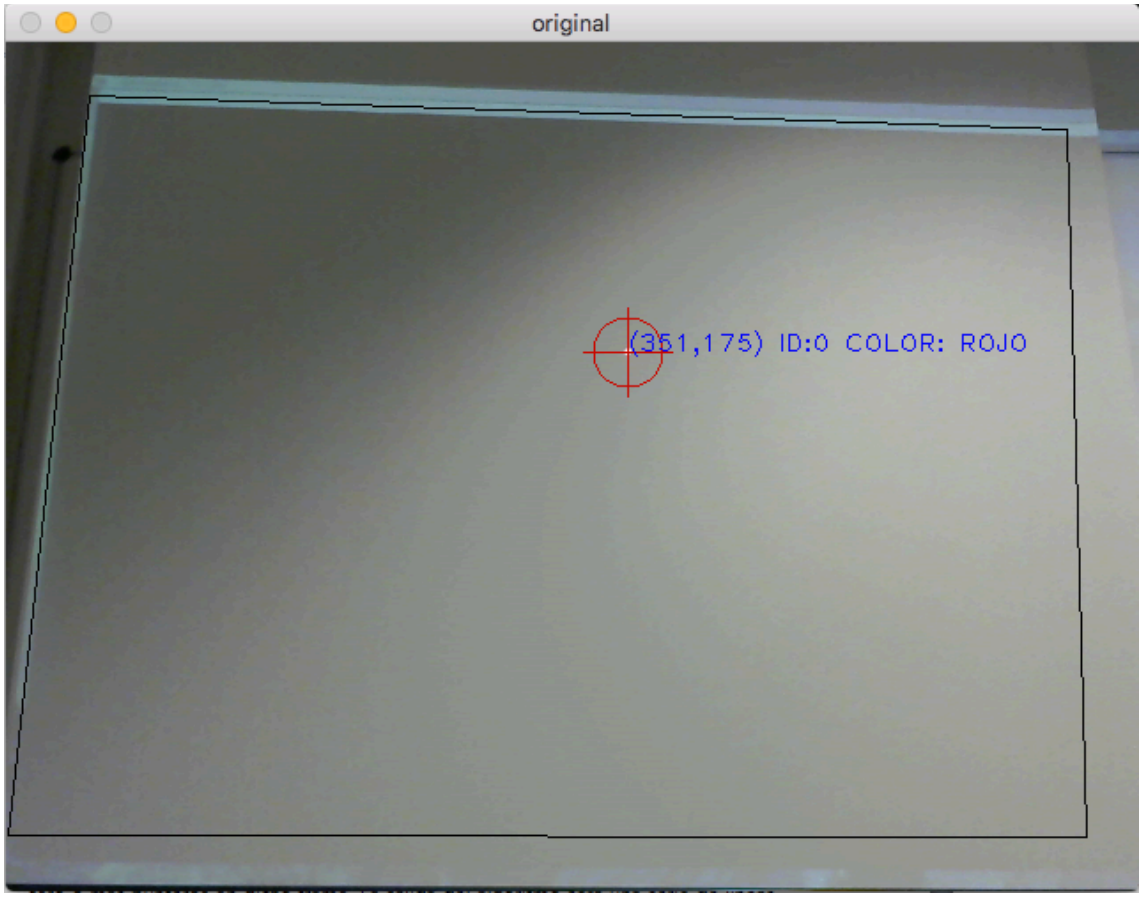


Figura 41. Entorno iluminado sin filtros sobre fondo negro

En esta prueba se han obtenido unos resultados de fiabilidad de la posición de láser de un 100%. Es decir, el fichero de texto de resultados ha devuelto el frame y la posición del láser en todos los frames en los que se encontraba el puntero. También, se ha obtenido una calidad de detección de color del 100%.

Como se ha comentado varias veces en el transcurso de esta memoria, la detección del puntero y el color son fuertemente dependientes del entorno. Para ello se han desarrollado técnicas que ayudan a resolver estos problemas, como por ejemplo la utilización de filtros para reducir la luz del entorno.

Esta prueba ha sido realizada con las siguientes características:

- Habitación iluminada

- Fondo negro sin movimiento

- Al valor máximo recibido por el ajuste del histograma se le ha restado un umbral de 15 (a partir de ahora nos referiremos a este valor como H : 15).

Entorno un poco más oscuro y con filtros polarizados

Esta prueba ha sido realizada en la misma habitación que la anterior, pero con la diferencia de que se han apagado solo las luces que caen directamente sobre el proyector. Además se han incorporados los filtros polarizados delante de la cámara.

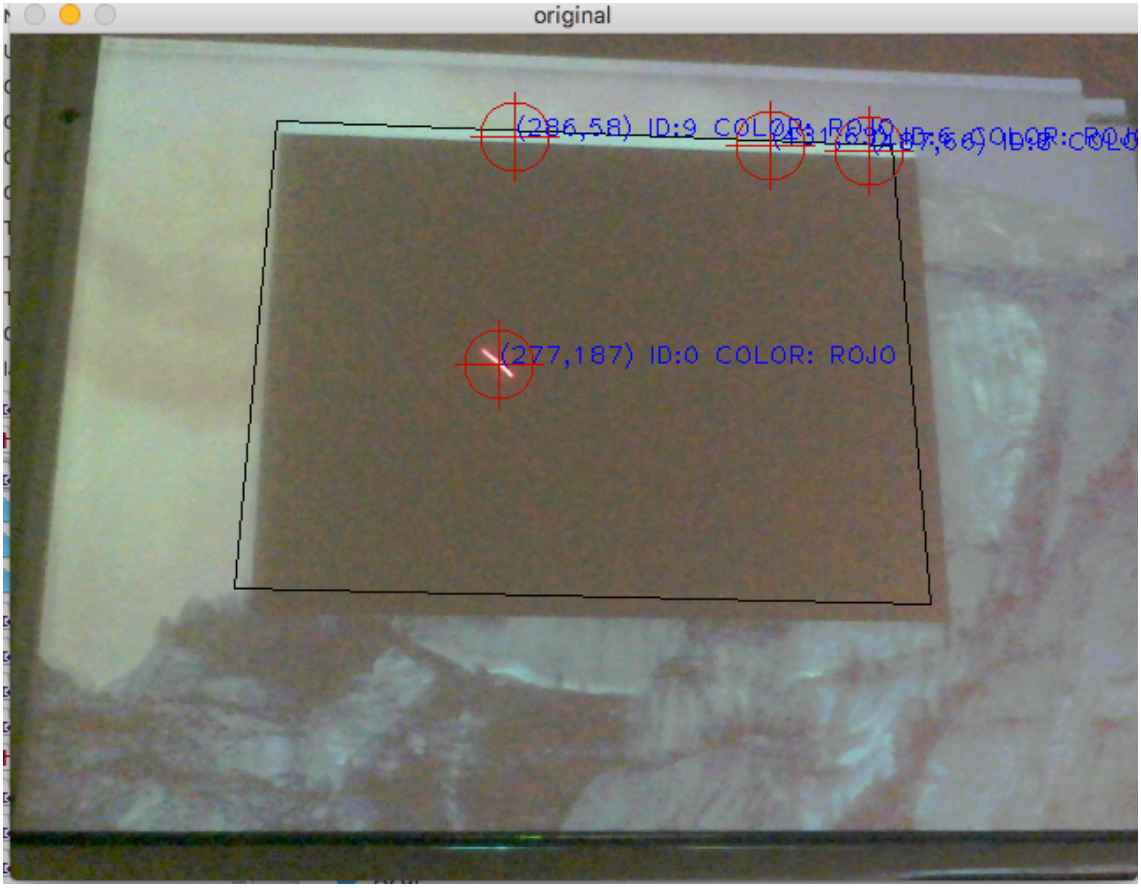


Figura 42. Prueba sobre fondo negro en un entorno no tan iluminado y con filtros

Como se puede apreciar en la imagen 33 hay errores de detección. Estos son debidos a un mal ajuste del umbral para el histograma. En este caso se ha vuelto a utilizar un valor H: 15, pero al ver los resultados directamente o comprobar el fichero de texto:

Frame 1: (0.652632, 0.398992) color: ROJO; (0.676724, 0.010605) color: ROJO;
Frame 2: (0.622353, 0.400449) color: ROJO; (0.766391, 0.009573) color: ROJO; (0.220225, 0.020815) color: ROJO; (0.401507, 0.014188) color: ROJO;
Frame 3: (0.603069, 0.405301) color: ROJO; (0.795302, 0.012172) color: ROJO; (0.367393, 0.016238) color: ROJO; (0.518594, 0.011454) color: ROJO;
Frame 4: (0.583789, 0.414059) color: ROJO; (0.518594, 0.011454) color: ROJO;
Frame 5: (0.578245, 0.422139) color: ROJO; (0.518594, 0.011454) color: ROJO; (0.867941, 0.012180) color: ROJO;
Frame 6: (0.569951, 0.434226) color: ROJO; (0.432841, 0.012305) color: ROJO; (0.766391, 0.009573) color: ROJO;

Se ve con claridad que hay un error grave de detección. Esto es debido a que, en este caso, hay zonas en la parte superior de la ventana que están dentro del umbral de brillo para reconocerlas como punteros. Esto se puede solucionar fácilmente reduciendo el H: 15 a un H: 10. De esta forma acortamos el umbral y no dejamos pasar objetos no tan brillantes como es el puntero láser. Es más, al reducir a H: 10 obtenemos un resultado perfecto, un 100% de detección de posición y un 100% de detección de color. Ver figura 43.

