Je propose de créer un nouveau projet qui factorise les compétence **d'OpenCV** en relation avec la bibliothèque **GTK**.

__OpenCV__ nous permet de visualiser les images avec la fonction __imshow()__. Toutefois, cette visualisation est très rudimentaire. Je préfère réaliser une IHM plus sophistiqué à l'aide de la librairie __GTK__.

Même si cela paraît évident à priori de fusionner ces deux technologies, les démarches sont totalement différentes, notamment pour la capture vidéo. Dans le cas __d'OpenCV__ nous pouvons réaliser une boucle de capture d'images alors que __GTK__ interdit ce type de fonctionnement.

En effet, une IHM ne fonctionne que sous la forme d'événements qui doivent être systématiquement disponibles à tout instant afin de pouvoir être pris en compte directement.

Heureusement, nous pouvons proposer un fonctionnement connexe à la gestion événementielle en proposant une __tâche de fond__ qui est traité en parallèle du fonctionnement normal (__intégré dans la boucle de capture d'événements__).

Installation de l'API OpenCV

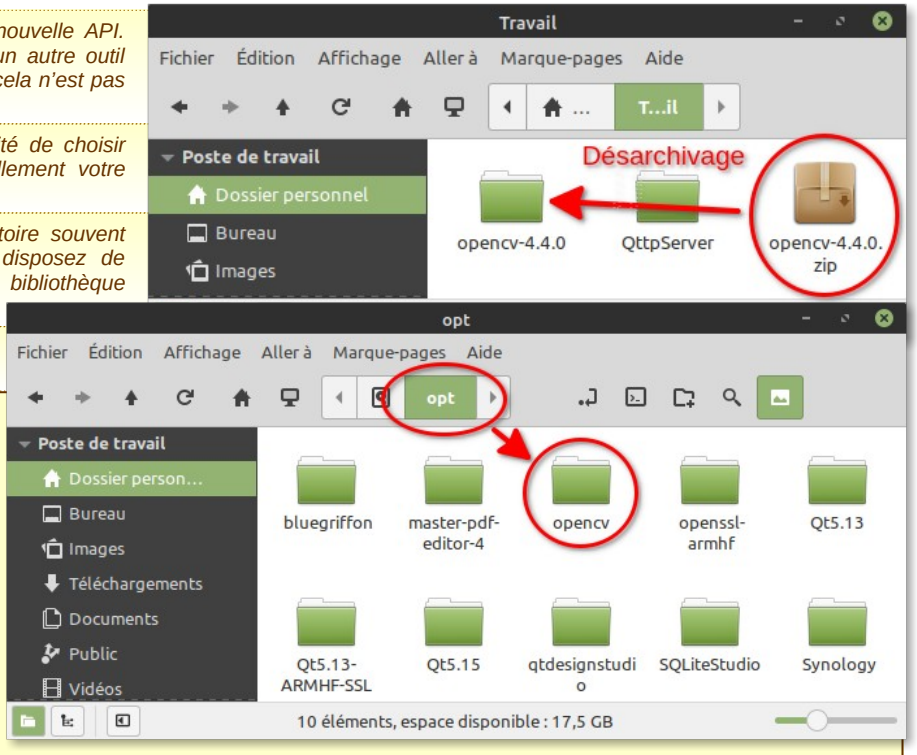
Tout d'abord nous devons installer l'API **OpenCV**. Pour cela, nous devons récupérer les sources de l'API afin de les compiler après coup. Après avoir désarchivé le paquetage en format « **zip** », placez-vous dans le répertoire de base, créer un nouveau répertoire temporaire que vous nommerez par exemple « **release** ».

Vous pouvez dès lors compiler et installer votre nouvelle API. Mais attention, pour cela vous devez passer par un autre outil spécifique « __CMAKE__ », que vous devez installer si cela n'est pas déjà fait.

Grâce cet outil « __CMAKE__ » vous avez la possibilité de choisir votre mode de compilation et où se situera réellement votre librairie.

L'idéal est de placer cette librairie dans un répertoire souvent utilisé « __/opt__ » par exemple. Du coup, si vous disposez de plusieurs comptes, vous trouverez toujours la bibliothèque « __OpenCV__ » au même emplacement.

Voici ci-dessous les commandes à suivre pour l'installation complète :




```
~$ sudo apt install cmake


~$ cd opencv-4.4.0/
~/opencv-4.4.0$ mkdir release
~/opencv-4.4.0$ cd release/


~/opencv-4.4.0/release$ cmake -D
CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/opt/opencv ..
// N'oubliez pas les deux points.
~$ make
~$ sudo make install
```

Une fois que la bibliothèque __OpenCV__ est définitivement compilée et installée, nous la retrouvons bien dans le répertoire « __/opt__ ». Pour tous les projets que nous ferons ultérieurement, il est important de bien noter sa localisation.

