Turbococco v.2

Informations

- Tenue du wiki : Pablo Fava
- Contact : pablo.fava@etu.sorbone-universite.fr
- Etudes : Licence Troisième année Mécanique - Sciences de la Terre
- Dates du Projet : du 4/02 au 31/04
- encadrants : Loïc Labrousse et Pierre Thery
- Dans le Cadre de l'ISTEP, dirigé par Fabrice Minoletti

Contexte

Mise en contexte : les coccolithophores sont de très bons indicateurs car ce sont des algues qui sont dans la zone de mélange pour pouvoir faire la photosynthèse, à l'inverse des foraminifères qui sont des organismes qui attendent majoritairement sous la zone de mélange. Les coccolithophores sont donc plus adaptés comme marqueur de la température passée.

Problème : leur taille de l'ordre du micromètre rend leur isolement par filtrage complexe.

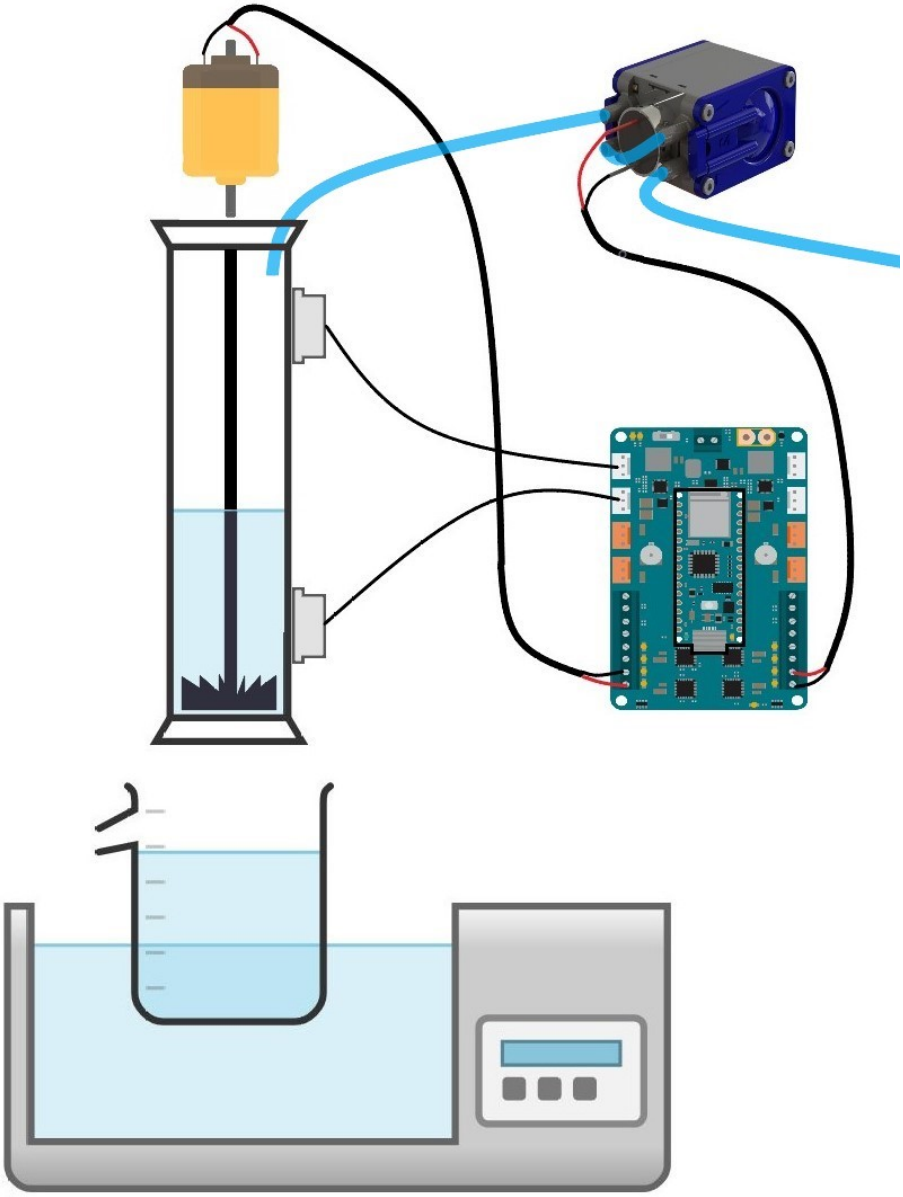
Le protocole mis au point par Fabrice Minoletti et son équipe consiste donc à faire vibrer une micro membrane filtrante à l'aide d'une cuve à ultrasons. Le filtre étant très fin, l'opération s'étend sur des laps de temps conséquents. L'idée est donc d'automatiser le remplissage du cylindre et d'ajouter une agitation pour faciliter le filtrage.

Objectifs

- Transférer le montage précédent sur une Arduino nano 33ioT et créer une application dans le cloud arduino pour contrôler le montage à distance

- réaliser une série de tests pour déterminer l'efficacité de l'agitation de la solution et du filtrage

- faciliter la réplication du montage.



Représentation graphique du montage réalisée par Ismaël

Matériel

- 1 carte arduino nano 33ioT
- deux capteurs de niveau d'eau
- un moteur
- un shield moteur
- câbles arduino
- bouchons et fixations et impression 3D
- pompes
- smartphone

Machines utilisées

- imprimantes 3D

- fer à souder
- perceuse manuelle

Construction

(Fichiers, photos, code, explications, paramètres d'usinage, photos, captures d'écran...)

Ce que l'on a récupéré

Tout le montage fonctionne autour d'une carte Arduino. Grâce au groupe aillant travaillé sur le projet l'an dernier, la structure du modèle est déjà opérationnelle. La carte Arduino est reliée à 4 composants : une pompe, un moteur agitateur et deux capteurs de niveau. Le principe général est d'aider la filtration par la micro membrane et d'automatiser son remplissage.

Ainsi, deux concepts sont importants :

- lorsque le capteur positionné en bas du tube ne détecte plus de solution, la pompe est activée pour un remplissage jusqu'à ce que le capteur haut détecte le niveau d'eau.
- sur une boucle de 120 secondes, le moteur est activé pendant les 5 premières secondes afin de remettre les particules en suspension et désencombrer la membrane filtrante.