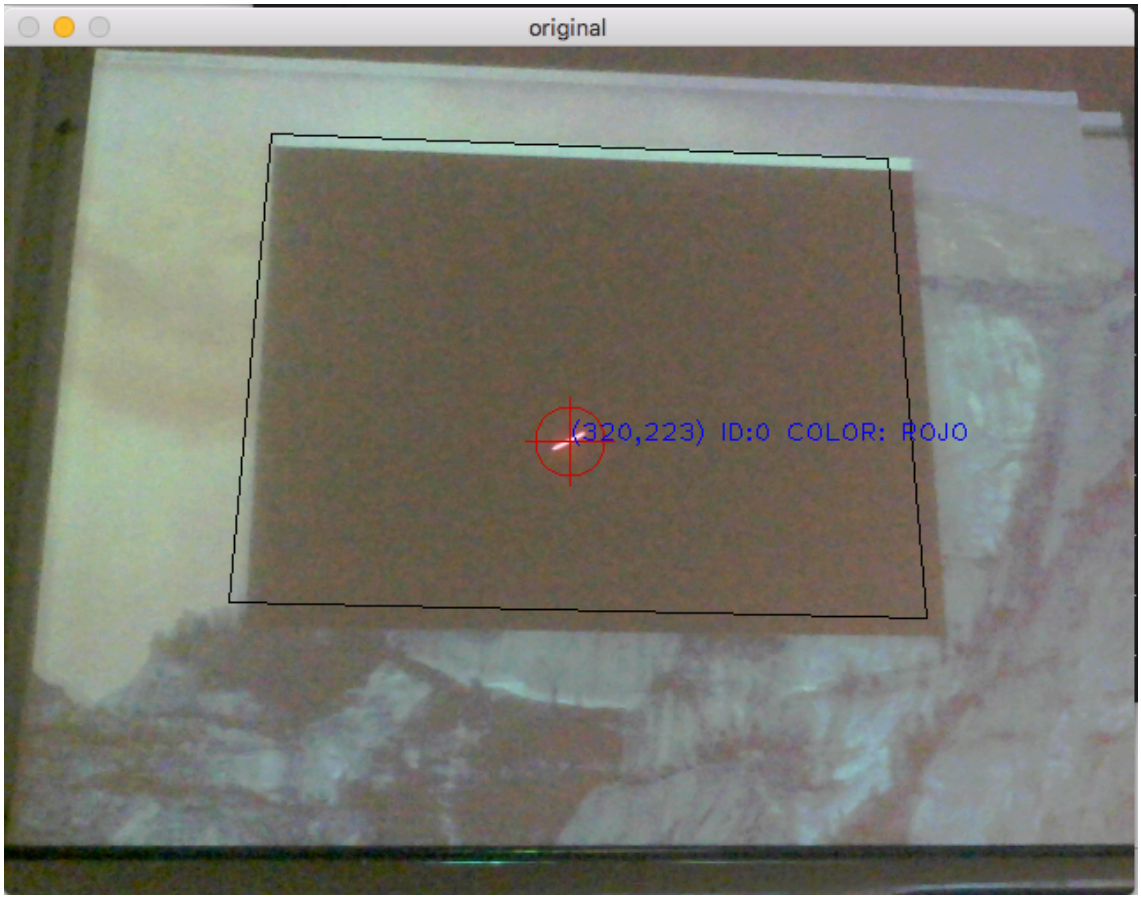


Figura 43. Pruebas con fondo negro, no tan iluminada y con filtro. Resultado perfecto.

Esta prueba ha sido realizada con las siguientes características:

- Habitación un poco menos iluminada.

- Fondo negro sin movimiento.

- H : 10.

Entorno iluminado sobre fondo blanco estático sin filtros

Durante el transcurso de esta prueba nos encontramos por primera vez con errores en la localización del puntero. Estos errores son debidos a que el puntero, según en qué zona del proyector se encuentre estará más o menos iluminado. La detección se ha hecho con un valor H : 15. La figura 44 muestra un ejemplo.

Figura 44. La imagen de la izquierda muestra el mismo puntero con menos intensidad que la imagen de la derecha.

Para solucionar este problema podemos aumentar el valor de H :15 a H : 40. Para este caso, es una solución factible, ya que no interfiere en el uso de la aplicación y no detecta punteros inexistentes. Se puede ver una comparativa de dos fragmentos de los ficheros de resultados para esta prueba con un valor H:15 y un valor H:40 en la figura 45.

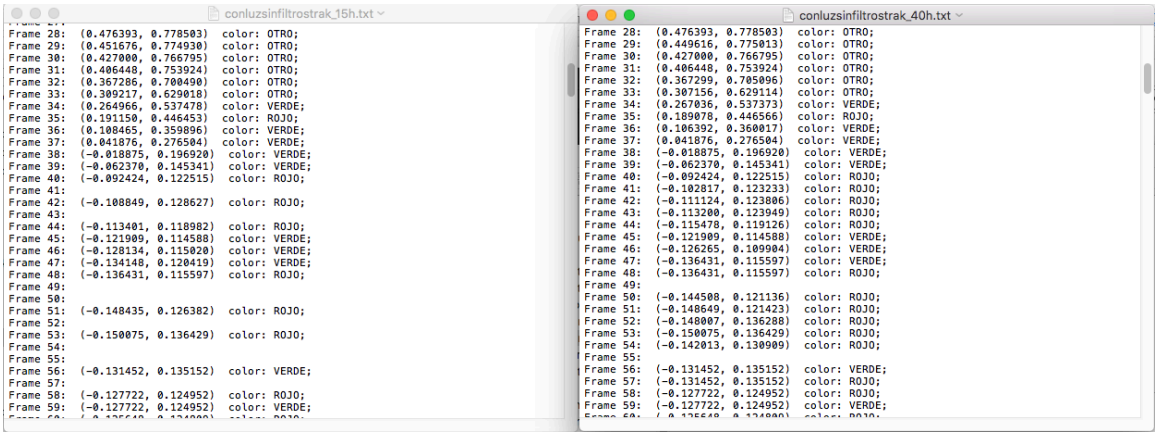


Figura 45. Comparativa de ficheros de resultados para la misma prueba con valor H: 15 y H: 40

Como se puede ver en la imagen anterior cuando hay pocos huecos seguidos de líneas vacías se interpretan como errores en la detección de las coordenadas del puntero, ya que en ninguna prueba se ha puesto un puntero parpadeante. También se puede apreciar que ha habido una mejora al aumentar el valor del umbral de H. En esta prueba solo ha habido un 76% de fiabilidad de color.

Esta prueba ha sido realizada con las siguientes características:

- Habitación iluminada.

- Fondo blanco si movimiento.

- H : 10 y H: 40.

Entorno iluminado, utilizando la aplicación TuioSmoke sin filtros

Al ejecutar esta prueba en H : 15 también se han detectado errores de localización por el mismo caso que en las pruebas anteriores: umbral pequeño para la detección de ese puntero en esa aplicación. Por lo que se ha subido el valor a H : 30. Y los resultados han mejorado. Ver figura 46.

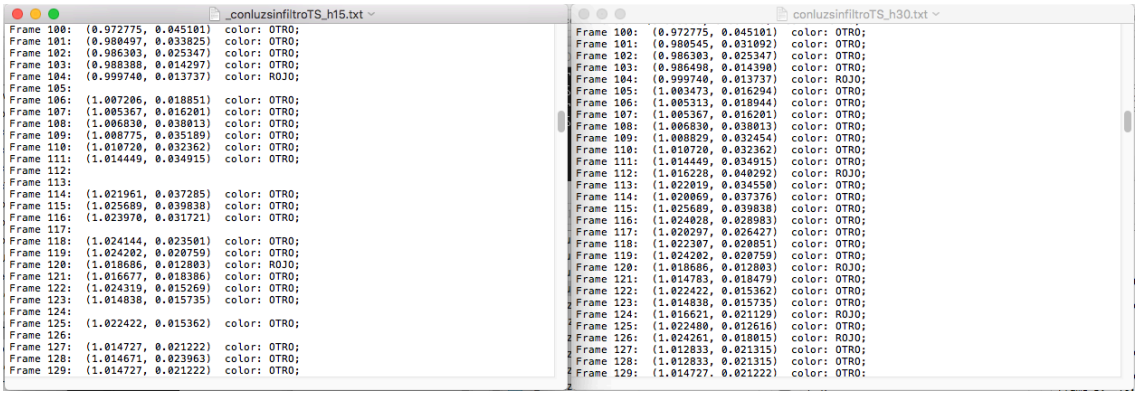


Figura 46. Comparativa entre la prueba con H: 15 y H: 30

Con este ajuste se ha mejorado la calidad de la detección pero aún no ha llegado a ser perfecta (100%) por lo que aumentamos el valor de H:30 a H : 40. Al hacer esto, y ver el fichero de salida comprobamos que han aparecido nuevos errores pero esta vez por exceso, es decir, se han detectado más punteros de los que realmente hay. Se puede comprobar en las líneas del archivo de salida siguientes:

Frame 406: (0.863006, 0.492257) color: OTRO; (0.924038, 0.340926) color: VERDE;
Frame 407: (0.864622, 0.517103) color: OTRO; (0.917776, 0.407611) color: VERDE;
Frame 408: (0.864553, 0.527027) color: VERDE; (0.930100, 0.435125) color: VERDE;

Está detectando 2 punteros cuando en realidad solo hay uno. Esto quiere decir se ha aumentado demasiado el umbral H. Ahora lo ajustamos a H : 35. El resultado para la detección es perfecto ahora, un 100%. Mostramos las mismas líneas del archivo para H : 35:

Frame 406: (0.863006, 0.492257) color: OTRO;
Frame 407: (0.864622, 0.517103) color: OTRO;
Frame 408: (0.864553, 0.527027) color: VERDE;

Como se puede comprobar en las líneas del fichero anteriores, ahora solo detecta el puntero que debería.

