MINISTÈRE
DE L'ÉDUCATION
NATIONALE

EAE SIN 3

SESSION 2018

AGRÉGATION
CONCOURS EXTERNE

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

CONCEPTION PRÉLIMINAIRE D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Durée : 6 heures

Calculatrice électronique de poche - y compris calculatrice programmable, alphanumérique ou à écran graphique – à fonctionnement autonome, non imprimante, autorisée conformément à la circulaire n° 99-186 du 16 novembre 1999.

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Dans le cas où un(e) candidat(e) repère ce qui lui semble être une erreur d'énoncé, il (elle) le signale très lisiblement sur sa copie, propose la correction et poursuit l'épreuve en conséquence.

De même, si cela vous conduit à formuler une ou plusieurs hypothèses, il vous est demandé de la (ou les) mentionner explicitement.

NB : *La copie que vous rendrez ne devra, conformément au principe d'anonymat, comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé comporte notamment la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de signer ou de l'identifier.*

MINISTÈRE
DE L'ÉDUCATION
NATIONALE

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

EAE SIN 3

SESSION 2018

AGRÉGATION
CONCOURS EXTERNE

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

CONCEPTION PRÉLIMINAIRE D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Partie C Page 34 Programme Python Ligne 10

Au lieu de

N=D-G+1

Lire

N=D+1

Ce document est composé :

- d'un dossier sujet de 18 pages (numérotées de 2 à 19) ;
- d'un dossier technique (DT) de 18 pages (numérotées de 20 à 37) ;
- d'un dossier réponse (DR1 à DR4) de 5 pages (numérotées de 38 à 42).

Conseils aux candidats :

Les différentes sous-parties du sujet sont indépendantes.

Un parcours attentif de l'ensemble du document est conseillé avant de composer.

La présentation du code doit respecter les mots clés du langage cible ainsi que l'indentation des structures algorithmiques.

Les réponses doivent être présentées avec clarté, rigueur et concision.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours	Section/option	Epreuve	Matière
EAE	1417A	103	1268

Solutions informatiques de contrôle et de maintenance dans un entrepôt logistique

Présentation

Les compétences de la Division Savoye du groupe Legris Industries concernent l'ingénierie-intégration pour la conception d'architectures de solutions et de services logistiques sur mesure capables de répondre à tous les procédés de la chaîne logistique (Supply Chain).

Ainsi, Savoye propose des solutions innovantes qui concernent :

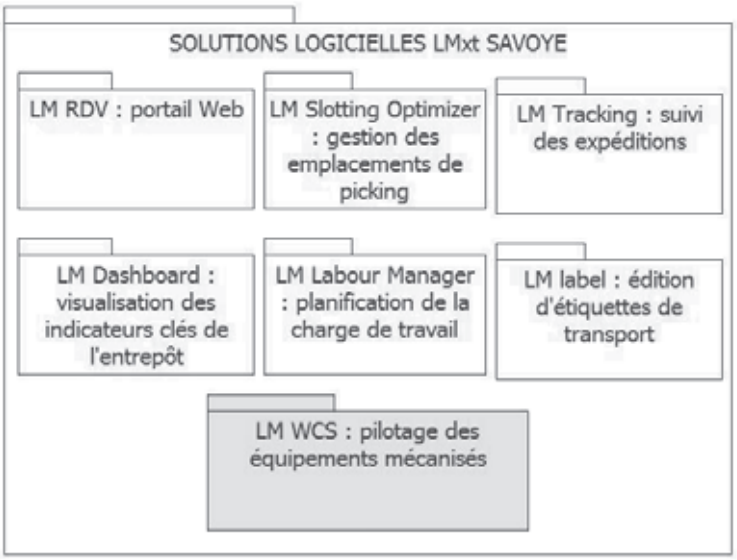
- l'emballage automatisé d'expédition ;
- les systèmes informatiques ;
- la préparation de commandes ;
- le stockage dynamique ;
- le stockage automatisé de palettes.

Ces solutions permettent la planification et la gestion des réseaux de distribution constitués d'entrepôts logistiques. Le transport et la livraison des produits finis constituent le dernier maillon de cette chaîne.

Concernant les systèmes informatiques, Savoye propose une suite logicielle complète et modulaire composée, entre autres, du *Warehouse Managment System* (WMS : LM^{xt}) qui permet de gérer les flux en optimisant les ressources des entrepôts logistiques.

Tournez la page S.V.P.

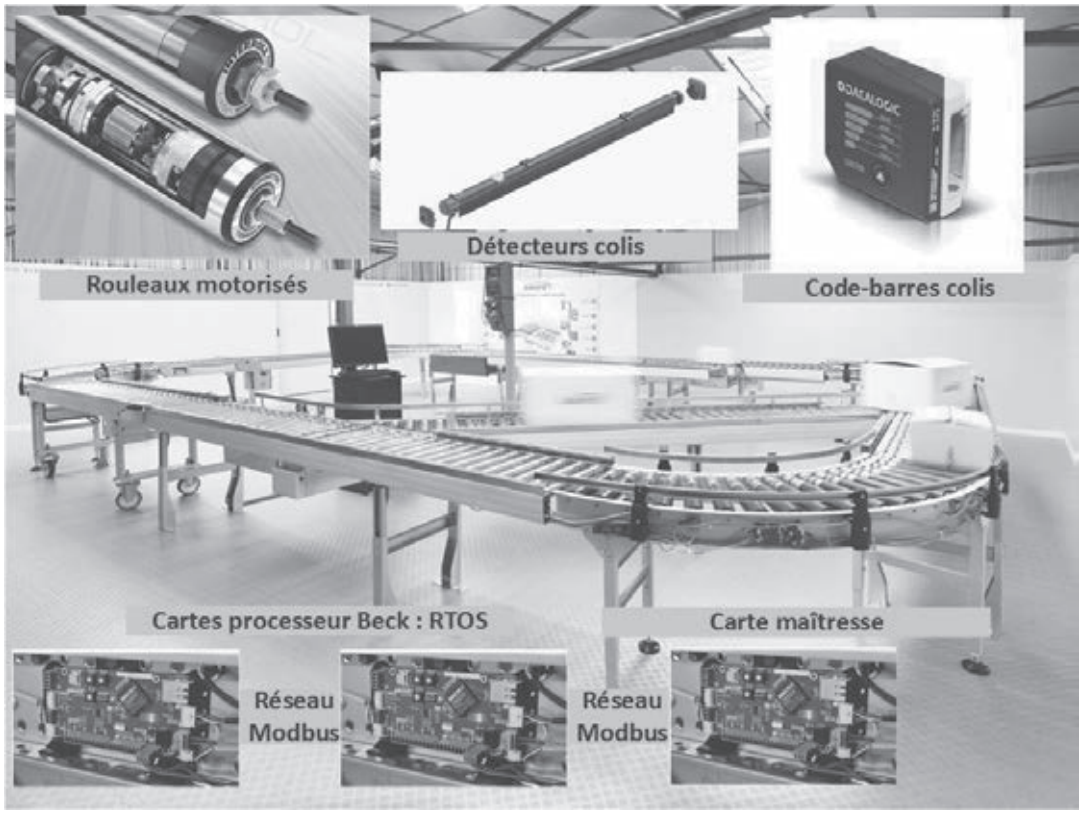
Le diagramme de package représente les différents modules logiciels :



Parmi les équipements mécanisés pilotés par le *WCS* (Warehouse Control System) tels que les machines d'emballage, les navettes et les élévateurs ; les convoyeurs « intelis » *plug&play* permettent un déploiement rapide de l'entrepôt. Deux types de convoyeurs sont proposés :

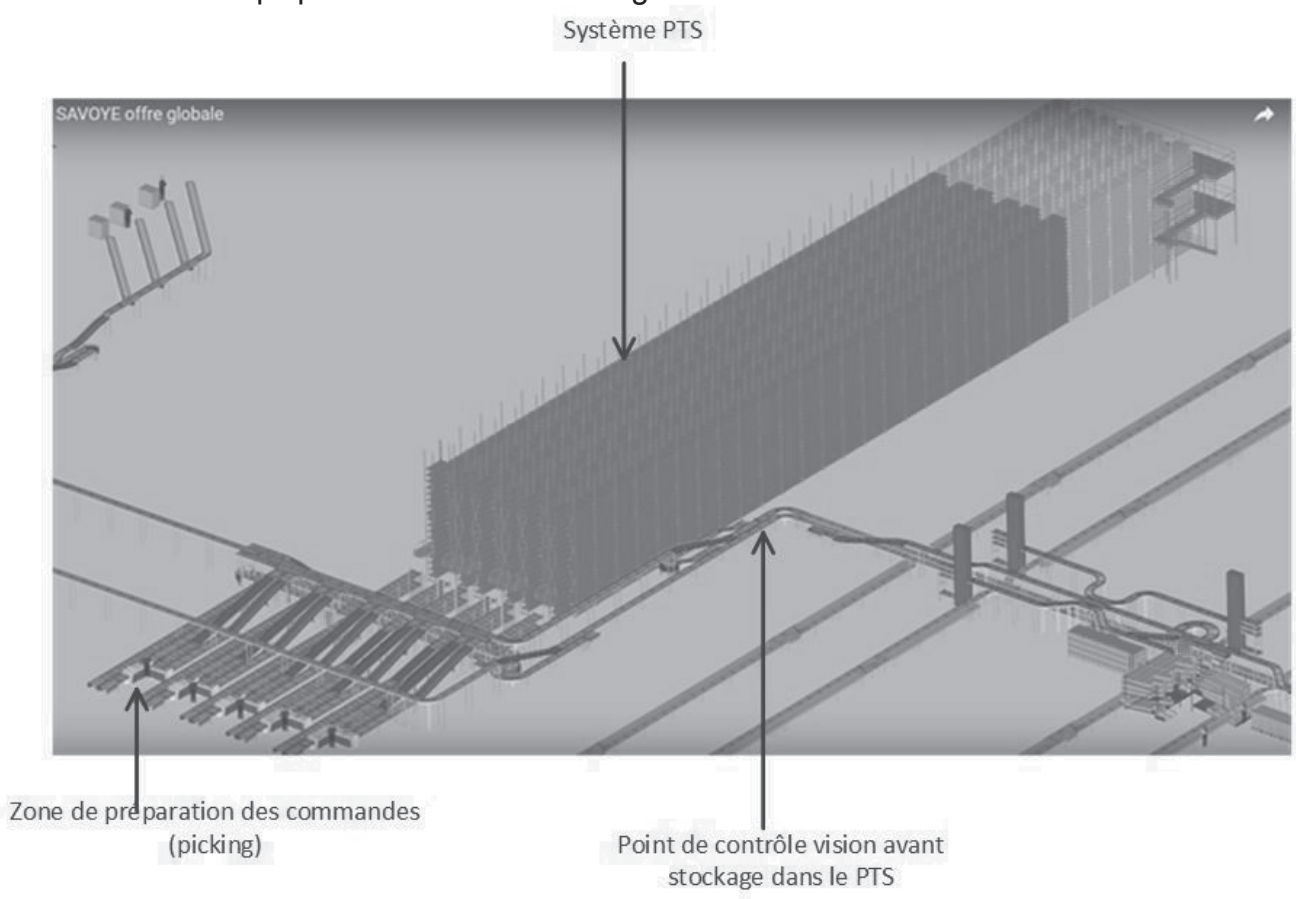
- les TRM, transporteurs à rouleaux motorisés ;
- les TBM, transporteurs à bande motorisée.

Ils sont équipés de capteurs, d'actionneurs, d'éléments intelligents et communicants, leur permettant de s'adapter (vitesse, accumulation, etc.) au volume d'activité.



Le lecteur de code-barres communique avec la carte maîtresse embarquée par l'intermédiaire d'un *hub* ou d'un *switch*. Ce dernier envoie les informations (poids, dimensions, etc.) du colis au système *WCS*.
L'introduction des colis sur la chaîne de convoyeur est manuelle ; les erreurs de positionnement entraînent un ralentissement du processus et elles sont difficiles à diagnostiquer.

Par ailleurs, le système *PTS* « Picking Tray System » est un stockage automatisé qui repose sur un système de navettes et d'ascenseurs qui acheminent les bacs ou les cartons de produits à prélever vers le préparateur de commandes. Pour optimiser le remplissage des bacs ou des cartons, les produits doivent être contrôlés afin de détecter des déformations qui pénaliseraient le stockage.



Expression du besoin

Dans l'objectif d'améliorer la gestion des flux de colis et la maintenance des équipements, le service R&D de Savoye a développé deux systèmes embarqués (voir diagramme des exigences dans le document technique DT1) :

- le premier système permet d'effectuer du contrôle vision de produits en vue de remplacer le système de capteurs dimensionnels et de détecter les déformations de colis pour le système *PTS* ;

- le deuxième système permet d'effectuer de la surveillance vidéo sur les convoyeurs d'introduction afin d'apporter une aide au diagnostic des problèmes.

Tournez la page S.V.P.

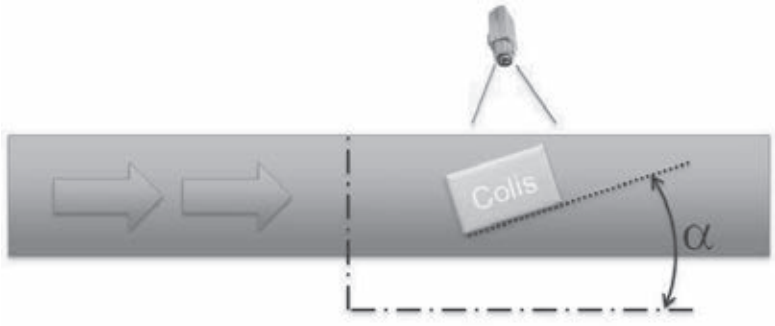
Contrôle vision

Cette fonctionnalité réalise :

- la mesure des dimensions d'un colis ;
- la mesure de l'orientation d'un colis sur le convoyeur ;
- le niveau de déformation d'un colis ;
- la reconnaissance de patrons prédéfinis de colis.

