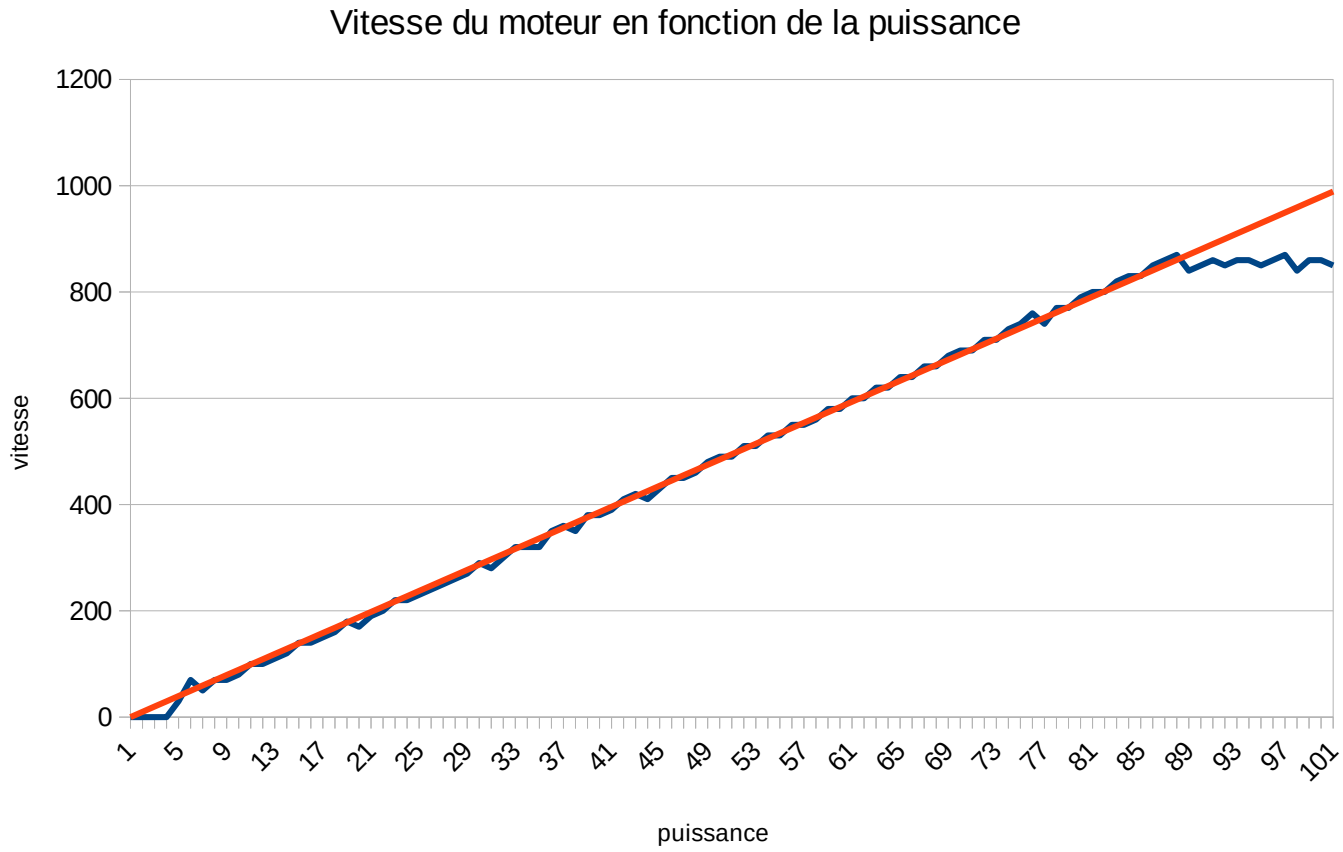
Addwyn LE LANN
M2 Info ILIADE

Rapport projet boussole avec robot EV3 et RobotC

*Fichier **motorCharacterization.c** permettant la caractérisation du moteur*

```
#pragma config(Sensor, S1, TouchSensor, sensorEV3_Touch)
#pragma config(Sensor, S2, GyroSensor, sensorEV3_Gyro)
#pragma config(Motor, motorA, motor, tmotorEV3_Large, PIDControl, encoder)

/*
 * @author: Addwyn Le lann m2 iliade
 * Fichier préliminaire permettant de déterminer les constantes liées au moteur.
