Projet de Bureau d'étude

Yousre Ouali & Hicheme BEN GAIED
HE2B: ISIB
54667@etu.he2b.be & 58930@etu.he2b.be

Résumé

L'objectif de notre projet est de programmer le robot KUKA afin qu'il puisse taper sur un clavier. On a décidé de faire un compte Twitter au robot et lui faire Twitter des phrases. Ces phrases auront pour sujet Pokemon. Il nous dira quelles sont les évolutions des Pokemon.

Mots-clefs : Robot, KUKA, Clavier, Twitter, Pokemon, Souris, Bras mécanique, KRL, Python

I. Introduction

Notre objectif est de programmer un robot KUKA de manière à lui permettre d'écrire sur un clavier de manière autonome.

I. KUKA

Dans le cadre de mon projet d'étude, nous avons créé un système qui permet à un bras mécanique KUKA d'effectuer la saisie de texte sur un clavier. L'automatisation des tâches est devenue indispensable dans de nombreux domaines, et l'utilisation de bras mécaniques offre de nombreuses opportunités pour améliorer l'efficacité et la précision des processus.

KUKA est avant tout une entreprise de robotique. Elle est une société mondiale spécialisée dans l'automatisation. Le siège de l'entreprise est situé en Allemagne.

II. Description des différentes parties du projet

Lors de ce projet, nous sommes passer par plusieurs étapes. Voici une description brève des étapes ;

- Assurer une connexion continue avec le robot.
- Configurer les entrées du robot pour recevoir des coordonnées.
- Programmer en KRL afin que le robot puisse utiliser ces coordonnées pour effectuer des mouvements.
- Déplacer le robot en utilisant les coordonnées envoyées.
- Prendre les coordonnées des touches du clavier.
- Contrôler les mouvements du robot de manière à ce qu'il puisse taper sur le clavier en lui fournissant simplement la phrase à écrire.
- Établir les étapes initiales nécessaires pour que le robot sache où appuyer afin d'effectuer une publication.

III. Matériel et Méthode

Les outils utilisés sont les suivants ;

- Un robot KUKA
- Un ordinateur
- Un clavier mécanique
- Un support pour le clavier

En ce qui concerne le language utilisé lors de ce projet. On a dû utiliser deux languages de programmation, **Python** et le **KRL**.

Python a été utilisé pour instancié le serveur. Ce serveur permet de maintenir la connexion, recevoir/envoyer des coordonnées, vérifier des coordonnées et séquencer la phrase que le robot doit taper. Les informations que le serveur envoie au robot est sous format **XML**.

Python a été utilisé pour créer une instance du serveur. Ce serveur assure la maintenance de la connexion, la réception et l'envoi des coordonnées, la vérification des coordonnées et la séquentialisation de la phrase que le robot doit taper. Les informations transmises du serveur au robot sont formatées en **XML**.

Le support de clavier est indispensable pour respecter la zone de travail du robot. En effet, le robot dispose d'une zone de travail définie. Lorsqu'il se trouve en dehors de cette zone, le robot devient inutilisable. Cette zone de travail est essentielle pour garantir la sécurité. Elle permet notamment d'éviter tout contact indésirable entre le robot et le plan de travail, par exemple.

Notez que la phrase que le robot doit taper ne doit pas être trop longue. Si la phrase est excessive en longueur, cela prolongera le temps nécessaire pour que le robot la tape.

I. Connexion

Pour la connexion, le robot est connecté via un **VLAN**, ce qui signifie qu'il n'est pas directement connecté au réseau principal. Afin de communiquer avec le robot, il est nécessaire de se connecter au VLAN spécifique associé.

Une fois que la communication entre le serveur et le robot est établie, on peut passer à l'étape suivante, qui consiste à créer un fichier XML. Ce fichier XML contiendra toutes les informations nécessaires pour établir la connexion, telles que l'adresse IP du serveur, le port utilisé par le serveur et le protocole de communication utilisé.

