Rapport final

-

Vision en ligne de commande

Auteur
Louis Lynch

Professeur superviseur
Alain April

Date
2011-08-05

-

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Résumé

Depuis toujours longtemps les systèmes UNIX se sont développés autour du concept de flux de donnée, de filtre et de connecteur. C'est concept qui encourage les développeurs à faire de petits logiciels exprès. Il est alors possible de relier une série de ces logiciels indépendants pour créer une infinité de systèmes et de solutions. Ce mode de développement encourage la réutilisation d'outils très performants et permet l'automatisation d'opérations souvent utilisées.

Les liens entre processus sont rendus possible grâce à l'architecture filtre et connecteurs, mais surtout grâce à l'interpréteur de commande. L'interpréteur de commande est une interface un peu difficile d'approche pour des néophytes, mais devient un atout considérable pour un utilisateur expérimenté. Les interpréteurs de commandes accompagnées d'outils complets et flexibles permettent à de nombreux développeurs d'effectuer à la volée des opérations complexes, qui prendraient beaucoup de temps avec une interface graphique, en quelques minutes, voire même quelques secondes dans certains cas.

Les manipulations possibles avec sur des flux contenant du texte, ou encore la manipulation de fichiers du disque sont rendues possibles du fait que des outils sont développés à ces fins depuis des dizaines d'années. Le même processus peut certainement s'appliquer dans le contexte de la manipulation d'images. Soit dans le but de faire l'acquisition, le traitement, la publication ou encore un simple enregistrement des données.

La manipulation de flux de donnée en ligne de commande a fait ses preuves dans de nombreux domaines. Il est certainement possible d'utiliser ce modèle de développement pour

le traitement d'images. Ce projet a donc pour but de faire une expérience pour confirmer que le concept de filtres et connecteurs entre processus experts est applicable au traitement de vision artificiel. Tant pour le développement qu'au point de vue de la manipulation de flux et de données.

1.2 Groupe OCTETS

Le groupe OCTETS est un regroupement de clubs scientifiques de robotique de l'ETS. Les clubs membres pour le moment sont : Capra, Dronolab, SONIA et Walking Machine. Le but de ce regroupement est de partager et développer communément des logiciels utilisés par plusieurs clubs.

1.3 Outils actuels de vision

Un ensemble d'outils existent pour faciliter le développement d'application de traitement de vision artificiel. Ces outils sont fonctionnels et répondent à un ensemble de besoins des différents clubs, mais ils ont aussi quelques limitations et semblent parfois lourds pour de petites tâches.

1.3.1 Serveur de vision

Parmi les outils mis en commun, il y a principalement un serveur de vision, un client de vision et un éditeur de vision. C'est dans le serveur de vision que nous implémentons nos filtres de visions. Le serveur de vision charge une liste de filtres à partir de la description fournie par le biais d'un fichier de configuration portant l'extension "filterchain". Cette liste de filtre est alors instanciée. L'instanciation est faite de manière à relier les filtres les uns à la suite des autres dans un ordre spécifié dans la configuration. Les filtres possèdent différentes propriétés ajustables directement lors de l'exécution. Ces propriétés peuvent s'enregistrer dans le fichier "filterchain".

Le serveur de vision est aussi responsable de faire l'acquisition des images. Il peut faire son acquisition d'image en utilisant plusieurs sources de données différentes. Par exemple, il peut faire une capture directement à partir d'une caméra, il peut aussi charger des images à partir d'un répertoire donné et finalement il peut lire les images qui proviennent d'un vidéo. Une fois la première image acquise, il la passe à chacune des chaînes de filtres actives. Une fois arrivée dans la chaîne de filtre, une image est passée successivement de filtre en filtre

pour être traitée.

Les filtres peuvent aussi émettre des données sous forme de description. Par exemple, un filtre pourrait être chargé d'identifier des points rouges dans l'image et de transmettre la coordonnée de ceux qui sont trouvés. C'est là que le nom serveur de vision prend son sens. Le serveur de vision est capable de publier les données fournies par le serveur de vision. Le but étant de permettre à un client de venir s'y connecter pour recevoir les descriptions émises par le serveur. C'est de cette manière que les logiciels d'intelligence artificielle reçoivent des données de haut niveau nécessaires pour prendre leurs décisions.

