Pilotage d'un robot à travers une application desktop

Illustration

Jaccard Anthony
Impasse du Saugy 1
1555 Villarzel
Anthony.JACCARD@cpnv.ch

SI-CA2a

07.05.2019

Table des matières

1 Introduction

1.1 Cadre, description et motivation

Le but de ce projet sera de créer une application permettant de piloter un robot et d'afficher une photo prise par ce dernier. J'ai choisi ce projet car je suis passionné de robotique depuis presque aussi longtemps que je suis passionné d'informatique. Je compte d'ailleurs poursuivre mes études dans l'informatique embarquée car c'est le domaine de l'informatique qui se rapproche le plus de la robotique. En ce sens-là, ce projet est pertinent car il me permettra notamment de me familiariser avec certaines technologies qui me seront certainement utiles dans mon futur parcours. De plus, lors d'une discussion avec un professeur du CPNV, ce dernier a soulevé le fait que des robots étaient utilisés dans le CPNV lors des portes ouvertes et qu'il pourrait être intéressant d'utiliser un robot fabriqué par un élève au lieu de n'utiliser que des robots grands publics. Comme la fabrication du robot et le développement de l'application auraient étés un objectif trop ambitieux pour les 90 heures du TPI, le robot a été fabriqué en grande partie dans le cadre du module de robotique et terminé hors des heures de cours.

1.2 Organisation

Élève : Anthony Jaccard
E-Mail : anthony.jaccard@cpnv.ch
Téléphone : 079/460.66.48

Chef de projet : Pascal Hurni
E-Mail : pascal.hurni@cpnv.ch

Expert 1 : Christophe Regamey
E-Mail : christophe.regamey@bluewin.ch

Expert 2 : Yves Bertino
E-Mail : yves@bertino.ch

1.3 Objectif

Créer une application desktop permettant de :

- Piloter en temps réel un robot accessible via une connexion SSH locale
- Demander au robot de prendre une photo et d'afficher celle-ci dans l'application
- Indiquer l'état de la connexion avec le robot
- Conserver des logs des communications avec le robot

1.4 Planification initiale

Date de début : 07.05.2019

Date de fin : 05.06.2019
La planification initiale détaillée a été envoyée au chef de projet ainsi qu'aux experts le premier jour

