Factores para la Diferenciación de Usuarios sobre Mesas Interactivas. Estudio de un Enfoque basado en Interacciones Simultáneas

Fernando García Sanjuan

Trabajo Final de Máster

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información
Departamento de Sistemas Informáticos y Computación

Dirigido por:
Dr. Francisco Javier Jaén Martínez
Dr. Alejandro Catalá Bolós

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Julio 2013

Resumen

Diferenciar entre los usuarios que están trabajando simultáneamente entorno a una superficie interactiva podría ser beneficioso para tareas colaborativas. Así, las aplicaciones podrían soportar de manera más efectiva ciertos aspectos orientados a la territorialidad, a la gestión eficiente del espacio disponible y a la representación de los contenidos.

En este trabajo se comenta una serie de factores software que podrían usarse para realizar dicha diferenciación, y se diseña un algoritmo para la agrupación de controles según su dueño basado en uno de estos factores: la manipulación simultánea de controles. Este factor no ha sido explorado en detalle hasta este momento, y, mediante su combinación con otros factores, podría resultar relevante para el objetivo buscado. Básicamente, se basa en la idea de que los usuarios suelen manipular los elementos de interfaz con una sola mano y, por lo tanto, dos controles que están siendo manipulados al mismo tiempo es muy probable que pertenezcan a usuarios distintos.

Se han realizado tres versiones diferentes del algoritmo anterior, cada una basada en una heurística diferente. La comparación de las mismas bajo un experimento simulado muestra que la heurística que involucra más información sobre la distancia física entre los controles se comporta mejor de acuerdo a una serie de métricas definidas.

Los resultados se muestran relevantes para seguir explorando y refinando este enfoque, expandiéndolo con otros factores potenciales, y así conseguir construir un subsistema de diferenciación potente y robusto.

Palabras clave

Diferenciación de usuarios, Heurísticas, Colaboración, Superficies interactivas, Mesas táctiles.

Agradecimientos

A Raquel, por su constante ánimo y comprensión, y cuya sonrisa me alegra siempre el día.
A mi madre, por estar siempre a mi lado y animarme en todo momento a dar lo mejor de mí.
A Javi y Alejandro, por las puertas que me han abierto y siguen abriéndome, y por su paciencia, dedicación y valiosos consejos a lo largo de todo el proyecto.
A Patricia y Vicente, por todas las risas que hemos compartido durante el curso, que lo han hecho mucho más llevadero.
A mis compañeros de laboratorio, en especial a Abel y Jose Antonio por sus consejos.
Y a Joan, Adrià y Jose, que este año no nos hemos visto mucho, pero espero que en los siguientes no sea así.

Índice

Índice de Figuras

ÍNDICE DE FIGURAS

Índice de Tablas

Índice de Algoritmos

1 Introducción

1.1. Motivación

Las mesas interactivas proporcionan una forma natural de trabajar de forma colectiva, y diversos estudios han mostrado su potencial para realizar actividades colaborativas (Buisine y otros, 2012; Dillenbourg y Evans, 2011; Hornecker y otros, 2008; Morris y otros, 2006a). En este tipo de tareas, la interacción es más usuario-usuario que usuario-máquina (Streitz y otros, 2001). Esto sugiere que la interacción explícita con el ordenador debería reducirse al mínimo, de forma que los usuarios pudieran enfocar toda su atención a la tarea que tienen entre manos y a la comunicación con el resto de compañeros. Por tanto, las operaciones a realizar por la máquina deberían ser invocadas mediante interacciones implícitas de los usuarios. Dicho de otra manera, el sistema debería ser capaz de interpretar señales del entorno y, en función de éstas, ejecutar unas acciones u otras (Schmidt, 2000).

Cuando diversos individuos están trabajando conjuntamente en un mismo espacio, muchas veces se producen interferencias entre ellos. Se ha observado que cada persona suele territorializar su espacio de trabajo y usar el área que tiene enfrente casi exclusivamente (Ryall y otros, 2004; Scott y otros, 2004; Tse y otros, 2004). Además, parece ser que los usuarios prefieren manejar sus propios elementos de interfaz, hecho descrito por Morris y otros (2006b), en el que muestran que, incluso si los usuarios pueden compartir controles, eligen replicarlos. No obstante, algunas veces la naturaleza de la aplicación puede desencadenar interferencias entre las acciones de los usuarios, y los protocolos de comportamiento y buena educación no parecen ser suficientes para solventarlas (Morris y otros, 2004). Esto da lugar a que sea necesario aplicar ciertas normas explícitas o bien hacer que los propios sistemas eviten dichas interferencias.

Según Decouchant y otros (2013), estos sistemas que pretendan soportar la colaboración deberían adaptarse, por ejemplo, en términos del estado de las actividades, recursos disponibles y localización de los colaboradores. Esta interacción sensible al contexto es relevante para proveer una interfaz más orientada al usuario, considerando el contexto de uso y el contexto de las interfaces de usuario (Seo y Lee, 2013a,b). Sin embargo, aunque se involucra a varios usuarios en bastantes tareas que se realizan sobre mesas interactivas, las aplicaciones no suelen ser conscientes de quién está interactuando o qué controles pertenecen a cada usuario. Por lo tanto, no toman en consideración esta valiosa información que podría resultar útil para mitigar algunos problemas típicos que suelen aparecer en estas interfaces o proveer ciertas mejoras. Por ejemplo, un problema particular es el abarrotamiento de la superficie cuando hay diversos controles en un área limitada, o el problema comentado anteriormente de las interferencias entre usuarios. Se podrían crear estrategias para reducir estos problemas considerando quién es el propietario de cada control. Una opción, surgida a partir de los estudios sobre territorialidad anteriores, podría ser agrupar los elementos de interfaz alrededor de su propietario con el fin de obtener una mejor distribución de los mismos y dejar más espacio disponible (por ejemplo, apartando los elementos que no han sido recientemente utilizados).

En general, ser consciente de quién está usando la superficie permitiría a las interfaces de usuario adaptarse automáticamente a la forma en que los usuarios interaccionan (en términos de visualización de contenido y gestión del espacio disponible). Por tanto, es interesante estudiar la forma de diferenciar entre usuarios, ya que podría abrir nuevas consideraciones de diseño en cómo el contenido puede ser mejor presentado en estas mesas multiusuario, y, con el tiempo, mejorar la calidad de las aplicaciones diseñadas.

Diversos investigadores han especificado una serie de características deseables para los sistemas de diferenciación de usuarios (por ejemplo, Harrison y otros (2012); Zhang y otros (2012)). Un punto en el que suelen coincidir es que este tipo de sistemas deberían evitar requerir el uso de *wearables* (dispositivos sujetos al cuerpo de los usuarios) y minimizar o suprimir el hardware periférico. Por tanto, el objetivo principal que guía este trabajo es establecer las bases para el diseño futuro de una solución basada en software para conseguir que los diferentes controles de una aplicación sobre una superficie interactiva sean conscientes de quién es su usuario dueño. O, dicho de otra manera, proponer una solución al problema de la diferenciación de usuarios sin utilizar más hardware que la propia mesa.

Para conseguir esto se han realizado dos tareas. Por una parte, se han identificado ciertos factores potenciales para que, mediante su combinación, se pueda alcanzar de forma satisfactoria la diferenciación. Una vez comple-

tado esto, se ha profundizado en uno de dichos factores que no ha sido considerado previamente en ningún algoritmo de diferenciación: el uso simultáneo de diferentes controles, el cual podría combinarse con otros para incrementar la robustez del proceso de asignación de propietarios a los controles. Cuando se interactúa con elementos digitales, los usuarios suelen usar una sola mano aunque las interfaces multitáctiles permitan interacciones bimanuales (Terrenghi y otros, 2007). Por lo tanto, en algunas situaciones se podría pensar que, si dos elementos de interfaz están siendo manipulados al mismo tiempo, muy probablemente pertenecen a distintos usuarios. Además, algunas actividades colaborativas colocalizadas han mostrado una alta tendencia a la paralelización de tareas mientras los usuarios interactúan oralmente unos con otros (Morris y otros, 2010). Por tanto, podría ser posible usar este factor de forma satisfactoria para diferenciar entre usuarios, pues varios controles serían manipulados al mismo tiempo, restringiéndose así su número de propietarios posibles. Por supuesto, el uso de este factor para diferenciar usuarios solamente será efectivo si la interfaz de usuario no requiere que se realicen interacciones con ambas manos sobre controles distintos. En dicho caso, un usuario podría estar manipulando a la vez dos controles y el sistema cometería un error al determinar que dichos controles pertenecen a usuarios distintos.

1.2. Organización del Documento

El documento actual está organizado de la siguiente manera: En este capítulo, a continuación se explican los diferentes trabajos que han inspirado el nuestro.

El capítulo 2 explorará una serie de factores que se han considerado potencialmente relevantes para la tarea de la diferenciación de usuarios sobre superficies interactivas.

En el capítulo 3 profundizaremos en uno de los factores identificados previamente (la simultaneidad de manipulaciones de los elementos de la interfaz) y propondremos un algoritmo para segmentar (o particionar) los controles en una superficie de acuerdo a su propietario más plausible, basado en dicho factor. Se propondrán distintas versiones del algoritmo basadas en heurísticas, que posteriormente, en el capítulo 4, serán comparadas de acuerdo a un conjunto de funciones de bondad definidas.

Finalmente, en el capítulo 5 se mostrarán las conclusiones y consideraciones para el trabajo futuro.

Figura 1.1: Particionamiento explícito del territorio por parte de los usuarios. Tomada de Kim y otros (2009).

1.3. Trabajos Relacionados

Este trabajo ha sido influenciado por una gran variedad de estudios sobre técnicas de diferenciación de usuarios en mesas interactivas, estudios sobre territorialidad y adaptación de interfaces.

Diversos trabajos han mostrado (a veces, de forma implícita) los beneficios potenciales de diferenciar usuarios en aplicaciones de territorialidad. Hinrichs y otros (2005) presentan un modo de organizar elementos gráficos sobre mesas interactivas para hacerlos accesibles a todos los usuarios, independientemente de su localización. En su trabajo comentan que sería interesante poder agrupar los controles de un mismo usuario que no sean necesarios para la tarea actual y apartarlos parcialmente en el territorio de su dueño para conseguir más espacio de trabajo libre.

Otros trabajos, simplemente, asignan un territorio explícito a cada usuario. Kim y otros (2009) permiten al usuario dibujar su territorio (ver figura 1.1), de modo que todo lo que ocurra dentro se clasifica como suyo. Uno de los problemas de este enfoque es que el hecho de tener que dibujar el propio territorio puede resultar incómodo al usuario, puesto que le supone realizar una acción adicional que desvía su atención de su tarea principal. (Klinkhammer y otros, 2011) solventan este problema con una mesa equipada con sensores de proximidad, de forma que, cuando un usuario se aproxima, se le crea un territorio explícito para que trabaje (ver figura 1.2). Ambos enfoques requieren protocolos sociales de buen comportamiento que eviten que un usuario interfiera con el espacio de otro, lo cual ya se ha comprobado que no suele ser la solución (Morris y otros, 2004).

Figura 1.2: Territorios explícitos creados enfrente de cada usuario. Tomada de Klinkhammer y otros (2011).

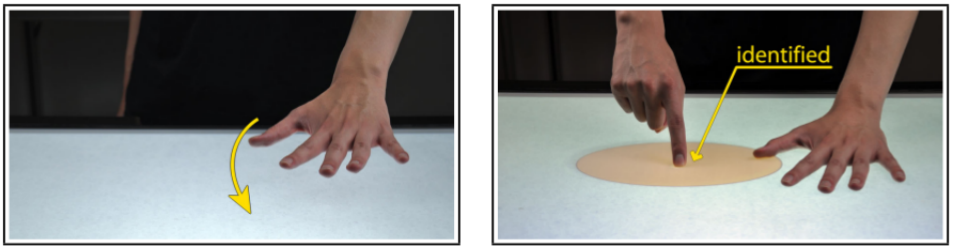


Figura 1.3: IdLenses. (a) Realizando un gesto, la lente puede ser invocada en cualquier momento. (b) Los contactos dentro de la lente están identificados, ya que cada lente pertenece a un usuario concreto. Tomada de Schmidt y otros (2010b).

Existen también otros trabajos que se benefician de conocer la identidad de los usuarios para adaptar sus controles. Schmidt y otros (2010b) presentan idLenses, un control que permite proporcionar a la aplicación información sobre un usuario determinado. Esta propuesta precisa de una fase de registro previa, donde el usuario queda vinculado permanentemente a una "lente", que puede ser invocada en cualquier momento haciendo uso de algún mecanismo de identificación externo (por ejemplo, HandsDown (Schmidt y otros, 2010a)). De esta forma, se establece una región móvil y con capacidad de ser ocultada donde se dispone de una entrada y salida personalizadas (ver figura 1.3). Esta solución, de nuevo, presenta el problema de la posible interferencia de un usuario externo con el territorio de otro.

Ryall y otros (2006) hacen uso de la identidad del usuario para crear iDwidgets, que permiten que el comportamiento o la apariencia de los controles cambie dependiendo de quién es su dueño. No obstante, la identidad del usuario se pasa como una entrada más al control, detectada a través de otras tecnologías como, por ejemplo, la mesa DiamondTouch (Dietz y Leigh, 2001), de la cual se hablará en la sección 1.3.1. En su trabajo proponen cuatro dimensiones a la hora de personalizar los controles:

1. Variación del comportamiento (botones multiusuario, interpretación semántica, comportamiento diferenciado, acceso privilegiado).

2. Variación del contenido (menús, listas).

3. Variación de la apariencia (propiedades, estética, orientación).

4. Entrada multiusuario (efecto acumulativo, entrada simultánea, secuencias de entrada modales...).

En cuanto a los trabajos previos que sí que han abordado la tarea de proveer una asignación de controles a sus dueños en superficies multitáctiles compartidas, distinguimos dos tipos de soluciones: las basadas en hardware y las basadas en software. Las soluciones hardware aumentan las mesas interactivas con dispositivos externos que permitan detectar usuarios, o bien equipan a los usuarios con dispositivos *wearables* (situados sobre su cuerpo). Los enfoques software, por el contrario, no requieren más hardware que la propia mesa interactiva, y exploran algún factor que les ayude en la tarea de la diferenciación, aunque solamente suelen centrarse en uno (comúnmente, la posición y la orientación de los contactos producidos por los dedos).

A continuación se recogen varias características deseables para un sistema de diferenciación de usuarios, identificadas por Harrison y otros (2012) y Zhang y otros (2012).

- Evitar equipar al usuario con dispositivos *wearables*.

- Compacto. Minimizar el uso de hardware externo (lo cual permite hacer la solución más independiente de la plataforma tecnológica).

- Fácil de desplegar.

- Barato.

- Bajo consumo energético.

- Rapidez. La diferenciación de usuarios no debe ser la tarea principal, por tanto, no debe requerir muchos recursos computacionales.

- Robustez (precisión), para no distraer a los usuarios de sus tareas principales.

- Escalabilidad. Es decir, permitir la identificación de un gran número de usuarios, tanto interactuando secuencialmente como simultáneamente, y también permitir que éstos se puedan situar en distintos lugares de la superficie e incluso moverse.

Teniendo en cuenta las cualidades anteriores, las soluciones software parecen ser más deseables, excepto por el hecho de que son menos robustas que las basadas en hardware.

1.3.1. Soluciones Hardware

Con respecto a soluciones hardware, la mesa DiamondTouch (Dietz y Leigh, 2001) usa un transmisor situado en la parte de abajo de la mesa que envía una señal a unas pequeñas antenas situadas justo bajo la superficie, ocupando toda ésta (ver figura 1.4). Dicha señal es única para cada antena, lo cual permite identificar pequeñas zonas en la superficie que determinan en qué posición se ha producido un contacto al tocar con un dedo o un objeto. Estas antenas forman un circuito con unos receptores situados en las sillas donde se sientan los usuarios. Este circuito se cierra al producirse un contacto permitiendo así la identificación del usuario. No obstante, el número de sillas está limitado a cuatro para producir un cierto aislamiento entre ellas y así impedir interferencias con las señales.

Harrison y otros (2012) presentan un prometedor método para la diferenciación de usuarios basado en las propiedades eléctricas del cuerpo de los usuarios, utilizando un sensor externo. Aunque su trabajo está diseñado para tabletas (ver figura 1.5), los principios usados podrían extenderse a mesas en un futuro cercano.

Dohse y otros (2008) proponen el uso de una cámara externa situada arriba de la superficie para inferir la posición de los usuarios capturando sus manos y brazos. Una solución similar es la adoptada por Martínez y otros (2011), pero utilizando el sensor de profundidad proporcionado por una Microsoft Kinect[1]. Al no diferenciar contactos basándose en su proximidad, presentan una mayor robustez frente a configuraciones donde se tengan varios

[1] `http://www.microsoft.com/en-us/kinectforwindows/`

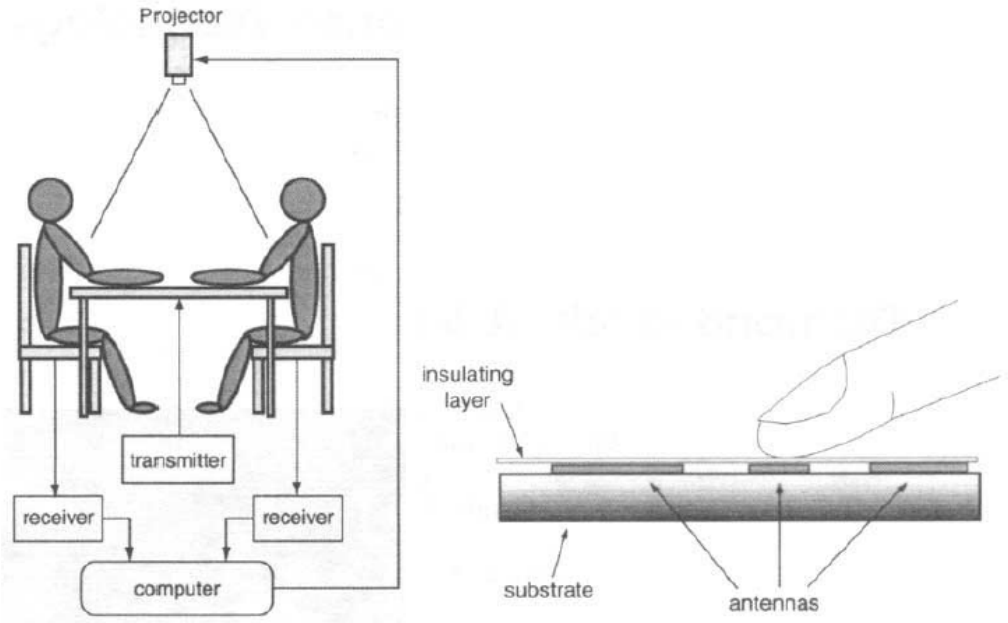


Figura 1.4: Esquema conceptual de la DiamondTouch. Tomada de Dietz y Leigh (2001).

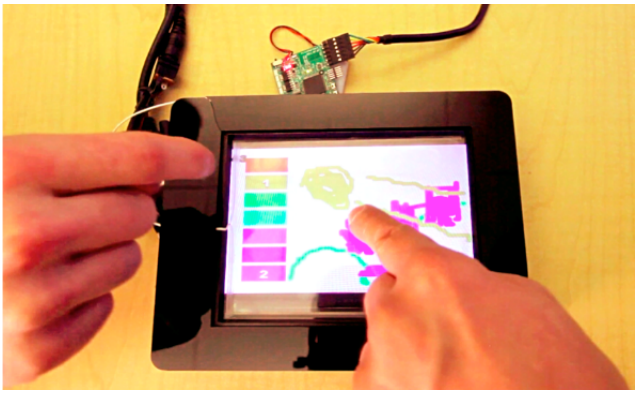


Figura 1.5: Tableta con sensor equipado en el trabajo *Capacitive Fingerprinting*. Tomada de Harrison y otros (2012).

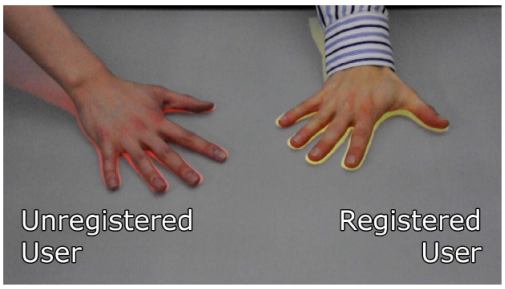


Figura 1.6: Reconocimiento de la palma de la mano en HandsDown. Tomada de Schmidt y otros (2010a).

usuarios trabajando en el mismo lado de la mesa, aunque si las manos se cruzan, las cámaras pueden confundirlas y perder precisión.

Schmidt y otros (2010a) presentan HandsDown, una forma de identificación de usuarios basada en observar el contorno de las manos. Permite realizar un seguimiento posterior de estos usuarios utilizando también una cámara externa, pero el seguimiento se pierde cuando se retira la mano de los límites de la mesa. Si esto ocurre, el usuario deberá volver a colocar la palma de la mano sobre la superficie para ser reconocido por el sistema en el que, previamente, debe de haberse registrado (ver figura 1.6). Este tipo de interacción no resulta muy natural para el usuario. En definitiva, este proyecto propone una forma distinta de realizar un registro/identificación en un sistema.

Otras propuestas usan sensores de proximidad enganchados a la mesa para detectar a los usuarios y seguirlos incluso si se mueven alrededor de la misma (Annett y otros, 2011; Klinkhammer y otros, 2011; Tănase y otros, 2008). Sin embargo, si se apartan de ésta, el sistema se olvida de ellos y son considerados como nuevos individuos al volver a aproximarse. En el trabajo sobre Medusa (ver figura 1.7) se exponen una serie de problemas asociados a usar sensores de proximidad (Annett y otros, 2011):

- Poca fiabilidad en los datos capturados; se producen muchos falsos positivos (objetos fantasma).
- Oclusiones por parte de los usuarios.
- Interferencias con los emisores y receptores de luz infrarroja de las cámaras de la mesa (para aquellas mesas que las tengan integradas).
- Necesidad de recalibrar los sensores cuando la mesa se mueve.

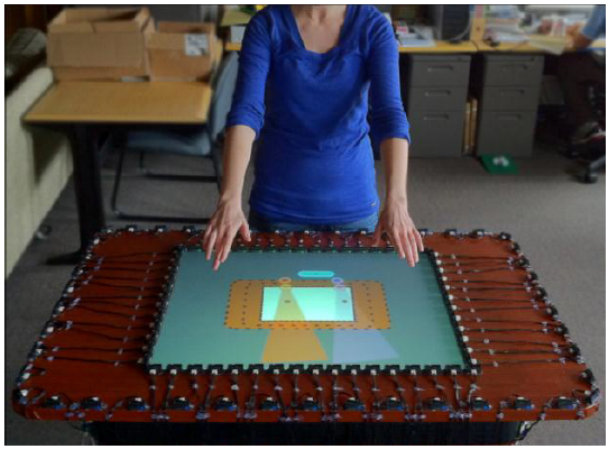


Figura 1.7: Medusa. Tomada de Annett y otros (2011).

Otros trabajos utilizan guantes con etiquetas codificadas (Marquardt y otros, 2011), tal y como se muestra en la figura 1.8-izquierda. Otros, como las IdWristbands (Meyer y Schmidt, 2010), muñequeras con LEDs transmitiendo un código identificador (figura 1.8-centro). Y también existe una solución usando un anillo que transmite pulsos de luz infrarroja llamado IR Ring (Roth y otros, 2010), el cual se puede ver en la figura 1.8-derecha. Estos tres dispositivos son reconocidos por los sistemas de visión de algunas mesas (por ejemplo, la Microsoft PixelSense 1.0[2]). Aunque la detección es bastante precisa, el uso de *wearables* puede resultar incómodo para ciertos usuarios. En estos trabajos se suele realizar una especie de registro previo donde a cada usuario se le asigna un dispositivo. De este modo, el sistema puede no sólo distinguir entre usuarios distintos, sino conocer sus identidades. A esto se le conoce con el nombre de identificación, que es una característica más ambiciosa que el objetivo perseguido en este trabajo final de máster, que sólo se centra en hacer distinción de usuarios.

