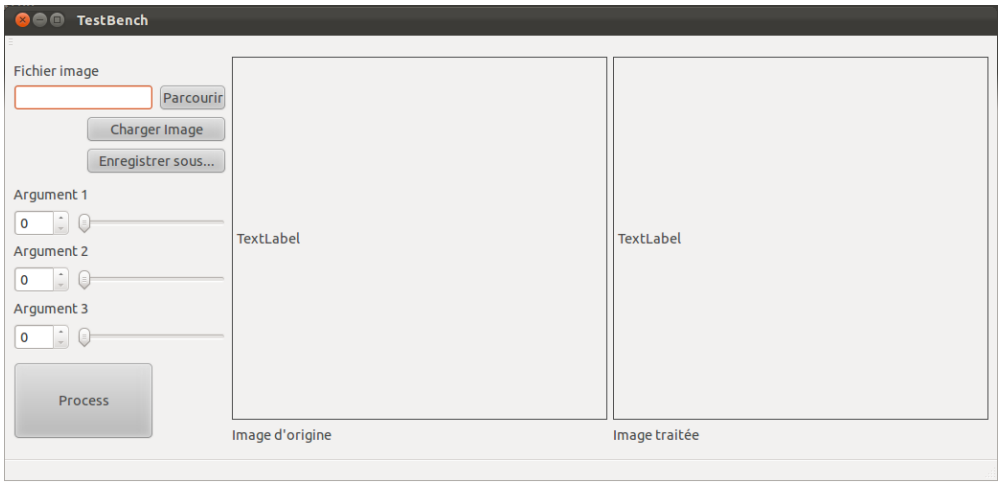
Traitement d'image

Initialement, le traitement d'image devait représenter la part la plus importante de notre projet, le but étant double : extraire de l'image un maximum d'informations sur la route, dans le but de les croiser avec les informations GPS, et fournir l'information de sortie destinée à l'utilisateur, à savoir les indications de directions par coloration de la route à emprunter. Malgré une motivation certaine, des recherches documentaires poussées sur les algorithmes déjà existants de détection de route nous ont fait comprendre qu'il s'agissait d'une tâche beaucoup plus compliquée que ce que nous pensions. Après quelques tentatives maladroites aux résultats très approximatifs, afin d'être sûrs que le programme final soit capable de détecter un virage, nous avons décidé pour ce faire de détecter sur un signal vidéo des panneaux de signalisation caractéristiques d'une intersection, que nous aurions nous-mêmes conçus et mis en place sur le bord de la route. Ainsi, la partie traitement d'image du projet s'est scindée en deux : d'une part la détection de panneaux de signalisation et d'autre part la recherche d'algorithmes de détection de route (que nous ne voulions absolument pas abandonner, car elle représente le véritable défi dans la conception d'un GPS en réalité augmentée).

Détection de route

La première étape de notre travail dans cette partie a été de mettre en place un outil d'aide au développement du traitement d'image en lui-même. Cela part du constat simple que l'algorithme de détection de la route allait se décomposer en une succession d'opérations plus ou moins simples mais dont le nombre de paramètres à fournir et ajuster pour chacune d'entre elles est relativement grand. Ainsi, tester et optimiser ces opérations et surtout la conjonction de ces opérations requiert un nombre de tests très important. C'est pour cela qu'il était nécessaire de commencer par coder une structure de banc de test dans laquelle implémenter les différents blocs algorithmiques avec les interfaces simples permettant la modification et l'enregistrement des paramètres, le chargement d'une image, l'affichage et l'enregistrement du résultat et le test. Le tout sans être contraint à chaque fois de recompiler le programme. La bibliothèque GUI (ou plutôt devrait-on dire le *framework*) utilisé pour implémenter ce banc de test est Qt.



Armés de cet outil, nous avons pu effectuer les premiers tests avec des fonctions simples.

