TABLE TRACANTE

L'objectif est de dessiner une figure avec la table traçante. Il faudra utiliser le protocole GCODE utilisé par les machines à commande numérique. La programmation se fera avec Netbeans en utilisant le port Série. En cas de tracés inconvenant, l'équipe sera éliminée et se verra attribué la note de 0.

1 Découverte de la machine à commande numérique	• Le Charlyrobot
2 Protocole GCODE	• Liste des codes utilisés • Tracé d'une fenêtre
3 Simulation sous Hterm	• Configuration avec Hterm
4 Programmation sous Netbeans	• Déplacement avec le clavier. • Génération de dessin automatique

A l'issue de l'étude, une présentation de 10 minutes du mini-projet sera effectuée le vendredi après midi en fin de semaine.

1 Découverte de la machine à commande numérique

La commande électronique du charlyrobot d'ancienne génération a été entièrement refait. Il dispose de 3 contrôleurs (drivers) de moteur pas à pas TB6600, piloté par un Arduino Uno. L'Arduino est programmé avec un interpréteur de commande GRBL GCODE.

L'énergie électrique 12V est fournie par une alimentation de PC recyclée.

L'ensemble permet désormais d'utiliser la machine avec le tout dernier logiciel de commande que vous allez réaliser.

2 langage de programmation de trajectoire GCODE

Le langage GCODE est expliqué ici :

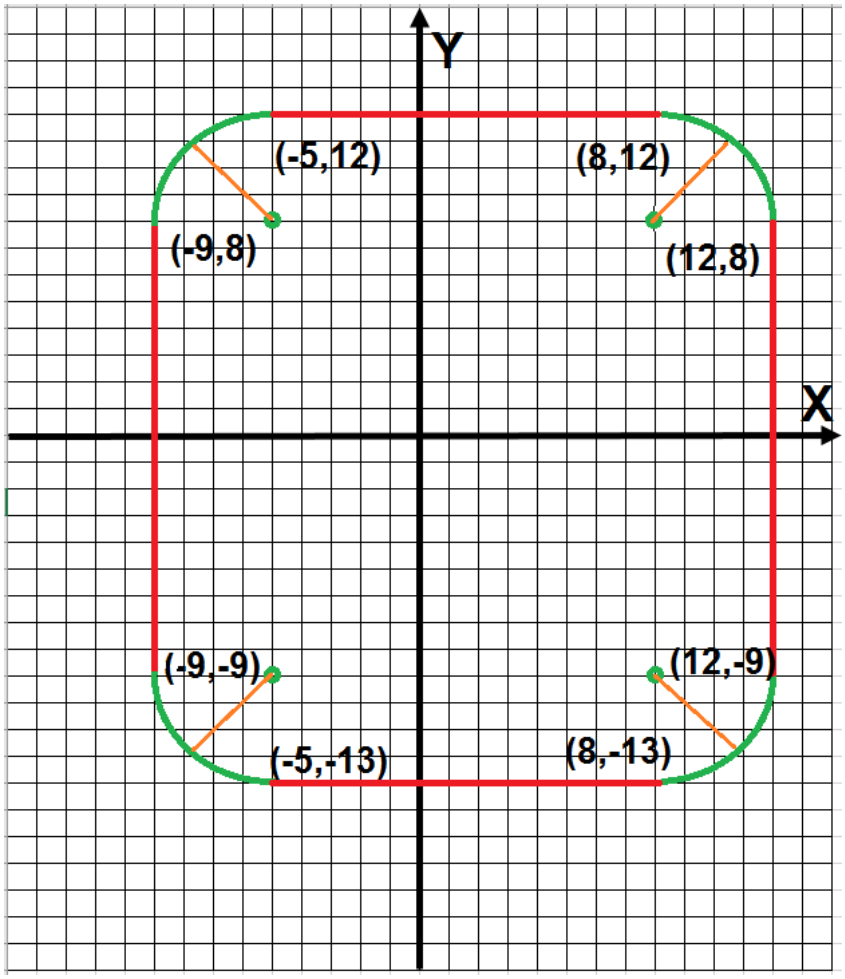
http://linuxcnc.org/docs/html/gcode/gcode_fr.html

