LibAsserv 2014 Documentation

Version 0.1

Matthieu Pizenberg

22 February 2015

Table des matières

CHAPITRE 1

Introduction

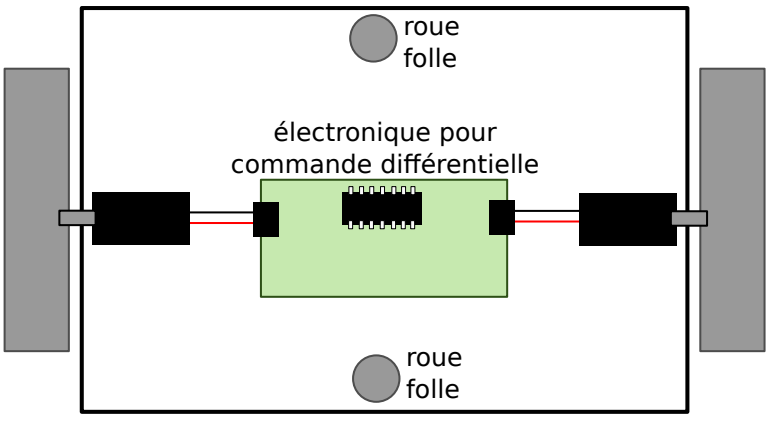
Bienvenu sur documentation de la bibliothèque utilisée pour l'asservissement du robot Fatman pour la coupe de France de robotique 2014.

Vous trouverez le code source sur le dépot Github suivant : https://github.com/7Robot/lib_asserv_2014

Vous trouverez la version pdf de cette documentation ici : https://github.com/7Robot/lib_asserv_2014/raw/documentation/docs/LibAsserv2014.pdf (elle n'est pas nécessairement à jour par rapport à la version web).

1.1 Plateforme à asservir

Cette bibliothèque répond à un besoin précis : **pouvoir contrôler le déplacement d'un robot, en vitesse ou en position**. Ce robot doit être assimilable à une plateforme différentielle, considérée comme holonome.



Plateforme différentielle Une plateforme différentielle est constituée de 2 roues commandées indépendamment ce qui permet souvent de la considérer holonome.

Plateforme holonome Pour faire simple, on peut dire qu'une plateforme est dite holonome si elle peut se déplacer dans toutes les directions depuis sa position courante.

1.2 Objectifs de l'asservissement

Le robot est utilisé pour participer à la coupe de France de robotique 2014. Il doit pouvoir se déplacer de manière autonome sur un plateaux au dimensions suivantes : 2m x 3m. Il n'est pas seul sur le plateaux, il peut y avoir jusque 4 robots. Il y a aussi des éléments de jeu fixe et mobiles sur le plateau.

Compte tenu de ces conditions, il est inutile de chercher la vitesse. Par contre il peut être très intéressant d'avoir des bonnes capacités d'accélération et de frein pour naviguer le plus efficacement possible.

La limite de l'**intelligence** de l'asservissement est par contre assez difficile à définir. Il peut être intéressant de le rendre autonome (évitement des obstacles intégré par exemple) mais plus complexe. Mais a contrario il peut être aussi plus souple de garder le contrôle global de la trajectoire par un composant plus au niveau du robot (IA sur Raspberry Pi par exemple) et de ne contrôler avec cette bibliothèque d'asservissement que les déplacement de **bas niveau**.

J'ai opté pour ce 2^{e} choix aussi parce que ça simplifie la bibliothèque d'asservissement.

1.3 Implémentation de la bibliothèque d'asservissement

L'objectif étant de faire une bibliothèque réutilisable et indépendante de la configuration du robot, on a choisi de la développer en C pur. Ceci nous permet de l'utiliser avec n'importe qu'elle plateforme de programmation supportant la compilation de langage C.

A 7Robot, nous utilisons une plateforme basée sur des microcontroleurs PIC (Microchip) pour contrôler les moteurs du robot. Nous utilisons le logiciel MPLAB X avec des PICkit pour compiler notre code et le charger sur des dsPIC33f.

CHAPITRE 2

Utilisation de la bibliothèque

Je vais expliquer ici comment utiliser la bibliothèque d'asservissement dans le cadre de l'asservissement d'un robot.

CHAPITRE 3

Déplacement du robot

CHAPITRE 4

Asservissement du robot

CHAPITRE 5

Odométrie

CHAPITRE 6

Régulateur PID

CHAPITRE 7

Construire cette documentation

L'objectif est d'avoir une documentation facilement maintenable tout en étant suffisamment riche en fonctionnalités pour pouvoir décrire l'ensemble du code. Il est donc important d'avoir le support de fonctionnalités telles que :

— Une syntaxe simple
— Ecrire du code source
— Inclure des images
— Utiliser des mathématiques
— Une navigation simple dans la documentation

J'avais le choix entre 2 types de documentations :

1. Une documentation générée à partir de fichiers Markdown.
2. Ou une documentation générée à partir de fichiers rST (reStructuredText).

J'ai opté pour la 2e car elle gère mieux l'affichage des formules mathématiques (à la mode LaTeX) et qu'elle permet d'exporter la documentation vers un pdf.

Sphinx est un outil pour générer facilement de la documentation à base de fichiers respectant la syntaxe rST (reStructuredText). Vous pourrez trouver une introduction à la syntaxe rST à cette adresse : http://sphinx-doc.org/rest.html. Elle est à la fois simple et puissante !

7.1 Installation de Sphinx

Vous pouvez trouver tous les détails de l'installation pour votre configuration ici : http://sphinx-doc.org/install.html. Pour moi ça a été extrêmement simple, j'ai utilisé le gestionnaire de paquets de python (pip) :

```
$ sudo pip install sphinx
```

Et c'est tout !

