

# La vision en FRC avec GRIP

Étienne Beaulac

Colloque de la communauté *FIRST* au Québec  
École Cavalier-de LaSalle, Montréal, le 20 octobre 2018



# Vue d'ensemble

- 1 Aperçu de la vision en FRC
  - Qu'est-ce que la vision ?
  - Pourquoi l'utiliser en FRC ?
- 2 Acquérir
  - Choisir une caméra
  - Configurer sa caméra
- 3 Analyser
  - Créer un pipeline avec GRIP
  - Intégrer GRIP au code
- 4 Agir
- 5 Références

# Aperçu de la vision en FRC

Qu'est-ce que la vision ?

Selon Wikipédia [9] :

*La **vision par ordinateur** est une branche de l'intelligence artificielle dont le principal but est de permettre à une machine d'analyser, traiter et comprendre une ou plusieurs images prises par un système d'acquisition.*

# Aperçu de la vision en FRC

Pourquoi l'utiliser en FRC ?

Chaque année, des bandes rétro réfléchissantes marquent des endroits clés du terrain. Elles sont très faciles à détecter.



FIGURE 1 – Réflexion et rétro réflexion [10]

# Aperçu de la vision en FRC

Pourquoi l'utiliser en FRC ?

Figure 2-10: Vision Guides on the Yellow TOTE

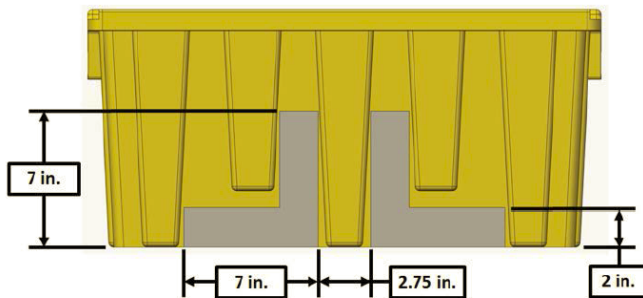
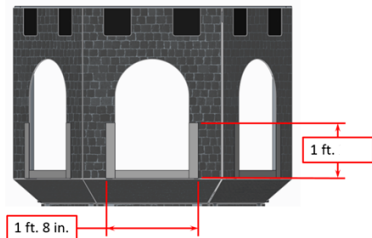


FIGURE 2 – Cibles de vision de *Recyclage Express* (2015) [1]

# Aperçu de la vision en FRC

Pourquoi l'utiliser en FRC ?



**FIGURE 3** – Cibles de vision de *La forteresse* (2016) [2]

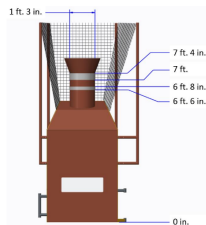


**FIGURE 4** – Détection des cibles de *La forteresse* [10]

# Aperçu de la vision en FRC

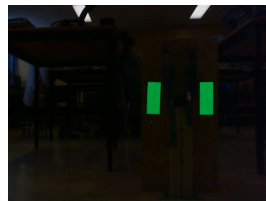
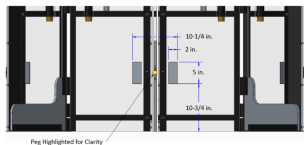
Pourquoi l'utiliser en FRC ?

Figure 3-38: BOILER vision target measurements



**FIGURE 5** – Cibles de vision de *À toute vapeur* (2017) [3]

Figure 3-39: LIFT peg vision target dimensions

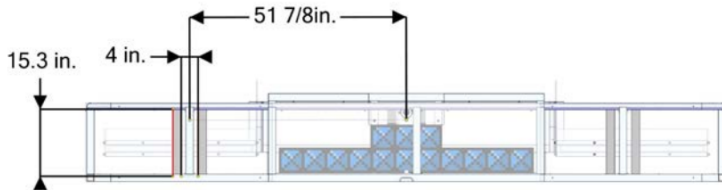


**FIGURE 6** –  
Détection des cibles  
de *À toute vapeur*

# Aperçu de la vision en FRC

Pourquoi l'utiliser en FRC ?

