

# Brevet de technicien supérieur

*Systèmes Numériques option Informatique et Réseau*

Dossier technique



# Sommaire

<b>I - Explication du projet</b>	<b>1</b>
A) Organisation des tâches .....	1
B) Mes tâches .....	2
C) Etude des contraintes .....	3
 <b>II - Réalisation du projet</b>	 <b>4</b>
A) Analyse du projet .....	4
B) Mise en place de l'environnement de développement .....	4
C) Envoie des données depuis un pc .....	5
D) Réalisation du code d'envoi de la carte Nucléo .....	10
E) Envoie des données depuis la carte Nucléo .....	11
 <b>III - Conclusion</b>	 <b>13</b>
A) Ressenti sur le projet .....	13
B) Aboutissements .....	13
 <b>Annexe</b>	 <b>14</b>

# I - Explication du projet

## A) Organisation des tâches

Mon objectif dans ce projet est d'assurer l'envoi des données depuis le système embarqué du bateau, vers le système à terre.

Afin d'atteindre cet objectif, je devrais réaliser un système capable d'envoyer les données via le réseau 4G, directement sur le système de réception de mon camarade. Pour ce faire, j'utiliserais une carte Nucléo et un bouclier arduino SIMCE7600.

Plus précisément, le diagramme des cas d'utilisations (*cf annexe 1*) indique la répartition des tâches de chacun des membres des 2 groupes, ainsi que la manière dont elles s'inscrivent dans le projet. Ma partie étant visible par la couleur bleue.

## B) Mes tâches

Étudiant 1	Liste des fonctions assurées par l'étudiant
EC <input type="checkbox"/> IR <input checked="" type="checkbox"/>	Transmission des informations via le réseau 4G
Laporte	➤ Définition des données à transmettre.
Kevin	➤ Choix de l'outil de développement.
	➤ Définition du protocole de transmission (format des données à transmettre).
	Travail avec l'étudiant 2
	➤ Définition du protocole de l'échange des données avec le système embarqué.
	Voir avec les étudiants qui gère le système embarqué.
	➤ Analyse des commandes AT pour dialoguer avec le serveur MQTT.
	➤ Analyse UML
	➤ Réalisation du programme permettant d'envoyer les informations.
	➤ Réalisation du programme permettant de recevoir les informations de la partie embarquée
	➤ Tests unitaires.

Plus simplement, j'ai dû mettre en place le système embarqué qui envoie les données du bateau au système à terre, via le réseau 4G.

Pour cela, il m'a fallu étudier les commandes AT, MQTT et les IDE pour la carte Nucléo.

Il a ensuite été nécessaire de discuter avec mon camarade Joshua avant que l'envoi de mes données au système à terre qu'il gère puisse s'effectuer.

## C) Etude des contraintes

Étudiant 1	Liste des fonctions assurées par l'étudiant		
EC <input type="checkbox"/> IR <input checked="" type="checkbox"/>	Transmission des informations via le réseau 4G	Revue 1	F0
Laporte	➤ Définition des données à transmettre.	Revue 1	F0
Kevin	➤ Choix de l'outil de développement.		
	➤ Définition du protocole de transmission (format des données à transmettre).	Revue 1	F1
	Travail avec l'étudiant 2		
	➤ Définition du protocole de l'échange des données avec le système embarqué.		
	Voir avec les étudiants qui gère le système embarqué.		
	➤ Analyse des commandes AT pour dialoguer avec le serveur MQTT.	Revue 2	F2
	➤ Analyse UML	Revue 3	F2
	➤ Réalisation du programme permettant d'envoyer les informations.	Revue 3	F1
	➤ Réalisation du programme permettant de recevoir les informations de la partie embarquée	Revue3	F2
	➤ Tests unitaires.	Revue3	F2

Flexibilité :

- F0 : flexibilité nulle, niveau impératif
- F1 : flexibilité faible, niveau peu négociable
- F2 : flexibilité bonne, niveau négociable
- F3 : flexibilité forte, niveau très négociable

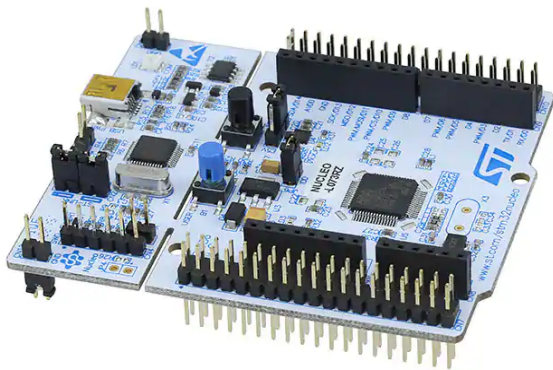
En plus de ses contraintes de réalisation, il m'a été imposé d'utiliser les commandes AT et MQTT, ainsi qu'une carte SIM 7600E et une carte Nucléo L073RZ, cette dernière m'imposant indirectement de travailler avec des logiciels STM32.

