Apéndice A

Gestión del proyecto

En este anexo se detalla la evolución del proyecto desde su concepción hasta la realización de los experimentos y las herramientas utilizadas para el desarrollo del mismo.

A.1. Metodología

Debido a la naturaleza de investigación del presente proyecto, no se ha utilizado ninguna metodología de diseño conocida y/o estandarizada. Sin embargo, esto no ha impedido que se realizara una división en función de las diversas tareas en las que podía ser dividido el proyecto: aprendizaje, implementación, experimentos con datasets conocidos y experimentos con dataset propio.

En la primera de ellas se realizó un estudio del estado del arte de cada uno de los ámbitos sobre los que trataba el proyecto, así como un aprendizaje de todas las técnicas y métodos desconocidos en los que dichos ámbitos se apoyaban. Todo ello nos proporcionó la capacidad de seleccionar los elementos por los que estaría formado el proyecto. La elección de cada uno de ellos vino determinada principalmente por los buenos resultados obtenidos en investigaciones anteriores.

A la hora de implementar el sistema de clasificación no existía ningún tipo de limitación en cuanto al lenguaje de programación empleado, por lo que se decidió hacerlo en C. Con esta elección se podía optimizar el uso de la memoria causado por el elevado número de información, en forma de descriptores, con el que se iba a tratar así como el acceso a la misma. Además, tal como se ha señalado anteriormente, los dos módulos utilizados para la extracción de los puntos de interés y para la clasificación estaban ya desarrollados en C, lo que significaba una ventaja a la hora de conectarlos al sistema y de cara futuras modificaciones de los mismos si es que estas fueran necesarias. De este modo, se desarrollaron dos módulos encargados de la conexión de estos dos elementos con el resto del sistema mediante

el uso de ficheros auxiliares y otro módulo encargado de todo lo referente al modelo de bolsa de palabras.

La división inicial en tareas vino determinada por el hecho de que, en un primer momento, no se concibió la posibilidad de desarrollar un algoritmo propio de agrupamiento. Conforme se avanzaba en el desarrollo fue cuando se observaron diversos problemas o inconvenientes derivados del uso de la distancia euclídea y se esbozo una primera idea que sintetizaba dicho algoritmo. Así pues, la división inicial debió de ser reestructurada para dar cabida al desarrollo e implementación de un algoritmo propio. Se realizó una primera implementación en *Matlab* con el objetivo de aplicarlo sobre ejemplos sencillos con los que poder visualizar los resultados y así poder valorar la posibilidad de incluirlo en el sistema. Debido a los buenos resultados obtenidos explicados en el capitulo **??**, se decidió realzar la implementación en C de dicho método. La inclusión de este algoritmo así como la implementación inicial fue el apartado en el que más tiempo se invirtió.

En cuanto a lo referente a los experimentos, primero se probó el sistema con datasets conocidos para poder tener una comparación respecto a su eficacia y, una vez que se estuvo seguro de ésta, se ejecuto sobre el dataset propio de cámaras vestibles. La mayor parte de estos experimentos se llevaron acabo en HERMES; el cluster de supercomputación del I3A. Debido al uso de HERMES por parte de otros usuarios y al coste temporal que ya de por si implicaba el sistema desarrollado, estos experimentos tomaron un tiempo considerable. En la figura A.1 se muestra un diagrama de Gantt a modo de resumen del tiempo invertido en cada una de las tareas.

El proyecto se ha desarrollado entre Octubre de 2013 y Junio de 2014, siendo el número total aproximado de horas dedicadas igual a 950.