1.3.2 Client de vision

Le client de vision est une application qui sert à se connecter sur le serveur de vision. Avec cet outil il est possible d'avoir un aperçu du rendu de chacun des filtres de la chaine. C'est aussi possible de changer la valeur des propriétés de la chaîne pour faire des ajustements. C'est aussi possible à partir de cette interface activer ou désactiver des chaines de filtres.

1.3.3 Éditeur de vision

L'éditeur de vision quant à lui sert à créer des chaines de filtres. Il permet de charger une banque d'images ou un vidéo et de voir bâtir une chaine de filtres à partir de filtres existants. C'est possible avec cet outil de choisir les filtres à utiliser, leur ordre d'exécution et les valeurs associées à chacune de leurs propriétés. L'outil permet aussi de voir le résultat de chacune des étapes.

1.4 Limitations des outils

Les outils qu'utilisent les membres du groupe OCTETS ont quelques limitations. Par exemple, il est impossible de faire du traitement sur des images qui n'ont pas la taille fixe de 640x480x3.

Aussi les chaines de filtres doivent avoir une source et une sortie, ni plus ni moins. Cette contrainte fait en sorte qu'il est impossible de faire des embranchements dans le flux d'images. Par exemple, une image doit commencer par le filtre 1, puis passer par le filtre 2 et ensuite se rendre au filtre 3, si le filtre 3 avait besoin de l'image de sortie du filtre 1, ce serait impossible pour lui de l'obtenir. Dans une telle situation, il faut obligatoirement faire

un immense filtre qui inclue les étapes 1 à 3 en un seul module. Cette limitation nuit donc à la flexibilité du système et le reste du système ne bénéficie pas des avantages de la réutilisation.

Dans d'autres cas, c'est une perte de performance qui est observée. Supposons que plusieurs chaines de filtres soient configurées pour rouler en parallèle et qu'elles commencent toutes par une opération de filtre Gausien, dans ce cas il faut refaire l'opération dans chacune des chaines de filtres.

1.5 Efficacité des outils

L'éditeur de vision est assurément un outil très efficace pour ce qui est de mettre au point une chaine de filtre à partir de filtres existant. Il permet de faire des essais avec différente combinaison de filtres et de paramètre. Le tout est modifiable en temps réel pendant la lecture d'un vidéo. C'est-à-dire qu'il est possible, changer les paramètres et observer au même moment les effets de ces changements.

Là où cet outil est moins performant, c'est au moment de la création d'un nouveau filtre. Quand le fonctionnement du filtre lui-même n'est pas encore au point, on passe de l'implémentation à l'exécution de manière très fréquente. Pour exécuter un filtre après une modification dans le code, il faut redémarrer le serveur de vision, une application qui prend plusieurs secondes pour décoller. Ensuite il faut aussi faire quelques manipulations pour aller observer le rendu du filtre en questions. L'opération est à répéter chaque fois qu'un changement est désiré.

En ajoutant le temps nécessaire au chargement de l'application et à la compilation de la solution, c'est une opération qui ralentit considérablement le développement et la mise au point de nouveaux filtres.

Chapitre 2

Description du projet

2.1 Solution proposée

Avec ce projet c'est une façon de penser différente qui est proposée. Une solution qui s'inspire de la méthode de manipulation de flux qui existe sous les systèmes Unix. Dans les systèmes Unix, les différents systèmes sont mis sur pied en connectant plusieurs logiciels experts ensemble pour arriver à un résultat.

Le concept permet de bâtir une sorte de langage spécifique au domaine d'application. C'est avec l'aide de l'interpréteur de commande et une série de petits logiciels indépendant et bien pensé qu'il sera possible de créer un genre mini langage pour faciliter la manipulation des flux d'image.

Avec ce projet de vision en ligne de commande l'objectif est de découpler les différents modules du système en créant une série de petits programmes experts. C'est différents logiciels agissants sur un flux de donnée et peut être agencé de plusieurs manières différentes. C'est en utilisant un langage de script, tel que bash, qu'il est possible de créer de nouveau système en réorganisant la chaine de processus.

