Panda robot bekapcsolása egységes robotprogramozási rendszerbe

Connectiong the Panda robot to a unified robot programing system

BOCSI Kristóf[1] B.Sc., fejlesztőmérnök, NACSA János[1] Ph.D., tud. főmunkatárs, CSEMPÉSZ János[1] M.Sc., fejlesztőmérnök

[1] Termelésinformatikai és Termelésirányítási Kiválósági Központ (EPIC),
Számítástechnikai és Automatizálási Kutatóintézet (SZTAKI), Eötvös Loránd Kutatási Hálózat (ELKH),
1111 Budapest, Kende u 13-17, +36 1 279 6000, bocsi.kristof@sztaki.hu, www.sztaki.hu

Abstract

The 7 DOF collaborative Panda robot [5] is usually programmed using ROS (Robot Operating System) [1]. This paper presents the panda_ursztaki ROS package, that is used to connect this robot to the unified robot programing system called URSZTAKI, which was originally created for UR robots [6]. An assembly task is presented as an example.

Keywords: robot, collaborative robot, unified control, panda robot

Kivonat

A hét szabadsági fokú, kollaboratív Panda robot [5] programozása leggyakrabban ROS (Robot Operating System) [1] segítségével történik. Jelen dolgozat célja, hogy bemutassa a panda_ursztaki nevű ROS csomagot, aminek segítségével ez a robot is bekapcsolható a korábban UR robotokhoz [6] létrehozott, URSZTAKI nevű, egységes vezérlési rendszerbe. A működés egy szerelési feladat részletezésével kerül bemutatásra, melyet mindkét robotfajtán el lehet végezni.

Kulcsszavak: robot, kollaboratív robot, egységes vezérlés, panda robot

1. AZ URSZTAKI KERETRENDSZER

Az URSZTAKI keretrendszer célja, egy olyan moduláris vezérlés létrehozása az UR robotokhoz, aminek segítségével a különböző felszereltségű robotok egységesen vezérelhetők, a robotok script nyelvének ismerete nélkül.

A program script fájlokra épül, amik a vezérlőn futnak. Alapműködése, hogy a hálózaton keresztül szám tömböket kap, amik alapján utasításokat hajt végre, állapotáról szintén szám tömbökkel küld visszajelzést.

Utasítások kiadása taskok segítségével történik. Ezek előre definiált mozgásokat reprezentálnak, a paramétereik meghíváskor adhatók meg. Az egyes taskok összetettsége eltérő. Az egyszerűbb taskok közé tartozik például a lineáris mozgás és a megfogó mozgatása. A bonyolultabb taskokra példa a munkadarab felvétele és lerakása.

A programmal két TCP porton keresztül lehet kommunikálni. Az egyik port utasítások kiadására a másik a program által küldött státuszüzenetek fogadására használható.

A rendszer része továbbá egy Node-Red [8] flow. Ennek célja egy magas szintű kommunikációs interfész biztosítása tetszőleges külső alkalmazás felé, valamint ezen keresztül érhető el az UR robotok RTDE (Real Time Data Exchange) [7] funkciója. Ennek lényege, hogy a vezérlés 125Hz-es frekvencián kibocsájtja a robot aktuális munka- és csuklótérbeli pozícióját. Ennek olvasására egy Python scriptet használunk, ami az olvasott értékeket egy JSON-ben küldi tovább a Node-Red-nek.

2. A PANDA ROBOT PROGRAMOZÁSA

2.1. A Panda robot

A Franka Emika Panda robotja (1. ábra) egy hét csuklós kollaboratív robot.

1. ábra. *A Panda robot*

A robot programozása több lehetőség is rendelkezésre áll. Ezek közül a franka_ros [3] nevű ROS csomagot és MoveIt [4] nevű, szintén ROS alapú pályatervezőt használtuk. A fejezet további része ezeknek a működését mutatja be.

2.2. A franka_ros

A franka_ros a robot programozására használható hivatalos ROS csomag. Ennek segítségével lehetséges többek között ROS alapú kontroller pluginok [2] létrehozása és a robot megfogójának mozgatása. A ROS kontroller pluginok C++ osztályok, amiknek tartalmazniuk kell a következő függvényeket: init, starting update, stopping. Egy kontroller mindig három állapot valamelyikében van: [2]

- Nincs betöltve: Az adott kontrollert az éppen futó alkalmazás nem használja
- Megállított: A éppen futó alkalmazás ezt a kontrollert is használja, de éppen nem ez aktív.
- Fut: A futó alkalmazás ezt a kontrollert használja (ez nem feltétlenül jelenti azt, hogy a robot éppen mozog)

Ha egy kontroller éppen fut az update függvénye 1kHz-es frekvencián minden időlépésben meghívódik.

