Travaux d'Initiative Personnelle Encadré – Bibliographie et Expérience

Louis-Victor LADAGNOUS et Léo SAMUEL

Sujet : *Lévitation acoustique*

- Comment peut-on minimiser les pertes liées aux contenants ?
- Peut-on réduire les rejets de polluants lors d'une réaction avec la lévitation acoustique ?
- Comment peut-on faciliter la manipulation de produit dangereux au moyen de la lévitation acoustique ?

I - Bibliographie :

1 • TinyLev : A mutli-emitter single-axis acoustic levitator

Asier MARZO, Adrian BARNES et Bruce W. DRINKWATER

https://aip.scitation.org/doi/10.1063/1.4989995

Cette thèse réalisée par des chercheurs de l'université de Bristol nous a permis d'avoir une première approche sur le sujet. Grâce à ces travaux, nous avons maintenant une vision globale du phénomène. Nous nous servirons plus tard de cette thèse pour notre expérience.

2 • Ultraino : An open phased-array system for narrowband Airborne ultrasound transmission

Asier MARZO, Adrian BARNES et Bruce W. DRINKWATER

https://ieeexplore.ieee.org/document/8094247

Ce document ne nous est pas entièrement utile, mais il aborde le sujet de la visualisation des ondes ultra-sonores. Cela nous sera utile notamment pour la réalisation d'un programme informatique pour calculer et visualiser l'intensité et la présence des ondes ultra-sonores dans notre système afin de valider notre expérience.

3 • Enjeux des usage industriels et commerciaux des ondes non ionisantes électromagnétiques et acoustiques

Dominique DRON, Yves MAGNE et Ilarion PAVEL

https://www.economie.gouv.fr/files/files/directions_services/cge/ondes.pdf

Ce compte-rendu nous permet d'avoir une vision des enjeux de l'utilisation des ondes acoustiques. Cela nous a aidé pour élaborer nos problématiques.

4 • Voletons au gré du son !

Mathieu PEYROT, Malo REMBAUD, Nicolas TAFZI et Anys ZOGHELY

https://odpf.org/images/archives_docs/25eme/memoires/EquipeC/memoire.pdf

Ce mémoire réalisé par des lycéens à l'occasion des XXV^{ème} olympiades de physique nous permet d'avoir une étude théorique très abordable sans être simpliste. De plus nous avons à notre disposition un compte-rendu de l'expérience de lévitation. Cependant, les coûts liés aux composants sont très importants. Notre but sera de proposer une expérience similaire tout en restant dans un budget raisonnable afin de rendre possible l'application tout en répondant aux enjeux économiques.

5 • Ondes et vibrations

Étienne GRAVIER

Edition Ellipses – ISBN : 9782340-028562

Ce livre nous permet d'étendre notre étude théorique au niveaux L2 de physique et plus particulièrement sur la quantification du phénomène de pression acoustique lié aux ondes stationnaires.

L'étude, qui complète le document précédent, nous amène à l'équation de d'Alembert.

Nous partons premièrement de la relation suivante où P est la pression, P_0 la pression du fluide au repos et p_1 la différence de pression par rapport à P_0 :

$$P(x,t) = P_0 + p_1(x,t)$$

Et

$$P(x+dx,t) = P_0 + p_1(x+dx,t)$$

On note $dm = \rho_0 S dx$ et s le déplacement dû à la perturbation d'une tranche de fluide par rapport à sa position d'équilibre. On néglige le poids et les frottements.

Avec le principe fondamental de la dynamique, on obtient :

$$\rho_0 S dx \frac{\partial^2 s(x,t)}{\partial t^2} = S[-P(x+dx,t) + P(x,t)]$$

Ou encore :

$$\rho_0 S dx \frac{\partial^2 s(x,t)}{\partial t^2} = S[-p_1(x+dx,t) + p_1(x,t)] \simeq -S\frac{\partial p_1(x,t)}{\partial x} dx$$

On a alors

$$\frac{\partial p_1(x,t)}{\partial x} = -\rho_0 \frac{\partial^2 s(x,t)}{\partial t^2} \qquad (1)$$

On néglige les échanges thermiques entre les particules et on suppose l'entropie du système (notée S) constante. On note $\chi_S = -\frac{1}{V}\left(\frac{\partial V}{\partial P}\right)_S$ le coefficient de compressibilité isentropique.

