Prototipo de sistema de medida de tiempo de permanencia en comercios

David García Albert Antonio Albiol Colomer

14 de julio de 2015

Índice general

1. Introducción **2**

2. Descripción del sistema **3**
2.1. Cámara 3D . . . 3
2.2. PC . . . 5
2.3. Software de tratamiento de imagen . . . 6
2.3.1. KheightDB . . . 7
2.3.2. matchDB . . . 13
2.4. Base de datos MySQL . . . 21
2.4.1. Descripción de las tablas . . . 21
2.4.2. Configuración de accesos de usuario . . . 24
2.4.3. Acceso remoto . . . 24
2.4.4. Volcado de datos . . . 25
2.4.5. Inserciones en las tablas Camera y Host . . . 25
2.5. Software de gestión . . . 26
2.5.1. SSH . . . 26
2.5.2. Samba . . . 26
2.5.3. rsyslog . . . 27
2.5.4. Scripts . . . 28
2.6. Ficheros de configuración . . . 30
2.6.1. sqlserver_config.xml . . . 30
2.6.2. kheight_config.xml . . . 31
2.6.3. matchDB_config.xml . . . 33
2.6.4. Fichero tCounter . . . 34

3. Resultados **36**
3.1. Conjunto de datos . . . 36
3.2. Test del sistema . . . 37

4. Conclusiones **44**

Capítulo 1

Introducción

La necesidad de conocer mejor los hábitos y costumbres de consumidores que suelen frecuentar un comercio, lleva a la necesidad de buscar métodos que nos permitan medir con precisión ciertos patrones de comportamiento que siguen a la hora de realizar sus compras o simplemente el hecho de visitar las tiendas en su tiempo de ocio. Con tal fin, este proyecto pretende implementar un sistema que pueda registrar los accesos y salidas para estimar los tiempo de permanencia en comercios utilizando un modo no invasivo y discreto que evite ralentizar el tránsito de entrada y salida. Una opción muy interesante es el uso de técnicas de procesamiento de imagen que tan solo requieren de una cámara instalada en la entrada y apenas son visibles para los consumidores. Por tanto, en este proyecto se ha decantado por utilizar cámaras 3D, como la Microsoft kinect, con un PC con el software necesario, para abordar este reto, teniendo en cuenta que el coste final del sistema no sea elevado y que permita a empresarios no solo de grandes comercios, sino también de pequeños y medianos, añadir un servicio extra para potenciar sus ventas, optimizar recursos y en definitiva, gestionar mejor su negocio.

El uso de cámaras 3D permiten obtener una matriz de color, matriz *RGB*, a la vez de la información de profundidad, matriz *Depth*, que nos permite conocer las coordenadas exactas de cada píxel en la imagen *RGB*. Además, al contemplar dicha información extra, permite diseñar un sistema de reidentificación robusto frente a problemas típicos en los sistemas de vídeo 2D como las oclusiones. El PC, aporta un gran potencial por las infinitas posibilidades de ejecutar cualquier programa bajo un sistema operativo que gestione el sistema, almacene la información y en general ofrezca diversos servicios como de acceso remoto o la posibilidad de integrar el sistema en una red de área local.

El desarrollo de este proyecto parte de un trabajo desarrollado por el grupo de procesamiento de imagen y vídeo de Valencia (GPIV) perteneciente a la Universidad Politécnica de Valencia (UPV)[14]. En dicho trabajo, se desarrolló un software de reidentificación de personas basado en cámaras 3D kinect para realizar el seguimiento y reidentificación de personas con el fin de controlar los accesos y salidas en comercios. En el proyecto que nos ocupa, se pretende definir un sistema completo que pueda instalarse en un comercio y que lleve a cabo dichas tareas de forma automática y ponga de forma accesible la información registrada.

Capítulo 2

Descripción del sistema

El sistema diseñado para determinar el tiempo de permanencia en los comercios, captura el instante en el que entra y sale una persona con el fin de obtener cierta información como:

- Tiempo de permanencia
- Hora de entrada y salida
- Número de personas en el comercio
- Velocidad a la cual entran o salen del comercio
- Otra información derivada de las anteriores

Para ello, se ha pensado en un sistema no invasivo y transparente para el consumidor o visitante, que no suponga un obstáculo ni distracción en el momento de entrada o salida. En la figura 2.1 se muestra un diagrama de bloques que describe los principales bloques del sistema. Además, coexisten otros procesos y ficheros que añaden servicios de conectividad y extracción de información y que a continuación se irán describiendo su implementación y funcionamiento, los ficheros de configuración, el formato de las tablas, etc.

2.1. Cámara 3D

Las cámaras 3D como la Microsoft Kinect[4] que se emplean como dispositivo de entrada en las videoconsolas Xbox360[13], permiten segmentar la figura de una persona, estimar la pose, detectar movimiento e incluso hacer una reconstrucción 3D del escenario. Por tanto, las cámaras 3D como la kinect permiten resolver con facilidad el problema de las oclusiones de forma eficiente. La clave que permite resolver en cierta manera, de forma sencilla la segmentación de las personas dentro del escenario que representa la entrada de un comercio, es la información que aporta la matriz de profundidad (*Depth*) gracias al sensor de infrarrojos (IR) del que dispone la cámara Kinect, además del sensor *RGB* convencional de cualquier cámara 2D.

Por otra parte, las librerías OpenNI[7], publicadas por la empresa PrimeSense (empresa que desarrolló la kinect y adquirida por Apple[1] en 2013), proporciona la interfaz de acceso a las matrices *RGB* y *Depth* que son la fuente de información principal de nuestro sistema y en concreto, del proceso *kheightDB*, encargado de registrar las entradas y salidas así como de realizar inserciones en la base de datos. Por tanto, el trabajo [14] nos proporciona un

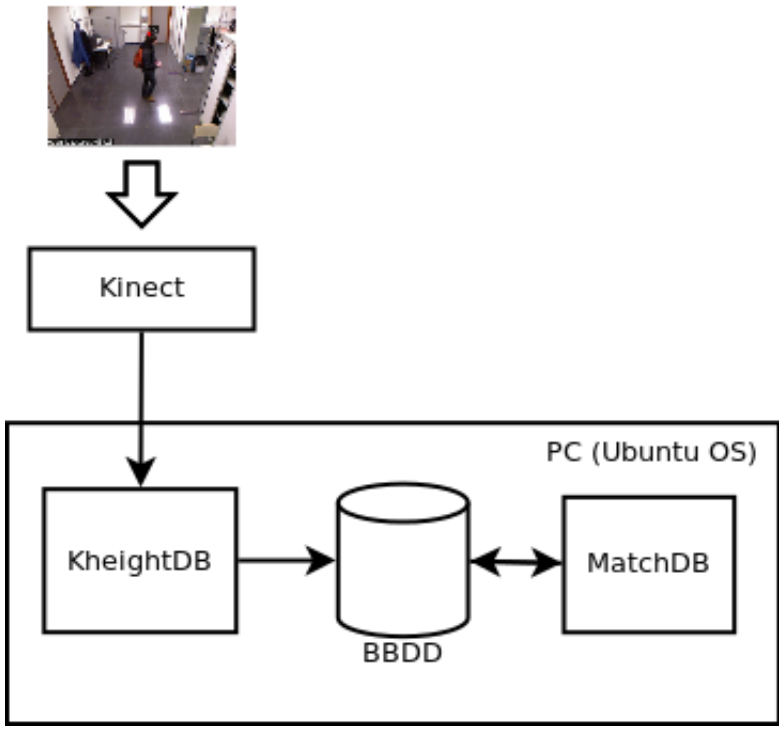

Figura 2.1: Diagrama de bloques del sistema

proceso llamado "captura_openni" que emplea dichas librerías y que nos servirá de enlace entre la cámara kinect y nuestro proceso de registro *kheightDB*.

Al disponer de dos sensores, RGB y IR en las cámaras kinect, disponemos de dos matrices o ficheros de información por cada *frame*: La imagen en color y habitual de cualquier otra cámara y otro fichero que asocia cada pixel de la imagen con unas coordenas XYZ. Finalmente, remarcar que el sensor IR tiene limitado su uso a espacios interiores debido al bajo contraste de dicho sensor y por tanto, no es posible utilizar la cámara en espacios con exposición a la luz solar directa. A priori, dicha limitación no supone ningún inconveniente en este proyecto ya que el sistema está destinado a funcionar en el interiores de comercios, aunque sí se debe tener en cuenta no disponer en la imagen zonas de luz solar directa o reflejos para evitar zonas en la imagen en las que no se disponga de información en la matriz *Depth*.

Calibración de la cámara

Previamente se debe realizar una calibración tras cada instalación por diversos motivos:

- Realizar un cambio a nuestro sistema de coordenadas calculando la matriz de transformación. Ver figura 2.2
- Obtener la altura de la cámara
- Definir la máscara de la zona de acceso al comercio

El método a seguir es siempre el mismo y debe repetirse para cada cámara que se instale en el sistema. Los comandos utilizados se desarrollaron durante el trabajo [14] del GPIV. A continuación se detalla los pasos a seguir para calibrar la cámara kinect:

Figura 2.2: Sistema de coordenadas de nuestro sistema y línea de cruce (*crossline*) de entrada y salida del comercio.

- Realizar una captura del escenario una vez tengamos instalada la kinect de forma definitiva mediante el comando "captura_openni -Q -T 3 -r". Detener transcurridos unos segundos.
- Definir la máscara como en la figura 2.3 tras ejecutar el comando "set_mask Img_RGB_00000XX.jpg donde el fichero "Img_RGB_00000XX.jpg" es una de las imágenes capturadas en el paso anterior. Pulsar la tecla 's' para guardar y 'q' para salir y obtendremos un fichero "mask_00.tif".
- Lanzar el comando "callibrateGround -b Img_Depth_00000XX.cimg -m mask_00.tif" para generar los ficheros "cam_height.dlm", que indica la altura de la cámara en milímetros y "matrix.dlm", que contiene la matriz de transformación.

.

A continuación sería necesario indicar la altura y matriz de transformación en el fichero de configuración "kheight_config.xml" como se explica en el punto 2.6.

2.2. PC

Para interconectar todas las partes y ejecutar los procesos necesarios se ha empleado un ordenador personal (PC) con arquitectura x86 compatible con sistemas operativos GNU/-Linux. El sistema operativo empleado es Ubuntu 12.04 Long Term Support (LTS)[12] con el fin de poder garantizar una compatibilidad con máquinas sin grandes prestaciones y un soporte de larga duración para evitar en la medida de lo posible cambios de versión en las librerías que puedan causar incompatibilidades. Utilizar un sistema operativo de código abierto, gratuito y libre evita costes derivados de licencias y se dispone de un gran repositorio herramientas de gestión como de escritorio. Los requerimientos mínimos del

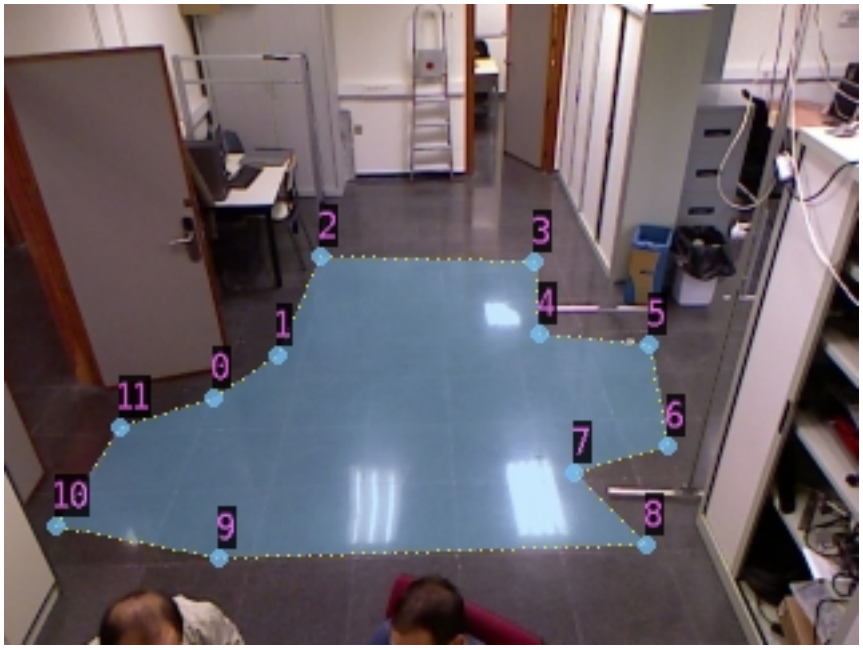

Figura 2.3: Máscara definida durante la calibración de la cámara en el laboratorio del GPIV