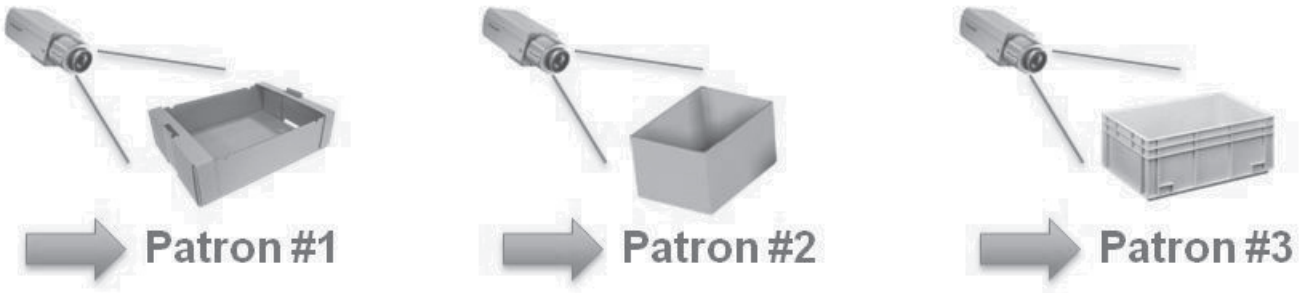
Contrôle 1 : Mesure des dimensions

Contrôle 2 : Mesure de l'orientation par rapport à l'axe du convoyeur



Contrôle 3 : Niveau de déformation du colis

Contrôle 4 : Reconnaissance de patrons

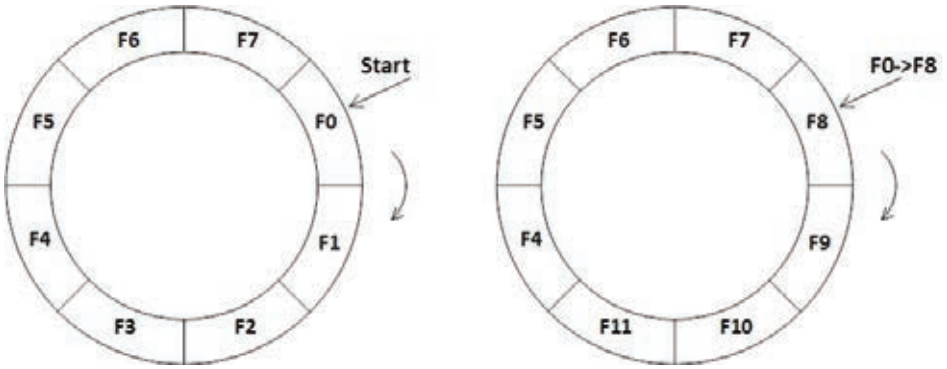


Surveillance vidéo

Actuellement, les opérateurs remplissent les colis de produits liés à une commande. Une fois remplis, les colis sont acheminés vers leur prochaine destination par des convoyeurs. À ce stade, certains défauts peuvent apparaître et sont détectés par les cartes *Beck* maîtresses. Celles-ci envoient alors un message au superviseur pour l'alerter et donner le type d'erreur détectée.

Afin d'améliorer la maintenance, la société SAVOYE souhaite valider une solution permettant de sauvegarder des séquences vidéo à l'apparition d'un défaut.

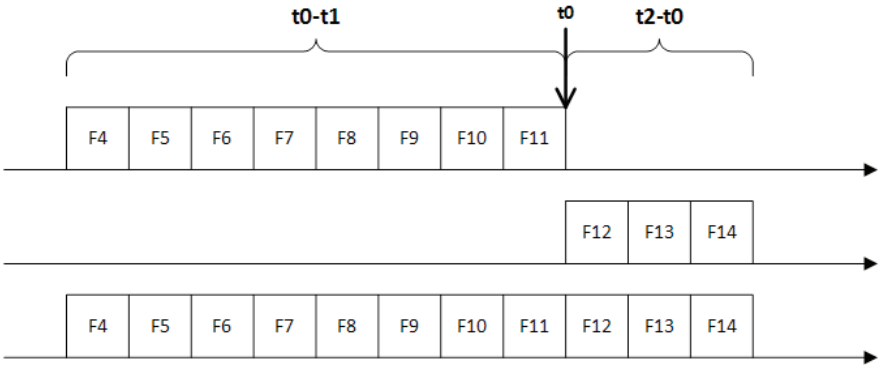
L'application enregistre les images vidéo (frames) dans un buffer circulaire pendant une durée **t1**. Si aucun défaut n'apparaît au bout de cette durée, les images vidéo les plus anciennes (F0, F1, F2, F3) sont supprimées et remplacées par les plus récentes (F8, F9, F10, F11).



Lorsqu'un événement du type défaut est échangé entre le superviseur (WCS) et le convoyeur à l'instant **t0**, l'application enregistre un fichier vidéo composé des images prises entre **t0-t1** et **t0+t2**.

La durée **t2** correspond au temps d'enregistrement des images vidéo après le défaut.

La vidéo enregistrée est alors la concaténation du buffer circulaire et des images vidéo associées à la durée **t2**.



Les fichiers vidéo sont sauvegardés au format « **.h264** ».

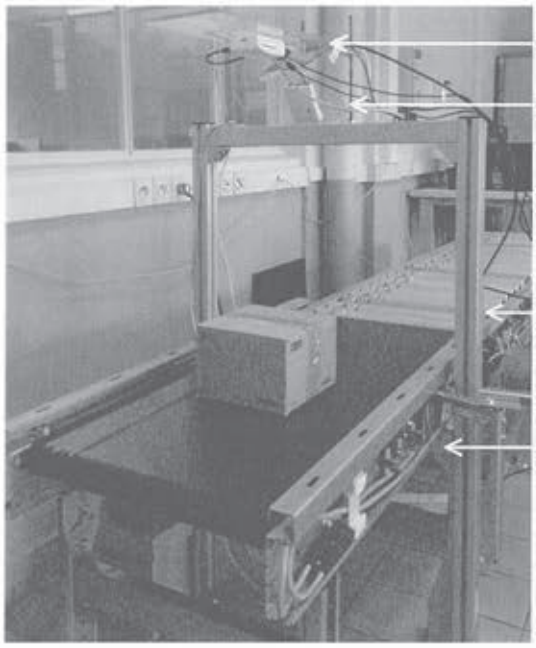
Les chemins des fichiers, les événements (date, type) et les caractéristiques du convoyeur sont sauvegardés dans une base de données relationnelle.

Les événements sont récupérés dans les trames TCP-IP échangées entre le *WCS* et la carte maîtresse du convoyeur.

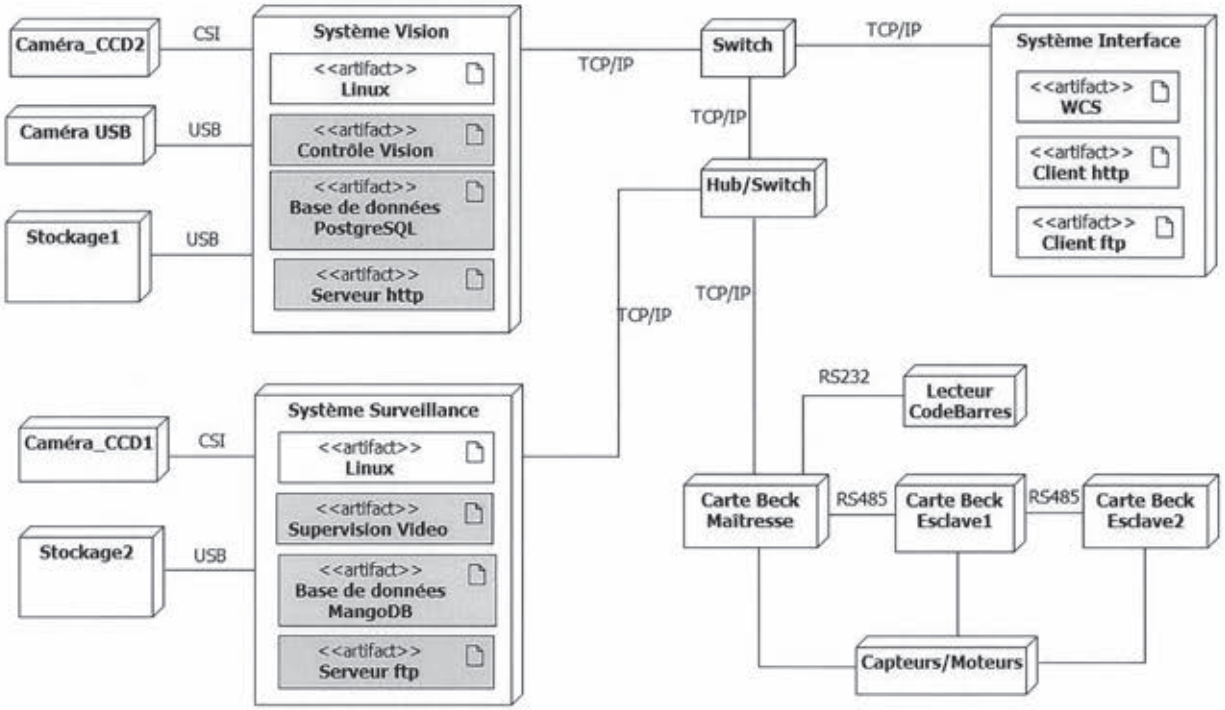
Tournez la page S.V.P.

Prototypage

Afin de valider les solutions de vision et de surveillance, les deux systèmes embarqués sont positionnés sur un convoyeur isolé « intelis ».



Les applications logicielles « Contrôle vision » et « Surveillance vidéo » permettent de traiter les données issues des caméras et du réseau TCP/IP.
La base de données permet de sauvegarder les informations associées aux configurations, aux résultats et aux communications.
Les éléments de stockage permettent d'archiver localement les images et les vidéos.



Dans ce contexte, ce sujet propose d'étudier les étapes de la conception des briques logicielles des deux systèmes embarqués à intégrer sur les convoyeurs.

Partie 1. Étude préliminaire

1.1 Acquisition par caméra CMOS

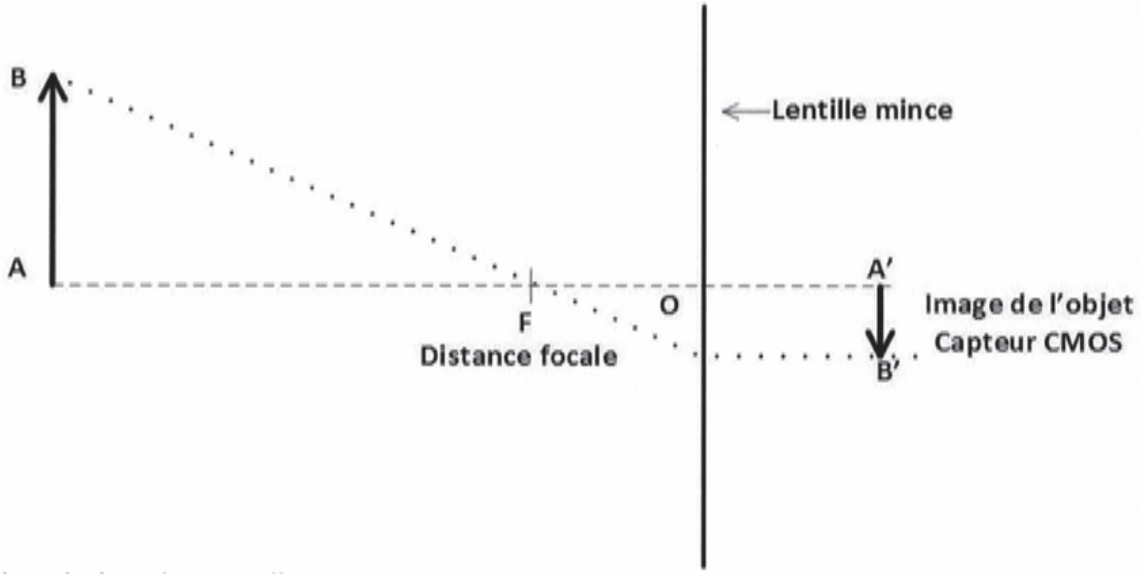
Cette sous-partie propose, dans un premier temps, de spécifier et de justifier le choix des caméras. La caméra permettant de mesurer la hauteur du colis ne sera pas étudiée.
Dans un second temps, il s'agira d'écrire le programme C++ associé à leur configuration et commande.

Le choix des caméras s'est porté sur les produits Omnivision (document technique DT3) compatibles avec les cartes embarquées. Le *firmware* associé est intégré dans le système d'exploitation de ces dernières et l'interface de programmation **Raspicam C++** (document technique DT5) permet de configurer et de piloter ces caméras depuis un programme C++.

La hauteur de positionnement pour la prise d'images, $\mathbf{H_p}$, est de 1000 mm à la verticale du convoyeur et la hauteur maximale d'un colis est donnée par $\mathbf{H_c}$=400 mm (voir document technique DT2).