 * Cela nous donne la vitesse de rotation du moteur en fonction de la puissance.
 * La puissance est testée de 0% à 100% avec un pas de 1%.
 */

// Variables globales
string s_val;
long fileHandle;

task motorCharacterization() {
  // Tableau contentant les différentes vitesse par puissance
  float tab [101];
  // Tabulation entre les différentes puissances
  for (int i=0;i<=100;i++) {
    // Règle la puissance du moteur
    setMotorSpeed(motorA, i);
    // Légère attente pour laisser le moteur tourner à la puissance
    delay(100);
    // Conversion en degré par seconde
    tab[i]=getMotorRPM(motorA)*6;
  }

  // Sauvegarde des données dans le fichier
  for (int i=0; i<=100;i++) {
    stringFormat(s_val, "%ld\n", tab[i]);
    fileWriteData(fileHandle, s_val, strlen(s_val));
  }

  // Fin du programme et fermeture du fichier
  fileClose(fileHandle);
  stopAllTasks();
}

task main() {
  // Créer un fichier pour récupérer les résultats
  fileHandle = fileOpenWrite("puissance");
  // Lance notre tâche d'analyse
  startTask(motorCharacterization);
}
```

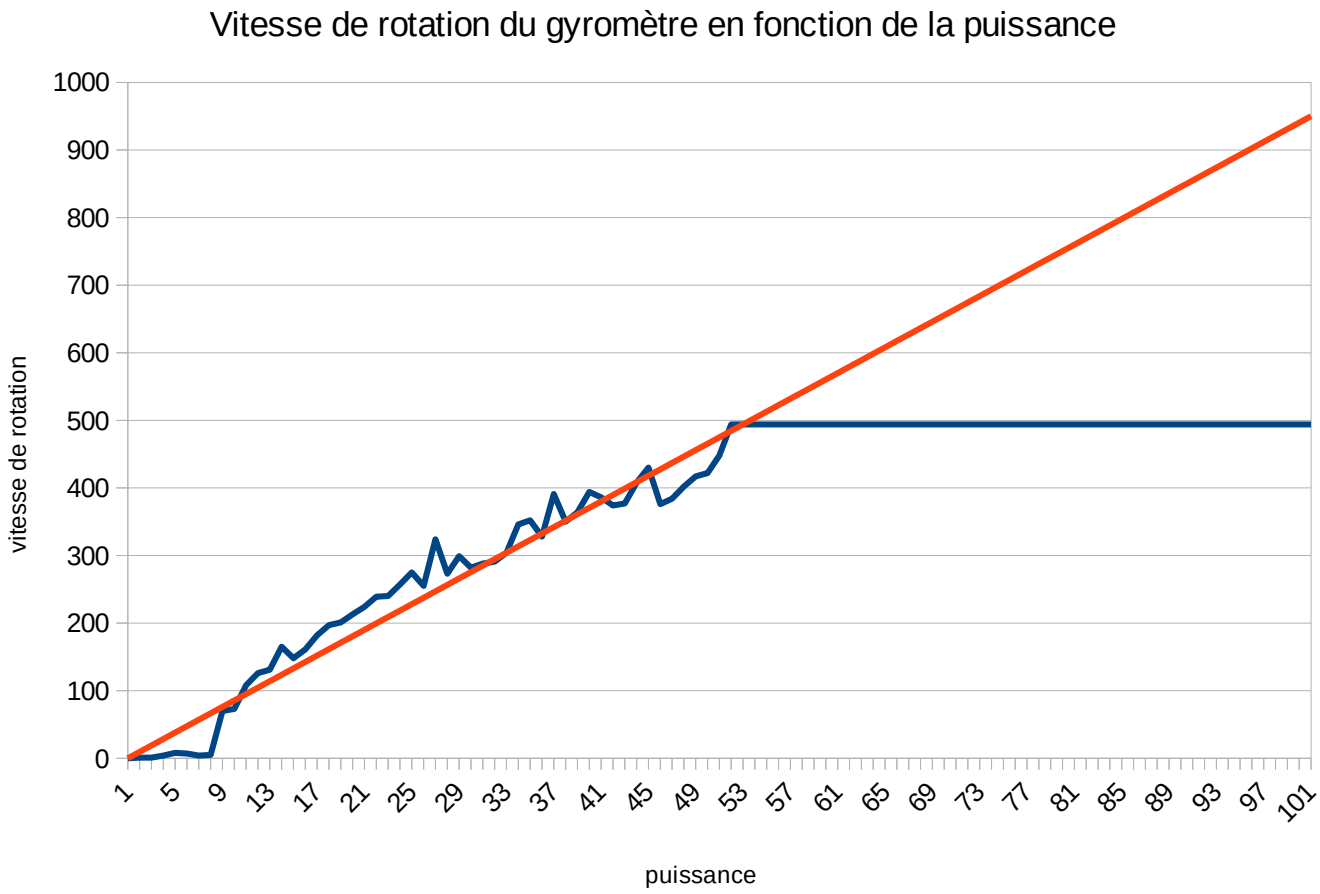


On peut constater que la vitesse du moteur est proportionnelle à sa puissance fournie.