## II - Réalisation du projet

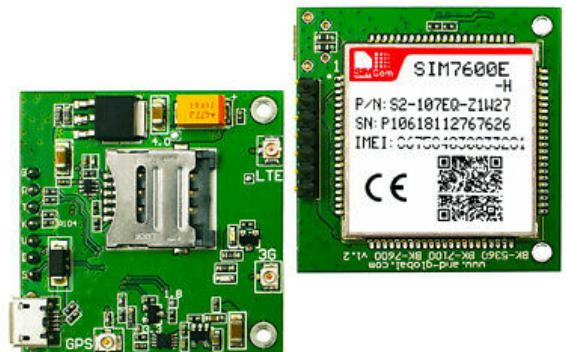
### A) Analyse du projet

L'objectif du projet est d'envoyer les données reçues par le système embarqué à un système situé à terre.

Pour cela, il a été choisi d'utiliser une carte Nucléo L073RZ et une carte SIM 7600E.



(figure 1 : Carte Nucléo L073RZ)



(figure 2 : Carte SIM 7600E)

Afin de mettre en place ce système, j'ai choisi de commencer avec la SIM 7600E, afin d'abord d'envoyer une donnée au système à terre avec un pc. Ensuite, l'objectif est d'envoyer des données grâce à la Nucléo. Enfin, la finalité du projet signifie recevoir des données depuis le système embarqué, cependant, le contexte actuel a entraîné des retards dans mon projet et dans celui de mes camarades qui travaillent sur le système embarqué, rendant impossible la réalisation de cette partie et des tests de cette dernière.

### B) Mise en place de l'environnement de développement

Logiciels pour la Nucléo (*sous Windows*) :



STM32CubeMX (v6.0.1)

STM32CubeIDE (v1.6.1)



Logiciels de terminaux :



Tera Term (v4.105) (*sous Windows*)

GTKTerm 2 (v0.2.3) (*sous Ubuntu 20.4*)



## C) Envoie des données depuis un pc

J'ai tout d'abord connecter la carte SIM 7600E à mon pc via un câble usb. Ainsi, j'ai tenté de lancer les commandes AT via un code sous qt, pour automatiser le processus, sans grand succès. Je suis donc passé à la méthode plus classique, c'est-à-dire un terminal GTKterm sous Linux, afin de communiquer directement avec la carte SIM 7600E. Il faut cependant noter qu'il est nécessaire de redémarrer le terminal quelques fois pendant le processus avant de finalement réussir à envoyer les données souhaitées.

Tout d'abord, il est nécessaire de parler des commandes AT.

La firme Hayes, fabricant de modems, a développé un protocole pour la commande d'un modem externe à partir d'un ordinateur. Le protocole définit diverses commandes permettant par exemple :

- de composer un numéro de téléphone
- de commander le raccordement du modem à la ligne (l'équivalent de décrocher le téléphone)
- de connaître l'état de la ligne : tonalité d'invitation à transmettre, ligne occupée ...
- de spécifier le type de transmission et le protocole de liaison à utiliser
- de régler le volume sonore du haut-parleur interne du modem
- d'envoyer les caractères transmis simultanément vers l'écran
- d'afficher certains renseignements concernant le modem
- de manipuler les registres internes du modem

Les commandes AT (ou Commandes Hayes) sont des commandes que l'on peut directement envoyer au modem lorsque celui-ci est en mode Command.

