

## Contraintes relatives aux fichiers gco.

Respecter quelques contraintes d'exploitation est impératif pour que le fonctionnement de la machine à graver soit correct. Ces contraintes résultent d'une optimisation maximale du programme opérationnel de la carte Arduino NANO ainsi que du format des fichiers fournis par le convertisseur d'images **LASER GRBL**.

### ► Contraintes sur le format des fichiers.

Diverses modifications sont impérativement à effectuer sur un fichier fourni par **LASER GRBL**, la liste sous forme d'une procédure ordonnée est développée ci-dessous :

- 1) À partir de l'image au format **bmp** créer le fichier **nc** en respectant le protocole fourni sur les fiches **Utiliser LASER GRBL v3.0.4**.
- 3) Quel que soit le nom donné au fichier **nc** lors de l'enregistrement, si nécessaire le renommer en **ImgN.gco** ou bien ce dernier sera ignoré par le logiciel d'exploitation du lecteur de carte **SD** sur Arduino NANO. Seuls les fichiers ayant pour identificateur **Img** seront reconnus, et à condition de **N** soit compris entre [0 et 9].

**On ne peut lister au maximum que dix images simultanément.**

En revanche, rien ne s'oppose à enregistrer sur la carte SD tout ce que l'on veut, que ce soit dans la racine ou dans des dossiers. Par exemple on peut archiver les fichiers **gco**, les dessins originaux etc.

- 4) Ajouter en tête de programme une ligne de **remarque qui précise quel est le fichier concerné** avec un **maximum de 25 caractères** le ';' et les espaces étant compris dans ce nombre.
- 5) En deuxième ligne du listage ajouter une **remarque qui précise le nombre de lignes exact du fichier**. Pour le connaître, commencer par introduire les deux premières lignes avec les ';' puis ouvrir le fichier avec un quelconque éditeur qui indique l'ordre des lignes de texte. (**RepetierHost** par exemple.) La dernière ligne étant visualisée, en noter l'ordre et le préciser en ligne n°2.

**REMARQUE :** Si les deux lignes de remarques ne sont pas ajoutées en tête du listage sur le fichier **gco**, le programme d'exploitation téléversé sur la carte Arduino NANO l'ignorera. Toutefois, des affichages d'informations destinées à l'opérateur seront absents.

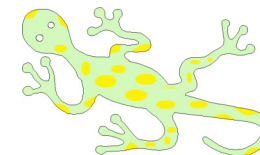
## Machine à pyrograver au LASER. Aspect LOGICIEL

Par Nulentout (Achevé le 20 Juin 2020.)

### TABLE DES MATÈRES

<i>Quelques détails d'écriture du programme.</i>	P02
<i>Logique de l'ordre des chapitres dans ce manuel.</i>	P03
<i>Problèmes posés par les "float" en C++.</i>	P04
<i>Mouvements : Problème du rapport DX / DY. 1/10</i>	P05
<i>Mouvements : Problème du rapport DX / DY. 10/10</i>	P14
<i>Déplacements dans toutes les directions. 1/5</i>	P15
<i>Déplacements dans toutes les directions. 5/5</i>	P19
<i>Explorer le fichier Image. 1/8</i>	P20
<i>Explorer le fichier Image. 8/8</i>	P27
<i>Constats et choix informatiques.</i>	P28
<i>Analyseur Syntaxique. 1/3</i>	P29
<i>Analyseur Syntaxique. 3/3</i>	P31
<i>Traitement des deux lignes G0 sans paramètre S.</i>	P32
<i>Procédure Affiche-etat-machine.</i>	P33
<i>Implantation des textes en EEPROM. 1/4</i>	P34
<i>Implantation des textes en EEPROM. 1/4</i>	P37
<i>Relations Cinématiques / Informatiques.</i>	P38
<i>Quelques aspects informatiques "en vrac".</i>	P39
<i>Contraintes relatives aux fichiers gco.</i>	P40

**IDE :** Environnement propre à la philosophie Arduino et qui permet d'en développer les programmes d'exploitation.



## Quelques détails d'écriture du programme :

- Globalement, les variables sont déclarées par types et en ordre alphabétique croissant. Elles sont ordonnées par tailles croissantes en occupation mémoire.
- Dans l'écriture du programme source, des lignes de "séparations de zones" sont utilisées globalement de la façon suivante :

```
//=====
```

Regroupe des procédures de nature analogues comme  
Affichages, Motorisation, gestion du LASER, servitudes ...

```
//+++++
```

Dans une procédure encadre une macro-séquence fonctionnelle regroupant un nombre significatif d'actions multiples.

```
//*****
```

Même finalité que ci-avant, mais pour un regroupement légèrement moins important. (Sous-division dans le listage.)

```
//-----
```

Au sein d'une procédure précise un groupement de quelques instructions servant une finalité commune. Ce sont les séparateurs statistiquement les plus nombreux.

- Une multitude de toutes petites procédures encombrant le programme. Elles sont systématiquement insérées pour optimiser la taille du programme OBJET dès qu'une séquence se répète et que la passer en procédure fait gagner des octets.
- Souvent, une procédure qui n'est utilisée qu'une seule fois et qui coûte quatre octets pour son appel peut sembler inutile. C'est pour des raisons de lisibilité qu'elle est extraite de sa procédure d'appel. Cette méthode est directement liée au critère suivant :
- Les longues procédures intégrant un grand nombre d'instructions sont difficiles à analyser. Elles sont évitées au maximum au profit de plusieurs procédures associées chacune à une finalité particulière, ces modules étant ensuite chaînés dans un traitement plus global n'étant alors constitués que d'appels fonctionnels.
- Chaque fois qu'une variable peut être "enfermée" en local dans une procédure, elle est retirée de l'espace global car il en résulte un gain d'octets systématique pour le corps du programme. *Les textes boulimiques en octets sont tous inscrits en EEPROM.*
- Quand c'est possible, une fonction ou une procédure fille est placée dans le programme *avant la procédure qui en fait l'appel.*

## Quelques aspects informatiques "en vrac":

- Pour minimiser le bruit par effet de résonnances issu du fonctionnement des moteurs pas à pas, la sélection du nombre de pas par tour est choisie à 1600 avec un coefficient de division de 8. (Voir la fiche *Pilote de moteurs pas à pas TB6600. 1/3.*) Ce choix issu d'un compromis expérimental entre nuisance sonore et rapidité des mouvements engendre des conséquences informatiques. Pour ne pas trop pénaliser la rapidité des déplacements, une précision de 0,5mm par axe sera imposée aux images. Ce choix simplifie considérablement le développement logiciel sans influence réelle sur la qualité de la pyrogravure.
- Gérer un lecteur de carte **SD** est boulimique en octets, car il faut prendre en compte un formatage issu de WINDOWS. Pouvoir naviguer dans des sous-dossiers, et utiliser des identificateurs quelconque serait trop pénalisant en taille de ressources englouties sans compter la gestion de l'affichage, et ce, au détriment de la qualité opérationnelle du programme d'exploitation. C'est la raison pour laquelle les identificateurs sont tous de la forme *ImgN.gco*, différenciés par le chiffre *N* allant de zéro à neuf. L'écran listant le contenu d'une carte ne pourra au maximum autoriser que dix images.

### ➤ *Stratégie pour gérer le LASER et les déplacements.*

- Quand la variable du LASER vaut 0% les déplacements se font à vitesse rapide. Quand la variable *Pourcentage* est différente de zéro les mouvements se font à vitesse lente.
- Le LASER de puissance ne s'allume que durant les déplacements quel que soit le mode d'utilisation de la machine, sauf pour l'option *LASER continu* du menu *SYSTEME*.

### ➤ *Gain de place pour le programme.*

- Initialement la mesure des tensions sur les CAN du CLAVIER résidait dans le menu *SYSTEME*. Pour gagner de la place, cette option d'une utilisation marginale a été intégrée dans le programme démonstrateur *P30\_pilotage\_par\_USB.ino* lui-même téléversé très occasionnellement durant la découverte du comportement du prototype et la validation de certaines fonctions.
- Si une évolution du programme imposait de dégager de la place, dans le menu *SYSTEME* l'option *Test capteurs ORG* n'est pas très utile et l'on peut s'en passer allègrement. Il serait dans ce cas opportun d'éliminer cette option vraiment marginale.

## Relations Cinématiques / Informatiques.

Globalement métallique, la structure de cette machine est bien plus bruyante que si du bois atténuant les vibrations était sélectionné pour concrétiser son ossature. Par nature les moteurs pas à pas, intrinsèquement d'un fonctionnement binaire, génèrent beaucoup de vibrations. Pour minimiser la nuisance sonore, 1600 pas par tour ont été ajustés sur les pilotes TB6600. Il en découle les éléments suivants :

- Poulie crantée du commerce : Étant spécifiquement prévue pour la réalisation de petites machines à commandes numériques, leur diamètre moyen est d'environ 12,73mm avec vingt dents. On obtient donc un développement de 40mm par tour du moteur : **Périmètre moyen  $\approx 12,73 \times 3.14159 \approx 39.992$  mm soit 40mm.**
- La définition possible sur le matériel sera de  $40 / 1600 \approx 0,025$ mm par incrément pour un nombre de 1600 incrément par tour.
- Pour que chaque pas commandé par logiciel, soit de 0,5mm la valeur d'optimisation du rapport Nuisance / Rapidité, le pilote TB6600 doit recevoir 20 impulsions. ( $20 \times 0,025 \approx 0,5$ mm.)
- Chaque appel à la procédure **Faire\_un\_pas\_sur\_X** ou **Y** devra générer 20 impulsions sur l'entrée **PUL+** du module TB6600.
- Le diamètre de la poulie motrice est précis, et sur un déplacement programmé de 205mm on ne constate visuellement aucun écart.

### ➤ Sens de rotation des moteurs.

Pour les moteurs 17HS4401 qui équipent le prototype, les deux sens de rotation seront corrects si l'on respecte les branchements de la Fig.3 proposée sur la fiche **Pilote de moteurs pas à pas TB6600. 2/3**. Si le moteur installé est d'une référence différente et qu'il tourne dans le mauvais sens, il suffit d'invertir **LOW** et **HIGH** dans les procédures **Faire\_un\_pas\_sur\_X** ou **Y** :

```
void Faire_un_pas_sur_X() {
  if (!STOP) {
    if (DX_Positif) {Ancien_X = Ancien_X + 5; digitalWrite(Sens_sur_X, LOW);}
    else {Ancien_X = Ancien_X - 5; digitalWrite(Sens_sur_X, HIGH);}
    for (byte l=0; l<20; l++) {digitalWrite(Un_pas_sur_X, HIGH); Delay_1mS();
    digitalWrite(Un_pas_sur_X, LOW);}
    if (!Mouvements_Rapide) delay(Delai_par_pas_elementaire);
    Teste_demande_sortie();}
}
```

Modification analogue à effectuer sur **Faire\_un\_pas\_sur\_Y** si c'est l'axe transversal qui est inversé pour son sens de rotation.

## Logique de l'ordre des chapitres dans ce manuel.

- **Problèmes posés par les "floats" en C++.**  
Précise une spécificité rencontrée par le compilateur en "2/8".
- **Mouvements : Problème du rapport DX / DY.** 1/8 à 8/8  
Analyse la logique des déplacements pas à pas pour effectuer des mouvements de type vectoriel pour une seule direction.
- **Déplacements dans toutes les directions.** 1/5 à 5/5  
Analyse la logique des mouvements effectués dans toutes les directions et quel que soit le sens des mobilités désirées.
- **Explorer le fichier Image.** 1/7 à 7/7  
Détaille la structure de **Traite\_ligne\_a\_ligne\_le\_fichier()** la procédure fondamentale qui sert à extraire les caractéristiques de l'image, (*Coordonnées Origine et nombre de lignes.*) et qui sert à **PYROGRAVER()** ou à **VERIFIER\_LE\_FICHIER()**.
- **Constats et choix informatiques.**  
Prépare le décryptage de l'agencement de l'analyseur syntaxique.
- **Analyseur Syntaxique.** 1/3 à 3/3  
C'est le cœur de l'interprétation des lignes extraites une à une dans **gcode** pour ensuite animer les systèmes de la machine.
- **Traitement des deux lignes G0 sans le paramètre S.**  
Analyse impérative pour comprendre le fonctionnement de la procédure fondamentale **Traite\_ligne\_a\_ligne\_le\_fichier()**.
- Procédure **Affiche-etat-machine.**  
Cette procédure présente le titre de l'image en cours de traitement, le numéro de la ligne analysée, les coordonnées du LASER, et son pourcentage d'utilisation. Cette procédure est utilisée par la fonction **ANALYSE** des fichier, et par **PYROGRAVER**. Bien que ce soit une procédure d'affichage, elle gère aussi l'économiseur d'écran et le mode PAS à PAS.
- **Relations Cinématiques / Informatiques.**  
Établit les relations géométriques entre les déplacements, le nombre de pas par tour sur les moteurs, et le diamètre des poulies.
- **Quelques aspects informatiques "en vrac".**  
Regroupe des informations relativement pertinentes qui ne sont pas directement spécifiques aux autres thèmes du livret.

## Problèmes posés par les "float" en C++.

Compte tenu de la façon dont sont codés les **float** par le compilateur d'Arduino, il en résulte des limites et des particularités d'exploitation qu'il faut prendre en compte en développement. Les nombres à virgule "flottante" peuvent prendre des valeurs aussi élevées que 3.4028235E+38 et aussi faibles que -3.4028235E+38. Ils sont stockés sur 4 octets soit 32 bits en mémoire.

**Les variables float ont seulement six à sept chiffres significatifs, cette limitation concerne l'intégralité de la partie entière et de la partie décimale incluses.**

ATTENTION : Avec le compilateur d'Arduino, les variables de type **double** augmentent la grandeur possible des données manipulées, **mais n'augmente pas le nombre de chiffres significatifs.**

### ► Comparaison sur des données de type float.

Mathématiquement parlant, les nombres réels ne sont pas exacts sur Arduino, et **peuvent donner des résultats étranges** quand ils sont comparés. Considérons un exemple où **Cumul** est un **float** : À l'instant où la comparaison est effectuée **Cumul** vaut 1.000 et la comparaison **if (Cumul > 0.999999)** donne **false** !

Pour parer ce type de difficulté, généralement on effectue un test sur la valeur absolue d'une différence effectuée entre les données et en vérifiant que cette dernière est inférieure à un très petit nombre.

Considérons la cas particulier rencontré dans le programme qui gère les mouvements vectoriels de la machine à graver pyrotechnique. La comparaison en cause devait s'écrire en toute logique : **if (Cumul > 1) { ...** et ce test ne fonctionne pas car effectué sur une variable de type **float**. La solution retenue est : **if (int(Cumul + 0.01) > 0) { ...**

Les opérations mathématiques sur les variables de type **float** sont plus lente que les calculs mathématiques sur les nombres entiers, et sont à éviter si elles s'intègrent dans une boucle doit s'exécuter à la vitesse maximale pour une fonction critique par exemple. Il est généralement préférable dans ce cas de convertir les calculs effectués en virgule flottante en calculs sur des nombres entiers.