Première tentative - extraction à partir d'un échantillon de texture :

La première idée que nous avons testé consistait à vouloir extraire la route en partant du principe qu'elle présente une texture relativement uniforme sur toute la surface de l'image. On pourrait alors partir d'un échantillon de cette texture de route que l'on prélève sur une partie de la photo pour essayer de la retrouver sur le reste de l'image. On peut même supposer la position de l'échantillon fixée car, la position de la caméra sur la voiture est fixe et qu'ainsi, un point proche du bas de la photo et légèrement décalé sur la droite pointe sur la route, en évitant l'éventuelle signalétique blanche. La probabilité de trouver la route y est donc proche de 1. A partir de cette échantillon, pour l'extraction en elle-même, deux idées plutôt simples nous sont venues. D'une part, un algorithme de backprojection de l'histogramme des couleurs de l'échantillon et d'autre part une sélection par distance de la couleur à la couleur moyenne de ce dernier.

La backprojection consiste tout d'abord à calculer l'histogramme de l'échantillon, c'est à dire de compter pour chaque couleur (ou pour chaque plage de couleur) le nombre de pixels de l'échantillon qui sont de cette couleur. On normalise ensuite ces valeurs puis, à chaque pixel de l'image totale on associe la valeur de l'histogramme associé à la couleur du pixel. Ainsi on obtient une matrice pour laquelle la valeur d'un pixel représente la probabilité d'appartenir à la même texture que l'échantillon. Enfin on applique un seuil sur ces valeurs, plaçant à 1 les pixels au-dessus d'une certaine probabilité, et à 0 les autres.

La sélection par distance consiste, elle, à mettre à 1 les pixels dont la distance dans l'espace des couleurs à la couleur moyenne de l'échantillon est inférieure à un certain seuil.

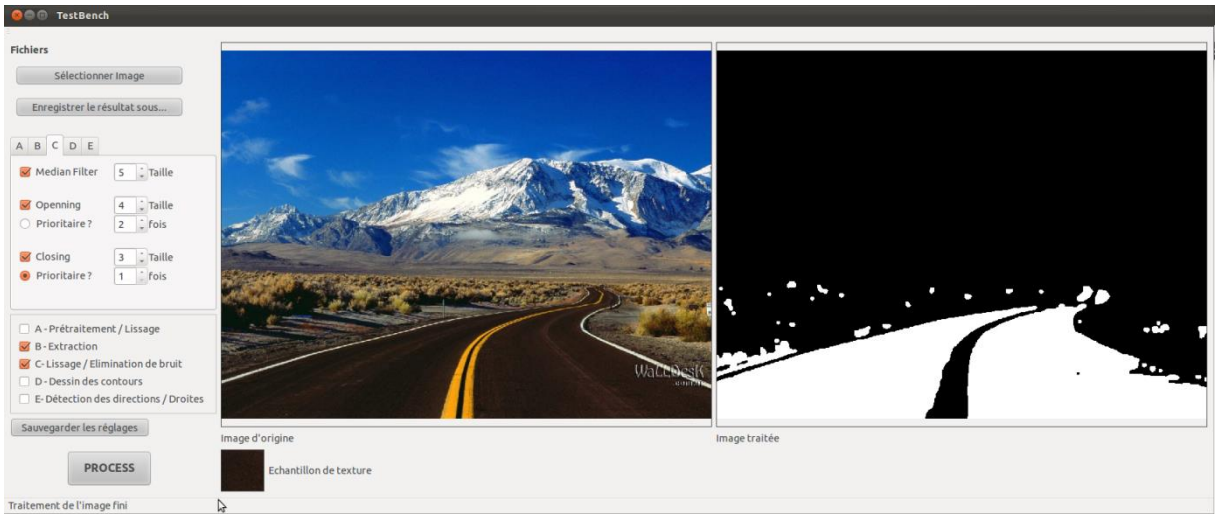
Ces deux méthodes sont applicables sur différents espaces de couleurs, donnant alors des résultats différents ! En effet, nombreuses sont les façons de représenter numériquement une couleur. La plus connue est le format RGB (Red Green Blue : trois valeurs correspondant aux valeurs des couleurs rouge, vert et bleu), mais on peut aussi passer l'image en niveaux de gris pour l'extraction, ou dans différents autres formats de couleurs que l'on ne détaillera pas ici.
Dès lors, rien que cette phase d'extraction nécessite de régler la méthode d'extraction, l'espace de couleur dans lequel on se place pour l'appliquer, le seuil associé (seuil de probabilité ou de distance), mais aussi la taille de l'échantillon et sa position sur l'image.

