Robothon - Polytech Sorbonne

Release 0.1

Author

Jun 06, 2024

Table des matières

Bienvenue sur la documentation du projet industriel “robothon”.

Projet dans le cadre du module projet industriel de la spécialité robotique.

Etudiants : DAI Yannick, DAT Mathieu, RAZANAJATOVO Faliana, TEIXEIRA Pierre

Clients : KHORAMSHAHI Mahdi, DONCIEUX Stéphane

Tuteurs académiques : BAUDRY Aline, PLUMET Frédéric

1

Vue d'ensemble du projet

2

Contexte du projet

3 Approche pour mener à bien le projet

Ressources

- https://github.com/peterso/robothon-grand-challenge?tab=readme-ov-file
- https://wiki.ros.org/

ce projet industriel a été pour nous une opportunité d'aprendre à utiliser une plateforme telle que Docker pour faciliter l'organisation du travail dans une équipe, mais aussi pour nous familiariser avec l'OS Linux, et être habitué à l'utilisation du terminale pour l'exécution et la compilation de programmes, mais aussi la découverte d'un nouvel outil, très intéressant et assez important dans la robotique car la majorité des systèmes robotisés l'utilise (on parle ici de ROS).

Comme il s'agit d'une découverte, on a passé une bonne partie de notre temps à se documenter et à apprendre. Pour mener à bien le projet, on s'est basé sur les différentes solutions développées par les équipes des éditions précédentes de la compétition Robothon, pour s'y inspirer, en prenant les bonnes solutions, en apprenant de leurs erreurs et en optimisant les raisonnements.

A travers 3 éditions, on a pu remarquer que l'équipe de "*Platonics*", dont les codes et modules étaient les mieux documentés et rendus open-source à 100%, avait développé des solution très intéressantes, qu'on a pas mal utilisé comme base pour les différentes tâches à accomplir comme la localisation de la task-board avec SIFT.

Dans la dynamique du travail collaboratif, on a créé un repository sur gitlabsu (https://gitlabsu.sorbonne-universite.fr/robothon-sorbonne) avec différents packages en fonction des tâches et des différents installations qu'il fallait qu'on fasse pour s'harmoniser au niveau du groupe.

Installation de l'environnement Docker avec ROS et tous les packages nécessaires

Logiciels installés (incluant environnement, et OS à avoir) :

- Linux (Ubuntu, peu importe la version car on va utiliser Docker),
- Docker (pour pouvoir contenir ROS Noetic qui n'est utilisable que sous Ubuntu version 20.04),
- ROS1 Noetic ("*Robotic Operating System*" installé dans Docker),
- Terminator (gestionnaire de terminal pour faciliter la compilation de plusieurs programmes en ayant un oeil sur les autres en train de tourner)

Qu'est-ce que "Docker" ?

Docker est une plateforme open source qui automatise le déploiement d'applications dans des conteneurs légers. Un conteneur regroupe l'application et toutes ses dépendances dans un seul paquet, garantissant que l'application fonctionne de manière cohérente, quel que soit l'environnement, donc indépendante à l'OS dans lequel il est. Docker permet de créer, déployer et gérer ces conteneurs de manière efficace, offrant ainsi une solution portable et flexible pour le développement et le déploiement d'applications. Il simplifie également la gestion des versions et des configurations, facilitant la collaboration entre les équipes de développement et d'exploitation.

Dans notre cas, on l'utilise pour la gestion de version, donc pouvoir utiliser ROS Noetic même avec un OS Ubuntu version 22.04, qui de base n'est pas compatible avec cette version de ROS, mais on utilise aussi Docker pour harmoniser les environnements sur chacun de nos appareils (les ordinateurs auront donc la même configuration et les mêmes pachages une fois que l'image Docker est compilée).

Commandes pour accéder aux différents packages qui ont été développés :

Installation de Docker incluant déjà tous les packages liés au projet :

```
$ git clone https://gitlabsu.sorbonne-universite.fr/robothon-sorbonne/docker.git --
↪recursive
```

Se placer dans le répertoire docker :

```
$ cd docker
```

Installation de l'image Docker :

```
$ ./install_docker.bash
```

Une fois l'installation de l'image, il suffit de lancer Docker :

```
$ ./start_docker.sh
```

Pour avoir accès à terminator, il suffit de saisir :

```
$ terminator
```

4 Environnement logiciel

Ceci présente l'environnement logiciel : ROS, Docker, git, gitlab, etc.

Est-ce que la compilation de schéma avec *Mermaid* fonctionne en *LaTeX* ?

FIGURE4.1 – Ceci est une figure centrée avec une légende

FIGURE4.2 – Graphe pour l'exécution d'une tâche

FIGURE4.3 – Superviseur

!(./img/new/Graphe.png)

5

Localisation de la task-board à l'aide de SIFT

```
class LocalizationService(){
    func __init__(self)
    func compute_localization_in_pixels(self, img: Image)
    func publish_annoted_image(self)
    func handle_request(self, req)
    func run(self)
}
```

6 Tâche pour appuyer sur le bouton bleu

```
class Panda_R(Panda):
    def __init__(self):
        super().__init__()
        self.listener = tf.TransformListener()

    # Méthode permettant de placer l'effecteur final à une hauteur de 0.25m
    # par défaut (mais ci-dessous, saisir dans le terminal "home" le place à 0.5m)

    def home(self, z=0.25):
        pos_array = np.array([0.4, 0, z])
        quat = np.quaternion(0, 1, 0, 0)
        goal = array_quat_2_pose(pos_array, quat)
        goal.header.seq = 1
        goal.header.stamp = rospy.Time.now()

        ns_msg = [0, 0, 0, -2.4, 0, 2.4, 0]
        self.go_to_pose(goal)
        self.set_configuration(ns_msg)
        self.set_K.update_configuration({"nullspace_stiffness":10})

        rospy.sleep(rospy.Duration(secs=1))

        self.set_K.update_configuration({"nullspace_stiffness":0})
```

```
$ Position : home
```

```
$ Position : home2
```

```
def run(self):
        while not rospy.is_shutdown():
            position = input("Position : ")
            if position in self.keys.keys():
                char=self.keys[position]
                if char == 'home2':
                    self.panda.home(0.25)
```

(continues on next page)

(continued from previous page)

```
                elif char == 'home':
                    self.panda.home(0.5)
                    self.localization()
                elif char == "quit":  # STOP
                    rospy.signal_shutdown("User initiated shutdown")
                elif char=='test':

                    pose = PoseStamped ()
                    pose.pose.position.x = 0
                    pose.pose.position.y = 0.2
                    pose.pose.position.x = 0

                    rospy.loginfo(pose)
                    self.panda.go_to_pose_EE(pose)
                elif char=='circle':
                    # theta goes from 0 to 2pi
                    theta = np.linspace(0, 2*np.pi, 2000)

                    # the radius of the circle
                    r = 0.1

                    x = r*np.cos(theta) + self.panda.curr_pos[0]
                    y = r*np.sin(theta) + self.panda.curr_pos[1]

                    z = np.full((2000), self.panda.curr_pos[2])
                    self.panda.set_stiffness(4000, 4000, 4000, 50, 50, 50, 10)

                    self.panda.play_trajectory(x, y, z)
                elif char=='door':
                    self.door()
                else:
                    self.panda.go_to_frame(char[0], 'tool', char[1])

            self.rate.sleep()
```

7

Tâche pour le déplacement du slider en fonction d'un retour visuel

8

Ouverture de la trappe avant le prélèvement de la sonde

9

Tâche de la sonde pour un prélèvement au niveau d'un capteur