Comment on souhaite le transformer

Afin d'avoir davantage de possibilités, nous avons transféré le montage sur une carte Arduino 33ioT, qui aura pour plus tard la possibilité de communiquer avec Arduino Cloud et ainsi être pilotée à distance par un smartphone. Connectée à un Shield moteur, elle permet de ne pas avoir de pin board.

Journal de bord - Avancée du projet à chaque étape, difficultés rencontrées, modifications et adaptations

10/02/2023

On commence à transférer le projet de l'Arduino Uno vers la nano mais on se rend vite compte qu'on n'a pas la bonne Arduino nano, il nous faut la 33ioT. Fabrice la commande pour la semaine suivante. ça nous bloque beaucoup parce qu'on ne peut pas commencer l'application non plus mais on est à peu près certain.e.s que tout fonctionne en-dehors de cela donc la session de la semaine suivante devrait être assez efficace.

17/02/2023

Ismaël a réussi à souder l'Arduino 33IOT et tout fonctionne. Seul problème : les capteurs ne détectent pas la présence d'eau. On essaie de serrer + les vis, sinon il faudra peut-être changer de capteur ou trouver une autre solution.

En attendant on doit modéliser la pièce manquante pour que l'hélice soit plus solide et voir si on a une tige de longueur suffisante pour passer sous le niveau du capteur bas.

On a réglé le problème en changeant les capteurs. Ils sont un peu moins réactifs mais ça ne gêne pas le fonctionnement global donc on les garde.

On a modélisé la pièce aux bonnes dimensions, mais on n'a pas de tige assez longue pour les nouveaux tubes du laboratoire. Pierre Théry va en commander à nouveau ou alors on peut en fabriquer une en tiges de métal mais on pense que ça ne sera pas assez rigide.

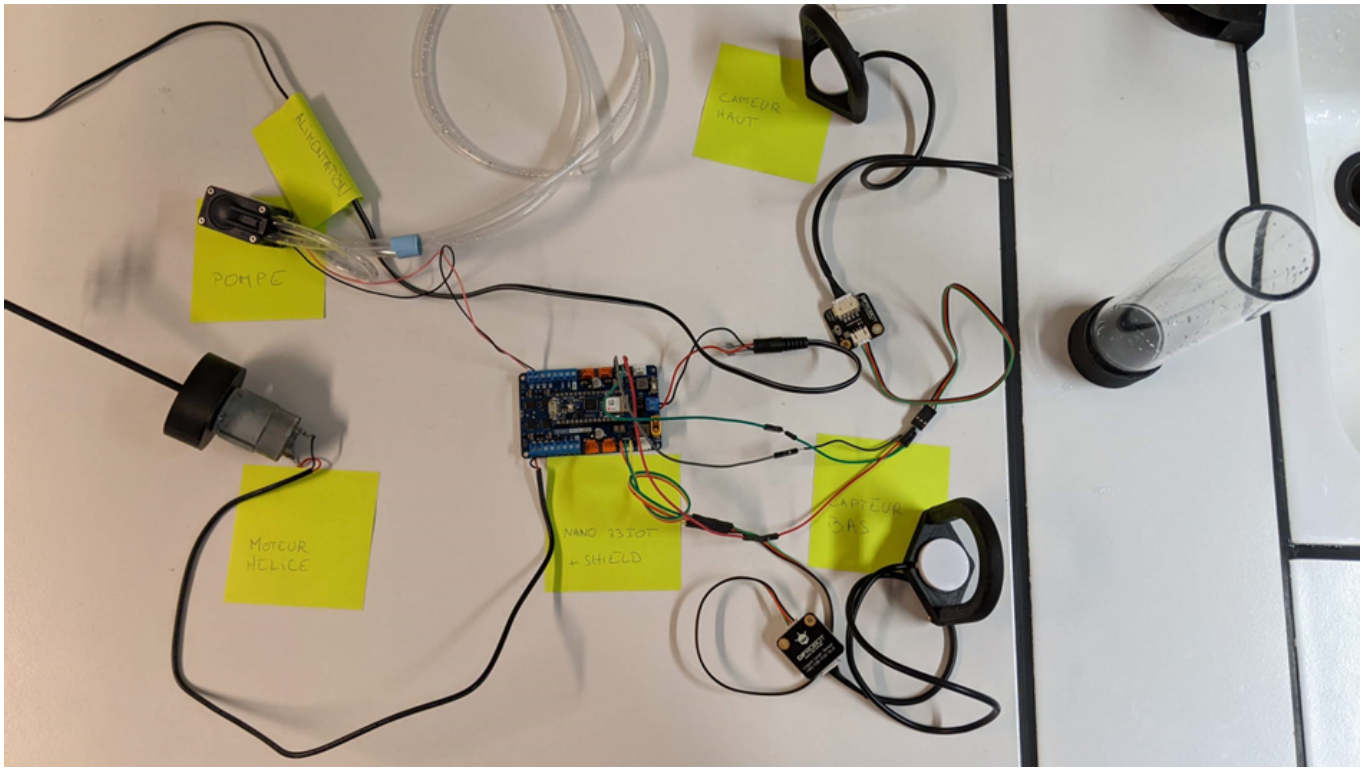


Photo du système total fonctionnel.

Léger problème d'étanchéité au bas du tube mais ça devrait se résoudre avec les tubes plus grands.

On a donc pu commencer à fouiller un peu pour créer l'application de contrôle à distance. On a établi les fonctionnalités nécessaires avec Fabrice :

