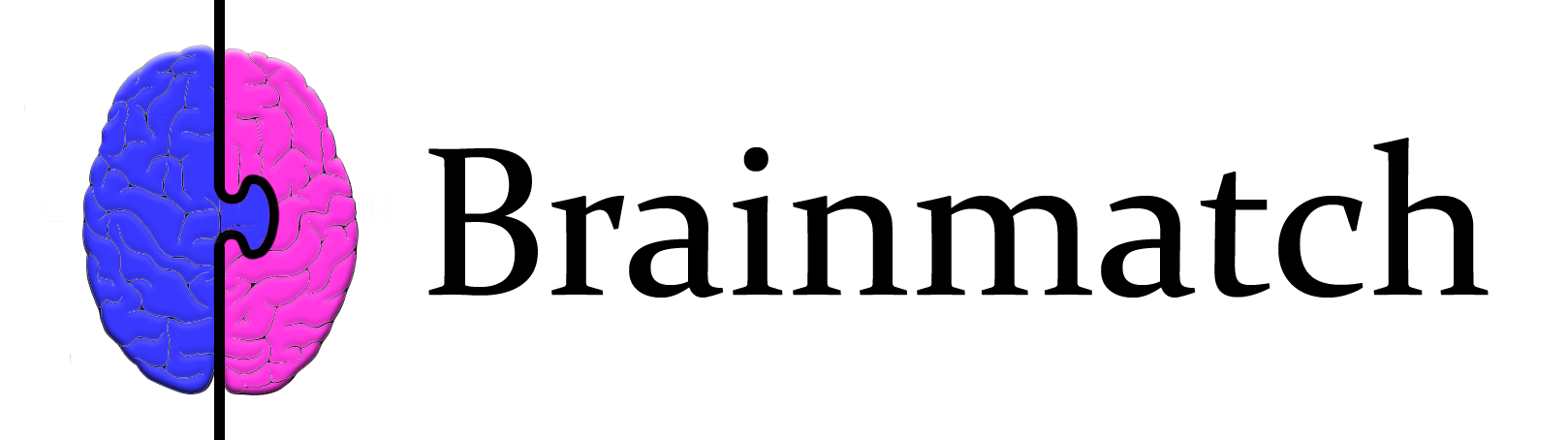
Benjamin Azoulay

Elodie Dehache

Emir Lekouara

Pierre Vazquez

# Rapport du Projet Transverse



# Sommaire :

# I- Introduction

# II- Théorie

## A) L’algorithme SURF

## B) L’implémentation

# III- Conclusion

# IV- Bibliographie

# I- Introduction

A chaque zone du cortex cérébrale correspond une émotion. Lorsque le cerveau réagit à son environnement, les zones excitées varient selon le ressenti, l’intérêt et le point de vue d’un individu. Or, en tant qu’humains, nous sommes attirés par des individus aux intérêts et points de vue similaires aux nôtres. Cela a permis aux chercheurs d’émettre l’hypothèse qu’il est possible de prédire l’entente entre deux personnes en comparant leurs réactions cérébrales. Cette hypothèse fut testée dans une étude publiée ce Janvier dans *Nature*. Cette étude prend pour individus 279 étudiants d’une promotion de première année à UCLA. Les chercheurs ont tout d’abord déterminé, parmi les étudiants, ceux qui sont amis, amis d’amis etc. jusqu’à obtenir le graphe social de la promotion. Des 279 étudiants, 42 ont été sélectionnés pour passer des IRMf. Lorsqu’une région du cerveau s’active, plus d’oxygène est nécessité. Il y a donc un influx sanguin vers la région du cerveau activée, ce qu’une IRMf peut détecter. Durant l’examen, les étudiants ont visionné des vidéos de divers genres : comédies, documentaires, clip musicaux… Si les étudiants prenaient plaisir à voir les vidéos ou s’ils s’ennuient, cela était visible sur l’IRMf. Une fois les images obtenues, elles ont été comparées et un nouveau graphe à été fait selon leur degré de similitude entre elles. Or, ce graphe correspondait au graphe social de la promotion établi précédemment : plus les réactions concordes, plus les individus sont proches.

Réciproquement, on peut donc prédire les relations entre deux personnes selon la similarité de leurs réactions cérébrales. Nous avons donc eu l’idée d’établir un programme de comparaison d’IRM renvoyant un pourcentage de compatibilité. Ce teste serait fiable dans la mesure où le cerveau ne ment pas.

