

Projet Technologique

---

## VISION STÉRÉOSCOPIQUE

---

Geoffrey MEILHAN  
Mohamed ALAMI  
Kenji FONTAINE

28 avril 2017

## Table des matières

<b>1 Description du projet</b>	<b>3</b>
<b>2 Domaine : Vision stéréoscopique</b>	<b>3</b>
<b>3 Cahier des charges</b>	<b>3</b>
3.1 Besoins fonctionnels . . . . .	3
3.2 Besoins non fonctionnels . . . . .	4
<b>4 Architecture du code : partie QT et OpenCV</b>	<b>5</b>
<b>5 Architecture du code : partie Unity</b>	<b>6</b>
<b>6 Tests</b>	<b>8</b>
<b>7 Problèmes rencontrés</b>	<b>11</b>
<b>8 Conclusion</b>	<b>12</b>

# 1 Description du projet

La stéréoscopie est un ensemble de techniques visant à créer ou améliorer la perception de relief à partir de deux images planes.

Le projet consiste à développer un module permettant d'évaluer une distance à partir de deux images planes en entrée. Ce projet prendra la forme d'une implémentation du module sur un système mobile à roues. L'objectif final étant de concevoir un robot suiveur, capable de suivre une personne à une distance donnée.

## 2 Domaine : Vision stéréoscopique

Aujourd’hui devenu peu coûteux et peu encombrant, les systèmes de vision stéréoscopiques sont de plus en plus répandus.

On trouve de nombreux domaines d’application dont les suivants :

- Système de freinage automatique chez les voitures (Toyota par exemple).
- Détection d’obstacle chez les voitures autonomes.
- Reconnaissance d’objets chez les robots.
- La réalité virtuelle

## 3 Cahier des charges

### 3.1 Besoins fonctionnels

Cette section décrit les besoins fonctionnels qui doivent être remplis par le module créé.

#### OpenCV et QT

Cette première partie décrit les besoins fonctionnels relatifs à la partie OpenCV et QT du projet.

##### Split

Le module doit pouvoir séparer verticalement une image en deux images de même taille.

##### Détection de bords

Le module doit pouvoir détecter et mettre en valeur les bords d’une image.

##### Carte de disparité

Le module doit pouvoir calculer une carte de disparité à partir de deux images sources.

##### Carte de profondeur

Le module doit pouvoir, à partir d’une carte de disparité, calculer une carte de profondeur.

##### Conversion QImage/cv : :Mat

Le module doit pouvoir convertir une image d’un format à l’autre.

## **Unity**

### **Simulation**

L'environnement Unity doit permettre d'obtenir des images stéréoscopiques.

## **Robot**

### **Carte de profondeur**

Le robot doit pouvoir calculer une carte de profondeur à partir de deux images en entrée en utilisant le module crée.

### **Estimer une distance**

Le robot doit pouvoir obtenir une distance à partir d'une carte de profondeur.

### **Déplacement**

Le robot doit pouvoir avancer ou reculer en fonction d'une distance estimée.

## **3.2 Besoins non fonctionnels**

### **OpenCV et QT**

#### **Robuste**

Les distances estimées doivent être ordonnées de la même façon que les distances réelles, sous des conditions variables de luminosité, ombres ou mouvements.

#### **Rapide**

L'estimation de la distance doit être suffisamment rapide afin de suivre une personne marchant à vitesse normale (5km/h). Par manque de test, la rapidité n'est pas quantifiée.

## **Unity**

### **Automatisation**

L'acquisition d'image de synthèse doit être automatisable. Pour pouvoir prendre plusieurs images d'un seul coup par exemple.

## **Robot**

#### **Rapide**

L'estimation de la distance doit être suffisamment rapide afin de suivre une personne marchant à vitesse normale (5km/h). Par manque de test, la rapidité n'est pas quantifiée.

## 4 Architecture du code : partie QT et OpenCV

### convert.cpp convert.h

Ces fichiers permettent de convertir des images stockées sous le format d'openCV (cv : :Mat) en image sous format QT(QImage) et inversement. Il existe deux fonctions principales : **mat2QImage** et **qImage2Mat**.