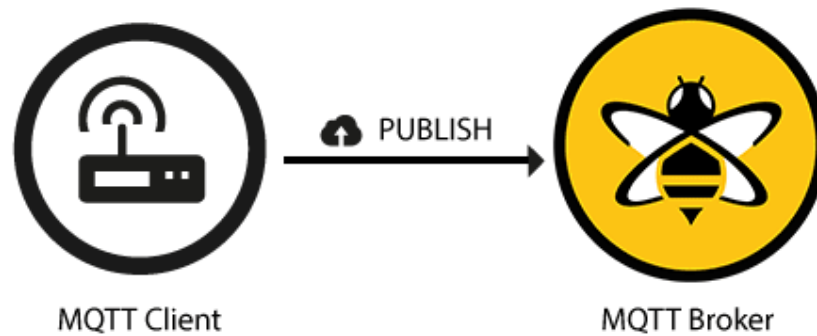
Chaque commande est envoyée sous la forme d'une ligne de texte encodée en ASCII, terminée par le caractère '\r' seul (code ASCII 0x13). Le modem retourne une réponse sous la forme d'une ou plusieurs lignes selon la commande envoyée, chaque ligne se terminant par les caractères '\r' suivi de '\n' (codes ASCII 0x13 et 0x10).



(figure 3 : Logo du fabricant de modem Hayes)

Cependant, les commandes AT sont ici utilisées avec MQTT, ce qui est prévu par les développeurs.

MQTT est en effet le point commun entre ma partie et la partie de mon camarade Joshua, qui gère le système à terre. Une explication plus conséquente de MQTT se trouvant dans sa partie, nous verrons ici rapidement son fonctionnement.



(figure 4 : Schéma MQTT)

Mon modem vient ainsi créer un client, qui identifiera auprès de MQTT afin de savoir quel modem aura envoyé une donnée définie.

Il viendra ensuite publish cette donnée après avoir entré le topic dans lequel cette dernière sera sauvegardée.

La suite se déroule du côté de mon camarade Joshua.

Maintenant qu'on en sait plus sur les commandes AT et MQTT, nous pouvons voir comment je m'en suis servi lors du projet.

```
RDY
+CPIN: SIM PIN
```

(figure 5 : Démarrage SIM 7600E)

Lors du démarrage de la SIM 7600E, elle nous envoie ces deux messages afin de notifier qu'elle est prête à recevoir des instructions.

```
at+cpin=0000
OK

+CPIN: READY

SMS DONE

PB DONE
```

(figure 6 :Déblocage SIM 7600E)

Comme lors du démarrage d'un téléphone, on débloque la carte SIM grâce à un code PIN (ici 0000). La carte nous informe alors que c'est un succès grâce aux messages **SMS DONE** et **PB DONE**, signifiant que la carte SIM peut être utilisée.

```
at+cscs=?
+CSCS: ("IRA", "GSM", "UCS2")

OK
at+cscs?
+CSCS: "IRA"

OK
at+cscs="gsm"
OK
```

(figure 7 : Configuration du réseau de la SIM 7600E)

Afin de pouvoir utiliser le réseau 4G en France, il est nécessaire de passer la carte sur le réseau GSM, car elle est de base configurée sur le réseau IRA.

```
at+cmgf=1
OK
at+cmgs="+33687449655"
> bonjour les BTS

+CMGS: 35

OK
```

(figure 8 : Envoie d'un SMS depuis la SIM 7600E)

Pour vérifier que la carte SIM est bien configurée, on envoie un SMS sur un téléphone portable. Le code réponse **+CMGS: 35** n'a pas de réelle importance ici, c'est le **OK** qui compte.



```
at+cmqttstart
OK
+CMQTTSTART: 0
```

(figure 9 : Démarrage MQTT sur la SIM 7600E)

Désormais que la carte SIM est connectée au réseau 4G, il est temps de démarrer MQTT, afin de communiquer avec le système à terre. La carte SIM nous informe que MQTT a bien démarré avec le message **+CMQTTSTART: 0**.

```
at+cmqttaccq=0,"client1"
OK
```

(figure 10 : Définition client pour la SIM 7600E)

On définit alors le client qui servira ensuite lors de la connexion, le 0 de la commande étant celui qu'on retrouve au départ de la commande suivante. Ce dernier sert également à identifier l'appareil qui envoie la donnée.

```
at+cmqttconnect=0,"tcp://193.253.222.162:2021",60,1,"mqtt","snirpass"
OK
+CMQTTCONNECT: 0,0
```

(figure 11 : Connexion MQTT de la SIM 7600E au serveur MQTT à terre)

On se connecte alors au serveur MQTT à terre grâce à cette commande. On a alors une réponse positive **+CMQTTCONNECT: 0,0**.

```
at+cmqtttopic=0,12
>topic/projet
OK
```

(figure 12 : Sélection du topic pour la SIM 7600E)

On sélectionne ensuite le topic,

```
at+cmqttpayload=0,15
>bonjour les bts
OK
```

(figure 13 : Génération du message envoyé par la SIM 7600E)