Vous utiliserez **seulement** les commandes suivantes :

g90, g0, g2, g3

Consulter la documentation afin de connaître le rôle de chaque commande

Exemple de déplacement d'outil dans le plan XY

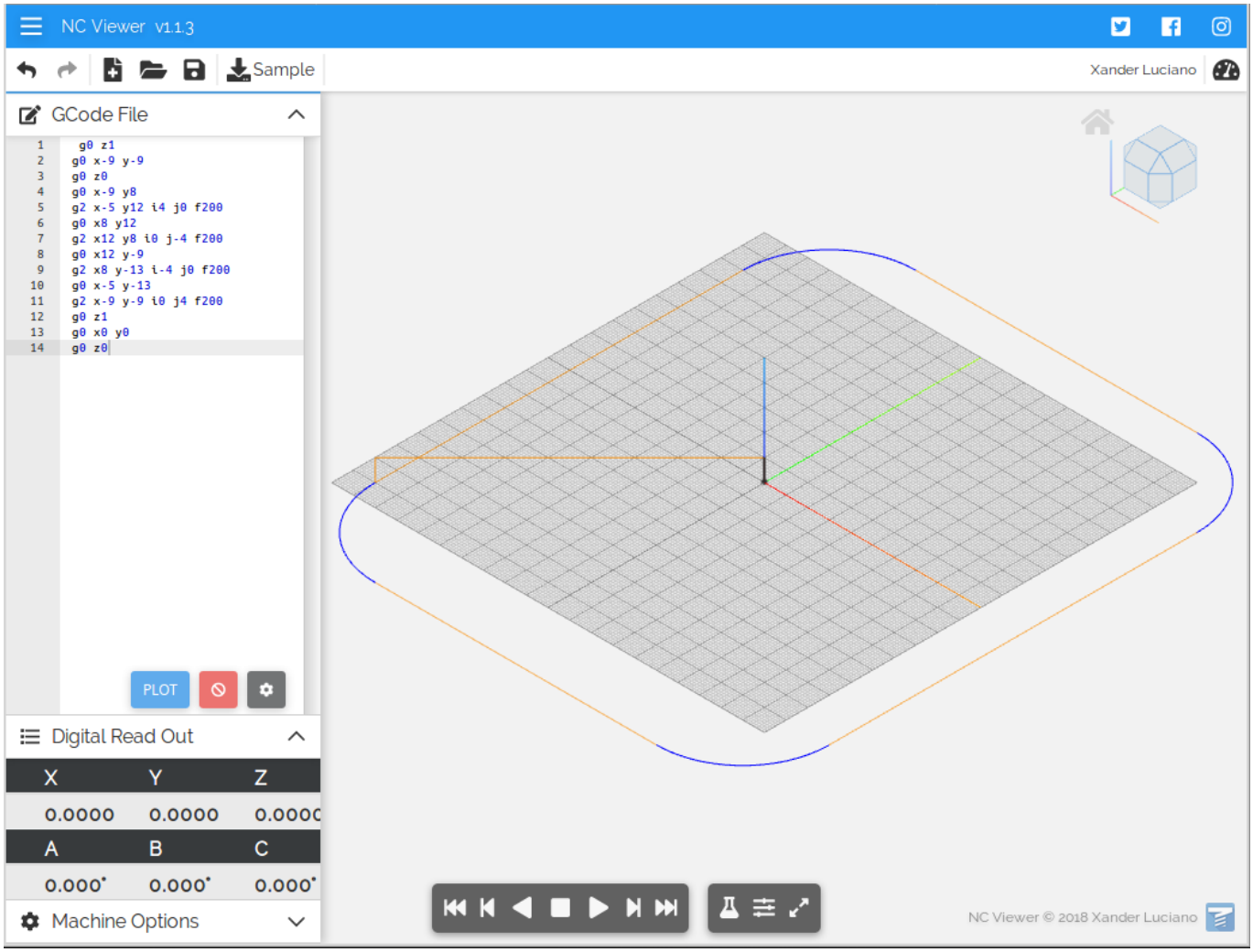



```
#define NBCDE 15
   char *gcode[NBCDE] = {"g90", //coord absolu (cotes au mm)
      "g0 z1", // monte le crayon de 1mm
      "g0 x-9 y-9", //deplacement x y
      "g0 z0", //descend le crayon de 1mm
      "g0 x-9 y8", //trace une droite verticale
      "g2 x-5 y12 i4 j0 f200", // trace un arc de cercle sens horaire
      "g0 x8 y12",
      "g2 x12 y8 i0 j-4 f200",
      "g0 x12 y-9",
      "g2 x8 y-13 i-4 j0 f200",
      "g0 x-5 y-13",
      "g2 x-9 y-9 i0 j4 f200",
      "g0 z1", // monte le crayon de 1mm
      "g0 x0 y0", // retourne a l'origine
      "g0 z0"}; //descend le crayon de 1mm
```

La difficulté est de bien comprendre les commandes G2 et G3 pour tracer des cercles ou des arcs de cercles.

