HAPTION
1 bis, rue Pierre et Marie Curie
92140 Clamart

API VIRTUOSE V2.25

DOCUMENTATION

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 Qu'est-ce que l'API VIRTUOSE ?

L'API VIRTUOSE est une bibliothèque de fonctions C permettant de s'interfacer avec un système à retour d'effort de la gamme VIRTUOSE.

L'API VIRTUOSE supporte les produits suivants :

- VIRTUOSE 6D35-45
- VIRTUOSE 6D35-40
- VIRTUOSE 3D35-40
- VIRTUOSE 6D10-20
- VIRTUOSE 3D10-20

Elle permet de piloter simultanément plusieurs produits de la gamme, ainsi que le simulateur VIRTUOSE.

Elle est disponible pour différentes architectures et systèmes d'exploitation :

- Architecture PC sous Microsoft Windows 98/NT/2000
- Architecture PC sous Linux (noyau 2.4)
- Architecture SGI sous Irix 6

Selon les cas, elle se présente sous forme d'une librairie statique et/ou dynamique (dll sous Windows).

1.2 Composition du document

Ce document regroupe les chapitres suivants :

1. Introduction (ce chapitre)
2. Manuel d'Installation
3. Manuel Utilisateur
4. Manuel de Référence
5. Glossaire

2 MANUEL D'INSTALLATION

Comme il l'a été précédemment expliqué, l'API Virtuose est une librairie C. Ainsi son installation consiste simplement à inclure les fichiers qui la constituent au sein de l'environnement de développement. Ces fichiers (header et binaires) diffèrent en fonction du système d'exploitation utilisé.

Ainsi, les fichiers à inclure à l'environnement de développement sont :

- Sous Microsoft Windows 98/NT/2000 :
  - virtuoseAPI.h
  - virtuoseAPI.lib
  - virtuoseAPI.dll
- Sous Linux 2.4.x et IRIX 6 (à partir de leur répertoire respectif sur le CD d'installation):
  - virtuoseAPI.h
  - virtuoseAPI.so
  - libvirtuose.a

3 MANUEL UTILISATEUR

3.1 Contenu de l'API VIRTUOSE

L'API VIRTUOSE se compose des éléments suivants :

- Un fichier d'en-tête C et C++ à inclure dans votre projet : VirtuoseAPI.h
- Une librairie statique au format Windows, Linux (ELF), ou Irix : VirtuoseAPI.lib
- Une librairie dynamique au format Windows (DLL)

3.2 Conventions

3.2.1 Programmation

Les fonctions de l'API VIRTUOSE respectent les conventions suivantes :

- Les noms de fonctions sont en langue anglaise, ils commencent toujours par « *virt* » et ne comportent aucun caractère « souligné » (*underscore*), la séparation entre mots étant matérialisée par des majuscules (ex : *virtGetPosition*).
- À l'exception de la fonction « *virtOpen* », toutes les fonctions prennent comme premier argument un objet de type « *VirtContext* » ; cet objet est créé par la fonction « *virtOpen* » et détruit par « *virtClose* ».
- À l'exception des fonctions « *virtOpen* », « *virtGetErrorCode* » et « *virtGetErrorMessage* », toutes les fonctions de l'API renvoie 0 en cas de succès et -1 en cas d'échec.
- Les fonctions de l'API ont toutes un prototype unique en C ; lorsqu'elles sont utilisées dans une application C++, leur prototype est spécifié comme « *extern "C"* » de façon à éviter toute surcharge.

3.2.2 Modes d'interaction

L'API VIRTUOSE comporte plusieurs modes d'interaction avec le système à retour d'effort. Selon le type d'application envisagé, le mode choisi sera différent :

1. Couplage force/position : dans ce mode très basique, l'application transmet au système les valeurs d'effort à appliquer au repère courant, et reçoit en retour la position et la vitesse du repère courant
2. Couplage position/force : ce mode évolué permet le couplage direct avec un objet virtuel ; dans ce cas, l'application transmet au système la position et la vitesse du centre de l'objet, et reçoit en retour le torseur d'efforts à appliquer au centre de l'objet lors de l'intégration de la dynamique. Les gains optimaux d'asservissement sont calculés par le contrôleur embarqué, à partir de la masse et du torseur d'inertie de l'objet couplé.
3. Couplage position/force avec guidage virtuel : ce mode ajoute au mode précédent un guidage virtuel de différent type (translation, rotation, ...).

3.2.3 Définition des repères

Les conventions d'orientation sont matérialisées par la notion de « repère ». Toute grandeur géométrique (position, vitesse, effort, inertie, etc.) est définie par rapport à un repère particulier.

Par définition, tous les repères utilisés dans l'API VIRTUOSE sont directs. Cela constitue une différence fondamentale par rapport aux conventions traditionnellement utilisées en informatique graphique. Nous proposons plus loin un schéma de conversion permettant de faire cohabiter les deux types de conventions.

Par défaut et à l'initialisation de l'API, tous les repères sont confondus et définis comme suit :

- Origine au centre du système à retour d'effort, c'est-à-dire au point d'intersection entre l'axe 1 (rotation de la base) et l'axe 2 (premier mouvement vertical).
- Axe X horizontal, aligné selon la position médiane de l'axe 1 (rotation de la base) et orienté vers l'utilisateur.
- Axe Y horizontal, perpendiculaire à l'axe X et orienté vers la droite de l'utilisateur.
- Axe Z vertical et orienté vers le haut

L'API définit les repères suivants :

1. Le repère d'environnement, qui correspond à l'origine de la scène ; il est fixé par l'application indépendamment de l'API.
2. Le repère d'observation, qui représente en général le repère écran ; il est positionné par rapport au repère d'environnement.
3. Le repère de base, qui représente le centre du système à retour d'effort ; il est positionné par rapport au repère d'observation.
4. Le repère outil, qui correspond à la base de l'outil fixé à l'extrémité du Virtuose ; il est positionné par rapport au repère d'environnement.
5. Le repère avatar, qui correspond à la position de la main de l'utilisateur par rapport au système haptique, et qui permet de prendre en compte la géométrie de l'outil fixé à son extrémité ; il est positionné par rapport au repère outil.

La position des repères suivants peut être modifiée à l'initialisation par l'API :

- Repère d'observation (par rapport au repère d'environnement)
- Repère de base (par rapport au repère d'observation)
- Repère avatar (par rapport au repère outil)

La position du repère suivant peut être modifiée à tout moment par l'API :

- Repère d'observation (par rapport au repère d'environnement)

La position du repère suivant ne peut jamais être modifiée :

- Repère outil (par rapport au repère d'environnement)

3.2.4 Unités physiques utilisées

Toutes les valeurs utilisées dans l'API sont exprimées en unités physiques et en convention métrique, à savoir :

- Durées en secondes (s)
- Dimensions en mètres (m)
- Angles en radians (rad)
- Vitesses linéaires en mètres par seconde ($m.s^{-1}$)
- Vitesses angulaires en radians par seconde ($rad.s^{-1}$)
- Efforts en Newtons (N)
- Couples en Newtons-mètres (N.m)
- Masses en kilogrammes (kg)
- Inerties en $kg.m^2$

3.2.5 Expression des grandeurs géométriques

L'API utilise différentes grandeurs géométriques, qui sont exprimées de la façon suivante :

- Les positions sont exprimées sous la forme de vecteur déplacement, qui regroupe sept composantes, à savoir un terme de translation (x, y, z) suivi d'un terme de rotation sous forme de quaternion normalisé (qx, qy, qz, qw) (cf. glossaire).
- Les vitesses sont exprimées sous forme de torseur cinématique (vx, vy, vz, wx, wy, wz) (cf. glossaire)
- Les efforts sont exprimés sous forme de torseur (fx, fy, fz, cx, cy, cz) (cf. glossaire)

3.3 Mise en œuvre de l'API VIRTUOSE

3.3.1 Initialisation

La première étape dans la mise en œuvre de l'API VIRTUOSE consiste à établir la communication avec le périphérique. La fonction `virtOpen` réalise cette opération, et elle crée en même temps toutes les variables internes nécessaires au fonctionnement de l'API, en renvoyant un pointeur de type `VirtContext`. Ce pointeur doit ensuite être utilisé pour chaque appel de fonction de l'API : il sert d'une part à accéder à l'état courant de la communication, et d'autre part à identifier le VIRTUOSE auquel est adressée la requête dans le cas d'une application à plusieurs interfaces haptiques.

Il est indispensable de tester la valeur de retour de la fonction `virtOpen`, car le fonctionnement de toute l'API est subordonné au succès de la connexion :

```
VirtContext VC;
VC = virtOpen("192.168.1.1");
if (VC == NULL)
{
      fprintf(stderr, "Erreur dans virtOpen: %s\n",
            virtGetErrorMessage(virtGetErrorCode(NULL));
      return -1;
}
```

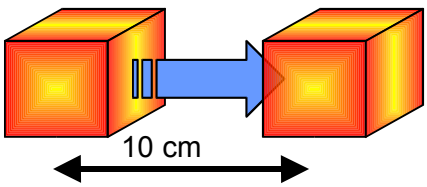
Ensuite, il est conseillé de réaliser explicitement la configuration du couplage, dans la mesure où la configuration par défaut n'est pas forcément conforme à l'utilisation qu'on veut faire du bras :

1. Le facteur d'échelle en déplacement (fonction `virtSetSpeedFactor`)

L'intérêt d'un facteur d'homothétie en vitesse est l'augmentation de l'amplitude des translations effectuées durant les simulations. Cette fonctionnalité est très appréciable dans le cadre des simulations nécessitant la manipulation d'objets dans un scène de grandes dimensions.