2 Analyse

2.1 Cahier des charges détaillé

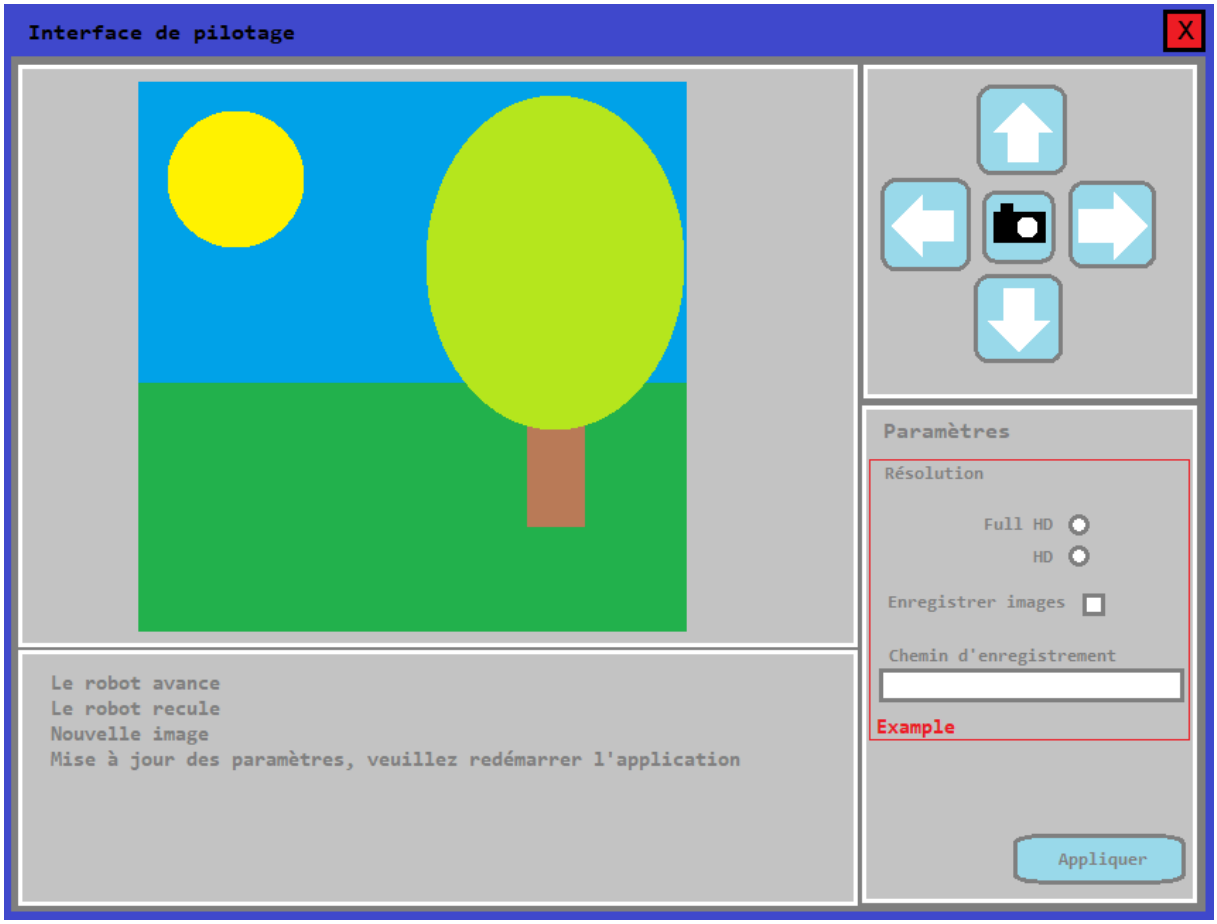
2.1.1 Uses Cases

2.1.2 Scenarii

Action	Effet
Clic sur un bouton directionnel (haut, bas, gauche, droite) ou appui sur une touche directionnelle du clavier (haut, bas, gauche, droite)	Mouvement du robot correspondant (avant, arrière, rotation anti-horaire, rotation horaire)
Combinaison de deux touches du clavier (haut/bas et droite/gauche)	Mouvement du robot correspondant (virage avant droite, virage avant gauche, virage arrière droite, virage arrière gauche)

Clic sur le bouton «Prise de vue»	Prise de vue et affichage de l'image
Modification d'un champ de texte	Modification d'un paramètre
Clic sur le bouton « Appliquer »	Envoi des modifications des paramètres au robot

2.1.3 Interface



2.1.4 Schéma UML

2.2 Stratégie de test

2.2.1 Tests fonctionnels

- Tests de pilotage

S'assurer que le robot répond comme il faut aux instructions de pilotage (interface ou clavier)

- Tests de prise de vue

S'assurer que le robot prend une photo et que cette dernière est bien affichée dans l'interface graphique

- Tests de paramétrage

S'assurer que les paramètres sont bien pris en compte et sont enregistrés sur le bon périphérique (ordinateur ou robot)

2.2.2 Tests de performances

- Test de réactivité au pilotage

La latence entre le clic sur une icône de pilotage ou l'appui d'une touche directionnelle et la réaction du robot doit être suffisamment faible pour permettre de bonnes conditions de pilotage

- Test de réactivité à la prise de vue

La latence entre le clic sur l'icône caméra ou l'appui sur la barre espace et l'apparition de l'image sur l'écran doit rester faible

2.3 Budget initial

Le TPI consistant à développer une application, il n'engendrera aucun coût. La fabrication du robot en revanche a nécessité l'achat de deux servomoteurs, deux roues, une batterie, un chargeur ainsi qu'un peu de matériel divers pour la fabrication du support. Ces coûts ont cependant déjà étés pris en charge par le CPNV.