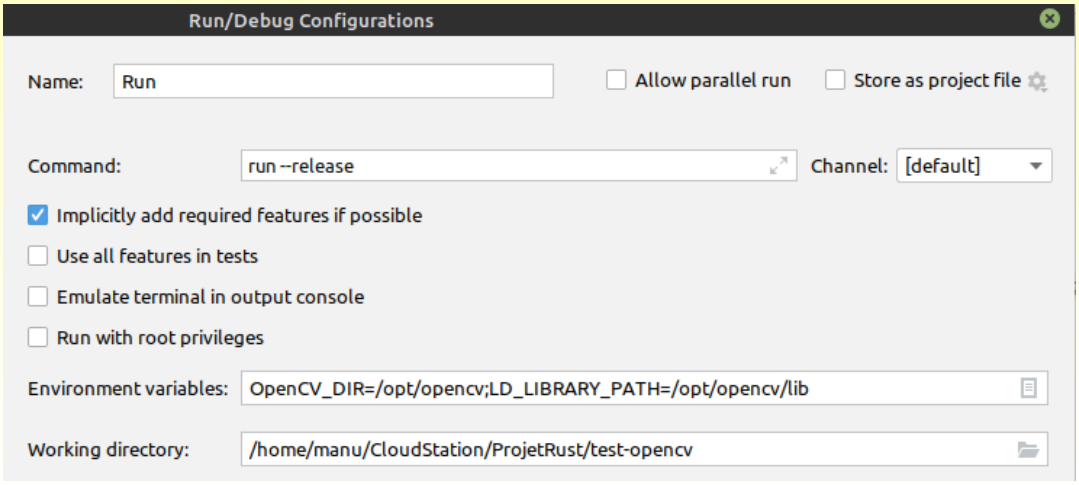
Finalisation de l'installation pour réaliser des applications avec Rust

Grâce aux installations précédentes, nous possédons maintenant la librairie **OpenCV** parfaitement opérationnelle. Toutefois, vu que nous sommes passé par « **make** » et « **make install** », l'API a été construire dans le langage **C++**. Afin de pouvoir exploiter les compétences de cette bibliothèque, il est nécessaire d'installer des outils supplémentaires qui permettront de transcrire les programmes **Rust** vers les modules **C++**.

Les deux outils à installer s'appellent __clang__ et __libclang-dev__ qui sont des compilateurs __C++__, ce qui veut dire que les programmes que nous réaliserons en relation avec __OpenCV__ seront en réalité transcrit en langage __C++__.

```
~$ sudo apt install clang libclang-dev
```

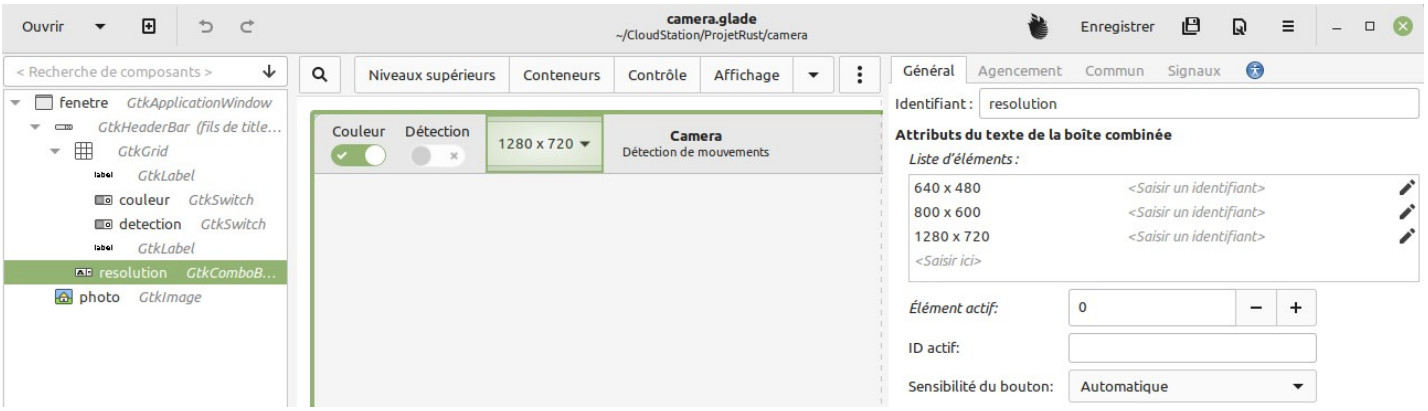
*Un dernier réglage à effectuer pour produire des programmes **Rust** avec analyses d'images consiste à spécifier où se situe exactement la librairie **OpenCV** à l'aide de deux variables d'environnement **OpenCV_DIR** et **LD_LIBRARY_PATH**.*



*Nous allons maintenant pouvoir réaliser nos premiers programmes en **Rust**. Ils seront identiques à ceux que j'ai déjà proposé dans le langage **C++**. Vous remarquerez d'ailleurs que le nom des fonctions et des classes utilisées sont exactement les mêmes.*

RÉALISATION DE L'IHM

L'application nous permet de visualiser la vidéo suivant trois formats **640x480**, **800x600** et **1280x720**. Nous pouvons choisir le mode couleur ou la version en noir et blanc. Enfin, il est possible d'activer la détection des mouvements ou pas.



Cargo.toml

```
[package]
name = "camera"
version = "0.1.0"
edition = "2021"

[dependencies]
gtk = "0.15.5"
opencv = "0.63.1"
```

