Sebastien Doyez, Lucien Reyser, Akram Benazzouz

Rapport Projet Automatique et Réseaux

MEA 4

POLYTECH

Ce projet se concentre sur l'étude automatique et la mise en place d'une partie réseau, afin de contrôler un bras manipulateur. Celui-ci nous permettra de mettre en place des concepts de réseaux que nous avions vu lors de la MEA3 comme les trames de types UDP, ou encore la mise en place de systèmes client/ serveurs. De même, nous nous pencherons sur une analyse finale du système, avec un regard critique au point de vue automatique et de la mise en place d'un tel système, dans un milieu industriel.

Sommaire :

I) Mise en place du projet :

L'objectif de ce projet était de réaliser une simulation quasi réelle d'envoi de consigne à un robot manipulateur. Pour ce faire, nous allons utiliser un système de client - serveur afin d'envoyer une consigne en position, que nous convertirons en consigne en vitesse.

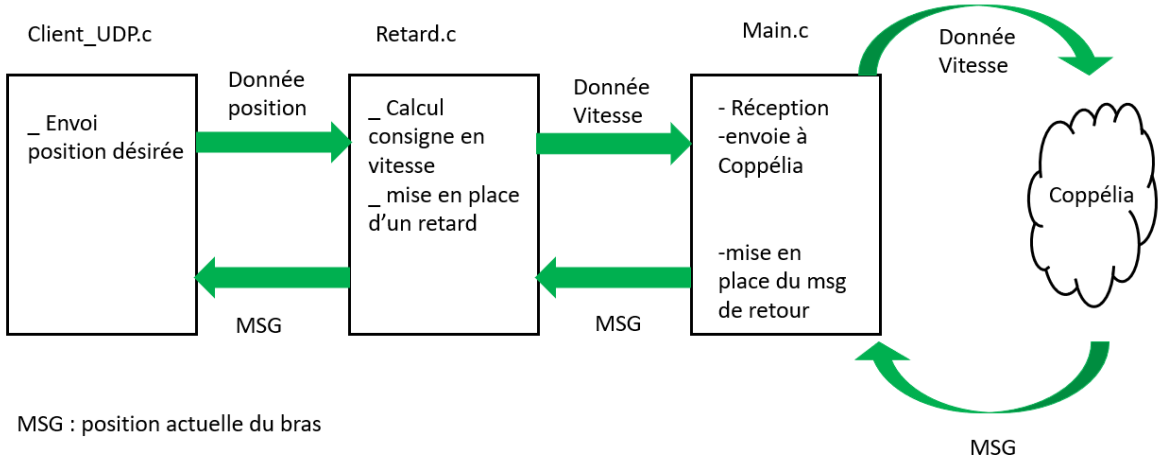


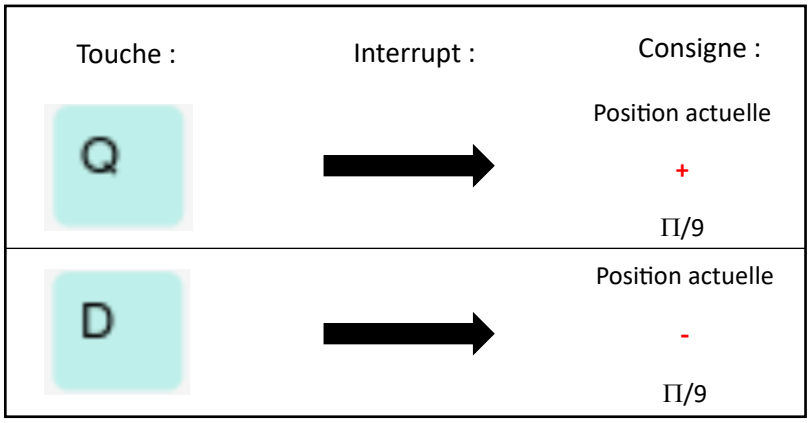
Figure 1 : Illustration de la mise en place du projet

Le but de Client_UDP.c est d'envoyer des consignes de positions désirées à intervalles de temps réguliers, ceci sera réalisé à l'aide d'un client uniquement. Nous n'utiliserons dans ce projet que de l'UDP, c'est-à-dire, des communications non bloquantes (on recevra bien un message de retour, qui sera la position actuelle du bras, mais si celle-ci n'arrive pas en temps voulu, cela ne bloquera pas les prochains envois). La réception de ces positions se fera par le serveur de Retard.c, ce fichier doit jouer le rôle d'une part de correspondance entre client_UDP et le Main, mais il doit aussi introduire un retard, que nous qualifierons de « constamment variable », et convertir la consigne en position en vitesse. Ensuite ce sera au client de ce fichier de renvoyer ces données au fichier main.c. Celui-ci fera la liaison code – Coppélia.