# II- La théorie

## A) L’algorithme SURF

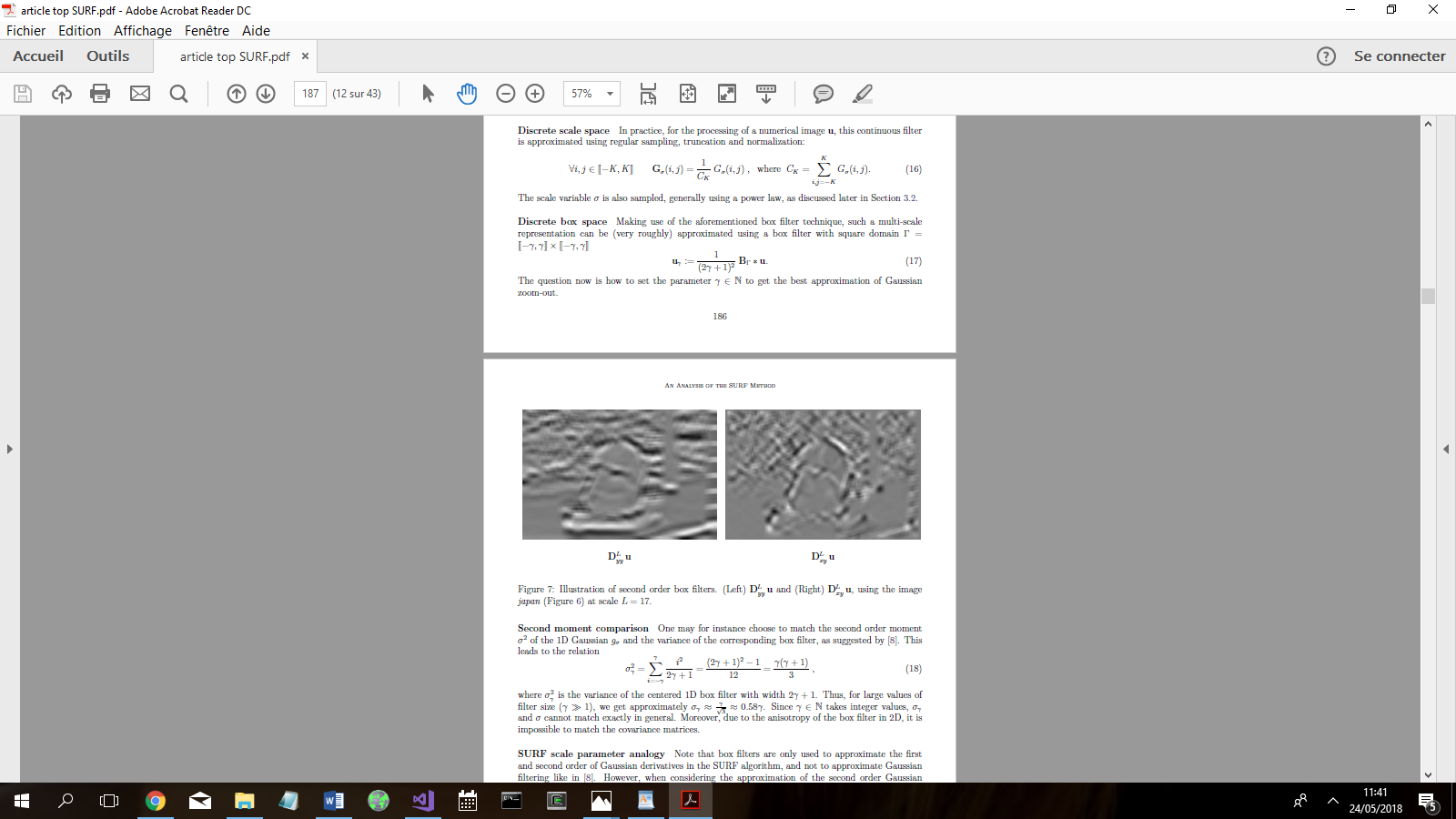
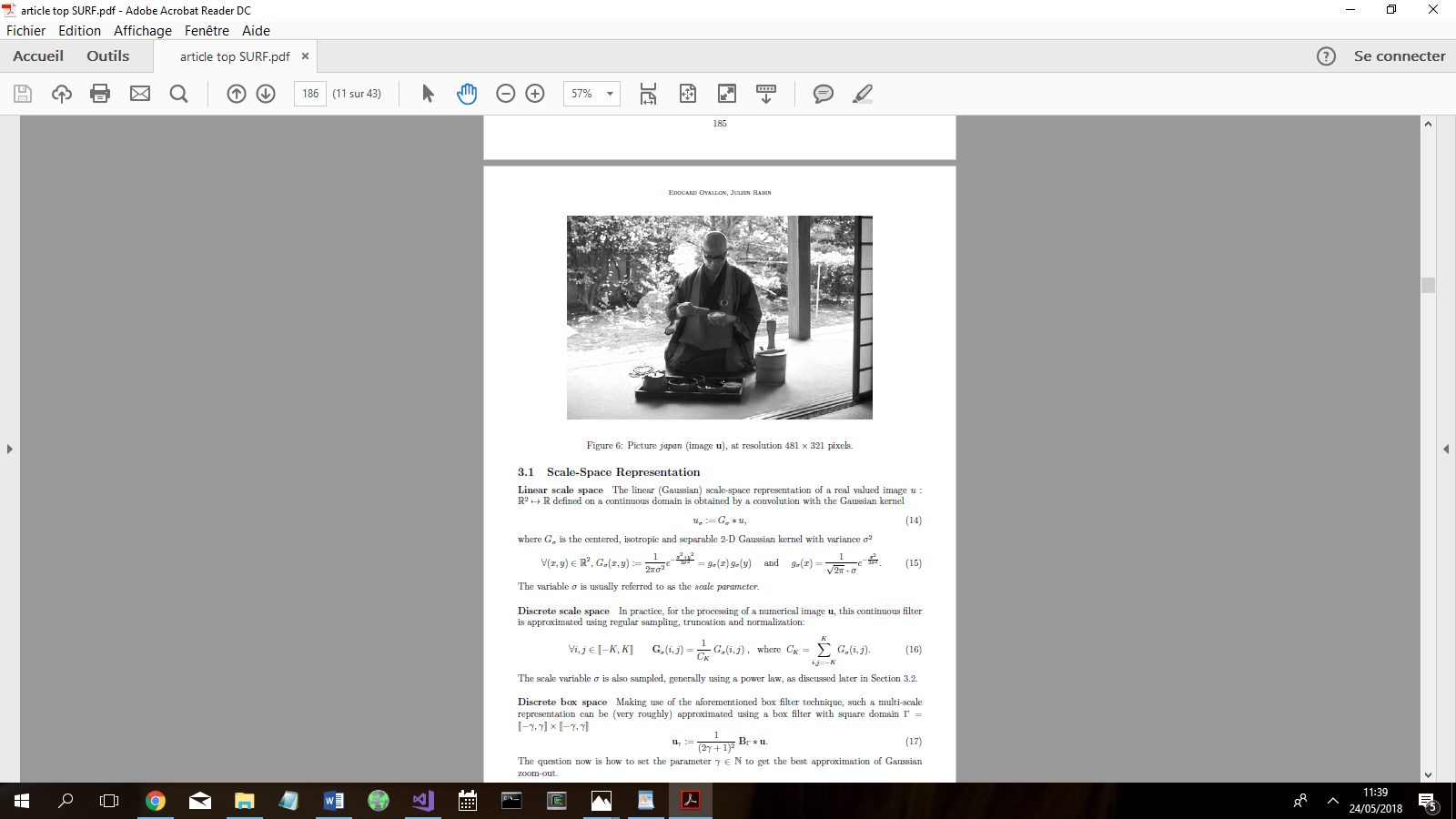
L’algorithme SURF se déroule en trois principales composantes :