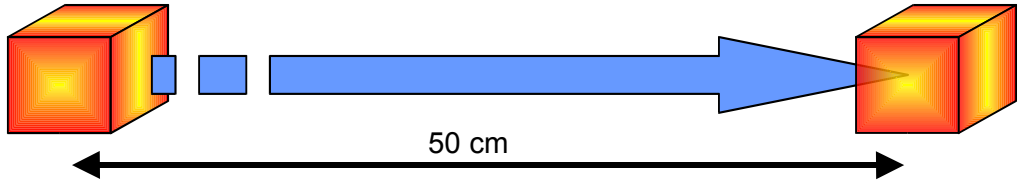
Le schéma suivant illustre l'effet d'un facteur d'homothétie en vitesse sur les déplacements en translation de l'objet virtuel manipulé :

Cas 1 : Pas d'homothétie en vitesse



Pour un déplacement horizontal de 10 cm de l'interface haptique, le déplacement du cube virtuel est équivalente.

Cas 2 : Homothétie de 5 en vitesse



Pour un déplacement horizontal de 10 cm de l'interface haptique, le déplacement du cube virtuel est 5 fois plus grand au sein du monde virtuel.

2. Le facteur d'échelle en efforts (fonction `virtSetForceFactor`)

L'intérêt d'un facteur d'homothétie en effort est la possibilité de jouer sur l'intensité des efforts à fournir par l'opérateur durant les manipulations grâce à une diminution des efforts restitués par l'interface haptique. Cette fonctionnalité est très appréciable dans le cadre des simulations nécessitant la manipulation d'objets de grande masse.

3. Le mode d'indexation (fonction `virtSetIndexingMode`)

Lorsque l'opérateur arrive proche d'une butée, il a la possibilité de remettre dans une position confortable en agissant sur le bouton de décalage. Il existe trois types de décalage:

- le type `INDEXING_ALL`, autorisant un décalage sur les translations et sur les rotations,
- le type `INDEXING_TRANS`, autorisant un décalage uniquement sur les translations,
- le type `INDEXING_NONE`, n'autorisant aucun décalage; l'action sur le bouton de décalage n'a aucun effet.

4. Le pas d'intégration du simulateur (fonction `virtSetTimeStep`)
5. La position du repère de base par rapport au repère d'observation (fonction `virtSetBaseFrame`)
6. La position du repère d'observation par rapport au repère d'environnement (fonction `virtSetObservationFrame`)
7. La vitesse du repère d'observation par rapport au repère d'environnement (fonction `virtSetObservationFrameSpeed`)
8. Le type de couplage (fonction `virtSetCommandType`)

Le code suivant correspond à une configuration d'usage fréquent :

```
float identity[7] = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,1.0f};
virtSetIndexingMode(VC, INDEXING_ALL);
virtSetForceFactor(VC, 1.0f);
virtSetSpeedFactor(VC, 1.0f);
virtSetTimeStep(VC, 0.003f);
virtSetBaseFrame(VC, identity);
virtSetObservationFrame(VC, identity)
virtSetObservationFrameSpeed(VC, identity)
```

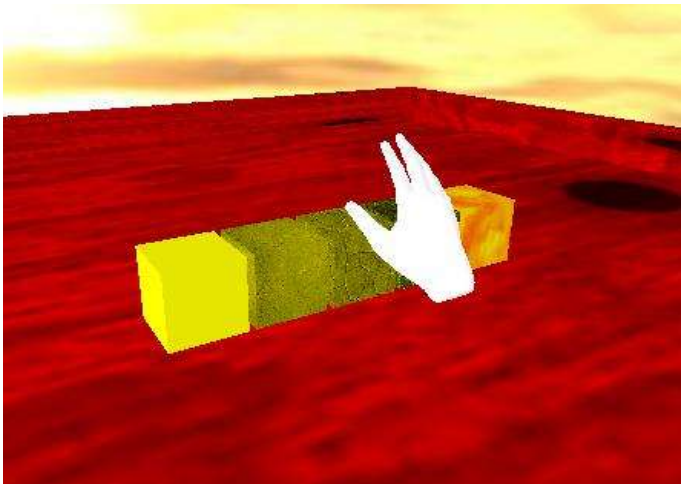
```
virtSetCommandType(VC, COMMAND_TYPE_ADMITTANCE);
```

Enfin, l'initialisation doit se terminer par l'autorisation de mise sous puissance (fonction virtSetPowerOn). Cette fonction active le bouton-poussoir « MARCHE » du VIRTUOSE, qui permet à l'utilisateur de mettre les moteurs sous tension :

```
virtSetPowerOn(VC, 1);
```

3.3.2 Utilisation comme pointeur

Dans de nombreuses applications, il est nécessaire de disposer d'un mode « pointeur », dans lequel la position du VIRTUOSE est représentée à l'écran par un objet purement graphique (l'avatar), sur lequel ne s'exerce aucun effort. Ce mode est utile pour désigner un objet sur lequel on veut se coupler, ou encore pour appeler une fonction à l'aide d'un menu contextuel.



Afin d'implémenter ce mode de fonctionnement, on définit une fonction appelée périodiquement, qui va servir à mettre à jour les consignes du périphérique haptique et à récupérer sa position. Cette fonction est appelée par l'application, par exemple sous la forme d'un callback.

En mode simple `COMMAND_TYPE_IMPEDANCE` cette fonction doit forcer à 0 la consigne d'effort, et renvoyer la position du bras. Elle peut prendre la forme suivante :

```
int update_avatar(VirtContext VC, float *position)
{
      float null [6] = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f};
      virtSetForce(VC, null);
      virtGetPosition(VC, position);
      return 0;
}
```

En mode couplé `COMMAND_TYPE_ADMITTANCE`, la fonction lit la position et la vitesse du bras, la renvoie à celui-ci comme consigne (ainsi, l'erreur de position et de vitesse est nulle, et donc aucun effort n'est généré) :

```
int update_avatar(VirtContext VC, float *position)
{
      float speed[6];
      virtGetPosition(VC, position);
      virtGetSpeed(VC, speed);
      virtSetPosition(VC, position);
      virtSetSpeed(VC, speed);
      return 0;
}
```

Dans les deux cas, la fonction renvoie dans la variable `position` la position courante de l'avatar, en translation et en rotation. Cette position est affectée par le choix des repères de base et d'observation, ainsi que par le bouton de décalage (ou bouton d'indexation) : lorsque ce dernier est enfoncé, l'avatar ne suit plus les mouvements du bras.

3.3.3 Couplage à un objet

En mode couplé `COMMAND_TYPE_ADMITTANCE`, on peut attacher le VIRTUOSE directement à un objet, et le bras devient alors une contrainte dynamique à intégrer dans la simulation de la scène.

Le couplage à un objet est effectué selon la procédure suivante :

1. L'appel de la fonction `virtAttachVO` réalise le couplage, en calculant les paramètres optimaux et en garantissant la stabilité ; pour cela, elle a besoin de connaître le pas d'intégration du simulateur, ainsi que la masse et le tenseur d'inertie de l'objet
2. La position et la vitesse du point de référence de l'objet sont envoyés comme consignes (fonction `virtSetPosition` et `virtSetSpeed`)

Le code suivant réalise le couplage avec un objet de masse `m`, d'inertie `mxmymz`, et dont le centre d'inertie est défini par la position `P` et la vitesse `V` :

```
virtAttachVO(VC, m, mxmymz);
virtSetPosition(VC, P);
virtSetSpeed(VC, S);
```

Afin de faire évoluer l'objet, on définit une fonction de mise à jour appelée périodiquement par l'application, par exemple sous forme de callback :

```
int update_object(VC,float *P,float *S,float *torseur_effort)
{
      virtSetPosition(VC, P);
      virtSetSpeed(VC, S);
      virtGetForce(VC, torseur_effort);
      return 0;
}
```

3.3.4 Temporisation

Certaines temporisations logicielles sont nécessaire entre le positionnement d'un paramètre et sa lecture, le temps au contrôleur de prendre en compte ce nouveau paramètrage. Cela concerne :

- le type d'indexation,
- les rèpères de base, avatar, observation et base du mécanisme virtuel,
- les coefficients de déplacement et d'effort,
- les enchaînements de mécanismes virtuels.

Exemples:

```
// type d'indexation
VirtIndexingType      type_indexation;
virtSetIndexingMode(VC, INDEXING_TRANS);
Sleep(10);
virtGetIndexingMode(VC, &type);

// repère de base du mécanisme virtuel
float      virtVmBaseFrame[7];
virtVmSetBaseFrameToCurrentFrame(VC);
Sleep(10);
virtGetVmBaseFrame(VC, virtVmBaseFrame);

// enchaînement de deux mécanismes virtuels :
// déplacement vers un point situé à 20 cm sur l'axe x et enchaînement d'un
// mécanisme rotation suivant le même axe
virtVmSetType(VC, VM_TYPE_CartMotion);
virtVmSetBaseFrameToCurrentFrame(VC);
virtVmActivate(VC);
virtVmWaitUpperBound(VC);
virtVmDeactivate(VC);
Sleep(10);

virtVmSetType(VC, VM_TYPE_Rx);
virtVmActivate(VC);
```

3.3.5 Evolution logicielle de la version 2.25

1. Il existe maintenant un guide virtuel plan défini par les axes Ox et Oy du repère de base du mécanisme virtuel. L'opérateur et l'objet virtuel sont contraints à rester sur ce plan après activation du mécanisme.

2. Deuxième guide virtuel, le mécanisme manivelle. Le bras de levier est déterminé au moment de l'activation par la position du bras par rapport à l'axe Ox du repère de base du mécanisme virtuel.

3. En cours de simulation, il est possible de modifier dynamiquement le coefficient de déplacement (ou facteur d'échelle) et de travailler à différentes échelles dans une scène virtuelle. On peut imaginer d'effectuer en même temps un zoom de la caméra.

4. Le repère d'observation est lui aussi modifiable dynamiquement. Il se peut, après couplage, qu'il existe un petit effet d'inertie si le repère d'observation subit un fort changement d'orientation.

5. Un objet virtuel peut maintenant être saisi non plus au centre de gravité mais à une position précisée par la simulation: par exemple, la saisie d'un marteau par son manche. On aurra donc un effet de levier du marteau sur les rotarions.