2.3. A MoveIt pályatervező

A MoveIt nevű ROS csomag pályák tervezésére és ezek végrehajtására használható API-t biztosít a felhasználónak. A pályatervező belső működésének leírása [4]-ben található.

3. A PANDA_URSZTAKI

A panda_ursztaki nevű ROS csomag célja a Panda robot bekapcsolása a fent bemutatott URSZTAKI keretrendszerbe. A projekt része még a bkb_force_control nevű csomag, ami egy erővisszacsatolást használó kontrollert tartalmaz.

3.1. A fő program megvalósítása

A panda_ursztaki csomag tartalmazza a fő node-ot (ROS alapú programot). Ez a program veszi át az UR-ek esetén a vezérlőn futó script helyét. Ez a program felelős a felhasználóval történő kommunikációért, valamint a kiadott utasítások végrehajtásáért. A megvalósított taskok négy csoportba oszthatók:

- Egyszerű robot mozgások: Ezek a taskok valamilyen egyszerű mozgást hajtanak végre. Megvalósításuk a MoveIt pályatervező függvényeivel történik. Példa: moveL, moveJ.
- Erő visszacsatolást használó taskok: Ezek a taskok a mozgás során figyelembe veszik a robotra ható külső erőt. A MoveIt erre nem alkalmas, ezért ezekhez a mozgásokhoz saját kontrollert hoztunk létre. Ennek a működését a „touch" nevű task segítségével mutatjuk be. Ennek működése a következő: a robot adott v_x v_y v_z sebességvektorral mozog, amíg a külső erő el nem éri a megadott értéket, ezután az adott távolsággal visszafelé mozog (d_z). A megvalósításhoz létrehozott „touch" kontroller a MoveIt alapértelmezett kontrollerével (joint trajectory controller) együtt töltődik be az alkalmazás indításakor. A task indításakor elindul a kontroller, a többi futó kontroller pedig megáll. A kontroller indítása után a robot munkatérbeli sebessége egy állandó gyorsulással növekszik, amíg el nem éri a megadott sebességvektort, majd ezzel a sebességgel halad, amíg a külső erő eléri az előre megadott értéket (ezt az egyes csuklókban mért nyomaték alapján számolja a robot vezérlése). A megadott erő elérése után a robot megáll, az aktuális pozíció és a megadott d_z távolság alapján megtörténik annak a pontnak a kiszámítása, ahova a robotnak vissza kell mozognia. Ezután a robot az előzővel ellenkező irányba gyorsul a megadott sebesség eléréséig, majd ezzel a sebességgel halad, amíg meg nem közelíti az elérendő pontot

$$d \leq \frac{v^2}{2 \cdot a} \tag{1}$$

  távolságra. Ezen a távolságon belül az álladó *a* gyorsulással csökken a sebessége.
- Megfogó mozgatás: Ezek a taskok a megfogó mozgatására használhatók. Ezek megvalósítása a franka_ros megfelelő funkciójával történik.
- Összetett taskok: Ezek a taskok több egyszerűbb task egymás utáni végrehajtásával vannak megvalósítva. Példa: munkadarab lerakása. Ennél a tasknál a robot *z* irányban lefelé mozog amig a külső *z* irányú erő el nem éri a megadott értéket. Ezután a megfogó kinyílik, majd a robot a megadott távolsággal felfelé mozog. Ez a task a „touch", „megfogó mozgatás" és „relatív mozgás" taskok egymás utáni lefuttatásával van megvalósítva.