* Détection de points-clés dans l'image objet et dans l'image scène. Les points clés associent dans un même objet les coordonnées du point, une taille, un angle et une octave.
* Calcul d'un descripteur pour chaque point-clé de l'image objet et de l'image scène. Un descripteur est une valeur calculée sur 64 ou 128 éléments. Nous calculons ici en 64 éléments.
* Détection des concordances entre les descripteurs des points-clé de l'image objet et de l'image scène. Chaque concordance caractérise par l'index du point objet, l'index du point scène correspondant et une grandeur "distance".

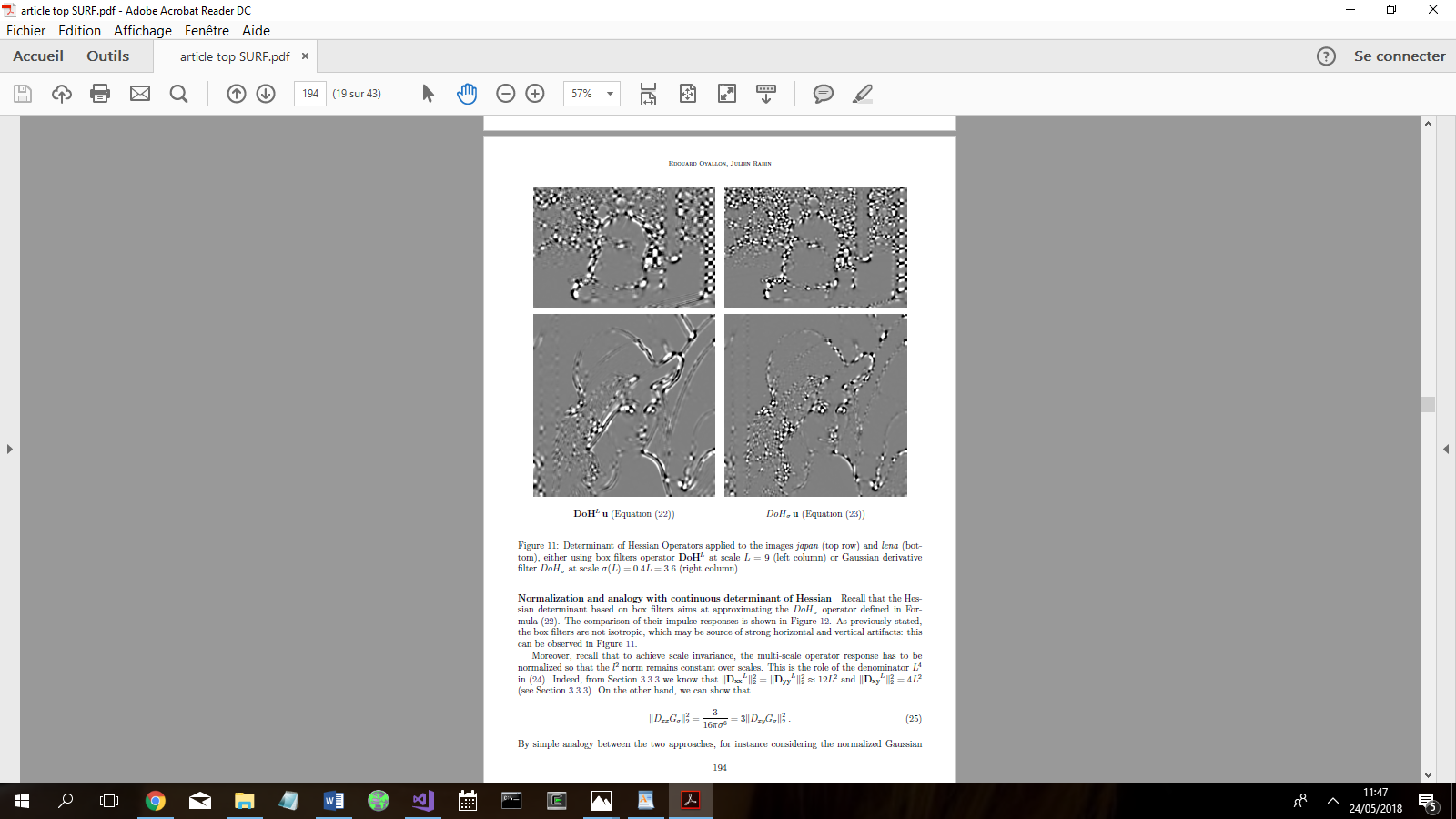
Il s’appuie sur des principes de filtrage d’une image par des transformations mathématiques (les *Box filters*) afin de définir les points clés (keypoints).