6. Auparavant, l'opération ne commandait un objet virtuel qu'en position. Actuellement, l'opérateur peut commander l'objet en vitesse. L'opérateur doit définir le rayon d'une sphère dont le centre est le centre de l'espace de travail du périphérique. Lorsque la poignée est à l'intérieure, la commande reste en position. Quand elle est à l'extérieure, l'opération contrôle la vitesse de l'objet virtuel (norme et direction).

7. Il existe un autre type d'indexation, INDEXING_ALL_FORCE_FEEDBACK_INHIBITION, permettant d'inhiber le retour d'effort lorsque vous effectuez un décalage (appui sur le bouton de décalage). Si l'objet couplé est en collision avec une autre pièce de la scène à ce moment, le retour d'effort disparaît en peu à peu. Il réapparait aussi peu à peu lorsque l'opération de décalage est terminé.

8. Le contrôle de la connexion est activé une fois la boucle haptique lancé. Ainsi, la boucle haptique peut ne pas suivre immédiatement la phase d'initialisation du virtuose.

9. Il est possible de travailler sans aucun décalage; ce qu'on appelle le mode colocalisé. En conséquence, le bouton de décalage est inopérant, un déplacement du bras sans puissance déplace l'objet virtuel auquel on est attaché. D'autre part, le retour de la puissance impose à l'opérateur d'effectuer un segment (le bras est contraint sur un segment) pour faire revenir le bras à la position au moment de l'arrêt de la puissance.

3.4 Résolution de problèmes

Nous citons ci-dessous quelques problèmes classiques qui peuvent survenir lors des premiers couplages avec un objet virtuel. Pour chacun d'entre eux, nous donnons des règles simples à adopter :

3.4.1 Compatibilité avec les versions 1.x

L'API VIRTUOSE est modifié sur les points suivants :

1. Les types de commande COMMAND_TYPE_FORCE et COMMAND_TYPE_POSITION sont remplacés par COMMAND_TYPE_IMPEDANCE et COMMAND_TYPE_ADMITTANCE.
2. La lecture de l'état des boutons se réalise par l'intermédiaire d'une seule fonction.
3. Certaines paramètres, notamment la position du vecteur de base et d'observation, ne peuvent être positionné qu'avant la sélection du type de commande.

3.4.2 Fonctions obsolètes

Les fonctions `virtSetLimitTorque` et `virtGetLimitTorque` sont obsolètes. La saturation des efforts est maintenant implémentée dans la fonction `virtSaturateTorque`; elle a été nettement améliorée.

3.4.3 Instabilité

L'API VIRTUOSE calcule les paramètres de couplage en respectant deux critères :

1. Stabilité du système couplé (simulateur – objet virtuel – bras)
2. Optimalité (raideur maximale)

Si une instabilité survient néanmoins lors d'un couplage, il est bon de faire certaines vérifications :

- Le pas d'intégration donnée à l'API par la fonction `virtSetTimeStep` est-il vraiment identique à celui qui est utilisé par le simulateur ?
- Le simulateur ne prend-il pas du retard par rapport à ce pas d'intégration, c'est-à-dire le temps réel est-il vraiment synchrone avec le temps simulé ?
- Les paramètres de masse et d'inertie utilisés par le simulateur pour son intégration sont-ils les mêmes que ceux transmis au `Virtuose` ? Les unités sont-elles respectées ?

Par ailleurs, certains moteurs de détection de collision peuvent renvoyer des informations très bruitées, avec des normales de contact mal définies, qui créent des réactions à haute fréquence, que l'on peut interpréter comme une instabilité.

3.4.4 Collage intempestif

Ce phénomène est caractéristique d'une perte de performances de la liaison avec le Virtuose. C'est particulièrement le cas lorsque le bras est branché sur une branche réseau

partagée avec de nombreuses stations de travail, ou encore lorsque le bras et le simulateur ne se trouvent pas sur la même branche réseau.

Dans ce cas, il est nécessaire de modifier la topologie du réseau, la meilleure solution étant de consacrer une carte réseau à la seule communication avec le VIRTUOSE.

3.4.5 Raideur trop faible

Ce phénomène apparaît lorsqu'on manipule des objets de grande taille, ou bien lorsque le facteur d'échelle en déplacement est inadapté. Il se traduit par une difficulté à maîtriser le déplacement de l'objet couplé, qui a trop d'inertie par rapport aux efforts exercés par le VIRTUOSE.

Dans ce cas, il est bon de diminuer l'inertie et la masse de l'objet couplé, soit en jouant sur sa taille (facteur d'échelle global pour toute la scène), soit en diminuant sa densité. On peut également, dans une certaine mesure, réduire le facteur d'échelle en effort, ce qui a pour effet de décupler les efforts exercés par le VIRTUOSE (voir ci-dessous).

3.4.6 Retour d'effort trop faible

Ce phénomène est observé lorsque le facteur d'échelle en effort est trop faible. Il peut aussi se produire avec certaines combinaisons de masse, d'inertie, et de facteur d'échelle en déplacement.

Dans tous les cas, il est souhaitable de remonter la valeur du facteur d'échelle en effort, ou à défaut de se rapprocher d'un cas nominal (objet de 10 cm et 300 g, échelle 1).

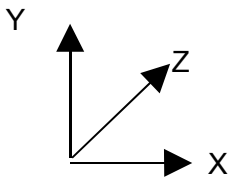
4 CHANGEMENT DE CONVENTIONS

Un problème classique est le passage entre une convention graphique (repères indirects, axe Z vers le fond de l'écran) et une convention robotique (repères directs, axe Z vertical).

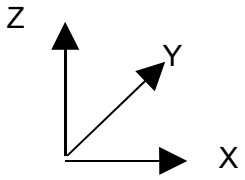
Nous proposons ci-dessous un schéma simple de conversion entre ces deux conventions, illustré par l'exemple du moteur physique Vortex™ distribué par la société Critical Mass Labs.

4.1 Exemple de conversion

Nous définissons les fonctions suivantes, qui ont pour effet de permuter les axes Y et Z, de façon à passer d'un repère graphique à un repère robotique et inversement :



Convention Vortex



Convention Virtuose

- Les déplacements Vortex sont définies sous la forme de position + quaternion.
- Les vitesses Vortex sont définies sous la forme de vitesse linéaire + vitesse angulaire.

```
void transformVortexPosQuatToVirtPosEnv(float *vortexPos,
                                        float *vortexQuat,
                                        float *virtPosEnv)
{
      virtPosEnv[0] = vortexPos[0];
      virtPosEnv[1] = vortexPos[2];
      virtPosEnv[2] = vortexPos[1];
      virtPosEnv[3] = -vortexQuat[1];
      virtPosEnv[4] = -vortexQuat[3];
      virtPosEnv[5] = -vortexQuat[2];
      virtPosEnv[6] = vortexQuat[0];
}

void transformVirtPosEnvToVortexPosQuat(float *virtPosEnv,
                                        float *vortexPos,
                                        float *vortexQuat)
{
      vortexPos[0] = virtPosEnv[0];
      vortexPos[1] = virtPosEnv[2];
      vortexPos[2] = virtPosEnv[1];
      vortexQuat[0] = virtPosEnv[6];
      vortexQuat[1] = -virtPosEnv[3];
      vortexQuat[2] = -virtPosEnv[5];
      vortexQuat[3] = -virtPosEnv[4];
}
```

```
void transformVortexLinAngVelToVirtSpeedEnv(float *vortexLinVel,
                                            float *vortexAngVel,
                                            float *virtSpeedEnv)
{
       virtSpeedEnv[0] = vortexLinVel[0];
       virtSpeedEnv[1] = vortexLinVel[2];
       virtSpeedEnv[2] = vortexLinVel[1];
       virtSpeedEnv[3] = -vortexAngVel[0];

       virtSpeedEnv[4] = -vortexAngVel[2];
       virtSpeedEnv[5] = -vortexAngVel[1];
}

void transformVirtTorsToVortexForceTorque(float *virtTors,
                                          float *vortexForce,
                                          float *vortexTorque)
{
       vortexForce[0] = virtTors[0];
       vortexForce[1] = virtTors[2];
       vortexForce[2] = virtTors[1];
       vortexTorque[0] = -virtTors[3];
       vortexTorque[1] = -virtTors[5];
       vortexTorque[2] = -virtTors[4];
}

void transformVortexInertiaTensorToVirtInertiaTensor(float *vortexTensor,
                                                     float *virtTensor)
{
       virtTensor[0] = vortexTensor[0];
       virtTensor[1] = vortexTensor[2];
       virtTensor[2] = vortexTensor[1];
       virtTensor[3] = vortexTensor[6];
       virtTensor[4] = vortexTensor[8];
       virtTensor[5] = vortexTensor[7];
       virtTensor[6] = vortexTensor[3];
       virtTensor[7] = vortexTensor[5];
       virtTensor[8] = vortexTensor[4];
}
```

5 MANUEL DE RÉFÉRENCE

virtAttachVO
virtClose
virtDetachVO
virtDisplayHardwareStatus
virtGetBaseFrame
virtGetButton
virtGetCommandType
virtGetDeadMan
virtGetEmergencyStop
virtGetError
virtGetErrorCode
virtGetErrorMessage
virtGetForce
virtGetForceFactor
virtGetIndexingMode
virtGetObservationFrame
virtGetPosition
virtGetSpeed
virtGetSpeedFactor
virtGetTimeLastUpdate
virtGetTimeoutValue
virtOpen
virtSetBaseFrame
virtSetCommandType
virtSetForce
virtSetForceFactor
virtSetIndexingMode
virtSetObservationFrame
virtSetObservationFrameSpeed
virtSetPosition
virtSetSpeed
virtSetSpeedFactor
virtSetTimeoutValue

virtSetTimeStep
virtGetTimeStep
virtVmSetType
virtVmActivate
virtVmDeactivate
virtVmSetSamplingTimeStep
virtTrajRecordStart
virtTrajRecordStop
virtVmStartTrajSampling
virtVmGetTrajSamples
virtPhysicalPosition
virtAvatarPosition
virtVmSetDefaultToTranparentMode
virtVmSetDefaultToCartesianPosition
virtVmGetBaseFrame
virtConvertRGBToGrayscale
virtSetTextureForce
virtSetTexture
virtVmWaitUpperBound
virtDisableControlConnexion
virtSetPeriodicFunction
virtStartLoop
virtStopLoop
virtIsInBounds
virtGetAlarm
virtActiveSpeedControl
virtDeactiveSpeedControl
virtSetCatchFrame
virtGetCatchFrame