main.rs

```
use gtk::*;
use gtk::prelude::*;
use gdk_pixbuf::*;
use std::rc::Rc;
use std::cell::Cell;
use gtk::glib::Bytes;
use opencv::{core::*, video::*, videoio::*, imgproc::*, Result};
use opencv::types::VectorOfMat;

fn main() -> Result<()> {
  gtk::init().unwrap();
  let vue = Builder::from_string(include_str!("../camera.glade"));
  let fenetre: Window = vue.object("fenetre").unwrap();
  let ihm = IHM::placer(vue);
  let vue = ihm.clone();
  fenetre.connect_destroy(move |_| {
    vue.continuer.set(false);
```

```
    gtk::main_quit() }
  );
  fenetre.show();
  ihm.changer_resolution();
  ihm.capture_video()?;
  gtk::main();
  Ok(())
}

#[derive(Debug, Clone)]
struct IHM {
  image: Image,
  couleur: Switch,
  detection: Switch,
  resolution: ComboBoxText,
  continuer: Rc<Cell<bool>>,
  changement: Rc<Cell<bool>>
}

impl IHM {
  fn placer(vue: Builder) -> Self {
    IHM {
      image: vue.object("photo").unwrap(),
      couleur: vue.object("couleur").unwrap(),
      detection: vue.object("detection").unwrap(),
      resolution: vue.object("resolution").unwrap(),
      continuer: Rc::new(Cell::new(true)),
      changement: Rc::new(Cell::new(false))
    }
  }

  fn changer_resolution(&self) {
    let ihm = self.clone();
    self.resolution.connect_changed(move |_| { ihm.changement.set(true);  });
  }

  fn changer_image(&self, image: &Mat) -> Result<()> {
    let octets = image.data_bytes()?;
    let vecteur = octets.to_vec();
    let bytes = Bytes::from(&vecteur);

    let (largeur, hauteur) = match self.resolution.active().unwrap() {
      0 => (640, 480),
      1 => (800, 600),
      2 | _ => (1280, 720)
    };

    let image = Pixbuf::from_bytes(&bytes, Colorspace::Rgb, false, 8, largeur, hauteur, largeur*3);
    self.image.set_from_pixbuf(Some(&image));
    Ok(())
  }

  fn capture_video(&self) -> Result<()> {
    let ihm = self.clone();
    let mut camera = VideoCapture::new(0, CAP_ANY)?;
    camera.set(CAP_PROP_FRAME_WIDTH, 640.)?;
    camera.set(CAP_PROP_FRAME_HEIGHT, 480.)?;
    let mut fond = create_background_subtractor_mog2(500, 16., true)?;
    let point = Point::new(-1, -1);
    let brosse = get_structuring_element(MORPH_RECT, Size::new(15, 15), point)?;
    let scalaire = Scalar::all(1.);
    let (mut image, mut rgb, mut gris) = (Mat::default(), Mat::default(), Mat::default());
    let (mut erosion, mut masque) = (Mat::default(), Mat::default());
    let mut contours = VectorOfMat::default();

    glib::idle_add_local(move || {
      if ihm.changement.get() {

        let (largeur, hauteur) = match ihm.resolution.active().unwrap() {
          0 => (640, 480),
          1 => (800, 600),
          2 | _ => (1280, 720)
        };

        camera.set(CAP_PROP_FRAME_WIDTH, largeur as f64).unwrap();
        camera.set(CAP_PROP_FRAME_HEIGHT, hauteur as f64).unwrap();
        ihm.changement.set(false);
      }
```

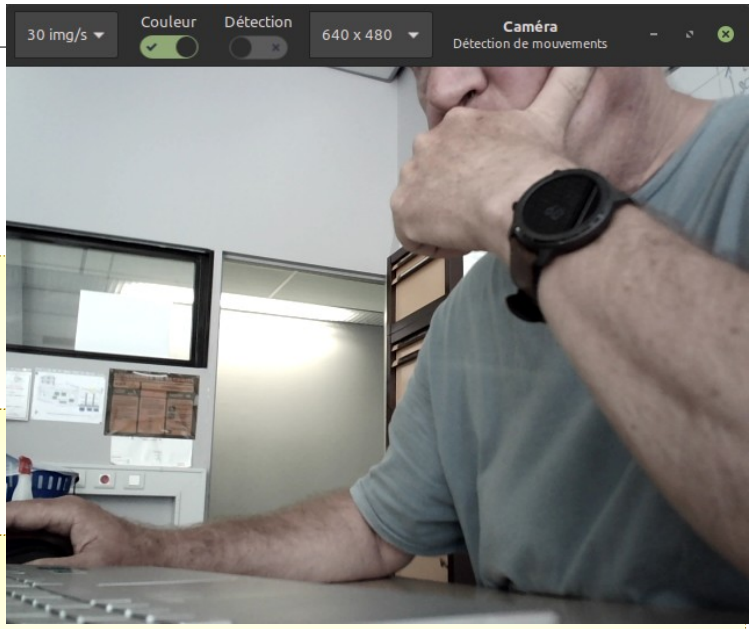
```
    camera.read(&mut image).unwrap();
    if ihm.couleur.is_active() {
      cvt_color(&image, &mut rgb, COLOR_BGR2RGB, 0).unwrap();
    }
    else {
      cvt_color(&image, &mut gris, COLOR_BGR2GRAY, 0).unwrap();
      cvt_color(&gris, &mut rgb, COLOR_BGR2RGB, 0).unwrap();
    }
    if ihm.detection.is_active() {
      BackgroundSubtractorMOG2::apply(&mut fond, &image, &mut masque, -1.).unwrap();
      erode(&masque, &mut erosion, &brosse, point, 1,BORDER_DEFAULT, scalaire).unwrap();
      dilate(&erosion, &mut masque, &brosse, point, 3,BORDER_DEFAULT, scalaire).unwrap();
      find_contours(&masque, &mut contours, RETR_TREE, CHAIN_APPROX_SIMPLE, Point::new(0, 0)).unwrap();
      for contour in &contours {
        let zone = bounding_rect(&contour).unwrap();
        rectangle(&mut rgb, zone, Scalar::new(255., 0., 255., 0.), 3, FILLED, 0).unwrap();
      }
    }
    ihm.changer_image(&rgb).unwrap();
    Continue(ihm.continuer.get())
  });
  Ok(())
 }
}
```

