



Documentation: Informatique



Programme principal : Coupe 2013

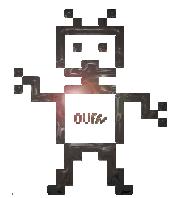
Cette documentation a pour but de présenter le programme informatique mis en place dans le robot pour la coupe de France de robotique 2013

I. Présentation Générale

Le projet de dev Oufff2013 est composé de la manière suivante :

Oufff2013\	: Racine du projet
- Conception	: Dossier contenant les fichiers de conception
- Projets	: Dossier contenant les environnements des IDE
- MPLAB_ocore_v1	: Environnement de dev de MPLAB
- VC2005	: Environnement de dev de MS VC2005
- VC2011	: Environnement de dev de MS VC2011
- src	: Code source
- BSP	: Code relatif au BSP
- Main	: Code du programme principal
- Strategy	: Code relatif à toutes les stratégies du robot
- Microchip	: Code MPLAB
- uC-CPU	: Code MPLAB
- uC-LIB	: Code MPLAB
- uCOS-II	: Code MPLAB

Tous les projets contenus dans le répertoire « Projets » font appels aux fichiers du répertoire « src ». Seules les constantes d'environnement du projet peuvent varier d'un projet à l'autre. Dans la suite de ce document, nous allons nous concentrer sur les fichiers contenus dans le répertoire « src/Main ».



Documentation:

Informatique

Présentation du contenu du répertoire « src/Main » :

src\Main

- App.h : Déclaration de toutes les variables globales
- App.c : Init et démarrage de toutes les tâches
- AppConfig.h : Définition de toutes les constantes de config
- AppCustomTypes.h : Définition de tous les types personnalisés
- AppGlobalFunc.* : Fonctions communes à toutes les tâches
- AppGlobalVars.h : Permet l'include de toutes les variables globales
- AppHooks.c : Fichier contenant les actions pour le hook
- AppIncludes.h : Contient tous les includes nécessaires du projet
- LibMoving.* : Lib de mouvement
- os_cfg.h : Configuration de l'OS
- Strategy\ : Répertoire contenant les fichiers des stratégies
- | - StrategyColorA.c : Stratégie pour la couleur A
- | - StrategyColorB.c : Stratégie pour la couleur B
- | - HomolColorA.c : Stratégie d'homologation pour la couleur A
- | - HomolColorB.c : Stratégie d'homologation pour la couleur B
- TaskAsser.* : Code de la tâche Asservissement
- TaskMain.* : Code de la tâche Principale
- TaskMvt.* : Code de la tâche Mouvement
- TaskOdo.* : Code de la tâche Odométrie
- TaskSensors.* : Code de la tâche gérant tous les capteurs
- TaskTempo.* : Code de la tâche gérant le temps

NB : Les fichiers notés « NomDuFichier.* » symbolise les fichiers « NomDuFichier.c » et « NomDuFichier.h »

II. Définition des types personnalisés

a) Les Enumérations

EnumColor	Donne la couleur
Valeurs Possibles	
c_NotSet	Couleur non définie
c_ColorA	Couleur A. Pour la coupe 2013 c'est du bleu
c_ColorB	Couleur B. Pour la coupe 2013 c'est du rouge



EnumCmd	Définit le type de la commande à passer entre toutes les tâches	
Valeurs Possibles		
Cmd_NotSet	0	Commande non définie
Mvt_UseAngleOnly	10	Effectuer un mouvement avec le mode d'asservissement 1
Mvt_UseDistOnly	11	Effectuer un mouvement avec le mode d'asservissement 2
Mvt_UseMixedMode	12	Effectuer un mouvement avec le mode d'asservissement 3
Mvt_UsePivotMode	13	Effectuer un mouvement avec le mode d'asservissement 4
Mvt_UseSpline	14	Effectuer un mouvement en spline
MvtSimple_MoveInMM	15	Mvt simple (pas de division) : Aller en ligne droite
MvtSimple_RotateInDeg	16	Mvt simple (pas de division) : Tourner d'un angle en degré
MvtSimple_RotateToAngleInDeg	17	Mvt simple (pas de division) : Tourner pour se mettre dans un angle précis par rapport au repère du terrain
Mvt_Stop	18	Arrêter l'action en cours
App_Wait	30	Attendre
App_IfGoto_System	31	Test une condition système et agit en conséquence
App_IfGoto_Strategy	32	Test une condition stratégique et agit en conséquence
App_SetNewPos	33	Modifier la position du robot
App_SetStrategyFlags	34	Permet de modifier un flag de stratégie

EnumCmdType	Définit le type de chaque commande	
Valeurs Possibles		
CmdType_NotSet	Type de commande non définie	
CmdType_Blocking	La commande est bloquante	
CmdType_NonBlocking	La commande est non bloquante	

EnumMsg	Donne tous les messages utilisés dans le programme	
Valeurs Possibles		
Msg_NotSet	Le message n'est pas défini	

b) Les Structures

StructPos		Structure utilisée par la TaskOdo pour donner la position actuelle du robot. Cette structure est aussi utilisée pour envoyer les ordres à la TaskAsser.
Contenu		
float	x	Position en x du robot
float	y	Position en y du robot
float	angle	Angle que fait le robot par rapport à l'origine (suivant x)
CPU_INT16U	right_encoder	Incrémentation de la roue droite pour un mouvement en pivot
CPU_INT16U	Left_encoder	Incrémentation de la roue gauche pour un mouvement en pivot
unsigned char	CurrentState	Flag pour indiquer le statut du mouvement (arrêté / en mvt)
Utilisation		
Pas d'information complémentaire		



Documentation:

Informatique



StructCmd		Structure utilisée pour passer des commandes d'une tâche à une autre		
Contenu				
<code>EnumCmd</code>	<code>Cmd</code>	Commande à exécuter		
<code>short</code>	<code>Param1</code>	Paramètre 1 de la commande		
<code>float</code>	<code>Param2</code>	Paramètre 2 de la commande		
<code>float</code>	<code>Param3</code>	Paramètre 3 de la commande		
<code>float</code>	<code>Param4</code>	Paramètre 4 de la commande		
<code>unsigned int</code>	<code>ActiveSensors Flag</code>	Indique quels sont les capteurs à utiliser pour l'anticollision lors de la prochaine commande		
<code>EnumCmdType</code>	<code>CmdType</code>	Action bloquante ou non		
Utilisation				
<code>Cmd</code>	<code>Param1</code>	<code>Param2</code>	<code>Param3</code>	<code>Param4</code>
<code>Cmd_NonSet</code>	/	/	/	/
<code>Mvt_UseAngleOnly</code>	Vitesse	/	/	Angle en °
<code>Mvt_UseDistOnly</code>	Vitesse	x	y	/
<code>Mvt_UseMixedMode</code>	Vitesse	x	y	Angle en °
<code>Mvt_UsePivotMode</code>	Vitesse	Roue à bloquer : <code>RIGHT_WHEEL</code> <code>LEFT_WHEEL</code>	/	Angle en °
<code>Mvt_UseSpline</code>	?	?	?	?
<code>Mvt_Simple_MoveInMM</code>	Vitesse	Distance en mm	/	/
<code>Mvt_Simple_RotateInDeg</code>	Vitesse	/	/	Angle en °
<code>Mvt_Simple_RotateToAngle_InDeg</code>	Vitesse	/	/	Angle en °
<code>Mvt_Stop</code>	/	/	/	/
<code>App_Wait</code>	Heure	Minute	Seconde	Milliseconde
<code>App_IfGoto_System</code>	Condition	Goto si vrai	Goto si faux	/
<code>App_IfGoto_Strategy</code>	Condition	Goto si vrai	Goto si faux	/
<code>App_SetNewPos</code>	/	x	y	Angle en °
<code>App_SetStrategyFlags</code>	?	?	?	?