### edit.cpp edit.h

Ces fichiers forment le "core" de notre module. Sont incluent la plupart des algorithmes utilisés lors des calculs de carte de disparité ou de profondeur, de détection de bords, de détections de points d'intérêt, etc...

**split** : Cette fonction permet de séparer verticalement une image en entrée, en deux images de même taille. Elle prend en entrée une image de type QImage. Cette QImage sera coupée en deux et chaque partie ainsi obtenue sera respectivement stockée dans les cv : :Mat dont les adresses sont données en paramètres.

**sobel** : Cette fonction permet de détecter les bords d'une image. Elle prend en entrée une image de type cv : :Mat. La fonction va effectuer sur l'image source une détection de bords, puis stockera le résultat ainsi obtenu dans une autre image cv : :Mat dont l'adresse est donnée en tant que paramètre.

**surf et surfmatch** : Ces deux fonctions permettent d'effectuer une détection de points d'intérêts. La fonction **surf** prend une seule image de type cv : :Mat en entrée. Elle va détecter les points d'intérêt sur l'image en entrée puis les mettre en valeur. Le résultat obtenu sera stocké dans une autre image cv : :Mat dont l'adresse est donnée en tant que paramètre.

La fonction **surfMatch** fait la même chose mais avec 2 images en entrée, toujours de type cv : :Mat. Elle va, en plus de la détection de points d'intérêts, effectuer une mise en correspondance des points d'intérêts entre eux. Le résultat obtenu sera stocké dans une cv : :Mat de destination dont l'adresse est en paramètre de la fonction.

**dispMap** : Cette fonction permet de calculer une carte de disparité à partir de deux images cv : :Mat en entrée. Elle fait appel à l'algorithme **StereoBM** de la bibliothèque **OpenCV**. La carte de disparité ainsi obtenue est stockée dans une image cv : :Mat dont l'adresse est donnée en paramètre.

### main.cpp

Il s'agit du code exécuté lors du lancement du programme. On distingue 2 parties. Une première partie avec GUI correspond au code généré par défaut par QT. Elle permet de lancer le programme avec l'interface graphique.

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Une deuxième partie permet d'exécuter des parties différentes du code en fonction de l'argument rentré lors de l'appel du programme. Si l'appel au programme se fait par la commande :

```
$ ./projet 0
```

Alors le programme va traiter toutes les images dans les sous-dossiers écrits en dur dans le code, dans ce cas précis il récupère les images dans le dossier source/cam et écrit dans le dossier result/cam avec l'appel suivant :

```
for (int j = 0 ; j < 3 ; j++) {
    for (int i = 0 ; i < 5 ; i++) {
        std::ostringstream ossI, ossO;
        ossI << "source/cam" << j << "-dist" << i << ".png"; //path fichier entrant
        ossO << "result/cam" << j << "-dist" << i << ".png"; //path fichier sortant
```

Le nombre d'images traitée est  $i*j$  et est rentré en dur dans le code. Cette partie a été écrite pour faciliter le traitement d'un grand nombre d'images, afin de tester les cartes de profondeur sur diverses situations.

Une troisième partie très similaire à celle décrite plus haut, à la différence pret que les images en entrée sont au nombre de deux et ont déjà était split. L'implémentation est la même avec la quantité d'images écrite en dur dans le code, tout comme les dossiers contenant les images. L'appel se fait par :

```
$ ./projet 1
```

Enfin un dernier appel au programme peut être fait permettant de tester plusieurs valeurs de SADWindowSize et de ndisparités pour une image rentrée dans le code. Celà nous a permis de faire des tests sur les valeurs optimales de StereoBM sur une certaine image, avant notre implémentation des trackbars. L'appel se fait par :

```
$ ./projet 2
```

**sgbm.cpp sgbm.h**

Ces fichiers contiennent les deux fonctions permettant de créer l'interface dynamique de gestion des paramètres du calcul de carte de disparité.

**dispSGBM** : Cette fonction charge des images sources en mémoire, initialise les paramètres de calcul de l'algorithme SGBM ainsi que les trackbars permettant de modifier dynamiquement les paramètres.