On continue en générant le message que l'on va envoyer au serveur à terre, qu'on valide avec un CTRL Z.

```
at+cmqttpub=0,0,60
OK
+CMQTTPUB: 0,0
```

(figure 14 : Envoie du message par la SIM 7600E)

Enfin, on envoie le message, et le système nous répond avec un **+CMQTTPUB: 0,0**.

```
pi@acquidonnes:~ $ node serv.js
Connecting MQTT
Subscribed to topic/projet
Database Connected!
bonjour les bts
Message reçu: bonjour les bts
```

(figure 15 : Démarrage SIM 7600E)

Côté système à terre, géré par mon camarade Joshua, on lance d'abord le serveur avec la commande **node serv.js**, puis on voit notre message arriver en direct.

<input type="checkbox"/>	 Modifier	 Copier	 Effacer	28	bonjour	topic/projet	bonjour,les,bts	1	2021-03-31 08:46:10
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	29	bonjour	topic/projet	bonjour	1	2021-03-31 08:47:07
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	33	bonjour les bts	topic/projet	bonjour les bts	1	2021-03-31 09:03:13

(figure 16 : Démarrage SIM 7600E)

On constate alors que le message envoyé est bien inscrit dans la base de données. On peut même observer 3 tests de chaînes de caractères incluant des caractères spéciaux tels que des espaces et des virgules afin de tester simplement les limites des données que l'on peut envoyer.

## D) Réalisation du code d'envoi de la carte Nucléo

Le code d'envoi sur la carte Nucléo a été réalisé avec CubeIDE, suite à l'échec lors de l'utilisation d'Atollic. Ce sont deux IDE de développement pour CubeMX, ce dernier permettant d'activer ou de désactiver des pins sur les cartes Nucléo.

```
#define CPIN "AT+CPIN=0000\r\n" // 15 -- OK _ PB DONE
#define CPCS "AT+CPCS=\"GSM\"\r\n" // 16 -- OK
#define CMGF "AT+CMGF=1\r\n" // 12 -- OK
#define CMGS "AT+CMGS=\"+33687449655\"\r\n" // 24 -- +CMGS:xx _ OK
#define SMS "Connexion au réseau réussie" // 32 -- OK _ +CMQTTSTART: 0
#define START "AT+CMQTTSTART\r\n" // 16 -- OK
#define ACCQ "AT+CMQTTACCQ=0,\"client1\"\r\n" // 27 -- OK
#define CONNECT "AT+CMQTTCONNECT=0,\"tcp://193.253.222.162:2021\",60,1,\"mqtt\",\"snirpass\"\r\n" // 72 -- OK _ +CMQTTCONNECT= 0,0
#define TOPIC "AT+CMQTTTOPIC=0,12\r\n" // 21 -- OK
#define TOPICDEF "topic/projet\r\n" // 15 -- OK
#define PAYLOAD "AT+CMQTTPAYLOAD=0,5\r\n" // 23 -- OK
#define PAYLOADDEF "49 50" // x -- OK
#define PUB "AT+CMQTTTTPUB=0,0,60\r\n" // 21 -- OK _ +CMQTTTTPUB: 0,0
#define CTRLZ "\x1A" // 2 --
```

(figure 17 : Define des commandes AT sous CubeIDE)

On commence alors par des **#define**, ce qui nous permet d'insérer nos commandes AT utilisées précédemment dans CubeIDE. En commentaire, on retrouve la taille de la chaîne de caractères ainsi que la réponse attendue.

Ainsi, on va pouvoir les envoyer à la carte SIM grâce au code suivant :

```
HAL_UART_Transmit(&huart1, CPIN, 15, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, CPCS, 16, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, CMGF, 12, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, CMGS, 25, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, SMS, 28, 100);
HAL_Delay(100);
HAL_UART_Transmit(&huart1, CTRLZ, 2, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, START, 16, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, ACCQ, 27, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, CONNECT, 72, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, TOPIC, 21, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, TOPICDEF, 15, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, PAYLOAD, 22, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, PAYLOADDEF, 5, 100);
HAL_Delay(10000);
HAL_UART_Transmit(&huart1, PUB, 21, 100);
```