II) Le client :

Notre but étant de contrôler le robot en utilisant un serveur. Ainsi dans un premier temps, le client sera à l'origine de l'envoie de commandes, c'est une commande en position qui sera par la suite transformée en vitesse dans le retard.c et exécuté par le robot. Le client enverra une structure comportant les positions souhaitées ainsi qu'un « id » qui nous permet d'identifier la trame. Ainsi, toutes les secondes nous enverrons une trame comportant la commande que devra exécuter le robot et avant chaque envoi nous gardons en mémoire l'heure d'envoi de la trame afin de mesurer le retard. À la suite de cela, nous attendrons avec un timeout de recevoir une trame que nous avons fait le choix d'appeler « MSG » comportant la position actuelle du robot. Nous vérifions bien que l'identifiant du message reçu correspond à celui du message envoyé et si c'est le cas nous faisons la différence entre l'heure où le message est arrivé et l'heure où il est parti afin de mesurer le temps entre l'envoi de la trame et la réception de celle comportant les positions actuelles du robot.

Nous avons aussi fait le choix de pouvoir modifier la commande envoyée par appui des touches 'q' ou 'd' du clavier. Grâce à une fonction d'interrupt et un getchar nous réalisons l'envoi de commandes directement par ces touches du clavier.



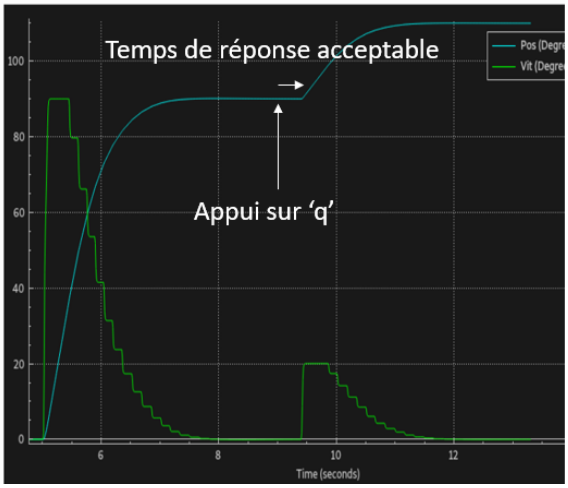


Figure 2 : illustration de l'interrupt au clavier via les touches 'q' ou 'd'

On remarque ici qu'il y a un temps de réponse (1 seconde ici) entre l'appui d'une touche et le changement de consigne sur CoppeliaSim. Celui-ci est du au retard injecté auquel on ajoute le temps de parcours des trames à travers les clients / serveurs.

III) Le Retard :

Étant en réseau local, il est difficile de constater le retard lié aux échanges client/serveur ainsi, pour se rapprocher de la réalité nous simulons un retard constamment variable. Qu'entendons-nous par un retard constamment variable ? Il s'agit d'un retard qui augmentera de manière aléatoire (avec un retard maximum de 1 seconde) toutes les 1 seconde de la manière suivante :