**update** : Cette fonction permet de mettre à jour les paramètres lorsque l'utilisateur agit sur une des trackbars. Elle rafraîchit également l'affichage des cartes de disparité.

## 5 Architecture du code : partie Unity

### move.cs

Ce script permet à l'objet auquel il est attaché de se déplacer dans une scène Unity en utilisant les flèches directionnelles. Par exemple, le code suivant est exécuté lorsque l'utilisateur appuie sur la flèche avant.

```
if (Input.GetKeyDown(KeyCode.UpArrow)) {
    Vector3 position = this.transform.position;
    position.y++;
    this.transform.position = position;
}
```

### capture.cs

Pour que ce script fonctionne il faut qu'il y ait 3 objets avec les tags suivant : camera\_left, camera\_right, bob. Ce script est à attacher à n'importe quel objet d'une scène. Ce script va déplacer

les objets et rendre en tant qu'image la vue des caméras.

Dans notre simulation, camera\_left et camera\_right représentent nos caméras stéréo et bob représente un humain à une certaine distance de ces caméras. Le script permet de prendre des captures d'écran en faisant varier la distance entre les caméras ainsi que la distance caméras-humain.

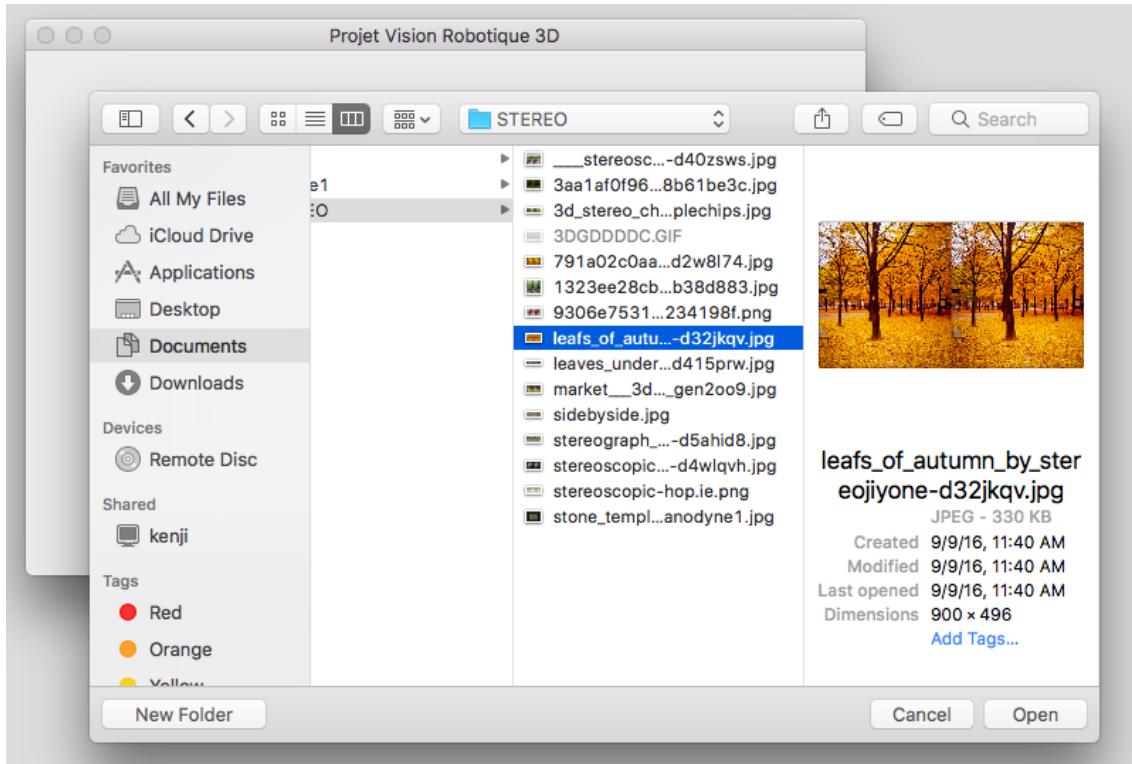
Voici la boucle principale du script :

```
for (int j = 0; j < 3; j++) {
    for (int i = 0; i < 5; i++) {
        // Render a picture and save it
        move ("bob", 'z', 1); // Move object with tag "bob"
    }
    // reset bob to initial position and move each camera
    reset ("bob", initial);
    move ("camera_left" , 'x', -0.02f);
    move ("camera_right" , 'x', 0.02f);
}
```

