```
struct complex_number
{
      float real_component
      float imaginary_component
}
```

- `void` **`avoid_obstacle`**()

réutiliser la fonction donnée dans le main.c d'exemple avec une condition (while) qui fait que l'on sort de cette fonction quand on n'est plus TOO_CLOSED

- `float` **`detect_sound_phase_shift`** `()`

La fonction calcule la phase pour les 4 micros de la FFT à la fréquence de la cible (à partir de `complex_number micRight_fft_data, complex_number micLeft_fft_data, complex_number micFront_fft_data, complex_number micBack_fft_data`, car comme ce sont des structures déclarées comme "static", elles sont directement accessibles). Elle renvoie la différence de phase entre les micros de droite et de gauche: `phase_difference`.

```
//position of the microphones in the buffer given to
#define MIC_LEFT 1
#define MIC_RIGHT 0the customFullbufferCb
#define MIC_FRONT 3
#define MIC_BACK 2
```

- `float` **`calculate_phase`**`(complex_number fft_at_frequency)`

Cette fonction calcule la phase à partir des parties réelle et imaginaire de la transformée de Fourier pour une fréquence donnée en sachant que:

phase = arctan($\frac{Im}{Re}$) si Re>0

phase = arctan($\frac{Im}{Re}$) + π sinon

- `void` **`process_audio_data`**`(int16_t *data, uint16_t num_samples)`
  - `void do_fft_optimized (uint16_t size, float* complex_buffer)`

Elle récupère les données du micro et elle les accumule (comme pour ProcessingAudioData). Lorsqu'elle en a 1024, elle fait un FFT optimisée et détecter la fréquence pour MicFront. Si l'indice du tableau correspond à celui d'une fréquence aux alentours de 440 Hz. (The relation between the frequency and the position in the buffer is *frequency = position∗15,625* if the position isn't greater than FFT_SIZE/2.) Si c'est le cas, la fonction active `frequency_on` =1 stocke les valeurs complexes (en static) pour chaque micro de la fréquence de la cible avec la structure complex_number.

- `int` **`get_calibrated_prox`**`(unsigned int sensor_number)`

On appelle cette fonction pour récupérer la valeur des capteurs IR.

- `void` **`displacement_start`**

- `void` **`go_to_sound`**`(float phase_difference)`

Cette fonction contrôle les deux moteurs à l'aide d'un régulateur. Elle soustrait la différence de phase à l'un des moteurs et l'additionne à l'autre. Par conséquent, si le robot est dans la bonne direction, la phase_difference est nulle ⇒ les 2 moteurs vont à la même vitesse ⇒ le robot va tout droit. On additionne / soustrait la différence de phase à une vitesse de telle sorte que v + delta_phase < v_max.

/!\ on ne sort pas de `go_to_sound` tant que phase_difference n'est pas égale à 0
Quand on sort de cette fonction et même de la thread displacement, les moteurs continuent de tourner car leur vitesse a été "set" par la fonction go_to_sound pour que le robot continue tout droit pendant que l'on calcule la FFT.

- `int8_t` **`check_proximity`**`()`

renvoie 1 si l'un des capteurs de proximité sont plus grand que TOO_CLOSED et 0 sinon.

Liens utiles

http://chibios.org/dokuwiki/doku.php?id=chibios:book:kernel_semaphores (fonctions à appeler et architecture à implémenter pour les sémaphores)

http://people.cs.aau.dk/~bnielsen/TOV07/material/fsm-intro.pdf

Liste de questions à poser aux assistants

- est-ce que l'on est obligé d'utiliser un float pour l'angle?
- comment fonctionne le code pour déterminer la fréquence d'échantillonnage de l'IMU (on n'arrive pas à comprendre comment ces opérations bit à bit donnent 10 comme résultat)
    - opération & logique bit à bit avec 0xff (= 11111111 en binaire) permet de faire sample rate initialement envoyé module 256
    - ensuite info du sample rate codée sur 8 bits est décalée de 8 bits pour correspondre aux 8 bits de poids forts d'une variable 16 bits
    - dans la fonction de configuration de l'imu, le programme récupère la variable int config qui contient plusieurs infos (espèce de structure sans être déclarée) et refait un décalage pour extraire les 8 bits de poids fort et donc le sample rate
    - la fréquence initiale étant de 1kHz (d'après datasheet), on obtient 10 Hz si on divise par un SAMPLE_RATE_DIV = 10

