Introduction au Middleware MOOS

Simon Rohou

Décembre 2018

V1.05
Supports de cours disponibles sur
`www.simon-rohou.fr/cours/moos-ivp/`

ENSTA Bretagne

ENSTA Bretagne

Section 1

Introduction

Introduction

Middleware : définition

De l'architecture logicielle en robotique :

Un **Middleware** (intergiciel) est un logiciel tiers qui créé un réseau d'échanges d'informations entre différentes applications informatiques.

Il permet de diviser la partie logicielle du robot en une suite de processus pouvant être démarrés simultanément ou séquentiellement au lancement d'une mission.

Introduction

Utilité d'un middleware

L'intérêt d'un middleware ? C'est pouvoir :

- **séparer** les applications
  et donc clarifier la répartition du travail au sein d'une équipe
- **paralléliser** facilement les processus
  et tirer profit des n cœurs du processeur utilisé
- **distribuer** les applications sur différentes machines
  car elles sont toutes liées à un serveur
- **rejouer** des missions
  car il est facile de logger tous les changements de variables
- **réutiliser** les applications déjà implémentées
  qui répondent à nos problématiques

Introduction

Middlewares existants

Quelques middlewares courants :

- **ROS** : Robot Operating System
- **MOOS** : Mission Oriented Operating Suite
- **YARP** : Yet Another Robot Platform
- **MIRA** : Middleware for Robotic Applications
- **LCM** : Lightweight Communications and Marshalling
- **RTMaps** : Real Time, Multisensor applications
- ...

ENSTA Bretagne

Introduction

MOOS : présentation

MOOS en quelques mots...

- « *a Mission Oriented Operating Suite* »
- architecture *publish-subscribe*
- initialement développé par Paul Newman (Université d'Oxford)
- implémentation en C++
- désormais interfaçable avec Python

Introduction

MOOS-IvP : une extension de MOOS

Quelle différence entre **MOOS** et **MOOS-IvP** ?

- **IvP** est une extension de **MOOS**
- des chercheurs du MIT [1] sont actifs dans son développement
- abréviation de *Interval Programming* ($\neq$ Interval Analysis)
- introduit de nombreux outils de prise de décision fondamentaux en robotique autonome
- **IvP** fourni aussi de nouveaux outils de contrôle (*ex :* PID), de simulation (*ex :* modélisation de courants marins), de communications (*ex :* interactions inter-robots), etc.

1. Michael R. Benjamin, Henrik Schmidt, John J. Leonard

Introduction

MOOS-IvP : les pour et les contre

- utilisation facile
- documentation très complète
- bon support technique
- économe en ressources $\Longrightarrow$ adapté à de l'embarqué
- utilisé par de grands organismes
  (MIT, Oxford, Bluefin Robotics, US Navy, CMRE, CGG, RTSys)
- contributions possibles sur Github
  (MOOS déjà présent, MOOS-IvP en migration)

- communauté souvent restreinte à la recherche
- absence de forums de discussions
- absence d'outils de tests unitaires ou d'intégration

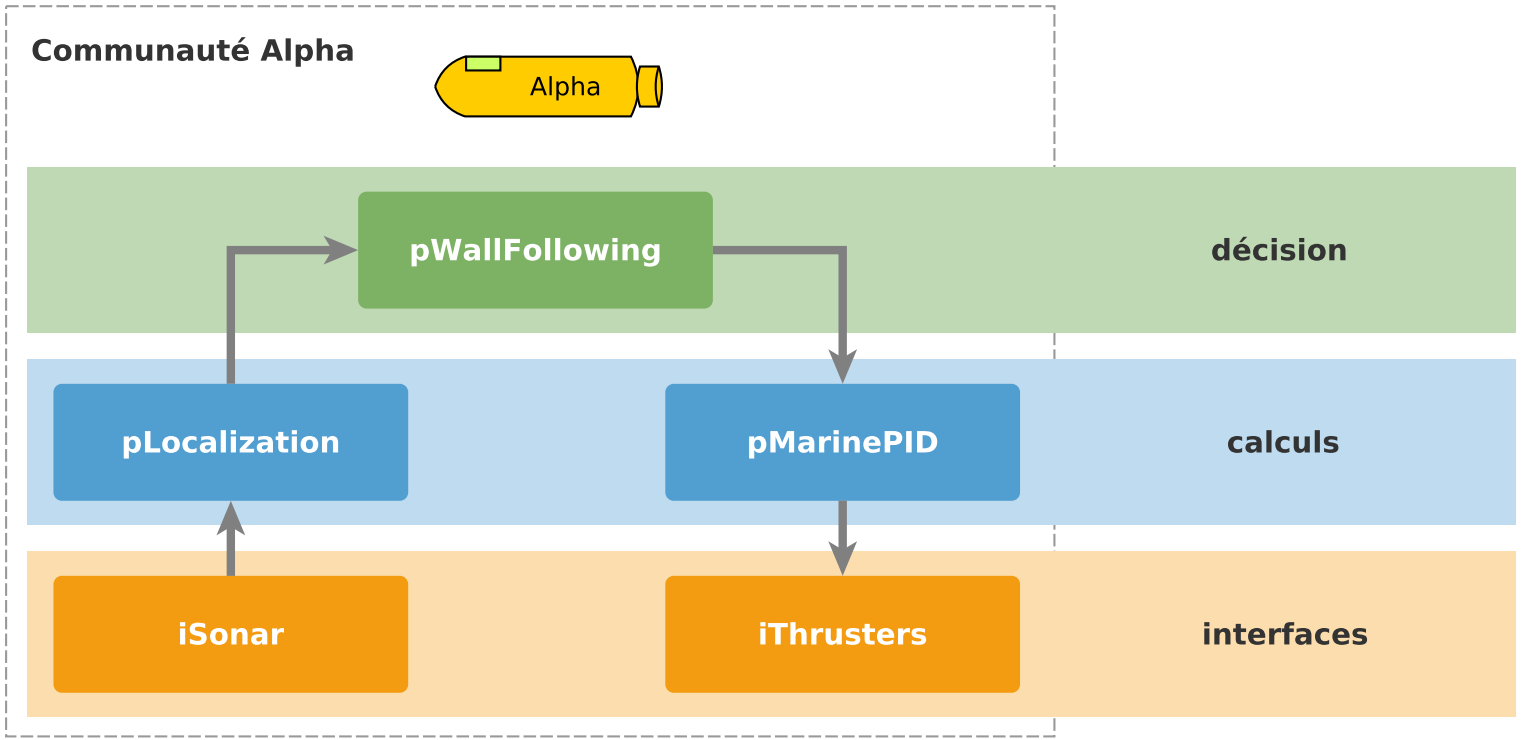
Section 2

Architecture publish-subscribe

Architecture publish-subscribe

Notion de communauté

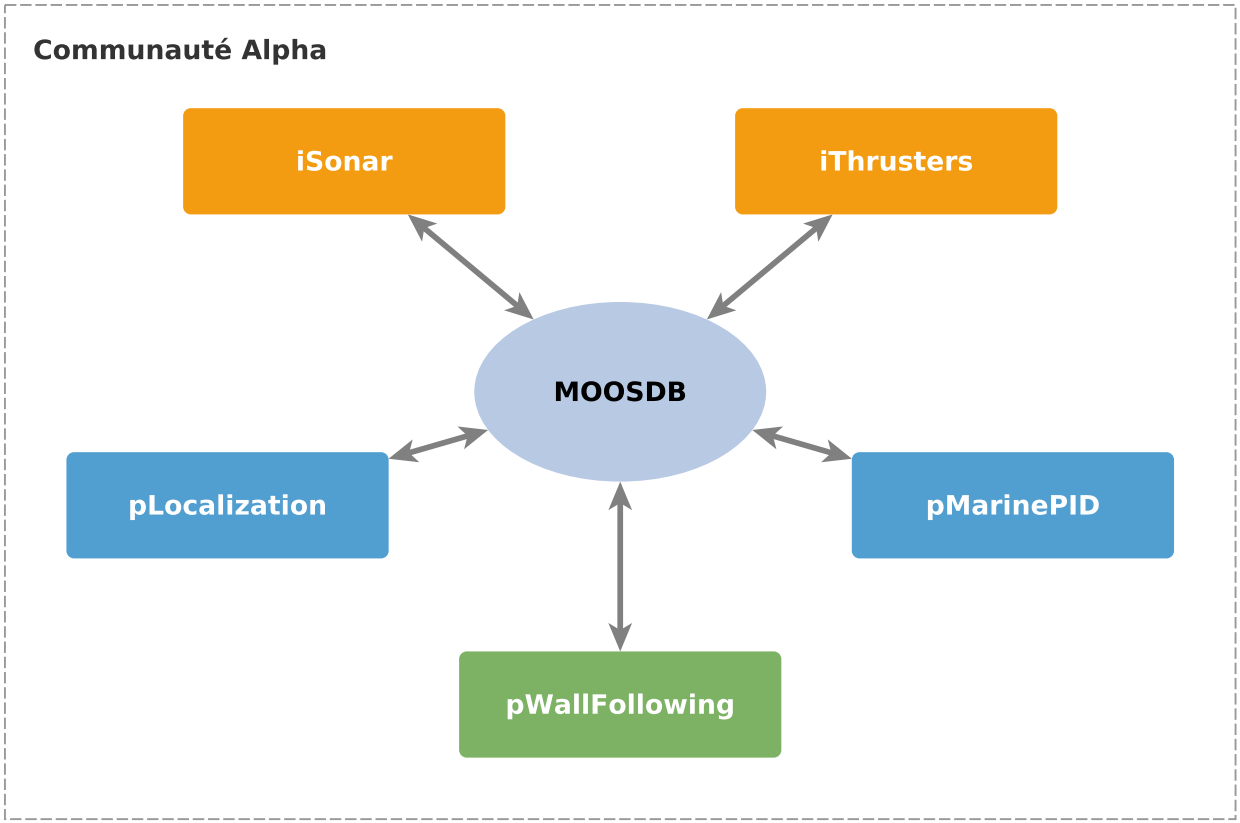
En général on associe un robot à une communauté de programmes. L'exemple ci-dessous rassemble 5 applications permettant à un robot sous-marin de faire un suivi de mur.