Siddalinga (2010) utiliza unas balizas montadas arriba de la superficie (ver figura 1.9-izquierda) que emiten unas señales ultrasónicas recibidas por un dispositivo que lleva el usuario en su mano (figura 1.9-derecha). Mediante métodos de triangulación, se consigue localizar al usuario y, por tanto, dependiendo de su posición, es posible distinguirlo de otros.

Todas las propuestas previas requieren hardware adicional aparte de la mesa de trabajo, a veces empotrado en ésta y otras, externo. Algunas desventajas son identificadas por sus respectivos autores, tales como que los

[2]Anteriormente conocida como Microsoft Surface. `http://www.microsoft.com/en-us/pixelsense/gettingstarted10.aspx`

Figura 1.8: Soluciones hardware mediante *wearables*. A la izquierda, guantes equipados con etiquetas codificadas; al centro, idWristbands; y a la derecha, IR Ring. Ilustraciones tomadas de Marquardt y otros (2011), Meyer y Schmidt (2010), y Roth y otros (2010), respectivamente.

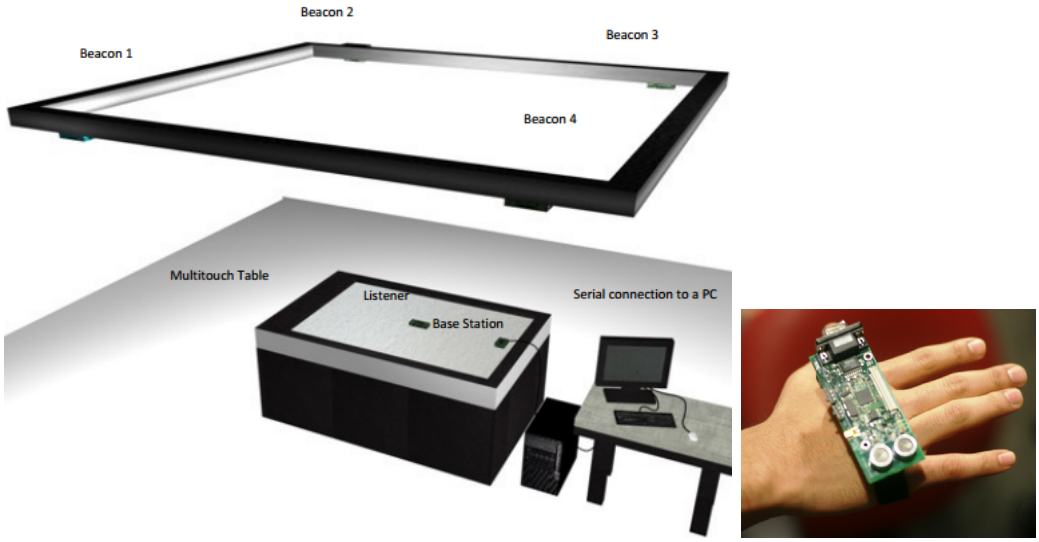


Figura 1.9: A la izquierda, balizas sobre la superficie emitiendo ondas ultrasónicas escuchadas por el receptor de la derecha. Tomada de Siddalinga (2010).

usuarios están atados a una localización fija y no se pueden mover (Dietz y Leigh, 2001; Dohse y otros, 2008); el reconocimiento no es muy preciso cuando varios usuarios están trabajando en el mismo lado de la mesa, dado que las manos si se cruzan pueden confundir a las cámaras externas o a los sensores (Annett y otros, 2011; Dohse y otros, 2008; Martínez y otros, 2011; Schmidt y otros, 2010a); y el número de usuarios que pueden diferenciar es limitado (Dietz y Leigh, 2001; Harrison y otros, 2012; Marquardt y otros, 2011; Meyer y Schmidt, 2010; Roth y otros, 2010). También, el uso de *wearables* (Marquardt y otros, 2011; Meyer y Schmidt, 2010; Roth y otros, 2010) puede resultar incómodo para algunas personas. Harrison y otros (2012) indican, además, que en su trabajo no se permite interacción multitáctil ni simultánea (sólo se puede tocar la superficie con un dedo a la vez) debido al sensor biométrico que utilizan.

En la tabla 1.1 se muestra una comparativa de las anteriores propuestas. La columna "Hardware" indica el tipo de dispositivo externo aplicado (empotrado a la mesa, externo o *wearable*). "Usuarios" marca el número de usuarios que pueden interactuar con la mesa al mismo tiempo (si no hay valor, significa que no hay límite en este aspecto). En "Movilidad" se indica el grado de movilidad que tienen los usuarios, que puede ser: "ninguna" (si los usuarios no pueden moverse de su sitio), "parcial" (si pueden moverse alrededor de la mesa pero sin apartarse de ella y sin cruzarse unos con otros) o "total" (si todo tipo de movimientos está permitido). La columna "Precisión mismo lado" indica la precisión del sistema cuando varios usuarios están trabajando hombro con hombro (el valor "regular" indica que pueden producirse errores si se cruzan las manos de los usuarios, mientras que "N/A" aparece si esta característica no está soportada en el sistema). Finalmente, la columna "Registro" nos dice si, para poder distinguir entre usuarios (no es necesario saber quienes son, sólo saber que son distintos) es necesario un paso previo antes de iniciar la interacción en el que se indique al sistema quién es cada usuario.

1.3.2. Soluciones Software

Respecto a las soluciones puramente software, Wang y otros (2009) presentan una propuesta para la diferenciación de usuarios utilizando la orientación de los dedos para inferir su posición y también con qué mano están interactuando. Sin embargo, este factor no sería muy útil si varios usuarios estuvieran trabajando en el mismo lado de la superficie y, si se movieran, el sistema no podría seguirlos. La figura 1.10 muestra cómo se puede realizar esta distinción. En esta línea, Dang y otros (2009) también presentan una forma de encontrar la mano a la que pertenecen unos contactos determina-

Trabajo	Hardware	Usuarios	Movilidad	Precisión mismo lado	Registro
DiamondTouch (Dietz y Leigh, 2001)	Empotrado + Externo	4	Ninguna	N/A	No
Capacitive Fingerprinting (Harrison y otros, 2012)	Externo	1	Total	N/A	No
Dohse y otros (2008)	Externo		Ninguna	Regular	No
Collaid (Martínez y otros, 2011)	Externo		Parcial	Regular	No
HandsDown (Schmidt y otros, 2010a)	Externo		Parcial	Regular	Sí
Medusa (Annett y otros, 2011)	Empotrado		Parcial	Regular	No
Tănase y otros (2008)	Empotrado		Parcial	Regular	No
Klinkhammer y otros (2011)	Empotrado		Parcial	Regular	No
Guantes (Marquardt y otros, 2011)	Wearable	Dispositivos disponibles	Total	Buena	No
IdWristbands (Meyer y Schmidt, 2010)	Wearable	Dispositivos disponibles	Total	Buena	No
IR Ring (Roth y otros, 2010)	Wearable	Dispositivos disponibles	Total	Buena	No
Ondas ultrasónicas (Siddalinga, 2010)	Wearable + Externo	Dispositivos disponibles	Total	Buena	No

Tabla 1.1: Comparación de las soluciones hardware para la diferenciación de usuarios.

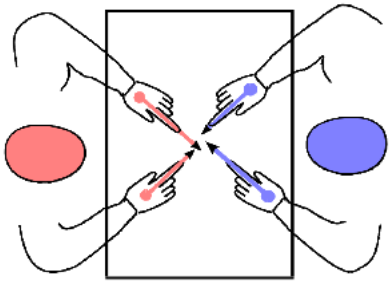


Figura 1.10: Inferencia de usuarios a partir de la orientación de sus contactos. Tomada de Wang y otros (2009).

dos, dada la posición y la orientación de los dedos, aunque en su trabajo no lo utilizan para realizar distinción de usuarios.

Zhang y otros (2012) proponen también realizar la diferenciación de usuarios usando la orientación de los dedos y predicen la posición de los usuarios utilizando un algoritmo de aprendizaje automático basado en *Support Vector Machines*. Aunque su solución proporciona una alta precisión, sólo estudian las interacciones realizadas con el dedo índice y con tres usuarios como máximo situados en posiciones específicas. En este trabajo, los autores remarcan la potencialidad de combinar su método de distinción con otros enfoques, tal vez, contemplando varios factores de distinción.

Blažica y otros (2013) presentan MTi, un método de identificación (que no diferenciación) biométrica basado en los contactos de los cinco dedos de la mano. Esta solución requiere del registro previo del usuario, en el que se le indica al sistema qué combinación de cinco contactos pertenecen a un individuo concreto. Además, presenta el problema de que no se puede realizar el seguimiento de los usuarios, es decir, que su trabajo viene a ser una alternativa a escribir una contraseña. Cuando un usuario se mueve de su posición, debe volver a identificarse para que el sistema sea consciente de su identidad.

Cuando se trabaja con controles gráficos en los que los contenidos se representan en una dirección fija (como elementos de texto), los controles están enfocados cara a su propietario (Kruger y otros, 2003; Morris y otros, 2010). Morris y otros (2010) se basan en una técnica basada en la dirección a la que apuntan estos controles para detectar usuarios. No obstante, su solución funciona únicamente con este tipo de controles. Esto no sería útil en aquellos entornos en los que se trabajara con controles 360º, los cuales tratan de mitigar el problema de acceso a los elementos desde cualquier posición alrededor de la mesa (por ejemplo, el presentado por Catala y otros (2012)). Además, su propuesta considera sólo cuatro direcciones (norte, sur, este y

Trabajo	Factor	Usuarios	Movilidad	Precisión mismo lado	Registro
Wang y otros (2009)	Posición + orientación contactos	4	Ninguna	N/A	No
See me, See you (Zhang y otros, 2012)	Posición + orientación contactos	3	Ninguna	Regular	No
MTi (Blažica y otros, 2013)	Relación entre dedos de la mano		Ninguna	Buena	Sí
Morris y otros (2010)	Orientación controles	4	Parcial	N/A	No

Tabla 1.2: Comparación de las soluciones software para la diferenciación de usuarios.

oeste) y, por lo tanto, el número de usuarios que pueden estar interactuando en la superficie al mismo tiempo, para que la distinción se realice de forma satisfactoria, está limitado a cuatro.

En la tabla 1.2 se muestra una comparativa de las anteriores propuestas de forma similar a como se ha hecho anteriormente con las soluciones hardware. La columna "Factor" indica el elemento diferenciador de usuarios considerado en la propuesta. El valor "parcial" en la columna "Movilidad" en esta tabla significa que los usuarios pueden moverse con libertad por el lado de la mesa en el que se encuentren sin que el sistema se olvide de ellos, pero no pueden cambiar de lado. Respecto a la movilidad, cabe destacar que, en todas estas soluciones el usuario puede retirarse de la mesa a voluntad siempre que luego regrese al mismo sitio que tenía inicialmente (a diferencia de algunas propuestas hardware donde el usuario no podía apartarse de la superficie sin que el sistema se olvidara de él). Otra ventaja de estos enfoques software respecto a algunos hardware es que ninguno de los primeros requiere un registro previo del usuario (la detección se realiza ad hoc).

1.4. Conclusiones

En este capítulo se ha motivado la utilidad de poder diferenciar entre los usuarios utilizando al mismo tiempo una superficie interactiva. Como se ha explicado, tener en cuenta esta información permitiría a los elementos

de interfaz adaptar su comportamiento y contenidos a sus dueños. Este tipo de adaptación permite al usuario, entre otras cosas, centrarse en la tarea principal que tiene entre manos sin tener que perder el tiempo organizando la interfaz.

Se han explorado varias soluciones existentes a este problema, y se han clasificado en dos grupos. Por una parte, están aquellos enfoques que hacen uso de sensores, cámaras u otros dispositivos externos a la propia superficie para poder realizar la diferenciación. Por otra, aquellas propuestas basadas únicamente en software que, prescindiendo de hardware adicional, pretenden hacer la distinción de usuarios basándose en algún factor que los diferencie (el cual, normalmente, viene a ser la posición y la orientación de los contactos sobre la mesa).

Según algunos autores, las propuestas basadas en software parecen ser más deseables. No obstante, el uso de un solo factor discriminador no proporciona soluciones suficientemente robustas. Esta carencia podría ser mitigada mediante la combinación de diversos factores, de forma que, si uno falla al diferenciar correctamente entre usuarios, otro pueda corregirlo. Nuestra propuesta en este trabajo consiste, primero, en identificar qué factores podrían usarse para realizar esta tarea (ver capítulo 2) y, segundo, proponer un algoritmo de diferenciación basado en un factor todavía no explorado en detalle en la literatura: la simultaneidad de interacciones (ver capítulo 3).

2

Factores para la Diferenciación de Usuarios

En este capítulo se describen una serie de factores que podrían ser útiles para realizar una distinción entre los usuarios interactuando al mismo tiempo con una superficie interactiva. Muchos de estos factores pueden ser medidos en cualquier aplicación multiusuario en la que haya controles que puedan ser movidos y rotados por la superficie, y su medición se realiza a partir de la interacción de los usuarios con éstos.

En el capítulo 1 se han comentado básicamente dos de estos factores, que ya han sido explorados por otros investigadores. Éstos son: la posición y la orientación de los contactos realizados sobre la superficie, y también la orientación de los controles en el caso de que estos tengan una dirección intrínseca (no sería válido para controles 360º). A lo largo de este capítulo se profundizará en las características de estos factores así como en cómo utilizarlos para la tarea entre manos.

2.1. Forma del Contacto

Algunas mesas, como la Microsoft PixelSense, miden las dimensiones (ancho y alto) de cada contacto que detectan sobre la superficie. Asumiendo que algunos usuarios interactúan con la punta mientras que otros usan una mayor parte de sus dedos, y también considerando que las dimensiones de los dedos difieren entre usuarios, sería posible diferenciar entre ellos mediante este factor. La figura 2.1 muestra la representación (centro) de los contactos producidos al situar dos manos sobre una Microsoft PixelSense 1.0 (izquierda). Y, a la derecha, se muestra cómo se podrían medir, sobre un contacto, las dos dimensiones comentadas anteriormente.

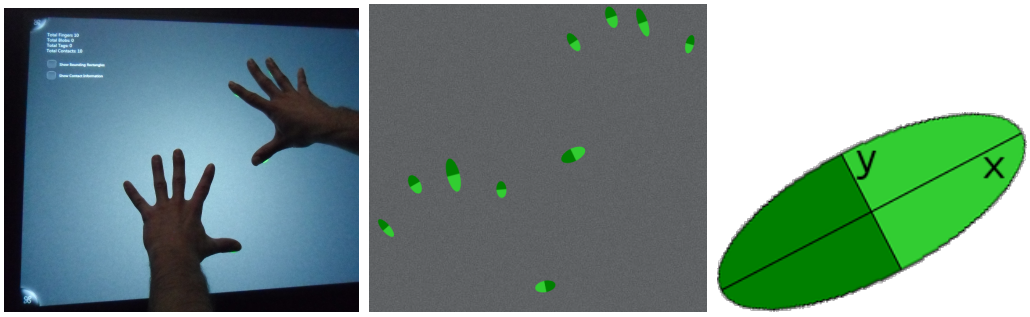


Figura 2.1: Representación de un contacto. A la izquierda, dos manos situadas sobre una Microsoft Pixelsense 1.0. Al centro, la representación de los contactos de dichas manos. A la derecha, dimensiones x e y de uno de los contactos.

2.2. Posición y Orientación del Contacto

Este factor ha sido previamente estudiado y encontrado útil para la diferenciación de usuarios. Dang y otros (2009) proponen una manera de utilizar este factor para determinar a qué mano pertenecen unos contactos determinados (ver figura 2.2). Aunque en su trabajo no realizan distinción de usuarios, utilizando la información que puede observarse en la ilustración, podría inferirse una posición aproximada del usuario al que pertenece la mano, pues las prolongaciones de los vectores que representan la orientación de los dedos parecen converger en un punto (exceptuando la del dedo pulgar). Aunque los usuarios rotasen la mano y, por tanto, dichas prolongaciones no apuntasen a la posición en la que se encuentra el usuario, aún podría utilizarse dicha información para distinguir dichos contactos de los producidos por otra persona (cuyos dedos presentarían rotaciones bastante distintas).

Wang y otros (2009) sí que explican cómo poder utilizar la información de la posición y orientación de los dedos para diferenciar entre usuarios situados en lados distintos de una mesa (como se mostraba en la figura 1.10). Aunque es complicado determinar en qué posición se encuentran exactamente, sí que se puede determinar fácilmente que dos contactos pertenecen, por ejemplo, a usuarios situados uno enfrente del otro, ya que el ángulo que formarán las orientaciones de sus respectivos dedos será cercano a 180º. Este fenómeno se puede observar en la figura 2.3-c (nótese la gran ampitud del ángulo de orientación entre ambos contactos). En contraposición, las partes a y b de la misma figura muestran que, las orientaciones de varios contactos pertenecientes a la misma persona (aunque use ambas manos), forman entre sí ángulos mucho menores.

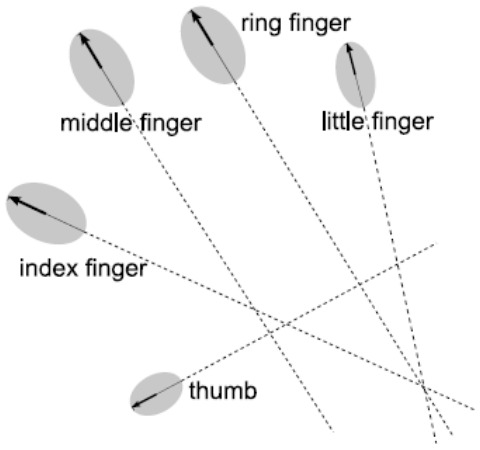


Figura 2.2: Contactos de una mano junto con sus orientaciones. Las prolongaciones de los vectores que representan la orientación indican, en cierto grado, dónde se encuentra el usuario. Tomada de Dang y otros (2009).

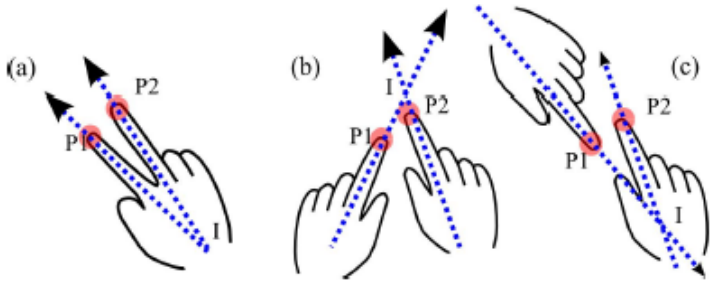


Figura 2.3: Vectores de las orientaciones de dos contactos pertenecientes a la misma mano (a), a ambas manos de un mismo usuario (b) y a las manos de dos usuarios enfrentados (c). Tomada de Wang y otros (2009).

En la propuesta de Zhang y otros (2012), se utiliza un clasificador basado en *Support Vector Machines* para conseguir la diferenciación entre tres usuarios. Para entrenar al clasificador, la superficie se divide en sesenta y cuatro celdas y, sobre cada una de ellas, va apareciendo un elemento que un usuario debe seleccionar. Dicho usuario está colocado en una posición concreta de la mesa. Por tanto, la clasificación solamente funciona en una aplicación real si los usuarios que están interactuando lo hacen desde las mismas posiciones con las que se ha entrenado al clasificador. Aunque los autores sólo realizan pruebas con tres usuarios al mismo tiempo, obtienen precisiones bastante altas. En la figura 2.4 se muestra que, con tres usuarios situados como se muestra a la izquierda, las orientaciones de los contactos no se suelen confundir. Esto se muestra a la derecha, donde, por cada una de las sesenta y cuatro celdas consideradas, se indica en triángulos las orientaciones medias de los tres usuarios junto con su desviación (representada por la anchura del triángulo), y, como se puede observar, la mayoría no se solapan. Esto lo que indica es que el factor considerado es prometedor para alcanzar el objetivo propuesto.

En todos los trabajos anteriores, sin embargo, se muestra que la información obtenida a partir de este factor no resulta suficientemente poderosa en escenarios complejos donde varios usuarios pueden interactuar en distintas posiciones entorno a la superficie y sus manos pueden solaparse. Estas debilidades podrían compensarse si se combinara este factor con algunos otros.

2.3. Relación entre Dedos de la Mano

Blažica y otros (2013) exploran este factor en su trabajo MTi, en el cual identifican a un usuario mediante ciertas relaciones entre los contactos de los cinco dedos de su mano. Su trabajo se centra en asociar estos contactos a una identidad concreta de un usuario, no simplemente en diferenciarlo de otros interactuando al mismo tiempo sobre la mesa. No obstante, la información que presentan podría también ser utilizada para este propósito. El problema de este enfoque reside en tener que utilizar los cinco dedos cada vez que se realice una interacción, lo cual puede limitar bastante el ámbito de las aplicaciones en el que este factor sea relevante. En MTi se utilizan seis tipos de relaciones entre los dedos, detallados a continuación (del a al f) y que vienen representados gráficamente en la figura 2.5 a través de ejemplos. Los tres primeros (a, b y c) son propios de estos autores, mientras que el resto (d, e y f) los toman del trabajo de Micire y otros (2011), en el que tienen como meta distinguir entre la mano derecha y la izquierda en interacciones.

(a) Distancia entre cada par de dedos (10 distancias consideradas).

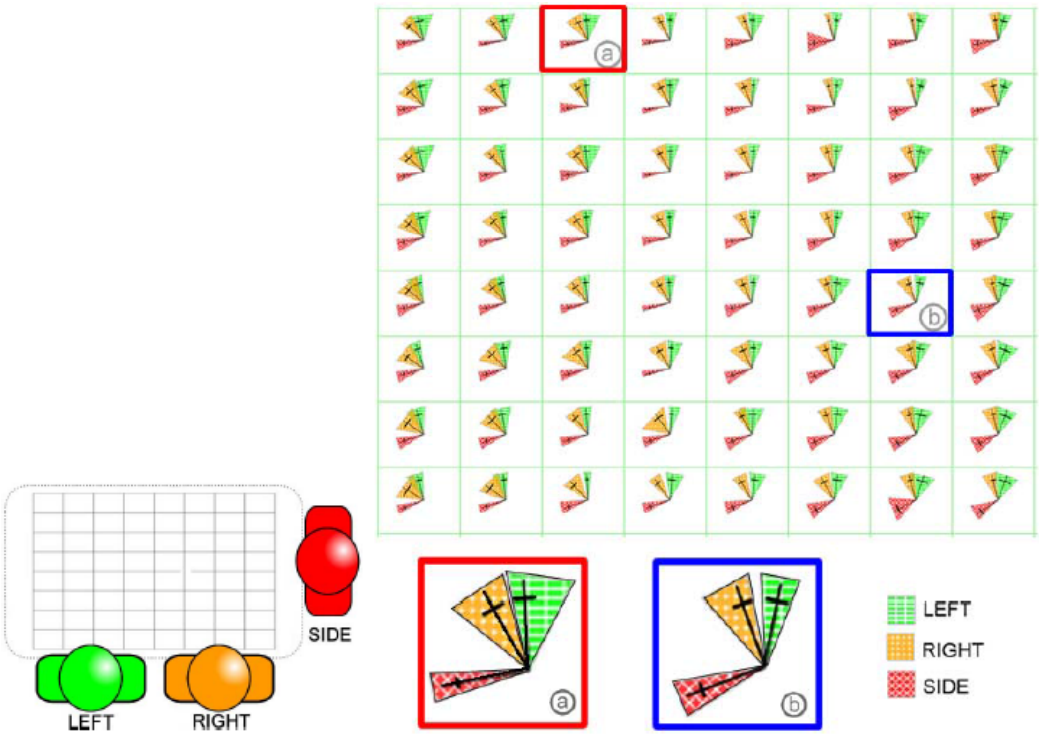


Figura 2.4: Orientaciones de contactos medidas en el trabajo "*See Me, See You*". A la izquierda, las posiciones en las que se colocan los participantes del experimento. A la derecha, la media y la desviación de las orientaciones de sus contactos. Tomada de Zhang y otros (2012).