Un plateau se dessine lorsque la puissance est égal ou supérieur à 87 %.
Il est donc inutile de faire tourner le moteur au-dessus de cette puissance car il ne tournera pas plus vite.
On obtient donc une pente d'environ 10 %.

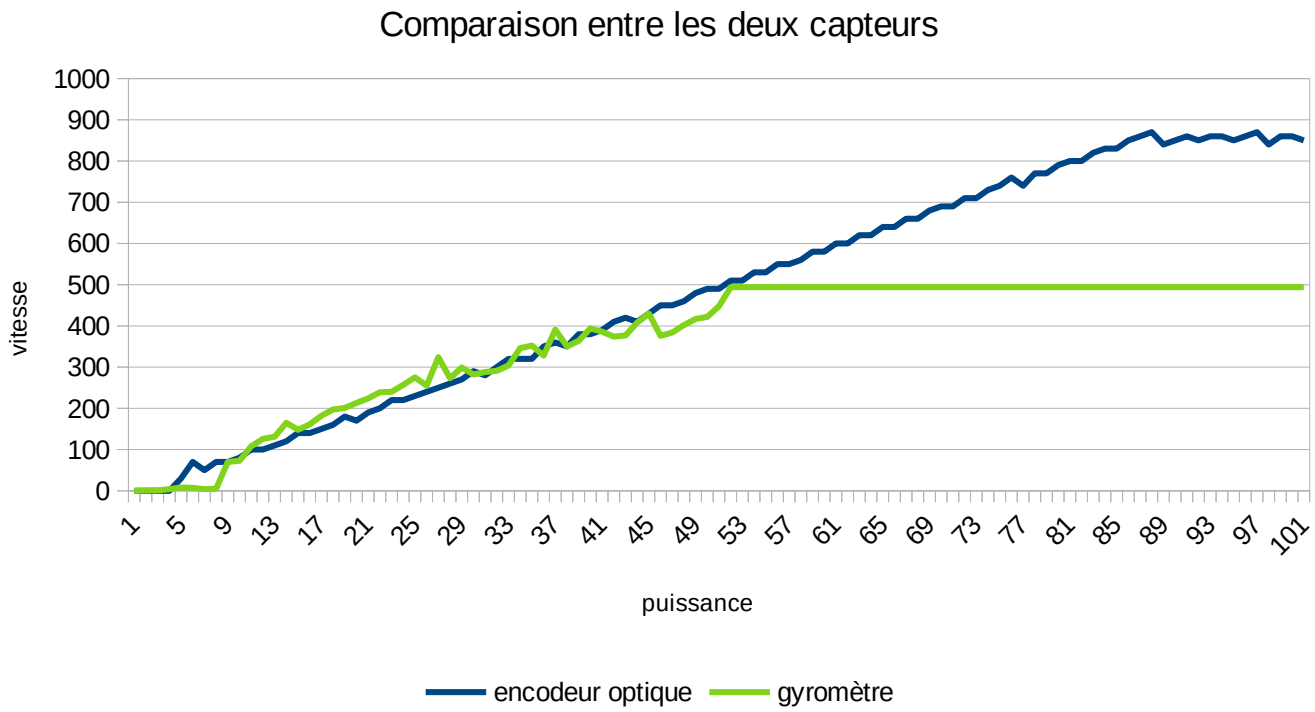
Cette analyse nous permet de faire ressortir certaines constantes :
minPower : 5 – puissance minimale à laquelle le moteur commence à tourner
maxPower : 87 – puissance maximale au-delà duquel la vitesse n'augmente plus
maxSpeed : 860 – vitesse de rotation maximale atteignable avec le moteur
pente : 9,88505747126437 – pente de la fonction affine entre le rapport vitesse et puissance du moteur

*Fichier **gyroRateCharacterization.c** permettant la caractérisation du gyromètre*

```
#pragma config(Sensor, S1, TouchSensor, sensorEV3_Touch)
#pragma config(Sensor, S2, GyroSensor, sensorEV3_Gyro)
#pragma config(Motor, motorA, moteurA, tmotorEV3_Large, PIDControl, encoder)

/*
 * @author: Addwyn Le lann m2 iliade
 * Fichier préliminaire permettant de déterminer les constantes liées au gyromètre.