Architecture publish-subscribe

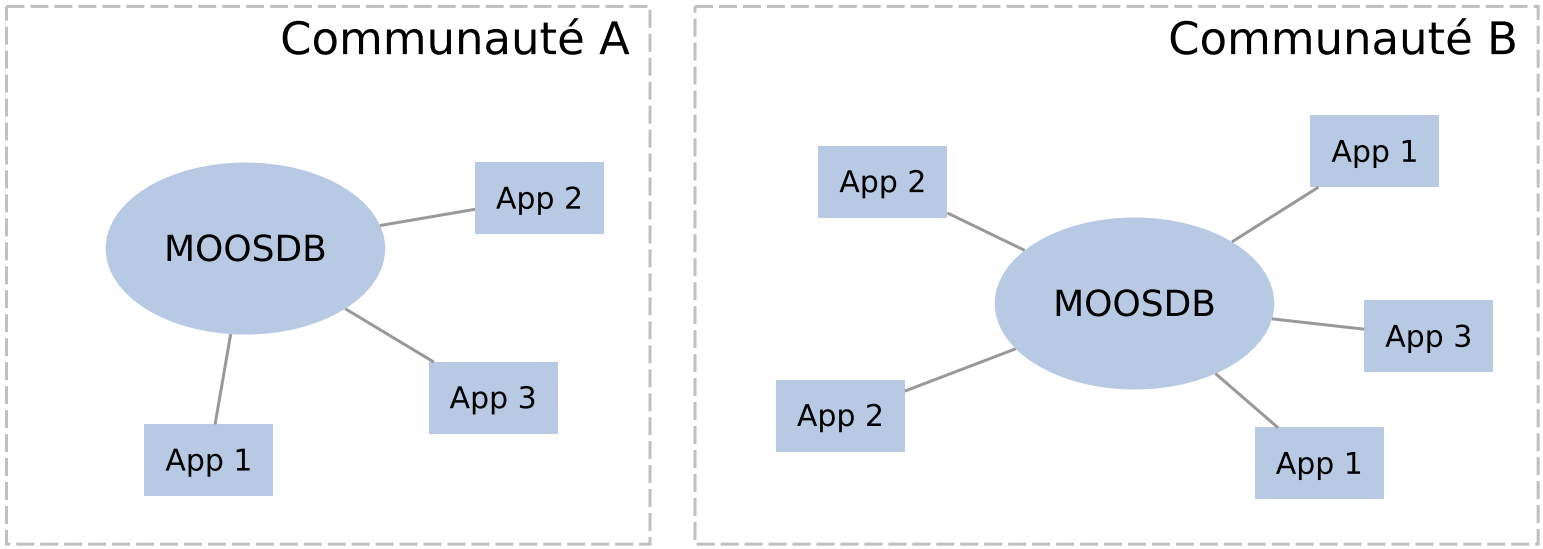
Notion de communauté

Dans MOOS, les programmes communiquent à travers une base de données centrale appelée MOOSDB :



Architecture publish-subscribe

Notion de communauté

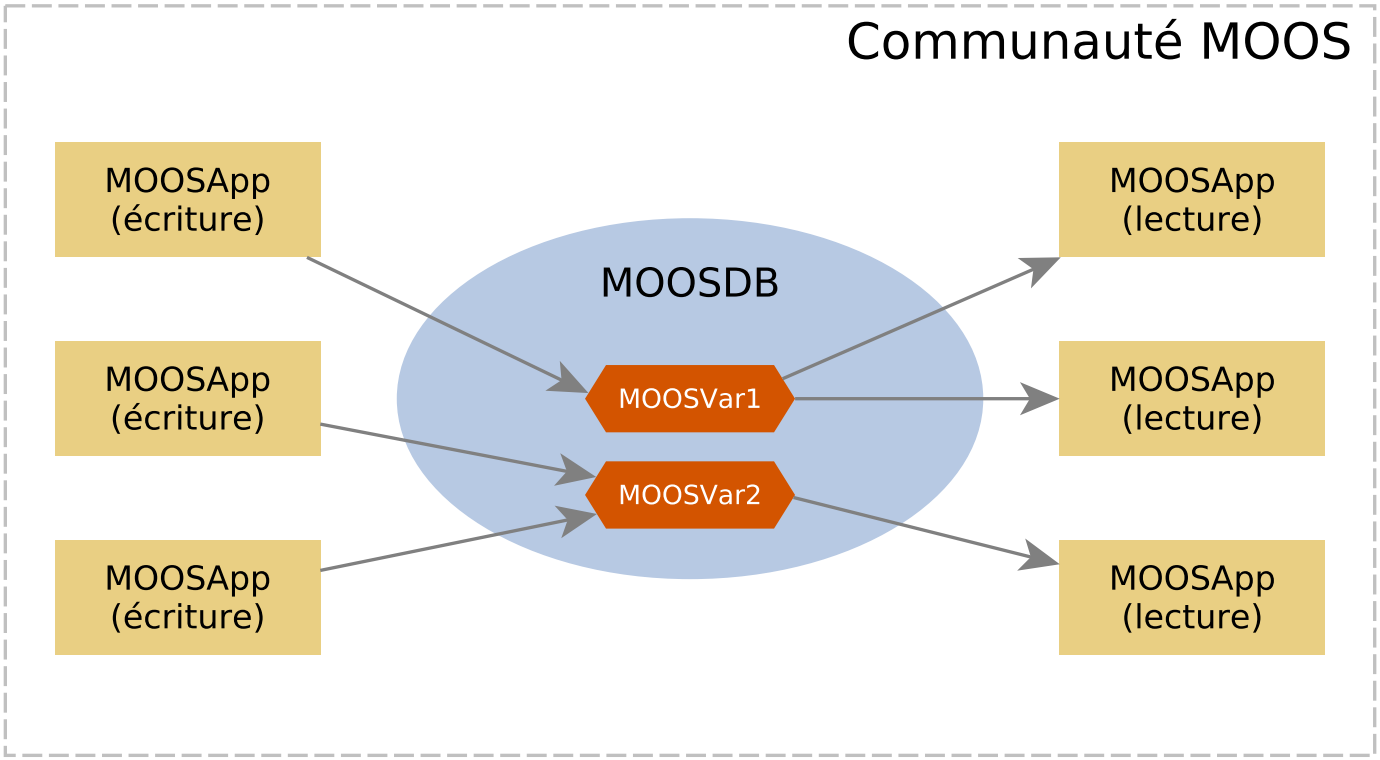


Plusieurs communautés peuvent cohabiter sur une même machine.

Un programme peut avoir plusieurs instances dans une communauté.

Architecture publish-subscribe

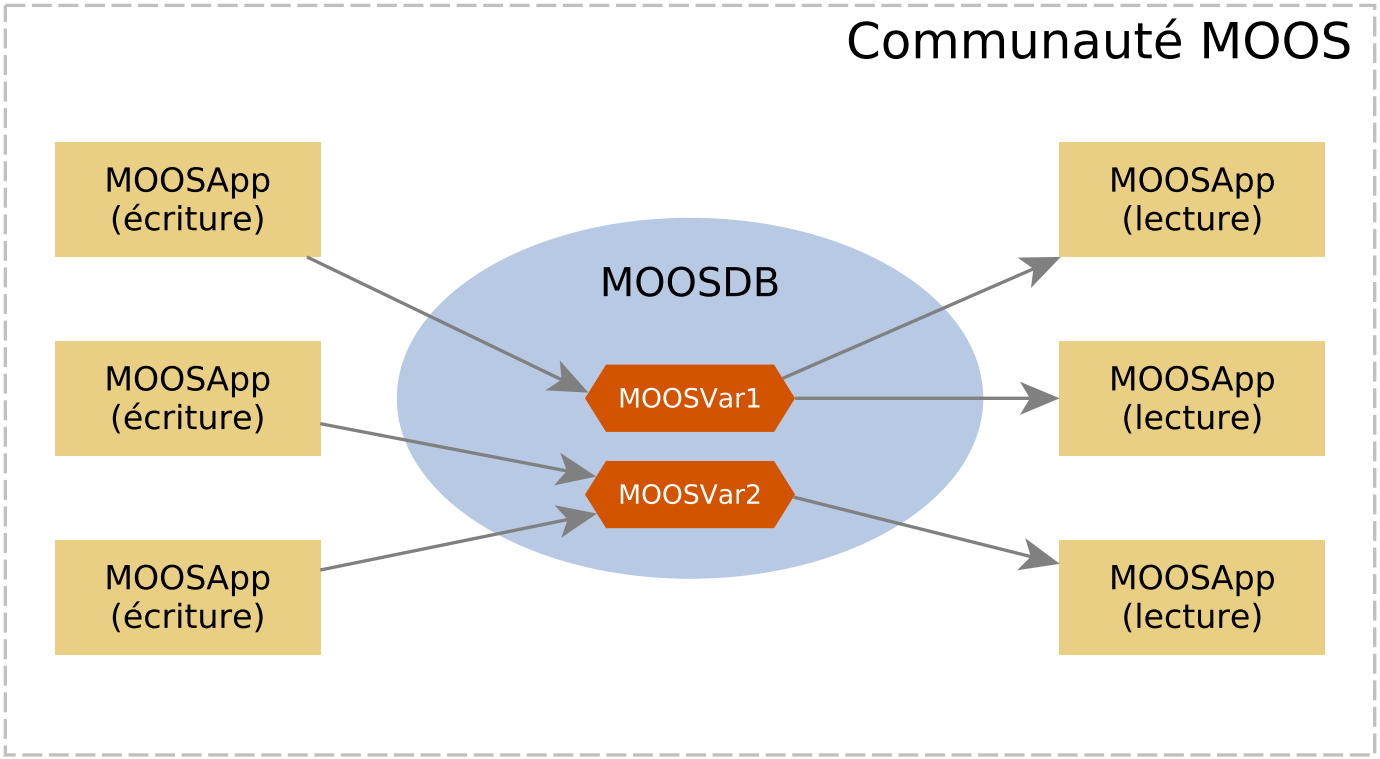
MOOSDB



MOOSDB : base de données centrale dans laquelle il est uniquement possible de voir l'état courant des variables qu'elle contient.