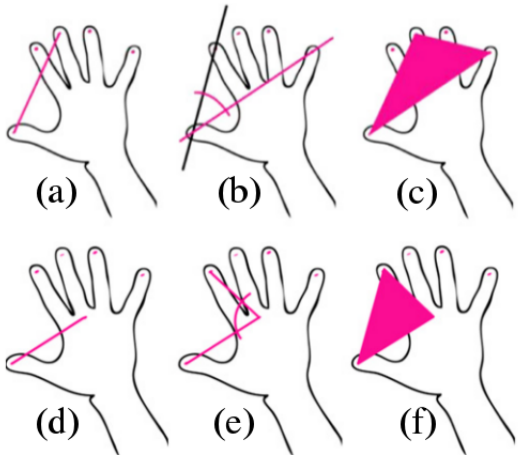


Figura 2.5: Ejemplos de cada uno de los tipos de relaciones entre los dedos consideradas en MTi. Tomada de Blažica y otros (2013).

(b) Ángulo entre la línea que une el pulgar y el índice y la línea entre el pulgar y el resto de los dedos (9 ángulos medidos).

(c) Área de las posibles figuras geométricas que surgen uniendo los dedos con líneas (10 triángulos, 5 cuadriláteros y 1 pentágono).

(d) Distancia entre el centroide de los cinco contactos y cada uno de éstos (5 distancias).

(e) Ángulo definido por dos contactos adyacentes, con el vértice en el centroide de los cinco contactos (5 ángulos).

(f) Área de los triángulos definidos por dos contactos adyacentes y el centroide de todos los cinco contactos de la mano (4 áreas).

2.4. Punto Inicial de la Manipulación

Cuando el usuario está interactuando con controles que pueden ser movidos o rotados con un gesto simple, es posible que el punto relativo al centro del control donde empieza la traslación o la rotación sea siempre el mismo (dentro de un cierto umbral) para un mismo usuario. Esto podría ser debido a la localización del usuario o a sus preferencias. Si esta asunción se probase relevante, la distinción entre usuarios podría realizarse de acuerdo a la "región" dentro de un control donde los usuarios realizan sus contactos.

Figura 2.6: Varios usuarios interactuando alrededor de una mesa interactiva con controles TangiWheel (Catala y otros, 2012).

Posiblemente, este factor por sí solo no sea capaz de distinguir entre todos los usuarios involucrados en la aplicación, pero podría establecer una adivinación preliminar que podría ser refinada considerando otro tipo de información.

2.5. Velocidad de las Manipulaciones

En contextos como en los que se enmarcan los menús de marcado de Lepinski y otros (2010) o los TangiWheel (ver figura 2.6) descritos por Catala y otros (2012), se permite libremente la rotación y reposicionamiento de los controles. En estos casos, sería razonable pensar que algunas personas llevarán a cabo dichas manipulaciones más rápidamente que otras, ya sea debido a su nivel de experiencia usando la aplicación, su vivacidad, etc. Por tanto, la velocidad de realización de estas acciones podría resultar un factor potencial a ser considerado bajo manipulaciones o gestos específicos.

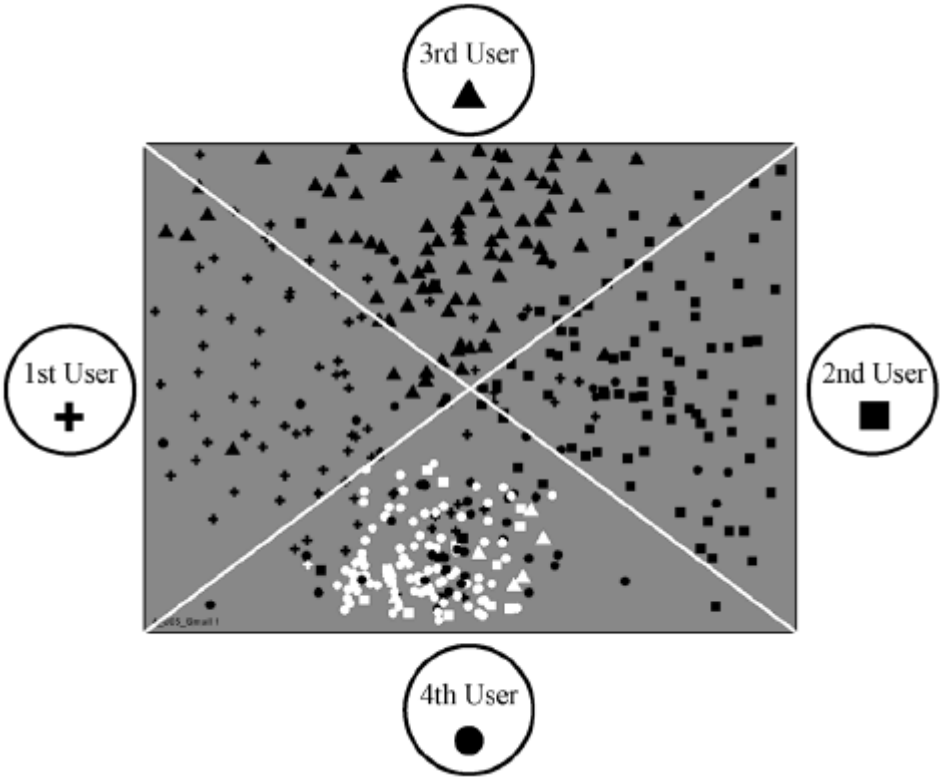


Figura 2.7: Territorialización efectuada por cuatro usuarios interactuando simultaneamente en una superficie. Tomada de Ryall y otros (2004).

2.6. Posición de los Controles

Los usuarios territorializan el espacio de trabajo disponible (Ryall y otros, 2004; Scott y otros, 2004; Tse y otros, 2004). Por tanto, se puede inferir que los elementos pertenecientes al mismo usuario estarán más cerca entre ellos que del resto de usuarios en la mesa. Así, la diferenciación entre usuarios de acuerdo a este factor podría realizarse agrupando los controles basándose en su posición, de manera que cada grupo representase al dueño de dichos controles. Si la superficie estuviese muy abarrotada, esta técnica podría fallar debido a que controles de diferentes usuarios estarían relativamente cerca. Ésta es otra motivación para combinar diversos factores.

En la figura 2.7 se puede observar cómo la mayoría de los contactos producidos por un usuario en una aplicación colaborativa sobre una superficie interactiva se producen en el territorio que implícitamente éste crea enfrente suyo.

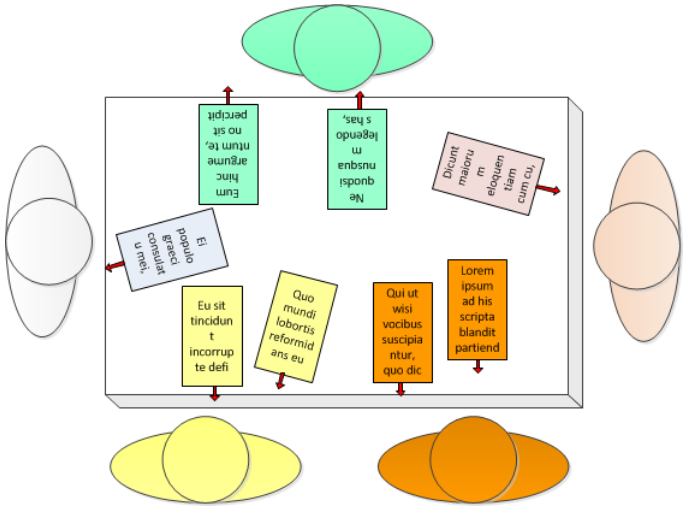


Figura 2.8: Posible asignación de controles de texto a sus dueños basándose en la dirección del texto (señalizada con flechas).

2.7. Orientación de los Controles

Si los elementos de interfaz tienen una dirección fija (por ejemplo, controles de texto), es muy probable que estén encarados a su propietario (Kruger y otros, 2003; Morris y otros, 2010). Por lo tanto, todos los controles pertenecientes a un mismo usuario apuntarán más o menos al mismo punto. De este modo, sería posible conocer cuántos usuarios diferentes están interactuando en la mesa y también inferir su localización. Morris y otros (2010) utilizan esta técnica con controles de texto para diferenciar entre cuatro usuarios situados cada uno en un lado de la mesa. Extendiendo su propuesta, se podría distinguir entre varios usuarios colocados también en el mismo lado si ésta fuera lo suficientemente grande (obteniendo un resultado como el que se muestra en el ejemplo de la figura 2.8). Incluso si este factor no pudiese realizar una distinción perfecta entre todos, se podría combinar con otros de los factores vistos en este capítulo para robustecer el sistema.

2.8. Uso Simultáneo de Diferentes Controles

Cuando los usuarios interactúan con elementos digitales, suelen utilizar una sola mano aunque las superficies multitáctiles permitan interacciones bimanuales (Terrenghi y otros, 2007). De este modo, en algunas situaciones se podría pensar que, si dos controles están siendo manipulados al mismo tiem-

po, seguramente pertenecen a usuarios distintos. Además de esto, algunas actividades colaborativas colocalizadas han mostrado una cierta tendencia a la paralelización de las tareas mientras los participantes interactúan oralmente entre ellos (Morris y otros, 2010). Por estas razones, sería posible utilizar satisfactoriamente este factor para diferenciar entre usuarios, puesto que muchos controles serían manipulados al mismo tiempo, restringiéndose así su número de dueños posibles. No obstante, este factor podría no resultar muy efectivo en aquellas interfaces en las que fuera necesario el uso simultaneo de ambas manos en controles distintos, ya que éstos serían clasificados como pertenecientes a usuarios distintos. El capítulo 3 se centra en detalle en este factor.

2.9. Conclusiones

En este capítulo se han presentado una serie de factores potenciales para permitir realizar la diferenciación de los usuarios interactuando al mismo tiempo sobre una superficie interactiva sin utilizar ningún tipo de hardware adicional. Por sí solos, es posible que estos factores no sean capaces de realizar una distinción completa entre todos los usuarios, pero sí que podrían formar grupos de usuarios distintos. Y, mediante la combinación de varios factores, estos grupos podrían desgranarse hasta conseguir, finalmente, una separación total de los participantes. También se ha visto que algunos de ellos no pueden aplicarse en todo tipo de situaciones o, simplemente, hay momentos en los que un factor determinado puede no proporcionar información suficiente o correcta.

Como trabajo futuro, sería necesario realizar experimentos con el fin de determinar si los factores considerados en este capítulo pueden contribuir de forma efectiva a la tarea de la diferenciación de usuarios. Y, si resultaran potencialmente útiles, sería interesante encontrar la manera de realizar la combinación de los mismos, tal y como se ha comentado anteriormente. También sería conveniente averiguar en qué tipo de aplicaciones y/o bajo qué condiciones un factor resulta más relevante y cuándo menos.

3

Manipulación Simultánea de Controles

En este capítulo se profundiza en el factor "uso simultáneo de diferentes controles" introducido en la sección 2.8, para el que se propone un algoritmo que permita al sistema diferenciar entre usuarios cuando varios de ellos estén interactuando simultáneamente con los controles de una aplicación.

3.1. Algoritmo de Diferenciación

El algoritmo diseñado tiene como objetivo segmentar el espacio de controles existente o, lo que es lo mismo, clasificar los controles existentes en diversos grupos (o segmentos), de forma que se evite que dos controles estén en el mismo si han sido manipulados al mismo tiempo previamente. De ahora en adelante nos referiremos a este solapamiento temporal como "colisión de uso" o, simplemente "colisión". Así, un grupo determinado es un contenedor de controles libre de colisiones. La situación ideal será tener todos los controles pertenecientes a un mismo usuario dentro de un mismo segmento, y tantos de éstos como usuarios haya interactuando en la mesa.

Por supuesto, esta solución funciona cogiendo como punto de partida que los usuarios tan sólo manipulan uno de sus controles a la vez. Incluso, el caso de tratar con controles específicos que requieran ser manipulados a la vez por un mismo usuario no supondría una amenaza para nuestro diseño, pues, conociendo esta situación, las manipulaciones sobre dichos controles se obviarían y no serían, por tanto, tratadas como colisiones de uso.

Pasamos a continuación a explicar el funcionamiento del algoritmo diseñado. Sea *Control* el conjunto que representa a los controles de una aplicación y *Grupo* el que representa a los segmentos o contenedores de controles, se define H (*Hit*) como una función que indica si dos controles han colisiona-

do en el pasado durante el transcurso de la sesión. La ecuación 3.1 la expresa en términos matemáticos.

$$\begin{aligned} &H : Control \times Control \longrightarrow \{True, False\} \\ &H\left(a, b\right) = \begin{cases} True & \text{si } a \text{ y } b \text{ han colisionado} \\ False & \text{en otro caso} \end{cases} \\ &\text{donde } Control \subset (\mathbb{N} \cup \{0\}) \end{aligned} \tag{3.1}$$

Al inicio de la ejecución, todos los controles se encuentran dentro de un mismo grupo, y cada nuevo control que entra en el sistema es automáticamente clasificado en éste. En cuanto dos elementos colisionan, el algoritmo de segmentación se dispara para obtener una nueva separación de controles libre de colisiones (una nueva colección de grupos). Esto se consigue extrayendo uno de los dos controles que han colisionado de su grupo y moviéndolo a otro donde no haya controles que hayan colisionado con él previamente. A este paso se le llama "migración". Si no existiese un contenedor libre de colisiones para dicho control, entonces se crearía uno nuevo (esto se conoce como "partición"). Además, si después de la migración todos los controles de dos grupos no han colisionado entre ellos, el algoritmo realizará una "fusión" de ambas particiones. Este último paso evita que el número de grupos crezca desproporcionadamente. Los pasos principales del algoritmo, ejecutado después de cada colisión, se muestran en el algoritmo 3.1. En resumen, este algoritmo mantiene siempre ciertas dos propiedades o estados:

- Si dos grupos existen es porque existe al menos un control en uno de ellos que ha colisionado con al menos un control del otro grupo. En la ecuación 3.2 se expresa términos matemáticos.

$$\begin{aligned} &\forall k_i k_j \in Grupo | k_i \neq k_j : \exists c_i \in k_i \exists c_j \in k_j | H(c_i, c_j) \\ &\text{donde } Grupo \subset (\mathbb{N} \cup \{0\}) \end{aligned} \tag{3.2}$$

- Los controles dentro de un grupo no han colisionado entre ellos (expresado matemáticamente en la ecuación 3.3).

$$\forall k \in Grupo \forall c_i c_j \in k : H(c_i, c_j) \tag{3.3}$$

En la situación óptima, si todo control de cada usuario colisiona con todos los controles de los demás, el algoritmo llega a la situación deseada donde existe un grupo por usuario y los controles dentro de cada grupo pertenecen a la misma persona, aunque esta situación podría darse antes de que todas las colisiones tuvieran lugar. En la mayor parte de los casos, el algoritmo de

Algoritmo 3.1 Algoritmo usado para mover controles entre grupos cuando ocurre una colisión.

Entrada: c_i, c_j controles involucrados en la colisión.

Precondición: c_i y c_j no han colisionado en el pasado. En caso contrario, el estado anterior y posterior a la ejecución del algoritmo será el mismo.

1: **Selección:** Si c_i y c_j se encuentran dentro del mismo grupo, elegir control para ser movido $m = select(c_i, c_j)$. Si no, Salir.
2: **Partición:** Si m no tiene ningún grupo de destino libre de colisiones, crear uno nuevo grupo y establecerlo como contenedor destino para m. Ir al paso 4.
3: **Migración:** Seleccionar el grupo de destino para m de entre los que tiene disponibles: $destination(m)$.
4: **Fusión:** Intentar mezclar todo par de grupos si todos los controles de ambos grupos no han colisionado entre ellos.

segmentación alcanzará soluciones subóptimas, pero que aún así serán mejores que seguir un enfoque totalmente desinformado (en términos de gestión de la territorialidad).

Para lidiar con los pasos 1 y 3 del algoritmo 3.1, se han creado tres versiones del mismo basadas en heurísticas para realizar la clasificación de forma más precisa, realizando los movimientos de los controles a los contenedores que tienen más probabilidades de pertenecer. Esto es importante, ya que, en una aplicación real donde hay gente manipulando controles, será complicado que se produzcan todas las colisiones necesarias para que se produzca la clasificación óptima.

3.2. Versiones Heurísticas

Los usuarios tienden a mantener sus controles agrupados (Scott y otros, 2004). Bajo esta hipótesis, proponemos varias versiones más informadas del algoritmo que toman en consideración uno de los factores explicados en el capítulo 2: la posición de los controles.

Un grupo determinado puede ser visto como un conjunto de puntos representando las posiciones de cada control en su interior. Por lo tanto, el centroide de un determinado grupo k con n_k controles dentro (cuyas posiciones se numeran de $c_{1_{pos}}$ a $c_{n_{pos}}$) se calcula tal y como se muestra en la

ecuación 3.4.

$$\begin{aligned} &centroid : Grupo \longrightarrow \mathbb{Z}^2 \\ &centroid(k) = \frac{c_{1_{pos}} + c_{2_{pos}} + \cdots + c_{n_{pos}}}{n_k} \\ &c_1, c_2, \cdots, c_n \in k \end{aligned} \tag{3.4}$$

Bajo la hipótesis comentada anteriormente, la distancia de cada control al centroide de su grupo (aquél que contiene los controles de un mismo usuario) debería ser menor que la distancia a los centroides del resto.

Para futuros usos, redefinimos las funciones *argmin* y *argmax* para que devuelvan un solo elemento en vez de un conjunto. Las ecuaciones 3.5 y 3.6 definen estas nuevas versiones llamadas *argmin'* y *argmax'*, respectivamente.

$$\underset{\substack{x_i \in X \\ i \in \mathbb{N}}}{\arg\min'} f(X) = \begin{cases} x_a & \text{si } \{x_a\} = \underset{X}{\arg\min} f(X) \\ x_a & \text{si } \{x_a, x_b, \cdots\} = \underset{X}{\arg\min} f(X) \end{cases} \tag{3.5}$$

$$\underset{\substack{x_i \in X \\ i \in \mathbb{N}}}{\arg\max'} f(X) = \begin{cases} x_a & \text{si } \{x_a\} = \underset{X}{\arg\max} f(X) \\ x_a & \text{si } \{x_a, x_b, \cdots\} = \underset{X}{\arg\max} f(X) \end{cases} \tag{3.6}$$

Ahora pasamos a definir algunas funciones que serán necesarias para la especificación formal de las distintas versiones heurísticas del algoritmo. Éstas son *isAvailable* (ver ecuación 3.7), la cual indica si un control puede ser movido a un grupo determinado (esto es, si dicho grupo está libre de colisiones para el control); y *availables* (ver ecuación 3.8), que devuelve el conjunto de contenedores disponibles (libres de colisiones) para un control determinado. También, sea k_c el grupo actual de un control c.

$$\begin{aligned} &isAvailable : Control \times Grupo \longrightarrow \{True, False\} \\ &isAvailable(c, k) = \begin{cases} True & \text{si } k \neq k_c \wedge \forall c' \in k : \neg H(c, c') \\ False & \text{en otro caso} \end{cases} \end{aligned} \tag{3.7}$$

$$\begin{aligned} &availables : Control \longrightarrow \wp(Grupo) \\ &availables(c) = \{k | isAvailable(c, k)\} \end{aligned} \tag{3.8}$$

A partir de la función *availables*, el paso 3 del algoritmo 3.1 se define de forma inmediata por la función *destination* expresada matemáticamente en la ecuación 3.9, donde se selecciona como grupo de destino aquél cuyo

centroide esté más cerca del control a ser movido. Nótese que c_{pos} representa la posición de un control determinado c.

$$\begin{aligned} &destination : Control \longrightarrow Grupo \\ &destination(c) = \underset{k \in availables(c)}{\arg\min'} \parallel c_{pos} - centroid(k) \parallel \end{aligned} \tag{3.9}$$

A continuación, en esta sección se explica, para cada versión considerada, cómo se define el paso 1 del algoritmo 3.1.

3.2.1. Versión CRCM (*Closest Remote Centroid Migration*)

Cuando ocurre una colisión entre dos controles, esta versión escoge como candidato para ser migrado aquel control con un grupo disponible más cercano (es decir, más cercano al centroide del grupo). En caso de empate, se seleccionará el que tenga un mayor número de destinos disponibles. Esto último intenta prevenir la creación excesiva de nuevos segmentos, aunque el caso en el que dos controles estén a la misma distancia exacta de un centroide es altamente improbable.

La ecuación 3.10 define matemáticamente la distancia de un control c al centroide de su grupo disponible más cercano, d_c^{Avail}. A partir de ésta, la función de selección para dos controles que colisionan c_i y c_j se podría definir según se muestra en la ecuación 3.11.

$$d_c^{Avail} = \begin{cases} \underset{\forall k | isAvailable(c,k)}{\text{mín}} \parallel c_{pos} - centroid(k) \parallel & \text{si } \exists k : isAvailable(c,k) \\ indefinido & \text{en otro caso} \end{cases} \tag{3.10}$$

$$\begin{aligned} &select : Control \times Control \longrightarrow Control \\ &select(c_i, c_j) = \begin{cases} \underset{c \in \{c_i, c_j\}}{\arg\min'} d_c^{Avail} & \text{si } \begin{array}{l} d_{c_i}^{Avail} \neq d_{c_j}^{Avail} \wedge \\ \exists k_1, k_2 : \\ \quad isAvailable(c_i, k_1) \wedge \\ \quad isAvailable(c_j, k_2) \end{array} \\ \underset{c \in \{c_i, c_j\}}{\arg\max'} |availables(c)| & \text{en otro caso} \end{cases} \end{aligned} \tag{3.11}$$

3.2.2. Versión FLCM (*Furthest Local Centroid Migration*)

Ésta es una propuesta diferente donde, dados los dos controles que producen una colisión, el seleccionado para ser movido es aquél que se encuentra más lejos del centroide de su propio grupo. En el improbable caso de que ambos controles estuvieran a una misma distancia de su centroide correspondiente, se seleccionaría aquél con un número mayor de destinos posibles (igual que se hacía en la versión CRCM). La lógica detrás de esta propuesta es que, cuanto más lejos está el control del centroide de su segmento, hace que el grupo esté menos cohesionado y es más probable que pertenezca a otro distinto.

En la ecuación 3.12 se expresa matemáticamente la distancia entre el control c y el centroide de su grupo, $d_c^{Current}$. Y, la función de selección del control a mover (entre c_i y c_j) está definida en la ecuación 3.13.

$$\begin{aligned} &d_c^{Current} = \parallel c_{pos} - centroid(k) \parallel \\ &\text{donde } k \in Grupo \wedge c \in k \end{aligned} \tag{3.12}$$

$$select(c_i, c_j) = \begin{cases} \underset{c \in \{c_i, c_j\}}{\arg\max'} d_c^{Current} & \text{si } d_{c_i}^{Current} \neq d_{c_j}^{Current} \\ \underset{c \in \{c_i, c_j\}}{\arg\max'} |availables(c)| & \text{en otro caso} \end{cases} \tag{3.13}$$

3.2.3. Versión RLCD (*Remote-Local Centroids Difference*)

Tanto en la versión CRCM como en la FLCM, es posible llegar a una situación donde el estado al que se llega después de mover un control es peor que el previo. Por ejemplo, en CRCM, si el control seleccionado para ser movido c_i tiene una distancia hacia el centroide del grupo destino mayor que la distancia al centroide del que era su segmento actual, o sea, en términos matemáticos, si $\arg\min'_{c \in \{c_i, c_j\}} d_c^{Avail} = c_i$, pero $d_{c_i}^{Avail} > \parallel c_{i_{pos}} - centroid(k_{c_i}) \parallel$ donde $k_{c_i} = \{k \in Grupo | c_i \in k\}$, entonces la mejor opción para c_i sería permanecer en su grupo actual, que seria el del centroide más cercano a dicho control.

