SY27 Rapport d'étude

Suivi de routes multi hypothèses

Projet réalisé par :
Johan MATHE Jake NICOLE
Dernière version : 16 janvier 2008

Table des matières

Table des figures

Liste des algorithmes

Chapitre 1

Présentation

1.1 Présentation du projet

Le but de ce projet est d'améliorer l'affichage du positionnement d'une voiture sur sa route. Pour ce faire nous utilisons bien évidement le capteur de position GPS embarqué sur la voiture et une base de données contenant les routes. Le gain de précision dans ce projet est apporté par la prise en compte d'un capteur proprioceptif de la voiture nous informant sur le cap de la voiture. La position de la voiture est rendue plus précise avec une fusion des données recueillies.

Ensuite, une fois que l'imprécision sur le positionnement est diminuée, il reste à se positionner sur une route existante car à priori la voiture circulera sur une voie. Cette étape est réalisée par une recherche des routes environnantes et après calcul de cohérence, la plus probable est choisie.

L'utilisateur de cet équipement pourra voir sa position sur la voie de circulation ainsi que l'indice de confiance que le système lui accorde. Ainsi, lorsque les conditions d'utilisation du positionnement GPS seront restreintes (ex : forte urbanisation, dégradation ionosphérique importante...), l'utilisateur prendra connaissance de sa position estimée tout en sachant que celle-ci peut être erronée.

Le système garde une certaine inertie qui lui permet lors de mauvaise réception GPS (ex : Tunnel) de garder une certaine cohérence par rapport à l'avance du véhicule. Cependant, ce fonctionnement ne confère pas une extrême précision sur le positionnent, c'est juste un confort de visualisation apporté à l'utilisation. Seul l'indice de confiance pourra être réellement signe de précision.

1.2 Evolutions du projet par rapport à la première étude

Nous devions à l'origine travailler grâce à des données de type EGNOS. L'indisponibilité de celles ci nous a amenés à ne pas les prendre en compte, mais notre système est designé de sorte à ce qu'elles soient très facilement intégrables au code.

1.3 Diagramme synoptique de l'approche mise en place

Avant de passer à la deuxième partie ou l'on rentrera dans les détails théoriques liés à l'estimation multi-hypothèses, on peut tout d'abord voir un schéma synoptique préliminaire représentant les grandes étapes de l'algorithme.

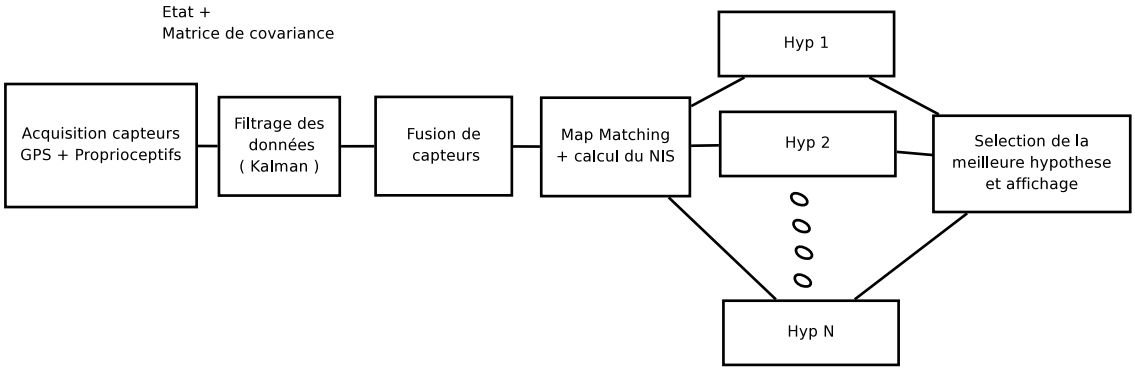

FIG. 1.1 – Illustration du synoptique du programme

Chapitre 2

Travaux effectués

2.1 Modèle de la voiture :

La prise en compte, dans le positionnement, du capteur proprioceptif n'est possible que si on l'intègre dans un modèle dynamique de la voiture. C'est pourquoi une représentation de la voiture en termes de dynamique est nécessaire.