L'utilisation de la ligne de commande permet de structurer un système en reliant plusieurs logiciels compilés indépendamment. C'est en utilisant plusieurs langages qu'il est possible de résoudre des problèmes de manière optimale en un minimum de temps.

2.2 Objectifs

L'objectif du projet est de faire la promotion du concept auprès du groupe OCTETS et de démontrer la faisabilité du concept multiprocessus. Il faut tester les limites et connaitre les forces de ce modèle de développement. Il faut aussi voir si le concept est utilisable dans un environnement temps réel pour tous les clubs partageant le répertoire de vision.

L'étape de ce projet est de concevoir un environnement simple d'utilisation pour le développement. Quand vient le temps de programmer un script ou un programme s'exécutant en ligne de commande, les invocations sont toujours effectuées rapidement. Certain développeur travail dans vim et sont très près de l'interface ligne de commande, ce qui permet de tester un script sans avoir a quitter l'éditeur. Le but est de permettre cette rapidité d'invocation qui est possible lors du développement d'application en ligne de commande au développement d'application de traitement de vision.

L'indépendance des logiciels/filtres est aussi un aspect important. Il faut pouvoir utiliser un filtre indépendamment. En encourageant cette manière de développer il sera plus facile de faire le tri de ce qui doit être maintenu et ce qui n'est pas nécessaire de maintenir.

Un autre objectif est de permettre à un programmeur de démarrer un petit projet très simple et de bénéficier de reste des outils disponibles dans l'environnement en limitant au minimum le couplage et la courbe d'apprentissage.

Un autre objectif est de rendre la manipulation d'image le plus accessible possible. Il faut que ce soit rapide de composer une solution à un problème en réutilisant une banque d'outils. Il faut rendre ces opérations aussi simples que de manipuler des fichiers ou des flux de données avec les outils de base fournis avec les systèmes UNIX.

Il faut aussi permettre à un développeur de créer des utilitaires de vision de manière très rapide et efficace. C'est important de pouvoir concevoir de nouveaux outils sans changer le comportement d'un outil partagé par l'ensemble des clubs. Il faut permettre à un sous logiciel d'exister sans que tout le reste vienne avec.

2.3 Avantages

Le but principal objectif est de pouvoir exécuter un filtre ou une chaine de filtre sans avoir à instancier une série d'outils qui prennent beaucoup de temps à démarrer. Dans certain cas il est trop tôt pour déboguer le rendu du filtre, la seule chose qui est voulue c'est de vérifier si l'exécution simple d'une analyse d'image fonctionne bien sans qu'une erreur de segmentation ou autre ne soit lancée. Dans d'autres cas, c'est sur la logique du filtre que le travail se fait. Dans cette situation relancer le traitement le plus rapidement possible pourrait se traduire par un gain de temps substantiel.

Cette expérience a pour but de rendre les outils de vision plus flexible et plus facilement réutilisables. Dans le cadre du développement, certaines opérations sont redondantes. Par exemple, toutes les applications commencent par faire l'acquisition d'images. Les mécanismes d'acquisition d'image sont souvent répliqués. Une application en cours de développement commence souvent par lire les images à partir d'une caméra ou d'une liste d'images. Cette opération demande de supporter une forme de configuration pour permettre à l'utilisateur de changer de sources de donnée.

Avec une approche par processus tous les éléments du système sont indépendants. Les processus ont chacun leurs responsabilités et sont experts à ne faire qu'une seule chose. En utilisant plusieurs logiciels experts, c'est possible modifier la structure du système pour l'adapter en fonction du problème à résoudre.

Le principal avantage de cette méthode, c'est que le développeur d'une application n'a pas à se soucier d'intégré son filtre au système. Lors du développement d'un nouveau filtre, il est facile de démarrer, ce n'est pas nécessaire de se coupler à autre chose que la librairie commune. Un nouveau filtre peut s'intégrer dans différents systèmes, sans que l'auteur du programme y ait pensé au moment de la conception.