Veamos un ejemplo de la situación descrita anteriormente con algunos valores numéricos de prueba, tal y como se muestra en la figura 3.1. En ésta, c_i y c_j están representadas como círculos; el centroide de su contenedor actual, como un cuadrado; y los centroides de sus respectivos grupos disponibles más cercanos, como triángulos. En este ejemplo, si se aplicara la heurística de

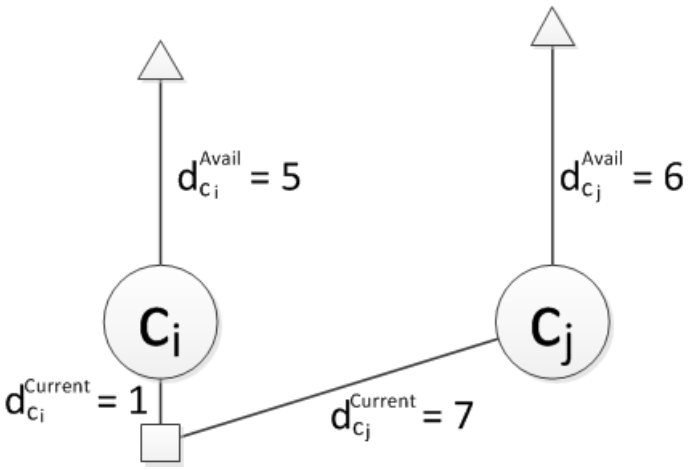


Figura 3.1: Ejemplo de distancias medidas para dos controles que colisionan c_i y c_j. Los centroides de sus destinos más cercanos están representados por triángulos; y el centroide de su grupo actual, por un cuadrado.

CRCM, el control seleccionado para ser movido sería c_i, puesto que presenta un menor valor de distancia al centroide de su segmento disponible más cercano. Desde la perspectiva de FLCM esta decisión sería incorrecta, ya que c_i está más cerca que c_j del centroide de su propio grupo, y por tanto c_j sería el candidato elegido. La misma situación ocurre si se aplica FLCM, porque mover el control elegido c_j lleva a un estado no óptimo del sistema de acuerdo con el enfoque de CRCM. Por lo tanto, este ejemplo sugiere que una combinación de CRCM y FLCM resultaría en una mejor heurística en términos de efectividad en la migración de controles que colisionan.

Dados dos controles c_i, c_j que colisionan, definimos dos medidas de error, $\Delta^{Avail} = \left| d_{c_i}^{Avail} - d_{c_j}^{Avail} \right|$ y $\Delta^{Current} = \left| d_{c_i}^{Current} - d_{c_j}^{Current} \right|$, donde Δ^{Avail} representa la diferencia entre las distancias de ambos controles con respecto a su respectivo destino más cercano, y $\Delta^{Current}$ expresa la diferencia de distancias con respecto al centroide de su grupo actual. Si Δ^{Avail} es mayor que $\Delta^{Current}$, entonces se deberá migrar aquél control que se encuentra más cerca de su destino disponible ($\arg\min'_{c \in \{c_i, c_j\}} d_c^{Avail}$), pues esta decisión contribuirá a reducir la diferencia que es mayor en magnitud (Δ^{Avail}). En caso contrario, el control seleccionado deberá ser aquél que se encuentre más lejos del centroide de su segmento actual ($\arg\max'_{c \in \{c_i, c_j\}} d_c^{Current}$), ya que así se contribuirá a reducir la mayor diferencia ($\Delta^{Current}$). La función de selección para la heurística RLCD está definida en términos matemáticos en la ecua-

ción 3.14.

$$select(c_i, c_j) = \begin{cases} & availables(c_i) \neq \emptyset \wedge \\ \underset{c \in \{c_i, c_j\}}{\arg\min'} d_c^{Avail} & \text{si } availables(c_j) \neq \emptyset \wedge \\ & \Delta^{Avail} > \Delta^{Current} \\ \underset{c \in \{c_i, c_j\}}{\arg\max'} d_c^{Current} & \text{en otro caso} \end{cases} \tag{3.14}$$

3.3. Implementación

Se ha realizado una implementación en C# del algoritmo 3.1 y de las diferentes versiones heurísticas del paso 1. A lo largo de esta sección se presentarán una serie de algoritmos más detallados que el anterior, aunque, con el fin de hacerlos más inteligibles, se escribirán en un híbrido entre pseudocódigo y código C# real.

3.3.1. Estructuras de Datos

La figura 3.2 muestra el diagrama de clases de diseño sobre el que se ha implementado la funcionalidad del algoritmo de diferenciación.

La clase principal en este diagrama es la clase *Classifier*, que es la que se encarga de clasificar cada control en el grupo que le corresponde. O, mejor dicho, dada la naturaleza del algoritmo, su función es segmentar el espacio definido por el conjunto de controles.

La clase *ClassifierInput* representa la información de un control necesaria para que el clasificador funcione. Ésta es, un nombre único para el control ($name$) y una posición ($posX, posY$) para las versiones heurísticas, que hacen uso de las distancias entre los controles. En la descripción del algoritmo realizada anteriormente en este capítulo, esta posición era denotada como c_{pos}.

Para determinar si dos controles han colisionado, hacemos uso de la función H (ver ecuación 3.1). En términos de implementación, dicha función hará uso de una matriz de colisiones (*collisionMatrix*). Ésta es una matriz booleana simétrica de tamaño $n \times n$ (siendo n el número de controles sobre la superficie) inicializada a *false* (o a 0), ya que, inicialmente, ningún control ha colisionado con ningún otro y, por tanto, pueden estar contenidos todos dentro de un mismo grupo. La figura 3.3 representa esta matriz en el estado en que solamente los controles c_0 y c_1 han colisionado entre ellos. Dos controles nunca podrán ser clasificados en el mismo grupo (y, por tanto, pertenecer al mismo usuario) si su intersección en la matriz tiene un 1. Por tanto, la diagonal principal siempre debe contener ceros.

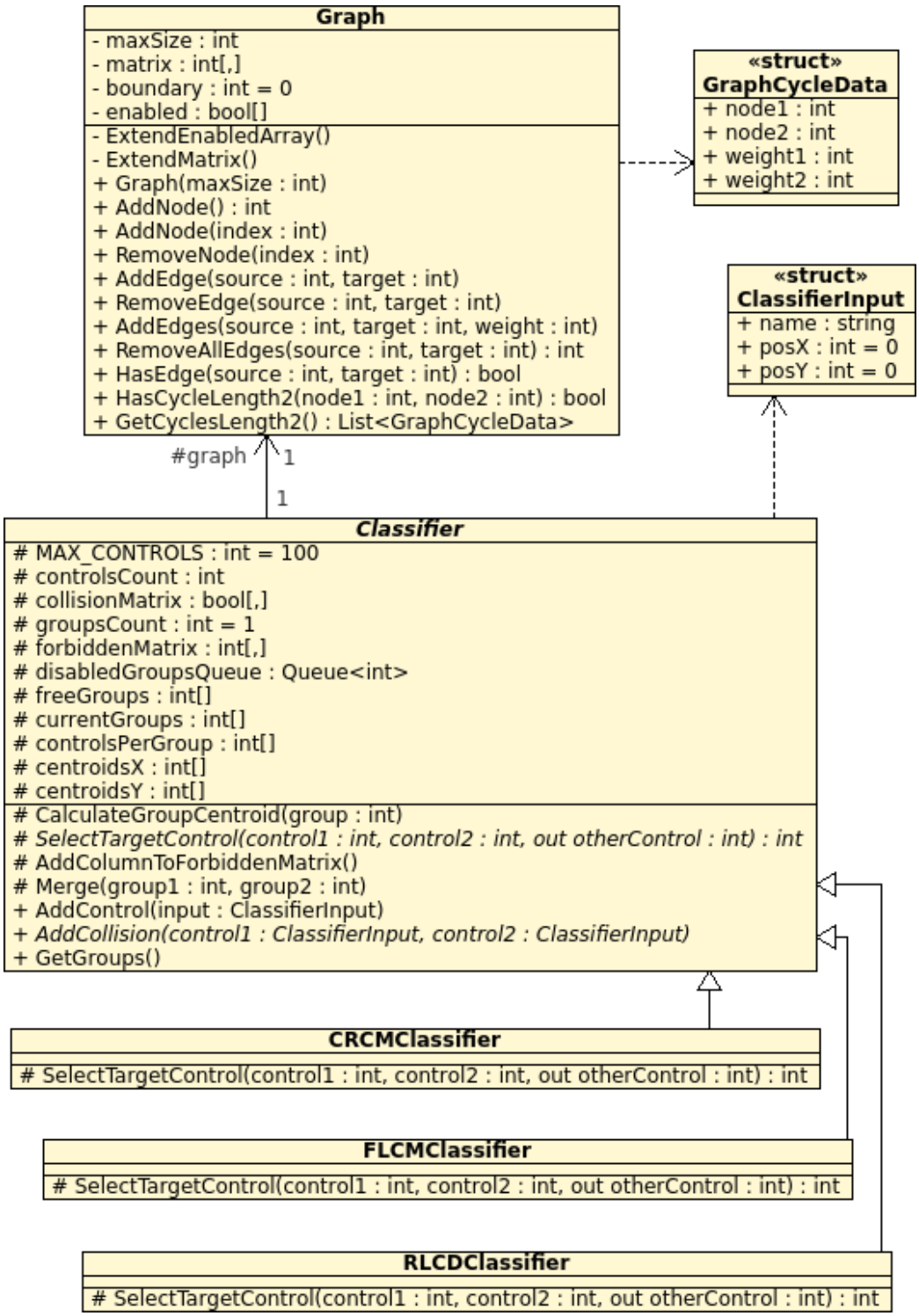


Figura 3.2: Diagrama de clases correspondiente a la estructura básica del clasificador.

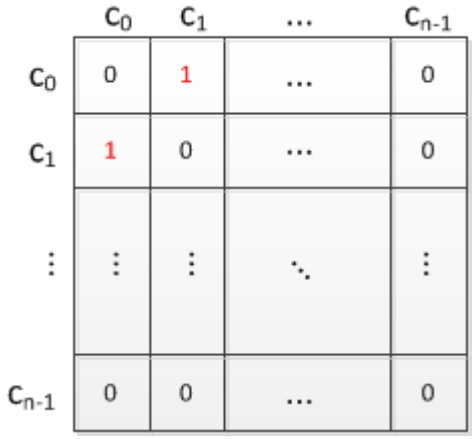


Figura 3.3: Matriz de colisiones cuando sólo c_0 y c_1 han colisionado entre ellos.

Para determinar si un control tiene disponible un grupo determinado, se hace uso de la función *isAvailable*. Para mantener esta información, se ha creado otra matriz (*forbiddenMatrix*), esta vez de tamaño $n \times K$ (siendo K el número total de grupos). Para cada control, cada elemento de su fila contendrá un entero que indicará cuántos controles en un determinado grupo han colisionado con él. Además, en su propio grupo actual tendrá marcado un 1, puesto que, en caso de que deba cambiarse de grupo, el destino no puede ser su contenedor actual. Por tanto, un valor mayor que cero implica que el control correspondiente no podrá ser clasificado en ese grupo cuando deba migrar de su contenedor actual. La figura 3.4 muestra un ejemplo de dicha matriz. En ésta, el control c_1 tiene prohibidos los grupos 0 y 1. Esto es porque se encuentra en el primero y hay tres controles en el segundo que han colisionado con él. No podría estar contenido en el segundo grupo porque el valor en la matriz es mayor que 1, lo que implicaría encontrarse en un contenedor con controles que han colisionado con él, lo cual viola la norma fundamental del algoritmo.

Para el paso 1 del algoritmo 3.1, dependiendo de la heurística aplicada, puede ser necesario mover el control que tenga más destinos disponibles. Para ello se hace uso de la función *availables* (ver ecuación 3.8). En términos de implementación, podría recorrerse la matriz de prohibiciones, pero, para mejorar el coste computacional, se ha decidido mantener un vector (*freeGroups*) que indica, para cada control, cuántos grupos tiene disponibles. En vez de utilizarse un vector (donde encontrar el valor máximo tiene coste lineal) podría haberse escogido un *max-heap*, pero, dado que el número de controles pre-

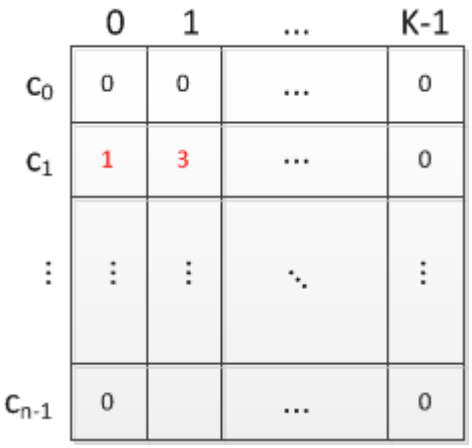


Figura 3.4: Matriz de prohibiciones.

sentes en una aplicación real sobre superficies interactivas está físicamente limitado, se ha considerado que la opción adoptada no perjudicará la eficiencia de forma notable. La figura 3.5-arriba muestra un ejemplo de este vector en el que ningún control tiene grupos disponibles, excepto el control c_1, que tiene 2.

Otros vectores adicionales que se ha utilizado en la implementación son *currentGroups* y *controlsPerGroup*. El primero indica, para cada control, un entero que representa en qué grupo se encuentra actualmente. Inicialmente, cuando un control nuevo entra en la aplicación, se introduce en el grupo 0. Esta información es lo que, en la descripción del algoritmo, se ha denotado por k_c. La figura 3.5-centro muestra un ejemplo de este vector en la situación inicial, donde todos los controles están incluidos en el grupo 0.

El vector *controlsPerGroup* denota, para cada grupo, cuántos controles tiene en su interior (es lo que anteriormente venía denotado por n_k). En la figura 3.5-abajo se puede observar una muestra de este vector en la que el grupo 0 tiene tres controles y el resto no tienen ninguno.

Las tablas 3.1 y 3.2 explican los atributos (entre los que se encuentran estas y otras estrucuras) y métodos, respectivamente, de la clase *Classifier*. Cabe destacar que, por motivos de eficiencia computacional, las matrices y vectores son estáticos. Se han creado para soportar inicialmente un máximo de cien controles (que es un número razonable teniendo en cuenta las limitaciones físicas de las mesas interactivas actuales), aunque hay servicios que permiten extender la memoria asignada en caso de superar dicho número. No obstante, para hacer más simples y claras las explicaciones de esta sección y los ejemplos de la sección 3.4, se considerarán como estructuras dinámicas, es

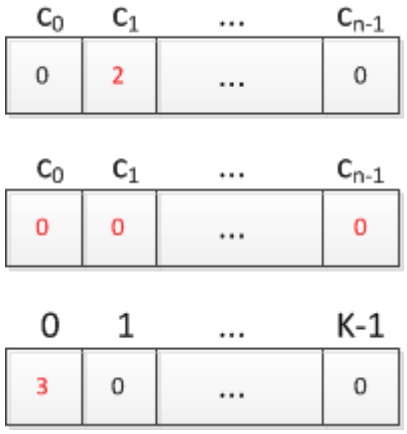


Figura 3.5: De arriba a abajo, vectores *freeGroups*, *currentGroups* y *controlsPerGroup*.

decir que, por ejemplo, sólo se mostrarán las columnas y filas de las matrices que sean relevantes en un momento dado (que no estén vacías).

Además de las descritas anteriormente, *Classifier* hace uso de una estructura de datos más, llamada *Graph*. Esta clase representa un grafo dirigido representado por su matriz de adyacencia, y su función es detectar cuándo se pueden fusionar dos grupos en el paso 4 del algoritmo 3.1. En este grafo, hay un nodo por cada grupo existente, y un arco del nodo a al b cuando hay algún control en el grupo a que tenga el b como contenedor disponible. El peso de cada arista indica el número de nodos que pueden ir de a a b. Inicialmente, sólo hay un nodo y ninguna arista.

Por la forma en la que se construye y mantiene la matriz de prohibiciones, un control sólo puede moverse a un grupo donde no haya ningún control que haya colisionado previamente con él. Por tanto, si tenemos dos grupos a y b tales que exista un bucle entre ellos en el grafo, y además, el peso de la arista que sale de cada grupo corresponde con el número de controles que contiene, entonces los dos grupos se podrán mezclar en uno solo. Esto es así porque todos los controles incluidos en a podrán moverse a b, y viceversa. Esta condición viene representada gráficamente en la figura 3.6. La información de un bucle se pasará al *Classifier* mediante la estructura *GraphCycleData*. Esta clase contiene dos enteros para representar a los dos nodos (grupos) a fusionar, y otros dos enteros con el peso de las dos aristas involucradas en el ciclo.

Los atributos y métodos de la clase *Graph* están explicados en las tablas 3.3 y 3.4, respectivamente.

Nombre	Descripción
MAX_CONTROLS	Constante que representa el tamaño máximo de las estructuras de datos estáticas (vectores, matrices...).
collisionMatrix	Matriz de colisiones. Indica si dos controles han sido manipulados al mismo tiempo en algún momento.
controlsCount, groups-Count	Número de controles n y de grupos K, respectivamente. Sirven como límite para las matrices y vectores estáticos.
forbiddenMatrix	Matriz de prohibiciones. Indica qué grupos están disponibles para cada control en el momento en el que éstos deban migrar. En el paso 4 del algoritmo 3.1, cuando se fusionen dos grupos, en vez de eliminar las columnas correspondientes de esta matriz, simplemente, se inhabilitan poniendo todos sus elementos a -1 (para mejorar la eficiencia). Sólo se considerará una columna nueva en esta matriz si no hay invalidadas.
disabledGroupsQueue	Cola de grupos correspondientes a las columnas invalidadas de la matriz de prohibiciones. Cada vez que se inhabilite alguna, se guardará su índice en esta cola. Así, cuando el algoritmo requiera crear un grupo nuevo, antes de considerar una columna adicional de la matriz, se intentará restaurar alguna de las columnas inhabilitadas extrayendo su identificador de esta cola.
freeGroups	Vector que indica, para cada control, el número de grupos a los que puede moverse en caso de colisión. Cada elemento de este vector coincide con el número de ceros que existe en la matriz de prohibiciones para la fila que represente a dicho control.
currentGroups	Vector que indica el grupo actual de cada control. Inicialmente se inicializa a 0 porque todos los controles se colocan en este grupo.
controlsPerGroup	Indica el número de controles en cada grupo. Su información se puede derivar del vector *currentGroups*, pero se mantiene por motivos de eficiencia. Inicialmente, sólo tendrá un elemento (correspondiente al grupo 0, con un valor n.
centroidsX, centroidsY	Vectores que contienen, para cada grupo, las coordenada X e Y, respectivamente, de su centroide.

Tabla 3.1: Atributos de la clase *Classifier*.

Nombre	Descripción
CalculateGroupCentroid	Calcula el centroide de un grupo determinado, de acuerdo con la función *centroid* (ver ecuación 3.4).
SelectTargetControl	Selecciona el control a ser movido de entre los dos que colisionan. Debe ser implementada en cada una de las subclases de *Classifier*. Se corresponde, en cada caso, con la función *select* (ver ecuaciones 3.11, 3.13 y 3.14 para ver las de la versión CRCM, FLCM y RLCD, respectivamente).
AddColumnToForbiddenMatrix	Utiliza una columna más de la matriz de prohibición y, en caso de que se haya llegado al límite estático establecido, se copian los contenidos de la matriz en otra con el doble de tamaño, que es la que se establece como nueva matriz de prohibición. Para retrasar esta situación al máximo, este método será llamado cuando no queden columnas invalidadas que rehabilitar en la matriz.
Merge	Realiza la fusión de dos grupos, dejando en la matriz de prohibiciones el primer grupo como válido e inhabilitando el segundo.
AddControl	Añade un control nuevo al diccionario *controlsIntsDict*, asociándole un número natural como identificador, y también realiza la asociación inversa en *intsControlsDict*. El control se incluye en el grupo de índice 0.
AddCollision	Se procesa la colisión ejecutando los pasos del algoritmo 3.1.
GetGroups	Obtiene una lista con tantos elementos como grupos actuales haya creados. Cada elemento de esta lista es otra con los controles que están contenidos en el grupo correspondiente.

Tabla 3.2: Métodos de la clase *Classifier*.

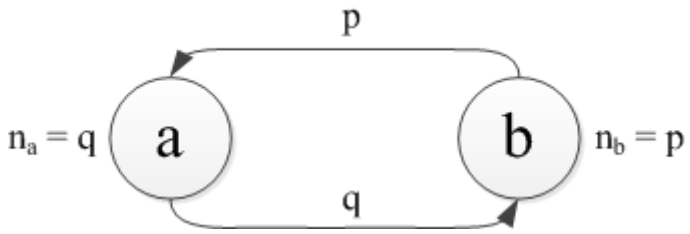


Figura 3.6: Condición que debe cumplirse en el grafo para poder fusionar los dos grupos *a* y *b*.

Nombre	Descripción
maxSize	Tamaño inicial de la matrix de adyacencia estática.
matrix	Matriz de adyacencia.
boundary	Como la matriz es estática, este índice sirve como límite para la matriz, indicando cuántas columnas son relevantes.
enabled	Vector de booleanos que indica si un nodo está activo o no. Para evitar manipulaciones más ineficientes sobre la matriz, cuando un nodo se "elimine", se marcará a *false* su correspondiente elemento de este vector. Y, en el momento en el que se vuelva a añadir, simplemente bastará con reactivarlo (ponerlo a *true*).

Tabla 3.3: Atributos de la clase *Graph*.

Nombre	Descripción
ExtendEnabledArray	Duplica el tamaño del vector *enabled*, en el caso de que se llegue al límite de elementos marcado por el atributo *maxSize*.
ExtendMatrix	Duplica el tamaño de la matriz de adyacencia, en el caso de llegar al límite *maxSize* de elementos.
Graph	Constructor. Crea una matriz de adyacencia del tamaño especificado como argumento.
AddNode	Si no se le pasan argumentos, se crea un nodo nuevo considerando una fila y columna nuevas en la matriz, y se devuelve el número de nodos después de crearlo. Si, por el contrario, se le especifica un entero como argumento, se vuelve a habilitar el nodo desactivado cuyo identificador corresponde con dicho entero.
RemoveNode	Desactiva el nodo especificado (es decir, se marca como *false* en el vector *enabled*).
AddEdge	Añade una arista dirigida desde el primer nodo especificado como argumento al segundo, o bien se aumenta en 1 el peso de la misma.
RemoveEdge	Decrementa el peso en 1 o borra (en su caso) la arista que va desde el primer nodo especificado como argumento al segundo.
AddEdges	Añade una arista que va del primer nodo pasado como argumento al segundo, y con el peso especificado en el tercer argumento. Si ya existe ésta, se suman los pesos.
RemoveAllEdges	Pone a 0 el peso de la arista que va del nodo especificado como primer argumento al segundo, lo que es lo mismo que eliminarla.
HasEdge	Comprueba si existe una arista entre el nodo especificado como primer argumento y el segundo.
HasCycleLength2	Comprueba si existe un ciclo de longitud 2 entre los nodos pasados como argumentos. O, lo que es lo mismo, comprueba que haya una arista del primero al segundo y a la inversa.
GetCyclesLength2	Obtiene una lista con todos los ciclos encontrados que involucren a dos nodos. Esta lista contiene, por cada ciclo, un objeto *GraphCycleData*. Si no hubiera ciclos, devuelve una lista vacía.

Tabla 3.4: Métodos de la clase *Graph*.

Algoritmo 3.2 Algoritmo ejecutado al añadir un control nuevo (descripción del método *AddControl* de la clase *Classifier*).

Entrada: Control a añadir c

```
// Agregar a los diccionarios
controlsIntsDict[c] = controlsCount
intsControlsDict[controlsCount] = c

// Entra en el grupo 0. Prohibir dicho grupo para este control
currentGroups[controlsCount] = 0
forbiddenMatrix[controlsCount, 0] = 1

// Los demás grupos están disponibles para este control salvo que estén inhabilitados
freeGroups[controlsCount] = groupsCount - 1 -
    disabledGroupsQueue.Count

// Incrementar el número de controles del grupo 0
controlsPerGroup[0]++

// Incrementar el número de controles
controlsCount++

// Recalcular centroide del grupo 0
CalculateGroupCentroid(0)
```

3.3.2. Detalles del Algoritmo

En esta sección se explicará en detalle cómo se efectúan algunas de las operaciones del clasificador de controles (clase *Classifier*), introducida en la sección anterior. El algoritmo 3.2 describe el funcionamiento del método *AddControl*, que se ejecuta cuando un control nuevo es introducido en la aplicación.