PC no son demasiado exigentes ya que con una versión *server* con una instalación básica y sin entorno gráfico es suficiente. Aún así, los requerimientos mínimos son:

- Procesador, 2 GHz
- Memoria RAM, 4 GB
- Disco duro, 128 GB
- Tarjeta de red, 100 Mbps
- USB 2.0

2.3. Software de tratamiento de imagen

Las principales librerías desarrolladas para el procesamiento de imagen parten de [14]. Dicho código ha servido de punto de partida para la implementación de las principales partes del sistema que nos ocupa. El programa o proceso que procesa directamente las imágenes de la Kinect se llama *kheightDB* y es el encargado de segmentar las personas dentro del escenario. Cuando una persona aparece en la imagen es seguida por ente interno del proceso *kheightDB* llamado *PersonTracker*. Cuando se considera que una persona ha entrado o salido del comercio, bien sea porque ha cruzado la línea de entrada fijada (*crossline*) o porque se ha salido de los límites del escenario, el *PersonTracker* registra la trayectoria, tiempo de inicio de aparición en el escenario, la hora y el número de imágenes (*frames*) en los cuales ha aparecido, en un registro denominado *track*. Dicho *track* contiene información de color durante su tiempo en escena y otra información de interés para posteriormente poder ser comparado con otros *tracks*. La información que identifica a un *track* se denomina *bodyprint* o firma de color. Por tanto, el proceso *kheightDB* debe recoger las imágenes registradas por la cámara kinect e ir grabando los *tracks* en el disco duro y

realizar las entradas en la tabla "Event". Previamente, el proceso *kheightDB* debe cargar dos ficheros de configuración, "sqlserver_config.xml" y "kheight_config.xml". El primero determina los parámetros de la base de datos donde se guarda la información de los *tracks* de entrada y salida. Y el segundo de ellos contiene parámetros relativos a los ficheros de calibración de la cámara, límites del escenario y otros parámetros relacionados. En el punto 2.6 se describen con detalle todos los parámetros de configuración de cada uno de ellos.

Por otro lado, se encuentra el proceso *matchDB* que es el encargado de hacer los emparejamientos o *matching*. Dicho proceso debe conectarse a la base de datos e ir analizando los *tracks* de salida que todavía no han sido emparejados e ir comparando con todos aquellos *tracks* de entrada libres de emparejamiento. Ante un *track* de salida, pueden darse tres situaciones: La más deseada es que dicho *track* de salida encuentre un *track* de entrada que obtenga una puntuación máxima (caso ideal pero muy poco probable) o muy próxima al máximo valor (entre 0 y 1, 1 es el valor máximo obtenido tras comparar un *track* consigo mismo). Otra posibilidad es que dos *tracks* obtuvieran una puntuación (*score*) por encima del umbral prefijado y que delimita cuando dos *tracks* corresponden a una misma persona. O por último, puede que dos *tracks* no sean la misma persona y por tanto se descarte su emparejamiento debido a la baja puntuación. En cualquier caso, cuando el proceso *matchDB* finaliza la búsqueda de parejas (*matching*), realiza una inserción en la tabla "Pairing" indicando para cada *track* de salida su correspondiente *track* de entrada (si lo hubiese encontrado) y la correspondiente actualización en el campo "processed" en la tabla "Event" de los *tracks* correspondientes, que indica si el *track* ha sido procesado y con que resultado. Más adelante en la sección 2.4 se detallan todas las tablas empleadas. Y por último, al igual que *kheightDB*, el proceso *matchDB* dispone de dos ficheros de configuración en formato XML[3], uno que indica parámetros de configuración de la base de datos y otro parámetros filtrado de los *tracks* que se pretenden emparejar.

A continuación se detalla el código de los dos procesos mencionados y que suponen el núcleo del sistema y que tratan directamente las imágenes y la información de los *tracks*.

2.3.1. KheightDB

Como ya se ha dicho anteriormente, el proceso recoge las imágenes de la cámara Kinect, las procesa y extrae toda información sobre lo que sucede en el escenario (entra o sale una persona). El lenguaje de programación en el cual están implementadas todo el código es C++ estándar y emplea librerías de tratamiento de imagen como OpenCV[6] y CImg[11]. También se hace uso de otras librerías para el conector C++ de a MySQL[5]. Además, con el fin de hacer un seguimiento más preciso del sistema, se hace uso de las librería de syslog[8], que es un estándar de ficheros estructurados de notificaciones integrado con sistemas operativos BSD[2].

A continuación se detalla las partes código implementadas y se explican su funcionalidad dentro del sistema con el fin de mostrar el papel que desempeña cada una.

Ficheros de configuración

En el cuadro 2.1 se puede leer con detalle las acciones del código de la función encargada de cargar los parámetros de configuración procedentes del fichero "kheight_config.xml". Los argumentos `distance2XYZ`, `heightMap` y `personTracker` son instancias de clases implementadas en el trabajo del GPIV y están explicadas en [14]. Las dos primeras contienen información relativa a las características del escenario o más bien, la zona de la imagen,

```
Distance2XYZ distance2XYZ;
HeightMap heightMap;
PersonTracker personTracker;

/*** ReidentDB. Implement new version of hMapConfig function with
    new param ***/
int rr = hMapConfigDB("kheight_config.xml", distance2XYZ, heightMap
    ,
     personTracker, camId);
if (rr < 0) {
  std::cerr << "Error reading kheight_config.xml\n";
  exit(-1);
}
```

Cuadro 2.1: Función para cargar el fichero "kheight_config.xml"

en la cual tendrán relevancia cualquier suceso que pueda darse.

El tercer argumento, `personTracker`, contiene requisitos que deben cumplir los *tracks* o personas que se detectan en la zona de la imagen definida como escenario. Por ejemplo, un *track* debe cumplir un tiempo mínimo en el escenario y recorrer lo que se considere una distancia suficiente como para determinar que una persona ha entrado o salido del comercio. Esta información la posee el *Tracker* que es quien decide si un *track* es válido o descartado cuando este finaliza su aparición en el escenario.

Por último, el argumento `camId` es un entero y hace referencia al identificador asignado a la cámara que está registrando. Es un valor determinado por el cliente y su finalidad es simplemente tener registrado en la base de datos que cámara en cuestión ha grabado un *track*. Puede ser de ayuda en el caso de haber más de una cámara grabando al mismo tiempo.

- `distance2XYZ`: Clase definida en "distance2XYZ.h"
- `heightMap`: Clase definida en "heightMap.h"
- `personTracker`: Clase definida en "person_tracker.h"
- `camId`: Es un entero que hace referencia a un identificador de cámara.

La función `hMapConfigDB` se implementa en "xmlconfig.cpp" y se encarga de recorrer el fichero *.xml* para extraer los valores de configuración y validarlos. Como se observa en el cuadro 2.2, la función busca primeramente los nodos padre y luego lee y valida los nodos hijos que contienen los valores. En el punto 2.6 se explica con detalle a que hace referencia cada parámetro.

Los mismo sucede con el fichero de configuración "sqlserver_config.xml". Esta vez tan solo se pasa un objeto de la clase `SqlServer` que contiene los parámetros de conexión a la base de datos que contiene las tablas correspondientes, tales como el nombre de *host*, la base de datos (*db*) , usuario (*user*) y contraseña (*passwd*). La función descrita en el cuadro 2.3, recorre los nodos y valida que estén los parámetros necesarios para poder realizar la conexión. La clase `SqlServer` es implementada para este proyecto y se encuentra en el fichero "SqlServer.cpp" tal y como se muestra en el cuadro 2.4.

```
node = GetSon(sensor_node, (char*) "camera_extrinsics");
if (node == NULL) {
  std::cerr << "Missing " << "camera_extrinsics" << "\n";
  return -2;
}

node_level2 = GetSon(node, (char*) "cameraHeight_metres");
if (node_level2 == NULL) {
  std::cerr << "Missing " << "cameraHeight_metres" << "\n";
  return -2;
}
buf = (char*) xmlNodeGetContent(node_level2);
float camHeightMetres = atof(buf);
if (camHeightMetres < 0) {
  std::cerr << "Error: Negative cameraHeight_metres value\n";
  return -2;
}

node_level2 = GetSon(node, (char*) "matrix_filename");
if (node_level2 == NULL) {
  std::cerr << "Missing " << "matrix_filename" << "\n";
  return -2;
}
```

Cuadro 2.2: Función "hMapConfigDB"

```
SqlServer myserver;
rr = loadSqlServerConfig("sqlserver_config.xml", myserver);
if (rr < 0) {
  std::cerr << " sqlServer parameters error\n";
  exit(-1);
}
```

Cuadro 2.3: Carga de "sqlserver_config.xml"

```
class SqlServer {
private:
  std::string host;
  std::string db;
  std::string user;
  std::string passwd;

public:
  SqlServer();
  virtual ~SqlServer();

  void setHost(std::string _h);
  void setDatabase(std::string _db);
  void setUser(std::string _u);
  void setPasswd(std::string _pw);

  std::string getHost();
  std::string getDatabase();
  std::string getUser();
  std::string getPasswd();

};
```

Cuadro 2.4: Implementación de la clase *SqlServer*

Carga del fichero *crossline.dlm*

Seguidamente se carga lo que definimos *crossline* o línea de cruce que define la línea imaginaria que determina cuando una persona o *track* está dentro o fuera del comercio. Para entrar o salir del comercio, la persona o *track* debe cruzar la *crossline*. En caso de que un *track* a lo largo de su trayectoria no cruzase la *crossline* este es descartado. En el cuadro 2.5 se muestra como se lee un fichero **.dlm* que contiene unicamente dos líneas en texto plano con la información de las coordenas del puntó máximo y mínimo de las coordenadas X e Y separados por una coma. Estos valores determinan dos puntos en el espacio vectorial *local* (ver 2.1). La lectura del fichero "crossline.dlm" se realiza con la función de la librería *CImg* que simplifica a una línea poder acceder de forma directa al contenido. Posteriormente se realiza los cálculos de los coeficientes A, B, y C de la ecuación de la recta que define si un *track* a cruzado la *línea*.

Arranque del demonio de video (Video Daemon)

A partir de este momento, el proceso *kheightDB* inicia el *Video Daemon* que se define como un memoria virtual compartida donde se depositan las imágenes por parte de la Kinect desde que se inicia el proceso "captura_openni". Cada imagen se procesada para realizar el seguimiento de aquellos *tracks* que están abiertos o en *vida* y se guardan o rechazan aquellos que *mueren* o finalizan.

