Información Importante

Universidad de la Sabana

Facultad de Ingeniería

Proyecto: "Visualizaciones de datos orientados en el tiempo sobre una mesa interactiva"

Proyecto para optar al grado de: Ingeniero Informático

Hazan Pérez Cardona

Profesor Director: Ricardo Sotaquirá Gutiérrez

Junio 2014
Chía, Colombia

Agradecimientos

Primero quisiera agradecer a mi familia por tenerme paciencia a la hora de realizar este trabajo de grado. También agradezco a mi director, Ricardo Sotaquirá, por sen un excelente guía y mostrarme lo que debería hacer en cada paso.

Agradezco también a las comunidades de sofware libre, especialmente a la comunidad de Ubuntu Colombia por ser tan atentos con mis preguntas técnicas. Agradezco a la comunidad de los foros de Processing por ayudarme con parte del código que implementé en el programa.

Y finalmente agradezco a la Universidad de la Sabana por proveer el espacio y el conocimiento para desarrollar este proyecto.

Tabla de Contenido

Introducción

La tecnología en el mundo actual converge muchas disciplinas, no sólo dentro de las ingenierías sino también el diseño gráfico, la psicología, el diseño industrial, la matemática, entre otras. Es dentro de esta convergencia que se consiguen nuevas formas de aprovechar la tecnología, y no sólo de acceder, sino también comprender la información.

Con este proyecto busco esta convergencia para crear nuevas e interesantes formas de mostrar la información e interactuar con ella, en este caso específico información de tráfico, y a partir de estas visualizaciones encontrar información más allá de lo evidente a primera vista.

Marco de referencia

Generalidades

El HCI (Human-Computer Interaction) en términos generales, podríamos decir que es la disciplina que estudia el intercambio de información mediante software entre las personas y las computadoras. Ésta se encarga del diseño, evaluación e implementación de los aparatos tecnológicos interactivos, estudiando el mayor número de casos que les pueda llegar a afectar. El objetivo es que el intercambio sea más eficiente: minimizar errores, incrementar la satisfacción, disminuir la frustración y, en definitiva, hacer más productivas las tareas que rodean a las personas y los computadores.

El Diseño de Interacción es un campo de desarrollo interdisciplinario que define el comportamiento de los productos y sistemas con los que interactúa el usuario. La práctica generalmente se centra en sistemas de tecnología complejos, como el software, dispositivos móviles y otros dispositivos electrónicos; sin embargo, también se puede aplicar a otro tipo de productos y servicios, e incluso a organizaciones. Todo esto con la finalidad de generar una experiencia de usuario que sea agradable. Para lograr esta finalidad, se deben realizar pruebas con usuarios que no estén implicados en el desarrollo del software, ya que, como estos serán los usuarios finales, su percepción del producto es la que cuenta.

La Visualización de información es el estudio de representaciones

visuales (interactivas) de datos abstractos para reforzar la cognición humana. Los datos abstractos incluyen información numérica y no numérica, como texto e información geográfica. Sin embargo, la visualización de información difiere de la visualización científica: “Es visualización de información cuando la representación espacial es escogida, y es visualización científica cuando la representación espacial es supuesta”.

La minería de datos, es un campo de las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos. Utiliza los métodos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos. El objetivo general del proceso de minería de datos consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible para su uso posterior. Además de la etapa de análisis en bruto, que involucra aspectos de bases de datos y gestión de datos, procesamiento de datos, el modelo y las consideraciones de inferencia, métricas de Intereses, consideraciones de la Teoría de la complejidad computacional, post-procesamiento de las estructuras descubiertas, la visualización y actualización en línea.

Hallazgos

Dada la investigación hecha, los modelos de visualización necesarios para el proyecto que se desea desarrollar, estos son modelos de visualización basados en el tiempo [Aigner 2011][Aigner 2007], están muy ligados a la minería de datos y al mismo tiempo son re-

presentaciones puntuales y únicas de los conjuntos de datos que representan. Existen soluciones genéricas [Zhao 2011] [Krstajić 2011], pero no deseamos implementar alguna de éstas, ya que no se pueden adaptar por completo al conjunto de datos que vamos a usar, y deseamos acercarnos más al lado del diseño gráfico para la realización de esta investigación.

Problema

Actualmente no existen herramientas de visualización de información que muestren datos orientados en el tiempo de una forma sencilla y animada, y en este caso, para visualizar y entender datos que tienen que ver con tráfico en una ciudad. Lo que se tiene son estudios estadísticos, usualmente con datos desactualizados [Cámara de Comercio de Bogotá 2010], los cuales son estudios completos pero muy generalizados. No se cuenta con datos detallados ni herramientas que permitan ver fácilmente esa información.

Justificación

Vivimos en un mundo donde el desarrollo tecnológico permite la recolección y almacenamiento de información sin mucho esfuerzo humano, pero dependemos de esa información para tomar decisiones en momentos críticos. Sin poder ver adecuadamente esta información las decisiones pueden tomar mucho tiempo, llevando a resultados desfavorables o incluso nefastos.

Además de esto, la tecnología actual permite formas de interacción más intuitivas que el uso de un mouse y un teclado, dándole a los usuarios la posibilidad de acceder a la información más rápido, más efectiva y sin la necesidad de escalar una curva de aprendizaje muy alta.

Como proyecto de investigación este trabajo busca explorar en el área de visualizaciones animadas, ya que en la investigación previa no se encontraron investigaciones que exploraran en esta dirección. También aportará a los campos de Visualización de Datos Orientados en el Tiempo y Experiencia de Usuario.

Objetivos

Objetivo general

Desarrollar un modelo de visualización de información de los datos orientados en el tiempo, que funcione en una mesa interactiva y sea entendible, fácil de manejar y minimalista, programado en Processing; con posibilidades de ser extensible a futuro en el caso que sea necesario.

Objetivos específicos

- Implementar una fuente de datos de imágenes, tomadas cada minuto, las cuales se usarán para la aplicación.

- Implementar y proponer una visualización animada y propia de los datos mencionadas, aplicando conceptos de HCI, interacción y diseño.

- Implementar la interacción con la plataforma de software reacTIVision mediante el uso de *fiducials*, para que la aplicación sea usada sobre una mesa interactiva.

- Implementar una función de compartir que permita exportar una imagen de la visualización a un servicio externo, como un correo electrónico.

- Probar el software implementado ante un grupo de tres personas con conocimientos sobre informática y diseño, para evaluar el funcionamiento y la usabilidad de la aplicación.

Alcance de la Investigación

Esta investigación pretende desarrollar una prueba de concepto de una aplicación para ver información de tráfico, en una sección de la ciudad de Bogotá, en un rango corto de tiempo (5 horas), y que pueda funcionar sobre una mesa interactiva.

Metodología

Este proyecto se basó en varias tecnologías y paradigmas existentes. Toda la programación se realizó sobre Processing (http://processing.org), un lenguaje de programación interpretativo de alto nivel desarrollado sobre Java, que permite programar nuevos formas de interacción, y animaciones, de una forma bastante sencilla. La página web de Processing contiene varios tutoriales en los que se demuestran varias características del lenguaje. El IDE de Processing también viene con muchos ejemplos de código, de los cuales se puede aprender y usar.

Para la interacción de la aplicación sobre la mesa interactiva se utilizó reacTIVision [Kaltenbrunner 2007], el cual se usó en dos partes: el programa interpretador de símbolos *fiducial*, junto con el montaje de la mesa interactiva; y la librería de reacTIVision para Processing, la cual trae un demo de funcionamiento de la misma en la cual se entienden las características y el alcance de reacTIVision desde el desarrollo de software.

Para que la aplicación fluyera y se manejara de la forma adecuada, se usó el paradigma de las máquinas de estado finito, que es un modelo computacional que realiza cómputos de forma automática sobre una entrada para producir una salida. En este caso la entrada es la posición del *fiducial* reacTIVision, y la salida es lo que muestra el programa.