Figure 3-23: Vision Target locations



**SOLIDWORKS**  
Modeling Solutions Partner

FIGURE 7 – Cibles de vision de *Prochain niveau* (2018) [4]



# Acquérir

Choisir une caméra

- Caméra USB (webcam)
  - Microsoft Lifecam HD-3000 (20 crédits,  $\approx 40\$$ )
  - Genius Widedcam F100 ( $\approx 50\$$ )
- Caméra réseau
  - Axis camera ( $\approx 250\$$ )
- Téléphone intelligent
- PixyCam (Arduino), JeVois, etc.



FIGURE 8 –  
Microsoft Lifecam  
HD-3000

# Acquérir

Choisir une caméra

Un anneau de DEL vertes (10 crédits,  $\approx$  5-10 \$) permet d'illuminer les bandes rétro réfléchissantes.



FIGURE 9 – Une caméra équipée d'un anneau de DEL [5]

# Acquérir

## Configurer sa caméra

Pour ne voir que les bandes vertes, il est important de diminuer la luminosité et l'exposition de la caméra.

```
// Demarre l'envoi d'images
UsbCamera camera = CameraServer.getInstance().
    startAutomaticCapture();
camera.setBrightness(2); // Entre 0 et 100
camera.setExposureManual(5); // Entre 0 et 100
```

Pour accéder à la caméra :

<http://roboRIO-TEAM-FRC.local:1181/?action=stream>,  
où TEAM est le numéro de l'équipe.

Les sources vidéos subséquentes seront disponibles sur les ports 1182, 1183, etc.

# Analyser

## Créer un pipeline avec GRIP

- GRIP est un logiciel créé par l'équipe qui développe WPILib, la librairie Java/C++ utilisée en FRC.
- Son interface graphique permet de voir en temps réel l'effet de ses choix.
- Il permet d'exporter ses *pipelines* directement en Java, C++ ou Python.
- On peut également exécuter GRIP à même la DriverStation, puis envoyer les résultats au robot.
- Site web :  
<https://wpiroboticsprojects.github.io/GRIP/>

# Analyser

## Créer un pipeline avec GRIP

Un *pipeline* est une suite d'opérations où le résultat d'un bloc est l'entrée du suivant.

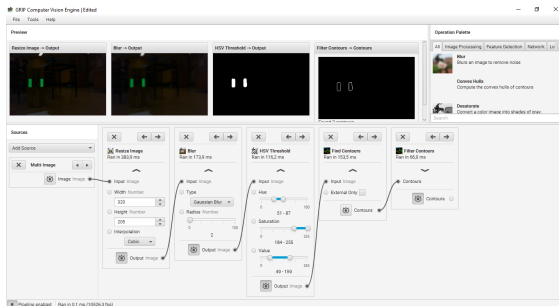


FIGURE 10 – Un *pipeline* GRIP

# Analyser

Créer un pipeline avec GRIP

Pour détecter les bandes vertes, les opérations utilisées sont habituellement les suivantes :

- 1 Redimensionner l'image (*Resize*)
- 2 Flou (*Blur*)
- 3 Seuil HSV (*HSV Threshold*)
- 4 Trouver les contours (*Find contours*)
- 5 Filtrer les contours (*Filter contours*)

# Analyser

Créer un pipeline avec GRIP

## 1. Redimensionner l'image (*Resize Image*)

On s'assure d'avoir une image de taille fixe, peu importe les paramètres de la caméra. De plus, une image plus petite sera traitée plus rapidement.

## 2. Flou (*Blur*)

Un léger flou permet d'ignorer les imperfections de l'image. Un *gaussian blur* ou *box blur* est adéquat.

# Analyser

Créer un pipeline avec GRIP

## 3. Seuil HSV (*HSV Threshold*)

On conserve uniquement les pixels qui nous intéressent. Le modèle HSV est préférable au modèle RGB.