Pour notre implémentation, en vue de l’analyse de radiographies médicales, il était suffisant de se placer dans un range de [0 ;255] donc en noir et blanc pour pouvoir analyser les images.

Les opérations mathématiques de filtrages reposent sur la génération d’une image intégrale de second ordre, ce qui nous permettra ensuite de créer une matrice Hessienne (ou de calculer un Laplacien) dont on conservera le maximum afin de mettre en évidence les points de selles.



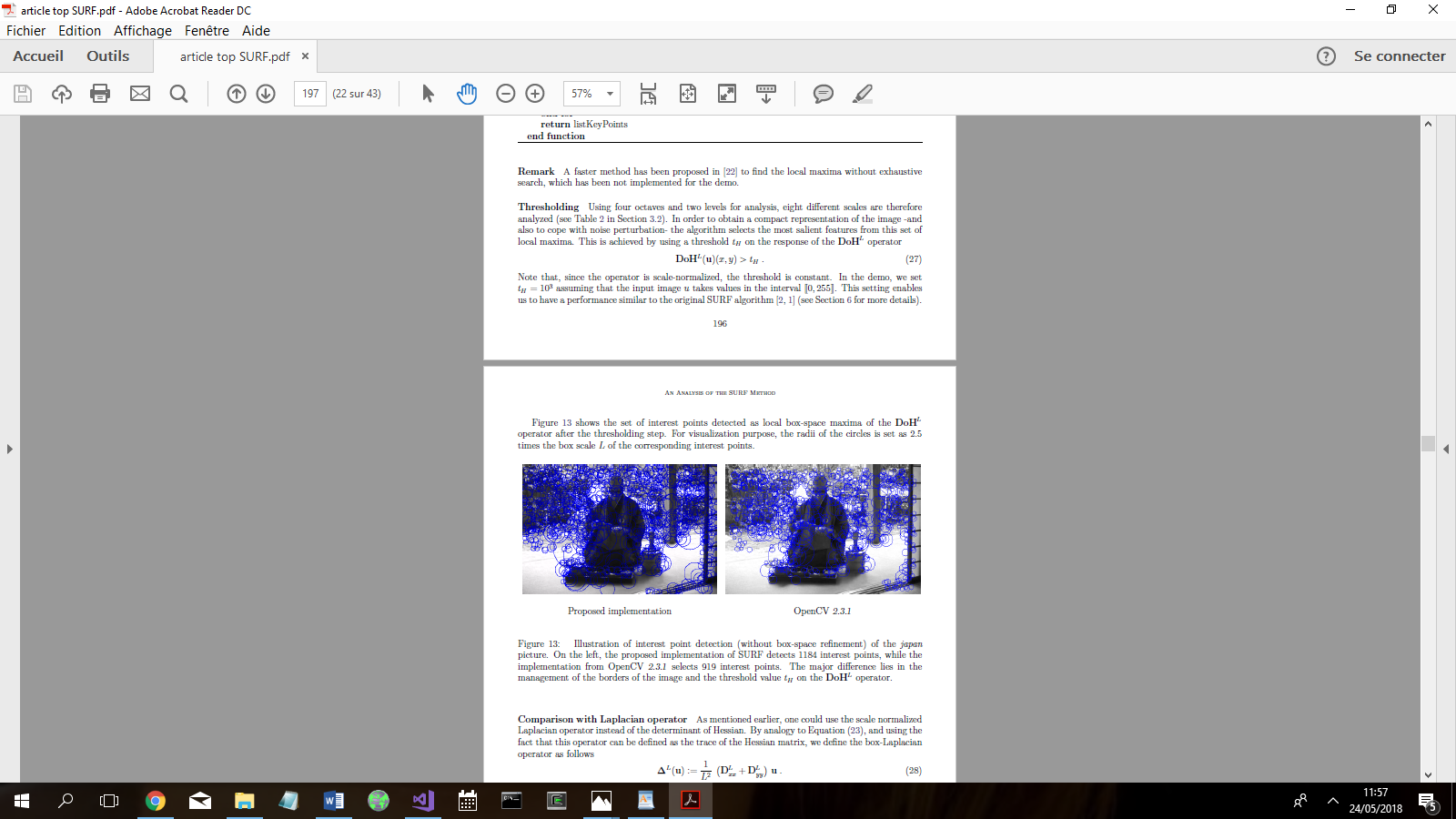
Il est important de noter que la théorie de l’algorithme SURF permet de garantir une invariance des points d’intérets : chaque opération est normalisée afin que l’échelle, l’orientation, la résolution du point n’influent pas sur le matching. Par exemple pour l’échelle L, on normalise le Dyy u obtenu en multipliant par un facteur 1/L^4. Apres normalisation, on obtient donc :