- bouton démarrer le processus total

- bouton allumer et éteindre la pompe

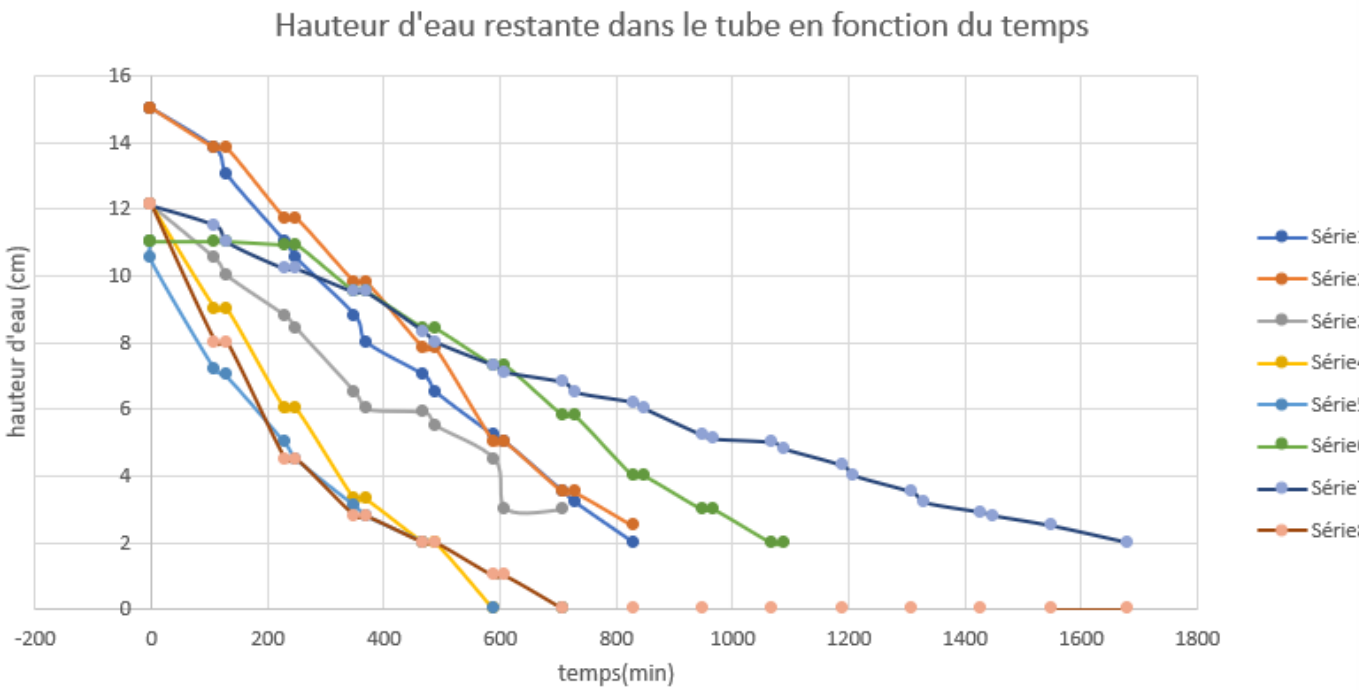
- bouton allumer et éteindre le moteur agitateur

- bouton éteindre tout le système

il faut également refaire la soudure de la Arduino nano 33ioT

24/03/2023

Après une série de tests par Téa, résultats assez étranges, globalement la filtration est moins rapide avec agitation moteur. On a rentré les résultats sur excel pour les analyser plus nettement :



Les temps les plus longs sont atteints lorsque le tube est agité par un moteur. Cela nous pose vraiment problème car le projet serait à moitié inefficace. On établit les sources d'erreurs potentielles :

- la position dans la cuve pourrait influer car il y a 3 sources d'ultrasons pour 4 emplacements

- le sens de rotation de l'hélice du moteur

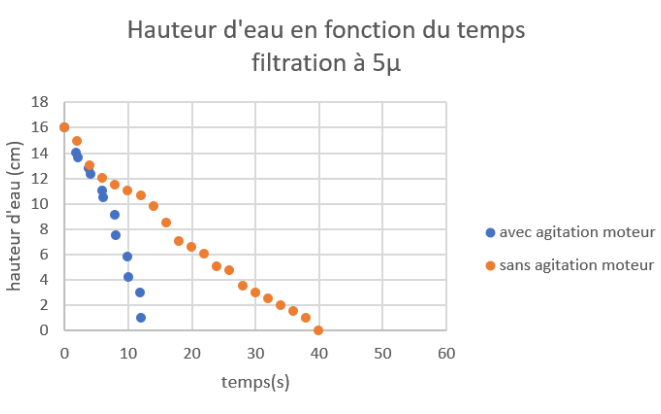
On lance donc l'impression d'un bouchon avec emplacement moteur supplémentaire pour pouvoir faire plusieurs essais en parallèle et plus facilement identifier le problème.

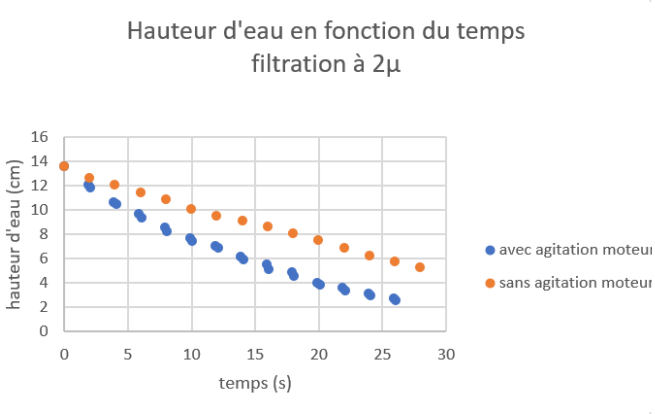
31/03/2023

Problème avec les impressions de la semaine passée, on n'avait pas pris en compte la dilatation lors de l'impression donc le moteur ne rentre pas dans l'emplacement prévu.

On recommence l'impression aux bonnes dimensions

Entre temps, Téa a refait des tests avec une membrane plus petite et une nouvelle cuve. Cette fois-ci l'agitation moteur permet bien un filtrage nettement plus rapide. Cela rend donc caduque le besoin d'un nouveau bouchon avec emplacement moteur. Voilà les graphiques obtenus avec les données de Téa :





De plus, les lames au microscope sont beaucoup moins polluées de particules sans intérêt.

Nous n'avons pas eu le temps de réitérer suffisamment l'expérience et s'assurer que le résultat serait identique à chaque répétition. Cela peut être un bon axe de perfectionnement du système.

14/04/2023

Nouvel objectif : rendre tout le système pilotable à distance grâce au Arduino ioT Cloud

On transfère le code déjà créé pour la carte Arduino et on l'adapte aux exigences du cloud. Beaucoup de problèmes de compilations, utilisation de Chat GPt4 pour les résoudre.