Sin embargo para esta prueba se ha obtenido un 89% en la detección del color. Este 89% se debe a que el color del humo de la aplicación se mezcla a veces con el color del puntero haciendo que sea una tarea más complicada detectar el color del puntero. Se puede ver una captura de la prueba en la figura 47.

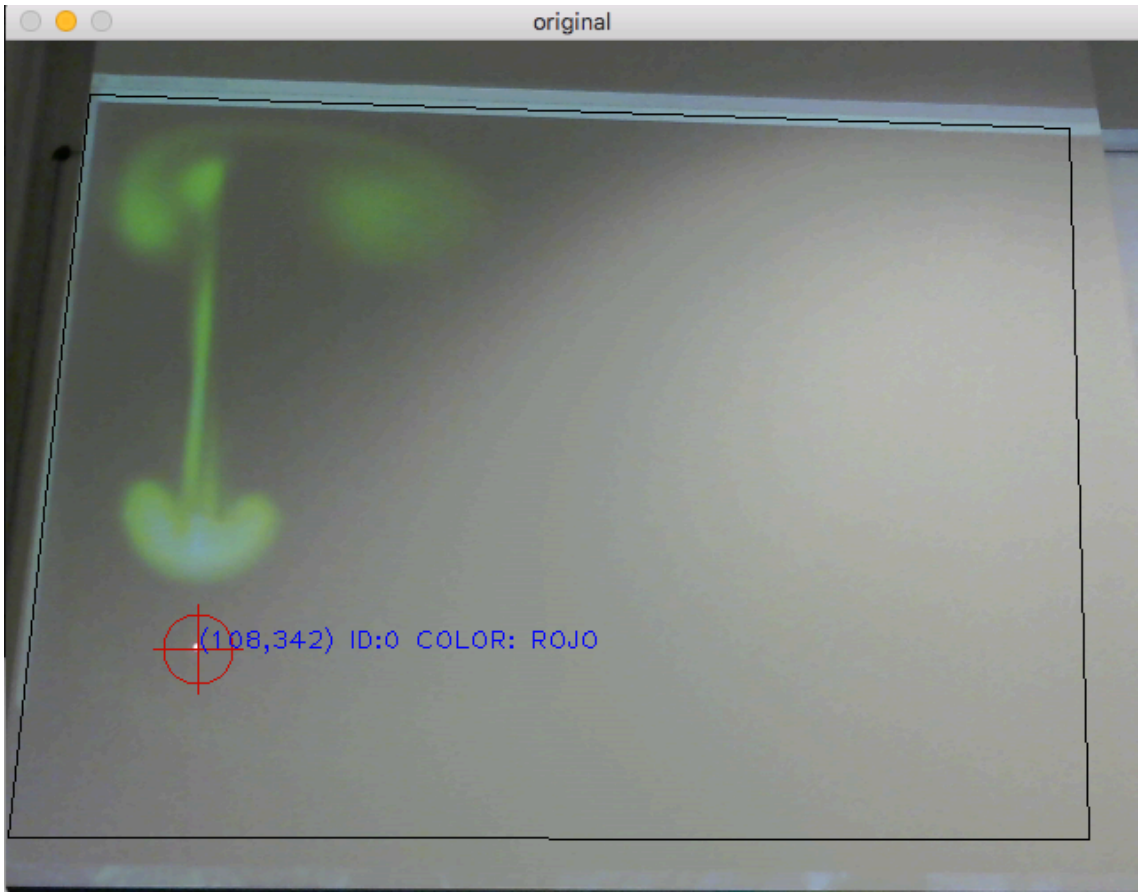


Figura 47. Prueba TuioSmoke en un entorno iluminado y sin filtros

Tracker web en un entorno un poco más oscuro y con filtros polarizados

La primera conclusión que obtenemos al ver el fichero de salida para esta prueba es que no ha habido una detección de coordenadas del 100%. También se ha obtenido un 100% de detección del color del puntero. Como se ha hecho en las pruebas anteriores se ha ido incrementando el valor de H hasta conseguir un resultado óptimo. En este caso se ha probado con valores de 15, 20, 25, 30 y 35. Siendo este último valor el que ha conseguido un 100% de detección de las coordenadas. En la figura 49 se pueden ver los mismos fragmentos del archivo de resultados para los diferentes valores de H.

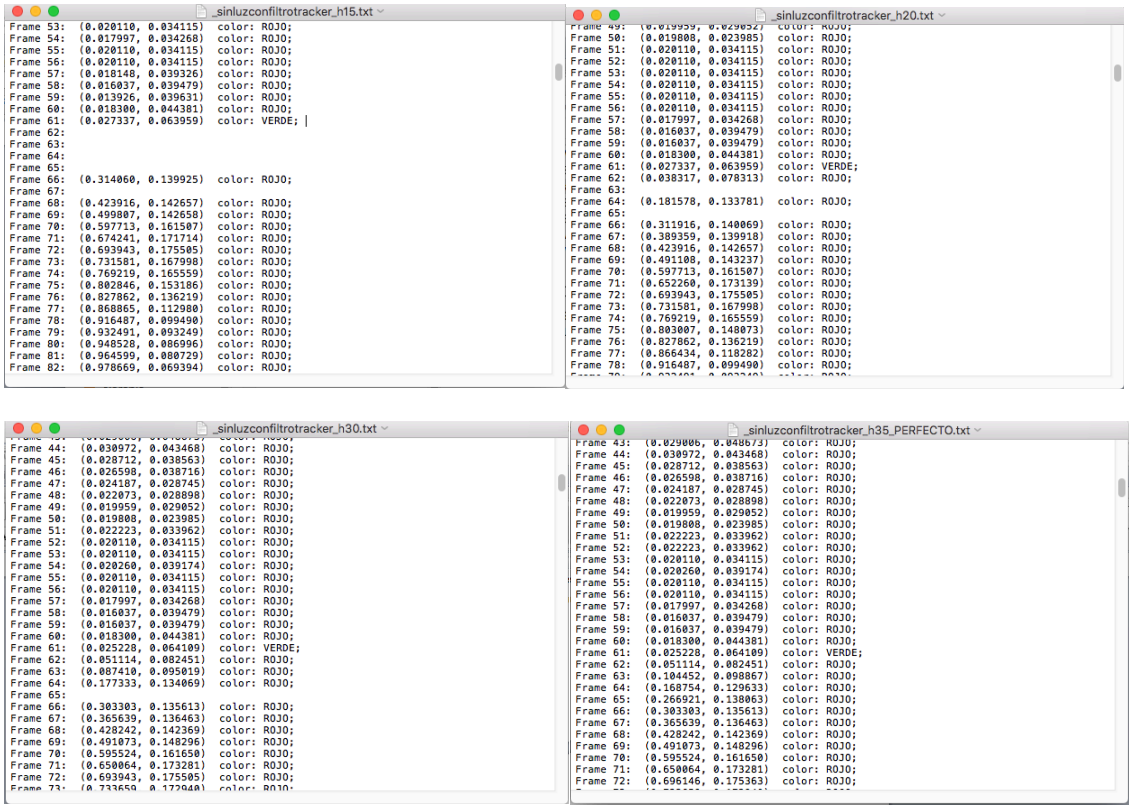


Figura 48. Mismo archivo de resultados para distintos valores de H. de arriba a abajo en sentido de las agujas del reloj. H: 15, 20, 30, 35.

En la figura 49 se muestra un frame donde se ve que un movimiento veloz del puntero hace que pierda luminosidad y gane en anchura. Este tipo de frames son los que son más susceptibles a pérdida de localización. De todos modos para esta prueba hemos conseguido detectar el puntero para todos los frames obteniendo un 100% de detección de las coordenadas. Dependiendo del entorno, del láser, de la cámara, del

fondo, de la aplicación y de la distancia este tipo de errores podrán ser más fácilmente corregibles o no.

En la figura 50, se muestra la ejecución de la aplicación en H : 35.