NOM

virtAttachVO – couple un objet avec le VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtAttachVO(VirtContext VC, float mass, float *inertia)
```

DESCRIPTION

La fonction `virtAttachVO` réalise le couplage de l'objet désigné avec le VIRTUOSE, et calcule des gains d'asservissement optimaux et garantissant la stabilité globale du système.
L'objet est défini par la position de son centre (point de réduction du torseur d'efforts), qui doit impérativement être envoyée à l'aide de la fonction `virtSetPosition` avant l'appel de `virtAttachVO`.
Cette fonction n'est accessible que dans les modes de commande `COMMAND_TYPE_ADMITTANCE et COMMAND_TYPE_IMPEDANCE`.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `mass` correspond à la masse de l'objet, exprimée en kg.
Le paramètre `inertia` correspond à la matrice d'inertie de l'objet, stockée en lignes sous forme d'un vecteur à 9 composantes.

VALEUR DE RETOUR

En cas de succès, la fonction `virtAttachVO` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtDetachVO, virtSetPosition, virtSetSpeed, virtGetForce

NOM

virtClose – fermeture de la connexion au contrôleur du Virtuose

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtClose(VirtContext VC);
```

DESCRIPTION

La fonction `virtClose` ferme la connexion au contrôleur du Virtuose.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtClose` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtOpen, virtGetErrorCode

NOM

virtDetachVO – découple un objet de la liaison avec le VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtDetachVO(VirtContext VC)
```

DESCRIPTION

La fonction `virtDetachVO` annule le couplage en cours d'un objet avec le VIRTUOSE, et remet à zéro les gains d'asservissement.
Cette fonction n'est accessible que dans les modes de commande `COMMAND_TYPE_ADMITTANCE` et `COMMAND_TYPE_IMPEDANCE`.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtDetachVO` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtAttachVO

NOM

`virtDisplayHardwareStatus` – Permet d'effectuer un bilan du fonctionnement du Virtuose niveau hardware.

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtDisplayHardwareStatus (VirtContext VC, FILE *file)
```

DESCRIPTION

La fonction `virtDisplayHardwareStatus` permet d'effectuer un bilan du fonctionnement du Virtuose niveau hardware en termes de pannes et d'erreurs au niveau de chacun des composants matériels. Les composants considérés sont les variateurs, les capteurs de position et le capteur d'effort (fonctionnement et saturation pour ce dernier).

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `file` n'est pas utilisé. Le flux de sortie est celui défini par la fonction `virtSetOutputFile` ou stdout par défaut.

VALEUR DE RETOUR

En cas de succès, la fonction `virtDisplayHardwareStatus` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

VirtSetOutputFile

NOM

virtGetBaseFrame – position courante du repère de base

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetBaseFrame(VirtContext VC, float *pos)
```

DESCRIPTION

La fonction `virtGetBaseFrame` renvoie la position courante du repère de base par rapport au repère d'observation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos` correspond au déplacement du repère de base par rapport au repère d'observation.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetBaseFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetBaseFrame

NOM

virtGetButton – état d'un bouton

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetButton(VirtContext VC, int button, int *state)
```

DESCRIPTION

La fonction `virtGetButton` renvoie l'état d'un des boutons placés sur l'outil monté à l'extrémité du Virtuose.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `button` correspond au numéro de bouton.
Le paramètre `state` est renseigné par la fonction (0 : bouton relâché, 1 : bouton enfoncé).

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetButton` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

VirtGetDeadMan, virtGetEmergencyStop

NOM

virtGetCommandType – état du mode de couplage

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetCommandType(VirtContext VC, VirtCommandType *type)
```

DESCRIPTION

La fonction `virtGetCommandType` renvoie le mode de couplage actif.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `type` est rempli par la fonction. Les valeurs possibles sont :

`COMMAND_TYPE_IMPEDANCE`
Couplage force/position

`COMMAND_TYPE_ADMITTANCE`
Couplage position/force simple

`COMMAND_TYPE_VIRTMECH`
Couplage position/force avec mécanisme virtuel

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetCommandType` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetCommandType

NOM

virtGetDeadMan – état du capteur de sécurité

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetDeadMan(VirtContext VC, int *deadman)
```

DESCRIPTION

La fonction `virtGetDeadMan` renvoie l'état du capteur de sécurité placé à l'extrémité du bras (aussi appelé « capteur d'homme mort »).

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `deadman` est rempli par la fonction. La valeur 1 signifie que le capteur de sécurité est enclenché (utilisateur présent), la valeur 0 que le capteur est déclenché (utilisateur absent).

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetDeadMan` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtGetEmergencyStop, virtGetPowerOn

NOM

virtGetEmergencyStop – état de la chaîne d'arrêt d'urgence

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetEmergencyStop(VirtContext VC, int *stop)
```

DESCRIPTION

La fonction `virtGetEmergencyStop` renvoie l'état de la chaîne d'arrêt d'urgence.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `stop` est rempli par la fonction. La valeur 1 signifie que la chaîne est fermée (le système est opérationnel), la valeur 0 qu'elle est ouverte (le système est arrêté).

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetEmergencyStop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtGetDeadMan, virtGetPowerOn

NOM

virtGetError – accès à l'état de fonctionnement du virtuose

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetError (VirtContext VC, int* error)
```

DESCRIPTION

La fonction `virtGetError` permet de tester l'état de fonctionnement du virtuose, que ce soit en termes de pannes ou de simples erreurs au niveau de chacun des composants matériels. Les composants testés sont les variateurs, les capteurs de position et le capteur d'effort.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `error` est rempli par la fonction avec la valeur d'état de fonctionnement du Virtuose. La valeur 0 signifie que le virtuose est en bon état de fonctionnement. La valeur 1 signifie une panne ou une erreur de fonctionnement.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetError` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_HARDWARE_ERROR`
Ce message d'erreur signifie la présence d'une ou plusieurs pannes ou erreurs au niveau de l'interface haptique.

VOIR AUSSI

`virtDisplayHardwareStatus`

NOM

virtGetErrorCode – accès au code de la dernière erreur signalée

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetErrorCode(VirtContext VC);
```

DESCRIPTION

La fonction `virtGetErrorCode` renvoie le code de la dernière erreur signalée sur la liaison avec le contrôleur Virtuose correspondant au paramètre `VC`.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`, ou à `NULL` dans le cas d'un échec de la fonction `virtOpen`.

VALEUR DE RETOUR

Par définition, la fonction virtGetErrorCode renvoie toujours un code d'erreur (voir ci-dessous).

ERREURS

Les erreurs ci-dessous ont une signification générique. Dans certains cas, le diagnostic dépend de la fonction qui a signalé l'erreur. Se reporter alors à la description de la fonction en cause.

`VIRT_E_NO_ERROR`
Aucune erreur signalée.

`VIRT_E_OUT_OF_MEMORY`
Erreur d'allocation mémoire.

`VIRT_E_INVALID_CONTEXT`
Un parameter de type VirtContext erroné a été transmis à l'API.

`VIRT_E_COMMUNICATION_FAILURE`
Une erreur a été signalée sur la communication avec le contrôleur du Virtuose.

`VIRT_E_FILE_NOT_FOUND`
Le nom donné en paramètre ne correspond à aucun fichier.

`VIRT_E_WRONG_FORMAT`
Une erreur a été détectée dans un format de données.

`VIRT_E_TIME_OUT`
Le délai maximum d'attente de réponse a été dépassé.

`VIRT_E_NOT_IMPLEMENTED`
La function n'est pas implémentée dans cette version de l'API.

`VIRT_E_VARIABLE_NOT_AVAILABLE`
La variable demandée n'est pas disponible sur ce modèle de Virtuose.

`VIRT_E_INCORRECT_VALUE`
Une valeur transmise à l'API a une valeur incorrecte ou en-dehors des limites autorisées.

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_HARDWARE_ERROR`
Problème de fonctionnement d'un des composants matériels du virtuose.

VOIR AUSSI

virtOpen, virtGetErrorMsg

NOM

virtGetErrorMessage – message explicatif d'un code d'erreur

SYNOPTIQUE

```
#include "virtuoseAPI.h"
const char *virtGetErrorMessage(int code)
```

DESCRIPTION

La fonction `virtGetErrorMessage` renvoie un message explicatif du code d'erreur donné en paramètre.

PARAMÈTRES

Le paramètre `code` contient le code d'erreur dont on veut obtenir une explication.

VALEUR DE RETOUR

Dans tous les cas, la fonction `virtGetErrorMessage` renvoie un message d'erreur.

ERREURS

Aucune.

VOIR AUSSI

virtGetErrorCode

NOM

virtSetTimeStep – fixe la valeur du pas d'échantillonnage de la simulation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTimeStep(VirtContext VC, float timeStep)
```

DESCRIPTION

La fonction `virtSetTimeStep` permet de communiquer au contrôleur embarqué la valeur du pas d'échantillonnage de la simulation. Cette valeur est utilisée par celui-ci afin de garantir la stabilité du système couplé.
La fonction doit être appelée avant la sélection du type de commande.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `timeStep` correspond au pas d'échantillonnage de la simulation, exprimé en secondes.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetTimeStep` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `timeStep` est incorrecte. Cette erreur est générée dans les cas suivants :

- valeur négative ou nulle
- valeur inférieure au pas d'échantillonnage utilisé par le contrôleur embarqué
- valeur trop grande pour assurer la stabilité du système

`VIRT_E_CALL_TIME`
La fonction est appelée après la sélection du type de commande.