Enfin, une fois que le fichier XML est correctement configuré, nous pouvons passer au programme KRL. Nous avons créé un objet RSI (Robot Sensor Interface) de type **ST_ETHERNET**. Cet objet spécifique au langage KRL permet d'initialiser la connexion. Une fois que la connexion est établie, le robot commence par envoyer le premier fichier XML au serveur. Ce premier fichier contient des informations telles que les coordonnées ainsi qu'un IPOC (Internal Point of Control). Cette valeur IPOC est une séquence numérique qui doit être renvoyée au robot dans un délai imparti. Si cette valeur n'est pas renvoyée dans les plus brefs délais, la connexion est interrompue.

II. Configuration du fichier XML

Comme expliqué précédement, le fichier XML permet de configuré la connexion du robot au serveur. Toutefois, ce fichier contient d'autre informations. Voyons voir de plus près l'architecture de ce fichier.

La balise racine de ce fichier se nomme "**ROOT**". Dans la balise racine se trouve 3 autres balises. Il y a une balise qui se nomme "**CONFIG**", une autre "**SEND**" et la dernière s'appelle "**RECEIVE**".

La balise **CONFIG** permet de configurer les paramètres de communication. On va y retrouver 6 autres balises ;

- "**IP_NUMBER**" contenant l'addresse IP du serveur.
- "**PORT**" permettant d'avoir le port utilisé par le serveur.
- "**PROTOCOL**" qui défini le protocole utilisé pour la communication. On a deux types de protocoles utilisable, on a le droit d'utiliser le protocole **TCP** ou **UDP**.
- "**SENDTYPE**" est l'identifieur du système externe. Cette valeur va être identifier pour chaques paquets reçu au robot.
- "**PROTOCOLLENGTH**" est une balise permettant d'activier la transmission de la longueur de byte du protocole. Elle n'a que deux valeurs possibles ("**ON**" ou "**OFF**").
- Et enfin, la balise "**ONLYSEND**" permet de définir la direction des échanges de donnée. Elle peut être mis à "**TRUE**", ceci permet au robot d'envoyer des données

mais pas d'en recevoir. Tandis que, lorsqu'il est mis à "**FALSE**", alors le système peut envoyer et recevoir des données.

Concernant la balise **PROTOCOLLENGTH**, il a été initialisé à **OFF** et la balise **ONLYSEND** a été initialisé à **FALSE**.

Ensuite, la balise **SEND** permet de définir ce que le robot peut envoyer. Cette balise contient une sous-balise "**ELEMENTS**". Cette balise contient aussi d'autres balises "**ELEMENT**". La balise **ELEMENT** a des attributs tels que ;

- **TAG** est le nom du tag qui sera généré par le robot.
- **TYPE** est le type de donnée.
- **INDX** est le numéro de sorite de l'objet.
- Et enfin, **UNIT** est l'unité utilisé pour la donnée.

Pour ce projet, on avait seulement besoin des coordonnées du robot. Le système dispose déjà d'un objet préconfiguré appelé "**DEF_RIst**". C'est dans l'attribut "TAG" qu'on spécifie le nom de cet objet. Le type de cet objet est "DOUBLE", le numéro de sortie est "INTERNAL" et l'unité est "0". Cela nous permettra de récupérer les coordonnées du robot de manière précise et cohérente dans votre programme.

Remarque, il existe plein d'autres mots-clés comme **DEF_RSol** (envoie les commandes de positions cartésienne), ou le **DEF_MACur** (envoie les courants moteur du robot de l'axe A1 à A6).

Et enfin, la balise **RECEIVE** est la balise qui définie la sortie du robot. Cette balise a la même architecture que la balise **SEND**. Toutefois, la balise **ELEMENT** contient un attribut en plus, c'est le **HOLDON**. Ce nouveau attribut va définir le comportement de l'objet reçu au robot lorsque celle-ci à une erreur. Elle a deux valeurs possibles ;

- **0** permet de réinitialiser la valeurs reçu.
- **1** permet de maintenir l'ancienne valeur valide alors que la nouvelle valeur a une erreur.