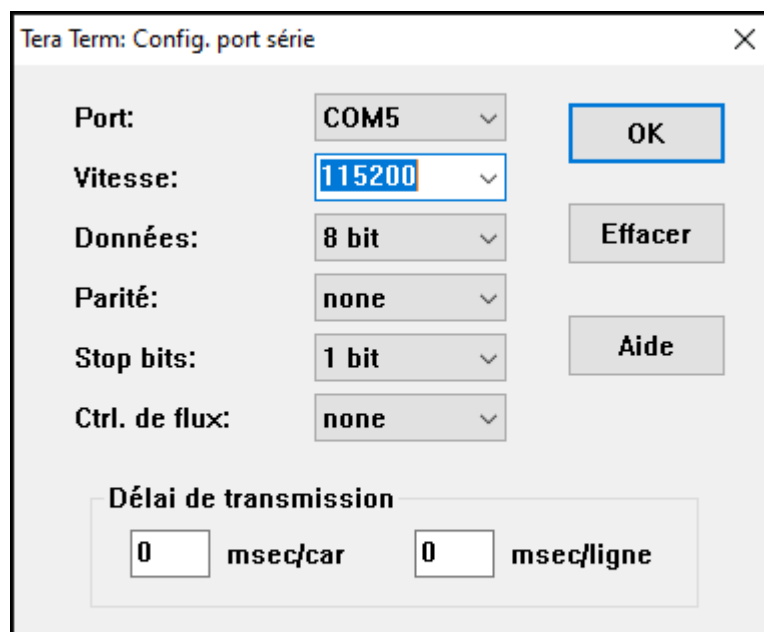
(figure 18 : Démarrage SIM 7600E)

On utilise donc la fonction HAL\_UART\_Transmit, native de CubelIDE afin d'envoyer les commandes définies précédemment, vers la carte SIM.

Après chaque HAL\_UART\_Transmit, on insère un HAL\_DELAY de 10 secondes, car les commandes AT peuvent mettre jusqu'à 10 secondes à répondre. Pour s'assurer que le code qu'on lance respecte cela, les HAL\_DELAY de 10 secondes sont donc utiles. Cependant, dans la continuité du projet, il sera nécessaire de se passer de ces HAL\_DELAY tout en s'assurant de l'intégrité du code.

## E) Envoie des données depuis la carte Nucléo

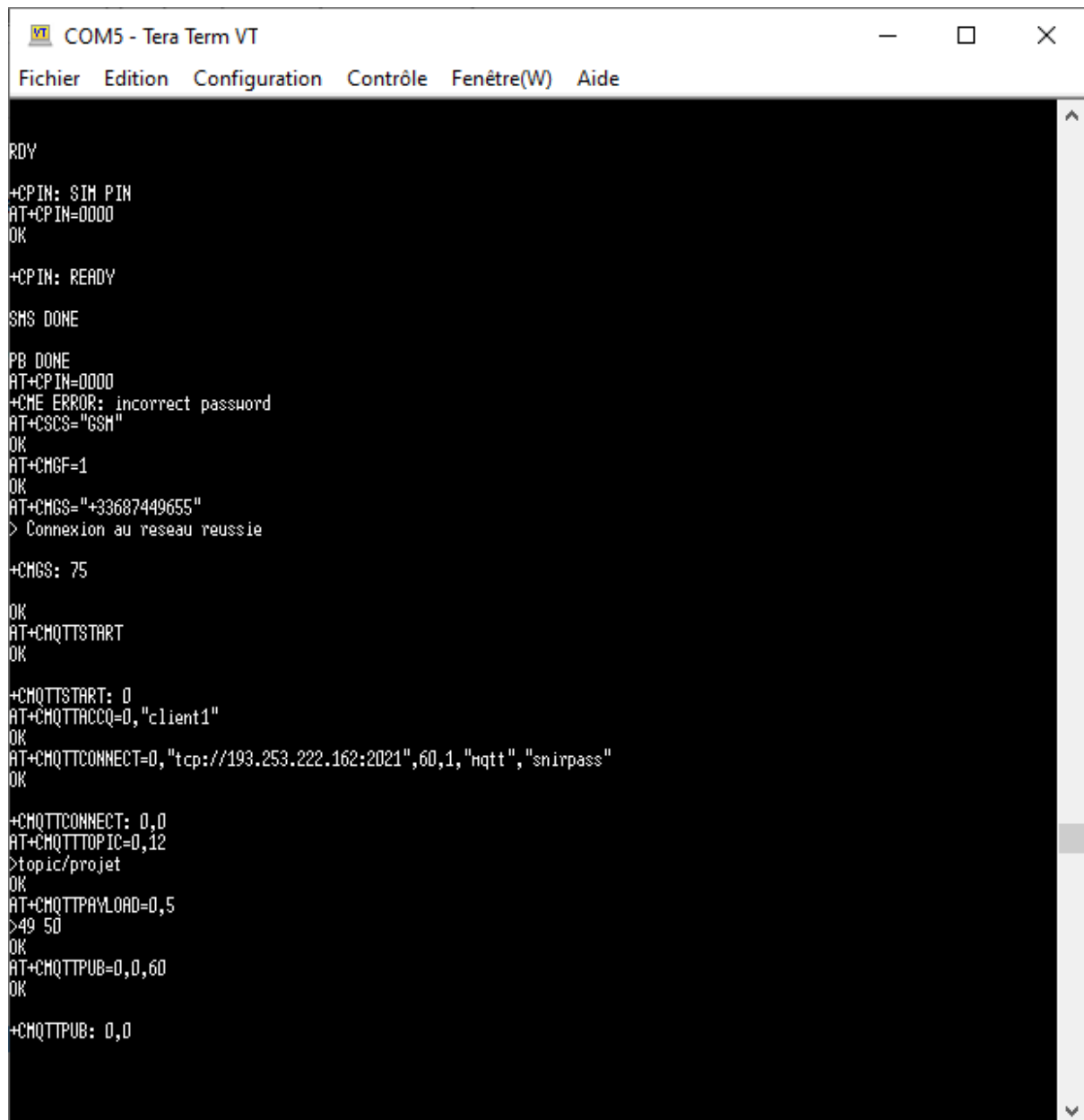
Lors de l'envoi des données, on peut récupérer les échanges entre les deux cartes en se connectant au pin Rx de la SIM 7600E, il suffit alors d'ouvrir un terminal (Tera Term ici) et de régler la vitesse du terminal 115200 bauds.