Para hacer la interfaz se creó un arco, el cual se calcula matemáticamente con trigonometría. Para calcular adecuadamente la rotación del *fiducial* con respecto al arco se hizo un cambio en el paradigma y los ángulos no se calculan con pi (π) sino con tau (τ) [Hartl 2010]. Tau equivale a 2 pi.

$$\tau \equiv 2\pi$$

Resultados

Durante el desarrollo de este proyecto se encontraron varios obstáculos de los cuales se tuvo que encontrar alternativas para poder resolverlos. De esta misma forma se obtuvieron tres resultados:

1. Obtener una fuente de datos con datos, valga la redundancia, reales y confiables, con los cuales alimentar la aplicación

2. Realizar una visualización de estos datos orientados en el tiempo que tuviera sentido para quien maneje la aplicación

3. Implementar la interacción con la aplicación de modo que sea usable sobre una mesa interactiva.

Fuente de datos

Inicialmente se planeaba implementar una Base de Datos y poblarla con información de tráfico de una calle de la ciudad de Bogotá. Esta base de datos tendría información de velocidad media a una hora y minuto específicos, y este dato estaría georeferenciado con latitud y longitud para luego ser puesto sobre un mapa. El mayor inconveniente al tratar de hacer esta base de datos fue la falta de dicha fuente de datos que proveyera los datos formateados de esta forma específica. Se intentó hacer un *workarround* tomando imágenes de el servicio Google Maps, con su capa de tráfico activa,

pero estos datos son cualitativos (Google usa un código de colores: verde cuándo es fluido, amarillo cuando el trafico es suave, rojo cuando es pesado y rojo con negro cuando es muy pesado), por lo que ya no se contaría con datos de velocidad reales.

Además la extracción de los datos a partir de Google Maps debía ser hecha a mano, por lo que se decidió no crear una base de datos como la que fue descrita. En cambio se decidió tomar directamente las imágenes y mostrarlas en el proyecto.

Para crear esta colección de imágenes se decidió usar Waze en vez de Google Maps, ya que la información que muestra Waze es un poco más certera que la que muestra Google, además de ser más agradable visualmente hablando.

Ya que el desarrollo de este proyecto se hizo sobre un escritorio Linux, para tomar las imágenes se creó un script que tomaba automáticamente capturas de pantalla, cada cierto tiempo, a una ventana en específico. La ventana sobre la que se hicieron las capturas de pantalla era un navegador con el mapa de Waze cargado en una posición específica, la autopista norte de Bogotá entre la calle 180 y el peaje. El script, además de tomar las capturas de pantalla, se encargaba de cortar la imagen y guardarla con un nombre y un formato específico en una carpeta del computador.

El script es el siguiente:

```
watch -n 60 'import -window 0x5206b7f -crop 535x980+872+96 -display :0
traficoWaze$(date +%F_%T).png'
```

Watch es un programa que ejecuta un comando cada cierto tiempo, que se determina con la bandera “-n”. El comando repetido aquí es un programa que hace parte de ImageMagic. El programa import hace una captura de pantalla sobre una ventana específica con el comando “-window”, y corta la imagen que recién ha capturado con el comando “-crop”. “-display” selecciona la pantalla de X sobre la cual se ejecuta el comando, y la parte del final es el nombre del archivo que se guarda.

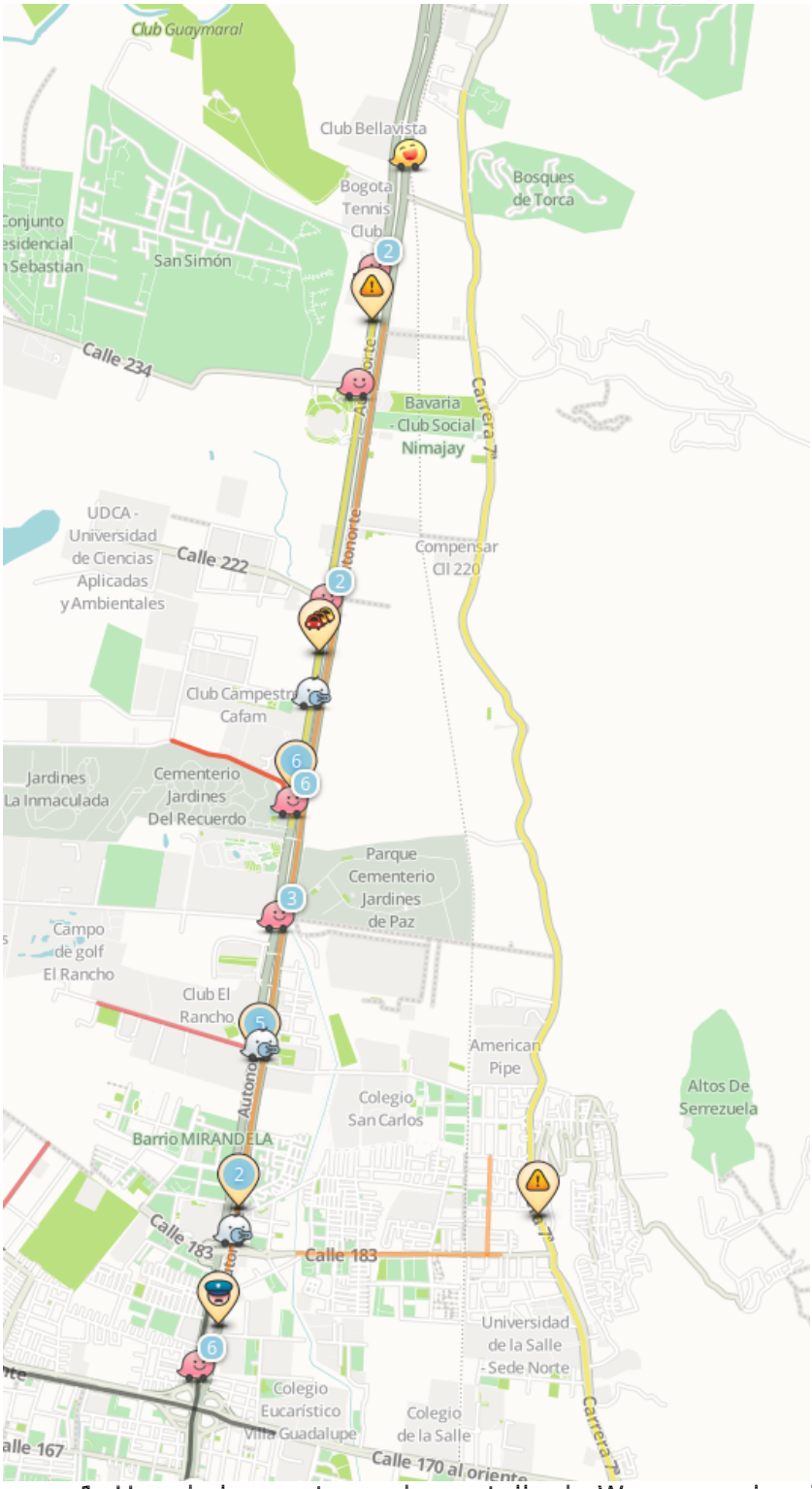


Imagen 1: Una de las capturas de pantalla de Waze con el script

Una vez que se tuvieran las imágenes, se podía proceder a animarlas.

La animación

Inicialmente, como se tenía la idea de tener la base de datos, se habían planeado animaciones diferentes para mostrar los datos cualitativos de una forma interesante, pero dada la fuente de datos esas ideas no fueron implementadas.

La animación que se hace con las imágenes es muy similar a un video. Se toman las imágenes en secuencia temporal y se combinan entre ellas para crear la ilusión de movimiento.

Esta parte de animación se implementó en Processing junto con algunas librerías de Java para poner las imágenes en la secuencia temporal adecuada. Una vez que las imágenes están en orden solo es cuestión de mostrarlas de forma que parecen animadas.

Processing da las herramientas necesarias para que hacer la animación sea bastante fácil. El lenguaje permite combinar dos imágenes progresivamente hasta que una se ponga encima de la otra de forma suave. El reto se encuentra en permitir este solapamiento de forma interactiva.