Seulement, la texture du bitume est parfois très granulaire et présente un fort gradient de couleur. Cela peut considérablement réduire l'efficacité de l'extraction. D'où l'idée, avant l'extraction d'une opération de lissage de l'image. Cela consiste, à affecter à un pixel soit la couleur moyenne, soit la valeur médiane d'un petit carré l'environnant. Les paramètres à régler ici sont le type de lissage (moyenne ou médiane) et la taille du filtre (ie la taille du petit carré).

Même avec ceci, l'image obtenue après extraction présente beaucoup de bruit : nombreux sont les pixels, plus ou moins isolés et en dehors de la route qui sont retenus et nombreux restent les pixels de la route qui n'ont pas été sélectionnés. Un second lissage s'impose, que l'on effectue par l'enchaînement ou non d'un filtre de la médiane, et d'opération d'openning et closing (que l'on ne décrira pas ici). Pour cette étape, il s'agit, pour optimiser les réglages, de déterminer quels filtres on applique, combien de fois, dans quel ordre et quelles sont leurs tailles.

Photos avant et après lissage

Néanmoins, cette méthode est loin d'être satisfaisante. En effet, les réglages obtenus détectent à peu près la route, mais de manière trop approximative. Des éléments n'appartenant pas à la route persistent, qu'il s'agisse de simples petites tâches ou que ce soit carrément une grosse partie du ciel. De plus colorer la route sans être capables d'en déduire des informations sur sa forme est largement insuffisant.