C'est la flexibilité des connexions et des structures qui sont intéressantes avec ce projet. Cette flexibilité encourage la réutilisation de différents outils qui sont complètement découplés les uns des autres. Ce découplage se traduit par l'écriture de sous-programmes plus simple à maintenir. Aussi l'écriture d'applications "jetables" est possible. C'est possible de faire une expérience avec un simple corps de programme c++ de tester certaines choses, sans polluer le code ou la structure des autres outils.

Un autre des points fort avec l'approche par filtre et connecteur c'est de pouvoir créer des

utilitaires qui se manipulent très rapidement. Par exemple, faire l'enregistrement d'un vidéo provenant avec une caméra et le visionner peut se faire plus rapidement en ligne de commande qu'en utilisant les autres outils existants à OCTETS. L'automatisation de certaines tâches est donc possible par la même occasion. Cette automatisation prend la forme d'un simple script.

Les structures de filtres/processus possibles avec l'approche ligne de commande sont plus permissives que la simple chaine de filtres qui s'exécutent séquentiellement. C'est possible de faire des structures qui s'apparentent à des arbres ou des graphes.

Depuis un moment déjà c'est une idée qui court à OCTETS de faire un système de plug-in pour que tous les filtres ne soient pas tous partagés dans un répertoire commun. Cette fragmentation n'est pas possible avec le système actuel, mais se ferait implicitement avec un système distribué en plusieurs processus. La fragmentation de logiciel permettrait à un club de ne pas partager les outils ou filtres qui ne sont pas utilisés par les autres clubs.

Chapitre 3

Scénarios d'utlisation

3.1 Description

Dans cette section quelques exemples sont présentés afin de faire comprendre de manière plus concrète la vision et les manipulations que permet une implémentation qui réduit la portée d'un filtre à un seul processus expert découplé de l'ensemble du système.

3.2 Scénarios d'utilisation

Les outils de ligne de commande permettent d'effectuer des opérations complexes et de manipuler des données. Dans cette section quelques scénarios d'utilisation sont présentés pour permettre au lecteur de mieux comprendre les objectifs du projet.

En prenant en compte le fait que les principales opérations qu'effectue un système de reconnaissance de vision artificiel sont :

- Faire l'acquisition d'images
- Faire le traitement des images
- Publier ou présenter les résultats
- Enregistrer des données

Les outils permettent de rapidement faire des tests sur n'importe quelle source d'image. Ils permettent aussi d'appliquer des transformations simples et de comparer les résultats.

Voici quelques exemples de problèmes qui peuvent être rapidement accomplis avec le système.

Ici, c'est la base, il s'agit de faire l'acquisition d'images pour les passer à un autre processus.

```
camera | viewer
load-image *.png | viewer
generate-random-images | viewer
load-video | viewer
cat raw-data | viewer
netcat imageserver 3333 | viewer
```

FIGURE 3.1 – Faire l'acquisition d'images

Dans cet exemple c'est la structure possible pour le traitement qui est développée.

```
# exemple simple
camera | blur | viewer

# exemple simple en chaine
camera | blur | threshold | viewer

# exemple complex de filtre avec des branches
mkfifo a
mkfifo b
mkfifo aa
mkfifo bb
camera | split a b &
cat a | blur > aa &
cat b | threshold > bb &
merge aa bb | filtre-a-deux-source | viewer
rm -rf a b aa bb
```

FIGURE 3.2 – Faire le traitement des images

Le prochain exemple montre quelques manières intéressantes de publier et enregistrer des données.

Les cas d'utilisations précédents montrent quelques problèmes qui peuvent être résolus de manière très simple en utilisant le démo. Problèmes, qui ne sont pas possibles avec l'implémentation actuelle. Ce sont ce genre d'exemples qui ont motivé la réalisation du projet, sans avoir d'exemple d'utilisation il est difficile de voir de quelle manière peut être utile un système de vision en ligne de commande.