A pesar de que cada versión heurística realiza el paso 1 (selección) del algoritmo 3.1 de forma diferente, lo único que cambia es la ejecución del método *SelectTargetControl*, que se corresponde con la función *select*. Esta función puede verse en las ecuaciones 3.11, 3.13 y 3.14 para la versión CRCM, FLCM y RLCD, respectivamente. El algoritmo 3.3 muestra las operaciones comunes a todas las heurísticas del paso de selección.

Algoritmo 3.3 Detalle del paso 1 del algoritmo 3.1 (selección).

Entrada: *ci*, *cj* controles involucrados en la colisión
Precondición: *ci* y *cj* no han colisionado en el pasado. En caso contrario, el algoritmo finaliza sin avanzar al siguiente paso
Devuelve: El control a ser movido *m*, el otro control *o*

```
// Mirar si hace falta mover controles de grupos. Puede ser que la colisión se haya
    producido por controles de distintos usuarios pero que ya estén en grupos distintos.
    En este caso, sólo hay que penalizar el acceso de ci al grupo de cj y viceversa
if currentGroups[ci] != currentGroups[cj] then
  // Si ci podía ir al grupo de cj, ahora ya no
  if forbiddenMatrix[ci, currentGroups[cj]] == 0 then
    freeGroups[ci]--
    graph.RemoveEdge(currentGroups[ci],
      currentGroups[cj])
  end

  // Si cj podía ir al grupo de ci, ahora ya no
  if forbiddenMatrix[cj, currentGroups[ci]] == 0 then
    freeGroups[cj]--
    graph.RemoveEdge(currentGroups[cj],
      currentGroups[ci])
  end

  // Aumentar penalización
  forbiddenMatrix[ci, currentGroups[cj]]++
  forbiddenMatrix[cj, currentGroups[ci]]++

  // Marcar colisión
  collisionMatrix[ci, cj] = true
  collisionMatrix[cj, ci] = true

  Salir
end

// Hay que mover de categoría uno de los controles
m = SelectTargetControl(ci, cj)
o = {ci, cj} - m
```

Algoritmo 3.4 Detalle del paso 2 del algoritmo 3.1 (partición).

Entrada: El control a ser movido m

Precondición: freeGroups[m] == 0. Si no se cumple, no se ejecuta el algoritmo

Devuelve: El grupo de destino d

```
if disabledGroupsQueue.Size() == 0 then
  // No hay categorías invalidadas. Creamos una nueva
  graph.AddNode()
  Agregar nueva columna con índice groupsCount a forbiddenMatrix
      inicializada a 0
  d = groupsCount
  groupsCount++
else
  // Habilitamos el grupo cuyo índice saquemos de disabledGroupsQueue
  d = disabledGroupsQueue.pop()
  graph.AddNode(d)
  forbiddenMatrix[i,d] = 0, ∀i ∈ [0,controlsCount[
end

// Incrementamos el número de grupos libres para cada control
for i = 0 to controlsCount - 1 do
  freeGroups[i]++
  graph.AddEdge(currentGroups[i], d)
end

// Realizamos el mantenimiento de los atributos y estructuras de datos
maintainDataStructures(m, d)
```

El resto de pasos son también comunes a todas las versiones. El paso 2 (partición) está descrito en el algoritmo 3.4; y el paso 3 (migración), en el algoritmo 3.5. Al final de ambos, hacen una llamada a la función *maintainDataStructures*, descrita en el algoritmo 3.6, la cual se encarga de mantener actualizados los diferentes atributos y estructuras de datos de la clase *Classifier* después de ejecutar uno de los dos pasos anteriores (que, recordemos, son mutuamente excluyentes).

Finalmente, el paso 4 (fusión) aparece más en detalle en el algoritmo 3.7. Éste hace uso de un método de la clase *Classifier*, *Merge*, que realiza la fusión de dos grupos concretos. Su descripción algorítmica puede verse en el algoritmo 3.8.

Algoritmo 3.5 Detalle del paso 3 del algoritmo 3.1 (migración).

Entrada: El control a ser movido m

Precondición: freeGroups[m] != 0. Si no se cumple, no se ejecuta el algoritmo

Devuelve: El grupo de destino d

```
// Nos apuntamos todos los grupos libres para m em V. v contendrá el cardinal de V
v = 0
V = vector de groupsCount elementos (no es necesario
    inicializarlo)
for j = 0 to groupsCount - 1 do
  if forbiddenMatrix[m, j] == 0 then
    V[v] = j
    v++
    if v == freeGroups[m] then
      // Ya no quedan más categorías libres
      break
    end
  end
end

// El grupo al que ir se calcula mediante la función destination
d = destination(m) (ver ecuación 3.9)

// Como m se mueve de un grupo A a otro B, hay una arista menos de A a todos los
    grupos libres para m (que no sean B), y hay una arista más de B a todos los grupos
    libres para m (menos al propio B).
for j = 0 to v - 1 do
  if V[j] != d then
    graph.RemoveEdge(currentGroups[m], V[j])
    graph.AddEdge(d, V[j])
  end
end

// Realizamos el mantenimiento de los atributos y estructuras de datos
maintainDataStructures(m, d)
```

Algoritmo 3.6 Descripción de la función *maintainDataStructures*, utilizada por los algoritmos 3.4 y 3.5.

Entrada: El control a ser movido m, el control que se quedará en el grupo actual o, el grupo de destino d

```
// Si penalizamos el acceso a un grupo, decrementamos el número de grupos libres
if forbiddenMatrix[m, d] == 0 then
  graph.RemoveEdge(currentGroups[m], d)
  freeGroups[m]--
end

// Si se deposita m en d, entonces d pasa a ser prohibido para o
if forbiddenMatrix[o, d] == 0 then
  graph.RemoveEdge(currentGroups[o], d)
  freeGroups[o]--
end

// Los controles que colisionan con m podrán ir al grupo del que m sale, pero no al que
    entra ahora
for h ∈ collisionMatrix[m, i] do
  if h == true then
    forbiddenMatrix[i, currentGroups[m]] =
      max(0, forbiddenMatrix[i, currentGroups[m]] - 1)
    if forbiddenMatrix[i, currentGroups[m]] == 0 then
      freeGroups[i]++
      graph.AddEdge(currentGroups[i], currentGroups[m])
    end
    if forbiddenMatrix[i, d] == 0 then
      freeGroups[i]--
      graph.RemoveEdge(currentGroups[i], d)
    end
    forbiddenMatrix[i, d]++
  end
end

// Actualizamos el resto de estructuras y recalculamos centroides
forbiddenMatrix[m, d]++
forbiddenMatrix[o, d]++
controlsPerGroup[currentGroups[m]]--
controlsPerGroup[d]++
currentGroups[m] = d
collisionMatrix[m, o] = true
collisionMatrix[o, m] = true
CalculateGroupCentroid(currentGroups[m])
CalculateGroupCentroid(currentGroups[o])
```

Algoritmo 3.7 Detalle del paso 4 del algoritmo 3.1 (fusión).

```
// Si se puede, fusionar grupos. A y B son los grupos involucrados en el ciclo. a y b son
    los pesos de las aristas que salen de cada uno de esos grupos hacia el otro en el ciclo.
do
  occurred = false
  foreach {A,a,B,b} in graph.GetCyclesLength2() do
    // Al fusionar dos grupos, el grafo se actualiza y puede que algunos ciclos antes
        detectados ya no existan. Por tanto, comprobamos que A y B no han sido
        víctimas de una fusión en esta iteración del algoritmo. Para ello, comprobamos
        que ninguno de los dos esté inactivo y que todavía exista un ciclo entre ellos.
    if forbiddenMatrix[0, A] >= 0 and
       forbiddenMatrix[0, B] >= 0 and
       graph.HasCycleLength2(A, B) then
      if controlsPerGroup[A] == a and
         controlsPerGroup[B] == b then
        Merge(A, B)
        occurred = true
        // Recalcular centroide del grupo resultante.
        CalculateGroupCentroid(A)
      end
    end
  done
while occurred = true
```

Algoritmo 3.8 Descripción del método *Merge* de la clase *Classifier*.
Entrada: Los dos grupos a ser fusionados: A y B

```
// Cambiar el número de controles por grupo. B se queda sin controles porque los que
    tenía pasan a A
controlsPerGroup[A] += controlsPerGroup[B]
controlsPerGroup[B] = 0

for i = 0 to controlsCount - 1 do
  // Los controles de un grupo tenían libre el otro. Ahora, al fusionarlos, todos los
      controles de estos grupos tienen un grupo libre menos. Además, todos los que no
      tenían prohibido B pero sí A (o viceversa) no podrán ir al grupo resultante,
      así que tendrán también un grupo libre menos. Y, los controles que tenían libre
      ambos grupos ahora perderán uno libre (B), pues se quedarán con A solamente
  if forbiddenMatrix[i, B] == 0 then
    freeGroups[i]--
    graph.RemoveEdge(currentGroups[i], B)
  else if forbiddenMatrix[i, A] == 0 then
    freeGroups[i]--
    graph.RemoveEdge(currentGroups[i], A)
  end

  // Si un control tenía prohibida la entrada en A o en B, ahora la tendrá prohibida en
      A (el grupo resultante), y si la tenía prohibida en ambos, ahora la tendrá en A
      por partida doble
  forbiddenMatrix[i, A] += forbiddenMatrix[i, B]

  // Mover los controles de B a A
  if currentGroups[i] == B then
    currentGroups[i] = A
  end

  // Desactivar la columna correspondiente al grupo B
  forbiddenMatrix[i, B] = -1
end

for j = 0 to groupsCount - 1 do
  // Mover a A las aristas que salen de B
  if graph.HasEdge(B, j) then
    x = graph.RemoveAllEdges(B, j)
    graph.AddEdges(A, j, x)
  end
end

// Eliminar B del grafo y marcarlo como grupo inhabilitado
graph.removeNode(B)
disabledGroupsQueue.push(B)
```

3.4. Ejemplo de Funcionamiento

En esta sección se muestra, a través de un ejemplo, cómo el clasificador de controles gestiona la distinción de usuarios y cómo evolucionan sus estructuras de datos. Por simplicidad y con el fin de hacerlo más genérico, para este ejemplo no se van a tener en cuenta las nociones de distancias físicas entre los controles. Por tanto, el único paso que difiere entre las diferentes versiones del clasificador (el de la selección del control a mover) se resolverá en este ejemplo de forma arbitraria. Cabe destacar que esta decisión no afecta a la corrección del algoritmo, el cual, si se produce un número de colisiones suficientemente grande, será capaz de clasificar todos los controles en los grupos correspondientes. La importancia de las versiones heurísticas recae simplemente en hacer que los movimientos entre grupos sean más informados y, por tanto, que el algoritmo sea capaz de converger con un menor número de colisiones. Otra razón para realizar este movimiento de forma aleatoria en este ejemplo es poder mostrar cómo se comportaría el clasificador si se toma la opción más desfavorable en el paso de selección.

Se consideran dos usuarios interactuando al mismo tiempo en una superficie interactiva, cada uno con tres controles. Los de uno están representados por triángulos (c_0, c_1, c_2) y los del otro, por cuadrados (c_3, c_4, c_5). Tal y como se muestra en la figura 3.7, inicialmente todos están incluidos en el mismo grupo (representado por un círculo y denotado por el número 0). Justo a la derecha, se muestra la matriz de prohibiciones que, en este punto inicial, solamente tiene una columna correspondiente al único contenedor existente, y está inicializada a unos porque ningún control puede moverse a un grupo en el que se encuentra actualmente. Bajo éstos, a la izquierda aparece el grafo para la operación de fusión, que en este momento solamente tiene un nodo. Finalmente, a la derecha del grafo se puede observar una matriz cuadrada que representa la matriz de colisiones. Esta matriz está inicializada a ceros porque todavía no ha colisionado ningún control. Para mejor claridad, las celdas que contienen ceros se han dejado vacías.

Las figuras 3.8-3.15 representan los sucesivos pasos del algoritmo en el ejemplo, y todas disponen en la misma posición las diferentes estructuras, para no confundir al lector. Se ha decidido no representar en estas ilustraciones los vectores *freeGroups*, *currrentGroups* y *controlsPerGroup*, ni tampoco la cola *disabledGroupsQueue*, puesto que la información que contienen puede derivarse de los elementos que sí que aparecen en los dibujos, sobretodo en un ejemplo reducido como el que se presenta en esta sección.

Supongamos que las colisiones que se producen son las que se muestran a continuación (en el orden indicado). Posteriormente se explica más en detalle qué provoca cada una.

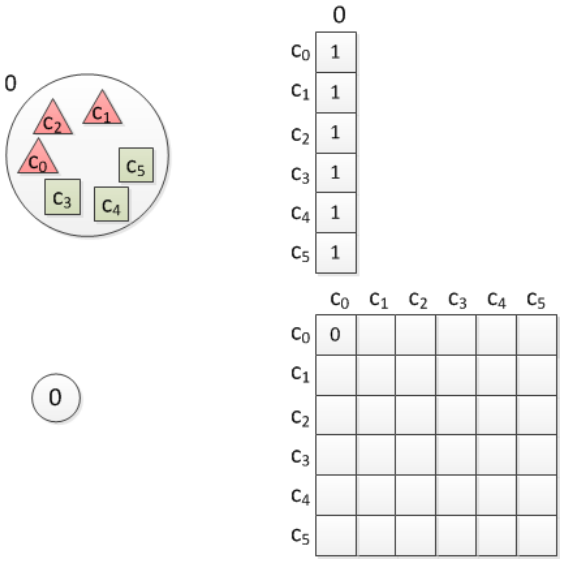


Figura 3.7: Situación inicial del clasificador, donde no se ha producido ninguna colisión. De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

1. c_0 y c_3: Se crea el grupo 1 y se mueve c_0 a éste.
2. c_3 y c_1: c_1 migra al grupo 1.
3. c_2 y c_4: c_4 migra al grupo 1.
4. c_0 y c_4: Se crea el grupo 2 y se coloca c_0 en él.
5. c_1 y c_4: Se mueve c_1 al grupo 2.
6. c_0 y c_5: Como están en grupos distintos, no se realiza ninguna partición ni migración.
7. c_2 y c_5: c_2 es migrado al grupo 2, los grupos 0 y 1 se fusionan, y el algoritmo converge a una solución.

Considerando la primera colisión, si c_0 colisiona con c_3, uno de los dos debe salir del grupo. Supongamos que se selecciona c_0. Como no tiene grupos libres a los que ir (todos tienen una prohibición de 1), es necesario crear uno nuevo (partición). Además, se marca que c_3 no podrá moverse al grupo 1 puesto que éste contiene ahora un control con el que ha colisionado. Por el mismo motivo, c_0 no podrá volver al grupo 0 mientras c_3 continúe en él. En

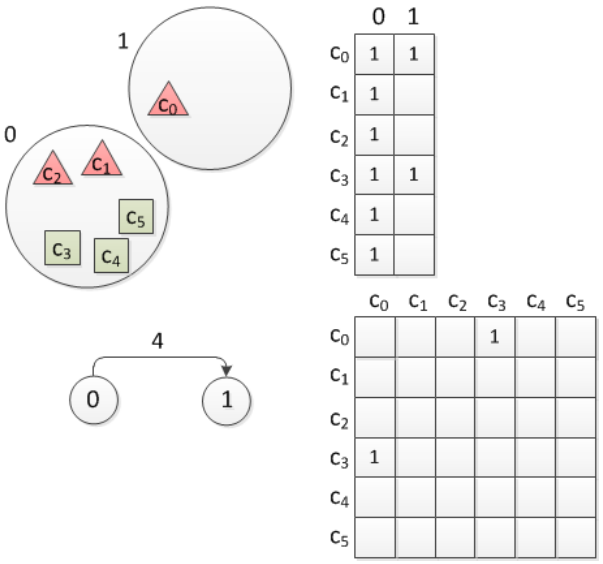


Figura 3.8: Situación del clasificador tras producirse la primera colisión (c_0 y c_3). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

el grafo, se indica que hay cuatro controles en el grupo 0 que pueden moverse al 1 (es decir, todos menos c_3). La figura 3.8 muestra esta situación.

En este punto, si se produce una colisión entre c_3 y c_1, hay dos opciones. Si se selecciona c_1, éste puede moverse al grupo 1 puesto que no hay ningún control en este contenedor que haya colisionado con él. No obstante, si se elige seleccionar c_3, éste no puede ir a ningún grupo, y se debería crear uno nuevo. Supongamos que la heurística correspondiente elige mover c_1 (migración). Por tanto, es necesario incrementar la penalización de c_3 al grupo 1 (es decir, no podrá moverse a dicho grupo porque en él hay dos controles con los que ha colisionado). Además, el grafo indica ahora que hay un control menos en el grupo cero que puede ir al 1, porque acaba de ser movido. La figura 3.9 representa este nuevo estado.

A continuación, c_2 y c_4 colisionan. si movemos el primero, éste se movería al grupo 1 y el algoritmo habría convergido, pues ya estarían los controles de cada usuario correctamente segmentados y, aunque se produjeran nuevas colisiones entre los controles de ambos usuarios, el estado de los grupos nunca cambiaría. Por tanto, supongamos que el candidato para ser movido es c_4. Esta decisión supondrá su migración al grupo 1. Aparentemente, esto sería incorrecto, pues iría a parar a un grupo con controles de un usuario distinto, pero todavía no se han producido evidencias de que esto sea así (no han

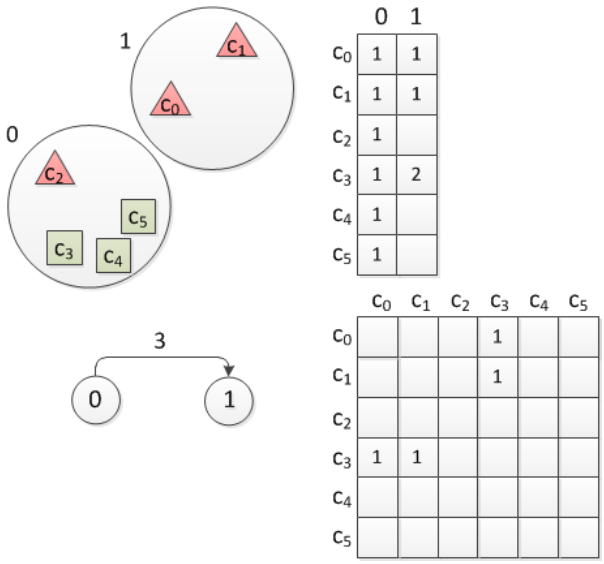


Figura 3.9: Situación del clasificador tras producirse la segunda colisión (c_1 y c_3). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

colisionado entre ellos). En la figura 3.10 se puede ver la nueva agrupación.

Si la aplicación sigue su curso, es de esperar que en algún momento se produzca una colisión entre c_0 y c_4. Por ejemplo, se selecciona c_0, el cual no tiene ningún grupo libre, lo que fuerza al algoritmo a ejecutar el paso de partición y mover a dicho control al nuevo contenedor. Una vez hecho esto, el mantenimiento de las estructuras requiere que se recorra la matriz de colisiones y, para cada control que haya colisionado con c_0 (c_3 en este caso), reducir su penalización al grupo 1 (siempre que esta penalización sea mayor que 0) e incrementarla para el nuevo grupo en el que c_0 entra. El resultado de estas operaciones puede observarse en la figura 3.11.

La siguiente colisión que se produce es entre c_1 y c_4, y se selecciona el primero para moverse al grupo 2. Por reducir la penalización de los controles que hayan colisionado con c_1 en el grupo del que éste sale, el grupo 1 queda en este momento libre para c_3, pero su penalización al grupo 2 se incrementa. Ver la figura 3.12 para una aclaración visual.

Pongamos ahora el caso de que colisionan dos elementos que ya se encuentran en grupos distintos, por ejemplo, c_0 y c_5. Según el algoritmo 3.3, la ejecución acabará en el paso 1 del algoritmo general 3.1. Esta colisión provoca, simplemente, que se aumente la penalización de ambos controles al grupo del otro. Como consecuencia, c_5 tendrá ahora prohibida la entrada al grupo

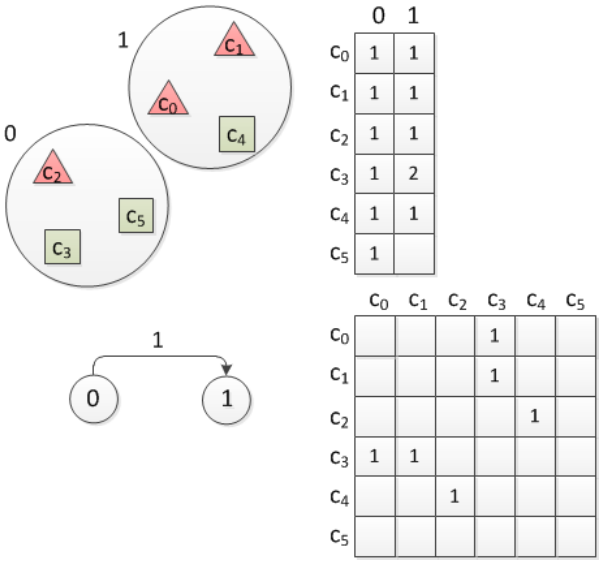


Figura 3.10: Situación del clasificador tras producirse la tercera colisión (c_2 y c_4). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

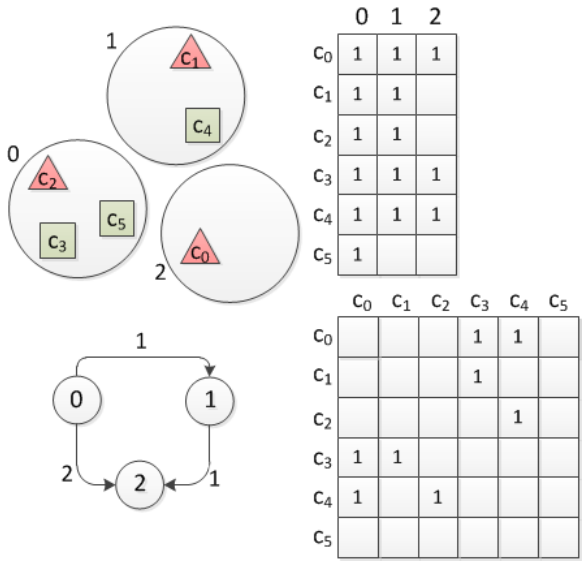


Figura 3.11: Situación del clasificador tras producirse la cuarta colisión (c_0 y c_4). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

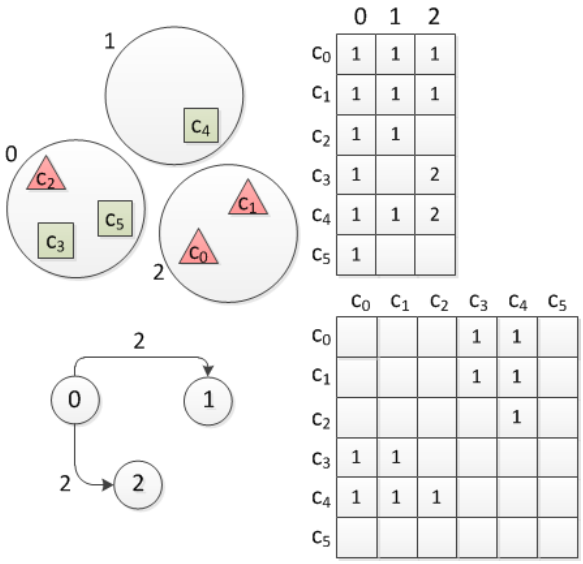


Figura 3.12: Situación del clasificador tras producirse la quinta colisión (c_1 y c_4). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

2 y, por tanto, habrá una arista menos (o lo que es lo mismo, se reducirá el peso de la misma) del grupo 0 (contenedor de c_5) al grupo 2 (contenedor de c_0). La situación a la que llega el clasificador después de este paso es la que refleja la figura 3.13.

