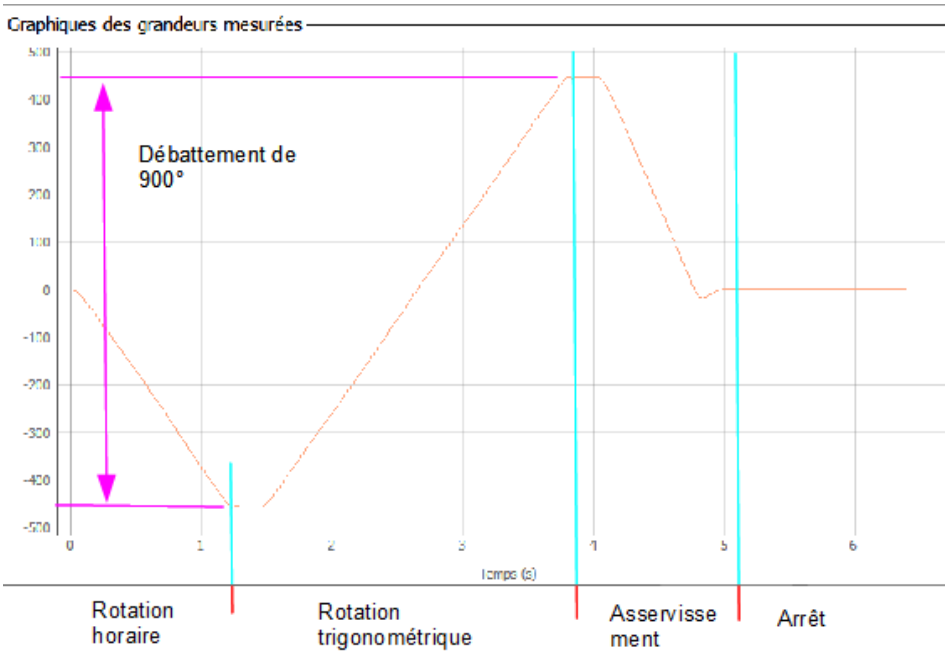
Volant à retour de force de simulateur

1) On détermine l'information de position en comptant le nombre de passage de 0 à 1 ou de 1 à 0 sur les signaux des deux voies (fronts montant et descendant) ce qui permet d'obtenir 120 informations par tour du moteur. Pour connaître le sens, il faut déterminer quel signal est en avance sur l'autre.

2) Le plus petit angle mesuré au niveau du moteur est de 360/120 = 3°. Sachant que le rapport de réduction est de 11/180, le plus petit angle mesuré au niveau du volant est de 3*11/180=180°. Cette valeur est très faible pour une application telle qu'un jeu vidéo.

On comprend l'intérêt d'utiliser le codeur sur le moteur plutôt que sur le volant pour gagner en précision au niveau du volant. Le gain est de 0,18 °=tops.

3)



4) On mesure un débattement de 900° ce qui correspond au cahier des charges. On pourrait utiliser des capteurs de fin de course pour détecter la butée.

5) Dans Rotation moteur +, on fait tourner le moteur en mettant *do/ pwm 400*
Dans Détection butée droite, on lit l'information du capteur : do/ Lire nbtops La transition permettant de sortir de l'état orthogonal est déclenchée lorsque la variable nbtops est égale à *nbtops_prec* (le volant ne bouge plus). Ainsi T3 correspond à *when nbtops=nbtops_prec.*
Cependant pour que cet événement puisse avoir lieu, il faut qu'au départ la variable *nbtops_prec* soit mise à 0. c'est pourquoi T1 correspond à */nbtops_prec=0*. Pour terminer, après chaque lecture du nombre de tops, après 10 ms, on met à jour la variable *nbtops_prec*, T2 vaut alors :
after(10ms)[nbtops !=nbtops_prec]/nbtops_prec=nbtops

6)

```
while(etat!="arret final"):
    if etat=="Rotation horaire":
        nb_tops_prec=0
        nb_tops=0
        sous_etat="test rotation"
        while(etat!="Rotation trigo"):
            pwm(400)
            if sous_etat=="test rotation":
                while(sous_etat!="test arret"):
```

```
                    sous_etat=test_rotation(sous_etat)
            elif sous_etat=="test arret":
                tic=time.time()
                etape="entry"
                while(sous_etat!="test rotation" and etape!="exit"):
                    sous_etat,etape=test_arret(sous_etat,etape,tic)
                if etape=="exit":
                    etat="Rotation trigo"
    elif etat=="Rotation trigo":
        nb_tops_max=nb_tops
        sous_etat="test rotation"
        while(etat!="Retour zero"):
            pwm(-400)
            if sous_etat=="test rotation":
                while(sous_etat!="test arret"):
                    sous_etat=test_rotation(sous_etat)
            elif sous_etat=="test arret":
                tic=time.time()
                etape="entry"
                while(sous_etat!="test rotation" and etape!="exit"):
                    sous_etat,etape=test_arret(sous_etat,etape,tic)
                if etape=="exit":
                    etat="Retour zero"
    elif etat=="Retour zero":
        nb_tops_min=nb_tops
        consigne=(nb_tops_min+nb_tops_max)/2
        while(etat!="arret final"):
            etat=asservissement(consigne)
```