3.3 Exemple d'implémentation

En prenant l'exemple de l'implémentation d'un nouveau filtre, on voit ici tout code du squelette nécessaire à l'implémentation. C'est relativement simple à comprendre et ça permet un découplage complet du reste de l'application.

```
# publication vers une socket
camera | netcat -kl 3333

# visualisation a partir d'une socket
netcat hostname 3333 | viewer

# enregistrement dans un fichier
camera > enregistrement

# enregistrement compressee
camera | gzip > enregistrement.gz

# visualisation des donnees compressee
zcat enregistrement.gz | viewer

# enregistrement sur un serveur distant
camera | gzip | ssh hostname 'cat > enregistrement.gz'
```

FIGURE 3.3 – Publication et enregistrement

Noter que les instructions ‘img = createImage()‘ et ‘releaseImage(img)‘ sont utilisés pour la gestion de la mémoire et que les instructions ‘readImage(in)‘ et ‘writeImage(out)‘ permettent l’échange des images entre les processus.

```
#include "pipe-commons.h"

int main() {
    // allocate memory
    IplImage* in = createImage();
    IplImage* out = createImage();

    // process each input
    while(readImage(in)) {
        process(in, out);
        writeImage(out)
    }

    // release memory
    releaseImage(in);
    releaseImage(out);

    return 0;
}
```

FIGURE 3.4 – Exemple de création d’un nouveau filtre de vision

Après la création de la première version du filtre, il faut en faire l’exécution. Dans la plupart des cas, la première exécution ne sera pas au point. Il n’est donc pas nécessaire de faire le chargement d’une application extrêmement complexe. Souvent à cette étape le simple fait d’exécuter le filtre en lui faisant traiter quelques images nous permettrait de trouver des erreurs de dépassement de tableau ou de gestion de la mémoire. Ensuite, il faudra répéter l’opération un bon nombre de fois jusqu’à ce que le filtre fonctionne correctement. Ce n’est

pas avant que la mise au point soit terminée que l'éditeur de vision deviendra efficace.

Une fois que ce nouveau filtre sera compilé, il sera possible de lui fournir plusieurs sources de données différentes. Il sera aussi possible de transférer le résultat vers un autre filtre/système. Voici quelques exemples de connexions qui seront ensuite possibles :

```
camera|mon-filtre|viewer
load-image *.png|mon-filtre|record video.avi
load-video video.avi|mon-filtre|server
```

FIGURE 3.5 – Exemple de connexion entre différents filtres

Dans cet exemple l'acquisition des images est déléguée à un autre processus. Le processus recevant le résultat du traitement est aussi différent. Pour reproduire ces trois comportements avec le système actuellement utilisé, il faut aller modifier le fichier de configuration entre chaque invocation ainsi qu'utiliser une autre application telle que le client de vision pour voir le résultat. Ces manipulations demandent du temps et ne peuvent difficilement se faire en parallèle, alors qu'avec une approche par processus c'est possible de lancer autant de processus que nécessaire sans restrictions.

Chapitre 4

Réalisation

4.1 Utilisations des outils

Tout d'abord, les outils utilisés pour la réalisation de ce projet sont tous des outils lignes de commande. Beaucoup de développeurs C/C++ utilisent vim comme éditeur. Aussi la rédaction de rapport et de documentation en latex est préférée par bon nombre de développeurs. Quant à la gestion du versionnement, les outils sur Linux sont la plupart du temps utilisés en ligne de commande. D'autres outils tels que gnuplot pour tracer des graphiques, ainsi que des outils de base des systèmes UNIX sont utilisés. Les outils et interfaces en ligne de commandes deviennent très puissants quand on prend le temps de bien apprendre comment ils fonctionnent.

L'avantage d'outils pour faire la manipulation de flux d'images prend tout son sens quand dans le cadre d'un développement qui est fait en utilisant uniquement des outils lignes de commande. Par exemple, il est possible de compiler et d'exécuter à la fois le filtre en développement sans avoir à sortir de l'instance de Vim.

4.2 Itérations

Les approches itératives ont pour avantage de permettre à la conception d'évoluer au même rythme que la compréhension du problème dans notre esprit. En alternant les activités de conception et d'implémentation il est plus facile de voir de quelles manières l'implémentation pourra être rendue possible. Le résultat est un système mieux structuré qui résout les problèmes de manière intelligente.