Architecture publish-subscribe

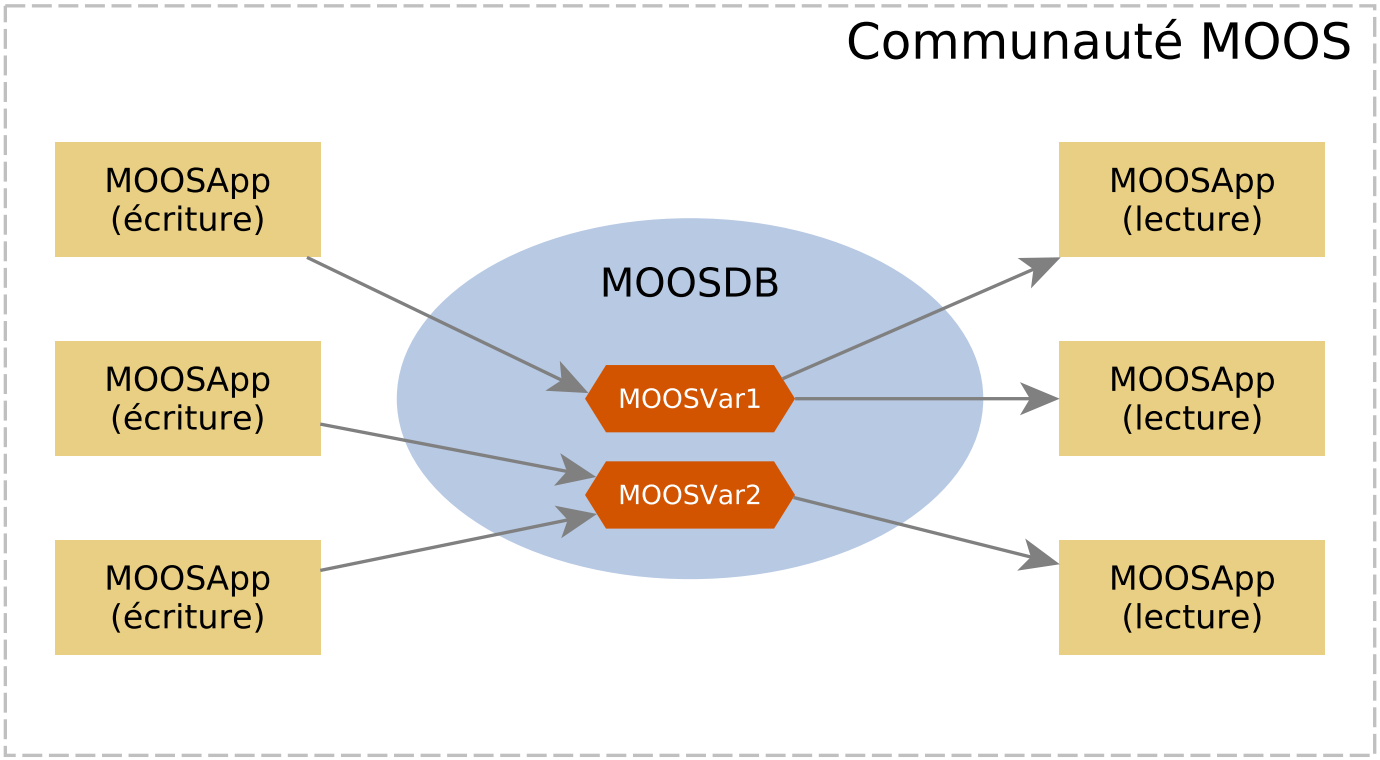
MOOSApp



MOOSApp : programme réalisant une tâche spécifique et capable de communiquer avec une MOOSDB.

Architecture publish-subscribe

MOOSVar



MOOSVar : variable partagée entre plusieurs MOOSApp.

Elle contient une donnée du type : *string*, *double* ou binaire

Architecture publish-subscribe

Communications

Lorsqu'une application A souhaite communiquer une information C à une application B, le processus suivant est déroulé :

1. B s'inscrit aux évolutions de la variable C dans la MOOSDB
2. A publie une nouvelle valeur pour C dans la MOOSDB
3. B reçoit une notification indiquant un changement sur C ; il est alors possible de récupérer la nouvelle valeur de la variable ainsi que son émetteur (MOOSApp), sa date de mise à jour, son type, etc.

Architecture publish-subscribe

Communications

Sur une même machine :

- communications par mémoire partagée

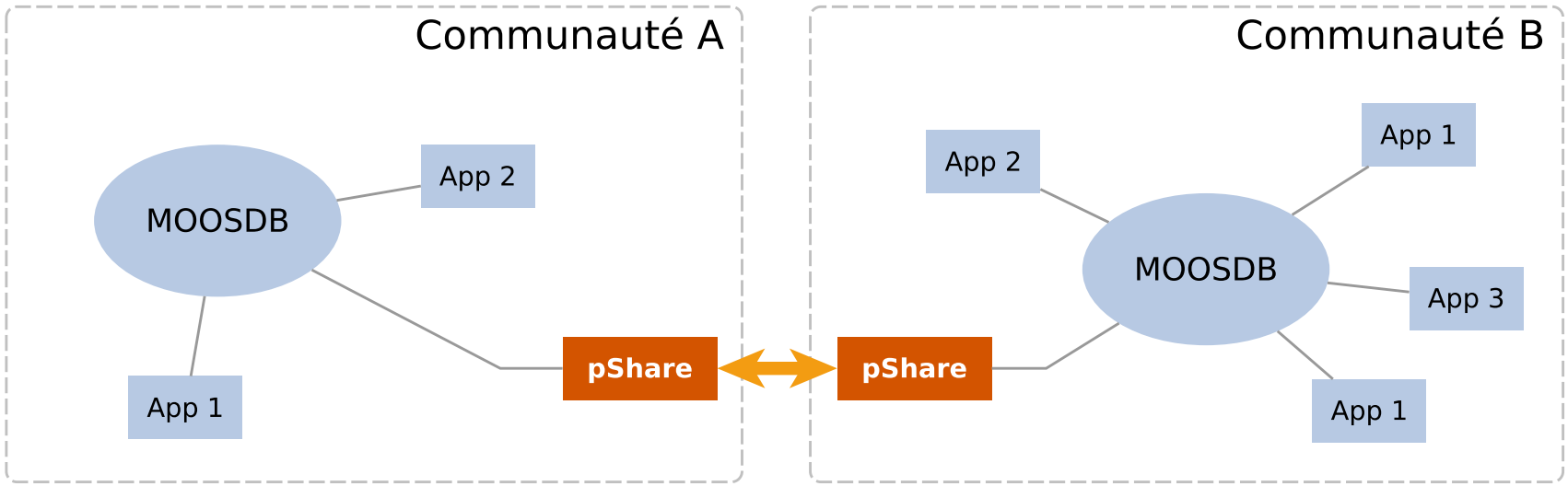
Entre plusieurs machines distantes :

- communications par protocole UDP
- chaque communauté est référencée par une adresse IP + port
- nécessite certaines MOOSApp dédiées (`pShare`, `uFldShoreBroker`, `uFldNodeBroker`, etc.)

Architecture publish-subscribe

Communications

Représentation des communications entre communautés :
(vue simplifiée)



Architecture publish-subscribe

Règles de nommage

On regroupe les MOOSApp dans trois catégories :

- **i** : *interface*, pour des connexions entre la communauté et des éléments externes de software/hardware
  ex : `iMatlab`, `iSonar`, `iThrusters`
- **p** : *process*, pour la logique métier pure (calculs, régulations, prises de décisions, ...)
  ex : `pLocalization`, `pWallFollowing`, `pHelmIvP`
- **u** : *utility*, regroupant des outils utilisés en dehors des missions réelles (visualisation, simulation, etc.)
  ex : `uPokeDB`, `uSimCurrent`, `uMS`

Par convention, une MOOSApp est préfixée par { **i** , **p** , **u** }.