Il reste quelques problèmes, on espère les résoudre avant l'oral : le code compile mais les moteurs ne s'activent pas lors de l'activation des widgets sur l'application arduino

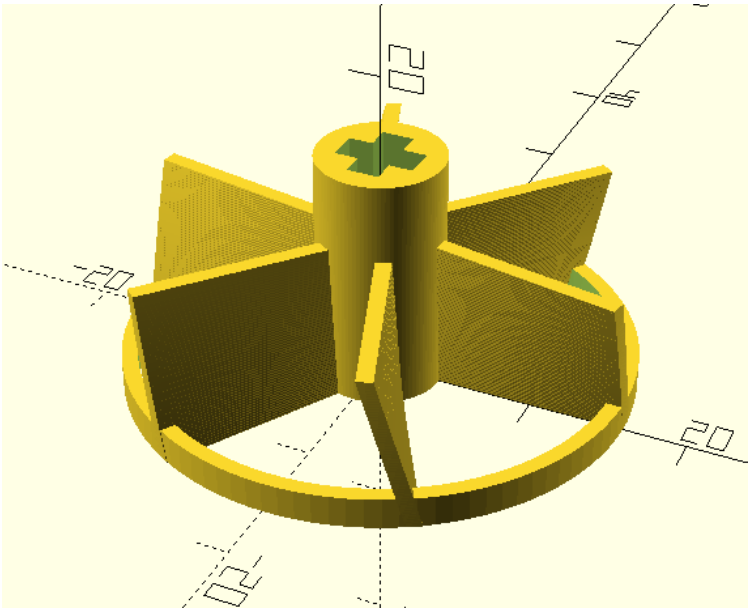
Codes Openscad

N.B : ajuster les dimensions pour prendre en compte la modification à l'impression

hélice agitation moteur

```
$fn=100;
nbPales=6;
rAxeRot=1.5;
rIntTub=32/2;
rTube=18;
//color("red")
//render()

translate([0,0,1]) {
    difference() {
        cylinder (h = 2, r=rIntTub, center=true);
        cylinder (h = 3, r=rIntTub-1, center=true);
    }
}
difference() {
    union (){
        linear_extrude(height = 10, twist = 10, slices = 60) {
            //circle(r=5);
            for(i = [0 : 1 : nbPales]) {
                rotate(a = i*360/nbPales) {
                    square(size = [rIntTub, 1]);
                }
            }
        }
       cylinder (h = 15, r=4, center = false);
    }
    translate([0,0,7.5]){
```

```
    linear_extrude(height = 15, twist = 0, slices = 60){
    translate ([0,0,3.5]) {

        square(size = [5, 2],center=true);//rectangle 1
        rotate (a=90){
        square(size = [5, 2],center=true);
                    }
    //cylinder (h = 30, r=rAxeRot);



}}}}
/*
    translate ([0,0,-1]) {
        cylinder (h = 30, r=rAxeRot);
    }
*/
```

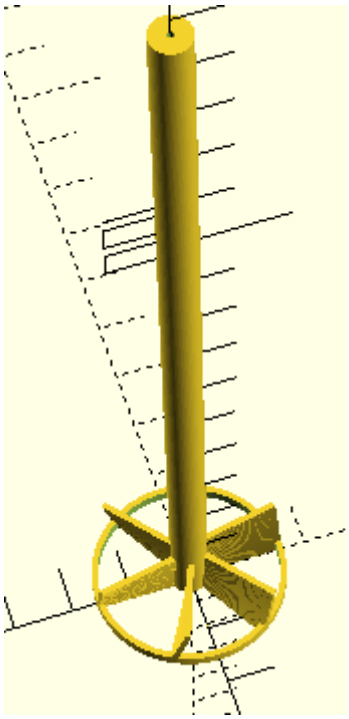


tige complète

```
$fn=100;
nbPales=6;
rAxeRot=1.5;
```

```
rIntTub=32/2;
rTube=18;
//color("red")
//render()

translate([0,0,1]) {
    difference() {
        cylinder (h = 2, r=rIntTub, center=true);
        cylinder (h = 3, r=rIntTub-1, center=true);
    }
}
difference() {
    union (){
        linear_extrude(height = 10, twist = -10, slices = 60) {
            //circle(r=5);
            for(i = [0 : 1 : nbPales]) {
                rotate(a = i*360/nbPales) {
                    square(size = [rIntTub, 1]);
                }
            }
        }
       cylinder (h = 147, r=3, center = false);

    }
    translate([0,0,140]){
    cylinder (h = 30, r=0.5);
    /*translate([0,0,7.5]){
    linear_extrude(height = 15, twist = 0, slices = 60){
    translate ([0,0,3.5]) {

        square(size = [3.8, 1],center=true);//rectangle 1
        rotate (a=90){
        square(size = [3.8, 1],center=true);*/
                    }}

    //vincent mouchi claire lix
```

```
//NB : le trou pour connexion moteur était un peu trop faible, à agrandir
```