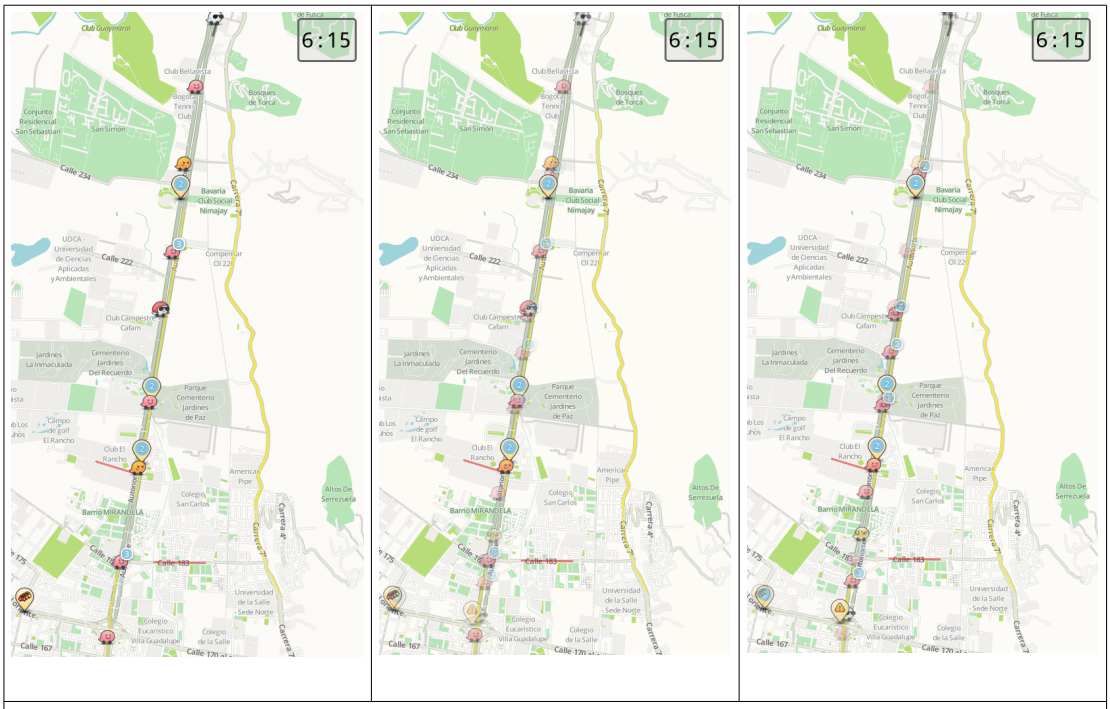


Imagen 2: Aqui se ve la transición suave entre una imagen y la siguiente

Hubo tres retos para lograr el efecto video: el manejo correcto de la memoria (que el programa no ocupara toda la memoria RAM del computador), que el video pudiera ir hacia adelante o hacia atrás en el tiempo, y que la interacción manejara la velocidad de la animación.

Por la forma como Processing maneja las imágenes, no era posible cargar todas las imágenes que hacen parte de la animación en memoria, por lo que fue necesario hacer un arreglo ordenado de las rutas de los archivos y luego cargar la imagen como tal en el momento que es mostrada. Claro que se cargan dos imágenes, la que se está mostrando ahora y la que se va a mostrar después, ya que

las dos se combinan para hacer el efecto de movimiento.

Esta solución no era obvia, por lo que pregunté en los foros de Processing, y amablemente me dieron la respuesta en forma de código. Para ordenar los archivos temporalmente, usando la fecha de último modificado, se debe crear un HashMap con los archivos y compararlos entre ellos usando la interfaz Comparator. Adicionalmente se aplicó un filtro para que solo se reconocieran los archivos de imagen, evitando así excepciones en la aplicación.

Para que el video pueda ir hacia adelante o hacia atrás solo es necesario recorrer el arreglo de rutas de archivos de imagen hacia adelante o hacia atrás, la parte algo difícil es cuando el video da la vuelta, es decir, cuando vuelve a empezar. La animación no se detiene cuando llega a la última hora registrada, sencillamente vuelve a empezar.

Y para que la interacción maneje la velocidad del video sencillamente se programa es la velocidad en que las dos imágenes se combinan entre sí.

Toda esta parte del desarrollo del proyecto se asemeja bastante a un simple reproductor de video con control de velocidad, por lo que no se propone nada nuevo en esta parte.

La interacción

Una vez que se tiene la animación preparada, se procede a implementar la interacción. Como es un objetivo de este proyecto que la aplicación funcione sobre una mesa interactiva, y aprovechando la infraestructura de la Universidad, se decidió que la plataforma sobre la que se implementaría la interacción fuera reacTIVision.

"reacTIVision es un sistema de visión de computador de código abierto, multiplataforma, para el rastreo rápido y robusto de marcadores tipo fiducial puestos sobre objetos físicos, y también para el rastreo tactil multi-touch. Fue diseñado como un toolkit para el rápido desarrollo de interfaces de usuario tangibles basadas en mesas (TUI por sus siglas en inglés) y superficies interactivas multi-touch." [Kaltenbrunner 2007]

Processing también cuenta con una librería para reacTIVision, por lo que integrar lo que ya se había hecho con la animación no tuvo inconvenientes. El uso de reacTIVision hace que la interacción con la aplicación sea bastante innovadora, ya que se puede crear una interfaz de usuario interesante, por fuera de los conceptos tradicionales, minimalista, entendible y fácil de usar.

Solo se usa un *fiducial*, o marcador, para controlar la aplicación. Dependiendo de la posición sobre la mesa y de la rotación que tenga el *fiducial* se controlan los diferentes estados del programa.

Imagen 3: Aplicación detenida

Cuando el marcador se encuentra en la posición inicial la animación está detenida. Cuando se rota el marcador hacia la derecha o izquierda, la animación se mueve hacia adelante o hacia atrás en el tiempo.

Imagen 4: Rotaciones para avanzar y retroceder en el tiempo

Rotar el marcador hacia el centro de nuevo detiene la aplicación. Si se arrastra hacia arriba se activan las opciones de compartir. Para este proyecto solo se implementó la opción de enviar un correo.