Q1. À partir du document technique DT3, compléter le tableau (document réponse DR1) qui résume les caractéristiques principales de la caméra.

La figure ci-dessous présente le principe optique simplifié associé à la caméra CMOS :



Avec la relation de grandissement :

$$\frac{\|\overline{A'B'}\|}{\|\overline{AB}\|} = \frac{\|\overline{OA'}\|}{\|\overline{OA}\|} \qquad (1)$$

Et en faisant l'hypothèse : $\|\overline{OA'}\| \cong \|\overline{OF}\|$

Q2. Conformément aux principes optiques exposés ci-dessus, donner les expressions de la longueur et de la largeur maximum d'un colis en fonction de H_p, H_c, L_f, H_s et L_s associées au système d'acquisition de contrôle vision. Calculer la taille maximale de l'objet mesurable par ce système d'acquisition avec H_c=400mm.

Tournez la page S.V.P.

Q3. Conformément à la tolérance de mesure des dimensions (largeur et longueur) d'un colis, déterminer la définition minimale en pixels. Afin d'assurer la précision voulue, il sera nécessaire d'appliquer un coefficient de sécurité de 5.

Q4. En déduire le poids d'une image brute en couleur 8 bits. Comparer avec celui d'une image de définition 2592 x 1944. Conclure sur le choix de la définition de l'image.

La prise d'image est réalisée lorsque le colis est immobile et l'éclairement **E** réfléchi par le colis est compris entre 50 et 100 lx.

Le document technique DT4 présente le principe de fonctionnement d'un pixel CMOS, la fonction de conversion de ce dernier (sensibilité) et la stratégie de lecture et d'exposition d'une image vidéo.
Ci-dessous sont rappelées les définitions respectives du rapport signal sur bruit (SNR, Signal Noise Ratio) et de l'échelle de mesure (DR, Dynamic Range) du capteur à partir des caractéristiques du document constructeur :

$$SNR_{max} = 20 \times \log_{10} \sqrt{FWC} = 20 \times \log_{10} \left(\frac{V_{max}}{V_{noise}} \right) \quad (2)$$

FWC : Full Well Capacity, nombre d'électrons à saturation

$$DR = 20 \times \log_{10} \left(\frac{FWC}{Read\ Noise} \right) \quad (3)$$

Read Noise : bruit total en entrée du CAN exprimé en électrons

Q5. Déterminer l'amplitude maximale de la tension de sortie du pixel en considérant le bruit en entrée du CAN (Convertisseur Analogique-Numérique) égal à celui associé au « dark current ». En déduire l'exposition lumineuse maximale $\mathbf{H_{max}}$ conformément à la sensibilité du capteur.

Q6. En déduire la valeur minimale de l'exposition lumineuse $\mathbf{H_{min}}$ mesurable.

Q7. Déterminer la durée minimale d'exposition $\mathbf{t_{emin}}$ afin de placer le capteur dans ses conditions maximales de sensibilité, c'est-à-dire à saturation dans le cas le plus défavorable.

Q8. D'après le document technique DT4, déterminer le temps d'intégration minimum d'une image à pleine résolution. Conclure en comparant avec la valeur de la question précédente.

Q9. Calculer le bruit associé à la photodiode. Quelle est l'utilité du paramètre DR ? Préciser les principaux critères permettant le choix d'une caméra CMOS pour une application de vision industrielle.

La configuration de la caméra définie dans le document technique DT3 permet d'optimiser le traitement d'image.

Q10. En s'appuyant sur les documents techniques DT3 et DT5, compléter le programme principal (document réponse DR2) permettant de configurer la caméra en utilisant la classe **Raspicam** conformément aux caractéristiques déterminées auparavant.

Q11. Après configuration d'une image de taille 640 x 480 (en mode RGB), la taille de la variable « data » du programme principal (document réponse DR2) est de 921 600 octets. En déduire le principe de stockage mémoire des lignes et des composantes de couleur d'une image dans la variable « data ».

1.2 Stockage des images vidéo

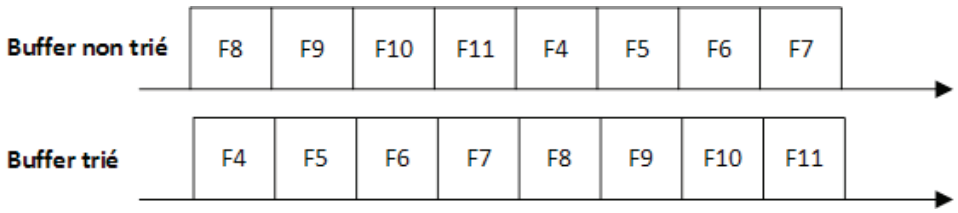
Cette sous-partie propose d'étudier, à l'aide du langage Python, la création et l'utilisation d'un buffer circulaire pour l'application de surveillance vidéo (voir description paragraphe « Surveillance vidéo »).

Le type natif « list » de Python (document technique DT6) est utilisé pour construire le buffer circulaire. Afin de simplifier cette étude, des entiers seront stockés dans ce buffer en lieu et place des images vidéo.

Q12. Compléter le programme (document réponse DR3) permettant de supprimer l'élément le plus ancien stocké dans la liste puis de stocker le nouvel élément lorsque le nombre K d'éléments à stocker dépasse la capacité N du buffer circulaire.

Q13. Compléter la sortie console (document réponse DR3) illustrant le fonctionnement complet du remplissage du buffer circulaire avec N=8 et K=13.

Avant l'enregistrement des éléments du buffer circulaire, il est nécessaire de trier ces derniers dans l'ordre croissant comme l'illustre le dessin ci-dessous.



Q14. Écrire un code Python permettant de trier le buffer par permutation circulaire des éléments en considérant la variable « i » égale au numéro du dernier élément stocké dans le buffer de N éléments.

Q15. Dans le cas du C++, quelle bibliothèque pourrait être exploitée afin de bénéficier de structures avancées comme le type « list » de Python ?

Tournez la page S.V.P.

Partie 2. Contrôle vision

Dans cette partie, il s'agit de valider le choix des traitements à réaliser en justifiant la valeur des paramètres. Seuls les contrôles dimensionnels, d'orientation et de déformation seront étudiés.

Le document technique DT9 présente le diagramme d'activités des traitements permettant de discriminer le colis dans l'image acquise.
Le document technique DT10 présente les images associées aux différents traitements.

Dans le domaine du logiciel libre, la librairie OpenCV, développée initialement par Intel, propose une suite logicielle complète de traitement d'images largement utilisée en vision industrielle. Le document technique DT7 présente un extrait de l'interface de programmation C++.

2.1 Définitions et niveaux de gris

Les instructions suivantes permettent d'initialiser la variable **imSrc** avec les valeurs de l'image couleur acquise par la caméra, dont la largeur et la hauteur sont fixées respectivement à L=640 et H=480. La référence de l'image est située en haut à gauche.

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
using namespace cv;
int main()
{
    //Matrice associée à l'image capturée
    Mat imSrc=imread("colis.ppm");
    return(0);
}
```

Le modèle d'une image est rappelé ci-dessous. Chaque pixel de l'image **imSrc** est associé à l'élément de la matrice $\mathbf{I_s}$ où les indices i,j représentent respectivement le numéro de ligne et de colonne.

$$I_s(i,j) = (R,G,B) \quad (4)$$
$$\text{avec : } i \in [0, H-1], j \in [0, L-1]$$

Pour une image couleur, la valeur d'un pixel est représentée par un tuple de 3 éléments représentant respectivement le rouge (**R**ed), le vert (**G**reen) et le bleu (**B**lue) codés sur 8 bits.

Chaque composante est accessible par l'opérateur *crochet* dans lequel est précisée la valeur de l'index du tuple. Par exemple pour la composante R d'un pixel (i,j) :

$$R(i,j) = I_s(i,j)[0]$$

Le premier traitement consiste à convertir l'image couleur en niveaux de gris. Le choix s'est orienté vers un algorithme permettant d'augmenter le contraste et l'intensité lumineuse de l'image en niveaux de gris.

La première étape consiste à calculer l'intensité totale de chaque composante de couleur. Le calcul des variables T_{green} et T_{blue} est identique à celui de T_{red} en choisissant les composantes respectives G(i,j) et B(i,j) :

$$T_{red} = \sum_{i=0}^{H-1} \sum_{j=0}^{L-1} R(i,j) \quad (5)$$

Le deuxième calcul consiste à évaluer le poids de chaque composante par rapport aux autres. Le calcul des variables W_{green} et W_{blue} est identique à celui de W_{red} :

$$W_{red} = \frac{T_{red}}{T_{red} + T_{green} + T_{blue}} \quad (6)$$

Le niveau de gris de chaque pixel est calculé suivant la somme des composantes de couleur de l'image **Is** pondérées par les coefficients respectifs Wred, Wgreen et Wblue.

Q16. Donner l'expression de l'élément de la matrice $\mathbf{I_g}(i,j)$ associée à l'image en niveaux de gris **imGray** en fonction de $\mathbf{I_s}(i,j)$ et des coefficients Wred, Wgreen et Wblue.

Q17. À l'aide du document technique DT7, écrire l'instruction permettant de créer un objet **imGray** de la classe **Mat** de taille (L=640, H=480) destiné à stocker une image en niveaux de gris. Écrire l'instruction permettant d'affecter au pixel (0,0), la valeur 255.

Q18. Écrire l'instruction permettant d'accéder au pixel (H-1, L-1) par l'intermédiaire de l'attribut public **data** de l'objet **imGray**. Expliquer pourquoi l'instruction **imGray**[H-1][L-1] sera rejetée par le compilateur.

Q19. Écrire la fonction C++ définie par le prototype ci-après et permettant de convertir l'image couleur **imSrc** (définie comme un objet **Mat**) en niveaux de gris conformément à l'algorithme défini par les relations (5) et (6). Le résultat de la conversion sera stocké dans l'objet **imGray**.

```
void grayScale_NEGG(Mat imSrc,Mat imGray)
```

2.2 Détection de contours

Le produit de convolution de l'image **Ig** (de taille L x H) avec un masque **M** d'ordre **Q** est rappelé ci-dessous :

$$(I_g * M)(i,j) = \sum_{l=-q}^{q} \sum_{k=-q}^{q} I_g(i+l, j+k) \times M(l+q, k+q) \quad (7)$$

Q impair : $Q = 2q + 1$

$$i \in [q, H-1-q], j \in [q, L-1-q]$$

Les contours des différents objets de l'image sont obtenus en calculant un gradient discret dans les directions de la largeur et de la hauteur. Cependant, cette opération est très sensible au bruit, une étape de filtrage est alors nécessaire.

Tournez la page S.V.P.

Le détecteur de contour de Canny permet à la fois de réaliser le filtrage par lissage et le gradient discret en utilisant les masques de Sobel respectivement dans les directions horizontale et verticale :

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Q20. Donner les expression des gradients $\mathbf{G_x}(i,j)$ et $\mathbf{G_y}(i,j)$ à partir des masques de Sobel. Donner également la norme du vecteur gradient $\mathbf{G}(i,j)$.

Q21. Ci-après, la figure représente une vue agrandie de l'image **Ig**. Calculer la valeur numérique de **Gx**, **Gy** et **G** pour la position [i=245, j=557]. En déduire les pixels concernés par le contour.

	j=555			
i=244	95	91	79	47
...	92	92	84	48
...	93	93	85	48
...	91	92	89	49

Q22. Préciser les types primitifs du langage C++ possibles permettant d'effectuer les calculs sans troncature. Dans le cas d'une image destination de format 8 bits, quelle précaution à prendre lors de l'affectation des résultats du calcul du gradient ?

Q23. Le document technique DT7 présente la fonction **cv::Canny(...)**. Expliquer le principe de la binarisation de l'image associée au gradient.

Q24. Déterminer les valeurs des seuils haut et bas de la fonction associée au détecteur de Canny, **cv::Canny(...)** en s'appuyant sur les résultats ci-dessus. Écrire l'instruction permettant d'appeler cette fonction avec les arguments adéquats.

2.3 Algorithme d'Otsu

En pratique, la détermination des seuils précédents doit être automatique et l'appel de la fonction **cv::threshold(...)** avant la détection des contours, permet de déterminer le seuil haut (*threshold2*) dans l'histogramme des niveaux de gris. Le seuil bas (*threshold1)* est compris entre 1/2 et 1/3 du seuil haut.

Cette fonction repose sur l'algorithme d'Otsu qui permet de classifier les pixels selon deux classes C_0 et C_1 qui maximisent la **variance inter-classe** définie ci-dessous.

$$\sigma_B^2(k) = \omega_0(k)\omega_1(k)[\mu_1(k) - \mu_0(k)]^2 \qquad (8)$$

Le but de cet algorithme étant de déterminer la valeur de la variable *k* permettant de séparer les deux classes précédentes.

Avec la probabilité des classes C_0 et C_1 :

$$\omega_0(k) = \sum_{i=0}^{k} p_i \qquad \omega_1(k) = \sum_{i=k+1}^{255} p_i$$

Où p_i représente la probabilité d'un niveau de gris i par rapport au nombre N de pixels et n_i le nombre d'occurrences de ce niveau de gris dans l'image.

$$p_i = \frac{n_i}{N}$$

Les valeurs moyennes sur l'ensemble des pixels et sur les pixels de la classe C_0 sont données par :

$$\mu_T = \sum_{i=0}^{255} i \times p_i \qquad \mu(k) = \sum_{i=0}^{k} i \times p_i$$

Et les moyennes des niveaux de gris de la classe C_0 et de la classe C_1 sont données par :

$$\mu_0(k) = \frac{\mu(k)}{\omega_0(k)} \qquad \mu_1(k) = \frac{\mu_T - \mu(k)}{\omega_1(k)}$$

Q25. Déterminer la valeur de la somme des probabilités des classes C_0 et C_1. Exprimer ω_1 en fonction de ω_0.

Q26. Exprimer la variance inter-classe en fonction de ω_0, μ_T et μ.

Q27. En considérant un tableau **histoN[256]** contenant les probabilités p_i, écrire une fonction C++ permettant de calculer la variance inter-classe. Cette fonction prend comme paramètres la valeur du seuil **k** et le tableau **histoN**. Quelles exceptions doivent être prises en compte ?

Q28. En considérant les valeurs de la variance inter-classe stockées dans le tableau **sigmaB[256]**. Écrire le code permettant de déterminer les seuils haut et bas.

Q29. Les figures du document technique DT11 présentent l'histogramme du gradient de Sobel ainsi que la variance inter-classe pour toutes les valeurs possibles du seuil **k**. En déduire les valeurs du seuil haut et bas.

2.4 Dimensions et orientation d'un colis

La fonction **cv::findContours(...)** permet de labelliser les régions délimitées par les contours. Suite à l'exécution de cette fonction, la variable **contours** contient les groupes de points associés à chaque région.