bouchon avec pas de vis et insert moteur

```
$fn=100;
hInt=20;
l=80;
diam=42.3;
/*
color("red")
translate ([-diam/2,-0.5,-hInt/2]) {
    cube([diam,1,20], center=false);
}
color("green")
render()
translate ([-l/2,-0.5,5]) {
    cube([l,1,4], center=false);
}
color("blue")
render()
```

```
difference () {
    difference () {
        cylinder (d=60, h=10);
        cylinder (d=45, h=10);
    }
    translate ([-l/2,3,-l/2]) {cube([l,l,l], center=false);}
    translate ([-l/2,-l-3,-l/2]) {cube([l,l,l], center=false);}
}
*/
translate ([0,0,-1]) {
    difference () {
        cylinder (d=50, h=3);
        translate ([0,0,-0.5]) {cylinder (d=3, h=5);}
    }
}
SVL42Cap ();
extend_mot();

difference(){

    cylinder(d=16,h=2,center=true);
    cylinder(d=3,h=2,center=true);}

module extend_mot (){//extension pour insertion moteur
    difference (){
    translate([0,0,-14])cylinder (d=50,h=14);
    translate([0,0,-14])cylinder(d=21,h=10);
    translate([0,0,-4]) cylinder(d=16,h=4);

            }

          }



/*difference () { // Pour observer une coupe
    SVL42Cap ();
    union () {
```

```
     translate ([-l/2,5,-l/2]) {cube([l,l,l], center=false);}
       translate ([-l/2,-l-5,-l/2]) {cube([l,l,l], center=false);}
    }
}*/

module SVL42Cap () {
    difference () {
        cylinder (d=50, h=hInt);
        moduleInt ();
    }
    difference () { // sommet du capuchon
        color("red")
        render()
        union () {
            cylinder (d=50, h=1.5);
            difference () {
                cylinder (d=50, h=1.5);
                cylinder (d=3, h=2);
            }
        }

        //translate ([-15,2,-5]) {cylinder (d=1, h=10);}
        //translate ([-15,0,-5]) {cylinder (d=1, h=10);}
        //translate ([-15,-2,-5]) {cylinder (d=1, h=10);}
    }
}
module moduleInt () {
    translate ([0,0,hInt]) {
        difference () {
            cylinder (d=45, h=3, center=true);
            cylinder (d=37, h=4, center=true);
        }
    }
    metric_thread (diameter=diam+4, pitch=4, length=hInt, internal=true, thread_size=2, groove=true,
angle=10, leadin=0);
    translate ([0,0,-0.5]) { cylinder (d=diam, h=hInt+1); }
}
```

```
/*module extension_moteur (){

   difference (){
cylinder (d=50,h=10, center=true);
cylinder(d=21,h=10, center=true);
              }

   difference(){
cylinder(d=50,h=4, center=true);
cylinder(d=16,h=4, center = true);}

   }*/
/*
metric_thread (diameter=8, pitch=1, length=4, internal=true);

diameter -    outside diameter of threads in mm. Default: 8.
pitch    -    thread axial "travel" per turn in mm.  Default: 1.
length   -    overall axial length of thread in mm.  Default: 1.
internal -    true = clearances for internal thread (e.g., a nut).
              false = clearances for external thread (e.g., a bolt).
              (Internal threads should be "cut out" from a solid using
              difference ()).
n_starts -    Number of thread starts (e.g., DNA, a "double helix," has
              n_starts=2).  See wikipedia Screw_thread.
thread_size - (non-standard) axial width of a single thread "V" - independent
              of pitch.  Default: same as pitch.
groove      - (non-standard) subtract inverted "V" from cylinder (rather than
              add protruding "V" to cylinder).
square      - Square threads (per
              https://en.wikipedia.org/wiki/Square_thread_form).
rectangle   - (non-standard) "Rectangular" thread - ratio depth/(axial) width
              Default: 1 (square).
angle       - (non-standard) angle (deg) of thread side from perpendicular to
              axis (default = standard = 30 degrees).
taper       - diameter change per length (National Pipe Thread/ANSI B1.20.1
              is 1" diameter per 16" length). Taper decreases from 'diameter'
              as z increases.
leadin      - 0 (default): no chamfer; 1: chamfer (45 degree) at max-z end;
```

```
            2: chamfer at both ends, 3: chamfer at z=0 end.
leadfac     - scale of leadin chamfer (default: 1.0 = 1/2 thread).
*/

/*
 * ISO-standard metric threads, following this specification:
 *          http://en.wikipedia.org/wiki/ISO_metric_screw_thread
 *
 * Copyright 2017 Dan Kirshner - dan_kirshner@yahoo.com
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * See <http://www.gnu.org/licenses/>.
 *
 * Version 2.3.  2017-08-31  Default for leadin: 0 (best for internal threads).
 * Version 2.2.  2017-01-01  Correction for angle; leadfac option.  (Thanks to
 *                       Andrew Allen <a2intl@gmail.com>.)
 * Version 2.1.  2016-12-04  Chamfer bottom end (low-z); leadin option.
 * Version 2.0.  2016-11-05  Backwards compatibility (earlier OpenSCAD) fixes.
 * Version 1.9.  2016-07-03  Option: tapered.
 * Version 1.8.  2016-01-08  Option: (non-standard) angle.
 * Version 1.7.  2015-11-28  Larger x-increment - for small-diameters.
 * Version 1.6.  2015-09-01  Options: square threads, rectangular threads.
 * Version 1.5.  2015-06-12  Options: thread_size, groove.
 * Version 1.4.  2014-10-17  Use "faces" instead of "triangles" for polyhedron
 * Version 1.3.  2013-12-01  Correct loop over turns -- don't have early cut-off
 * Version 1.2.  2012-09-09  Use discrete polyhedra rather than linear_extrude ()
 * Version 1.1.  2012-09-07  Corrected to right-hand threads!
 */

// Examples.
```

```
//
// Standard M8 x 1.
// metric_thread (diameter=8, pitch=1, length=4);

// Square thread.
// metric_thread (diameter=8, pitch=1, length=4, square=true);

// Non-standard: long pitch, same thread size.
//metric_thread (diameter=8, pitch=4, length=4, thread_size=1, groove=true);

// Non-standard: 20 mm diameter, long pitch, square "trough" width 3 mm,
// depth 1 mm.
//metric_thread (diameter=20, pitch=8, length=16, square=true, thread_size=6,
//               groove=true, rectangle=0.333);

// English: 1/4 x 20.
//english_thread (diameter=1/4, threads_per_inch=20, length=1);

// Tapered.  Example -- pipe size 3/4" -- per:
// http://www.engineeringtoolbox.com/npt-national-pipe-taper-threads-d_750.html
// english_thread (diameter=1.05, threads_per_inch=14, length=3/4, taper=1/16);

// Thread for mounting on Rohloff hub.
//difference () {
//   cylinder (r=20, h=10, $fn=100);
//
//   metric_thread (diameter=34, pitch=1, length=10, internal=true, n_starts=6);
//}



// ------------------------------------------------------------------------
function segments (diameter) = min (50, ceil (diameter*6));



// ------------------------------------------------------------------------
// diameter -    outside diameter of threads in mm. Default: 8.
// pitch    -    thread axial "travel" per turn in mm.  Default: 1.
// length   -    overall axial length of thread in mm.  Default: 1.
```

```
// internal -    true = clearances for internal thread (e.g., a nut).
//               false = clearances for external thread (e.g., a bolt).
//               (Internal threads should be "cut out" from a solid using
//               difference ()).
// n_starts -    Number of thread starts (e.g., DNA, a "double helix," has
//               n_starts=2).  See wikipedia Screw_thread.
// thread_size - (non-standard) axial width of a single thread "V" - independent
//               of pitch.  Default: same as pitch.
// groove      - (non-standard) subtract inverted "V" from cylinder (rather than
//               add protruding "V" to cylinder).
// square      - Square threads (per
//               https://en.wikipedia.org/wiki/Square_thread_form).
// rectangle   - (non-standard) "Rectangular" thread - ratio depth/(axial) width
//               Default: 1 (square).
// angle       - (non-standard) angle (deg) of thread side from perpendicular to
//               axis (default = standard = 30 degrees).
// taper       - diameter change per length (National Pipe Thread/ANSI B1.20.1
//               is 1" diameter per 16" length). Taper decreases from 'diameter'
//               as z increases.
// leadin      - 0 (default): no chamfer; 1: chamfer (45 degree) at max-z end;
//               2: chamfer at both ends, 3: chamfer at z=0 end.
// leadfac     - scale of leadin chamfer (default: 1.0 = 1/2 thread).
module metric_thread (diameter=8, pitch=1, length=1, internal=false, n_starts=1,
                      thread_size=-1, groove=false, square=false, rectangle=0,
                      angle=30, taper=0, leadin=0, leadfac=1.0)
{
  // thread_size: size of thread "V" different than travel per turn (pitch).
  // Default: same as pitch.
  local_thread_size = thread_size == -1 ? pitch : thread_size;
  local_rectangle = rectangle ? rectangle : 1;

  n_segments = segments (diameter);
  h = (square || rectangle) ? local_thread_size*local_rectangle/2 : local_thread_size / (2 * tan(angle));

  h_fac1 = (square || rectangle) ? 0.90 : 0.625;

  // External thread includes additional relief.
  h_fac2 = (square || rectangle) ? 0.95 : 5.3/8;
```

```
tapered_diameter = diameter - length*taper;

difference () {
  union () {
    if (! groove) {
      metric_thread_turns (diameter, pitch, length, internal, n_starts,
                           local_thread_size, groove, square, rectangle, angle,
                           taper);
    }

    difference () {

      // Solid center, including Dmin truncation.
      if (groove) {
        cylinder (r1=diameter/2, r2=tapered_diameter/2,
                  h=length, $fn=n_segments);
      } else if (internal) {
        cylinder (r1=diameter/2 - h*h_fac1, r2=tapered_diameter/2 - h*h_fac1,
                  h=length, $fn=n_segments);
      } else {

        // External thread.
        cylinder (r1=diameter/2 - h*h_fac2, r2=tapered_diameter/2 - h*h_fac2,
                  h=length, $fn=n_segments);
      }

      if (groove) {
        metric_thread_turns (diameter, pitch, length, internal, n_starts,
                             local_thread_size, groove, square, rectangle,
                             angle, taper);
      }
    }
  }

  // chamfer z=0 end if leadin is 2 or 3
  if (leadin == 2 || leadin == 3) {
    difference () {
```

```
        cylinder (r=diameter/2 + 1, h=h*h_fac1*leadfac, $fn=n_segments);

        cylinder (r2=diameter/2, r1=diameter/2 - h*h_fac1*leadfac, h=h*h_fac1*leadfac,
                $fn=n_segments);
      }
    }

    // chamfer z-max end if leadin is 1 or 2.
    if (leadin == 1 || leadin == 2) {
      translate ([0, 0, length + 0.05 - h*h_fac1*leadfac]) {
        difference () {
          cylinder (r=diameter/2 + 1, h=h*h_fac1*leadfac, $fn=n_segments);
          cylinder (r1=tapered_diameter/2, r2=tapered_diameter/2 - h*h_fac1*leadfac,
h=h*h_fac1*leadfac,
                  $fn=n_segments);
        }
      }
    }
  }
}
// ------------------------------------------------------------------------
module metric_thread_turns (diameter, pitch, length, internal, n_starts,
                        thread_size, groove, square, rectangle, angle,
                        taper)
{
  // Number of turns needed.
  n_turns = floor (length/pitch);

  intersection () {

    // Start one below z = 0.  Gives an extra turn at each end.
    for (i=[-1*n_starts : n_turns+1]) {
      translate ([0, 0, i*pitch]) {
        metric_thread_turn (diameter, pitch, internal, n_starts,
                          thread_size, groove, square, rectangle, angle,
                          taper, i*pitch);
      }
    }
```