*La grande particularité de ce code est l'utilisation de la fonction **idle_add_local()** qui permet, comme son nom l'indique, de réaliser du traitement en tâche de fond. Le contenu de cette fonction, tant qu'elle est active (en mode continu), est réalisé à la fin de chaque boucle de la fonction principale.*

*Attention, vous ne devez en aucun cas prévoir une boucle interne à cette fonction, sinon cela bloque tout le système et notamment la prise en compte de la gestion événementielle. Comme précisé précédemment, le contenu de cette fonction est sollicité à chaque boucle de la fonction principale **main()** jusqu'à la clôture complète de l'application principale.*

*Nous obtenons finalement un fonctionnement en boucle au travers de cette fonction **idle_add_local()**, et c'est à ce niveau là que nous mettons en œuvre la récupération de chacune des images de la vidéo avec les différents traitements désirés.*

SERVICE VIDÉO – VISUALISATION À DISTANCE



Je vous propose de poursuivre notre étude en séparant notre application précédente en deux entités distinctes, afin d'avoir la gestion de la vidéo sur un système numérique et la visualisation sur un autre. La gestion de la vidéo se fait alors au travers d'un service monté sur une **Raspberry** par exemple. La consultation de la vidéo se fait donc à distance au travers du réseau local ou même par Internet.

La particularité de cette approche, c'est que le flux vidéo se fait au travers du réseau. Dans ce contexte, vu la quantité d'informations transitées, il me paraît judicieux de limiter le nombre d'applications clientes connectées à une seule entité. La qualité de la visualisation sera alors optimale.

*Par rapport au projet précédent, je propose de rajouter le choix de la fréquence des images afin de bien maîtriser le flux pour une haute résolution, surtout si le service se situe sur une **Raspberry**.*

*Contrairement à la librairie **Qt** qui travaille sous forme d'événements, la communication réseau en **Rust** est différente puisque vous devez en permanence consulter les requêtes possibles venant du client. Dans ce principe, nous devons toujours avoir une alternance entre la requête et la réponse venant du service. Il n'est pas possible d'envoyer plusieurs réponses pour une seule requête puisque le système n'est pas capable de discriminer chacune de ses réponses.*

Lorsque nous avons travaillé sur le réseau dans les projets précédents, le client se connectait, soumettait sa requête, le service renvoyait alors sa réponse et se déconnectait du client juste après cet envoi.

Ici, la démarche est totalement différente puisque nous devons rester connecté pour envoyer le flux vidéo en continu jusqu'à ce que le client clôture son application.

*Lors de la clôture du client dans l'environnement de **GTK**, il faudra trouver une solution côté service qui est en attente permanente d'une requête venant du client alors que l'arrêt de l'application cliente se fait de façon anarchique. Enfin, il ne faut activer la prise en compte de la caméra que si un client se connecte (il peut y avoir plusieurs services dans la même **Raspberry**).*

Cargo.toml côté service

```
[package]
name = "service-camera"
```

```
version = "0.1.0"
edition = "2021"

[dependencies]
opencv = "0.63.1"
serde = {version="1.0.137", features=["derive"]}
serde_json = "1.0.81"
```

camera.rs côté service

```
use opencv::{core::*, video::*, videoio::*, imgproc::*, types::VectorOfMat};
use serde::Deserialize;

#[derive(Deserialize, Debug)]
pub struct Communication {
  pub largeur: f64,
  pub hauteur: f64,
  pub couleur: bool,
  pub detection: bool,
  pub frequence: u32,
  pub change: bool
}

pub struct Video {
  camera: VideoCapture,
  capture: Mat,
  inter: Mat,
  pub rgb: Mat,
  masque: Mat,
  point: Point,
  brosse: Mat,
  scalaire: VecN<f64, 4>,
  contours: VectorOfMat,
  fond: Ptr<dyn BackgroundSubtractorMOG2>,
  couleur: bool,
  detection: bool,
  pub frequence: u32
}

impl Video {
  pub fn generer(trame: &Communication) -> Self {
    let mut video = VideoCapture::new(0, CAP_ANY).unwrap();
    video.set(CAP_PROP_FRAME_WIDTH, trame.largeur).unwrap();
    video.set(CAP_PROP_FRAME_HEIGHT, trame.hauteur).unwrap();
    let point = Point::new(-1, -1);
    Video {
      camera: video,
      capture: Mat::default(),
      inter: Mat::default(),
      rgb: Mat::default(),
      masque: Mat::default(),
      point,
      brosse: get_structuring_element(MORPH_RECT, Size::new(15, 15), point).unwrap(),
      scalaire: Scalar::all(1.),
      contours: VectorOfMat::default(),
      fond: create_background_subtractor_mog2(500, 16., true).unwrap(),
      couleur: trame.couleur,
      detection: trame.detection,
      frequence: trame.frequence
    }
  }

  pub fn change(&mut self, trame: &Communication) {
    self.camera.set(CAP_PROP_FRAME_WIDTH, trame.largeur).unwrap();
    self.camera.set(CAP_PROP_FRAME_HEIGHT, trame.hauteur).unwrap();
    self.couleur = trame.couleur;
    self.detection = trame.detection;
    self.frequence = trame.frequence;
  }

  pub fn lire_video(&mut self) {
    self.camera.read(&mut self.capture).unwrap();
    self.gerer_couleur();
    if self.detection { self.gerer_detection() }
  }

  fn gerer_couleur(&mut self) {
    if self.couleur {
```

```
    cvt_color(&mut self.capture, &mut self.rgb, COLOR_BGR2RGB, 0).unwrap();
   }
   else {
    cvt_color(&mut self.capture, &mut self.inter, COLOR_BGR2GRAY, 0).unwrap();
    cvt_color(&mut self.inter, &mut self.rgb, COLOR_BGR2RGB, 0).unwrap();
   }
  }

  fn gerer_detection(&mut self) {
   BackgroundSubtractorMOG2::apply(&mut self.fond, &self.capture, &mut self.masque, -1.).unwrap();
   erode(&self.masque, &mut self.inter, &self.brosse, self.point, 1,BORDER_DEFAULT, self.scalaire).unwrap();
   dilate(&self.inter, &mut self.masque, &self.brosse, self.point, 3,BORDER_DEFAULT, self.scalaire).unwrap();
   find_contours(&self.masque, &mut self.contours, RETR_TREE, CHAIN_APPROX_SIMPLE, Point::new(0, 0)).unwrap();
   for contour in &self.contours {
    let zone = bounding_rect(&contour).unwrap();
    rectangle(&mut self.rgb, zone, Scalar::new(255., 0., 255., 0.), 3, FILLED, 0).unwrap();
   }
  }
}
```