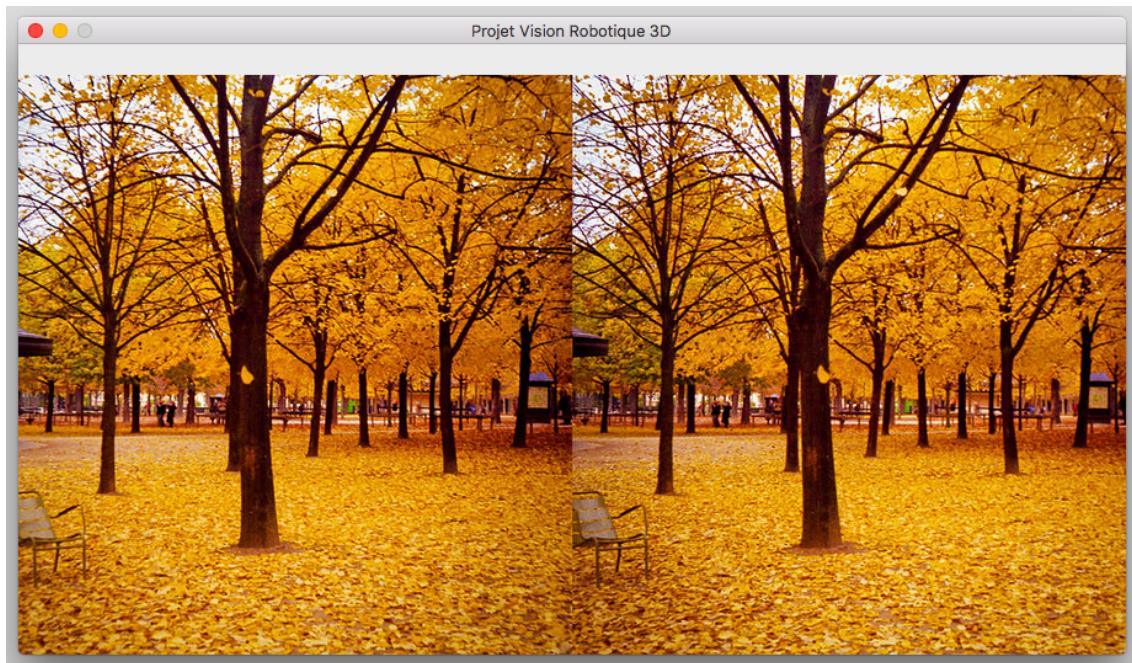
## 6 Tests

### Ouverture d'une image

Pour ouvrir une image, il faut aller dans le menu **File** puis appuyer sur "Open File".



Et le résultat :