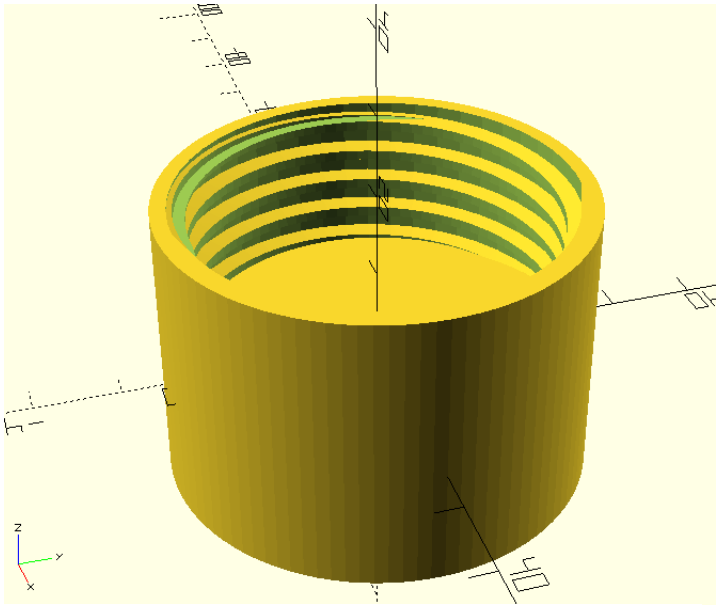
```
    // Cut to length.
    translate ([0, 0, length/2]) {
      cube ([diameter*3, diameter*3, length], center=true);
    }
  }
}


// ------------------------------------------------------------------------
module metric_thread_turn (diameter, pitch, internal, n_starts, thread_size,
                           groove, square, rectangle, angle, taper, z)
{
  n_segments = segments (diameter);
  fraction_circle = 1.0/n_segments;
  for (i=[0 : n_segments-1]) {
    rotate ([0, 0, i*360*fraction_circle]) {
      translate ([0, 0, i*n_starts*pitch*fraction_circle]) {
        //current_diameter = diameter - taper*(z + i*n_starts*pitch*fraction_circle);
        thread_polyhedron ((diameter - taper*(z + i*n_starts*pitch*fraction_circle))/2,
                           pitch, internal, n_starts, thread_size, groove,
                           square, rectangle, angle);
      }
    }
  }
}


// ------------------------------------------------------------------------
module thread_polyhedron (radius, pitch, internal, n_starts, thread_size,
                          groove, square, rectangle, angle)
{
  n_segments = segments (radius*2);
  fraction_circle = 1.0/n_segments;

  local_rectangle = rectangle ? rectangle : 1;

  h = (square || rectangle) ? thread_size*local_rectangle/2 : thread_size / (2 * tan(angle));
```

