Linor

Traitement d'image

Nathan Latino, Sol Rosca (INF3b)

Mai 2020

Table des matières

1. Guide utilisateur

Vidéo de démonstration sur Youtube. Le lien se trouve dans le fichier `docs/youtube.txt`.

1.1. Hiérarchie

Cmd

```
.
├── main.py
├── Settings.py
├── setup.py
└── src
    ├── functions.py
    ├── GUI.py
    ├── objects
    │   ├── Image.py
    │   ├── Line.py
    │   └── Point.py
    ├── steering
    │   ├── directkeys.py
    │   └── Steering.py
    └── tools
        ├── Capture.py
        ├── Helpers.py
        └── Smoothing.py
```

- `main.py` : Point d'entrée à exécuter pour lancer le programme
- `settings.py` : Configuration initiale du programme
- `setup.py` : Installeur du programme
- `src/` : Contient les sources du projet
- `src/objects` : Le coeur de la logique fonctionnelle du programme
- `src/steering` : Partie du programme en charge de piloter un jeu
- `src/tools` : Abstractions utiles à toutes les autres parties du programme

1.2. Dépendances

- Python 3.7
- API win32 2.27

Ne fonctionne pas sous Linux. Dépendance à l'API win32.

1.3. Installation

Il est vivement conseillé de faire usage d'un environnement virtuel

1. Téléchargez ou clonez les sources du projet sur github.
2. Après décompression de l'archive, ouvrez un terminal à la racine du projet.
3. (optionel) Créez un nouvel environnement virtuel et activez le.
4. Installation des dépendances avec `$ pip install -e .`

1.4. Configuration

Il est recommandé d'utiliser deux écrans ou un écran d'une résolution supérieure à 1920x1080 pour utiliser confortablement ce prgramme.

Le programme se base sur le fait que l'écran principal (celui où se trouve en principe la barre des tâches) est celui qui affiche le média à traiter et que ce dernier se trouve dans le **coins supérieur gauche** et est affiché dans une fenêtre d'une **résolution de 1024 x 768**.

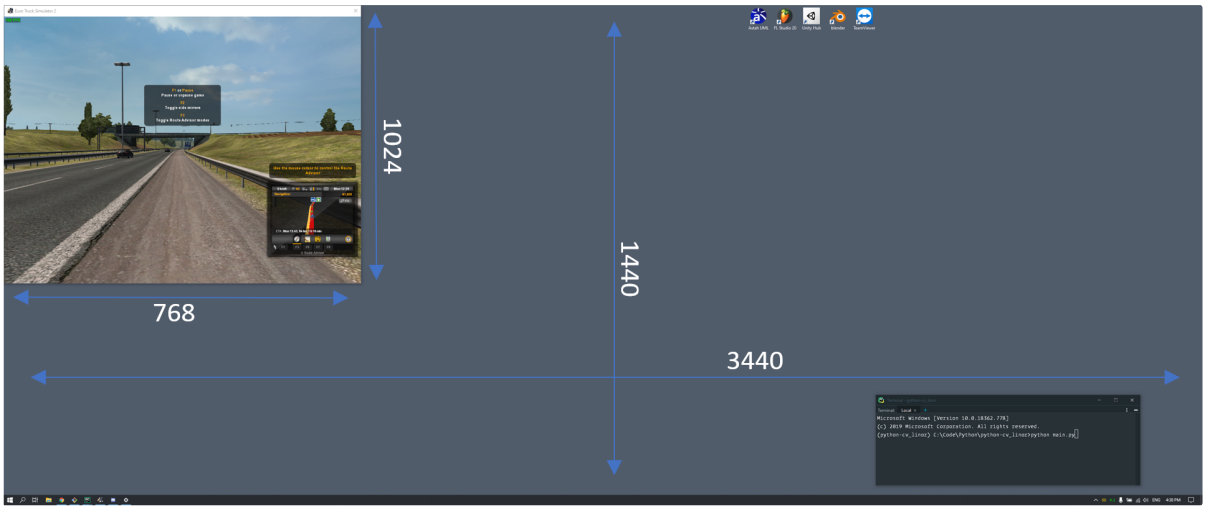

Figure 1 Dimensions et mise en place

À la racine du projet se trouve un fichier `settings.py` où il est possible de modifier le comportement précédemment décrit à l'aide des clés `x-offset` et `y-offset` du dictionnaire `settings`. Il est également possible de changer la résolution à l'aide de la clé `resolution`.