El subproceso que determina si un *track* ha finalizado o no, es el *PersonTracker* ya visto en 2.3.1. A su vez, para cada *track* existe un objeto de la clase `TrackFeature` que contiene la información relativa del mismo y que está asociada a una persona. Por tanto, el *Video Daemon* realiza un seguimiento de los *tracks* en escena y determina si guarda o

```
CImg<float> CrossLine;
CrossLine.load_dlm("crossline.dlm");

std::stringstream cline;
cline << "Crossline is define from p2( " << CrossLine(2) << ", "
    << CrossLine(3) << " ) to p1( " << CrossLine(0) << ", "
    << CrossLine(1) << " )";
std::cout << cline.str() << std::endl;

std::string s_cline(cline.str());
syslog(LOG_INFO, "%s", s_cline.c_str());

/*** Calculate A,B,C coeficents ***/
coef_A = CrossLine(1) - CrossLine(3);
coef_B = CrossLine(2) - CrossLine(0);
coef_C = CrossLine(0) * CrossLine(3) - CrossLine(2) * CrossLine
   (1);
```

Cuadro 2.5: Carga del fichero "crossline.dlm" utilizando la función "load_dlm" de CImg

descarta aquellas que finalizan. La declaración de la clase `PersonFeature` se encuentra en "track_feature.h". Por cada iteración del *Video Daemon*, si un *track* finaliza se hace una llamada a la función "TrackFeatureCallback" (ver cuadro 2.6) que determina si el *track* es una entrada o salida (*dirección*), si ha recorrido al menos la distancia mínima requerida y si ha cruzado la línea definida como *crossline*. Con dicha información se determina si el *track* es válido o se descarta. Si el *track* no es válido y se descarta, por motivos de *debug*, el *track* se guarda en un directorio de *tracks* descartados '../tracks/discards' pero no se realiza la inserción en la tabla "Event" de la base de datos. En caso de ser un *track* válido, se guarda dentro del directorio '../tracks/*id*', donde *id* corresponde al número de *track* que se asignó durante su aparición, y en este caso si se realiza la inserción en la tabla "Event".

```
void TrackFeatureCallback(const TrackFeature & tF, int id,
    bool save_cloud_dense = false);
```

Cuadro 2.6: Declaración de la función "TrackFeatureCallback"

Dentro del directorio '../tracks/*id* se guardan una serie de ficheros que sirven para identificar o comparar dicho *track* con otros. En el directorio se encuentran los siguientes ficheros que contienen información del *track*:

- "keyImage.jpg": Se guarda el *frame* más representativo de toda la trayectoria del *track*
- "t_count_time.dlm": Es un fichero de texto plano con los pesos de cada *frame* para el cálculo del *bodyprint*
- "t_frames.dlm": Contiene el número de *frames* registrados del *track*
- "t_mean_time.tif": Es el *bodyprint* en cuestión y se utiliza para comparar con otros *tracks*

```
float p1 = (coef_A * xini + coef_B * yini + coef_C);
float p2 = (coef_A * xend + coef_B * yend + coef_C);
float dx = xend - xini;
float dy = yend - yini;
float PathLength = (dx * dx + dy * dy); // dist between
    points

minPathLength = minPathLength * minPathLength; // avoid sqrt
    () operator

if (PathLength < minPathLength)
    return;

if (p1 < 0 && p2 > 0) {
    // Save tracks on disks
    tF.save(id, save_cloud_dense);
    syslog(LOG_INFO, "Track Inwards %d", id);
    direction = true;

} else if (p1 > 0 && p2 < 0) {
    // Save tracks on disks
    tF.save(id, save_cloud_dense);
    syslog(LOG_INFO, "Track Outwards %d", id);
    direction = false;

} else {
    syslog(LOG_NOTICE, "Track %d NOT valid", id);
    tF.save_discard(id, save_cloud_dense);
    return;
}
```

Cuadro 2.7: Validación de un *track*

```
sql::Driver * driver = sql::mysql::get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(host, user,
    passwd));
std::auto_ptr<sql::Statement> stmt(con->createStatement());
```

Cuadro 2.8: Inserción en la tabla "Event"

```
stringstream sql;

sql.str("");
sql << "INSERT INTO";
sql << " Reident.Event"; /* DataBase and Table */
sql << " VALUES (NULL"; /* Auto increment value */
sql << ", " << id; /* track */
sql << ", " << camId; /* camId */
sql << ", curdate()"; /* date */
sql << ", " << localfps; /* fps */
sql << ", " << durationF; /* duration in frames */
sql << ", curtime()"; /* endTime */
sql << ", " << direction; /* direction */
sql << ", 0"; /* processed */
sql << ");"; /* End of the COMMAND */
```

Cuadro 2.9: Construcción de la variable de tipo stream

- "trajectory.dlm": Contiene las coordenas XYZ de la trayectoria del *track* durante su registro

Inserción en la tabla "Event"

Tras considerarse que el *track* es válido se realiza la inserción en la tabla "Event" que contiene la información de todos los *tracks* que se han registrando durante el funcionamiento del proceso *kheightDB*. El primer paso es conectarse a la base de datos como se muestra en el cuadro 2.8. Seguidamente, se construye una variable de tipo `stringstream` que conforma la sentencia que realiza la entrada a la base de datos con todos los campos del *track* cumplimentados debidamente (ver cuadro 2.9). Y por último, se ejecuta la instrucción. En caso de que la inserción en la base de datos devuelva un error o que el número de filas insertadas en la tabla fuera nulo, el bloque `try-catch` del código 2.10 recoge la excepción y detendría el proceso mostrando información reportada por la base de datos.

2.3.2. matchDB

A diferencia del proceso *kheightDB* no se ha partido de un código fuente ya existente el cual se ha modificado para adaptarlo a las nuevas características del sistema pero si se han reutilizado librerías y funciones ya implementadas en [14] para manejar los *tracks* y poder compararlos. La función del proceso *matchDB* es *emparejar* los *tracks* de salida con sus correspondientes *tracks* de entrada, al mismo tiempo que se hacen inserciones en una tabla "Pairing" y actualizar el campo *processed* de la tabla "Event" de los *tracks* que se procesan.

```
  } catch (sql::SQLException &e) {
    //
    //         MySQL Connector/C++ throws three different exceptions:
    //
    //         - sql::MethodNotImplementedException (derived from sql
        ::SQLException)
    //         - sql::InvalidArgumentException (derived from sql::
        SQLException)
    //         - sql::SQLException (derived from std::runtime_error)
    //
    syslog(LOG_ERR, "# ERR: SQLException");
    syslog(LOG_ERR, "# ERR: %s", e.what());
    syslog(LOG_ERR, "# MySQL error code: %d", e.getErrorCode());
//     syslog(LOG_ERR, "# SQLState: %s", e.getSQLState());

    exit(-1);
  }
```

Cuadro 2.10: Captura de la excepción de MySQL

Ficheros de configuración

Al igual que *kheightDB* se cargan dos ficheros de configuración en formato XML. El fichero "sqlserver_config.xml" es el mismo que utiliza el proceso anterior. El fichero "matchDB_config.xml" contiene parámetros relativos al método de comparación que se debe utilizar, fechas y tiempos de los *tracks* que se quieren procesar, un listado de las cámaras a tener en cuenta, el tiempo máximo de estancia en el comercio y el umbral de puntuación. Puede leerse con detalle el contenido del fichero en 2.6.

Para función encargada de leer el fichero de configuración, se ha implementado una clase llamada `Query` destinada a contener los requisitos que deben cumplir los *tracks* de la tabla "Event" (ver cuadro 2.12). Otra clase de interés en este proceso es `MatchFeature` que contiene información de *emparejamientos* que se realizan y que procederán a guardarse tanto en disco como en la base de datos (ver cuadro 2.13). La clase `SqlServer` ya ha sido explicada en el 2.3.1 en la parte correspondiente a ficheros de configuración.

Carga de *tracks* y comparación

Seguidamente se entra en un bloque `try-catch` donde en primer lugar, se realiza la conexión a la base de datos. En este caso, se crean vectores de tipo `auto_ptr` propios de C++ llamados "res_outs" y "res_ins", que contendrán los *tracks* seleccionados de salida y entrada respectivamente (ver cuadro 2.14). Con el fin de mantener un código más claro y limpio, se ha implementado funciones que devuelven un `string` con la consulta a la tabla que se desea realizar. En este caso, para extraer todos los *tracks* de salida que según el fichero "matchDB_config.xml" se han de intentar *emparejar*, se hace una llamada a la función "Event_OutTracks", a la cual se le pasa como argumento un objeto de tipo `Query`. Tras ejecutar la consulta se obtiene todos los *tracks* de salida ordenados en el vector "res_outs" para posteriormente rellenar los campos del objeto "match" de tipo `MatchFeature` y seleccionar los posibles *tracks* de entrada candidatos para el *emparejamiento* (ver cuadro 2.15).

Una vez se tienen los *tracks* de entrada candidatos para una salida concreta, se procede

```
SqlServer myserver;
Query myquery;
MatchFeature match;

int r = 0;
r = loadMatchConfigDB("matchDB_config.xml", myquery);
if (r < 0) {
  cerr << "error in matchDB_config.xml file\n";
  return r;
}

r = loadSqlServerConfig("sqlserver_config.xml", myserver);
if (r < 0) {
  cerr << "error in sqlserver_config.xml file\n";
  return r;
}
```

Cuadro 2.11: Lectura de los ficheros de configuración

a volcar la información alojada en disco de los *tracks* a *comparar* en dos objetos llamados "tF_In" y "tF_Out" de tipo `TrackFeature`. El método "compareTrack" perteneciente a un objeto de tipo `TrackFeature`, requiere como argumentos un objeto de tipo `TrackFeature` y un entero que describe el método de comparación. Dicho método devuelve un `float` comprendido entre 0 y 1 que corresponde a la correlación entre ambos *tracks*. El valor 1 sería el resultado tras comparar un *track* consigo mismo. Durante el proceso de comparación, los distintos *tracks* de entrada para una misma salida, se guardan de forma temporal las dos puntuaciones de mayor puntuación en dos variables nombradas "score1" y "score2" que se utilizan finalmente para determinar si el *track* de salida seleccionado posee un serio candidato *track* de entrada que pueda corresponder y se deba hacer el *emparejamiento* y la composición de un fichero *.jpg* que contenga amgas "keyImage.jpg". La imagen contiene la composición de las "keyImage.jpg" de ambos *tracks* de salida y entrada, con un rotulo indicando el *score* obtenido de la comparación de ambos, sus números de *track* y ratio. El nombre del fichero contiene el número de *track* de salida, *track* de entrada con mayor *score*, *track* de entrada con el segundo *score* y el *score* perteneciente al *track* de entrada de mayor puntuación. Todas las imágenes pertenecientes a los emparejamientos se guardan dentro de un directorio llamado '../pairs'. Véase la figura 2.4.

Atendiendo al método de comparación, existen diversos métodos que se listan en el punto 2.6. El método recomendado y utilizado por defecto es "Classic" y corresponde al valor 0 (cero). Por otra parte, para que un *track* se considere válido, su *score* debe superar un umbral prefijado en el fichero de configuración "matchDB_config.xml" en el campo "threshold_score". Dicho valor es subjetivo pero se fuerza a que sea mayor de 0.5 y se recomienda no fijarlo demasiado estricto.