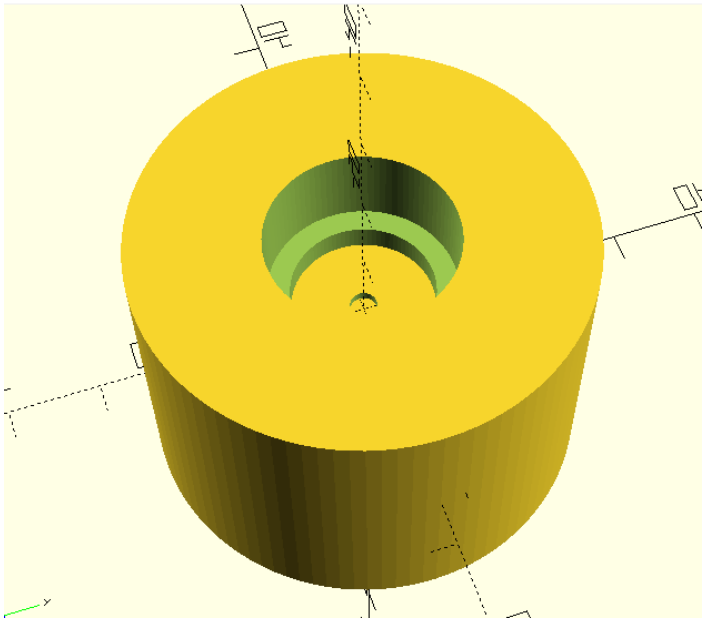
```
outer_r = radius + (internal ? h/20 : 0); // Adds internal relief.
//echo (str ("outer_r: ", outer_r));

// A little extra on square thread -- make sure overlaps cylinder.
h_fac1 = (square || rectangle) ? 1.1 : 0.875;
inner_r = radius - h*h_fac1; // Does NOT do Dmin_truncation - do later with
                             // cylinder.

translate_y = groove ? outer_r + inner_r : 0;
reflect_x   = groove ? 1 : 0;

// Make these just slightly bigger (keep in proportion) so polyhedra will
// overlap.
x_incr_outer = (! groove ? outer_r : inner_r) * fraction_circle * 2 * PI * 1.02;
x_incr_inner = (! groove ? inner_r : outer_r) * fraction_circle * 2 * PI * 1.02;
z_incr = n_starts * pitch * fraction_circle * 1.005;

/*
 (angles x0 and x3 inner are actually 60 deg)

                    /\  (x2_inner, z2_inner) [2]
                   /  \
(x3_inner, z3_inner) /    \
             [3]   \     \
                   |\     \ (x2_outer, z2_outer) [6]
                   | \    /
                   |  \  /|
          z        |[7]\/ / (x1_outer, z1_outer) [5]
          |        |   | /
          |   x    |   |/
          |  /     |   / (x0_outer, z0_outer) [4]
          | /      |  /     (behind: (x1_inner, z1_inner) [1]
          |/       | /
 y________|        |/
(r)                / (x0_inner, z0_inner) [0]

*/
```

```
x1_outer = outer_r * fraction_circle * 2 * PI;

z0_outer = (outer_r - inner_r) * tan(angle);
//echo (str ("z0_outer: ", z0_outer));

//polygon ([[inner_r, 0], [outer_r, z0_outer],
//        [outer_r, 0.5*pitch], [inner_r, 0.5*pitch]]);
z1_outer = z0_outer + z_incr;

// Give internal square threads some clearance in the z direction, too.
bottom = internal ? 0.235 : 0.25;
top    = internal ? 0.765 : 0.75;

translate ([0, translate_y, 0]) {
  mirror ([reflect_x, 0, 0]) {

    if (square || rectangle) {

      // Rule for face ordering: look at polyhedron from outside: points must
      // be in clockwise order.
      polyhedron (
        points = [
                [-x_incr_inner/2, -inner_r, bottom*thread_size],         // [0]
                [x_incr_inner/2, -inner_r, bottom*thread_size + z_incr], // [1]
                [x_incr_inner/2, -inner_r, top*thread_size + z_incr],    // [2]
                [-x_incr_inner/2, -inner_r, top*thread_size],            // [3]

                [-x_incr_outer/2, -outer_r, bottom*thread_size],         // [4]
                [x_incr_outer/2, -outer_r, bottom*thread_size + z_incr], // [5]
                [x_incr_outer/2, -outer_r, top*thread_size + z_incr],    // [6]
                [-x_incr_outer/2, -outer_r, top*thread_size]             // [7]
               ],

        faces = [
                [0, 3, 7, 4],  // This-side trapezoid

                [1, 5, 6, 2],  // Back-side trapezoid
```

```
            [0, 1, 2, 3],  // Inner rectangle

            [4, 7, 6, 5],  // Outer rectangle

            // These are not planar, so do with separate triangles.
            [7, 2, 6],     // Upper rectangle, bottom
            [7, 3, 2],     // Upper rectangle, top

            [0, 5, 1],     // Lower rectangle, bottom
            [0, 4, 5]      // Lower rectangle, top
           ]
  );
} else {

  // Rule for face ordering: look at polyhedron from outside: points must
  // be in clockwise order.
  polyhedron (
    points = [
            [-x_incr_inner/2, -inner_r, 0],                      // [0]
            [x_incr_inner/2, -inner_r, z_incr],                  // [1]
            [x_incr_inner/2, -inner_r, thread_size + z_incr],    // [2]
            [-x_incr_inner/2, -inner_r, thread_size],            // [3]

            [-x_incr_outer/2, -outer_r, z0_outer],               // [4]
            [x_incr_outer/2, -outer_r, z0_outer + z_incr],       // [5]
            [x_incr_outer/2, -outer_r, thread_size - z0_outer + z_incr], // [6]
            [-x_incr_outer/2, -outer_r, thread_size - z0_outer]  // [7]
           ],

    faces = [
            [0, 3, 7, 4],  // This-side trapezoid

            [1, 5, 6, 2],  // Back-side trapezoid

            [0, 1, 2, 3],  // Inner rectangle

            [4, 7, 6, 5],  // Outer rectangle
```

```
                    // These are not planar, so do with separate triangles.
                    [7, 2, 6],     // Upper rectangle, bottom
                    [7, 3, 2],     // Upper rectangle, top

                    [0, 5, 1],     // Lower rectangle, bottom
                    [0, 4, 5]      // Lower rectangle, top
                   ]
          );
        }
      }
    }
}
```