VOIR AUSSI

virtGetTimeStep

NOM

virtGetTimeStep – lecture du pas d'échantillonnage de la simulation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetTimeStep(VirtContext VC, float* step)
```

DESCRIPTION

La fonction `virtGetTimeStep` permet de lire la valeur du pas d'échantillonnage de la simulation préalablement fixée par la fonction `virtGetTimeStep`. Elle retourne zéro si celle-ci n'est pas définie.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `step` correspond au pas d'échantillonnage de la simulation, exprimé en secondes.

VALEUR DE RETOUR

La fonction `virtGetTimeStep` renvoie 0 dans tous les cas.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetTimeStep

NOM

virtGetForce – renvoie le torseur d'efforts à appliquer à l'objet couplé

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetForce(VirtContext VC, float *force_p_env_r_ps)
```

DESCRIPTION

La fonction `virtGetForce` renvoie le torseur d'efforts à appliquer à l'objet auquel est couplé le VIRTUOSE, permettant la simulation dynamique de la scène.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `force_p_env_r_ps` est renseigné par la fonction `virtGetForce`. Il correspond aux efforts à appliquer à l'objet, sous forme d'un torseur 6 composantes projeté dans le repère d'environnement et réduit au centre de l'objet.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetForce` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetPosition, virtSetSpeed, virtAttachVO

NOM

virtGetForceFactor – renvoie le facteur d'homothétie en effort

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetForceFactor(VirtContext VC, float *factor)
```

DESCRIPTION

La fonction `virtGetForceFactor` permet de lire le facteur d'homothétie en effort, qui correspond à un changement d'échelle entre les efforts exercés au niveau du VIRTUOSE et ceux de la simulation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `factor` est rempli par la fonction `virtGetForceFactor`. Un facteur plus petit que l'unité correspond à une amplification des efforts depuis le VIRTUOSE vers la simulation.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetForceFactor` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetForceFactor, virtGetSpeedFactor

NOM

virtGetIndexingMode – accès au mode courant d'indexation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virt(VirtContext VC, VirtIndexingMode *mode)
```

DESCRIPTION

La fonction `virtGetIndexingMode` permet d'accéder au mode d'indexation (aussi appelé mode de décalage).

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `mode` est rempli par la fonction. Les valeurs possibles sont :

`INDEXING_ALL`
L'indexation agit sur les translations et les rotations.

`INDEXING_TRANS`
L'indexation agit sur les translations seulement.

`INDEXING_NONE`
L'indexation est inopérante.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetIndexingMode` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetIndexingMode

NOM

virtGetObservationFrame – position courante du repère d'observation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetObservationFrame(VirtContext VC, float *pos)
```

DESCRIPTION

La fonction `virtGetObservationFrame` renvoie la position courante du repère d'observation par rapport au repère d'environnement.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos` correspond au déplacement du repère d'observation par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetObservationFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetObservationFrame

NOM

virtGetPosition – renvoie la position courante de l'avatar ou de l'objet couplé au VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetPosition(VirtContext VC, float *pos_p_env)
```

DESCRIPTION

La fonction `virtGetPosition` renvoie la position courante de l'avatar, ou de l'objet couplé au VIRTUOSE si un couplage est en cours.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos_p_env` est renseigné par la fonction `virtGetPosition`. Elle est exprimée sous forme d'un vecteur déplacement à 6 composantes, projeté dans le repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetPosition` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetPosition, virtAttachVO

NOM

virtGetPowerOn – état de l'alimentation des moteurs

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetPowerOn(VirtContext VC, int *poweron)
```

DESCRIPTION

La fonction `virtGetPowerOn` renvoie l'état de l'alimentation des moteurs.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `poweron` est rempli par la fonction. La valeur 1 signifie que les moteurs sont alimentés, la valeur 0 qu'ils ne le sont pas.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetPowerOn` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetPowerOn, virtGetEmergencyStop, virtGetDeadMan

NOM

virtGetSpeed – renvoie la vitesse de l'avatar, ou de l'objet couplé au VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetSpeed(VirtContext VC, float *speed_p_env_r_ps)
```

DESCRIPTION

La fonction `virtGetSpeed` renvoie la vitesse de l'avatar, ou bien de l'objet couplé au VIRTUOSE si un couplage est actif.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `speed_p_env_r_ps` est renseigné par la fonction `virtGetSpeed`. Elles est exprimée sous la forme d'un torseur à 6 composantes, projeté dans le repère d'environnement et réduit au centre de l'avatar ou de l'objet couplé au VIRTUOSE, selon le cas.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetSpeed` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetSpeed, virtAttachVO

NOM

virtGetSpeedFactor – renvoie le facteur d'homothétie en déplacement

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetSpeedFactor(VirtContext VC, float *factor)
```

DESCRIPTION

La fonction `virtGetSpeedFactor` permet de lire la valeur du facteur d'homothétie en déplacement, qui correspond à un changement d'échelle entre les déplacements du VIRTUOSE et ceux de la simulation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `factor` est renseigné par la fonction `virtGetSpeedFactor`. Un facteur plus grand que l'unité correspond à une dilatation de l'espace de travail du Virtuose.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetSpeedFactor` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetSpeedFactor, virtGetForceFactor

NOM

virtGetTimeLastUpdate – accès à la valeur du délai d'attente

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetTimeLastUpdate (VirtContext, unsigned int *tout);
```

DESCRIPTION

La fonction virtGetTimeLastUpdate renvoie le temps depuis la dernière acquisition.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `tout` est rempli par la fonction avec la valeur en ticks du temps écoulé depuis la dernière acquisition

VALEUR DE RETOUR

En cas de succès, la fonction virtGetTimeLastUpdate renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

NOM

virtGetTimeoutValue – accès à la valeur du délai d'attente

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetTimeoutValue(VirtContext VC, float *tout)
```

DESCRIPTION

La fonction `virtGetTimeoutValue` renvoie la valeur courante du délai d'attente avant abandon utilisé pour la communication avec le contrôleur embarqué.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `tout` est rempli par la fonction avec la valeur en secondes du délai d'attente.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetTimeoutValue` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetTimeoutValue

NOM

virtOpen – connexion de l'API au contrôleur du Virtuose

SYNOPTIQUE

```
#include "virtuoseAPI.h"
VirtContext virtOpen(const char *host)
```

DESCRIPTION

La fonction `virtOpen` réalise la connexion de l'API au contrôleur du Virtuose.

PARAMÈTRES

Le paramètre `host` contient l'URL (*Uniform Ressource Locator*) du contrôleur VIRTUOSE auquel l'API doit se connecter.
Dans la version courante de l'API, un seul type de protocole est supporté, l'URL est donc de la forme suivante :

```
"udpxdr://identification:numero_de_port+interface"
```

`udpxdr` désigne le protocole à utiliser.
`identification` contient le nom du contrôleur VIRTUOSE s'il peut être résolu par un serveur DNS, ou à défaut son adresse IP sous forme de quatre entiers décimaux séparés par des point (ex. "192.168.0.1").
`numero_de_port` est un entier décimal spécifiant le port à utiliser. Par défaut, il vaut 0, et dans ce cas l'API cherche un port libre en partant de 3131.
`interface` correspond au lien physique à utiliser (ignoré dans le cas du protocole `udpxdr`).

Dans le cas où seul `nom_ou_adresse_ip` est renseigné, l'URL est complété de la façon suivante :

```
"udpxdr://nom_ou_adresse_ip:0"
```

Attention : cette complétion automatique est limitée au seul protocole `udpxdr`.
Le pré-préfixe standard "url:" défini par la norme est supporté mais ignoré.

VALEUR DE RETOUR

En cas de succès, la fonction `virtOpen` renvoie un pointeur de type `VirtContext` (pointeur sur une structure de données masquée). Sinon, elle renvoie `NULL` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_SYNTAX_ERROR`
Le paramètre `host` ne correspond pas à un URL valide.

`VIRT_E_COMMUNICATION_FAILURE`
La fonction `virtOpen` n'a pas réussi à communiquer avec le contrôleur du Virtuose

`VIRT_E_OUT_OF_MEMORY`

La fonction `virtOpen` n'a pas réussi à allouer la mémoire nécessaire pour l'objet `VirtContext`.

`VIRT_E_INCOMPATIBLE_VERSION`
La version de l'API et celle du contrôleur sont incompatibles.

VOIR AUSSI

virtClose, virtGetErrorCode

NOM

virtSetCommandType – changement du mode de couplage

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetCommandType(VirtContext VC, VirtCommandType type)
```

DESCRIPTION

La fonction `virtSetCommandType` permet de modifier le mode de couplage.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `type` correspond au mode de couplage souhaité. Les valeurs possibles sont :

`COMMAND_TYPE_NONE`
Aucun mouvement possible

`COMMAND_TYPE_IMPEDANCE`
Couplage force/position

`COMMAND_TYPE_ADMITTANCE`
Couplage position/force simple

`COMMAND_TYPE_VIRTMECH`
Couplage position/force avec mécanisme virtuel

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetCommandType` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `type` ne correspond par à un mode valide.

VOIR AUSSI

virtGetCommandType

NOM

virtSetDebugFlags – activation d'un ou plusieurs niveaux de diagnostic

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetDebugFlags(VirtContext VC, unsigned short flags)
```

DESCRIPTION

La fonction `virtSetDebugFlags` permet d'activer les différents niveaux de diagnostic disponibles dans l'API VIRTUOSE.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `flags` est un champ de bits correspondant aux différents niveaux de diagnostic à activer :

- DEBUG_SERVO : informations relatives à la stabilité du sytème
- DEBUG_LOOP : informations relatives à l'exécution

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetDebugFlags` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

NOM

virtSetForce – envoie l'effort à appliquer au VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetForce(VirtContext VC, float *force_p_env_r_ps)
```

DESCRIPTION

La fonction `virtSetForce` envoie la force à appliquer au VIRTUOSE, dans le cas d'un couplage en effort.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `force_p_env_r_ps` contient la force à appliquer, sous la forme d'un torseur à 6 composantes, projeté dans le repère d'environnement et réduit au centre du repère outil.
Cette fonction n'est accessible qu'en mode de commande `COMMAND_TYPE_SIMPLE`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetForce` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetCommandType

NOM

virtSetForceFactor – fixe le facteur d'homothétie en effort

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetForceFactor(VirtContext VC, float factor)
```

DESCRIPTION

La fonction `virtSetForceFactor` permet de changer le facteur d'homothétie en effort, qui correspond à un changement d'échelle entre les efforts exercés au niveau du VIRTUOSE et ceux de la simulation.
La fonction doit être appelée avant la sélection du type de commande.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `factor` contient le facteur d'homothétie à appliquer. Un facteur plus petit que l'unité correspond à une amplification des efforts depuis le VIRTUOSE vers la simulation.
La fonction doit être appelée avant la sélection du type de commande.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetForceFactor` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `factor` est incorrecte. Cette erreur est générée dans les cas suivants :

- facteur d'homothétie négatif ou nul
- facteur d'homothétie trop grand ou trop faible pour garantir la stabilité du système

`VIRT_E_CALL_TIME`
La fonction est appelée après la sélection du type de commande.

VOIR AUSSI

virtGetForceFactor, virtSetSpeedFactor

NOM

virtSetIndexingMode – modification du mode d'indexation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetIndexingMode(VirtContext VC, VirtIndexingType mode)
```

DESCRIPTION

La fonction `virtSetIndexingMode` permet de modifier le mode d'indexation (aussi appelé mode de décalage).