## Implantation des textes en EEPROM. 4/4

La zone teintée en rose pastel correspond aux 148 octets actuellement non utilisés dans la mémoire non volatile du microcontrôleur. On peut aussi distinguer dans les zones repérées en vert clair, des textes non significatifs composés de suites de 'x'. Ce sont des "résidus" résultant de chaînes de caractères qui au cours du développement du programme ont été abandonnées. Pour optimiser l'utilisation de l'EEPROM, ces zones ont été "récupérées" par d'autres textes. De longueurs différentes, les "trous" restant disponibles sont alors mis en évidences par ces chaînes particulières.

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0512	a	i	:		S	e	c	D	o	n	n	é	e	s		
0528	i	m	a	g	e	P	o	i	n	t	e	r		O	R	G
0544		i	m	a	g	e	?	G	r	a	v	u	r	e		
0560	:		±		x	x	x	x	x	x		T	e	s	t	c
0576	a	p	t	e	u	r	s		O	R	G	U		s	u	r
0592		c	a	p	t	e	u	r		O	R	G	N	O	N	
0608	a	r	m	é		!	P	o	s	i	t	i	o	n		Y
0624	L	i	g	n	e		à		t	r	o	i	s		B	P
0640		L	i	g	n	e		à		d	e	u	x		B	P
0656		B	P		d	é	d	u	i	t		C	h	a	q	u
0672	e		B	.	P	.	/		1	0	0	1	0	s	U	
0688		p	a	r		B	.	P	.		C	L	A	V	I	E
0704	R	é		R	e	l	i	r	e		l	a		c	a	r
0720	t	e		S	D	.	F	o	c	a	l	i	s	e	r	
0736	l	e		L	A	S	E	R	.	M	o	d	e		L	A
0752	S	E	R		c	o	n	t	i	n	u	P	A	U	S	E

Fig.29

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0768		e	n		l	i	g	n	e		C	o	d	e		A
0784	l	l	u	m	e	r		l	'	é	c	r	a	n	.	x
0800	x	I	n	c	r	é	m	e	n	t		=		S	u	i
0816	t	e		?	D	é	g	a	g	e	m	e	n	t		p
0832	l	a	t	e	a	u	.	E	t	e	i	n	d	r	e	
0848	l	'	é	c	r	a	n	.	.	.	.	.	.	.	.	.
0864	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0976	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0992	.	.	.	.	.	.	.	.	.	.	.	.	.	.	V	e
1008	i	o	n	:		2	7		0	5		2	0	2	0	



## Implantation des textes en EEPROM. 3/4

Sur les deux copies d'écran de Fig.28 et Fig.29 le programme adopte un listage sous forme textuelle. Les caractères accentués sont alors mal affichés. Pour les mettre en évidence ces deux images ont donc été modifiées manuellement. La version du programme est située tout en "haut" de l'EEPROM. Ainsi on peut ajouter librement du texte sans avoir à la déplacer, facilitant ainsi l'élaboration des programmes.

```

ADRS 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000 I m g o u i - - - P b s u
0016 r l e L e c t e u r S D .
0032 C o d e i l l é g a l . L i g
0048 n e < P = X = Y =
0064 F I N d e p r o c é d u r e
0080 ? D é l a i p a r p a s
0096 : l s m m A f f i c h a g e
0112 ? E r r e u r d e l e c t u
0128 r e V a l e u r h o r s l i
0144 m i t e s P Y R O G R A V U R E
0160 E N C O U R S x x x : O
0176 U I : N O N U O r i g i n
0192 e X = U O r i g i n e
0208 Y = R é s o u d r e l e
0224 p r o b l é m e p u i s > > >
0240 R E S E T ! > C a p t u r
0256 e O R I G I N E < P b C a
0272 p t e u r O R G I N E ! P o
0288 s i t i o n X o u Y <
0304 0 . V a l e u r X o u Y
0320 > M A X . p o u r 1 0 0 .
0336 S y n t a x e i n c o r r e c
0352 t e . F i n e n l i g n e
0368 A N A L Y S E d u F I C H I
0384 E R M o d e M A N U E L . M o
0400 d e P A S à P A S : T
0416 a i l l e l i g n e s T m p
0432 A n a l y s e m i n O P T I O
0448 N S S Y S T E M E C h a q u e
0464 B . P . : M A X I P W L
0480 A S E R : p o u r 1 0 0
0496 M d f e n P Y R O . D é l

```

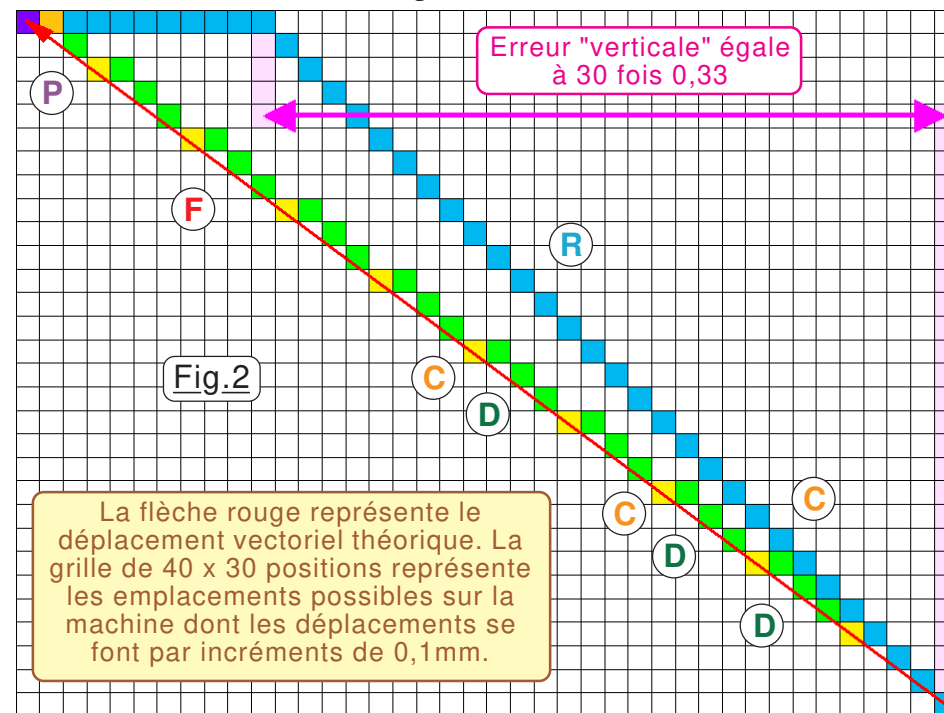
Fig.28

## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 1/10

Générer un tracé vectoriel impose de coordonner les mouvements sur X et Y en respectant "la pente". L'exemple de la Fig.1 est issu d'une copie d'écran et montre que pour la ligne de code 5154 d'un dessin, la "pente"  $DX / DY = 40 / 30 = \approx 1,33$ . Sur la Fig.2 le déplacement vectoriel théorique est représenté par la flèche rouge **F**. Les moteurs pas à pas par nature ne peuvent générer que des incréments "entiers" de grandeur  $\approx 0,1\text{mm}$  sur la graveuse

Fig.1 L = 5150 DeltaX = -230.00 DeltaY = -50.00 Rapport = 4.60  
L = 5154 DeltaX = -40.00 DeltaY = +30.00 Rapport = 1.33

pyrotechnique. Le rapport arrondi à l'entier le plus proche vaut 1. De ce fait, pour 1 pas effectué sur la plus grande des deux variations, soit X, on effectuera un pas sur la plus petite Y pour tenter de respecter la pente. On obtient le résultat **R** avec une inclinaison exagérée. La position finale **P** n'est atteinte ensuite que par des pas effectués sur X. Pour la ligne 5150 le résultat serait encore plus catastrophique, car au lieu de  $0,33 \times 30$  la perte à chaque incrément sera de  $0,6 \times 50$  suivi du déplacement résiduel "horizontal".



## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 2/10

L'erreur cumulée en ne tenant pas compte de la partie décimale de la valeur de la "pente" est inacceptable. Pour prendre en compte ce résidu, la technique logicielle possible est représentée sur la Fig.2 par le déplacement colorié en vert et en jaune. L'idée consiste à effectuer sur le déplacement le plus important une correction **C** régulièrement. (*Coloriée en jaune sur le croquis.*) Sur cet exemple, on effectue un déplacement **D** de trois pas, colorié en vert, puis un déplacement horizontal correctif **C**. Comme c'est le cas sur la Fig.2 il peut arriver que le déplacement le plus rapide étant terminé, (*Dans cet exemple sur la position coloriée en orange.*) sur le plus "lent" on ne soit pas exactement aux coordonnées d'arrivées théoriques, mais à proximité. On complète donc, systématiquement par une vérification de la position finale, suivie s'il le faut de déplacements jusqu'à atteindre la position ciblée.

### ➤ Technique pour optimiser la correction.

Plusieurs approches sont possibles pour traiter ce problème, en cherchant à aboutir à un code le plus compact possible. Dans le programme gérant la pyrograveuse, l'algorithme utilisé est le suivant :

- 01) Une variable **Decimale** est initialisée à **0,33** pour cet exemple.
- 02) On calcule une variable "incrémentale" **Rapport** en prenant uniquement *la partie entière* de **DX / DY** soit ici **3**.
- 03) On force à **false** un booléen nommé **Fini**.
- 04) On initialise à zéro une variable nommée **Cumul**.
- 05) On initialise à zéro une variable nommée **Compteur**.
- 06) **Compteur** inférieur à **Rapport** ?
  - OUI : Position X atteinte ?
  - NON : Faire un pas sur X > **Compteur++** > **Cumul = Cumul + Decimale** > Retour en 6
- 07) Faire un pas sur Y. *Ci dessous le test désiré est **Cumul > 1** ?*
- 08) **int(Cumul + 0.01) supérieur à 0 ?** @ (*Test sur un float.*)
  - OUI : Position X atteinte ?
  - NON : Faire un pas sur X > **Cumul = abs(Cumul - 1)**.
- 09) **Fini** = Position X atteinte. Si NON : retour en 5
- 10) Tant que Y non atteint : Faire un pas sur Y.

@ : Voir la fiche nommée **Problèmes posés par les "float" en C++**.

## Implantation des textes en EEPROM. 2/4

Lorsque **P01\_Ecrire\_les\_textes\_en\_EEPROM.ino** est téléversé, mis à part un délai avant la fin des échanges d'information sur la ligne USB il ne se passe rien de visuel. Aussi, lorsque l'ATmega328 est pourvu de ce programme, on valide le Moniteur de l'**IDE** qui alors affiche à l'écran la liste des textes inscrits en EEPROM suivi de l'affichage du contenu de l'intégralité de cette dernière. **ATTENTION : L'écran sera cohérent que si on adopte 115200 bauds en vitesse série** sur le Moniteur ou si l'on change la valeur dans l'instruction **Serial.begin(115200)**. La copie d'écran de la Fig.27 a été modifiée, car les accentués n'apparaissent pas sur l'écran du moniteur vidéo, phénomène également lié aux problèmes posés par les codages des caractères différents entre les divers acteurs informatiques impliqués dans ce domaine.

```

Img oui ---Pb sur le Lecteur SD.
Code illégal.Ligne <P = X = Y =
FIN de procédure ?Délai par pas : 1smm
Affichage ?Erreur de lectureValeur hors limites
PYROGRAVURE EN COURSxxx : OUI : NON
U Origine X = U Origine Y = Résoudre le
problèmepuis >>> RESET !
> Capture ORIGINE <Pb Capteur ORIGINE !
Position X ou Y < 0.Valeur X ou Y > MAX.
pour 100.Syntaxe incorrecte.
Fin en ligneANALYSE du FICHIER
Mode MANUEL.Mode PAS à PAS :
Taille lignesTmp Analyse min
OPTIONSSYSTEMEChaque B.P. : MAXI PW LASER :
pour 100Mdf en PYRO. Délai : Sec
Données imagePointer ORG image ?
Gravure : xxxxxxTest capteurs ORG
U sur capteur ORGNON armé !
Position YLigne à trois BP
Ligne à deux BP BP déduit
Chaque B.P. / 10010sU par B.P.
CLAVIER& Retire la carte SD.
Focaliser le LASER.Mode LASER continu
PAUSE en ligne Code Allumer l'écran.
xxIncrément = Suite ?Dégageant plateau.
Eteindre l'écran.
Version : 02 06 2020

```

"Résidus"  
non utilisés

Fig.27

## Implantation des textes en EEPROM. 1/4

Il impose pour qu'elle soit effective, et que les divers logiciels de démonstration et programmes d'exploitation puissent afficher des textes cohérents sur l'écran OLED, que le programme de servitude **P01\_Ecrire les textes en EEPROM.ino** soit téléversé avant tous les autres sur la carte Arduino NANO. S'il s'agit d'une carte neuve, ou que son EEPROM n'a encore jamais servie, il n'y a rien de spécial à faire. Si l'ATmega328 a déjà été modifié dans sa mémoire non volatile, il sera utile de valider la ligne de remarque // for (int i=0; i < 1024; i++) EEPROM.write(i, 255); qui commence par effacer intégralement la zone concernée. Ainsi les copies d'écran Fig.28 et Fig.29 seront conformes à ce que l'on doit obtenir. Cette ligne de remarque placée en tête de programme est facile à trouver car terminée par une suite de @@@@ @@@@ @@@@ @@@@.

### ➤ L'usage des accentués.

Outre la variété manifeste des polices de caractères disponibles, la bibliothèque **<U8glib>** qui accompagne l'afficheur OLED sélectionné pour cette application, et fournie dans le répertoire **<Les bibliothèques>**, cette "library" met à notre disposition les principaux accentués, et il serait bien dommage de ne pas en profiter. La Fig.26 présente la fonte sélectionnée pour le programme **P20\_Programme\_exploitation.ino** et précise en décimal ou en hexadécimal l'intégralité des codes pour chaque caractère disponible. Cette information est absolument indispensable car pour introduire des accentués dans l'EEPROM de l'ATmega328, il faut les définir comme des OCTETS et non les insérer dans les chaînes de caractères de l'IDE qui dans ce domaine a toujours présenté de petites difficultés.

