HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

LABO – MET

Entraînement réglé par un moteur synchrone à aimants permanents

Michel Girardin

12.03.2021

Table des matières

1. Théorie

Onduleur triphasé

La *Figure 1* rappelle le schéma de l'onduleur triphasé qui permettra de contrôler le moteur synchrone à aimants permanents.

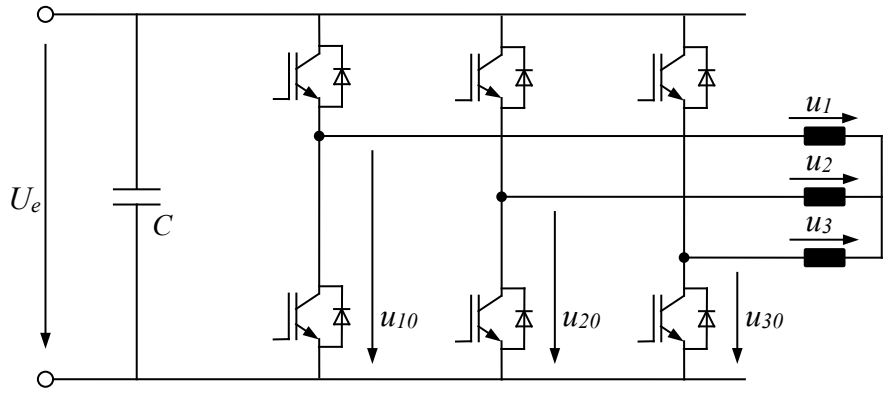


Figure 1 Schéma de l'onduleur triphasé

Les trois tensions de phase du moteur dépendent directement des trois tensions de branche de l'onduleur selon les relations ci-dessous.

$$u_1 = \frac{1}{3}(2\,u_{10} - u_{20} - u_{30}) \quad (1)$$

$$u_2 = \frac{1}{3}(2\,u_{20} - u_{10} - u_{30}) \quad (2)$$

$$u_3 = \frac{1}{3}(2\,u_{30} - u_{10} - u_{20}) \quad (3)$$

Phaseurs spatiaux appliqués au moteur synchrone

La théorie des phaseurs spatiaux s'applique très simplement au moteur synchrone à aimants permanents. La *Figure 2* montre les 3 référentiels utilisés pour exprimer les courants de phase du moteur. On voit également, sur cette figure, les relations utiles pour passer d'un référentiel à un autre (il s'agit de la transformée de Park et de son inverse).

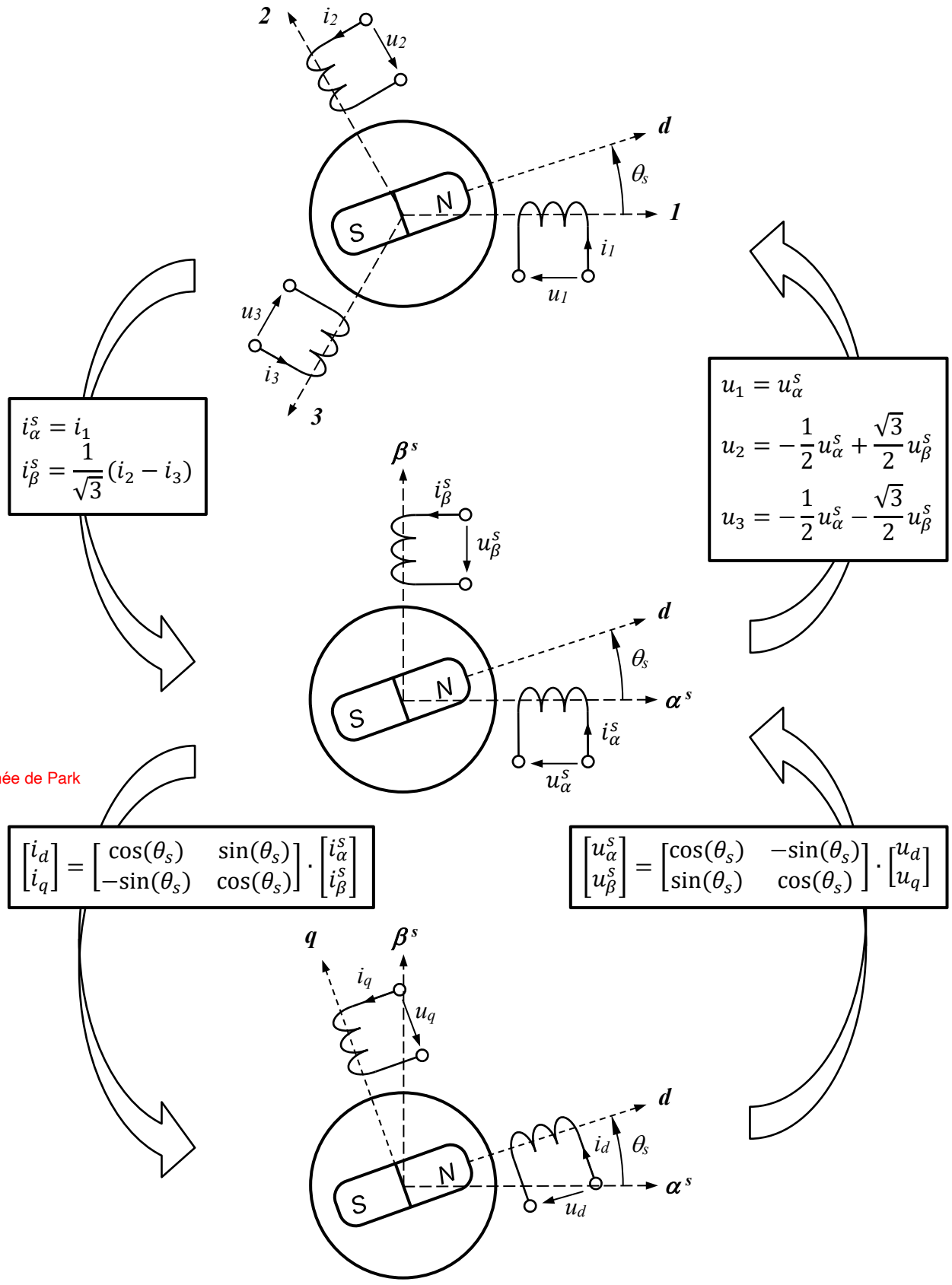


Figure 2 Phaseurs spatiaux appliqués au moteur synchrone à aimants permanents

Les schémas blocs utilisés pour ces transformations sont présentés ci-dessous.

Les transformations entre les grandeurs de phase et le phaseur spatial dans le référentiel fixe sont présentées à la *Figure 3*.

Figure 3 Schémas blocs pour la transformation en phaseur spatial dans le référentiel fixe

Les transformations entre le phaseur spatial dans le référentiel fixe et le phaseur spatial dans le référentiel tournant sont présentées à la *Figure 4*.

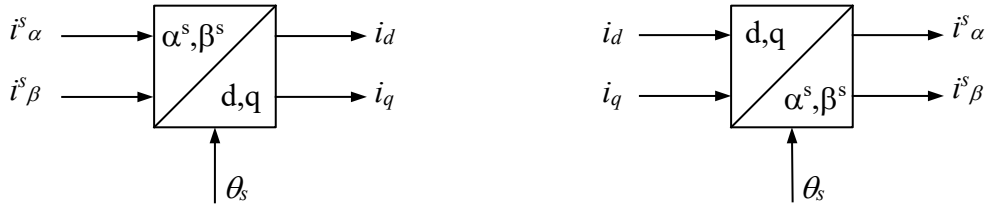


Figure 4 Schémas blocs pour les changements de référentiel du phaseur spatial.