Section 3

Les principaux outils

Les principaux outils

MOOSDB

MOOSDB :
La base de données à exécuter au lancement d'une mission.
Elle définit une nouvelle communauté.

```
> MOOSDB
```

```
> MOOSDB --moos_port=9000
```

Note : par défaut la MOOSDB est exécutée sur le port 9000

Les principaux outils

uPokeDB

Publication d'une variable dans la MOOSDB par ligne de commande :

```
> uPokeDB DESIRED_SPEED=2.0
```

Résultat :

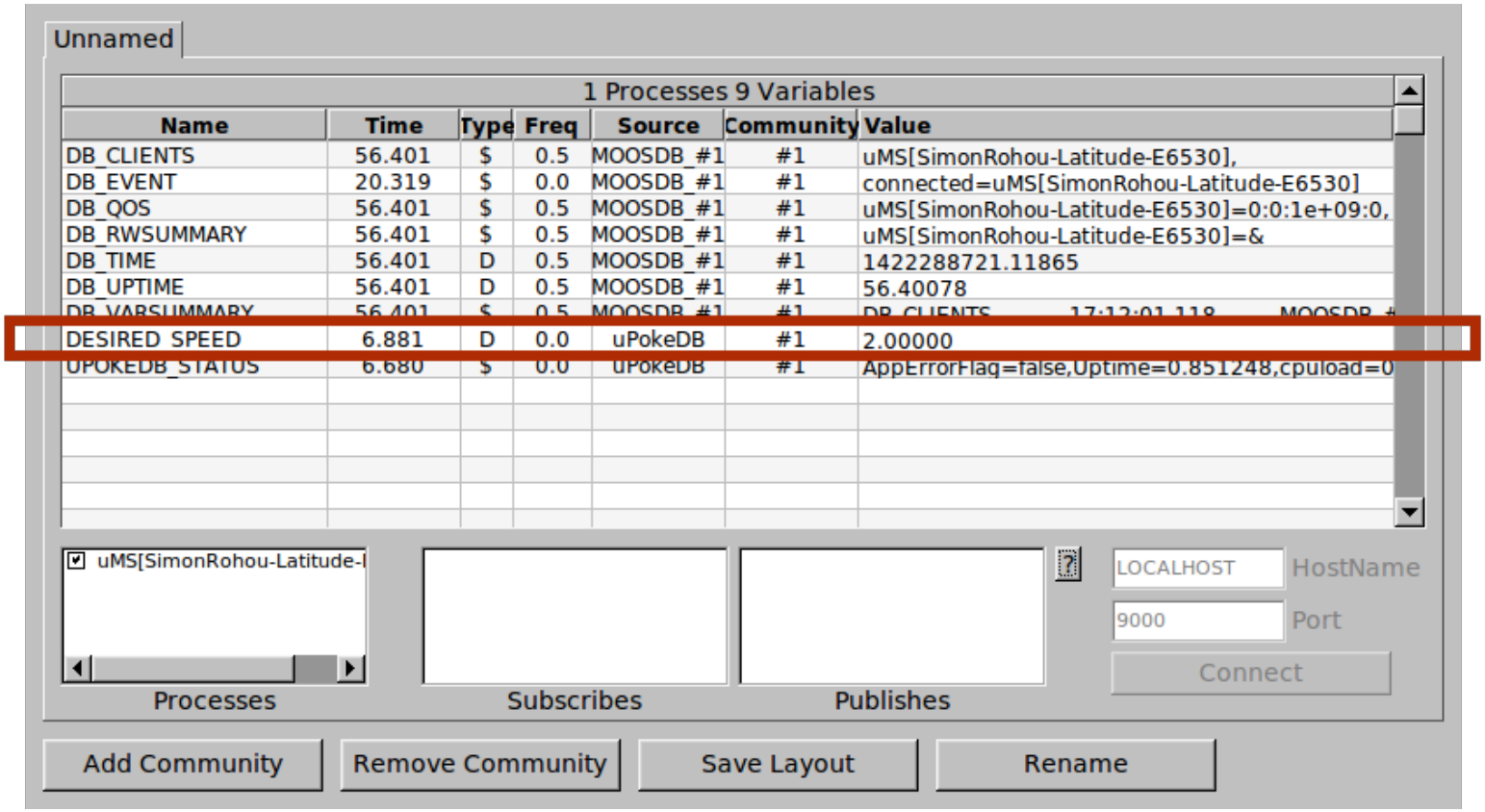
```
PRIOR to Poking the MOOSDB
VarName              (S)ource   (T)ime     VarValue
---------------    ---------- ----------  -------------
DESIRED_SPEED


AFTER Poking the MOOSDB
VarName              (S)ource   (T)ime     VarValue
---------------    ---------- ----------  -------------
DESIRED_SPEED        uPokeDB    37.28       2.00000
```

Les principaux outils

uMS

Visualisation des variables de la MOOSDB :



Les principaux outils

pAntler

Exécution de plusieurs MOOSApp avec leurs configurations :

```
> pAntler mission.moos
```

Le fichier `mission.moos` contient :

- la configuration de la communauté MOOS
- la liste des MOOSApp à exécuter
- le bloc de configuration de chaque MOOSApp

Les principaux outils

pAntler : mission.moos

Exemple : le fichier `mission.moos`
Configuration pour le lancement de trois programmes.

```
1   Community = alpha // le nom de la communauté MOOS
2   ServerHost = localhost // l'adresse du serveur
3   ServerPort = 9000 // le port de la communauté
4
5   ProcessConfig = ANTLER
6   {
7     // la base de données :
8     Run = MOOSDB          @ NewConsole = false
9     // un nouveau programme de localisation du robot :
10    Run = pLocalization   @ NewConsole = true
11    // une interface de visualisation des variables MOOS :
12    Run = uMS             @ NewConsole = false
13  }
14  ...
```

Les principaux outils

pAntler : mission.moos

Exemple : le fichier `mission.moos` **(suite)**

Les paramètres d'une MOOSApp sont à renseigner ici :

```
15    ...
16
17    ProcessConfig = pLocalization
18    {
19      MAX_SPEED = 2
20      MAX_RANGE = 10.5
21      SLAM = true
22      LOC_METHOD = intervals
23      BEACONS_NAME = beacon1=alpha, beacon2=bravo, beacon3=charlie
24      BEACONS_POS = 0.0:0.0, 50.0:10.0, -90.0:30.0
25    }
```

Les principaux outils

pAntler : deux instances d'une MOOSApp

Exemple : le robot dispose de deux sonars.

```
1   ProcessConfig = ANTLER
2   {
3     ...
4     Run = iSonar    @ NewConsole = true ~iSonar_bottom
5     Run = iSonar    @ NewConsole = true ~iSonar_front
6   }
7
8   ProcessConfig = iSonar_bottom
9   {
10    SERIAL_PORT = /dev/ttyUSB0 // paramètre propre à iSonar
11  }
12
13  ProcessConfig = iSonar_front
14  {
15    SERIAL_PORT = /dev/ttyUSB3 // paramètre propre à iSonar
16    MAX_RANGE = 50             // paramètre propre à iSonar
17  }
```

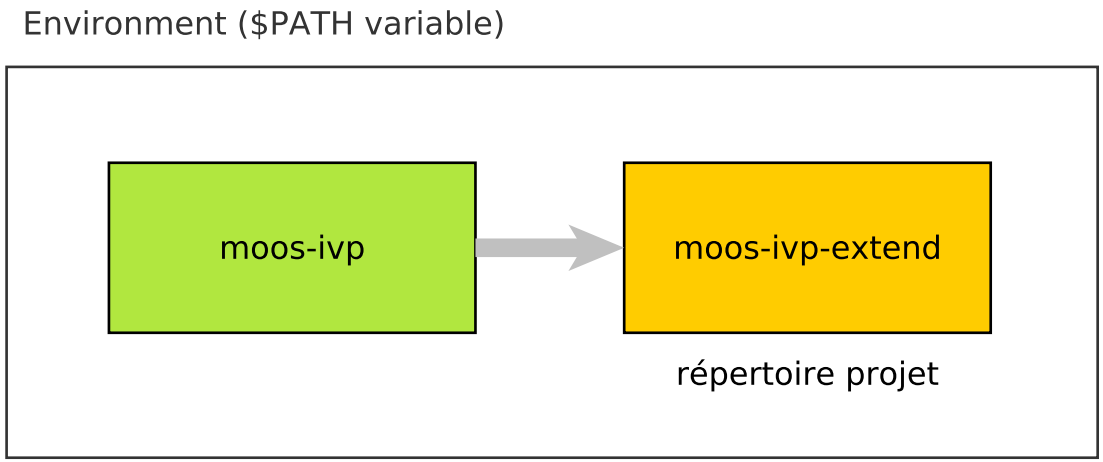
Section 4

Implémentation

Implémentation

Un projet utilisant MOOS-IvP

Le développement d'un projet se fait dans un répertoire dédié.
Exemple : `moos-ivp-extend`.