Documentation:

Informatique



StructMsg		Structure utilisée pour passer des messages entre tâches
Contenu		
BOOLEAN	IsRead	Flag utilisé pour la gestion des messages
EnumCmd	Cmd	Commande à envoyer via le message
EnumCmdType	CmdType	Type de la commande
INT8U	Param1	Paramètre de la commande
Utilisation		
Cmd		Param1
Sensors_SetHolderStatus		HOLDER_CLOSE HOLDER_OPEN_LEFT_ONLY HOLDER_OPEN_RIGHT_ONLY HOLDER_OPEN HOLDER_GRAB HOLDER_GRIP
Sensors_SetHolderLevel		HOLDER_LEVEL_MIDDLE HOLDER_LEVEL_HIGH HOLDER_LEVEL_LOW

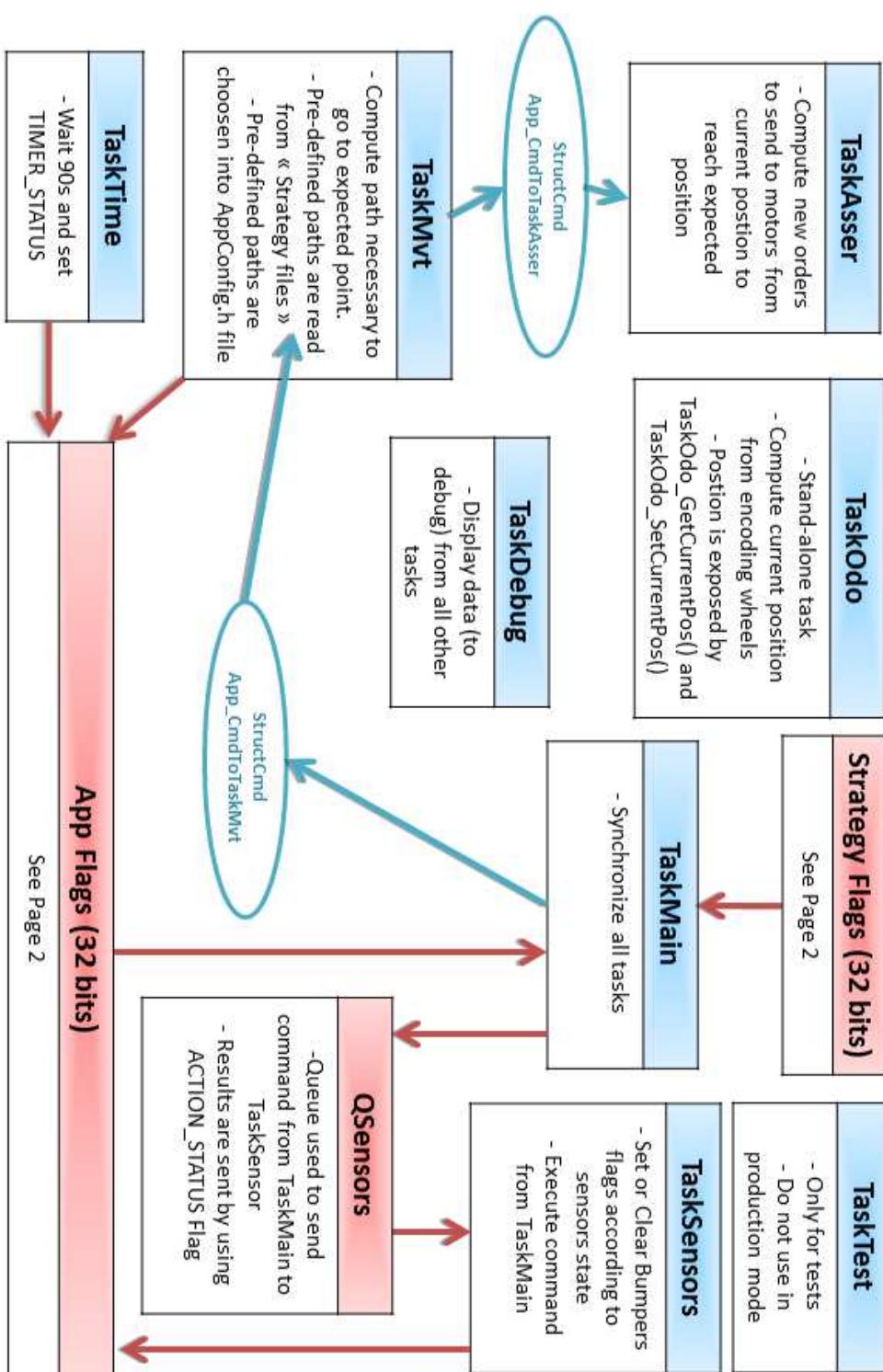


Documentation:

Informatique

III. Présentation Globale

Le programme se compose de la manière suivante :



Projet 2013

Documentation:

Informatique



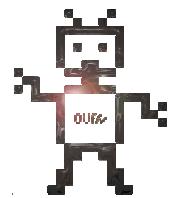
Strategy Flags (32 bits)			
<u>Actions Done</u>	<u>Forbidden Zones</u>	<u>Not Used</u>	<u>Not Used</u>
0 -	8 -	16 -	24 -
1 -	9 -	17 -	25 -
2 -	10 -	18 -	25 -
3 -	11 -	19 -	27 -
4 -	12 -	20 -	28 -
5 -	13 -	21 -	29 -
6 -	14 -	22 -	30 -
7 -	15 -	23 -	31 -

App Flags (32 bits)			
<u>State</u>	<u>Bumpers (GP2)</u>	<u>Bumpers (SW)</u>	<u>Not Used</u>
0 - START_BUTTON	8 - GP2_FRONT	16 -	24 -
1 -	9 - GP2_BACK	17 -	25 -
2 -	10 - GP2 HOLDER	18 -	26 -
3 -	11 -	19 -	27 -
4 -	12 -	20 -	28 -
5 -	13 -	21 -	29 -
6 - ACTION_STATUS	14 -	22 -	30 -
7 - TIMER_STATUS	15 -	23 -	31 -



Documentation:

Informatique



Le fichier « AppIncludes.h » permet d'inclure toutes les ressources nécessaires à l'exécution du programme. Il faut ajouter ce fichier dans les en-têtes des fichiers du projet pour utiliser toutes les fonctionnalités (constantes et fonctions) de la OufffTEAM.

Le fichier « App.c » est le point d'entrée du programme. C'est à partir de ce fichier que l'on lance toutes les tâches de l'application. Le processus de démarrage est le suivant :

App.c – main(...)

- | | |
|---|-----------------|
| 1. Initialisation des registres | BSP_IntDisAll() |
| 2. Initialisation du kernel µC-OSII | OSInit() |
| 3. Initialisation des variables internes du programme | AppInitVar() |
| 4. Création des outils de synchronisation de tâches | AppCreateIPCS() |
| 5. Démarrage de toutes les tâches actives | AppTaskStart |
| 6. Démarrage de l'OS | OSStart() |

Le lancement des tâches s'effectue de la manière suivante :

App.c – AppTaskStart(...)