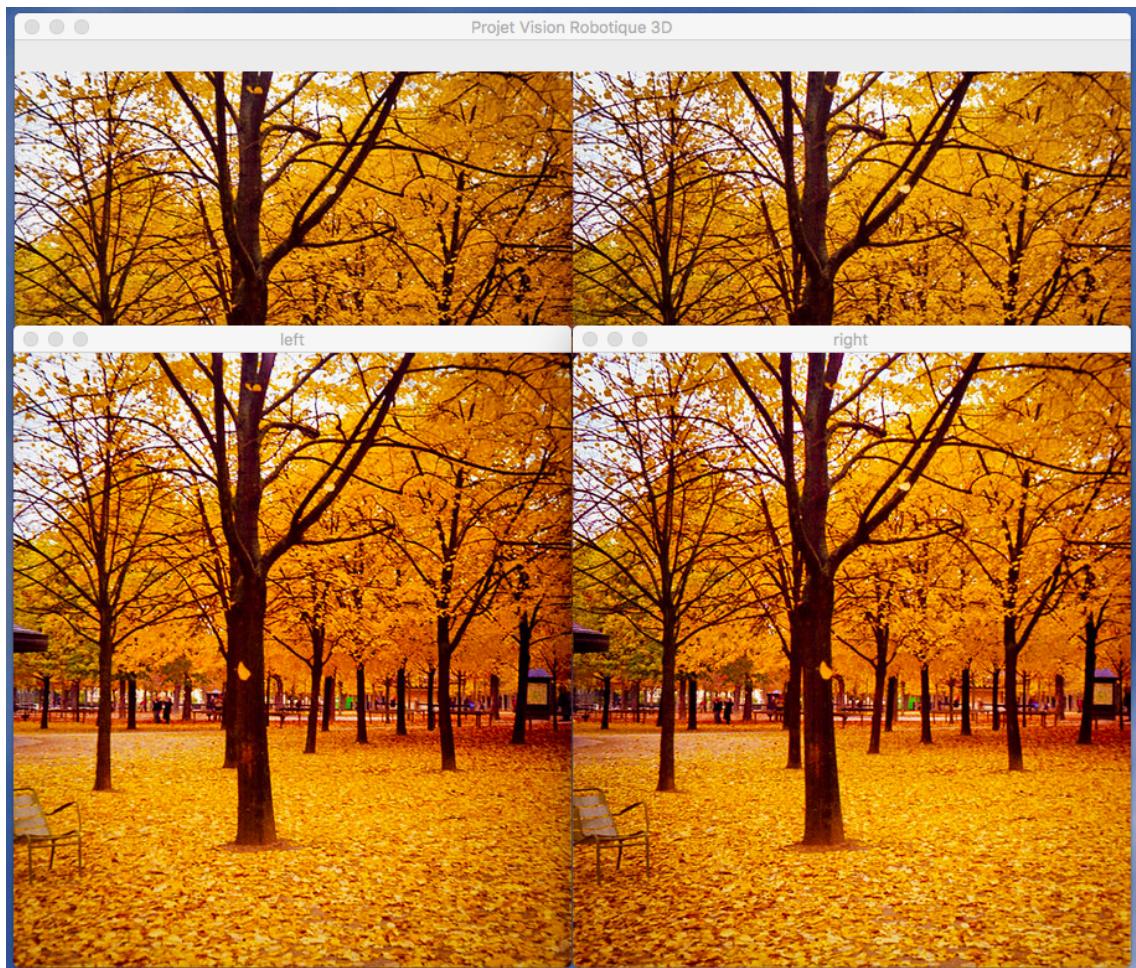
## Conversion de format

Pour convertir une image, il faut en avoir une d'ouverte. Puis aller dans le menu **Edit** et cliquer sur "Convert test". Cette fonction va convertir l'image de base, chargée comme QImage, en cv : :Mat puis la reconvertisr en QImage avant de l'afficher. Cela permet de tester la conversion dans les deux sens.

## Split

Pour séparer une image en deux, il faut tout d'abord en avoir ouvert une. Puis aller dans le menu **Edit** et cliquer sur "Split test".