```
std::vector< std::vector<cv::Point> > contours;
findContours(imCanny,contours,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
```

Q30. Expliquer la déclaration de la variable **contours** et donner l'instruction permettant d'indexer la première région.

Tournez la page S.V.P.

La fonction **cv::contourArea(…)** retourne la valeur de la surface délimitée par les points d'une région.
La méthode **contours.size()** retourne le nombre de régions contenues dans l'image.

Q31. Écrire la fonction **findMaxArea(…)** dont le prototype est donné ci-après, permettant de déterminer l'index de la région la plus grande. Conclure sur la stratégie de détection d'un colis.

```
void findMaxArea(std::vector< std::vector<cv::Point> >contours,int &iMaxArea)
```

Soit le code suivant :

```
void drawBoundingRectangle(Mat src,std::vector< std::vector<cv::Point> >
contours,cv::Point2f pt[],int iMaxArea,RotatedRect &rect)
{
    unsigned char j;
    rect=cv::minAreaRect(contours[iMaxArea]);
    rect.points(pt);
    //Dessin du bounding rectangle
    for(j=0;j<4;j++)
    {
        line(src,pt[j],pt[(j+1)%4],Scalar(255,0,0),1);
    }
}
```

Q32. Expliquer ce que réalisent les instructions du code ci-dessus. Expliquer le rôle de l'opérateur modulo.

Dans le document technique DT10, l'image n° 4 présente le résultat final avec la superposition du rectangle englobant *(bounding rectangle)* et du colis.

Q33. Donner les instructions permettant de déterminer les dimensions du colis et son orientation dans l'image. Conclure sur la validité de ce contrôle. Critiquer cette solution dans le cas d'un colis déformé.

2.5 Déformation d'un colis

La fonction **cv::approxPolyDP(…)** du code ci-dessous permet de déterminer les points de la ligne polygonale fermée englobant au plus près la plus grande région de l'image.

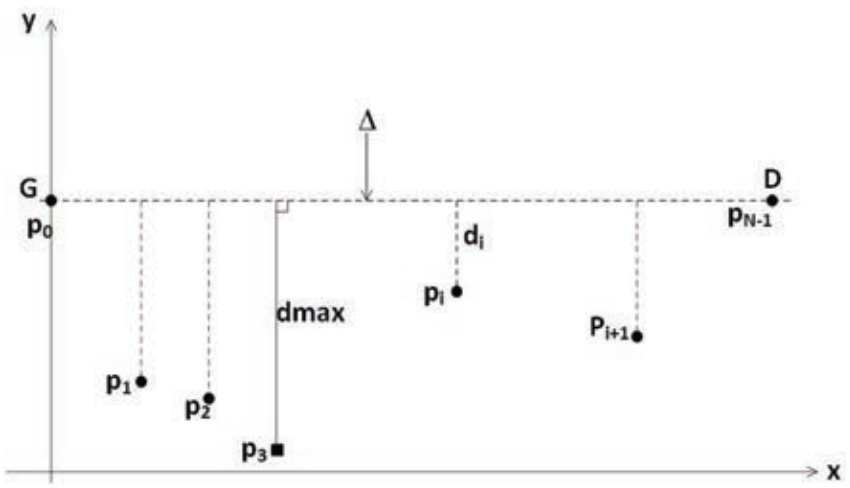
```
std::vector<cv::Point> poly;
approxPolyDP(contours[iMaxArea],poly,5,true);
```

Q34. Expliquer le rôle des paramètres de cette fonction.

La fonction **cv::approxPolyDP(…)** est implémentée selon l'algorithme de Ramer-Peucker-Douglas, dont le traitement élabore la liste de points formant la ligne polygonale simplifiée à partir des points de la région de taille maximale.

Dans cette sous-partie, il s'agit de borner les valeurs d'**epsilon** (document technique DT12).

Soit la figure ci-dessous qui illustre le principe algorithmique à partir d'un tableau de N points (borne G à D) définis par leur abscisse **x** et ordonnée **y** :



La première étape consiste à déterminer la droite Δ passant par le premier et dernier point dont l'équation est rappelée ci-dessous :

$$by_i - ax_i - c = 0 \quad avec\ b = 1$$

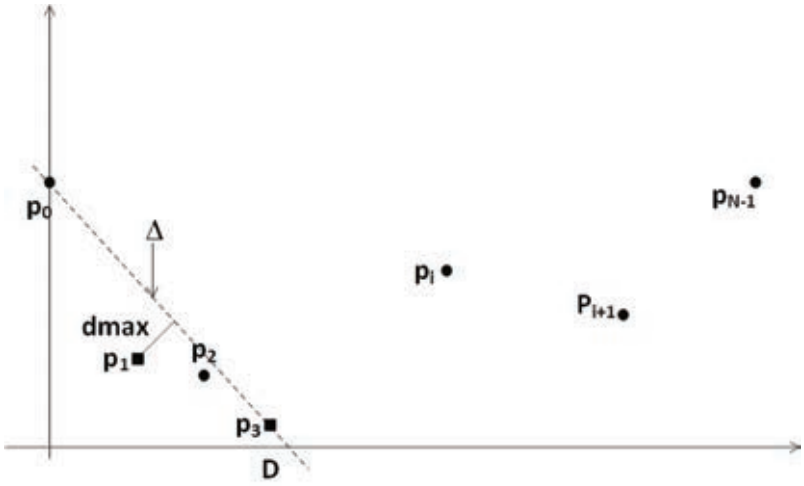
Cette étape est réalisée par la fonction **equaline(p_0,p_{N-1})** qui retourne les valeurs de a et c.

La seconde étape consiste à calculer les distances perpendiculaires $\mathbf{d_i}$ entre cette droite et les points p_i d'après la relation ci-dessous :

$$d_i = \frac{|-ax_i + by_i - c|}{\sqrt{x_i^2 + y_i^2}} \qquad (9)$$

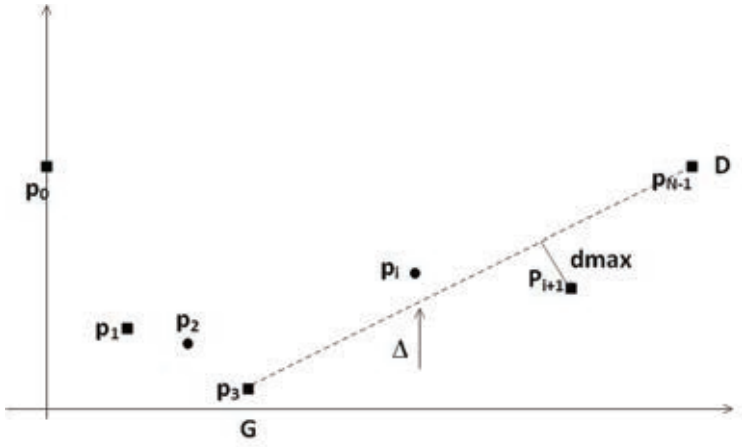
Ce calcul est réalisé par la fonction **perpDist(…)**.

L'algorithme recherche ensuite la distance maximale et stocke le point associé, ici p_3 (représenté par un carré) pris comme pivot.
L'algorithme est récursif et reprend les opérations identiques sur la séquence de points se trouvant à gauche du pivot (p_0, p_1, p_2, p_3) avec les nouvelles valeurs de G et D et la nouvelle droite Δ. Le point p_1 est à son tour stocké.

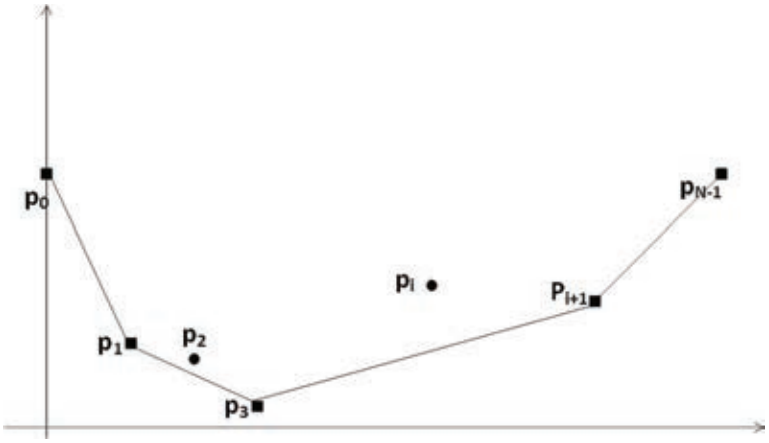


Tournez la page S.V.P.

Lorsqu'il n'y a plus de point à gauche du pivot, l'algorithme s'applique à l'ensemble des points situés à droite avec les nouvelles valeurs de G et D ainsi que la nouvelle droite Δ.
Le point p_{i+1} est alors stocké.



Les points p_0 et p_{N-1} sont également stockés pour former la ligne polygonale simplifiée.



Q35. Soit le tableau de points **tabPts** du document technique DT12, déterminer l'équation de la première droite et la valeur de **dmax** associée à cette dernière.

Q36. En s'appuyant sur le code Python du document technique DT12, donner la condition d'arrêt de la récursivité et la structure de données dans laquelle sont stockés les points retenus pour le tracé de la ligne polygonale simplifiée.

Q37. Déterminer la complexité dans le meilleur des cas et le pire des cas. Justifier votre réponse.

Q38. Quel est le rôle du paramètre **epsilon** ? Préciser l'unité. En considérant les images de test de la fonction **cv::approxPolyDP(…),** préciser la valeur maximale et minimale de ce paramètre.

Q39. Conclure sur une méthode permettant de déterminer le taux global de déformation d'un colis. Quel raffinement est à apporter pour détecter une déformation sur un seul côté du colis ?

Partie 3.Surveillance vidéo

L'objectif de cette partie est de mettre au point une application multitâche permettant de déclencher la sauvegarde d'une vidéo lors de l'apparition d'un défaut par la carte D.

3.1 Analyse réseau de la solution existante

Le diagramme de déploiement (page 7) montre l'architecture matérielle avec tous les équipements connectés via un hub ou un switch (à définir). La partie réseau est toujours configurée de la manière suivante :
Adresse réseau : 10.239.64.0
Masque de réseau : 255.255.192.0

Q40. Donner le nombre de nœuds réseaux exploitables avec ce type d'adressage ainsi que les adresses IP minimale et maximale utilisables. Le masque réseau est-il en concordance avec le type d'adresse IP ? Cela pose-t-il problème ? Justifier votre réponse.

La solution de convoyeurs est intégrée dans un réseau dont les caractéristiques sont différentes à chaque fois (adresse réseau différente, masque différent).

Q41. Proposer une solution permettant de segmenter le réseau sans pour autant modifier les adresses IP du système de convoyeurs et du réseau d'accueil.

3.2 Architecture de la solution surveillance vidéo

Q42. Expliquer la présence du hub dans l'architecture matérielle.