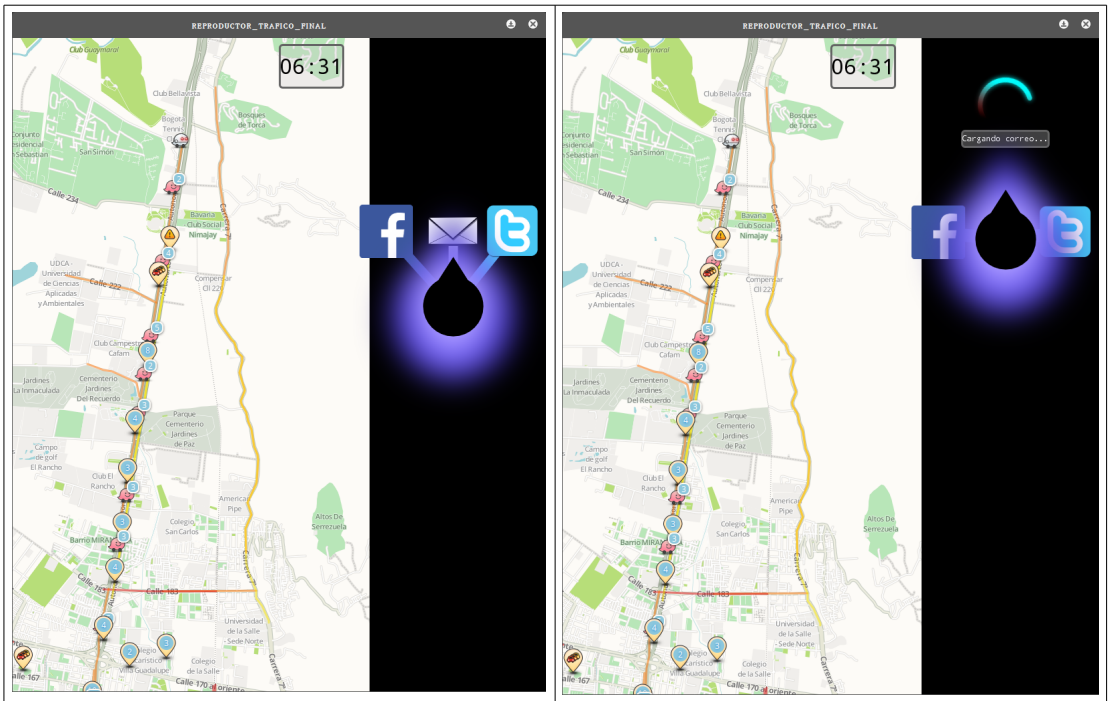


Imagen 5: Opciones de compartir

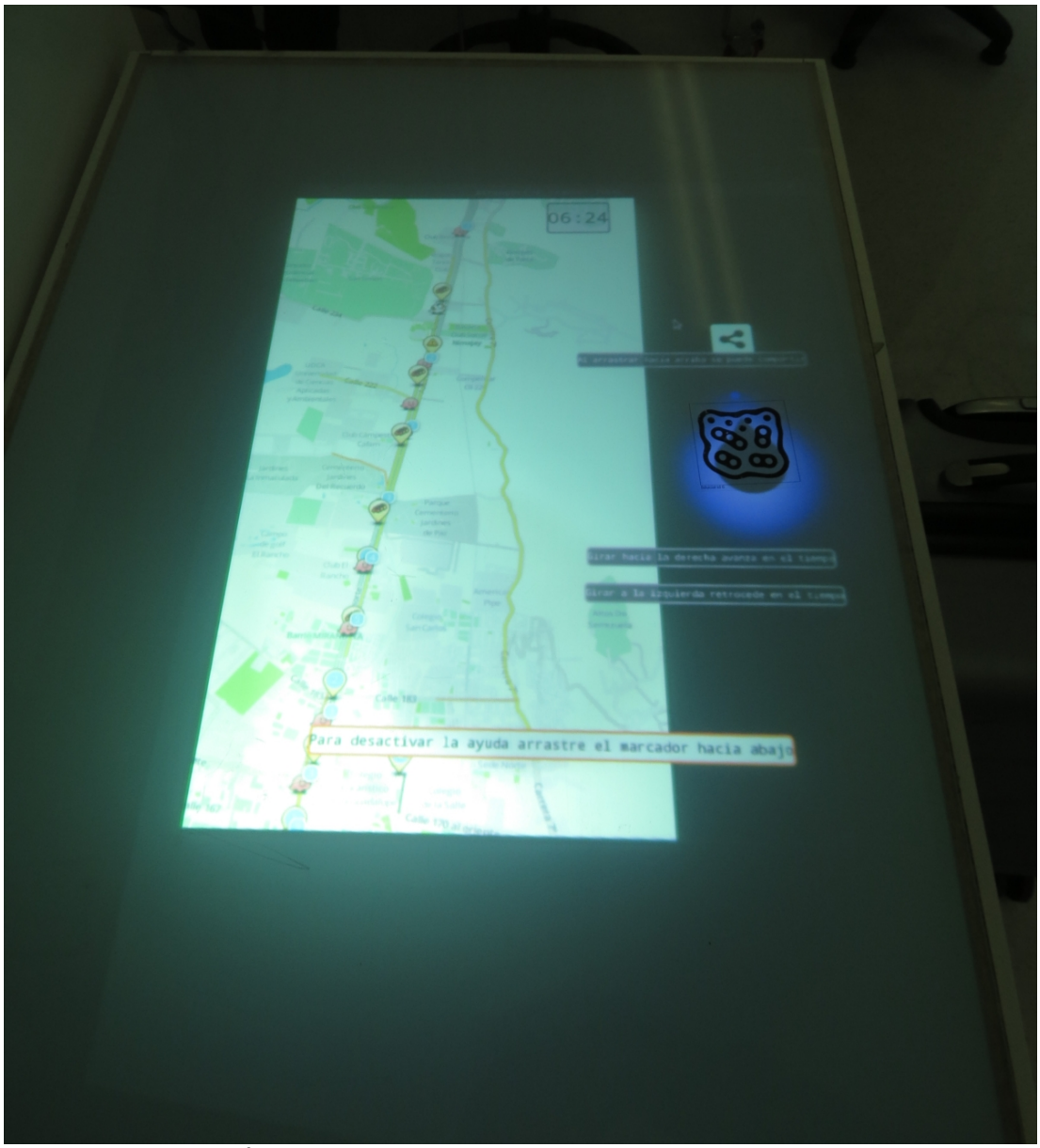
En cualquier caso que el marcador no se encuentre en la posición adecuada, o que no esté sobre la mesa, un mensaje de error se activa avisando a el usuario de esto.

Desde el punto de vista técnico, usar reacTIVision no difiere mucho de la implementación necesaria para usar un Mouse, por lo que su implementación en este proyecto no fue mayor inconveniente.

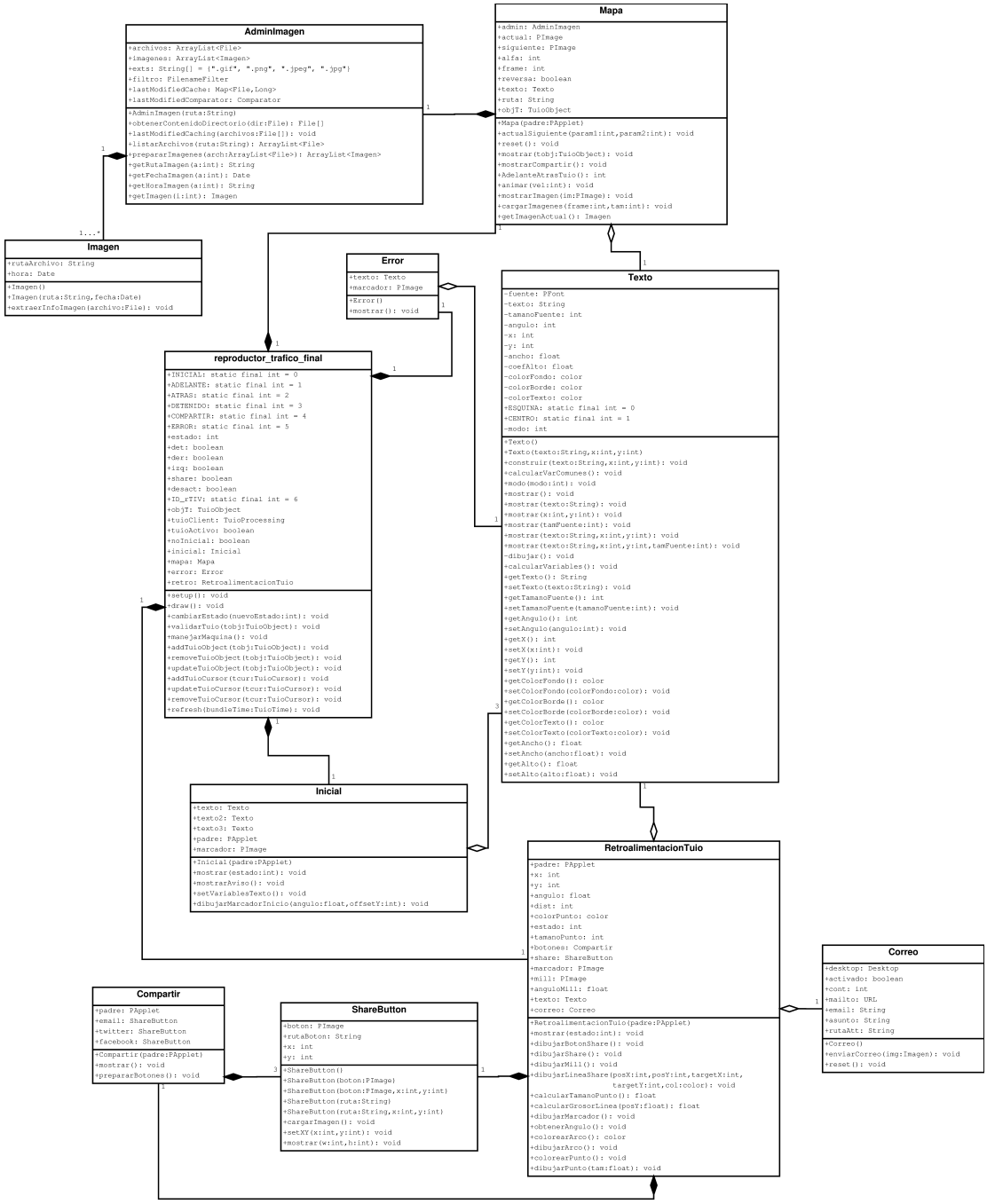
Anexos

Anexo 1: Configuración de la Mesa Interactiva en el Laboratorio LUX

Anexo 2: Configuración de los equipos dentro de la Mesa Interactiva



Anexo 3: Ejecución en vivo del software resultado de este proyecto sobre la Mesa Interactiva. En esta imagen se puede ver el *fiducial* de reacTIVision.