*Les requêtes venant du client sont proposées en format **JSON**. Nous disposons alors de la structure **Communication** qui factorise les différents paramétrages possibles pour la mise en œuvre du flux vidéo. La prise en compte de ces paramètres n'est effectif que si un changement est proposé, ce qui est le cas au démarrage, à la connexion du client.*

*Nous retrouvons les mêmes traitements que dans le projet précédent. La seule différence, c'est que l'algorithme complet est encapsulé dans la structure **Video**.*

main.rs côté service

```
use std::net::{TcpListener, TcpStream};
use camera::*;
use opencv::core::*;
use serde_json::*;

fn main() {
  match TcpListener::bind(("0.0.0.0", 5555)) {
    Ok(service) => {
      println!("En attente d'un client");

      for connexion in service.incoming() {
        match connexion {
          Ok(mut client) => {
            println!("Un client se connecte");
            let mut video = Video::generer(&recuperer_trame(&mut client).unwrap());
            let mut iterateur = 0;
            loop {
              video.lire_video();
              if iterateur%(30/video.frequence) == 0 {
                let octets = video.rgb.data_bytes().unwrap();
                client.write(octets).unwrap();
                match recuperer_trame(&mut client) {
                  Ok(trame) => if trame.change { video.change(&trame) },
                  Err(_) => {
                    println!("Déconnexion du client");
                    break
                  }
                }
              }
              iterateur+=1;
            }
          },
          Err(_) => println!("Un client n'a pas réussi à se connecter"),
        }
      }
    }
    Err(_) => println!("Impossible de démarrer le service !")
  }
}

fn recuperer_trame(client: &mut TcpStream) -> Result<Communication> {
  let mut trame = [0; 256];
  let nombre = client.read(&mut trame).unwrap();
  serde_json::from_slice::<Communication>(&trame[..nombre])
}
```

Le service est en fonctionnement permanent. Lorsque qu'un client se connecte, la caméra est alors activée avec les différents réglages proposés par le client et le service envoi son flux vidéo en adéquation avec la fréquence choisie. Le flux vidéo ne s'arrête qu'à la condition que la réception de la trame ne s'effectue pas correctement, ce qui se produit lors d'une interruption de connexion intempestive venant du client.

Cargo.toml côté client

```
[package]
name = "client-camera"
version = "0.1.0"
edition = "2021"

[dependencies]
gtk = "0.15.5"
serde = {version="1.0.137", features=["derive"]}
serde_json = "1.0.81"
```