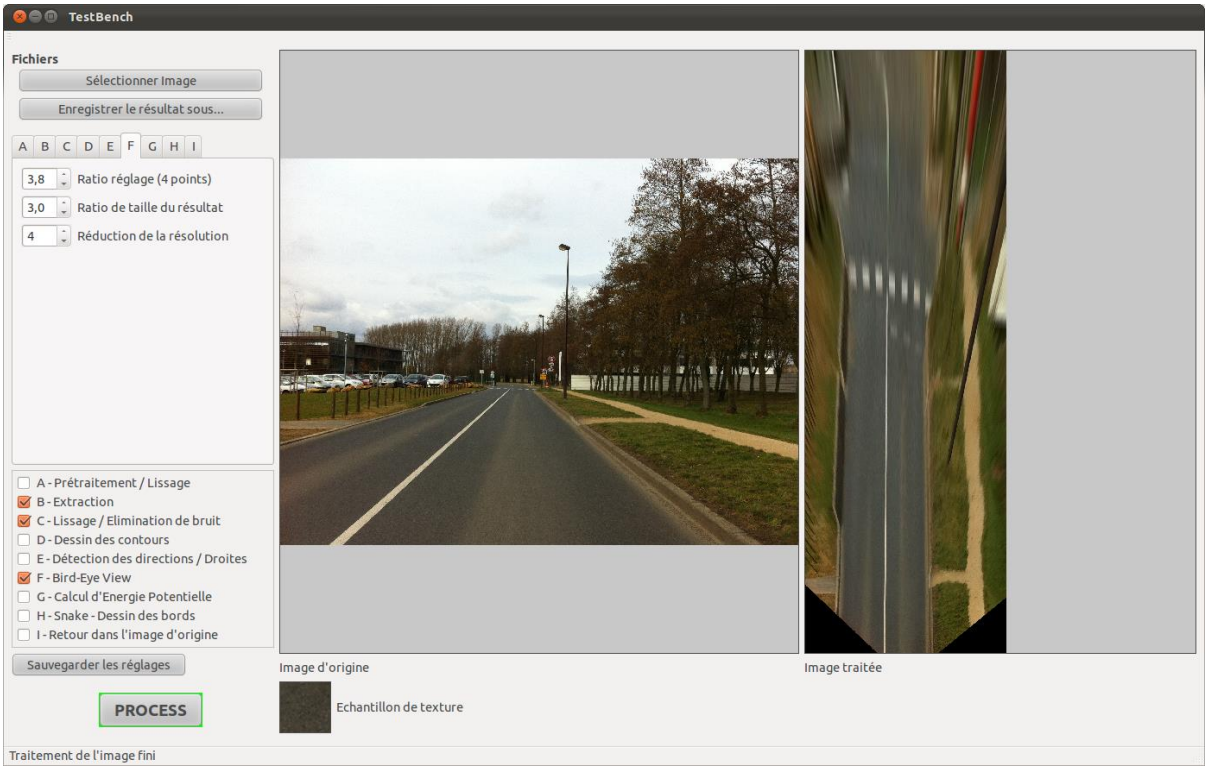


Enfin, nous nous sommes rendus compte de la difficulté d'optimiser les réglages autrement que photo par photo. Ainsi, la flexibilité de l'algorithme en souffre cruellement. Il parait également illusoire de vouloir automatiser l'optimisation des réglages, n'ayant pas de modèle précis et facilement vérifiable de la forme attendue pour les résultats.
Dès lors, force est de constater qu'il nous faut trouver des méthodes radicalement différentes.

<u>Seconde tentative - traitement après passage en *bird-eye view* :</u>

Une recherche documentaire supplémentaire nous met alors sur la voie de deux méthodes très utilisées pour la détection de route. Il s'agit de la détection du point de fuite et du passage en bird-*eye view*. La détection du point de fuite consiste en la détection des deux droites principales qui forment les bords de la route. Leur point d'intersection est alors appelé le point de fuite. La seconde méthode est le passage en *bird-eye view*, c'est à dire la transformation géométrique de l'image prise

par la caméra située au niveau du pare-brise pour donner une image en vue de dessus de la route, comme la verrait un oiseau en train de survoler la voiture.



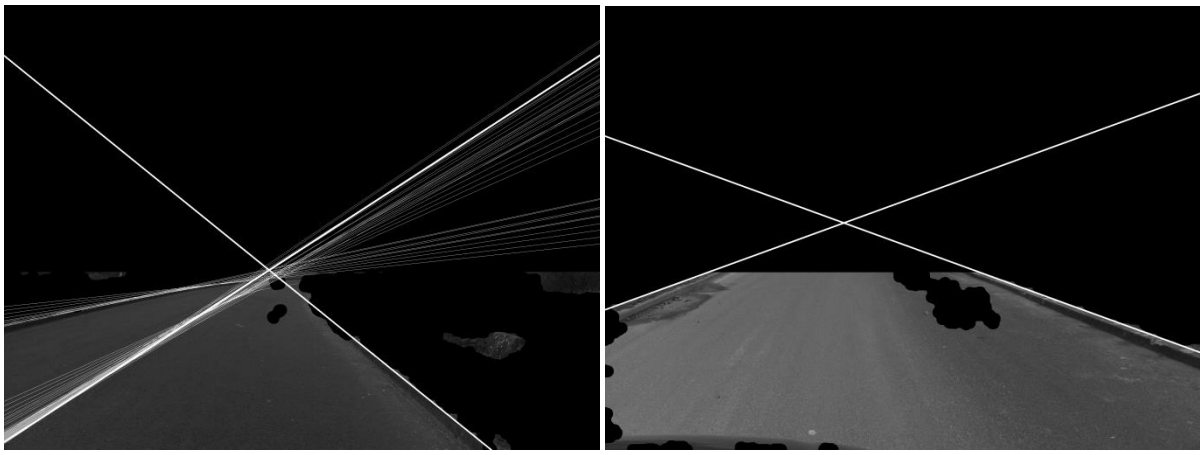
Le passage en *bird-eye view* nous a semblé être la meilleure solution. Il sera en effet bien plus facile de détecter la forme de la route et d'en extraire les caractéristiques à partir d'une "photo aérienne". En effet dans ce cas sa largeur est quasi-constante et sa direction quasiment verticale sur la photo. Cependant, si cette transformation est relativement facile à mettre en place avec la librairie OpenCV, elle nécessite d'avoir une connaissance assez clair de l'environnement dans lequel la photo a été prise. Pour se la représenter facilement, on peut se dire que cette transformation (une homographie du plan) envoie 4 points du plan sur 4 autres points. Pour la définir, il nous faut donc trouver 4 points de l'image d'origine dont on connait la position qu'ils auront sur l'image de sortie. On pourrait calculer ces quatre points à partir des caractéristiques précises de la position de la caméra sur la voiture, mais cette solution manque de flexibilité et ne permet pas de travailler sur un panel de photo qui n'ont pas toutes été prises de la même voiture dans les même conditions. Nous avons donc adopté une autre approche, et les 4 points d'origine seront, dans notre cas 2 points du bord gauche de la route, et leurs symétriques sur le bord droit. Leurs images après transformation forment alors un rectangle centré latéralement sur l'image de sortie.
On en déduit que pour connaître les caractéristiques de la transformation, et ainsi être en mesure d'obtenir la *bird-eye view*, il nous faut détecter les deux bords de la route, ou du moins leurs tangentes en des points proches de la voiture. Nous sommes donc revenus à la méthode du point de fuite, et ce dans le but de déterminer les réglages de la transformation en bird-eye view.