Ce coefficient nous permet de faire le lien entre s et p_1. On obtient alors :

$$p_1(x,t) = -\frac{1}{\chi_S}\frac{\partial s(x,t)}{\partial x} \qquad (2)$$

En combinant les équations (1) et (2), on obtient :

$$\frac{\partial^2 p_1}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2 p_1}{\partial t^2} = 0$$

On a alors l'équation de d'Alembert vérifiée par p_1. On parle ici d'une onde de pression.

Cette étude théorique nous permettra ensuite de paramétrer notre modèle informatique.

II - Expérience :

Le but de notre montage est de faire léviter des objets de petites tailles, par exemple des grains de polystyrène ou des gouttes d'eau.

Matériel (Seulement les composants, les consommables ne sont pas spécifiés) :

- 12 transducteurs Ultrasons (KPUS-40T-16T-K768)
- 1 Arduino Uno
- 1 driver pour moteur (L298)

Réalisation de la structure :

La structure du système a été imprimée avec une imprimante 3D. Il est constitué de deux socles de forme hexagonale où seront disposer les transducteurs. Les deux socles sont reliés par l'intermédiaire de trois bras. La matière utilisée est le PLA afin de minimiser les coûts et faciliter la fabrication du prototype.

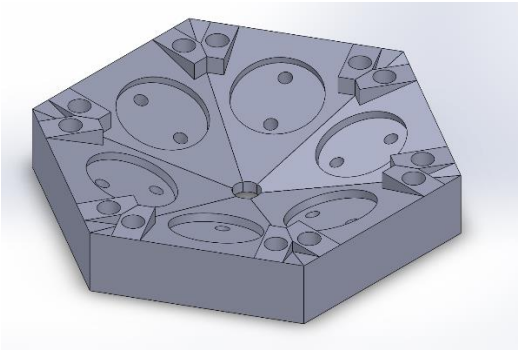

Figure 1 : modelisation du socle

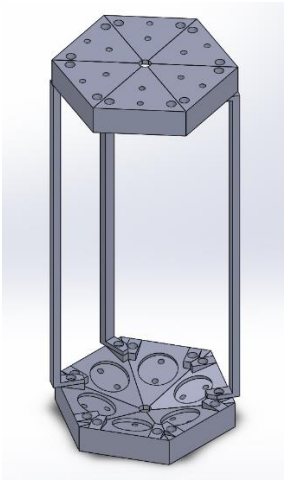

Figure 2 : assemblage numérique de la structure

Figure 3 : Assemblage de la structure avec les transducteurs

Réalisation du circuit :

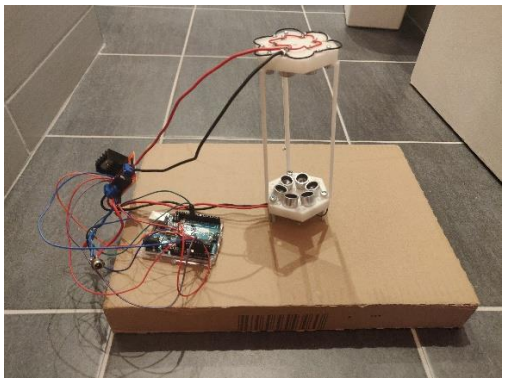

Figure 4 : Montage experimental

Le principe est d'alimenter les transducteurs avec une tensions alternatives ($f = 40\ kHz$). Pour se faire, il suffit d'utiliser la modulation par largeur d'impulsion de l'Arduino afin de générer cette tension. Cependant, l'Arduino ne peut fournir une tension maximum $Ucc = 5\ V$. Cette tension n'est pas suffisante et il faut amplifier le signal avec le driver et une alimentation externe continue de $U = 12\ V$.