Anexo 4: Diagrama de clases de la aplicación resultado

Código Fuente

reproductor_trafico_final.pde

```
//Nuevo reproductor de trafico
//Esta es la clase principal del programa,
//la aplicación inicia por aquí y también es
//la que se encarga de crear la máquina de
//estados finitos.

//Es obligatorio tener los métodos de eventos
//de reacTIVision en esta clase.

//imports
import TUIO.*;
import java.util.Vector;

//variables globales estáticas
static final int INICIAL=0;
static final int ADELANTE=1;
static final int ATRAS=2;
static final int DETENIDO=3;
static final int COMPARTIR=4;
static final int ERROR=5;
//variables globales
int estado;

boolean det=false;
boolean der=false;
boolean izq=false;
boolean share=false;
boolean desact=false;

//con que fiducial va a funcionar el programa
static final int ID_rTIV=6;

TuioObject objT=null;
TuioProcessing tuioClient;
boolean tuioActivo=false;

boolean noInicial=false;

//Clases

Inicial inicial;
Mapa mapa;
Error error;
```

```
RetroalimentacionTuio retro;

void setup() {
  size(800, 980); //x original 1070
  background(0);
  estado=INICIAL;
  inicial= new Inicial(this);
  mapa= new Mapa(this);
  error=new Error();

  retro=new RetroalimentacionTuio(this);

  tuioClient = new TuioProcessing(this);
}

void draw() {
  background(0);

  manejarMaquina();

  switch(estado) {
  case INICIAL:
    //hacer el tutorial
    inicial.mostrar(estado);
    break;

  case ADELANTE:
    mapa.mostrar(objT);
    retro.mostrar(estado);
    if (!desact) {
      inicial.mostrar(estado);
      der=true;
    }
    break;

  case ATRAS:
    mapa.mostrar(objT);
    retro.mostrar(estado);
    if (!desact) {
      inicial.mostrar(estado);
      izq=true;
    }
    break;

  case DETENIDO:
    mapa.mostrar(objT);
    retro.mostrar(estado);
    if (!desact) {
      inicial.mostrar(estado);
```

```
      det=true;
    }
    break;

  case COMPARTIR:
    mapa.mostrarCompartir();
    retro.mostrar(estado);
    if (!desact) {
      inicial.mostrar(estado);
      share=true;
    }
    break;

  case ERROR:
    error.mostrar();
    break;

  default:
    break;
  }
}

//Método de cambio de estado
void cambiarEstado(int nuevoEstado) {
  estado=nuevoEstado;
}

//Clases de reacTIVision

//restringir el reacTIVision a el fiducial ID_rTIV y a la
//mitad derecha de la pantalla
void validarTuio(TuioObject tobj) {
  float minX=0.66;
  float minY=0.25;
  float maxX=0.99;
  float maxY=0.68;

  if (tobj.getSymbolID()==ID_rTIV) {
    objT=tobj;
    if
(tobj.getX()>minX&&tobj.getX()<maxX&&tobj.getY()>minY&&tobj.getY()<maxY) {
      tuioActivo=true;
    } else {
      tuioActivo=false;
    }
  } else {
    tuioActivo=false;
  }
}
```

```
//condiciones
void manejarMaquina() {

  float minAdelante=TAU/10;
  float minAtras=(3*TAU)/4;
  float maxAdelante=TAU/4;
  float maxAtras=(9*TAU)/10;
  int alturaCompartir=(height/2)-25;

  if ((!tuioActivo||objT==null)&&noInicial) {
    cambiarEstado(ERROR);
  } else if (!noInicial) {
    cambiarEstado(INICIAL);
  } else if (objT.getAngle()>minAdelante && objT.getAngle()<maxAdelante) {
    cambiarEstado(ADELANTE);
  } else if (objT.getAngle()<maxAtras && objT.getAngle()>minAtras) {
    cambiarEstado(ATRAS);
  } else if (objT.getScreenY(height)<alturaCompartir) {
    cambiarEstado(COMPARTIR);
  } else {
    cambiarEstado(DETENIDO);
  }

  if (tuioActivo) {
    if (det && der && izq && share && estado!=ERROR) {
      if (objT.getScreenY(height)>height/2+100) {
        desact=true;
      }
    }
  }
}

//Métodos de llamado TUIO, es obligatorio tenerlos aqui...
// called when an object is added to the scene
void addTuioObject(TuioObject tobj) {
  validarTuio(tobj);
}

// called when an object is removed from the scene
void removeTuioObject(TuioObject tobj) {
  tuioActivo=false;
}

// called when an object is moved
void updateTuioObject (TuioObject tobj) {
  validarTuio(tobj);
}
```

```
// called when a cursor is added to the scene
void addTuioCursor(TuioCursor tcur) {
}

// called when a cursor is moved
void updateTuioCursor (TuioCursor tcur) {
}

// called when a cursor is removed from the scene
void removeTuioCursor(TuioCursor tcur) {
}

// called after each message bundle
// representing the end of an image frame
void refresh(TuioTime bundleTime) {
}
```

AdminImagen.pde

```
//Esta clase controla como se muestran las imágenes.
//También ordena por último modificado de forma ascendente

import java.io.File;
import java.util.Date;
import java.util.ArrayList;

import java.io.FilenameFilter;

import java.util.Arrays;
import java.util.Comparator;
import java.util.Map;
import java.util.WeakHashMap;

class AdminImagen {

  //atributos

  ArrayList<File> archivos; //lista de rutas de archivo
  ArrayList<Imagen> imagenes; //lista de Imágenes (clase Imagen)

  //Filtrar por extensión de imagen: extensiones
  String[] exts= {
    ".gif", ".png", ".jpeg", ".jpg"
  };

  //Filtro para leer unicamente archivos de imagen
  FilenameFilter filtro = new FilenameFilter() {
    boolean accept(File dir, String nombre) {
      nombre = nombre.toLowerCase();
```

```
      for (int i = exts.length; i-- != 0; )  if (nombre.endsWith(exts[i]))
return true;
      return false;
    }
  };

  //Ordenar por ultimo modificado

  Map<File, Long> lastModifiedCache = new WeakHashMap(1000); //Hashmap
usado para ordenamiento

  //comparador usado para el ordenamiento
  Comparator lastModifiedComparator = new Comparator<File>() {
    int compare(File f1, File f2) {
      return Long.signum(lastModifiedCache.get(f1) -
lastModifiedCache.get(f2));
    }
  };

  //Constructor
  AdminImagen(String ruta) {
    archivos=listarArchivos(ruta);
    imagenes=prepararImagenes(archivos);
  }

  //este método se usa para filtrar los elementos del directorio
  File[] obtenerContenidoDirectorio(File dir) {
    return dir.listFiles(filtro);
  }

  //este método se usa para el ordenamiento por ultimo modificado
  void lastModifiedCaching(File[] archivos) {
    //System.gc();
    for (File f : archivos) lastModifiedCache.put(f, f.lastModified());
  }

  //Llenar la lista de archivos (ArrayList) con los archivos de imagen
  //que se van a mostrar. Se llena y ordena automáticamente.
  ArrayList<File>listarArchivos(String ruta) {
    File file=new File(ruta);
    ArrayList<File> lista = new ArrayList<File>();
    if (file.isDirectory()) {
      File[] arch = obtenerContenidoDirectorio(file);
      lastModifiedCaching(arch);
      Arrays.sort(arch, lastModifiedComparator);
      for (int i=0; i<arch.length; i++) {
        lista.add(arch[i]);
      }
      return lista;
```

```
    } else return null;
  }
  //Extraer la fecha de última modificación de cada imagen y
  //llenar la lista de imagen (ArrayList)
  ArrayList<Imagen> prepararImagenes(ArrayList<File> arch) {
    ArrayList<Imagen> im=new ArrayList<Imagen>();
    for (File f : arch) {
      String ruta=f.getAbsolutePath();
      Date fecha=new Date(f.lastModified());
      Imagen imagen=new Imagen(ruta, fecha);
      im.add(imagen);
    }
    return im;
  }
  //getters
  String getRutaImagen(int a) {
    return imagenes.get(a).rutaArchivo;
  }

  Date getFechaImagen(int a) {
    return imagenes.get(a).hora;
  }

  String getHoraImagen(int a) {
    Date fecha = getFechaImagen(a);
    String hora = nf(fecha.getHours(), 2)+":"+nf(fecha.getMinutes(), 2);
    return hora;
  }

  Imagen getImagen(int i) {
    return imagenes.get(i);
  }
}
```

Compartir.pde

```
//Clase compartir.
//Esta clase solo se llama cuando la animación está detenida

class Compartir {

  PApplet padre;

  ShareButton email;
  ShareButton twitter;
  ShareButton facebook;

  Compartir(PApplet padre) {
    this.padre=padre;
```

```
    email=new ShareButton("email.png");
    twitter= new ShareButton("Twitter-Button.png");
    facebook= new ShareButton("facebook-logo.png");
  }

  void mostrar() {
    imageMode(CENTER);
    noTint();
    fill(0);
    prepararBotones();
    email.mostrar(80, 45);
    twitter.mostrar(80, 80);
    facebook.mostrar(80, 80);
  }

  void prepararBotones() {
    email.cargarImagen();
    email.setXY(width-140, (height/2)-200);

    twitter.cargarImagen();
    twitter.setXY((width*3/4)+150, (height/2)-200);

    facebook.cargarImagen();
    facebook.setXY((width*3/4)-40, (height/2)-200);
  }

}
```

