Proyecto Vision

Proyecto final para las materias "Vision Artificial" y "Complementos de Informatica" El proyecto consiste en una clase capaz de obtener imagenes de distintos medios, detectar rostros, obtener puntos de interés y analizar su simetria. Ademas, puede registrar tanto el video adquirido como los puntos de interés detectados. Estos datos provistos pueden ser utilizados para detectar momentos donde la asimetria sea maxima o donde se detecte claramente cierta expresion facial (sonrisa, ceño fruncido, levantar cejas, etc). La idea de la implementación está basada en este paper.

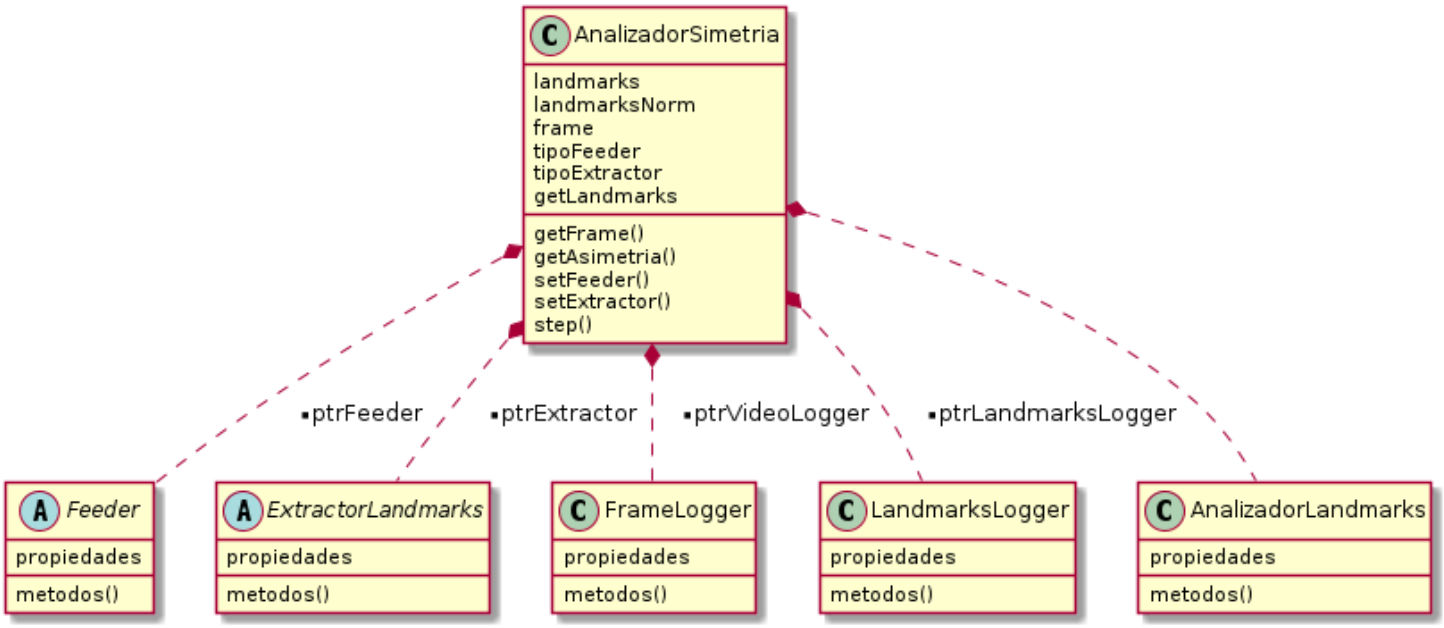
Uso del programa de ejemplo

El programa lee la configuración del archivo config.yaml

- Presionando la tecla `o` se utiliza el extractor de landmarks de openCV
- Presionando la tecla `d` se utiliza el extractor de landmarks de dlib
- Presionando la tecla `w` se intenta obtener imágenes de una webcam
- Presionando la tecla `k` se intenta obtener imágnees de una kinect
- Presionando la tecla `v` se intenta abrir un archivo de video (por defecto, 'video.avi').
- Presionando la tecla `q` se termina la ejecución.