 * Cela nous donne la vitesse délivrée par le gyromètre en degrés par
 * seconde en fonction de la puissance.
 * La puissance est testée de 0% à 100% avec un pas de 1%.
 */

// Variables globales
string s_val;
long fileHandle;

task gyroRateCharacterization() {
  // Tableau contentant les différentes vitesse par puissance
  long tab [101];
  // Tabulation entre les différentes puissances
  for (int i=0;i<=100;i++) {
    // Règle la puissance du moteur dans le sens antihoraire
    setMotorSpeed(motorA, -i);
    // Légère attente pour laisser le moteur tourner à la puissance
    // Et ainsi prendre la mesure après que le moteur est fait son demi-tour
    delay(200);
    // Sauvegarde de la vitesse dans le tableau
    tab[i] = getGyroRate(S2);
    // Règle la puissance du moteur dans le sens horaire
    // Permet de dérouler le câble du gyromètre
    setMotorSpeed(motorA, i);
    // Laisse le câble se dérouler
    delay(200);
  }

  // Sauvegarde des données dans le fichier
  for (int i=0; i<=100;i++) {
    stringFormat(s_val, "%ld\n", tab[i]);
    fileWriteData(fileHandle, s_val, strlen(s_val));
  }

  // Fin du programme et fermeture du fichier
  fileClose(fileHandle);
  stopAllTasks();
}

task main() {
  // Créer un fichier pour récupérer les résultats
  fileHandle = fileOpenWrite("gyro");
  // Lance notre tâche d'analyse
  startTask(gyroRateCharacterization);
}
```



On peut constater que la vitesse de rotation du gyromètre est proportionnelle à la puissance fournit au moteur.
Mais il intervient assez vite un plateau à partir duquel le gyromètre n'est plus capable de mesurer une vitesse de rotation supérieur à 500 degrés par seconde.
Ce plateau se dessine lorsque la puissance est égal ou supérieur à 52 %.
Il est donc inutile de faire tourner le moteur au-dessus de cette puissance car il ne tournera pas plus vite.
On obtient donc une pente d'environ 10 %.
Cette analyse nous permet de faire ressortir certaines constantes :
minPower2 : 9 – puissance minimale à laquelle le gyromètre mesure une rotation
maxPower2 : 52 – puissance maximale au-delà duquel le gyromètre ne mesure plus d'augmentation de la rotation
maxSpeed2 : 494 – vitesse de rotation maximale mesurée par le gyromètre
pente2 : 9,5 – pente de la fonction affine entre le rapport vitesse de roation et puissance du moteur



On constate que ce soit les mesures réalisées par l'encodeur optique ou le gyromètre, les données sont toujours proportionnelles à la puissance du moteur. On obtient également une quasi parfaite superposition des deux courbes ayant une pente similaire de 10 %. Mais on peut comprendre que le gyromètre étant plus précis que l'encodeur optique, quand réalité le moteur ne tourne pas plus vite après une puissance supérieure de 50 %. Il est donc inutile d'utiliser une puissance supérieure à 50 %, car on constate que la pente est la même, ainsi avec la formule PD on obtiendra les mêmes résultats en économisant de la batterie.