Inserciones en la base de datos

Tras recorrer todos los *tracks* de entrada para una misma salida y determinar si a un *track* le corresponde un emparejamiento o no, se realizan las inserciones en la base de datos que corresponde. Un *track* puede contemplar tres estados en el campos *processed* de la tabla "Event":

```
class Query {

private:
  ComparatorType method; // method comparator [0..8]
  string start_date;     // start date
  string stop_date;      // stop date
  string start;      // Start time to search
  string stop;     // Stop time to search
  int max_stay;      // max stay time in minutes
  float thresh;    // threshold to match
  vector<int> cam_in;
  vector<int> cam_out;

public:

public:
  Query();
  virtual ~Query();

  int setStartDate(std::string _date);
  int setStopDate(std::string _date);
  int setStartTime(std::string _time);
  int setStopTime(std::string _time);
  int setMethod(int _method);
  int setThreshold(float _th);
  int setCameraIn(int _cin);
  int setCameraOut(int _cout);
  int setMaxStay(int _stay);

  ComparatorType getMethod();
  std::string getStartDate();
  std::string getStopDate();
  std::string getStartTime();
  std::string getStopTime();
  float getThreshold();
  int getCameraIn(); // only returns the first element. For
     compatibility
  int getCameraOut(); // only returns the first element. For
     compatibility
  vector<int> getVectorCameraIn();
  vector<int> getVectorCameraOut();
  int getMaxStay();

};
```

Cuadro 2.12: Clase Query

```
class MatchFeature {
public:
  MatchFeature();
  virtual ~MatchFeature();

  string Out_name; // track input name (EventIn)
  string Out_time; // what time outward tracks happens
  int Out_cam; // what does camera save the outward track?
  string Out_date; // what date outwards track happen

  string Win1_name; // inward track name with (best score) (EventIn)
  string Win1_time; // what time inward track happens (in_time)
  int Win1_cam; // what does camera save the inward track?
  string Win1_date; // what date outwards track happen

  string Win2_name; // track output name (second best score)

  float score; // win1's score (get from compareTrack)
  float ratio; // score_win1 / score_win2
  string pname; // images compose's path name (url)
  int validate; // 0 no processed, 1 processed but not match, 2
     processed and match

  void listAllInfo();
  void resetAll();

};
```

Cuadro 2.13: Clase MatchFeature

```
    sql::Driver * driver = sql::mysql::get_driver_instance();
    auto_ptr<sql::Connection> con(driver->connect(myserver.getHost(),
        myserver.getUser(), myserver.getPasswd()));
    con->setSchema(myserver.getDatabase());
    auto_ptr<sql::Statement> stmt(con->createStatement());

    auto_ptr<sql::ResultSet> res_outs;
    auto_ptr<sql::ResultSet> res_ins;

    // Outwards track
    string sql_outs = Event_OutTracks(myquery);
    stmt->execute(sql_outs);
    res_outs.reset(stmt->getResultSet());
    if (!res_outs->next()) {
      cerr << "no results for this parameters. Review matchDB_config.
         xml\n";
      return -1;
    }
```

Cuadro 2.14: Conexión a la base datos

Figura 2.4: Fichero 169_133_134_score_0.715.jpg correspondiente al emparejamiento de dos *tracks*

```
match.Out_name = res_outs->getString("track");
match.Out_time = res_outs->getString("endTime");
match.Out_cam = res_outs->getInt("camId");
match.Out_date = res_outs->getString("date");
cout << "outward track " << match.Out_name << "...";

stringstream tf_path;
tf_path << "tracks/" << setfill('0') << setw(4) << match.
    Out_name;
TrackFeature tF_Out(tf_path.str());

// Outward tracks
string sql_ins = Event_InTracks(myquery, match);
stmt->execute(sql_ins);
res_ins.reset(stmt->getResultSet());
```

Cuadro 2.15: Búsqueda de los *tracks* de entrada candidatos

```
CImg<float> score; // current track
float score1 = 0.0;
float score2 = 0.0;
string taux_name("");
stringstream taux_path;

while ( res_ins->next() )
{
  taux_path.str("");
  taux_name = res_ins->getString("track");
  taux_path << "tracks/" << setfill('0') << setw(4) <<
     taux_name; // ex: "tracks/0123"

  TrackFeature tF_In(taux_path.str());
  score = tF_Out.compareTrack(tF_In, myquery.getMethod(), 0);

  if (score(0) > score2) {
    if (score(0) > score1) {
      match.Win2_name = match.Win1_name;
      score2 = score1;
      match.Win1_name = taux_name;
      score1 = score(0);
      match.Win1_time = res_ins->getString("endTime");
      match.Win1_cam = res_ins->getInt("camId");
      match.Win1_date = res_ins->getString("date");
    } else {
      match.Win2_name = taux_name;
      score2 = score(0);
    }
  }
} // end while
```

Cuadro 2.16: Comparativa de *tracks*

```
// Update Pairing DDBB
string sql;
sql = preparePairDB(match);
stmt->execute(sql);

// UPdate Event DDBB
sql = prepareEventDB(match.Out_name, match.validate);
stmt->execute(sql);
sql = prepareEventDB(match.Win1_name, match.validate);
stmt->execute(sql);

// Save compose image in ./pairs
saveValidTrack(match);
```

Cuadro 2.17: Inserciones en las tablas "Pairing" y "Event"

- "0" : El *track* no se ha procesado y por tanto está pendiente de ello
- "1" : El *track* se ha procesado pero no se ha podido emparejar con un *track* de entrada
- "2" : El *track* se ha procesado y sí se ha emparejado con un *track* de entrada

En caso de que un *track* de salida encuentre un *track* de entrada que obtenga un *score* superior al valor "threshold_score", se procede como se describe en el cuadro 2.17. Primeramente, se realiza una llamada a la función "preparePairDB" con el objeto "match" de tipo `MatchFeature` que devuelve un `string` que contiene la sentencia a mandar a la base de datos y que realiza dicha inserción. Para el caso de la tabla "Event", se procede de forma similar pero con la diferencia en este caso de no realizar una única inserción en un tabla si no de modificar el campo *processed* de ambos *tracks*, el de salida y el de entrada, a un valor de "2" como corresponde anteriormente descrito. La función encargada de formar la sentencia correcta es "prepareEventDB". Finalmente, en el caso que nos ocupa, se procede a guardar una imagen compuesta con ambas "keyImage.jpg" y sus valores de *score* dentro del directorio '../pairs'.

Si un *track* de salida no encuentra un *track* de entrada que obtuviese un *score* superior al umbral, o que entre todos los *tracks* de entrada no hubiese ninguno que cumpliese con los requisitos impuestos en el fichero de configuración "matchDB_config.xml", es decir, que la lista de *tracks* de entrada candidatos para un *track* de salida, estuviese vacía, entonces dicho *track* de salida se marca como procesado pero no emparejado (*processed=1*). La función encargada de formar la sentencia para la tabla "Event" es la misma utilizada anteriormente, "prepareEventDB", a diferencia del argumento "match.validate" que corresponde a un valor de "1" que indica que dicho *track* de salida no se ha podido emparejar con un *track* de entrada. Para este caso, no se realiza ningún guardado de datos de forma local en disco que indique nada al respecto, por lo que una vez recorridos todos los *tracks* de salida seleccionados, el proceso *matchDB* finaliza indicando el total de *tracks* emparejados con éxito y los no emparejados.

2.4. Base de datos MySQL

La gran variedad de servidores de bases de datos que existe hoy en día nos ofrece la posibilidad de elegir un servidor que sea de *código abierto*, que utilice un lenguaje definido como estándar (SQL) y además sea multiplataforma. Aunque tal vez MySQL[5] no sea el servidor más ligero que ha elegir, si es cierto que se trata del servidor de base de datos por excelencia. MySQL garantiza poder albergar toda la información de forma centralizada y gestionada por un único servicio y nos permite acceder de forma remota si es necesario. Otra de las ventajas de MySQL es disponer de un conector para C++ mediante el cual resulta más fácil gestionar las llamadas a la base de datos desde los mismos procesos que están implementados en C++ en este proyecto, e integrando el servicio de base de datos dentro del código ya existente.

Al emplear un sistema operativo Ubuntu, la instalación y puesta en marcha del servidor MySQL se realiza de forma rápida. Tan solo se deben dar permisos a los usuarios que accedan a la tabla que contiene las tablas. El nombre elegido para la base de datos (o esquema) es "Reident". Dentro de dicho esquema se han creado cuatro tablas que se describen a continuación:

- Camera: Contiene la información relativa a las cámaras que hay instaladas en el comercio
- Event: En esta tabla se realizan las inserciones de todos los *tracks* que se registran durante que el proceso *kheightDB* se ejecuta
- Host: De forma similar a la tabla Camera, contiene información relativa a los posibles PC's que puedan haber instalados en una instalación
- Pairing: Alberga información relativa a los emparejamientos realizados por el proceso *matchDB*

2.4.1. Descripción de las tablas

A continuación se procede a describir todos los campos de las tablas que se han nombrado anteriormente.

Tabla Camera

- camId: Identificador único de la cámara siempre positivo y auto-incremental.
- localId: Identificador local de la cámara. Un PC puede tener conectadas más de una cámara y cada una de ellas tener un *id* distinto al identificador global (camId).
- host: Nombre del host completo (ej: vista.dcom.upv.es) al que está conectado la cámara.
- p1x: Coordenada X de p1 perteneciente al *crossline* de dicha cámara. Los puntos p1 y p2 describen una recta que corresponde con la línea del *crossline*.
- p1y: Coordenada Y de p1 perteneciente al *crossline* de dicha cámara.
- p2x: Coordenada X de p2 perteneciente al *crossline* de dicha cámara.
- p2y: Coordenada Y de p2 perteneciente al *crossline* de dicha cámara.

- locationX: Coordenada X de localización global de la cámara.
- locationY: Coordenada Y de localización global de la cámara.
- orientation: indica si la cámara esta apuntando hacia el interior o exterior del comercio. "1", indica que está apuntando hacia el exterior y por tanto la cámara está "dentro". "0", indica que apunta hacia el interior y por tanto la cámara está "fuera".
- comment: Comentario descriptivo de la cámara.

Tabla Event

- id: Identificador único del *track* registrado, siempre positivo y auto-incremental.
- track: Número de *track* asociado en la escena y asignado por *kheightDB*.
- camId: Identificador de la cámara que registró el track.
- date: Fecha en la que se registro el *track*. El formato de la fecha debe seguir la norma ISO 8601 representado primeramente los periodos de tiempo más largos y posteriormente los más cortos (YYYYMMDD). Puede emplearse variantes con guiones o barras al estilo YYYY-MM-DD o YYYY/MM/DD y siempre respetando 4 cifras para el año (YYYY) y dos para el mes (MM) y día (DD).
- fps: Frames por segundo que el proceso *kheightDB* va tomando las imágenes desde la memoria virtual compartida. Este dato es importante para poder calcular los tiempos y velocidades de un *track* y no siempre tienen porque ser los mismos ya que por limitaciones del hardware interese reducir el número de imágenes por segundo que se procesan.
- duration: Número de frames capturados durante el periodo de vida de un *track*. Es útil para determinar la duración del mismo.
- endTime: Hora del sistema en el que se registró el *track*. La hora debe respetar el formato de HH:MM:SS utilizando siempre dos cifras para cada campo.
- direction: Dirección del *track*. "1" se considera una entrada y "0" una salida.
- processed: Indica si el *track* ha sido procesado por *matchDB*. "0" no procesado, "1" procesado pero no emparejado y "2" procesado y emparejado con otro *track*.

Tabla Host

- hostId: Identificador único del *host* o PC, siempre positivo y auto-incremental. En un sistema o instalación podrían darse más de un PC al cual se conectan 1 ó más cámaras. También podría darse el caso en el que un *host* albergase la base de datos y los ficheros de los *tracks* registrados y en el cual se realizasen los emparejamientos.
- folder: Hace referencia al directorio en el cual se encuentran los ficheros de los *tracks* registrados. En caso de compartir recursos en red para acceder de forma remota sería necesario conocer el recurso en red de los *tracks* para realizar los emparejamientos con *matchDB*.

- alias: Hace referencia al modo de en que se registran los *tracks*. Si el *host* tiene una cámara conectada con vista lateral, si la cámara enfoca hacia el interior o exterior del comercio.

Tabla Pairing

- pair_id: Identificador único del emparejamiento, siempre positivo y auto-incremental.
- CameraIn: *Id* de la cámara que registró el *track* de entrada. Corresponde al campo "camId" de la tabla "Camera".
- EventIn: Número asociado al *track* de entrada. Corresponde al campo "track" de la tabla "Event".
- CameraOut: *Id* de la cámara que registró el *track* de salida. Corresponde al campo "camId" de la tabla "Camera".
- EventOut: Número asociado al *track* de salida. Corresponde al campo "track" de la tabla "Event".
- in_time: Hora de entrada al comercio de la persona asociada al emparejamiento de *tracks*. Corresponde a la hora de registro del *track* de entrada.
- out_time: Hora de salida del comercio de la persona asociada al emparejamiento de *tracks*. Corresponde a la hora de registro del *track* de salida.
- url: Ruta parcial a la imagen compuesta por los dos *tracks* emparejados. La ruta debe comenzar siempre por 'pairs' seguido del nombre del fichero *jpg* con su extensión.
- score: Puntuación obtenida al comparar dos *tracks* de entrada y salida.
- ratio: Corresponde al cociente entre las dos mayores puntuaciones obtenidas. Tras comparar todos los *tracks* de entrada con el *track* de salida, el emparejamiento se realiza con aquel que ha obtenido la mayor puntuación (*score*) y el ratio es el resultado de *score1*/*score2* donde *score1* y *score2* son respectivamente la primera y segunda máxima puntuación obtenida tras haber comparado con todos los *tracks* de entrada que correspondían a un único *track* de salida.
- dateIn: Fecha en la cual se registró el *track* de entrada. El formato de la fecha debe seguir la norma ISO 8601 representado primeramente los periodos de tiempo más largos y posteriormente los más cortos (YYYYMMDD). Puede emplearse variantes con guiones o barras al estilo YYYY-MM-DD o YYYY/MM/DD y siempre respetando 4 cifras para el año (YYYY) y dos para el mes (MM) y día (DD).
- dateOut: Fecha en la cual se registró el *track* de salida. El formato de la fecha debe seguir la norma ISO 8601 representado primeramente los periodos de tiempo más largos y posteriormente los más cortos (YYYYMMDD). Puede emplearse variantes con guiones o barras al estilo YYYY-MM-DD o YYYY/MM/DD y siempre respetando 4 cifras para el año (YYYY) y dos para el mes (MM) y día (DD).

