ESTIMATION DE RISQUES PAR GEO-LOCALISATION ET ANALYSE DU RESEAU ROUTIER

Maître de stage: Nicolas Turro
Tuteur: Jean-Guillaume Dumas
Responsable du parcours: Françoise Jung
Elève: Cyril Lamine

M1 MAI 2011

TABLE DES MATIERES

COORDONNEES

Lamine Cyril :
Add : 31avenue Gabriel Péri
38400 SMH
Tél. : 06.88.96.75.93
Mail : Laminec@e.ujf-grenoble.fr (mail de la fac je préfère que vous me joignez sur ce email en cas de besoin.)
cyril.lamine@inria.fr

Turro Nicolas:
Mail : nicolas.turro@inrialpes.fr

INRIA RHONE ALPES :

Add : INRIA Grenoble - Rhône-Alpes
Inovallée
655 avenue de l'Europe Montbonnot
38 334 Saint Ismier Cedex

Tél : 04.76.61.52.00
Fax : 04.76.61.52.52

PRESENTATION DE L'ENTREPRISE

INRIA :

"Qu'il s'agisse d'environnement, d'éducation, de santé, d'économie ou de recherche, l'INRIA tente de répondre aux questions de la société de demain."

Michel Cosnard *est président-directeur général de INRIA*

INRIA (institut national de recherche en informatique et automatique) est composé de plusieurs centres de recherches à travers la France. INRIA est composé de 5 grands pôles de recherches qui sont :

- STIC pour les sciences de la vie et de l'environnement.
- Mathématiques appliquées, calcul & simulation.
- Perception, cognition, interaction.
- Réseaux Systèmes et services, calcul distribué.
- Algorithmique, programmation logiciels et architectures.

CADRE DE TRAVAIL :

J'ai effectué mon stage à l'INRIA Rhône Alpes. INRIA se situe à Montbonnot. Pendant ce stage, j'ai intégré l'équipe SED. L'INRIA me subventionne pour les repas et les transports en commun. J'ai également accès à la bibliothèque ainsi qu'à des salles de sports.

PRESENTATION DE L'EQUIPE :

ArosDyn : "Analyse robuste de scènes dynamiques complexes par fusion bayésienne de données de capteurs. Application à la sécurisation de la conduite automobile"

Dans le cadre du projet ArosDyn, l'équipe SED doit développer un logiciel embarqué d'analyse robuste de scène dynamique de conduite automobile. Les algorithmes permettront le suivi et la détection de plusieurs corps mobiles dans la scène routière. Le logiciel devra comprendre des fonctions de prédiction et d'évaluation des risques lors de la conduite. Pour cela, différents capteurs seront utilisés : «un télémètre laser à balayage et un système de stéréovision. Ces deux capteurs offrent une grande complémentarité dans leurs caractéristiques. Ils permettent en outre une portée de détection suffisante pour des applications routières, même extra-urbaines. La perception proprioceptive est quant à elle déléguée à une centrale inertielle, des capteurs odométriques et un GPS»

Ce projet résulte d'un partenariat entre Toyota et l'INRIA. Toyota fournissant le véhicule sur lequel est embarqué le système.

Organisation : Les principaux acteurs de l'ADT ArosDyn sont les équipes-projets e-Motion, PERCEPTION, le service SED de l'INRIA Grenoble Rhône-Alpes et l'équipe-projet EVOLUTION de l'INRIA Sophia-Antipolis. La société Probayes et l'équipe-projet PRIMA de l'INRIA Grenoble Rhône-Alpes participent à cette ADT de manière plus ponctuelle. L'ADT ArosDyn est coordonnée par un comité de pilotage. Un comité technique assure le développement et la mise en œuvre des modules nécessaires pour accomplir les objectifs de cette ADT, qui se déroulera sur une période de trois ans. La réunion de démarrage de l'ADT ArosDyn a eu lieu le 27 janvier 2009.

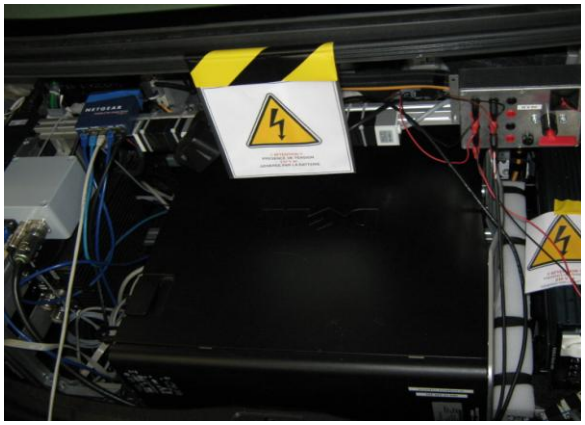
Texte tiré en partie de http://arosdyn.gforge.inria.fr/

MON LIEU DE TRAVAIL :

Je travaille dans le hall robotique. Le hall robotique est la salle où est entreposée la voiture Lexus. C'est dans cette voiture que mon programme devrait être intégré. Il y a aussi différents modules électroniques et robotiques pour d'autres équipes de travail. Il y a bien sûr d'autres postes de travail que le mien. Je travaille seul quand il n'y a pas d'expériences ou de mesures à faire par les chercheurs. Les chercheurs sont tous enthousiastes ; ce qui rend mon cadre de travail agréable. Mon maître de stage travaille souvent dans le hall robotique ce qui me permet de lui poser des questions quand j'en éprouve le besoin.

MON ENVIRONNEMENT DE TRAVAIL :

Je développe des programmes sur une machine que l'INRIA à mis à ma disposition. Je travaille sous l'OS Fedora (Linux) ce qui n'est pas vraiment déroutant pour moi. Évidemment j'ai accès à internet. Je dispose d'un login et d'un mot de passe. J'ai une adresse internet à l'INRIA.



REFORMULATION DU SUJET DE STAGE

ESTIMATION DE RISQUES PAR GEOLOCALISATION ET ANALYSE DU RESEAU ROUTIER

CONTEXTE

Dans le cadre du projet Arosdyn d'assistance à la conduite automobile présenté précédemment, je dois développer un programme d'estimation de risques d'accidents en fonction de la topologie routière. Ce programme fera partie du système embarqué dans la Lexus. Il devra permettre la prédiction de risque d'accident liée à la topologie routière. Cette prédiction sera liée uniquement aux coordonnées GPS du véhicule. Enfin ce programme devra rendre une estimation de risque.

La sortie du système complet (GPS + vision stéréo + laser) devra être soit une alerte pour le conducteur, soit une action sur la conduite du véhicule si un risque de collision élevé se présente.

OBJECTIFS DU PROJET/STAGE

1ère partie :

Grâce aux coordonnées GPS, je dois pouvoir déterminer sur quel tronçon de route se trouve le véhicule. Je dois surtout déterminer quels sont les « obstacles » proches qui représentent un risque et surtout quantifier ce risque. Pour cela je dispose de fichiers XML provenant d'Open Street Map.

Je dois pouvoir afficher la route et la position du véhicule avec le logiciel de visualisation Mapnik. Je pense qu'afficher les éléments dits « à risque » sur la carte Mapnik peut être intéressant sans que la carte soit surchargée bien sûr. Je dois pouvoir vérifier la direction dans laquelle le véhicule se déplace et à quelle vitesse aussi.

Le but étant de faire en sorte d'exclure toutes les données non « proches » pour optimiser le temps de recherche.

Je dois réaliser cela en C++. La maîtrise des fichiers XML est essentielle. Une partie mathématique tournera autour de la géo-localisation de la voiture, son sens de déplacement et les calculs de distances avec les autres objets à risques (tous fixes en théorie donc plus simples).

2ème partie :

Le module pourra être intégré dans un Framework plus global de l'application :

- En s'interfaçant avec le middleware Hugr pour produire ses entrées/sorties (le but étant de travailler sur une entrée de données dynamique).
- Je devrai également récupérer les données dans une base de données OpenStreetMap afin d'avoir accès à plus de données qu'avec un simple fichier XML (le recours à une API spécifique s'imposera probablement car il est possible de recréer un serveur local OpenStreetMap contenant toutes les données du site.)

ORGANISATION

Cette partie présente l'ordre dans lequel le stage devait se dérouler et comment il s'est déroulé en réalité. En premier sera développé l'ordre logique dans lequel les tâches doivent être accomplies et ensuite, nous finirons par les diagrammes de Gantt : prévisionnel et final.

DEROULEMENT DE MON STAGE

Réalisation de la bibliographie et prise en main du logiciel

- Réalisation d'une étude comparative sur les différentes bases de données cartographiques disponibles.
- Prise en Main du Format OSM (OpenStreetMap).
- Prise en main des librairies de visualisation :
  - Mapnik
  - Qt

Implémentation des différents objectifs de la première partie du stage.

- Affichage de la carte désirée dans l'interface Mapnik.
- Affichage de la position du véhicule à partir de données GPS sous forme longitude et latitude en degrés. Ainsi que sa direction et sa vitesse sous forme d'un vecteur.
- Détection et affichage de la route sur laquelle se trouve le véhicule.
- Détection du type de voie sur laquelle le véhicule est (nationale, autoroute, route de campagne et autres) ainsi que des éléments importants en matière de danger sur le réseau routier.
- Une fois les obstacles détectés, il faut afficher et calculer la distance avec les obstacles.
- Calcul du risque d'accident.

Expérimentation

- Prise en main d'un Middleware Hugr
- Interfacer l'afficheur Mapnik avec Hugr.
- Récupérer la carte sur une distance correctement choisie en fonction de la position du véhicule.

Documentation et oral

- Rédaction du rapport final au cours du stage.
- Mise en page, fusion des différentes parties du rapport, relecture.
- Préparation de la soutenance orale.

DIAGRAMME DE GANTT PREVISIONNEL

Semaine 1 à 3 Du / au /	Semaine 4 à 6 Du / au /	Semaine 7 à 9 Du / au /	Semaine 10 à 12 Du / au /	Semaine 13 à 15 Du / au /
Etude comparative des BD de cartographie et rédaction de cette partie pour le rapport final Temps 1 semaine	Recherche type de route, point clef du réseau routier. Et affichage Temps 1 semaine	Tests et ajouts de certaines fonctions supplémentaires si besoin. Temps 1 semaine	Interfacer le middleware avec l'interface graphique Mapnik Temps 2 semaines	Nettoyer le code l'optimiser si possible. Test intensif. Rapport sur cette partie. Temps 2 semaines
Installations des packages Prise en main des librairies. Découvertes du code Temps 1 semaine	Calcul des distances avec les éléments et calcul du risque. Temps 1 semaine	Documentation découverte et prise en main du middleware Débuter le rapport sur cette partie Temps 2 semaines	Obtention de la carte (correctement choisie) à partir des données osm complètes Temps 1 semaine	Préparation de la soutenance orale. Réalisation du support de présentation PowerPoint. Reprise du rapport final. Finition des tests reprise du code si besoin (probable) Temps 2 semaines
Affichage de la carte, du véhicule, vitesse, direction. Recherche du tronçon de route actuel Temps 1 semaine	Nettoyer le code, optimiser si possibles et rapport de cette partie Temps 1 semaine			

DIAGRAMME DE GANTT FINAL

Semaine 1 à 3 Du 18/04 au 6/05	Semaine 4 à 6 Du 9/05 au 27/05	Semaine 7 à 9 Du 30/05 au 17/06	Semaine 10 à 12 Du 20/06 au 8/07	Semaine 13 à 15 Du 11/07 au 29/07
Etude comparative des BDD de cartographie et rédaction de cette partie pour le rapport final Temps 1 semaine	Recherche du type de routes, et autres points clefs du réseau routier. Et affichage dynamique via un thread Temps 1 semaine	Nettoyage du code à nouveau. Création d'un serveur osm via XML installation d'osmosis. Traitement de la carte pour l'alléger. Choix d'une sous carte Temps 1 semaine	Vérification des données existantes dans la BDD. Création d'une requête. Augmentation des données de la base. postGIS plus approprié pour nous dans le plug-in Temps 1 semaine	Suppression de la requête spécifique dans le plug-in et création à la place des deux nouveaux layer via le fichier XML Portage du code sur le véhicule Temps 1 semaine
Installations des packages Prise en mains des librairies. Découverte du code Temps 1 semaine	Utilisation du plug-in, nettoyage du code, mise en ligne sur svn, Correction de bug divers Temps 1 semaine	Installation de BDD postGIS et osm2pgsql. Remplissage de la BDD via XML. Reprise du code pour avoir un code sans affichage graphique. Conversion des données via proj4. Premier essai de l'interface avec la BDD Création d'un thread permettant la mise à jour régulière de la sous carte Temps 2 semaines	Rédaction du rapport et de la présentation PowerPoint quasi intégrale. Amélioration de la précision du code. Création d'une sous image a partir d'une image de la sous carte. Temps 2 semaines	Portage du code sur le véhicule. Mise au propre du code. Interfaçage du code avec HUGR. Création d'un fichier de test pour HUGR. Création d'une sortie des dangers pour HUGR. Test du code en temps réel dans MontBonnot. Préparation de la soutenance et finition du rapport et du PowerPoint. Temps 2 semaines
Affichage de la carte, du véhicule, vitesse et direction. Recherche du tronçon de route actuel Temps 1 semaine	Affichage des dangers sur un label via un thread. Recherche de la prochaine route. Optimisation et reprise d'algorithmes Temps 1 semaine			

COMPARAISON DES DIFFERENTES BASES DE DONNEES DE CARTOGROPHIE ROUTIERE

Dans un premier temps, il faut réaliser une comparaison entre les différentes bases de données de cartographie. Ceci nous permettra de choisir la meilleure base de données (la plus adaptée à notre problème). Pour réaliser cette comparaison nous nous appuierons sur quatre points bien particuliers :

- Possibilités d'accès aux informations.
- Cadre légal de leurs utilisations.
- Contenu des données.
- Format et documentation associé.

OPEN STREET MAP

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

C'est un logiciel libre donc gratuit où tout le monde peut récupérer ce qu'il veut mais aussi ajouter. L'obtention des données est simple, on peut les récupérer directement à partir de la carte mondiale en utilisant le menu export :

- Sous format OSM (XML). Format qui semble le plus intéressant pour nous
- Sous format image JPEG, PNG et PDF.
- Sous format image Mapnik.
- Récupérer avec du HTML.

L'accès aux données peut se faire également via différentes API (voir glossaire)

CADRE LÉGAL DE LEUR UTILISATION

Le logiciel est libre ainsi que sa base données donc son utilisation est possible pour tous et dans tous les cas.

CONTENU DES DONNÉES

La base de données dispose d'informations très complètes. Nous disposons du réseau routier avec la connaissance de points spécifiques tels que les arrêts de tramway et de bus, les ronds points, le nombre de voies parallèles, la circulation à double sens ou non, les péages, les échangeurs, les passages piétons, les passages à niveaux entre voie ferrées et routes, le type de route (nationales, autoroutes, départementales, etc...).

- OSM a obtenu le support de Mapquest qui permet une recherche d'itinéraire efficace. Il est également possible d'avoir accès à Cloud Made qui permet de récupérer uniquement les points d'un parcours donné. Après vérification il est possible de rattacher ces points à des points pris sur la carte OSM.

- Il existe également OpenRouteService (non libre) qui calcul également des itinéraires via OSM (identique à cloud Made).
- OSM a obtenu les fonds de carte aérienne de Bing "Microsoft".

Open Street Map est une sorte de Wikipédia de la cartographie. Il est actuellement en plein essor. Actuellement il dispose de 300 000 contributeurs donc son contenu est en constante augmentation.

FORMAT ET DOCUMENTATION ASSOCIÉS

Le format que nous retiendrons est le format .OSM (XML). Les fichiers sont crées avec simplement trois types de départ des "node", des "way" et des "relations".

Les balises XML permettant de nommer les types de way ou node sont libres ce qui permet une grande flexibilité pour la création. Bien sûr, il y a un standard déjà mis en place mais on peut créer un nouveau nom sans restriction. Le standard mis en place est bien documenté et facilement accessible sur leur site. Le format des fichiers .osm sont transposables sous d'autres plateformes de cartographies. Par exemple, ils sont visualisables sous GEOportail.