Une explication est fournie ici : https://www.youtube.com/watch?v=gXSlyxMiZ6s

Vous pouvez utiliser le simulateur NC Viewer pour visualiser par simulation la trajectoire de l'outil.
https://ncviewer.com/



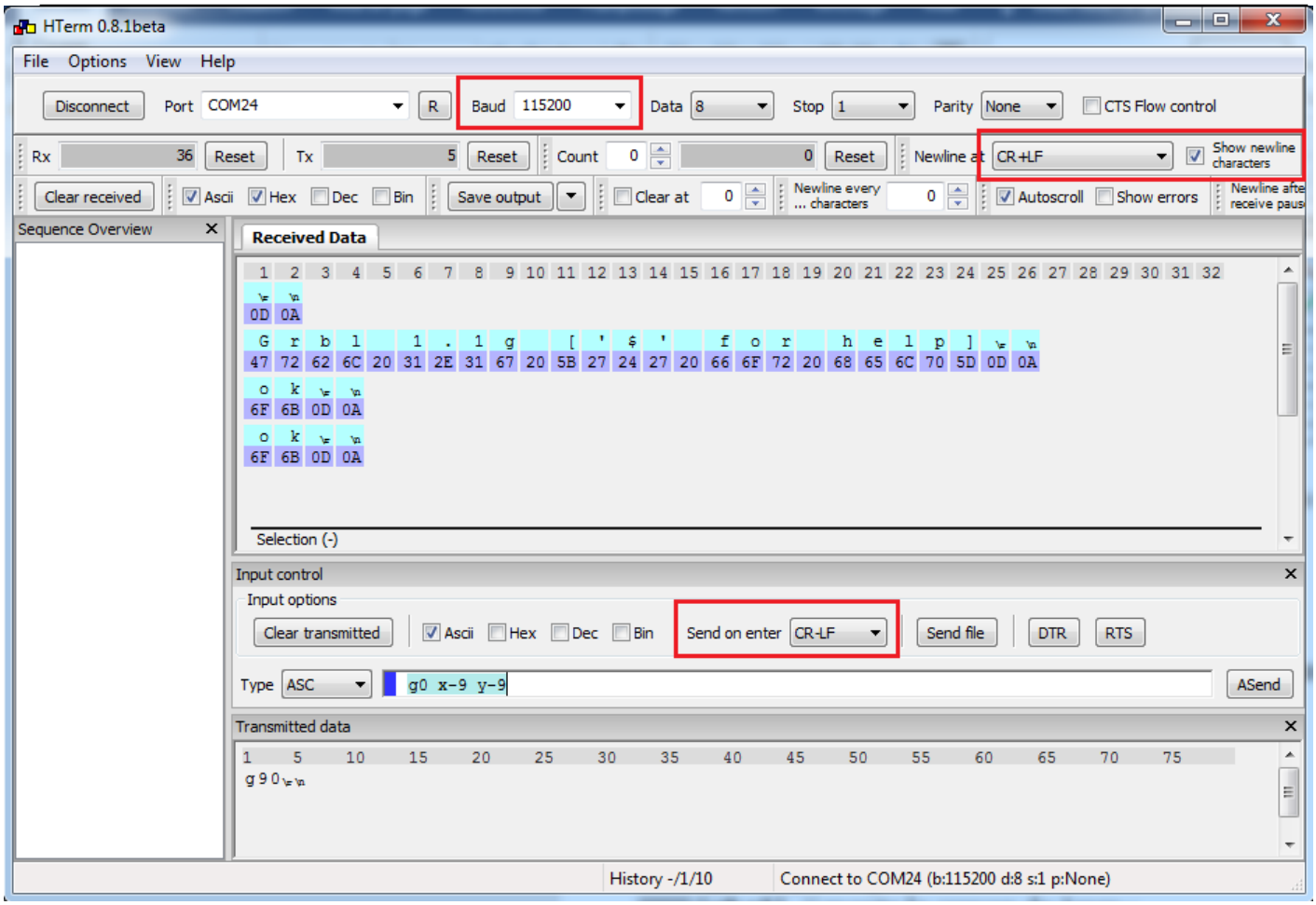
3 Simulation sous Hterm

Avant de se lancer dans la programmation, il faudra vérifier les commandes par simulation avec le terminal Hterm

3.1 Vérifier le port de communication série dans la console :

```
ls /dev/tty*
```

Normalement vous devez trouver ttyUSB0 ou ttyACM0.