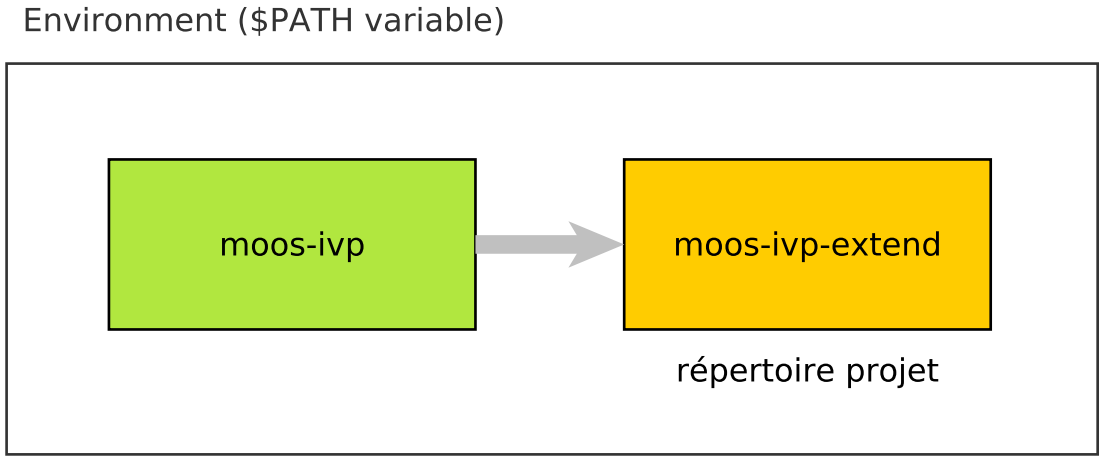


Un squelette de ce répertoire peut être téléchargé :

```
> svn co https://oceanai.mit.edu/svn/moos-ivp-extend/trunk ~/moos-ivp-extend
```

Implémentation

Un projet utilisant MOOS-IvP



Pour la compilation et l'exécution du projet, il est nécessaire d'avoir accès à `moos-ivp` et `moos-ivp-extend`. Sous Linux, il faut mettre à jour les variables d'environnement du système :

```
export PATH=$PATH:~/moos-ivp/bin
export PATH=$PATH:~/moos-ivp-extend/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/moos-ivp/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/moos-ivp-extend/lib
```

Implémentation

MOOSApp : génération automatique

Génération automatique d'une nouvelle MOOSApp dans le répertoire projet `moos-ivp-extend` :

```
> GenMOOSApp_AppCasting Localization p "ENSTA Bretagne"
```

Un nouveau répertoire `pLocalization` contiendra les fichiers à compléter. Dans cet exemple, l'application créée sera de type *process* (`p`). L'auteur *ENSTA Bretagne* sera renseigné dans l'entête de chaque fichier.

Implémentation

MOOSApp : génération automatique

Exemple : le répertoire `pLocalization` contiendrait :

- `main.cpp` : lancement de la MOOSApp
- `Localization.cpp` : classe principale
- `Localization.h` : classe principale
- `Localization_Info.cpp` : documentation
- `Localization_Info.h` : documentation
- `pLocalization.moos` : un exemple de fichier de configuration
- `CMakeLists.txt` : configurations de compilation

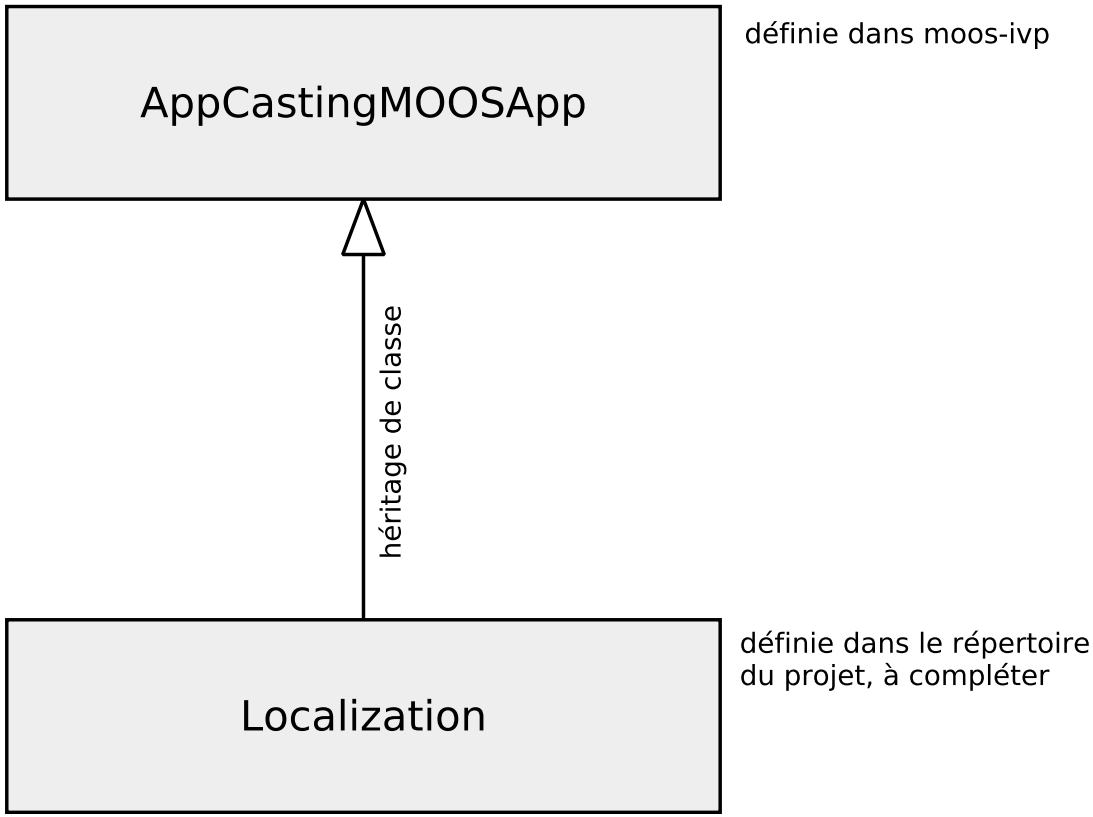
Ces fichiers sont à compléter.
Une application MOOS s'implémente en C++ [2].

2. une implémentation en Python est désormais possible

Implémentation

Héritage de la classe AppCastingMOOSApp

La classe `Localization` est à compléter. Elle dérive de la classe MOOS `AppCastingMOOSApp` et est instanciée dans `main.cpp`.



Implémentation

Héritage de la classe AppCastingMOOSApp

Dans la classe MOOS du programme :

Méthodes d'initialisation de l'application :

1. Chargement des paramètres : `OnStartUp()`
2. Connexion à la MOOSDB : `OnConnectToServer()`
3. Abonnement aux variables MOOS : `registerVariables()`

Puis, exécution du programme *via* deux méthodes à compléter :

- une fonction de réception de nouveaux mails
  appelée de manière évènementielle : `OnNewMail()`
- une fonction récurrente
  appelée automatiquement et régulièrement à une fréquence souhaitée : `Iterate()`

Implémentation

Héritage de la classe AppCastingMOOSApp

```
1   class Localization : public AppCastingMOOSApp
2   {
3     public:
4       Localization();
5
6     protected:
7       // Chargement des configurations du fichier .moos
8       bool OnStartUp();
9       // Initialisations après connexion à la MOOSDB :
10      bool OnConnectToServer();
11      // Enregistrement des abonnements aux MOOSVar :
12      void registerVariables();
13      // Méthode exécutée dès la réception d'une MOOSVar :
14      bool OnNewMail(MOOSMSG_LIST &NewMail);
15      // Exécution du programme principal :
16      bool Iterate();
17  };
```

Implémentation

MOOSApp - méthode registerVariables()

Exemple :
L'application `pLocalization` s'abonne à la variable `SONAR_DATA` :

```
1  void Localization::registerVariables()
2  {
3    AppCastingMOOSApp::RegisterVariables(); // héritage
4    Register("SONAR_DATA");
5  }
```

Implémentation

MOOSApp - méthode OnNewMail()

Notification de changement sur SONAR_DATA :

```
1   bool Localization::OnNewMail(MOOSMSG_LIST &NewMail)
2   {
3     ...
4
5     // Plusieurs MOOSVar peuvent être reçues en même temps
6     // Un iterator permet de les parcourir :
7     MOOSMSG_LIST::iterator p;
8     for(p = NewMail.begin() ; p != NewMail.end() ; p++)
9     {
10      CMOOSMsg &msg = *p; // l'objet correspondant à la MOOSVar
11      string key = msg.GetKey(); // le nom de la MOOSVar
12
13      if(key == "SONAR_DATA") {
14        ... // traitement des données
15      }
16    }
17  }
```

Implémentation

MOOSApp - méthode OnNewMail()

Une MOOSVar est accessible dans un objet `CMOOSMsg &msg`.
Récupération des informations sur la variable :

```
1   // la communauté émettrice
2   string comm = msg.GetCommunity();
3   // data (double value)
4   double dval = msg.GetDouble();
5   // data (string value)
6   string sval = msg.GetString();
7   // nom de la MOOSApp émettrice
8   string msrc = msg.GetSource();
9   // date d'émission de la variable
10  double mtime = msg.GetTime();
```

Implémentation

MOOSApp - méthode Iterate()

La méthode `Iterate` est automatiquement appelée à intervalles de temps réguliers :

```
1  bool Localization::Iterate()
2  {
3    AppCastingMOOSApp::Iterate();
4
5    ... // à remplir
6
7    AppCastingMOOSApp::PostReport();
8    return true;
9  }
```

La fréquence d'appel est fixée par le paramètre `AppTick`.

Implémentation

MOOSApp - publication d'une MOOSVar

Exemple : publication d'une variable `ESTIM_X` :

```
1    double x_estimation = 5.0;
2    Notify("ESTIM_X", x_estimation);
```

`Notify()` est applicable dans `OnNewMail()` et `Iterate()`

Implémentation

Paramétrage d'une application : fréquences d'exécutions

Dans le fichier de configuration `.moos` :

- `AppTick` : fréquence d'exécution de la méthode `Iterate()`
- `CommsTick` : fréquence de communications avec la MOOSDB

```
1   //------------------------------------------
2   // pLocalization config block
3
4   ProcessConfig = pLocalization
5   {
6     AppTick   = 4 // Hz
7     CommsTick = 4 // Hz
8
9     ...
10  }
```

Implémentation

Paramétrage d'une application : paramètres personnalisés

Rappel : des paramètres peuvent être définis dans le fichier `.moos`.

```
15   ProcessConfig = pLocalization
16   {
17     AppTick   = 4 // Hz
18     CommsTick = 4 // Hz
19
20     SLAM = true
21     MAX_SPEED = 2
22     MAX_RANGE = 10.5
23     LOC_METHOD = intervals
24     BEACONS_NAME = beacon1=alpha, beacon2=bravo, beacon3=charlie
25     BEACONS_POS = 0.0:0.0, 50.0:10.0, -90.0:30.0
26   }
```

Ces paramètres sont récupérés dans la méthode `OnStartUp()`.