main.rs côté client

```
use gtk::*;
use gtk::prelude::*;
use gdk_pixbuf::*;
use std::rc::Rc;
use std::cell::Cell;
use gtk::glib::Bytes;
use serde::Serialize;
use std::io::{Read, Write};
use std::net::TcpStream;
use std::str::FromStr;

fn main() {
  gtk::init().unwrap();
  let vue = Builder::from_string(include_str!("../client-camera.glade"));
  let fenetre: Window = vue.object("fenetre").unwrap();
  let ihm = IHM::placer(vue);
  let vue = ihm.clone();
  fenetre.connect_destroy(move |_| {
    vue.continuer.set(false);
    gtk::main_quit() }
  );
  fenetre.show();
  ihm.changer();
  ihm.capture_video();
  gtk::main();
}

#[derive(Serialize, Clone)]
struct Communication {
  largeur: f64,
  hauteur: f64,
  couleur: bool,
  detection: bool,
  frequence: u32,
  change: bool
}

#[derive(Debug, Clone)]
struct IHM {
  image: Image,
  couleur: Switch,
  detection: Switch,
  resolution: ComboBoxText,
  frequence: ComboBoxText,
  continuer: Rc<Cell<bool>>,
  changement: Rc<Cell<bool>>
}

impl IHM {
  fn placer(vue: Builder) -> Self {
    IHM {
      image: vue.object("photo").unwrap(),
      couleur: vue.object("couleur").unwrap(),
      detection: vue.object("detection").unwrap(),
      resolution: vue.object("resolution").unwrap(),
      frequence: vue.object("frequence").unwrap(),
      continuer: Rc::new(Cell::new(true)),
      changement: Rc::new(Cell::new(false))
    }
  }

  fn changer(&self) {
    let changement = self.changement.clone();
    self.resolution.connect_changed(move |_| { changement.set(true);  });
    let changement = self.changement.clone();
```

```
    self.frequence.connect_changed(move |_| { changement.set(true);  });
    let changement = self.changement.clone();
    self.couleur.connect_changed_active(move |_| { changement.set(true);  });
    let changement = self.changement.clone();
    self.detection.connect_changed_active(move |_| { changement.set(true);  });
  }

  fn changer_image(&self, image: Vec<u8>) {
    let bytes = Bytes::from(&image);
    let (largeur, hauteur) = match self.resolution.active().unwrap() {
      0 => (640, 480),
      1 => (800, 600),
      2 | _ => (1280, 720)
    };
    let image = Pixbuf::from_bytes(&bytes, Colorspace::Rgb, false, 8, largeur, hauteur, largeur*3);
    self.image.set_from_pixbuf(Some(&image));
  }

  fn capture_video(&self) {
    let ihm = self.clone();
    let mut reglages = Communication {
      largeur: 640.,
      hauteur: 480.,
      couleur: true,
      detection: false,
      frequence: 10,
      change: true
    };

    match TcpStream::connect("172.16.20.31:5555") {
      Ok(mut service) => {
        glib::idle_add_local(move || {
          if ihm.changement.get() {
            (reglages.largeur, reglages.hauteur) = match ihm.resolution.active().unwrap() {
              0 => (640., 480.),
              1 => (800., 600.),
              2 | _ => (1280., 720.)
            };
            let texte = ihm.frequence.active_text().unwrap();
            reglages.frequence = u32::from_str(&texte[..texte.len()-6]).unwrap();
            reglages.couleur = ihm.couleur.is_active();
            reglages.detection = ihm.detection.is_active();
            reglages.change = true;
            ihm.changement.set(false);
          }
          let requete = serde_json::to_string::<Communication>(&reglages).unwrap();
          service.write(requete.as_bytes()).unwrap();
          reglages.change = false;

          let octets = recuperer_image(&mut service, &reglages);
          ihm.changer_image(octets);
          Continue(ihm.continuer.get())
        });
      },
      Err(_) => println!("Impossible de se connecter au service caméra")
    }
  }
}

fn recuperer_image(service: &mut TcpStream, reglages: &Communication) -> Vec<u8> {
  let limite = (reglages.largeur * reglages.hauteur * 3.) as usize;
  const TAILLE: usize = 4096;
  let mut octets = vec![];
  let mut tampon = [0; TAILLE];
  loop {
    let nombre = service.read(&mut tampon).unwrap();
    octets.extend_from_slice(&tampon[..nombre]);
    if octets.len()>=limite { break; }
  }
  octets
}
```

*Nous retrouvons pratiquement la même ossature que le projet précédent avec toutefois la prise en compte de la fréquence du flux vidéo. Nous retrouvons la fonction **idle_add_local()** qui permet de réaliser des traitements en tâche de fond. C'est dans cette fonction spécifique que nous proposons la connexion réseau avec la soumission des requêtes et la récupération du flux vidéo.*

*Vu la quantité énorme d'octets reçus pour chaque image de la vidéo, dans **recuperer_image()**, nous prévoyons une récupération par bloc de **4096** octets sachant que le maximum est de **65535**. Nous contrôlons exactement le nombre d'octets à recevoir.*

PLACER LA CAMÉRA SUR UNE RASPBERRY

Si vous placez le service précédent dans une Raspberry, avec la caméra bien connectée, vous remarquerez que la partie détection ne fonctionne pas, et le service s'arrête alors instantanément. C'est logique puisque les ressources utilisées par la classe **BackgroundSubtractorMOG2** sont trop importantes pour un nanoPC. Je rappelle que cette architecture stocke par défaut 500 images pour permettre une bonne analyse de l'image de fond.

> *C'est beaucoup trop pour une **Raspberry**. Il est possible de limiter considérablement le nombre d'images à conserver, mais ce système aurait alors beaucoup moins d'intérêt. Je pense qu'il est préférable d'utiliser la première solution de détection que nous avions déjà abordée lors de l'étude sur **OpenCV**.*

> *La solution de détection pour la **Raspberry** qui est beaucoup plus simple et qui fonctionne très bien consiste simplement à faire **une soustraction entre deux images consécutives**. Dans cette solution, toute la partie statique, dite image de fond, se retrouve en noir. Ce qui apparaît dans ce différentiel est uniquement la partie en mouvement. Il suffit alors de faire un masque et de retrouver les contours comme nous l'avons fait sur la solution précédente.*