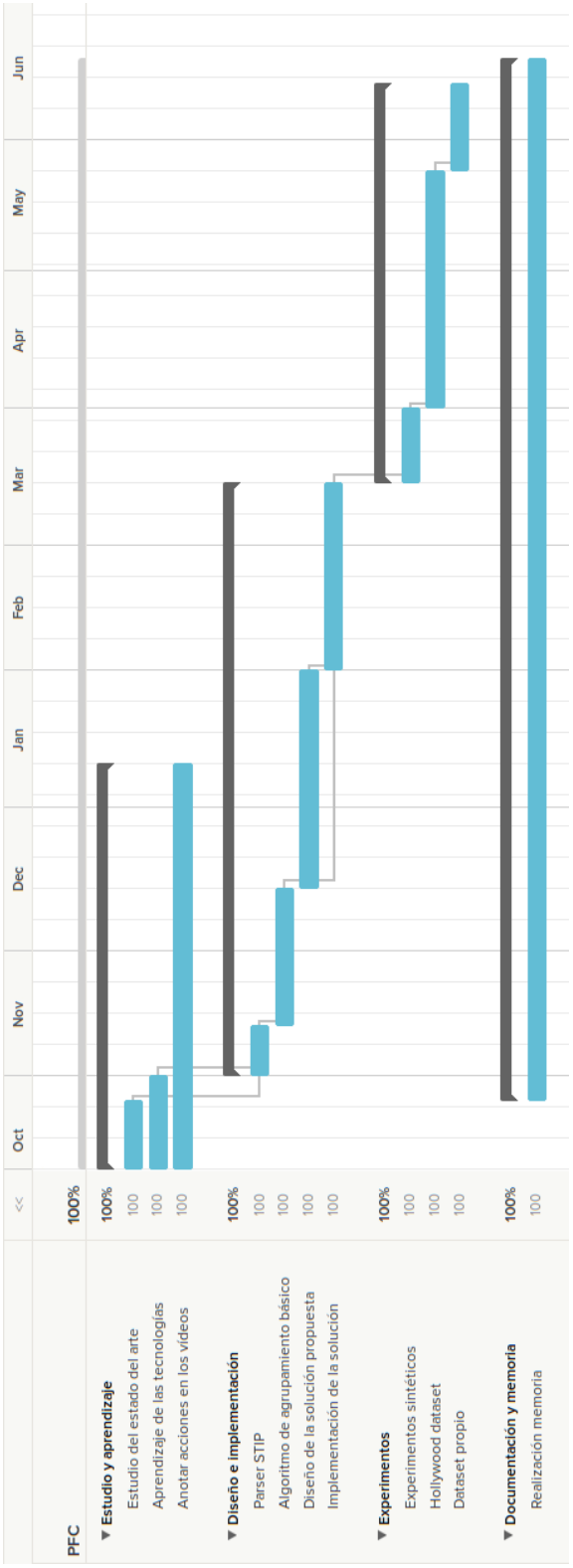

Figura A.1: Diagrama de Gantt.

A.2. Herramientas utilizadas

A continuación se detallan las herramientas que se han utilizado a lo largo del proyecto. Para la extracción de los puntos STIP de los vídeos se ha utilizado el programa disponible en la web del investigador Ivan Laptev en su versión 2.0. Dicho programa generaba errores al ser ejecutado en el cluster de supercomputación del I3A, HERMES. Se inspeccionó la ejecución de programa mediante el depurador *GDB* (The GNU Project Debugger) hasta hallar que el motivo de su mal funcionamiento se debía a una llamada $futex$ (fast user-space locking) la cual se asocia a una dirección de memoria indicada en el código del programa a la espera de que el valor contenido en ella cambie. Tal como el manual de la propia función indica, su uso genera código no portable ya que, si la dirección de memoria indicada no cae dentro del espacio de usuario, el kernel puede decidir dormir el proceso. Dado que el código fuente no era suministrado, no se pudo arreglar el problema; teniendo así que realizar la extracción de puntos de interés en un ordenador portátil.

El apartado relativo a la clasificación se ha realizado apoyándose en la librería de código abierto $LibSVM$ [**?**]. Para la programación del código en C se ha utilizado un editor de texto como es *Gedit*. Las copias de seguridad del proyecto se han realizado mediante *Dropbox* y *Ubuntu One*, mientras que para el control de versiones se ha utilizado $GitHub$.