Les autres options sont toutes modifiables via l'interface du programme.

1.5. Utilisation

Pour lancer le programme, une fois le média à capturer à la bonne place, il suffit d'aller à la racine du projet et en s'assurant que l'environnement du projet est actif taper:

```
python3 main.py
```

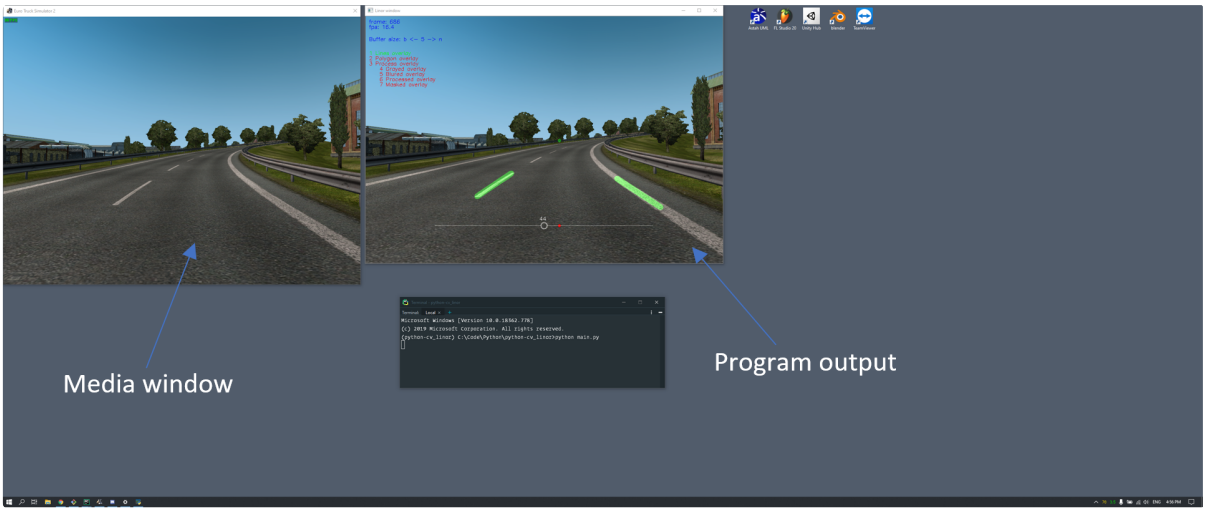

Figure 2 En action

Une fois le programme en route, la détection commence et le rendu est affiché dans une nouvelle fenêtre.

Une fonctionnalité détaillée au prochain point permet d'ouvrir une seconde vue qui illustre ce que le programme voit:

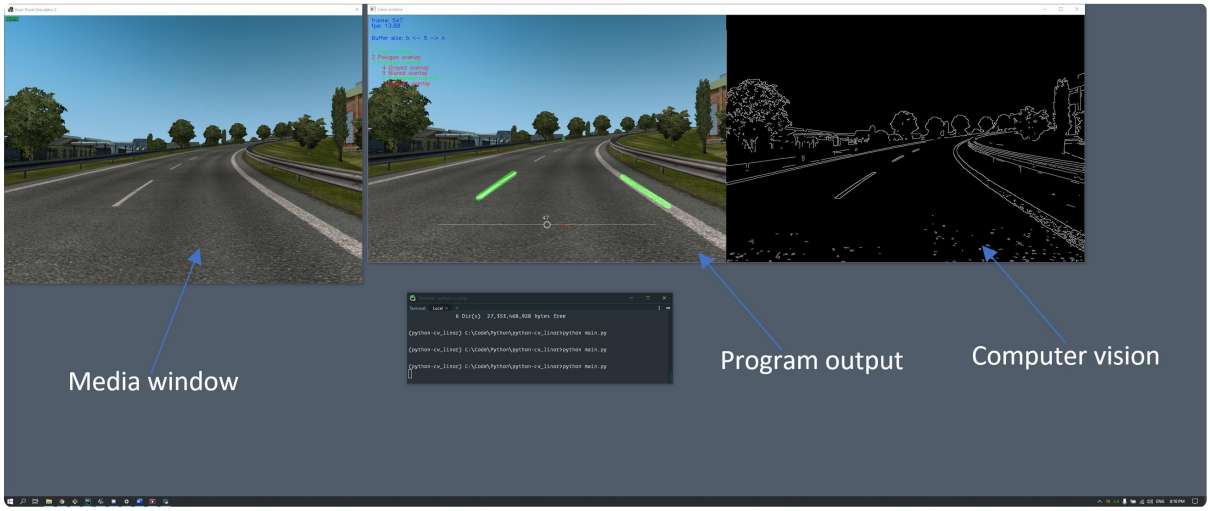

Figure 3 Illustre la nécessité d'un second écran ou d'une plus grande résolution

1.6. Interface

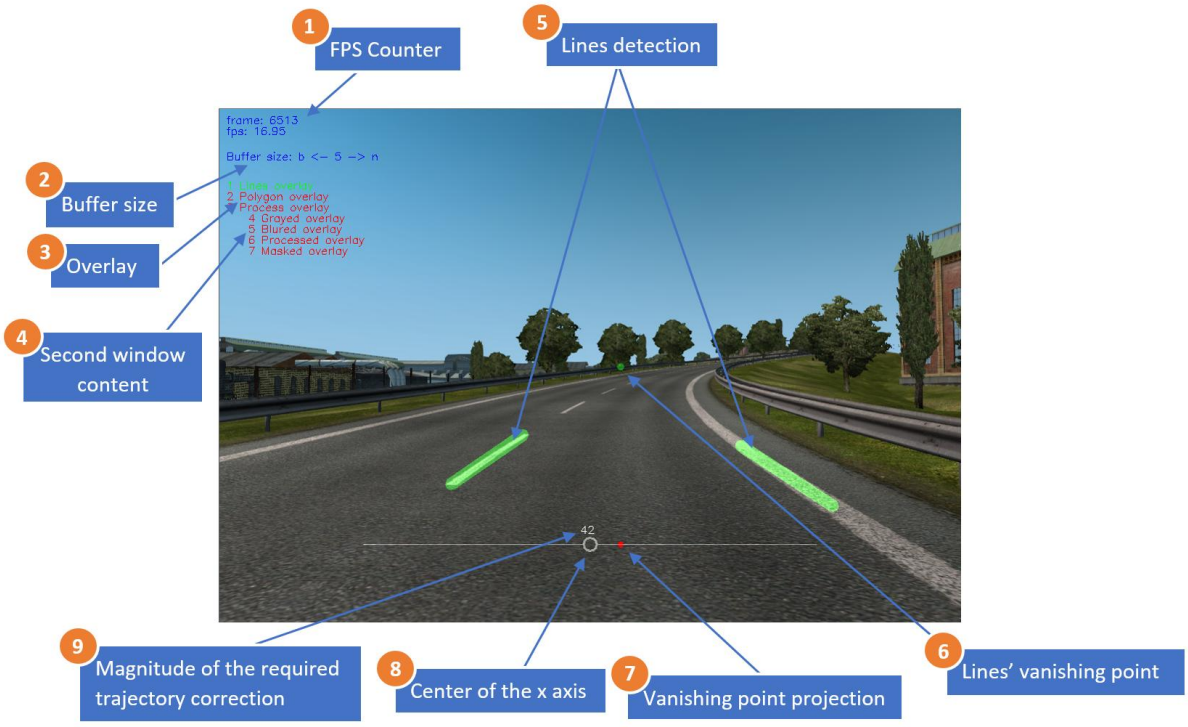

Figure 4 Éléments de l'interface

1. Compteur d'image + images par seconde
2. Taille du tampon utilisé pour lisser la détection
   - Augmenter la taille avec la touche `n`
   - Réduire la taille avec la touche `b`
3. Menu des overlays (activation via le clavier):
   - `1` Lines overlay: Affiche le rendu de la détection
   - `2` Polygon overlay: Affiche le polygone de détection
4. Contenu de la seconde fenêtre (activation avec `3`):
   - `4` Niveaux de gris
   - `5` Flou gaussien
   - `6` Canny filter
   - `7` Polygone de détection
5. Rendu de la détection
6. Point de fuite des lignes détectées
7. Projection du point de fuite sur un axe monodimensionnel
8. Centre de l'axe x de l'image
9. Intensité de la correction requise

1.6.1. Rendu de la détection

L'élément 5 (touche `1` pour activer/désactiver) de la **figure 4** illustre le résultat de interprétation du programme. C'est à dire deux lignes vertes (ou rouge en fonction de la situation). Le programme recherche des lignes dans l'environnement qu'il catégorise selon la valeur de leur pente. Si la pente est positive, la ligne devrait (en principe) faire partie des éléments de gauche, si elle est négative alors de celles de droite. Finalement une moyenne pour chaque groupe de lignes est faite pour en obtenir les résultantes à afficher.

Si pour une raison ou pour une autre la détection ne trouve pas de ligne d'un coté ou de l'autre, c'est le résultat de l'image précédente pour ce coté qui est gardé. Dans ce cas, les lignes affichées prennent une teinte rouge pour signifier la baisse de fiabilité de l'interprétation:

Figure 5 Baisse de fiabilité

Le pire cas étant celui qui arrivera l'instant qui suit la précédente figure, lorsqu'il n'y aura plus rien pour permettre au programme de déduire la direction à suivre. Dans ce cas, le programme continue de donner la dernière information qu'il avait en attendant de trouver de nouveaux éléments valables.

1.6.2. Point de fuite

L'élément 6 de la **figure 4**. Il s'agit tout simplement du point où convergent les lignes détectées par le programme. Le données de ce point sont utilisées pour déduire l'intensité du décalage vis-à-vis de la trajectoire. C'est en réalité uniquement sa composante horizontale qui est intéressante et qui sera détaillée plus loin dans ce guide.

1.6.3. Taille du tampon temporel

L'élément 2 de la **figure 4** est la taille (en nombre d'images) du tampon de lissage temporel des données. Ce dernier permet de lisser le résultat de l'interprétation du programme.

Plus la valeur est élevée (touche `n`), plus le programme sera fluide dans l'animation de l'affichage de la détection des bandes mais en contrepartie il y perd en réactivité. Si la valeur est vraiment trop grande il se crée un retard entre la direction suggérée et la réalité.

Si au contraire elle est trop faible (touche `b`), les résultats deviennent chaotiques car le programme est trop réactif et se laisse influencé par les éléments visuels ayant une faible temporalité.

Un bon compromis permet de gommer une partie des faux positifs ainsi que de lisser l'animation du résultat.