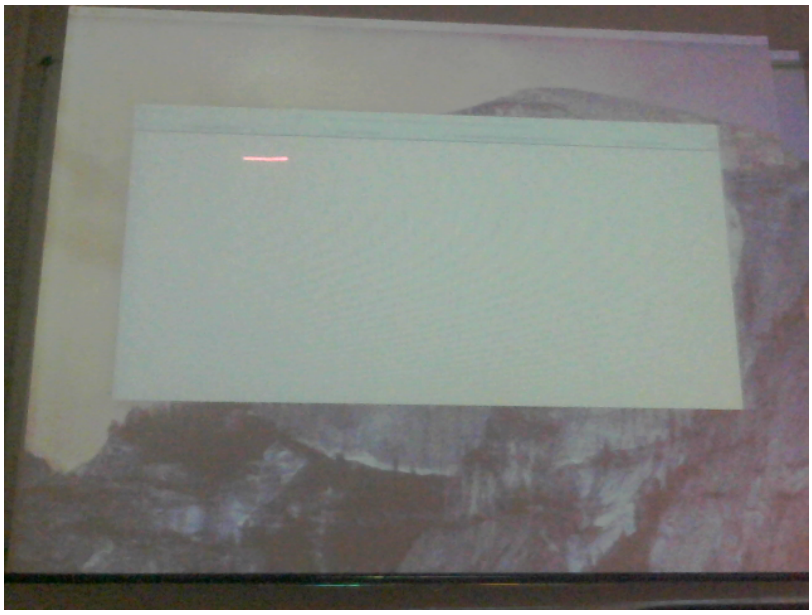


Figura 49. Frame susceptible a pérdida.

Figura 50. Captura de pantalla de la prueba con la aplicación tracker.

TuioSmoke en un entorno un poco más oscuro y con filtros polarizados

Como se ha visto en las dos pruebas anteriores en un entorno un poco más oscuro y con filtros la detección de color suele tener unos valores más altos de fiabilidad. En este caso se ha obtenido un 99% de fiabilidad de la detección de color. Pero al observar en los ficheros de salida de resultados también se pueden ver huecos aislados que representan fallos en la detección de posición de los punteros. Como se ha hecho en las pruebas anteriores hemos aumentado el valor de H : 15 a H 30 y así se ha conseguido una detección total de los punteros en cada frame. En la figura 51 se puede ver una comparación de los ficheros de salida. Si se observa las dos imágenes se puede ver que no hay mejora, los frames en los que no se captaba el puntero en H 15 se sigue sin captar en H 20. En H 25 ya hay una ligera mejora y en H 30 se consigue un resultado casi perfecto en cuanto a detección de posición. En cuanto a detección de color se ha logrado un 100%.

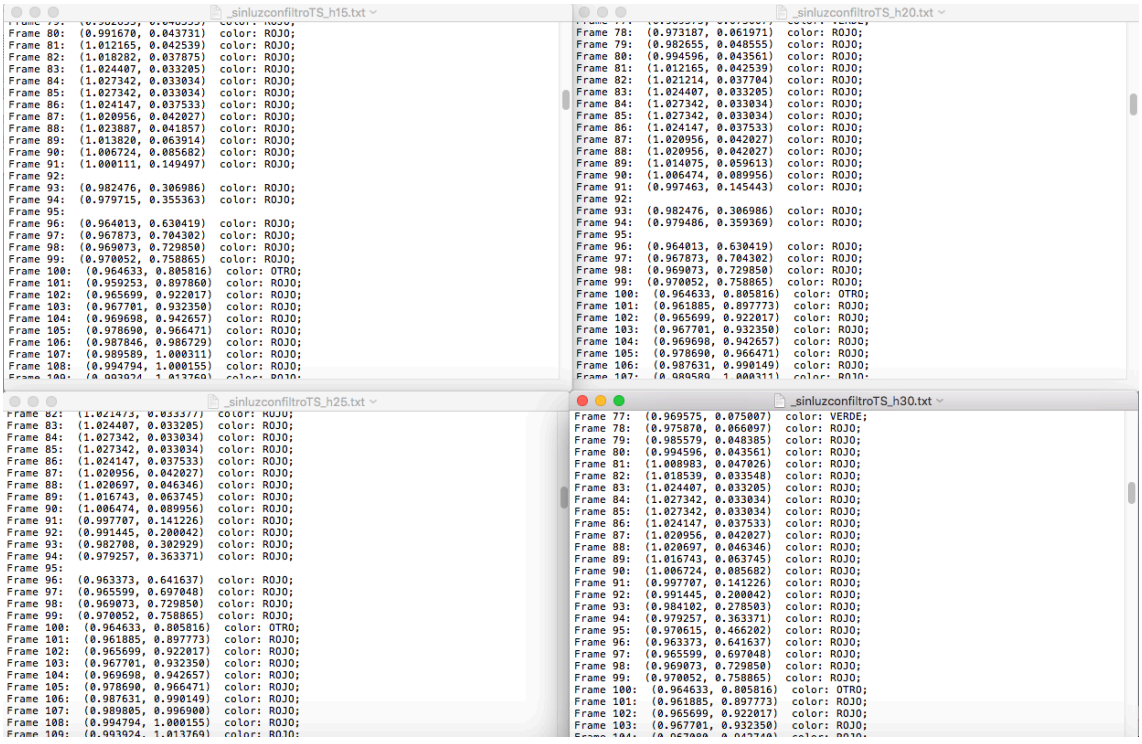


Figura 51. Comparativa de ficheros de salida H 15, 20, 25 y 30.

Se ha intentado ajustar más lo valores que afectan a la detección para intentar lograr una detección 100% perfecta pero en este caso ha sido imposible, siempre se escapa algún frame. De todas formas, y rompiendo una lanza a favor del algoritmo, la pérdida es tan minúscula que no tiene ningún efecto en la utilización de la aplicación.

Esta ha sido la primera prueba que se ha tenido que intentar corregir el valor de la sensibilidad para la detección del movimiento. Este valor normalmente tiene un valor de 0. Cuanto más alto sea el valor, más brusco tiene que ser el movimiento para que sea detectado.

Cuanto más alto sea el valor de la sensibilidad más se podrá subir el valor del umbral H. Pero esto tiene unas consecuencias, como que puede haber pérdidas en la detección porque el puntero "se mueve poco" o el movimiento no es suficientemente brusco como para ser detectado. En esta prueba al aumentar el SV (*Sensitivity Value*) también hemos podido aumentar el valor H, esto ha solucionado la detección en unos frames, pero también ha hecho que falle en otros donde con una sensibilidad menor no había error.

En la figura 52 podemos ver una captura de la prueba.

Figura 52. Prueba TuioSmoke en un entorno menos iluminado y con filtros

Prueba con objetos brillantes obstaculizando la detección

El algoritmo es capaz de detectar si un objeto es completamente estático (ni con el mejor de los pulsos se podría conseguir para un puntero láser). Esto es de utilidad cuando aparece un brillo indeseado en medio de la cámara y no se ha conseguido eliminarlo de otro modo(ver figura 53). Como es natural, no se podrán detectar los punteros láser cuando estén dentro de ese brillo que tendrá la misma o más luminosidad que el puntero.

Esta escena se ha probado con la aplicación TuioSmoke enfocando la cámara al techo de una habitación con la luz encendida. Como se puede apreciar en las figuras 54 y 55 la lámpara tiene mucho brillo, pero aun así no se detecta como puntero. Sin embargo, el láser si es detectado como contacto y así se reproduce en la aplicación TuioSmoke (ver figura 55)