1. Si le flag AX12_REG_PROGRAMMING est activé, le processus de réglage des AX12 est démarré. ATTENTION, si ce processus est actif, toutes les autres tâches sont désactivées.
2. Si le flag APP_TASKODO_ENABLED est activé, le processus d'odométrie est actif.
3. Si le flag APP_TASKASER_ENABLED est activé, le processus d'asservissement est actif.
4. Si le flag APP_TASKMVT_ENABLED est activé, le processus de gestion évolué des trajectoires est actif.
5. Si le flag APP_TASKSENSORS_ENABLED est activé, le processus permettant de récupérer les informations des capteurs et piloter les effecteurs est actif.
6. Si le flag APP_TASKMAIN_ENABLED est activé, le processus d'intelligence artificiel est actif.
7. Si le flag APP_TASKTEMPO_ENABLED est activé, le processus de gestion du temps de match est actif.

NB1 : Toutes les définitions des constantes APP_... sont faites dans le fichier « AppConfig.h ».

NB2 : La définition de la constante _TARGET_440H permet d'activer ou non le mode debug sur la carte de développement.



Documentation:

Informatique



Les variables communes à toutes les tâches sont définies ci-dessous. Elles sont déclarées dans le fichier « App.h » et utilisables via le fichier « AppGlobalVars.h ».

Queues		
OS_EVENT	*AppQueueSensors	File d'attente pour stocker les messages entre les processus TaskMain et TaskSensors
void	*AppQSensorsStk[...]	Pile pour stocker les pointeurs des messages pour la file d'attente AppQueueSensors. La taille maximale de cette pile est définie par la variable APP_QUEUE_SENSORS_SIZE.
StructMsg	AppMsgStack[...]	Pile pour stocker tous les messages de l'application. La taille maximale de l'application est définie par la variable APP_QUEUES_TOTAL_SIZE qui est la somme de toutes les tailles des files de messages.
Variables globales		
StructCmd	App_CmdToTaskMvt	Variable pour stocker la prochaine commande (en provenance de « TaskMain ») devant être exécutée par « TaskMvt ».
unsigned int	App_CmdToTaskMvtId	ID pour donner la validité du message. Il sert de révision de l'information. Si l'ID de l'ordre en cours dans « TaskMvt » est inférieur à App_CmdToTaskMvtId, cela signifie qu'un nouvel ordre est en attente.
StructCmd	App_CmdToTaskAsser	Variable pour stocker la prochaine commande (en provenance de « TaskMvt ») devant être exécutée par « TaskAsser ».
unsigned int	App_CmdToTaskAssertId	ID pour donner la validité du message. Il sert de révision de l'information. Si l'ID de l'ordre en cours dans « TaskAsser » est inférieur à App_CmdToTaskAsserId, cela signifie qu'un nouvel ordre est en attente.
Mutex / Sémaphores		
OS_EVENT*	App_MutexCmdToTaskMvt	Mutex pour limiter l'accès (RW) à la variable App_CmdToTaskMvt.
OS_EVENT*	App_MutexCmdToTaskAsser	Mutex pour limiter l'accès (RW) à la variable App_CmdToTaskAsser.
OS_EVENT*	App_MutexUART1	Mutex pour limiter l'accès (RW) à l'UART1
OS_EVENT*	App_MutexUART2	Mutex pour limiter l'accès (RW) à l'UART2
OS_EVENT*	App_MutexPMP	Mutex pour limiter l'accès (RW) au PMP
Enumeration		
EnumColor	AppCurrentColor	Couleur actuelle (lue depuis le BSP)
Flags		
OS_FLAG_GRP	*AppFlags	Contient tous les FLAGS (sur 32bits) de l'application

Toutes ces variables sont créées et initialisées par les fonctions :

App.c – AppInitVar(. . .)
App.c – AppCreateIPCS(. . .)



Documentation:

Informatique



IV. Présentation Détailée

a) TaskOdo

Description :

Cette tâche permet de calculer à tout moment la position du robot. Elle tourne en continu et calcule de manière cyclique la position actuelle du robot. A temps régulier, elle capture la valeur des encodeurs sur les roues « folles » et effectue ensuite les calculs nécessaires pour obtenir la position en (abscisse, ordonnée, angle).

Fonctionnement :

Ce processus n'interagit avec aucun autre processus. Il met à jour de manière régulière la position actuelle (en interne). Les autres processus viennent lire cette variable (mutexée) pour obtenir la position courante. Pour éviter les risques de blocages concurrents, tous les accès à la variable de positionnement se font explicitement via l'utilisation de la fonction **TaskOdo.c** – `TaskOdo_GetCurrentPos(...)` qui prend en paramètre la structure à remplir avec les informations courantes. Sur le même principe, la fonction **TaskOdo.c** – `TaskOdo_SetCurrentPos(...)` permet de modifier la position actuelle depuis un autre processus.

Algo principal :

```
| Création des Mutex/Semaphores spécifiques à TaskOdo
| Initialisation de la position initiale (tout est mis à zéro)
| Initialisation et démarrage des Timers liés à cette tâche
| Faire(Toujours)
|   | Récupération du Mutex
|   | Si récupération ok
|   |   | Calcul de la nouvelle position (ou calibration en fonction du mode choisi)
|   |   | Si le compteur de debug est à 0
|   |   |   | Afficher les informations de debug sur la sortie UART2
|   |   |   | Incrémenter le compteur de debug
|   |   |   | Le remettre à 0 si il est > à 50
|   |   | FinSi
|   | FinSi
| FinFaire
```

Algo pour calculer la nouvelle position [**TaskOdo.c** – `position_manager_process(...)`]:

```
| Init des variables de la fonction
| Attendre la fin de la capture des informations des codeurs
| Lecture des informations du codeur droit
| Lecture des informations du codeur gauche
| Calcul de l'écart sur la roue droite
| Calcul de l'écart sur la roue gauche
| Conversion des données en mm et radians (pour faire le calcul)
| Calcul de la nouvelle position
| Prendre le Mutex (bloquer l'accès à la variable)
|   | Enregistrement de la nouvelle position dans la structure adéquate
|   | Vérification et correction de l'angle
|   | Mise à jour des incrémentations des codeuses (pour le mouvement pivot)
|   | Mise à jour du flag de mouvement
| Lâcher le Mutex (libérer l'accès à la variable)
| Stocker la variable courante pour le prochain calcul
```

Auteur : [Fifi_one \(philippe.bechet@cpe.fr\)](mailto:Fifi_one (philippe.bechet@cpe.fr))

Team : [OufffTEAM \(oufffteam@gmail.com\)](mailto:OufffTEAM (oufffteam@gmail.com))

Coupe : 2012



Documentation:

Informatique



b) TaskAsser

Description :

Cette tâche permet de calculer les ordres à envoyer aux moteurs afin d'atteindre la position souhaitée.

Fonctionnement :

Cette tâche vient lire la position actuelle du robot via le processus TaskOdo et calcule en conséquence l'ordre à envoyer aux moteurs afin d'atteindre la position désirée. Elle se déroule en continu et se met à jour dès que la position du robot évolue. « TaskAsser » reçoit la position à atteindre à partir de la tâche « TaskMvt » avec l'utilisation de la variable App_CmdToTaskAsser (App_CmdToTaskAsserId permet de gérer la mise à jour de la commande).