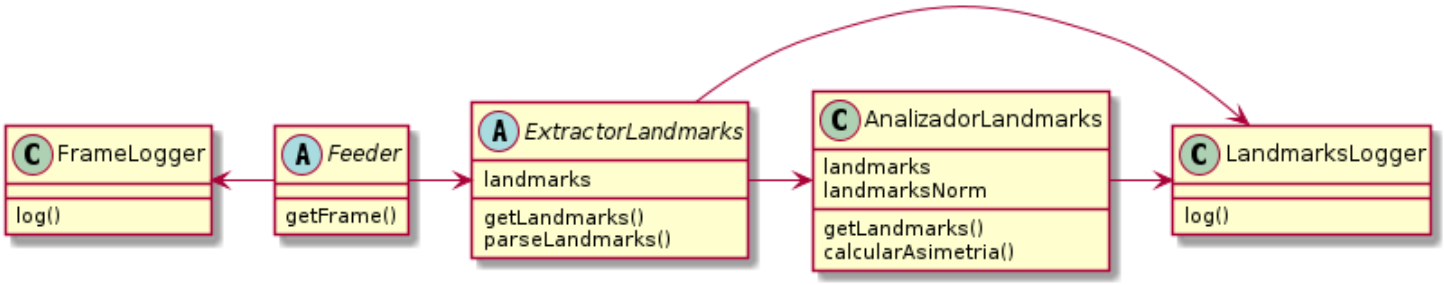
Desarrollo

El software consiste en una clase base que se compone de otras subclases. Luego, los objetos concretos de estas clases interactúan entre sí.



.

Esquema de composición de la clase base.



.

Esquema de las relaciones internas entre clases.

El sistema instancia los objetos necesarios a partir de un archivo de configuración provisto, con las clases concretas necesarias segun el caso. Si no se pasan argumentos, intenta cargar una configuración por defecto, abriendo una webcam y utilindo un archivo de entrenamiento por defecto.

Además, se incluye el código para probar diversas funcionalidades de la clase.

Clases

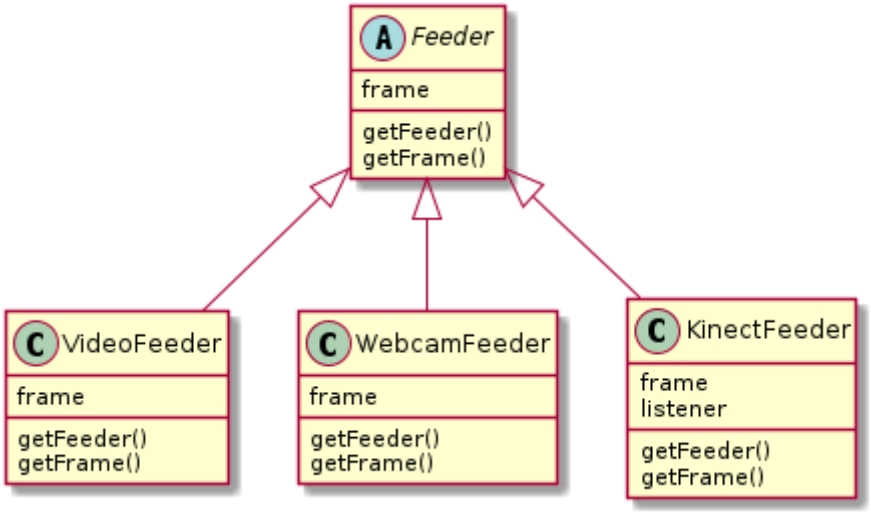
A continuacion, se detallan las distintas clases base que compondran al sistema.

Clase Feeder

Clase encargada de obtener los fotogramas a analizar. De ésta se derivan tres clases para proveer frames de distintos medios. Las tres clases que heredan de Feeder, serán:

- VideoFeeder, pensada para trabajar con videos.
- WebcamFeeder, pensada para adquirir fotogramas de una webcam.
- KinectFeeder, pensada para adquirir fotogramas de una kinect utilizando libfreenect2.