D'après le schéma ci-dessous, nous pouvons établir le lien entre la position de la voiture et ses paramètres propres :

Modèle Vélocipède :

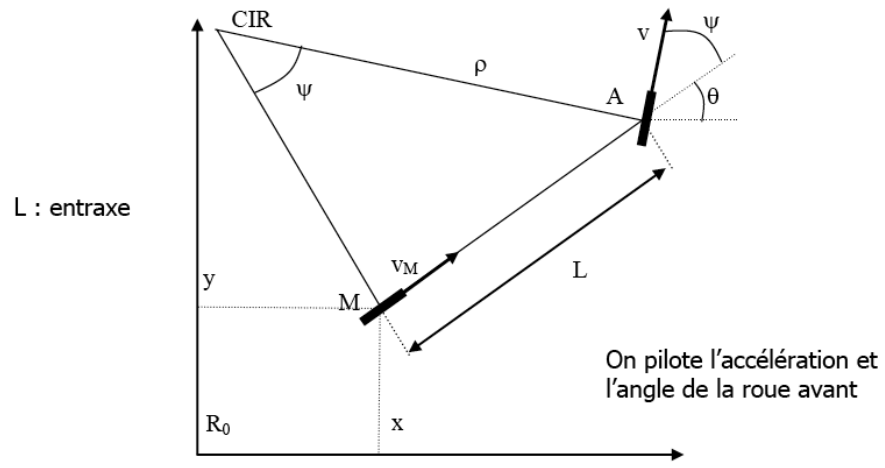

FIG. 2.1 – Illustration du modèle vélocipède

$$X = Vt\cos(\theta)$$

$$Y = Vt\sin(\theta)$$

La représentation d'état est la forme minimale d'un modèle, notre système est développé pour une application embarquée, donc tout gain d'espace est essentiel. Ainsi la représentation de la voiture peut se faire de la manière suivante :

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} X_{k+1} + T_e v_k \cos(\theta) + \alpha_X \\ Y_{k+1} + T_e v_k \sin(\theta) + \alpha_Y \\ \theta_{k+1} + \alpha_\theta \end{bmatrix} = f(X_k) + Q_A$$

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X_k + Q_B$$

Le modèle d'évolution de la voiture est donc issu de la mise en équation de la position de la voiture en fonction de ses paramètres. L'état X_k donne toutes les informations sur la voiture. Les paramètres α sont considérés comme des bruits de modèle. Ils permettent ici de mettre en doute la qualité du modèle.

Le modèle d'observation représente les éléments mesurables du système, ici avec les équipements de la voiture, nous pouvons mesurer la position GPS et le cap du véhicule. Les paramètres Q_B représentent l'imprécision sur la mesure du GPS et du cap.

Ce modèle de représentation permet ainsi d'unifier toutes les données fournies sur la voiture afin d'apprécier au plus juste la position du véhicule.

2.2 Filtrage du positionnement :

La position peut être à présent extraite de l'état de la voiture mais pour obtenir une meilleure dynamique, l'utilisation d'un filtre récursif est tout à fait appropriée. L'utilisation d'un filtre de Kalman semble évidente car il utilise la représentation d'état d'un système ainsi que les covariances du modèle et des mesures pour ajuster sa sortie.

Dans notre cas, l'application du filtrage de Kalman étendu est nécessaire car notre modèle d'évolution n'est pas linéaire à cause des fonctions trigonométriques. Pour utiliser ce filtre, nous prendrons la Jacobienne du modèle d'évolution présentée ci-dessous :

$$F_k = \left.\frac{\partial f}{\partial X}\right|_{X_k} = \begin{bmatrix} 1 & 0 & -v_k \sin(\theta_k) \\ 0 & 1 & v_k \cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix}$$

2.2.1 Filtrage de Kalman

Si l'on considère ici Q_A le bruit lié à l'imprécision du modèle Q_B le bruit de mesures, on

$$y_k, Q_\beta$$

$$P_{1/0} = I$$

$$\hat{x}_{1/0} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

$$K = P_{1/0} C^T (C P_{1/0} C^T + Q_\beta)^{-1}$$

Calcul de l'estimation

Algorithme 1 Calcul de l'estimation

y_k, Q_B <- mesure
$\hat{x}_{k/k} = \hat{x}_{k/k-1} + K(y_k - C\hat{x}_{k/k-1})$
$P_{k/k} = (I - KC)P_{k/k-1}(I - KC)^T + KQ_BK^T$

Algorithme 2 Calcul de la prédiction

$A_k = \frac{\partial f(\hat{x}_{k/k})}{\partial x}$
$\hat{x}_{k+1/k} = f(\hat{x}_{k/k})$
$P_{k+1/k} = A_k P_{k/k} A^T + Q_A$
$K = P_{k/k-1} C^T (C P_{k/K-1} C^T + Q_B)^{-1}$

2.3 Fusion de données

On est ici confronté à un problème classique de fusion de données. L'objectif de notre étude étant d'élire une hypothèse de route parmi d'autres, l'importance est donc de normaliser de partir d'une théorie de la fusion multicapteurs. On prendra donc ici une approche statistique et on utilisera la distance de MAHALANOBIS.