Les points obtenus par Cloud Made sont dans un fichier au format GPX qui est format standard pour les GPS et donc très bien renseigné.

GEO PORTAIL / IGN

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

Il est possible d'obtenir des jeux de test gratuit des différentes bases de données de l'IGN. Cet accès est présent sur leur site internet. L'obtention des données peut être faite pour différents logiciels qui sont :

- IGN MAP
- GEO-CONCEPT
- MAP INFO
- ARC VIEW

Évidemment, ceci implique différents formats de fichiers qui seront explicités plus loin.

CADRE LÉGAL DE LEUR UTILISATION

Les jeux de tests sont utilisables par tous et pour tous semble-t-il ; néanmoins les cartes complètes font l'objet d'un copyright. Par conséquent seuls les chercheurs du domaine public et les étudiants ont le droit de télécharger les cartes complètes gratuitement pour travailler dessus à condition que ce soit à but non lucratif bien sûr. Pour obtenir le droit d'accès à l'intégralité des cartes, il m'aurait fallu remplir une fiche avec attestation d'un enseignant puis les renvoyer par la poste etc... (Comme les données ne semblent pas correspondre à notre attente, je ne l'ai pas fait).

CONTENU DES DONNÉES

Seules les données de la base de données CARTO seront explicitées puisque c'est la base de données la plus complète proposée pour les réseaux routiers.

Cette base de données comprend les réseaux routiers sous forme de points et de tronçons de route. Les réseaux ferrés sous forme de points et de tronçons de voie ferrée. Il y a également les limites administratives, les aérodromes, les tronçons hydrographiques, les zones occupées au sol.

FORMAT ET DOCUMENTATION ASSOCIÉS

- Le format des fichiers associés à Map Info est .MID/.MIF.
- Les formats des fichiers associés à IGNMAP sont .AVL / .DBF / .LYR / .PRJ / .SHP / .SHX .
- Les formats des fichiers associés à ARCVIEW sont les même.
- Les formats des fichiers associés à GEO-CONCEPT sont les .GTX / .GCM / .GCR / .SBL .

Les fichiers les mieux documentés sont ceux sur le .MID/MIF. ils sont destinés au logiciel Map info. Nous disposons de la documentation complète sur ce format de fichier. Cette documentation est disponible sur

http://reference.mapinfo.com/software/mapinfo_pro/english/8.5/MI_UG.pdf.

Grâce à cela, il serait possible de travailler avec les données des cartes IGN mais elles semblent bien moins complètes sur les points qui nous intéressent. C'est-à-dire les points d'intérêts (notés POIs par la suite) du réseau routier, tels que les feux tricolores, les ronds points et autres.

GOOGLE MAPS

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

Il est possible uniquement d'utiliser des API pour récupérer les images de Google Maps. Néanmoins, il est impossible d'accéder directement à la base de données à proprement parler. Autrement dit, il est impossible de récupérer les métadonnées sur la voirie.

CADRE LÉGAL DE LEUR UTILISATION

La licence n'autorise pas la récupération de données autres que celles générées par l'utilisateur avec bien sûr des limitations. Dans le cas où ces limitations sont respectées, alors il est possible d'utiliser les données utilisateur sur n'importe quel site libre d'accès (c'est à dire sans login et password)

CONTENU DES DONNÉES ET FORMAT ET DOCUMENTATION ASSOCIÉS

Inconnus puisque que nous ne pouvons qu'utiliser mais pas accéder à leur base de données. Par conséquent nous n'avons pas accès aux métadonnées de Google Maps

MAPPY

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

Il est possible uniquement d'utiliser des API pour récupérer les images de Mappy. Néanmoins il est impossible d'accéder directement à la base de données à proprement parler. Autrement dit il est impossible de récupérer les métadonnées sur la voirie.

CADRE LÉGAL DE LEUR UTILISATION

ARTICLE 8 – RESTRICTIONS D'USAGE

L'utilisateur s'engage, pendant la durée d'utilisation des services, à ne pas utiliser l'API MAPPY: dans une application de navigation. Est considérée comme application de navigation, toute application/service qui inclut un guidage temps-réel, c'est à dire toute information, fournie à l'utilisateur final en fonction de sa position actuelle, sur l'action qu'il doit entreprendre à l'approche d'un point de décision sur son trajet. En général, la navigation inclut des capteurs, un processus de recalcule automatique en cas de déviation de trajet, l'affichage des cartes.

« Condition d'utilisation Mappy »

CONTENU DES DONNÉES ET FORMAT ET DOCUMENTATION ASSOCIÉS

Inconnus puisque que nous ne pouvons qu'utiliser mais pas accéder à leur base de données. Par conséquent, nous n'avons pas accès aux métadonnées de Mappy.

TELEATLAS

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

Le coût des cartes nécessaires se révèle très important.

CADRE LÉGAL DE LEUR UTILISATION

Une fois une carte achetée avec la licence adéquate, il est possible de faire ce que l'on veut avec. TeleAtlas est le principal fournisseur de GoogleMaps.

CONTENU DES DONNÉES

TeleAtlas MultiNet est une base de données cartographique qui regroupe l'intégralité des réseaux de transport conjugués aux limites administratives et à l'occupation du sol. Adapté à l'ensemble des logiciels SIG du marché, MultiNet propose jusqu'à 8 thèmes majeurs :

- Limites administratives et postales
- Réseau routier, franchissements et intersections
- Numéros d'adresses postales aux intersections et carrefours
- Logiques et attributs de circulation
- Réseau ferré
- Hydrographie
- Habillage (occupation su sol)
- Toponymie, équipements et Points d'intérêts (POI)

http://www.intercarto.com/cms/bases-de-donnees-sig/bd-routieres/teleatlas-multinet.html

FORMAT ET DOCUMENTATION ASSOCIÉS

.DBF / .PRJ / .SHP / .SHX. Les fichiers étant très nombreux cela peux sembler chaotique, mais il existe une nomenclature qui permet de connaître l'utilité de chacun d'eux.

Leur contenu semble idéal pour nos travaux.

NAVTEQ

POSSIBILITÉS D'ACCÈS AUX INFORMATIONS

Il faut acheter les cartes voulues à un prix inconnu (23 000 euros pour la France selon un forum) L'accès aux prix et autres données passe par une identification plus que précise. Les mises à jour de cartes pour GPS sont facilement accessibles bien que l'on ignore ce qu'elles contiennent (si ce sont bien les données NAVTEQ NAVSTREETS).

J'ai réussi à obtenir un exemple des fichiers NAVTEQ sur la ville de Luxembourg sous forme de données Oracle. Néanmoins il est impossible de savoir si se sont des données NAVSTREET ou non.

CADRE LÉGAL DE LEUR UTILISATION

Une fois une carte achetée avec la licence adéquate son utilisation est totalement disponible. L'utilisateur final a aussi le droit de rajouter ce qu'il veut dans la base de données qu'il possède.

CONTENU DES DONNÉES

Les données NAVSTREETS contiennent :

- Signes (panneaux)
- Z-niveaux
- Manœuvres de conduites
- Codes d'emplacement de trafic
- Routes (autoroutes) majeures et accès
- Routes (autoroutes) secondaires et accès
- Chemins de fer

- Zones, frontières administratives
- Océans, îles, polygones de voie navigable et segments
- Caractéristiques d'utilisation de terre
- Polygones de construction/point de repère
- 16 point d'intérêts/couches
- Métadonnées
- Rues (jeu complet d'attributs)

Des attributs supplémentaires trouvés dans la couche de Rues incluent :

- Catégorie de vitesse
- Emplacement de diviseur
- Direction de voyage
- Accès automobiles
- Accès bus
- Accès taxis
- Accès aux parcs automobiles
- Accès piétons
- Accès camions
- Accès par trafic
- Accès livraisons
- Accès pour les véhicules de secours
- Figure de trafic spéciale
- Indescriptible
- Manœuvre
- Diviseur légal
- Codes de trafic

FORMAT ET DOCUMENTATION ASSOCIÉS

Les fichier NAVSTREETS sont disponibles au format .SHP, .TAB(= MID/MIF format MapInfo) ou encore en format de base de données oracle.

AUTRES

Les bases de données suivantes ont été rapidement exclues pour différentes raisons, telles que non recouvrement du territoire français ; ou aucune donnée ni aide ne sont accessibles ; cela est dû au fait que la base de données est privée :

TIGER,ViaMichelin, FREE vector map.

NOTRE CHOIX

Nous avons donc décidé d'utiliser Open Street Map pour plusieurs raisons. Premièrement, il semble contenir toutes les données que l'on veut. De plus, dans le cas où des informations seraient manquantes, il est possible que nous les ajoutions nous-mêmes. L'autre point important est que l'accès aux données dans leur intégralité est simple et gratuit. Enfin, le format de données XML est simple à comprendre.

LES PROJECTIONS

Nous avons vu précédemment différents types de données et nous avons choisi de travailler avec les données Open Street Map (OSM dans la suite). De ce fait, nous disposons de données géographiques sous forme latitude, longitude. De même, nous disposons de données GPS provenant de la Lexus qui sont sous cette forme. Néanmoins ces données sont, en fait, difficilement interprétables pour un humain. Il nous faut donc les convertir en quelque chose de plus facilement interprétable comme l'utilisation des distances en mètres ou en kilomètres.

Il existe plusieurs manières de représenter un point sur la planète. En général nous voulons avoir des cartes planes, car c'est plus simple en raison des supports que nous utilisons. En effet que ce soit du papier ou un écran d'ordinateur ; ils sont plats dans la plupart des cas. Par conséquent, il faut un moyen de projeter l'ellipsoïde, qu'est notre planète, sur un plan. Pour cela il existe une multitude de projections différentes.

Une projection permet donc de convertir des données dans une autre forme (parfois plus facilement interprétable). Je ne présenterai ici que la projection de Mercator et la projection UTM (voir glossaire) basée sur le système géodésique WGS84. Ces projections permettent d'obtenir des données en mètres plutôt que des angles et cela sur une surface plane.

Pour repérer un point sur la planète tout le monde utilise les coordonnées géographiques (latitude, longitude). La latitude et la longitude sont en fait deux angles qui permettent à partir de l'équateur et du méridien zéro (le méridien de Greenwich) de se repérer sur la planète comme si elle était une sphère.

- La latitude est une mesure angulaire s'étendant de 0° à l'équateur à 90° aux pôles.
- La longitude est donc une mesure angulaire sur 360° par rapport à un *méridien de référence*, avec une étendue de -180° à +180°, ou respectivement de 180° ouest à 180° est.

Nous trouvons ces coordonnées sous deux formes distinctes couramment utilisées. La première étant des angles en degrés décimaux. Dans leurs secondes formes, les angles sont donnés sous la forme degrés minutes secondes (que l'on nomme degrés sexagésimaux).

Exemple : l'INRIA Rhône-Alpes se trouve à

- En degrés décimaux : latitude = 45,217679 longitude = 5,806614.
- En degrés sexagésimaux : latitude = 45° 13' 3,644" longitude =5° 48' 23,8104"

Les méthodes de conversion sont les suivantes :

Degrés décimaux ⟹ Degrés sexagésimaux.

Formulation générale :

Latitude ou longitude en degrés décimaux = degrés + $\frac{minutes}{60} + \frac{\text{secondes}}{3600}$.

45° 13' 3,644" = 45 + $\frac{13}{60} + \frac{3,644}{3600}$ = 45+ 0 ,2166666 + 0,00101233 =45,21767(9)

Degrés décimaux ⟸ Degrés sexagésimaux.

Soit une latitude de 45.217679°

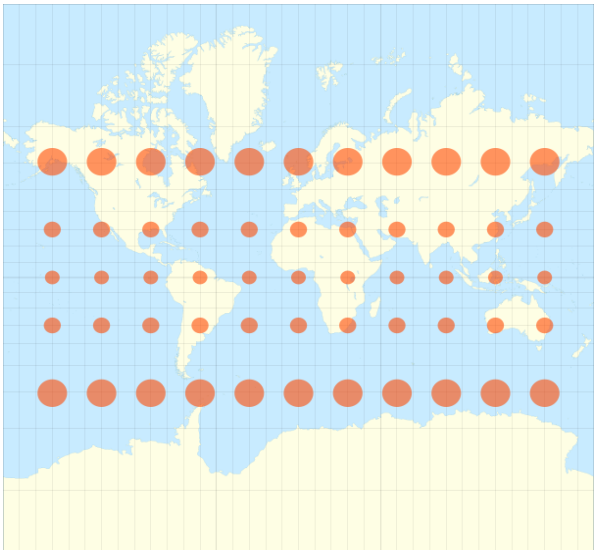
1. Le nombre avant la virgule indique les degrés => 45°
2. Multiplier le nombre après la virgule par 60 => 0, 217679 * 60 = 13,06074
3. Le nombre avant la virgule devient la minute (13')
4. Multiplier le nombre après la virgule par 60 => 0, 06074 * 60 = 3,6444
5. Le résultat correspond aux secondes (3,6444").
6. Notre longitude sera de 45° 13' 3,644"

Le GPS qui équipe la Lexus fournit des coordonnées géographiques sous forme de degrés décimaux. Il faut maintenant trouver un moyen de pouvoir placer ces données sur une carte. Je dois donc construire la carte. Je ne dispose en fait que d'un fichier .osm avec les coordonnées géographiques de tous les points qui forment le réseau routier et le cadastre. Donc, si je projette tous les points de la même façon, j'aurai une carte cohérente. Néanmoins selon la transformation employée, je risque de perdre certains rapports, notamment les distances. Par exemple, une projection plane standard serait une erreur. De plus, les points qui sont dans le fichier OSM sont convertis automatiquement par Mapnik. Par conséquent, il m'a fallu trouver la bonne transformation.

LA PROJECTION DE MERCATOR

Au cours de mon stage, j'ai d'abord travaillé avec une projection de Mercator. La projection de Mercator est une projection cylindrique tangente à l'équateur du globe terrestre sur une carte plane. La projection de Mercator conserve les angles donc elle est une projection conforme.

« L'inévitable étirement Est-Ouest en dehors de l'équateur est accompagné par un étirement Nord-Sud correspondant, de telle sorte que l'échelle Est-Ouest est partout égale à l'échelle Nord-Sud. Une carte de Mercator ne peut couvrir les pôles : ils seraient infiniment hauts. »
http://fr.wikipedia.org/wiki/Projection_de_Mercator



Voici une image issue de la même référence qui illustre cette explication de manière simple et efficace.

LES FORMULES

$$x = \lambda - \lambda_0$$

$$y = \ln\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right)$$

$$y = \frac{1}{2}\ln\left(\frac{1 + \sin\varphi}{1 - \sin\varphi}\right)$$

$$y = \sinh^{-1}(\sin\varphi)$$

$$y = \tanh^{-1}(\sin\varphi)$$

$$y = \ln(\tan(\varphi) + \sec\varphi)$$

Les formules donne la position x et y (position sur une carte de Mercator) en fonction de λ et φ qui ne sont autres que la latitude et la longitude. λ_0 Le centre de la carte en latitude.

Les formules de la fonction inverse sont les suivantes.

$$\varphi = 2\tan^{-1}(e^{y}) - \frac{1}{2}\pi$$
$$\varphi = \tan^{-1}(\sinh(y))$$
$$\lambda = x + \lambda_0$$

EXEMPLE D'APPLICATION

Dans le cadre de mon projet, j'utilise la librairie Qt pour faire l'affichage graphique de ma carte et cela passe par une pixmap. Je dispose donc d'un tableau de $n \times m$ (n la largeur et m la hauteur) case on peut voir cela comme une carte de taille $n \times m$. Par conséquent, je dois pouvoir placer n'importe quel point en fonction de cette taille. Pour l'exemple, je suppose que je projette le cadre suivant dans la carte (latmin, lonmin) ; (latmax, lonmax).