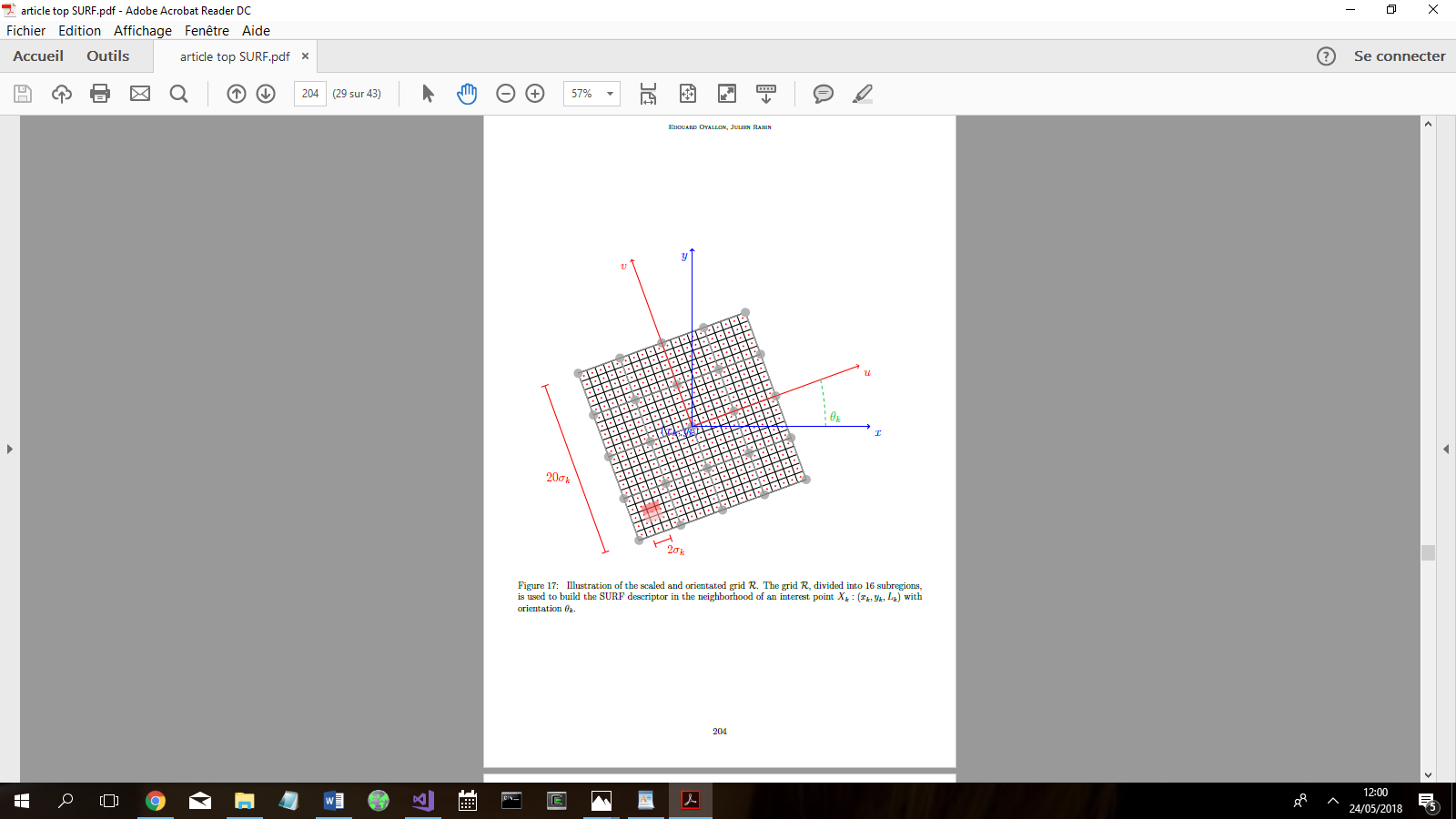
On choisit enfin le maximum de ces « images intégrales ». Ce n’est qu’après ces étapes de normalisation et de maximisation qu’on considère un point comme d’intérêt pour notre algorithme.

On obtient des points d’intérêts keypoints ayant en paramètres : l’échelle(L), l’orientation(gradient), la valeur du déterminant(DoH).

Pour visualiser ces points d’intérêts, on a représenté ces points par des cercles de radian proportionnels à 2.5xL :



Une fois les points d’intérêts détectés, l’algorithme génère un vecteur descriptor par rapport au point d’intérêt et son voisinage.



Sur cette figure, on peut voir comment on génère un descriptor : on génère un vecteur descriptor en fonction du point (xk,yk) (en bleu sur le schema), on aura divisé le voisinage du point en sous régions (dans surfdef.h on aura fixé celui-ci au nombre de 20 max). Il prend en compte les signe du laplacien et l’orientation : un vecteur 16x4 qui équivaut au gradient normalisé d'une grille R divisée en sous régions 4x4, on obtient le descriptor final en concaténant les 16 vecteurs obtenus. On obtient un descriptor de 16x4=64 éléments.