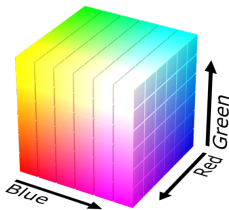


FIGURE 11 – Le modèle RGB [8]

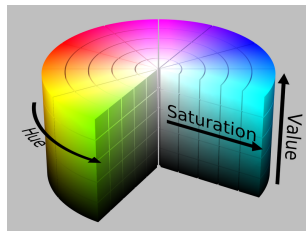


FIGURE 12 – Le modèle HSV [7]



# Analyser

Créer un pipeline avec GRIP

## 4. Trouver les contours (*Find contours*)

On convertit les amas de pixels blancs en contours, afin de pouvoir en extraire de l'information (centre, dimensions, aire).

## 5. Filtrer les contours (*Filter contours*)

On conserve uniquement les contours qui respectent certains critères :

- Taille (longueur et/ou largeur)
- Aire
- Ratio  $\frac{\text{largeur}}{\text{hauteur}}$
- Solidité

# Analyser

Créer un pipeline avec GRIP

GRIP comprend plusieurs autres fonctionnalités :

- vérifier l'efficacité du *pipeline* : Tools > Analyze
- *Find Blobs* et *Find Lines*
- envoyer les résultats via HTTP et NetworkTables
- *erode* et *dilate*
- etc.

# Analyser

## Intégrer GRIP au code

Le bouton **Tools** > **Generate** permet de créer le code équivalent au *pipeline* GRIP (Java, C++ et Python). Nous utiliserons l'option *WPILib VisionPipeline*.

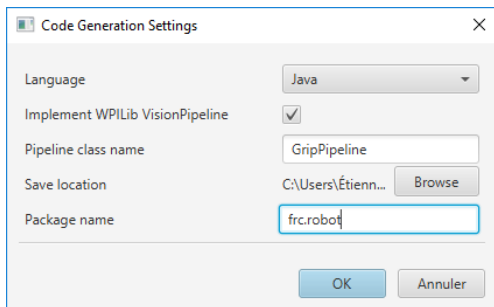


FIGURE 13 – Génération de code à partir de GRIP

# Analyser

Intégrer GRIP au code

On crée un VisionThread qui s'occupera d'exécuter le GripPipeline.

```
USBCamera camera = CameraServer.getInstance().  
    startAutomaticCapture();  
VisionThread thread = new VisionThread(camera, new  
    GripPipeline(), this);  
thread.start();
```

Il est essentiel d'utiliser un autre thread pour la vision.

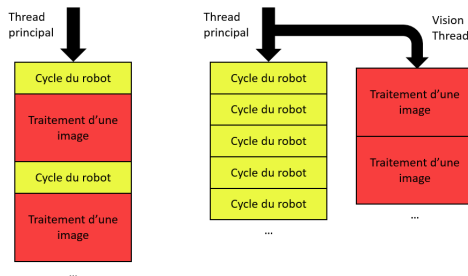


FIGURE 14 – Un seul thread comparativement à deux threads

# Analyser

Intégrer GRIP au code

L'utilisation de `VisionThread` demande l'implémentation de l'interface `VisionRunner.Listener<GripPipeline>` ayant comme méthode `copyPipelineOutputs(GripPipeline)`.

```
public void copyPipelineOutputs(GripPipeline pipeline) {  
    // Calculs et extraction d'information. Par exemple :  
    Mat image = pipeline.hsvThresholdOutput();  
    ArrayList<MatOfPoint> contours = pipeline.  
        filterContoursOutput();  
}
```

# Analyser

## Intégrer GRIP au code

La classe `Mat` représente une image, tandis que `MatOfPoint` représente un contour. On peut convertir un contour en rectangle (`Rect` : `x`, `y`, largeur, hauteur, aire).

```
// Rectangle qui correspond au contour
Rect rect = Imgproc.boundingRect(contour);

// Centre du rectangle
double centre = rect.x + rect.width / 2.0;

// Centre normalise
centre = 2 * centre / ((double) LARGEUR_IMAGE - 1;
```