Dans ce projet, tous les eléments ont un **HOLDON** à **1** afin d'assurer une localisation toujour valide.

III. Programmation en KRL

Passons maintenant dans le language KRL, on va définir les différents principes et outils utilisés dans le programme.

Au début du programme, on commence d'abord par déclarer les variables utiles. Une variable de type **RSIERR** est déclaré. Cette variable permet, en autre, de **récupérer** la réponse d'une commande. Des variables de type **INT** seront utilisées afin de stocker des entiers. Il y aura un entier (nommé **hEthernet**) contenant l'identifiant de l'objet permettant l'accés à l'objet **RSI**. Cette valeur est automatiquement assigné par le système lorsque l'objet RSI est créer. D'autres entiers seront utilisés pour identifié l'objet RSI. Et enfin, une variable de type **FRAME** sera déclaré. C'est une variable permettant de faire la liaison avec le capteur de coordonnée du système. Elle permet notamment de changer les coordonnées selon des nouvelles coordonnées reçu par le robot.

Une fois que les déclarations de variables sont terminées, on peut passer à la configuration de l'objet RSIERR en utilisant les méthodes mises à disposition.

La communication entre le controlleur du robot et un système externe est implémenté en utilisant l'objet **RSI ST_ETHERNET**. Cette objet doit être crée et configuré dans le programme KRL afin de établir la communication. Pour activer la communication entre ces systèmes, l'utilisateur doit créer et configurer le fichier XML. Ce fichier de configuration est spécifié et charger lorsque l'objet RSI ST_ETHERNET est crée. Cette objet prend trois paramètres.

Le premier paramètre est l'identifiant de l'objet permettant l'accés à l'objet **RSI**. Dans notre cas, cette variable est appellée **hEthernet**.

Le deuxième paramètre est un entier correspondant au nombre de containeur dans lequel l'objet RSI est crée. Nous ne voulons pas créer de containeur, on va donc préciser qu'aucun containeur doit être crée.

Et enfin, le troisème paramétre est le chemin d'accés vers le fichier de configuration XML.

L'objet **RSI ST_ETHERNET** peut être configuré à l'aide de la commande **ST_SETPARAM**. Dans notre cas, la seule configuration effectuée concerne le nombre maximum de paquets de données pouvant arriver en retard au contrôleur du robot. Ce paramètre est appelé **eERXmaxLatePackages**, et on l'a défini à 500. Cela signifie que le nombre maximal de paquets de données autorisés à arriver en retard est de 500. Cette configuration permet de gérer les éventuels retards de transmission des données tout en maintenant la stabilité et la performance de la communication entre le serveur et le robot.

Lorsque la partie concernant la connexion est terminé, on peut commencer à mapper les entrés de données du robot. Pour réaliser cela, on utilise l'objet **RSI ST_MAP2SEN_PREA**. Cette objet permet donc de rendre disponible la lecture de donnée que le système externe envoie. Il prend cinq paramètres ;

— Le premier paramètre est l'identifiant de l'objet. C'est donc un entier. Dans le programme KRL, il sera nommé **hMap2Prea**.

— Le deuxième paramètre est le numéro du containeur de l'objet RSI, dans notre cas, c'est le containeur 0.

— Le troisème paramètre est l'identifiant de l'objet permettant l'accés à l'objet **RSI** (hEthernet).

— Ensuite, le quatrième paramètre est l'index de la source de signal. Ceci est directement lié à la configuration du fichier XML, dans la balise **RECEIVE**.

— Et enfin, le cinquième paramètre est l'indice de la variable de système **$SEN_PREA[]**.

Remarque importante, les variables seront accessible avec la variables du système **$SEN_PREA[X]**. Où le terme "X" est l'indice qu'on utilise pour accéder à la variable reçu.

Dans le but d'assurer la sécurité matérielle, on active un cycle de correction qui ajuste les mouvements du robot en fonction du système de référence utilisé, en l'occurrence le système #WORLD. Pour activer cette correction, on utilise la commande **ST_ON1(#WORLD,1)**. Cette commande indique au robot de mettre en marche le cycle de correction lié au système de référence #WORLD, ce qui permet d'obtenir des mouvements plus précis et adaptés à notre configuration spécifique.