1.6.4. Polygone de détection

Le *polygone de détection* `3` est la zone considérée par le programme lors de la recherche d'éléments à caractériser:

Figure 6 Polygone de détection

La figure qui suit illustre dans sa partie **gauche** l'activation de la visualisation du *polygone de détection* `2` et sur la partie de **droite** se trouve la *seconde fenêtre* `3` avec son contenu set sur le mode détection de bords avec *Canny filter* `6`:

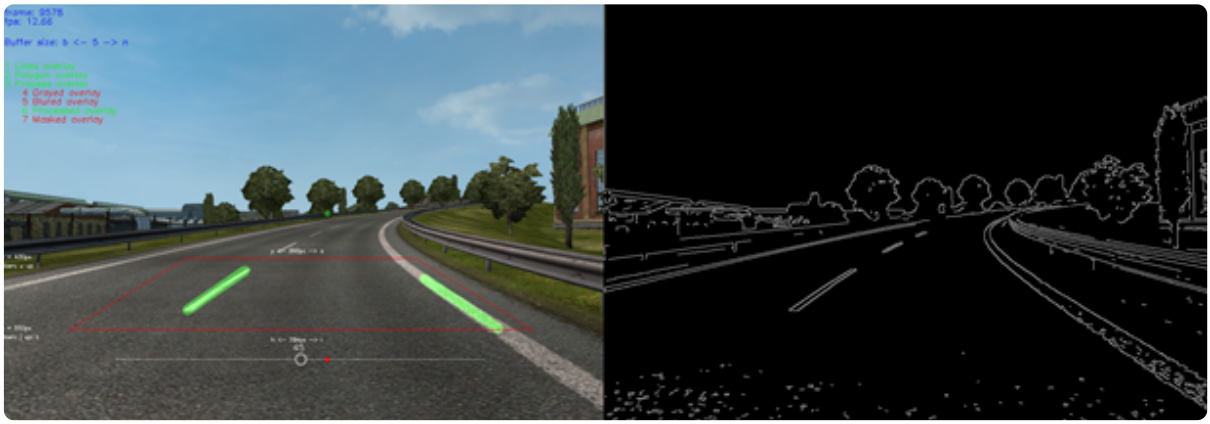

Figure 7 Fenêtre secondaire

Même si l'écran de droite montre que toute l'image est filtrée, le programme, lui, ne s'intéresse qu'à la portion délimitée par le trapèze rouge apparaissant sur la partie gauche de l'écran ainsi qu'illustré par la figure suivante qui active le *polygone de détection* `7` :

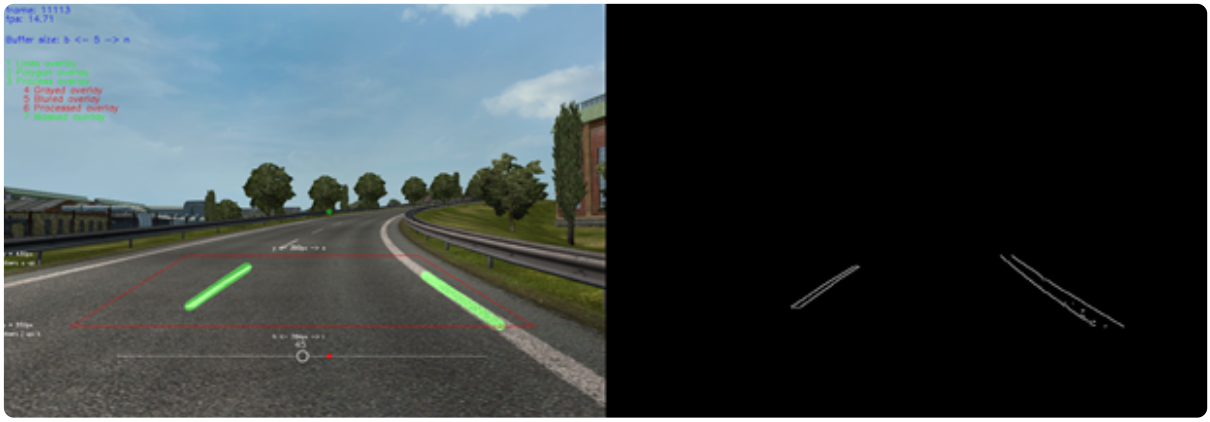

Figure 8 Influence du polygone de détection

Il est possible de modifier le polygone de détection en cours d'exécution du programme avec les raccourcis suivants:

- Hauteur de la petite base: `i` (+) et `u` (-)
- Hauteur de la grande base: `k` (+) et `j` (-)
- Largeur de la petite base: `y` (+) et `o` (-)
- Largeur de la grande base: `h` (+) et `l` (-)

La modification du polygone permet d'adapter la détection à d'autres médias comme un flux vidéo, un autre jeu ou encore un mode différent du même jeu:

Figure 9 Changement considérable du contexte de traitement

Pour un résultat optimal, un compromis entre la taille, la forme et la position du polygone doit être trouvé. Un trop grand polygone donnera de nombreux faux positif et à l'inverse un trop petit polygone réduira la réactivité du programme et l'empêchera de trouver les informations qu'il cherche.

Un trapèze avec une grande hauteur rendra le programme plus apte à anticiper la trajectoire mais si il est trop haut, par exemple, un des problèmes qui survient est que lors des côte, il interprètera la "ligne d'horizon" comme une ligne à traiter qui viendra influencer le résultat:

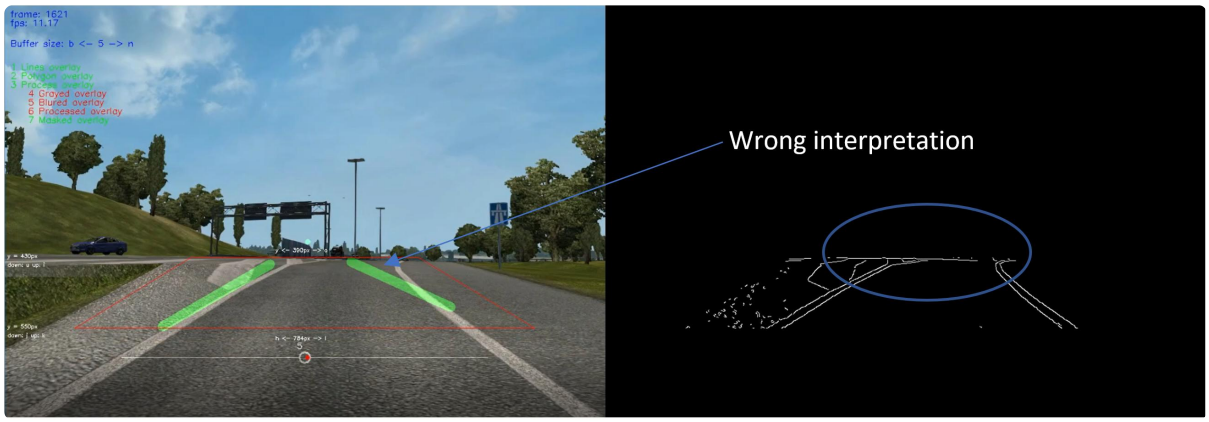

Figure 10 Erreur d'interpretation

Sur la figure précédente on voit que la ligne horizontale sur l'écran de droite influence le résultat de la moyenne temporelle de la détection de la bande de droite qui dévie significativement de la bande réelle.

Une des amélioration possible du programme serait une dynamisation de la forme du polygone en fonction des données à un instant t. Si sur un des côtés, une des bandes (ou autre élément parallèle à la direction) n'est pas détectée car trop en dehors de la zone de détection, le polygone pourrait grandir de ce coté pour tenter de voir si il n'y a pas quelque chose qui traîne un peu plus loin.

1.6.5. Interprétation des données

Les éléments 7, 8 et 9 de la **figure 4** sont une aide à l'interprétation du résultat du programme. Au centre de l'axe horizontal se trouve un cercle (élément 8) qui marque le milieux de la composante horizontale de l'écran. Le point rouge (élément 7) est simplement la projection du point de fuite (élément 6) sur cet axe. Ce point illustre l'intensité de la correction nécessaire pour suivre la trajectoire détectée. l'élément 9 donne une représentation numérique de cette même intensité. C'est cette valeur qui doit être traitée et transformé en input pour piloter un éventuel jeu.

Figure 11 Direction à suivre