Algo principal :

```
| Initialisation des variables de la tâche
| Faire(Toujours)
|   | Temporisation de ASSER_SAMPLING secondes
|   | Lecture de la position actuelle du robot
|   | Si(App_CmdToTaskAsserId est > à l'ID de la commande en cours de traitement)
|   |   | Prendre le Mutex (bloquer l'accès à la variable)
|   |   |   | Récupération du contenu de App_CmdToTaskAsser comme ordre courant
|   |   | Lâcher le Mutex (libérer l'accès à la variable)
|   |   | Mettre à jour l'ID de la commande en cours
|   | Sélectionner(PourLaCommandeCourante)
|   |   | Cas NouvellePosition :
|   |   |     | Mettre à jour les données internes à la tâche avec les nouvelles valeurs
|   |   |     | Mettre à jour les données de TaskOdo avec les nouvelles valeurs
|   |   | FinCas
|   |   | Cas MvtEnAngleSeulement :
|   |   |     | Mettre à jour les données internes à la tâche avec les nouvelles valeurs
|   |   | FinCas
|   |   | Cas MvtEnDistanceSeulement :
|   |   |     | Mettre à jour les données internes à la tâche avec les nouvelles valeurs
|   |   | FinCas
|   |   | Cas MvtSimple et MvtComplexe :
|   |   |     | Mettre à jour les données internes à la tâche avec les nouvelles valeurs
|   |   | FinCas
|   |   | Cas MvtStop :
|   |   |     | Lire les données de positionnement actuel (depuis TaskOdo)
|   |   |     | Mettre à jour les données internes avec les valeurs de TaskOdo
|   |   | FinCas
|   |   | Cas MvtPivot :
|   |   |     | Si le pivot est sur la roue de droite
|   |   |     |     | Calculer le nombre d'incrément à faire sur la roue droite
|   |   |     | Sinon Si le pivot est sur la roue de gauche
|   |   |     |     | Calculer le nombre d'incrément à faire sur la roue gauche
|   |   |     | FinSi
|   |   | FinCas
```



```
    |   FinSelectionner
|---+   FinSi
|     Lire la position courante depuis TaskOdo
|     Selectionner(ModeDeMouvement)
|       Cas AngleSeulement :
|         Calculer l'ordre à envoyer aux moteurs en angle seulement
|       FinCas
|       Cas DistanceSeulement :
|         Calculer l'ordre à envoyer aux moteurs en distance seulement
|       FinCas
|       Cas MouvementMixte :
|         Calculer l'ordre à envoyer aux moteurs en mouvement mixte
|       FinCas
|       Cas MouvementPivot :
|         Calculer l'ordre à envoyer aux moteurs en pivot
|       FinCas
|     FinSelectionner
|     Vérifier la validité des ordres à envoyer aux moteurs
|     Envoyer les ordres aux moteurs
FinFaire
```

c) TaskMvt

Description :

Cette tâche est en charge de toute la gestion du déplacement du robot. C'est elle qui génère et qui valide les trajectoires que doit réaliser le robot afin d'atteindre le point choisi. Elle embarque toute l'intelligence relative aux déplacements du robot. C'est cette tâche qui prend en compte les informations de positionnement et qui génère les nouvelles trajectoires lors d'une collision du système d'anticollision.

Fonctionnement :

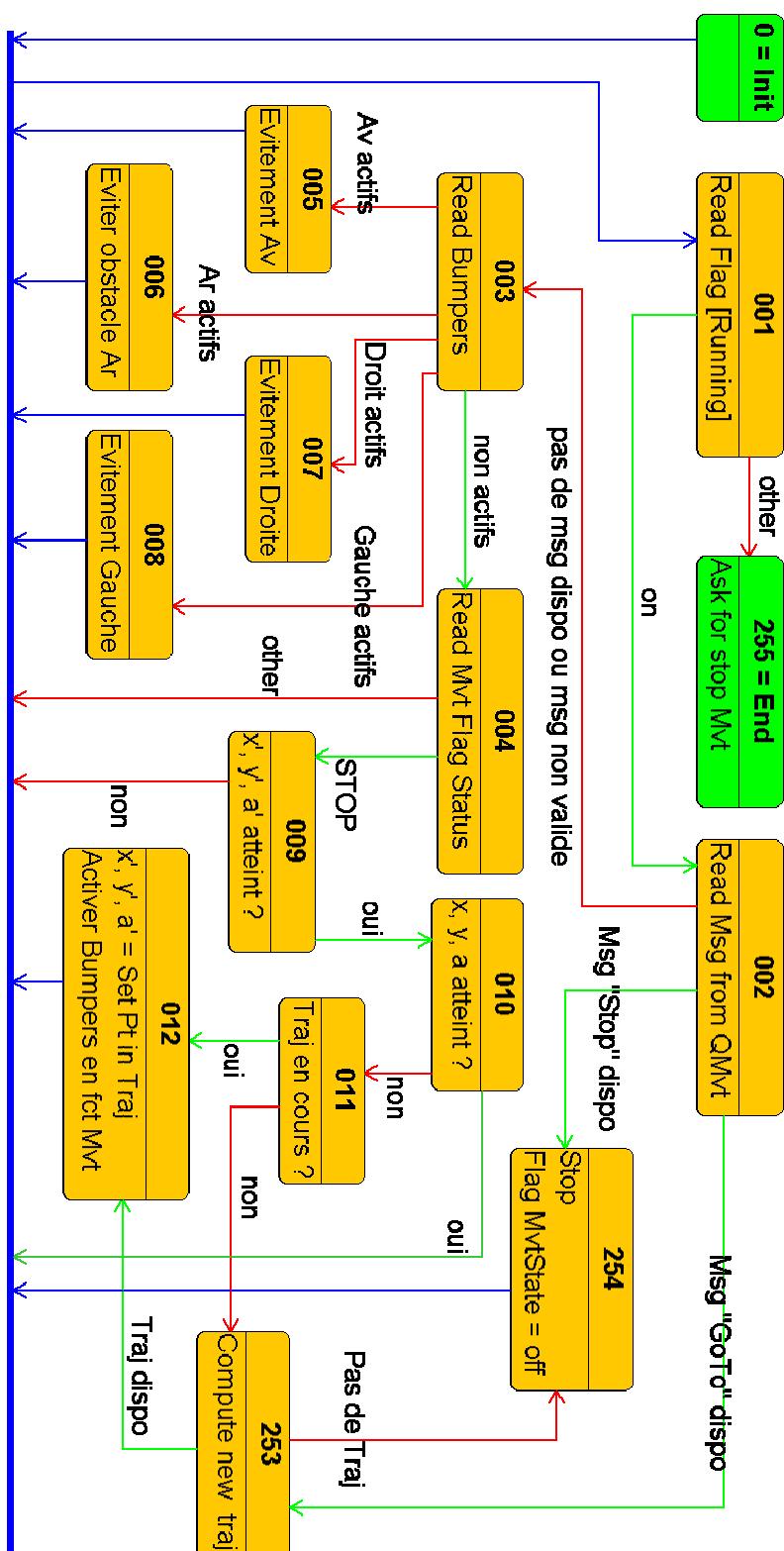
Cette tâche reçoit un point à atteindre depuis TaskMain. Elle crée une trajectoire adaptée et envoie les ordres de déplacement à TaskAsser. Toutes les méthodes liées aux calculs de trajectoire sont présentes dans le module LibMoving (détailé dans la suite de ce document)

Documentation:

Informatique



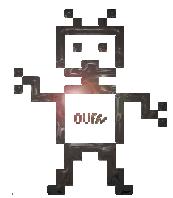
Algo principal (schéma) :





Documentation:

Informatique



Algo principal (détailé) :

```
| Initialisation des variables de la tâche
| Faire
|   | Attente de 10 ms pour relâcher le processeur
|   | Récupération de l'EtatCourant (à partir de la variable NextState)
|   | Lecture des FLAGS système
|   | Selectionner(EtatCourant)
|   |   | Cas 0 (=EtatInitial) :
|   |   |     Si(BoutonStartEstPressé)
|   |   |       | NextState = 1
|   |   |     Sinon
|   |   |       | NextState = 0
|   |   |       | Si(App_CmdToTaskMvt définit une nouvelle position)
|   |   |         | Mettre à jour la position du robot
|   |   |       | FinSi
|   |   |     FinSi
|   |   | FinCas
|
|   | Cas 1 (=LectureEtatTimer) :
|   |   Si(LeTempsEstTerminé)
|   |     | NextState = 255
|   |   Sinon
|   |     | NextState = 2
|   |   FinSi
|   | FinCas
|
|   | Cas 2 (=LireMessageProvenantDuMain) :
|   |   Si(App_CmdToTaskMvtId est > à l'ID de la dernière commande exécutée)
|   |     | Récupération du message
|   |     | Mise à jour de l'ID de la dernière commande exécutée
|   |     | Selectionner(LaNouvelleCommandeReçue)
|   |     |   | Cas Stop :
|   |     |     | NextState = 254
|   |     |   FinCas
|   |     |   Cas Mvt_AngleSeulement :
|   |     |   Cas Mvt_Simple :
|   |     |   Cas Mvt_DistanceSimple :
|   |     |   Cas Mvt_ModeMixte :
|   |     |   Cas Mvt_Pivot :
|   |     |   Cas Mvt_DefinirNouvellePosition :
|   |     |     | NextState = 253
|   |     |   FinCas
|   |     |   CasParDéfaut :
|   |     |     | NextState = 3
|   |     |   FinCas
|   |     | FinSelectionner
|   |   Sinon
|   |     | NextState = 3
|   |   FinSi
|
| FinCas 3 (=Lecture des bumpers) :
|   | Lecture des états des bumpers
|   | Si(Bumpers de face activés)
|   |     | NextState = 5
```

Auteur : Fifi_one (philippe.bechet@cpe.fr)

Team : OufffTEAM (oufffteam@gmail.com)

Coupe : 2012

Documentation:

Informatique



```

    Sinon Si(Bumpers de gauche activés)
    |   nextState = 7
    Sinon Si(Bumpers de droite activés)
    |   nextState = 8
    Sinon Si(Bumpers de dos activés)
    |   nextState = 6
    Sinon
    |   nextState = 4
    FinSi
FinCas

Cas 4 (=Lecture du FLAG de mouvement) :
Si(l'état du mouvement est STOP)
|   nextState = 9
Sinon
|   nextState = 1
FinSi
FinCas

Cas 5 (=Séquence d'évitement FRONT) :
Envoyer le message d'arrêt à TaskAser

Faire
|   Tempo de 500ms
|   Lecture du FLAG des bumpers de FRONT
TantQue(Bumpers FRONT activés)
    nextState = 253
FinCas

Cas 6 (=Séquence d'évitement BACK) :
Envoyer le message d'arrêt à TaskAser

Faire
|   Tempo de 500ms
|   Lecture du FLAG des bumpers de BACK
TantQue(Bumpers BACK activés)
    nextState = 253
FinCas

Cas 7 (=Séquence d'évitement RIGHT) :
    nextState = 254
FinCas

Cas 8 (=Séquence d'évitement LEFT) :
    nextState = 254
FinCas

Cas 9 (=Vérification de la position du robot) :
Si(Le point actuel est-il atteint ?)
|   nextState = 10
Sinon
|   Si(Un point est en cours d'atteinte)
|       |   Incrémenter le compteur de TimeOut
|   Sinon
|       |   Annuler le TimeOut

```



```

FinSi
| Si(Le compteur de TimeOut est > au TIMEOUT)
|   | Si(Le mouvement actuel est bloquant)
|   |   | Libérer le FLAG du statut d'action
|   | FinSi
|   | Désactiver la trajectoire actuelle
|   | nextState = 1
|   | FinSi
|   | nextState = 1
| FinSi
| FinCas

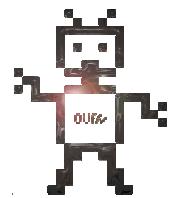
Cas 10 (=Est-ce que le point final est atteint) :
Lecture de la position actuelle
Si(L'un des paramètres est = à USE_CURRENT_VALUE)
|   | Mettre à jour la commande avec la position actuelle
| FinSi
| Si(Le point final est atteint)
|   | Si(Le mouvement actuel est bloquant)
|   |   | Libérer le FLAG du statut d'action
|   | FinSi
|   | Désactiver la trajectoire actuelle
|   | Réinitialiser tous les FLAGS
|   | nextState = 1
| Sinon
|   | nextState = 11
| FinSi
| FinCas

Cas 11 (=Existe-t-il un point suivant dans la trajectoire ?) :
Si(Il existe un point suivant)
|   | nextState = 12
| Sinon
|   | nextState = 253
| FinSi
| FinCas

Cas 12 (=Utilisation du point suivant dans la trajectoire) :
Sélection du point suivant
Si(Il existe un point suivant)
|   | Envoyer le nouveau point à atteindre à TaskAsser
| FinSi
| nextState = 1
| FinCas

Cas 253 (=Calculer une nouvelle trajectoire) :
Effacer la trajectoire courante
Calculer la nouvelle trajectoire
Si(Une trajectoire a été trouvée)
|   | nextState = 12
| Sinon
|   | nextState = 254
| FinSi
| FinCas

```



Documentation: Informatique

```
Cas 254 (=Ordre de stopper le mouvement) :  
    Envoyer le message de STOP à TaskAsser  
    Si(Le mouvement actuel est bloquant)  
        Libérer le FLAG du statut d'action  
    FinSi  
    Désactiver la trajectoire actuelle  
    Inhiber tous les capteurs de mouvement du robot  
    NextState = 1  
FinCas

Cas 255 (=Fin du match) :  
    Désactiver la trajectoire actuelle  
    Envoyer le message de STOP à TaskAsser  
FinCas

Cas par défaut :  
    Désactiver la trajectoire actuelle  
    NextState = 254  
FinCas
FinSelectionner
TantQue(L'état courant est différent de 255)

Faire(Toujours)
    Attente de 1 heure
FinFaire
```

d) TaskSensors

Description :

Cette tâche est en charge de toute la gestion des capteurs et des actionneurs du robot. C'est elle qui fait l'interface entre le monde extérieur et le robot. C'est cette tâche qui se charge de lever les flags d'anticollision.

Fonctionnement :

Cette tâche reçoit un ordre depuis TaskMain (via QSensors) et l'exécute. Une fois l'action terminée, et si l'action en cours est bloquante, TaskSensor lève le flag ACTION_STATUS afin de signaler aux autres tâches que l'action est terminée.