Implémentation

MOOSApp - méthode OnStartUp()

La méthode `OnStartUp` est automatiquement appelée.

```
1   bool Localization::OnStartUp()
2   {
3     ... // préparation de la liste des paramètres
4
5     STRING_LIST::reverse_iterator p;
6     for(p = sParams.rbegin() ; p != sParams.rend() ; p++)
7     {
8       string orig  = *p; string line  = *p;
9       string param = toupper(biteStringX(line, '='));
10      string value = line;
11
12      if(param == "MAX_SPEED") // enregistrement d'une valeur numérique (double)
13        m_max_speed = atof(value.c_str()); // cast : std::string > const char* > double
14
15      else if(param == "LOC_METHOD") // enregistrement d'une chaîne de caractères (string)
16        m_loc_method = value; // pas de cast nécessaire
17
18      else // notification de paramètre inconnu (warning)
19        reportUnhandledConfigWarning(orig);
20    }
21
22    ...
23  }
```

Section 5

Et après ?

Et après ?

De l'autonomie avec IvP

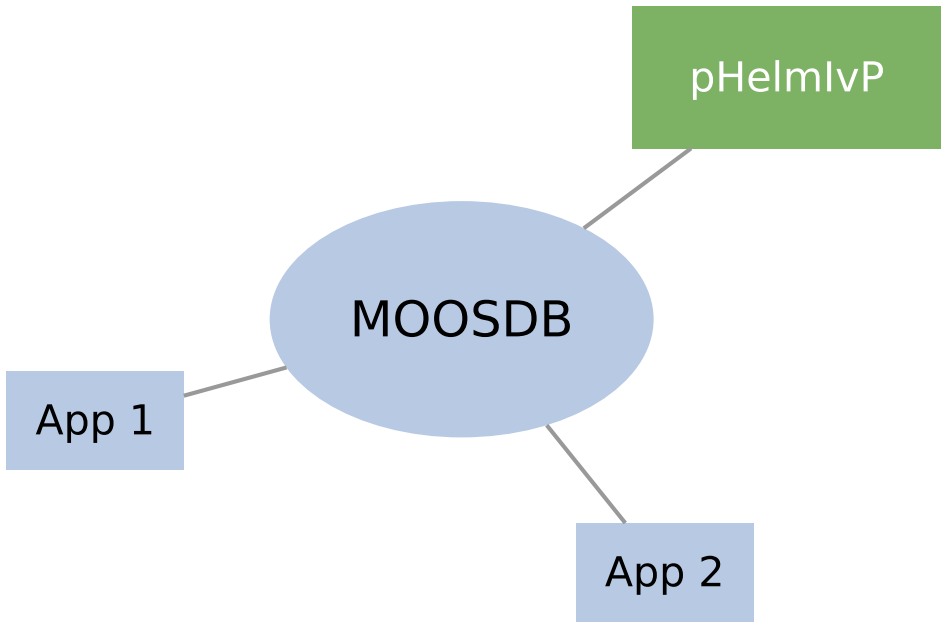
L'extension IvP présente beaucoup d'outils intéressants voire incontournables pour un utilisateur de MOOS.

En voici un (très) bref aperçu.

Et après ?

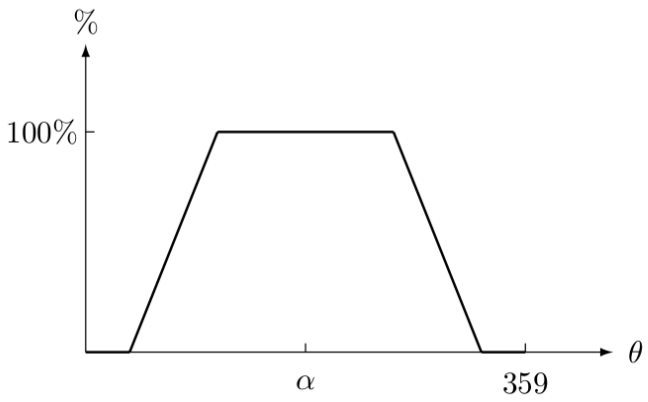
De l'autonomie avec IvP

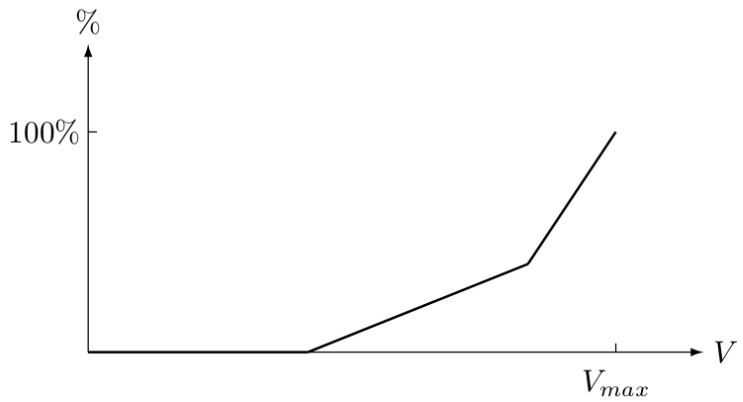
IvP permet à un robot d'adopter un comportement en fonction d'une situation donnée. Le robot devient alors autonome en prenant des décisions.
Ce processus est hébergée par une nouvelle MOOSApp :



Et après ?

De l'autonomie avec IvP

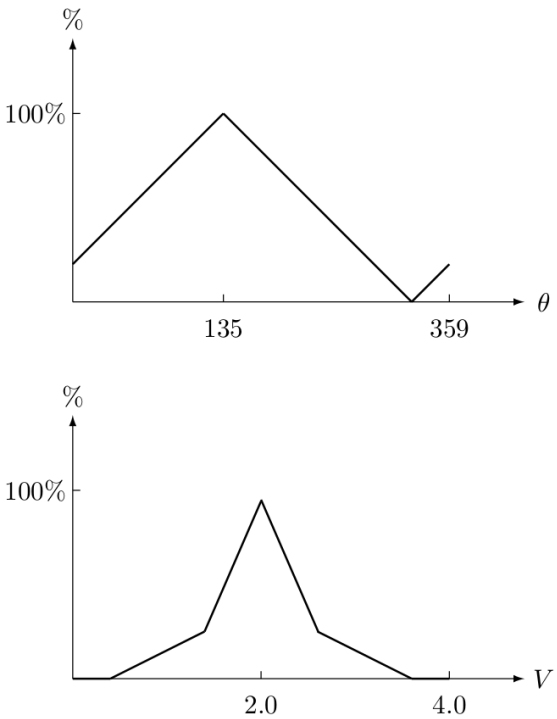


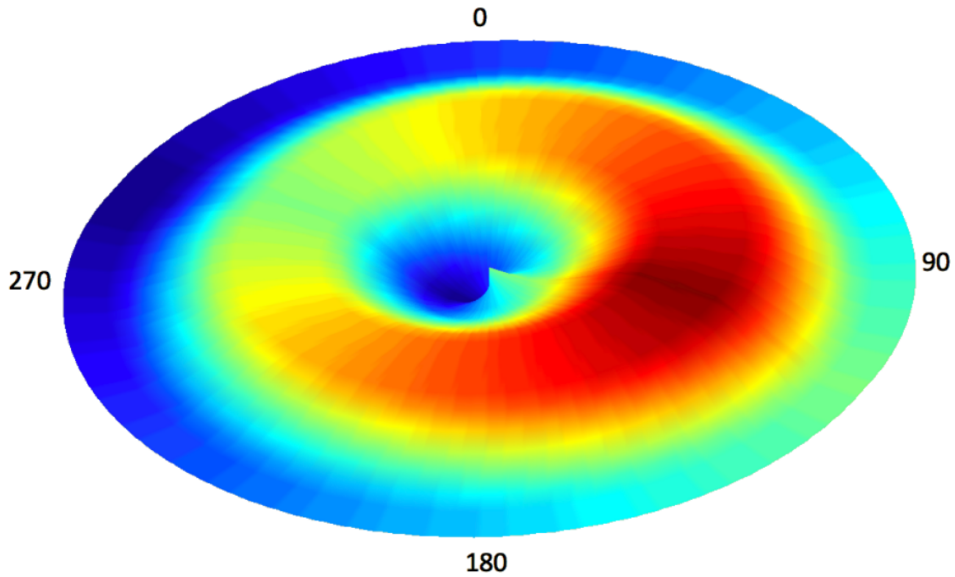


Des fonctions de prise de décision

Et après ?