2.3.1 Distance de Mahalanobis

Soit un vecteur X de moyenne m_x de moyenne m_x et de covariance P_x, la théorie nous présente la distance de MAHALANOBIS qui a surtout pour but fondamental de rendre les données cohérentes avant de les fusionner. Ainsi, on obtient :

$$D = \sqrt{(x - m_x)^T P^{-1} (x - m_x)}$$

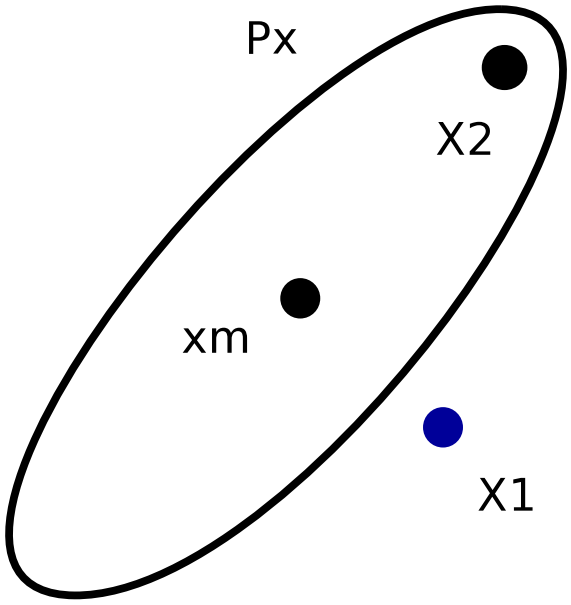

FIG. 2.2 – Illustration de l'utilisation de la distance de Mahalanobis

On voit très bien sur la figure 2.4 page 10 que la distance prend en compte une notion de qualité de valeurs estimées.

Dans notre cas, on simplifiera aisément le calcul de cette distance en modifiant notre matrice de covariance de sorte à ce qu'elle soit diagonale et que l'ellipse de covariance associée à la position

devienne un cercle de rayon égal au rayon minimal de cette ellipse. Pour cela on diagonalise la matrice et on choisit la valeur propre minimale. Pour cela, il suffit de calculer les racines du polynome caractéristique associé à la matrice u 2x2 associée à la position :

$$\det(u - XI)$$

On peut donc écrire :

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y & 0 \\ \sigma_x\sigma_y & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \equiv P_s = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_r^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

La distance de Mahalanobis devient ainsi :

$$D = \sqrt{X^T P_s^{-1} X} = \sqrt{\frac{d_{abs}^2}{\sigma_r^2} + \frac{\theta^2}{\sigma_\theta^2}}$$

Notre vecteur étant un VTA gaussien, on peut prouver que la distance de Mahalanobis au carré est assimilable à une loi $\chi^2(2)$ d'espérance k et de variance $2k$. On tombe donc dans le cadre bien connu des statistiques. Ainsi :

$$NIS = D^2 = \frac{d_{abs}^2}{\sigma_r^2} + \frac{\theta^2}{\sigma_\theta^2}$$

$$NIS \sim \chi^2(2)$$

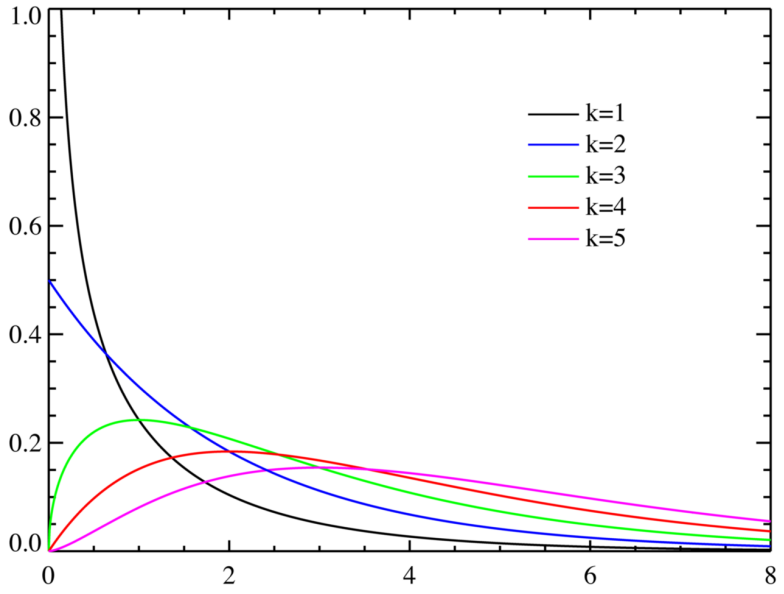

FIG. 2.3 – Densité de probabilité du chi-deux

2.4 Multi-hypothèses

L'originalité de ce projet réside dans le fait que l'on effectue un suivi de routes multi-hypothèses. Le principe est le suivant : on dispose de N hypothèses, N étant un entier fixé à l'avance. Il nous faut trouver un critère d'ordre pour trier ces hypothèses et extraire la plus pertinente. C'est ici qu'intervient le bagage théorique expliqué dans la section 2.3 page 8