7) On propose les deux fonctions suivantes :

```
def test_rotation(sous_etat):
    nb_tops=lire_tops()
    if nb_tops==nb_tops_prec:
        sous_etat="test arret"
    else:
        sous_etat="test rotation"
    nb_tops_prec=nb_tops
  return sous_etat
```

```
def test_arret(sous_etat,etape):
    nb_tops=lire_tops()
    if nb_tops!=nb_tops_prec:
        sous_etat="test rotation"
    else:
        sous_etat="test arret"
        if time.time()-tic>0.5:
            etape="exit"
  return sous_etat,etape
```

8) Si on bloque à la main le volant avant qu'il n'atteigne la butée, le changement d'état va s'opérer et le volant sera mal centré. On peut utiliser l'information sur le courant du moteur pour savoir si la butée est atteinte ou non compte-tenu du profil de courant qui sera différent.

9) On doit d'abord stocker la valeur de nbtops pour la rotation horaire *: after(0.5s)/nbtopsmin=nbtops*, puis stocker le nbtops maximal : *after(0.5s)/nbtopsmax=nbtops* et enfin calculer la consigne avant de lancer l'asservissement : *entry/consigne=(nbtopsmax+nbtopsmin)/2*

10) La fonction asservissement est appelée à chaque instant.

```
def asservissement(consigne):
  nbtops=lire_nbtops()
  val=(consigne-nbtops)*3
  if val<-1024:
    val=-1024
  elif val>1023:
    val=1023
 pwm(val)
```

Jeu sur Wiiboard

1) Dans l'état Gagné, on doit *Afficher un message 1* et dans l'état Perdu *Afficher un message 2*. Pour passer de l'état Gagné à l'état Jeu ou de l'état Perdu à l'état Jeu, il faut appuyer sur R pour recommencer d'où T1=T4 : *Appui sur R*. Pour quitter le jeu, il faut appuyer sur Q, ainsi T5=T6 : *Appui sur Q*. Enfin la variable gain permet de savoir si la partie est gagnée ou perdue, d'où T3 : *when gain=1* et T2 : *after(20s) [gain=0]*.

2) L'action associée à Entry/ est gain=0. Dans le comportement do/, on doit calculer aléatoirement la position du disque cible : Générer aléatoirement (x0,y0,R).

3) Dans les états Hors disque et Dans disque, on doit Obtenir la position (x,y) et rafraichir l'écran.

4) Les transitions *hors_disque* et *dans_disque* permettent de tester si le point x,y est dans le disque ou non. Il suffit de tester la condition suivante $[(x-x0)^2+(y-y0)^2 <= R^2]$ pour la transition *dans_disque*.

La transition jT1 correspond à l'événement contraire à *dans_disque* soit *hors_disque*, et inversement pour jT2. Pour jT3, on sort de cet état composite si on est resté 3 secondes dans l'état Dans disque et on met la variable gain à 1, soit jT3 : after(3s) / gain=1

5) Il faut d'abord acquérir et calculer les positions x et y normalisés avec la fonction calc_xy)() puis faire un décalage pour passer de -1,1 à 0,800 ou 0,600.

```
x,y= calc_xy ()
x =400* x +400
y=y *300+300
```

6) Voici une proposition de programmation testée sur le système

```
def jeu_wiiboard():
    gain=0
    R=randint(10,50)
    x0=randint(0,800)
    y0=randint(0,600)
    affiche_disque(x0,y0,R)
    tic=time.time()
    tini=0
    dans_disque=0
    while(time.time()-tic<20 and gain==0):
        x,y=calc_xy()
        x=400*x+400
        y=y*300+300
        affiche_point(x,y)
        if dans_disque==0:
            if ((x-x0)**2+(y-y0)**2<=R**2):
                dans_disque=1
                tini=time.time()
        if dans_disque==1:
            if time.time()-tini>=3:
                gain=1
            if (x-x0)**2+(y-y0)**2>R**2:
                dans_disque=0
```