C'est donc une méthode par itération qui a été préconisée pour la réalisation de ce projet.

Dans un premier temps, c'est l'élaboration du concept qui a été faite. Ensuite, une première expérience a été possible. Dans un premier temps, c'est la conception d'une solution du genre prototype ou démo qui a été faite. Ensuite l'implémentation du prototype a eu lieu et les résultats ont été documentés et présentés au groupe OCTETS. Lors de cette rencontre, les membres d'OCTETS ont fait part de leurs commentaires et ont parlé des attentes qu'ils ont d'un nouveau système de vision. Finalement, la conception d'une nouvelle version du logiciel utilisant de la mémoire partagée a été entreprise.

4.2.1 Élaboration du concept

L'élaboration du concept s'est faite sur une très longue période de temps. Depuis un moment déjà l'idée d'orienter le développement en utilisant des flux et des connecteurs faisait l'objet de discussion sur la liste de diffusion du groupe OCTETS.

Pour essayer d'évaluer les avantages possibles d'une architecture utilisant des flux connecteurs, quelques scénarios avaient déjà été envisagés. Ces scénarios ont permis de voir qu'on pourrait régler certains problèmes si on bénéficiait d'une approche flexible, permettant de connecter certains utilitaires ensemble.

C'est avec quelques scénarios en tête et certains objectifs bien précis que le projet fût lancé.

4.2.2 Conception du démo

Le démo était une manière de voir comment on pourrait arriver à manipuler des flux d'images comme prévu dans les scénarios d'utilisation. L'objectif était de rendre disponible une plateforme permettant de travailler avec le mécanisme en ligne de commande.

Il fallait avec la conception du démo prévoir une méthode qui permettrait de faire l'acquisition d'images, de traiter des images, de les manipuler, de les publier, de les enregistrer et de les visualiser. Il fallait surtout que toutes ces opérations soient découplées dans des programmes binaires complètement indépendants et qu'une communication soit possible entre ces petits logiciels.

Lors de la conception de ce démo, un choix important a dû être fait. C'est une question qui allait grandement influencer le temps de développement. Étant donnée la nature expérimentale de l'application, il a été jugé bon de faire un choix conceptuel qui réduirait le temps

de développement pour permettre de se familiariser avec l'approche le plus rapidement possible sans nécessairement avoir la solution la plus performante. La méthode optimale aurait été de faire une implémentation utilisant de la mémoire partagée pour éviter la copie des données. Cette méthode est efficace, mais demande un temps de développement qui excède celui alloué pour la réalisation de ce projet. Dans le but d'obtenir des résultats le plus rapidement possible pour voir si l'expérience est viable, une solution copiant l'information a été choisie.

La copie d'information entre les processus est la norme avec les outils UNIX (tels que : grep, cat, sed, wc, nl, etc...), mais pour le traitement d'image comme le traitement se fait sur une grande quantité de données il aurait été préférable de ne pas copier les données inutilement. Avant de prendre la décision, une expérience a été menée pour évaluer le temps de transfert nécessaire. L'expérience a révélé que le temps moyen d'échange pour une image 640x480x3 prenait environ 1,2 milliseconde sur un ordinateur ayant un dual core 2.1 Ghz. Ce temps de transfert a été jugé suffisamment petit pour permettre l'utilisation de cette méthode pour l'implémentation du démo.

4.2.3 Démo : Implémentation du mécanisme

Pour l'implémentation de ce démo, l'accent a été mis sur les manipulations rapides de flux d'images. Il fallait tester les limites et avantages de la méthode sans obtenir une solution parfaite. Il fallait vérifier si les outils en lignes de commande pouvaient réellement résoudre des problèmes plus rapidement que la solution conventionnelle.

Cette version démo devait fournir suffisamment d'outils pour démontrer son utilité. Initialement, l'implémentation a été lancée avec comme objectif de pouvoir charger des images, les transformer et les affichera. C'est donc avec des outils de base tels que : image-loader, blur et viewer que le système a commencé à prendre forme.