Détection des droites principales :

En vision assistée par ordinateur, la recherche de droites est une tâche assez répandue et qui jouit donc d'algorithmes standards déjà présents dans les bibliothèques telles qu'OpenCV. La détection de droite se fait de la manière suivante On commence par un algorithme d'extraction des contours

appelé Canny. Cet algorithme part d'une image en niveaux de gris, en calcule le gradient et en déduit les pixels qui appartiennent à des contours. Il prend en entrée deux paramètres entiers déterminant la quantité de contours retenus. A partir de cette image monochrome des contours, on lance un algorithme de transformée de Hough. C'est un algorithme classique dont je ne détaillerai pas le fonctionnement ici. Il prend en entrée un seuil et renvoie une liste de couples (rho, thêta) qui représentent l'angle polaire et la distance à l'origine de chaque droite détectée.
Le problème, c'est que dans une photo de route, la route est rarement le seul endroit de l'image qui présente des droites. Tout l'enjeu est alors de sélectionner les bonnes droites, ce qui n'est pas une mince affaire ! Nous avons alors eu l'idée d'effectuer une sélection sur l'image avant la détection des droites. Pour ce faire, nous avons réutilisé le travail effectué dans la démarche d'extraction à partir d'un échantillon de texture. En effet, en utilisant son résultat comme masque, que l'on complète en noircissant toute la partie supérieure de l'image (les trois cinquièmes supérieurs), on ne conserve pour la détection de droite quasiment que des zones de route. Même si ce premier filtrage n'est pas parfait il permet au moins de contre-sélectionner une grande partie de l'image. Dès lors, il ne reste "plus" qu'à sélectionner les deux bonnes droites.

Pour cela, nous sommes d'abord partis du principe que les bords de la file sur laquelle on roule sont les droites dont les angles sont les plus proches de la verticale, avec tout de même une plage d'angles que l'on juge trop proches (des droites verticales ou presque pourraient être détectées : il s'agit là de les éliminer). La largeur de cette plage est un paramètre réglable dans le TestBench. Néanmoins, cette approximation de la droite la plus proche angulairement de la verticale n'est pas entièrement satisfaisante car en pratique on observe souvent plutôt un faisceau de droites proches de celle-ci. La droite que l'on veut sélectionner est dans ce faisceau. L'opération de sélection consiste alors à extraire ce faisceau (on retient les droites dont l'angle est inférieur à 3° de plus que celui de la droite la plus proche). De ce faisceau classé angulairement, on retient celle qui se trouve à environ 1/3 du classement. Enfin, dans le cas où on ne trouverait pas de droite d'un côté de l'image et où on en détecte bien une de l'autre côté, on prend la droite symétrique par rapport à la droite verticale qui coupe l'image en deux.



On obtient des résultats plutôt concluants (voir photos ci-dessus) par cette méthode qui peut encore être grandement améliorée. Elle nous permet toutefois d'avoir de très bons résultats sur une petite série de photos prises sur le campus de l'Ecole et qui constitue le vivier d'images tests avec lesquelles nous optimisons les paramètres.