La ejecución de el sistema de reconocimiento ha sido realizada en el citado cluster del I3A con la salvedad de que los puntos STIP fueron extraídos en el ordenador portátil y guardados en ficheros temporales de los que el programa se alimentaba. De esta forma los puntos de interés solo tuvieron que ser extraídos una sola vez ahorrando un tiempo muy valioso. Del uso de Hermes se derivaron una serie de problemas relacionados con la librería *OpenCV* así como la librería GSL (GNU Scientific Library) [**?**] utializada para invertir las matrices de covarianza que se iban calculando. Estos inconvenientes se solucionaron mediante la instalación bajo demanda de las dependencias necesarias. Adicionalmente, se utilizó $Matlab$ para la realización de los experimentos sintéticos.

En lo referente a la documentación del proyecto, la memoria se ha realizado mediante $Texmaker$, un editor de LaTeX de libre distribución. La imágenes y figuras incluidas en la memoria han sido creadas o retocadas con el programa $GIMP$ (GNU Image Manipulation Program).

Apéndice B

Optimización del cálculo de la distancia de Mahalanobis

A la hora de programar el sistema de reconocimiento de acciones presentado en este documento se pudo observar que el cálculo de la distancia de Mahalanobis de un punto a un cluster definida por la expresión **??** iba a ser la operación más repetida. La función que encapsula la búsqueda del cluster más cercano para una muestra y distancia dada es la función *FindNearestCluster* y, como se puede observar en el algoritmo 1 y en el algoritmo 2, es llamada varias veces por cada muestra tanto a la hora de crear un vocabulario como a la hora de entrenar o testear el sistema. Si la distancia seleccionada es la distancia de Mahalanobis, esta función realizara el calculo apoyándose en una función auxiliar llamada *CalculateMahalanobisDistance*. Es por ello que una optimización de la misma se antojaba como un elemento clave para disminuir el tiempo de ejecución del algoritmo.

```
...
/* elegir aleatoriamente 100000 muestras */
C ← SelectRandom(···);
...
/* construir el vocabulario */
repeat
    foreach x ∈ C do FindNearestCluster(x);
    ...
until no haya cambios ;
...
```

Algoritmo 1: *ConstructVocabulary*()

Data: X
...
foreach $x \in X$ **do** `FindNearestCluster(x);`
...

Algoritmo 2: $FetchWords()$

Se ha hecho un análisis de la eficiencia de la función $CalculateMahalanobisDistance$ comparando varias implementaciones; siendo medida cada una de ellas en micro segundos. Dicho análisis se ha realizado en equipo con procesador Intel® Core™ i7-2670QM a una frecuencia de 2.20GHz.

B.1. Implementaciones

La operación a evaluar consiste principalmente en dos multiplicaciones enmarcadas en la ya típica expresión de matriz por vector $y = A * x$. Suponiendo que A es una matriz de *double* de tamaño *sizexsize* ya inicializada y que x e y son sendos vectores de *double* de tamaño *size* ya inicializados, una aproximación clásica sería la siguiente:

```
1 for(i=0; i<size; i++)
2 {
3   y[i] = 0;
4   for(j=0; j<size; j++)
5   {
6     y[i] += A[i][j]*x[j];
7   }
8 }
```

Código B.1: Método 1

En el método anterior las direcciones de memoria son accedidas de forma ordenada y consecutiva, pero al acceder mediante índices a dichas posiciones y siendo estos índices las variables de control de los bucles, el compilador no lo puede predecir y generar un código consecuente con ello. En esta implementación se ha optado por sustituir el acceso por índices a la matriz y los vectores por referencias a memoria. Los índices solo son utilizados fuera del bucle j más usado, a la hora de almacenar los resultados calculado sobre una matriz auxiliar.

```
1 double ytemp;
2 double *Adir = &A[0][0];
3
4 for(i=0; i<size; i++)
5 {
```

```
6    double *xdir = &x[0];
7    ytemp=0;
8
9    for(j=0; j<size; j++)
10   {
11     ytemp += (*Adir++) * (*xdir++);
12   }
13   y[i] = ytemp;
14 }
```