Ce que l'on veut déterminer c'est la position x et y du point en pixel. Pour cela, on utilise les formules ci-dessous :

$$x = \frac{n}{360} \times \left(\left(\lambda - \frac{(latmin + latmax)}{2}\right)\right)$$

$$y = \frac{m}{2} - \ln\left(\tan\left(\frac{\pi}{4} + \frac{\text{radian}\varphi}{2}\right)\right) \times \left(\frac{\text{n}}{2\pi}\right)$$

EMPLOI DES PROJECTIONS VIA DES «BOITES NOIRES» AU COURS DE MON STAGE

J'ai implanté ces calculs pour faire la conversion des données GPS. Il se trouve que le résultat n'était pas celui attendu car je m'étais trompé de projection. J'ai en fait, trouvé comment ne pas avoir à implanter les calculs ; il m'a suffi de faire appel à quelques méthodes Mapnik pour que la conversion se fasse seule. Il suffit de créer un objet Mapnik coordtransform(n, m, taille de la carte). Et puis d'utiliser la méthode forward d'un de ces objets. J'utilise donc une transformation par défauts sans la connaître. En fait, elle est spécifiée dans ficher osm.

Plus tard, lors du changement de plug-in de OSM vers postGIS ; il m'a fallu à nouveau trouver comment convertir des données géographiques en données UTM avec wgs84 (voir glossaire). Ce fût obligatoire, car les fonds de carte .shp que j'emploie dans cette partie sont dans ce format de projection. Là aussi, j'ai essayé d'implanter les calculs à la main (des calculs assez fastidieux avec des développements limités à l'ordre 6). Là encore, impossible de trouver la transformation exacte utilisée par Mapnik. Néanmoins, en utilisant la librairie Proj4 et en connaissant les deux transformations sous leur forme suivante. J'ai pu convertir les données au bon format.

La projection UTM
proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs
La projection géographique avec le système géodésiques WGS84
proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
Note :
La projection UTM et ses calculs sont développés dans la partie annexes

PARTIE TECHNOLOGIQUE

MAPNIK

Mapnik est une librairie implantée en C++ et en Python qui permet d'afficher des données cartographiques. C'est une librairie open source qui peut générer des cartes sous format pnb ou même PDF, mais surtout elle possède une interface graphique permettant de visionner une carte à partir des métadonnées. C'est pourquoi, nous avons décidé d'utiliser mapnik car une grande part du travail était déjà réalisée.

L'interface graphique s'appelle le viewer. Le viewer est une interface Qt4. Dans cette interface graphique, les fonctions de translation de la carte à la souris ou du zoom et de quelques autres actions sont déjà implantées. Il est également possible de charger certains fichiers et bien sûr d'imprimer une carte puisque c'est son but premier.

Mapnik fait appel à diverses librairies, qui sont toutes en open source également. Par exemple, Proj4 qui permet de faire des calculs de projection assez compliqués. La librairie AGG pour que le dessin ait un meilleur rendu. Le viewer Mapnik permet surtout la lecture des données de cartographie de plusieurs sortes. Ceci se traduit par l'existence de plusieurs plug-in. Durant de mon stage, j'aurais travaillé avec deux d'entre eux. Tout d'abord, le plug-in OSM qui permet de lire un fichier OSM directement. Puis avec le plug-in PSQL. Celui-ci permet donc de lire les métadonnées dans un BDD PostGIS

Mapnik a besoin d'un fichier XML de rendu pour fonctionner. L'affichage étant déterminé par ce fichier si l'on veut que toutes les autoroutes soient roses, il suffit de le spécifier à l'intérieur. Chaque fichier XML est attribué à un fichier OSM, ou une BDD, différente. Donc si l'on veut avoir le même rendu pour deux cartes différentes il faudra quand même deux fichiers XML.

LES CALQUES (LES LAYERS)

Les layers sont les containers des différentes données dont nous disposons. Ils sont importants pour nous car il nous faudra réaliser les algorithmes de recherche dessus.

L'affichage de Mapnik est une superposition de layer. Il fonctionne sur le même principe que le célèbre adobe Photoshop. C'est-à-dire qu'il utilise des layer (calque par analogie) dans lequel il stocke différentes données qu'il affichera. Par exemple, il peut y avoir un layer pour les routes, un layer pour les autoroutes et un layer avec les POIs. La superposition des trois donne une carte routière relativement complète bien que les tunnels, les ponts et autres ne soient pas inclus.

L'avantage de ce système est de pouvoir facilement afficher ou non différents layers. Le Viewer possède un panneau qui permet de choisir ce que l'on veut afficher ; par exemple, nous pouvons désactiver les rivières, les zones militaires, les parcs d'attractions, les bois ainsi que les bâtiments afin de faciliter la lisibilité de la carte.

Les layers sont définis par une classe dans le code Mapnik. Le problème étant souvent de savoir quel est le numéro du layer qui contient les informations que nous voulons. Parcourir toutes les données de tous les layers prend un temps considérablement long.

Il est possible de connaître leurs numéros quand on utilise toujours le même style d'affichage (toujours le même fichier xml voir plus bas).

Conseil : il est possible de voir le numéro d'un layer dans la console en le désactivant dans le viewer

FICHIER OSM

Les fichiers .osm sont en fait une extension du type de fichier XML. Ils fonctionnent donc évidemment sous la forme de système de balise. Ce qui est vraiment essentiel est le système de tag qui permet d'attribuer une fonction différente à chaque balise de type node. Il existe aussi des balises de type way qui sont en fait des ensembles de node qui permettent de définir une route. Il est possible d'ajouter un tas d'informations par exemple, une route peut avoir :

- Une vitesse maximale
- Un type
- Un nom
- Sens de circulation unique ou non
- le nombre de voies de circulation

Cette liste est non exhaustive.

Voici quelques exemples de ce qui composent un fichier .OSM :

```
<node id="442070349" lat="45.2176163" lon="5.8075529" user="gottferdom"
                uid="200426" visible="true" version="2" changeset="5730021"
                timestamp="2010-09-09T06:53:26Z">
        <tag k="highway" v="bus_stop" />
        <tag k="name" v="INRIA" />
</node>
```

Note : ici le nœud numéro 442070349 est défini par ses coordonnées lat, lon. Puis d'autres informations inutiles à décrire pour nous. Ensuite, on voit que ce nœud possède un tag qui le déclare en tant qu'arrêt de bus et le nom de l'arrêt est INRIA

```
<node id="583340214" lat="45.2214062" lon="5.8128599" user="gottferdom"
                uid="200426" visible="true" version="2" changeset="5541291"
                timestamp="2010-08-20T07:01:16Z">
        <tag k="highway" v="traffic_signals" />
</node>
```

Note : ici le nœud numéro 583340214 est défini par ses coordonnées lat, lon. Ensuite on voit que ce nœud possède un attribut (un tag) qui le déclare en tant que feu tricolore.

```
<way id="41179173" user="gottferdom" uid="200426" visible="true" version="4"
                        changeset="7063569" timestamp="2011-01-23T16:39:24Z">
  <nd ref="502961782" />
  <nd ref="502961784" />
  <nd ref="502961785" />
  <nd ref="502961786" />
  <nd ref="858866258" />
  <nd ref="502961787" />
  <tag k="highway" v="residential" />
  <tag k="maxspeed" v="15" />
  <tag k="name" v="Lotissement les Étoiles" />
  <tag k="oneway" v="yes" />
  <tag k="source" v="cadastre-dgi-fr source : Direction Générale des Impôts - Cadastre ;
mise à jour : 2009" />
  </way>
```

Note : ici la voie numéro 41179173 est composée de différents nœuds dont les nœuds 502961782 et 502961784. Nous pouvons voir ici que cette voie a pour nom : le lotissement des étoiles. Cette voie est à sens de circulation unique, de type résidentiel et que la vitesse maximale autorisée est de 15 Km/h

Ci-dessous un nœud qui appartient à la voie. On peut donc constater qu'il ne possède aucun tag (en dehors du nom de la source absolument inutile pour nous comme pour l'affichage de n'importe quelle carte).

```
<node id="502961782" lat="45.2139821" lon="5.8079445" user="gottferdom"
                uid="200426" visible="true" version="5" changeset="6592624"
                 timestamp="2010-12-09T08:04:59Z">
        <tag k="source" v="cadastre-dgi-fr source : Direction Générale des
  Impôts - Cadastre. Mise à jour : 2009" />
</node>
```

FICHIER XML ASSOCIE AU RENDU MAPNIK

Ce fichier permet de reconnaître les balises du fichier .osm et leur assigne une couleur et un style de tracé (pointillé, continu, largeur du trait, image et autre). Il est facilement personnalisable lorsque l'on travaille sur une carte assez simple, mais devient très vite un travail de longue haleine. Le fichier xml que j'utilise avec ma base de données postGIS fait plus de 6000 lignes ; mais est généré automatiquement grâce à Mapnik-stylesheet. J'ai essayé de modifier certaines choses sur le tout premier fichier que j'avais. Celui-ci était très court. Cela m'a permis de vérifier que je comprenais bien le fonctionnement du fichier. J'ai par exemple mis des images .png sur chaque point d'intérêt dans ma toute première version.

Voici un exemple simple et commenter de fichier XML.

Déclaration de la projection employée

Ici on déclare que pour chaque parking nous utiliserons une image parking.png avec les tailles associées (16x16)

Création d'un style. Ici on défini tout ce que comporte le tag aménity

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Map>
<Map bgcolor="#d0d0d0" srs="+proj=latlong +datum=WGS84">
    <Style name="amenity">
        <Rule>
            <Filter>[amenity]='parking'</Filter>
            <PointSymbolizer file="./parking.png" type="png" width="16"
height="16"/>
        </Rule>
        <Rule>
            <Filter>[highway]='bus_stop'</Filter>
            <PointSymbolizer file="./bus_stop.png" type="png" width="16"
height="16"/>
        </Rule>
        <Rule>
            <Filter>[highway]='traffic_signals'</Filter>
            <PointSymbolizer file="./feu_tricolor.png" type="png" width="16"
                                                    height="16"/>
        </Rule>
    </Style>

    <Style name="roads-casing">
        <Rule>
            <Filter>[highway] = 'residential' or [highway] = 'service'</Filter>
            <LineSymbolizer>
                <CssParameter name="stroke">#999</CssParameter>
                <CssParameter name="stroke-width">4</CssParameter>
                <CssParameter name="stroke-linejoin">round</CssParameter>
                <CssParameter name="stroke-linecap">round</CssParameter>
            </LineSymbolizer>
        </Rule>
        <Rule>
            <Filter>[highway] = 'secondary' or [highway] = 'secondary_link'</Filter>
            <LineSymbolizer>
                <CssParameter name="stroke">#a37b48</CssParameter>
                <CssParameter name="stroke-width">6</CssParameter>
                <CssParameter name="stroke-linejoin">round</CssParameter>
                <CssParameter name="stroke-linecap">round</CssParameter>
            </LineSymbolizer>
        </Rule>
    </Style>
```

Idem ici on définit le style des routes de type 'residential'

```
        <Layer name="roads" status="on" srs="+proj=latlong +datum=WGS84">
                <StyleName>residential</StyleName>
                <StyleName>secondary</StyleName>
                <Datasource>
                        <Parameter name="type">osm</Parameter>
                        <Parameter name="file">map.osm</Parameter>
                </Datasource>
        </Layer>
        <Layer name="amenity" status="on" srs="+proj=latlong +datum=WGS84">
                <StyleName>amenity</StyleName>
                <Datasource>
                        <Parameter name="type">osm</Parameter>
                        <Parameter name="file">map.osm</Parameter>
                </Datasource>
        </Layer>
</Map>
```

Ici on crée un layer, en fonction de la projection, qui regroupera les différents types de routes.

On déclare ensuite le fichier source dans lequel chercher et son type. Dans notre cas avec postGIS on utilise des requêtes (voir plus, à la fin de la partie postGIS)

OSMOSIS, TRAITEMENT DES DONNEES POUR LES ALLEGER

Pour réaliser mes tests, j'utilise les données OSM de toute la région Rhône-Alpes. Les données de la région sont sous forme d'un fichier xml de plusieurs millions de lignes. Ce fichier pèse 184.7 MB lorsqu'il est compressé avec BZ2. Décompressé, il représente plus de 1.3 GB (je ne connais pas la taille exacte car mon système refuse de le décompresser en entier pour de raison de place mémoire). Évidemment cela représente une quantité de données énormes à traiter pour nos calculs.

Lorsqu'on effectue une recherche avec les données sous forme de fichier xml cela prend trop de temps, car chaque recherche nécessite de parcourir le fichier en entier. Pour éviter ce problème, nous avons donc décidé de changer de format de données initiales. Le fichier xml a été remplacé par une base de données PostGIS. De cette manière, il est possible de créer des requêtes qui ne travaillent que sur un sous-ensemble précis de la base de données.

La première solution pour alléger le coup de calcul, est de ne s'intéresser qu'à un périmètre restreint autour de notre véhicule. Il faut que le programme fasse appel à son Plug-in postGIS pour charger les données en mémoire. Cela prend du temps ; par conséquent, il ne faut pas le faire trop souvent. Il faut donc trouver un bon compromis entre dans la taille du périmètre à prendre et le temps de chargement et de traitement des données. Il faut que la carte soit assez grande pour ne pas avoir à être rechargée trop régulièrement mais assez petite pour qu'elle ne contienne pas trop de points.

OSM contient une quantité énorme de données qui sont absolument inutiles pour notre projet, par exemple les bâtiments. Les bâtiments sont représentés par tout un ensemble de points et sont en fait très coûteux pour nous. Grâce à Osmosis nous allons pouvoir ne conserver que les données qui nous intéressent.

A partir du fichier rhone-alpes.osm.bz2, nous allons créer trois fichiers distincts :

- Un qui ne contient que les routes, en général (route, nationale, autoroute ...).
- Un qui ne contient que les points d'intérêts dont le tag est amenity.
- Un qui ne contient que les points d'intérêts dont le tag est highway.

Pour cela, j'ai utilisé les commandes :

- *bzcat ./../rhone-alpes.osm.bz2 | ./osmosis --read-xml enableDateParsing=no file=/dev/stdin --tf reject-relations --tf accept-ways highway=* --used-node*
- *bzcat ./../rhone-alpes.osm.bz2 | ./osmosis --read-xml enableDateParsing=no file=/dev/stdin --tf reject-relations --tf accept-nodes amenity=* --tf reject-ways --wx file="map_POI_amenity .osm"*
- *bzcat ./../rhone-alpes.osm.bz2 | ./osmosis --read-xml enableDateParsing=no file=/dev/stdin --tf reject-relations --tf accept-nodes highway=* --tf reject-ways --wx file="map_POI_amenity .osm"*

Puis avec l'option merge on fusionne ces fichiers pour en obtenir un seul. Les commandes nécessaires pour cela sont :

- *./osmosis --rx map_POI_amenity.osm --rx map_POI_amenity.osm --rx map_Route.osm --merge --merge --wx map_alleger_2.osm.*

Une fois réalisé nous obtenons un fichier de la région qui ne pèsent "que" 379.5 MB on a donc divisé sa taille par au moins 4.

OSM2PGSQL ET LES MODIFICATIONS APPORTEES

Grâce aux logiciels osm2pgsql il est possible de remplir une base données postGIS en une ligne de commande. Néanmoins, il m'a fallu essayer plusieurs transformations pour trouver la bonne. J'ai également modifié le fichier default_style pour pouvoir avoir en données complémentaires la vitesse maximale autorisée.

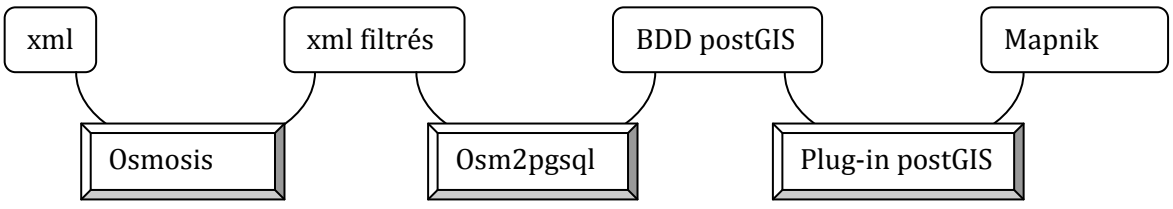
Le logiciel osm2pgsql est connu pour réaliser de nombreux pré-calculs qui optimisent l'emploi de la base de données qu'il va créer. Néanmoins, il supprime aussi une part des données car le but est au départ d'utiliser cette base de données pour réaliser une "carte du monde" par conséquent il a des informations (jugée non intéressante pour le rendu d'une carte) qui sont tronquées via le fichier default_style.

Osm2pgsql réalise aussi la conversion des données lat, lon qui sont dans le xml dans d'autres projections, par défaut, il se trouve que c'est la projection dite 900913 dans le fichier 900913.sql. Il existe une quantité de projections très importantes. J'en ai essayé plusieurs avant de me rendre compte que c'était celles-ci que je devais conserver, car les fonds de carte shp que j'ai récupéré sont dans ce format aussi.

La ligne de commande appropriée est donc :

- *./osm2pgsql -S default.style --slim -d gis -C 1024 /chemin-vers-le-fichier/map_alleger_2.osm*

Note : il est possible de remplir la base de données postGIS avec Osmosis, mais cela change en partie la structure des tables et surtout la table de références spatiale ce qui ne m'arrangeait pas.



POSTGIS

PostGIS est un complément (plugin) qui active la manipulation d'informations de géométrie (points, lignes, polygones) par le SGBD PostgreSQL,(voir glossaire) conformément aux standards établis par l'Open Geospatial Consortium. Il est utilisé par les systèmes d'information géographiques.
Le nom provient de la contraction de PostgreSQL et de GIS (acronyme anglais de SIG). En bref, PostGIS permet le traitement d'objets spatiaux dans les serveurs PostgreSQL, autorisant le stockage en base de données pour les SIG, un peu comme le SDE de ESRI ou l'extension spatiale d'Oracle.
http://fr.wikipedia.org/wiki/PostGIS

RESUME DES COMMANDES IMPORTANTES POUR INSTALLER UNE BASE DE DONNEES POSTGIS

sudo -u postgres -i
createuser username # answer yes for superuser
createdb -E UTF8 -O username gis
createlang plpgsql gis
exit

sudo -u postgres –i
psql -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql -d gis
psql -f ~/bin/osm2pgsql/900913.sql -d gis
exit

cd chemin-vers-osm2pgsql

./osm2pgsql -S default.style --slim -d gis -C 1024 /chemin-vers-le-fichier/map_alleger_2.osm

LES TABLES IMPORTANTES, LEURS LIENS ET LES REQUETES QUE J'AI CREES

La table planete_osm_line contient toutes les routes tandis que la table planete_osm_point contient tous les POIs. Il existe une table planete_osm_roads, mais elle ne contient que les avenues principales. Nous préférerons donc la table planete_osm_line pour nos recherches.

Les clefs primaires des deux tables les plus importantes pour nous (cité dans le paragraphe précédent) sont « osm_id » et « way ». Ces tables contiennent toutes les deux les mêmes champs. Exemple avec une table complète :