Si en este momento colisionan c_2 y c_5, y el primero es el escogido para migrar, será trasladado al grupo 2, que es el único que tiene libre. Tras ejecutarse el paso 3, la situación es la que se puede observar en la figura 3.14, donde c_4 tiene de nuevo el grupo 0 disponible. Como se puede ver en la figura, en el grafo hay un ciclo entre los grupos 0 y 1, y los pesos de las aristas salientes coinciden con el número de controles del grupo del que salen. Esto indica que, todos los controles de cada uno de esos dos grupos podrían moverse al otro. Por tanto, el paso 4 (fusión) permite hacer, en este caso, la mezcla de ambos contenedores. Como se puede observar en la figura 3.15, el grupo 1 queda inhabilitado en la matriz de prohibiciones, y las prohibiciones que tenía previamente se suman al grupo 0, que ahora contiene los controles del grupo desactivado. Llegados a este punto, el algoritmo ha convergido. Ante nuevas colisiones, las estructuras de datos seguirán actualizándose, pero no se producirá ninguna partición, migración o fusión más.

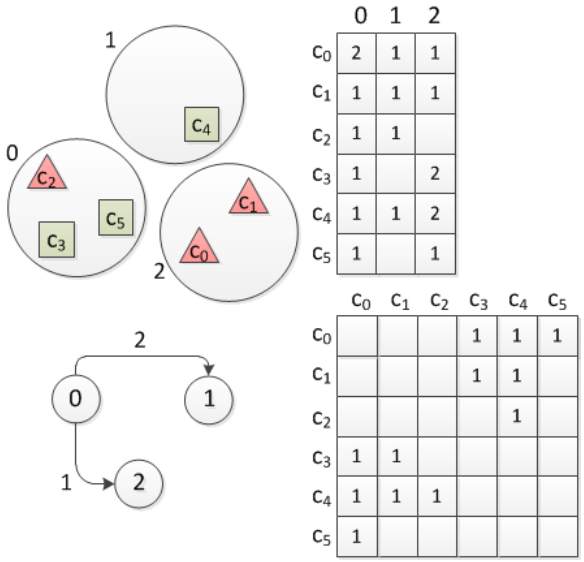


Figura 3.13: Situación del clasificador tras producirse la sexta colisión (c_0 y c_5). De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

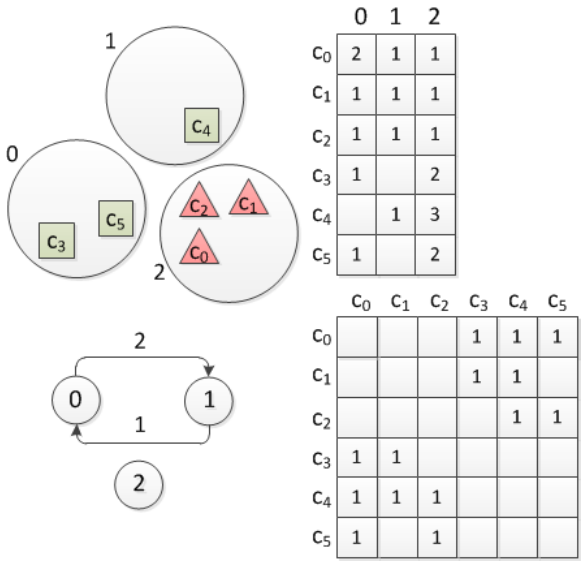
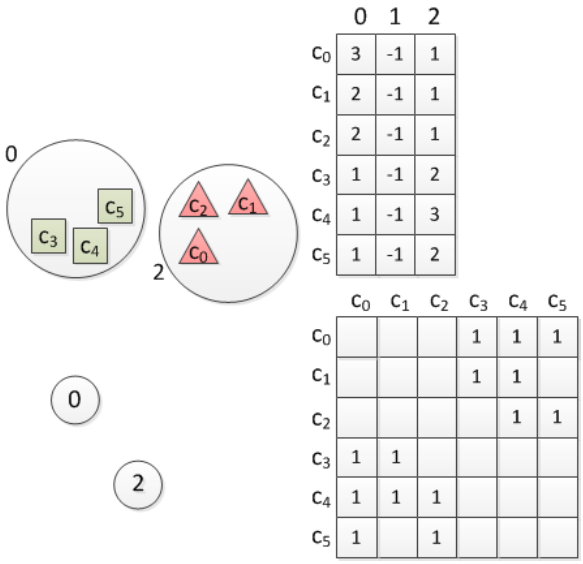


Figura 3.14: Situación del clasificador tras producirse la séptima colisión (c_2 y c_5) y antes del paso de fusión. De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.



	0	1	2
c_0	3	-1	1
c_1	2	-1	1
c_2	2	-1	1
c_3	1	-1	2
c_4	1	-1	3
c_5	1	-1	2

	c_0	c_1	c_2	c_3	c_4	c_5
c_0				1	1	1
c_1				1	1	
c_2					1	1
c_3	1	1				
c_4	1	1	1			
c_5	1		1			

Figura 3.15: Situación del clasificador tras producirse la séptima colisión (c_2 y c_5) y después de realizar el paso de fusión de grupos. De arriba a abajo y de izquierda a derecha: grupos creados, grafo, matriz de prohibiciones, matriz de colisiones.

3.5. Conclusiones

En este capítulo se ha presentado un algoritmo basado en la simultaneidad de interacciones para hacer frente a la tarea de diferenciación de usuarios. Su objetivo es conseguir clasificar los controles de una aplicación sobre superficies interactivas de acuerdo con el usuario que los está manipulando, bajo la hipótesis de que cada control únicamente es manipulado por un usuario, y que un usuario sólo manipula un control a la vez.

Primero, se ha presentado dicho algoritmo de forma genérica en cuatro pasos básicos, los cuales se han detallado matemáticamente. A continuación, se ha hecho hincapié en la forma de implementarlo, indicando qué estructuras de datos se han utilizado, e introduciendo una clase llamada *Classifier* como contenedora de la lógica del algoritmo de diferenciación. Después, cada uno de los pasos del algoritmo básico ha sido explicado con más detalle utilizando las estructuras de datos introducidas previamente.

El algoritmo propuesto se ejecuta cada vez que dos controles son manipulados al mismo tiempo (a lo que se ha denominado "colisión"), obteniendo a cada ejecución un nuevo estado, es decir, una nueva agrupación de controles. De esta forma, si se produce el número suficiente de colisiones, el algoritmo

convergerá a una situación en la que haya tantos grupos creados como usuarios interactuando, y todos los controles dentro de un grupo pertenezcan a la misma persona. Esta propuesta no olvida nunca las colisiones producidas, es decir, si dos controles colisionan serán considerados para siempre como pertenecientes a usuarios disintos. Aunque el hecho de que el mismo usuario manipule al mismo tiempo dos controles distintos viola nuestra hipótesis de partida, cabría la posibilidad de que sucediera. Una posible forma de recuperarse ante esta situación sería hacer que una colisión pudiera "olvidarse" tras transcurrir un cierto periodo de tiempo, ya que cabe esperar que los controles realmente pertenecientes a usuarios distintos estén continuamente colisionando entre ellos.

Otro punto débil del algoritmo es que los estados intermedios que se obtienen pueden no ser óptimos, debido a que tras cada colisión sólo se mueve de grupo uno de los dos controles involucrados en la misma. Por tanto, puede ser que en un momento dado haya más grupos creados que usuarios reales. Esto podría solucionarse explorando todos los cambios posibles de controles entre grupos después de cada colisión, pero sería más ineficiente. Nuestra propuesta busca obtener una solución aceptable sin que se perjudique mucho el coste computacional, puesto que no hay que olvidar que la tarea de la diferenciación de usuarios es una tarea secundaria y, por tanto, debe dejar libre el procesador para ejecutar las aplicaciones principales.

Una mejora para el algoritmo en este aspecto ha sido diseñar el clasificador de controles de forma extensible, de forma que ciertas operaciones del algoritmo puedan ser redefinidas mediante subclases de *Classifier*. En este capítulo se han presentado tres versiones diferentes que, utilizando heurísticas basadas en otro factor (la posición de los controles, ver sección 2.6), pretenden realizar el movimiento de los controles de forma más inteligente, con la finalidad de reducir el número de colisiones necesarias para llegar a la convergencia. El capítulo 4 las evaluará para mostrar su efectividad en este aspecto.

4 Evaluación

El análisis del rendimiento de las versiones heurísticas del algoritmo propuesto en el capítulo 3 puede realizarse en términos de su complejidad computacional y en términos de su efectividad distribuyendo los controles en grupos. Es decir, no solamente de acuerdo al tiempo que tarda en realizar una computación después de producirse una colisión, sino también analizando la adecuación del estado alcanzado después de haberse procesado una colisión usando cada una de las versiones propuestas.

En este capítulo se ha realizado un estudio comparativo entre las tres versiones descritas en el capítulo 3: CRCM (sección 3.2.1), FLCM (sección 3.2.2) y RLCD (sección 3.2.3). Además, se ha añadido una versión menos informada que sirve de referencia para comparar, donde el control seleccionado para moverse es siempre aquél que tenga más grupos libres a los que ir (con el fin de que el número de grupos no crezca de forma desorbitada), y el grupo de destino es el primero que se encuentre libre.

Se ha implementado un simulador de colisiones con el que se pueden realizar un gran número de repeticiones para cada configuración de parámetros experimentales sin la intervención de usuarios reales.

4.1. Equipamiento

El simulador implementado permite la especificación de ciertos parámetros de simulación: las dimensiones de la superficie de trabajo, el número de usuarios involucrados y su respectivo número de controles. En nuestros experimentos, el valor para el parámetro "dimensiones de la superficie" se ha fijado a 1024×768 píxeles para simular una superficie comercial real como es la Microsoft Pixelsense 1.0.

El simulador sitúa un cierto número de controles de forma aleatoria sobre

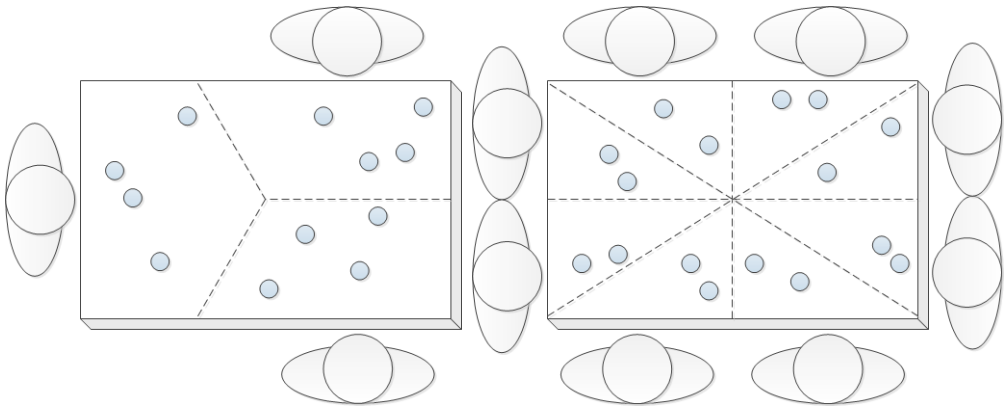


Figura 4.1: Distribución de los controles hecha por el simulador de colisiones en una situación hipotética con $U = 3$ y $C = 4$ (izquierda) y con $U = 8$ y $C = 2$ (derecha).

la superficie, atendiendo a dos parámetros: el número de usuarios U que pueden estar interactuando al mismo tiempo con la aplicación, y el número de controles por usuario C que cada uno posee. La distribución de los controles se ha realizado bajo la hipótesis de territorialidad ya comentada en capítulos anteriores, bajo la cual se supone que los usuarios territorializan el espacio frente a ellos y no suelen invadir las áreas de otros Scott y otros (2004). Así, el simulador divide la superficie en U secciones, como si los usuarios estuvieran situados a su alrededor (simulando una situación real como la que puede verse en la figura 2.6), asignando el mismo espacio de trabajo a cada uno. A continuación, coloca C controles de un mismo usuario hipotético agrupados frente a la posición en la que éste se sentaría, en posiciones aleatorias. Esta distribución del espacio puede observarse en la figura 4.1 con un ejemplo para tres usuarios con cuatro controles por usuario ($U = 3, C = 4$, a la izquierda) y con ocho usuarios con dos controles por cada uno ($U = 8, C = 2$, a la derecha). Con esta configuración realizada, se generan aleatoriamente todas las posibles colisiones entre los controles de diferentes usuarios. Todas estas secuencias de interacciones simuladas formarán parte de los conjuntos de datos usados como entrada en el estudio.

4.2. Procedimiento de los Experimentos

Se han conducido un conjunto de experimentos para averiguar si hay diferencias significativas entre las versiones heurísticas presentadas cuando el número de usuarios U y el número de controles C para cada usuario varían.

U	Low	2, 3
	Medium	4, 5, 6
	High	7, 8, 9
C	Low	1, 2, 3
	Medium	4, 5, 6, 7
	High	8, 9, 10

Tabla 4.1: Clasificación de U y C por densidades.

El análisis comparativo de las heurísticas propuestas se ha realizado, por una parte, en términos del coste computacional medido empíricamente (ver sección 4.3.2, y, por otra parte, en términos de una serie de métricas que evalúan la eficacia de cada versión distribuyendo los controles entre grupos (ver sección 4.4.1).

En el diseño experimental, el número de usuarios U ha sido variado de 2 a 9, y el número de controles por usuario C, de 1 a 10. Con el fin de simplificar el análisis, estos valores han sido organizados en grupos de densidad (baja, media y alta), tal y como se muestra en la tabla 4.1. De entre todas las posibles combinaciones de valores para C y U, se ha seleccionado aleatoriamente un subconjunto de 3 candidatos representativos para cada una de las diferentes densidades. Con estas combinaciones, se han generado los correspondientes conjuntos de datos de acuerdo con las condiciones descritas en la sección 4.1, cambiando tanto la posición de los controles como el orden en el que se producen las colisiones. Todo esto ha resultado en 50 pruebas por combinación, lo que viene a resultar en 450 pruebas diferentes.

Una prueba, pues, se refiere a una combinación específica de U y C junto con una secuencia ordenada de eventos de colisión entre los controles. Todo evento de colisión producido en la secuencia de una prueba es procesado por todas las versiones del algoritmo.

4.3. Estudio de la Complejidad

4.3.1. Análisis Teórico

En cuanto a la complejidad computacional teórica, la elección de una versión u otra es irrelevante, pues todas ellas tienen un coste $O(K^4)$, considerando K el número de segmentos o grupos actuales. En el peor caso, que correspondería a la situación donde cada control se encuentra en un grupo distinto y, por tanto, existe un segmento por control, K tiende al número de controles n en la aplicación. En este caso, la complejidad sería del orden

de $O(n^4)$, tal y como se detalla a continuación a partir del análisis de los algoritmos descritos en la sección 3.3.2. Consideremos los siguientes casos:

1. Si la colisión se produce entre controles que ya han colisionado previamente, o que se encuentran en grupos distintos, el coste es $\Theta(1)$.

2. Si la colisión se produce entre controles dentro de un mismo grupo, una parte del algoritmo tiene un coste $O(3n + K^2\frac{K^2-K}{2})$, más además un sobrecoste en los siguientes casos:

   a) Si no hay necesidad de crear un grupo nuevo, ya que uno de los controles que han colisionado puede moverse a un grupo existente y activo, se aplica la heurística para elegir cuál es el que debe migrar, lo que añade un coste de $O(4K)$. Por otra parte, si no se puede mover a ningún grupo existente, pero hay alguno inhabilitado, el sobrecoste de reactivarlo y poner el control dentro es $\Theta(K + 2n)$. En cambio, si no existen grupos inhabilitados, el sobrecoste de crear uno nuevo y mover el control es $\Theta(n)$.

   b) Si se pueden fusionar dos grupos, se añade un sobrecoste de $O(2n+K)$.

Así, en el peor caso cuando los controles que colisionan están en el mismo grupo es cuando hay que elegir mediante heurísticas qué control mover y a qué contenedores de los disponibles y, además, se pueden fusionar dos de los grupos existentes. En este caso, el coste total sería el que se muestra en la ecuación 4.1.

$$O\left(5n + \frac{K^4 - K^3}{2} + 5K\right) = O\left(5n + \frac{K^4 - K^3 + 10K}{2}\right) \tag{4.1}$$

Teniendo en cuenta que K tiende a n, el coste se podría expresar con la cota mostrada en la ecuación 4.2.

$$O\left(5n + \frac{n^4 - n^3 + 10n}{2}\right) = O\left(\frac{n^4 - n^3 + 20n}{2}\right) \approx O\left(n^4\right) \tag{4.2}$$

Cabe destacar que el coste anterior no tiene en cuenta que las estructuras de datos estáticas pueden sobrepasar su capacidad y, por tanto, haya que asignar más memoria. Esta situación es extrema, puesto que, en una situación real, el número de controles sobre la mesa están limitados por las dimensiones físicas de la superficie. No obstante, en caso de necesitar una reasignación de memoria, el coste de esta operación sería de $\Theta(n^2)$ para copiar el contenido de todos los vectores y matrices a las nuevas estructuras, aunque sería un coste puntual añadido a la operación de añadir un control nuevo.

4.3.2. Evaluación Experimental

A pesar de ser $O(n^4)$ un gran coste teórico, empíricamente el tiempo no resulta un factor crítico dadas las restricciones físicas de las aplicaciones que pueden hacer uso de este algoritmo. Recordemos que, en una situación en la que varios usuarios estén interactuando al mismo tiempo sobre una mesa interactiva, el número de dichos usuarios y el número de controles que posee cada usuario son limitados.

En la ejecución de las diferentes pruebas descritas en la sección 4.2, se ha almacenado, para cada combinación de los parámetros U y C, el tiempo t (en segundos) que han tardado en procesar cada una de las colisiones producidas (que, recordemos, son todas las posibles entre todos los controles de diferentes usuarios). Considerando $numColisiones$ como el número total de colisiones procesadas en cada prueba, se puede calcular el número de colisiones que sería posible procesar por unidad de tiempo, de la siguiente forma: $\frac{numColisiones}{t}$. Como el coste depende del número de controles que haya en un momento determinado sobre la superficie, se ha calculado este valor como $n = U\Delta C$, de forma que el menor valor de n es 2 y se obtiene con $U = 2$ y $C = 1$; y el mayor valor es 90, el cual se obtiene con $U = 9$ y $C = 10$. Para aquellas pruebas que presentan un mismo valor de n (por ejemplo, la que combina $U = 3$ con $C = 5$ y la que combina $U = 5$ con $C = 3$), se ha calculado el valor medio de colisiones por segundo que se pueden procesar. Los resultados se muestran en la figura 4.2 donde, a la izquierda, se puede ver la variación completa de número de controles n, y a la derecha se ha realizado un enfoque sobre los valores más altos de n (para una mejor visualización).

Como cabría esperar, a medida que se aumenta la cantidad de controles n, el algoritmo tarda más en procesar las colisiones y, por tanto, se pueden procesar menos por unidad de tiempo. No obstante, en el peor caso considerado (90 controles, que es un valor demasiado alto para ser realista actualmente), se pueden procesar 81.1 colisiones por segundo en la versión CRCM, 101.1 en FLCM; y 91.2 en RLCD, que son valores muy elevados. Además, como se puede observar en las gráficas, las tres versiones heurísticas se comportan de forma similar, pudiendo procesar unas pocas menos colisiones por segundo que la versión de control utilizada para comparar. Esto es así debido a que la evaluación de las heurísticas conlleva un coste. No obstante, vistos los resultados, se puede afirmar que el tiempo de ejecución parece no ser un factor crítico a la hora de elegir una versión u otra. Además, como se explicará en la sección 4.4.1, utilizar heurísticas presenta ciertas ventajas respecto a la versión desinformada.

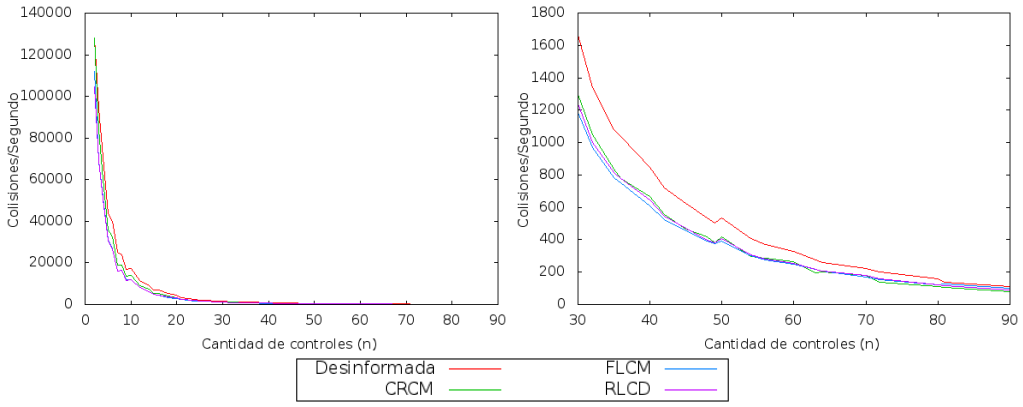


Figura 4.2: Evolución del número de colisiones que se pueden procesar por segundo, dependiendo del número de controles que haya sobre la superficie. A la izquierda, figura completa; a la derecha, detalle.

4.4. Estudio de la Efectividad en la Distribución de Controles

En esta sección se va a tratar de evaluar qué versión provee una mejor distribución de los controles entre los grupos.

4.4.1. Funciones de Bondad

Para estudiar la bondad de las diferentes versiones, se han implementado varias funciones. Para cada colisión observada, éstas representan cuán lejos se encuentra el estado alcanzado después de la colisión de la situación ideal en la que todos los controles han sido clasificados satisfactoriamente. Tal y como se han definido estas funciones, cuanto más cercano sea su valor a cero, mejor es la versión considerada. Un valor de cero correspondería a una clasificación ideal en la que todos los grupos contienen únicamente controles de un solo usuario y hay tantos grupos como usuarios.

Sea u_k el número de usuarios distintos que tienen controles dentro del grupo k; K, el número de grupos existentes actualmente; n_k, el número de controles dentro de k; U, el número de participantes; y C, el número de controles que posee cada usuario. Las funciones medidas están definidas a continuación.

- Función F_1 (exceso de usuarios): Idealmente, sólo debería haber un segmento por usuario. Esta función mide la diferencia entre el número

actual de usuarios con controles dentro de un grupo con respecto a la situación deseada. La ecuación 4.3 expresa esta función matemáticamente.

$$F_1 = \frac{\frac{\sum_{k \in Grupo}(u_k - 1)^2}{U}}{K} \tag{4.3}$$

- Función F_2 (exceso de controles): En la situación ideal, debería haber C controles por usuario en cada grupo. Esta función computa la diferencia entre el número de controles actual dentro de un contenedor con respecto a la situación deseada. En la ecuación 4.4 esta definida en términos matemáticos.

$$F_2 = \frac{\frac{\sum_{k \in Grupo}(n_k - C)^2}{C}}{K} \tag{4.4}$$

- Función F_3 (dispersión de controles): Los controles dentro de un grupo deberían pertenecer a un mismo usuario y, por tanto, encontrarse físicamente cerca los unos de los otros. Esta función mide la distancia entre los controles y el centroide de su respectivo contenedor. Cuanto más agrupados estén, más cercano a cero será el valor de esta función. Matemáticamente, F_3 está expresada en la ecuación 4.5.

$$F_3 = \frac{\sum_{k \in Grupo} \sum_{c \in k} \frac{\|centroid(k) - c_{pos}\|^2}{n_k}}{K} \tag{4.5}$$

- Función F_4 (falta o exceso de grupos): En la situación ideal, el número de contenedores K deberia ser igual al número de usuarios U, puesto que cada uno de ellos debería albergar a los controles de un usuario determinado. Por tanto, esta función mide la escasez (valores negativos) o exceso (valores positivos) de grupos con respecto a esta situación óptima. En la ecuación 4.6 está expresada esta función en forma matemática.

$$F_4 = K - U \tag{4.6}$$

Además de las funciones previas, se han realizado diversos refinamientos a la función F_1 dado que la heterogeneidad de cada grupo en términos de los usuarios incluidos es un factor clave de calidad. En este sentido, se desea prestar atención al grado de "densidad" de esta heterogeneidad en términos de la proporción de controles de un determinado usuario en un grupo con

respecto a la proporción de controles en ese mismo contenedor pertenecientes a otros usuarios. Esta densidad de heterogeneidad puede expresarse como se indica en la ecuación 4.7, siendo n_k^u el número de controles en un grupo k pertenecientes al usuario u.

$$\rho_k^u = \frac{n_k^u}{n_k} \tag{4.7}$$

A continuación se definen una serie de funciones de bondad adicionales que tienen en cuenta la anteriormente mencionada densidad de heterogeneidad de cada grupo.