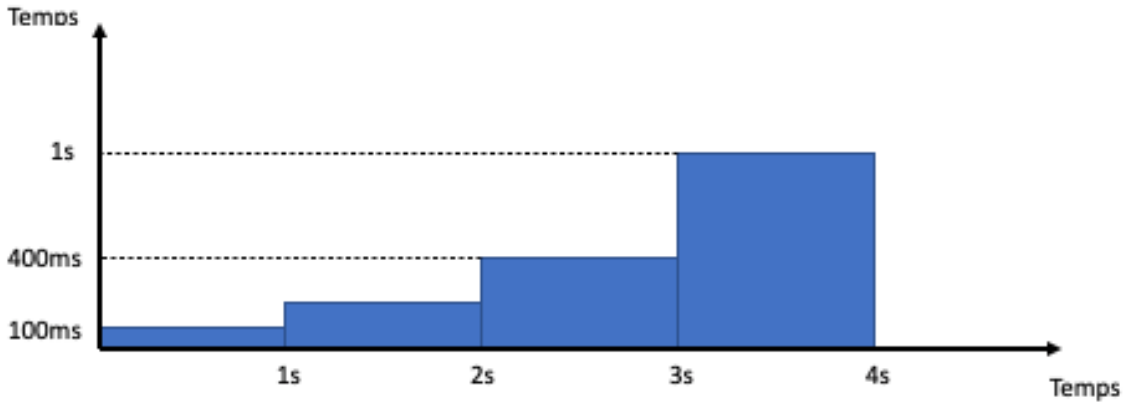
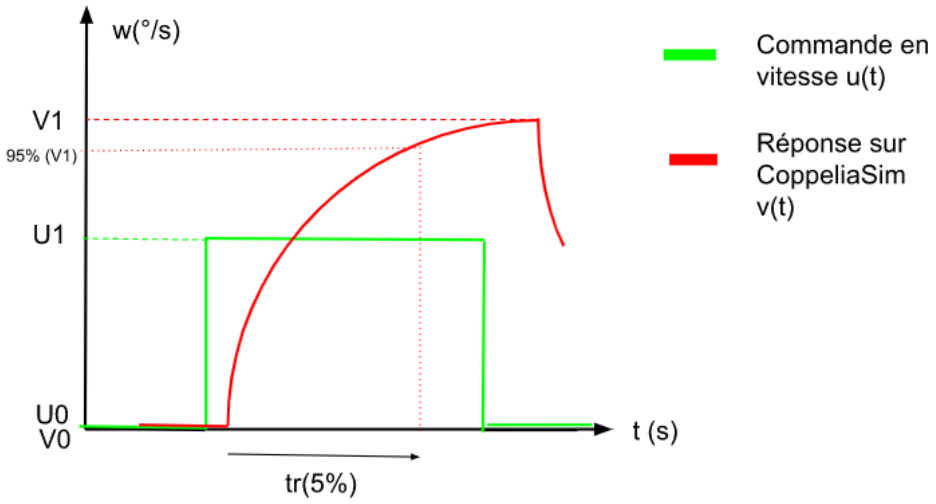


Figure 3 : illustration d'un retard « constamment variable »

Ainsi, notre but est de corriger ce retard car dans le cas contraire, il aura pour effet de créer des oscillations et donc de rendre le système instable. Ainsi, une fois la position désirée reçue, étant donné que la commande envoyée au robot sera une commande en vitesse, la première étape est de calculer la vitesse à appliquer grâce à la position désirée. Pour se faire, nous soustrayions la position actuelle du robot à la position désirée et nous multiplions le tout par un correcteur proportionnel Kc qui dépendant du retard du système et aura pour but d'annuler l'instabilité causée par le retard. Pour obtenir Kc, nous utilisons une look up table qui nous donne Kc en fonction du retard du système. En estimant G et tau, on peut déterminer les valeurs de Kc stabilisant le contrôle pour des retards variants.

Dans un premier temps, estimons G et du robot de la scène Coppelia Sim. Pour cela, commandons en vitesse un des moteurs du bras robot par un signal carré.



G sera calculé par G = $\frac{(V1-V0)}{(U1-U0)}$ et τ par sa relation avec le temps de réponse à 5%,

Soit tr5% = 3τ → τ = tr5% / 3

On relève une consigne de 188,3°/s envoyé à Coppelia et cette dernière affiche 188°/s. Notre Gain G vaut donc 0,998.

En outre,

0.95*188 = 178,6°

t (0°) = 1,39 s

t (178,6°) = 1,57 s

soit tr5% = 0,18 s

τ = 0,06 s

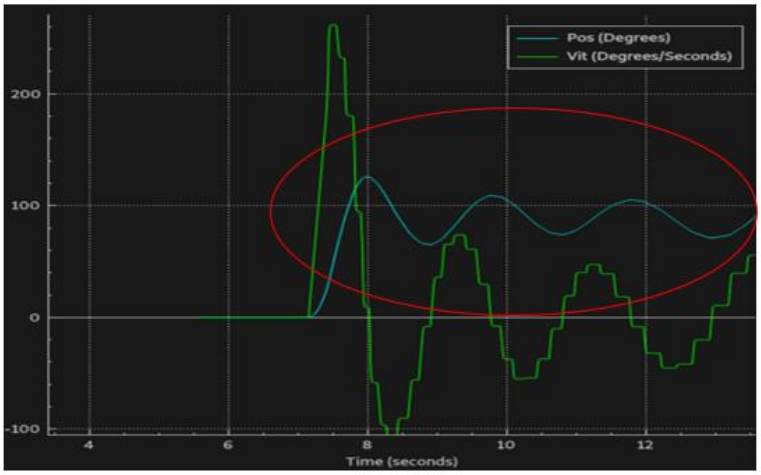
Les différentes valeurs de Kc de la look up table ont été obtenues via la formule suivante :