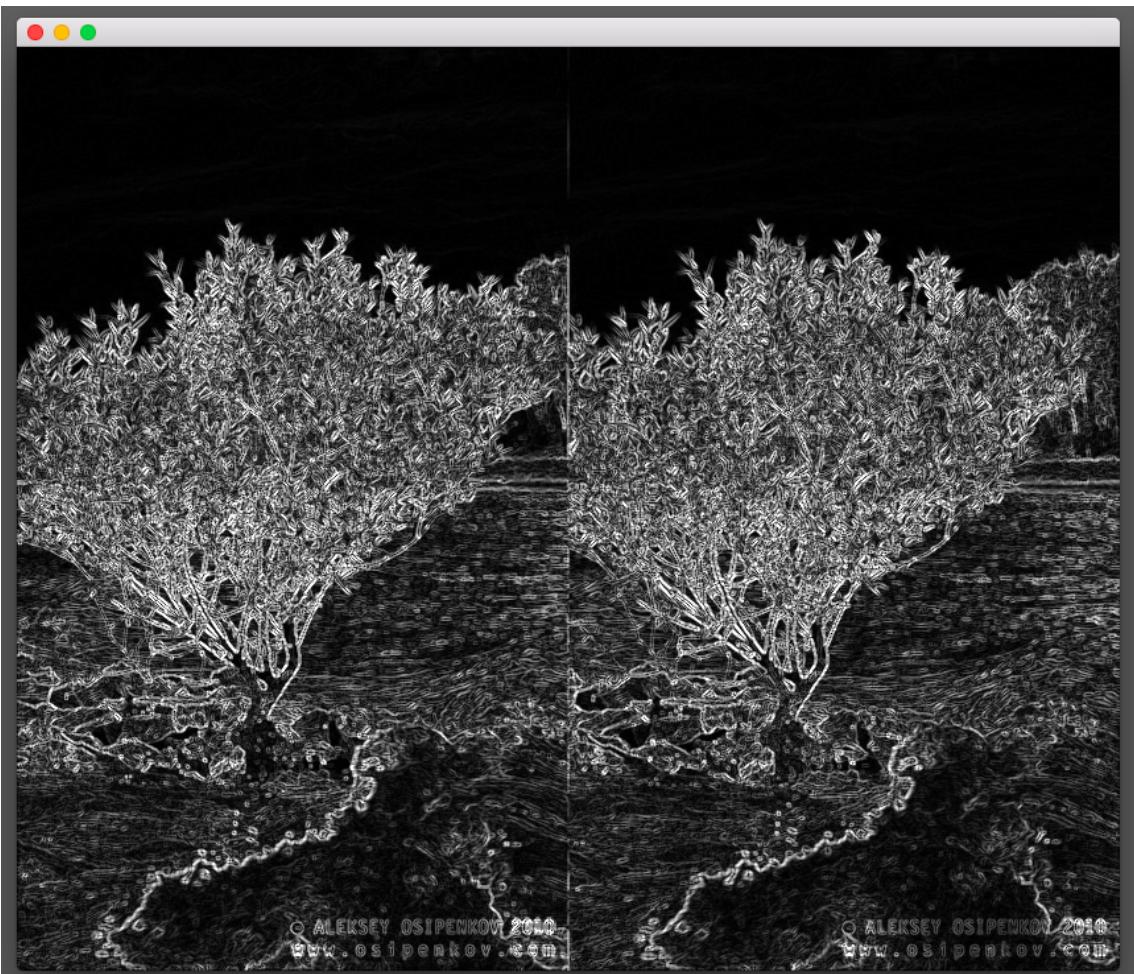
Voici le résultat obtenu :



## Détection de bords

Pour effectuer une mise en valeur des bords d'une image, aller dans le menu **Edit** et cliquer sur **Sobel test**.

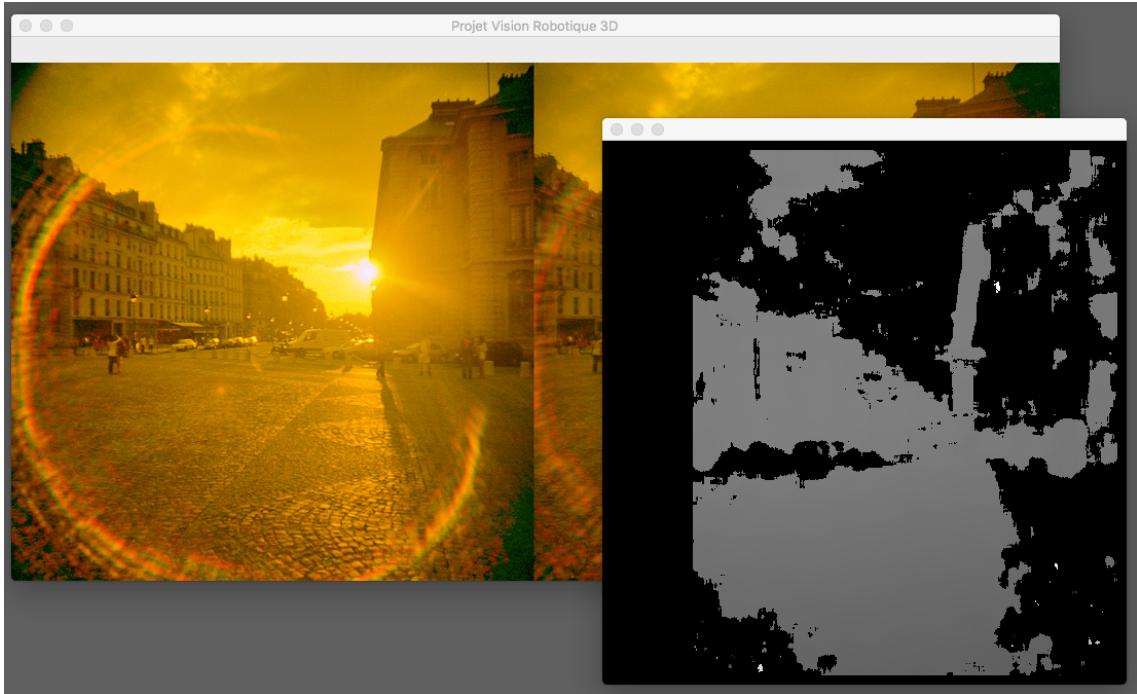
Voici le résultat obtenu :