Modèle du moteur synchrone à aimants permanents dans le référentiel lié au rotor

Dans un référentiel lié au rotor les tensions dans les axes d et q s'expriment par :

$$u_d = R_s\, i_d + \frac{d\Psi_d}{dt} - \omega_s \Psi_q \tag{4}$$

$$u_q = R_s\, i_q + \frac{d\Psi_q}{dt} + \omega_s \Psi_d \tag{5}$$

Les flux et leur dérivée sont donnés par :

$$\Psi_d = L_d\, i_d + \Psi_f \qquad \text{où } \Psi_f \text{ est le flux créé par les aimants permanents} \tag{6}$$

$$\Psi_q = L_q\, i_q \tag{7}$$

$$\frac{d\Psi_d}{dt} = L_d\, \frac{di_d}{dt} \tag{8}$$

$$\frac{d\Psi_q}{dt} = L_q\, \frac{di_q}{dt} \tag{9}$$

Les équations de tension deviennent :

$$u_d = R_s\, i_d + L_d\, \frac{di_d}{dt} - \omega_s\, L_q\, i_q \tag{10}$$

$$u_q = R_s\, i_q + L_q\, \frac{di_q}{dt} + \omega_s\, L_d\, i_d + \omega_s\, \Psi_f \tag{11}$$

Pour un rotor isotrope, $L_d = L_q = L_s$.
De plus, la tension induite de mouvement $(\omega_s \cdot \Psi_f)$ peut s'écrire $(\Omega_m \cdot k_E)$

$$u_d = R_s\, i_d + L_s\, \frac{di_d}{dt} - \omega_s\, L_s\, i_q \tag{12}$$

$$u_q = R_s\, i_q + L_s\, \frac{di_q}{dt} + \omega_s\, L_s\, i_d + \Omega_m\, k_E \tag{13}$$

où : ω_s est la pulsation électrique du stator
Ω_m est la vitesse de rotation mécanique du rotor

$$\omega_s = p\, \Omega_m \qquad \text{où p est le nombre de paires de pôles du moteur} \tag{14}$$

On voit qu'il existe un couplage entre les axes d et q. Les tensions de chaque axe sont influencées par le courant de l'autre axe.

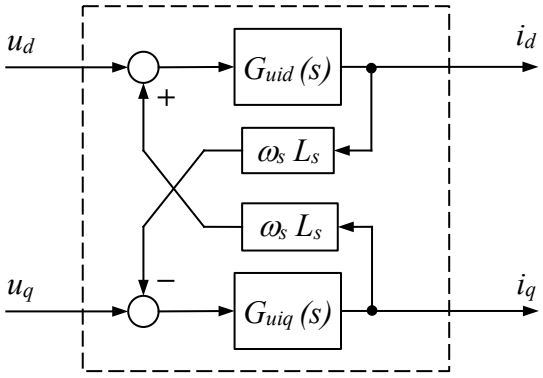


Figure 5 Schéma bloc du moteur synchrone avec le couplage entre les axes d et q

Le découplage des axes consiste à compenser le couplage interne du moteur par une commande a priori. En principe, comme le courant i_d est maintenu à 0, la commande a priori sur la tension u_q n'est pas indispensable.

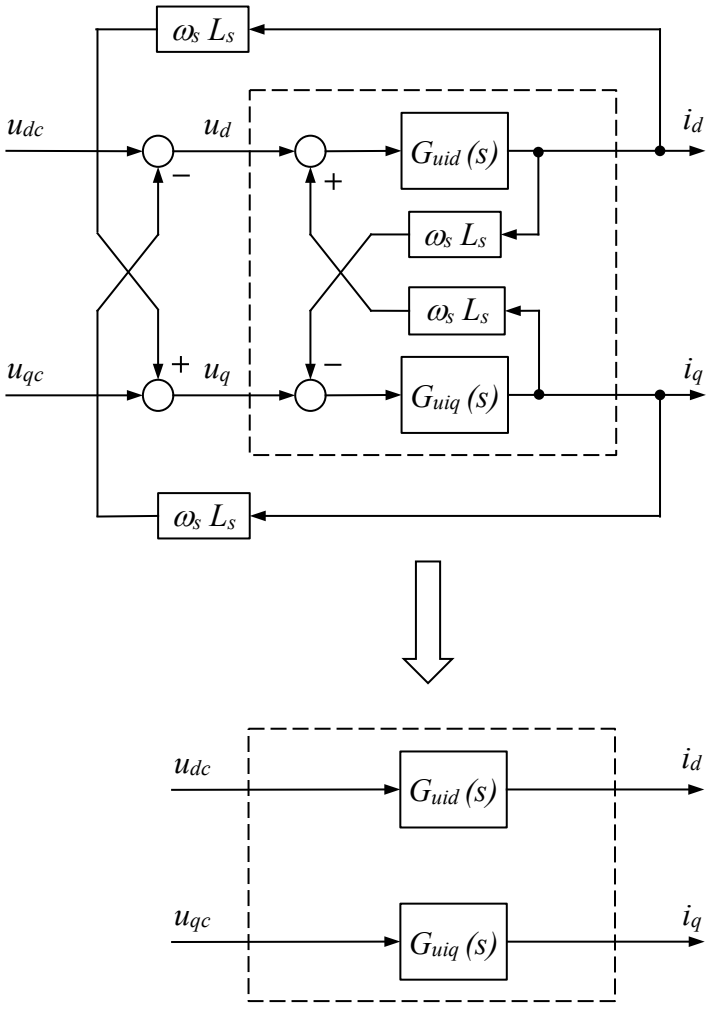


Figure 6 Schéma bloc du moteur synchrone avec les commandes a priori de découplage

Control vectoriel dans le domaine des entraînements réglés

Dans le domaine des entraînements réglés, la méthode la plus efficace est la régulation en cascade (ou régulation imbriqué) avec une boucle de régulation interne rapide permettant de limiter le couple ou le courant, et une boucle externe superposée pour le réglage de vitesse. Cette structure peut être étendue avec la superposition d'une boucle de réglage de position. Le fait que chaque variable peut être limitée par la consigne correspondante est un sérieux avantage de la régulation en cascade.

Dans le cas des moteurs triphasés, le réglage de couple est garanti par le réglage de la composante *q* du phaseur spatial du courant dans le référentiel tournant i_q. Le flux rotorique est contrôlé par la composante *d* du courant dans le référentiel tournant i_d.

Pour un moteur synchrone à aimants permanents, i_d est habituellement maintenu à zéro. Dans le cas d'un moteur asynchrone, i_d est utilisé pour produire et régler le flux rotorique.

A la *Figure 7*, la régulation de courant est faite dans le bloc Ri. Les consignes sont les deux composantes du phaseur spatial du courant dans le référentiel tournant i_{dc} et i_{qc}. La mesure du courant se fait sur les trois phases i_1, i_2 et i_3. La position mécanique du rotor θ est mesurée par un capteur de position directement sur l'arbre du moteur. La position électrique θ_s est calculée en fonction du nombre de pôles et de la position de l'axe magnétique. Ensuite, le régulateur de courant fournit la modulation des trois signaux digitaux d_1, d_2 et d_3 pour commander les trois branches de l'onduleur triphasé.

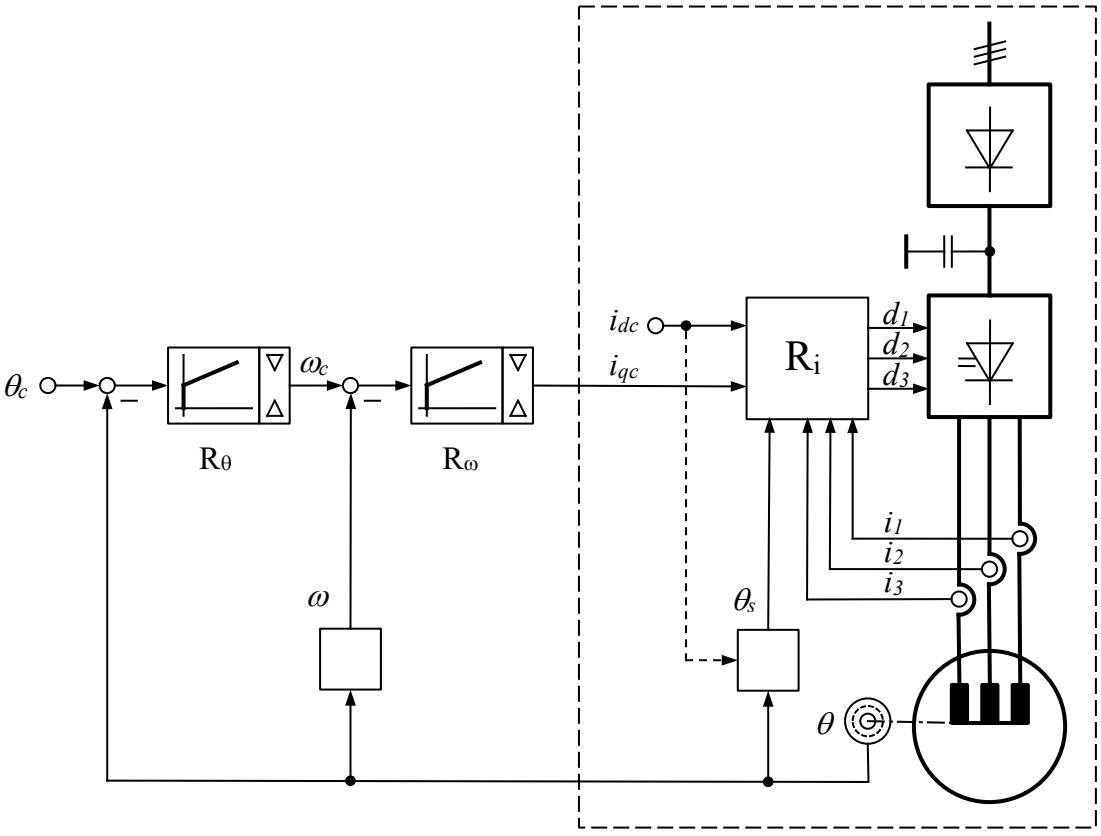


Figure 7 Schéma bloc de la régulation superposée de position, vitesse et courant

Régulateurs dans le référentiel tournant

La topologie la plus utilisée pour le control vectoriel du courant repose sur l'emploi de deux régulateur PI dans le référentiel tournant, comme le montre la *Figure 8*. Dans ce cas, les mesures des courants de phase sont d'abord converties en phaseur spatial dans le référentiel fixe, puis converties dans le référentiel tournant. Les deux composantes de la mesure du courant dans le référentiel tournant sont comparées avec les deux consignes. Les deux régulateurs PI fournissent les deux composantes du phaseur spatial de tension dans le référentiel tournant.