On peut maintenant passer à la création d'une boucle permettant de récupérer les nouvelles coordonnées envoyées par le système externe. Dans cette boucle, on utilise une variable de type **FRAME**. Cette variable va contenir les nouvelles coordonnées à l'aide de la variable $SEN_PREA[X]. Une fois les coordonnées initialisées, on peut effectuer un mouvement de type PTP (Point To Point). Ce type de mouvement permet de se déplacer d'un point à un autre de manière directe, sans suivre une trajectoire linéaire. Il nous permet simplement de nous déplacer vers un point dans l'espace défini par les coordonnées spécifiées.

IV. Mouvement

Dans cette partie, nous allons discuter des différents types de mouvements effectués par le bras Kuka afin de répondre aux différents besoins de notre projet. Les mouvements que nous allons aborder vont permettre d'écrire sur le clavier et de publier le tweet.

IV.1 Mouvement clavier

Le mouvement du clavier a été décomposé en 3 parties. La première étape consiste à se positionner au-dessus de la touche à écrire. Le bras Kuka se déplace en X et Y uniquement

pour cette phase. Une fois positionné au-dessus de la touche, le bras effectue un déplacement en Z pour appuyer sur la touche. Après avoir pressé la touche, le bras effectue à nouveau un déplacement en Z pour se retirer de la touche. Ce cycle est répété jusqu'à ce que le bras ait fini d'écrire la phrase souhaitée. Les coordonnées de chaque touche sont stockées dans un dictionnaire appelé "Dico" dans le fichier MovementManager.py. Ces coordonnées sont transmises au robot en fonction de la touche à écrire. Elles permettent également d'effectuer plusieurs vérifications en comparant la position à atteindre et la position actuelle du bras :

1. Vérifier que le robot se trouve bien au-dessus de la touche concernée.
2. Vérifier que le robot a bien appuyé sur la touche du clavier.
3. Vérifier que le robot s'est bien relevé après l'appui.

Ces comparaisons permettent de garantir la précision et l'exactitude des mouvements à effectuer par le bras Kuka lors de l'écriture sur le clavier.

IV.2 Soumission du tweet

Une fois la phrase entièrement écrite, le robot doit maintenant soumettre le tweet. Pour cela, nous utilisons la touche de tabulation huit fois afin de sélectionner le bouton "Tweet". Une fois ce bouton sélectionné, nous ordonnons au robot de se diriger vers la touche "Enter" du clavier et d'appuyer dessus pour soumettre notre tweet. Les coordonnées de la touche "Enter" et "Tabulation" sont également stockées dans un dictionnaire appelé "MousePos", qui se trouve dans le fichier MovementManager.py. Les mêmes vérifications que celles effectuées pour l'écriture sur le clavier sont appliquées pour soumettre le tweet. Une fois notre tweet soumis, le robot peut écrire un autre message s'il le souhaite en répétant le même schéma.

V. Twitter

Lors du lancement du programme, le robot va directement appuyer sur la touche "N". Cette touche est un raccourci proposé par Twitter pour ouvrir rapidement et facilement la fenêtre d'écriture des tweets. Les coordonnées permettant d'utiliser cette option sont stockées dans le dictionnaire "MousePos" sous le nom de "new", situé dans le fichier MovementManager.py. Une fois la fenêtre ouverte, le robot va commencer à rédiger son tweet, puis le soumettre.