Estas clases tendrán un método general para devolver un fotograma del tipo `cv::Mat` que serán utilizados por los objetos de las clases (FrameLogger y ExtractorLandmarks).



Ejemplo en UML de la clase "feeder"

Clase FrameLogger

Esta clase está encargada de registrar los frames provistos por el Feeder. Por el momento, va registrando los fotogramas en un video por defecto. Debe tener un metodo de actualizacion que consista en guardar el archivo de imagen en algun lugar en particular, con un nombre que lo identifique unicamente, y que de alguna manera quede linkeado a una "base de datos".

Clase ExtractorLandmarks

Esta clase está encargada de obtener los puntos de interés de un rostro a partir de las imágenes provistas por el Feeder. Esta clase abstracta en principio tiene dos implementaciones:

- clase ExtractorLandmarksOpenCV, utilizando openCV
- clase ExtractorLandmarksDlib, utilizando dlib

Estas dos clases utilizarían distintos algoritmos para la deteccion de puntos de interes, basados en distintos papers y con distintos entrenamientos.

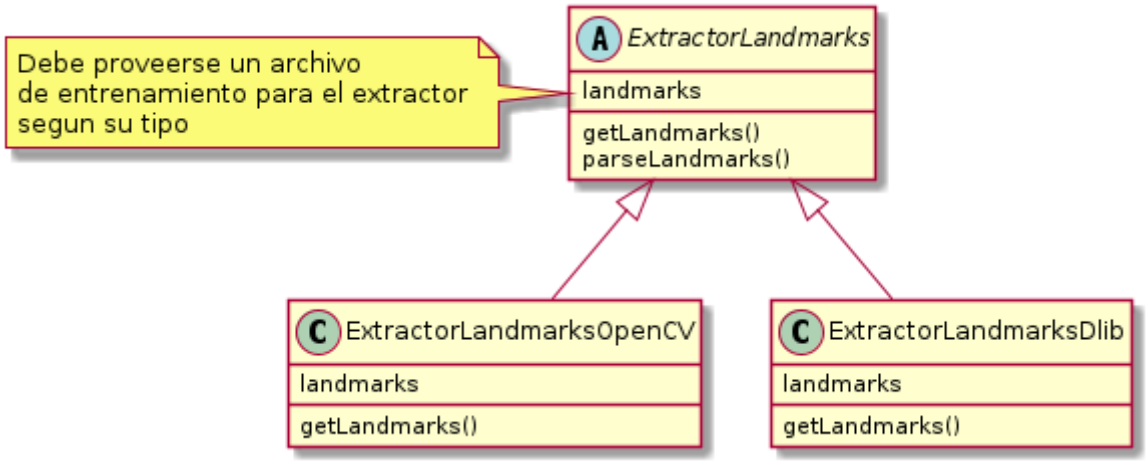


Diagrama UML de la clase ExtractorLandmarks

Clase ExtractorLandmarksOpenCV

Esta clase está basada en las librerias de openCV. Necesita los archivos `haarcascade_frontalface_alt2.xml` y `lbfmodel.yaml` .

Clase ExtractorLandmarksDlib

Esta clase está basada en las librerias de dlib. Necesita el archivo `shape_predictor_68_face_landmarks.dat` , obtenido del código de ejemplo de dlib.