2.4 Planification

Le fichier de planification est fourni en annexe car l'intégrer ici aurait rendu sa lecture compliquée

3 Conception

3.1 Dossier de conception

- Matériel : Un robot (conception et fonctionnalités décrites plus bas), un chargeur Li-Po, un point d'accès Wifi, un ordinateur Windows 10 du CPNV, une carte réseau Wifi USB (TP-LINK T2UH), un multimètre
- Logiciels :
  - Programmation : Arduino IDE, Visual Studio Code + extension python
  - Documentation : Word, Excel, Visio, Paint
  - Réseau : OpenSSH
- Langage : Python

3.2 Le Robot

3.2.1 Conception

- Composants matériels : Raspberry Pi 2 B, Arduino Uno, 2x servomoteurs continus, batterie Li-Po 11.1V, régulateur à découpage, caméra
- Logiciel : Raspbian (Raspberry Pi), micrologiciel

Le robot est constitué d'un Raspberry Pi connecté grâce à une connexion I2C à une carte Arduino Uno. Le Raspberry Pi sert d'interface avec le monde extérieur

et envoie des commandes à l'Arduino qui se charge de piloter les servomoteurs en conséquence.

3.2.2 Fonctionnalités pré-existantes

En l'état, il est possible de se connecter en SSH au robot grâce au Raspberry Pi et de s'en servir pour lui envoyer des ordres sous forme de commandes. En pratique, ces ordres sont simplement des commandes I2C (i2cset) camouflées sous une forme plus simple grâce aux alias. Ces commandes sont :

- **idle** : met le robot à l'arrêt
- **fRun** : (forward run) fait avancer le robot
- **bRun** : (backward run) fait reculer le robot
- **lTurn** : (left turn) fait tourner le robot dans le sens anti-horaire
- **rTurn** : (right turn) fait tourner le robot dans le sens horaire
- **fTurnR** : (forward turn right) fait faire au robot un virage à droite tout en avançant
- **fTurnL** : (forward turn left) fait faire au robot un virage à gauche tout en avançant
- **bTurnR** : (backward turn right) fait faire au robot un virage à droite tout en reculant
- **bTurnL** : (backward turn left) fait faire au robot un virage à gauche tout en reculant

3.2.3 Fonctionnalités à implémenter

Par défaut, aucune commande n'existe pour prendre une photo à l'aide de la caméra du Raspberry Pi et de la stocker. Il faudra donc, dans le cadre de ce TPI, créer un script Python permettant de le faire. De plus, ce script devra pouvoir faire appel à un fichier de configuration qui pourra être modifié à distance par l'application afin de configurer la prise de vue (résolution, dossier d'enregistrement des clichés… etc)

3.3 Protocole de prise de vue en réseau

3.3.1 Description

Le protocole permet de demander au robot de prendre une photo et de la stocker dans un dossier connu puis d'aller la récupérer à travers le réseau et de l'enregistrer sur la machine exécutant l'application

3.3.2 Fonctionnement

L'ordinateur sur lequel est exécuté l'application exécute via SSH le script python du robot qui lui fait prendre une photo et la stocker dans un dossier connu. Une fois cela fait, l'ordinateur copie l'image dans le dossier renseigné dans la configuration de l'application à l'aide du protocole SCP et prévient le contrôleur qu'une nouvelle image à afficher est disponible.

3.4 Moyens pour la capture vidéo (non-implémentés)

3.4.1 Capture en rafale

Le script du raspberry Pi ne prend plus une image à la fois mais prend des images à intervalle régulier et les stocke dans le dossier habituel lorsqu'il est activé.
Sur l'ordinateur, un script est également appelé à intervalle régulier et va vérifier si une nouvelle image a été enregistrée. Dès qu'il en détecte une nouvelle, il la récupère et prévient le contrôleur qu'il y a une nouvelle image à afficher.