2.4.2. Configuración de accesos de usuario

Con el fin de preservar la seguridad y mantener un control de quienes pueden acceder al servidor, es necesario hacer una gestión de usuarios que puedan realizar inserciones y consultas a las tablas tanto de forma local como remota, según requieran las características de la instalación. El control de acceso a la base de datos requiere de un usuario y contraseña que se comparte en los ficheros de configuración del servidor, "sqlserver_config.xml", que se detalla en 2.6.

Para configurar los usuarios del servidor MySQL es necesario conectarse mediante la línea de comandos `mysql` como "root" y ejecutar las sentencias que se deseen. Para el sistema de este proyecto, tan solo se ha provisto de un usuario adicional del "root" y que posee todos los privilegios sobre el esquema "Reident". Para ello se ejecutaron las siguientes instrucciones. Usuario: 'gpiv' y contraseña 'gpivvalencia'.

```
mysql> CREATE USER 'gpiv'@'localhost'
    -> IDENTIFIED BY 'gpivvalencia';

mysql> GRANT ALL PRIVILEGES ON Reident.* TO
    -> 'gpiv'@'localhost' WITH GRANT OPTION;

mysql> CREATE USER 'gpiv'@'%'
    -> IDENTIFIED BY 'gpivvalencia';

mysql> GRANT ALL PRIVILEGES ON Reident.* TO
    -> 'gpiv'@'%' WITH GRANT OPTION;
```

Cuadro 2.18: Creación de usuarios en el servidor MySQL

2.4.3. Acceso remoto

Al diseñar el sistema se ha tenido en cuenta la posibilidad de poder acceder al *host* que alberga el servidor MySQL con el fin de consultar las tablas e incluso realizar los emparejamientos con *matchDB* desde otra estación de trabajo externa al sistema. Para permitir el acceso remoto a la base de datos se debe modificar el fichero '/etc/mysql/my.cnf' y comentar la línea `skip-external-locking` y modificar `bind-address` a la dirección 0.0.0.0.

```
...
datadir    = /var/lib/mysql
tmpdir     = /tmp
lc-messages-dir = /usr/share/mysql
# skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
# bind-address     = 127.0.0.1
bind-address     = 0.0.0.0
#
...
```

Cuadro 2.19: Modificación del fichero "my.cnf"

2.4.4. Volcado de datos

Es posible que en un momento dado se quiera volcar toda la información que se dispone en las tablas en un único fichero con el fin de importar los datos en otro servidor o simplemente realizar una copia de seguridad. A continuación, se muestra como utilizar el comando "mysqldump" para el volcado de los datos en un fichero en formato *sql*.

```
$ mysqldump -u root -p Reident > ./ddbb-backup.sql
```

En caso de querer restaurar los datos contenidos en un fichero *sql* en nuestra base de datos, por ejemplo cuando se hace la instalación del sistema y se necesita volcar el esquema de tablas inicial, se haría de la siguiente manera:

```
$ mysql -u root -p Reident < ./ddbb-backup.sql
```

2.4.5. Inserciones en las tablas Camera y Host

Cuando el sistema se instala por primera vez, es necesario realizar ciertas inserciones en las tablas que albergan información relativa a configuración de cámaras y hosts presentes en el sistema. El modo de realizar las inserciones es mediante el cliente de "mysql" y conectarse a la base de datos. Las tablas que requieren de inserciones manuales en la instalación inicial del sistema son "Camera" y "Host", que contienen información del sistema y que puesto en marcha permanecen fijas. Por tanto, a continuación pondremos ejemplos de como se realizan las inserciones para el sistema instalado en el laboratorio del GPIV.

En primer lugar se debe conectar a la base de datos mediante la herramienta de línea de comandos, "mysql".

```
$ mysql -u root -p Reident
```

Una vez logueados dentro de la línea de comandos "mysql" se realizan las inserciones. Es indistinto utilizar mayúsculas o minúsculas para los comandos SQL pero sí es relevante en los propios datos. Nota: Aquellos campos que queramos dejarlos vacíos se pueden rellenar con "NULL" y los campos que contienen valores auto-incrementales se debe utilizar la palabra reservada "DEFAULT".

```
mysql> INSERT INTO `Camera` VALUES (DEFAULT,1,'server',
    -> 0.5,5,0.5,2.5,NULL,NULL,1,'Lateral view');
```

Al ejecutar la instrucción anterior el *prompt* nos debe devolver el número de filas que han sido introducidas en la tabla y que debe coincidir el número de filas introducido. Para este ejemplo nos devolvería la siguiente línea.

```
Query OK, 1 row affected (0.05 sec)
```

En caso de querer comprobar todas las filas y campos que contiene una tabla, se debe ejecutar la siguiente instrucción (en este ejemplo mostraría la tabla "Camera").

```
mysql> SELECT * FROM Camera;
```

2.5. Software de gestión

Para ayudar a la gestión y mantenimiento del sistema se han instalado servicios que pueden ser útiles para realizar un seguimiento más cercano y acceder a la información sin necesidad de tener que desplazarse al lugar físico donde se encuentre el sistema. Tal vez, ahora pueda parecer que el sistema tenga un acceso fácil pero es probable que en versiones para clientes la instalación se haga en lugares más inaccesibles intentando minimizar el impacto visual.

2.5.1. SSH

Secure Shell[10] o más conocido como SSH es una herramienta de conectividad encriptada que reemplaza a *rlogin* y *telnet*. SSH permite acceder de forma segura al sistema para realizar tareas de gestión y consulta. Como acostumbran todos los sistemas operativos Ubuntu, la instalación del servidor SSH se realiza con una simple orden desde una consola (`sudo aptitude install openssh-server`). El comando "`scp`" puede resultar de gran ayuda cuando se quiera copiar ficheros desde nuestra máquina al sistema o viceversa. Nota: También es posible navegar entre directorios de una máquina remota y a través de SSH, si utilizamos un gestor de ventanas como Gnome o Kde. Por tanto, a través de SSH se puede acceder al sistema y realizar cualquier tarea de forma remota sin necesidad de presentarse físicamente en el comercio. Puede ser útil en caso de detectarse anomalías en el funcionamiento del sistema.

2.5.2. Samba

Samba implementa los protocolos SMB/CIFS de Windows y facilita la compatibilidad con estaciones de trabajo externas que no corran bajo un sistema GNU/Linux. A través de Samba se pueden compartir recursos e integrar el sistema en una red que opere bajo un dominio Windows. El propósito de la instalación de Samba en el sistema es compartir en red el directorio donde se almacena los *tracks* para que estén accesibles de forma remota y se pueda lanzar el proceso *matchDB* desde otra máquina.

Para compartir el directorio donde se encuentran los *tracks* se ha añadido las siguientes líneas al fichero "`/etc/samba/smb.conf`".

```
[tracksTest]
        comment = "tracks files for testing"
        path = /home/deivid/capturasTest
        read only = yes
        guest ok = no
        browseable = yesbrowseable = yes
```

Cuadro 2.20: Compartiendo un recurso con samba

```
...
openlog(NULL, LOG_CONS | LOG_PID | LOG_NDELAY | LOG_PERROR,
    LOG_LOCAL1);
syslog(LOG_INFO, "Program started...");
...
syslog(LOG_INFO, "host: %s", host.c_str());
...
syslog(LOG_INFO, "camera id: %d", camId);
...
syslog(LOG_NOTICE, "Track %d NOT valid", id);
...
syslog(LOG_ERR, "# ERR: SQLException");
syslog(LOG_ERR, "# ERR: %s", e.what());
...
```

Cuadro 2.22: Mensajes syslog

Seguidamente, se debe asignar a un usuario del sistema, que tenga permisos de acceso al recurso compartido, una contraseña asociada al protocolo *smb*. Por último, reiniciar los servicios *smbd* y *nmbd* para que el nuevo recurso quede accesible para el usuario "deivid".

```
$ sudo smbpasswd -a deivid
```

Cuadro 2.21: Asignación de contraseña para samba

2.5.3. rsyslog

Interesa poder tener un seguimiento de aquello que ha ido ocurriendo mientras el proceso *kheightDB* ha estado ejecutándose con el fin de ayudar a la depuración del programa y tener un mayor control. Así que aprovechando la estructura del sistema Ubuntu que ya dispone de un servidor de registro de procesos como *rsyslog*[9], se ha optado por integrar las librerías y hacer uso del mismo. Como ya se vio en el punto 2.3.1, tras ciertas decisiones que se tomaban siempre se notificaba mediante un mensaje *syslog* que quedaba registrado en el directorio de *logs*. Por ejemplo, cuando *kheightDB* arranca y se conecta a la base de datos, se manda un mensaje de tipo INFO indicando los parámetros del servidor MySQL que se han leído del fichero de configuración (ver cuadro 2.22). Otro caso en el cual se informa mediante *rsyslog* es cuando se detecta un *track*, bien sea de entrada o salida, ha finalizado. Se informa por consola en caso de ejecutarse y además se manda un mensaje de tipo INFO a los ficheros de registro. Los niveles de alerta utilizados en *kheightDB* son INFO, NOTICE y ERROR. NOTICE se utiliza cuando un *track* es descartado por alguna razón y ERROR únicamente en caso de suceder un problema en la conexión con la base de datos.

Para poder tener separados los ficheros de *logs* por niveles de alerta, es necesario indicarle a *rsyslog* donde los queremos alojar y con que niveles de alerta clasificarlos. Esto se consigue creando el fichero de configuración '/etc/rsyslog.d/30-kheightDB.conf' con las líneas siguientes:

```
Jul 22 13:06:01 DGA kheightDB[4744]: Program started...
Jul 22 13:06:01 DGA kheightDB[4744]: Tracks start from 0
Jul 22 13:06:23 DGA kheightDB[4777]: Program started...
Jul 22 13:06:23 DGA kheightDB[4777]: Track starts from 876
Jul 22 13:06:23 DGA kheightDB[4777]: fps = 10.000000
Jul 22 13:06:23 DGA kheightDB[4777]: host: localhost
Jul 22 13:06:23 DGA kheightDB[4777]: Database: Reident
Jul 22 13:06:23 DGA kheightDB[4777]: user: gpiv
...
```

Cuadro 2.24: Fichero kheightDB.info

```
if $programname == 'kheightDB' and $syslogfacility-text == 'local1'
   and $syslogseverity-text == 'info' then /var/log/ReidentDB/
   kheightDB.info
& ~

if $programname == 'kheightDB' and $syslogfacility-text == 'local1'
   and $syslogseverity-text == 'notice' then /var/log/ReidentDB/
   kheightDB.not
& ~

if $programname == 'kheightDB' and $syslogfacility-text == 'local1'
   and $syslogseverity-text == 'err' then /var/log/ReidentDB/
   kheightDB.err
& ~
```

Cuadro 2.23: Configuración de rsyslog

Por tanto, en los sistemas instalados podemos encontrar en la ruta '/var/log/ReidentDB' tres ficheros que indican todos los mensajes que *kheightDB* ha reportado desde sus inicios. Además, *rsyslog* se encarga de forma automática de la rotación de los ficheros para que el administrador del sistema no tenga que estar pendiente de borrar los *logs* antiguos y no provocar que el sistema se quede sin espacio en disco.