A partir de ce point, il y a plusieurs possibilités pour convertir le phaseur spatial de tension dans le référentiel tournant (u_d, u_q) en commande modulée des trois branches de l'onduleur d_1, d_2 et d_3.

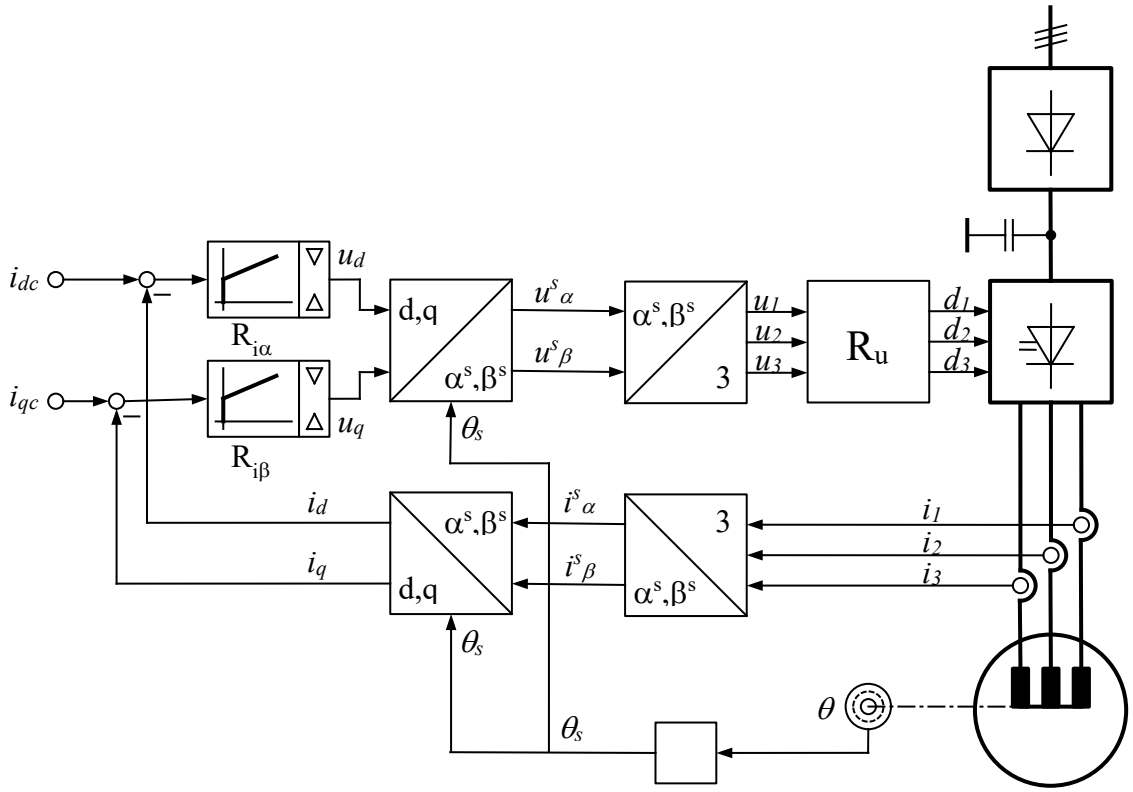


Figure 8 Schéma bloc de la régulation de courant dans le référentiel tournant

Trois signaux PWM (méthode de sous-oscillation)

Une première méthode, présentée à la *Figure 9*, consiste à convertir le phaseur spatial de la tension du référentiel tournant au référentiel fixe, puis de déterminer les trois tensions de phase à appliquer au moteur.

On utilise ensuite trois modulateurs (PWM) pour commander séparément les trois branches de l'onduleur.

En mode linéaire, sans sur-modulation, la plus grande tension de phase est donnée par :

$$\widehat{U}_{1max} = \frac{U_e}{2} \tag{15}$$

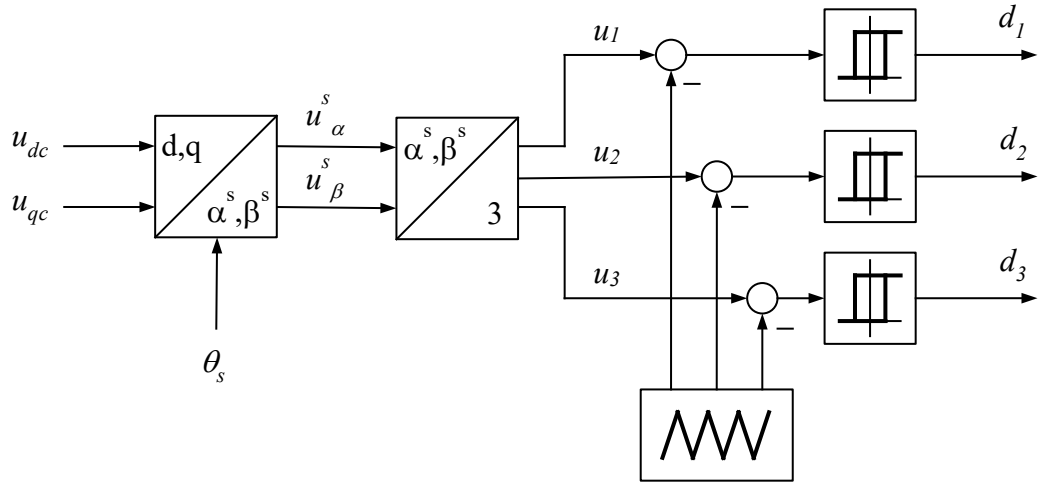


Figure 9 Trois signaux PWM (méthode de sous-oscillation)

Space Vector PWM (SV-PWM)

Une deuxième méthode, appelée space vector PWM (SV-PWM), est très répandue dans le domaine des entraînements réglés. Cette méthode permet de déterminer directement les états de commutation de l'onduleur à partir du phaseur spatial de tension dans le référentiel fixe, comme le montre la *Figure 10*.

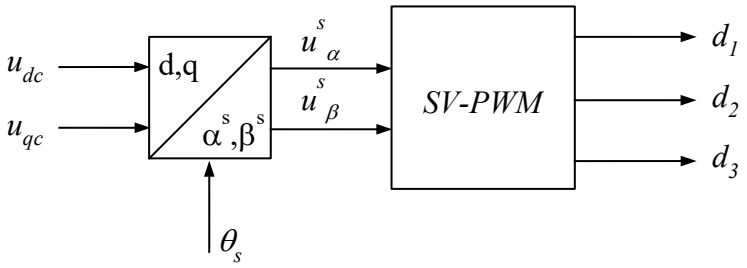


Figure 10 Schéma bloc de la méthode 'space vector PWM'

On peut montrer que les 8 états de commutation de l'onduleur permettent de générer 8 phaseurs spatiaux de tension (dans le référentiel fixe). Comme le montre la *Figure 11*, il y a 6 phaseurs spatiaux donnés par les sommets d'un hexagone et 2 phaseurs spatiaux nuls (états 0 et 7).