**Fig.26**

```

u8g_font_proFont12, ProFont
BBX Width 6, Height 11, Capital A 8
Font data size: 2907
32/0x20  !"#$%&'()*+,-./
48/0x30  0123456789:;<=>?
64/0x40  @ABCDEFGHIJKLMNO
80/0x50  PQRSTUVWXYZ[\]^_
96/0x60  `abcdefghijklmnopqrstuvwxyz
112/0x70  pqrstuvwxyz{|}~
128/0x80  ,f...t^%$&'()*+,-./
144/0x90  ',"#%&'()*+,-./:;<=>?
160/0xa0  i j k l m n o p q r s t u v w x y z
176/0xb0  ò ó ô õ ö ø ù ú û ü ý þ ÿ
192/0xc0  À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
208/0xd0  Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
224/0xe0  à á â ã ä å æ ç è é ê ë ì í î ï
240/0xf0  ð ñ ò ó ô õ ö ø ù ú û ü ý þ ÿ
  
```

## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 3/10

Traduire l'algorithme de la fiche 2/4 en optimisant le code objet aboutit à la procédure décrite ci-dessous. Pour changer, on va prendre un exemple différent avec un **DeltaX = -40** et une valeur sur **DeltaY = +12**. Dans le démonstrateur on ne se préoccupe pas du sens des déplacements, mais uniquement du nombre d'incréments à effectuer sur les moteurs. La "pente"  $\Delta X / \Delta Y = 40 / 12 = \approx 3,33$ . **ATTENTION** : L'algorithme de la fiche 2/4 raisonne en positions X et Y à atteindre. La procédure et les explications de cette fiche sont relatives aux incréments effectués par les moteurs.

```

int DeltaX = 40; int DeltaY = 12; (Saisies au clavier dans P12)
boolean Fini; byte Compteur; int Rapport; // Peut aller jusqu'à 2250.
float Cumul, Decimale; int NbPX, NbPY; // Nombres de pas réalisés.
void Effectuer_un_mouvement_vectoriel() { (Valeur est utilisée ici pour
la clarté, mais dans la réalité c'est Cumul qui est employé pour optimisation.)
  
```

```

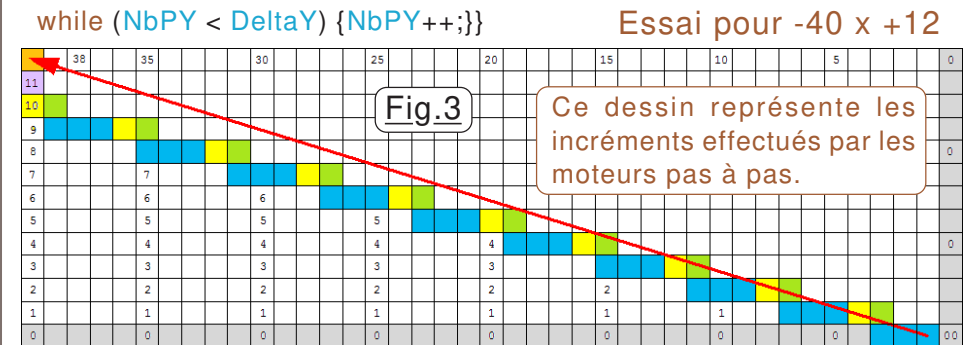
  Valeur = float(DeltaX) / float(DeltaY);
  Decimale = Valeur - ((int) Valeur); Rapport = Valeur - Decimale;
  Cumul = 0.0; NbPX=0; NbPY=0; Fini = false;
  
```

```

  while (!Fin) { Compteur = 0;
    while (Compteur < Rapport) {
      if (NbPX < DeltaX)
        {NbPX++; Compteur++; Cumul = Cumul + Decimale;}
      else Compteur = Rapport; // Forcer la sortie si NbPX = DeltaX.
    }
    NbPY++;
    if (int(Cumul + 0.01) > 0) {
      if (NbPX < DeltaX) {NbPX++; Cumul = abs(Cumul - 1);}
      if (Cumul > 1) Cumul = abs(Cumul - 1);}
    Fini = NbPX == DeltaX;
    while (NbPY < DeltaY) {NbPY++;}
  }
  
```

(Dans ce listage les instructions d'affichage sur le Moniteur USB sont enlevées.)

L'organigramme commenté est sur la fiche 5/8





## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 4/10

Quelques précisions s'imposent relatives aux contraintes opérationnelles imposées par la machine et au comportement de la procédure **Effectuer\_un\_mouvement\_vectoriel()**.

Les couleurs adoptées sur la Fig.4 correspondent à celles des déplacements colorés sur la Fig.3 avec mise en évidence des comparaisons qui engendrent des actions particulières. Par exemple en vert c'est le **Compteur** arrivé à la valeur de **Rapport** qui impose un pas sur Y. En jaune c'est la valeur de **Cumul** qui dépasse 1.0000 et qui impose de faire un pas de plus sur X.

*Lorsque le programme de type gco est exploité ligne à ligne, il importe de ne jamais laisser le LASER plusieurs fois de suite à la même position, car, s'il est actif, le "pixel" concerné sera plus prononcé que ses voisins.*

Dans ce but, il faut vérifier que la procédure de vectorisation respecte ce critère. Par exemple, sur la Fig.4 quelques "stagnations" sur X et sur Y permettent de s'assurer que lorsque un axe ne change pas de position, à chaque ligne l'autre est incrémenté. Par ailleurs, si le déplacement se fait uniquement sur X ou uniquement sur Y, le calcul de  $\Delta X / \Delta Y$  engendrerait une division par zéro. C'est la raison pour laquelle ces deux cas particuliers sont traités à part.

L'observation de la Fig.3 montre que plus on se déplace dans la direction la plus grande, plus le tracé passe "en dessous" du mouvement théorique. Il y aura forcément un "rattrapage" en "escalier" à la fin de la procédure. Il faut s'assurer en testant divers cas critiques que cette correction résiduelle reste acceptable pour des  $\Delta X$  égaux à 2250 et pour les pentes d'environ  $1^\circ$  à  $45^\circ$ .

>>> DeltaX = 40				
>>> DeltaY = 12				
DeltaX/DeltaY=3.333				
Rapport = 3				
Decimale = 0.333				
X	Y	C	Cml.	
01	00	1	0.3333	
02	00	2	0.6667	
03	00	3	1.0000	
03	01	3	1.0000	
04	01	3	0.0000	
05	01	1	0.3333	
06	01	2	0.6667	
07	01	3	1.0000	
07	02	3	1.0000	
08	02	3	0.0000	
09	02	1	0.3333	
10	02	2	0.6667	
11	02	3	1.0000	
11	03	3	1.0000	
12	03	3	0.0000	
13	03	1	0.3333	
14	03	2	0.6667	
15	03	3	1.0000	
15	04	3	1.0000	
16	04	3	0.0000	
17	04	1	0.3333	
18	04	2	0.6667	
19	04	3	1.0000	
30	07	2	0.6667	
31	07	3	1.0000	
31	08	3	1.0000	
32	08	3	0.0000	
33	08	3	0.0000	
34	08	2	0.6667	
35	08	3	1.0000	
35	09	3	1.0000	
36	09	3	0.0000	
37	09	1	0.3333	
38	09	2	0.6667	
39	09	3	1.0000	
39	10	3	1.0000	
40	10	3	0.0000	
40	11	3	0.0000	
40	12	3	0.0000	

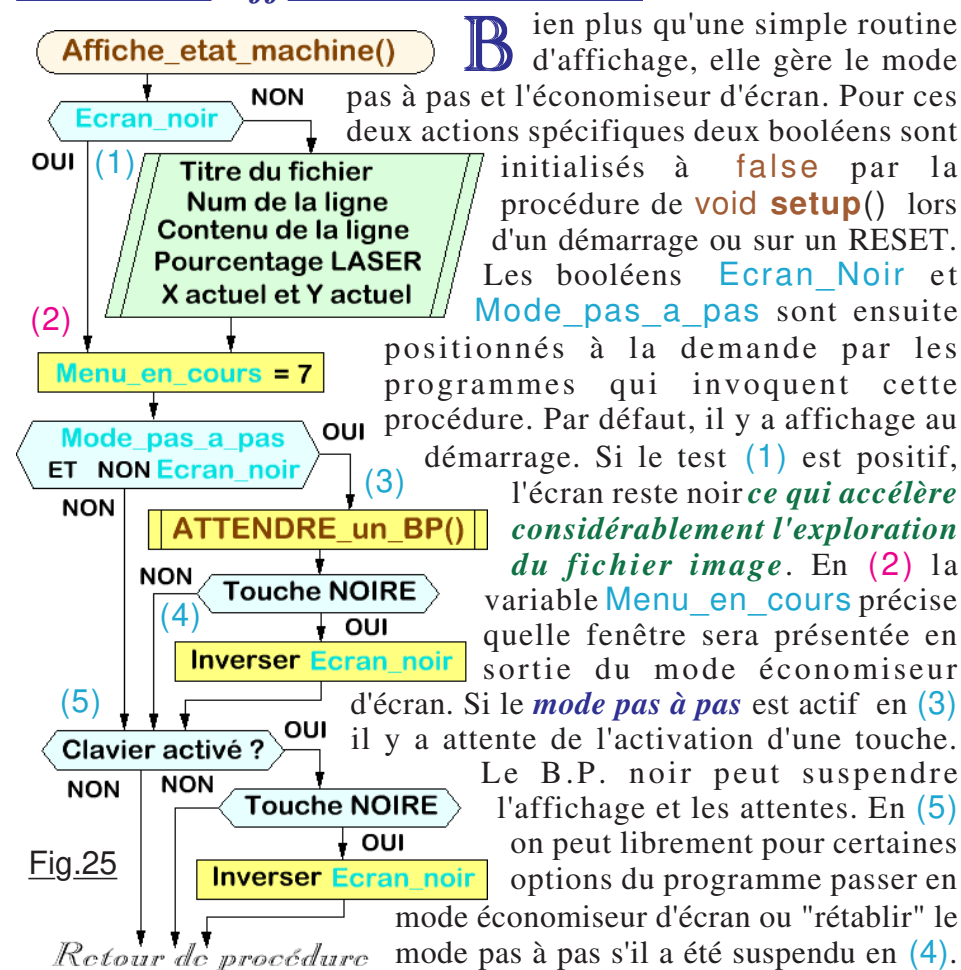
Fig.4

## Influence des procédures sur Lent / Rapide et sur STOP. P 33

Procédure	Mouvement		STOP	
	Entré	Sortie	Entrée	Sortie
Capure_origine_machine	---	Rapide	---	false
Position_degagee	---	Rapide	---	false
Degage_le_plateau	---	Rapide	false	false
Positionne_sur_Origine_Image	---	Rapide	---	false
FIN demandée manuellement	---	---	---	true

Traite\_G() >>>> si Go laisse à Lent si G1 laisse à Rapide

## Procédure Affiche-etat-machine.





### Traitement des deux lignes G0 sans paramètre S.

Deux fois dans un fichier *gcode* on rencontre une instruction de type **G0** qui ne précise que **X** et **Y**, et non suivis de **S**. La première se trouve au début du fichier pour *imposer le déplacement rapide sur l'origine de l'image*. La deuxième en fin de listage pour *ramener le LASER en origine machine*. Pour ces deux singularités il ne faut pas que `Traite_G()` ne tente d'extraire la valeur de **S**, car une boucle infinie figerait alors définitivement le programme.

#### ► PYROGRAVURE() ou Vérification du fichier image.

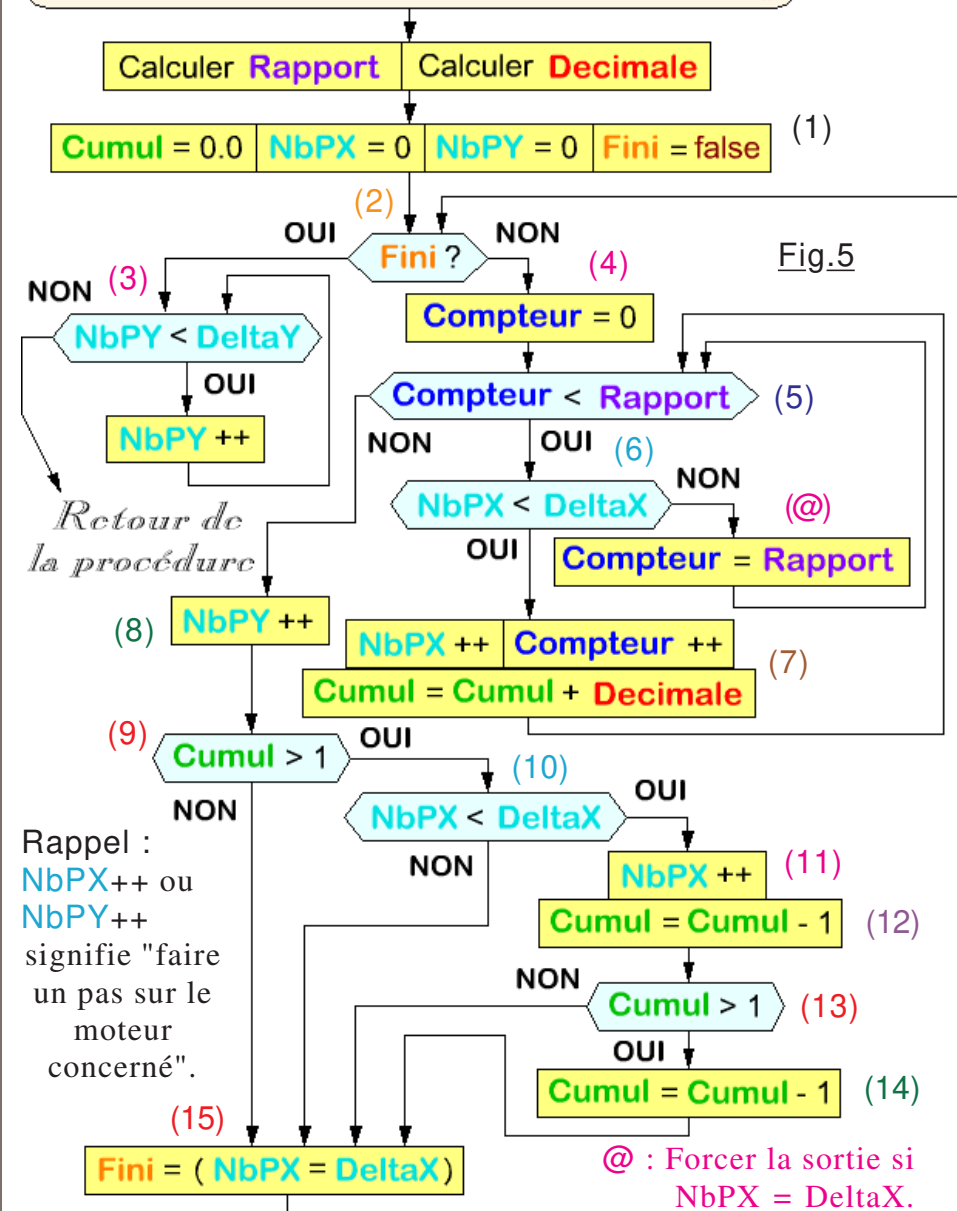
Logique oblige, le fichier image sera entièrement exploré. La procédure `Traite_ligne_a_ligne_le_fichier()` commence par initialiser à **true** la variable `Ligne_Premier_G0`. Puis les lignes sont analysées les unes après les autres jusqu'à trouver la première qui commence par **G0**. La procédure `Traite_G()` est alors invoquée. Examinant `Ligne_Premier_G0` elle en déduit qu'il faut mémoriser les valeurs de **X** et de **Y** dans les coordonnées de l'*Origine Image*, et surtout ne pas chercher à traiter le paramètre **S** qui n'est pas présent. La variable `Ligne_Premier_G0` est alors positionnée à **false** pour ne plus tenter d'extraire l'*Origine Image*.