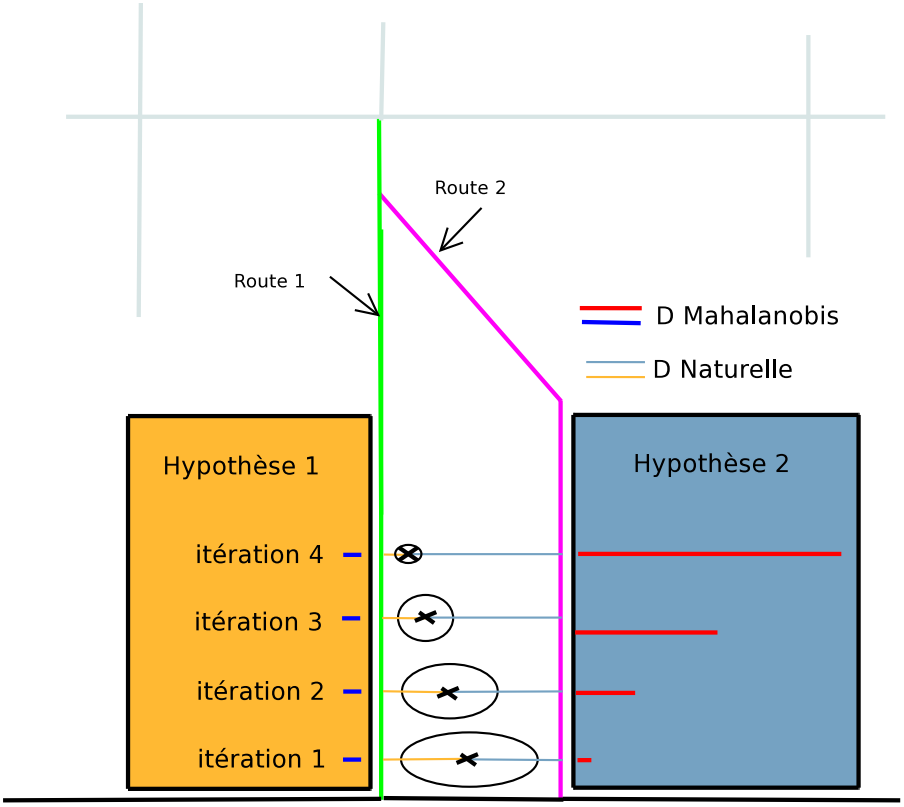

FIG. 2.4 – Illustration du NIS et de la distance de Mahalanobis

2.4.1 Définition d'une hypothèse

Pour être en mesure de sélectrionner une hypothèse il faut d'abord bien définir ce qu'est une hypothèse. Dans notre cas, les hypothèses sont les N routes les plus proches au sens de celles qui minimisent les NIS de l'état à l'itération k de l'objet.

Une hypothèse est un objet mathématique qui contient donc :

1. Une route sur la carte
2. Une valeur de NIS
3. Une valeur de notre indicateur statistique W

2.4.2 Mise en place d'un critère d'ordre

Le critère d'ordre pour une hypothèse choisi ici correspond au NIS de la route associée à celle ci à une légère différence près :

Algorithme 3 Algorithme de recherche des N hypothèses les plus proches

```
mettre a jour cache de route()
pour toutes routes
     calculer nis route
fin pour
trier routes
renvoyer N meilleures routes
```

2.4.2.1 Mise en place d'une mémorisation de la route

L'important est de donner un poids statistique plus important à une route sur laquelle on obtient un bon NIS depuis plusieurs itérations. Pour cela, il suffit à chaque itération de prendre en compte le résultat de l'hypothèse précédente. Ainsi, si $W_i[k]$ correspond à la valeur de l'indicateur statistique à l'itération k pour une hypothèse, $w_i[k+1]$ vaudra :

$$w_i[k+1] = W_i[k]e^{-\alpha NIS_i[k]}$$

ou encore, si l'on passe au logarithme (pour des raisons de stabilité numérique sur des petits calculateurs) :

$$\log(w_i[k+1]) = \log(W_i[k]) - \alpha NIS_i[k]$$

Avec ici α un coefficient permettant de donner plus ou moins d'inertie au système.