3.4.2 Transfert continu

L'API de la caméra du Raspberry Pi gère l'écriture dans les flux et Python permet d'utiliser les interfaces réseau pour créer de tels flux. En théorie, il est donc possible de créer un flux continu entre le robot et l'ordinateur ce qui permettrait d'avoir un retour vidéo aussi fluide que la connexion le permet.

4 Réalisation

4.1 Dossier de réalisation

4.1.1 Arborescence du programme

- src : dossier racine de l'application en elle-même
  - img : contient les images utilisées pour l'interface de l'application
  - app : contient les fichiers de configuration et de log pour l'application
    - images : dossier de stockage des images récupérées du robot

4.1.2 Matériel

- 1 robot
- 1 point d'accès Wifi
- Un ordinateur avec carte Wifi
- Matériel d'entretien du robot (multimètre, chargeur)

4.1.3 Code source

- **PilotApp.py** : Racine du programme
- App.py : Description de la classe App qui décrit les fonctionnalités et le comportement du programme lui-même
- **View.py** : Description de la classe View et des classe dérivées ImageView et ControlView qui décrivent les différentes vues composant l'interface graphique
- Setting.py : Description de la classe Setting décrivant l'apparence et le comportement des paramètres de l'application et du robot
- **NetControls.py** : Description de la classe NetControls qui commande le robot à distance
- **NetCamera.py** : Description de la classe NetCamera qui ordonne la prise de vue au robot et récupère le fichier

- **SettingSaver.py** : Description de la classe SettingSaver qui se charge de récupérer la valeur des paramètres et de les enregistrer dans les fichiers de configuration de l'application et du robot
- **Logger.py** : Description de la classe Logger qui enregistre les logs dans le fichier prévu à cet effet

4.1.4 Technologies et librairies utilisées

4.1.4.1 Python

Python est un langage de programmation haut niveau, interprété et orienté objet avec une syntaxe simple basée sur l'indentation pour la définition des blocs de code. Il est extensible via l'utilisation de modules.

L'application développée dans le cadre de ce TPI ainsi que les scripts utilisés sur le Raspberry Pi seront entièrement programmés en Python 3 et seuls les modules standards seront utilisés.

4.1.4.2 SSH

SSH est un protocole de console à distance dérivé de Telnet implémentant en plus des mécanismes d'encryption des communications (par mot de passe ou clés asymétriques) afin d'en améliorer la sécurité (d'où le nom Secure SHell). Il permet donc de prendre le contrôle d'un ordinateur à distance à travers une interface en ligne de commande.

Dans mon programme, SSH est utilisé afin de permettre à l'application de se connecter au robot (plus précisément au Raspberry Pi) et de lui envoyer les ordres de pilotage ou de prise de vue afin que pour détecter l'enregistrement d'une nouvelle image sur le robot. Afin de simplifier la connexion, l'ouverture de session se fait à l'aide d'une paire de clé dont l'exemplaire public se trouve déjà sur le robot. La clé privée doit donc être transmise en même temps que l'application afin que la connexion puisse se faire.

4.1.4.3 SCP

SCP (Secure CoPy) est un utilitaire de copie sur le réseau basé sur le protocole RCP et sécurisé à l'aide de SSH. Il permet de récupérer un fichier enregistré sur un ordinateur distant de manière sécurisée car encryptée.

Dans le cadre de l'application, il est utilisé afin de récupérer les clichés pris par le Raspberry Pi afin qu'ils puissent être affichés dans l'interface de l'application.

4.1.4.4 Subprocess (Python)

Le module subprocess est un module standard de Python permettant l'exécution de programmes externes à l'application ainsi que la redirection des flux d'entrées et sorties de celle-ci afin de pouvoir les utiliser de manière automatisée au sein même de l'application.