2.5.4. Scripts

Para poner en marcha el sistema es necesario tener corriendo de forma permanente dos procesos, uno encargado de capturar las imágenes con la kinect y otro que las analice y procese y vaya realizando las inserciones en la base de datos. El primero de ellos se trata de *captura_openni* que se encarga de gestionar la interfaz con la kinect mediante OpenNI y deposita las imágenes en una memoria virtual compartida. El segundo proceso es *kheightDB* que se encarga de recoger esas imágenes y procesarlas, perseguir los *tracks* y guardarlos o descartarlos según el caso. Si por algún motivo alguno de los dos procesos se detuviese el sistema se detendría y no realizaría inserciones en la tabla "Event". Por tanto, para evitar esta posible situación se han implementado cuatro *scripts* que se encargan de arrancar los procesos con sus parámetros y reiniciarlos en caso de fallo de alguno de ellos.

Script de arranque

El *script* encargado de poner en marcha ambos procesos es "lanzador.sh", este lanza a su vez "seq_captura.h" y "seq_kheight.sh" teniendo en cuenta uno directorios de ejecución previamente indicados. "seq_captura.h" lanza con un bucle infinito el proceso *captura_openni* con una serie de opciones definidas previamente. En este caso se utiliza la opción "`-Q`" para que no muestre mensajes por pantalla, la opción "`-T 3`" para que tome 1 muestra de cada 3 (diezmado) y la opción "`-t ~/C1`" que siempre se utiliza y sirve para pasar el *token* de la memoria virtual compartida.

```
#!/bin/bash

DIREXE=$HOME/capturas
DIRSCRIPTS=$HOME/scripts

echo "Lanzador..."

$DIRSCRIPTS/seq_captura.sh &
$DIRSCRIPTS/seq_kheight.sh &
```

Cuadro 2.25: Script lanzador.sh

El *script* "seq_kheight.sh" se encarga de lanzar el proceso *kheightDB* dentro del directorio indicado en la cabecera (normalmente "`~/capturas`") y utiliza la opción "`-t ~/C1`" al igual que *captura_openni* para poder acceder al mismo espacio de memoria. La opción "`-q`" indica *quiet* y sirve para que no muestro mensajes por consola.

```
#!/bin/bash

DIREXE=$HOME/capturas

echo "Kheight control Script"


while [ 1 ]; do
  cd $DIREXE
  kheightDB -t ~/C1 -q
  finvideoserver -t ~/C1
  sleep 4
done
```

Cuadro 2.26: Fichero seq_kheight.sh

```
#!/bin/bash

echo "Captura control Script.."

while [ 1 ]; do
  sleep 2
  captura_openni -t ~/C1 -Q -T 3
  finvideoserver -t ~/C1
```

```
  sleep 2
done
```

Cuadro 2.27: Fichero seq_captura.sh

Script de finalización

Para finalizar los procesos se ha implementado otro *script*, "matador.sh" que se ocupa de garantizar que ambos procesos se detengan y ninguno de ellos quede corriendo en segundo plano. También es necesario liberar la memoria virtual compartida y para ello debe ejecutar el comando "`finvideoserver`".

```
#!/bin/bash

echo "Matador..."

killall seq_captura.sh
killall seq_kheight.sh

  finvideoserver -t ~/C1
  sleep 2

killall kheightDB
killall captura_openni
```

Cuadro 2.28: Script matador.sh

2.6. Ficheros de configuración

Los ficheros de configuración nos permiten ajustar parámetros de los procesos *kheightDB* y *matchDB* sin tener que recurrir constantemente a la compilación, además, de ser un modo rápido y eficaz para ajustar las características del sistema utilizando un formato legible y estructurado. La estructura de los ficheros sigue un formato definido por el estándar XML(eXtensible Markup Language)[3] y es conocido por ser ampliamente utilizado para dar soporte a multitud de aplicaciones.

En el caso que nos ocupa se han definido tres ficheros *xml* que contienen parámetros que son cargados en el arranque de los procesos y utilizados posteriormente durante toda la ejecución de los mismos. Por tanto, para leer el contenido de los ficheros *xml* es necesario contar con librerías que realicen un barrido en búsqueda de las etiquetas que variarán en función de cada fichero *xml* tal y como se ha visto en los extractos de código mostrados el punto 2.3. En cuanto al contenido de los ficheros, decir que sigue una estructura con etiquetas que delimitan un cuerpo, que a su vez, contiene elementos que albergan el contenido. A continuación se describen los tres tipos de ficheros de configuración, indicando su cuerpo y elementos de cada uno de ellos y como afecta cada parámetro en el comportamiento de los procesos.

2.6.1. sqlserver_config.xml

Este fichero es común a los procesos *kheightDB* y *matchDB* y define los parámetros de conexión a la base de datos. Su cuerpo es la etiqueta `<sqlserver>` y sus elementos son

únicamente cuatro y se definen por si mismos. En este ejemplo, el contenido del elemento `<host>` hace referencia a "localhost" ya que el servidor MySQL está alojado en la misma máquina donde se pretende lanzar los procesos y no es necesario indicar una dirección IP o un nombre de dominio.

```
<sqlserver>
  <host> localhost </host>
  <database> Reident </database>
  <user> gpiv </user>
  <passwd> unpassword </passwd>
</sqlserver>
```

Cuadro 2.29: Fichero sqlserver_config.xml

2.6.2. kheight_config.xml

Para el proceso *kheightDB* es necesario definir una gran variedad de parámetros. El cuerpo está definido por la etiqueta `<kheight_sensor>` y dentro de la misma se define la jerarquía. En primer lugar, se encuentra la etiqueta `<fps>` (*frames* por segundo) que define la velocidad a la cual se toman las imágenes de la memoria virtual compartida. El valor por defecto de la cámara kinect es 30 fps pero como se indica en el punto 2.5.4, el *script* de arranque se lanza para que solamente se tome 1 muestra de cada 3.

El elemento `<camera_extrinsics>` envuelve una subestructura con define parámetros relacionados con la calibración e instalación de la cámara.

- `<cameraHeight_metres>`: Define la altura (en metros) a la cual está instalada la cámara kinect. Este valor se obtiene de forma automática tras la calibración de la misma. Ver punto 2.1.
- `<matrix_filename>`: Este elemento indica el fichero que contiene la matriz de rotación que se obtiene tras la calibración de la cámara. El fichero "matrix.dlm" es un fichero de texto plano que puede abrirse y leer directamente los valores de la matriz de rotación. Ver punto 2.1.
- `<cameraId>` : El identificador único de la cámara que tiene instalada el sistema en el cual se pretende lanzar el proceso *kheightDB*. Hace referencia al campo "camId" de la tabla "Camera". Ver punto 2.4.

```
<fps> 10 </fps>

<camera_extrinsics>
  <cameraHeight_metres> 2.7 </cameraHeight_metres>
  <matrix_filename> matrix.dlm </matrix_filename>
  <cameraId> 99 </cameraId>
</camera_extrinsics>
```

Cuadro 2.30: Fichero kheight_config.xml - camera_extrinsics

```
<hMap>
  <map_xmin> -3.0 </map_xmin>
  <map_xmax> 3.0 </map_xmax>
  <map_ymin> 0 </map_ymin>
  <map_ymax> 5 </map_ymax>
  <map_binSize> 0.05 </map_binSize>
  <heightBin> 0.02 </heightBin>
  <minimumImageArea> 500 </minimumImageArea>
  <maximumPersonHeight> 2 </maximumPersonHeight>
  <minimumPersonHeight> 1.10 </minimumPersonHeight>
  <minimumHeightContrast> 0.15 </minimumHeightContrast>
  <amaproi_mask> map_roi.tif </amaproi_mask>
  <ceiling_mask></ceiling_mask>
  <avoid_touching_map_border> 0 </avoid_touching_map_border>
</hMap>
```

Cuadro 2.31: Fichero kheight_config.xml - hMap

El elemento `<distance2XYZ>` hace referencia a parámetros derivados de proyectos anteriores del GPIV y que aquí no se utilizan. Los valores de `<focal>` y `<step>` no se deben alterar. En cambio, el elemento `<hMap>` si que posee elementos de gran relevancia para el comportamiento del proceso *kheightDB* de los cuales a destacar los siguientes.

- `<map_xmin>`: Coordenada X mínima en el sistema coordenado interno de la cámara kinect hasta la cual, un *track* se considera que está dentro del escenario y por tanto se tiene en cuenta.
- `<map_xmax>`: Coordenada X máxima en el sistema coordenado interno de la cámara kinect hasta la cual, un *track* se considera que está dentro del escenario y por tanto se tiene en cuenta.
- `<map_ymin>`: Coordenada Y mínima en el sistema coordenado interno de la cámara kinect hasta la cual, un *track* se considera que está dentro del escenario y por tanto se tiene en cuenta.
- `<map_ymax>`: Coordenada Y máxima en el sistema coordenado interno de la cámara kinect hasta la cual, un *track* se considera que está dentro del escenario y por tanto se tiene en cuenta.
- `<maximumPersonHeight>`: Altura máxima permitida para un *track*/persona. A veces se hace necesario restringir la altura de las personas cuando en la entrada o salida al comercio puede darse el caso que se transporten objetos o bultos que lleven a confusión al *personTracker*. Por tanto, cualquier *track* que supere la altura máxima será descartado.
- `<minimumPersonHeight>`: Altura mínima permitida para un *track*/persona. Al igual que sucede en el parámetro anterior, es necesario fijar una altura mínima para evitar falsos *tracks*.

```
<pTracker>
  <maxMissedDetectionTime> 0 </maxMissedDetectionTime>
  <minTrackLife> 0.5 </minTrackLife>
</pTracker>
```

Cuadro 2.32: Fichero kheight_config.xml - pTracker

El agente *personTracker* que corre dentro del proceso *kheightDB*, es el encargado de "perseguir" los *tracks* y en elemento `<pTracker>` podemos definir con más detalle su comportamiento.

- `<maxMissedDetectionTime>`: Durante el seguimiento de un *track* es posible que por algún motivo se pierda el seguimiento del mismo, tal vez por una oclusión. Este parámetro define el tiempo que debe transcurrir en segundos, para que tras perderse la *pista* de un *track* se considere que ha finalizado.
- `<minTrackLife>`: Por otro lado, un *track* debe tener al menos el tiempo de vida en segundos que se indica este parámetro.

2.6.3. matchDB_config.xml

Al proceso *matchDB* también le corresponde su fichero de configuración. El proceso *matchDB* es el encargado de realizar los emparejamientos y requiere de una serie de parámetros que ayudan a filtrar dentro de la base de datos aquellos *tracks* que puedan ser serios candidatos a ser pareja de un *track* de salida del comercio. Por tanto, todos los parámetros que se describen van directamente relacionados con el método a emplear para comparar *tracks* además de fechas y periodos de tiempo para buscar un *track* de entrada.

La etiqueta que define el cuerpo es `<matchDB_param>` y bajo la cual parten el resto de elementos del documento. Los siguientes elementos definen los periodos de tiempo en los cuales deben estar comprendidos los *tracks* de salida para buscar su *pareja*. Puede verse un ejemplo del documento *xml* en el cuadro de código 2.33.