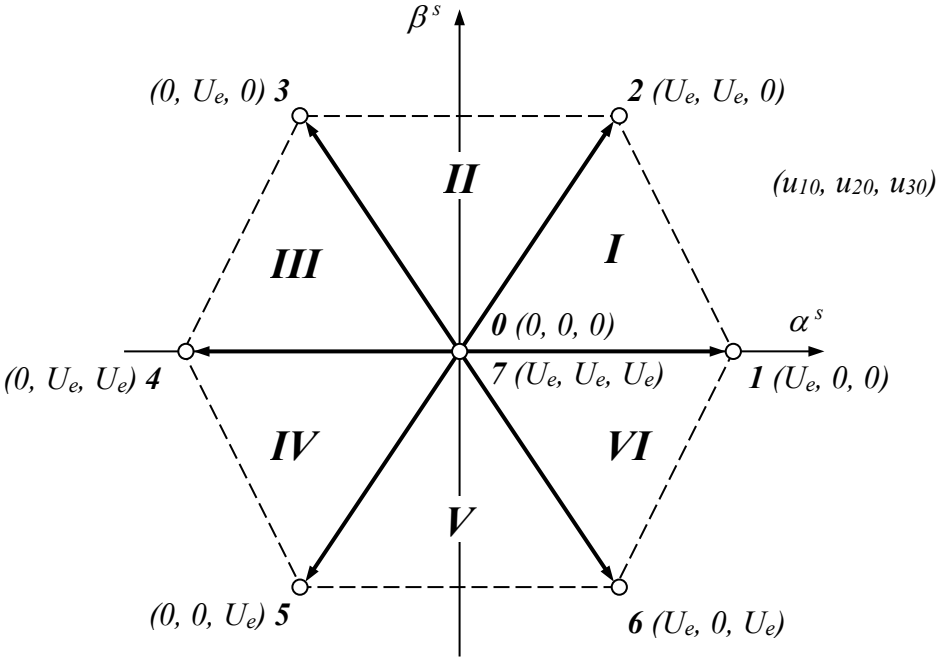
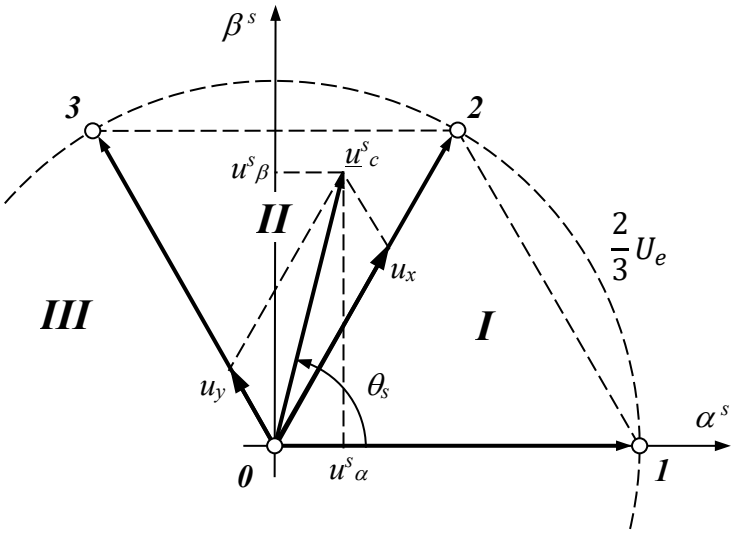


Figure 11 *Hexagone des états de commutation de l'onduleur.*

A l'aide des moyennes glissantes, un phaseur spatial u^s_c quelconque est généré en alternant 3 états de commutation (SV-PWM). Les rapports cycliques de chaque état de commutation sont déterminés par les projections u_x et u_y du phaseur spatial u^s_c sur les 2 phaseurs spatiaux les plus proches que l'onduleur peut fournir (*Figure 12*).



Ux - retard
Uy -avance

Figure 12 *Phaseur spatial à l'intérieur de l'hexagone des états de commutation.*

Les rapports cycliques se calculent avec :

$$D_x = \frac{u_x}{\frac{2U_e}{3}}, \quad D_y = \frac{u_y}{\frac{2U_e}{3}}, \quad D_0 = 1 - D_x - D_y \tag{16}$$

Afin d'éviter l'utilisation de fonctions trigonométriques dans la routine d'interruption, on peut déterminer les rapports cycliques ainsi que le secteur dans lequel se trouve le phaseur spatial à générer en utilisant la série de tests présentée à la *Figure 13*.

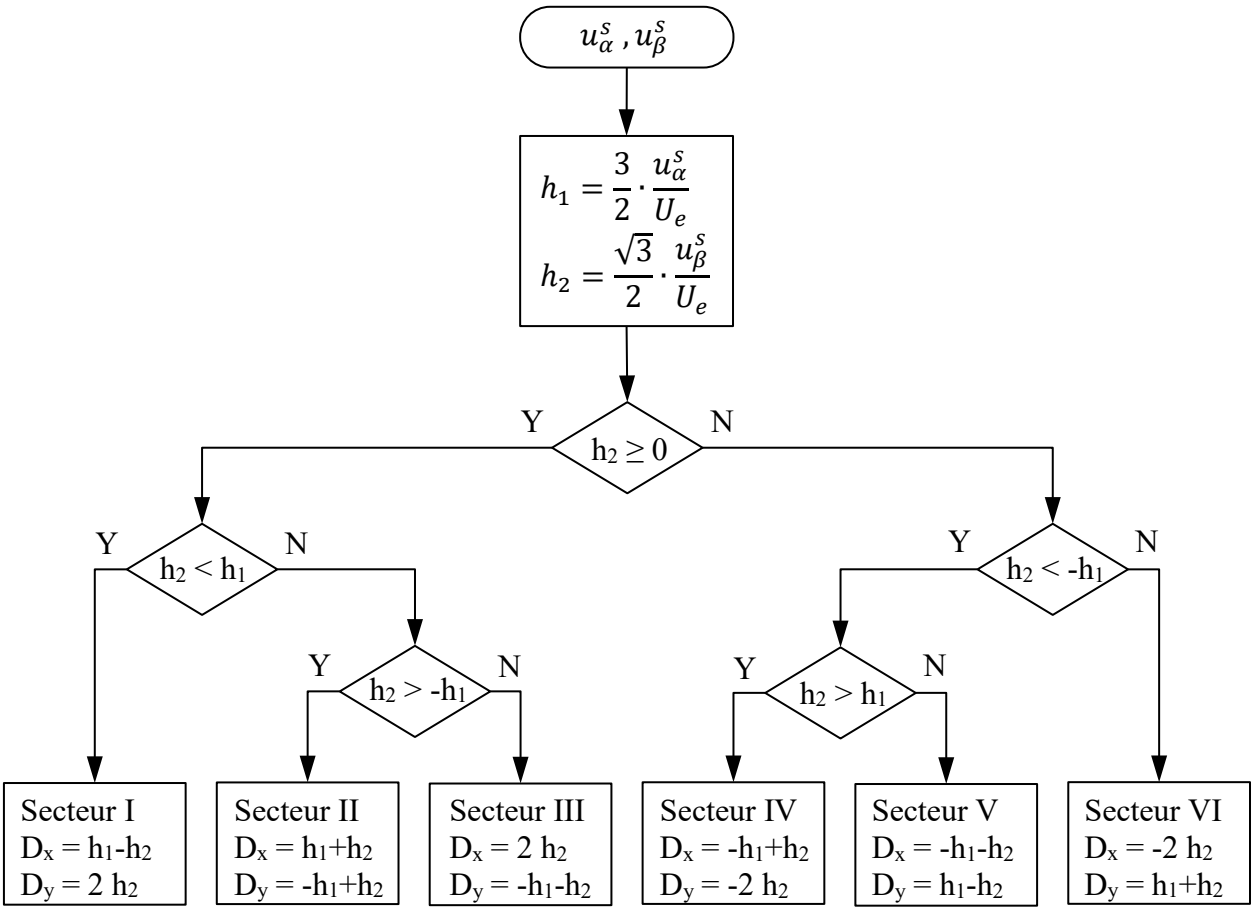


Figure 13 Algorithme pour déterminer le secteur et les rapports cycliques.

La *Figure 14* montre une séquence optimale des trois états de commutation (dans le secteur I). Dans une période, la séquence est parcourue deux fois.

Le tableau de la *Figure 15* indique les séquences et les niveaux de comparaison dans les 6 secteurs.