(figure 19 : Configuration port série Tera Term)

Ensuite, il suffit de lancer le code depuis CubelIDE, de préférence en mode débogueur si l'on veut ajouter des breakpoints pour comprendre le code (à noter que dans ce mode, le premier HAL\_UART\_Transmit, qui envoie la commande CPIN, s'exécute au lancement).

NB : Lors du lancement du code, le débogueur se connectant à la carte Nucléo, le retour affiche des caractères spéciaux dû à l'incompréhension de la carte SIM.



The image shows a screenshot of a Tera Term VT window titled "COM5 - Tera Term VT". The window has a menu bar with "Fichier", "Edition", "Configuration", "Contrôle", "Fenêtre(W)", and "Aide". The main area displays a series of AT commands and responses from a SIM card. The text is as follows:

```
RDY
+CPIN: SIM PIN
AT+CPIN=0000
OK
+CPIN: READY
SMS DONE
PB DONE
AT+CPIN=0000
+CHE ERROR: incorrect password
AT+CSCS="GSM"
OK
AT+CMGF=1
OK
AT+CMGS="+33687449655"
> Connexion au reseau reussie
+CMGS: 75
OK
AT+MQTTSTART
OK
+MQTTSTART: 0
AT+MQTTACCQ=0,"client1"
OK
AT+MQTTCONNECT=0,"tcp://193.253.222.162:2021",60,1,"mqtt","snirpass"
OK
+MQTTCONNECT: 0,0
AT+MQTTTOPIC=0,12
>topic/projet
OK
AT+MQTTPAYLOAD=0,5
>49 50
OK
AT+MQTTPUB=0,0,60
OK
+MQTTPUB: 0,0
```

(figure 20 : échanges entre la Nucléo et la carte SIM)

### III - Conclusion

#### A) Ressenti sur le projet

Mon premier ressenti sur le projet fut la surprise, à laquelle s'est mêlée la curiosité. Puis, lors de la découverte du projet, je me suis retrouvé à devoir utiliser des nouvelles technologies pour moi, ce qui au final fut très intéressant.

Malgré une organisation compliquée au départ, j'ai finalement réussi à trouver un axe de travail, et ai réussi à aller le plus loin possible, compte tenu de la situation actuelle qui a entraînée un retard notable dans le projet.

#### B) Aboutissements

Au terme du temps imparti, le projet est dans sa globalité bien avancé. Le contexte actuel aura entraîné un retard pour moi, ainsi que pour le groupe qui devait m'envoyer des données, ainsi il m'est impossible de travailler avec les données que je suis supposé recevoir, et d'axer mon code sur la réception de ses données afin de l'automatiser pour qu'il se lance à chaque donnée reçue.

# Annexe

Annexe 1 : diagramme des cas d'utilisations