Correo.pde

```
//Clase Correo
//Esta clase permite abrir el cliente de correo predeterminado para la
//función de compartir

import java.awt.Desktop;
import java.net.URI;

class Correo {
  Desktop desktop;
  boolean activado;
  int cont;

  URI mailto;
  String email;
  String asunto;
  String rutaAtt;

  Correo() {
    activado=false;
```

```
    cont=0;
    email="fulanodetal@ejemplo.com";
    asunto="Imagen%20de%20Trafico";
  }

  void enviarCorreo(Imagen img) throws Exception {
    String ruta= img.rutaArchivo;
    //String fecha =
    if (Desktop.isDesktopSupported()
      && (desktop = Desktop.getDesktop()).isSupported(Desktop.Action.MAIL)
      && !activado) {
      mailto= new URI("mailto:"+email+"?
subject="+asunto+"&attachment="+ruta);
      desktop.mail(mailto);
      activado=true;
    } else if (!activado) {
      throw new RuntimeException("El cliente de correo no puede ser
abierto...");
    }
  }

  void reset() {
    activado=false;
    cont=0;
  }
}
```

Error.pde

```
//Esta clase muestra un mensaje de error cuando no esta en otro estado
//de la máquina

class Error {
  Texto texto;

  PImage marcador;

  Error() {
    texto=new Texto("Por favor coloque el marcador en este lugar",
width/2, height/3);
    texto.setColorTexto(color(255, 0, 0));
    marcador=loadImage("marcador error.png");
  }

  void mostrar() {
    texto.modo(texto.CENTRO);
    texto.mostrar(30);
    imageMode(CENTER);
    image(marcador, width-140, (height/2)-14);
```

```
  }
}
```

Imagen.pde

```
//Este es un simple objeto contenedor.
//Guarda una ruta a un archivo de imagen
//y su fecha de creación

import java.io.File;
import java.util.Date;

class Imagen {
  String rutaArchivo;
  Date hora;

  Imagen(String ruta, Date fecha) {
    rutaArchivo=ruta;
    hora=fecha;
  }

  Imagen() {
    rutaArchivo="";
    hora=null;
  }

  //crear un objeto Imagen a partir de un File
  void extraerInfoImagen(File archivo) {
    rutaArchivo=archivo.getAbsolutePath();
    hora=new Date(archivo.lastModified());
  }
}
```

Inicial.pde

```
//Estado Inicial.
//Esta clase es un tutorial de manejo del resto del programa

//Mostrar textos de ayuda la primera vez que se corre
class Inicial {
  Texto texto;
  Texto texto2;
  Texto texto3;
  PApplet padre;

  PImage marcador;

  Inicial(PApplet padre) {
    texto=new Texto("Bienvenido.", width/2, height/2);
```

```
    texto.setTamanoFuente(20);
    texto.setColorTexto(255);

    texto2=new Texto("Por favor coloque el marcador en este lugar para
iniciar", width/2, height/2);
    texto2.setTamanoFuente(20);
    texto2.setColorTexto(255);
    texto2.modo(texto2.CENTRO);

    texto3=new Texto();
    texto3.setTamanoFuente(20);
    texto3.setColorTexto(255);

    marcador=loadImage("marcador error.png");
  }

  void mostrar(int estado) {

    if (det && der && izq && share && estado!=ERROR && tuioActivo) {
      mostrarAviso();
    }

    switch(estado) {
    case INICIAL:
      texto.modo(texto.CENTRO);
      texto.mostrar(width/2, (height/3)-50);
      texto2.modo(texto2.CENTRO);
      texto2.mostrar(width/2, height/3);

      dibujarMarcadorInicio(0, 0);

      if (tuioActivo) {
        cambiarEstado(DETENIDO);
        noInicial=true;
      }
      break;

    case DETENIDO:
      setVariablesTexto();
      texto.mostrar("Girar hacia la derecha avanza en el tiempo",
width*3/4, height/2+150, 12);

      texto2.mostrar("Girar a la izquierda retrocede en el tiempo",
width*3/4, height/2+200, 12);

      texto3.mostrar("Al arrastrar hacia arriba se puede compartir",
width*3/4, height/3, 12);
      break;
```

```
    case ADELANTE:
      setVariablesTexto();
      texto.mostrar("Girar hacia el centro detiene la animación",
width*3/4, height/3, 14);

      texto2.mostrar("La cantidad de giro controla la velocidad",
width*3/4, height/2+150, 14);
      break;

    case ATRAS:
      setVariablesTexto();
      texto.mostrar("Girar hacia el centro detiene la animación",
width*3/4, height/3, 14);

      texto2.mostrar("Al llegar a la última hora registrada, vuelve a
empezar", (width*3/4)-50, height/2+150, 14);
      break;

    case COMPARTIR:
      setVariablesTexto();
      texto.mostrar("Con el marcador se selecciona la opcion de compartir",
(width*3/4)-50, height/5, 14);

      texto.mostrar("(por ahora solo la opción de correo está habilitada)",
width*3/4, height/2+100, 12);
      break;
    }
  }

  void mostrarAviso() {
    //mostrar el aviso de fin de ayuda...

    texto.modo(texto.CENTRO);
    texto.setColorFondo(color(240));
    texto.setColorBorde(color(255, 108, 10));
    texto.setColorTexto(color(0));
    texto.mostrar("Para desactivar la ayuda arrastre el marcador hacia
abajo", width/2, height-100, 16);
  }

  void setVariablesTexto() {
    color fondo=color(50, 200);
    color borde=color(150);
    color cTexto=color(255);

    texto.modo(texto.CENTRO);
    texto.setColorFondo(fondo);
    texto.setColorBorde(borde);
    texto.setColorTexto(cTexto);
```

```
    texto2.modo(texto.CENTRO);
    texto2.setColorFondo(fondo);
    texto2.setColorBorde(borde);
    texto2.setColorTexto(cTexto);

    texto3.modo(texto.CENTRO);
    texto3.setColorFondo(fondo);
    texto3.setColorBorde(borde);
    texto3.setColorTexto(cTexto);
  }


  void dibujarMarcadorInicio(float angulo, int offsetY) {
    imageMode(CENTER);
    noTint();
    pushMatrix();
    translate(width-140, (height/2)-offsetY);
    rotate(angulo);
    image(marcador, 0, -14);
    popMatrix();
  }
}
```