La solution retenue pour analyser le trafic réseau est d'utiliser la librairie **libpcap** (**lib**rary **p**acket **cap**ture). Celle-ci permet d'utiliser une interface réseau au choix (eth0 dans notre cas), de la mettre dans un état *promiscuous*, puis de remonter toutes les trames réseaux « vues » par l'interface.

Le code partiel de l'application est donné dans le document réponse DR4.

Q43. Expliquer ce qu'est le mode *promiscuou*s pour une interface réseau ? Que se passe-t-il si ce mode n'est pas utilisé ?

La fonction **pcap_loop(…)** (voir document technique DT13) de la librairie **libpcap** utilise une fonction de *callback* pour traiter les trames.

Q44. Expliquer ce qu'est une fonction de *callback* et l'avantage que cela représente.

Q45. La fonction **traiterTrame(…)** est passée en paramètre de la fonction **pcap_loop(…)**. En analysant le prototype de ces deux fonctions, expliquer celui de **traiterTrame(…)**.

Le format de la trame liée à la détection d'un défaut est donné dans le document technique DT14. À chaque trame reçue, la fonction **traiterTrame(…)** est appelée.

Q46. Grâce aux commentaires donnés dans le code, proposer un code de la fonction **traiterTrame(…)** permettant de positionner la variable globale **Defaut** à 1 lorsqu'une trame de défaut est détectée.

Tournez la page S.V.P.

Le module de surveillance doit permettre de déclencher l'enregistrement d'une séquence vidéo à l'apparition d'un défaut. Deux traitements en parallèle sont ainsi effectués :
- l'écoute du réseau pour détecter le message de défaut ;
- la gestion d'une fenêtre glissante vidéo afin d'effectuer un enregistrement en cas de défaut.

Q47. Expliquer pourquoi un développement multitâche est ici plus approprié qu'un développement de plusieurs applications fonctionnant en parallèle. Argumenter la réponse conformément au fonctionnement d'un OS intégrant des IPC (Inter Process Communication).

Un thread dédié à la production d'une séquence vidéo a été mis en place.

Q48. Le thread principal contient la primitive bloquante **pthread_join()** permettant au thread principal d'attendre la fin du thread. Expliquer sa présence.

Le cahier des charges impose d'avoir une capture vidéo démarrant avant l'apparition du problème et se terminant un certain temps après (voir page 6). Sans apparition d'un défaut, l'image la plus ancienne est supprimée et une nouvelle image issue du flux vidéo est ajoutée.

Un exemple de code est fourni dans le document technique DT15 pour gérer un flux vidéo avec la bibliothèque OpenCV.
Une classe disponible dans la bibliothèque **Boost** est utilisée (document technique DT16) afin d'implémenter un buffer circulaire destiné à stocker les images vidéo.

Q49. Donner le(s) avantage(s) à utiliser un conteneur de ce type par rapport à un simple tableau.

La documentation de l'objet **circular_buffer** indique que les temps d'insertion, d'accès et de suppression sont en O(1).

Q50. Expliquer ce que cela veut dire et l'intérêt que cela représente.

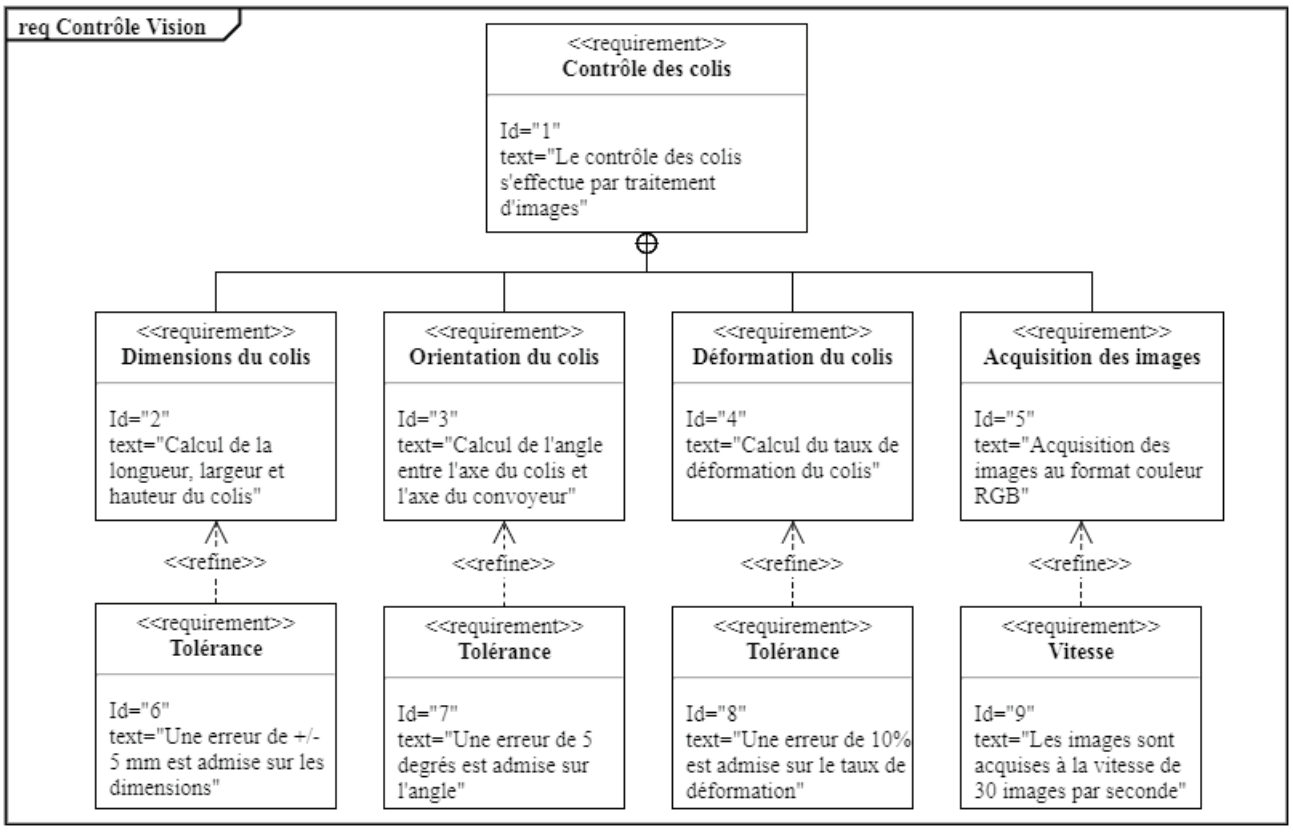
Afin d'économiser de l'espace, la cadence d'acquisition vidéo est de 10 images par seconde. Le temps d'enregistrement vidéo a été fixé à 50 secondes avant le défaut et 10 secondes après.

Q51. En s'appuyant sur les documents techniques DT15 et DT16, compléter la première partie du code (document réponse DR4) permettant d'initialiser une capture vidéo et un buffer circulaire pour le stockage des images vidéo.

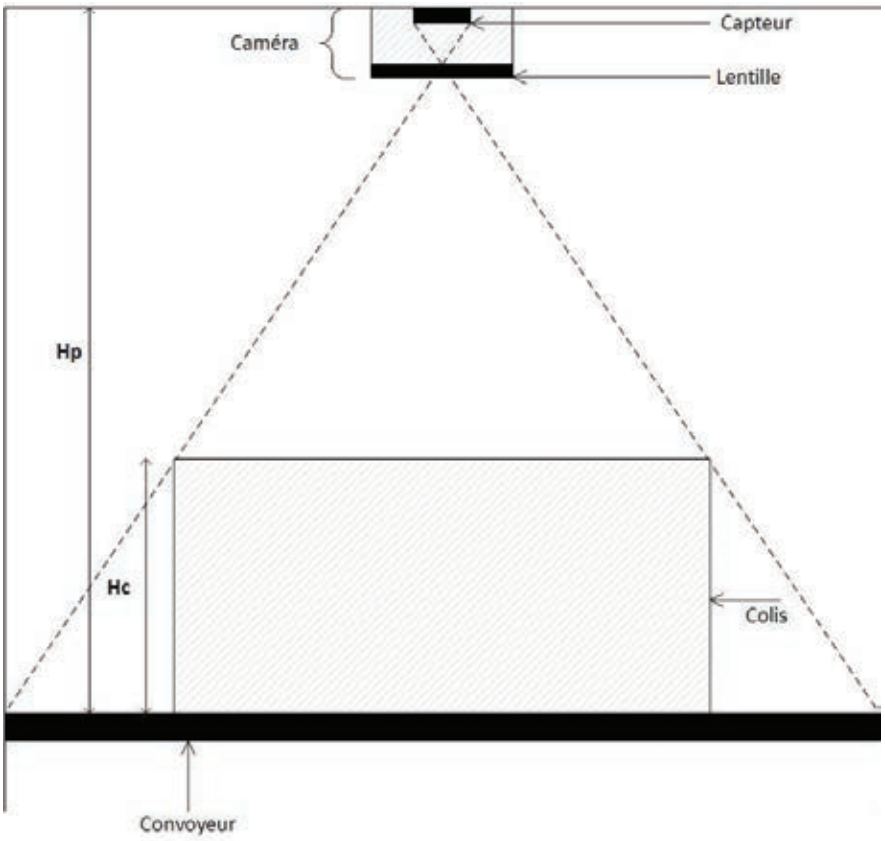
Q52. Compléter la partie du code de la boucle infinie du thread permettant de gérer la fenêtre glissante et la partie de code permettant de créer la vidéo en cas de signalisation d'un défaut.

Q53. Conclure sur l'architecture de la solution de surveillance vidéo en critiquant les parties matérielles et logicielles.

Document technique DT1 : Diagramme des exigences



Document technique DT2 : Système d'acquisition vision



Tournez la page S.V.P.

C

Document technique DT3 : Caméra CMOS Omnivision

- 1.4 µm x 1.4 µm pixel with OmniBSI technology for high performance (high sensitivity, low crosstalk, low noise)
- optical size of 1/4"
- automatic image control functions:
  - automatic exposure control (AEC)
  - automatic white balance (AWB)
- support for output formats: 8-/10-bit raw RGB data
- support for video or snapshot operations
- **active array size:** 2592 x 1944
- **power supply:**
  - **core:** 1.5 V ±5% (internal regulator)
  - **analog:** 2.6 ~ 3.0 V
  - **I/O:** 1.7 ~ 3.0 V
- **temperature range:**
  - **operating:** -30°C to 70°C
  - **stable image:** 0°C to 50°C
- **output formats:** 8-/10-bit raw RGB data
- **lens size:** 1/4"
- **lens chief ray angle:** 24°
- **input clock frequency:** 6 ~ 27 MHz
- **S/N ratio:** 36 dB
- **dynamic range:** 68 dB
- **sensitivity:** 680 mV/(lux-sec)

maximum image transfer rate:
- **QSXGA (2592x1944):** 15 fps
- **1080p:** 30 fps
- **960p:** 45 fps
- **720p:** 60 fps
- **VGA (640x480):** 90 fps
- **QVGA (320x240):** 120 fps

shutter: rolling shutter/global shutter

maximum exposure interval:
1968 x t_{ROW}

pixel size: 1.4 µm x 1.4 µm

well capacity: 4.3 Ke-

dark current: 16 mV/sec @ 60°C

fixed pattern noise:
< 1% of $V_{PEAK-TO-PEAK}$

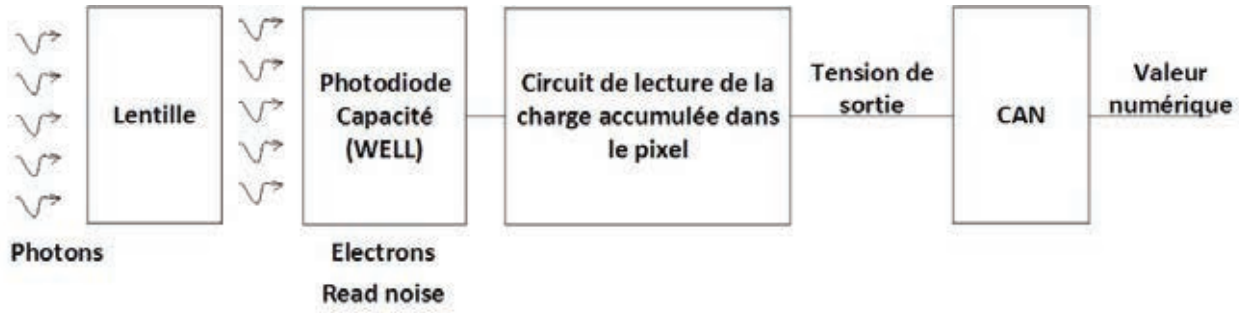
image area: 3673.6 µm x 2738.4 µm

die dimensions: 5520 µm x 4700 µm

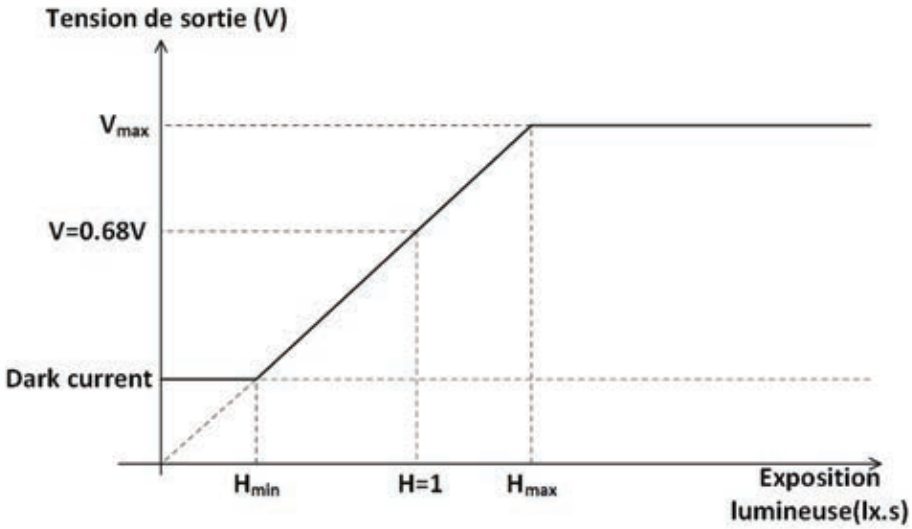
Configuration de la caméra	
Définition de l'image	640 X 480
Format de l'image (RGB ou GRAYSCALE)	RGB
Luminosité	70%
ISO	100
Contraste	100%
Temps d'exposition	Fixed
Vitesse de l'obturateur (shutter)	100%
Codage de l'image	BMP
Contrôle AEC	OFF
Contrôle AWB	ON

Document technique DT4 : Principe d'une caméra CMOS

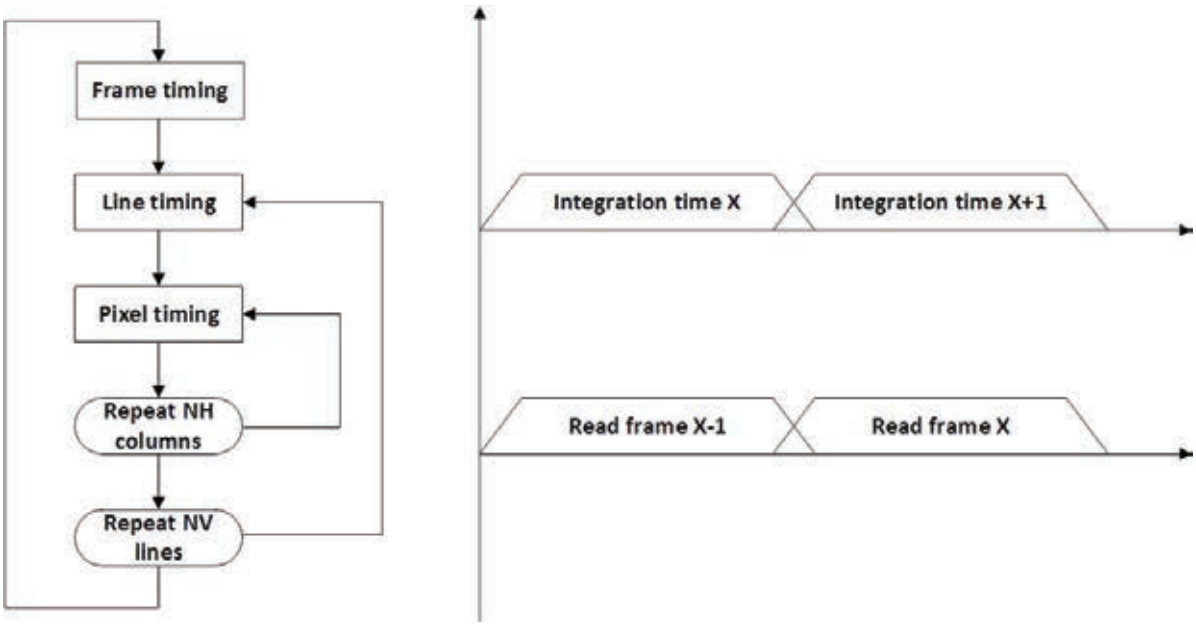
Structure d'un pixel



Fonction de conversion

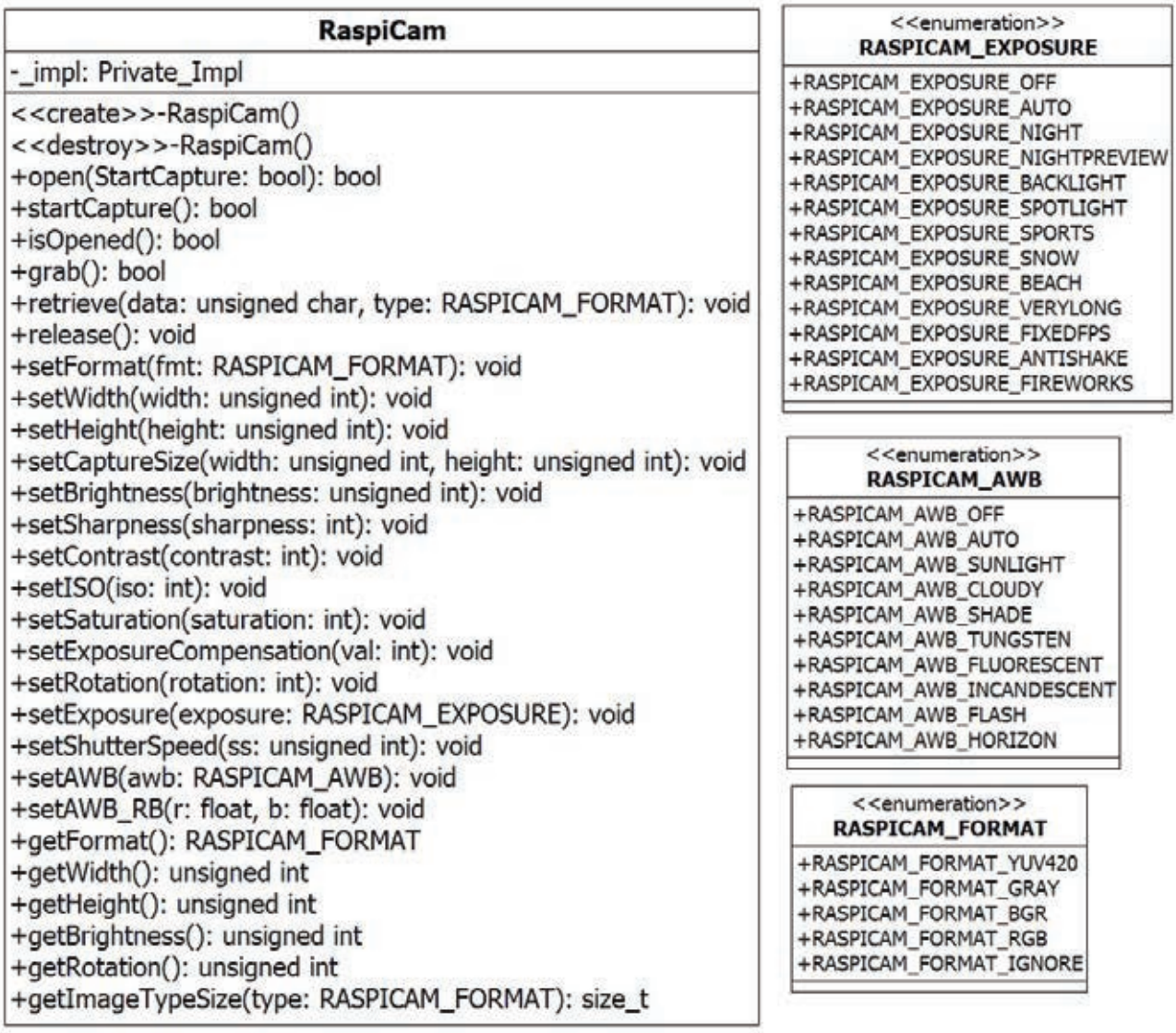


Séquence de lecture d'une image vidéo----Lecture et exposition simultanées



Tournez la page S.V.P.

Document technique DT5 : Extrait du module Raspicam C++



Extrait du fichier Raspicam.cpp