L'utilisation des programmes SSH et SCP décrit plus haut est rendue possible par l'utilisation des subprocess. Sans ces derniers, il aurait fallu utiliser des librairies (dont certaines non-intégrées par défaut à Python) permettant l'utilisation de ces protocoles par le langage lui-même et qui n'ont pas toutes la réputation d'être très stables, intuitives voire simplement compatibles avec mon infrastructure.

4.1.4.5 Tkinter (Python)

Tkinter est un module standard de Python servant d'interface entre le langage et Tk, une bibliothèque d'interface graphique multi-plateforme originellement créé pour le langage de script TCL et intégrant un grand nombre de « widget » permettant la création rapide d'interfaces basiques

L'interface graphique de mon application est intégralement conçue en utilisant Tkinter et certaines classes utilisées afin de composer l'interface héritent de classes Tkinter

4.1.5 Problèmes rencontrés

4.1.5.1 Plantages de l'ordinateur à cause de la carte Wifi USB

Après une mise à jour de l'ordinateur lors du TPI, l'utilisation de la carte Wifi USB pour se connecter au routeur faisait planter l'ordinateur. Après plusieurs redémarrages et la venue de M. Jäggi du service informatique, il s'est avéré que la panne ne survenait que lorsque la carte Wifi était branchée sur un port USB3.

4.1.5.2 Connexion SSH ne chargeant pas les alias ou faite avec un PATH restreint

L'ouverture d'une session SSH à l'aide des subprocess Python diffère légèrement d'une ouverture « conventionnelle » dans un shell dans le sens où les alias ne semblent pas être chargés, ce qui complique le contrôle du robot. Afin de pouvoir les utiliser, il a fallu utiliser les options de la commande « ssh » permettant l'allocation d'un pseudo-terminal. Le problème s'étant présenté ensuite fut qu'une telle utilisation de la commande créait un pseudo-terminal dont le PATH (la liste des dossiers où le terminal peut aller chercher les commandes entrées par l'utilisateur) n'incluait pas le dossier contenant les commandes de gestion de la connexion I2C. Avec l'aide de M. Hurni, j'ai appris que la configuration d'un terminal Linux se faisait par l'exécution de fichiers « profile » et que différents fichiers de ce type étaient chargés en fonction du type de connexion. En fouillant dans la documentation de l'utilitaire OpenSSH, j'ai découvert l'existence d'un argument permettant de demander le bon type de connexion ce qui m'a permis d'utiliser les commandes de pilotage comme désiré

4.1.5.3 Lenteur de l'ouverture d'une session SSH

Le problème majeur de la fonction la plus simple d'utilisation du module subprocess de Python est que la session est ouverte, la commande exécutée, puis la session est close. Cela signifie que lorsqu'on voudra exécuter une nouvelle commande, la session devra être ré-ouverte ce qui implique une latence non-négligeable dans un contexte de pilotage. De plus, pendant tout le temps où la commande est exécutée, l'application est en attente et ne répondra aux événements utilisateur que lorsque la

commande sera terminée. Malheureusement, après quelques recherches sur internet, il s'avère que la création d'une connexion SSH persistante à l'aide des subprocess de python n'est pas possible et que même si elle l'était, elle impliquerait des instabilités logicielles potentielles qui pourraient aller jusqu'à faire planter l'application si des mesures complexes ne sont pas mises en œuvre.