Mapa.pde

```
//Esta clase se encarga de mostrar el mapa, animarlo y mostrar
//la hora y fecha de la imagen que está mostrando en ese
//momento

import TUIO.*;

class Mapa {
  //Atributos
  AdminImagen admin;

  PImage actual;
  PImage siguiente;

  int alfa; //alfa inicia en 1
  int frame;  //frame inicia en 0

  boolean reversa;  //inicia como false

  Texto texto;  //muestra la hora de la imagen

  String ruta; //ruta donde se encuentran las imágenes

  TuioObject objT; //objeto TUIO usado para los cálculos subsiguientes
```

```
  Mapa(PApplet padre) {
    ruta=sketchPath + "/Waze4/";
    alfa=1;
    frame=0;
    reversa=false;
    admin=new AdminImagen(ruta);
    actualSiguiente(frame, frame+1);
    texto=new Texto(admin.getHoraImagen(0), width/2, 10); //original x
(width-200)/2
    objT=null;
  }

  //Métodos

  void actualSiguiente(int param1, int param2)
  {
    actual=loadImage(admin.getRutaImagen(param1));
    siguiente=loadImage(admin.getRutaImagen(param2));
  }

  void reset() {
    alfa=1;
    frame=0;
    reversa=false;
  }

  //este método solo se llama dentro del draw()
  void mostrar(TuioObject tobj) {
    imageMode(CORNER);
    objT=tobj;
    animar(adelanteAtrasTuio());
    texto.modo(texto.ESQUINA);
    texto.mostrar(admin.getHoraImagen(frame));
  }

  void mostrarCompartir() {
    imageMode(CORNER);
    mostrarImagen(actual);
    texto.mostrar(admin.getHoraImagen(frame));
  }

  int adelanteAtrasTuio() {
    if (objT!=null) {
      //TAU = 2PI. Para más información visite http://tauday.com
      if (objT.getAngle()>TAU/10 && objT.getAngle()<TAU/4) {
        int retorno=(int)map(objT.getAngle(), TAU/10, TAU/4, 0, 128);
        reversa=false;
        return retorno;
```

```
      } else if (objT.getAngle()<(9*TAU)/10 && objT.getAngle()>(3*TAU)/4)
{
        int retorno=(int)map(objT.getAngle(), (3*TAU)/4, (9*TAU)/10, 128,
0);
        reversa=true;
        return retorno;
      } else {
        return 0;
      }
    } else return 0;
  }

  void animar(int vel) {
    if (alfa<256) {
      tint(255, 255);
      mostrarImagen(actual);
      blendMode(BLEND);
      tint(255, alfa);
      mostrarImagen(siguiente);
      alfa=alfa+vel;
    } else {
      int tam=admin.imagenes.size();
      if (!reversa) {
        frame=(frame+1)%tam;
      } else {
        if (frame==0) {
          frame=tam-1;
        } else {
          frame=frame-1;
        }
      }
      alfa=255;
      mostrarImagen(actual);
      alfa=0;
      cargarImagenes(frame, tam);
      mostrarImagen(actual);
    }
  }

  void mostrarImagen(PImage im) {
    int x=0;
    int y=0;
    image(im, x, y);
  }

  void cargarImagenes(int frame, int tam) {
    if (!reversa) {
      if (frame==tam-1) {
        actualSiguiente(frame, 0);
```

```
      } else if (frame==tam) {
        actualSiguiente(0, 1);
      } else {
        actualSiguiente(frame, frame+1);
      }
    } else {
      if (frame==0) {
        actualSiguiente(0, tam-1);
      } else {
        actualSiguiente(frame, frame-1);
      }
    }
  }

  Imagen getImagenActual() {
    return admin.getImagen(frame);
  }
}
```

RetroalimentacionTuio.pde

```
//Clase Retroalimentación visual de TUIO
//Esta clase se encarga de hacer toda la retroalimentación visual

class RetroalimentacionTuio {
  PApplet padre;

  int x;
  int y;
  float angulo;

  int dist; //ancho del arco

  color colorPunto;
  int estado;

  int tamanoPunto;

  Compartir botones;
  ShareButton share;

  PImage marcador;
  PImage mill;

  float anguloMill;
  Texto texto;

  Correo correo;
```

```
RetroalimentacionTuio(PApplet padre) {
  this.padre=padre;
  dist=200;
  x=width-140;
  y=(height/2);
  tamanoPunto=12;
  botones= new Compartir(padre);

  share=new ShareButton("share.png", x, y-dist);
  marcador=loadImage("marcador.png");
  mill=loadImage("mill.png");

  texto= new Texto();
  anguloMill=0;

  correo = new Correo();
}

//este método se llama en el estado inicial

void mostrar(int estado) {
  this.estado=estado;

  if (this.estado!=COMPARTIR) {
    correo.reset();
  }
  switch(estado) {
  case ADELANTE:
    dibujarArco();
    dibujarPunto(tamanoPunto);
    dibujarMarcador();
    break;
  case ATRAS:
    dibujarArco();
    dibujarPunto(tamanoPunto);
    dibujarMarcador();
    break;
  case DETENIDO:
    dibujarArco();
    //dibujarShare();
    dibujarPunto(tamanoPunto);
    dibujarMarcador();
    dibujarBotonShare();
    break;
  case COMPARTIR:
    dibujarShare();
    //dibujarPunto(calcularTamanoPunto());
    dibujarMarcador();
    break;
```

```
    }
  }

  void dibujarBotonShare() {
    noTint();
    imageMode(CENTER);
    share.mostrar(64, 64);
  }

  void dibujarShare() {
    int posX=objT.getScreenX(width);
    int posY=objT.getScreenY(height);

    //Dibujar unas líneas
    dibujarLineaShare(posX, posY, botones.email.x, botones.email.y,
color(255));
    dibujarLineaShare(posX, posY, botones.twitter.x, botones.twitter.y,
color(73, 200, 245));
    dibujarLineaShare(posX, posY, botones.facebook.x, botones.facebook.y,
color(59, 87, 157));

    //mostrar los iconos
    botones.mostrar();

    //retroalimentacion sobre los iconos
    texto.modo(texto.CENTRO);
    texto.setColorTexto(color(255));
    if (posY<botones.email.y+40) {
      if (posX>botones.email.x-26 && posX<botones.email.x+26) {
        correo.cont++;
        if (correo.cont>150) {

          try {
            correo.enviarCorreo(mapa.getImagenActual());
            texto.mostrar("Correo Abierto", x, 150, 12);
          }
          catch(Exception e) {
            //texto.mostrar(e.getMessage, width/2, height/2, 16);
            println(e);
          }
        } else {
          dibujarMill();
          texto.mostrar("Cargando correo...", x, 150, 12);
        }
        //llamar el método para hacer un correo...
      } else if (posX>botones.twitter.x-26 && posX<botones.twitter.x+26) {
        texto.mostrar("Twitter", x, 150, 12);
        correo.reset();
      } else if (posX>botones.facebook.x-26 && posX<botones.facebook.x+26)
```

```
{
        texto.mostrar("Facebook", x, 150, 12);
        correo.reset();
      } else {
        texto.mostrar("...", x, 150, 12);
        correo.reset();
      }
    }
  }

  void dibujarMill() {
    anguloMill+=0.20;
    pushMatrix();
    translate(x, 100);
    rotate(anguloMill);
    image(mill, 0, 0);
    popMatrix();
  }

  void dibujarLineaShare(int posX, int posY, int targetX, int targetY,
color col) {
    stroke(col);
    float grosor=calcularGrosorLinea(objT.getY());
    strokeWeight(grosor);
    line(posX, posY, targetX, targetY);
  }

  float calcularTamanoPunto() {
    float posY=objT.getY();
    float tamano=12;
    if (posY<0.65 && posY>0.3) {
      tamano=map(posY, 0.65, 0.3, 12, 40);
    } else tamano=12;
    return tamano;
  }

  float calcularGrosorLinea(float posY) {
    float grosor=0;
    if (posY<0.65 && posY>0.3) {
      grosor=map(posY, 0.65, 0.3, 1, 20);
    } else grosor=0;
    return grosor;
  }

  void dibujarMarcador() {
    int hor=objT.getScreenX(width);
    int ver=objT.getScreenY(height);

    imageMode(CENTER);
```

```
  noTint();
  pushMatrix();
  translate(hor, ver);
  rotate(angulo+HALF_PI);
  image(marcador, 0, -14);
  popMatrix();
}

void obtenerAngulo() {
  angulo=objT.getAngle()-HALF_PI;
}

color colorearArco() {
  float anguloT=objT.getAngle();
  float anguloMin=3*TAU/4;
  float anguloMax=TAU/4;
  //float anguloMed=TAU;

  color retorno;

  color min=color(0);
  color max=color(200);

  float amt1=map(anguloT, anguloMin, TAU, 1.5, 0);
  float amt2=map(anguloT, 0, anguloMax, 0, 1.5);
  if (anguloT<=TAU && anguloT>=anguloMin) {
    retorno=lerpColor(min, max, amt1);
  } else if (anguloT>=0 && anguloT<=anguloMax) {
    retorno=lerpColor(min, max, amt2);
  } else {
    retorno=color(0);
  }
  return retorno;
}

void dibujarArco() {
  noFill();
  color colorArco=colorearArco();
  stroke(colorArco);
  strokeWeight(20);
  arc(x, y, dist, dist, TAU/2, TAU);
}

void colorearPunto() {

  float anguloT=objT.getAngle();
  float anguloMin=3*TAU/4;
  float anguloMax=TAU/4;
```

```
    color min=color(255, 139, 23);
    color med=color(0, 0, 250);
    color max=color(50, 250, 22);

    float amt1=map(anguloT, anguloMin, TAU, 0, 1);
    float amt2=map(anguloT, 0, anguloMax, 0, 1);
    if (anguloT<=TAU && anguloT>=anguloMin) {
      colorPunto=lerpColor(min, med, amt1);
    } else if (anguloT>=0 && anguloT<=anguloMax) {
      colorPunto=lerpColor(med, max, amt2);
    } else {
      colorPunto=color(0);
    }
  }

  void dibujarPunto(float tam) {
    int posY=objT.getScreenY(height);
    int posX=objT.getScreenX(width);
    noStroke();
    obtenerAngulo();
    colorearPunto();
    fill(colorPunto);
    if (estado==ADELANTE || estado == ATRAS) {
      ellipse(x+cos(angulo)*(dist/2), y+sin(angulo)*(dist/2), tam, tam);
    } else if (estado==DETENIDO) {
      int trueY;
      if ((posY-(dist/2))>y+sin(angulo)*(dist/2)) {
        trueY=(int)(y+sin(angulo)*(dist/2));
      } else {
        trueY=posY-(dist/2);
      }
      ellipse(x+cos(angulo)*(dist/2), trueY, tam, tam);
    } else if (estado==COMPARTIR) {
      ellipse(posX, posY-(dist/2), tam, tam);
    }
  }
}
```