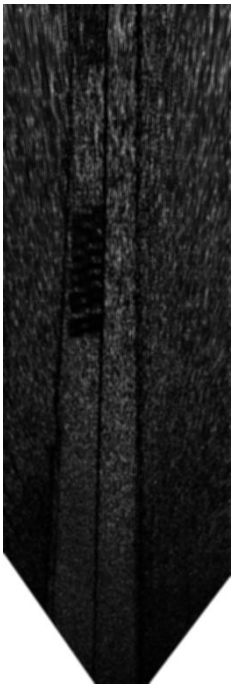
Passage en bird-eye view :

Une fois les bords de la route sont détectée, il est relativement facile à l'aide d'OpenCV d'effectuer la transformation. On choisit sur ces deux droites deux points sur la limite inférieure de l'image et deux points à une certaine hauteur un peu au-dessus. On choisit également la taille de l'image de sortie. On prend ensuite les 4 points d'un rectangle correspondant dont on peut régler les proportions afin de calculer l'homographie. Ce n'est qu'alors que l'on effectue la transformation. Les résultats sont plutôt impressionnants dès lors que la détection des deux droites principales a correctement fonctionné.

Cette vue de dessus est la base de tous les traitements et les algorithmes d'extraction d'information ultérieurement utilisés. Cette transformation étant réversible, il est facile de dessiner sur cette vue pour ensuite replacer ce dessin dans l'image originale. Par exemple, si on arrive à colorer sur la bird-eye view la route sur laquelle on roule, on n'aura aucun mal à replacer ce coloriage dans la vue initiale. C'est d'ailleurs l'objectif que nous nous fixerons par la suite.

Détection de la route sur laquelle on roule :

Il est évident que pour concevoir un GPS en réalité augmentée, il ne faut pas seulement détecter la route sur laquelle on se trouve, mais toutes les routes présentes sur l'image afin de déterminer et cartographier les intersections. Cependant, réussir à délimiter la route sur laquelle on roule semble être un bon début qui, vu le temps restant à ce niveau d'avancement du projet, restait le seul objectif potentiellement réalisable.
Après un rapide essai dénué de tout espoir, nous nous sommes rendus compte que la démarche d'extraction à partir d'un échantillon de texture était complètement vouée à l'échec dans ce cas. Il fallait alors complètement changer de stratégie, ne plus traiter indifféremment les pixels un à un mais essayer de suivre la route itérativement. Nous ne nous intéressons plus dès lors à la couleur en elle-même mais nous essayons de suivre les bords de la route. Le point de départ semble a priori évident : il s'agit du bord inférieur de l'image. Connaissant les caractéristiques de transformation en bird-eye view, nous avons connaissance des abscisses des deux points de départs sur cette bordure.
Il nous reste alors à trouver un moyen de suivre les bords de route. La caractéristique d'un contour (quel qu'il soit) est un changement rapide voire brutal de couleur. C'est ce qui nous amène donc à considérer le gradient de l'image. Dans un souci de simplicité, nous passons d'abord d'image en niveaux de gris, puis nous en calculons le gradient (étant une opération très fréquente en traitement d'image, ce calcul est directement implémenté par OpenCV).



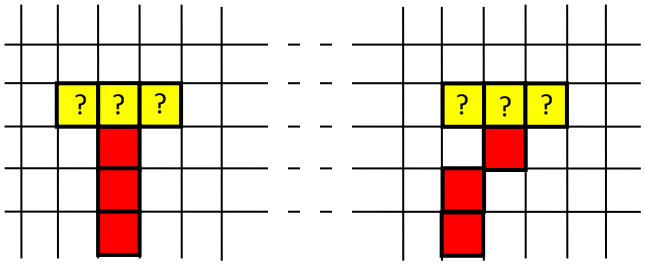
Reste alors à trouver un moyen de suivre ce chemin de plus fort gradient. Par analogie avec la physique, nous pensons à attribuer à chaque pixel une valeur d'énergie potentielle qui serait définie à partir de la norme du gradient calculé en ce point. Ainsi, suivre la bordure reviens plus où moins à suivre la vallée dans la carte d'énergie potentielle ainsi construite.
Nous utilisons la forme suivante de l'énergie potentielle (exemple ci-contre):

$$E = \frac{a}{1 + b.\|grad\|^2}$$

a et b sont des paramètres variables qui participent aux phases d'optimisation.