- Función F_5 (proporción de controles: máximo): Toma el valor máximo de densidades en cada contenedor y obtiene el valor medio de entre todos los grupos. En otras palabras, mide la bondad de una versión determinada como cuán efectiva es en términos de incrementar la densidad media máxima de cada grupo. La ecuación 4.8 expresa esta función matemáticamente.

$$F_5 = 1 - \frac{\sum_{k \in Grupo} \text{máx}_u \text{ usuario con controles en } _k \, \rho_k^u}{K} \tag{4.8}$$

- Función F_6 (proporción de controles: máximo - mínimo): Toma la diferencia entre los valores de densidad máximo y mínimo en cada grupo y obtiene un valor medio. Lo que mide es la efectividad de una versión determinada para incrementar la diferencia entre los valores máximo y mínimo de densidad dentro de un segmento. En la ecuación 4.9 se expresa matemáticamente.

$$\begin{aligned} &F_6 = 1 - \frac{\sum_{k \in Grupo} x}{K}, \text{ donde} \\ &x = \begin{cases} \left(\max_u \rho_k^u\right) - \left(\min_u \rho_k^u\right) & \text{si } u_k > 1 \\ \rho_k^u & \text{en otro caso} \end{cases} \\ &\text{y siendo } u \text{ un usuario con controles dentro de } k. \end{aligned} \tag{4.9}$$

- Función F_7 (proporción de controles: máximo1 - máximo2): Toma la diferencia entre los dos valores máximos de densidad en cada contenedor y a continuación obtiene el valor medio de dichas diferencias. Esta función mide la efectividad de una heurística determinada en reducir el número de densidades máximas en un grupo. Los grupos con altos valores de densidad para diferentes usuarios provocarán potencialmente

un gran número de migraciones de controles en el futuro para alcanzar la situación ideal en la que todos los controles dentro de un segmento determinado pertenezcan a un mismo usuario. Esta función está expresada en términos matemáticos en la ecuación 4.10.

$$
\begin{aligned}
&F_7 = 1 - \frac{\sum_{k \in Grupo} x}{K}, \text{ donde} \\
&x = \begin{cases} \rho_k^u & \text{si } u_k = 1 \\ max1 - max2 & \text{en otro caso} \end{cases} \\
&max1 = \underset{u \text{ usuario con controles en } k}{\text{máx}} \rho_k^u \\
&max2 = \underset{u \text{ usuario con controles en } k \text{ excepto } umax}{\text{máx}} \rho_k^u \\
&umax = \underset{u \text{ usuario con controles en } k}{\arg\max'} \rho_k^u
\end{aligned}
\tag{4.10}
$$

- Función F_8 (proporción de controles: máximo - resto): Ésta es una versión más elaborada basada en F_6 y F_7, en la cual no sólo se calculan las diferencias entre las densidades máxima y mínima o entre las dos máximas dentro de cada grupo, sino que se toma la diferencia entre el valor más alto y la media del resto de densidades. La ecuación 4.11 la expresa matemáticamente.

$$
\begin{aligned}
&F_8 = 1 - \frac{\sum_{k \in Grupo} x}{K}, \text{ donde} \\
&x = \begin{cases} \rho_k^u & \text{si } u_k = 1 \\ max - rest & \text{en otro caso} \end{cases} \\
&max = \underset{u \text{ usuario con controles en } k}{\text{máx}} \rho_k^u \\
&rest = \frac{\sum_{u \text{ usuario con controles en } k \text{ excepto } umax} \rho_k^u}{u_k - 1}
\end{aligned}
\tag{4.11}
$$

Después de cada colisión procesada en cada experimento, se ha realizado el cómputo de cada una de las funciones de bondad sobre el estado de la clasificación, y se han guardado estos valores.

4.4.2. Resultados y Discusión

Cada experimento se ha nombrado de la forma *UxCy*, donde x es el número de usuarios involucrados e y, el número de controles que cada uno posee. Las tablas 4.2-4.9 muestran una ordenación de las versiones heurísticas (de la mejor a la peor) basada en cada función de bondad. Con el fin de aumentar la legibilidad de las tablas, los nombres de las versiones han sido substituidos por números. La versión 0 representa la anteriormente mencionada versión de control (que no contempla el factor que mide las distancias entre controles), y las versiones 1, 2 y 3 representan a las CRCM, FLCM y RLCD, respectivamente.

En las tablas, las diferentes versiones forman grupos (separados por guiones) si no hay diferencias significativas entre ellas. Dentro de cada grupo, están también ordenadas de acuerdo a su valor medio (de menor a mayor). Por ejemplo, en la tabla 4.2, la entrada para el experimento *U6C2* es "32-1-0". Este valor implica que, en términos de la función F_1, hay tres grupos de versiones estadísticamente diferentes, siendo la 3 (RLCD) y la 2 (FLCM) las mejores, seguidas por la 1 (CRCM) y, finalmente, por la 0 (versión de control), que es la que peor se comporta. No obstante, a pesar de que no hay diferencias significativas entre la 3 y la 2, el valor medio de F_1 para la versión 3 es menor que para la 2, y por eso ha sido listada primero en el grupo "32".

Para determinar dichos grupos de versiones, se han comprobado estadísticamente las diferencias entre sus valores para las funciones de bondad. Cuando se podía asumir normalidad en los datos (basada en test Shapiro-Wilk), se ha realizado un ANOVA para determinar si habían diferencias significativas entre las versiones. Y, en caso afirmativo, se han realizado comparaciones por parejas, usando el método de Bonferroni, para determinar qué versiones eran estadísticamente diferentes unas de otras. Sin embargo, cuando no se podía aceptar la asunción de normalidad, se ha realizado un test Kruskal-Wallis con sus comparaciones entre pares respectivas. De un modo u otro, la idea es ser capaz de decidir si la efectividad de una versión heurística difiere significativamente del resto para una función de bondad dada.

El análisis de la tabla 4.2 (F_1, exceso de usuarios) muestra que no hay una separación clara entre versiones para densidades bajas tanto de U como de C, aunque a veces la versión desinformada tomada como control parece separarse del resto. Esto indica que, cuando el número de usuarios interactuando o el número de controles por usuario es pequeño, utilizar una versión informada tiende a separar más rápidamente los controles de los diferentes usuarios que la versión desinformada. A medida que las densidades de U y C crecen, se hace más clara la diferencia entre versiones. La desinformada presenta los peores resultados, seguida por la versión 1 (CRCM) que sólo toma

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	3201	U7C3	3-2-10
		U3C1	0123	U5C3	32-10	U8C2	3210
		U3C3	3-201	U6C2	32-1-0	U9C1	23-01
	Medium	U2C5	312-0	U4C6	3-2-1-0	U7C4	3-2-1-0
		U2C6	312-0	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	3-21-0	U6C7	3-2-1-0	U9C5	3-2-1-0
	High	U2C8	3-120	U4C10	3-2-1-0	U7C10	3-2-1-0
		U2C9	312-0	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-21-0	U6C9	3-2-1-0	U9C9	3-2-1-0

Tabla 4.2: Clasificación de las versiones de mejor a peor para la función de bondad F_1.

en consideración la distancia a los grupos disponibles para escoger qué control mover en caso de colisión. La 2 (FLCM), que elige siempre el control más lejano al centroide del grupo actual, parece ser algo mejor, pero la combinación de las versiones CRCM y FLCM en la RLCD es la que supera a las otras, manteniendo agrupados, mayormente, los controles de un mismo usuario. La figura 4.3 muestra los valores medidos de F_1 tras cada colisión para un experimento con densidades medias tanto de U como de C. Tal y como se puede observar, el uso de heurísticas reduce claramente el exceso de usuarios dentro de los grupos.

Respecto a F_2, el exceso de controles (ver tabla 4.3), no hay diferencias significativas con respecto a las versiones 2 (FLCM) y 3 (RLCD), las cuales son mejores que las otras dos para combinaciones de densidades medias y altas de U y C. Esto significa que, cuando se intenta tener C controles en cada grupos (lo cual sería la situación ideal), la decisión correcta a tomar es mover el control que se encuentra más cerca del centroide de su propio contenedor. Para bajas densidades de controles por usuario, no hay diferencias claras entre versiones. Adicionalmente, algunas combinaciones de tamaño medio de usuarios y controles muestra un conjunto variable de versiones ganadoras. Estos resultados variables nos llevan a concluir que F_2 no es una función discriminante para medir la bondad de las versiones, tal y como indica la figura 4.4.

La función F_3 (ver tabla 4.4) mide la distancia media entre los controles clasificados en el mismo grupo y su centroide. Dado que esta función mide las distancias entre controles, es de esperar que las versiones heurísticas pre-

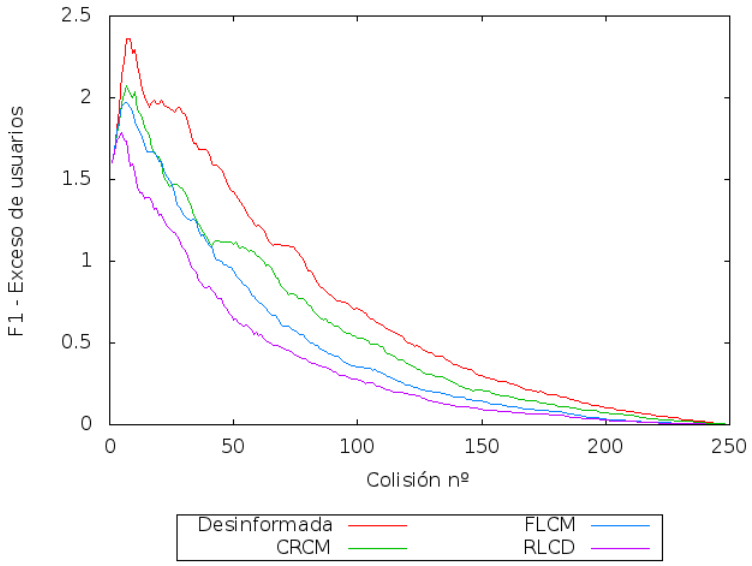


Figura 4.3: Evolución de F_1 con las colisiones producidas en el experimento *U5C5*.

		U					
		Low		Medium		High	
C	Low	U2C2	1302	U4C1	3201	U7C3	2301
		U3C1	0123	U5C3	3201	U8C2	2310
		U3C3	0132	U6C2	2-310	U9C1	23-01
	Medium	U2C5	1320	U4C6	3210	U7C4	23-01
		U2C6	13-20	U5C5	32-10	U8C6	23-0-1
		U3C7	3120	U6C7	32-01	U9C5	23-01
	High	U2C8	13-02	U4C10	321-0	U7C10	32-1-0
		U2C9	13-20	U5C8	32-1-0	U8C8	32-01
		U3C10	321-0	U6C9	32-10	U9C9	32-01

Tabla 4.3: Clasificación de las versiones de mejor a peor para la función de bondad F_2.

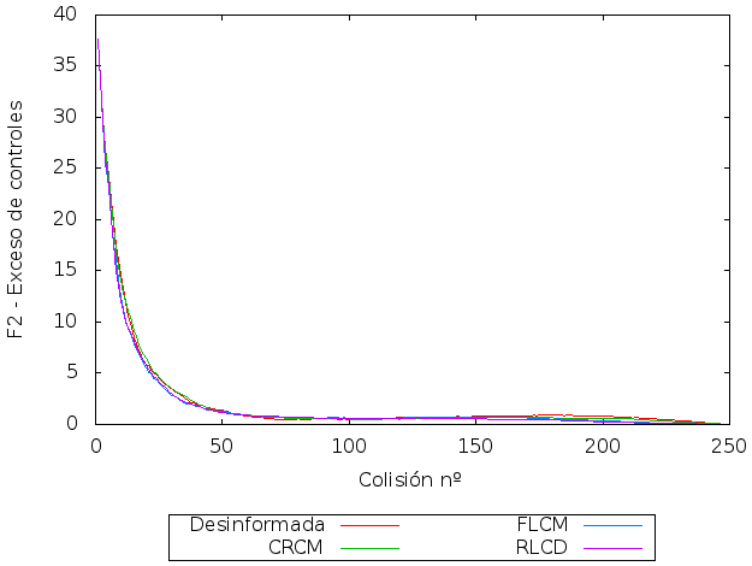


Figura 4.4: Evolución de F_2 con las colisiones producidas en el experimento *U5C5*.

senten un mejor valor que la versión desinformada (la cual escoge en función del número de categorías disponibles). Y, efectivamente, los resultados revelan que, para cualquier combinación de densidades medias y altas, hay una separación clara entre las versiones. La RLCD (número 3) es la vencedora a la hora de mantener juntos los controles más próximos los unos a los otros (ver figura 4.5), seguida en este orden por la 2 (FLCM) y la 1 (CRCM). La separación no es tan marcada para bajas densidades de C y medias/bajas densidades de U, pero la tendencia es la misma. No obstante, para una baja densidad de número de usuarios no hay diferencias significativas entre versiones, aunque RLCD parece mostrar valores medios menores de F_3.

Las diferencias entre versiones no son muy fuertes considerando F_4 (la falta o exceso de grupos), tal y como se puede observar en la figura 4.6 así como en la tabla 4.5. Aunque la RLCD parece superar a la versión desinformada para combinaciones de densidades medias y altas de U y C, es necesario considerar densidades altas de C para poder diferenciar entre las diferentes versiones heurísticas. En este caso, la que genera un número de segmentos más próximo al caso ideal es la versión 3 (RLCD), seguida por las otras dos heurísticas y, finalmente, la versión de control. A pesar de ser una métrica para evaluar cuán cerca está una versión del caso ideal, esta función es importante para medir la bondad de una heurística, pues un gran exceso en el número de grupos podría ser perjudicial para el coste computacional, que

		U					
		Low		Medium		High	
C	Low	U2C2	3210	U4C1	3201	U7C3	3-2-1-0
		U3C1	3210	U5C3	32-1-0	U8C2	3-21-0
		U3C3	3210	U6C2	32-1-0	U9C1	3210
	Medium	U2C5	3210	U4C6	3-21-0	U7C4	3-2-1-0
		U2C6	3210	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	321-0	U6C7	3-2-1-0	U9C5	3-2-1-0
	High	U2C8	3120	U4C10	3-21-0	U7C10	3-2-1-0
		U2C9	3120	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-21-0	U6C9	3-2-1-0	U9C9	3-2-1-0

Tabla 4.4: Clasificación de las versiones de mejor a peor para la función de bondad F_3.

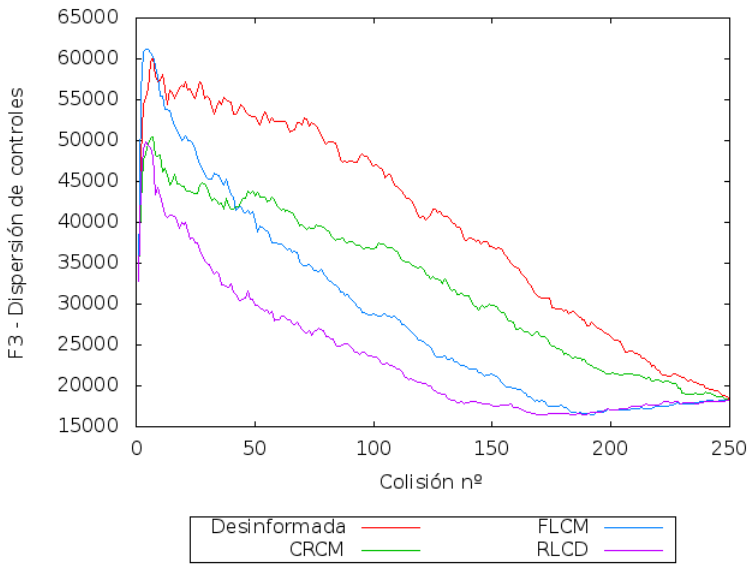


Figura 4.5: Evolución de F_3 con las colisiones producidas en el experimento *U5C5*.

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	1023	U7C3	1302
		U3C1	0123	U5C3	1320	U8C2	10-32
		U3C3	1032	U6C2	1-302	U9C1	01-23
	Medium	U2C5	1320	U4C6	3-12-0	U7C4	3120
		U2C6	3120	U5C5	321-0	U8C6	31-2-0
		U3C7	3-12-0	U6C7	321-0	U9C5	132-0
	High	U2C8	31-02	U4C10	3-21-0	U7C10	3-21-0
		U2C9	31-20	U5C8	3-21-0	U8C8	3-21-0
		U3C10	321-0	U6C9	32-1-0	U9C9	3-21-0

Tabla 4.5: Clasificación de las versiones de mejor a peor para la función de bondad F_4.

es función del número de grupos total K: $O(K^4)$. La figura 4.6 muestra la evolución de F_4 con respecto al número de colisiones procesadas. Al principio hay menos grupos de los deseados, y, a partir de la sexagésima colisión, hay un número excesivo de éstos. Sin embargo, este exceso es bastante pequeño y, por lo tanto, no es un problema crítico para el coste computacional.

Las funciones F_5 (tabla 4.6), F_6 (tabla 4.7), F_7 (tabla 4.8) y F_8 (tabla 4.9) miden la heterogeneidad interior a un grupo en términos de la proporción de controles pertenecientes a cada usuario que tiene controles en dicho grupo. A pesar de que estas funciones difieren entre ellas, esencialmente miden un factor similar (la heterogeneidad de densidad de controles), y, por tanto, en teoría deberían mostrar una capacidad similar para discriminar entre versiones. Los resultados corroboran esta suposición, y, tal y como ocurre con las otras funciones de bondad explicadas anteriormente, las diferencias entre versiones se hacen más evidentes cuando se manejan combinaciones de densidades medias y altas tanto de U y de C. En este caso, tal y como se puede observar en la figura 4.7, la versión RLCD es superior al resto, seguida, en este orden, por las versiones 2 (FLCM), 1 (CRCM) y 0 (desinformada). Además, cuando se trata con bajas densidades de U o C, las versiones CRCM y FLCM se degradan y resultan tan efectivas como la versión de control, pero, aun así, la RLCD destaca como la mejor en la mayor parte de los casos.

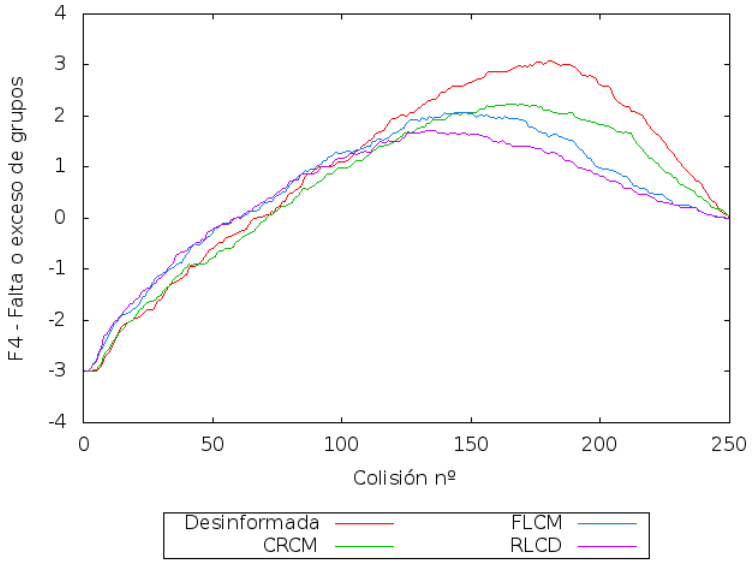


Figura 4.6: Evolución de F_4 con las colisiones producidas en el experimento *U5C5*.

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	3021	U7C3	3-2-1-0
		U3C1	0123	U5C3	32-1-0	U8C2	32-1-0
		U3C3	3-210	U6C2	32-1-0	U9C1	32-10
	Medium	U2C5	312-0	U4C6	3-2-1-0	U7C4	3-2-1-0
		U2C6	3120	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	3-21-0	U6C7	3-2-1-0	U9C5	3-2-1-0
	High	U2C8	31-20	U4C10	3-2-1-0	U7C10	3-2-1-0
		U2C9	31-20	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-2-1-0	U6C9	3-2-1-0	U9C9	3-2-1-0

Tabla 4.6: Clasificación de las versiones de mejor a peor para la función de bondad F_5.

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	3021	U7C3	3-2-1-0
		U3C1	0123	U5C3	32-1-0	U8C2	32-1-0
		U3C3	3-210	U6C2	32-1-0	U9C1	32-10
	Medium	U2C5	312-0	U4C6	3-2-1-0	U7C4	3-2-1-0
		U2C6	3120	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	3-21-0	U6C7	3-2-1-0	U9C5	32-10
	High	U2C8	31-20	U4C10	3-2-1-0	U7C10	3-2-1-0
		U2C9	31-20	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-2-1-0	U6C9	32-1-0	U9C9	3-2-1-0

Tabla 4.7: Clasificación de las versiones de mejor a peor para la función de bondad F_6.

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	3021	U7C3	3-2-1-0
		U3C1	0123	U5C3	32-1-0	U8C2	32-1-0
		U3C3	3-210	U6C2	32-1-0	U9C1	32-10
	Medium	U2C5	312-0	U4C6	3-2-1-0	U7C4	3-2-1-0
		U2C6	3120	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	3-21-0	U6C7	3-2-1-0	U9C5	32-1-0
	High	U2C8	31-20	U4C10	3-2-1-0	U7C10	3-2-1-0
		U2C9	31-20	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-2-1-0	U6C9	32-1-0	U9C9	3-2-1-0

Tabla 4.8: Clasificación de las versiones de mejor a peor para la función de bondad F_7.

		U					
		Low		Medium		High	
C	Low	U2C2	3120	U4C1	3021	U7C3	3-2-1-0
		U3C1	0123	U5C3	32-1-0	U8C2	32-1-0
		U3C3	3-210	U6C2	32-1-0	U9C1	32-10
	Medium	U2C5	312-0	U4C6	3-2-1-0	U7C4	3-2-1-0
		U2C6	3120	U5C5	3-2-1-0	U8C6	3-2-1-0
		U3C7	3-21-0	U6C7	3-2-1-0	U9C5	32-1-0
	High	U2C8	31-20	U4C10	3-2-1-0	U7C10	3-2-1-0
		U2C9	31-20	U5C8	3-2-1-0	U8C8	3-2-1-0
		U3C10	3-2-1-0	U6C9	32-1-0	U9C9	3-2-1-0

Tabla 4.9: Clasificación de las versiones de mejor a peor para la función de bondad F_8.