Código B.2: Método 2

Por otro lado y dado que el vector x es necesitado por completo en cada iteración del bucle interior, se ha podido aprovechar este hecho y calcular dos posiciones del vector y en una sola iteración; reduciendo así el ancho de banda de memoria utilizado. Es decir, se ha realizado un 2-desenrollado del bucle.

```
1  double *Adir1 = &A[0][0];
2  double *Adir2 = &A[1][0];
3
4  double *ydir = &y[0];
5
6  for(i=0; i<size/2; i++)
7  {
8    double ytemp1 = 0;
9    double ytemp2 = 0;
10
11   double *xdir = &x[0];
12
13   for(j=0; j<size; j++)
14   {
15     ytemp1 += (*Adir1++) * (*xdir);
16     ytemp2 += (*Adir2++) * (*xdir);
17
18     xdir++;
19   }
20   *ydir = ytemp1;
21   ydir++;
22   *ydir = ytemp2;
23   ydir++;
24
25   Adir1 += size;
26   Adir2 += size;
27 }
```

Código B.3: Método 3

Según lo explicado en el párrafo anterior, se puede aumentar el paralelismo desenrollando más veces el bucle y realizando más multiplicaciones por iteración.

```
1  double *Adir1 = &A[0][0];
2  double *Adir2 = &A[1][0];
3  double *Adir3 = &A[2][0];
4  double *Adir4 = &A[3][0];
5
6  double *ydir = &y[0];
7
8  for(i=0; i<size/4; i++)
9  {
10   double ytemp1 = 0;
11   double ytemp2 = 0;
12   double ytemp3 = 0;
13   double ytemp4 = 0;
14
15   double *xdir = &x[0];
16
17   for(j=0; j<size; j++)
18   {
19     ytemp1 += (*Adir1++) * (*xdir);
20     ytemp2 += (*Adir2++) * (*xdir);
21     ytemp3 += (*Adir3++) * (*xdir);
22     ytemp4 += (*Adir4++) * (*xdir);
23
24     xdir++;
25   }
26   *ydir = ytemp1;
27   ydir++;
28   *ydir = ytemp2;
29   ydir++;
30   *ydir = ytemp3;
31   ydir++;
32   *ydir = ytemp4;
33   ydir++;
34
35   Adir1 += 3*size;
36   Adir2 += 3*size;
37   Adir3 += 3*size;
38   Adir4 += 3*size;
39 }
```

Código B.4: Método 4

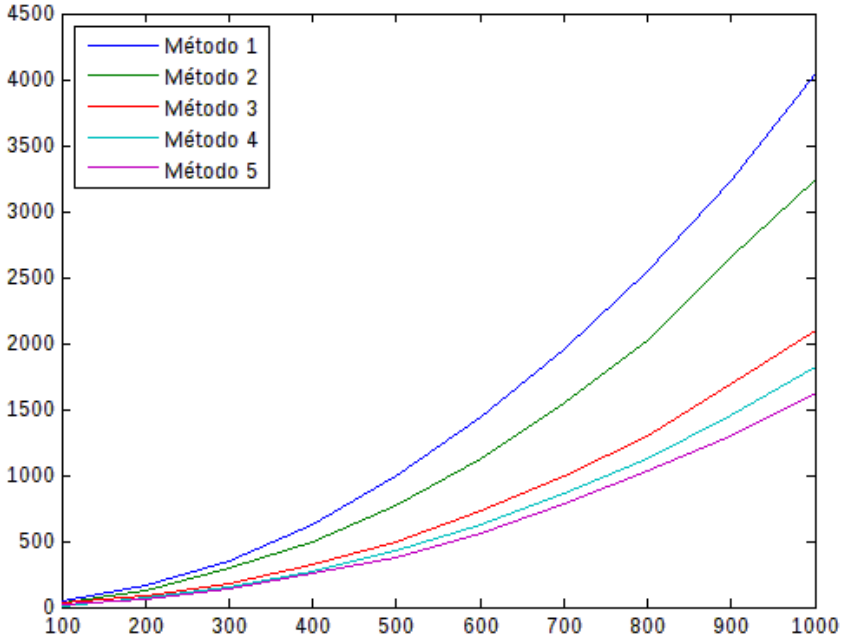

Figura B.1: Gráfica del coste temporal de cada una de las cinco implementaciones.