```
#include "raspicam.h"
#include "private/private_impl.h"
namespace raspicam{
    ...
    ...
    void RaspiCam::retrieve(unsigned char *data,RASPICAM_FORMAT type){
        _impl->retrieve(data,type);
    }
    size_t RaspiCam::getImageTypeSize(RASPICAM_FORMAT type)const{return _impl-
>getImageTypeSize(type);}
    void RaspiCam::setShutterSpeed(unsigned int ss){
          _impl->setShutterSpeed(ss);
    }
};
```

Extrait du fichier private_impl.cpp

```
void Private_Impl::setShutterSpeed(unsigned  int shutter){
            if (shutter>330000)
                shutter=330000;
            State.shutterSpeed=shutter;
            if (isOpened())commitShutterSpeed();}
```

Document technique DT6 : Le type « list » de Python

Sequences

These represent finite ordered sets indexed by non-negative numbers. The built-in function len() returns the number of items of a sequence. When the length of a sequence is n, the index set contains the numbers 0, 1… n-1. Item i of sequence a is selected by a[i].

Sequences also support slicing: a[i:j] selects all items with index k such that i <= k < j. When used as an expression, a slice is a sequence of the same type. This implies that the index set is renumbered so that it starts at 0.

Sequences are distinguished according to their mutability. Mutable sequences can be changed after they are created. The subscription and slicing notations can be used as the target of assignment and del (delete) statements.

List

List is an intrinsic mutable sequence types.
The items of a list are arbitrary Python objects. Lists are formed by placing a comma-separated list of expressions in square brackets.
The list data type has some more methods :

list.append(x)
Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x)
Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.

list.clear()
Remove all items from the list. Equivalent to del a[:].
The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the start argument.

list.count(x)
Return the number of times x appears in the list.

list.reverse()
Reverse the elements of the list in place.

Tournez la page S.V.P.

Document technique DT7 : Librairie OpenCV C++

Primitive types	
CV_8UCX	8-bit unsigned integers (0..255), X channels (1-4)
CV_8SCX	8-bit signed integers (0..255), X channels (1-4)
CV_16UCX	16-bit unsigned integers (0..255), X channels (1-4)
CV_16SCX	16-bit signed integers (0..255), X channels (1-4)
CV_32UCX	32-bit unsigned integers (0..255), X channels (1-4)
CV_32SCX	32-bit signed integers (0..255), X channels (1-4)
CV_32FCX	32-bit floating point numbers (0..255), X channels (1-4)
CV_64FCX	64-bit floating point numbers (0..255), X channels (1-4)
uchar	8-bit unsigned integers (0..255)

Mat class

n-dimensional dense array class.

The class **Mat** represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms…
2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on.

The data layout of the array **M** is defined by the array M.step[], so that the address of element $(i_0,...,i_{M.dims-1})$, where $0 \le i_k < M.size[k]$, is computed as:

$$addr(M_{i0,...,i_{M.dims-1}}) = M.data + M.step[0] * i_0 + M.step[1] * i_1 + ... + M.step[M.dims-1] * i_{M.dims-1}$$

In case of a 2-dimensional array, the above formula is reduced to:

$$addr(Mi,j) = M.data + M.step[0] * i + M.step[1] * j$$

Constructors

Mat::Mat()
Mat::Mat(int **rows**, int **cols**, int **type**)
Mat::Mat(Size **size**, int **type**)
Parameters
rows: number of rows in a 2D array.
cols: number of columns in a 2D array.
size: 2D array size: Size(cols, rows) . In the Size() constructor, the number of rows and the number of columns go in the reverse order.
type: array type. Use CV_8UC1... CV_64FC4 to create 1-4 channel matrices, or CV_8UC(n), ..., CV_64FC(n) to create multi-channel (up to CV_CN_MAX channels) matrices.

Destructor

Mat::~Mat()

Assignment operator

Mat& Mat::operator=(const Mat& m)
Parameters
m: assigned, right-hand-side matrix. Matrix assignment is an O(1) operation. This means that no data is copied but the data is shared.

Public attributes

int cols : columns number
int rows : rows number
int dims : matrix dimension (two for a 2D matrix)
uchar *data : pointer to the data (the image is stored row by row as a 1D dynamic array, from the top-left pixel to the bottom-right pixel)

Access to one channel matrix element

template<typename T> T& Mat::at(int **i**) const
template<typename T> const T& Mat::at(int **i**) const
template<typename T> T& Mat::at(int i, int j)
template<typename T> const T& Mat::at(int **i**, int **j**) const
Parameters
i : index along the dimension 0
j : index along the dimension 1
Returns a reference to the specified array element.

The example below initializes a Hilbert matrix:

```
Mat H(100,100,CV_64F);
for(int i = 0; i < H.rows; i++)
    for(int j = 0; j < H.cols; j++)
        H.at<double>(i,j)=1./(i+j+1);
```

Keep in mind that the size identifier used in the **at** operator cannot be chosen at random. It depends on the image from which you are trying to retrieve the data.

If matrix is of type CV_8U then use Mat.at<uchar>(y,x).
If matrix is of type CV_8S then use Mat.at<schar>(y,x).
If matrix is of type CV_16U then use Mat.at<ushort>(y,x).
If matrix is of type CV_32S then use Mat.at<int>(y,x).

Access to three channels matrix element

template<typename T> T& cv::Mat::at(const **Vec**< uchar, **n** > & **idx**)
Parameters
idx: array of Mat::dims indices
Vec< uchar, n >: template class for short numerical vectors with **n** elements of uchar type.

Example:

```
Mat mat2(4,4,CV_8UC3);
unsigned int i,j;
for(i=0;i<4;i++)
{
     for(j=0;j<4;j++)
     {
          mat2.at< Vec<uchar,3> >(i,j)[0]=i+1;//Blue channel
          mat2.at< Vec<uchar,3> >(i,j)[1]=i+j;//Green channel
          mat2.at< Vec<uchar,3> >(i,j)[2]=j+2;//Red channel
      }
}
```

InputArray

This is the proxy class for passing read-only input arrays into OpenCV functions.

It is defined as:

typedef const **_InputArray**& cv::InputArray;

Where **_InputArray** is a class that can be constructed from **Mat**, **Mat_<T>**, **std::vector<T>**, **std::vector<std::vector<T> >** or **std::vector<Mat>.**

When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, vector<T> etc…

Tournez la page S.V.P.

OutputArray

This type is very similar to InputArray except that it is used for input/output and output function parameters.

It is defined as:

```
typedef const  _OutputArray& cv::OutputArray
```

Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too.

here are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators:

```
typedef OutputArray OutputArrayOfArrays;
typedef OutputArray InputOutputArray;
typedef OutputArray InputOutputArrayOfArrays;
```

Mat cv::**imread**(const String & **filename**)

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (**Mat::data**==NULL).

Currently, the following file formats are supported:
ppm,
Windows bitmaps - *.bmp, *.dib (always supported),
JPEG files - *.jpeg, *.jpg, *.jpe,
Portable Network Graphics - *.png.
Parameters
filename: name of file to be loaded

void cv::**Canny**(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=False)

Finds edges in an image using the Canny algorithm.

The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges.

Hysteresis: Canny does use two thresholds (threshold1 and threshold2):

If a pixel gradient is higher than the threshold2, the pixel is accepted as an edge
If a pixel gradient value is below the threshold1, then it is rejected.
If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.
Canny recommended a threshold2:threshold1 ratio between 2:1 and 3:1.

Parameters
image : 8-bit input image.
edges: output edge map; single channels 8-bit image, which has the same size as image .
threshold1: first threshold for the hysteresis procedure.
threshold2: second threshold for the hysteresis procedure.
apertureSize: aperture size for the Sobel operator.
L2gradient: a flag, indicating whether a more accurate $\|L2\| = \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default $\|L1\| = |dI/dx| + |dI/dy|$ is enough (L2gradient=false).

Point class

template<typename **_Tp**> class cv::**Point_**< **_Tp** >

Template class for 2D points specified by its coordinates x and y.

Constructors

Point_ ()

Point_ (_Tp _x, _Tp _y)

Public attributes

_Tp x

_Tp y

For your convenience, the following type aliases are defined:

```
typedef Point_<int> Point2i;
typedef Point2i Point;
typedef Point_<float> Point2f;
typedef Point_<double> Point2d;
```

Example :

```
Point2f a(0.3f, 0.f), b(0.f, 0.4f);
```

void cv::**line**(InputOutputArray **img**, Point **pt1**, Point **pt2**, const Scalar &**color**)

Draws a line segment connecting two points.

The function line draws the line segment between pt1 and pt2 points in the image.

Parameters

img: image

pt1: first point of the line segment.

pt2: second point of the line segment.

color: line color.

void cv::**findContours**(InputOutputArray **image**, OutputArrayOfArrays **contours**, int **mode**, int **method**)

Finds contours in a binary image.

The function retrieves contours from the binary image using the **Suzuki algorithm**. The contours are a useful tool for shape analysis and object detection and recognition.

Parameters

image: source, an 8-bit single-channel image.

contours: detected contours. Each contour is stored as a vector of points.

mode: contour retrieval mode, see cv::RetrievalModes.

method: contour approximation method.

cv::RetrievalModes

RETR_EXTERNAL: retrieves only the extreme outer contours.

RETR_LIST: retrieves all of the contours without establishing any hierarchical relationships.

double cv::**contourArea**(InputArray **contour**, bool **oriented** = false)

Calculates a contour area.

The function computes a contour area. Similarly to moments, the area is computed using the Green formula. Thus, the returned area and the number of non-zero pixels, if you draw the contour using drawContours or fillPoly, can be different. Also, the function will most certainly give a wrong results for contours with self-intersections.

Parameters

contour: input vector of 2D points (contour vertices), stored in std::vector or Mat.

oriented: oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise).

Tournez la page S.V.P.

Example

```
vector<Point> contour;
contour.push_back(Point2f(0, 0));
contour.push_back(Point2f(10, 0));
contour.push_back(Point2f(10, 10));
contour.push_back(Point2f(0, 10));
double area0=contourArea(contour);//area0=10*10
```

RotatedRect cv::**minAreaRect**(InputArray **points**)

Finds a rotated rectangle of the minimum area enclosing the input 2D point set.

The function calculates and returns the minimum-area bounding rectangle (possibly rotated) for a specified point set.

Parameters

points: input vector of 2D points, stored in std::vector<> or Mat depending on the contour orientation (clockwise or counter-clockwise).

RotatedRec class

The class represents rotated (i.e. not up-right) rectangles on a plane.

Each rectangle is specified by the center point (mass center), length of each side (represented by cv::Size2f structure) and the rotation angle in degrees.

Constructors

RotatedRect ()

Public Member Functions

void points (Point2f **pts[]**) const

Public attributes

float angle
Point2f center
Size2f size

template<typename **_Tp**> class cv::**Size_**< **_Tp** >

Template class for specifying the size of an image or rectangle.

The class includes two members called width and height.

Constructors

Size_ ()
Size_ (_Tp _width, _Tp _height)

Public attributes

_Tp height
_Tp width

OpenCV defines the following Size_<> aliases:

```
typedef Size_<int> Size2i;
typedef Size2i Size;
typedef Size_<float> Size2f;
```

void cv::**approxPolyDP**(InputArray **curve**, OutputArray **approxCurve**,double **epsilon**, bool closed)

Approximates a polygonal curve(s) with the specified precision. The functions approxPolyDP approximate a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the **Douglas-Peucker algorithm**.

Parameters

curve: input vector of a 2D point stored in std::vector or Mat
approxCurve: result of the approximation. The type should match the type of the input curve.
epsilon: parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.
closed: if true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.color.

template < class T, class Alloc = allocator<T> > class vector;

Vectors are sequence containers representing arrays that can change in size.

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically.

Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

Constructors

explicit vector (const allocator_type& **alloc** = allocator_type());//Constructs an empty container, with no elements.
explicit vector (size_type n)// Constructs a container with n elements. Each element is a copy of val (if provided).
vector (size_type **n**, const value_type& **val**, const allocator_type& **alloc** = allocator_type());

Parameters

alloc : allocator object. The container keeps and uses an internal copy of this allocator.Member type allocator_type is the internal allocator type used by the container, defined in vector as an alias of its second template parameter (Alloc). If allocator_type is an instantiation of the default allocator (which has no state), this is not relevant.
n: initial container size (i.e., the number of elements in the container at construction). Member type size_type is an unsigned integral type.
val : value to fill the container with. Each of the n elements in the container will be initialized to a copy of this value. Member type value_type is the type of the elements in the container, defined in vector as an alias of its first template parameter (T).

std::vector::operator[ ]

reference operator[] (size_type n);
const_reference operator[] (size_type n) const;
Returns a reference to the element at position n in the vector container.

Parameter

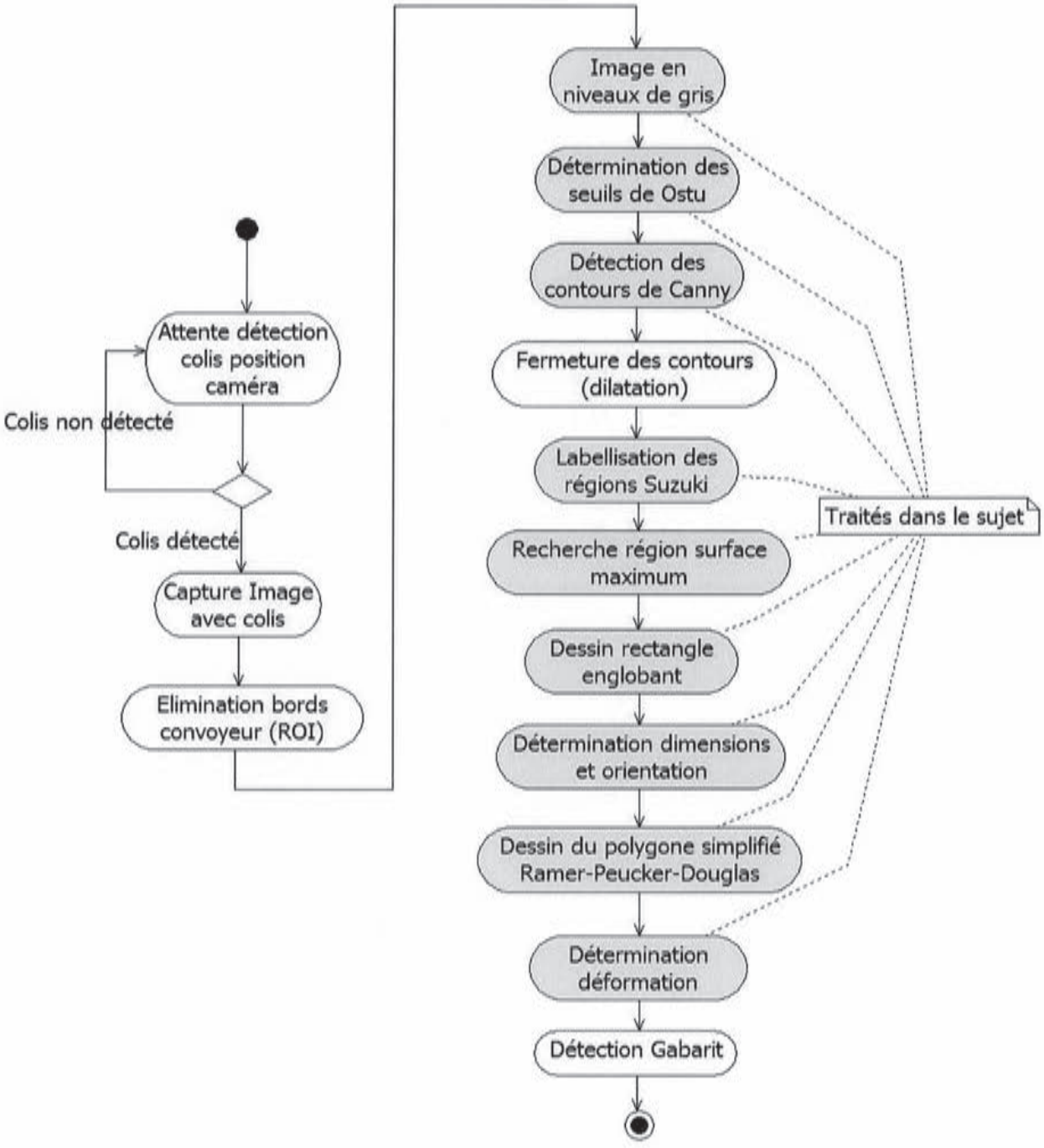
n: Position of an element in the container.
Notice that the first element has a position of 0 (not 1).

Example

```
#include <vector>
int main ()
{
  //constructors used in the same order as described above:
  std::vector<int> first; // empty vector of ints
  std::vector<int> second (4,100); // four ints with value 100
  //assign some values
  second[0]=200;
}
```

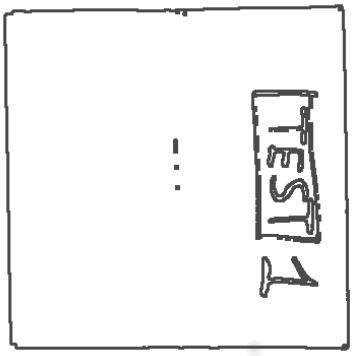

Tournez la page S.V.P.

Document technique DT9 : Principe du contrôle vision



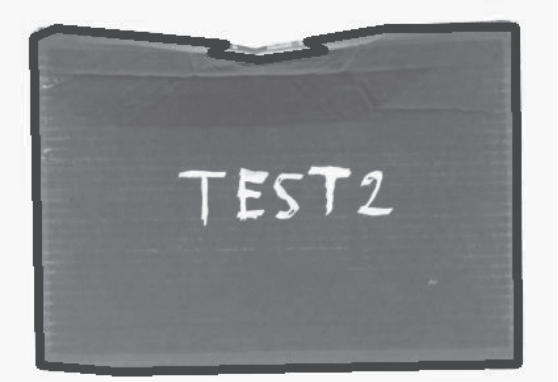
Document technique DT10 : Images test du contrôle vision

1-niveaux de gris	2-détecteur de Canny



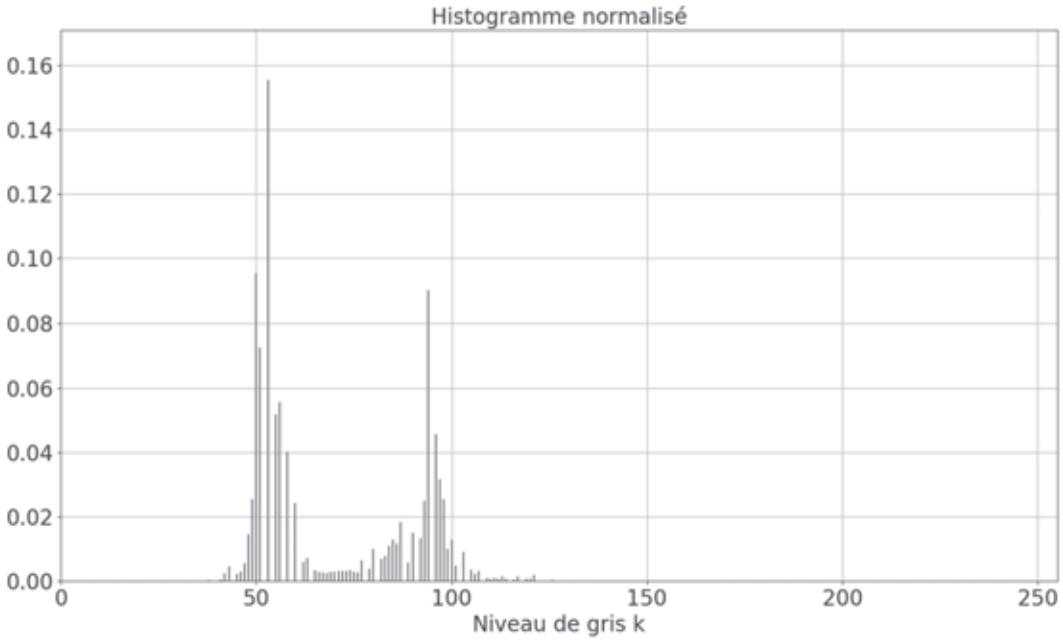
3-contours de Suzuki	4-*bounding* rectangle

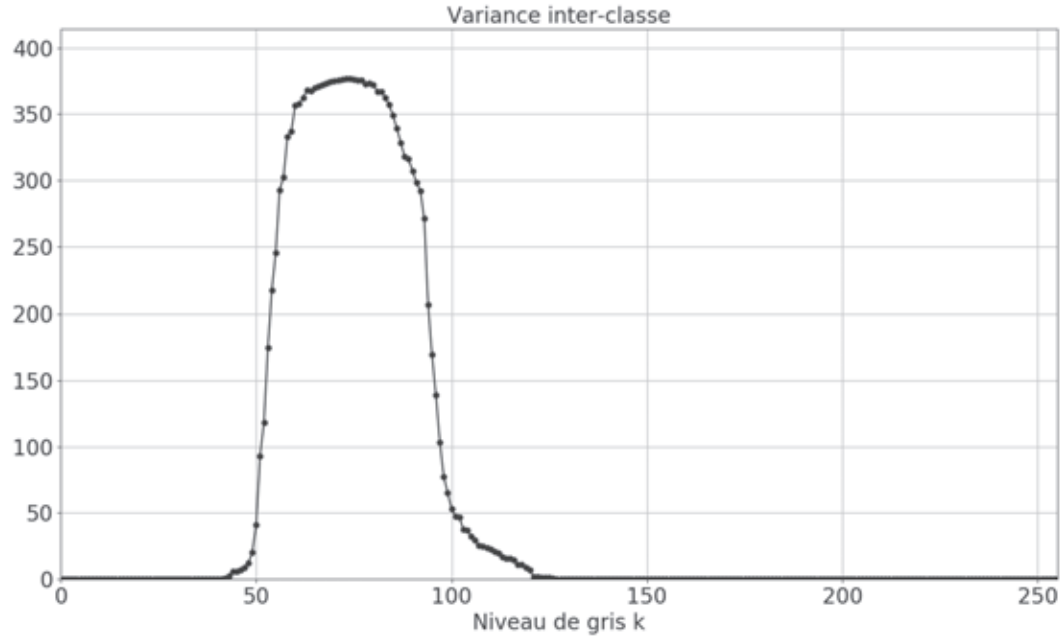
5-colis déformé avec *bounding* rectangle	6-colis déformé avec ligne polygonale



Tournez la page S.V.P.

Document technique DT11 : Méthode de seuillage d'Otsu





Document technique DT12 : Algorithme de Ramer-Peucker-Douglas

Programme Python