Il faut ensuite souder les transducteurs en dérivation puis les relier au driver.

Le schéma du circuit est en annexe *(figure 5)*

Résultat :

L'expérience n'est actuellement pas concluante. Seulement, nous n'avons pas les moyens matériels de vérifier notre circuit pour le moment, notamment la nature et les caractéristiques (fréquence, amplitude et offset principalement) du signal fournis par l'Arduino afin de corriger les éventuels défauts. De plus le code *(figure 6)* récupéré de la thèse **TinyLev** *(Bibliographie n°1)* est très complexe et nous travaillons actuellement dessus afin de mieux le maîtriser.

Annexe :

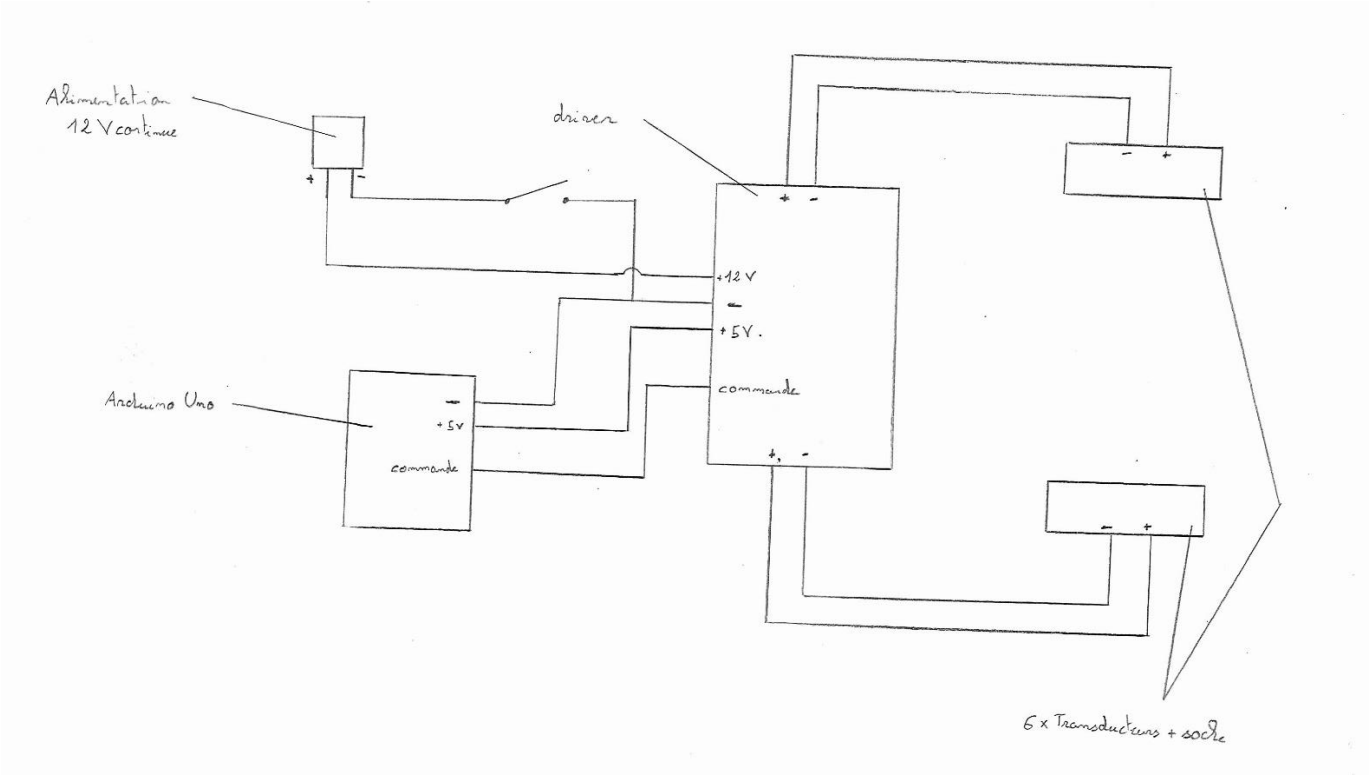

Figure 5 : Schéma du circuit électrique

```
#include <avr/sleep.h>
#include <avr/power.h>

#define N_PORTS 1
#define N_DIVS 24
#define WAIT_LOT(a) __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop"); __asm__
__volatile__ ("nop");__asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");__asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");  __asm__
__volatile__ ("nop"); __asm__ __volatile__ ("nop");__asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop")
#define WAIT_MID(a) __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop"); __asm__
__volatile__ ("nop");__asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");__asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__
__volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop")
#define WAIT_LIT(a) __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop"); __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop"); __asm__
__volatile__ ("nop");  __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop");  __asm__ __volatile__ ("nop")
#define OUTPUT_WAVE(pointer, d)  PORTC = pointer[d*N_PORTS + 0]
#define N_BUTTONS 6
#define STEP_SIZE 1
#define BUTTON_SENS 2500
#define N_FRAMES 24

static byte frame = 0;
static byte animation[N_FRAMES][N_DIVS] =
{{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0xa,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0xa,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0xa,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0xa,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0xa,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0xa,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x5,0x5,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0xa},
{0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0x6,0x6,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0x6,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0x6,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x5,0x9,0x9,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0x6,0x6,0x6,0x6,0x6},
```

```
{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x9,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0x6,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x9,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0x6,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x9,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0x6,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x9,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0x6,0x6},
