C++
PRÉDICTOR
MÉTHODE PARTICULAIRE

- Quelques règles du C++
- Erreurs rencontrées à ne pas reproduire !
- Là ou nous en sommes rendu

QUELQUES RÈGLES DU C++ À SAVOIR !

- Librairies de base: (.h .cpp)
  - Iostream = input/output -> gérer les flux
  - Vector = Tableaux dynamiques
  - Math = Formules mathématiques
  - Std = Librairies standard de C++,
  - Fstream = Gestion des fichiers.

- Droits d'accès (.h)
  - Public: l'attribut ou la méthode peut être appelé depuis l'extérieur de l'objet.
  - Private: l'attribut ou la méthode ne peut pas être appelé depuis l'extérieur de la classe.

- Constructeur / Destructeur (.h . cpp)
  - Constructeur : Initialisation attributs (donner une valeur par défaut)
  - Destructeur: Libération de la mémoire (Malloc, vector, tableau, …)

- Const
  - Mot servant pour les méthodes => Indique au compilateur méthode ''d'affichage''

- Vector (Tableau dynamique)
  - vector<double> tableau;
  - vector<string> listeNoms(12, "Sans nom");
  - vector.push_back(); => permet de d'ajouter à la fin
  - vector.pop_back(); => Permet de supprimer la dernière case
  - vector.at<double>(0,0) => accéder à une variable du vector

- Namespace (.cpp)
  - Comme import numpy as np => même pas besoin de mettre np. !
  - ‘using namespace std’
  - Par convanction que dans les .cpp
  - Problème: peut se perdre entre plusieurs ‘namespace’

- Définition et implémentation de méthodes
  - Définition (.h): renvoie(type)-nom-attribut méthode();
  - Implémentation (.cpp): Classe::méthode(){}
    - Utiliser les attributs de classe sans spécifier l’appartenance

- Print
  - Afficher plusieurs choses en une ligne cout<<’’_’’<<’’_’’<<endl: cout<<List_robot.capacity()<<endl;

Main.cpp | robot.cpp | robot.h | Cours_Cpp.cpp | Robot_States.txt

```cpp
#include <vector>
#define PI 3.141592653

struct State{
  int ID;
  double t;
  double x;
  double y;
  double theta;
}typedef State;

typedef std::pair<double, double> point;
//////// CLASSE
class Robot
{
private:
  int m_ID;
  std::vector<State> m_state;

public:
  double t;
  double theta;
  double theta_bar;
  double theta_dot;
  cv::Mat u;
  cv::Mat x_hat, Gx_hat;

  Robot(); // Constructeur par defaut
  //~Robot(); // Destructeur
  void P_theta(); //Porportionnel pour theta
  Robot(int,double);
  //Methodes
  void kalman_predict( cv::Mat& xup_k, cv::Mat& Pup_k, cv::Mat* x_k1, cv::Mat* P_k1);
  void kalman_correct( cv::Mat* xup_k1, cv::Mat* Pup_k1);
```

robot.cpp — ~/Documents/GIT/Predictor/Methode_particulaire/CPP

robot.h | robot.cpp | Main.cpp | Cours_Cpp.cpp

```
58
59  Robot::Robot(int ID,double dt)
60  :x(Mat::zeros(3, 1, CV_64F)), u(Mat::zeros(1, 1, CV_64F)),C(Mat::zeros(2, 3, CV_64F)),
61  A(Mat::zeros(3, 3, CV_64F)),B(Mat::zeros(3, 1, CV_64F)),
62  Galpha(Mat::zeros(3, 3, CV_64F)),y(Mat::zeros(2, 1, CV_64F)),Gbeta(Mat::zeros(2, 2, CV_64F)),
63  Gx_hat(Mat::zeros(3, 3, CV_64F)) ,x_hat(Mat::zeros(3, 1, CV_64F)),
64  t(0),m_ID(ID),dt(dt),theta_bar(0),v(1),theta(0),theta_dot(0),Kp(1),theta_mission(0)
65  {
66    x.at<double>(0,0) = 0;
67    x.at<double>(1,0) = 0;
68    x.at<double>(2,0) = 1;
69
70    x_hat.at<double>(0,0) = 0;
71    x_hat.at<double>(1,0) = 0;
72    x_hat.at<double>(2,0) = 1;
73
74
75    u.at<double>(0,0) = 1;
76
77    A.at<double>(0,0) = 1;
78    A.at<double>(0,2) = cos(theta*PI/180)*dt;
79    A.at<double>(1,1) = 1;
80    A.at<double>(1,2) = sin(theta*PI/180)*dt;
81    A.at<double>(2,2) = 0;
82
83    B.at<double>(2,0) = 1;
84
85    Galpha.at<double>(0,0) = pow(1,2);
86    Galpha.at<double>(1,1) = pow(1,2);
87    Galpha.at<double>(2,2) = pow(1,2);
88
89    Gbeta.at<double>(0,0) = 0;
90    Gbeta.at<double>(1,1) = 0;
91
92    Gx_hat.at<double>(0,0) = pow(0.1,2);
93    Gx_hat.at<double>(1,1) = pow(0.1,2);
94    Gx_hat.at<double>(2,2) = pow(0.1,2);
95  }
96
```

ERREURS

- ;
- Taille => segmentation fault !
  - Bonne définition de la taille => Constructeur, variables externes à la classe
- Variables de classes / Variables temporaires => classe.variable
- Penser à utiliser GDB
  - Core dumped => dit n'importe quoi dans les 'log'.

NOTRE TRAVAIL

- Classe Robot => Plus facile pour la méthode Particulaire
- Méthode de classe
  - Kalman (Correct + Predict)
  - Evolution (Euler)
  - Proportionnel pour theta=> PID si nécessaire
  - Affichage + écriture coordonnées => Unity Vibes
- Main
  - Définition de la mission
  - Aller retour => 3 amers en triangle

- Adaptation pour MOOS

BIBLIOGRAPHIE

- https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c OpenClassRooms