Attention : il faudra déplacer **les axes X, Y, Z** à la main afin de définir **l'origine du tracé.**
Le crayon touchera la feuille de papier avant l'envoi des commandes. Les seules commandes autorisées sur l'axe Z sont :

« g0 z1 » et **« g0 z0 »**

3.2 Bien analyser les échanges avec Hterm :

3.3 Que se passe-t-il à l'ouverture du port série (connexion) ?

3.4 Que retourne la machine lors de l'envoi d'une commande G90 ?

3.5 Les caractères CR LF sont-ils utilisés ?

3.6 Pourquoi la machine retourne-t-elle apparemment 2 OK à chaque envoi de commande ? (Voir annexe 2 en fin de document)

4 Programmation sous Netbeans

Programme N°1 :

Utiliser les touches du clavier (z,q,s,d,x) afin de déplacer les axes X,Y d'un pas de 10mm lors de chaque appui. Utiliser le caractère 0 pour sortir du programme.

Algorithme :

Début

Initialiser le port série : 115200 bauds, 8, n,1.

Compter (N) caractères entrants sur le port série

Boucle de gestion du clavier et envoi des commandes

Fermeture du port série

Fin

(N) : A déterminer à l'aide de la question 3.3 ou copie d'écran page 3.

En annexe 1 les fonctions à réaliser permettant de résoudre le problème.

Programme N°2 :

Générer automatiquement un dessin de votre choix en lien avec le BTS SN IR.

Annexe 1

Lire la liaison série caractère par caractère, puis enregistre les données dans le buffer jusqu'à ce que l'on détecte le caractère LF. (Réception des données RX)

```
void readSerialUntil(int fd, char buffer[], char code) {
    char car;
    char buf[1];
    int indice = 0;
    int n;
    buf[0] = 0;
        //a compléter

    printf("%s\n", buffer);
}
```

Attendre ok\r\n sur la liaison série

```
int waitOk(int fd) {
    char buf[5];
    int retour = -1;
    readSerialUntil(fd, buf, 10);

        //à compléter

}
```

Lire nb caractères sur le port série. (Réception des données RX)
Ne pas stocker les caractères entrants, on désire juste les compter.

```
void lirePortSerieNbchar(int fd,int nb) {
    int n = 0;
    int compteur = 0;
    char buf[1];

        //à compléter

    printf("-> %d, %s \n", compteur, buf);
}
```

Envoi une commande gcode sur le port série. (Emission des données TX)

```
void sendCde(int fd, char cde[]) {
    int n;

// A compléter
}
```

Rappel du paramétrage à effectuer dans Netbeans:

```
struct termios tty, orig;

//ouverture du port série : a completer

/* Lecture et sauvegarde des parametres courants */
    tcgetattr(fd, &tty);
    orig = tty;
    /* remplissage des champs de la structure termios*/
    cfsetispeed(&tty, B115200);
    cfsetospeed(&tty, B115200);
    tty.c_cflag |= (CLOCAL | CREAD);
    tty.c_cflag &= ~PARENB;
    tty.c_cflag &= ~CSTOPB;
    tty.c_cflag &= ~CSIZE;
    tty.c_cflag |= CS8;
    tty.c_lflag &= ~ICANON;
    tty.c_oflag = 0;

// initialiser le port série : a completer
// gestion de vos besoins avec la table tracante. : a completer
// configuration initiale du port et fermeture du port série. : a completer
```

Annexe 2

After each line of Gcode sent from the host to the controller, all the RepRap G-code interpreters (typically) respond with a line that starts with "ok" and ends with a newline. Most of the time it's just those 4 bytes -- "ok\r\n" -- but occasionally there's some debugging information at the end of the line.

Most of those RepRap interpreters work exactly like Michael Pruitt pointed out -- they have an internal buffer that can hold several lines of Gcode. The interpreter holds off on sending the "ok" reply until there is enough room in that buffer for the next line of Gcode. When there is plenty of space in the buffer, the interpreter may immediately send "ok" in response to several G1 codes, buffering them all up, sending those OKs *long* before it actually executes any of those commands.