Code Arduino nano par Ismaël

```
#include <ArduinoMotorCarrier.h>
        const unsigned long equivalence_ms_s=1000;                  // constante pour la convertion des ms en s
        unsigned long heure_start_ms, heure_start_sec;              // variables pour stocker l'heure du début en ms et s
        unsigned long heure_delay_ms, heure_delay_sec;              // variables pour stocker l'heure du retard en ms et s

            void setup() {
```

```
        Serial.begin(9600);                         // initialisation de la communication série à 9600 bites par s

    //Establishing the communication with the motor shield
  if (controller.begin())
  {
   Serial.print("MKR Motor Shield connected, firmware version ");
   Serial.println(controller.getFWVersion());
  }
  else
  {
   Serial.println("Couldn't connect! Is the red led blinking? You may need to update the firmware with FWUpdater sketch");
   while (1);
  }

  // Reboot the motor controller; brings every value back to default
  Serial.println("reboot");
  controller.reboot();
  delay(500);
   M1.setDuty(0);
   M2.setDuty(0);
            }
   int remplissage=1;                          // variable "remplissage" qui vaut 1 par défaut
   int chrono = 0;                            // variable "chrono" qui vaut 0 par défaut

   void loop() {

        int sensorValue0 = analogRead(A7);              // lit la valeur analogique reçue en A0, envoyée par le capteur BAS
        int sensorValue1 = analogRead(A2);              // lit la valeur analogique reçue en A2, envoyée par le capteur HAUT

        //Serial.println(sensorValue1);
        Serial.println(sensorValue0);

        float voltage0 = sensorValue0 * (5.0 / 1023.0);     // conversion de la valeur analogique (de 0 à 1023) en voltage (de 0 à 5V) - capteur BAS
        float voltage1 = sensorValue1 * (5.0 / 1023.0);     // conversion de la valeur analogique (de 0 à
```

```
1023) en voltage (de 0 à 5V) - capteur HAUT

            //Serial.println(voltage0);
            //Serial.println(voltage1);                ||||||||

                        if (remplissage == 1) {                          // si le niveau d'eau est sous le capteur du
HAUT, alors :
                  if ((voltage0< 2.5) & (voltage1 < 2.5)) {                      // si le capteur BAS ne détecte pas
l'eau, alors :
                     M1.setDuty(20);                    //On fait tourner le moteur à 20%
                     M2.setDuty(100);                    // On fait tourner la pompe à fond
                     remplissage = 1;                         // on reste dans cette boucle
                  }

                  if ((voltage0> 2.5) & (voltage1 < 2.5)) {                      // si le capteur BAS ne détecte pas
l'eau, alors :
                     M1.setDuty(20);                    // On fait tourner le moteur à 20%
                     M2.setDuty(100);                    // On fait tourner la pompe à fond
                  }

                  if ((voltage1 > 2.5)&(voltage0>2.5)){            // si le capteur HAUT détecte l'eau, alors :
                     M1.setDuty(0);                    // On coupe le moteur
                     M2.setDuty(0);                    // On coupe la pompe
                     remplissage=0;                         // on entre dans la boucle suivante
                     heure_start_ms = millis();                                          // dès que le capteur HAUT
détecte l'eau, lecture de l'heure du début en ms
                     heure_start_sec = (heure_start_ms/equivalence_ms_s);
                     chrono = 0;                                                  // conversion des ms en s
                  }
               }

               if (remplissage == 0) {                          // si le niveau d'eau est au dessus le capteur HAUT,
alors :
                  if (voltage0 > 2.5) {                       // si le niveau d'eau est au dessus le capteur BAS, alors

                     if (chrono <= 5) {                    // si le chrono est < 5 s, alors :
                           M1.setDuty(20);                                    // "pas de courant" en sortie 2 : le
circuit est fermé dans la carte relais, le moteur fonctionne
                           M2.setDuty(0);
```

```
                heure_delay_ms = millis();                                  // lecture du retard en ms
                heure_delay_sec = (heure_delay_ms/equivalence_ms_s);        // conversion des
ms en s
                chrono = heure_delay_sec - heure_start_sec;                 // calcul du chrono =
(heure du début  - heure du retard)
                remplissage = 0;
              }

              if  ((chrono <= 60) & (chrono > 5)) {// si le chrono est > 5 s, alors :
                M1.setDuty(0);                              // On eteint le moteur
                M2.setDuty(0);                              // On eteint la pompe
                heure_delay_ms = millis();                          // lecture de l'heure du retard
en ms
                heure_delay_sec = (heure_delay_ms/equivalence_ms_s);        // conversion des
ms en s
                chrono = heure_delay_sec - heure_start_sec;                 // calcul du chrono =
(heure du début - heure du retard)
                remplissage = 0;
              }

              if (chrono > 60)   {                          // si le chrono > 10 s, alors :
               chrono = 0;                                          // le chrono est remis à 0
               heure_start_ms = millis();                           // l'heure du début est remis à
jour en ms
               heure_start_sec = (heure_start_ms/equivalence_ms_s);         // conversion des
ms en s
               remplissage = 0;
              }
            }

            if (voltage0 < 2.5) {
              remplissage = 1;
            }
          }
          //Serial.println(remplissage);          ||||||||||||||||||
        }   // ferme void loop
```

Remerciements

Un grand merci à Fabrice Minoletti de nous avoir accueillis dans son laboratoire et de nous avoir fait confiance pour la réalisation de son projet. Merci à Loïc Labrousse pour sa disponibilité et à Pierre Thery pour son soutien en toute situation. Merci enfin et évidemment à Ismaël et Téa de constituer cette équipe efficace avec qui il a fait bon de travailler.

Revision #22
Created 3 February 2023 12:55:20 by Fava Pablo
Updated 15 May 2023 10:26:40 by Christian Simon