4.1.5.4 Caméra nécessitant un temps de « chauffe »

Dans la documentation officielle de la caméra du Raspberry Pi, il est recommandé de laisser à la caméra au minimum 2 secondes de « preview » afin qu'elle puisse procéder à un certain nombre de réglages automatiques. Comme la caméra n'est pour le moment activée que lors d'une prise de vue et désactivée une fois cette dernière faite, ce temps de « chauffe » doit se faire à chaque prise d'un cliché. Ces 2 secondes s'ajoutent alors aux autres délais lors d'une prise de vue. L'utilisation de l'interface de la caméra en ligne de commande pourrait permettre de lancer la caméra au lancement de l'application mais il faudrait à ce moment-là désactiver la caméra à la fermeture de celle-ci ce que je n'ai pas encore trouvé comment faire.

4.1.5.5 Débit de connexion faible

Pour une raison encore inconnue, la vitesse de transfert entre l'ordinateur et le robot lors de l'utilisation du SCP est très lente (~50kB/s) ce qui entraîne un temps de transfert de l'ordre de la dizaine de seconde pour une image PNG de dimension 600x600px. C'est donc du temps qui s'ajoute aux autres délais lors d'une prise de vue

4.1.5.6 Gestion des images par Tkinter

Tkinter ne gère nativement que les formats d'image «.gif», «.pgm», «.ppm». Parmi ces trois, seul le GIF est un format dans lequel l'API de la caméra du Raspberry Pi est officiellement capable d'enregistrer une image. En pratique, bien que les GIFs puissent être lu par l'application elle-même, l'image n'est pas d'excellente qualité et ne peut être ouverte que par certains logiciels. Cela ne pose pas problème pour l'application en tant que telle, mais l'utilisateur doit en être conscient s'il souhaite consulter les images hors de l'application

4.2 Description des tests effectués

4.2.1 Tests de fonctionnalités

4.2.2 Tests de performance

Pour chaque partie testée de votre projet, il faut décrire:

- *les conditions exactes de chaque test*
- *les preuves de test (papier ou fichier)*
- *tests sans preuve: fournir au moins une description*
- *Il est recommandé de partir des Scénarios décrits dans l'anayse, complétés éventuellement par les modifications apportées à l'analyse.*

4.3 Erreurs restantes

S'il reste encore des erreurs:

- *Description détaillée*
- *Conséquences sur l'utilisation du produit*
- *Actions envisagées ou possibles*

4.4 Dossier d'archivage

- PilotApp
  - doc :
    Dossier contenant tous les fichiers de conception du projet (documentation, mockup…)
  - src
    Dossier racine de l'application elle-même (voir 4.1.1).

5 Mise en service

5.1 Installation

5.1.1 Pré-requis

- Commande « scp » dans les PATH
- Commande « ssh » dans les PATH
- Interpréteur python 3

5.1.2 Installation du logiciel

- Télécharger les fichiers
- Placer la clé privée dans le dossier adéquat (~/.ssh/)
- Pour lancer le programme, exécuter le fichier « PilotApp.py » depuis le dossier où il est installé

5.2 Liste des documents fournis

- Rapport du projet : DocumentationAnthonyJaccard.pdf
- Résumé du rapport : ResumeAnthonyJaccard.pdf
- Planification : PlanificationAnthonyJaccard.pdf
- Fichiers d'installation (fichiers source, manuel d'utilisation texte et clé privée) : Installation.zip

6 Conclusions

Développez en tous cas les points suivants:

- *Objectifs atteints / non-atteints*
- *Comparaison entre ce qui avait prévu et ce qui s'est passé, en termes de planning et (éventuellement) de budget*
- *Points positifs / négatifs*
- *Difficultés particulières*
- *Suites possibles pour le projet (évolutions & améliorations)*

7 Annexes

7.1 Sources – Bibliographie

- https://www.ssh.com/
- https://picamera.readthedocs.io/en/release-1.13/
- http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html
- https://www.sololearn.com
- https://acrisel.github.io/posts/2017/08/ssh-made-easy-using-python/
- https://www.openssh.com/
- https://stackoverflow.com/

7.2 Manuel d'Utilisation

7.3 Archives du projet

CD, DVD… dans une fourre en plastique.

7.4 Glossaire