relation de stabilité pour un retard variable :

$$0 < K_c * G < [\tanh\left(\frac{Trpl}{2\tau}\right)]^{-1}$$

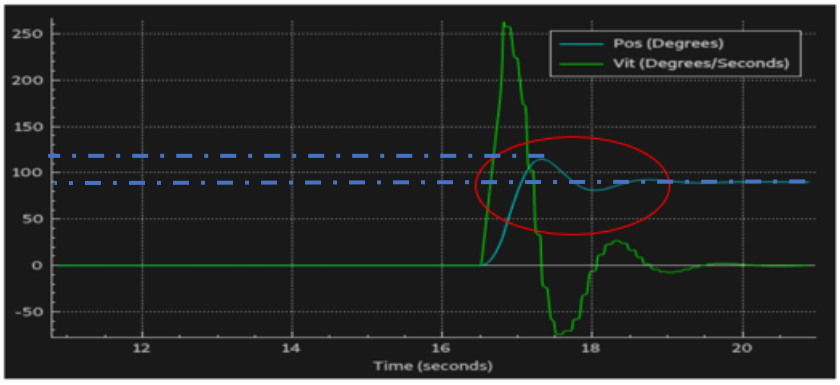
Et donc comme on peut le voir si dessous cela aura bel et bien l'effet escompté, cela supprimera l'instabilité du système.

Système non corrigé :



Système avec valeur de Kc non cohérente

Dépassement



Système corrigé :

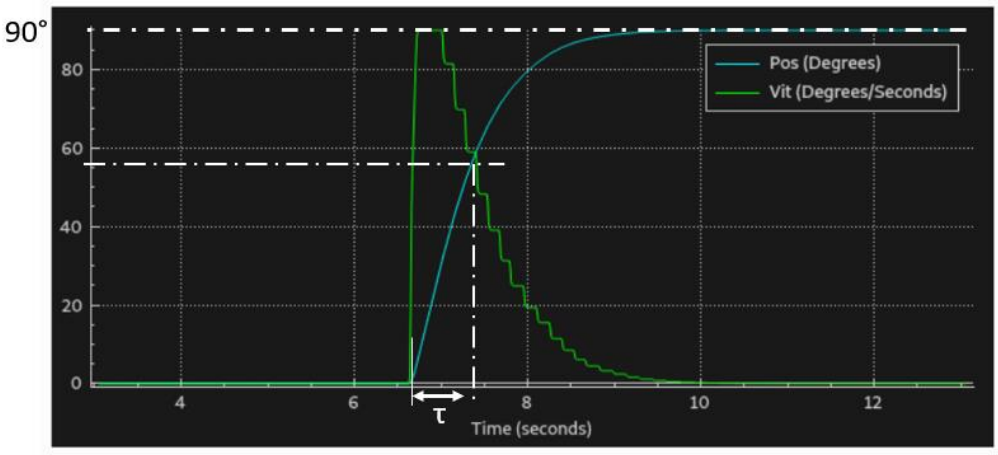


Figure 4 : Allure de la courbe en position de systèmes non corrigé et corrigé

Sur la figure 4, pour la première courbe, il s'agit d'un système non corrigé, nous retrouvons un système oscillant, il n'est pas instable d'un point de vue automatique, car il ne diverge pas dans le sens mathématique du terme, mais il est inexploitable et dangereux. Nous avons donc mis un correcteur proportionnel, non cohérent : on remarque que désormais, les oscillations s'atténuent au bout d'une seconde, mais de même le système est inexploitable : on ne peut pas imaginer des robot possédant des dépassements (sur le graphique 2 : 30° / 90° = 33,3 % de dépassement), travaillant dans le domaine industriels… Grâce à la relation de stabilité écrite à la page précédente, nous pouvons alors trouver une bonne valeur de Kc, qui va stabiliser le système. Bien évidemment, vu que le retard est constamment variable, la valeur de Kc doit évoluer dans le temps : nous avons donc créer une fonction de type look up table : qui prend en argument le retard injecté et renvoie la bonne valeur de Kc. On compare donc la valeur de retard, et comme nous avons calculé différentes valeurs de Kc, nous lui donnons la valeur qui fonctionnera le mieux.