```
Table "public.planet_osm_point"
      Column      |   Type   | Modifiers
------------------+----------+-----------
 osm_id           |  integer |
 access           |  text    |
 addr:housename   |  text    |
 addr:housenumber |  text    |
 addr:interpolation |  text  |
 admin_level      |  text    |
 aerialway        |  text    |
 aeroway          |  text    |
 amenity          |  text    |
 area             |  text    |
 barrier          |  text    |
 bicycle          |  text    |
 brand            |  text    |
 bridge           |  text    |
 boundary         |  text    |
 building         |  text    |
 capital          |  text    |
 construction     |  text    |
 covered          |  text    |
 culvert          |  text    |
 cutting          |  text    |
 denomination     |  text    |
 disused          |  text    |
 ele              |  text    |
 embankment       |  text    |
 foot             |  text    |
 generator:source |  text    |
 harbour          |  text    |
 highway          |  text    |
 historic         |  text    |
horse             |  text    |
 intermittent     |  text    |
 junction         |  text    |
 landuse          |  text    |
 layer            |  text    |
 leisure          |  text    |
 lock             |  text    |
 man_made         |  text    |
 military         |  text    |
 motorcar         |  text    |
 name             |  text    |
 natural          |  text    |
 oneway           |  text    |
 operator         |  text    |
 poi              |  text    |
 population       |  text    |
 power            |  text    |
 power_source     |  text    |
 place            |  text    |
 railway          |  text    |
 ref              |  text    |
 religion         |  text    |
 route            |  text    |
 service          |  text    |
 shop             |  text    |
 sport            |  text    |
 surface          |  text    |
 toll             |  text    |
 tourism          |  text    |
 tower:type       |  text    |
 tunnel           |  text    |
 water            |  text    |
 waterway         |  text    |
 wetland          |  text    |
 width            |  text    |
 wood             |  text    |
 z_order          |  integer |
 way              |  geometry |
Indexes:
    "planet_osm_point_index" gist (way)
  "planet_osm_point_pkey" btree (osm_id)
```

J'utilise donc deux requêtes différentes pour obtenir ce dont nous avons besoin. C'est-à-dire les routes, avec leur nom, les vitesses maximales autorisées ; si elles ont un sens de circulation unique ou non, si ce sont des ronds-points. Mais aussi les pistes cyclables, les voies piétonnes, les arrêts de bus et les feux de signalisation.

Ces requêtes sont en fait placées dans le fichier XML associé au rendu Mapnik. De cette manière, je crée deux nouveaux layers qui contiendront uniquement ce que je veux pour effectuer mes recherches et qui n'ont aucun affichage associé. Cela me permet donc de ne pas avoir à implanter des requêtes en dur dans le plug-in Mapnik pour postGIS.

```
<Layer name="My_roads" status="on" srs="&osm2pgsql_projection;">
   <StyleName>My_roads</StyleName>
   <Datasource>
     <Parameter name="table">
( select way, osm_id,  maxspeed, highway ,amenity, name, oneway, route, denomination, junction  from
&prefix;_line where highway is not null) as My_roads
          </Parameter>
     &datasource-settings;
   </Datasource>
</Layer>
```

Requête pour les routes