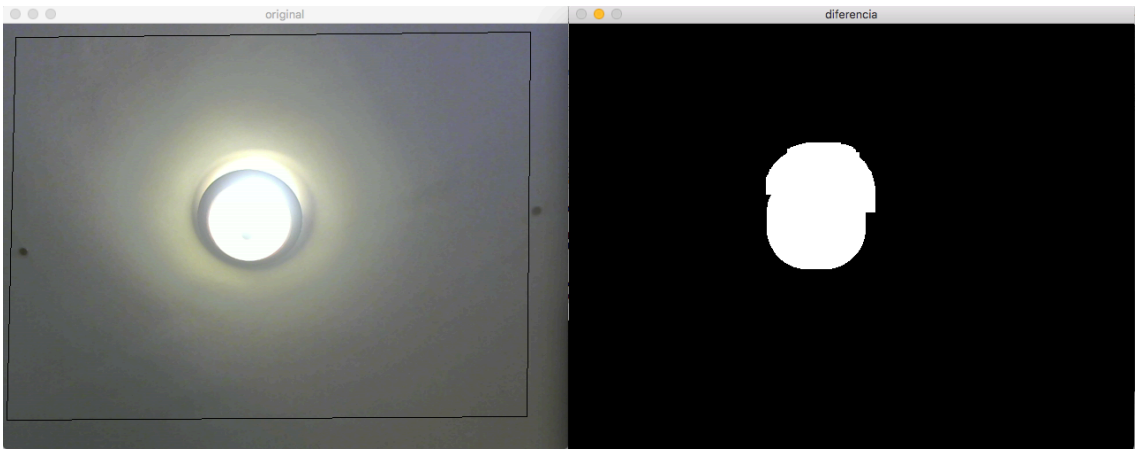


Figura 53. Brillo imposible de eliminar

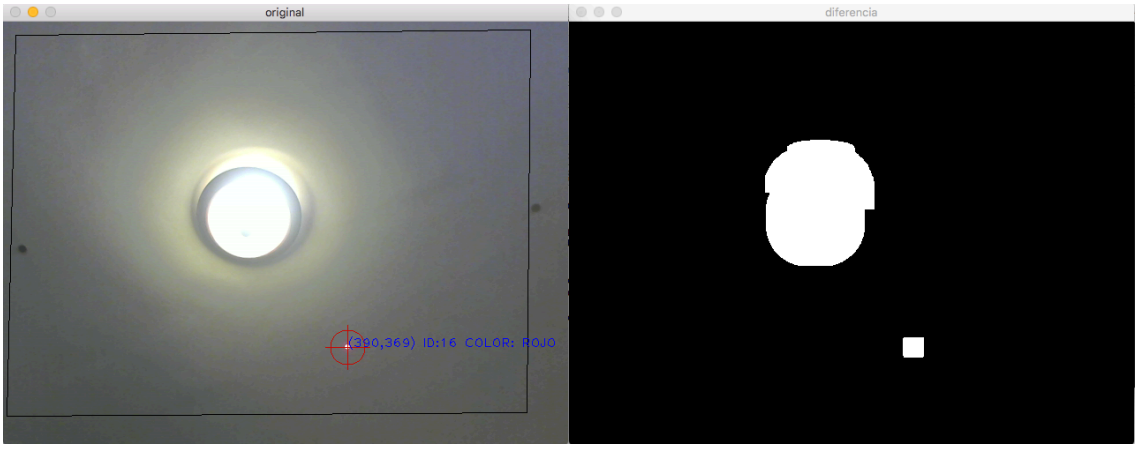


Figura 54. Solo detecta el puntero por que el brillo de la lámpara es estático.

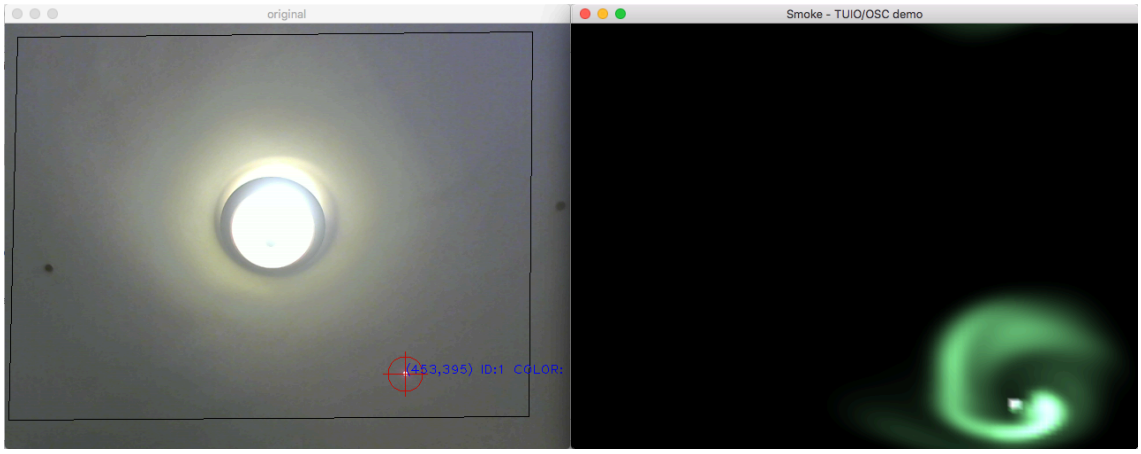


Figura 55. Resultado en la aplicación TuioSmoke

6. Conclusiones

Este proyecto ha tenido una duración aproximada de 6 meses de trabajo desde el inicio hasta que la aplicación fuese funcional. Ha habido días en los que se ha hecho más difícil compaginar el trabajo con el proyecto, aunque aún así se le ha dedicado todo el tiempo que ha sido posible. Este proyecto ha pasado por muchas fases, desde la búsqueda de información hasta las pruebas finales. Pero sin lugar a duda la fase que más se ha repetido en ese proyecto son las pruebas. Al mínimo cambio que se hacía en la aplicación se tenía que probar cómo afectaba a la aplicación en los diferentes entornos, aplicando filtros, ajustando los histogramas, modificando la distancia de la cámara, utilizando diferentes láseres, diferentes colores, etc.

Los mayores problemas a los que nos hemos enfrentado durante la realización del proyecto serían dos: la reducción de luz que capta la webcam y la detección del láser junto con las reducciones de luz.

La reducción de la luz en el sensor ha sido una tarea complicada porque en una primera instancia utilizábamos una cámara a la cual le faltaban los filtros infrarojos, esto hacía que la webcam captara imágenes muy saturadas, muy brillantes. Se intentó cambiar el tiempo de exposición de la cámara con OpenCV o el tamaño del diafragma, pero no se pudo lograr y hasta que se dio con la solución de los filtros polarizados para hacer uno de densidad neutra pasaron muchas pruebas y mucho tiempo. Encontrar los filtros fue un gran avance.

El segundo problema más importante a resolver fue la detección del láser. Se intentaron varios métodos como buscar el punto más brillante de la imagen (el cual se descartó), después se intentó localizar los puntos más brillantes pero que fueran circulares o elipses y que tuvieran un tamaño determinado (esto también se desechó ya que podían haber brillos circulares de tamaño similar que pasaban por láseres). Al final se decidió captar el movimiento de los objetos ya que los brillos suelen estar estáticos. Esta tampoco fue la decisión óptima ya que la aplicación tendrá un fondo en movimiento, por lo que se integró con el cálculo del histograma para solo quedarnos con el movimiento de los objetos más brillantes y si se quiere se puede activar un área máxima de X ya que el área del puntero puede variar dependiendo de la distancia al proyector.

Un tercer problema menos grave en la ejecución de la aplicación sería la detección del color del láser. Decimos que no es grave porque cada láser tiene su identificador, el color es más bien informativo. Este depende fuertemente del entorno, del fondo donde proyectemos los láseres, de la potencia de los láseres, de la potencia del

proyector y de la distancia de la cual se proyectan los láseres. Esto es así porque en un entorno con mucha luz la cámara no capta la elipse o el círculo de color del láser sino que lo percibe como una saturación, un punto blanco. El fondo sobre el cual proyectemos el láser también es un factor altamente importante ya que si lo proyectamos sobre colores oscuros el color de la luz del láser será mucho más claro y perceptible para la cámara. Por el contrario si los lanzamos sobre una superficie blanca, la webcam no será tan capaz de percibir el color. Para paliar con este problema se han explicado varias técnicas de reducción de luz a lo largo de la memoria, pero lo más conveniente es que no haya una luz excesiva en el entorno.

Conclusión del trabajo realizado

Después de todo el tiempo invertido, al fin, la aplicación cumple los requisitos establecidos al inicio del proyecto. Es capaz de identificar láseres ya sean del mismo o distinto color, obtiene sus características como el color, el área y la posición para una posterior decisión de si es o no es uno de los elementos que estamos buscando.

Sí que es verdad que el entorno condiciona en gran parte el funcionamiento de la aplicación, sobre todo la cantidad de luz que reciba la cámara puede ser la diferencia entre una ejecución perfecta y un mal funcionamiento. También la detección del color se ve afectada por la luz del entorno en el que nos encontremos, pues un entorno más oscuro nos ayudará a un mejor reconocimiento del color del láser. Hemos intentado reducir todo lo posible el efecto de la luz del entorno sobre nuestra aplicación pero en casos extremos no ha sido posible. A pesar de todo, y rompiendo una lanza a favor del trabajo realizado, esta aplicación está pensada para ser utilizada junto con un proyector el cual proyectará la aplicación táctil y los usuarios podrán interactuar mediante los láseres apuntando a la proyección. Es de todos sabido que para ver mejor lo que se está proyectando con el proyector nos ayudará un entorno más oscuro. Esto no quiere decir que tengamos que estar en la más absoluta oscuridad, pero un entorno con menos luz nos ayudará a ver mejor la aplicación multitáctil y al funcionamiento de la aplicación desarrollada en este proyecto.

Una vez ya finalizado todo el proyecto uno mismo se da cuenta de que no solo ha aprendido a realizar una tarea o un proyecto. Se van adquiriendo conocimientos sin darse cuenta, solo porque los necesitas para llegar a resolver cierto problema. No solo aprendes a programar una aplicación o a desarrollarla, te das cuenta de lo importantes que son muchos de los pasos que antes no creías que fueran tan significativos.

Trabajos futuros

Este proyecto podría ser ampliado de muchas formas y podría ser un buen comienzo para el desarrollo de otras aplicaciones similares.