2.4.2.2 Normalisation

Pour toujours pourvoir travailler dans le cadre théorique apporté par les statistiques, il nous faut ici normaliser ces valeurs entre 0 et 1. Nous appelerons $W_i[k]$ la valeur de l'indicateur statistique normalisée. On aura donc :

$$W_i[k] = \frac{w_{i[k]}}{\sum_{i=1}^{N} w_i[k]}$$

Ainsi, une fois cette normalisation effectuée, l'indicateur statistique représentera la probabilité que l'hypothèse soit valide. Il s'agira donc de prendre l'hypothèse disposant du coefficient W_i le plus important.

2.4.3 Algorithme mutli hypothèses

Nous abordons ici l'algorithme de nouvelle hypothèses

2.4.3.1 Représentation UML des objets

On voit ici la modélisation UML des objets mémoire pour la mise en place logicielle du suivi multi hypothèses. Il s'agit en effet d'un objet global contenant toutes les hypothèses sous forme de vecteur (pour pouvoir appliquer un tri sur ces objets). Les hypothèses contiennent une méthode appelée update() qui permet de mettre à jour toutes les nouvelles informations par rapport au nouvel état de l'objet suivi.

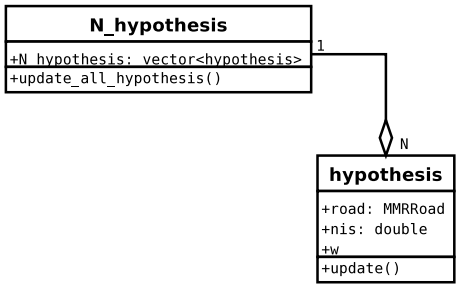

FIG. 2.5 – Diagramme UML de notre coeur applicatif

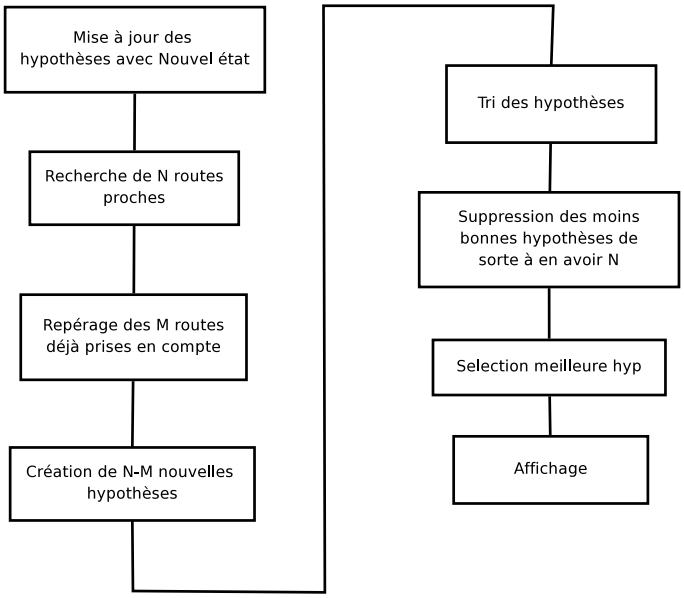

FIG. 2.6 – Algorithme de mise à jour des hypothèses

2.4.3.2 Algorithme de mise à jour des hypothèses

2.5 Implémentation

L'application a été codée en langage C++ à l'aide de 4 librairies utilisées d'une façon intensive :

- La librairie OpenGL pour l'affichage
- La librairie STL pour manipuler des modèles de données avancées comme les listes et les vecteurs. Cette librairie implémente aussi des possibilités de tri rapide (complexité en $n\log(n)$) il suffit pour cela de surcharger l'opérateur $<$. C'est à dire de fournir un critère d'odre total au compilateur.
- La librairie Boost qui est en fait une évolution de la librairie STL. Boost offre accès à des modèles de données encore plus avancés et des concepts comme les pointeurs intelligents etc.
- La librairie QT permet l'affichage multi plateformes

2.5.1 Rendu graphique

On peut voir ici le rendu des applications. La figure 2.7 montre le résultat sans la fusion multicapteurs et le traitement multiroutes. alors que la figure 2.8 nous montre le résultat avec le multiroute. On voit bien ici qu'en environnement urbain les résultats sont bien meilleurs.

FIG. 2.7 – Résultat de l'affichage sans le multiroute

FIG. 2.8 – Affichage avec multiroute

Chapitre 3

Bibliographie

JABBOUR MAGED, BONNIFAIT PHILIPPE : Localisation de véhicules en milieu urbain à l'aide d'un lidar et d'une base de données navigable. Laboratoire Heudiasyc.