```
<Layer name="My_amenity" status="on" srs="&osm2pgsql_projection;">
   <StyleName>My_amenity</StyleName>
   <Datasource>
     <Parameter name="table">
(select way, highway, name, amenity from  &prefix;_point ) as My_amenity
          </Parameter>
     &datasource-settings;
   </Datasource>
</Layer>
```

Requête pour les POIs

Commentaire :

Il m'a fallu beaucoup d'essais pour arriver à ce résultat. Au début j'ai travaillé directement dans le plug-in. Dans les plug-ins, les requêtes avaient des formes bien plus complexes. De plus, j'ai essayé de tout faire en seule requête ; mais le programme ne le supportait pas. Puis, après divers essais, je me suis rendu compte que le programme lisait les requêtes qui sont dans le XML pour les convertir en ce qu'il veut. J'ai donc décidé de mettre les requêtes sous cette forme beaucoup plus lisible, et qui ne nécessite pas de modifier le plug-in. Cela évite de le recompiler et de devoir le remplacer à son emplacement d'installation.

Conclusion :

Désormais nous disposons de tous les outils pour détecter et afficher les éléments routiers proches du véhicule. Cette méthode est extrêmement plus rapide et surtout beaucoup plus précise pour nous que les requêtes déjà existantes dans le fichier XML.

Il est tout à fait possible d'en créer de nouvelle pour obtenir plus de choses comme les stations service, les écoles et autres choses qui peuvent être considérées comme des risques d'accidents pour le conducteur.

HUGR

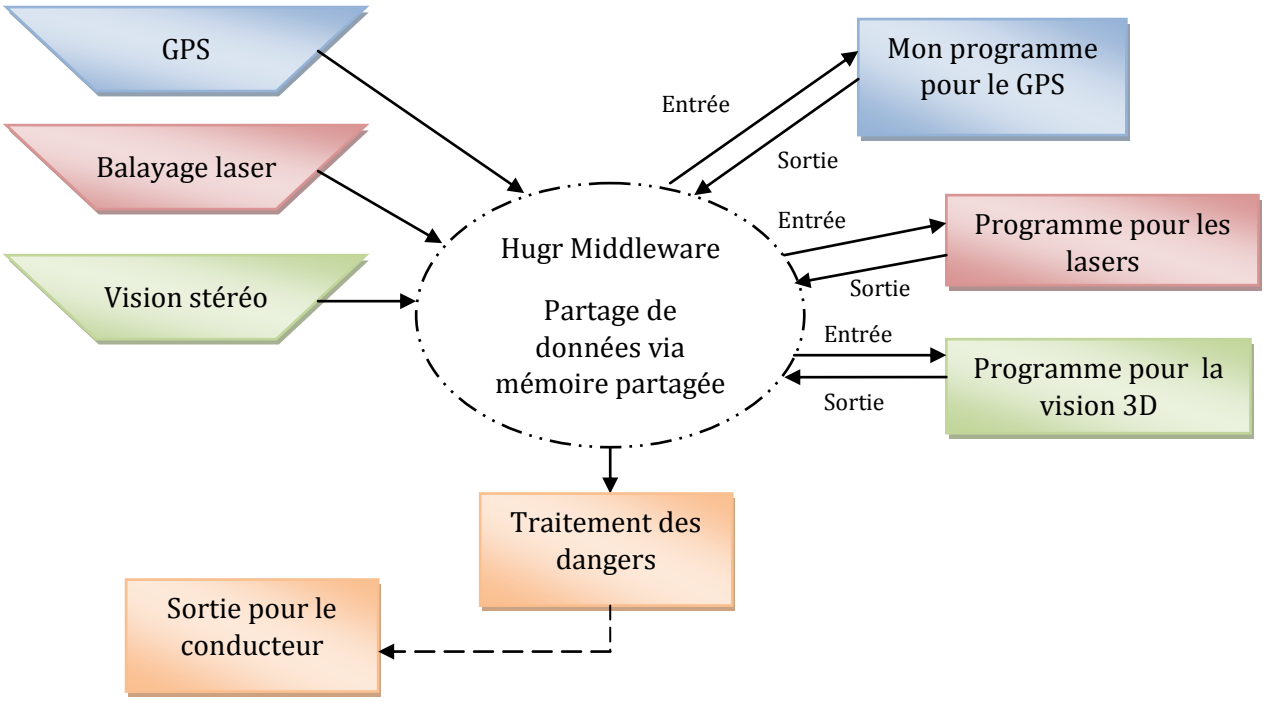
Hugr est le middleware qui doit interfacer mon programme avec le GPS mais les autres capteurs avec leur application respective. C'est un middleware de communication. Il permet donc l'échange de variables via une mémoire partagée. Il autorise entre autres possibilités :

- Un partage de variables de type tableau noir (mémoire partagé/socketUDP).
- Datation des données à chaque écriture.
- Enregistrement des données pour pouvoir les rejouer plus tard.
- Possibilité de synchronisation : lectures bloquantes.
- Orienté objet (C++).
- Open source (LGPL).

Outil présent dans Hugr :

- Hugr shell.
- Interface web.
- Utilitaire d'enregistrement et de relecture.
- Partage des données sur le réseau.

Schéma descriptifs simple :



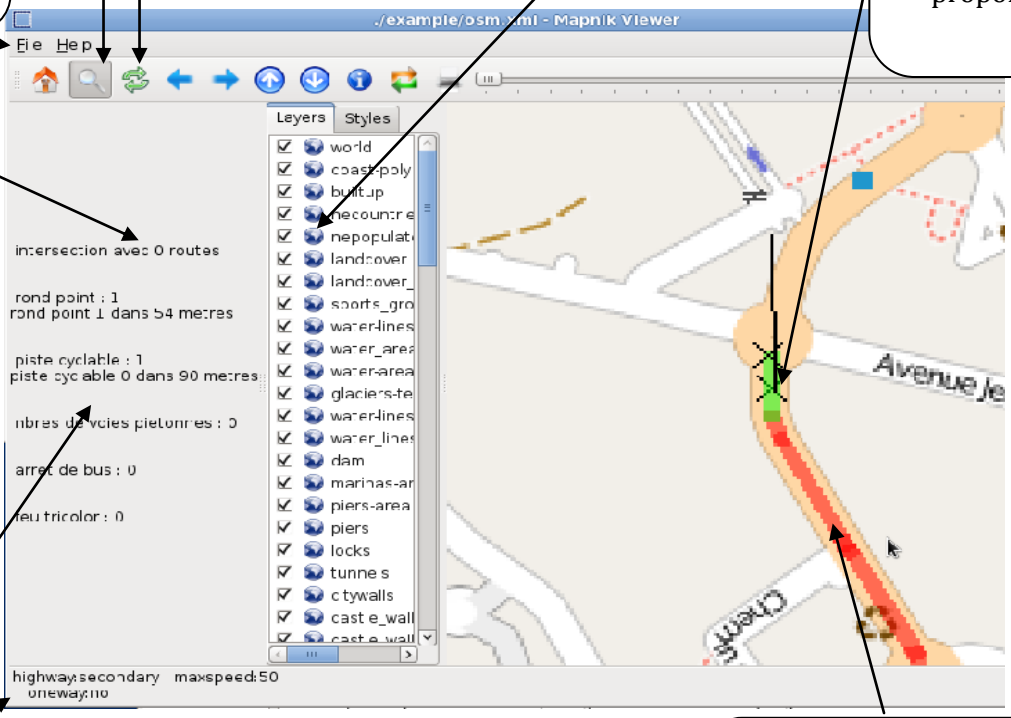
Concrètement, Hugr permet de récupérer les données obtenues par les capteurs de la Lexus (GPS, vision stéréo 3D, et télémètre laser). Puis, il les partage avec les programmes qui en ont besoins. Chaque programme doit retourner une évaluation de dangers et enfin ce danger doit être transmis au conducteur s'il est trop grand.

DEVELOPPEMENT ET CONCEPTION

Cette partie présente ma contribution directe au projet. Elle contient donc les explications des différents algorithmes, une présentation de l'affichage graphique et un schéma représentant l'exécution du code

AFFICHAGE SUPPLEMENTAIRE :

PRESENTATION DU VIEWER MAPNIK REMODELE POUR NOTRE UTILITE.



Barre de menu avec accès au chargement, à l'impression

Bouton de zoom

Bouton de translation

Panneau permettant de désactiver un layer de l'affichage

Affichage de la position actuelle et du vecteur de déplacement dont la longueur est proportionnelle à la vitesse

Label contenant les informations de dangers chiffrés

Compte à rebours des différents dangers qui ont été trouvés (en mètres)

Affichage du type de route, de la vitesse maximale et de la circulation à sens unique

Affichage de la route actuelle et sens de déplacement du véhicule (il se dirige vers la partie verte)

AFFICHAGE DE LA POSITION DU VEHICULE

Pour afficher la position du véhicule, je dispose en entrée de ces coordonnées GPS en degrés avec beaucoup de chiffres significatifs. Exemple : Latitude= 45.21701132 Longitude= 5.805734417

La totalité des affichages se font via les méthodes de Qpainter qui utilisent un Qpixmap(voir glossaire). L'emploi d'un Qpixmap justifie la nécessité de l'emploi d'une projection voir l'analogie entre carte et tableau de pixel décrit dans la partie PROJECTION.

J'ai d'abord réalisé une projection linéaire par mes propres moyens, néanmoins la projection n'est pas exacte étant donné qu'elle ne conserve ni les angles ni les distances. Puis, j'ai implanté une fonction faisant appel aux fonctions mapnik déjà existantes. En utilisant la bonne structure de données, il existe une méthode qui convertie les coordonnées GPS en Pixel (plan OXY).

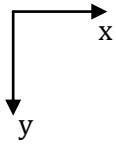
Au début, le point qui représentait le véhicule se déplaçait sur la carte.
Mon maître de stage m'a tout de suite demandé de modifier cela et de faire en sorte que le véhicule soit constamment le centre de l'image à chaque instant. Cette demande était motivée pour deux raisons

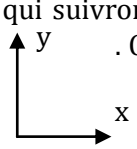
- La première : les données sont bruitées le point n'arrêtait pas de « vibrer » à l'écran ce qui était plus que désagréable.
- La seconde : il aurait fallu gérer le déplacement de la carte avant que le véhicule ne sorte de la limite visible sur l'écran.

Pour cela, il m'a suffit d'appeler la fonction de translation de la carte qui était déjà existante mais qui sert normalement à déplacer la carte à la souris ou avec un bouton de l'interface graphique. Donc à chaque fois que le véhicule se déplace la carte se déplace exactement de la même manière.

AFFICHAGE DE LA VITESSE DU VEHICULE ET DE SA DIRECTION

Pour afficher la direction du véhicule, il me faut connaître la position actuelle et la position précédente du véhicule. A partir de cela, je peux calculer le vecteur de déplacement précédent. Il me suffit de faire une homothétie de valeur = -1 sur le vecteur par rapport à la position courante du véhicule.

Important dans la quasi totalité des transformations qui suivront, il faut penser que le repère d'affichage QT est le suivant 

et non celui-ci 

. Cela m'a joué quelques tours en début de stage.

AFFICHAGE DE LA PORTION DE ROUTE SUR LAQUELLE NOUS SOMMES

Cette méthode fut extrêmement utile pour beaucoup de tests (utile pour vérifier que le programme récupère la bonne route)

J'ai tout d'abord travaillé avec les données brutes du plug-in car je ne comprenais pas comment les données obtenues dans le plug-in étaient rangées dans Mapnik. Après que mon maître de stage m'ait expliqué le principe du plug-in et avec un peu d'aide, j'ai compris la structure de données. Les données sont en fait rangées dans des « Featureset » qui contiennent des « Features » qui contiennent des « geometry ». Les geometry sont des ensembles de « points ».

De cette manière, je peux récupérer les coordonnées correctement (sans court-circuiter le plug-in). Ce format de coordonnées me permet d'utiliser une méthode de classe qui détermine si un point appartient à un « Features ». Chaque « Features » contient une geometry.

Les « Features » représentent les routes, mais aussi les bâtiments, les arrêts de bus, les feux tricolores et autres. Il suffit donc de regarder tous les « Features » dans un périmètre donné.

Ce qui rend le code plus intéressant :

Évidemment pour éviter d'afficher autre chose que les routes, je vérifie que certains mots apparaissent ou non dans la chaîne de caractères qui renseigne chaque Features. Dans le cas où ce n'est pas une route, j'affiche la dernière route parcourue.

Grâce à la chaîne de caractères qui décrit la route sur laquelle se trouve le véhicule, j'affiche les informations suivantes dans la fenêtre QT sur un label :

- Vitesse maximum autorisée sur cette route
- Type de route (résidentielle, autoroute, nationale, etc...)
- La voie (sens unique ou non).

De même, tout le traitement des données n'est pas entièrement refait si le véhicule reste sur la même route. Dans certains cas particuliers, une même route peut être définie par plusieurs « Features » qui posséderont donc la même chaîne de caractères descriptifs. Afin de mettre l'affichage à jour dans ces cas là et donc d'afficher le nouveau tronçon de la même route, j'incrémente un compteur qui, s'il arrive à 5, mets à jour les coordonnées de la route actuelle via un tableau. Tableau qui contient les données du tronçon sur lequel la voiture circule.

IMPORTANT :

- à chaque fois que l'on change de route, nous analysons les 150 premiers mètres de la route afin de ne pas être « aveugle » sur les N premiers mètres de la route.
- Evidement afin de ne pas s'encombrer de risques qui n'existent plus lorsqu'on change de route, le container de danger est vidé (avant de détecter les 150 mètres)
- La détection est interrompue lorsque la voiture se trouve sur un rond point pour éviter les cas aberrants, ou l'on détecte quatre intersections très proches.

AFFICHAGE DE LA DIRECTION DANS LAQUELLE NOUS PARCOURONS LA ROUTE

Cette fonction permet de savoir dans quel sens se déplace la voiture par rapport à la route. Elle ne se base en aucun cas sur les directions cardinales, mais uniquement sur la numérotation des points qui composent la route. La fonction nous est utile pour vérifier que nous allons travailler dans le bon sens.

Afin de savoir dans quel sens je parcours la route, il me suffit de calculer la distance entre le premier point de la voie et le véhicule, puis la distance entre le premier point de la voie et le véhicule à l'instant précédent. Il me suffit ensuite de faire la différence entre ces deux valeurs. Mais il se trouve que cela ne marche pas dans tous les cas sur certaines courbes (routes) le résultat est faux.

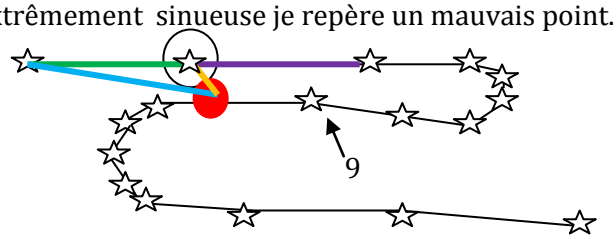
Du coup, il me faut d'abord connaître le point le plus proche précédent la voiture. Puis je reprends l'idée de la différence des distances grâce à cela, je peux déterminer dans quel sens je me dirige. En fonction du signe de la différence, on sait si on parcourt la voie du point 0 au point N ou si on parcourt la voie du point N au point 0.

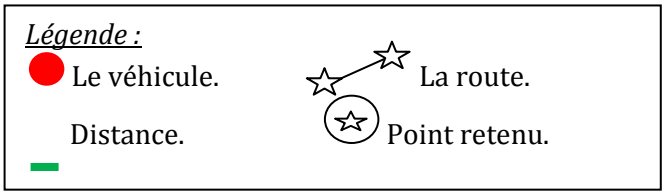
Dans l'algorithme précédent (affichage de la route sur laquelle on roule) on regarde dans quel cas nous sommes et nous affichons le dernier segment vers lequel nous nous dirigeons en vert grâce au résultat obtenu.

OBTENTION DU POINT LE PLUS PROCHE PRECEDENT LE VEHICULE

Il nous suffit de faire une boucle avec la condition suivante : si la distance entre le véhicule et un point est inférieur à la distance entre ce point et le point suivant alors on garde le point en question donc nous obtenons toujours le point le plus qui précédent la voiture.

Il existe un cas extrême qui peut fausser le résultat. Il est possible que sur une route extrêmement sinueuse je repère un mauvais point. Exemple :





Ici la longueur verte est plus courte que la bleue donc le premier essai est non concluant. En revanche dans le deuxième cas la longueur violette est supérieure à longueur orange. Donc il semble que le second point est le point le plus proche du véhicule alors que ca devrait être le neuvième.

REDEFINITION DE LA METHODE QUI PERMET DE SAVOIR SI UN POINT APPARTIENT A UNE GEOMETRIE.

Dans cette fonction, il y a une méthode qui calcule la distance d'un point à un segment. Si cette distance est plus petite que le rayon passer en paramètres ; on obtient alors la geometry à laquelle appartient ce segment. Le rayon dépendait de la taille de la fenêtre ce qui se veut absolument incohérent pour nous ; donc j'ai repris cette fonction en passant la distance voulue en paramètre de manière à l'adapter à nos besoins.

LES FONCTIONS SIMPLES ET EXPLICITES

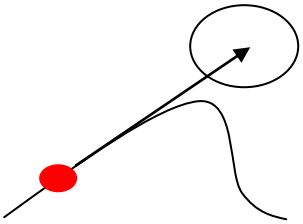
Il y a une multitude de petites fonctions que je ne décris pas car elles sont simples ou peu intéressantes sur le plan algorithmique. En voici une liste non exhaustive :