Une fois le retard injecté et la vitesse à appliquer au système calculé, nous envoyons la commande à un deuxième serveur et nous attendons la trame de la réception de la position actuelle du robot avec un timeout afin de ne pas être bloquant et enfin, nous renvoyons cette position sur le premier serveur afin de la récupérer au niveau du client qui envoie les positions désirées.

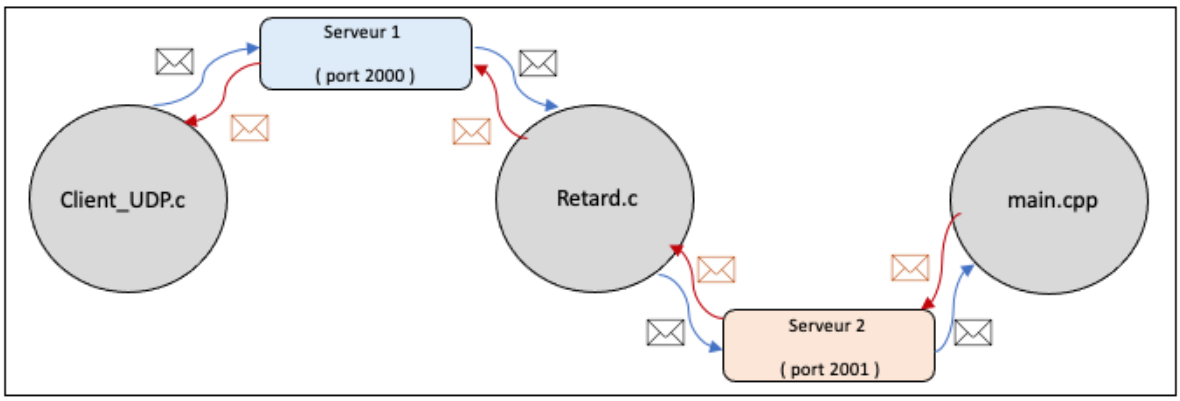


Figure 5 : illustration du fonctionnement du système

IV) Le main :

Le but du fichier main est d'établir la connexion avec CoppeliaSim. En effet, son fonctionnement est simple, celui-ci a pour but d'attendre l'arrivée d'un message, une fois ceci fait il envoi la commande en vitesse à CoppeliaSim, puis récupère la position actuelle du robot et renvoi cette position sur le serveur 2. Pour cela, nous avons donc dû mettre au point 2 fonctions : SetJointPos, qui sera la fonction utilisée pour envoyer les consignes en vitesses et une fonction getJoinPos, qui va nous servir à obtenir la position (angulaire) actuelle du bras manipulateur…

Afin que l'utilisateur puisse observer l'évolution du bras, nous afficherons les positions angulaires du bras dans le terminal, et grâce à la fonction fprintf(…), nous les écrirons dans un fichier texte, qui sera utilisé dans la partie « Analyse ».

Par la suite, nous utiliserons une nouvelle fois la fonction press(), qui lorsque la touche 'c' sera pressée, fermera les serveurs de communication. Nous avons également mis en place un fichier ./exe, qui est un exécutable, fait dans le but de simplifier l'utilisation du système que nous avons créé.

V) Analyse :

Le but de ce projet étant d'étudier l'effet des retards que peut provoquer l'utilisation d'un serveur sur un système mais également comment corriger cet effet. En effet, comme un retard est introduit, le bras va alors faire des dépassements plus ou moins important suivant la valeur de retard. Nous aurons donc des oscillations et cela peut provoquer d'important dégâts en fonction du système contrôlé. Ainsi il est primordial de ne pas les négliger et de mettre en place un correcteur capable de contrebalancer leurs effets.