Chapitre 4

Annexes

Annexe 1 : code du coeur du multi hypothèses

```
/*
 * Copyright (C) 2007 Jake NICOLLE - Johan MATHE
 * This library is free software; you can redistribute it
 * and/or modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either version
 * 2.1 of the License, or (at your option) any later version. This
 * library is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details. You should have received a copy of the GNU
 * Lesser General Public License along with this library; if not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301 USA $Id: film.cpp 141 2007-04-02 16:10:53Z
 * johmathe $ $Date: 2007-12-12 17:22:16 +0100 (Mer, 12 dÃ©c 2007) $
 */

#ifndef __MULTIHYP_HPP__
#define __MULTIHYP_HPP__

#include "../display/positioning_display_base.hpp"
#include "include/implementation/implementation_multihyp.hpp"
#include "hypothesis.hpp"
#include <iostream>
#include <Qt/QtOpenGL>

using namespace MultiHyp;
using namespace std;


int ComputeEllipse(const double &pxx, const double &pxy, const double &pyy, const float &
    proba, double & height,double &width, double &heading )
{
        double k, ro, a, b, c, lambda1, lambda2, delta, aux, deno, Ux, Uy, axeX, axeY;

        // le scalaire "k" definit l'ellipse avec l'equation :(x-mx)T*(1/P)*(x-mx)=k^2
        k=sqrt(-2*log(1-proba));
        // coeficient de correlation
        ro = pxy / sqrt(pxx * pyy);
        if ( fabs( ro ) > 1 )
```

```
39          {
40                  std::cout<<"correlation coefficient is not included between -1 and 1 -
                        Covariance matrix is not defined positive" <<std::endl;
41                  return 0;
42          }
43
44          a = 1/(pxx*(1- ro * ro));
45          b = -ro/(sqrt(pyy*pxx)*(1- ro * ro));
46          c = 1/(pyy*(1- ro * ro));
47
48          // calcul des deux valeurs propres
49          // la gde vp (lambda1) est associee au petit axe.
50          delta = (a-c)*(a-c)+4*b*b;
51          lambda1 = 0.5*(a+c+sqrt(delta));
52          lambda2 = 0.5*(a+c-sqrt(delta));
53
54          // vecteur directeur du grand axe
55          aux = (lambda2-a)/b; deno=sqrt(1+aux*aux);
56          Ux = 1/deno;
57          Uy = aux/deno;
58
59          // longueur des axes dans le repere propre
60          axeX = k/sqrt(lambda2); // demi axe
61          axeY = k/sqrt(lambda1); // demi axe
62
63          heading = M_PI/2 - atan2(Uy, Ux);
64          width = axeY * 2 * 3; // x3 (sigma) si PROBA = 0.4 ellipsoide a deux dimensions (
                test du khi2)
65          height = axeX * 2 * 3;
66
67          return 1;
68  }
69
70  class MultiHypDisplay : public PositioningDisplay{
71          public:
72
73                  /* Constructor */
74                  MultiHypDisplay(const int &timerInterval):PositioningDisplay::
                        PositioningDisplay(timerInterval){
75                     data.open("data.txt", ifstream::in);
76                     if(!data.is_open()) {cout<<" Imposible to open data file" <<endl;
                           exit(0);}
77                     init =false;
78                  }
79
80                  /* Destructor */
81                  ~MultiHypDisplay(){}
82
83                  /* Load the map from a file */
84                  void LoadMap(){
85
86                         MCC.LoadSerializedMapCache("map.dat");
87                         MCC.MC_current->ComputeRoadsSpecificAttributs();
88                         mm.MapUpdate(MCC.MC_current);
89
90                         // Set initial view
91                         map<unsigned long, Road<LAMBERT93> > ::iterator I=MCC.MC_current->
                               mroads.begin();
```

```
92                      vector<LAMBERT93>::iterator Ip=(*I).second.GeometryPoints.begin();
93                      SetRefPoint((*Ip).x(),(*Ip).y());
94                      SetLookPoint((*Ip).x(),(*Ip).y());
95
96                      // Load roads in memory of the graphic card
97                      for(I=MCC.MC_current->mroads.begin();I!=MCC.MC_current->mroads.end()
                            ;I++){
98                              vector<vector<GLfloat> > road;
99                              for(Ip=(*I).second.GeometryPoints.begin();Ip!=(*I).second.
                                    GeometryPoints.end();Ip++){
100                                     vector<GLfloat> point(3);