Un ejemplo de ampliación podría ser un mecanismo para la regulación de la luz que entre en la cámara. Como se ha visto en el capítulo 5, dependiendo del entorno es necesario recalcular el valor de H y posiblemente el de sensibilidad SV. Se podría desarrollar un regulador que al iniciar la aplicación disparara un láser contra el proyector encendido y con algún tipo de hardware hacer rotar los filtros polarizados al ángulo ideal para utilizar la aplicación, ya que esto ahora se tiene que hacer a mano o ir cambiando los valores H y SV.

Otra mejora podría ser integrar la webcam en el proyector para que siempre esté ajustada correctamente y no hacerlo de forma manual.

Gracias a este proyecto se puede tener una buena base para desarrollar aplicaciones cliente para ser utilizada con punteros láser como sustitutivo de los contactos en la pantalla. Así que, se puede utilizar todo lo que ya se ha construido en este proyecto para nuevos proyectos de mayor envergadura o para el desarrollo de aplicaciones cliente que entienda los mensajes OSC.

7. Manual de usuario

Antes de comenzar a utilizar la aplicación se le debe decir en qué dirección y puerto está escuchando la aplicación cliente que se va a utilizar mediante los punteros láser. Por ejemplo, si la aplicación cliente está en el mismo ordenador que el servidor, la dirección será localhost y el puerto será el que esté escuchando la aplicación (normalmente es el 3333). Pero si la aplicación estuviera en otra dirección y escuchara en otro puerto, bastaría solo con cambiar estos parámetros.

Ahora se conecta la webcam al puerto usb y se le dice en qué canal está la webcam, en el caso de que el ordenador tenga más de una cámara (en mi caso, el 0 es la cámara integrada del portátil y el 1 la usb).

Una vez configurada la dirección de la aplicación cliente y la webcam se debería proyectar la aplicación con la ayuda de un proyector sobre una pared blanca, o una pantalla. Al ejecutar la aplicación se mostrará una ventana que será la imagen que está captando la webcam. Esta imagen hay que ajustarla para un funcionamiento óptimo como se ha explicado al final del apartado 4.4. Para asignar nuevas coordenadas a las esquinas lo que hay que hacer es apuntar con el láser donde se quiere que esté la esquina superior izquierda y pulsar la tecla numérica 1. Lo mismo hay que hacer para la esquina superior derecha, pero esta vez pulsando la tecla numérica 2. Para la esquina inferior derecha se pulsa el 3, y 4 para la esquina inferior izquierda. Una vez seleccionadas todas las esquinas se hace automáticamente una conversión de coordenadas mediante la homografía.

El siguiente paso es opcional, aunque recomendable, y depende de la cantidad de luz que hay en la sala. Si la cantidad de luz es grande se recomienda poner los filtros polarizados rotados delante del objetivo de la webcam como se muestra en el apartado 4.1.3, hasta que el punto más brillante detectado sea el de los punteros láser

Una vez ajustada la webcam al proyector se pulsa la tecla B lo que guardará una captura del fondo, para una mejor detección del movimiento. Ahora hay que hacer un ajuste del histograma con el láser proyectado en el proyector, y para ello solo tenemos que pulsar la tecla H cuando el láser esté en la pantalla. Con este ajuste de histograma lo que estamos haciendo es que solo vamos a tener en cuenta los objetos más brillantes, como el láser.

En este punto podemos pulsar la tecla V para mostrar la ventana de depuración, si todo ha salido bien, la ventana debería mostrar una imagen negra y al proyectar el láser debería aparecer una pelotita blanca en la posición del láser.

Como se puede apreciar en la parte superior de la ventana hay una barra donde podemos elegir el tamaño máximo del área del láser, para no tener en cuenta, objetos más grandes que ese área.

Si esto es correcto ya podemos pulsar la tecla T para empezar con el tracking de los láseres y a utilizar la aplicación.

Resumen de teclas

- B : Capturar el fondo.
- H : Ajustar el histograma.
- T : Activar/Desactivar el tracking de los láseres. Si no está activo, no se enviaran los mensajes TUIO.
- V : Muestra las ventanas para la depuración.
- 1 : Marca la nueva esquina superior izquierda.
- 2 : Marca la nueva esquina superior derecha.
- 3 : Marca la nueva esquina inferior derecha.
- 4 : Marca la nueva esquina inferior izquierda.
- R : Deshace las nuevas coordenadas para volver a seleccionar unas nuevas.
- Esc : Salir de la aplicación

8. ANEXOS

A. La clase Blob

Para la realización del proyecto hemos decidido que es necesaria la implementación de una clase Blob con sus atributos. Ya que toda precaución es poca, incluso después de todos los mecanismos que hemos utilizado hasta ahora para evitar tener en cuenta algo que no sea el puntero láser, aún podrían aparecer blobs que no queramos. Por esa razón, vamos a localizarlos y a crear objetos de tipo Blob que tendrán atributos o características que nos ayudarán a diferenciar blobs que no nos son útiles del blob del láser. Los atributos de un objeto blob son:

- ID: Un identificador de blob.

- Área: Nos dirá el tamaño del blob.

- Color: El color del blob (nos será útil cuando tengamos que diferenciar los láseres).

- Está estático: Para saber si el blob está en el mismo sitio siempre.

- Contador: El contador se incrementa cada vez que el blob está en la misma coordenada, si el contador llega a X, decimos que el blob está estático

- Centro: Las coordenadas x y del centro del blob.

- Ultimo centro: Las coordenadas de su centro en el frame anterior (nos ayudará para calcular distancias entre blobs y saber si es el mismo)

- BoundingRect: un atributo de tipo Rect, que nos ayudará a encontrar la posición del blob. Sirve para encontrar contornos.

Estas características se podrán utilizar posteriormente a diferenciar los punteros láser de otros brillos en la imagen.

B. Detección de blobs circulares y ovalados

El puntero láser en la imagen es un pequeño círculo con un valor de luminosidad bastante elevado. Pero no solo queremos detectar círculos, aunque a primera impresión veamos que en el frame binarizado tenemos un punto que representa el láser, eso es porque el láser está estático. Pero cuando movemos el puntero por la imagen el láser se deforma cogiendo formas ovaladas incluso casi rectas. Para poder localizarlo en la imagen binarizada nos vamos a ayudar de un código para detección de blobs que nos permite ajustar los parámetros de los blobs que queremos localizar como el tamaño, la forma, el brillo, la convexidad (si es cóncavo o convexo) o la relación de inercia (Ver figura 56).

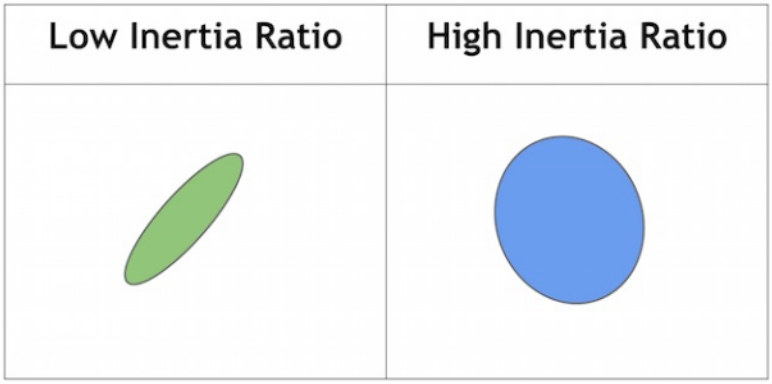


Figura 56. Relación de inercia. Se refiere a la longitud de la forma. Utiliza un valor entre 0 y 1 siendo 1 un círculo, un valor entre 0 y 1 una elipse y un 0 una línea

El código en OpenCV para definir estas características se muestra a continuación en las figuras 57 y 58.

```
286         /*******************************/
287         /******* BLOBS SETTINGS  *******/
288         /*******************************/
289
290         // Setup SimpleBlobDetector parameters.
291         SimpleBlobDetector::Params params;
292
293         // Change thresholds
294         params.minThreshold = 10;
295         params.maxThreshold = 255;
296
297         // Filter by Area.
298         params.filterByArea = true;
299         params.minArea = 10;
300         params.maxArea = 800;
301
302         // Filter by Circularity
303         params.filterByCircularity = true;
304         params.minCircularity = 0.1;
305
306         // Filter by Convexity
307         params.filterByConvexity = true;
308         params.minConvexity = 0.87;
309
310         // Filter by Inertia
311         params.filterByInertia = true;
312         params.minInertiaRatio = 0;
313
314
315         // Storage for blobs
316         vector<KeyPoint> keypoints;
317
318         /*************************************/
319         /******* END BLOBS SETTINGS  *******/
320         /*************************************/
321
```

Figura 57. Ejemplo de definición de características para la detección de blobs en OpenCv

```
387
388         /*********************************/
389         /********** BLOB DETECTOR *********/
390         /*********************************/
391         #if CV_MAJOR_VERSION < 3   // If you are using OpenCV 2
392
393             // Set up detector with params
394             SimpleBlobDetector detector(params);
395
396             // Detect blobs
397             detector.detect( invertida, keypoints);
398         #else
399
400             // Set up detector with params
401             Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);
402
403             // Detect blobs
404             detector->detect( invertida, keypoints);
405         #endif
406
407         Mat im_with_keypoints;
408         drawKeypoints( ventanabinaria, keypoints, im_with_keypoints, Scalar(0,0,255), DrawMatchesFlags::
                DRAW_RICH_KEYPOINTS );
409
410
```

Figura 58. Detectando los blobs y marcándolos con un círculo rojo.