En tête de procédure `Traite_ligne_a_ligne_le_fichier()` le booléen `Ligne_M5_trouvee` est initialisé à **false**. Puis, lorsque la ligne de *gcode* **M5** est rencontrée, cette variable passe à **true**. Arrivé à la fin du programme, la ligne **G0 X0 Y0** est rencontrée. La procédure `Traite_G()` teste alors si `Ligne_M5_trouvee` est à **true**. Si simultanément `Longueur_ligne_de_code` compte exactement huit caractères, c'est qu'il ne faut pas traiter le paramètre **S** car c'est la dernière instruction pour pyrograver l'image. Dans ce cas, *On se contente d'imposer les coordonnées correspondant à "machine dégagée"* soit `Position_X = 10.0` et `Position_Y = 192.0` par appel de la procédure `Prepare_degagement()`. Le début du traitement de **G0** a imposé la valeur 0 à la variable `Pourcentage`. De ce fait, le déplacement à l'origine machine logicielle sera effectué en vitesse rapide en sortie de l'*Analyseur\_Syntaxique()* lors son l'appel par `Traite_ligne_a_ligne_le_fichier()`. Ce fonctionnement ne sera garanti que si la structure le fichier image respecte le protocole d'utilisation de *GRBL* précisé dans la fiche nommée *Utiliser LASER GRBL v3.0.4* pour générer du code *gcode* qui sera conforme.

### Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 5/10

Représenter l'algorithme des fiches 2/8 et 3/8 sous la forme d'un organigramme aboutit au dessin Fig.5 donné ci-dessous.

**void Effectuer\_un\_mouvement\_vectoriel ()**



## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 6/10

Observant l'organigramme de la Fig.5 on peut avoir l'impression que certaines instructions sont redondantes. Par exemple le test (13). En réalité, cet algorithme est issu de nombreux essais pour tenir compte de tous les cas particuliers envisageables.

Dans cette simulation, NbPX et NbPY représentent les nombres d'incréments effectués par les moteurs. Une instruction telle que NbPX++ ou NbPY++ signifie que le moteur de l'axe concerné effectue un pas en respectant le sens du déplacement. *(La valeur est donc augmentée sans tenir compte du sens de déplacement.)*

- En (1) on initialise le contexte à "aucun mouvement n'est fait".
- En (2) on teste pour savoir si l'intégralité du mouvement dans la direction X a été effectuée. Si c'est le cas, le booléen Fini est forcé à true en (15). Si ce n'est pas le cas, on va effectuer des déplacements sur X en respectant le Rapport. Donc en (4) on commence par indiquer qu'aucun pas n'est encore effectif. *(Dans le programme définitif ce sera X ou Y, le mouvement dans la direction du déplacement le plus important.)*
- En (5), tant que l'on aura pas atteint le Rapport on effectue un pas en (7), à condition toutefois en (6) que le nombre total à effectuer ne soit pas réalisé. La valeur de Compteur est augmentée pour en tenir compte et on ajoute à Cumul le déficit Decimal.
- Quand en (5) on a satisfait le Rapport, en (8) on effectue un pas sur l'axe Y. *(Ou X si dans le PGM définitif NbPY > NbPX.)* Puis en (9) on regarde si le "déficit" dépasse la valeur d'un pas. Si en (10) le déplacement total n'est pas terminé, alors en (11) on fait un incrément de plus et en (12) on met à jour la valeur de Cumul.
- Sous certaines conditions, après avoir réalié (12) la valeur de Cumul peut encore dépasser un. Il est impératif de le vérifier en (13), et d'enlever encore une unité en (14) si c'est le cas.
- Après avoir traité globalement un groupe de déplacements pour aboutir à Rapport on teste en (15) pour savoir si c'est Fini.
- Si c'est le cas, alors (2) est satisfait, et en (3) on va effectuer autant d'incréments qu'il le faut dans la direction du déplacement qui n'est pas forcément satisfait jusqu'à terminer le mouvement.

(9) est codés if (int(Cumul + 0.01) > 0) : Voir la fiche spécifique nommée *Problèmes posés par les "float" en C++*.

## Analyseur Syntaxique. 3/3

Tester le bon fonctionnement de l'analyseur syntaxique impose l'exploration de fichiers corrects directement issus du logiciel GRBL et d'images modifiées pour générer des erreurs typiques. Pour générer un "Problème de lecture carte SD" il suffit de débrancher provisoirement GND du lecteur pendant le test. Les programmes spécifiques pour tester l'analyseur syntaxique sont les suivants :

Img0.gco : Test rapide sur fichier correct. (39 lignes.)

La puissance du LASER augmente progressivement.

Img1.gco : Logo en gris avec origine 30 et 40,5. (354 lignes.)

Img2.gco : Format A4 réduit à 202 mm de haut.

Exige 5101 lignes.

Img3.gco : X = 235.1 en ligne 12. (29 lignes.)

Img4.gco : Y = 225.1 en ligne 20. (30 lignes.)

Img5.gco : Non effective pour tests de visualisation.

Img6.gco : Y = -1.3 en ligne 9. (29 lignes.)

Img7.gco : X = -123 en ligne 23. (33 lignes.)

Img8.gco : E0 pour générer une erreur ligne 25. (37 lignes.)

Img9.gco : Grille de 202 x 228mm. (92.066 lignes.)

Img0.gco reste à 50 sur X pour deux lignes Y10 et Y20.5 et sortie rapide avec une analyse positive. Les deux dessins Img1.gco et Img2.gco (Taille petite et un grande) sont des images réelles converties par GRBL correctes avec uniquement les deux premières lignes de remarques ajoutées manuellement. Outre Img5.gco qui génère juste un avertissement sonore, les fichiers Img3.gco à Img8.gco engendrent rapidement une erreur typique que l'analyseur syntaxique doit détecter. Enfin, Img9.gco ne comporte pas d'erreur, mais impose la plus longue analyse potentielle. Sa vérification pour 92.066 lignes exige environ 51 minutes soit environ 30 lignes testées par seconde en moyenne quand la motorisation n'est pas activée.

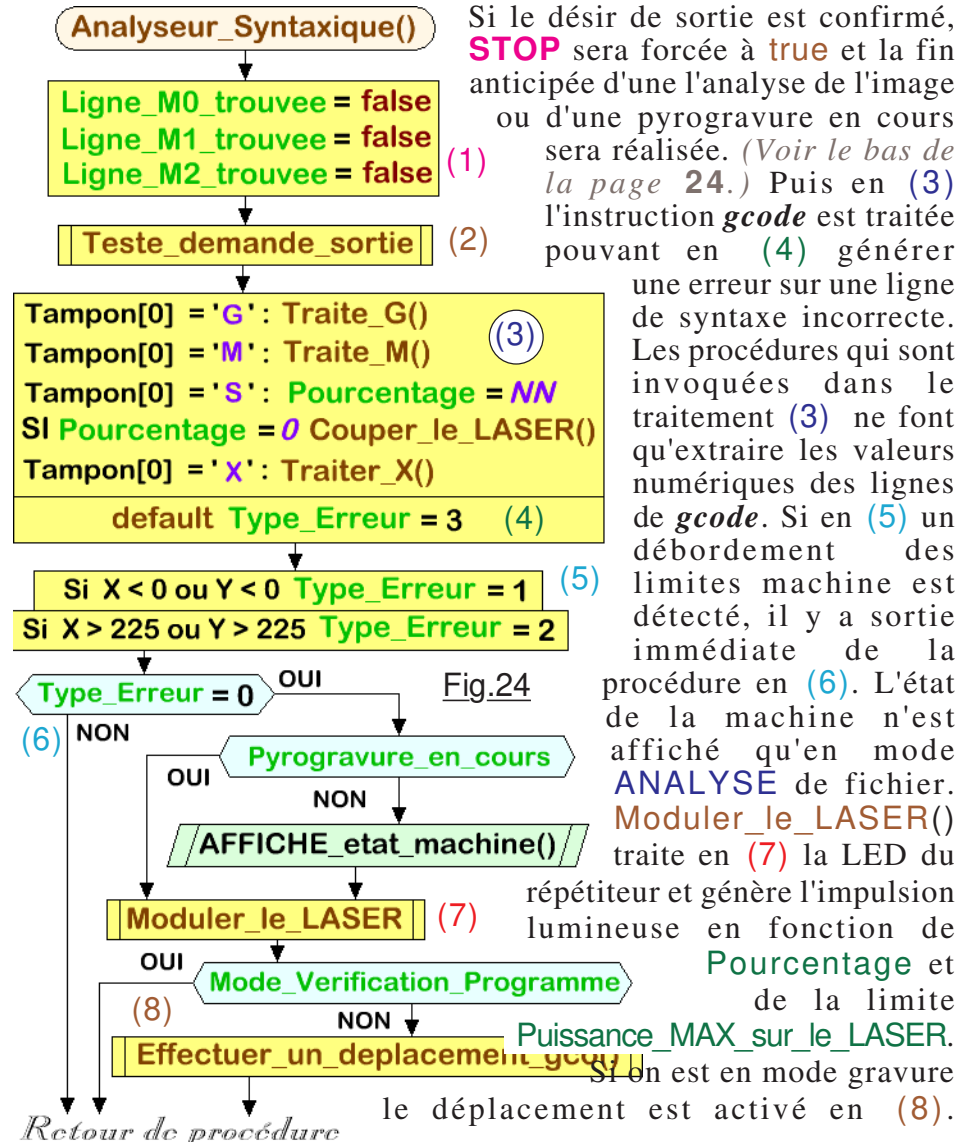
*Le codeur rotatif fait sortir à tout moment si on confirme.*

### LISTE DES ERREURS OPÉRATIONNELLES.

- ERR 1 : Ligne contenant un X ou un Y négatif.
- ERR 2 : Ligne contenant un X ou un Y dépassant la limite MAXI.
- ERR 3 : Ligne non vide non commencée par ';', 'F', 'G', 'M' ou 'X'.
- ERR 4 ou 6 : Problème de lecture du fichier sur la carte SD.
- ERR 5 : Problème sur les capteurs d'Origine machine.

## Analyseur Syntaxique. 2/3

Présenté sur la Fig.24 l'**Analyseur Syntaxique** commence en (1) par initialiser à **false** les trois lignes de codes particulières au programme d'exploitation de la pyrograveuse. Puis en (2) on teste si l'opérateur a sollicité une sortie prématurée par activation du B.P. central du C.I. Si c'est le cas, alors **BP\_central\_a\_traiter** est forcé à **false** puis il y a appel à **Demande\_FIN\_de\_procedure()**.



## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 7/10

### ► Tester la procédure **Effectuer\_un\_mouvement ...**

Avant d'écrire une procédure générale qui englobe tous les mouvements possibles, en préambule il faut s'assurer que la routine qui gère les déplacements vectoriel se comporte correctement dans divers cas de figures typiques. Pour le faire, on doit téléverser le démonstrateur **P12\_simuler\_des\_mvts\_vectoriels.ino** indépendant, qui dialogue avec l'opérateur par le truchement du Moniteur de l'**IDE**. On peut facilement tester toutes les combinaisons souhaitables en se limitant aux contraintes précisées en tête du programme. Le tableau ci-dessous propose un jeu d'essais initial :



DeltaX	DeltaY	$\Delta X / \Delta Y$	Remarque.
0	2250	---	Déplacement nul sur X.
2250	0	---	Déplacement nul sur Y.
40	12	3.333	Test de la fiche 3/8.
100	52	1.923	Décimal proche de 1.
1	1	1.000	Déplacement minimal possible.
2250	1	2250	N'incrémente que X et Y à la fin.
2250	2	1125	N'incrémente que X et Y à la fin.
2250	5	450.00	Pente très faible.
2250	10	225.00	Pente très faible.
2250	20	112.50	Pente faible.
2250	35	64.286	Pente $\approx 0.65^\circ$ .
2250	70	32.143	Pente $\approx 1,3^\circ$ .
2250	140	16.071	Pente $\approx 2,6^\circ$ .
2250	281	8.007	Pente $\approx 5,6^\circ$ .
2250	562	4.004	Pente $\approx 11.25^\circ$ .
2250	843	2.669	Pente $\approx 16.9^\circ$ . (@)
2250	1125	2.000	Pente $\approx 22.5^\circ$ .
2250	1406	1.600	Pente $\approx 28.1^\circ$ . (@)
2250	1687	1.334	Pente $\approx 33.75^\circ$ . (@)
2250	1968	1.143	Pente $\approx 39.4^\circ$ . (@)
2250	2247	1.001	Pente $\approx 45^\circ$ .
2250	2249	1.000	Pente $\approx 45^\circ$ .
2250	2250	1.000	Déplacement Maximal possible.

(@) : Pas de déplacement sur Y en fin de mouvement sur X.




## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 8/10

### ► Accepter l'imperfection pour optimiser à outrance.

L'une des difficultés la plus épineuse du défi que représente ce projet réside dans le fait que l'on doit faire entrer un éléphant dans une boîte d'allumettes, c'est à dire arriver à gérer tous les périphériques de la machine : Le clavier, l'écran, les moteurs, le lecteur de carte SD, avec uniquement 30720 octets de programme. Aussi, si l'on désire pouvoir "caser" un système d'exploitation de qualité, il faut rogner partout où c'est possible. La gestion des déplacements fait partie des entités "sacrifiées". Expérience : Quand la machine sera opérationnelle, graver le fichier  **Img3.gco** dont le titre est "**Img3 SOLEIL !**". Cette image programmée à la main a pour but de vérifier que le programme de déplacement vectoriel fonctionne correctement quelle que soit la direction et le sens du mouvement à effectuer. Copie d'écran effectuée dans le logiciel  RepetierHost la Fig.6 montre **les déplacements tels qu'ils sont programmés** dans le fichier **gco**. Les rayons vecteur sont régulièrement espacés angulairement, les écarts sur le périmètre font tous 10mm.

Mouvement sans graver pour passer de l'origine de la machine à l'origine de l'image dans

 RepetierHost.