- `<method>`: El *método* a emplear para realizar la comparación entre *tracks*. Dicho elemento contempla nueve posibles valores, 0-8, que corresponden con los valores del cuadro 2.1.
- `<start_date>`: Fecha a partir la cual se debe filtrar los *tracks* de salida en la tabla "Event" para intentar realizar un emparejamiento. La fecha debe seguir la norma ISO 8601 que define que los periodos de tiempo más largos deben colocarse primero. Se pueden emplear la barra (/) o el guión (-) para separar los campos pero siempre respetando 4 cifras par el año (YYYY) y dos cifras para el mes (MM) y día (DD).
- `<stop_date>`: Fecha hasta la cual se debe filtrar los *tracks* de salida que se encuentran en la tabla "Event". La fecha debe respetar el mismo formato descrito para la etiqueta `<start_date>`.

```
<start_date> 20140722 </start_date>
<stop_date> 20140722 </stop_date>

<start_time> 00:00:00 </start_time>
<stop_time> 23:59:59 </stop_time>

<camera_in> 99 </camera_in>
<camera_in> 22 </camera_in>
<camera_in> 23 </camera_in>

<camera_out> 99 </camera_out>
<camera_out> 32 </camera_out>

<max_stay_time_minutes> 20 </max_stay_time_minutes>

<threshold_score> 0.5 </threshold_score>
```

Cuadro 2.33: Fichero matchDB_config.xml

- `<start_time>`: Hora a partir la cual los *tracks* de salida de la tabla "Event" se deben filtrar para intentar realizar un emparejamiento. El formato de la hora debe respetar el formato HH:MM:SS con dos cifras para todos los campos (horas, minutos y segundos).
- `<stop_time>`: Hora hasta la cual los *tracks* de salida de la tabla "Event" se deben filtrar para intentar realizar un emparejamiento. El formato de la hora debe cumplir con los mismos requisitos del elemento `<start_time>`.
- `<camera_in>`: Corresponde al campo "camId" de la tabla "Camera". Este elemento filtra los *tracks* de entrada al comercio que han sido registrados por la cámara con "camId" que se indica en este parámetro. Es posible indicar más de un elemento `<camera_in>` en el caso de disponer más de una cámara registrando en el comercio. Puede verse como repite dicho elemento en el cuadro de código 2.33.
- `<camera_out>`: Corresponde al campo "camId" de la tabla "Camera". Este elemento filtra los *tracks* de salida del comercio han sido registrados por la cámara con "camId'. Se puede duplicar del mismo modo que el elemento `<camera_in>`.
- `<max_stay_time_minutes>`: Para cada *track* de salida, se buscan todos los *tracks* de entrada que han entrado al comercio en un periodo de tiempo anterior al *track* de entrada. Ese periodo de tiempo anterior se define en este elemento y se expresa en minutos.
- `<threshold_score>`: El valor umbral del *score* determina cual es la puntuación mínima que debe alcanzar la comparación de dos *tracks* para considerar que se corresponde a una misma persona y se realice el emparejamiento.

2.6.4. Fichero tCounter

El fichero "tCounter.dlm" es un fichero de texto plano que contiene un único número que se emplea para indicar al proceso *kheightDB* el número por el cual se debe iniciar

<method>	Descripción
0	COMPARATOR_CLASSIC: El más usado y recomendado. Si $NBO > 1$, suma todos los *bins* dentro de $NBO = 1$ y realiza la correlación. Usa la información RGB del *track*.
1	COMPARATOR_YC_VAR: Al igual que el método *classic* pero utilizando luminancia(Y) y crominancia(C) en vez de RGB.
2	COMPARATOR_CLASSIC_V2: No usado.
3	COMPARATOR_CLASSIC_V3: No usado.
4	COMPARATOR_NBO: Al igual que el método *classic* pero para $NBO > 1$.
5	COMPARATOR_RADIUS: Al igual que el método *NBO* pero utilizando información del *radio*.
6	COMPARATOR_EUCLIDEAN: Calcula la distancia euclídea utilizando la información de color RGB.
7	COMPARATOR_EUCLIDEAN_WITH_RADIUS: Al igual que el método *euclidean* pero utilizando información del *radio*.
8	COMPARATOR_GAUSSIAN:

Cuadro 2.1: Descripción de los métodos

la secuencia de registros de *tracks*. Resulta de utilidad cuando *kheightDB* se detiene por algún motivo y se desea reanudar siguiendo la misma secuencia de numeración. Si dicho fichero no existe es creado y mantenido por el propio proceso *kheightDB* y tan solo se requiere editarlo en caso de querer alterar el número por el cual arranca el proceso.

Capítulo 3

Resultados

El sistema descrito en los capítulos anteriores fue montado en una instalación de pruebas con el fin de evaluar el funcionamiento del sistema que se ha descrito en capítulos anteriores. Señalar que el objetivo de estas pruebas no era probar las funciones internas de procesado de imagen empleadas para el seguimiento de *tracks* y emparejamientos y que fueron desarrolladas por el GPIV[14], si no del sistema en su conjunto.

La colocación de las cámaras puede influir en el resultado de éxito de los emparejamientos debido a las diferencias de color que puede presentar una misma persona según el ángulo en que sea registrada. Por ejemplo, el uso de camisetas que contienen diferentes colores en el pecho y en la espalda, uso de mochilas, bolsos o chaquetas abiertas, pueden influir notablemente en el resultado del *bodyprint*. Por tanto, se contemplan diferentes opciones de colocación de la cámara que puedan mitigar dicho efecto:

- Cámara lateral, se registra el paso de personas de izquierda a derecha y viceversa
- Cámara frontal, se registra la entrada y salida desde el mismo punto y ángulo, por tanto, los *tracks* de entrada tendrán una vista de frente y los *tracks* de salida una vista de espalda
- Dos cámaras opuestas, se registra tanto los *tracks* de entrada como de salida con una vista de frente

3.1. Conjunto de datos

El escenario es el laboratorio del GPIV, un espacio transcurrido por un grupo de personas de forma diaria de las cuales un número de 10 entran y salen todos los días, a los que hay que sumar personas externas que han accedido al laboratorio durante periodos más o menos cortos de tiempo y visitantes esporádicos. Entre todos ellos se calcula un total de 40 personas distintas que pueden haber sido registradas en algún momento por el sistema.

A la entrada del laboratorio del GPIV se instaló un arco con dos cámaras kinect mirando en direcciones opuestas con el fin de registrar todas las entradas y salidas al laboratorio además de una tercera en un lateral. La disposición de las cámaras se puede ver en la figura 3.1, siendo la cámara 1 la de registro lateral y las cámaras 2 y 3 las entradas y salidas desde dos puntos de vista, frontal y trasera. Al mismo tiempo, esta prueba sirvió para comprobar la estabilidad del sistema en funcionamiento ininterrumpido incluso con cortes intermitentes del sistema debido al suministro de red eléctrica. La ventaja de

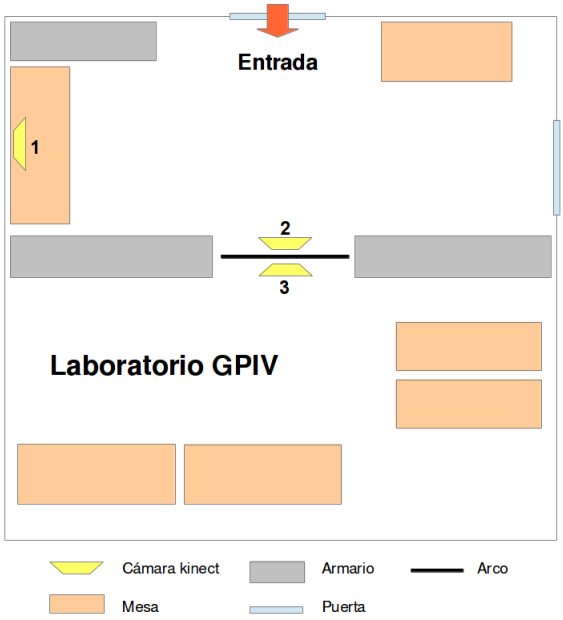

Figura 3.1: Distribución del laboratorio

```
SELECT
  AVG(e1.nTracks),
  AVG(e1.avgDur)
FROM
   (SELECT
     COUNT(id) AS nTracks,
     AVG(duration) AS avgDur,
     date
  FROM
     Event
  GROUP BY date) e1;
```

Cuadro 3.1: Sentencia SQL

este escenario es la cantidad de registros que se realizaron en la base de datos entre 3 cámaras durante 8 meses, de los cuales se repartieron en 83 días distintos, con una media de 62 *tracks* de entradas y salidas por día y un total de 5172. La media de duración de cada *track* es de 24 frames lo que supone un tiempo medio de paso de 2.4 segundos. Por tanto, se corrobora que el proceso *kheightDB* funciona y es útil para aportar información interesante sobre el tráfico de personas de acceso a cualquier espacio, pudiéndose extraer tanta información como se quiera. Los resultados globales expuestos anteriormente se han extraído realizando consultas a la base de datos empleando un lenguaje SQL, como por ejemplo, la sentencias que se muestra en el cuadro 3.1.

3.2. Test del sistema

Dentro de este conjunto de datos, cabe destacar una prueba de un grupo de 35 alumnos de la escuela al que se invitó a entrar y salir del laboratorio para simular un momento de máximo tráfico en el acceso a un comercio. Todos los alumnos entraron y salieron de

```
SELECT
    COUNT(id),
    MIN(endTime),
    MAX(endTime)
FROM
    Event
WHERE
    date = 20140219 AND camId = 1 AND track BETWEEN 608 AND 675;
```

```
+-----------+---------------+---------------+
| count(id) | min(endTime)  | max(endTime)  |
+-----------+---------------+---------------+
|        67 | 12:29:28      | 12:30:57      |
+-----------+---------------+---------------+
1 row in set (0.01 sec)
```

Cuadro 3.2: *Tracks* en la tabla Event tras la entrada y salida de un grupo de 35 personas al laboratorio

forma desordenada y despreocupada, intentando no preservar un orden ni velocidad de paso constante con el fin de simular una situación real. Por ejemplo, sabemos que el primer y último *track* son el 608 y 675, siendo registrados por la cámara 1 durante el día 19 de febrero de 2014. Haciendo una simple consulta a la base de datos, podemos saber cuantos *tracks* se registraron en dicho ensayo y cual fue la hora del primer y último *track* y que luego se utilizarán para probar el proceso *matchDB* que realiza los emparejamientos del conjunto *tracks* que contempla únicamente este ensayo. La sentencia SQL utilizada y los datos obtenidos tras la consulta a la tabla "Event", se pueden ver en el cuadro 3.2.

A continuación, conociendo la hora de inicio y fin del conjunto de *tracks* que pertenecen al grupo de 35 alumnos, podemos configurar el fichero "matchDB_config.xml" para fijar la fecha, hora y la cámara con la cual se registraron los *tracks*. El tiempo máximo de estancia en el comercio se fija en 5 minutos y el umbral del *score* en 0.8. El fichero de configuración queda de la siguiente manera:

```
<matchDB_param>
  <method> 0 </method>
  <start_date> 20140219 </start_date>
  <stop_date> 20140219 </stop_date>
  <start_time> 122928 </start_time>
  <stop_time> 123057 </stop_time>
  <camera_in> 1 </camera_in>
  <camera_out> 1 </camera_ou>
  <max_stay_time_minutes> 5 </max_stay_time_minutes>
  <threshold_score> 0.8 </threshold_score>
</matchDB_param>
```

El siguiente paso es lanzar el proceso *matchDB*. El directorio de ejecución debe ser el mismo donde se encuentre el subdirectorio '/. tracks' que contiene los *tracks* que se van a emparejar. Tras lanzar el proceso, la salida muestra para cada *track* de salida, el *track* de entrada al cual se empareja, el *track* con la segunda mejor puntuación y la puntuación final. En el caso que un *track* de salida no sea emparejado con una entrada, se informa en la línea correspondiente. Al finalizar con los emparejamientos, se hace un resumen a

Figura 3.2: Cámara lateral(1). Emparejamiento de los *tracks* 645 con 620.

modo informativo del número de *tracks* emparejados y sin emparejar. En el cuadro 3.3 puede verse la salida por consola para el conjunto de *tracks* del grupo de 35 alumnos. Un ejemplo de sentencia SQL para ver la actualización del campo "processed" en la tabla "Event" podría ser como la siguiente:

```
SELECT track, processed
FROM Event
WHERE date = 20140219 AND camId = 1 AND track BETWEEN 608 AND 675;
```

La ejecución del proceso *matchDB* genera un directorio '/. pairs/' en el cual se alojan para cada emparejamiento una imagen compuesta por las dos *keyImage* de los *tracks* emparejados y con información añadida sobre la de puntuación obtenida y el ratio. Puede verse diversos ejemplos en las figuras 3.2, 3.3, 3.4 y 3.5.