De l'autonomie avec IvP

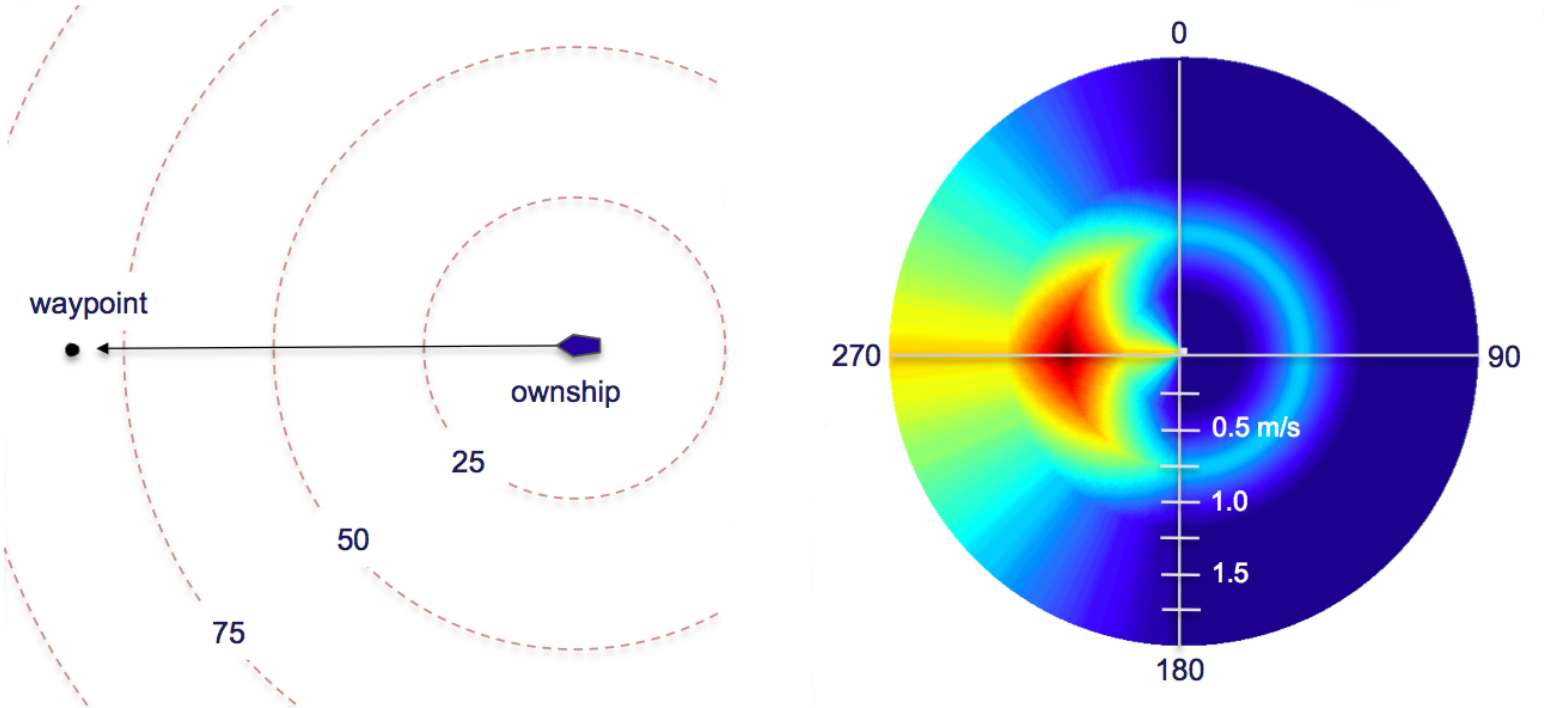




Fusion de prises de décisions

Et après ?

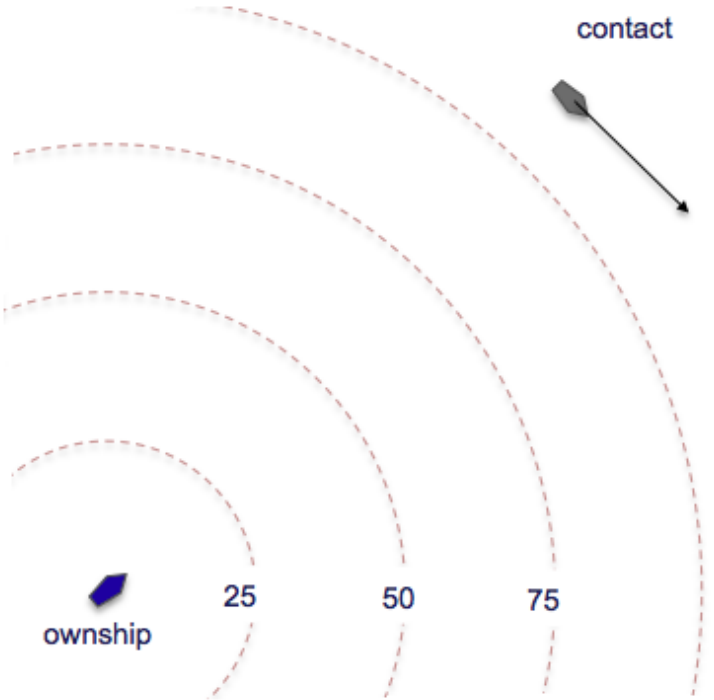
De l'autonomie avec IvP

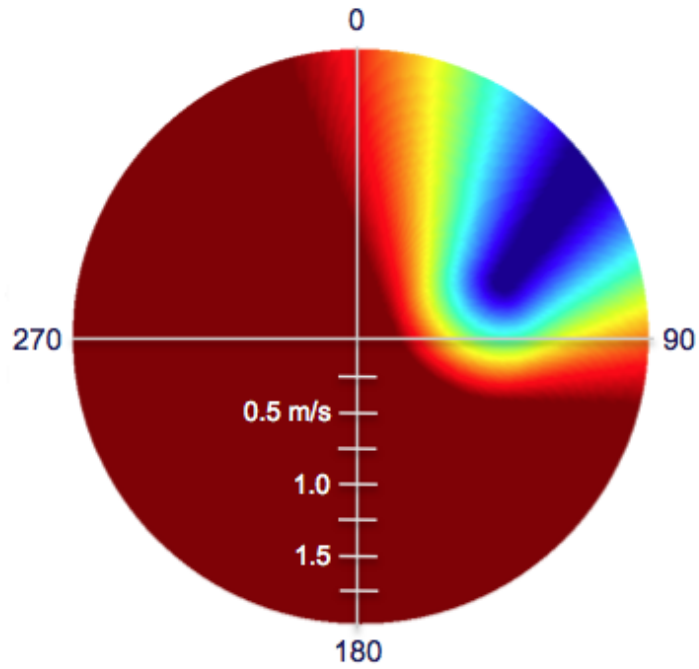


Suivi de cibles

Et après ?

De l'autonomie avec IvP





Évitement d'obstacles

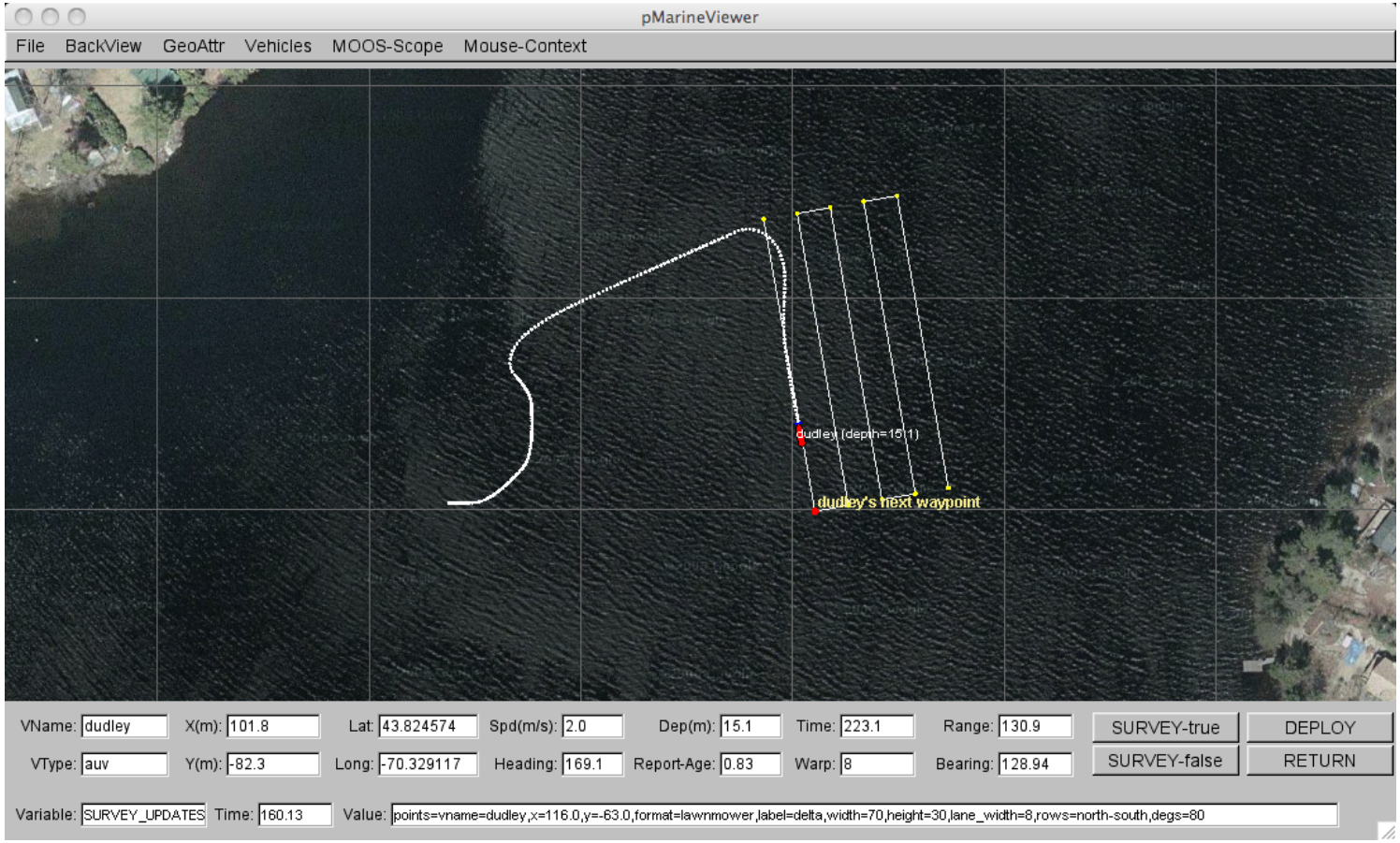
Et après ?