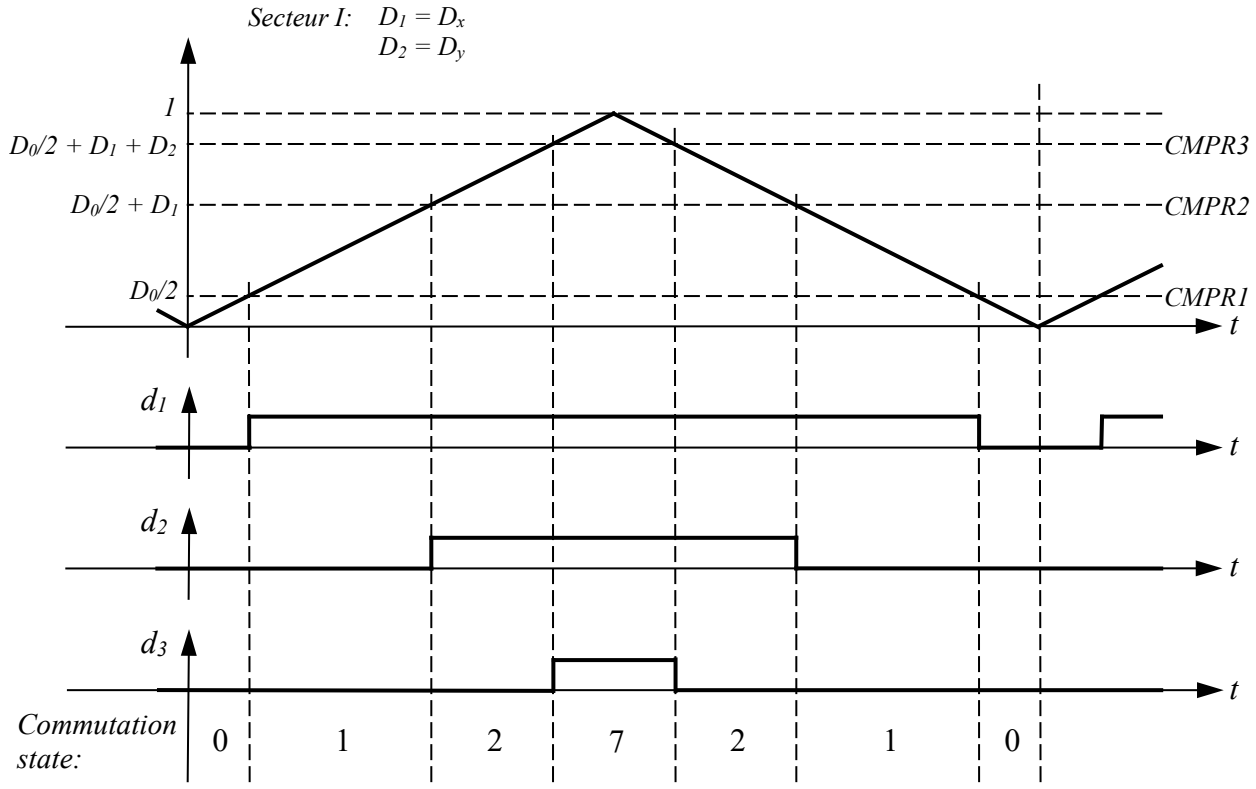


Figure 14 Séquence de commutation des 3 branches (secteur I).

Qu'une branche qui change par commutation (seulement pour état impaire)

0 3 2 7 … pour paire

Secteur	D_x	D_y	*CMPR1*		*CMPR2*		*CMPR3*	
I	D_1	D_2	$D_0/2$	d_1	$D_0/2+D_x$	d_2	$D_0/2+D_x+D_y$	d_3
II	D_2	D_3	$D_0/2$	d_2	$D_0/2+D_y$	d_1	$D_0/2+D_x+D_y$	d_3
III	D_3	D_4	$D_0/2$	d_2	$D_0/2+D_x$	d_3	$D_0/2+D_x+D_y$	d_1
IV	D_4	D_5	$D_0/2$	d_3	$D_0/2+D_y$	d_2	$D_0/2+D_x+D_y$	d_1
V	D_5	D_6	$D_0/2$	d_3	$D_0/2+D_x$	d_1	$D_0/2+D_x+D_y$	d_2
VI	D_6	D_1	$D_0/2$	d_1	$D_0/2+D_y$	d_3	$D_0/2+D_x+D_y$	d_2

Figure 15 Niveaux de comparaison et branche correspondante pour chaque secteur.