Par exemple le déplacement surligné en vert clair aboutit au point B qui se trouve à égale distance des jalons A et C

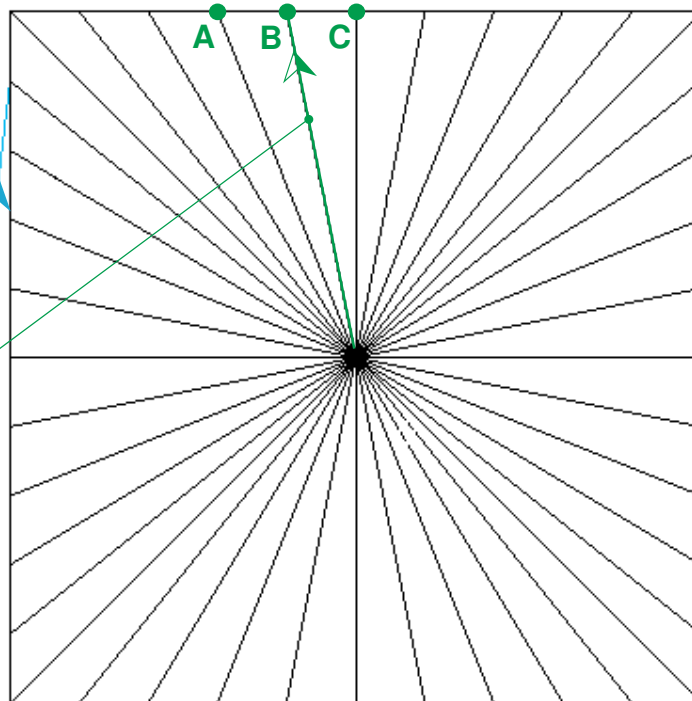



Fig.6

## Analyseur Syntaxique. 1/3

Réspecter le protocole d'utilisation de GRBL précisé dans la fiche nommée **Utiliser LASER GRBL v3.0.4** est un impératif pour générer du code **gco** qui sera intégralement accepté par l'analyseur syntaxique.

**Dans ces conditions, les seules formes possibles pour les lignes à traiter sont :**

- Manuellement on doit ajouter TITRE en ligne 1 et la taille en ligne 2.
  - En ligne 4 et 5 des commentaires ont été ajoutés avec un éditeur de texte.
  - En ligne 6 la commande G90 qui initialise le positionnement absolu, sera ignorée. (*Fonctionnement par défaut de la machine à graver.*)
  - En ligne 7 : Le déplacement rapide sur l'**origine de l'image** est traité normalement. C'est la seule ligne de type G0 **ainsi que celle à la fin** qui ne contient pas Snn.
  - En ligne 8 : LASER ON à 0%.
  - En ligne 9 : Ne contient que S0.
  - En ligne 10 : F1000 est ignorée.
  - Lignes analogues à la ligne 11, 19 : Instructions de type Xnnn.n Snn
  - Lignes analogues à la ligne 21 : Déplacements rapide à 0% de type G0 Xnnn.n Ynnn.n S0
  - Lignes analogues à la ligne 13, 17 : Déplacements lents à n% de type G1 Xnnn.n Sn
  - Les fichiers **gco** générés se terminent par une ligne vide. (*Ici en ligne 39.*)
  - GRBL ne génère aucun commentaire.
-  .n est soit le chiffre 0, soit 5.

```

1 ; Img0 Test rapide.
2 ; 39 lignes.
3
4 ; La puissance LASER
5 ; augmente par pas.
6 G90
7 G0 X50 Y10
8 M3 S0
9 S0
10 F1000
11 X50 S5
12
13 G1 X225 S10
14
15 ; Remarque 1.
16
17 G1 X50 S15
18 G1 X50 S20
19 X50 S25
20 X50 S30
21 G0 X50.0 Y20.5 S0
22 X50 S35
23 X50 S40
24 X50 S45
25 X50 S50
26 ; Remarque 2.
27 X50 S55
28 X50 S60
29 X50 S65
30 X50 S70
31 X50 S75
32 X50 S80
33 X50 S85
34 X50 S90
35 X50 S95
36 X50 S100
37 M5
38 G0 X0 Y0
39

```

On peut ajouter librement des lignes vides ou des remarques.

Fig.23

## Constats et choix informatiques.

**T**ributaire du contenu des fichiers fournis par **LASER GRBL**, la stratégie consiste à paramétrer le traitement pour optimiser le code **nc** fourni et simplifier au maximum le traitement effectué sur Arduino. Contrairement à ce que l'on pourrait penser, **même en mode ligne à ligne il y a des déplacements "vectorisés"**. Utiliser du balayage ligne à ligne horizontal ou vertical n'est pas un critère qui évitera d'avoir à coder des mouvements coordonnés en XY sur la machine.

### ► Utiliser les codes de GRBL v3.0.4.

**P**ar nature, une graveuse LASER reste une machine relativement simple car par conception il n'y a que deux directions de déplacements et pas de changements d'outils. De ce fait le nombre de commandes en **gcode** reste limité. La génération de fichiers par **LASER GRBL** ne contient que les commandes suivantes :

- G90 : Utiliser le positionnement absolu.
- G0 X Y : Déplacement rapide. **Pas de S sur le premier G0. (1)**
- S0 : Puissance nulle sur le LASER.
- M3 S0 : Allumer le LASER à la puissance 0 exprimée en %.
- F1000 : Déplacer à la vitesse V. (Valeur en mm/min)
- G0 X Y S : Déplacement rapide. (1)
- G1 X S : Déplacement à vitesse V. (Y inchangé.) (1)
- X S : Déplacement rapide. (Y inchangé.) (1)
- M5 : Éteindre le LASER.

La fin du programme se termine par une ligne vide.

**NOTE** : Le dernier déplacement est toujours G0 X0 Y0 pour imposer un retour sur l'origine image. Il ne comporte pas de **Snn**.

Le paramètre **S** sera compris entre 0 et 100 et conditionnera sur le programme d'exploitation d'Arduino une puissance du LASER qui variera entre nulle et maximale.

Par sécurité, si le listage est modifié manuellement, toute ligne de commentaire commençant donc par le caractère ';' sera ignorée.

Le logiciel d'exploitation ignorera le G90 car les déplacements sont toujours indiqués par les coordonnées à atteindre en ligne droite à partir de la position actuelle. La vitesse sera gérée par **Snn**.

(1) X et Y sont les coordonnées car les déplacements sont tous effectués en mode absolu par la commande initiale G90.

## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 9/10

Sur la Fig.7 le résultat obtenu en gravant sur du carton est bien différent. Visiblement les angles entre les fuyantes ne sont pas identiques, loin s'en faut. L'écart constaté a pour origine la façon dont est traitée la procédure de tracé vectoriel. Comme montré sur la Fig.2 on se contente de calculer "l'escalier" avec des marches identiques. Puis, comme à l'arrivée la position programmée n'est pas atteinte, on rattrape de "résidu" par un déplacement cartésien. Pour l'exemple de la Fig.7 le déplacement programmé est en vert clair, celui réellement effectué est en jaune, puis le résidu entre B et C est mis en évidence en rouge. Cette imperfection serait intolérable sur une machine à commande numérique, il faudrait impérativement répartir le "résidu" tout le long du mouvement. C'est faisable, mais cette solution a été abandonnée car bien trop consommatrice de code binaire. Hors, **sur la pyrograveuse** nous savons que **les mouvements avec gravure seront** soit **horizontaux**, soit **verticaux**. La simplification adoptée est parfaitement justifiée.

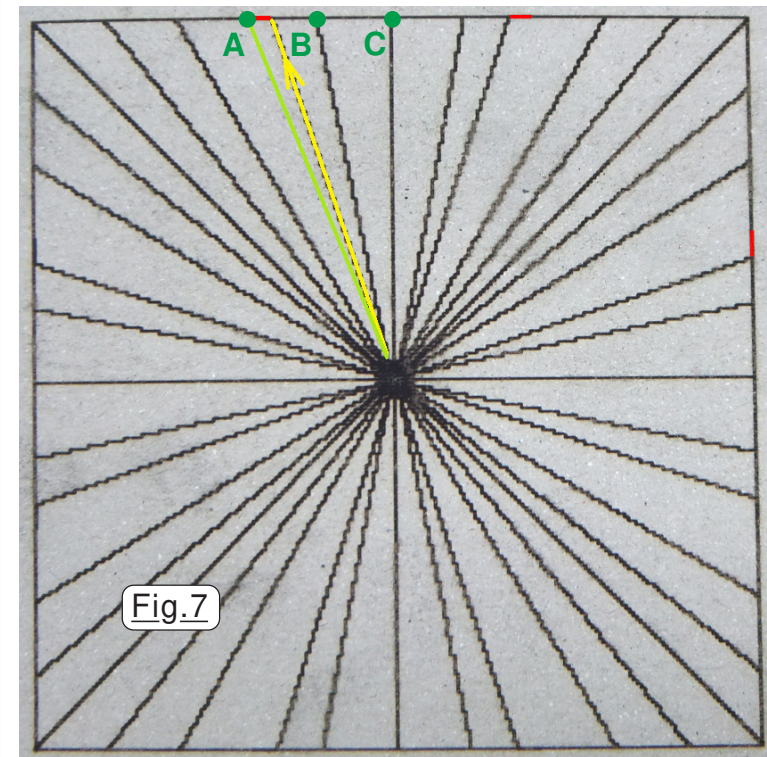


Fig.7

Img3.gco est écrit pour interdire que le LASER ne grave jamais deux fois sur les mêmes "PIXELS". Ce n'est pas très visible sur la Fig.7, mais il y a bien deux passages pour la gravure des résidus. C'est le programme vectoriel d'exploitation d'Arduino qui les provoque.



## Mouvements : Problème du rapport $\Delta X / \Delta Y$ . 10/10

Pire des cas : Une erreur de programme engendre une boucle infinie dans laquelle le microcontrôleur est définitivement piégé. Par exemple un cas particulier engendre un test de sortie qui ne sera plus satisfait dans une boucle de type `while`. Plus rien ne se passe sur la machine. Si au moment où se produit l'incident le LASER est à pleine puissance, sa cible du genre carton ou bois va alors prendre feu. *La seule façon de parer ce risque consiste à tester l'intégralité de la combinatoire des déplacements possibles sur la machine*. Si la sortie de ce test est effective, alors on peut considérer que le programme est fiable ce qui n'empêche pas :

- ☠ Que ce soit pour des raisons mécaniques, électronique ou informatique, l'arrêt des mouvements alors que le LASER est à pleine puissance n'est jamais totalement exclu. C'est la raison pour laquelle **ON NE LAISSE JAMAIS UNE PYROGRAVEUSE EN SERVICE SANS LA SURVEILLER**. *Toute personne dans le local doit porter des lunettes de protection contre le faisceau du LASER qui équipe la machine*. Si on va dans une pièce voisine, **un détecteur d'incendie est OBLIGATOIRE !**

Avec P13\_Tester\_tous\_les\_mvts\_vectoriels.ino qui est un autre démonstrateur indépendant, on va effectuer logiciellement tous les déplacements possibles en testant l'intégralité de la combinatoire. Un seul "quadrant" est pris en compte puisque `DeltaY` supérieur à `DeltaX` est interdit. C'est sans importance puisque la procédure travaillera en "absolu" sans tenir compte des signes. Le comportement du logiciel est montré sur la copie d'écran donnée ci-contre. Si la sortie à 44258 tests se produit, la procédure est validée. À 115200 bauds il faut moins de deux heures pour réaliser ce test.

```
DeltaX = 0   DeltaY = 0
DeltaX = 1   DeltaY = 0
DeltaX = 1   DeltaY = 1
DeltaX = 2   DeltaY = 0
DeltaX = 2   DeltaY = 1
DeltaX = 2   DeltaY = 2
DeltaX = 2   DeltaY = 2
DeltaX = 2250 DeltaY = 2242
DeltaX = 2250 DeltaY = 2243
DeltaX = 2250 DeltaY = 2244
DeltaX = 2250 DeltaY = 2245
DeltaX = 2250 DeltaY = 2246
DeltaX = 2250 DeltaY = 2247
DeltaX = 2250 DeltaY = 2248
DeltaX = 2250 DeltaY = 2249
DeltaX = 2250 DeltaY = 2250
Nb total de verif. = 44258
```

## Explorer le fichier Image 8/8

### ➤ Intervention des diverses variables.

Outre celles qui sont directement présentes dans la procédure, il faut aussi prendre en compte celles qui aiguillent les traitements de `Traite_ligne_a_ligne_le_fichier()` par l'entremise des séquences invoquées. La Fig.22 donne la liste de quelques variables "internes" à la procédure. La première en violet signale quand elle vaut `true` que si `G0` est détecté, c'est le premier de tout le fichier. Quand `Traite_G()` decode un `G0`, si `Ligne_Premier_G0` vaut `true`, alors les coordonnées qui suivent sont mémorisées dans les variables `Origine_Image_X` et `Origine_Image_Y`.

`Ligne_Premier_G0 = true;`  
`Ligne_M5_trouvee = false;`  
`Type_Erreur = 0;`  
`Fin_demandee = false;` Fig.22

Quand l'instruction `M5` est trouvée, le booléen `Ligne_M5_trouvee` est forcé à `true`. À partir de cette ligne, le programme sait alors que la prochain `G0` qui sera rencontré ne contiendra pas `Snn`, c'est celui de la fin du fichier image, et il ne faut pas chercher à extraire la valeur numérique de `Snn`.

Si `Type_Erreur` devient supérieur à zéro, c'est qu'un problème a été rencontré, déclenchant alors une sortie prématurée de la boucle. La variable `Num_Ligne` sert à `AFFICHE_etat_machine()` mais également à extraire le `Titre` et la `Taille` du fichier si la remarque a été ajoutée manuellement en tête de l'image. Cette information est utilisée pour évaluer globalement la durée que prendra la vérification de la syntaxe d'un fichier image.

Quand `Fin_demandee` est égal à `true`, c'est qu'un problème de lecture de la carte `SD` a été rencontré ou que l'opérateur a demandé manuellement une sortie anticipée. `PYROGRAVER()` ou `VERIFIER_LE_FICHER()` invoquent alors `Traiter_PB_Syntaxe()` qui `AFFICHE_etat_machine()` pour situer la ligne en cause.

La variable "externe" `STOP` est forcée à `true` quand l'opérateur valide une sortie anticipée ce qui peut arriver à chaque pas effectué lors d'un déplacement. Si `Delai_par_pas_elementaire` est notable, finir un déplacement long serait laborieux. `STOP = true` empêche alors les procédures `Faire_un_pas_sur_X()` et `Faire_un_pas_sur_Y()` d'actionner les moteurs. Il y a par conséquent sortie rapide du traitement de la ligne de `gcode` en cours.