Clase AnalizadorLandmarks

Esta clase normaliza los puntos de interés provistos por el extractor de landmarks, es decir, corrige la inclinacion de la cabeza, tomando como referencias los puntos a mitad de cada oreja. Con esto, calcula la simetria de la cara basándose en algunos puntos particulares(Elegidos medio aleatoriamente) Respecto a la simetria, la clase será la encargada de analizar los puntos de interés normalizados, haciendo algunos calculos geométricos y devolviendo distintas medidas sobre la simetria facial. En el paper de referencia, estas medidas se utilizan para luego alimentar un clasificador. Al no tener acceso a los datasets para poder "clasificar" distintos rostros, este ultimo paso se dificulta. Aun asi, debe ser posible obtener un puntaje analizando distintas medidas y comparando ambos lados del rostro. Además, el normalizador podria filtrar solo los landmarks necesarios para el calculo de simetria, reduciendo asi el tamaño de los datos guardados. Finalmente, la clase devuelve un vector de una estructura predefinida, habiendo una estructura por cada rostro detectado.

Clase LandmarksLogger

Esta clase está encargada de registrar cada vector de landmarks obtenido. Registra los datos en un archivo de tipo YAML, tomando un nombre de base y agregandole el tipo de feeder utilizado y un

timestamp.

Estructura Landmarks

Es la estructura que devuelve el extractor de landmarks para facilitar su uso. Está compuesta por las siguientes propiedades:

- vacio : Bandera para detectar si la estructura posee información
- rotacion : Indica la rotación de la cabeza, en caso de haberse normalizado.
- escala : Indica la escala de los puntos, en caso de haberse normalizado.
- menton : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan el mentón.
- ojoIzq : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan el ojo izquierdo.
- ojoDer : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan el ojo derecho.
- cejaIzq : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan la ceja izquierda.
- cejaDer : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan la ceja derecha.
- boca : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan la boca.
- nariz : `std::vector<cv::Point2f>` vector de puntos de openCV que demarcan la nariz.

Dependencias

- dlib (utilizada version 19.22)
- libfreenect2 (utilizada version 0.2.0)
- openCV (utilizada version 4.5.2)

Además, se utilizaron diversas librerias requeridas por estos paquetes (principalmente por libfreenect2) que están mencionadas en el comando de compilación.

TODO

- [ ] Sobrecarga de operadores `<<` y `>>` para la entrada y salida de archivos.
- [ ] Algoritmo de ordenamiento de los vectores de landmarks (utilizando como referencia la asimetria).
- [ ] Procesar datos guardados previamente.
- [ ] Generar una interfaz en Qt.
- [ ] Generalizar el Feeder para poder incorporar, por ejemplo, imágenes de profundidad , RGBD o IR desde kinect.

- [ ] Realizar una implementación mejorada del calculador de asimetría (es una funcion dummy, hace un par de calculos basicos).
- [ ] Utilizar threads para distintas tareas(principalmente las que requieren acceso al disco) y así optimizar el funcionamiento general.
- [ ] Desarrollar en mayor profundidad el programa de ejemplo mainProyectoVision.
- [ ] Manejo de señales, para que si se interrumpe salga limpiamente.

Desarrollo y compilación

Para el desarrollo de este software, se utilizó el editor de texto Visual Studio Code, ya que facilitaba y automatizaba muchas tareas con las correspondientes extensiones (Generador de UML, comentarios automáticos para Doxygen, snippets de código, autocompletar código, generación de distintas tareas de compilación). Para compilarlo, se utilizó el siguiente comando:

```
g++ -Wall -DUSE_AVX_INSTRUCTIONS=ON /home/agustin/Facultad/5to/ProyectoVision/*.cc
/home/agustin/Facultad/5to/ProyectoVision/include/src/*.cc
-o /home/agustin/Facultad/5to/ProyectoVision/Release/mainProyectoVision
-I/home/agustin/Facultad/5to/ProyectoVision/include -I/usr/include/
-I/usr/local/lib/ -I/usr/local/include -I/usr/local/include/opencv4
-L/usr/local/freenect2/lib -L/usr/lib -lopencv_core -lopencv_highgui
-lopencv_imgcodecs -lopencv_video -lopencv_videoio -lopencv_plot
-lopencv_objdetect -lopencv_imgproc -lopencv_face -ldlib -lturbojpeg
-ljpeg -lfreenect2 -llapack -lopenblas -O3
```

Generado con el archivo tasks.json en Visual Studio Code.