Si se desea filtrar y emparejar los *tracks* registrados por las cámaras 2 y 3 (ver figura 3.1) de cualquier otro día del laboratorio en el cual se dispone de registros de un día completo, se han de elegir los parámetros adecuados para el fichero "matchDB_config.xml", con día, hora y cámaras de entrada y salida que correspondan. En este caso, la cámara de entrada es la #2 y la de salida la #3. Se elige el día 27 de febrero de 2014 (20140227) y las horas comprendidas entre las 13:30 y las 18:00. El fichero "matchDB_config.xml" queda así:

```
<matchDB_param>
  <method> 0 </method>
  <start_date> 20140227 </start_date>
  <stop_date> 20140227 </stop_date>
  <start_time> 133000 </start_time>
  <stop_time> 180000 </stop_time>
  <camera_in> 2 </camera_in>
  <camera_out> 3 </camera_ou>
  <max_stay_time_minutes> 120 </max_stay_time_minutes>
  <threshold_score> 0.8 </threshold_score>
</matchDB_param>
```

Tras lanzar el proceso *matchDB* se pueden ver los resultados en forma de imágenes, por consola o en la misma tabla "Pairing". Ejemplos de emparejamientos del día y horas mencionados son las figuras 3.6 y 3.7. O en caso de querer utilizar los *tracks* registrados por la cámara #1, se deben modificar los campos <camera_in> y <camera_out>. Se pueden ver emparejamientos en las figuras 3.8 y 3.9.

```
$ matchDB
max time stay 5
outward track 642...match with 630 and 615=> 0.958187
outward track 643...match with 638 and 620=> 0.961728
outward track 644...match with 639 and 618=> 0.924789
outward track 645...match with 620 and 633=> 0.964343
outward track 646...match with 634 and 615=> 0.829987
outward track 648...match with 618 and 575=> 0.942049
outward track 647...match with 637 and 611=> 0.827111
outward track 649...match with 616 and 636=> 0.833613
outward track 650...match with 613 and 633=> 0.960261
outward track 651...match with 641 and 631=> 0.968781
outward track 652...no match found for track 652
outward track 654...match with 636 and 626=> 0.961003
outward track 653...match with 626 and 614=> 0.971969
outward track 655...match with 633 and 621=> 0.989267
outward track 656...match with 627 and 575=> 0.967815
outward track 657...match with 632 and 621=> 0.968161
outward track 658...match with 623 and 575=> 0.822989
outward track 659...match with 625 and 610=> 0.951544
outward track 660...match with 629 and 608=> 0.957542
outward track 661...match with 617 and 612=> 0.987916
outward track 662...match with 624 and 622=> 0.963549
outward track 663...match with 628 and 621=> 0.972858
outward track 664...match with 612 and 608=> 0.976668
outward track 665...match with 635 and 631=> 0.973801
outward track 666...match with 631 and 622=> 0.981232
outward track 667...match with 610 and 573=> 0.996717
outward track 668...match with 621 and 608=> 0.975471
outward track 669...match with 622 and 589=> 0.940401
outward track 670...match with 609 and 608=> 0.842566
outward track 671...match with 611 and 614=> 0.975219
outward track 672...match with 614 and 615=> 0.95989
outward track 673...match with 615 and 575=> 0.95878
outward track 674...no match found for track 674
outward track 675...match with 608 and 573=> 0.989436
I match 32 tracks and left 2 tracks without match
```

Cuadro 3.3: Ejecución del proceso matchDB

```
mysql> SELECT EventIn, EventOut, in_time, out_time, score, dateIn, dateOut FROM Pairing;
+---------+----------+-----------+-----------+----------+------------+------------+
| EventIn | EventOut | in_time   | out_time  | score    | dateIn     | dateOut    |
+---------+----------+-----------+-----------+----------+------------+------------+
|     630 |      642 | 12:29:55  | 12:30:22  | 0.958187 | 2014-02-19 | 2014-02-19 |
|     638 |      643 | 12:30:04  | 12:30:23  | 0.961728 | 2014-02-19 | 2014-02-19 |
|     639 |      644 | 12:30:06  | 12:30:24  | 0.924789 | 2014-02-19 | 2014-02-19 |
|     620 |      645 | 12:29:44  | 12:30:25  | 0.964343 | 2014-02-19 | 2014-02-19 |
|     634 |      646 | 12:29:59  | 12:30:26  | 0.829987 | 2014-02-19 | 2014-02-19 |
|     618 |      648 | 12:29:42  | 12:30:27  | 0.942049 | 2014-02-19 | 2014-02-19 |
|     637 |      647 | 12:30:03  | 12:30:27  | 0.827111 | 2014-02-19 | 2014-02-19 |
|     616 |      649 | 12:29:41  | 12:30:28  | 0.833613 | 2014-02-19 | 2014-02-19 |
|     613 |      650 | 12:29:37  | 12:30:29  | 0.960261 | 2014-02-19 | 2014-02-19 |
|     641 |      651 | 12:30:08  | 12:30:30  | 0.968781 | 2014-02-19 | 2014-02-19 |
|     636 |      654 | 12:30:01  | 12:30:31  | 0.961003 | 2014-02-19 | 2014-02-19 |
|     626 |      653 | 12:29:50  | 12:30:32  | 0.971969 | 2014-02-19 | 2014-02-19 |
|     633 |      655 | 12:29:58  | 12:30:34  | 0.989267 | 2014-02-19 | 2014-02-19 |
|     627 |      656 | 12:29:51  | 12:30:35  | 0.967815 | 2014-02-19 | 2014-02-19 |
|     632 |      657 | 12:29:57  | 12:30:36  | 0.968161 | 2014-02-19 | 2014-02-19 |
|     623 |      658 | 12:29:48  | 12:30:37  | 0.822989 | 2014-02-19 | 2014-02-19 |
|     625 |      659 | 12:29:49  | 12:30:38  | 0.951544 | 2014-02-19 | 2014-02-19 |
|     629 |      660 | 12:29:53  | 12:30:39  | 0.957542 | 2014-02-19 | 2014-02-19 |
|     617 |      661 | 12:29:42  | 12:30:41  | 0.987916 | 2014-02-19 | 2014-02-19 |
|     624 |      662 | 12:29:50  | 12:30:42  | 0.963549 | 2014-02-19 | 2014-02-19 |
|     628 |      663 | 12:29:52  | 12:30:43  | 0.972858 | 2014-02-19 | 2014-02-19 |
|     612 |      664 | 12:29:35  | 12:30:45  | 0.976668 | 2014-02-19 | 2014-02-19 |
|     635 |      665 | 12:30:01  | 12:30:46  | 0.973801 | 2014-02-19 | 2014-02-19 |
|     631 |      666 | 12:29:56  | 12:30:48  | 0.981232 | 2014-02-19 | 2014-02-19 |
|     610 |      667 | 12:29:32  | 12:30:49  | 0.996717 | 2014-02-19 | 2014-02-19 |
|     621 |      668 | 12:29:46  | 12:30:51  | 0.975471 | 2014-02-19 | 2014-02-19 |
|     622 |      669 | 12:29:47  | 12:30:51  | 0.940401 | 2014-02-19 | 2014-02-19 |
|     609 |      670 | 12:29:31  | 12:30:52  | 0.842566 | 2014-02-19 | 2014-02-19 |
|     611 |      671 | 12:29:33  | 12:30:53  | 0.975219 | 2014-02-19 | 2014-02-19 |
|     614 |      672 | 12:29:39  | 12:30:54  |  0.95989 | 2014-02-19 | 2014-02-19 |
|     615 |      673 | 12:29:39  | 12:30:54  |  0.95878 | 2014-02-19 | 2014-02-19 |
|     608 |      675 | 12:29:28  | 12:30:57  | 0.989436 | 2014-02-19 | 2014-02-19 |
+---------+----------+-----------+-----------+----------+------------+------------+
32 rows in set (0.00 sec)
```

Cuadro 3.4: Tabla Pairing

Figura 3.3: Cámara lateral(1). Emparejamiento de los *tracks* 669 y 622.

Figura 3.4: Cámara lateral(1). Emparejamiento de los *tracks* 671 y 611.

Figura 3.5: Cámara lateral(1). Emparejamiento de los *tracks* 672 y 614.

Figura 3.6: Cámaras frontal(2) y trasera(3). Emparejamiento de los *tracks* 437 con 303.

Figura 3.7: Cámaras frontal(2) y trasera(3). Emparejamiento de los *tracks* 422 con 277.

Figura 3.8: Cámara lateral(1). Emparejamiento de los *tracks* 1488 y 1470.

Figura 3.9: Cámara lateral(1). Emparejamiento de los *tracks* 1509 y 1506.

Capítulo 4

Conclusiones

En este proyecto se presenta un sistema basado en el procesamiento de imágenes para la reidentificación de personas que puede emplearse en comercios con diversos fines estadísticos. La disponibilidad de cámaras con sensores de profundidad a costes bajos y el potencial de un PC con un sistema operativo GNU/Linux, hacen posible que estos sistemas sean accesibles para cualquier tipo de comercio con mayor o menor recursos económicos. En cuanto al funcionamiento del sistema, se ha probado durante un periodo de tiempo de 8 meses durante los cuales ha demostrado ser estable, llegando a coleccionar un número de entradas y salidas de 5172 *tracks*, procedentes de 3 cámaras distintas y funcionando al mismo tiempo y una única base de datos. Los datos registrados han quedado accesibles mediante el servidor de la base de datos y los servicios de compartición de recursos instalados en el PC, para disponer de la información cuando resulte de interés al usuario final. Hasta el momento, en los comercios se había recurrido a sistemas de conteo y control de permanencia en el comercio que se realizaban de forma manual o suponían un obstáculo en el acceso al comercio. Por tanto, con el sistema descrito se aporta una nueva opción a métodos anteriores que resuelve dichos inconvenientes sin suponer una pérdida de la fiabilidad de la información.

El software de reidentificación ha demostrado ser robusto frente a los típicos problemas de los sistemas de registro mediante procesamiento de imagen, como son:

- Oclusiones (ver figura 4.1)
- Parecidos físicos en cuanto a tamaño y color entre personas
- Chaquetas abiertas, mochilas o bultos que puedan transportar a la entrada y/o salida

Pese a todo, el test realizado al grupo de 35 alumnos, obtuvo una tasa de acierto del 85 % y gran parte de los emparejamientos erróneos se debió a oclusiones por las condiciones de avalancha en que se llevó a cabo la entrada y salida. Se decidió proceder de tal manera con el fin de simular el peor caso para el sistema aun siendo conscientes de tratarse de una situación habitual en la mayoría de comercios durante gran parte del día.

Sería de interés para futuras versiones, contar con una interfaz con el fin de poder realizar cualquier tipo de consulta o extraer información detallada sin recurrir a conocimientos de sentencias SQL, a modo que el propio comerciante o personal encargado de gestionar la información del sistema pueda extraer aquello que le interese.

Figura 4.1: Oclusión típica

Bibliografía

[1] Apple Inc. http://www.apple.com. ©2015 Apple Inc.

[2] Berkeley Software Distribution (BSD). https://www.bsd.org/.

[3] Extensible Markup Language (XML). http://www.w3.org/XML/. ©2013-2015 W3C (MIT , ERCIM , Keio, Beihang) All Rights Reserved.

[4] Kinect for Windows. https://www.microsoft.com/en-us/kinectforwindows.

[5] MySQL. https://www.mysql.com/. ©2015, Oracle Corporation and/or its affiliates.

[6] OpenCV. http://opencv.org.

[7] OpenNI. http://structure.io/openni.

[8] RFC 5424. The Syslog Protocol. March 2009. https://tools.ietf.org/html/rfc5424.

[9] rsyslog, Rocket-fast System for Log processing. http://www.rsyslog.com/.

[10] SSH, Secure Shell. http://www.openssh.com/.

[11] The CImg Library. C++ Template Image Processing Toolkit. http://cimg.eu.

[12] Ubuntu. http://www.ubuntu.com/. ©2015 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.

[13] Xbox-360. http://www.xbox.com/en-US/xbox-360?xr=shellnav. ©2015 Microsoft.

[14] Antonio Albiol. Who is who at different cameras. People re-identification using Depth Cameras. *IEEE Xplore*, 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6334790.