ShareButton.pde

```
//Esta es una simple clase contenedora
//Para tener los diferentes botoncitos de Share

class ShareButton {

  PImage boton;
  String rutaBoton;
  int x;
  int y;
```

```
  void cargarImagen() {
    boton=loadImage(rutaBoton);
  }

  void setXY(int x, int y) {
    this.x=x;
    this.y=y;
  }

  void cargarVariables(String ruta, int x, int y) {
    rutaBoton=ruta;
    setXY(x, y);
  }

  ShareButton() {
    boton=null;
    cargarVariables("", 0, 0);
  }

  ShareButton(PImage boton) {
    this.boton=boton;
    cargarVariables("", 0, 0);
  }

  ShareButton(PImage boton, int x, int y) {
    this.boton=boton;
    cargarVariables("", x, y);
  }

  ShareButton(String ruta) {
    cargarVariables(ruta, 0, 0);
  }

  ShareButton(String ruta, int x, int y) {
    cargarVariables(ruta, x, y);
  }

  void mostrar(int w, int h) {
    cargarImagen();
    image(boton, x, y, w, h);
  }
}
```

Texto.pde

```
//Módulo de texto que muestra un recuadro con bordes redondeados y
//texto dentro del recuadro. Por defecto se crea un rectángulo gris
//ligeramente transparente, con bordes de un gris más claro, y texto
```

```
//negro con letra tamaño 32 puntos y tipo de letra "Droid Sans Mono".
//
//El módulo es lo suficientemente "inteligente" para ajustar el tamaño
//del rectángulo con respecto al tamaño y largo del texto (si es solo
//una línea de texto).

public class Texto {

  // Atributos

  private PFont fuente; //Archivo de fuente de Processing
  private String texto; //El texto que se va a mostrar
  private int tamanoFuente;
  private int angulo; // angulo del borde del rectángulo
  private int x; //Posición x donde va a aparecer el texto (esquina
izquierda)
  private int y; //Posición y donde va a aparecer el texto (esquina
superior)
  private float ancho; // ancho total del texto y del rectángulo
  private float alto; // alto total del texto y del rectángulo
  // al tamaño de la letra
  private float coefAlto; // multiplicador para ajustar el alto al tamaño
de
  // la letra
  private color colorFondo; // color del fondo del rectángulo
  private color colorBorde; // color del borde del rectángulo
  private color colorTexto; // color del texto

    //Estas variables se usan para el modo de dibujo del módulo
  public static final int ESQUINA=0;
  public static final int CENTRO=1;
  private int modo;

  //Este método es llamado por los constructores
  private void construir(String texto, int x, int y) {
    fuente = loadFont("DroidSansMono-32.vlw");
    this.texto = texto;
    this.y = y;
    this.x = x;
    angulo = 5;
    tamanoFuente = 32;
    calcularVarComunes();
    colorFondo = color(200, 80);
    colorBorde = color(100);
    colorTexto = color(0);
    modo=ESQUINA;
  }

  //método que se llama cada vez que se calculan las variables
```

```
//comunes del módulo, cada vez que se muestra algo en la ventana
private void calcularVarComunes() {
  coefAlto = 2;
  ancho=textWidth(texto);
  alto = tamanoFuente * coefAlto;
}

//constructores
public Texto() {
  construir("", 0, 0);
}

public Texto(String texto, int x, int y) {
  construir(texto, x, y);
}

//getters & setters
public String getTexto() {
  return texto;
}

public void setTexto(String texto) {
  this.texto = texto;
}

public int getTamanoFuente() {
  return tamanoFuente;
}

public void setTamanoFuente(int tamanoFuente) {
  this.tamanoFuente = tamanoFuente;
}

public int getAngulo() {
  return angulo;
}

public void setAngulo(int angulo) {
  this.angulo = angulo;
}

public int getX() {
  return x;
}

public void setX(int x) {
  this.x = x;
}
```

```
public int getY() {
  return y;
}

public void setY(int y) {
  this.y = y;
}

public color getColorFondo() {
  return colorFondo;
}

public void setColorFondo(color colorFondo) {
  this.colorFondo = colorFondo;
}

public color getColorBorde() {
  return colorBorde;
}

public void setColorBorde(color colorBorde) {
  this.colorBorde = colorBorde;
}

public color getColorTexto() {
  return colorTexto;
}

public void setColorTexto(color colorTexto) {
  this.colorTexto = colorTexto;
}

public float getAncho() {
  return ancho;
}

public void setAncho(float ancho) {
  this.ancho=ancho;
}

public float getAlto() {
  return alto;
}

public void setAlto(float alto) {
  this.alto=alto;
}

//este método cambia el origen desde donde se pinta el
```

```
//cuadro de texto
public void modo(int modo) {
  switch(modo) {
  case ESQUINA:
    this.modo=ESQUINA;
    break;

  case CENTRO:
    this.modo=CENTRO;
    break;

  default:
    this.modo=ESQUINA;
    break;
  }
}

//métodos para pintar el módulo, dependiendo del caso
public void mostrar() {
  mostrar(texto, x, y, tamanoFuente);
}

public void mostrar(String texto) {
  mostrar(texto, x, y, tamanoFuente);
}

public void mostrar(int x, int y) {
  mostrar(texto, x, y, tamanoFuente);
}

public void mostrar(int tamFuente) {
  mostrar(texto, x, y, tamFuente);
}

public void mostrar(String texto, int x, int y) {
  mostrar(texto, x, y, tamanoFuente);
}


public void mostrar(String texto, int x, int y, int tamFuente) {
  int trueX;
  int trueY;

  tamanoFuente = tamFuente;
  this.x=x;
  this.y=y;
  this.texto=texto;
  calcularVariables();
```

```
    switch(modo) {
    case ESQUINA:
      rectMode(CORNER);
      dibujar();
      break;

    case CENTRO:
      rectMode(CENTER);
      dibujar();
      break;
    }
  }

  private void dibujar() {
    fill(colorFondo);
    stroke(colorBorde);
    strokeWeight(3);
    rect(x, y, ancho, alto, angulo);

    fill(colorTexto);
    text(texto, x, y, ancho, alto);
  }

  // este método se llama la primera vez que se muestra algo
  public void calcularVariables() {
    textFont(fuente, tamanoFuente);
    textAlign(CENTER, CENTER);
    calcularVarComunes();
  }
}
```

Bibliografía

Aigner, W., Miksch, S., Müller, W., Schumann, H., & Tominski, C. (2007). Visualizing time-oriented data—A systematic view. Computers & Graphics, 31(3), 401–409. doi:10.1016/j.cag.2007.01.030

Zhao, J., Chevalier, F., & Balakrishnan, R. (2011). KronoMiner: Using Multi-Foci Navigation for the Visual Exploration of Time-Series Data.

Krstajić, M., Bertini, E., & Keim, D. a. (2011). CloudLines: compact display of event episodes in multiple time-series. IEEE transactions on visualization and computer graphics, 17(12), 2432–9. doi:10.1109/TVCG.2011.179

Fry, B. J., Design, C., Arts, M., Fry, B., Maeda, J., Cooper, M., & Lippman, A. B. (2004). computational information design, (May 1997).

Cámara de Comercio de Bogotá (2010). Observatorio de Movilidad, número 6.
http://camara.ccb.org.co/contenido/contenido.aspx?conID=3222&catID=721

Hartl, Michael (2010). No, really, Pi is wrong: The Tau Manifesto http://tauday.com/tau-manifesto

Martin Kaltenbrunner and Ross Bencina. 2007. reacTIVision: a computer-vision framework for table-based tangible interaction. In Proceedings of the 1st international conference on Tangible and embedded interaction (TEI '07). ACM, New York, NY, USA, 69-74. DOI=10.1145/1226969.1226983