Algo principal :

```
Si(le flag d'utilisation de la tirette est à VRAI)  
    TantQue(La tirette est non présente)  
        Lire la couleur  
    FinTantQue  
    TantQue(La tirette est présente)  
        Lire la couleur  
    FinTantQue  
FinSi  
Lire la couleur  
Lever le Flag Tirette pour lancer toutes les tâches  
Initialiser le Flag d'action  
Initialiser la pince du robot  
Faire
```

Auteur : Fifi_one (philippe.bechet@cpe.fr)

Team : OufffTEAM (oufffteam@gmail.com)

Coupe : 2012



Documentation:

Informatique



```
Lire l'état des capteurs (collision + pince)
EtatCourant = EtatSuivant
Selectionner(EtatCourant)
  Cas 0 : Attente d'un message provenant de TaskMain
    Si(Un message est en cours d'utilisation)
      Libérer le message
    FinSi
    Récupérer un nouveau message
    Si(Un message n'a pas été trouvé)
      EtatSuivant = 0
    Sinon
      Selectionner(La commande depuis le message courant)
        Cas Sensors_GrabObject :
          Attraper l'objet
          Si(l'action courante est bloquante)
            Lever le Flag pour indiquer la fin de l'action
          FinSi
        FinCas
        Cas Sensors_SetHolderStatus :
          Ouvrir/Fermer la pince en fonction du paramètre 1
          Si(l'action courante est bloquante)
            Lever le Flag pour indiquer la fin de l'action
          FinSi
        FinCas
        Cas Sensors_SetHolderLevel :
          Lever la pince en fonction du paramètre 1
          Si(l'action courante est bloquante)
            Lever le Flag pour indiquer la fin de l'action
          FinSi
        FinCas
        Cas par défaut :
          EtatSuivant = 0
        FinCas
      FinSelectionner
    FinSinon
  FinCas

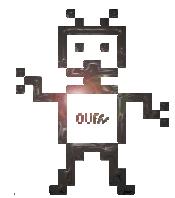
  Cas 255 : Etat final
  EtatSuivant = 255
  FinCas

  Cas par défaut
  EtatSuivant = 0
  FinCas
FinSelectionner
TantQue(EtatCourant != 255)
Faire(Toujours)
  Patienter 30 sec
FinFaire
```



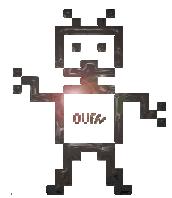
Documentation:

Informatique



Sous-fonctions de la tâche :

- TaskSensors_IsStartButtonPressed()
 - *Action* : Lecture de l'état de la tirette
 - *Retour* : **BOOLEAN**
 - OS_TRUE : La tirette est en place
 - OS_FALSE : La tirette n'est plus en place
 - *Paramètres* : Aucun
- TaskSensors_ReadColor()
 - *Action* : Lecture de la couleur
 - *Retour* : Via la variable **AppCurrentColor**
 - c_ColorA : Bleu
 - c_ColorB : Rouge
 - *Paramètres* : Aucun
- TaskSensors_CheckBumpers()
 - *Action* : Vérifier l'état des bumpers
 - *Retour* : Via le flag **AppFlags**
 - APP_PARAM_APPFLAG_GP2_FRONT
 - APP_PARAM_APPFLAG_GP2_BACK
 - *Paramètres* : Aucun
- TaskSensors_GrabObject()
 - *Action* : Fermer la pince (en cas de prise ratée, la pince se rouvre)
 - *Retour* : Aucun
 - *Paramètres* : Aucun
- TaskSensors_ControlHolder(...)
 - *Action* : Piloter le mode de fermeture de la pince
 - *Retour* : Aucun
 - *Paramètres* :
 - HOLDER_CLOSE : Fermer la pince
 - HOLDER_OPEN_LEFT_ONLY : Fermer la pince de gauche
 - HOLDER_OPEN_RIGHT_ONLY : Fermer la pince de droite
 - HOLDER_OPEN : Ouvrir la pince
 - HOLDER_GRAB : La pince essaye de bloquer l'objet
 - HOLDER_GRIP : La pince bloque l'objet pour le lever
- TaskSensors_SetHolderLevel(...)
 - *Action* : Permet de définir le niveau de la pince
 - *Retour* : Aucun
 - *Paramètres* :
 - HOLDER_LEVEL_MIDDLE : Mettre la pince dans un état intermédiaire
 - HOLDER_LEVEL_HIGH : Mettre la pince dans un état haut
 - HOLDER_LEVEL_LOW : Mettre la pince dans un état bas



- TaskSensors_CheckObject()
 - *Action* : Vérifier la présence d'un objet dans la pince
 - *Retour* : Via le flag `AppFlags`
 - APP_PARAM_APPFLAG_GP2 HOLDER
 - OS_TRUE : Un objet est présent dans la pince
 - OS_FALSE : Aucun objet n'est présent dans la pince
 - *Paramètres* :
 - HOLDER_LEVEL_MIDDLE : Mettre la pince dans un état intermédiaire
 - HOLDER_LEVEL_HIGH : Mettre la pince dans un état haut
 - HOLDER_LEVEL_LOW : Mettre la pince dans un état bas

e) TaskTempo

Description :

Cette tâche est en charge de la gestion du temps. C'est elle qui indique à toutes les autres tâches le statut actuel.

Fonctionnement :

Dès que la tirette est enlevée, le compte à rebours commence. Une fois le temps de match écoulé, TaskTempo lève le Flag `TIMER_STATUS` pour indiquer à toutes les tâches de se mettre en pause.

Algo principal :

```

| Attente de la tirette
| Calcul du temps de match
| Faire
|   | Se mettre en pause durant APP_PARAM_TEMPO_RESOLUTION millisecondes
|   | Lire le TempsActuel
| TantQue(TempsActuel < temps de match)

| Lever le Flag TIMER_STATUS
| Faire(Toujours)
|   | Attendre 1 heure
| FinFaire
  
```

f) TaskMain

Description :

Cette tâche est la tâche principale. C'est elle qui est en charge de toute l'intelligence du robot. C'est cette tâche qui se charge de coordonner les ordres envoyés aux tâches TaskSensors et TaskMvt.

Fonctionnement :

TaskMain coordonne l'envoi des ordres entre toutes les tâches. Elle est en charge de surveiller l'évolution des Flags afin de générer les ordres en conséquence. C'est le point central du programme.

Auteur : [Fifi_one \(philippe.bechet@cpe.fr\)](mailto:Fifi_one (philippe.bechet@cpe.fr))

Team : [OufffTEAM \(oufffteam@gmail.com\)](mailto:OufffTEAM (oufffteam@gmail.com))