*Fichier **projet.c***

```
#pragma config(Sensor, S1, TouchSensor, sensorEV3_Touch)
#pragma config(Sensor, S2, GyroSensor, sensorEV3_Gyro)
#pragma config(Motor, motorA, moteurA, tmotorEV3_Large, PIDControl, encoder)

/*
 * @author: Addwyn Le lann m2 iliade
 */

// Constantes
#define PENTE 9.885
#define MAX_SPEED 860
#define MIN_POWER 5
#define MAX_POWER 87

#define PENTE_2 9.5
#define MAX_SPEED_2 494
#define MIN_POWER_2 9
#define MAX_POWER_2 52

#define P 100
#define D -1

#define CONSIGNE_GYRO 0

// Variables globales
int consigne = 0;
TSemaphore mutexConsigne;
int changement = false;
```

```
// Déclare le prototype la fonction "choix" en avance pour pouvoir faire des appels récursifs
void choix();

// Permet de limiter la vitesse du moteur à notre constante calculé avec l'encodeur optique
void launchMotorSpeed(int speed) {
  if(abs(speed) > MAX_SPEED) {
    speed = MAX_SPEED;
  }
  setMotorSpeed(motorA, speed/PENTE);
}

// Permet de limiter la vitesse du moteur à notre constante calculé avec le gyromètre
void launchMotorSpeed2(int speed) {
  if(abs(speed) > MAX_SPEED_2) {
    speed = MAX_SPEED_2;
  }
  setMotorSpeed(motorA, speed/PENTE_2);
}

// Fonction qui indique le cap à exécuter pour le moteur
task keepHeading() {
  int consigneCopy;
  while(1) {
    delay(50);
    // Sémaphore pour empêcher les lectures et écritues sur la même consigne
    semaphoreLock(mutexConsigne);
    // Création d'une copie de la consigne, pour relâcher le sémaphore plus vite
    consigneCopy = consigne;
    semaphoreUnlock(mutexConsigne);
    if(changement) {
      // Fait tourner le moteur à la consigne indiquée avec une puissance de 20
      setMotorTarget(motorA, consigneCopy, 20);
      // Indique que l'action a été exécutée
      changement = false;
    }
  }
}

// Fonction qui éxécute le cap avec la formule Vitesse = P(consigne-cap)+D(vitesseAngulaire)
task keepHeadingPD() {
  // Mise par défaut des différents capteurs
  resetMotorEncoder(motorA);
  resetGyro(S2);

  int consigneCopy;
  int vitesse;

  while(1) {
    delay(50);
    semaphoreLock(mutexConsigne);
    consigneCopy = consigne;
    semaphoreUnlock(mutexConsigne);
    if(changement) {
      // Calcule de la vitesse en fonction des constantes P et D
      vitesse = P * (consigneCopy - getMotorEncoder(motorA)) + D * getMotorRPM(motorA)*6;
```

```
      launchMotorSpeed(vitesse);
      setMotorTarget(motorA, consigneCopy, getMotorSpeed(motorA));
      changement = false;
    }
  }
}

// Fonction qui utilise la formule P et D provenant du gyromètre
task keepHeadingPD2() {
  resetMotorEncoder(motorA);
  resetGyro(S2);

  int vitesse;

  while(1) {
    delay(50);
    // Force le cap à être toujours dirigé vers 0
    vitesse = P * (CONSIGNE_GYRO - getGyroDegrees(S2)) + D * getGyroRate(S2);
    launchMotorSpeed2(-vitesse/10);
  }
}

// Fonction qui affiche une IHM indiquant les différentes actions possibles
task IHM() {
  eraseDisplay();
  displayString(1, "R : Turn motor 10d right");
  displayString(2, "L : Turn motor 10d left");
  displayString(3, "U : Turn motor 90d right");
  displayString(4, "D : Turn motor 90d left");
  //displayString(5, "Enter : End program");
  displayString(5, "Enter : Return to menu");

  // Delay pour actualiser l'écran sur les consignes