B.2. Análisis del coste temporal

El coste en tiempo que supone cada una de las implementaciones mencionadas en el apartado anterior puede observarse en la figura B.1, donde el eje X representa el valor de la variable *size* y el eje Y el tiempo en microsegundos. El nombrado como Método 5 representa sucesivas versiones del Método 4 en el que se han desenrollado consecutivas iteraciones del bucle.

Los métodos que utilizan acceso a la matriz y a los vectores con punteros y post-incremento de las direcciones obtienen un mejor rendimiento que el cálculo clásico expuesto en un principio. Esto es debido a que el compilador puede generar código apropiado para acceder a direcciones de memoria consecutivas y, además, se juntan instrucciones en aritmética de punto flotante con instrucciones en aritmética entera, las cuales el procesador puede realizar de forma paralela. Se podría llegar a pensar que cuantas más iteraciones se desenrollen del bucle mejor rendimiento se va a obtener; pero esto sólo ocurre para tamaños de *size* realmente grandes. En el caso que concierne al presente proyecto, en el que $size \leq 162$, el Método 4 con solo cuatro multiplicaciones por iteración presenta mejores resultados que el Método 5 en la mayoría de los casos tal y como se puede deducir de la figura B.2.

Por lo tanto se puede concluir que el Método 4 es el más apropiado para este

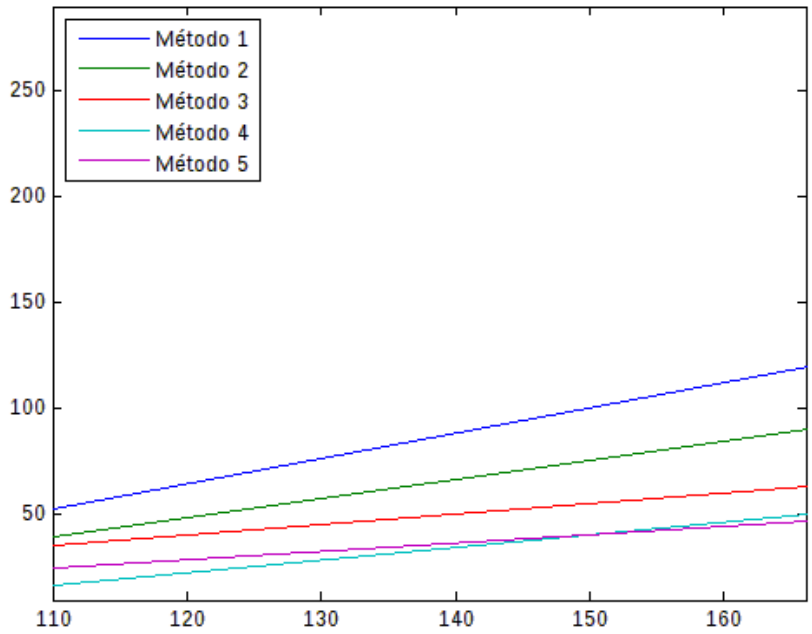

Figura B.2: Gráfica que muestra el zoom realizado en la esquina inferior izquierda de la gráfica B.1.

caso de estudio. A la hora de integrarlo en el sistema de reconocimiento, ya que los tamaños de todas las matrices a calcular no siempre serán múltiplos de cuatro, se ha añadido un código similar para tratar el resto de iteraciones. Como añadido, aprovechándose de la propiedad asociativa de la multiplicación de matrices y del hecho de que las matrices en C se almacenan en memoria por filas; a la hora de calcular la expresión **??** se ha realizado primero la multiplicación $B = {S'_{d'}}^{-1}(x'_{d'} - \mu'_{d'})$ y a continuación $D = (x'_{d'} - \mu'_{d'})^T B$, haciendo así un mejor uso de la propiedad de localidad espacial de la memoria cache del procesador y obteniendo un mejor rendimiento.