- Dessiner une croix avec un pixmap et des coordonnées.
- Dessiner une ligne épaisse.
- Dessiner le vecteur direction de la voiture.
- Afficher tous les points d'un fichier GPS.
- Ajouter des labels dans la fenêtre
- Parser des chaînes de caractères
- La lecture de fichier

CALCUL DE LA POSITION A N METTRE DEVANT LE VEHICULE

Cette méthode permet de trouver le point qui est à une distance de N mètre du véhicule. Mais surtout qui se trouve sur la route. Car si nous prenons un point qui se trouve à N mettre droit devant le véhicule alors au moindre virage nous ne détectons plus rien de pertinent pour nous.

Exemple visuel :



Légende :
- Le véhicule.
- Zone de détection.
- Vecteur de direction de la voiture
- Route.

Pour cela, il nous faut évidement encore connaître le sens de déplacement de la voiture sur ce tronçon de route. Nous commençons par récupérer le point le plus proche dans le sens de déplacement du véhicule (cf la fonction précédente). Puis nous calculons la distance entre le véhicule et ce point que nous mettons dans DIST. Puis tant que DIST est inférieur à N alors nous ajoutons à DIST la distance entre deux points consécutifs suivant.

Une fois cela fait, nous avons donc dépassé le nombre de mètres voulu donc nous repartons de l'avant dernier point retenu et nous remettons DIST à la bonne valeur associée. Puis il suffit par homothétie de placer un point sur le segment entre l'avant dernier point et le dernier. L'homothétie doit être de la taille du nombre de mètres manquants.

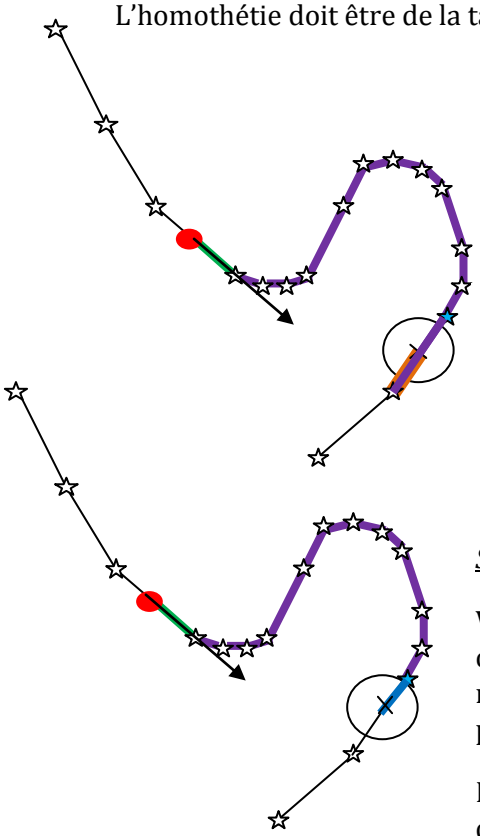


Schéma 1 :

Vert + violet > N (mètres). On repartira donc de l'étoile bleu.

Schéma 2 :

Vert + violet < N (mètres) on calcul le vecteur AB, on le normalise puis on le multiplie par : N-(vert+violet)

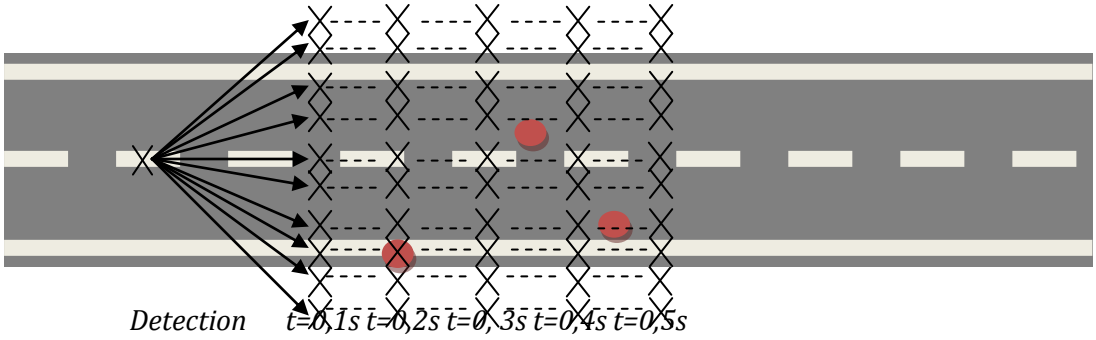
De cette manière on obtient la distance bleue et donc on a : N=vert+violet+bleu

Légende :
- Distance entre le véhicule et le point le plus proche.
- Distance entre deux points consécutifs.
- Distance en trop.
- Points qui forment la route
- Avant dernier point retenu
- Le véhicule.
- Le point N mètres devant le véhicule.
- Distance manquante après depuis l'étoile bleue.
- Direction de la voiture

EXPLICATION DE LA DETECTION

La détection des dangers se décompose en deux parties importantes tout d'abord comment obtenir les POIs. Ensuite il suffit d'un algorithme simple pour ne choisir que ceux que l'on veut retenir. Enfin il faut être sûr de ne pas en rater.

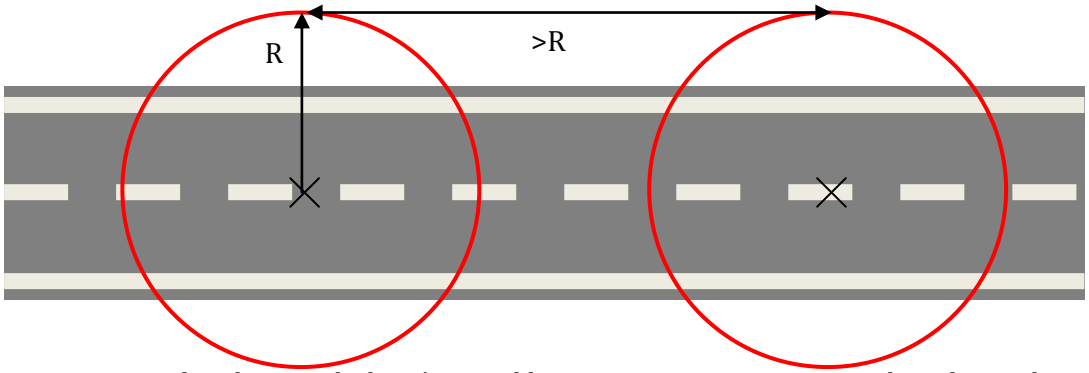
Premièrement, je fus capable de détecter ce qui se trouvait en un point précis. En réalité, le programme détecte un point avec un cercle extrêmement petit mais cela ne change en rien l'exemple au vue de la taille du rayon utiliser. Pour ne pas rater un éventuel danger, j'ai d'abord pensé à utiliser un nuage de point de détection. Puis j'ai remplacé le nuage de points par une ligne de point orthogonale à la route. En diminuant le nombre de point j'augmente l'efficacité du programme. Néanmoins cette idée fut rapidement abandonnée pour être améliorés.



Note : Exemple de l'acquisition des POIs avec une ligne de points exacts. Dans cet exemple, on voit bien que deux des trois dangers représentés par des ronds rouge ne sont pas détectés. On a beau détecter avec beaucoup de points toutes les 10 millisecondes il y aura des ratés.

Après avoir parcouru divers fichiers sources de Mapnik j'ai trouvé une méthode qui recherche tous les points dans un certain périmètre autour d'un point voulu ; donc il apparaît clairement que cet algorithme serait plus efficace que de rechercher avec une ligne de points discontinus.

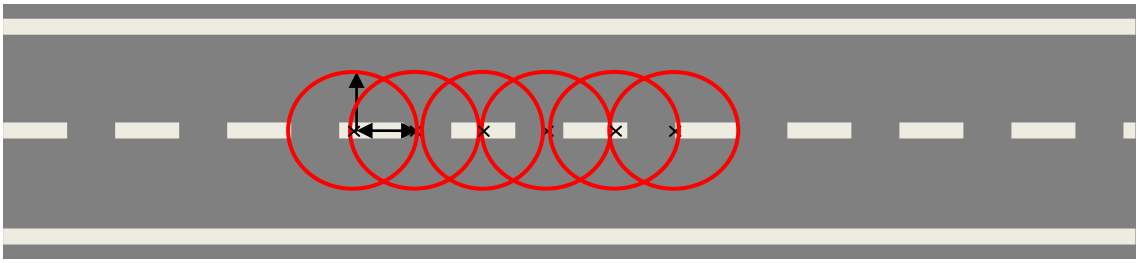
Donc, pour la détection des dangers, au final, nous récupérons tous les points dans un certain périmètre autour du point que nous calculons à N mètres devant la voiture. Ce qu'il faut maintenant c'est être sur de ne rien rater. Il faut donc que lorsque la voiture avance elle n'avance pas plus que de deux fois le rayon de détection sinon nous raterions d'éventuels dangers.



Note : Ici on constate donc la perte de données possible. Puisque tout ce qui se trouve hors des cercles rouges n'est pas détecté.

Sachant que la vitesse maximum autorisée est de 130 km/h soit environ 36 m/s. Il faut donc que notre périmètre de détection soit de 18 mètres si l'on fait une détection toutes les secondes. Il se trouve que nous avons des mesures toutes les ~10ms donc on pourrait prendre un rayon de 1,8 mètres.

Je pensais définir le rayon le rayon de détection en fonction de la vitesse. De plus, il aurait fallu avoir un rayon minimum conséquent car les POIs se trouvent généralement sur le bord des routes. Il existe beaucoup de routes dont la largeur minimum est de 10 mètres. Il aurait donc fallu déterminer un rayon minimum basé sur des heuristiques qui auraient pu engendrer la non détection de certains POIs.



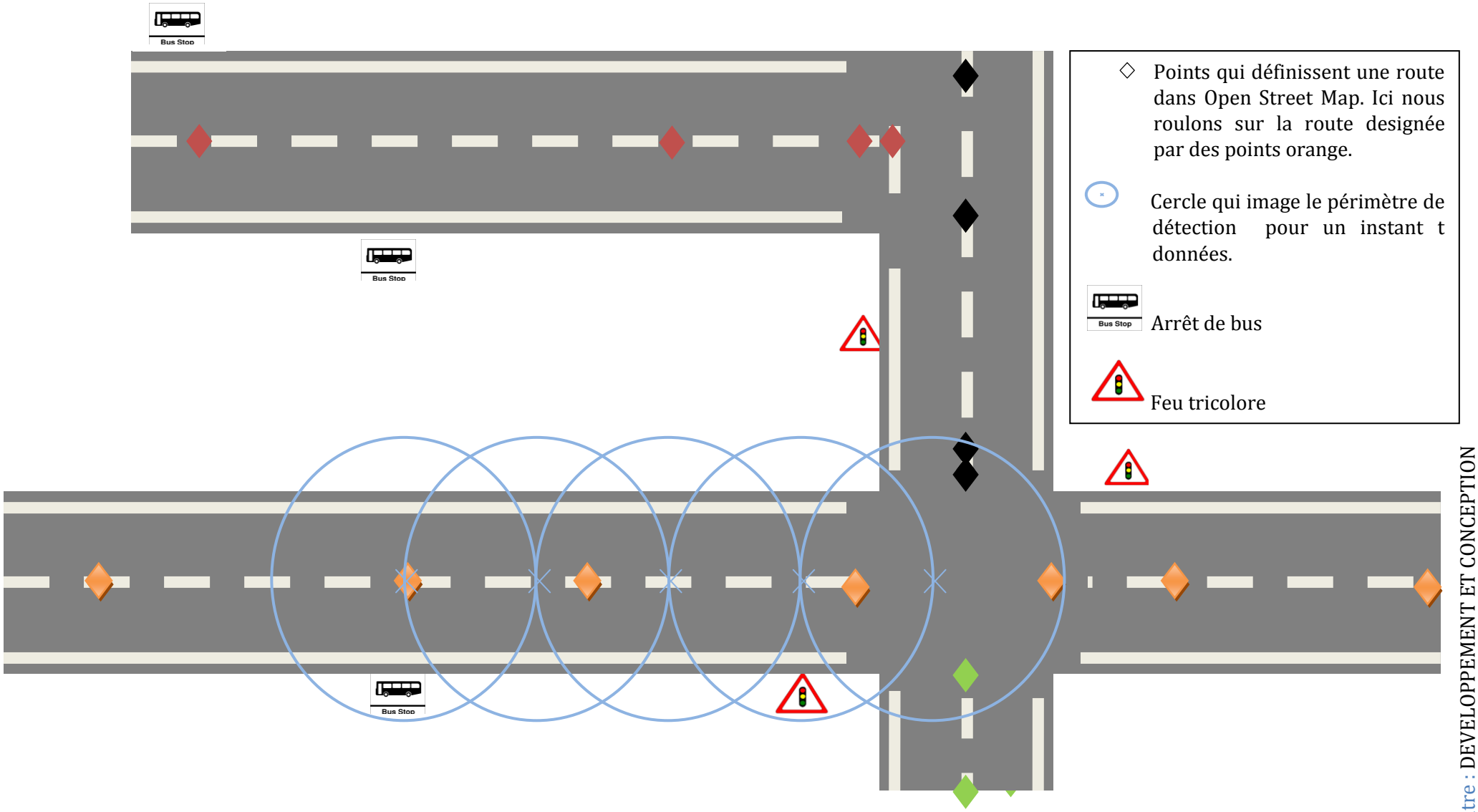
Note : dans le cas ou le rayon est égal à 1, 8m. Le problème vient du fait que le rayon ne dépasse pas la largeur de la route et donc il est impossible de détecter les intersections ou arrêt de bus et autres.

Dans mon exemple, je travaille avec un périmètre de 20 mètres. Ce périmètre a été choisi arbitrairement lors de mes premiers tests. Le programme fonctionne correctement avec un rayon de 20 mètres. Il se trouve que dans l'ensemble des tests il correspond à nos attentes il n'est ni trop grand ni trop petit. Avec l'accord de mon maître de stage j'ai conservé cette valeur « en dur ».

Si le rayon est trop grand alors on peut détecter des POIs qui appartiennent à des routes parallèles. Cela ne nous intéresse pas et pose problème au moment du traitement des données récupérées.

Note : (figure voir ci-dessous) : schéma explicatif de la méthode détection finale. Ici le programme détecte bien ce qu'il y a au bord de la route ; sans détecter ce qu'il se trouve sur les autres. Nous détectons aussi la présence d'autres routes donc on peut anticiper l'intersection. On obtient aussi la présence du feu tricolore. Les arrêts de bus ne seront comptés qu'une fois car ils doivent porter le même nom. De même, que tout ce qui peut être détecté deux fois.

Ici le rayon des cercles est considéré comme faisant 20 mètres. Ici, les cercles de détection sont trop espacés mais je n'en mets pas plus pour ne pas surcharger le schéma.


Bus Stop
Bus Stop
Bus Stop
◇ Points qui définissent une route dans Open Street Map. Ici nous roulons sur la route designée par des points orange.
Cercle qui image le périmètre de détection pour un instant t données.
Bus Stop
Arrêt de bus
Feu tricolore

AFFICHAGE DYNAMIQUE GERE PAR UN QTHREAD :

Dans Qt(voir glossaire) un thread ne peut pas agir directement sur l'affichage c'est pourquoi j'ai utilisé le système de signal/slot de Qt. j'ai créé une classe qui hérite de Qthread. Cette classe est « dirigée » par sa fonction run.

Dans celle-ci, j'émets un signal vers un slot qui appartient au widget d'affichage de la carte. Ce slot reçoit le signal et émet un signal à son tour de cette façon je peux agir du Qthread sur l'affichage graphique. Ce signal appelle une fonction qui permet de mettre à jour l'affichage et qui émet un signal vers le slot qui l'a appellé au bout de 10 millisecondes. Grâce à cela, mon affichage dynamique fonctionne et me laisse tout de même libre de faire ce que je veux dans ma fenêtre (comme de Zoomer par exemple). Ceci ne serait pas possible avec une fonction ne disposant pas de son propre fil d'exécution.

Ce que le thread réalise à chaque itération :

Je récupère un pixmap défini ailleurs sauf dans le cas de la première itération. C'est sur ce pixmap que je superposerai mon affichage. D'abord, je place ma voiture au centre de l'image par une croix noire. Ensuite, je fais afficher le tronçon de route sur lequel le véhicule se déplace. Avec son sens de déplacement évidemment. Ensuite, je dessine le vecteur de direction du véhicule et enfin je finis par réaliser la détection. La détection implique diverses recherches. Pour ce qui est de l'affichage, je place une croix à une distance choisie devant le véhicule sur la route courante. Cette croix représente le centre du cercle de détection. Et puis, je fais appel à un autre thread qui lui va gérer l'affichage des données dans des labels qui appartiennent à la fenêtre.

Puis maintenant que le pixmap a été modifié par toutes les fonctions qu'il l'utilise, il est prêt à être affiché. Je copie en faite une partie de l'image seulement (trois huitièmes pour être exacte explication de cette valeur arbitraire dans la partie sur le thread qui gère le chargement de carte). Puis j'agrandis l'image de manière à remplir la fenêtre. Ensuite je force le « repaint ». Enfin, je fini par lancer un programme qui rappellera la fonction dans 10 millisecondes.

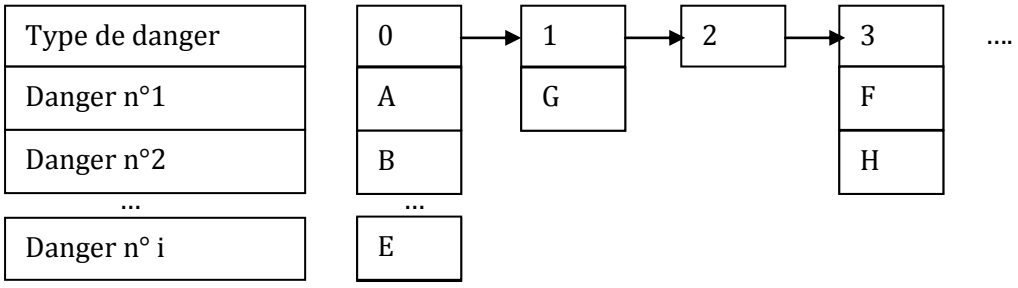
MISE A JOUR ET AFFICHAGE DES DANGERS VIA UN QTHREAD :

Là encore, j'ai créé une classe qui hérite directement de Qthread. Cette classe possède en plus une méthode que je peux appeler quand je veux. C'est grâce à cette méthode que je vais mettre à jour l'affichage des dangers quand il le faut.

Pour ce thread j'ai créé une classe danger qui est composée de :

- Un string (son nom)
- Un string (son type)
- Un double (distance entre la voiture et le danger)

Il est important aussi de savoir qu'il existe une variable globale qui est un vector < vector<danger> > cette variable est rangée de la manière suivante :



Cette méthode commence par mettre à jour les dangers existants. C'est-à-dire que tous les dangers voient leur distance diminuer du nombre de mètres parcourus depuis la dernière détection. Dans le cas où une distance est inférieure ou égale à zéro, alors on supprime le danger qui lui est associé. Cette méthode prend en fait un vector<string> en paramètre. Grâce à ce vector<string>, je vais regarder pour chaque string dans quelle catégorie de danger, je dois le ranger. Une fois trouvé, je regarde s'il existe déjà, si c'est le cas, alors je remets le compteur à la distance à laquelle la détection vient d'être réalisée. Sinon, quand le danger n'est pas déjà présent je l'ajoute au un vector< vector < danger>> nommé risque (voir précédemment) qui contient tous les dangers existants pour l'instant. En fait, pour certains, je ne vérifie pas qu'ils existent déjà car ils sont ponctuels, par exemple en feu rouge. Mais pour d'autres, je suis obligé de vérifier. Par exemple une piste cyclable car elle peut longer une route et par conséquent n'est pas obligatoirement ponctuelle.

Enfin je parcours mon vector< vector < danger> > et je crée un string de manière simple qui affiche le nombre de danger leur type et la distance à laquelle il se trouve.

CHARGEMENT D'UNE NOUVELLE CARTE VIA UN QTHREAD

A partir du moment où nous avons utilisés la base de données postGIS. Le temps de chargement de la carte était bien plus important. Ceci s'explique car je travaille sur toute la région Rhône-Alpes et non plus sur le quartier autour de l'INRIA Rhône-Alpes. Je ne peux plus me permettre d'avoir une mise à jour visuelle toutes 10 millisecondes. Après avoir envisagé plusieurs possibilités lors d'une discussion avec mon maître de stage, celui-ci m'a conseillé la méthode suivante :

Tout d'abord charger une carte quelque peu conséquente. Ensuite, comme cela prend un peu de temps, il faut que je conserve et que je travaille sur cette carte uniquement durant tout le temps où mon thread charge une nouvelle carte. Le problème vient du fait que, dans le cas où la carte est fixe, ma voiture se déplace sur la carte et donc ne reste plus au centre.