101                                     point[0]=(*Ip).x();
102                                     point[1]=(*Ip).y();
103                                     point[2]=0;
104                                     road.push_back(point);
105                             }
106                             AddLine(road,(*I).second.Id);
107                     }
108             }
109
110     protected :
111             /** Function called by the timer */
112             void timeOut(){
113                     static int cmp=0;
114                     int gps_status;
115
116
117                     if(init == false ){
118                             init=true;
119                             /* read data from file + Kalman intialisation */
120                             data>> de.U(0) >> de.Q(0,0) >> de.U(1) >> de.Q(1,1) >> s.X(0)
                                     >> s.X(1) >>s.P(0,0) >> s.P(0,1)>>s.P(1,0) >> s.P(1,1)
                                    >> gps_status;
121                             s.P(2,2)=M_PI;
122                             /* initialization of all hypothesis */
123                             hyps.init_all_hypothesis(s,mm);
124                             /*for(int i=0;i < hyps.size() ; i++)
125                             {
126                                     AddMMHypLine(hyps.hyps(i).road().road->Id);
127                                     std::cerr << hyps.hyps(i).road().nis << std::endl;
128                             }*/
129
130                     }
131                     else if(!data.eof()){
132                             /* read data from file */
133                             data >> de.U(0) >> de.Q(0,0) >> de.U(1) >> de.Q(1,1) >> gme.Z
                                    (0) >> gme.Z(1) >>gme.R(0,0) >> gme.R(0,1)>>gme.R(1,0)
                                    >> gme.R(1,1) >> gps_status;
134
135                             /* Kalman prediction step */
136                             de.Predict(&s,&s);
137                             /* Kalman correction step */
138                             if(gps_status != 0 && cmp++==5){ gme.Update(&s,&s);cmp=0;}
139                            /* str_overlay<<"Pos";
140                             str_overlay<<s.X;
141                             str_overlay<<"Pos cov";
142                             str_overlay<<s.P; */
143
```

```
144                             /* Set the looked point */
145                             SetLookPoint(s.X(0),s.X(1));
146 
147                             /*Set the position of the object */
148                             DeletePositions();
149                             AddPosition(s.X(0),s.X(1), s.X(2),0);
150 
151                             hyps.update(s,mm);
152                             DeleteMMLines();
153                             DeleteMMHypLines();
154                             AddMMLine(hyps.get_best_road());
155 
156                             /* Drawing potential lines */
157                             for(int i=0;i < hyps.size() ; i++)
158                             {
159                                     //AddMMHypLine(hyps.hyps(i).road().road->Id);
160                             }
161                             double a,b,phi;
162                             ComputeEllipse(s.P(0,0),s.P(0,1),s.P(1,1),0.40, a,b,phi );
163                             std::cout << a <<" "<<b<<" "<<phi<<std::endl;
164                             DeleteEllipses();
165                             AddEllipse(s.X(1),s.X(2),a,b,phi/180*M_PI,0);
166 
167                             /* Update the GL view */
168                             updateGL();
169 
170                     }else{
171 
172                             cout<<"End of file"<<endl;
173                     }
174             }
175 
176 
177     private :
178             /* all the hypothesis */
179             all_hypothesis hyps;
180 
181             /*Filter member classes */
182             UState s;
183             UDynamicEquation de;
184             UGPSMeasureEquation gme;
185 
186             /* Sensor data */
187             ifstream data;
188             bool init;
189 
190             /* Map cache members classes */
191             UMapCacheClient MCC;
192             UMapMatching mm;
193             UMapMatchResults mmr;
194 
195 };
196 
197 #endif //__MULTIHYP_HPP__
```