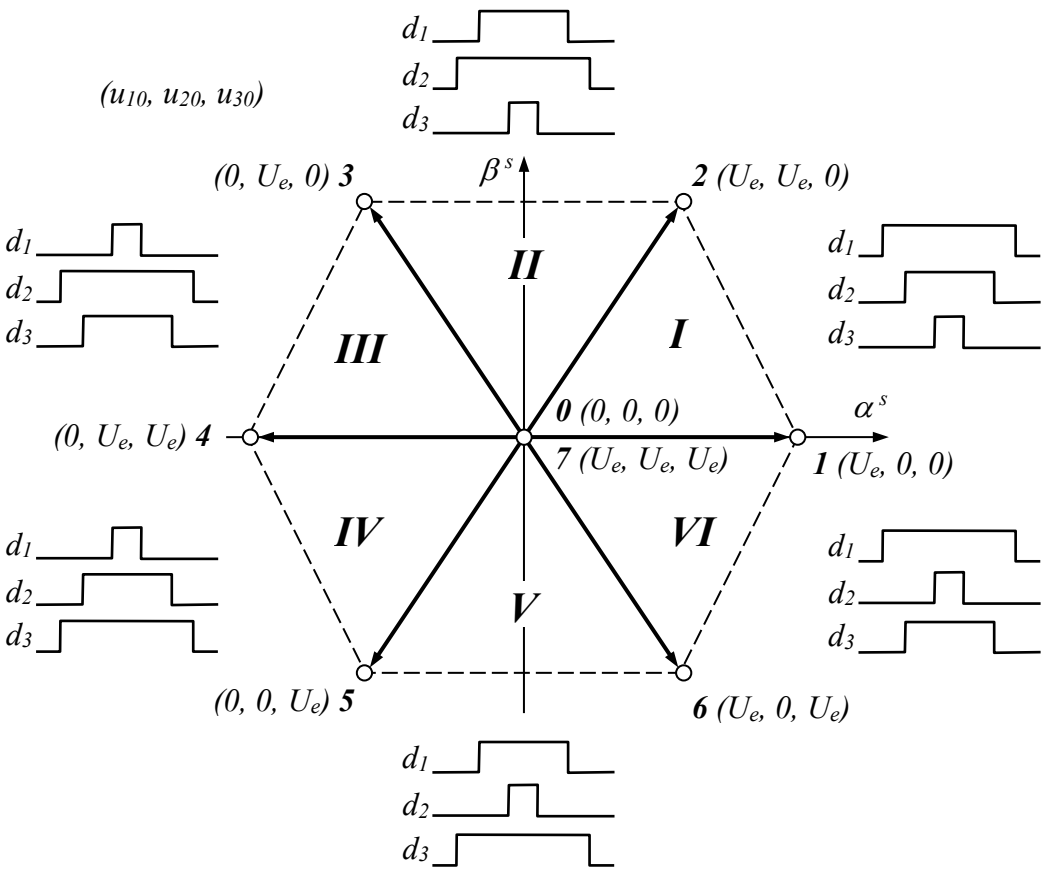


Figure 16 *Modulation des 3 branches de l'onduleur pour chaque secteur.*

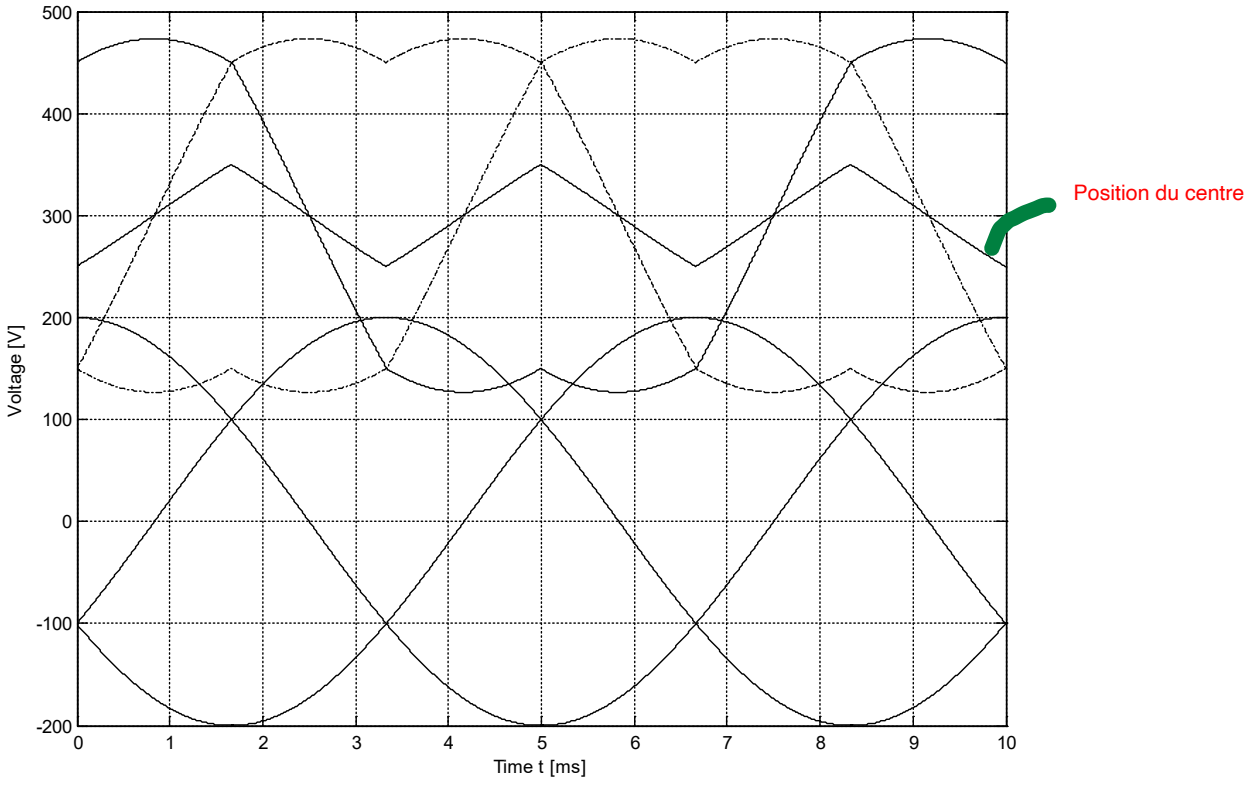


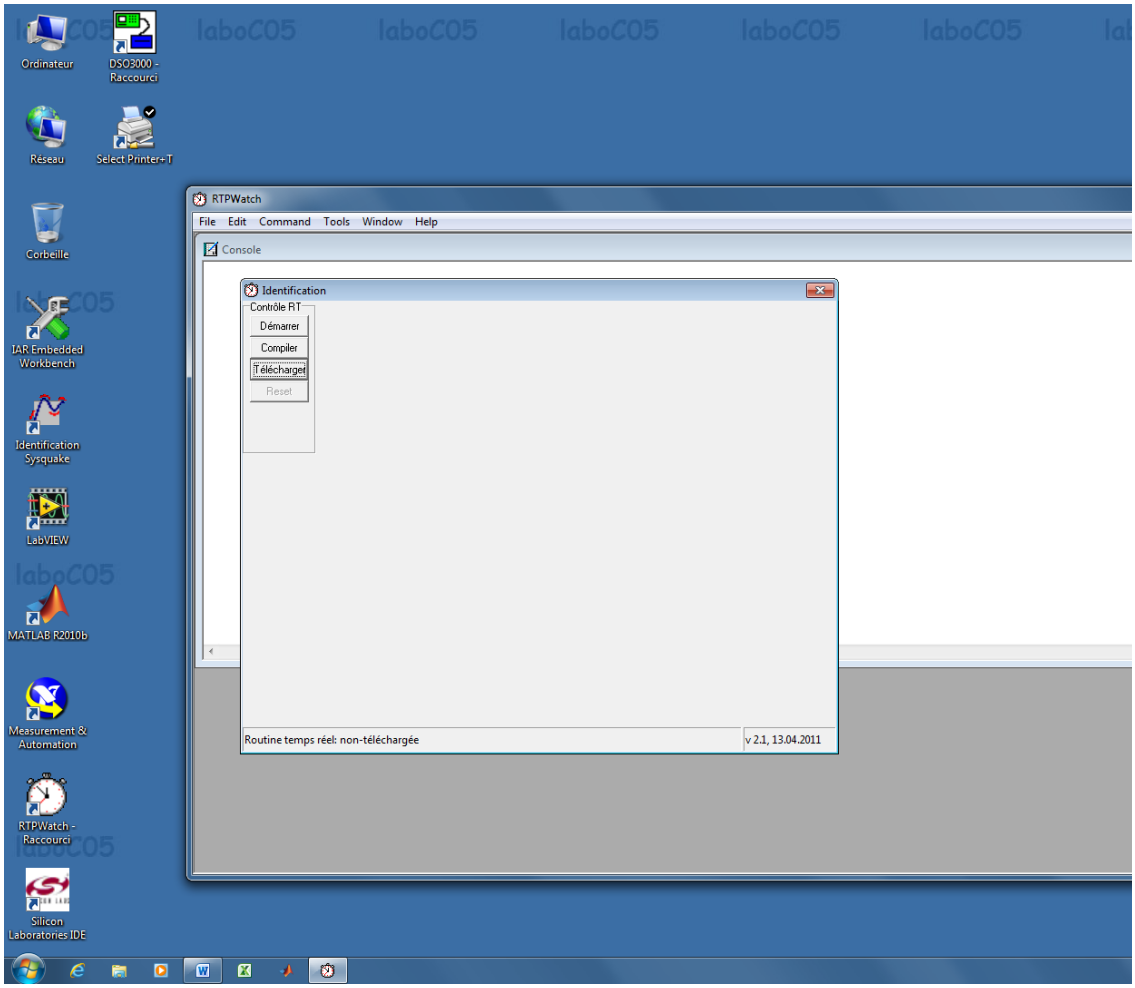
Figure 17 *Moyenne glissante des tensions de branche et de phase (SV-PWM).*

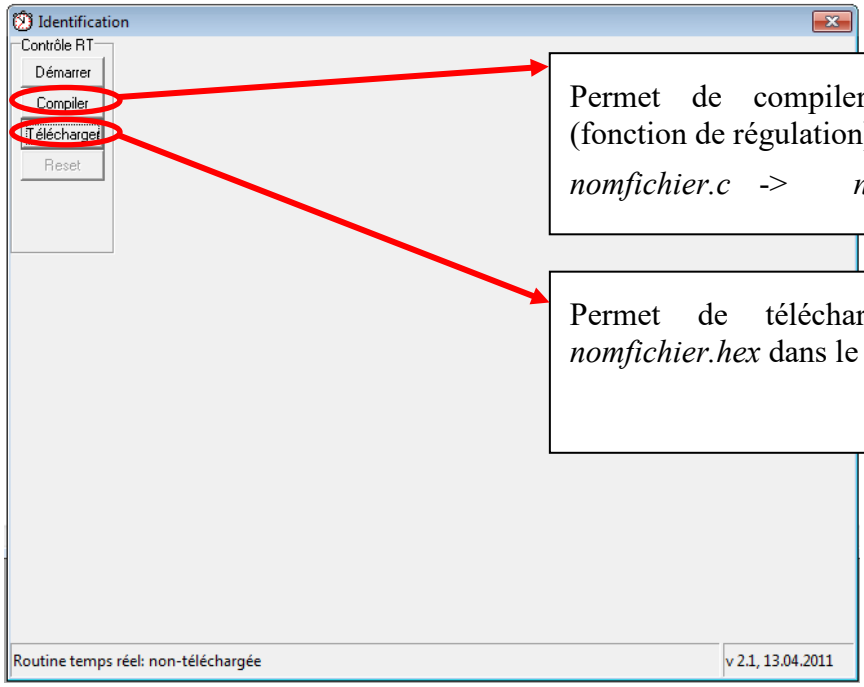
2. Manuel d'utilisation de l'onduleur triphasé

Utilisation du logiciel RTP Watch

Le logiciel RTP Watch permet de compiler le code C (routine d'interruption pour la régulation) et de le charger dans le DSP de la carte de commande de l'onduleur triphasé