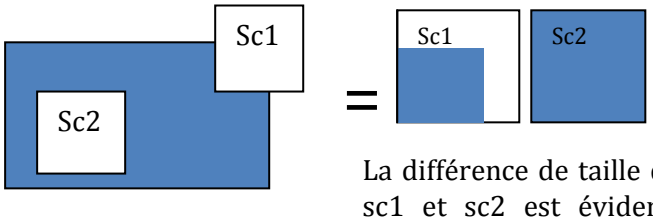
Or mon maître m'a dit au début du stage que la voiture doit être le centre de la carte (voir dans la partie affichage de la voiture). Pour cela, il me faut en fait prendre une sous partie de la carte qui sera centrée sur la voiture (copier une sous partie de la carte ne prend que très peu de temps). Je recharge une carte toute les six secondes sans souffrir de ralentissement pénalisant ou autres. On remarque un léger blocage quand la carte est actualisée mais rien de véritablement gênant.

Problème rencontré :

Si une des extrémités de la sous carte dépasse de la carte existante alors la fonction Qt tronque la sous carte aux valeurs uniquement possibles et dans ce cas, la carte est déformée ce qui entraîne des problèmes de superposition avec mon affichage.

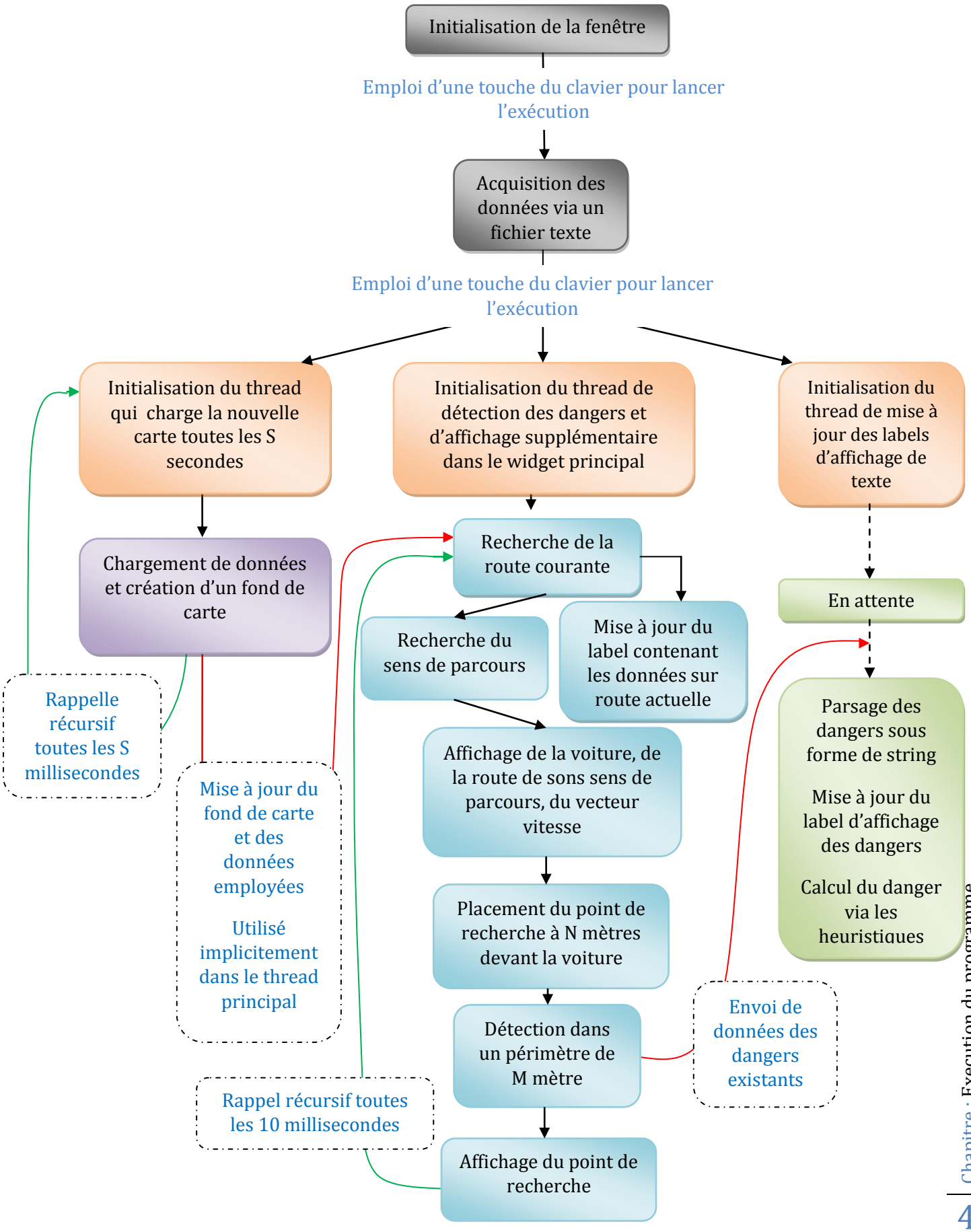
J'ai donc trouvé une valeur empirique qui semble fonctionnait relativement correctement. La sous carte que je conserve est donc de taille 3/16 de la taille de la carte initial.

Schéma explicatif



La différence de taille des carrés bleus entre sc1 et sc2 est évidente. Les carrés bleus seront les zones affichées dans la fenêtre au final. L'agrandissement visant à remplir la fenêtre sera différent. Ceci créera une carte déformée

EXECUTION DU PROGRAMME



HEURISTIQUE

Dans le cadre du projet que je dois réaliser, il me faut en sortie une évaluation chiffrée du risque. Pour cela, je dois connaître le rapport entre le taux de dangers des POIs, la vitesse à laquelle on aborde ces dangers et la distance restante entre le véhicule et les POIs. N'ayant aucune donnée réelle sur cette question, je propose une heuristique. Après concertation avec mon maître de stage, celui-ci m'a conseillé de prendre un modèle assez simple suivant la vitesse et certaines valeurs fixes.

PRESENTATION DU MODELE HEURISTIQUE DE CHIFFRAGE DES DANGERS

LES RATIOS DES POIS

Ratio de danger en fonction des POIs :

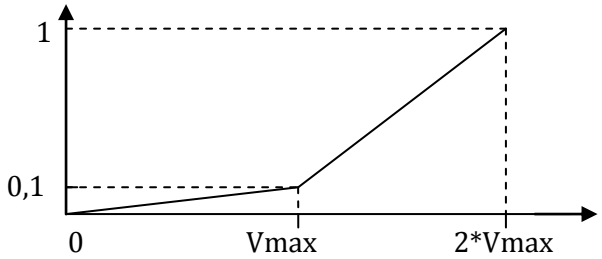
- Intersection : I = 0.5
- Rond point : RP = 0.5
- Feux rouge : F = 0.3
- Piste cyclable, voie piétonne et autre : A = 0.1

Ces valeurs ont été choisies de manière purement arbitraire de ma part. Néanmoins, il respecte une certaine logique car lorsqu'on longe une piste cyclable ou que l'on croise une voie piétonne la probabilité d'accident est moins importante à mon avis que au feu rouge ou dans les ronds points. De même, les intersections sont des zones de dangers importantes en matière de circulation. Une intersection possédant un feu rouge sera d'autant plus dangereuse qu'en général ce sont des intersections compliquées. Il y a souvent des passages piétons à proximité ainsi que plusieurs voies et de nombreuses priorités à prendre en compte par les conducteurs.

Pour que les dangers soient tous pris en compte nous ferons donc une somme des différents dangers. Donc une intersection avec feu rouge aura en réalité un ratio de 8.

LE FACTEUR VITESSE

La vitesse à laquelle on aborde un lieu dangereux peut être déterminante. Par exemple, le fait d'aborder un rond point à 90 ou à 50 Km/h n'implique pas du tout le même danger ; néanmoins nous ne pouvons pas faire des cas particuliers pour toutes les situations. Ce serait trop long, et surtout les valeurs que nous prendrions seraient arbitraires. Il est plus simple et probablement moins dangereux de faire un modèle général. L'idée retenue est que plus la vitesse du conducteur est supérieure à la vitesse maximum autorisée plus le risque augmente rapidement. Le ratio de la vitesse sera le suivant. (Important $v(x) \in [0,1]$)



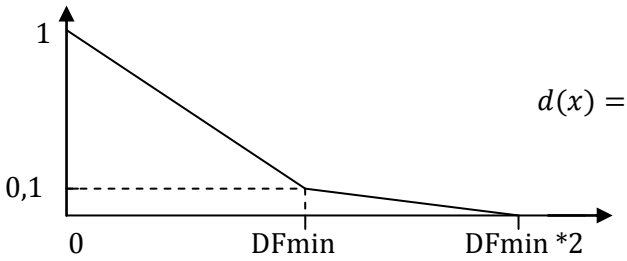
$$v(x) = \begin{cases} \dfrac{V}{Vmax} \times 0.1, & x \leq Vmax \\ \dfrac{0{,}9 \times V}{Vmax} + 0.8, & Vmax \leq x \leq 2 \times Vmax \end{cases}$$

DISTANCE

Pour la distance j'ai décidé de suivre la même fonction que la vitesse. Ici la vitesse maximum autorisée sera remplacée par la distance de freinage minimum et donc la limite sera la distance de freinage maximum divisée par deux. (Important $d(x) \in [0,1]$)

La distance de freinage minimum est simple à calculer. Il suffit de faire

$A = \frac{vitesse}{10}$ et $DFmin = A^2$ formule tirée du code de la route 2010.



$$d(x) = \begin{cases} \frac{D \times (-0.9)}{DFmin} + 1, & x < DFmin \\ \frac{D \times (-0.1)}{DFmin} + 0.2, & DFmin < x < 2 \times DFmin \end{cases}$$

CALCUL FINAL DU DANGER

Au final, je retourne le résultat de la formule suivante :

$$danger\ final = \sum danger \times probabilité\ de\ ne\ pas\ avoir\ de\ d'accident\ avec\ les\ dangers\ précédents$$

Exemple de formule développée à l'ordre 4 :

$$danger\ final = X_1 + [1 - X_1]X_2 + [1 - X_1 - (1 - X_1)X_2]X_3 + [1 - X_1 - (1 - X_1)X_2 - (1 - X_1 - (1 - X_1)X_2)X_3]X_4$$

Avec X_i les dangers

$$X_i = d(x) \times v(x) \times Ratio$$

Cette formule est symétrique et représente la probabilité d'avoir un accident en considérant que les éléments sont non liés. Evidement, cette formule est fausse car les risques dus au réseau routier sont liés, néanmoins il nous est impossible de connaître $P(X_i \cap X_j)$ cette formule est donc la meilleure que nous ayons.

Note : si les ratios et la distance changent en fonction des POIs, la vitesse instantanée elle ne dépend que du véhicule et donc est unique. Il suffit donc de la calculer une seule fois.

INTERFACAGE DU CODE AVEC LE MIDDLEWARE HUGR

UTILISATION DE HUGR

L'emploi de HUGR passe par trois actions différentes. Il faut bien sur activer HUGR pour avoir un espace de mémoire partagée. Il faut ensuite avoir une application qui émet des données (en général un capteur) et une autre qui les reçoivent et les utilisent

STRUCTURE DE DONNEES UTILISEE POUR LA COMMUNICATION

Le GPS émet des données sous la forme d'objet C++. C'est objet sont des Donnees_GPS. La classe Donnees_GPS contient les attributs suivants :

Heure, minutes, secondes, nombre_satellites, coordonee_nord, coordonees_est, coordonee_altitude, mode_fonctionemment, vitesse, cap, pos_lat, pos_lon.

Nous récupérons donc uniquement ce qui nous intéresse c'est-à-dire les champs vitesse, latitude et longitude.

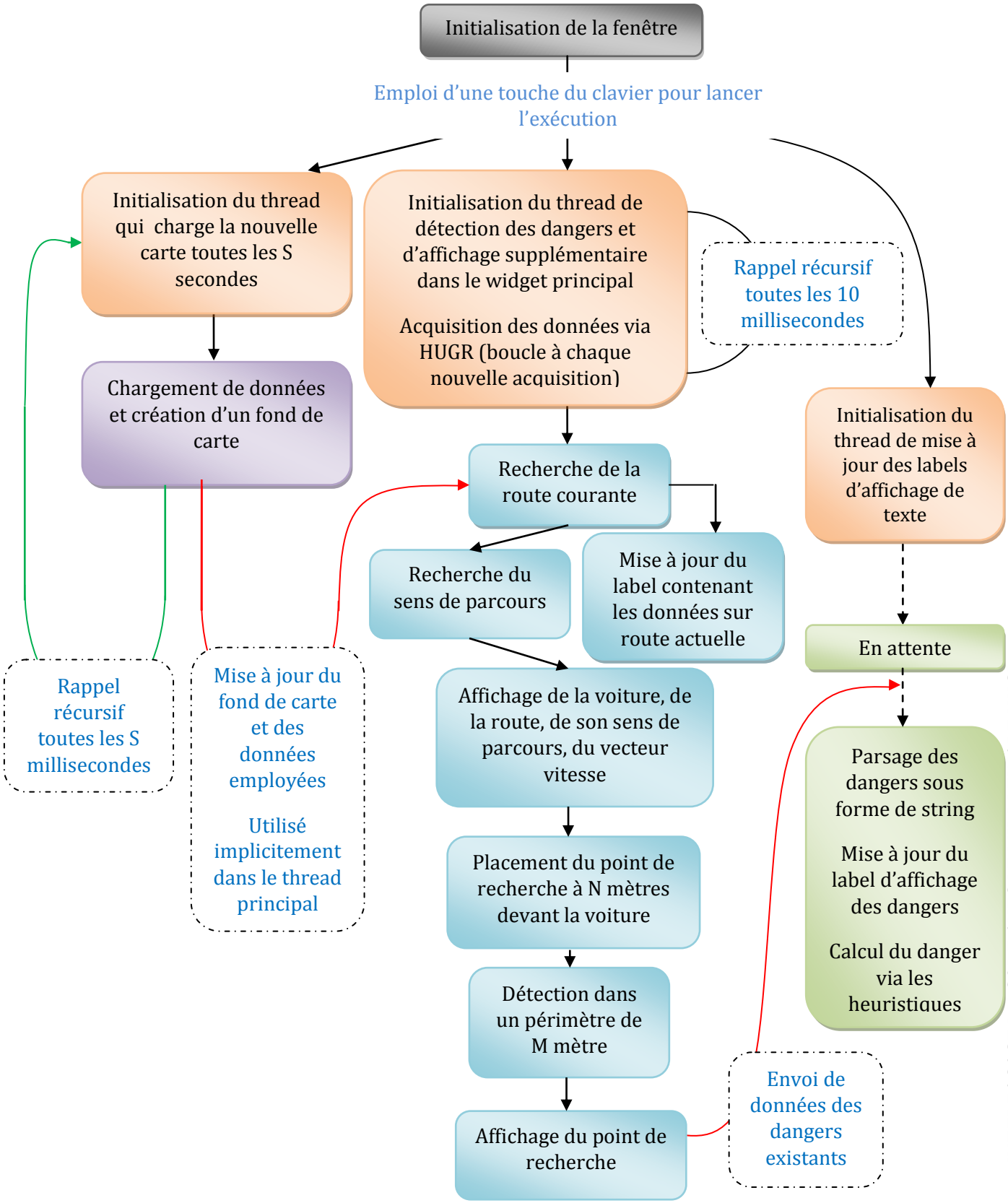
Note : ayant eu des problèmes de compatibilité entre le jeu de test que l'on m'a fourni (fichier de données déjà acquisent et que je devais rejouer) et la version de certaines librairies dont je disposais. Ce problème est en fait lié à la sérialisation Boost utilisée par hugr qui est incompatible d'une version à l'autre. J'ai créé mon propre jeu de test avec HUGR via le fichier texte de données GPS que j'utilisé précédemment. Le procédé n'est pas explicité ici car il ne me semble pas pertinent dans le cadre du projet.

CE QUI CHANGE DANS LE CODE

Dans mon code je n'ai plus besoin de la lecture de données dans un fichier. Je la remplace par l'obtention des données via HUGR. L'obtention des données se fait dans le thread d'affichage dynamique et a lieu toute les 10 millisecondes. De ce fait, la fonction d'affichage dynamique n'est plus itérative. Avant elle se rappelait toute seule et incrémentait un compteur qui me permettait de lire la donnée suivante désormais ce n'est plus le cas.

L'acquisition des données se fait en boucle et relance les algorithmes de détection dès qu'il y a de nouvelles données ou bien toutes deux secondes (ceci dans le but de ne pas avoir de problème d'exception). Le problème c'est que pour certains algorithmes, j'ai besoin de deux jeux de données. Pour cela, j'ai décidé d'avoir des vecteurs de taille deux dans lesquels je stocke la position actuelle et la position précédente du véhicule.

Note : le code a très peu changé. En dehors des quelques compteurs pour les vecteurs qui ont disparu pour être remplacer par des 0 et des 1. Ainsi que la lecture des données qui a été transformée et déplacée.


Initialisation de la fenêtre
Emploi d'une touche du clavier pour lancer l'exécution
Initialisation du thread qui charge la nouvelle carte toutes les S secondes
Initialisation du thread de détection des dangers et d'affichage supplémentaire dans le widget principal
Acquisition des données via HUGR (boucle à chaque nouvelle acquisition)
Rappel récursif toutes les 10 millisecondes
Initialisation du thread de mise à jour des labels d'affichage de texte
Chargement de données et création d'un fond de carte
Recherche de la route courante
Recherche du sens de parcours
Mise à jour du label contenant les données sur route actuelle
En attente
Rappel récursif toutes les S millisecondes
Mise à jour du fond de carte et des données employées
Utilisé implicitement dans le thread principal
Affichage de la voiture, de la route, de son sens de parcours, du vecteur vitesse
Parsage des dangers sous forme de string
Mise à jour du label d'affichage des dangers
Calcul du danger via les heuristiques
Placement du point de recherche à N mètres devant la voiture
Détection dans un périmètre de M mètre
Envoi de données des dangers existants
Affichage du point de recherche

TEST

Durant mon stage, j'ai réalisé de nombreux tests pour vérifier que mon code fonctionnait. Ce qui fut assez facile grâce à l'interface graphique et un fichier de trajectoire GPS. Cela m'a permis de corriger beaucoup de bugs de vérifier que je détectais ce qu'il fallait. Le seul inconvénient étant que je testais mon code toujours sur le même trajet.

Cependant aux cours des deux dernières semaines de mon stage comme mon code était opérationnel, j'ai eu la chance de pouvoir le tester en temps réel dans la Lexus. Nous sommes donc allés faire quelques essais du code dans MontBonnot avec Nicolas Turro.

Grâce à ce test j'ai pu me rendre compte que mon code a en général un peu de retard sur le temps réel environ 1 à 2 secondes ; cependant je pense que cela n'a pas lieu dans la version sans affichage. J'ai détecté quelques cas particuliers que je n'ai pas pris en compte dans mon code et qui pourraient être à revoir. En fait, j'ai surtout détecté un cas particulier important.

Lorsque le véhicule est sur l'autoroute et que nous passons au dessus d'une route ou qu'il y a une sortie d'autoroute ; le programme les détecte comme des routes. Il prend donc en compte des intersections comme danger alors qu'il n'y a pas de croisement. Le programme perçoit cela comme si le véhicule aborde une intersection à 130km/h et donc il rend une valeur de danger maximum.

Pour corriger cette erreur, il suffit de faire en sorte que dans le cas où le véhicule est sur un motor_highway il ne doit détecter que des trunks_ways ou des junctions.

Ceci étant, mon programme fonctionne relativement correctement. Il détecte bien la route sur laquelle nous sommes, il récupère les informations importantes et évalue les dangers de manière « assez » cohérente au vu de la simplicité des heuristiques. La mise à jour de la carte se fait correctement. Dans l'ensemble, je pense que c'est une réussite. Il reste certaine, choses à améliorer bien sûr mais le programme est fonctionnel dans l'ensemble.

Les choses que j'aurai pu améliorer si j'avais eu plus de temps, serait donc, la prise en compte de plus de cas particuliers et surtout réaliser un vrai travail sur les heuristiques. J'aurais également pu tenter de diminuer le temps de latence qui existe avec mon interface graphique.

CONCLUSION

Durant mon stage, j'ai employé des logiciels et des langages avec lesquels nous travaillons à l'université ou déjà employés dans mon stage de l'année passée. Ceci m'a permis d'en approfondir la connaissance. J'ai découvert aussi beaucoup, notamment sur les bases de données. Je n'avais jamais eu à en installer ni à en employer au travers d'un langage de programmation.