4.2.4 Démo : Implémentation d'outils

Après l'implémentation de ces outils, il a été possible de s'amuser avec les flux de donnée. Rapidement, de nouveaux besoins se sont manifestés. Par exemple, pour permettre au flux de donnée de se séparer pour parcourir deux sous-embranchements l'outil split a été développé.

Voici une liste, triée en ordre alphabétique, des outils binaires développés durant la phase d'implémentation du démo :

blur Effectue un simple filtre gausien sur l'image.

camera Capture des images a partir d'une caméra.

head Prend les 'n' premières images du flux, puis termine.

grayscale Transforme une image RGB en niveaux de gris.

image-loader Charge des images à partir de noms de fichiers.

image-saver Enregistre des images dans des fichers (ex. : png).

pipe-timer Calcul le temps de transfert nécessaire pour faire la copie d'une images vers au autre processus.

record Permet d'enregistrer ou de republier des images à intervalles réguliers. Exemple, pour conserver une image à toutes les deux secondes, record peut être utilisé comme ceci : 'camera | record 2s > sauvegarde'

tail Prend les n dernières images du flux.

head Prend les n premières images du flux.

merge Lis les images de plusieurs sources différentes pour les unifier dans un seul flux.

split Réécris les images provenant d'un flux vers plusieurs fichiers ou "named pipes".

threshold Fait une binarisatoin de l'image selon un certains seuil.

viewer Permet de lire des images pour ensuite les afficher.

Aussi certains scripts ont été développés pour simplifier certaines tâches. Ces scripts permettent parfois de créer de petites applications pratiques. C'est le cas d'un script nommé 'take-picture'. Ce script réutilise des logiciels binaires et les combine en ajoutant un peu de logique en bash pour permettre à un utilisateur de prendre une photo chaque fois qu'une touche du clavier est touchée. Quelques autres scripts ont été implémentés, notamment pour charger des images de tests, simplifier une chaine de transformation en un script simple, faire la sauvegarde et le chargement d'images sur un serveur distant, etc.

4.2.5 Limitations

Les résultats sont positifs du point de vue de la flexibilité, certaines tâches sont rendues possibles très simplement avec les outils, tâches qui ne sont pas envisageables avec le système actuel d'OCTETS. Le système à la base a été conçu pour que la première implémentation se fasse rapidement. L'objectif étant de voir si l'approche était viable. Les résultats sont suffisamment positifs pour s'attarder à trouver une solution aux limitations du démo.

Voici quelques une des limitations de la version démo :

– Impossible de visualiser toutes les étapes de chacun des filtres du système.
– Impossible de changer les paramètres à l'exécution.
– Temps de transfert des images est lent pour les processeurs embarqués.

Ces limitations sont principalement dues au fait qu'un processus au bout de la chaine n'a pas accès à l'espace mémoire des processus précédents. Il ne peut donc pas aller inspecter la valeur des paramètres de celui-ci, ni voir le rendu des images qu'ils produisent. Aussi les images doivent être copiées entre chaque processus, ce temps de copie n'est pas énorme, mais peu devenir important si on utilise beaucoup de connecteurs, c'est aussi problématique dans un environnement qui doit être le plus réactif possible.

4.2.6 Planification de la prochaine étape

Lors de rencontres avec les membres du club OCTETS, les fonctionnalités suivantes ont été identifiées comme étant les plus souhaitées par l'ensemble, ou du moins un sous-ensemble du groupe.

– Visualiser toutes les étapes d'une chaine de filtre.
– Modifier les paramètres des filtres à volonté.
– Faire la sauvegarde des paramètres d'une chaine de filtre.
– Modifier la structure de la chaine à l'exécution.

Lors de cette rencontre, le démo a été évalué en fonction de sa capacité à remplacer complètement le serveur de vision d'OCTETS. De manière générale l'intérêt des membres est suscité, mais certains doutes persistent quant à la faisabilité de remplacer tous les outils en réutilisant ce concept.

En effet pour que le projet soit accepté il faut pouvoir faire la démonstration qu'il est possible d'intégrer le coeur du système à tous les outils actuellement utilisés. Il reste à voir de quelle manière on peut remplacer complètement le serveur de vision et l'éditeur de vision.