IV. Problème rencontré

Durant notre projet, nous avons rencontré un problème lié à l'utilisation de la souris par le bras. En effet, notre objectif était d'utiliser la souris pour déplacer le curseur vers le bouton "Tweet" afin de soumettre le tweet rédigé par le robot, au lieu d'utiliser les huit tabulations. Pour cela, nous avons développé un boîtier adapté à la tête du robot et implémenté le code nécessaire pour cette fonctionnalité. Lors de nos tests, cela fonctionnait très bien, mais lorsque nous sommes passés en mode T2, qui permet au robot d'aller plus vite, un problème majeur et évident est apparu. La distance parcourue par le curseur variait en fonction de la vitesse. Malheureusement, nous n'avions pas eu la possibilité de calibrer la souris avec le mode T2, car nous ne disposions pas de la clé nécessaire pour activer ce mode. Nous pensons toutefois qu'il est possible de résoudre ce problème en ajoutant du code dans le fichier contenant les instructions KRL, de manière à imposer une vitesse au robot lors de l'utilisation de la souris.

Cependant, par manque de temps, nous avons finalement opté pour la méthode des tabulations et de la touche Enter afin de publier nos tweets.

V. Résultat final

Le résultat final de notre projet est que le robot KUKA est capable de taper sur les touches du clavier en fonction de la phrase fournie par le système externe. Dans notre

cas, nous avons mis en place une configuration où le serveur (système externe) effectue une requête à une API spécifique. Cette API fournit des informations sur les évolutions des Pokémon. Ainsi, lorsque le serveur reçoit une requête pour une évolution spécifique, il transmet cette information au robot KUKA qui exécute les mouvements nécessaires pour taper la phrase correspondante sur le clavier. Cela permet d'automatiser le processus d'envoi de commandes liées aux évolutions des Pokémon en utilisant le robot.

En plus de pouvoir taper sur les touches du clavier, le robot a également la capacité de déplacer la souris pour soumettre un Tweet. Cependant, ce type de mouvement est limité au niveau T1 du bras mécanique.

VI. Améliorations possibles

Il y a plusieurs améliorations possibles pour le projet, notamment :

1. Équiper le robot d'une caméra pour reconnaître les lettres du clavier : Cela permettrait d'éviter d'avoir à spécifier les coordonnées de chaque lettre individuellement. Le robot serait capable de reconnaître les lettres visuellement et d'interagir avec le clavier de manière plus intuitive.
2. Ajouter un moteur au bout du robot pour une frappe plus rapide : En équipant le bout du robot d'un moteur, il pourrait effectuer des mouvements de frappe sur les touches du clavier de manière plus rapide et précise. Cela éliminerait la nécessité de mouvements complexes sur l'axe Z, ce qui accélérerait le processus de rédaction de la phrase.
3. Générer des phrases avec une IA : Cette amélioration a été envisagée, mais il y a eu des obstacles liés aux coûts. L'idée était d'utiliser une IA pour générer des phrases de manière automatique, offrant ainsi une variété de sujets et de phrases uniques à chaque tweet. Cependant, l'utilisation d'une IA de ce type nécessite souvent un quota de requêtes payantes.
4. Équilibrer le mouvement de la souris avec le niveau T2 du robot KUKA : L'amélioration serait de calibrer et de synchroniser les mouvements de la souris avec le mode T2 du robot. Cela permettrait au robot de se déplacer plus rapidement tout en maintenant une précision suffisante lors de l'utilisation de la souris pour des actions telles que cliquer sur des boutons ou naviguer dans une interface graphique.
5. Permettre au robot de reconnaître sa position sur l'écran, ce qui lui permettrait de déplacer la souris de manière autonome. Cela impliquerait d'intégrer des fonctionnalités de vision par ordinateur et de traitement d'image au système du robot. En utilisant des techniques telles que la détection d'objets et la reconnaissance de motifs, le robot serait capable de localiser sa position relative sur l'écran et de naviguer avec précision. Cela faciliterait l'interaction du robot avec des interfaces graphiques complexes, lui permettant d'effectuer des actions spécifiques à des emplacements précis sur l'écran sans nécessiter de coordonnées préprogrammées.

Ces améliorations permettraient d'optimiser davantage le projet en termes de vitesse, précision et automatisation des tâches liées à l'utilisation du clavier et de la souris par le robot KUKA.

Références

[Kuka Roboter] 2009
KUKA System technology KUKA.RobotSensorInterface 2.3 / For KUKA System Software 5.4, 5.5, 5.6, 7.0