Projet Technologique

VISION STÉRÉOSCOPIQUE

Geoffrey MEILHAN
Mohamed ALAMI
Kenji FONTAINE

25 avril 2017

Table des matières

1 Description du projet **3**

2 Domaine : Vision stéréoscopique **3**

3 Cahier des charges **3**

4 Architecture du code : partie QT et OpenCV **3**

5 Architecture du code : partie Unity **4**

6 Tests **4**

7 Conclusion **4**

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

And got so far

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 adresse 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 stockée dans une autre image cv : :Mat dont l'adresse est donnée en tant que paramètre.

La fonction **surfMath** 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();
}
```

5 Architecture du code : partie Unity

But in the end

6 Tests

- ouvrir une image - détection de bords - split - depth/disp map - unity (scripts) -

It doesn't even matter

7 Conclusion

OPPA GANGNAM STYLE