Justifications à préciser dans le rapport

- Essayer de présenter les fonctions (définitions / argument(s) / variable(s) renvoyée(s) comme dans les librairies.

- Dans notre cas, comme la pulsation de notre signal est petite, nous n'avons pas de unwrapping. -> faire un rapide calcul pour approximer la distance à laquelle devrait se trouver la source sonore du robot pour que ce phénomène est lieu

- On utilise des nombres float pour faire la FFT et les calculs de phase à partir des valeurs complexes car nous avons vraiment besoin d'être très précis mathématiquement. Pour l'instant, la différence de phases entr les 2 micros est aussi stockée sous la forme d'un float.

grâce au bruit, on a seulement de deux micros (Left, right) pour déterminer la direction de la source sonore

pas besoin d'utiliser l'imu, avec le pid, on veut juste un décalage de phase nulle entre ces deux micros

goal = 0
feedback = différence de phase entre les deux micros

mettre threshold pour essayer de stabiliser le pid
cas particulier: Quand on est exactement dans la direction de la source, on ne peut pas savoir si la source sonore est devant ou derrière le robot. Dans ce cas, le bruit nous aide car en pratique, cela nous permet de discriminer le devant de l'arrière.

une seule consigne et mettre un signe pour que cela corresponde pour chaque moteur

Ã déterminer par la pratique

- constante seuil TOO_CLOSED
- est-ce que l'on peut suivre la cible avec seulement un P ?

Archives

```
struct closest_mics
{
      uint8_t micro1              //closest microphone
      float phase_micro1
      uint8_t micro 2             //2nd closest microphone
      float phase_micro2
};
```

- `float` **`measure_angle`**()
    - `float get_gyro_rate(uint8_t axis)`

La fonction `measure_angle` appelle `get_gyro_rate` pour l'axe z (2) et récupère ainsi une vitesse en rad/s. Elle intègre cette vitesse (en la multipliant par un Δt correspondant à la fréquence d'échantillonnage divisée par un multiple (MPU9250_SAMPLE_RATE_DIV)) et la convertit en degré avant de la renvoyer.

fréquence d'échantillonnage = 1kHz
dans ligne 146 de imu.c, on a MPU9250_SAMPLE_RATE_DIV = 100
⇒ d'où $\Delta t = 1/(10\ Hz)$
Inspiration algorithme: http://www.pieter-jan.com/node/7

- **`go_straight`**

dire aux moteurs de tourner pour aller tout droit

- `float` **`get_angle`**`()`

Cette fonction renvoie la valeur de `angle` correspondant à une valeur statique updatée par la fonction `find_sound_direction`.

- **`orientate_robot`** `(float delta_phase)`
    - `int16_t pi_regulator(float distance, float goal)`

On donne comme entrée à la fonction l'angle entre la position initiale du micro de devant du robot et la direction de la source. La fonction `orientate_robot` appelle ensuite `pi_regulator(measure_angle(),find_sound_direction()`.

- `void` **`find_sound_direction`**

La fonction appelle `detect_sound_phase_shift` et pondère les phases des deux micros les plus proches pour en déduire un angle.
-> Tester différence de phase entre les deux micros les plus proches quand la source est exactement en face d'un micro. Puis faire une règle de 3 avec ce résultat en sachant que si la source sonore est exactement au milieu entre 2 micros, on a une différence de 0 au niveau de la phase.
Elle actualise la valeur de l'angle correspondant à un float en static.