## Explorer le fichier Image 7/8

### ➤ Traite\_ligne\_a\_ligne\_le\_fichier()

Traite\_ligne\_a\_ligne\_le\_fichier()

Initialiser les variables concernées

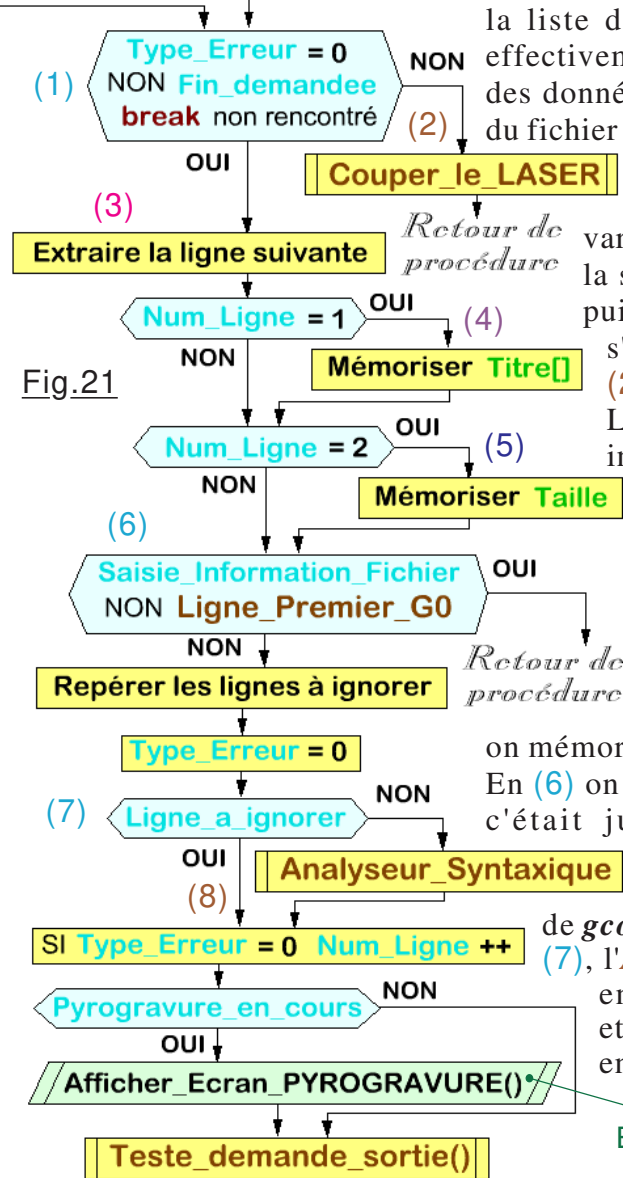


Fig.21

Lorsque la procédure de la Fig.21 est invoquée, le fichier image indexé dans la liste des entités valides est effectivement ouvert et le début des données y est pointé. La fin du fichier engendre un **break**. On débute en initialisant les nombreuses variables impliquées dans la séquence de la boucle, puis en (1) le test vérifie s'il ne faut pas sortir en (2) et couper le LASER. La liste des variables initialisées est présentée dans le tableau de la Fig.22 en chapitre 8/8. La ligne de **gcode** suivante est extraite en (3) suivie en (4) de l'extraction du **Titre**. Puis en (5) on mémorise la **Taille** du fichier. En (6) on sort de la procédure si c'était juste une saisie des caractéristiques de l'image. Si la ligne de **gcode** est significative en (7), l'**Analyseur Syntaxique** en (8) traite l'instruction et réalise les actions qui en découlent.

Affiche si NON  
Ecran\_noir ou invoque  
Eteindre\_Ecran()

## Déplacements dans toutes les directions. 1/5

Présentée dans **Mouvements : Problème du rapport  $\Delta X / \Delta Y$**  la procédure **Effectuer\_un\_mouvement\_vectoriel()** ne prend en compte que la grandeur des déplacements vectoriels sans se préoccuper du sens et en imposant aux essais un **DeltaY** inférieur ou égal à **DeltaX**. Il faut maintenant traiter tous les cas possibles.

- L'ensemble du traitement est basé sur l'hypothèse que chaque pas effectué par un moteur engendre un déplacement de 0,1mm. Bien que peu probable, pour le cas où les poulies motrices seraient légèrement différentes sur les deux axes, un coefficient de correction est prévu pour chaque direction.
- Pour faciliter le traitement informatique et pouvoir utiliser au maximum des variables de type **int**, les déplacements et les coordonnées seront gérés en dixièmes de millimètres.
- Passer des données par adresse (*Au lieu de variables globales*) diminue systématiquement la taille du code objet généré par le compilateur. C'est la raison pour laquelle un maximum de variables transitent sous forme de paramètres dans les diverses procédures.
- Par rapport à l'écriture simplifiée utilisée dans **P12**, dans cette nouvelle version, **Effectuer\_un\_mouvement\_vectoriel()** ne fait qu'inverser l'ordre des mouvements si **DeltaY** est supérieur à **DeltaX**. Ce sont les procédures **Faire\_un\_pas\_sur\_X()** et **Faire\_un\_pas\_sur\_Y()** qui se chargent de gérer le sens de déplacement et qui mettent à jour **Ancien\_X** et **Ancien\_Y**.

### ➤ Tester P14\_Simuler\_des\_Déplacements.

Vérifier **Effectuer\_un\_mouvement\_vectoriel()** se fait en deux étapes. La première consiste à téléverser le programme démonstrateur sur Arduino sans le modifier. Puis, par l'entremise du clavier, proposer aléatoirement des coordonnées à atteindre sur "toute la surface théorique" que peut balayer le LASER de la machine. Vérifier que les positions affichées soient égales à dix fois celles saisies. La deuxième étape consiste à valider **Affiche\_Position()** à la fin des routines **Faire\_un\_pas\_sur\_X()** et **Faire\_un\_pas\_sur\_Y()**. Puis passer en remarque **Affiche\_Position()** qui se trouve à la fin d'**Effectuer\_un\_deplacement\_gco()**. Le jeu d'essai de base consiste à tester toutes les combinaisons du tableau proposé dans la fiche 7/8, puis de recommencer en échangeant les valeurs de X et de Y.

## Déplacements dans toutes les directions. 2/5

Comme c'était le cas lors de la première écriture du sketch **Effectuer\_un\_mouvement\_vectoriel()** en fiche 8/8, il importerait de démontrer que cette procédure fonctionne pour tous les cas potentiellement possibles. Il n'est pas réalisable de tester "en surface" la totalité des mouvements envisageables, car la combinatoire explose, et leur nombre devient bien trop grand. On va se contenter d'une validation partielle qui englobe déjà une grande majorité de cas typiques. Le programme **P15\_Tester\_tous\_les\_deplacements.ino** se charge de ce travail. Il inclut plusieurs options disponibles pour afficher le déroulement de certaines séquences au prix d'un temps d'exécution bien plus important. La totalité des vérifications, comme le présente la Fig.8 issue d'une copie d'écran, comporte huit séquences typiques.

### ➤ Vérification par balayage circulaire.

Agencé pour couvrir sur 360 degrés la totalité de la surface possible de gravage de la machine, comme décrit sur la Fig.9 la technique consiste à tracer des rayons dont l'origine est systématiquement sur le centre du plateau. On part du coin arrière

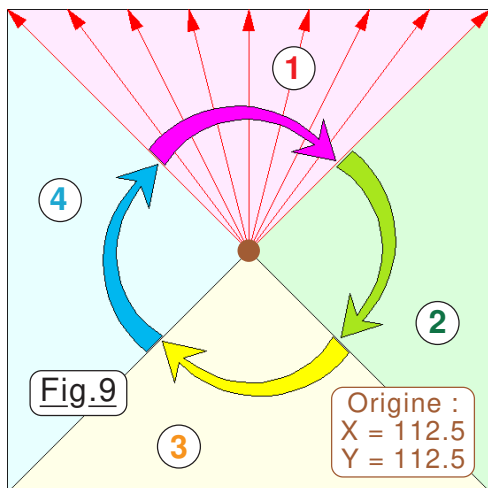


Fig.8

```

Debut de la simulation.
Nb total de verifications : 2250
Nb total de verifications : 4500
Nb total de verifications : 6750
Nb total de verifications : 9000
Nb total de verifications : 9719
Nb total de verifications : 10438
Nb total de verifications : 12688
Nb total de verifications : 14938
FIN de la simulation.
  
```

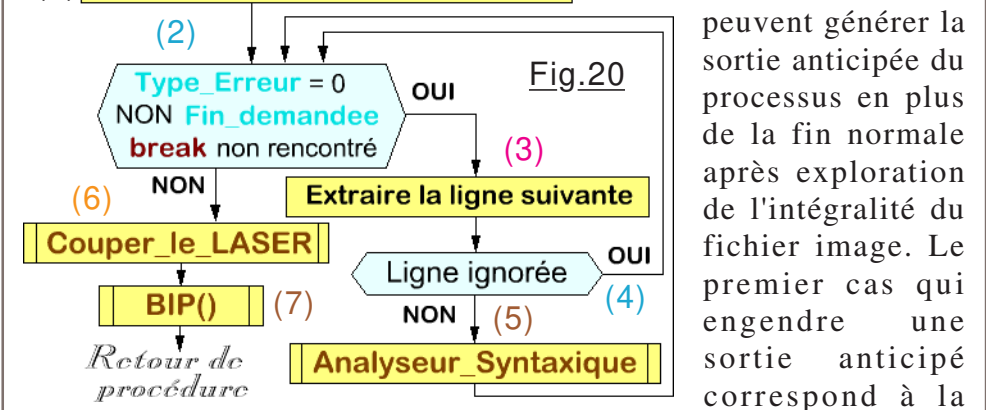
gauche, le rayon "tournant" arbitrairement dans le sens horaire par incréments de 0,5°. Balayer la totalité du carré de 225mm x 225mm exige les quatre séquences qui sur la Fig.1 sont coloriées en jaune et repérées sur la Fig.9 par des couleurs différentes. Les mouvements sont tous simulés en partant du centre vers le cadre limite. (Voir les quelques vecteurs symboliques rouges.)

## Explorer le fichier Image 6/8

### ➤ Traite\_ligne\_a\_ligne\_le\_fichier()

C'est incontestablement la procédure la plus conséquente du programme d'exploitation car elle doit filtrer un grand nombre de cas, que ce soit dans le contenu du fichier ou par les interventions de l'opérateur. La Fig.20 présente de façon épurée la structure globale de cette procédure fondamentale qui en (1) commence par initialiser les nombreuses variables intervenant durant les

Traite\_ligne\_a\_ligne\_le\_fichier() traitements effectués. En (2) le test vérifie s'il faut continuer. Plusieurs cas



peuvent générer la sortie anticipée du processus en plus de la fin normale après exploration de l'intégralité du fichier image. Le premier cas qui engendre une sortie anticipé correspond à la détection d'une erreur de syntaxe par (5) ou d'un problème de lecture de la carte **SD**. Le deuxième évènement correspond à une requête de l'opérateur qui désire achever le processus sans aller jusqu'à son terme. C'est dans ce cas particulier que la variable **STOP** sera forcée à **true**. La boucle (3) à (4) peut aussi rencontrer des instructions de type **break**. (Fin de fichier ou ligne G0 trouvée.) Chaque ligne extraite en (3) est analysée pour savoir si elle doit être prise en compte, le test (4) autorisant alors son traitement en (5) par l'**Analyseur Syntaxique**. C'est ce dernier qui en mode pyrogravure activera la motorisation et le LASER à condition toutefois qu'il ne rencontre pas une quelconque erreur durant sa séquence de traitement pour la ligne de **gcode** examinée. En (6), pour raison de sécurité, en sortie de procédure le LASER est coupé. Noter que si l'analyseur syntaxique en (5) détecte une erreur, il en affiche la nature et déclenche un BIP sonore qui double celui en (7) qui signale la fin du traitement.

## Explorer le fichier Image 5/8

Que ce soit mode vérification de fichier ou en pyrogravure, le module qui traite des erreurs affichera le type d'erreur rencontré en (5) puis l'état de la machine en (6) si un problème est rencontré en (1). En (6) on précise l'ordre et le contenu de la ligne provoquant la sortie anticipée. Si le problème se produit durant la pyrogravure, vérifié en (2), le plateau de la machine sera dégagé en (4). *La sortie d'un fichier correct ne sera commentée en (3) qu'avec le mode vérification.*

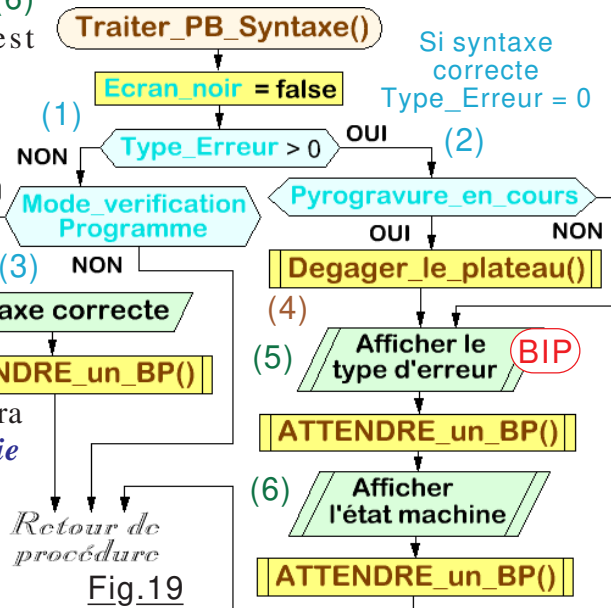


Fig.19

La procédure de capture des origines machine ne vérifie pas le bon fonctionnement des capteurs. Si par exemple l'un des deux est défectueux, faisant croire que la cellule photorésistante est masquée, le logiciel déduira à tort que l'origine est capturée. Si au contraire la ligne du capteur est coupée, le moteur va amener le chariot concerné en butée physique et engendrer un forçage mécanique. C'est la raison pour laquelle par défaut le programme commence par vérifier le bon fonctionnement des deux capteurs ce qui suppose que dans la pratique la machine soit en configuration dégagée. Si un mauvais fonctionnement est détecté le programme se contente d'allumer la LED triple rouge. Normalement c'est uniquement le bleu qui est allumé, signalant que les moteurs sont en VEILLE. *Si la LED tripe s'allumine en "violet" à la mise sous tension ou sur RESET, l'opérateur doit dégager manuellement les deux capteurs et éventuellement vérifier leur bon fonctionnement dans de menu SYSTEME.*

## Déplacements dans toutes les directions. 3/5

### ➤ Vérification par tracé de disques.

assez proche du protocole de la fiche 2/5, la technique consiste à tracer des disques circulaires centrés sur le plateau. Deux tests repérés en bleu pastel sur la Fig.6 sont effectués. Le premier consiste à tracer virtuellement, comme montré sur la Fig.10 ci-dessous,

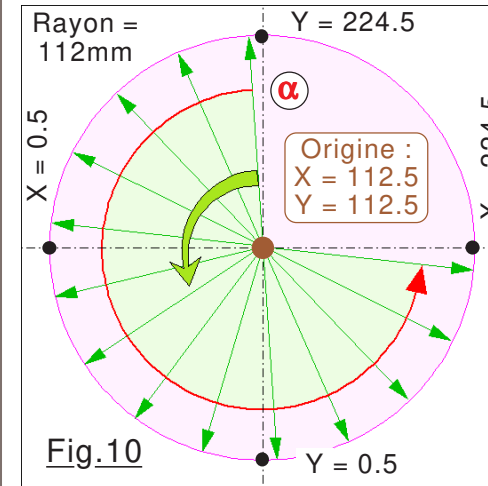


Fig.10

un disque complet dans le sens arbitraire horaire inverse. Le disque coloré en rose clair sur ce dessin est construit avec un rayon de 112mm tous les 0,5° et les mouvements sont tous commencés au centre comme symbolisé par les vecteurs verts. Le deuxième test, toujours avec des variations angulaires de 0.5° trace un disque bien plus petit de 1mm de rayon, également centré sur le plateau de la graveuse.