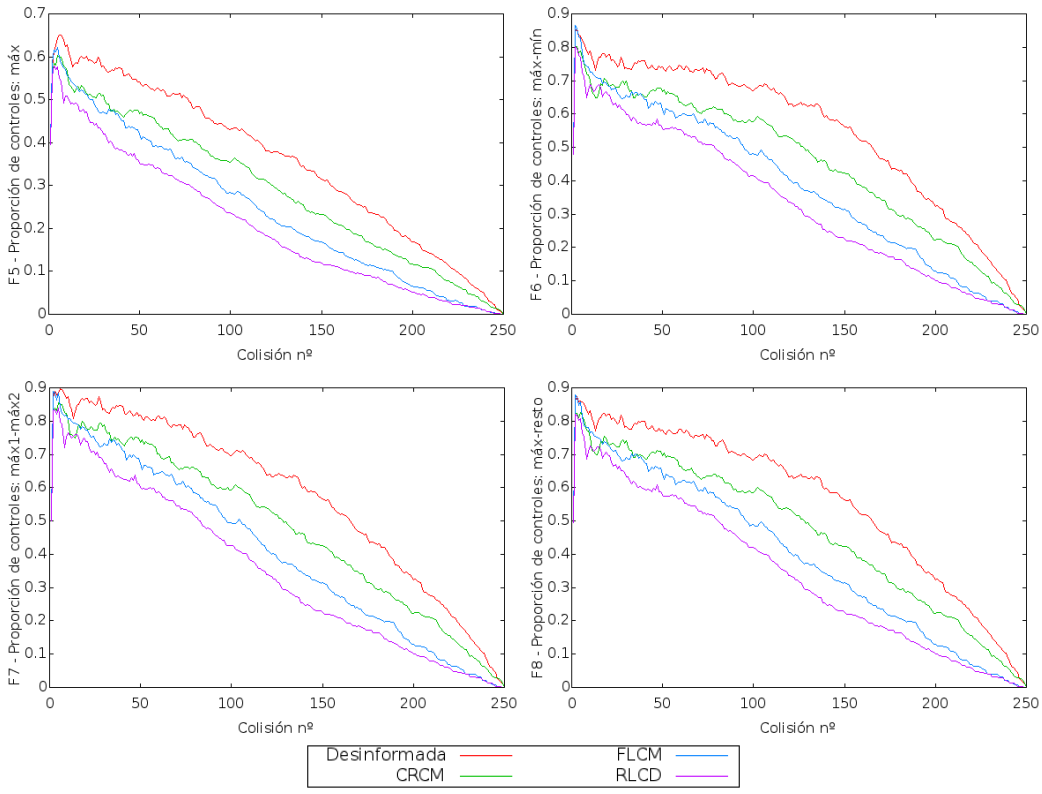


Figura 4.7: Evolución de F_5, F_6, F_7 y F_8 con las colisiones producidas en el experimento *U5C5*.

4.5. Conclusiones

En este capítulo se han evaluado las diferentes versiones heurísticas del algoritmo de diferenciación presentadas en el capítulo 3 con respecto a dos características: su coste computacional y su efectividad a la hora de mover los controles entre grupos, que indica cuán rápido convergen los algoritmos a la situación ideal donde se tenga tantos grupos como usuarios y todos los controles dentro de un grupo pertenezcan al mismo dueño. Se han comparado las tres versiones entre ellas y también con una versión desinformada que mueve los controles sin aplicar ninguna heurística basada en distancias.

Respecto al coste computacional, las tres versiones (y la desinformada) presentan el mismo coste asintótico $O(n^4)$, siendo n el número de controles en la aplicación. Pese a esta gran cota, se ha comprobado empíricamente que el número de colisiones que son capaces de procesar es bastante alto incluso con configuraciones experimentales que presentan un número de controles muy elevado, si se tiene en cuenta que el tamaño de las superficies interactivas actuales no permite colocar muchos elementos sobre ésta. También se ha podido observar que las tres heurísticas presentan resultados similares, mientras que la versión desinformada permite procesar un mayor número de colisiones por segundo. Esto ocurre porque la operación de elegir el control a ser migrado de grupo se realiza de forma trivial.

Atendiendo a la efectividad realizando las agrupaciones, se han definido una serie de funciones de bondad que sirven como métricas para esta característica. Según éstas, el uso de heurísticas presenta mejores resultados que escoger como control a ser movido aquél que simplemente presente un mayor número de destinos disponibles (o arbitrariamente en caso de empate). Este comportamiento se acentúa a medida que el número de usuarios interactuando en la aplicación U y el número de controles por usuario C aumentan. No obstante, es importante notar que, incluso con valores pequeños de estos dos parámetros, la distribución de controles realizada por algunas versiones heurísticas es normalmente mejor que la obtenida con la versión de control desinformada. Además, aunque tallas grandes de U y C podrían ser consideradas poco realistas actualmente debido a las acotadas dimensiones de las mesas interactivas, la combinación de valores medios de U y C todavía soporta estas conclusiones. Los resultados muestran también que la versión heurística RLCD, que es la más elaborada en términos de medición de la heterogeneidad dentro de cada grupo, es la ganadora bajo todas las circunstancias consideradas excepto cuando se trabaja con valores bajos de densidad de usuarios o controles. Esto implica que no siempre es preferible mover un control que está más cerca de un contenedor libre de colisiones si también está muy cerca del resto de controles en su grupo actual, y viceversa.

5 Conclusiones y Trabajo Futuro

5.1. Conclusiones

Este trabajo empieza exponiendo la necesidad de realizar una distinción entre los usuarios que están interactuando al mismo tiempo alrededor de una mesa interactiva. Esta información podría permitir al sistema adaptar sus interfaces automáticamente, permitiendo a los usuarios concentrarse en sus tareas primarias.

La revisión de otros trabajos relacionados con el tema parece llevar a la conclusión de que abordar este problema sin añadir hardware externo a la superficie interactiva parece ser preferible. Por tanto, en este trabajo se ha propuesto una solución puramente software para el problema de la diferenciación de usuarios, simplemente teniendo en cuenta ciertas características discriminantes que diferencian a unos usuarios de otros cuando interactúan con este tipo de aplicaciones.

Se han identificado algunas de estas características o factores diferenciadores, algunos ya explorados en la literatura. Tras la exploración de los mismos, se observa que utilizar un solo factor no permite distinguir correctamente entre todos los usuarios posibles que puedan situarse en la mesa, independientemente de su posición e interacciones, y se intuye que una combinación de varios de ellos pueda proporcionar resultados más robustos.

Se ha implementado un algoritmo de segmentación que mantiene los controles sobre la superficie en diferentes grupos si se asume que pertenecen a usuarios distintos, consiguiendo así la diferenciación de los usuarios manipulando dichos controles. Para ello, se ha explorado un factor todavía no estudiado en profundidad por otros autores: la manipulación simultanea de los controles. Observaciones realizadas por otros investigadores sugieren que

los usuarios no suelen manejar los elementos de interfaz con ambas manos, por tanto, la idea de este factor es que, si en un momento determinado, dos controles están siendo manipulados a la vez, es porque pertenecen a usuarios distintos.

El algoritmo sigue un diseño genérico de forma que ciertas operaciones básicas pueden ser implementadas de formas distintas, obteniendo así diferentes versiones. Esta característica es particularmente interesante, ya que permitirá en el futuro extender el algoritmo fácilmente con información sobre otros factores, simplemente sobrescribiendo estas operaciones. En este trabajo se han implementado tres versiones diferentes para el movimiento de los controles entre grupos basadas en diferentes heurísticas. Éstas han sido denominadas CRCM (*Closest Remote Centroid Migration*), FLCM (*Furthest Local Centroid Migration*) y RLCD (*Remote-Local Centroids Difference*), e incluyen los beneficios de otro factor: la posición en la que están situados los controles sobre la mesa. Mientras CRCM selecciona para migrar aquél control que se encuentre más cerca de un grupo disponible, FLCM elige el que se encuentre más lejos del centroide de su propio contenedor. RLCD es una versión híbrida de las dos primeras.

Posteriormente, se ha realizado una comparación en un entorno simulado de las tres versiones heurísticas entre ellas y también contra otra, menos informada, que se ha utilizado como control. Ésta selecciona como elemento a mover aquél que tenga más grupos disponibles a los que ir y, en el caso de un muy probable empate, lo selecciona arbitrariamente. La comparación se ha realizado de acuerdo a ocho funciones de bondad que miden la calidad de los movimientos efectuados entre los controles, y los resultados han mostrado que, en general, la versión RLCD es la que mejor se comporta. Las diferencias entre versiones se acentúan más a medida que el número de controles sobre la aplicación crece. Cuando se consideran pocos usuarios interactuando cada uno con pocos controles, la versión RLCD parece no destacar. Sin embargo, en estas situaciones cuando la superficie no está muy poblada, la necesidad de territorializar la superficie automáticamente no es muy grande y, por tanto, podría no ser necesario la aplicación de algoritmos de diferenciación de usuarios como el presentado en este trabajo.

Las diferentes versiones han sido comparadas también con respecto a su coste computacional. Por una parte, se ha observado que las distintas versiones heurísticas no parecen diferir mucho unas de otras en este aspecto, aunque la versión desinformada usada como control parece ser ligeramente más rápida. Esto era de esperar, puesto que la decisión de mover un control de un grupo a otro la toma de una forma más "ciega", lo que perjudica su comportamiento de acuerdo a las funciones de bondad comentadas anteriormente. No obstante, cualquiera de las versiones heurísticas permite procesar

un gran número de colisiones por segundo, por lo que la eficiencia computacional no parece ser un problema.

En resumen, el algoritmo diseñado en este trabajo inicia la exploración del factor "manipulación simultánea de controles" para conseguir la diferenciación de usuarios. Sin embargo, no se consigue una distinción completa hasta que se producen muchas colisiones entre los diferentes elementos de cada usuario.

5.2. Trabajo Futuro

Como trabajo futuro, se seguirá probando este algoritmo con la heurística RLCD, y se realizarán experimentos con usuarios para poder probar su efectividad real en una tarea que requiera realizar diferenciación de usuarios. También se combinará este factor con los otros identificados en este trabajo (y algunos ya explorados por otros autores), con el objetivo de comprobar si se incrementa la robustez y la precisión del diferenciador cuando se combinan varios factores.

Finalmente, también será objeto de estudio determinar en qué tipo de aplicaciones y tareas resulta más o menos conveniente usar un factor u otro, ya que puede ser que no todos sean útiles en todos los contextos.

5.3. Publicaciones

Este trabajo ha derivado en la publicación de un artículo de investigación:

- Fernando García, Javier Jaén, Alejandro Catalá. Evaluating Heuristics for Tabletop User Segmentation Based on Simultaneous Interaction. En *Expert Systems with Applications*, 40(14), págs. 5578-5587. Octubre de 2013 (JCR, IF:1.854, 31/114 1er tercil T1).

Además, también se han obtenido otras publicaciones relacionadas con el estudio de interfaces gráficas de usuario e interfaces tangibles de usuario (Fitzmaurice y otros, 1995; Ishii y Ullmer, 1997):

- Alejandro Catalá, Fernando García, Patricia Pons, Javier Jaén, Jose Antonio Mocholí. AGORAS: Towards collaborative game-based learning experiences on surfaces. En *Proceedings of the International Conference on Cognition and Exploratory Learning (CELDA 2012)*, págs. 147-154. IADIS. Octubre de 2012.

- Alejandro Catalá, Fernando García, Jose Azorín, Javier Jaén, Jose Antonio Mocholí. Exploring Direct Communication and Manipulation on Interactive Surfaces to Foster Novelty in a Creative Learning Environment. En *International Journal of Computer Science Research and Application*, 2(1), págs. 15-24. Edición especial, artículos extendidos seleccionados de ICVL2011 (aceptación del 20 %).

- Alejandro Catalá, Javier Jaén, Fernando García, Patricia Pons. Ownership and Dominance in Tabletop Based Collaboration for Creative Tasks. En *Actas de las Jornadas sobre aprendizaje colaborativo en entornos virtuales de la red*. Red temática sobre Aprendizaje Colaborativo en Entornos Virtuales (RACEV), 2012.

- Alejandro Catalá, Fernando García, Javier Jaén, Jose Antonio Mocholí. TangiWheel: A widget for Manipulating Collections on Tabletop Displays Supporting Hybrid Input Modality. En *Journal of Computer Science and Technology*, 27(4), págs. 811-829. Julio de 2012 (JCR, IF:0.656).

- Alejandro Catalá, Fernando García, Jose Azorín, Javier Jaén, Jose Antonio Mocholí. Exploring Direct Communication and Manipulation on Interactive Surfaces to Foster Novelty in a Creative Learning Environment. En *Proceedings of the International Conference on Virtual Learning (ICVL'11)*. Distinguido con el premio a la excelencia de la conferencia.

- Fernando García, Alejandro Catalá, Javier Jaén, Jose Antonio Mocholí. Una Interfaz Tangible para la Navegación Genérica de Estructuras de Colección. En *Actas de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD2011)*, págs. 1031-1044, 2011.

Bibliografía

Michelle Annett, Tovi Grossman, Daniel Wigdor, y George Fitzmaurice. Medusa: a proximity-aware multi-touch tabletop. En *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, páginas 337–346, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047240. URL `http://doi.acm.org/10.1145/2047196.2047240`.

Bojan Blažica, Daniel VladušIč, y Dunja Mladenić. Mti: A method for user identification for multitouch displays. *International Journal of Human-Computer Studies*, 71(6):691–702, Junio 2013. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2013.03.002. URL `http://dx.doi.org/10.1016/j.ijhcs.2013.03.002`.

Stéphanie Buisine, Guillaume Besacier, Améziane Aoussat, y Frédéric Vernier. How do interactive tabletop systems influence collaboration? *Computers in Human Behavior*, 28(1):49–59, Enero 2012. ISSN 0747-5632. doi: 10.1016/j.chb.2011.08.010. URL `http://dx.doi.org/10.1016/j.chb.2011.08.010`.

Alejandro Catala, Fernando Garcia-Sanjuan, Javier Jaen, y Jose A. Mocholi. Tangiwheel: A widget for manipulating collections on tabletop displays supporting hybrid input modality. *Journal of Computer Science and Technology*, 27(4):811–829, 2012.

Chi Tai Dang, Martin Straub, y Elisabeth André. Hand distinction for multi-touch tabletop interaction. En *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pági-

nas 101–108, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-733-2. doi: 10.1145/1731903.1731925. URL http://doi.acm.org/10.1145/1731903.1731925.

Dominique Decouchant, Sonia Mendoza, Gabriela Sánchez, y José Rodríguez. Adapting groupware systems to changes in the collaborator's context of use. *Expert Systems with Applications*, 40(11):4446–4462, Septiembre 2013. ISSN 0957-4174. doi: 10.1016/j.eswa.2013.01.043. URL http://dx.doi.org/10.1016/j.eswa.2013.01.043.

Paul Dietz y Darren Leigh. Diamondtouch: a multi-user touch technology. En *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, páginas 219–226, New York, NY, USA, 2001. ACM. ISBN 1-58113-438-X. doi: 10.1145/502348.502389. URL http://doi.acm.org/10.1145/502348.502389.

Pierre Dillenbourg y Michael Evans. Interactive tabletops in education. *International Journal of Computer-Supported Collaborative Learning*, 6:491–514, 2011. doi: 10.1007/s11412-011-9127-7.

K. C. Dohse, Thomas Dohse, Jeremiah D. Still, y Derrick J. Parkhurst. Enhancing multi-user interaction with multi-touch tabletop displays using hand tracking. En *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, ACHI '08, páginas 297–302, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3086-4. doi: 10.1109/ACHI.2008.11. URL http://dx.doi.org/10.1109/ACHI.2008.11.

George W. Fitzmaurice, Hiroshi Ishii, y William A. S. Buxton. Bricks: laying the foundations for graspable user interfaces. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, páginas 442–449, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1. doi: 10.1145/223904.223964. URL http://dx.doi.org/10.1145/223904.223964.

Chris Harrison, Munehiko Sato, y Ivan Poupyrev. Capacitive fingerprinting: exploring user differentiation by sensing electrical properties of the human body. En *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, páginas 537–544, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1580-7. doi: 10.1145/2380116.2380183. URL http://doi.acm.org/10.1145/2380116.2380183.

Uta Hinrichs, Sheelagh Carpendale, Stacey D. Scott, y Eric Pattison. Interface currents: Supporting fluent collaboration on tabletop displays. En *Proceedings of the 5th Symposium on Smart Graphics*, páginas 185–197, Frauenwörth Cloister, Germany, Agosto 2005. Springer-Verlag.

Eva Hornecker, Paul Marshall, Nick Sheep Dalton, y Yvonne Rogers. Collaboration and interference: awareness with mice or touch input. En *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, páginas 167–176, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-007-4. doi: 10.1145/1460563.1460589. URL `http://doi.acm.org/10.1145/1460563.1460589`.

Hiroshi Ishii y Brygg Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. En *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, CHI '97, páginas 234–241, New York, NY, USA, 1997. ACM. ISBN 0-89791-802-9. doi: 10.1145/258549.258715. URL `http://doi.acm.org/10.1145/258549.258715`.

KyungTae Kim, Tejas Kulkarni, y Niklas Elmqvist. Interaction workspaces: Indentity tracking for multi-user collaboration on camera-based multitouch tabletops. En *Workshop on Collaborative Visualization on Interactive Surfaces*, CoVIS'09, 2009.

Daniel Klinkhammer, Markus Nitsche, Marcus Specht, y Harald Reiterer. Adaptive personal territories for co-located tabletop interaction in a museum setting. En *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, páginas 107–110, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0871-7. doi: 10.1145/2076354.2076375. URL `http://doi.acm.org/10.1145/2076354.2076375`.

Russell Kruger, Sheelagh Carpendale, Stacey D. Scott, y Saul Greenberg. How people use orientation on tables: comprehension, coordination and communication. En *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, GROUP '03, páginas 369–378, New York, NY, USA, 2003. ACM. ISBN 1-58113-693-5. doi: 10.1145/958160.958219. URL `http://doi.acm.org/10.1145/958160.958219`.

G. Julian Lepinski, Tovi Grossman, y George Fitzmaurice. The design and evaluation of multitouch marking menus. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, páginas 2233–2242, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753663. URL `http://doi.acm.org/10.1145/1753326.1753663`.

Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring, y Saul Greenberg. Designing user-, hand-, and handpart-aware tabletop interactions with the touchid toolkit. En *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, páginas 21–30, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0871-7. doi: 10.1145/2076354.2076358. URL http://doi.acm.org/10.1145/2076354.2076358.

Roberto Martínez, Anthony Collins, Judy Kay, y Kalina Yacef. Who did what? who said that?: Collaid: an environment for capturing traces of collaborative learning at the tabletop. En *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, páginas 172–181, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0871-7. doi: 10.1145/2076354.2076387. URL http://doi.acm.org/10.1145/2076354.2076387.

Tobias Meyer y Dominik Schmidt. Idwristbands: Ir-based user identification on multi-touch surfaces. En *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, páginas 277–278, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0399-6. doi: 10.1145/1936652.1936714. URL http://doi.acm.org/10.1145/1936652.1936714.

Mark Micire, Eric McCann, Munjal Desai, Katherine M. Tsui, Adam Norton, y Holly A. Yanco. Hand and finger registration for multi-touch joysticks on software-based operator control units. En *Proceedings of IEEE Conference on Technologies for Practical Robot Applications*, TePRA 2011, páginas 88–93, Woburn, MA, USA, 2011. IEEE Computer Society. doi: 10.1109/TEPRA.2011.5753487. URL http://dx.doi.org/10.1109/TEPRA.2011.5753487.

Meredith Ringel Morris, Kathy Ryall, Chia Shen, Clifton Forlines, y Frederic Vernier. Beyond "social protocols": multi-user coordination policies for co-located groupware. En *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, páginas 262–265, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031648. URL http://doi.acm.org/10.1145/1031607.1031648.

Meredith Ringel Morris, Andreas Paepcke, y Terry Winograd. Teamsearch: Comparing techniques for co-present collaborative search of digital media. En *Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*, TABLETOP '06, páginas 97–104,

Washington, DC, USA, 2006a. IEEE Computer Society. ISBN 0-7695-2494-X. doi: 10.1109/TABLETOP.2006.32. URL http://dx.doi.org/10.1109/TABLETOP.2006.32.

Meredith Ringel Morris, Andreas Paepcke, Terry Winograd, y Jeannie Stamberger. Teamtag: exploring centralized versus replicated controls for co-located tabletop groupware. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, páginas 1273–1282, New York, NY, USA, 2006b. ACM. ISBN 1-59593-372-7. doi: 10.1145/1124772.1124964. URL http://doi.acm.org/10.1145/1124772.1124964.

Meredith Ringel Morris, Jarrod Lombardo, y Daniel Wigdor. Wesearch: supporting collaborative search and sensemaking on a tabletop display. En *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW '10, páginas 401–410, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-795-0. doi: 10.1145/1718918.1718987. URL http://doi.acm.org/10.1145/1718918.1718987.

Volker Roth, Philipp Schmidt, y Benjamin Güldenring. The ir ring: authenticating users' touches on a multi-touch display. En *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, páginas 259–262, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. doi: 10.1145/1866029.1866071. URL http://doi.acm.org/10.1145/1866029.1866071.

Kathy Ryall, Clifton Forlines, Chia Shen, y Meredith Ringel Morris. Exploring the effects of group size and table size on interactions with tabletop shared-display groupware. En *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, páginas 284–293, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031654. URL http://doi.acm.org/10.1145/1031607.1031654.

Kathy Ryall, Alan Esenther, Clifton Forlines, Chia Shen, Sam Shipman, Meredith Ringel Morris, Katherine Everitt, y Frederic D. Vernier. Identity-differentiating widgets for multiuser interactive surfaces. *IEEE Computer Graphics and Applications*, 26(5):56–64, Septiembre 2006. ISSN 0272-1716. doi: 10.1109/MCG.2006.108. URL http://dx.doi.org/10.1109/MCG.2006.108.

Albrecht Schmidt. Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2/3):191–199, 2000.

Dominik Schmidt, Ming Ki Chong, y Hans Gellersen. Handsdown: hand-contour-based user identification for interactive surfaces. En *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, páginas 432–441, New York, NY, USA, 2010a. ACM. ISBN 978-1-60558-934-3. doi: 10.1145/1868914.1868964. URL http://doi.acm.org/10.1145/1868914.1868964.

Dominik Schmidt, Ming Ki Chong, y Hans Gellersen. Idlenses: dynamic personal areas on shared surfaces. En *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, páginas 131–134, New York, NY, USA, 2010b. ACM. ISBN 978-1-4503-0399-6. doi: 10.1145/1936652.1936678. URL http://doi.acm.org/10.1145/1936652.1936678.

Stacey D. Scott, M. Sheelagh, T. Carpendale, y Kori M. Inkpen. Territoriality in collaborative tabletop workspaces. En *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, páginas 294–303, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031655. URL http://doi.acm.org/10.1145/1031607.1031655.

Dong Woo Seo y Jae Yeol Lee. Physical query interface for tangible augmented tagging and interaction. *Expert Systems with Applications*, 40(6): 2032–2042, Mayo 2013a.

Dong Woo Seo y Jae Yeol Lee. Direct hand touchable interactions in augmented reality environments for natural and intuitive user experiences. *Expert Systems with Applications*, 40(9):3784–3793, Julio 2013b.

Prasad Ramanahally Siddalinga. Sensor augmented large interactive surfaces. Tesis de máster, Iowa State University, Diciembre 2010. URL http://archives.ece.iastate.edu/archive/00000599/.

Norbert A. Streitz, Peter Tandler, Christian Müller-Tomfelde, y Shin'ichi Konomi. Roomware: Towards the next generation of human-computer interactions based on an integrated design of real and virtual worlds. En John M. Carroll, editor, *Human-Computer Interaction in the New Millennium*, volume 1, páginas 553–578. Addison Wesley, 2001.

Lucia Terrenghi, David Kirk, Abigail Sellen, y Shahram Izadi. Affordances for manipulation of physical versus digital media on interactive surfaces. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, páginas 1157–1166, New York, NY, USA, 2007. ACM.

ISBN 978-1-59593-593-9. doi: 10.1145/1240624.1240799. URL http://doi.acm.org/10.1145/1240624.1240799.

Edward Tse, Jonathan Histon, Stacey D. Scott, y Saul Greenberg. Avoiding interference: how people use spatial separation and partitioning in sdg workspaces. En *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, páginas 252–261, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031647. URL http://doi.acm.org/10.1145/1031607.1031647.

Cristian Andy Tănase, Radu-Daniel Vatabu, Stefan-Gheorghe Pentiuc, y Adrian Graur. Detecting and tracking multiple users in the proximity of interactive tabletops. *Advances in Electrical and Computer Engineering*, 8:61–64, 2008.

Feng Wang, Xiang Cao, Xiangshi Ren, y Pourang Irani. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. En *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, páginas 23–32, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-745-5. doi: 10.1145/1622176.1622182. URL http://doi.acm.org/10.1145/1622176.1622182.

Hong Zhang, Xing-Dong Yang, Barrett Ens, Hai-Ning Liang, Pierre Boulanger, y Pourang Irani. See me, see you: a lightweight method for discriminating user touches on tabletop displays. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, páginas 2327–2336, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208392. URL http://doi.acm.org/10.1145/2207676.2208392.