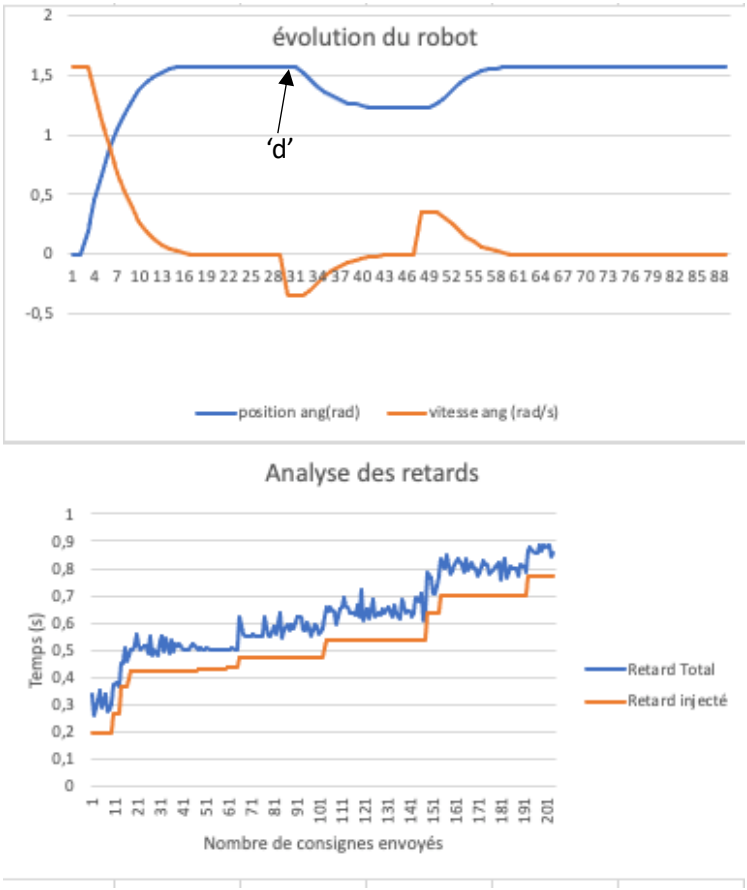


Figure 6 : graphiques représentant l'évolution du robot et du retard

Commentons les allures des courbes de la figure 6 : On remarque que pour la trajectoire de l'actionneur q0, il n'y a pas de dépassements, l'allure de la courbe de vitesse montre une évolution assez douce, sans à-coup (ce qui aurait pu endommager le bras manipulateur). Lorsque la touche 'd' est pressée, le temps de réponse du système est assez rapide (celui-ci dépend donc du retard injecté). Pour ce qui concerne les retards, on remarque le retard total est toujours supérieur à celui injecté, ce qui est cohérent. Le décalage entre les 2 courbes est du au temps de réponse de CoppeliaSim.

valeur moyenne du retard(retard.c) =	0,47376248
valeur moyenne du retard accumulé =	0,66212284
écart moyen =	0,183626
écart type retard(retard.c) =	0,17136021
écart type retard accumulé =	0,17581083

^

D'un point de vue mathématique, la valeur moyenne du retard injecté est de quasiment d'une demi seconde, ce qui est cohérent, celle du retard cumulé est de 0.66 s, soit environ 0.19 secondes d'écart moyen. L'écart type, sert à mesurer la dispersion, plus cette valeur est faible plus la population au sens statistique est homogène, ici la valeur de 0.17 pour les deux séries de retards : ce qui signifie que les retards sont assez proches les uns des autres et surtout que comme nous avons quasiment la même valeur d'écart type, nous avons le même étalement dans nos deux séries de retard. Cela est cohérent et cela nous amène à penser que l'écart entre le retard injecté et celui observé est principalement dû au temps de réponse de CoppeliaSim et des clients – Serveurs …

Cependant, avec un bon correcteur, il est possible d'y parvenir et donc d'obtenir un système stable en toutes circonstances (à l'exception du cas ou l'ordre d'envoi et de réception des trames est impacté par le retard, mais ici, cela ne pourra pas arriver puisque le retard est croissant). Comme on peut le voir sur les graphes, malgré un retard simulé constamment variable, on remarque que le système reste stable et n'est donc pas impacté par le retard.

Conclusion :

Ce projet nous a permis de mettre en œuvre des principes fondamentaux de réseaux que nous avions vu en 3ème année, tels que l'utilisation d'UDP, de système clients/ serveurs. D'un point de vue automatiques, ceci nous a permis de mettre en place un système réaliste : le fait de travailler en réseaux local, ne permettait pas de simuler les temps de diffusion des trames, c'est pour cela que nous avons mis en place un retard constamment variable, avec ce système fait en UDP, pour qu'il soit robuste à la perte de donnée… Nous avons également permis à l'utilisateur de changer la consigne du robot via l'appui de touches du clavier, ce qui nous à permit de mieux visualiser les temps de réponses du système.