En la figura 58 se está creando un objeto *SimpleBlobDetector* (que nos proporciona OpenCV) y le está pasando los parámetros que se le han definido en la figura 57. Una vez creado el objeto utiliza la función *Detect* y le pasa como parámetros el *frame* o imagen donde queremos detectar los blobs y un vector de tipo *KeyPoint* donde se

guardará las posiciones de los blobs que encuentre con esas características. En la línea 408 está dibujando los círculos encima de los blobs que ha encontrado.

Podemos ver unos ejemplos de cómo funciona en las siguientes figuras.

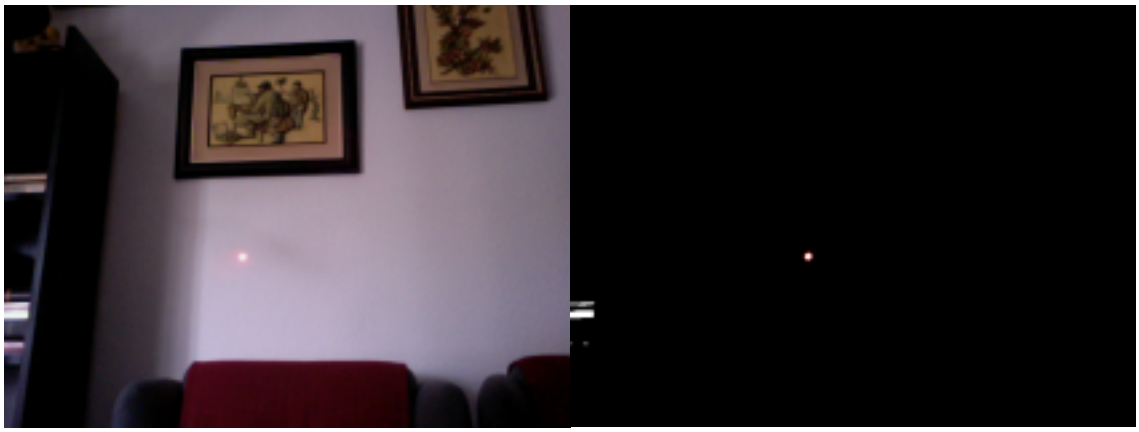


Figura 59. Comportamiento del detector de blobs

En la figura 59 de la derecha vemos que el láser no es la única zona saturada de la imagen, pero con el detector de blobs y sus características ajustadas se ha podido marcar solo el láser y obviar el resto de interferencias en la imagen. Pero, ¿qué sucede si el láser está en movimiento? ¿será capaz de detectarlo? Veamos las figuras 60 y 61.

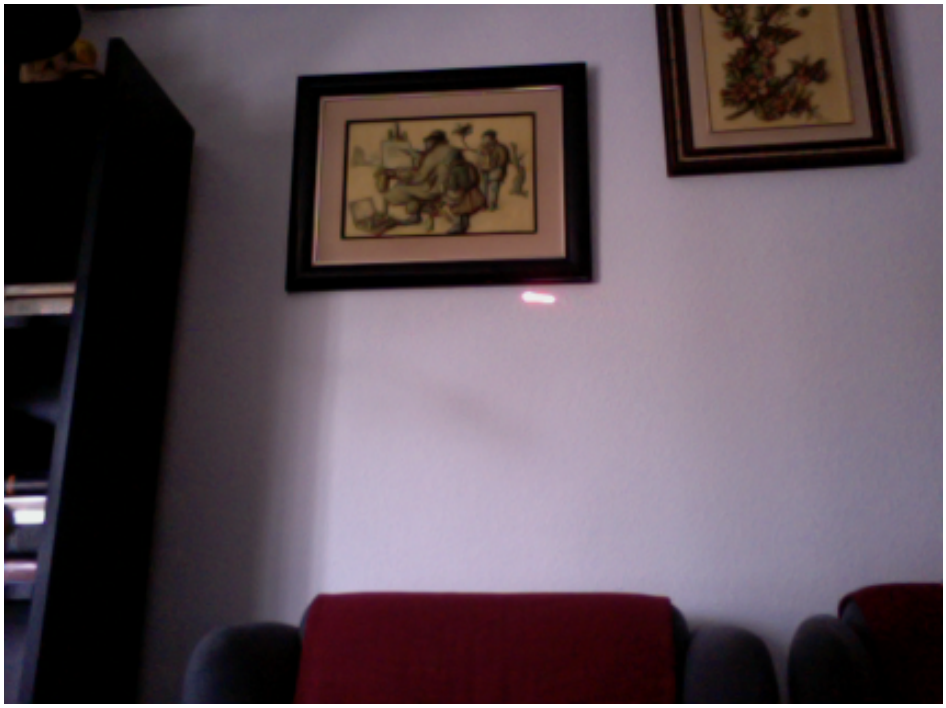


Figura 60. Láser en movimiento

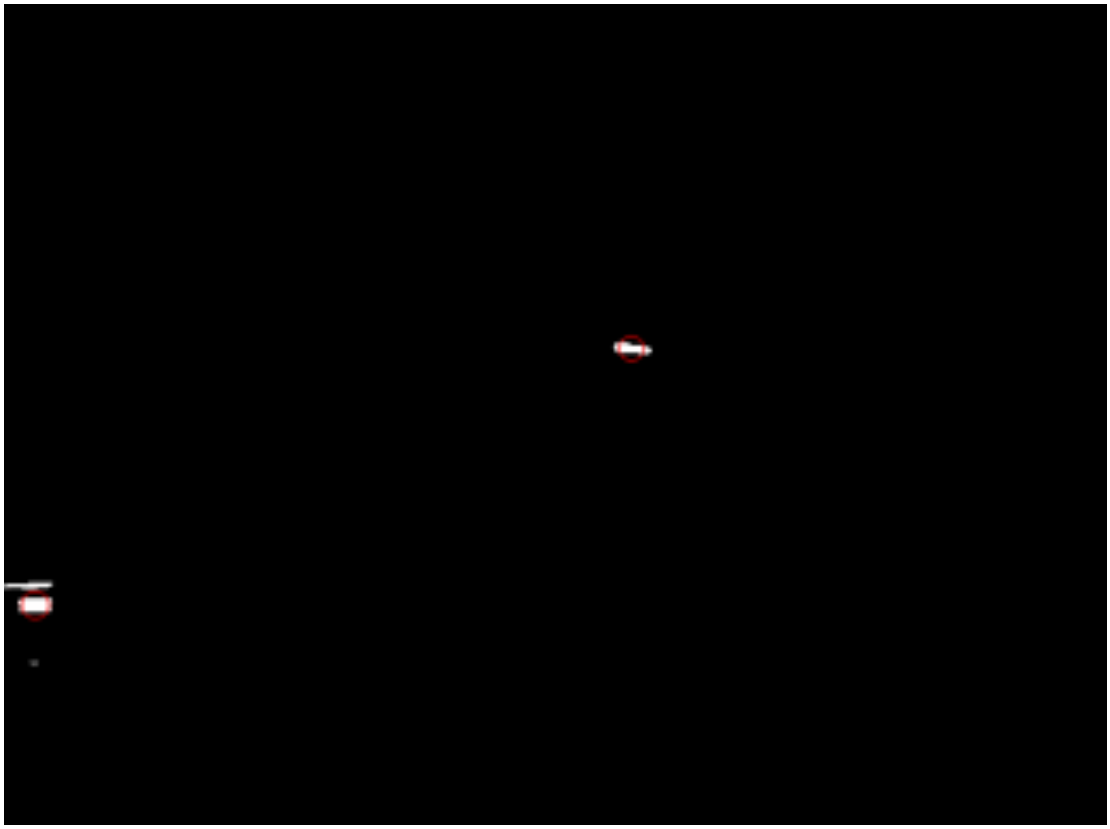


Figura 61. Binarización y detección de blobs de la figura 60

Como hemos definido en las características de los blobs que queremos detectar que su relación de inercia sea entre casi una línea y un círculo pasando por los óvalos y que su área sea entre unos tamaños mínimos y máximos, ha dado la casualidad de que un reflejo que no tiene nada que ver con el láser cumple las características al igual que el puntero. Así pues, no nos podemos fiar por completo de la detección de blobs, y por lo tanto no va a ser nuestra única herramienta para localizar la posición exacta de los punteros. Aunque puede ser de ayuda.

C. El protocolo TUIO

TUIO es un framework abierto que define un protocolo común y una API para superficies multi-touch. El protocolo TUIO permite la transmisión de una descripción abstracta de las superficies interactivas, incluyendo eventos de contacto y los estados de los objetos tangibles. Este protocolo codifica datos de control desde una aplicación de seguimiento y lo envía a cualquier aplicación cliente capaz de decodificar el mensaje.

Ya hay un gran número de aplicaciones que son capaces de codificar mensajes TUIO al igual que existen muchas otras capaces de decodificarlos. Esto facilita y hace más rápido el desarrollo de interfaces multi-touch.