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `mode` correspond au mode d'indexation souhaité. Les valeurs possibles sont :

`INDEXING_ALL`
L'indexation agit sur les translations et les rotations. Le décalage s'effectue lors de l'appui du bouton de décalage ou lorsqu'il n'y a pas de puissance (bouton de facade ou relâchement du bouton « homme mort »).

`INDEXING_TRANS`
L'indexation agit sur les translations seulement. A la remise sous puissance, le bras est contraint sur un segment en rotation pour revenir à l'orientation au moment de l'arrêt de la puissance.

`INDEXING_NONE`
L'indexation est inopérante, même hors puissance. A la remise sous puissance, le bras est contraint sur un segment pour revenir à la position au moment de l'arrêt de la puissance.

Les autres modes correspondent aux modes ci-dessus à la différence que le retour de d'effort est inhibé pendant le décalage.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetIndexingMode` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `mode` ne correspond par à un mode valide.

VOIR AUSSI

virtGetIndexingMode

NOM

virtSetObservationFrame – modification du repère d'observation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetObservationFrame(VirtContext VC, float *pos)
```

DESCRIPTION

La fonction `virtSetObservationFrame` permet de modifier la position du repère d'observation par rapport au repère d'environnement.
Elle doit être appelée avant la sélection du type de commande.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos` correspond au déplacement du repère d'observation par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetObservationFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Le paramètre `pos` n'est pas un déplacement valide.

`VIRT_E_CALL_TIME`
La fonction est appelée après la sélection du type de commande.

VOIR AUSSI

VirtGetObservationFrame
VirtSetObservationFrameSpeed

NOM

virtSetObservationFrameSpeed – modification de la vitesse du repère d'observation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetObservationFrameSpeed(VirtContext VC, float *speed)
```

DESCRIPTION

La fonction `virtSetObservationFrameSpeed` permet de modifier la vitesse du repère d'observation par rapport au repère d'environnement.
Elle doit être appelée avant la sélection du type de commande.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `speed` correspond à la vitesse de déplacement du repère d'observation par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetObservationFrameSpeed` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_CALL_TIME`
La fonction est appelée après la sélection du type de commande.

VOIR AUSSI

virtGetObservationFrame

NOM

virtSetPosition – envoie au contrôleur la position de l'objet couplé

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetPosition(VirtContext VC, float *pos_p_env)
```

DESCRIPTION

La fonction `virtSetPosition` envoie au contrôleur la position de l'objet auquel le VIRTUOSE est couplé.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos_p_env` est un vecteur déplacement à 7 composantes (cf. glossaire).

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetPosition` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `pos_p_env` est incorrecte. Cette erreur est générée dans les cas suivants :

- quaternion non normé

VOIR AUSSI

VirtAttachVO, virtSetSpeed

NOM

virtSetSpeed – envoie au contrôleur embarqué la vitesse de l'objet couplé

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetSpeed(VirtContext VC, float *speed_p_env_r_ps)
```

DESCRIPTION

La fonction `virtSetSpeed` envoie au contrôleur du VIRTUOSE la vitesse de l'objet auquel il est couplé.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `speed_p_env_r_ps` correspond à la vitesse du centre de l'objet, exprimée sous forme d'un torseur 6 composantes projeté dans le repère d'environnement et réduit au centre de l'objet.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetSpeed` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtAttachVO, virtSetPosition

NOM

virtSetSpeedFactor – fixe le facteur d'homothétie en déplacement

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetSpeedFactor(VirtContext VC, float factor)
```

DESCRIPTION

La fonction `virtSetSpeedFactor` permet de changer le facteur d'homothétie en déplacement, qui correspond à un changement d'échelle entre les déplacements du VIRTUOSE et ceux de la simulation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `factor` contient le facteur d'homothétie à appliquer. Un facteur plus grand que l'unité correspond à une dilatation de l'espace de travail du Virtuose.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetSpeedFactor` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `factor` est incorrecte. Cette erreur est générée dans les cas suivants :

- facteur d'homothétie négatif ou nul
- facteur d'homothétie trop grand ou trop faible pour garantir la stabilité du système

VOIR AUSSI

virtGetSpeedFactor, virtSetForceFactor

NOM

virtSetTimeoutValue – modification de la valeur du délai d'attente

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTimeoutValue(VirtContext VC, float tout)
```

DESCRIPTION

La fonction `virtSetTimeoutValue` permet de modifier la valeur du délai d'attente avant abandon utilisé dans la communication avec le contrôleur embarqué.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `tout` contient la valeur du nouveau délai d'attente à utiliser.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetTimeoutValue` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `tout` est incorrecte.

VOIR AUSSI

virtGetTimeoutValue

NOM

virtSetTimeStep – fixe la valeur du pas d'échantillonnage de la simulation

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTimeStep(VirtContext VC, float timeStep)
```

DESCRIPTION

La fonction `virtSetTimeStep` permet de communiquer au contrôleur embarqué la valeur du pas d'échantillonnage de la simulation. Cette valeur est utilisée par celui-ci afin de garantir la stabilité du système couplé.
Elle doit être appelée avant la sélection du type de commande.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `timeStep` correspond au pas d'échantillonnage de la simulation, exprimé en secondes.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetTimeStep` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `timeStep` est incorrecte. Cette erreur est générée dans les cas suivants :

- valeur négative ou nulle
- valeur inférieure au pas d'échantillonnage utilisé par le contrôleur embarqué
- valeur trop grande pour assurer la stabilité du système

`VIRT_E_CALL_TIME`
La fonction est appelée après la sélection du type de commande.

VOIR AUSSI

NOM

virtVmSetType– sélection du type de mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmSetType(VirtContext VC, virtVmType type)
```

DESCRIPTION

La fonction virtVmSetType permet de sélectionner le type de mécanisme virtuel.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `type` correspond au type de mécanisme virtuel. Les types disponibles sont :

VM_TYPE_CartMotion : guide virtuel à zéro degré de liberté correpondant à un point, défini par l'origine du repère de base du mécanisme virtuel,
VM_TYPE_SPLINE : guide virtuel à un degré de liberté basé sur une courbe spline,
VM_TYPE_Rx : guide virtuel à un degré de liberté correpondant à une liaison de type pivot d'axe Ox du repère de base du mécanisme virtuel,
VM_TYPE_Tx : guide virtuel à un degré de liberté correpondant à une liaison de type glissière d'axe Ox du repère de base du mécanime virtuel,
VM_TYPE_TxTy : guide virtuel à deux degré de liberté correpondant à deux liaisons de type glissière d'axe Ox et Oy du repère de base du mécanime virtuel,
VM_TYPE_TxRx : guide virtuel à deux degré de liberté correspondant à une liaison de type pivot-glissant d'axe Ox du repère de base du mécanime virtuel.
VM_TYPE_RxRyRz : guide virtuel à trois degré de liberté correspondant à une liaison de type rotule par rapport au repère de base du mécanime virtuel.
VM_TYPE_Crank : guide virtuel à deux degré de liberté correspond à un mouvement d'une manivelle autour de l'axe Ox du repère de base du mécanisme virtuel.

VALEUR DE RETOUR

En cas de succès, la fonction virtVmSetType renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
La valeur du paramètre `type` est incorrecte.

VOIR AUSSI

virtVmSetBaseFrame

NOM

virtVmActivate– activation du mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmActivate(VirtContext VC)
```

DESCRIPTION

La fonction `virtVmActivate` permet d'activer le mécanisme virtuel une fois le VIRTUOSE positionné.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtVmActivate` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmDeactivate

NOM

virtVmDeactivate – désactivation du mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmDeactivate(VirtContext VC)
```

DESCRIPTION

La fonction `virtVmDeactivate` permet de désactiver le mécanisme virtuel précédemment activé.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtVmDeactivate` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmActivate

NOM

virtVmSetSamplingTimeStep – fixe la valeur du pas d'échantillonnage de l'enregistrement

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmSetSamplingTimeStep (VirtContext VC, float timeStep
                               unsigned int* recordTime)
```

DESCRIPTION

La fonction `VirtVmSetSamplingTimeStep` permet de fixer le pas d'échantillonnage de l'enregistrement de la trajectoire.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `timeStep` correspond au pas d'échantillonnage exprimé en secondes.
Le paramètre `recordTime` correspond au temps maximum d'enregistrement exprimé en secondes.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmSetSamplingTimeStep` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Le paramètre du paramètre `timeStep` est incorrect. Cette erreur est générée dans les cas suivants :

- valeur négative ou nulle
- valeur inférieure au pas d'échantillonnage utilisé par le contrôleur embarqué

VOIR AUSSI

VirtTrajRecordStart, virtTrajRecordStop

NOM

virtTrajRecordStart – démarre l'enregistrement

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtTrajRecordStart (VirtContext VC)
```

DESCRIPTION

La fonction `VirtTrajRecordStart` permet de lancer l'enregistrement de la trajectoire.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtTrajRecordStart` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtTrajRecordStop

NOM

virtTrajRecordStop – arrête l'enregistrement

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtTrajRecordStop (VirtContext VC)
```

DESCRIPTION

La fonction `VirtTrajRecordStop` permet d'arrêter l'enregistrement de la trajectoire.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtTrajRecordStop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtTrajRecordStart

NOM

virtVmStartTrajSampling – effectue une demande de lecture de la trajectoire

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmStartTrajSampling (VirtContext VC, unsigned int* nbSamples)
```

DESCRIPTION

La fonction `VirtVmStartTrajSampling` permet de lancer une lecture d'une partie des points de trajectoire précédemment enregistrés.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `nbSamples` correspond au nombre de points.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmStartTrajSampling` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Le paramètre `nbSamples` est négatif ou supérieur à 3000 points.