De l'autonomie avec IvP

Évitement d'obstacles

Et après ?

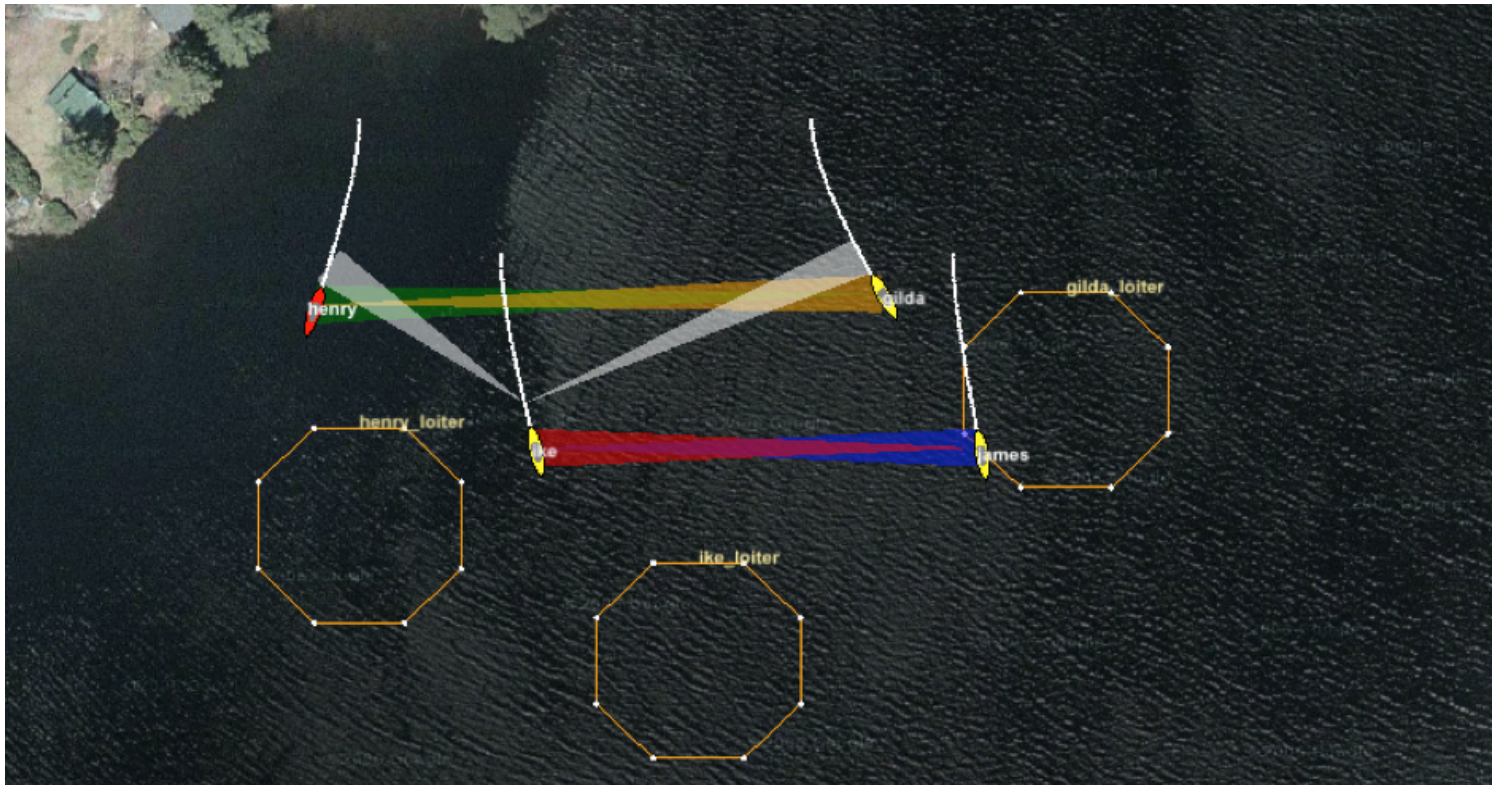
De l'autonomie avec IvP



Suivi de lignes

Et après ?

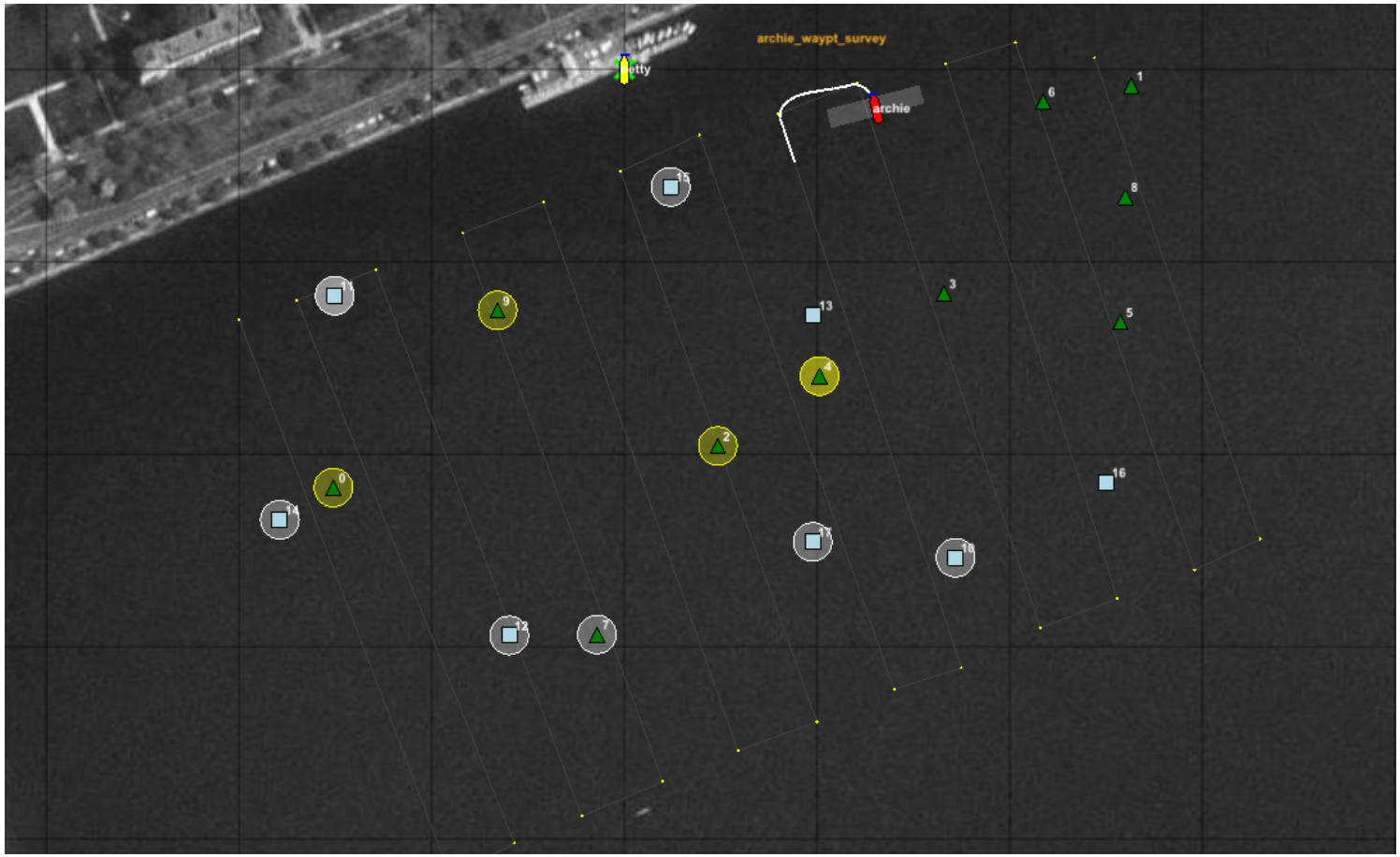
De l'autonomie avec IvP



Communications multi-robots

Et après ?

De l'autonomie avec IvP



Et bien d'autres...

Section 6

Installation de MOOS-IvP

Installation de MOOS-IvP

Procédure

Installation de `moos-ivp` :

```
> sudo apt-get install g++ subversion xterm cmake \
                       libfltk1.3-dev freeglut3-dev libpng-dev \
                       libjpeg-dev libxft-dev libxinerama-dev libtiff5-dev
> svn co https://oceanai.mit.edu/svn/moos-ivp-aro/releases/moos-ivp-17.7.2 ~/moos-ivp
> cd ~/moos-ivp
> ./build.sh
```

Configuration des chemins d'accès ; ajouter à la fin de `~/.bashrc` :

```
1  export PATH=$PATH:~/moos-ivp/bin
2  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/moos-ivp/lib
3  export PATH=$PATH:~/moos-ivp-extend/bin
4  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/moos-ivp-extend/lib
```

Préparation du répertoire de travail `moos-ivp-extend` :

```
> svn co https://oceanai.mit.edu/svn/moos-ivp-extend/trunk ~/moos-ivp-extend
> cd ~/moos-ivp-extend
> ./build.sh
```

Références

- Page officielle de la V10 de MOOS (Oxford)
- Site officiel de MOOS-IvP
- Documentation en ligne de MOOS-IvP

Publication :

– M. Benjamin, H. Schmidt, P. Newman, J. Leonard,
Nested Autonomy for Unmanned Marine Vehicles with MOOS-IvP,
Journal of Field Robotics, Volume 27, Issue 6, pp. 834-875,
November 2010.

ENSTA Bretagne