  while(1) {
    delay(100);
    displayString(7, "consigne : %d", consigne);
    displayString(8, "cap : %f", getMotorEncoder(motorA));
  }
}

// Fonction qui "écoute" les différentes touches préssées
task watchButtons() {
  while(1) {
    delay(50);

    // Tourne de 10 degrés vers la droite
    if(getButtonPress(buttonRight)) {
      semaphoreLock(mutexConsigne);
      consigne -= 10;
      semaphoreUnlock(mutexConsigne);
      changement = true;
      while(getButtonPress(buttonRight)){}
    }
```

```
    // Tourne de 10 degrés vers la gauche
    if(getButtonPress(buttonLeft)) {
      semaphoreLock(mutexConsigne);
      consigne += 10;
      semaphoreUnlock(mutexConsigne);
      changement = true;
      while(getButtonPress(buttonLeft)){}
    }

    // Tourne de 90 degrés vers la droite
    if(getButtonPress(buttonUp)) {
      semaphoreLock(mutexConsigne);
      consigne -= 90;
      semaphoreUnlock(mutexConsigne);
      changement = true;
      while(getButtonPress(buttonUp)){}
    }

    // Tourne de 90 degrés vers la gauche
    if(getButtonPress(buttonDown)) {
      semaphoreLock(mutexConsigne);
      consigne += 90;
      semaphoreUnlock(mutexConsigne);
      changement = true;
      while(getButtonPress(buttonDown)){}
    }

    // Quitte le mode en cours
    //if(getButtonPress(buttonEnter)) stopAllTasks();
    if(getButtonPress(buttonEnter)) {
      stopTask(IHM);
      stopTask(keepHeadingPD);
      choix();
      stopTask(watchButtons);
    }
  }
}

// Affiche une IHM pour le mode mobile
task IHM2() {
  eraseDisplay();
  //displayString(1, "Enter : End program");
  displayString(1, "Enter : Return to menu");

  while(1) {
    delay(100);
    displayString(3, "consigne : %d", CONSIGNE_GYRO);
    displayString(4, "cap : %f", getGyroHeading(S2));
  }
}

// Premet d'écouter les touches en mode mobile
task watchButtons2() {
  while(1) {
```

```
    delay(50);
    // Met fin au mode
    //if(getButtonPress(buttonEnter)) stopAllTasks();
    if(getButtonPress(buttonEnter)) {
      stopTask(IHM2);
      stopTask(keepHeadingPD2);
      launchMotorSpeed2(0);
      choix();
      stopTask(watchButtons2);
    }
  }
}

// IHM expliquant les touches pour dérouler le câble du gyromètre
void IHM_init() {
  eraseDisplay();
  displayString(1, "R : Turn motor right");
  displayString(2, "L : Turn motor left");
  //displayString(3, "Enter : End program");
  displayString(3, "Enter : Validate");
}

// Fonction qui init l'ensemble des modes
void initialize() {

  consigne = 0;
  changement = false;

  resetGyro(S2);
  resetMotorEncoder(motorA);
  setMotorSpeed(motorA, 0);

  IHM_init();
  // Permet de dérouler le câble
  while(1) {
    delay(50);

    // Tourne le moteur vers la gauche avec 20 de puissance
    if(getButtonPress(buttonLeft)) {
      setMotorSpeed(motorA, 20);
      while(getButtonPress(buttonLeft)){}
    }

    // Tourne le moteur vers la droite avec 20 de puissance
    if(getButtonPress(buttonRight)) {
      setMotorSpeed(motorA, -20);
      while(getButtonPress(buttonRight)){}
    }

    //if(getButtonPress(buttonEnter)) stopAllTasks();
    if(getButtonPress(buttonEnter)) {
      setMotorSpeed(motorA, 0);
      break;
    }
```

```
  }
}