### ➤ Calcul des positions situées sur le périmètre.

Déterminer les coordonnées de l'extrémité des rayons balayant le disque relève d'une trigonométrie élémentaire facile à coder en C++ comme le prouve le listage proposé ci-dessous. Dans cette procédure, les déplacements virtuels ne sont pas encore codés.

```
void Teste_le_grand_disque_centrales() {
  ① float Angle = 359.5;
  ② for (int l=0; l < 720; l++) {
    ③ Position_X = 112.5 + (112 * sin(radians(Angle)));
    ④ Position_Y = 112.5 + (112 * cos(radians(Angle)));
    ⑤ Serial.println(); Serial.print("Angle = "); Serial.print(Angle,1);
    ⑥ Serial.print(" X = "); Serial.print(Position_X,1);
    ⑦ Serial.print(" Y = "); Serial.print(Position_Y,1);
    ⑧ Angle = Angle - 0.5;
    ⑨ Nb_de_tests_realises++;
    ⑩ Affiche_Nb_tests();
  }
```

Fig.11



## Déplacements dans toutes les directions. 4/5

En tenant compte de l'origine de l'angle  $\alpha$ , on calcule les coordonnées avec  $X = R \sin(\alpha)$  et  $Y = R \cos(\alpha)$  calculés respectivement en ③ et ④. Attention : Ces formules ne sont correctes que si l'Angle est exprimé en radians d'où la correction. Pour balayer un tour complet, dans

la boucle **for** en ② on impose  $(360 / 0.5) = 720$  pas. En ① on commence presque à la verticale pour terminer par l'angle  $\alpha = 0$ . Par défaut, dans le programme les affichages ⑤, ⑥ et ⑦ sont en remarques. Chaque séquence compte en ⑨ et affiche en ⑩ le nombre de vecteurs tracés. Quand on valide les lignes de code ⑤, ⑥ et ⑦, on obtient le résultat de la Fig.12 extraite d'une copie d'écran, lorsque le moniteur affiche les positions calculées en fonction des Angles. Après chaque vecteur traité, on diminue de  $0,5^\circ$  la valeur de l'Angle  $\alpha$  en ⑧.

Lors de la vérification du comportement, à chaque pas effectué la valeur de l'Angle  $\alpha$  doit diminuer de  $0,5^\circ$ . X et Y doivent varier entre les valeurs minimales 0.5 et maximales 224.5 qui sont à vérifier pour les angles particuliers de  $270^\circ$ ,  $180^\circ$ ,  $90^\circ$  et zéro. Ces quatre positions situées sur les axes de symétrie du plateau de la machine ont été mises en évidence en jaune sur la Fig.12 confirmant le comportement attendu de la procédure.

Fig.12

Debut de la simulation.

Angle = 359.5	X = 111.5	Y = 224.5
Angle = 359.0	X = 110.5	Y = 224.5
Angle = 358.5	X = 109.6	Y = 224.5
Angle = 358.0	X = 108.6	Y = 224.4
Angle = 357.5	X = 107.6	Y = 224.4
Angle = 357.0	X = 106.6	Y = 224.3
Angle = 356.5	X = 105.7	Y = 224.3

Angle = 271.0	X = 0.5	Y = 114.5
Angle = 270.5	X = 0.5	Y = 113.5
Angle = 270.0	X = 0.5	Y = 112.5
Angle = 269.5	X = 0.5	Y = 111.5
Angle = 269.0	X = 0.5	Y = 110.5

Angle = 181.5	X = 109.6	Y = 0.5
Angle = 181.0	X = 110.5	Y = 0.5
Angle = 180.5	X = 111.5	Y = 0.5
Angle = 180.0	X = 112.5	Y = 0.5
Angle = 179.5	X = 113.5	Y = 0.5

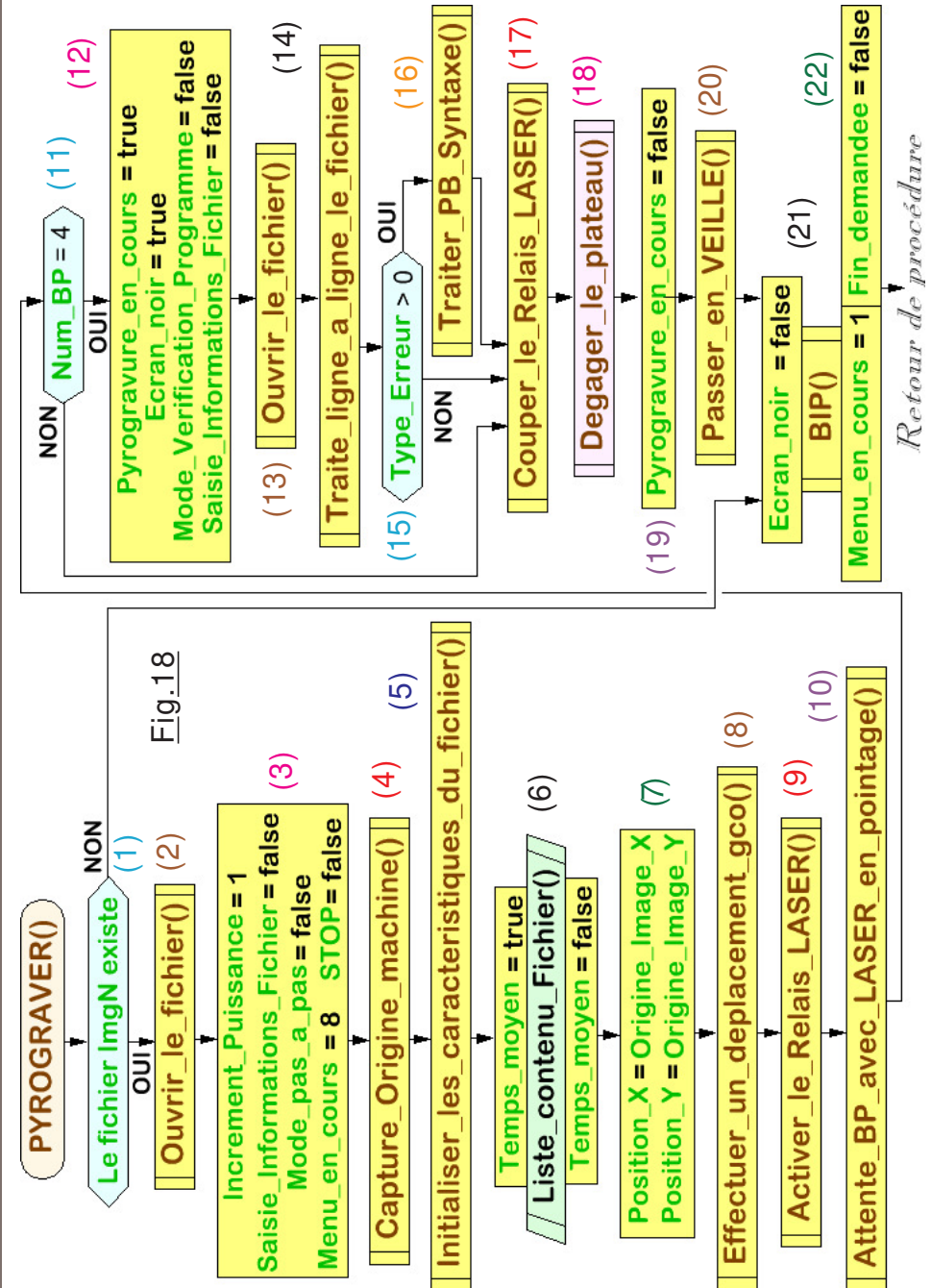
Angle = 91.0	X = 224.5	Y = 110.5
Angle = 90.5	X = 224.5	Y = 111.5
Angle = 90.0	X = 224.5	Y = 112.5
Angle = 89.5	X = 224.5	Y = 113.5
Angle = 89.0	X = 224.5	Y = 114.5


Angle = 2.0	X = 116.4	Y = 224.4
Angle = 1.5	X = 115.4	Y = 224.5
Angle = 1.0	X = 114.5	Y = 224.5
Angle = 0.5	X = 113.5	Y = 224.5
Angle = 0.0	X = 112.5	Y = 224.5

Nb total de verifications : 720  
FIN de la simulation.

## Explorer le fichier Image 4/8



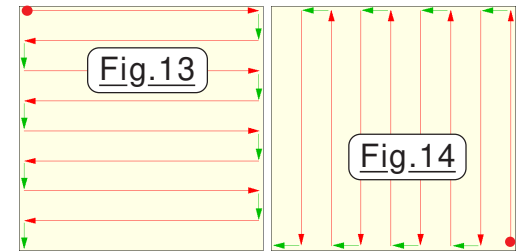
## Explorer le fichier Image 3/8

**A** vant de balayer ligne à ligne le **gcode**, vérifier en (1) que le fichier est bien présent sur la carte **SD**. Puis en (2) on fait appel à la procédure de la Fig.15 pour ouvrir le fichier *et éventuellement générer un message d'erreur si un problème est rencontré*. En (3) on précise qu'à ce stade la sortie manuelle n'est pas demandée, et on interdit le **Mode pas à pas**. En (4) les moteurs sont activés en mouvements rapides pour **déterminer la position mécanique définissant l'Origine Machine**. Il n'y a sortie de cette procédure que si les deux origines 0X et 0Y sont acquises optiquement. Consulter l'encadré en bas de page 5/8 précisant le comportement du logiciel en cas de problème rencontré sur les capteurs. En (5) on examine les premières lignes de l'image pour en afficher les caractéristiques en (6). C'est le **Temps\_moyen** estimé pour la gravure qui sera précisé. En (7) les coordonnées de l'origine sont transférées dans la position à atteindre et en (8) le LASER y est déplacé. En (9) le Relais du LASER est activé pour pouvoir en (10) effectuer la focalisation sur la cible et positionner cette dernière sur le plateau. Durant cette phase l'opérateur peut vérifier sur l'écran si c'est la bonne image qui a été indexée. Pour passer à la suite l'opérateur doit frapper une touche. (*LED verte qui clignote.*) Toute touche autre qu'**Y+ ↑** en (11) engendrera la fuite. En (12) on empêche l'affichage des instructions **gco** sur l'écran et OLED est éteint pour en augmenter la durée de vie. Puis en (13) on ouvre à nouveau le fichier pour enchaîner en (14) sur **le traitement ligne à ligne de l'intégralité du fichier avec une sortie anticipée en (15) suivie d'un message d'erreur en (16) si un problème de syntaxe ou de lecture sur la carte SD est rencontré**. En (17) l'énergie sur le LASER est supprimée et en (18) le plateau de la machine est dégagé. (*Normalement par une extinction du LASER, donc un retour à vitesse rapide.*) En (19) on désactive le booléen utilisé par des procédures comme **Traiter\_la\_rotation\_durant\_la\_gravure()**, **Traite\_M()** etc. En (20) on pilote la coupure d'énergie sur les moteurs. Que ce soit en sortie de pyrogravure ou en fuite par le test (11) dans tous les cas l'affichage est rétabli en (21) et un BIP signale à l'opérateur la fin du processus. Enfin, en (22) on rétablit l'affichage de la liste des images et on évite une sortie prématurée de ce menu qui doit se faire sur commande de l'opérateur par le B.P. central du **C.I.** .

## Déplacements dans toutes les directions. 5/5

### ➤ Balayages cartésiens.

**C** oloïré en vert pastel sur la Fig.1 le test complet se termine par deux séquences de déplacements couvrant toute la surface par des balayages longitudinaux soit sur **X** en Fig.13 partant de *l'arrière de la machine vers l'avant*, soit sur **Y** en Fig.14 partant cette fois *de la droite vers la gauche*. *Les deux sens de balayage* sont arbitraires et l'intégralité du plateau virtuel est balayée.



### ➤ Les options possibles pour le programme P15.

**A** ctivé dans sa version par défaut, le programme enchaîne les six séquences sans afficher les positions atteintes. Pour permettre une analyse fouillée du comportement des divers traitements, on peut ne valider que l'une des séquences dans la boucle de base **void loop()** en plaçant les autres en remarques avec **//**. Cependant, les coordonnées du LASER virtuel ne seront affichées sur le moniteur de l'**IDE** que si l'instruction **return;** en tête de la procédure **void Affiche\_position()** est modifiée en remarque avec **//**. Pour repérer facilement la ligne concernée une remarque de type **//@@@@@@@@** termine la ligne d'instruction.

Enfin, si l'on désire visualiser les coordonnées des mouvements relatifs aux tracés virtuels des disques circulaires, il faut, dans la procédure **Teste\_le\_grand\_disque\_central()** et à l'intérieur de son homologue **Teste\_le\_petit\_disque\_central()** valider les trois lignes d'instructions qui sont actuellement en remarques :

```
//Serial.println(); Serial.print("Angle = "); Serial.print(Angle,1);
//Serial.print(" X = "); Serial.print(Position_X,1);
//Serial.print(" Y = "); Serial.print(Position_Y,1);
```

### ➤ Mouvement vectoriel complet.

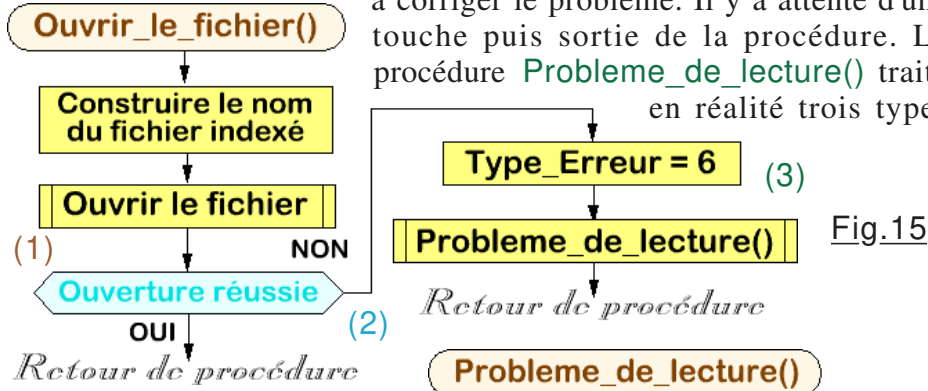
**D** ifférence du programme **P14** par rapport au démonstrateur **P12**, la prise en compte d'un **DeltaY** supérieur à **DeltaX** est traitée. Ce n'est pas très compliqué. Dans le traitement des mouvements vectoriels, si **DeltaY** est supérieur à **DeltaX** il suffit d'intervertir le rôle des moteurs sur les deux axes de déplacement.