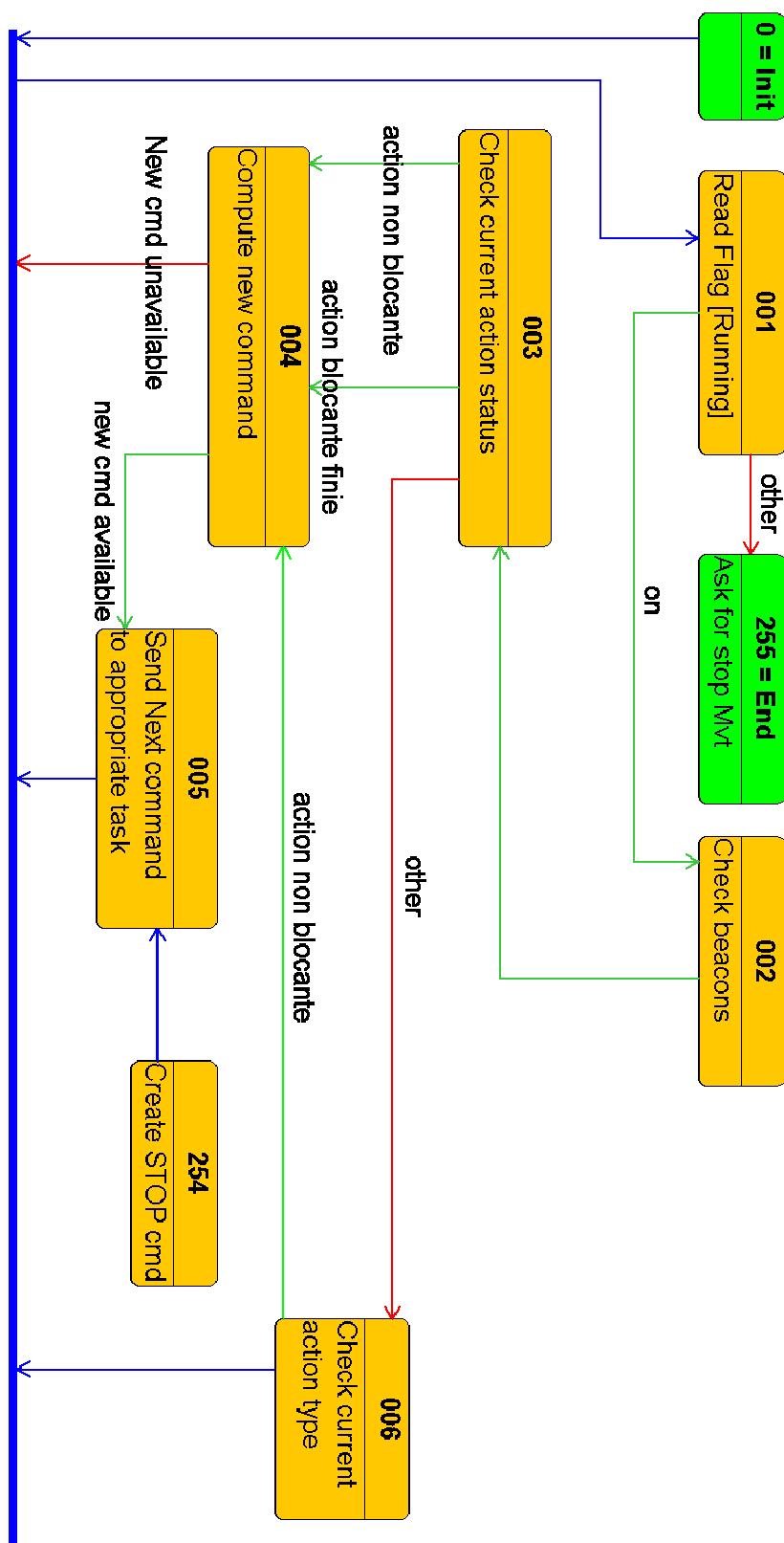
Coupe : [2012](#)

Documentation:

Informatique



Algo principal (schéma) :



Documentation:

Informatique



Algo principal :

```

    | Attente de 1 sec (attente des autres tâches)
    | TantQue(La couleur n'est pas valide)
    |   | Temporisation
    |   | FinTantQue

    Faire
    |   Si(CouleurCourante est différente de CouleurPrécédente)
    |   | CouleurPrécédente = CouleurActuelle
    |   | Si(Une position est disponible pour CouleurActuelle)
    |   |   | Envoyer la position à TaskMvt
    |   |   | FinSi
    |   | FinSi
    |   | Lire les Flags actuels
    | TantQue(Le Flag de démarrage n'est pas présent dans les Flag système)

    Faire
    |   EtatCourant = EtatSuivant
    |   Lire les Flags systèmes
    |   Selectionner(EtatCourant)
    |   | Cas 0 : Initialisation
    |   |   | EtatSuivant = 1
    |   |   | FinCas

    |   | Cas 1 : Vérification du Timer
    |   |   Si(Le Flag de TIMER est activé)
    |   |   |   | EtatSuivant = 255
    |   |   Sinon
    |   |   |   | EtatSuivant = 2
    |   |   | FinSi
    |   |   | FinCas

    |   | Cas 2 : Vérification des balises
    |   |   EtatSuivant = 3
    |   |   | FinCas

    |   | Cas 3 : Vérification du statut courant
    |   |   Si(L'action en cours est non bloquante)
    |   |   |   Si(Le Flag de fin d'action est levé)
    |   |   |   |   | EtatSuivant = 4
    |   |   |   Sinon
    |   |   |   |   | EtatSuivant = 6
    |   |   |   | FinSi
    |   |   |   Sinon
    |   |   |   |   | EtatSuivant = 4
    |   |   |   | FinSi

    |   | Cas 4 : Récupérer l'action suivante
    |   |   Si(Une nouvelle action est dispo)
    |   |   |   | EtatSuivant = 5
    |   |   Sinon
    |   |   |   | EtatSuivant = 1
    |   |   | FinSi

    |   | Cas 5 : Envoyer l'action suivante

```

Auteur : Fifi_one (philippe.bechet@cepe.fr)

Team : OufffTEAM (oufffteam@gmail.com)

Coupe : 2012

Documentation:

Informatique



```

Si(L'action actuelle est définie)
    Réinitialiser les variables d'échanges
    Selectionner(L'action actuelle)
        Cas MvtAngleSeulement :
        Cas MvtDistanceSeulement :
        Cas MvtModeMixte :
        Cas MvtPivot :
        Cas MvtSimple :
        Cas MvtStop :
        Cas MvtAttente :
        Cas MvtDefinirNouvellePosition :
            Envoyer la nouvelle commande à TaskMvt
        FinCas

        Cas AttraperUnObjet :
            Préparer le message
            Envoyer me message à TaskSensors
        FinCas

        Cas DefinirLeModeDePriseDeLaPince :
        Cas DefinirLeNiveauDeLaPince :
            Préparer le message
            Envoyer me message à TaskSensors
        FinCas

        Cas par défaut :
        FinCas
    FinSelectionner
    Abaisser le Flag d'action
FinSi
EtatSuivant = 1
FinCas

Cas 6 : Vérifier le Flag d'action
Selectionner(Le type de l'action actuelle)
    Cas ActionBloquante :
        EtatSuivant = 1
    FinCas

    Cas ActionNonBloquante :
        EtatSuivant = 4
    FinCas

    Cas par défaut :
        EtatSuivant = 1
    FinCas
FinSelectionner

Cas 254 : Action à faire si les balises sont activées
Préparer l'action à exécuter
EtatSuivant = 5
FinCas

Cas 255 : Etat Final
EtatSuivant = 255

```



Documentation:

Informatique



```
FinCas  
| Cas par défaut  
|   EtatSuivant = 1  
| FinCas  
| FinSelectionner  
TantQue(EtatCourant = 255)  
  
Faire(Toujours)  
| Attendre 1 heure  
FinFaire
```

g) LibMoving

Description :

Ce module contient toutes les méthodes de base qui permettent le calcul des trajectoires pour le robot.

Fonctionnement :

Les méthodes sont utilisées dans la tâche TaskMvt pour le calcul des trajectoires.