TUIO se basa en OSC (Open Sound Control), que es un estándar emergente para entornos interactivos que no solo se limita al control de instrumentos musicales.

El mensaje FRM es un identificador único para un frame individual, y además ha de ser incluido al principio de cada paquete TUIO. Cada frame está identificado con un entero de 32bits sin signo que representa el ID del frame.

Hay diferentes tipos de mensages TUIO como [10]:

PTR (pointer messages)

/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press [x_vel y_vel p_vel m_acc p_acc]

/tuio2/ptr int32 int32 int32 float float float float float [float float float float float]

Donde los identificadores del 1..5 son utilizados para definir los dedos de la mano derecha, los ID del 6..10 son los dedos de la mano izquierda. Siguiendo este orden: 1 índice, 2 corazón, 3 anular, 4 meñique, 5 pulgar, 6 índice, 7 corazón, etc...

El rango de identificación de 11-20 define una pequeña selección de los dispositivos de puntero común (lápiz 11, puntero láser 12, ratón 13, trackball 14, joystick 15). El rango de identificación de 21-30 define varias partes del cuerpo (21 apuntando mano derecha, mano derecha abierta 22, 23 de la mano derecha cerrada, 24 que señala la mano izquierda, la mano izquierda abierta 25, 26 de la mano izquierda cerrada, 27 pie derecho, pie izquierdo 28, 29 de cabeza, 30 persona). Cualquier tipo de identificación a partir de 64 y por encima se puede asociar libremente por la aplicación de

seguimiento. Parámetros de velocidad y aceleración son opcionales y la implementación de cliente tiene que considerar las dos posibles longitudes de mensaje.

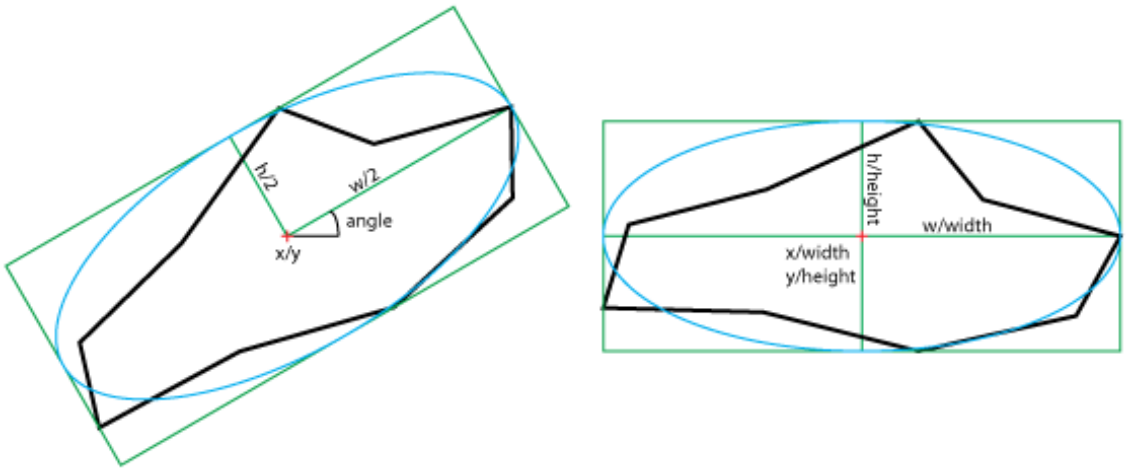
BND (bounds message)

/tuio2/bnd s_id x_pos y_pos angle width height area [x_vel y_vel a_vel m_acc r_acc]

/tuio2/bnd int32 float float float float float float [float float float float float]

Los mensajes BND codifican la información de la geometría básica de los objetos genéricos sin identificar (blobs). El formato del mensaje describe la elipse interior de un cuadro delimitador orientado, con su punto central, el ángulo del eje mayor, las dimensiones del eje mayor y menor, así como el área de la región. Por lo tanto este formato compacto lleva información sobre la región elíptica aproximada, y permite la reconstrucción orientada de la caja de contorno. La zona de la región se normaliza en píxeles / ancho * altura, proporcionando un acceso rápido al tamaño global de la región.

El mensaje BND generalmente identifica los límites de cualquier objeto físico sin etiqueta genérica, y también se puede utilizar para transmitir la información básica de geometría, tales como el ángulo y dimensiones de manchas de los dedos o tokens que han sido ya identificados por un mensaje PTR o TDC anterior. El identificador de sesión tiene que ser igual en ambos mensajes con el fin de que coincida con el componente con los límites correspondientes.



Cada paquete debe contener al menos un mensaje FRM y un mensaje de ALV, donde el mensaje FRM es el primer mensaje en un paquete, mientras que el mensaje de ALV es el último mensaje del paquete.

La siguiente tabla muestra algunas funciones que tiene el objeto TuioServer. El que utiliza este proyecto.

Devuelve	Función	Explicación
	TuioServer(char *host, int port)	Crea el objeto tuio server y se le pasa como argumento la dirección y el puerto donde se enviarán los mensajes
TuioObject *	addTuioObject(int sym, float xp, float yp, float a)	Crea un objeto tuio object con su id, coordenada x, coordenada y, y ángulo
void	updateTuioObject(TuioObject *tobj, float xp, float yp, float a)	Actualiza un objeto tuioObject. Pasándole el objeto a actualizar y sus nuevos atributos.
void	removeTuioObject(TuioObject *tobj)	Elimina el objeto
TuioCursor *	addTuioCursor(float xp, float yp)	Añade un objeto tuioCursor con sus coordenadas x, y
void	updateTuioCursor(TuioCursor *tcur, float xp, float yp)	Actualiza el objeto *tcur
void	removeTuioCursor(TuioCursor *tcur)	Elimina el objeto tuioCursor
void	initFrame(TuioTime ttime)	Tiene que estar al principio de todos los mensajes que enviemos.
void	commitFrame()	Ultima instrucción del mensaje. Envía los cambios.
long	getSessionID()	Devuelve el identificador de la sesión
TuioTime	getFrameTime()	Devuelve el tiempo que ha pasado desde el inicio de sesión.
void	stopUntouchedMovingObjects()	detiene los objetos que no están siendo tocados
void	stopUntouchedMovingCursors()	detiene los cursores
std::list< tuioObject * >	getTuioObjects()	Devuelve un array con todos los objetos activos
std::list< tuioCursor * >	getTuioCursors	Devuelve un array con todos los cursores activos
bool	isConnected()	Si el servidor está conectado devuelve True, false en caso contrario
void	setVerbose(bool verbose)	Si el parámetro es True, mostrará mensajes cuando actualicemos, borremos o añadamos objetos. Si es false no mostrará nada.

9. Bibliografía

1. **ReacTIVision**. [en línea] [Citado el: 23 de Mayo de 2016.]
http://reactivision.sourceforge.net/.

2. **Reactable Systems**. Reactable. [en línea] [Citado el: 23 de Mayo de 2016.]
http://www.reactable.com/.

3. **NUI Group**. Community Core Vision documentation. [en línea] [Citado el: 23 de Mayo de 2016.]
http://ccv.nuigroup.com/#docs

4. **t2i interaction laboratory.** Ortholumen. [en línea] [Citado el: 23 de Mayo de 2016.]
http://t2i.se/ortholumen-2/.

5. **OpenCV.org**. Open Source Computer Vision. [en línea] [Citado el:1 de Junio de 2016.]

6. **Gary Bradski & Adrian Kaehler**. Learning OpenCV. Computer Vision with OpenCV Library. ISBN: 978-0-596-51613-0.

7. **Tuio.org**. Open framework that defines a common protocol and API for tangible multitouch surfaces. [en línea] [Citado el: 4 de Junio de 2016].
http://www.tuio.org

8. **Nada Que Hacer**. Cómo hacer filtro de Densidad Neutra Variable ND Filter. [en línea] [Citado el: 1 de Junio de 2016.]
https://www.youtube.com/watch?v=E-sANzD0cSc

9. **HSV**. Wikipedia. [en línea] [Citado el: 24 de Mayo de 2016.]
https://es.wikipedia.org/wiki/Modelo_de_color_HSV

10. **TUIO 2.0 Protocol Specification.** [en línea] [Citado el: 23 de Mayo de 2016.]
http://www.tuio.org/?tuio20

11. **Técnicas de visión.** Calibrar la cámara con OpenCV. [en línea] [Citado el: 6 de Junio de 2016.]
http://tecnicasdevision.blogspot.com.es/2014/05/calibrar-una-camara-con-opencv.html

12. Tommaso Piazza. Ortholumen: Using Light for Direct Tabletop Input. Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on. [Fecha conferencia 10-12 de Octubre de 2007.]

13. Peter Vandoren. IntuPaint: Bridging the gap between physical and digital painting. Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on. [Fecha conferencia 1-3 de Octubre de 2008.]

14. Pfeiffer, Gustavo Thebit; Marroquim, Ricardo Guerra; Oliveira, Antonio Alberto Fernandes. "WebcamPaperPen: A Low-Cost Graphics Tablet", *Graphics, Patterns and Images (SIBGRAPI).* [Fecha de conferencia e conferencia 26-30 de Agosto de 2014.]