- Lancer RTPWatch
- Tools / Labo MET2

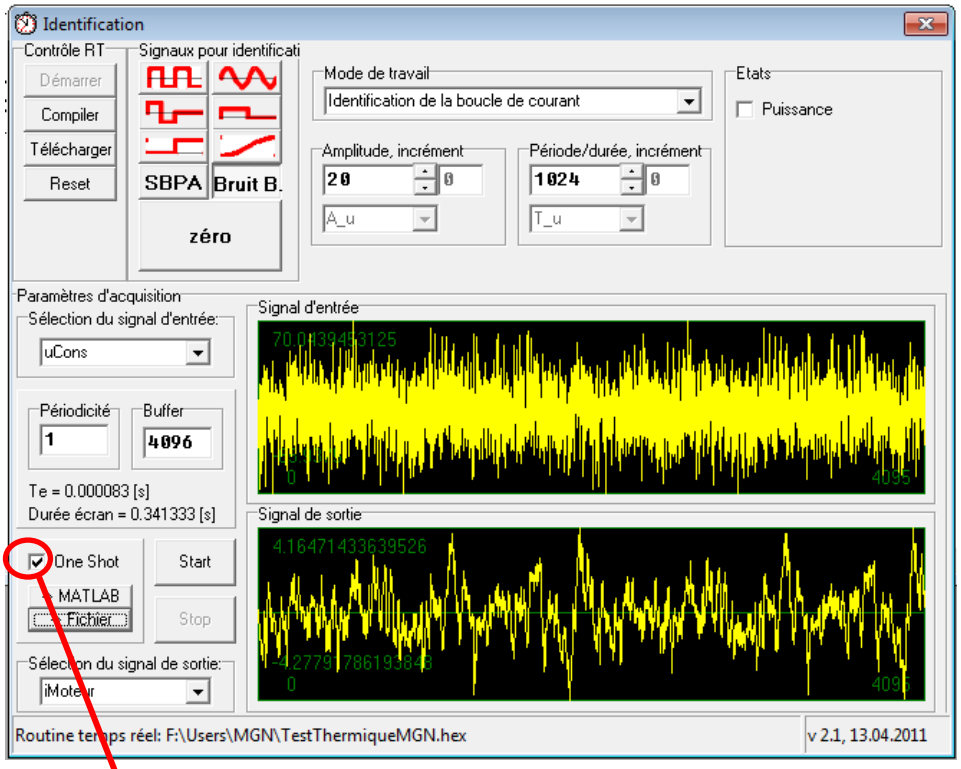




Permet de compiler le code C (fonction de régulation)
nomfichier.c -> *nomfichier.hex*

Permet de télécharger le code *nomfichier.hex* dans le DSP

Fenêtre de travail :



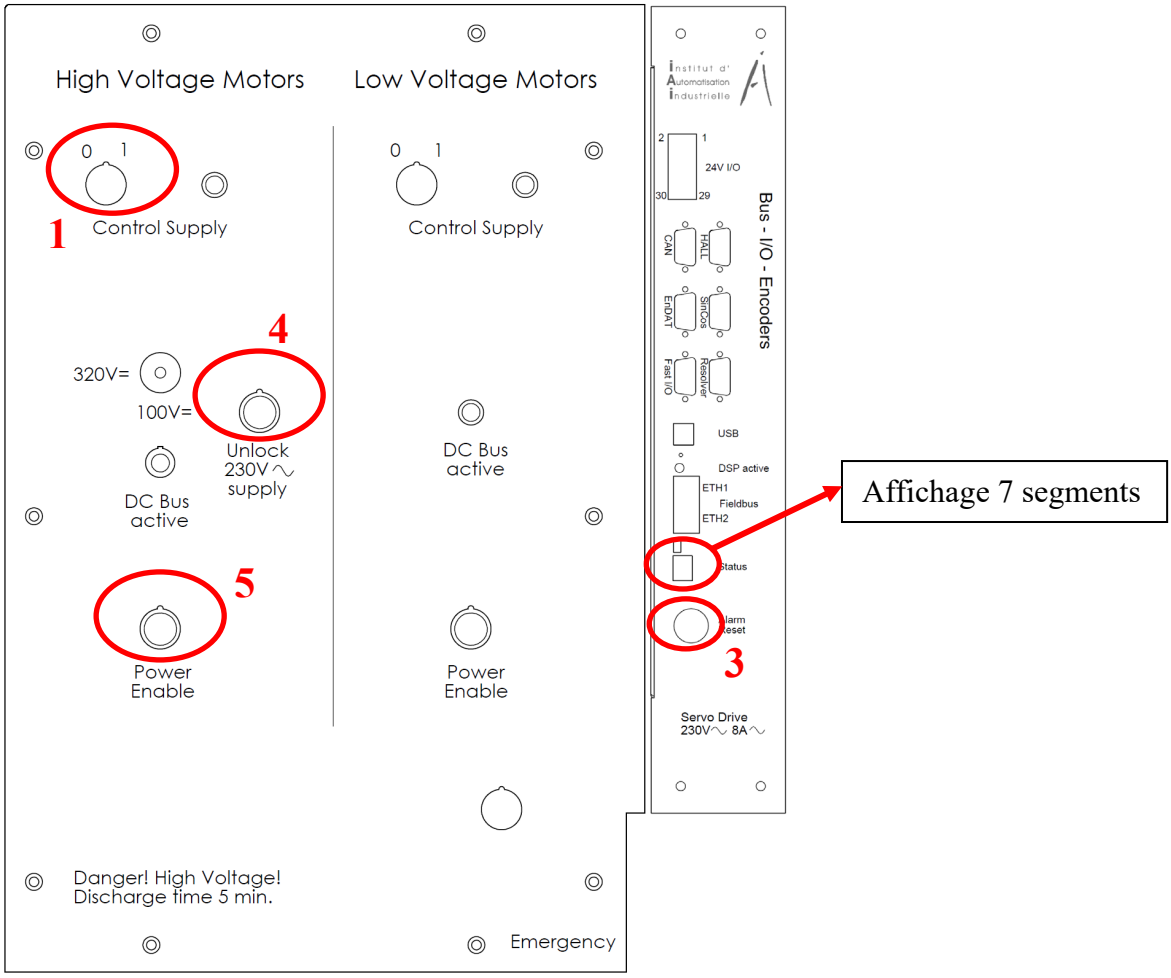
Paramètres de commande

Paramètres d'acquisition

Sélectionner 'One Shot' pour figer l'affichage et éviter les dépassements de capacité

Utilisation du l'onduleur triphasé

Pour ce laboratoire, il faut utiliser la partie 'High Voltage Motor' (partie gauche du pupitre de commande)



1. Tourner le commutateur '*Control Supply*' pour enclencher l'alimentation de commande du variateur.
   → L'affichage 7 segments indique 3 barres horizontales, il n'y a pas de code dans le DSP.
2. Charger la routine temps réel (*nomfichier.hex*) à l'aide du logiciel RTPWatch.
   → L'affichage 7 segments indique que le variateur est en défaut (*Alarm*).
3. Quittancer les défauts à l'aide du bouton '*Alarm Reset*'.
   → L'affichage 7 segments indique un petit 'o' si tout est OK.
4. Appuyer sur le bouton '*Unlock 230V supply*' pour activer l'alimentation de puissance.
5. Appuyer sur le bouton '*Power Enable*' pour activer la sortie de puissance du variateur.
   → L'affichage 7 segments indique un petit 'o' tournant.
6. Lorsque le mode de fonctionnement est correctement configuré dans RTPWatch, il faut encore libérer la sortie de puissance à l'aide de la case à cocher '*Puissance*'.
   Il est parfois nécessaire de répéter une fois cette opération.

3. Template pour le code C

MET_etudiant.c *(sur Cyberlearn)*

Déclarations

```
/**
 * \file            etudiant.c
 * \brief           Fonction de régulation écrite par les étudiants dont le .hex va être chargé
 * \author          Etudiant
 * \version 0
 * \date            12.03.2021
 */

//** Définitions
#define    RTPWATCH_MODE_NONE                   1       /*!< Aucun mode n'as été sélectionné dans RTPWatch */
#define    RTPWATCH_MODE_IDENT_BCL_INTERNE      2       /*!< Identification de la boucle de courant */
#define    RTPWATCH_MODE_REG_BCL_INTERNE        3       /*!< Regulation de la bouclde de courant */
#define    RTPWATCH_MODE_IDENT_BCL_EXT          4       /*!< Identification de la boucle position/vitesse */
#define    RTPWATCH_MODE_REG_BCL_EXT            5       /*!< Regulation de la boucle position/vitesse */
#define    RTPWATCH_MODE_REG_BCL_EXT_PID        6       /*!< Regulation de la boucle de position/vitesse par un PID */

//** Définitions de types
#define UINT32_WATCH(v)    (UpdateWatch(#v,(Uint32*)(&(v)),VAR_INT))
#define FLOAT_WATCH(v)     (UpdateWatch(#v,(Uint32*)(&(v)),VAR_FLOAT))

#define             Fe                  12000.0  // Fréquence d'échantillonnage
#define             Fe2                 6000.0   // Fréquence d'échantillonnage pour le resolver
#define             h                   83.3e-6  // Période d'échantillonnage

#define             Umax                120.0    // Tension maximale
#define             Imax                5.0      // Courant maximal
#define             Wmax                300.0    // Vitesse maximale
#define             NbrePairePole       3.0      // Nombre de paires de pôles

#define             PI                  3.1416
#define             PI_SUR_2            1.5708
#define             TWO_PI              6.2832
#define             Sqrt3               1.73205
#define             Div_sqrt3           0.57735

#define             INCREMENTAL         0        // Capteur relatif du moteur DC -> 12kHz
#define             RESOLVER            1        // Capteur absolu du moteur synchrone -> 6kHz


//** Macros

//** Variables Globales               /!\ toute initialisation est perdue! utiliser InitialisationEtudiant()
//                                    (idem pour les variables statiques)

long int nombreTours;

// Courant
float Kp_i, Gi_i;
// vitesse
float Kp_w, Gi_w;
// position
float Kp_theta;

float thetaMoteur_old, thetaElectrique, thetaMecanique;
float sinThetaElectrique, cosThetaElectrique;

float udc, uqc;
float idm, iqm;
```