## Explorer le fichier Image 1/8

Animier les diverses unités fonctionnelles de la machine impose de lire ligne à ligne le fichier image soumis à l'analyseur syntaxique qui en vérifiera la validité. Cette opération exige en préambule de s'assurer de la cohérence du contexte que ce soit pour vérifier un fichier ou pour pyrograver. (*Deux traitements similaires.*)

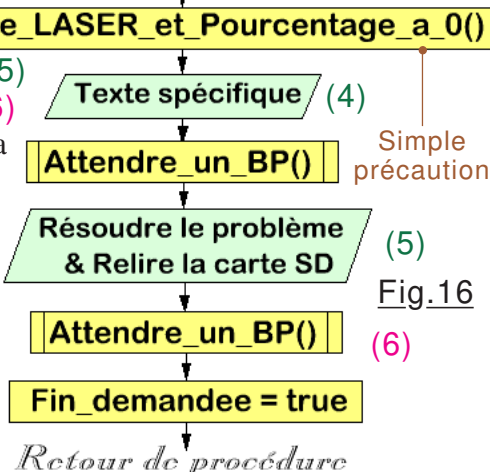
### ➤ Procédures de servitude.

Préambule au développement des organigrammes relatifs à la vérification des fichiers ou à la pyrogravures, deux modules doivent au préalable être décrits. Le premier en Fig.15 fait appel à la fonction de la bibliothèque **PetitSerial.h** qui en (1) ouvre effectivement le fichier pointé et retourne un booléen qui permet le test (2). Si l'ouverture n'est pas correcte, en (3) on prévient l'opérateur, et on l'incite à corriger le problème. Il y a attente d'une touche puis sortie de la procédure. La procédure **Probleme\_de\_lecture()** traite en réalité trois types



de problème. Elle incite à résoudre le problème en (5) avec attente d'une touche en (6) pour sortir. En fonction de la variable de traitement des erreurs **Type\_Erreur** les divers textes spécifiques en (4) sont :

Typ.	Pb sur le lecteur SD.
4	Pb sur le lecteur SD.
5	Pb Capteur ORIGINE !
6	Erreur en lecture.



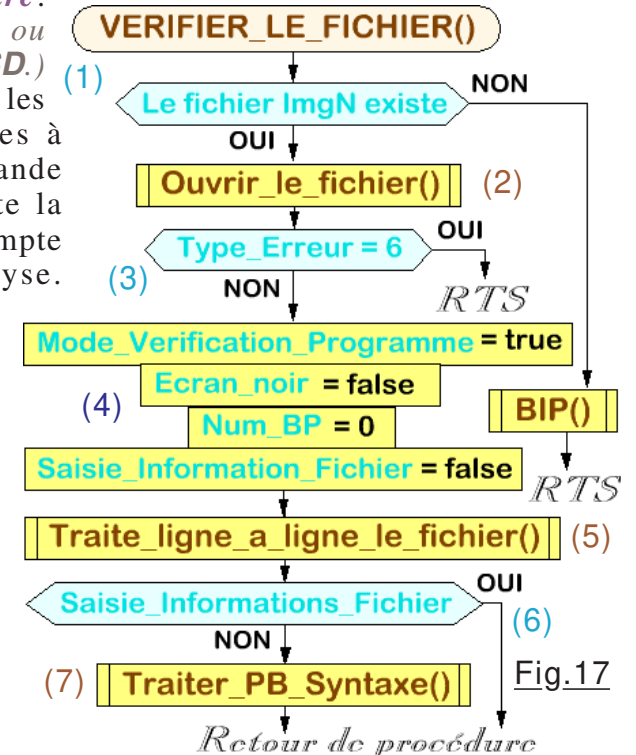
## Explorer le fichier Image 2/8

### ➤ VERIFIER LE FICHIER.

Observer la machine adopter prématurément la configuration dégagee après une heure de travail bien avant d'avoir terminé le travail car une erreur de syntaxe a été détectée s'avère très frustrant. Aussi, lorsqu'une pyrogravure va s'avérer longue à effectuer, il sera préférable de passer l'ensemble du fichier image au crible de l'analyseur syntaxique avant d'activer la procédure. C'est l'objet de cette option. La Fig.17 développe la structure globale de cette séquence qui vérifie en (1) que le fichier est bien présent sur la carte **SD**. Puis en (2) on fait appel à la procédure de la Fig.15 pour ouvrir le fichier *et éventuellement générer un message d'erreur si un problème est rencontré*. Si en (3) il n'y a pas de problème d'ouverture du fichier **gco**, on impose en (4) l'affichage ligne à ligne des instructions et des coordonnées machines, ainsi que l'appel éventuel du mode pas à pas. *L'intégralité du fichier sera examinée en (5) avec une sortie anticipée et message d'erreur si un problème est rencontré.*

(Problème de Syntaxe ou de lecture de la carte **SD**.) Si on désire juste avoir les informations relatives à l'image par la commande **Y- ↓**, en (6) on saute la séquence qui rend compte du résultat de l'analyse.

(Dans ce cas seules les quelques premières lignes informatives du fichier sont traitées.) En (7) on précise s'il y a une erreur ou si le fichier est correct mais uniquement en mode analyse par **Y+ ↑** du fichier image. Un BIP est généré puis il y a attente d'un clic sur l'un des B.P. du clavier.





## ATTENTION : Ne pas imprimer tout ce qui suit à partir de cette page !

L'aspect informatique a été résumé sous la forme d'un petit manuel bien plus commode pour la maintenance logicielle que des chapitres regroupés dans un tutoriel. Il restait toutefois une facette qui concerne tout croquis "volumineux" dont on ne peut faire l'économie si l'on pratique régulièrement Arduino. C'est l'objet de ce complément qui à mon sens n'a pas sa place dans le livret de maintenance du programme d'exploitation de la pyrograveuse.

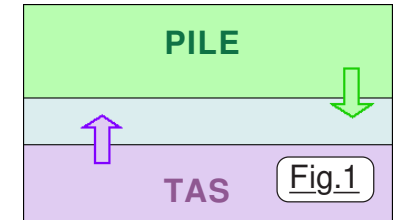
➤ **Abuser n'est pas gratuit ... ça peut faire mal.**

Compilons la dernière version du programme d'exploitation. **GLUPS ... il ne reste plus que 30 octet de disponibles pour le programme.** Du reste le compilateur ne se sent pas à l'aise, il nous avertit en signalant que 99% de l'espace de stockage de programmes est consommée. Et encore, il n'affiche pas les décimales car c'est 99,9% qui seraient révélés par le calcul précis. Pour faire court : Dans l'intégralité des zones mémoire disponibles de l'ATmega328 il ne reste non utilisé que 30 octets pour le programme, autant dire que **Dudule** ne pourra pas nous dire que l'on gaspille des ressources. On peut certifier que le taux d'occupation est presque déraisonnable. Pour circonstances atténuantes j'invoquerais le fait qu'avec 1054 octets entre la PILE et le TAS le programme devrait se montrer stable.

### Vérifier que l'impensable ne va pas se produire.

Un des problèmes les plus vicieux qui puisse survenir lors du développement d'un programme, c'est la collision entre la **PILE** et le **TAS**. Il n'est pas question dans cet exposé d'analyser en profondeur le fonctionnement interne d'un microcontrôleur. On va se contenter du minimum minimorum. Lorsque le programme fonctionne, il entasse dans une zone mémoire spéciale nommé le **TAS** les variables temporaires, comme les variables locales à une procédure, les paramètres passés par valeur etc. Simultanément, un pointeur d'adresse de retour des procédures et des interruptions entasse les adresses dans une autre zone nommée la **PILE**. La zone mémoire dédiée à ces deux fonctions est commune, et pour ne pas interférer elles sont situées aux "extrémités" de la RAM dédiée. Du coup le **TAS** se crée du bas vers le haut. La **PILE** au contraire ajoute ses valeurs du haut vers le bas. Plus il y a de données

dans le tas, plus il y a d'appels à procédures sans retour, et plus la zone (En bleu clair sur la Fig.1) qui sépare les deux antagonistes devient exiguë. Si vraiment la dynamique du croquis exige trop de place simultanément dans ces deux zones, elles se superposent et il y a écrasement de données vitales. On dit alors qu'il y a **COLLISION DE PILE**. C'est un problème particulièrement sournois car brusquement le logiciel se met à avoir un comportement totalement imprévu, alors que l'on a à peine modifié un fifrelin de code source. Par exemple on fait afficher "Bonjour". Puis on modifie par "Bonjour." en n'ajoutant qu'un point final à la chaîne de caractères. On relance le programme et **PAFFFFFFffffff, c'est du n'importe quoi.**



Particulièrement agassif, vous allez y passer le réveillon. (Façon de parler, car franchement au changement d'année je peux vous assurer que l'ordinateur est au chômage !) Vous aurez un mal fou à comprendre, car en toute logique n'ajouter qu'un modeste point final dans un texte affiché n'a aucune raison de perturber le déroulement global d'un programme. Donc si un jour un tel incident se produit, pensez à la **COLLISION DE PILE**.

➤ **Surveiller la COLLISION de PILE.**

Généralement, lorsque je finalise un programme "cossu", je sais par expérience que la PILE et le TAS sont très sollicités. On peut parfaitement imaginer que la combinatoire des cheminements dans les séquences de code n'ont jamais engendré de collision de PILE même si parfois on a sans le savoir "frisé la correctionnelle". Il n'est jamais prouvé qu'en utilisation normale, avec une marge parfois limite, que dans une configuration particulière TAS ou PILE débordent d'un ou deux Octets. C'est la sanction imparable. Aussi, pour considérer qu'un programme est fiable à ce point de vue, avant de le valider, une bonne pratique consiste à effectuer une mesure de la "marge" qui existe entre le TAS et la PILE. On ne considèrera que le risque de collision est écarté que si la marge calculée par une petite séquence particulière dépasse les 150 Octets. Ce n'est pas une preuve à proprement parler, mais une sorte de principe qui depuis que je programme n'a jamais été remis en cause. Voici comment procéder :

Techniquement, l'approche consiste à activer une séquence qui mesure la place restante en RAM dédiée quand le programme a effectué toutes ses initialisations. Il a ainsi "entassé" toutes ses variables et le **TAS** atteint probablement la hauteur maximale. On fait afficher la taille en Octets qui reste encore disponible pour la **PILE**. Si cette zone est inférieure à 150 Octets je considère que ce n'est pas suffisant. Une longue recherche d'optimisation du code source est alors engagée pour faire "maigrir" les exigences de place imposée sur le **TAS** et sur la **PILE**. *(Cette optimisation exige une bonne expérience en programmation.)* Lorsque les résultats obtenus sont satisfaisants, on neutralise les séquences qui servent à cette vérification pour alléger le programme et libérer le maximum de place mémoire. Je dois avouer que pour cette application, la manipulation est loin d'être redondante, car avec pratiquement 100% de zone programme utilisée, on se doute que le **TAS** et la **PILE** sont plus que sollicités. Du reste pour pouvoir conduire la manipulation il faut libérer de la place et transformer la procédure `void Afficher_Ecran_PYROGRAVURE()` {} en procédure vide. Donc, passer son corps en remarque par encadrement avec `/*` et `*/`. En tête de listage on trouve à la fin de la procédure `void setup()` :

```

@@@@@@@@@@ Ci-dessous code ajouté pour afficher la place disponible. @@@@@@@@@@
// u8g.firstPage(); do {u8g.setPrintPos(22, 35); u8g.print("PILE - TAS = ");
// u8g.print(SRAM_LIBRE());} while(u8g.nextPage()); ATTENDRE_un_BP();
@@@@@@@@@@@@*****@@@@@@@@@@@@

```

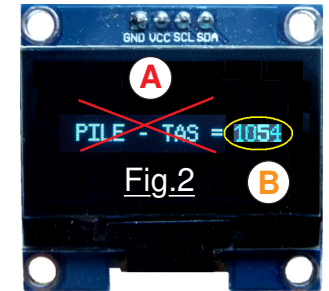
Les deux lignes de commentaire qui encadrent le code contiennent des chaînes de caractère du genre `@@@@@@@@@@` pour facilement les repérer dans le listage. Vous noterez que le code source colorisé en orange est ignoré par le compilateur car transformé initialement en remarque par le `//` placés en début de ligne. Suivant directement la séquence d'initialisation et juste avant `void loop()` on trouve :

```

@@@@@@@@@@ Ci-dessous fonction pour calculer la place disponible @@@@@@@@@@
// int SRAM_LIBRE() { // Fonction qui retourne la taille de SRAM disponible.
// extern int __heap_start, *__brkval; // Déclaration des deux pointeurs dédiés.
// byte BIDON; // Dernière variable allouée, donc occupe le "haut" du TAS.
// if (__brkval == 0) {return (int) &BIDON -(int) &__heap_start;}
// else {return (int) &BIDON -(int) __brkval;} }
@@@@@@@@@@@@*****@@@@@@@@@@@@

```

Cette séquence est comme pour la précédente entièrement neutralisée par des `//` placés en début de ligne. Lorsque l'on désire faire une mesure de la place qui reste actuellement disponible entre la **PILE** et le **TAS** il suffit de *valider ces lignes de code* et de passer en remarque tout en bas de **P20** le corps de `Afficher_Ecran_PYROGRAVURE`. Le programme démarre normalement. Puis, quand on clique dans la page d'accueil pour passer en exploitation, l'écran devient noir avec au centre une valeur numérique. C'est précisément la valeur de la zone disponible. Le programme attend un clic au clavier pour passer en exploitation, nous laissant ainsi le temps de lire la valeur. Actuellement, dans le cas de **P20** la validation de cette séquence annonce un espace confortable de 1054 octets, auquel il faut retrancher les 230 octets qui reconstitueront la procédure masquée. Il n'y a donc pas de "plantage vicieux" trop à craindre, même avec la dernière version "abusive" qui consomme 99,9% de l'espace réservé au programme. L'introduction de ce code de servitude



qui ne concerne que le programmeur consomme 140 Octets de programme et 24 octets de mémoire dynamique. C'est du "code parasite" qui gloutonne inutilement des ressources du microcontrôleur. Le laisser serait impertinent si la marge de place disponible est limite, car ces séquences viennent encombrer de la place mémoire. Par ailleurs, avoir à sauter cette phase à chaque RESET manque de convivialité. Aussi, chaque fois que ces séquences viennent polluer un croquis quel qu'il soit, il est plus que logique de les neutraliser en les transformant en remarque en rétablissant les `//`. Notez que la page écran dédiée et montrée en Fig.2 est scandaleusement incongrue. En effet, on est supposé craindre un manque de place entre la **PILE** et le **TAS**. Et pour calculer cette dernière on affiche bêtement en **A** le texte "**PILE - TAS =**" alors que seule ne présente de la pertinence le nombre **B**. Aussi, vous avez bien compris que ce luxe stupide n'est présent dans le code source uniquement parce-que **P20** "allégé" a montré qu'il restait largement de la place. Dans un contexte de "vaches maigres", il ne faudrait surtout pas faire afficher ce texte ou ... risque de COLLISION !