Ces tâches sont réalisées par l’exécutable extract\_surf .

Il faut ensuite matcher les points d’intérêts : On compare les descriptor avec le principe du *Nearest-Neighbor Distance Ratio* et de la distance euclidienne. Afin d’accélerer le processus de matching, on a ajouté que si deux descriptors ont un signe (du laplacien) différent, il ne seront surement pas un match. Cela permet de faire un premier tri dans les matchs.

On invoque la commande : match\_surf pour faire le matching.

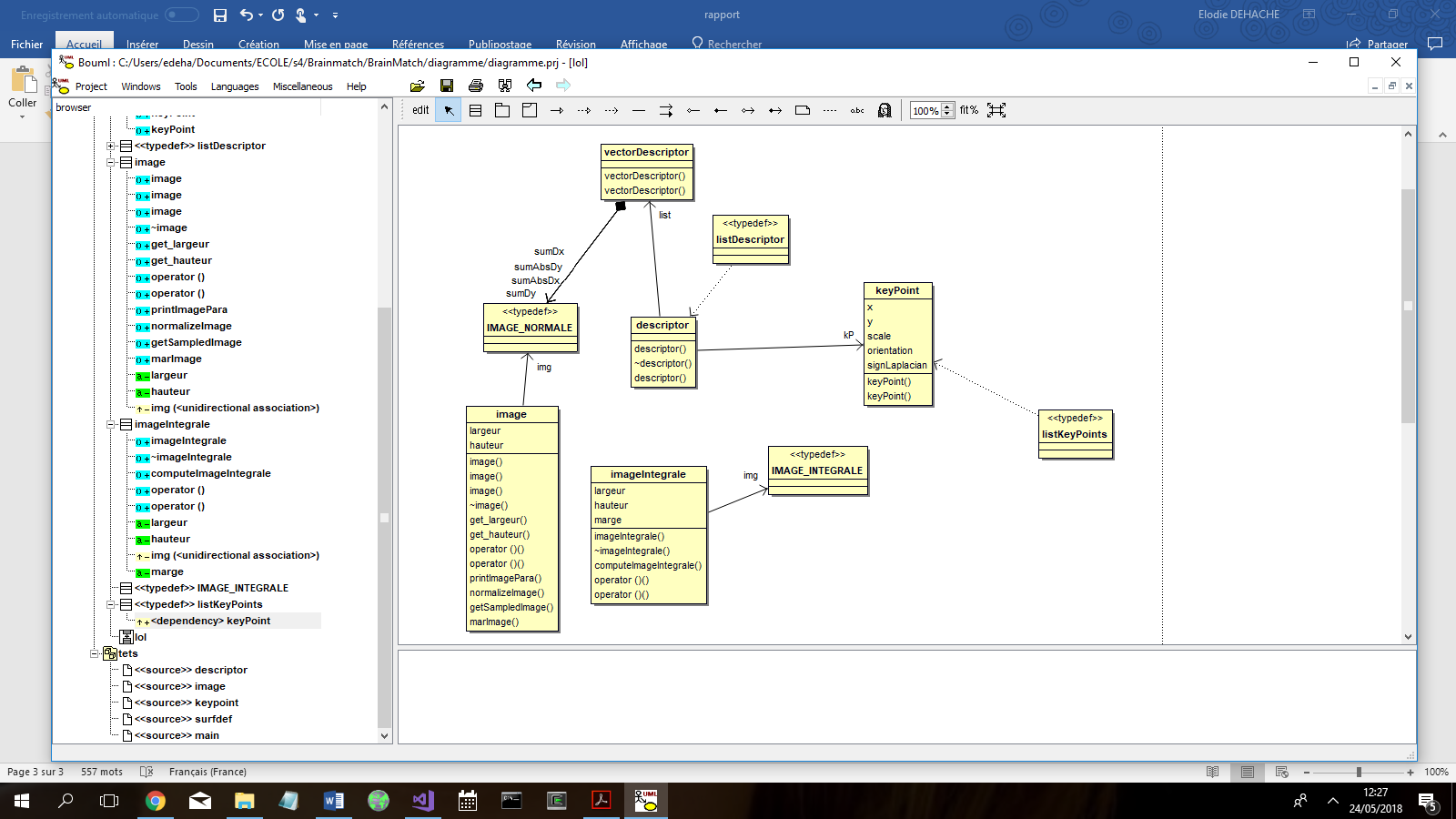
Il est important de noter que la compréhension des éléments manipulés par l’algorithme repose principalement sur des composantes mathématiques que nous ne détaillerons pas pour ne pas alourdir le rapport. La publication d’origine de SURF ainsi que de nombreux éléments théoriques sont cités en biographie si l’on souhaite approfondir les connaissances sur le fonctionnement.

## B) L’implémentation

La complexité de la théorie mathématique de l’algorithme SURF et le temps imparti ont empatis sur l’implémentation de notre projet, elle demandait une certaine expertise de la manipulation d’image qui a posé certains problèmes lors de la conception d’un interface utilisateur. Nous avons fait le choix de nous concentrer sur les performances techniques et le bon déroulement de l’algorithme. Au vu des nombreuses manipulations mathématiques requises, le C++ nous a semblé être un langage cohérent vers lequel nous orienter. Nous avons également beaucoup annoté le code au vu de l’aspect très dense mathématiquement de l’algorithme que l’on souhaitait implémenter.