```
#points
p0=[0,6.2];p1=[1,3.2];p2=[2,2];p3=[4,1.7];p4=[5,1.5];p5=[9,1.5];p6=[11,1.3]
p7=[12,1];p8=[14,3.8];p9=[16,6]
tabPts=numpy.array([p0,p1,p2,p3,p4,p5,p6,p7,p8,p9]);N=numpy.shape(tabPts)[0]
def ramer_peucker_douglas(tPts,G,D,epsilon,tR):
    """simplifie la ligne polygonale définie par N points
    contenus dans un tableau"""
    index=0
    dmax=0
    N=D-G+1
    for i in range(G+1,N-1):
        d=perpDist(tPts[i],equaline(tPts[G],tPts[D]))
        if(d>dmax):
            dmax=d
            index=i
    if(dmax<=epsilon):
        return(None)
    else:
        tR.append(tPts[index])
        ramer_peucker_douglas(tPts,G,index,epsilon,tR)
        ramer_peucker_douglas(tPts,index,D,epsilon,tR)
G=0;D=N-1
tR=[]
tR.append(tabPts[G])
tR.append(tabPts[D])
ramer_peucker_douglas(tabPts,G,D,0.2,tR)
```

Tests de la fonction cv ::approxyPolyDP(...)

Epsilon=0 N=580	Epsilon=5 N=24
	

Epsilon=10 N=13	Epsilon=40 N=8
	

Tournez la page S.V.P.

Document technique DT13 : Fonction PCAP_LOOP()

```
#include <pcap/pcap.h>

typedef  void (*pcap_handler)(u_char *user, const struct pcap_pkthdr *h,
                              const u_char *bytes);

int pcap_loop(pcap_t *p, int cnt,pcap_handler callback, u_char *user);
```

Description

pcap_loop() processes packets from a live capture or ``savefile" until cnt packets are processed, the end of the ``savefile" is reached when reading from a ``savefile", pcap_breakloop() is called, or an error occurs. It does not return when live packet buffer timeouts occur. A value of -1 or 0 for cnt is equivalent to infinity, so that packets are processed until another ending condition occurs.
callback specifies a pcap_handler routine to be called with three arguments: a u_char pointer which is passed in the user argument to pcap_loop(), a const struct pcap_pkthdr pointer pointing to the packet time stamp and lengths, and a const u_char pointer to the first caplen (as given in the struct pcap_pkthdr a pointer to which is passed to the callback routine) bytes of data from the packet. The struct pcap_pkthdr and the packet data are not to be freed by the callback routine, and are not guaranteed to be valid after the callback routine returns; if the code needs them to be valid after the callback, it must make a copy of them.

Parameters

p: the device handle,
cnt: the number of packets,
callback : the callback function to process packets,
user : the user value is a user-specified value to be passed to the callback function and can be NULL.
h : the struct pcap_pkthdr
bytes: data buffer

struct pcap_pkthdr

```
typedef int bpf_int32;
typedef u_int bpf_u_int32;

struct pcap_pkthdr
{
      struct timeval ts;      /* time stamp */
      uint32_t caplen;  /* length of portion present */
      uint32_t len;           /* length this packet (off wire) */
};
```

Document technique DT14 : Format de la trame de défaut

Libellé	*Valeurs*								*Format*
Nom message	‘ ‘	‘ ‘	‘D’	‘E’	‘F’	‘A’	‘U’	‘T’	*8 caractères*
Longueur des données	10								*1 byte*
Nom carte D ou tronçon	‘ ‘	‘ ‘	‘ ‘	‘A’	‘0’	‘1’			*6 caractères*
Adresse carte	1								*1 byte*
Adresse zone	2								*1 byte*
Numéro du défaut	10								*1 word*
Type du défaut	2								*1 byte*
Statut de l’alarme	1								*1 byte*

Exemple de programme

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
#include "unistd.h"

using namespace std;
using namespace cv;

int main()
{

   VideoCapture vcap(0);
   if(!vcap.isOpened())
   {
      cout << "Error opening video stream or file" << endl;
      return -1;
   }

   int frame_width=vcap.get(CV_CAP_PROP_FRAME_WIDTH);
   int frame_height=vcap.get(CV_CAP_PROP_FRAME_HEIGHT);
   VideoWriter video("out.avi",CV_FOURCC('x','2','6','4'),10,
Size(frame_width,frame_height),true);

   // VideoWriter est la classe permettant d'écrire dans un fichier.
   // Les paramètres du constructeur ci-dessus sont :
   // "out.avi" : chaine de caractères indiquant le nom du fichier à créer.
   // CV_FOURCC('x','2','6','4') : donne le format de fichier.
   // 10 : enregistrement à 10 images sec.
   // Size(frame_width,frame_height) : taille de l'image.
   // true : en couleur.

   for(;;)
   {

       Mat frame;//Objet de base pour stocker une image
       vcap >> frame;//lecture d'une image
       video.write(frame);//écriture de l'image dans le fichier
       usleep(100000);//suspend le thread pendant 100ms.
    }
    return 0;
}
```


Tournez la page S.V.P.

Document technique DT16 : Buffer circulaire avec Boost

Exemple d'utilisation du module Boost

```
#include <boost/circular_buffer.hpp>

 int main()
 {

    //Create a circular buffer with a capacity for 3 integers.
    boost::circular_buffer<int> cb(3);

    //Insert threee elements into the buffer.
    cb.push_back(1);
    cb.push_back(2);
    cb.push_back(3);

    int a=cb[0];//a == 1
    int b=cb[1];//b == 2
    int c=cb[2];//c == 3

    //The buffer is full now, so pushing subsequent
    //elements will overwrite the front-most elements.

    cb.push_back(4);// Overwrite 1 with 4.
    cb.push_back(5);// Overwrite 2 with 5.

    //The buffer now contains 3, 4 and 5.
    a=cb[0];//a == 3
    b=cb[1];//b == 4
    c=cb[2];//c == 5

    //Elements can be popped from either the front or the back.
    cb.pop_back();// 5 is removed.
    cb.pop_front();// 3 is removed.

    //Leaving only one element with value = 4.
    int d = cb[0];// d == 4

    return 0;

}
```

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

Né(e) le : / /

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

DR1 - DR2 - DR3

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR1 : Caractéristiques de la caméra (Q1)

Caractéristiques du capteur CMOS	
Taille d'un pixel du capteur	
Hauteur, Largeur du capteur H_s x L_s	
Angle d'ouverture horizontale de la lentille	53.50 degrés
Angle d'ouverture verticale de la lentille	41.41 degrés
Distance focale L_f	3.6 mm
Capacité de charge à saturation	
Rapport signal sur bruit	
Échelle de mesure	
Plancher de bruit par seconde	
Tension électrique d'un pixel pour 1 lx.s (1 lux.s)	
Nombre d'images par seconde avec une résolution maximale	

Document réponse DR2 : Code Raspicam C++

Début du programme

```
#include <fstream>
#include <iostream>
#include <raspicam/raspicam.h>
using namespace std;
using namespace raspicam;
int main()
{
```

À compléter (Q10)

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Fin du programme

```
    if(!camera.open())
    {
        cerr<<"Error opening camera"<<endl;
        return -1;
    }
    //capture
    camera.grab();
    //allocate memory
    unsigned char *data=new unsigned
char[camera.getImageTypeSize(RASPICAM_FORMAT_RGB)];
    cout<<"image size:"<<camera.getImageTypeSize (RASPICAM_FORMAT_RGB)<<endl;
    //extract the image in rgb format
    camera.retrieve(data,RASPICAM_FORMAT_RGB);//get camera image
    //save
    std::ofstream outFile ("colis.ppm",std::ios::binary);
    outFile<<"P6\n"<<camera.getWidth() <<" "<<camera.getHeight() <<" 255\n";
    outFile.write((char*)data,camera.getImageTypeSize(RASPICAM_FORMAT_RGB));
    //free ressources
    delete data;
    camera.release();
    return(0);
}
```


Tournez la page S.V.P.

Document réponse DR3 : Code Python buffer circulaire

Une liste comme un buffer circulaire

Début du programme

```
#Taille maxi du buffer circulaire
N=8
#Buffer circulaire
bCirc=[]
#K éléments à stocker
K=13
for i in range(K):
```

À compléter (Q12)

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Sortie console de la variable « bCirc »	À compléter (Q13)
[0]	.....
[0, 1]	.....
[0, 1, 2]	.....
[0, 1, 2, 3]	.....
[0, 1, 2, 3, 4]	.....
[0, 1, 2, 3, 4, 5]	.....
[0, 1, 2, 3, 4, 5, 6]	.....
[0, 1, 2, 3, 4, 5, 6, 7]	.....

Code Python : tri du buffer circulaire

À compléter (Q14)

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

Né(e) le : / /

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

DR4

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR4 : Application de surveillance vidéo

Début du programme

```
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <pcap.h>
#include <string.h>
#include <stdio.h>
//Inclusions nécessaires pour la partie vidéo
#include <boost/circular_buffer.hpp>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
//Pour la synchro entre la capture réseau et l'enregistrement vidéo
//La fonction traiterTrame() mettra cette variable à 1 en cas de défaut.
//Le thread vidéo la remet à zéro une fois la séquence vidéo enregistrée.
int Defaut=0;
```

À compléter (Q51)

```
//Fonction traitant chaque trame réseau
void traiterTrame(u_char *useless,const struct pcap_pkthdr *pkthdr,const u_char
*packet)
{
.....
.....
.....
.....
.....
.....
}
```

Fonction ThreadFenetreVideo

```
void* ThreadFenetreVideo(void* param)
{
    //Initialisation pour la capture vidéo
    VideoCapture vcap(0);
    if(!vcap.isOpened())
    {
        cout<<"Error opening video stream or file"<<endl;
        return -1;
    }
    int frame_width=vcap.get(CV_CAP_PROP_FRAME_WIDTH);
    int frame_height=vcap.get(CV_CAP_PROP_FRAME_HEIGHT);
    VideoWriter
video("out.avi",CV_FOURCC('x','2','6','4'),10,Size(frame_width,frame_height),tru
e);
    Mat frame;//Pour stocker les images.
```

À compléter (Q52)

```
    while(1)
    {
        .....
        .....
        .....
```

Suite Fonction ThreadFenetreVideo - À compléter (Q52)

```
            if(Defaut==1)//un défaut est détecté
            {

                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........
                ..........

            }
        }
}
```

Fin du programme

```
int main()
{
      //Mise en place de la gestion de la vidéo
      pthread_t idThreadFenetreVideo;
      if (pthread_create(&idThreadFenetreVideo, NULL, ThreadFenetreVideo, NULL)
!=0)
      {
            printf(« Problème à la création du thread vidéo… <EXIT> ») ;
            exit(-1) ;
      }
      ////////////////////////////////////////////////
      //Mise en place de la capture des trames réseaux.
      ////////////////////////////////////////////////

      //Initialisation pour la capture réseau
      pcap_t *handle;/* Session handle */
      char *dev="eth0";/* Device to sniff on */
      char errbuf[PCAP_ERRBUF_SIZE]; /* Error string */
      struct bpf_program fp; /* The compiled filter expression */
      char filter_exp[]="port 5555";/* The filter expression */
      bpf_u_int32 mask; /* The netmask of our sniffing device */
      bpf_u_int32 net; /* The IP of our sniffing device */

      //une partie des fonctions d'initialisation de la capture réseau n'a pas
été représentée ici pour des raisons de clarté.

      //fonction permettant d'installer le traitement des trames reçues
      pcap_loop(handle,0,traiterTrame,NULL);
      pthread_join(idThreadFenetreVideo, NULL);
      printf("Fin du programme principal\n");

      return 0;
}
```