Puis on lisse la matrice obtenue avec un filtre en moyenne ou un filtre gaussien dont la taille est un paramètre réglable. Sans cela les fluctuations d'énergie potentielle seraient trop importantes en raison du bruit et des textures "granulaires".

L'étape suivante est de définir les règles qui permettent d'avancer tout en prenant en compte l'énergie potentielle. Partant du bas de l'image, on progresse ligne par ligne en autorisant les déplacements latéraux d'au maximum un pixel de chaque côté. Il y a alors 9 configurations d'évolution possibles à chaque étape. On considère alors l'énergie potentielle globale comme la somme de celles des deux pixels.



On pourrait simplement chercher parmi les 9 configurations celle qui minimise l'énergie, ce qui reviendrait à suivre indépendamment les deux vallées ligne par ligne. Cependant ce ne serait pas judicieux. En effet, ce serait ignorer le fait que l'on connait des caractéristiques de la route :

- sa direction est globalement la verticale, ce qui implique que les déplacement latéraux ne peuvent pas être trop brusques
- sa largeur reste relativement constante, ce qui implique que les variations de largeur ne peuvent être trop brusques, et ne peuvent dépasser un certain seuil.

Ainsi, on peut classer les neufs configurations en 4 catégories : la configuration constante (les abscisses des deux pixels restent les mêmes), les configurations de déplacement (-1 ou +1 pour les deux abscisses, les configurations de variation de largeur de 1 pixel et les configurations de variation de largeur de 2 pixel.

A partir de cela, pour formaliser les contraintes précédentes en une règle à suivre à chaque étape, nous avons interdit les variations de largeur de 2 pixels et nous avons définit 3 paramètres et des comportements associés :

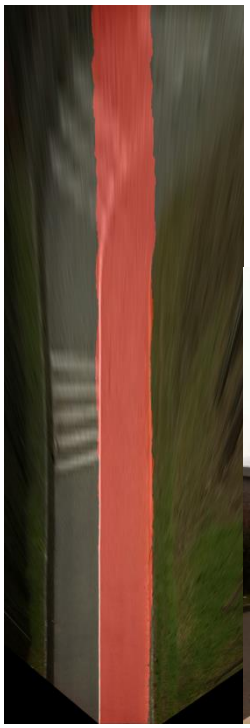
- une rigidité de déplacement s : on ne peut pas effectuer de déplacement pendant d-1 périodes après le dernier déplacement
- une rigidité de variation de largeur w : on ne peut pas effectuer de changement de largeur pendant w-1 périodes après la dernière en date
- un ratio de r de changement de largeur maximal : la variation de largeur relative à la largeur initiale ne doit pas dépasser r.

Il ne reste plus qu'à optimiser les réglages de ces paramètres et des paramètres entrant en jeu dans le calcul de l'énergie potentielle sur le vivier de 5 ou 6 images tests. Les résultats obtenus sont plutôt encourageants puisque la route est suivie d'assez près.

Cependant on remarque certains défauts : si la route tourne trop, on observe une "sortie de route", plus ou moins tardive selon les valeurs de s, w et r. De plus, la dégradation progressive de la qualité de l'image bird-eye view lorsque l'on s'éloigne du départ induit des écarts plus fréquents à la route. Même si cela n'améliore rien pour la détection de la route (visant à terme une cartographie des environs), on peut stopper les itérations lorsqu'on a atteint une certaine hauteur. Cela permet, lorsqu'on repasse dans l'image d'origine de ne colorer que le début de la route et produit un résultant plutôt probant (voir ci-dessous).








Finalement, même si ces résultats sont loin d'être parfaits, le programme réalisé constitue un début de recherche dans ce domaine et peut être perfectionné, tant au niveau de la détection du point de fuite (qui est un point assez fragile de la chaîne) qu'au niveau de la détection itérative de la route, qui n'en est qu'à ses balbutiements.