camera.rs côté service

```rust
use opencv::{core::*, videoio::*, imgproc::*, types::VectorOfMat};
use serde::Deserialize;

#[derive(Deserialize, Debug)]
pub struct Communication {
  pub largeur: f64,
  pub hauteur: f64,
  pub couleur: bool,
  pub detection: bool,
  pub frequence: u32,
  pub change: bool
}

pub struct Video {
  camera: VideoCapture,
  capture: Mat,
  inter: Mat,
  pub rgb: Mat,
  masque: Mat,
  avant: Mat,
  gris: Mat,
  point: Point,
  brosse: Mat,
  scalaire: VecN<f64, 4>,
  contours: VectorOfMat,
  couleur: bool,
  detection: bool,
  pub frequence: u32
}

impl Video {
  pub fn generer(trame: &Communication) -> Self {
    let mut video = VideoCapture::new(0, CAP_ANY).unwrap();
    video.set(CAP_PROP_FRAME_WIDTH, trame.largeur).unwrap();
    video.set(CAP_PROP_FRAME_HEIGHT, trame.hauteur).unwrap();
    let point = Point::new(-1, -1);
    let mut avant = Mat::default();
    video.read(&mut avant).unwrap();

    Video {
      camera: video,
      capture: Mat::default(),
      inter: Mat::default(),
      rgb: Mat::default(),
      masque: Mat::default(),
      gris: Mat::default(),
      avant,
      point,
      brosse: get_structuring_element(MORPH_RECT, Size::new(11, 11), point).unwrap(),
      scalaire: Scalar::all(1.),
      contours: VectorOfMat::default(),
      couleur: trame.couleur,
      detection: trame.detection,
      frequence: trame.frequence
    }
  }

  pub fn change(&mut self, trame: &Communication) {
    self.camera.set(CAP_PROP_FRAME_WIDTH, trame.largeur).unwrap();
    self.camera.set(CAP_PROP_FRAME_HEIGHT, trame.hauteur).unwrap();
```

```
    self.couleur = trame.couleur;
    self.detection = trame.detection;
    self.frequence = trame.frequence;
  }

  pub fn lire_video(&mut self) {
    self.camera.read(&mut self.capture).unwrap();
    self.gerer_couleur();
    if self.detection { self.gerer_detection() }
  }

  fn gerer_couleur(&mut self) {
    if self.couleur {
      cvt_color(&mut self.capture, &mut self.rgb, COLOR_BGR2RGB, 0).unwrap();
    }
    else {
      cvt_color(&mut self.capture, &mut self.inter, COLOR_BGR2GRAY, 0).unwrap();
      cvt_color(&mut self.inter, &mut self.rgb, COLOR_BGR2RGB, 0).unwrap();
    }
  }

  fn gerer_detection(&mut self) {
    let soustraction = ((&self.rgb-&self.avant) * 7.).into_result().unwrap().to_mat().unwrap();
    cvt_color(&soustraction, &mut self.gris, COLOR_BGR2GRAY, 0).unwrap();
    threshold(&self.gris, &mut self.masque, 70., 255., THRESH_BINARY).unwrap();
    erode(&self.masque, &mut self.inter, &self.brosse, self.point, 1,BORDER_DEFAULT, self.scalaire).unwrap();
    dilate(&self.inter, &mut self.masque, &self.brosse, self.point, 7,BORDER_DEFAULT, self.scalaire).unwrap();
    self.rgb.copy_to(&mut self.avant).unwrap();
    find_contours(&self.masque, &mut self.contours, RETR_TREE, CHAIN_APPROX_SIMPLE, Point::new(0, 0)).unwrap();
    for contour in &self.contours {
      let zone = bounding_rect(&contour).unwrap();
      rectangle(&mut self.rgb, zone, Scalar::new(255., 0., 255., 0.), 3, FILLED, 0).unwrap();
    }
  }
}
```

UTILLISER OPENCV DANS UNE RASPBERRY

Lorsque nous utilisons une camera et que nous devons réaliser des analyses d'images particulières comme pour récupérer le numéro d'une plaque minéralogique d'une voiture qui vient de passer devant la caméra, nous pouvons fabriquer ce dispositif avec un système numérique de type **Raspberry**.

*L'idéal, dans ce genre de système, c'est que la **Raspberry** soit juste un système numérique communicant sans clavier ni écran, comme devrait l'être tout **nano-PC** dont l'objectif principal est généralement de capturer des informations à partir de capteurs câblés et piloter éventuellement des actionneurs pour par exemple démarrer le lave-linge à distance depuis son smartphone.*

*Le capteur qui nous intéresse ici, est une simple webcam qui va nous permettre d'avoir une vision à distance. En effet, nous pourrons nous servir de notre PC pour consulter à distance une camera connecté à cette **Raspberry**. Pour que cela fonctionne correctement, la librairie **OpenCV** doit être installée directement dans la **Raspberry**.*

Attention, dans ce cadre là, vu que la Raspberry ne peut être utilisée qu'à distance et qu'elle dispose d'un processeur ARM totalement différent de ceux qui existent dans un PC, il est plus judicieux de réaliser une suite de développement en compilation croisée.

C'est-à-dire que la conception du programme se fait sur un PC normal en utilisant un compilateur prévu pour un autre type de processeur en association avec une librairie OpenCV prévue également pour cet autre type de processeur. Le programme ainsi constitué sera ensuite déployé sur la Raspberry cible qui sera ensuite exécutée à distance pour remplir sa fonction.