Nous avons commencé par fixer les paramètres autour desquels s’organise l’algorithme dans surfdef.h ainsi que quelques éléments récurrents (arrondir,etc).

Nous avons ensuite conçu nos classes : la théorie offrait un bon guide à la conception, en effet elle se sépare en différentes entités sur lesquelles on réalise des opérations ou qui interagissent entre elles, nous avons donc créé les classes images, keypoint, descriptor.



* Classe image : Il s'agissait de créer une classe pour les premières manipulations de l'image.

Les attributs sont donc centrés sur la taille de l'image, qui sont des paramètres dans tous

les calculs qui seront réalisés par la suite (logique car la théorie repose que le calcul d'intégrale,de manipulation de filtre de gauss qui prend cette taille en paramètre, du Laplacien,etc).

On a également créé une classe spécifique pour les images intégrées.

* Classe keypoint : elle permet de regrouper les paramètres qui entrent dans la structure de nos points d’intérêts

On stock ces points dans une liste (Rappel, ils ne sont ajoutés qu’après normalisation, etc)

* Classe descriptor : Classe du vecteur descripteur pout le point clé détecté ce vecteur µ(i,j) sera ensuite concaténé aux autre pour donner le descriptor final. Il contient les sommes des gradients locaux des sous régions(sumDx,sumDy,sumAbsDx,sumAbsDy).

Nous avons ensuite été confrontés au manque de temps pour réaliser un interface utilisateur. Nous avons donc 3 fichiers .cpp pouvant être exécutables par des lignes de commande (nous avons utilisé (cygwin):

* Extract.cpp
* Matchsurf.cpp
* Affiche.cpp

Etant donné le nombre de fichiers du programme, nous avons été obligés de réaliser un makefile afin de n’avoir en commandes finales que ces trois entités : extract\_surf, match\_surf et display\_surf.

*# C source*

*CSRC = fichiers\_externes/io\_png.c*

*# C++ source*

*CPP\_match = matchsurf.cpp fichiers\_externes/orsa.cpp matchlib.cpp fichiers\_externes/libMatch/match.cpp*

*CPP\_affiche = image.cpp affiche.cpp fichiers\_externes/libMatch/match.cpp*

*CPP\_extract = extract.cpp descriptor.cpp image.cpp keypoint.cpp*

*# objets*

*COBJ = $(CSRC:.c=.o)*

*CPPOBJ\_match\_surf = $(CPP\_match:.cpp=.o)*

*CPPOBJ\_display\_surf = $(CPP\_affiche:.cpp=.o)*

*CPPOBJ\_extract\_surf = $(CPP\_extract:.cpp=.o)*

*CPPOBJ = $(COBJ) $(CPPOBJ\_extract\_surf) $(CPPOBJ\_match\_surf) $(CPPOBJ\_display\_surf)*

*# binaries*

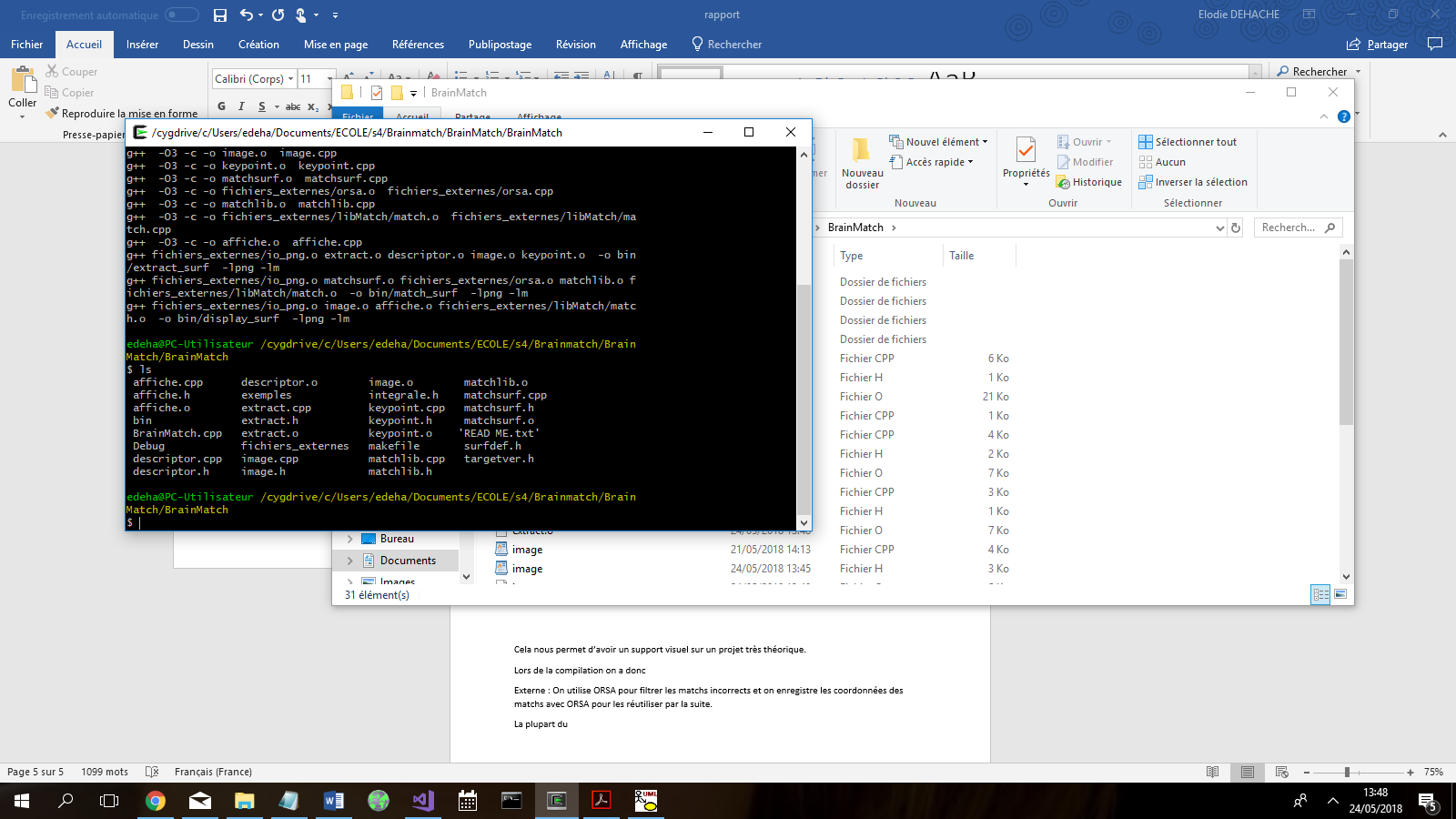
*EXEC = extract\_surf match\_surf display\_surf*