Cette partie du stage fut la partie qui m'a posé le plus de problème pour deux raisons :

- La première est que je n'avais jamais installé de base données car nous ne l'avons jamais fait en TP, et aussi que je m'étais trompé de projection la première fois.
- La seconde étant que les bases de données n'était pas L'UE ou j'excellais.

Au cours de mon stage, j'ai pu enfin me rendre compte que je commence à être capable de réaliser un projet en entreprise. Je n'en doutais pas vraiment au vu des projets que j'ai réalisés dans mon année scolaire. Néanmoins ceci est une preuve importante et réconfortante pour moi de mes capacités. Je pense aussi que j'ai bien mieux cerné le sujet que je devais réaliser que l'année passée et donc que j'en ai bien mieux évalué les difficultés.

Durant les semaines que j'ai passées à développer au sein de L'INRIA, j'ai pu constater, ce qui pour moi sont, les points forts et les points faibles de ma formation. Je pense que nous avons un enseignement de l'algorithmique et de la relation entre les mathématiques et informatique qui est plus que conséquent. De même que notre connaissance de la POO me semble relativement complète. Je n'ai d'ailleurs pas eu de réelles difficultés à élaborer les algorithmes pour mon programme. En revanche, il m'est arrivé à plus d'une reprise, de bloquer sur des parties plus informatique. Notamment sur l'édition de liens pour la compilation, ainsi que l'installation de librairie et les dépendances qui vont avec. De même, nous connaissons trop peu de commande Shell à mon avis.

Ce stage fut en tout cas réellement bénéfique pour mes connaissances. Il le fut tout autant pour ma motivation à continuer dans cette voie et me conforte dans le choix de mes études. Le fait de d'avoir travaillé sur un sujet qui soit réellement d'actualité pour de nombreuses entreprises (les GPS) rendra cette expérience précieuse sur mon CV.

REMERCIEMENTS

Je tiens à remercier l'INRIA pour l'opportunité qui m'a été offerte d'effectuer ces recherches.

Je remercie chaleureusement Nicolas TURRO pour son accueil, son encadrement, mais aussi ses critiques constructives qui m'ont permis de progresser dans mon stage. J'ai également apprécié sa bonne humeur quotidienne et sa disponibilité.

J'ai été encadré par des personnes disponibles et soucieuses de m'aider qui ont eu la gentillesse de me consacrer une partie de leur temps ; notamment Soraya Arias, Sandrine avakin, Jean-Francois Cuniberto ainsi que toute l'équipe SED. Ce fut avec plaisir que j'ai effectué ce stage. Je remercie également les autres stagiaires du Hall robotique pour les échanges de connaissances.

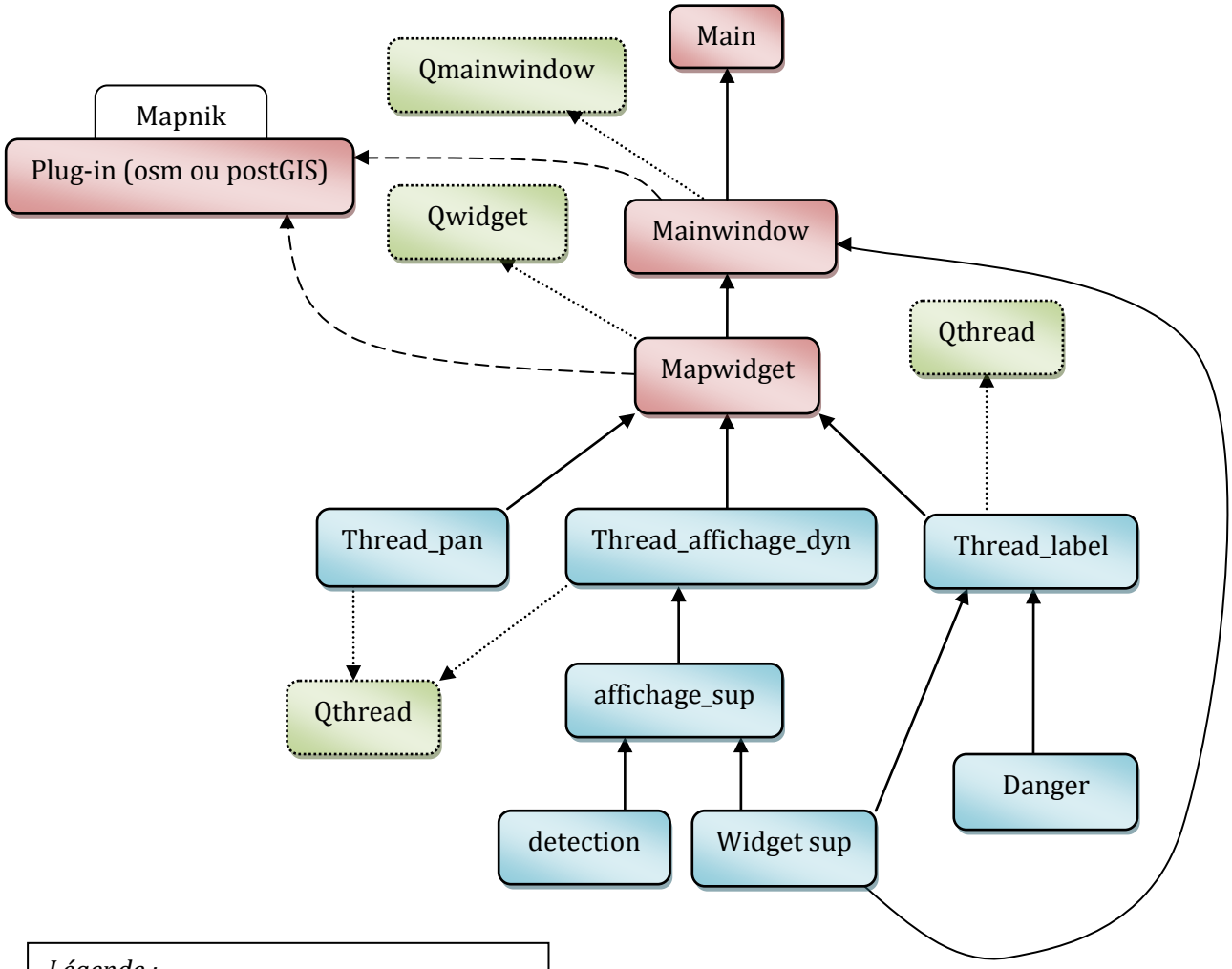
Je remercie également Françoise Jung et toute l'équipe pédagogique pour la mise en place des stages en entreprise.

Je remercie vivement et respectueusement mon tuteur Jean-Guillaume DUMAS pour le temps qu'il a dû consacrer à évaluer le travail que j'ai réalisé.

Je remercie aussi L. Françoise et Adeline Trener pour leur participation linguistique.

ANNEXE

DIAGRAMME DE CLASSES



Légende :

→ fichier faisant appel à au moins une méthode étant dans un autre fichier.

⋯→ Fichier qui hérite d'une class C++ connu.

- - -→ Ficher incluant majoritairement les fonctionnalités Mapnik ou faisant appel au plug-in.

Fichier créé par moi

Fichier modifié mais déjà existant dans le paquet Mapnik

Class de la librairie QT

Note : Le diagramme de classe est en fait un UML très simplifié, mais qui donne une idée des liens entre les différents fichiers qui composent mon code. Ici ils ne figurent que les fichiers que j'ai créés ou modifiés. Il y a d'autres fichiers dans le viewer mais je n'ai pas eu à les modifier. Par conséquent, afin de simplifier le diagramme j'ai choisi de ne pas les faire apparaître.

PROJECTION UNIVERSAL TRANSVERSE MERCATOR

L'utilisation des coordonnées en projection (ex : E et N UTM) plutôt que des coordonnées géographiques (Latitude /Longitude) est en général jugée avantageuse pour les raisons suivantes :

Les coordonnées sont basées sur un système décimal, plus facile à utiliser pour les calculs que le système sexagésimal. Cependant avec des longitudes et latitudes on peut toujours travailler en degrés "décimaux" sans avoir à utiliser des minutes et des secondes d'angles ;

Le système est "rectangulaire" et est mesuré en kilomètres. On peut donc directement calculer des distances approximatives à partir des coordonnées UTM.

LES FORMULES AVEC UNE PRECISION AU CENTIMETRE

Les formules exactes sont compliquées et peu utilisables. Nous proposons des formules approchées avec une précision de l'ordre du centimètre.

Par convention, le géoïde WGS 84 décrit la terre par un ellipsoïde de révolution d'axe Nord-Sud, de rayon à l'équateur a=6378,137 km et d'excentricité e=0,0818192. On considère un point de latitude géodésique φ et longitude λ. Notons λ_0 la longitude du méridien de référence.

Les angles sont exprimés en radian. Voici des valeurs intermédiaires à calculer:

$$\nu(\varphi) = 1/\sqrt{1 - e^2 \sin^2 \varphi}$$

$$A = (\lambda - \lambda_0) \cos\varphi$$

$$s(\varphi) = (1-\frac{e^2}{4}-\frac{3e^4}{64}-\frac{5e^6}{256})\varphi-(\frac{3e^2}{8}+\frac{3e^4}{32}+\frac{45e^6}{1024})\sin 2\varphi+(\frac{15e^4}{256}+\frac{45e^6}{1024})\sin 4\varphi-\frac{35e^6}{3072}\sin 6\varphi$$

$$T = \tan^2\varphi, \quad C = \frac{e^2}{1-e^2}\cos^2\varphi, \quad k_0 = 0,9996$$

Dans l'hémisphère Nord N_0 = 0 et dans l'hémisphère Sud N_0 = 10000*km*. Voici les formules de passage donnant les coordonnées UTM *E*, *N* en kilomètres:

$$E = 500 + k_0 a \nu(\varphi)\left(A + (1 - T + C)\frac{A^3}{6} + (5 - 18T + T^2)\frac{A^5}{120}\right)$$

$$N = N_0+k_0 a \left(s(\varphi)+\nu(\varphi)\tan\varphi\left(\frac{A^2}{2}+(5-T+9C+4C^2)\frac{A^4}{24}+(61-58T+T^2)\frac{A^6}{720}\right)\right)$$

Toute cette partie est tirée de « Wikipédia ».

Cette projection étant l'une des plus utilisée et au vue des données que j'obtenais du viewer je pensais que c'était la bonne transformation j'ai donc codé les formules précédentes en intégralité ce qui assez fastidieux et inutile au final. Néanmoins, il me semblait important d'en parler dans le rapport car c'est la projection à partir de laquelle j'ai compris beaucoup de chose sur l'utilité de ne pas travailler avec uniquement des degrés décimaux. De plus la projection que Mapnik utilise est assez proche de celle car j'obtiens bien la bonne longitude mais pas la bonne latitude

ALGORITHME DE DETECTION DES PROCHAINES ROUTES

Cet algorithme n'est pas utilisé dans l'interface, néanmoins je l'ai implanté et testé. Il fonctionne correctement donc j'ai décidé de l'ajouter dans les annexes de manière quelque peu succincte.

Le problème, lorsque je place un point à N mètre devant le véhicule c'est que lorsque la distance restante entre la voiture et la fin de la route est inférieure à N, alors je laisse le point au bout de la route. Initialement je n'ai pas détectée les 150 premiers mètres sur une route donc j'avais un partie de N mètres où le programme était aveugle. J'ai donc eu l'idée de lui faire explorer les débuts des routes suivantes de manière à toujours tout connaitre à N mètres devant le véhicule.

Mon code fonctionne très bien son problème est qu'il consomme des ressources, souvent pour rien. Par exemple, sur une route, on rencontre une intersection ; mon programme va continuer à détecter devant le véhicule mais aussi à droite et à gauche. Quelque soit la direction que la voiture, on a détecté les dangers de deux routes pour rien. Evidemment cela implique des difficultés supplémentaires pour traiter les données et aussi et surtout pour estimer les risques.

Tous ces défauts ont été corrigés par le fait de détecter les 150 premiers mètres lorsqu'on change de route.

CODE SANS AFFICHAGE GRAPHIQUE

Durant mon stage j'ai développé une version de mon code sans affichage graphiques. L'affichage graphique étant utile lors des tests. A terme le code sera utilisé dans sa version sans affichage graphiques ce qui permet un gain énormes en temps d'exécution. Ce code provient directement de la version graphique.

Je conserve cependant l'emploi de Qt pour l'emploi des Qthreads. Ce code renvoie une sortie texte à chaque exécution du thread principal. Cette sortie est de la forme suivante :

DANGER = 0.05352115466 nombre compris entre 0 et 1

GLOSSAIRE

ADT : L'Action de Développement Technologique.

Amenity: un attribut spécifique à Open Street Map

API: Une interface de programmation (*Application Programming Interface* ou *API*) est une interface fournie par un programme informatique. Elle permet l'interaction des programmes les uns avec les autres, de manière analogue à une interface homme machine, qui rend possible l'interaction entre un homme et une machine. "Wikipedia"

Bash : invite de commande linux

BDD: abréviation de Base de données

Framework: Structure

Freeze: phénomène d'arrêt (dans notre cas de l'image) pendant un cours moment

GTX : Format de fichier adapté à GEO-concept

Highway: route à grande vitesse. Dans Open Street Map c'est juste une variable

IGN: L'Institut Géographique National. Son rôle consiste à décrire, d'un point de vue géométrique et physique, la surface du territoire national et l'occupation de son sol, en faire toutes les représentations appropriées, diffuser les informations correspondantes. "www.ign.fr"

Lat, lon: abréviation latitude longitude

Layer: couche (épaisseur).

Mapnik: Mapnik est un logiciel libre pour les applications de développement cartographiques. Il est facilement extensible et convient à la fois à l'application de bureau et au développement web.

MapQuest: *MapQuest* est une société américaine éditrice de carte et offrant un service gratuit en ligne de cartes de navigation. Il y a peu MapQuest c'est lance dans le monde du logiciel libre.

Merge: traduction : mêler. Ici c'est en fait l'action de fusionner des fichiers.

Métadonnées : Une métadonnée est une donnée servant à définir ou décrire une autre donnée quelque soit son support (papier ou électronique).

Middleware: traduction intergiciel : logiciel servant d'intermédiaire de communication entre plusieurs applications, généralement complexes ou distribuées sur un réseau informatique.

OSM : Abréviation d'OpenStreetMap.

Pixmap: Les *pixmap* sont des structures de données qui contiennent des images. Ce sont souvent des tableaux de pixel.

POI(s): abréviation de point(s) d'intérêt(s)

PostgresSQL: *PostgreSQL* est un système de gestion de bases de données relationnelles et objet. C'est un outil libre disponible selon les termes d'une licence de type BSD.

Psql: *psql* est un client *PostgreSQL* en mode terminal. Il vous permet d'entrer vos requêtes de façon interactive, de les envoyer à *PostgreSQL*, et de voir le résultat.

QThread: un thread dont l'implantation général est défini dans la librairie Qt

SGBD: abréviation de système de gestion de base de données

Tag : étiquette ou balise

Thread: processus fonctionnant en parallèle d'un processus principal

UTM: Universal Transverse Mercator (projection standard dans la plupart des pays)

Viewer: visionneur. Ici viewer est le nom de programme qui affiche une fenêtre de visualisation des données d'où son nom.

WGS84: World Geodesic Systeme révision de 1984. C'est un système de coordonnées terrestres, basé sur un géoïde de référence prenant la forme d'un ellipsoïde de révolution.

Widget : composant d'une fenêtre graphique