*Pour réaliser tout ce processus, nous devons construire deux nouvelles librairies **OpenCV**, une pour la **Raspberry** afin qu'elle soit adaptée au processeur **ARMv7**, et une autre placée dans le **PC** de développement qui nous permet de réaliser nos programmes avec les outils que nous connaissons bien.*

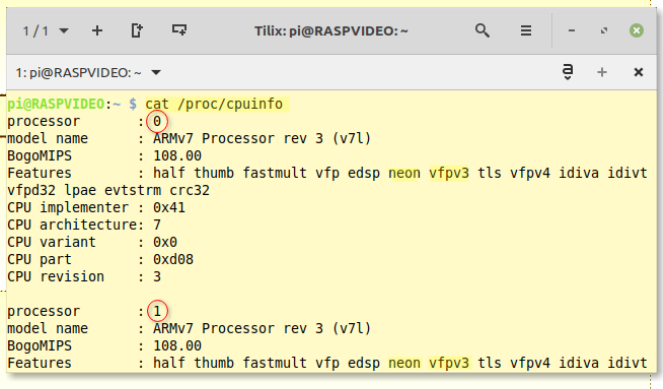
***OpenCV pour ARMv7 côté PC de développement** : côté **PC**, pour réaliser cette compilation croisée, nous devons disposer du bon compilateur adapté aux processeurs de type **ARMv7**. Installez-le :*

```
~$ sudo apt-get install crossbuild-essential-armhf
```

*Ensuite, nous réutilisons les mêmes sources **OpenCV** qui nous ont servis à créer notre librairie normale directement pour le PC, mais cette fois-ci la configuration sera bien entendu différente pour prendre en compte la compilation croisée d'une part, et les instructions spécifiques de la Raspberry d'autre part.*

*En effet, les dernières **Raspberry** intègrent des processeurs plus performants qui possèdent des instructions spécifiques de type **NEON** et **VFPV3**. Voici ci-dessous les commandes à réaliser pour la mise en œuvre de tout le processus :*

```
~$ cmake  -D CMAKE_TOOLCHAIN_FILE=../platforms/linux/arm-gnueabi.toolchain.cmake
          -D ENABLE_VFPV3=ON   -D ENABLE_NEON=ON   -D CMAKE_INSTALL_PREFIX=/opt/opencv-armhf   ..
```



```
~$ make -j4
~$ sudo make install
```

OpenCV dans la Raspberry *: Pour la Raspberry, vous devez également récupérer les sources* ***OpenCV*** *et réaliser votre compilation afin d'obtenir les binaires respectifs directement à l'intérieur qui seront exploités lorsque les programmes de traitement d'images seront déployés. Là aussi,* ***OpenCV*** *doit pouvoir prendre en compte toutes les compétences du processeur utilisé.*

Attention*, vu que le développement des applications se fait par compilation croisée, vous devez préciser au système d'exploitation le chemin où se trouve les librairies* ***OpenCV*** *ou les déplacer dans un répertoire déjà connu, comme «* ***/usr/lib*** *».*

```
~$ cmake -D CMAKE_BUILD_TYPE=RELEASE   -D ENABLE_VFPV3=ON  -D ENABLE_NEON=ON
        -D CMAKE_INSTALL_PREFIX=/opt/opencv-armhf   ..
~$ make -j4
~$ sudo make install
~$ sudo  cp  /opt/opencv-armhf/libopencv*  /usr/lib
```

MISE EN PLACE DU SERVICE SUR LA RASPBERRY - CROSS-COMPILATION

Les outils de **Rust** sont relativement puissants et complets. Ils permettent notamment de réaliser des programmes pour **plusieurs cibles différentes** tout en restant sur notre poste de développement. C'est ce que nous appelons faire de la **compilation croisée** sur le même poste de travail.

Il est nécessaire au préalable d'installer les compilateurs pour chacune des cibles souhaitées et ceci toujours grâce à outil ***rustup****. Dans la documentation de* ***Rust****, vous avez une très grande liste de cibles. Pour compiler un programme pour qu'il fonctionne sur une* ***Raspberry****, vous devez prendre le compilateur* ***armv7-unknown-linux-gnueabihf*** *:*

```
> rustup target add armv7-unknown-gnueabihf
```

Les commandes précédentes, bien entendu, sont à faire qu'une seule fois sur votre ordinateur hôte. Par contre, par défaut, lorsque vous exécutez un projet, la cible reste votre ordinateur hôte. Si vous souhaitez changer de cible, vous devez rajouter dans votre projet un répertoire « ***.cargo*** *» à l'intérieur duquel vous placez le fichier «* ***config.toml*** *» avec le script suivant :*

.cargo/config.toml

```
[build]
target = "armv7-unknown-linux-gnueabihf"

[target.armv7-unknown-linux-gnueabihf]
linker = "arm-linux-gnueabihf-gcc"
```

Lorsque vous construisez votre projet, le système tient compte de ce fichier, et la compilation correspond à un exécutable de type ***Raspberry****.* ***Attention****, pensez à changer la localisation de la librairie* ***OpenCV*** *pour la Raspberry à l'aide de deux variables d'environnement :* ***OpenCV_DIR=/opt/opencv-armhf;LD_LIBRARY_PATH=/opt/opencv-armhf/lib****. Déployer votre service :*

```
> scp service-camera  pi@192.168.1.25:/home/pi/rust
```

Une fois que votre service est placé dans la ***Raspberry****, vous devez ensuite faire en sorte que ce service soit opérationnel dès le démarrage du* ***nano-PC****. Pour cela, vous devez proposer un* ***fichier de configuration*** *dans le répertoire prévu à cet effet «* ***/etc/init/{nom du fichier.conf}*** *» à l'intérieur duquel vous placez les lignes suivantes :*

/etc/init/service-camera.conf

```
#!/bin/sh
description "Visualisation vidéo à distance"
start on startup
task
exec /home/pi/rust/service-camera
```

Pour que tous ces services soient parfaitement opérationnels, vous devez rajouter l'exécutable dans le système de lancement automatique local « ***/etc/rc.local*** *» qui prendra en compte alors les configurations des différents services comme le précédent :*

/etc/rc.local

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

/home/pi/rust/service-comptes &
/home/pi/rust/service-ressources &
/home/pi/rust/service-camera &
```