{0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x5,0x9,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0xa,0x6}};

void setup()
{
  DDRC = 0b00001111; //A0 to A3 are the signal outputs
  PORTC = 0b00000000;

  pinMode(10, OUTPUT); //pin 10 (B2) will generate a 40kHz signal to sync
  pinMode(11, INPUT_PULLUP); //pin 11 (B3) is the sync in

  for (int i = 2; i < 8; ++i){ //pin 2 to 7 (D2 to D7) are inputs for the buttons
   pinMode(i, INPUT_PULLUP);
  }
 // generate a sync signal of 40khz in pin 10
 noInterrupts();         // disable all interrupts
 TCCR1A = bit (WGM10) | bit (WGM11) | bit (COM1B1); // fast PWM, clear OC1B on compare
 TCCR1B = bit (WGM12) | bit (WGM13) | bit (CS10);   // fast PWM, no prescaler
 OCR1A =  (F_CPU / 40000L) - 1;
 OCR1B = (F_CPU / 40000L) / 2;
 interrupts();           // enable all interrupts

 // disable everything that we do not need
 ADCSRA = 0;  // ADC

 power_adc_disable ();
 power_spi_disable();
 power_twi_disable();
 power_timer0_disable();
 Serial.begin(115200);

byte* emittingPointer = &animation[frame][0];
byte buttonsPort = 0;
bool anyButtonPressed;
bool buttonPressed[N_BUTTONS];
short buttonCounter = 0;

 LOOP:
  while(PINB & 0b00001000); //wait for pin 11 (B3) to go low

  OUTPUT_WAVE(emittingPointer, 0); buttonsPort = PIND; WAIT_LIT();
  OUTPUT_WAVE(emittingPointer, 1); anyButtonPressed = (buttonsPort & 0b11111100) != 0b11111100; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 2); buttonPressed[0] = buttonsPort & 0b00000100; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 3); buttonPressed[1] = buttonsPort & 0b00001000; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 4); buttonPressed[2] = buttonsPort & 0b00010000; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 5); buttonPressed[3] = buttonsPort & 0b00100000; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 6); buttonPressed[4] = buttonsPort & 0b01000000; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 7); buttonPressed[5] = buttonsPort & 0b10000000; WAIT_MID();
  OUTPUT_WAVE(emittingPointer, 8); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 9); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 10); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 11); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 12); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 13); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 14); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 15); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 16); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 17); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 18); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 19); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 20); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 21); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 22); WAIT_LOT();
  OUTPUT_WAVE(emittingPointer, 23);
 goto LOOP;
}


void loop(){}
```

Figure 6 : Code en C pour l'Arduino