## Carte de disparité

Pour afficher une carte de disparité, il faut ouvrir une image. Puis, aller dans le menu **Edit** et appuyer sur "Disp map test". La séparation verticale est automatique, il n'est pas nécessaire de "split" l'image avant.

Et le résultat :



Nous n'avons pas écrit de fonction permettant de tester les différentes fonctionnalités de notre programme, même si on peut assimiler les différents appels à notre programme avec les arguments "0", "1" ou "2" comme des tests pour les fonctionnalités suivantes :

- Ouvrir une image
- Convertir une Qimage en cvMat et l'opération inverse
- Split une image en deux images de même résolution afin de les traiter
- Obtenir une carte de disparité
- Ouvrir une nouvelle fenêtre

Nous avons cependant testé toutes les fonctionnalités que notre programme est censé contenir dans le cahier des charges.

## 7 Problèmes rencontrés

### Installation OpenCV

Lors des premières séances, nous avons eu des difficultés à installer OpenCV. En effet, utilisant les trois OS majeurs (Windows, Mac, Linux), la procédure d'installation n'était pas la même et nous n'avons pas pu nous aider les uns les autres facilement. Ceci a retardé un peu l'implémentation des fonctions OpenCV, et nous avons même abandonné l'installation sous Windows.

### Carte de disparité

Afin de traiter la carte de disparité, nous avons dans un premier temps utiliser l'algorithme StereoBM. L'appel à celui-ci sur les images tests fournis par l'enseignant nous rentraitait un résultat satisfaisant lorsque nous n'utilisions aucun paramètre. Nous avons donc garder cette implémentation naïve (pas sûr pour cet adjectif) de StereoBM.

Lorsque nous en avons du traiter les images issues du robot, les valeurs par défaut de cet algorithme ne nous donnait pas de résultat concluant. Nous avons donc eu besoin d'implémenter une variation des paramètres de StereoBM dans notre code, afin de mieux contrôler le résultat. Après avoir essayé pendant plusieurs séances d'améliorer les cartes de profondeur avec cette méthode, nous avons abandonné au profit de l'utilisation de StereoSGBM. Ces deux algorithmes étant similaires dans leur implémentation, nous n'avons pas eu beaucoup de temps pour adapter

notre programme. Nous avons très rapidement obtenu un meilleur résultat avec beaucoup moins d'effort.

## Conversion d'images

Notre conversion d'images a encore des problèmes de source inconnues. En effet, lors de l'appel du split depuis l'interface graphique, les images ne sont pas traités et la fonction split affiche deux fois l'image de gauche. Cependant le split est le seul moment où nous avons remarqué ce comportement.

## 8 Conclusion

Au travers de ce projet nous avons pu acquérir plusieurs connaissances. Premièrement, nous avons pu découvrir un nouveau domaine, la reconnaissance visuelle. D'un point de vue technique, nous avons pu approfondir nos connaissances en C++ et nous avons également pu apprendre à nous servir d'une librairie open source, OpenCV. De plus, ce projet présentait un plus par rapport aux autres projets que nous pouvons réaliser dans le cadre scolaire dans le sens où il permettait d'aller se confronter aux contraintes physiques.