7.2 Initialiser la documentation

Les sources de la documentation vont se trouver dans un dossier `docs`. On commence donc par créer ce dossier. Ensuite il suffit d'utiliser la commande `sphinx-quickstart` qui permet

de configurer la documentation :

```
$ mkdir docs
$ cd docs/
$ sphinx-quickstart
```

Généralement vous pouvez laisser la configuration par défaut qui vous est proposée. Attention, il y a quand même certaines questions pour lesquelles il faut changer la réponse :

```
> Separate source and build directories (y/n) [n]: y
> autodoc: automatically insert docstrings from modules (y/n) [n]: y
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]: y
```

A la fin du scrip Sphinx aura initialisé la configuration en générant 2 dossiers : `build` et `source` ainsi qu'un makefile pour faciliter la génération de la doc avec une commande du type :

```
$ make html
```

7.3 Configuration de la documentation Sphinx

L'utilisation du script `sphinx-quickstart` a généré plusieurs fichiers permettant d'agir sur la configuration de la documentation dans le dossier `docs/`. Je vais détailler maintenant les quelques manip à faire pour améliorer cette configuration.

7.3.1 Héberger sa documentation sur Github

Si vous travaillez sur du code avec un dépot Git (j'espère !), il y a quelques manip à faire.

Une documentation qui n'est visible par personne n'est pas très utile. Du coup il faut héberger quelquepart la documentation générée. Si vous hébergez votre code sur Github c'est parfait ! Github peut aussi héberger la documentation dans ce que l'on appelle les "pages Github". Vous pouvez avoir une explication ici : https://help.github.com/articles/what-are-github-pages/

Pour activer les pages Github de votre dépot il suffit de créer un nouvelle branche vierge intitulée `gh-pages` et de la push :

```
$ git checkout --orphan gh-pages
$ git rm -rf .
$ echo "My page" > index.html
$ git add index.html
$ git commit -am "Premier commit sur les pages Github"
$ git push origin gh-pages -u
```

7.3.2 Configurer la compilation vers HTML

On veut pouvoir build la documentation dans la branche `gh-pages`. Or ca ne va pas être pratique de devoir changer constamment de branche pour la documentation. Ce que je suggère donc c'est d'avoir un deuxième dossier qui est un clone du même dépot mais dans lequel on reste toujours sur la branche `gh-pages` :

```
$ cd ../
$ mkdir mondepot_gh-pages
$ cd mondepot_gh-pages/
$ git clone -b gh-pages --single-branch git@github.com:moi/mondepot.git .
```

Maintenant il faut changer la configuration pour build dans ce dossier. On repart dans le dossier original et on modifie le fichier `Makefile` :

```
BUILDDIR = ../../mondepot_gh-pages

html:
    $(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)
```

7.3.3 Utiliser le thème rtd (readTheDocs)

C'est tout simple. Il suffit d'installer le theme via pip :

```
$ sudo pip install sphinx_rtd_theme
```

Ensuite dans le fichier `conf.py` :

```
import sphinx_rtd_theme
html_theme = "sphinx_rtd_theme"
html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]
```

Pour que le theme marche une fois la documentation poussée sur Github, il faut faire une dernière petite manip : il faut ajouter un fichier vide intitulé `.nojekyll` (dans la branche gh-pages) et le push (add et commit avant biensûr). Sinon, Github va ignorer les dossiers commençant par _ dont le dossier _static/ qui contient les styles CSS et images et donc le theme ne marchera pas (vous aurez une vielle page html toute moche).

7.3.4 Configurer la compilation vers PDF (Latex)

La compilation de la doc vers pdf (Latex) est un outils puissant mais qui demande un petit peu de réglages. J'ai choisi de compiler en français (babel french), dans un dossier temporaire et de ne garder que les pdf à la fin (supprimer directement les fichiers Latex générés automatiquement). Pour le type de document Latex on a le choix entre 2 types : `manual` qui correspond à une sorte de `report`, ou `howto` plus proche du style de `article`. J'ai choisi `howto` pour avoir une documentation plus compacte.

Modifs à faire dans le fichier `Makefile` :

```
# ajouter au début : le dossier temporaire pour la compilation Latex
PDFBUILDDIR = build

# nettoyage des fichiers générés
clean:
   rm -rf $(BUILDDIR)/*
   rm -rf $(PDFBUILDDIR)

# builds pour latex et latexpdf (compile le pdf en plus de générer le Latex)
latex:
   $(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(PDFBUILDDIR)/latex
   @echo "Build finished; the LaTeX files are in $(PDFBUILDDIR)/latex."
   @echo "Run \`make' in that directory to run these through (pdf)latex" \
         "(use \`make latexpdf' here to do that automatically)."

latexpdf:
   $(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(PDFBUILDDIR)/latex
   @echo "Running LaTeX files through pdflatex..."
   $(MAKE) -C $(PDFBUILDDIR)/latex all-pdf
   cp $(PDFBUILDDIR)/latex/*.pdf .
   rm -rf $(PDFBUILDDIR)
   @echo "pdflatex finished"
```

Et pour le fichier `conf.py` :

```
language = 'fr'

'papersize': 'a4paper',
'pointsize': '12pt',

# The language
'babel': '\\usepackage[french]{babel}'

# choisir 'howto' au lieu de 'manual' si vous le souhaitez
latex_documents = [
   ('index', 'LibAsserv2014.tex', 'LibAsserv 2014 Documentation',
   'Matthieu Pizenberg', 'howto'),
]
```