Annexe 2 : code d'une hypothèse

```
1 /*
```

```
 * Copyright (C) 2007 Jake NICOLLE - Johan MATHE
 * This library is free software; you can redistribute it
 * and/or modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either version
 * 2.1 of the License, or (at your option) any later version. This
 * library is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details. You should have received a copy of the GNU
 * Lesser General Public License along with this library; if not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301 USA $Id: film.cpp 141 2007-04-02 16:10:53Z
 * johmathe $ $Date: 2007-12-12 17:22:16 +0100 (Mer, 12 dÃ©c 2007) $
 */


#ifndef __HYPOTHESIS_H__
#define __HYPOTHESIS_H__

#include "../display/positioning_display_base.hpp"
#include "include/implementation/implementation_multihyp.hpp"
#include <iostream>

#include "include/implementation/implementation_multihyp.hpp"


using namespace Map;
using namespace MultiHyp;
using namespace std;

class hypothesis {

  private :
    /* Road associated with the hypothesis */
    UMapMatchResults _road;
    /* Distance between route and object */
    double d;
    /* Variance for route */
    double var_d;
    /* Cap */
    double cap;
    /* Cap Variance */
    double var_cap;
    /* Cap coefficient */
    double c_cap;
    /* Distance coefficient (c_d + c_cap = 1) */
    double c_d;
    /* w */
    /* Next raods */
    std::vector<unsigned long> NextRoads;

  public :
    /* Normalized innovation square */
    double nis;


    double w;
    hypothesis( UMapMatchResults a_road, double aw) {
```

```
60        _road = a_road;
61        w = aw;
62      }
63
64      ~hypothesis() {
65
66      }
67
68      void update(const UState &s, UMapMatching & mm ) {
69        _road = mm.MapMatch(s,_road.road);
70        nis = _road.nis;
71        w = nis ; // + 0.5*w;
72      }
73
74      inline UMapMatchResults & road(void) { return _road; }
75      //    inline double & w(void) { return _w; }
76  };
77
78  class all_hypothesis {
79
80    private :
81      vector<hypothesis> _all_hypothesis;
82      vector<UMapMatchResults> results_search;
83
84    public :
85      int Nhyp;
86      int best_hyp;
87      unsigned long get_best_road() {
88        double w_min = std::numeric_limits<double>::max();
89        std::cerr << "BEST HYP : " << best_hyp << " | ROAD : " << _all_hypothesis[0].road().
              road->Id << " | W :" << _all_hypothesis[0].w << " | "<< std::endl;
90        return _all_hypothesis[0].road().road->Id;
91      }
92
93      unsigned int size()
94      {
95      return _all_hypothesis.size();
96      }
97
98      void update(const UState &s , UMapMatching & mm) {
99        results_search = mm.MapMatchN(s,Nhyp);
100       int newhyps = 0;
101
102       for (int i=0; i< Nhyp ;i++)
103       {
104         bool already_chosen = false;
105         hypothesis hyp_i(results_search[i],1);
106         for(int j=0; j< _all_hypothesis.size(); j++)
107         {
108           if(hyp_i.road().road->Id == _all_hypothesis[j].road().road->Id)
109           {
110           already_chosen = true;
111           }
112         }
113
114         if(!already_chosen)
115         {
116         cerr << "CREATING NEW ROUTE" << endl;
```

```
117         hypothesis hyp_j(results_search[i],1);
118         _all_hypothesis.push_back( hyp_j );
119         newhyps++;
120         }
121       }
122       /* Computing nis for each hypothesis */
123       for(int i=0;i< _all_hypothesis.size();i++)
124         _all_hypothesis[i].update(s,mm);
125
126       /* Sorting hypothesis */
127       sort(_all_hypothesis.begin(),_all_hypothesis.end());
128       /* DISPLAY for each hypothesis */
129                       for(int i=0;i< _all_hypothesis.size();i++)
130                       {
131                               cerr.setf( ios_base::fixed, ios_base::floatfield );
132                               cerr.precision( 6 );
133                               cerr << "n: " << i << " dcap : " << _all_hypothesis[i].road()
                                    .dcap_normalized << " \t dabs_normalized :" <<
                                    _all_hypothesis[i].road().dabs_normalized << " \t nis :
                                    " << _all_hypothesis[i].nis << " \t w : " <<
                                    _all_hypothesis[i].w << endl;
134                       }
135
136       /* Deleting obsolete hypothesis */
137       if(newhyps)
138       {
139         for(int i=0;i<newhyps;i++)
140         {
141           cerr << "DELETING HYP" << endl;
142           _all_hypothesis.pop_back();
143         }
144       }
145     }
146
147
148     void init_all_hypothesis (const UState &s, UMapMatching & mm ) {
149       results_search = mm.MapMatchN(s,Nhyp);
150       /* Initialization of all hypothesis */
151       for ( int i=0; i< Nhyp; i++)
152       {
153         hypothesis hyp_i(results_search[i],100);
154         _all_hypothesis.push_back(hyp_i);
155         std::cerr << "pushing hyp "<< i << "data : " << hyp_i.road().road->Id << "distance
                " << hyp_i.road().pf[1]<< "distance " << hyp_i.road().pf[0]<< std::endl;
156
157       }
158     }
159     inline hypothesis& hyps(unsigned int i) { return _all_hypothesis[i]; }
160     all_hypothesis () { Nhyp = 10; }
161     ~all_hypothesis () {}
162
163 };
164
165 /*Map matching result Comparison function*/
166 bool operator<(const hypothesis & hyp1 , const hypothesis & hyp2)
167 {
168   return abs(hyp1.w) < abs(hyp2.w);
169 }
```

```
170
171
172
173 #endif // __HYPOTHESIS_H__
```