Extract prend en argument un fichier image .png et un fichier texte k1.txt l’algorithme y mettra les points d’intérêts.

Match prend en argument les deux images, deux fichiers textes des points cles, un fichier match.txt où sont répertoriés les matchs.

Display nous permet d’afficher les performances de l’algorithme, (à l’aide de fichiers externes que nous développerons ensuite) prends les deux fichiers de points clés, le fichier de match et il génère deux .png : un qui répertorie les matchs et un qui représente les points clés à l’aide d’ORSA (utilisé pour générer des images satellites, etc).

Cela nous permet d’avoir un support visuel sur un projet très théorique.



Lors de la compilation, les fichiers .o sont générés et les exécutables sont placés dans le /bin du projet. On retrouve dans le /bin les matchs, descriptors et images générées.

Externe : On aura utilisé IPOL pour le traitement d’image. On utilise ORSA pour filtrer les matchs incorrects et on enregistre les coordonnées des matchs avec ORSA pour les réutiliser par la suite. Orsa et io\_png nous permettent de pouvoir afficher les detectors et les matchs.

L’implémentation dans sa conceptualisation n’aura pas posé le problème majeur de notre projet. Celui-ci reposant sur des manipulations mathématiques, il aura demandé un gros travail de traduction et de rigueur mais cela aura guidé la conception. L’adaptation à la manipulation des images, de nouveaux concepts et leur mélange (orsa, etc) aura posé un des enjeux majeurs dans la compréhension du projet. Le développement en c++ fut certes laborieux, mais la difficulté majeure reposait principalement dans la compréhension de l’algorithme surf et ses concepts mathématiques.

# Conclusion

Au commencement de ce projet transverse, notre équipe s’orientait plus vers la création d’une application de rencontres entre personnes en fonction des similitudes observées lors d’une IRM. En effet un groupe de chercheurs a démontré en Janvier que plus on est proche d’une personne plus nos réactions avec elle lors d’une IRM sont similaires.

Après avoir contacté l’équipe de chercheurs ayant fait l’étude, afin de pouvoir accéder à leur base de données, nous avons pris conscience que le stockage d’une IRM était très complexe car il faut en moyenne 1 To d’espace disponible pour une seule IRM.

De plus avoir accès à des images médicales est très difficiles car ce sont des documents confidentiels.

C’est pourquoi même après avoir eu une réponse positive de l’équipe nous avons toutefois décidé de changer l’orientation du projet afin de créer, toujours avec le même principe, un comparateur d’images mais utilisable dans le milieu médical, ce qui nous permet d’occulter le problème du stockage des IRM.

La finalité serait de pouvoir proposer notre projet à des hôpitaux pour qu’ils puissent comparer une nouvelle IRM à leur base de données afin de faire des pré-diagnostics.

# IV- BIBLIOGRAPHIE



Etude sur laquelle est basée l’idée du projet :

<https://www.nature.com/articles/s41467-017-02722-7>

Documentation sur la comparaison d’images :

<https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html#KeyPoint::KeyPoint(Point2f%20_pt,%20float%20_size,%20float%20_angle,%20float%20_response,%20int%20_octave,%20int%20_class_id)>

<https://docs.opencv.org/2.4/doc/tutorials/features2d/feature_detection/feature_detection.html#feature-detection>

<https://docs.opencv.org/2.4/doc/tutorials/features2d/feature_homography/feature_homography.html#feature-homography>

<http://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/>

<https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html>

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2660387/figure/F3/>

<https://msdn.microsoft.com/fr-fr/library/c8whxhf1.aspx>

<http://orsa.sourceforge.net/doc/demo/an_introduction_to_ORSA.pdf>

<http://www.math-info.univ-paris5.fr/~moisan/epipolar/>

<http://www.ipol.im/>

<https://www.projet-plume.org/ressource/ipol>