VOIR AUSSI

virtVmGetTrajSamples

NOM

virtVmGetTrajSamples – récupère les points de trajectoire

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmGetTrajSamples (VirtContext VC, float* samples)
```

DESCRIPTION

La fonction `VirtVmGetTrajSamples` permet de récupérer les points de trajectoire précédemment enregistrés. Cette fonction est bloquante jusqu'à la fin de la réception des points depuis le contrôleur.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `samples` correspond à un tableau de points.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmGetTrajSamples` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmStartTrajSampling

NOM

virtGetPhysicalPosition – récupère la position du bras Virtuose

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetPhysicalPosition (VirtContext VC, float* pos)
```

DESCRIPTION

La fonction `VirtGetPhysicalPosition` permet de récupérer la position de la poignet du Virtuose par rapport à sa base.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos` correspond à la position. Elle est exprimée par rapport au repère de base du Virtuose.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtGetPhysicalPosition` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetBaseFrame

NOM

virtGetAvatarPosition – récupère la position du bras Virtuose avec décalage

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetAvatarPosition (VirtContext VC, float* pos)
```

DESCRIPTION

La fonction `VirtGetAvatarPosition` permet de récupérer la position du poignet du Virtuose par rapport au repère d'environnement. Cette position tient compte du coefficient de déplacement.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `pos` correspond à la position. Elle est exprimée par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtGetPhysicalPosition` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetSpeedFactor

NOM

virtVmSetDefaultToCartesianPosition – bloque le bras avant l'activation du guide virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmDefaultToCartesienPosition (VirtContext VC)
```

DESCRIPTION

La fonction `VirtVmDefaultToCartesienPosition` permet de bloquer le bras du virtuose avant activation du mécanisme virtuel. Le bras sera de nouveau bloqué après désactivation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmDefaultToCartesianPosition` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetDefaultToTransparentMode
virtVmActivate
virtVmDeactivate

NOM

virtVmSetDefaultToTransparentMode – libère le bras avant l'activation du guide virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmDefaultToTransparentMode (VirtContext VC)
```

DESCRIPTION

La fonction `VirtVmDefaultToCartesienPosition` permet de mettre le bras du virtuose en mode transparent, avant activation du mécanisme virtuel. Le bras sera de nouveau dans ce mode après désactivation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmDefaultToTransparentMode` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetDefaultToCartesianMode
virtVmActivate
virtVmDeactivate

NOM

virtVmSetBaseFrame – positionnement de la base du mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmSetBaseFrame (VirtContext VC, float *base)
```

DESCRIPTION

La fonction `VirtVmSetBaseFrame` permet de positionner le repère de base du mécanisme virtuel.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `base` correspond au repère de base. Il est exprimé par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmSetBaseFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmType
virtVmGetBaseFrame

NOM

virtVmGetBaseFrame – lecture de la base du mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmGetBaseFrame (VirtContext VC, float *base)
```

DESCRIPTION

La fonction `VirtVmGetBaseFrame` permet de lire le repère de base du mécanisme virtuel.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `base` correspond au repère de base. Il est exprimé par rapport au repère d'environnement.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmGetBaseFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmSetBaseFrame

NOM

virtConvertRGBToGrayscale – conversion RGB en niveau de gris

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtConvertRGBToGrayscale (VirtContext VC, float *rgb,
float* gray)
```

DESCRIPTION

La fonction `VirtConvertRGBToGrayscale` permet de convertir une couleur définie en RGB en niveau de gris. Les proportions réalisées sont :

29.9% de rouge,
58.7% de vert,
11.4% de bleu.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `rgb` est un tableau de trois composants dont le premier correspond à la couleur rouge, le second à la couleur verte et le troisième à la couleur bleu.

VALEUR DE RETOUR

Dans tous les cas, la fonction `VirtConvertRGBToGrayscale` renvoie `0`.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtVmSetTexture

NOM

virtSetTexture – application d'une texture en un point de contact

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTexture (VirtContext VC, float *H_texture_p_env,
float* intensite, int reinit)
```

DESCRIPTION

La fonction `VirtSetTexture` permet d'appliquer une texture en un point de contact suivant le niveau de gris ou plutôt de la variation des niveaux de gris en différents points. Elle permet de créer des sensations de reliefs, les bosses sont associées au blanc et les creux associés au noir.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `H_texture_p_env` est un vecteur déplacement à 7 composants correspondant à la position du point de contact.
Le paramètre `intensite` est un flottant correspondant à un niveau de gris entre 0 et 1.
Le paramètre `reinit` réinitialise le module de génération de texture au niveau du contrôleur. Il doit être positionner à 1 lors d'une nouvelle collision à 0 pour la prise en compte de texture.

VALEUR DE RETOUR

Dans tous les cas, la fonction `VirtSetTexture` renvoie `0`.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Le paramètre `intensité` n'est pas compris entre 0 et 1.

VOIR AUSSI

virtConvertRGBToGrayscale

NOM

virtSetTextureForce – application d'une force liée à une texture

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTextureForce (VirtContext VC, float
*W_texture_p_env_r_ps)
```

DESCRIPTION

La fonction `virtSetTextureForce` permet d'appliquer directement une force.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `W_texture_p_env_r_ps` correspond à la force à appliquer, sous la forme d'un torseur à 6 composantes, projeté dans le repère d'environnement et réduit au centre du repère outil.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetTextureForce` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

NOM

virtVmWaitUpperBound – attente d'une butée supérieure d'un mécanisme virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtVmWaitUpperBound (VirtContext VC)
```

DESCRIPTION

La fonction `VirtVmWaitUpperBound` n'est disponible qu'en mode COMMAND_TYPE_VIRTMECH et pour les mécanismes déplacement cartésien et splines.
Avant activation, la fonction est non bloquante et retourne un code d'erreur.
Après activation, la fonction est bloquante. En mécanisme déplacement cartésien, elle est débloquée lorsque la butée supérieure du segment rejoignant le point final est atteint. En mécanisme Spline, elle est débloquée lorsque la butée supérieure de la spline est atteinte.
Dans tous les cas, la fonction est débloquée sur désactivation du mécanisme.
Cette fonction est souvent utilisée en mode robot et permet d'enchainer les mécanismes virtuels.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtVmWaitUpperBound` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VM_TYPE`
Le type de mécanisme est incorrecte.

`VIRT_E_WRONG_MODE`
Le type de commande est incorrecte.

`VIRT_E_CALL_TIME`
L'appel de la fonction est effectuée avant activation du mécanisme.

VOIR AUSSI

virtVmSetType
virtVmActivate
virtVmDeactivate

NOM

virtDisableControlConnexion – désactivation du contrôle de connexion entre la virtuoseAPI et le contrôleur

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtDisableControlConnexion (VirtContext VC, int disable)
```

DESCRIPTION

La fonction `VirtDisableControlConnexion` désactive ou réactive le mécanisme de surveillance de la connexion entre la librairie virtuoseAPI et le logiciel intégré dans le contrôleur.
Ce contrôle a été mis en place pour palier à un plantage de la simulation et remettre le contrôleur dans un état déterminé. Il considère la simulation planté lorsque celle-ci n'envoit pas d'information au bout de 2 secondes.
Cette fonction permet ainsi le debugage de la simulation et de positionner des points d'arrêt sans interrompre le contrôleur.
Par contre, après l'utilisation de cette fonction puis plantage de la simulation, nous ne garantissons pas l'état du contrôleur. Il est aussi impérati d'utiliser la fonction `virtClose` à la fin de la simulation.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `disable` désactive le mécanisme s'il est positionné à 1, et permet de le réactiver s'il est positionné à 0.

VALEUR DE RETOUR

En cas de succès, la fonction `virtDisableControlConnexion` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtClose

NOM

virtSetPeriodicFunction – initialisation du mécanisme d'appel de fonction périodique

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetPeriodicFunction (VirtContext VC,
                             void (*fn) (VirtContext, void*),
                             float* period,
                             void* arg);
```

DESCRIPTION

La fonction `VirtSetPeriodicFunction` permet de fournir un pointeur sur la fonction qui sera appelé cycliquement, à la période fournit en paramètre.
L'appel de fonction est synchronisé sur la réception de trame en provenance du controleur, ce mécanisme fournit de bonne performance temps-réel.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `fn` est un pointeur de fonction.
Le paramètre `period` correpond à la période d'appel de la fonction, exprimé en secondes. Celle-ci doit correspondre à la période fournie lors de l'appel à la fonction `virtSetTimeStep`.
Le paramètre `arg` correspond à l'argument passé lors de l'appel.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtSetPeriodicFunction` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtStartLoop
virtStopLoop
virtSetTimeStep

NOM

virtStartLoop – démarrage du mécanisme d'appel de fonction périodique

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtStartLoop (VirtContext VC);
```

DESCRIPTION

La fonction `VirtStartLoop` permet de lancer le mécanisme d'appel de fonction périodique définit précédemment lors de l'appel à la fonction `VirtSetPeriodicFunction`.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `VirtStartLoop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetPeriodicFunction
virtStopLoop