Fonction d'initialisation

Cette fonction n'est appelée qu'une seule fois avant de générer les interruptions.

```
/**
 *          \fn         void InitialisationEtudiant()
 *          \brief      Cette fonction est appelée après le chargement du .hex
 */

void InitialisationEtudiant()
{
        FLOAT_WATCH(Kp_i);
        FLOAT_WATCH(Gi_i);
        FLOAT_WATCH(Kp_w);
        FLOAT_WATCH(Gi_w);
        FLOAT_WATCH(Kp_theta);
        FLOAT_WATCH(thetaMecanique);
        FLOAT_WATCH(thetaElectrique);
        FLOAT_WATCH(idm);
        FLOAT_WATCH(iqm);

        nombreTours = 0;
        thetaMoteur_old = 0.0;
        // courant
        Kp_i = 0.0;
        Gi_i = 0.0;
        // vitesse
        Kp_w = 0.0;
        Gi_w = 0.0;
        // position
        Kp_theta = 0.0;
        // Calcul de la vitesse avec dérivée + filtre
        thetaMecanique = 0.0;
        thetaElectrique = 0.0;
        sinThetaElectrique = 0.0;
        cosThetaElectrique = 0.0;
}
```

Routine d'interruption

Cette fonction est appelée à chaque interruption à la fréquence de 12 kHz.

```
/**
 *		\fn
 *		\brief		Cette fonction est appelée pour régler le moteur
 *
 *		\param		flags			: regroupe différentes informations:
 *								- flags.bit.Mode		mode de travail de RTPWatch
 *								- flags.bit.FirstMesure		indique la première mesure après un reset
 *		\param		consigne		: consigne envoyée par RTPWatch
 *		\param		courantBranche1		: courant du moteur (branche 1)
 *		\param		courantBranche2		: courant du moteur (branche 2)
 *		\param		thetaMoteur		: position du moteur
 *		\param		busDC			: tension du bus DC
 *
 *		\return		tension à appliquer
 */
RapportsCycliques Regulation(FlagsRegulation* flags, float consigne,float courantBranche1, float courantBranche2, float thetaMoteur, float busDC)
{
	RapportsCycliques rapportsCycliquesVariateur;
	float divBusDC;
	float delta_theta;
	float h1, h2, Dx, Dy, CMPR1, CMPR2, CMPR3;
	float usa, usb;

	/*------------------------------------------------------------------------------ **
	**			Traitement de la mesure de position			**
	**------------------------------------------------------------------------------*/
	// Une nouvelle mesure de position est disponible une fois sur deux dans le cas du resolver
	if(flags->bit.FirstMesure)				// set la première mesure comme condition initiale
		thetaMoteur_old = thetaMoteur;
	if(flags->bit.NewPosition && flags->bit.ChoixCapteur == RESOLVER)
	{
		delta_theta = thetaMoteur - thetaMoteur_old;
		if(delta_theta < -PI)
		{
			nombreTours++;
		}
		if(delta_theta > PI)
		{
			nombreTours--;
		}
		thetaMecanique = nombreTours * TWO_PI + thetaMoteur;
		thetaElectrique = thetaMoteur * NbrePairePole;

		thetaMoteur_old = thetaMoteur;
	}



	/*------------------------------------------------------------------------------ **
	**			Transformation de Park			**
	**			-> idm, iqm				**
	**------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------- **
**                      Régulation (Position, Vitesse, Courant)                   **
**                      -> udc, uqc                                               **
**--------------------------------------------------------------------------------*/

switch(flags->bit.Mode)
{
        case RTPWATCH_MODE_IDENT_BCL_INTERNE:
        {
        }
        break;

        case RTPWATCH_MODE_REG_BCL_INTERNE:
        {
        }
        break;

        case RTPWATCH_MODE_IDENT_BCL_EXT:
        {
        }
        break;

        case RTPWATCH_MODE_REG_BCL_EXT:
        {
        }
        break;

        default:
        {
        udc = 0.0;
        uqc = 0.0;
        }
}


/*-------------------------------------------------------------------------------- **
**                      Transformation de Park inverse                            **
**                      -> usa, usb                                               **
**--------------------------------------------------------------------------------*/

usa = 0,0;
usb = 0.0;



/*-------------------------------------------------------------------------------- **
**                      Calculs des rapports cycliques                            **
**                      -> raportsCycliquesVariateur.branche1                     **
**                      -> raportsCycliquesVariateur.branche2                     **
**                      -> raportsCycliquesVariateur.branche3                     **
**--------------------------------------------------------------------------------*/
divBusDC = 1.0/busDC;
h1 = 1.5*usa*divBusDC;
h2 = 0.5*Sqrt3*usb*divBusDC;
```

```
	if (h2 >= 0.0)
	{
		if (h2 < h1)                                    // SECTEUR 1
		{
			Dx = h1-h2;
			Dy = 2.0*h2;
			CMPR1 = 0.5*(1.0-Dx-Dy);
			CMPR2 = CMPR1+Dx;
			CMPR3 = 1.0-CMPR1;
			rapportsCycliquesVariateur.branche1 = 1.0-CMPR1;
			rapportsCycliquesVariateur.branche2 = 1.0-CMPR2;
			rapportsCycliquesVariateur.branche3 = 1.0-CMPR3;
		}
		else
		{
			if (h2 > -h1)                               // SECTEUR 2
			{
				Dx = h1+h2;
				Dy = -h1+h2;
				CMPR1 = 0.5*(1.0-Dx-Dy);
				CMPR2 = CMPR1+Dy;
				CMPR3 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche1 = 1.0-CMPR2;
				rapportsCycliquesVariateur.branche2 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche3 = 1.0-CMPR3;
			}
			else                                        // SECTEUR 3
			{
				Dx = 2.0*h2;
				Dy = -h1-h2;
				CMPR1 = 0.5*(1.0-Dx-Dy);
				CMPR2 = CMPR1+Dx;
				CMPR3 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche1 = 1.0-CMPR3;
				rapportsCycliquesVariateur.branche2 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche3 = 1.0-CMPR2;
			}
		}
	}
	else
	{
		if (h2 < -h1)
		{
			if (h2 > h1)                                // SECTEUR 4
			{
				Dx = -h1+h2;
				Dy = -2.0*h2;
				CMPR1 = 0.5*(1.0-Dx-Dy);
				CMPR2 = CMPR1+Dy;
				CMPR3 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche1 = 1.0-CMPR3;
				rapportsCycliquesVariateur.branche2 = 1.0-CMPR2;
				rapportsCycliquesVariateur.branche3 = 1.0-CMPR1;
			}
			else                                        // SECTEUR 5
			{
				Dx = -h1-h2;
				Dy = h1-h2;
				CMPR1 = 0.5*(1.0-Dx-Dy);
				CMPR2 = CMPR1+Dx;
				CMPR3 = 1.0-CMPR1;
				rapportsCycliquesVariateur.branche1 = 1.0-CMPR2;
				rapportsCycliquesVariateur.branche2 = 1.0-CMPR3;
				rapportsCycliquesVariateur.branche3 = 1.0-CMPR1;
			}
		}
		else                                            // SECTEUR 6
		{
			Dx = -2.0*h2;
			Dy = h1+h2;
			CMPR1 = 0.5*(1.0-Dx-Dy);
			CMPR2 = CMPR1+Dy;
			CMPR3 = 1.0-CMPR1;
			rapportsCycliquesVariateur.branche1 = 1.0-CMPR1;
			rapportsCycliquesVariateur.branche2 = 1.0-CMPR3;
			rapportsCycliquesVariateur.branche3 = 1.0-CMPR2;
		}
	}
	// Actualisation depuis RTPWatch
	flags->bit.ChoixCapteur = RESOLVER;

	return rapportsCycliquesVariateur;
}

//** end of file **//
```