# Analyser

Intégrer GRIP au code

Les coordonnées  $(x, y)$  d'un `Rect` représentent le coin supérieur droit. Son centre correspond donc à

$$\text{centre}_x = x + \frac{\text{width}}{2}$$

Ces coordonnées sont en pixels. Pour normaliser  $\text{centre}_x$  dans l'intervalle  $[-1, 1]$ , on utilise la formule

$$\text{centre}'_x = \frac{2 \times \text{centre}_x}{\text{largeur image}} - 1$$



Quel contour choisir lorsque GRIP en détecte plusieurs ?

- Le premier
- Le plus gros
- Le plus au centre
- Le meilleur score selon différents critères (ratio, solidité)

Les informations extraites des images peuvent être utilisées comme n'importe quel autre capteur.

- Le centre en  $x$  pour tourner
- Le centre en  $y$  pour avancer ou reculer (cibles en hauteur)
- La largeur ou la hauteur pour avancer ou reculer (plus la cible est petite, plus le robot est loin)

Pour ce faire, on peut utiliser différents systèmes de contrôle :

- *Bang-bang control*
- Régression linéaire
- Contrôleur PID
- Conversion en angle pour le gyro ou en distance pour les encodeurs (*sensor fusion*)

# Démonstration et questions

La présentation, le projet GRIP et le projet Java sont disponibles sur  
`github.com/etiennebeaulac/Atelier-GRIP-Public`.

Courriel : `etienne.beaulac@uqtr.ca`

# Références

Pour aller plus loin :

- Tutoriels de WPILib [10]
- Présentation de Jaci Brunning (OpenRIO, GradleRIO) [5]
- Présentation de l'équipe FRC 254 - Cheesy Poofs [6]

# Références

- [1] **FIRST.**  
2015 Game Manual - Recycle Rush.  
[https://www.firstinspires.org/sites/default/files/uploads/resource\\_library/frc/game-and-season-info/archive/2015/GameManual20150407.pdf](https://www.firstinspires.org/sites/default/files/uploads/resource_library/frc/game-and-season-info/archive/2015/GameManual20150407.pdf).
- [2] **FIRST.**  
2016 Game Manual - Stronghold.  
<https://firstfrc.blob.core.windows.net/frc2016manuals/GameManual/FRC-2016-game-manual.pdf>.
- [3] **FIRST.**  
2017 Game Manual - Steamworks.  
<https://firstfrc.blob.core.windows.net/frc2017/Manual/2017FRCGameSeasonManual.pdf>.
- [4] **FIRST.**  
2018 Game Manual - Power Up.  
<https://firstfrc.blob.core.windows.net/frc2018/Manual/2018FRCGameSeasonManual.pdf>.
- [5] **Jaci Brunning.**  
Computer Vision : Scratching The Surface.  
<https://docs.google.com/presentation/d/1vgMuifEYkToz7KGrrdd0VZSc6E1mU0K-z4cKvbQZ1kbo/>.
- [6] **FRC 254 Cheesy Poofs.**  
Integrating Computer Vision with Motion Control.  
<https://www.team254.com/documents/vision-control/>.
- [7] **SharkD.**  
HSV color solid cylinder saturation gray.  
[https://commons.wikimedia.org/wiki/File:HSV\\_color\\_solid\\_cylinder\\_saturation\\_gray.png](https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png).

# Références

- [8] **SharkD.**  
RGB color solid cube.  
[https://commons.wikimedia.org/wiki/File:RGB\\_color\\_solid\\_cube.png](https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png).
- [9] **Wikipedia.**  
Vision par ordinateur.  
[https://fr.wikipedia.org/wiki/Vision\\_par\\_ordinateur](https://fr.wikipedia.org/wiki/Vision_par_ordinateur).
- [10] **WPI.**  
Vision Processing | 2018 FRC Control System.  
<https://wpilib.screenstepslive.com/s/currentCS/m/vision>.