4.2.7 Conception utilisant de la mémoire partagée

Pour régler les limitations dues aux problèmes d'accès aux images et aux paramètres des différents processus, ainsi que dans le but d'éliminer la copie inutile des données entre les processus, une solution mettant à profit un espace mémoire partagé a été élaborée. Un document de conception a été produit et la phase d'implémentation de cette solution a commencé.

4.3 Discusion

Le démo a permis de tester bon nombre de cas d'utilisations et de prouver que l'approche suggérée était extrêmement flexible. Certaines tâches ont été rendues possibles avec ce démo qui est absolument impensable avec l'architecture actuelle.

Pour le moment la version présentée en démo a quelques lacunes par rapport à l'implémentation déjà existante, mais ces lacunes pourront être réglées si on réimplémente le système en utilisant de la mémoire partagée.

En réimplémentant le projet avec de la mémoire partagée, ce serait possible de modifier les paramètres de tous les filtres pendant l'exécution. Il serait possible de publier le rendu des transformations à chacune des étapes à des fins de débogage.

Pour ce qui est de l'éditeur de vision, ce serait possible de créer directement des sous-processus et de reconnecter leur sortie pour modifier la configuration. Les mêmes fonctionnalités que le serveur de vision pourraient alors être implémentées pour faire la visualisation des différentes étapes et la modification des paramètres en temps réel. La principale difficulté resterait de permettre de modéliser des arbres de filtres plutôt que des listes de filtres.

Ce modèle de développement a définitivement des chances de coexister, voir même remplacer complètement les outils actuels.

Chapitre 5

Annexes

5.1 Artéfacts

Conception initial du projet : Document résumant la problématique et détaillant la conception utilisée pour la résolution du problème. C'est à partir de ce document de conception qu'a été implémentée la version démo du projet.

Implémentation du démo : Le code de l'implémentation du démo et des outils qui le composent. Les scénarios d'utilisations et autres exemples de code bash ont tous été testés et réalisables en utilisant les outils fournis par cette implémentation.

Résultats du démo : Dans ce document certains détails d'implémentation sont rapidement expliqués. Ensuite, les limitations et les forces de la solution sont discutées. Une section fait aussi état des performances de l'implémentation.

Résumé de rencontres avec OCTETS : Document énumérant les sujets abordés avec les gens du groupe OCTETS à différentes occasions.

Rapport d'étape : Document obligatoire, remis à la mi-session.

Présentation : Présentation numérique ayant pour but d'expliquer le projet de vision en ligne de commande.

Rapprot final : Ce document. Il décrit le contexte du projet, les méthodes de développements et les résultats obtenus.

5.2 Outils de développement

Composition des documents et rapports

vim - éditeur de texte (code et rapports)

latex - compilation de rapport au format tex

gnuplot - outils pour tracer des graphes

Outils pour le versionnement

git - gestionnaire de versionnement

ssh - synchronisation du versionnement

Manipulations en ligne de commande

bash - interpréteur de commande multiplateforme

pricipalement : cat, netcat, find, xargs, time, ssh, gzip, zcat, mkfifo

Libraries utilisées pour le développement

opencv - librairie de vision artificiel

highgui - librairie permettant d'afficher des images

boost-interprocess - librarie pour la gestion de la mémoire partagé

std :iostream - librarie standard pour la manipulation de flux de données

Outils nécessaires à la compilation

g++ - compilateur de c++

Make - gestionnaire de compilation des artéfacts

5.3 Réferences

Livres

– Gary Bradski et Adrian Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*, O'Reilly Media, Inc., 2008, p. 576

Sites web

– **Wiki LaTeX :** http ://en.wikibooks.org/wiki/LaTeX
– **Advanced Bash-Scripting Guide :** http ://tldp.org/LDP/abs/html/
– **OpenCv :** http ://opencv.willowgarage.com/wiki/
– **Boost - Interprocess :** http ://www.boost.org/doc/libs/1_47_0/doc/html/interprocess.html
– **Sémaphores :** http ://en.wikipedia.org/wiki/Semaphore_(programming)
– **Packaging guide :** https ://wiki.ubuntu.com/PackagingGuide/Complete