Méthodes :

```
void LibMoving_MoveInMM(int Dist, INT8U Speed, StructCmd *NextSetpoint);  
- Param1 (Integer) : Distance en mm à parcourir par le robot en ligne droite  
- Param2 (INT8U) : Vitesse pour le mouvement (entre 0 et 100%)  
- Param3 (StructCmd*) : Structure contenant le point à atteindre  
  o Le type de mouvement est un mouvement simple  
  o L'action est bloquante  
- Retour : La valeur retournée dans NextSetpoint n'est pas contrôlée (par rapport à la taille du terrain) avant d'être retournée.  
  
void LibMoving_RotateInDeg(int AngleInDeg, INT8U Speed, StructCmd *NextSetpoint);  
- Param1 (Integer) : Angle en degré à parcourir  
- Param2 (INT8U) : Vitesse pour le mouvement (entre 0 et 100%)  
- Param3 (StructCmd*) : Structure contenant le point à atteindre  
  o Le type de mouvement est un mouvement simple  
  o L'action est bloquante  
- Retour : La valeur retournée dans NextSetpoint n'est pas contrôlée (par rapport à la taille du terrain) avant d'être retournée.  
  
void LibMoving_MoveToAngleInDeg(int AngleToGoInDeg, INT8U Speed, StructCmd  
*NextSetpoint);  
- Param1 (Integer) : Angle en degré à atteindre dans le repère du terrain  
- Param2 (INT8U) : Vitesse pour le mouvement (entre 0 et 100%)  
- Param3 (StructCmd*) : Structure contenant le point à atteindre  
  o Le type de mouvement est un mouvement simple  
  o L'action est bloquante  
- Retour : La valeur retournée dans NextSetpoint n'est pas contrôlée (par rapport à la taille du terrain) avant d'être retournée.
```



Documentation:

Informatique



```
void LibMoving_ComputeNewPath(StructCmd *ExpectedCmd, StructCmd *NewPath, INT8S *NewPathLength);
    - Param1 (StructCmd*) : Point à atteindre dans le repère du terrain
    - Param2 (StructCmd*) : Tableau pour recevoir tous les points de la trajectoire
    - Param3 (INT8S*) : Entier pour recevoir le nombre d'ordres de la trajectoire
    - Retour : Les valeurs retournées sont faites via les variables NewPath et NewPathLength.
```

```
void LibMoving_CreateEscapeSeq(INT8U EscapeSeqType, INT8U Speed, StructCmd *NewPath, INT8S *NewPathLength);
    - Param1 (INT8U) : Type de mouvement d'esquive à réaliser
        o APP_MOVING_ESCAPE_SEQ_RIGHT : Passer par la droite
        o APP_MOVING_ESCAPE_SEQ_LEFT : Passer par la gauche
        o APP_MOVING_ESCAPE_SEQ_BACK : Reculer
    - Param2 (INT8U) : Vitesse pour le mouvement (entre 0 et 100%)
    - Param3 (StructCmd*) : Tableau pour recevoir tous les points de la trajectoire
    - Param4 (INT8S*) : Entier pour recevoir le nombre d'ordres de la trajectoire
    - Retour : Les valeurs retournées sont faites via les variables NewPath et NewPathLength.
```

```
BOOLEAN LibMoving_IsSetpointReached(StructCmd *SetpointToTest);
    - Param1 (StructCmd*) : Point que le robot doit atteindre
    - Retour : Retourne OS_TRUE si le point est atteint (si la position actuelle du robot est comprise dans un cercle d'approche défini par les coefficients APP_MOVING_APPROACH_PRECISION_COEF et APP_MOVING_DIST_APPROACH_PRECISION), OS_FALSE sinon. Le type d'approche est calculé en fonction du type de mouvement.
```

h) AppGlobalFunc

Description :

Ce module contient toutes les fonctions génériques de bases.

Méthodes :

```
void AppDebugMsg(char *DebugMsg);
    - Param1 (char*) : Chaine de caractère à afficher sur la sortie debug (serial port)
    - Retour : Pas de retour.
```

```
BOOLEAN AppPostQueueMsg(OS_EVENT *PtrQueue, StructMsg *PtrMsgToPost);
    - Param1 (OS_EVENT*) : Pointeur vers la queue où il faut poster le message
    - Param2 (StructMsg*) : Message à poster. Le message est copié puis la copie est ajoutée dans la queue
    - Retour : OS_TRUE si le message est posté correctement, OS_FALSE sinon.
```

```
float AppConvertRadInDeg(float ValueInRad);
    - Param1 (float) : Angle (en radian) à convertir en degré
    - Retour : Retourne la valeur en degré
```



Documentation:

Informatique



```
float AppConvertDegInRad(float ValueInDeg);
- Param1 (float) : Angle (en degré) à convertir en radian
- Retour : Retourne la valeur en radian

float AppCheckAngleInRad(float ValueToCheck);
- Param1 (float) : Angle (en radian) à vérifier
- Retour : Retourne un angle en radian compris dans l'intervalle ]-π ; π]

float AppCheckAngleInDeg(float ValueToCheck);
- Param1 (float) : Angle (en degré) à vérifier
- Retour : Retourne un angle en degré compris dans l'intervalle ]-180 ; 180]

INT8U AppGetCurrentPos(StructPos *CurrentPos);
- Param1 (StructPos*) : Pointeur vers une structure pour recevoir la position actuelle
du robot
- Retour : La valeur actuelle du robot (en provenance de TaskOdo) est stockée dans
CurrentPos
```

i) Strategy

Description :

Le répertoire Strategy contient les fichiers de stratégies du robot.

Fonctionnement :

Tous les fichiers implémentent les fonctions de bases (déclaré dans le fichier Strategy.h) :

- Strategy_GetInitCmd(...)
- Strategy_GetNextAction(...)

Chaque fichier est lié à une constante d'utilisation qui permet de sélectionner la stratégie à utiliser. Seulement une seule constante d'utilisation peut être activée à la fois (une erreur de compilation se produit sinon). Les constantes actuelles sont :

- DEFAULT_STRATEGY_ENABLED : Stratégie par défaut (DefaultStrategy.c)
- HOMOL_STRATEGY_ENABLED : Stratégie d'homologation (Homol.c)
- STRATEGY_1_ENABLED : Stratégie n°1 (Strategy.c)

Méthodes :

INT8U Strategy_GetInitCmd(EnumColor CurrentColor, StructCmd *InitCmd)

Récupérer la commande initiale en fonction de la couleur sélectionnée.

- Param1 (EnumColor) : Couleur actuelle
- Param2 (StructCmd*) : Pointeur pour récupérer la commande initiale
- Retour : Retourne la commande initiale dans InitCmd

INT8U Strategy_GetNextAction(EnumColor CurrentColor, StructCmd *NextAction)

Récupérer la commande suivante en fonction de la couleur sélectionnée.

- Param1 (EnumColor) : Couleur actuelle
- Param2 (StructCmd*) : Pointeur pour récupérer la commande suivante
- Retour : Retourne la commande suivante dans NextAction



Documentation: Informatique



La fonction `Strategy_GetNextAction(...)` est structurée de la manière suivante :

- Une variable statique (`CurrentActionID`) permet de suivre toutes les étapes de la stratégie.
- Tous les cas de la stratégie doivent contenir les paramètres suivants :
 - o [Optionnel] `OSTimeDlyHMSM(...)` pour temporiser l'exécution de la tâche
 - o [Obligatoire] Définition de la commande (`NextAction->Cmd = Command`)
 - o [Optionnel] Définition des paramètres de la commande
 - o [Obligatoire] Définition du type de la commande (Bloquante ou non bloquante)
 - o [Obligatoire] Incrémenter le compteur `CurrentActionID` pour passer à l'étape suivante