NOM

virtStopLoop – arrêt du mécanisme d'appel de fonction périodique

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtStopLoop (VirtContext VC);
```

DESCRIPTION

La fonction `VirtStopLoop` permet d'arrêter le mécanisme d'appel de fonction périodique définit précédemment lors de l'appel à la fonction `VirtSetPeriodicFunction`.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction v`irtStopLoop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSetPeriodicFunction
virtStartLoop

NOM

virtIsInBounds – indique si l'opérateur se situe en butée sur le bras VIRTUOSE

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtIsInBounds (VirtContext VC, unsigned int *bounds);
```

DESCRIPTION

La fonction `VirtIsInBounds` permet de savoir si l'opérateur se situe en butée sur le bras VIRTUOSE.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `bounds` est rempli par la fonction. Il représente un ensemble de bit dont voici la signification :

- le bit `VIRT_BOUND_LEFT_AXE_1` correspond à la butée gauche sur l'axe 1,
- le bit `VIRT_BOUND_RIGHT_AXE_1` correspond à la butée droite sur l'axe 1,
- le bit `VIRT_BOUND_SUP_AXE_2` correspond à la butée supérieure sur l'axe 2,
- le bit `VIRT_BOUND_INF_AXE_2` correspond à la butée inférieure sur l'axe 2,
- le bit `VIRT_BOUND_SUP_AXE_3` correspond à la butée supérieure sur l'axe 3,
- le bit `VIRT_BOUND_INF_AXE_3` correspond à la butée inférieure sur l'axe 3,
- le bit `VIRT_BOUND_LEFT_AXE_4` correspond à la butée gauche sur l'axe 4,
- le bit `VIRT_BOUND_RIGHT_AXE_4` correspond à la butée droite sur l'axe 4,
- le bit `VIRT_BOUND_SUP_AXE_5` correspond à la butée supérieure sur l'axe 5,
- le bit `VIRT_BOUND_INF_AXE_5` correspond à la butée inférieure sur l'axe 5,
- le bit `VIRT_BOUND_LEFT_AXE_6` correspond à la butée gauche sur l'axe 6,
- le bit `VIRT_BOUND_RIGHT_AXE_6` correspond à la butée droite sur l'axe 6.

VALEUR DE RETOUR

En cas de succès, la fonction `virtStopLoop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

NOM

virtGetAlarm – récupère les alarmes en cours

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetAlarm (VirtContext VC, unsigned int *alarm);
```

DESCRIPTION

La fonction `VirtGetAlarm` permet de récupérer les alarmes en cours.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `alarm` est rempli par la fonction. Il représente un ensemble de bit dont voici la signification :

- le bit `VIRT_ALARM_OVERHEAT` indique qu'un des moteurs est en chauffe. Le retour d'effort est alors réduit pour maintenir une température acceptable.
- le bit `VIRT_ALARM_SATURATE` indique une saturation des moteurs; par défaut, le retour d'effort est au maximum de 35 Newton en force instantanée et de 3.1 Nm en couple (sur les VIRTUOSE 6D).
- le bit `VIRT_ALARM_CALLBACK_OVERRUN` indique que le temps d'exécution de la fonction périodique définie lors de l'appel de `virtSetPeriodicFunction` dépasse le temps fourni en paramètre. Aucun traitement n'est réalisé en cas de dépassement. Il est indispensable que le temps d'exécution de la callback soit inférieure au temps précisé.

VALEUR DE RETOUR

En cas de succès, la fonction `virtStopLoop` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtSaturateTorque
virtSetPeriodicFunction
virtSetTimeStep

NOM

virtActiveSpeedControl – active la commande en vitesse

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtActiveSpeedControl (VirtContext VC,
                            float radius,
                            float speedFactor);
```

DESCRIPTION

La fonction `virtActiveSpeedControl` permet d'activer la commande en vitesse.
On définie une sphère dont le centre correspond au centre de l'espace de travail du périphérique haptique et de rayon, celui fournit en paramètre.
A l'intérieure de la sphère, l'opérateur contrôle l'objet en position.
A l'extérieure de la sphère, l'opérateur contrôle l'objet en vitesse. Plus la position du poignée s'écarte de la sphère, plus la vitesse de l'objet augmente. A l'intersection, la vitesse est nulle.
La valeur du rayon pouvant être utilisée pour un virtuose 3D 15-25 est de 0.1. Pour un virtuose 6D 35-45 est de 0.2.
La valeur du coeficient dépend essentiellement des dimensions de la scène virtuelle (1.0 par exemple).

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `radius` correspond au rayon de la sphère précédemment décrite.
Le paramètre `speedFactor` est un coefficient multiplicatif sur la vitesse de l'objet virtuel.

VALEUR DE RETOUR

En cas de succès, la fonction `virtActiveSpeedControl` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Le rayon ou le coefficient multiplicatif ne sont pas positifs.

VOIR AUSSI

virtDeactiveSpeedControl

NOM

virtDeactiveSpeedControl – désactive la commande en vitesse

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtDeactiveSpeedControl (VirtContext VC)
```

DESCRIPTION

La fonction `virtDeactiveSpeedControl` permet de désactiver la commande en vitesse. L'opérateur contrôle l'objet en position sur tout l'espace de travail.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.

VALEUR DE RETOUR

En cas de succès, la fonction `virtDeactiveSpeedControl` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

NOM

virtSetCatchFrame – position du repère de prise d'un objet virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetCatchFrame (VirtContext VC, float* frame)
```

DESCRIPTION

La fonction `virtSetCatchFrame` permet de position le repère de prise par rapport au repère du centre de l'objet.
Ce décalage par rapport au centre doit être fixé avant la demande de couplage effectuée par la fonction `virtAttachVO`. Il est remis à zéro à chaque appel de la fonction `virtDettachVO`.
La partie orientation du repère de prise n'a aucune influence.

La fonction n'est disponible que dans le mode COMMAND_TYPE_VIRTMECH.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `frame` correpond au repère de prise de l'objet exprimé dans le repère du centre de gravité de l'objet virtuel.

VALEUR DE RETOUR

En cas de succès, la fonction `virtSetCatchFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

`VIRT_E_INCORRECT_VALUE`
Les valeurs du paramètre `frame` est incorrect (quaternion non normé)

VOIR AUSSI

virtAttachVO
virtDettachVO
virtGetCatchFrame

NOM

virtGetCatchFrame – lecture du repère de prise d'un objet virtuel

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetCatchFrame (VirtContext VC, float* frame)
```

DESCRIPTION

La fonction `virtGetCatchFrame` permet de récupérer le repère de prise par rapport au repère du centre de l'objet, préalablement fixer par la fonction `virtSetCatchFrame`.
La fonction n'est disponible que dans le mode COMMAND_TYPE_VIRTMECH.

PARAMÈTRES

Le paramètre `VC` correspond à l'objet `VirtContext` renvoyé par la fonction `virtOpen`.
Le paramètre `frame` correpond au repère de prise de l'objet exprimé dans le repère du centre de gravité de l'objet virtuel.

VALEUR DE RETOUR

En cas de succès, la fonction `virtGetCatchFrame` renvoie `0`. Dans le cas contraire, elle renvoie `-1` et la fonction `virtGetErrorCode` permet d'accéder au code d'erreur.

ERREURS

`VIRT_E_INVALID_CONTEXT`
Le paramètre `VC` ne correspond pas à un objet `VirtContext` valide.

VOIR AUSSI

virtAttachVO
virtDettachVO
virtSetCatchFrame

6 GLOSSAIRE

Quaternion

On peut passer d'une rotation angulaire d'angle θ autour d'un vecteur (a, b, c) à un quaternion par la formule suivante :

$$\begin{cases} qx = a\sin\left(\frac{\theta}{2}\right) \\ qy = b\sin\left(\frac{\theta}{2}\right) \\ qz = c\sin\left(\frac{\theta}{2}\right) \\ qw = \cos\left(\frac{\theta}{2}\right) \end{cases}$$

On peut ensuite normaliser le quaternion en divisant chacune de ses quatre composantes par sa norme calculée ainsi :

$$n = \sqrt{qx^2 + qy^2 + qz^2 + qw^2}$$

Torseur cinématique

Soit un solide (S) attaché à un repère (R1)={O1,x1,y1,z1} en mouvement dans un repère (R0)={O,x0,y0,z0}.

La vitesse d'un point M quelconque de (S) par rapport à (R0) est notée :

$$\vec{V}(M \in S/R_0) = \left(\frac{dOM}{dt}\right)_{R_0} = \left(\frac{dOO_1}{dt}\right)_{R_0} + x\left(\frac{d\vec{x}_1}{dt}\right)_{R_0} + y\left(\frac{d\vec{y}_1}{dt}\right)_{R_0} + z\left(\frac{d\vec{z}_1}{dt}\right)_{R_0}$$

L'indice R0 de la dérivation / temps signifie que, dans le calcul de la vitesse, on suppose les vecteurs de base de (R0) constants. On note : $\left(\frac{dOO_1}{dt}\right)_{R_0} = v_x\vec{x}_0 + v_y\vec{y}_0 + v_z\vec{z}_0$

Le vecteur rotation de S dans R0 est le vecteur Ω tel que :

$$\begin{cases} \left(\frac{d\vec{x}_1}{dt}\right)_{R_0} = \vec{\Omega} \wedge \vec{x}_1 \\ \left(\frac{d\vec{y}_1}{dt}\right)_{R_0} = \vec{\Omega} \wedge \vec{y}_1 \\ \left(\frac{d\vec{z}_1}{dt}\right)_{R_0} = \vec{\Omega} \wedge \vec{z}_1 \end{cases}$$

Il est indépendant du choix du repère R1 lié à S. On note : $\Omega = \omega_x x_1 + \omega_y y_1 + \omega_z z_1$

On peut en déduire la relation suivante :

$$V(M \in S/R_0) = V(O_1/R_0) + \Omega \wedge O_1M$$

Le champ des vitesses de S est celui du champ de moment d'un torseur appelé torseur cinématique dont les éléments de réduction sont :

$$(VS/R_0)_{O_1} = \begin{Bmatrix} \vec{V}(O_1 \in S/R_0) \\ \Omega(S/R_0) \end{Bmatrix} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$$

Dans ce document, on parlera du « torseur cinématique de S réduit en O1 et projeté dans R0 ».