// Permet de choisir entre les deux modes
void choix() {
  eraseDisplay();
  displayString(1, "R : Robot mobile");
  displayString(2, "L : Robot fixe");

  while(1) {
    delay(50);

    if(getButtonPress(buttonLeft)) {

      delay(200);
      initialize();
      delay(200);

      startTask(IHM);
      startTask(keepHeadingPD);
      startTask(watchButtons);

      break;
    }

    if(getButtonPress(buttonRight)) {

      delay(200);
      initialize();
      delay(200);

      startTask(IHM2);
      startTask(keepHeadingPD2);
      startTask(watchButtons2);

      break;
    }

  }
}

// Permet de quitter le programme à tout moment grâce au bouton poussoir
task watchStop() {
  while(1) {
    delay(20);
    if(getTouchValue(S1)) stopAllTasks();
  }
}

// Programme principal
task main() {
  resetMotorEncoder(motorA);
  resetGyro(S2);
  semaphoreInitialize(mutexConsigne);
```

```
  //startTask(keepHeading);

  startTask(watchStop);
  choix();

  while(1);
}
```

encodeur optique	gyromètre	9,89x	9.5x	x
0	0	0	0	0
0	1	9,89	9,5	1
0	1	19,78	19	2
0	4	29,67	28,5	3
30	8	39,56	38	4
70	7	49,45	47,5	5
50	4	59,34	57	6
70	5	69,23	66,5	7
70	70	79,12	76	8
80	73	89,01	85,5	9
100	108	98,9	95	10
100	126	108,79	104,5	11
110	131	118,68	114	12
120	165	128,57	123,5	13
140	148	138,46	133	14
140	161	148,35	142,5	15
150	182	158,24	152	16
160	197	168,13	161,5	17
180	201	178,02	171	18
170	213	187,91	180,5	19
190	224	197,8	190	20
200	239	207,69	199,5	21
220	240	217,58	209	22
220	257	227,47	218,5	23
230	275	237,36	228	24
240	255	247,25	237,5	25
250	324	257,14	247	26
260	273	267,03	256,5	27
270	299	276,92	266	28
290	282	286,81	275,5	29
280	288	296,7	285	30
300	291	306,59	294,5	31
320	304	316,48	304	32
320	346	326,37	313,5	33
320	352	336,26	323	34
350	328	346,15	332,5	35
360	391	356,04	342	36
350	350	365,93	351,5	37
380	364	375,82	361	38
380	394	385,71	370,5	39
390	386	395,6	380	40
410	374	405,49	389,5	41
420	377	415,38	399	42
410	408	425,27	408,5	43
430	430	435,16	418	44
450	376	445,05	427,5	45
450	384	454,94	437	46
460	402	464,83	446,5	47
480	417	474,72	456	48
490	422	484,61	465,5	49
490	448	494,5	475	50
510	494	504,39	484,5	51
510	494	514,28	494	52
530	494	524,17	503,5	53
530	494	534,06	513	54
550	494	543,95	522,5	55
550	494	553,84	532	56
560	494	563,73	541,5	57
580	494	573,62	551	58
580	494	583,51	560,5	59

600	494	593,4	570	60
600	494	603,29	579,5	61
620	494	613,18	589	62
620	494	623,07	598,5	63
640	494	632,96	608	64
640	494	642,85	617,5	65
660	494	652,74	627	66
660	494	662,63	636,5	67
680	494	672,52	646	68
690	494	682,41	655,5	69
690	494	692,3	665	70
710	494	702,19	674,5	71
710	494	712,08	684	72
730	494	721,97	693,5	73
740	494	731,86	703	74
760	494	741,75	712,5	75
740	494	751,64	722	76
770	494	761,53	731,5	77
770	494	771,42	741	78
790	494	781,31	750,5	79
800	494	791,2	760	80
800	494	801,09	769,5	81
820	494	810,98	779	82
830	494	820,87	788,5	83
830	494	830,76	798	84
850	494	840,65	807,5	85
860	494	850,54	817	86
870	494	860,43	826,5	87
840	494	870,32	836	88
850	494	880,21	845,5	89
860	494	890,1	855	90
850	494	899,99	864,5	91
860	494	909,88	874	92
860	494	919,77	883,5	93
850	494	929,66	893	94
860	494	939,55	902,5	95
870	494	949,44	912	96
840	494	959,33	921,5	97
860	494	969,22	931	98
860	494	979,11	940,5	99
850	494	989	950	100