3.2. Az RTDE funkció megvalósítása

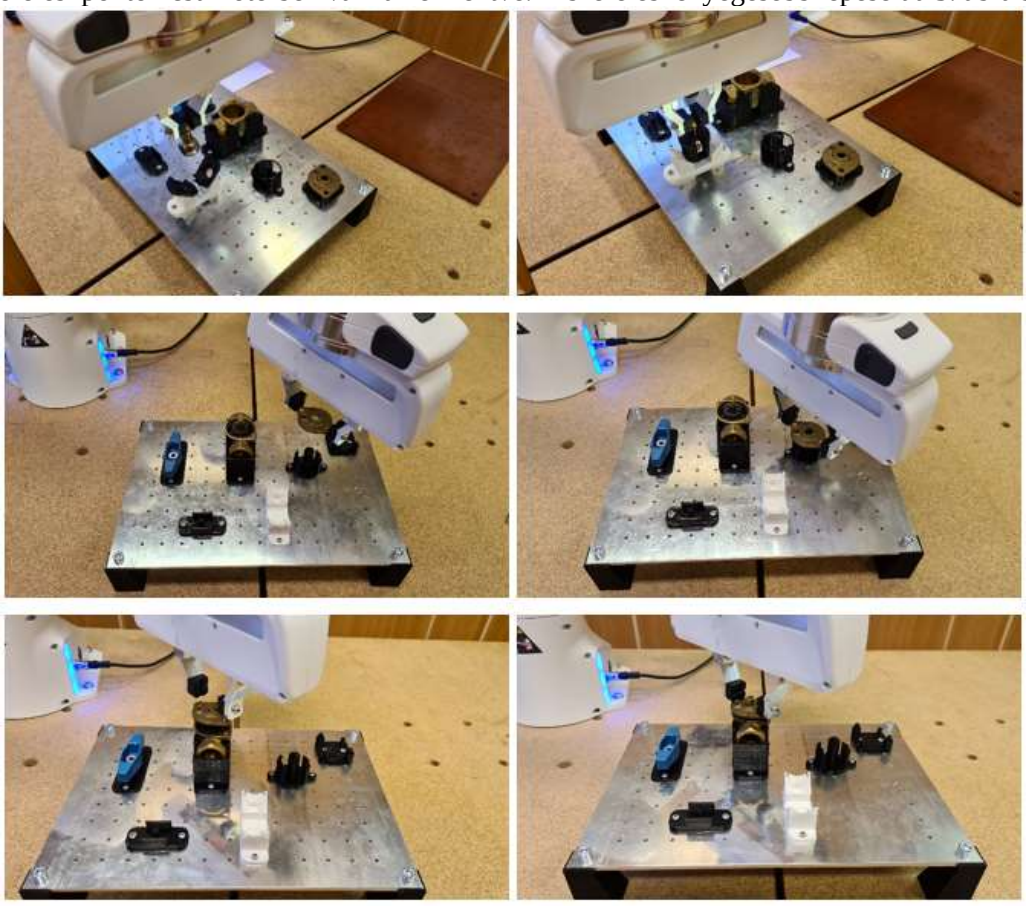
Az eredeti alkalmazás RTDE funkcióját jelen esetben a Python script helyett egy ROS node veszi át. Ez a node a „joint_states" topic-ot olvassa. Ez alapján végzi el a direkt kinematikai számításokat és az így megkapott pozíciót küldi tovább a Node-Red felé (az eredeti alkalmazás Node-red flow-ját megtartottuk).

4. GÖMBCSAP SZERELÉSI FELADAT MEGVALÓSÍTÁSA

Az elkészült program tesztelése egy $\frac{1}{2}$ colos gömbcsap automatikus összeszerelésével történt. A gömbcsap összeszerelése korábban UR5 robottal is megtörtént szintén az URSZTAKI program segítségével. A szerelési elrendezés a 2. ábrán látható.

2. árba A gömbcsap szerelési feladat elrendezése

A szerelési pontok csuklótérben vannak elmentve. A szerelés lényegesebb lépéseit a 3. ábra szemlélteti.



3. ábra: A szerelés lényegesebb lépései (felül: a gömb behelyezése a tömítőfelek közé, középen: a kisseb o-gyűrű felvétele a fedéllel, alul: a fedél lenyomása

Felül látható a gömb behelyezése a tömítőfelek közé, amik ennek hatására rácsukódnak és az összeállítás így egyben felvehető. Középen látható a kisebb o-gyűrű felvétele a fedéllel. Az első ábrákon látható az így létrejött részösszeállítás rányomása a házra. Ezt a műveletet a korábban bemutatott „touch” task segítségével hajtjuk végre.

KÖSZÖNETNYILVÁNÍTÁS

A publikációban szereplő projektet a Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal INEXT - Kutatások az ipari digitalizáció által nyújtott potenciál minőségi kiaknázására (ED_18-22018-0006) projektje támogatta. Nacsa János külön köszöni a „Felsőoktatási Intézményi Kiválósági Program – Digitális ipari technológiák kutatása a Széchenyi István Egyetemen" projekt (TUDFO/47138-1/2019-ITM) támogatását.

IRODALMI HIVATKOZÁSOK

[1] ROS https://www.ros.org/about-ros/ (2021. 02. 03.)
[2] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke and E. Fernandez Perdomo "ros_control: A generic and simple control framework for ROS" The Journal of Open Source Software, 2017.
[3] franka_ros https://frankaemika.github.io/docs/franka_ros.html (2021. 02. 03.)
[4] David Coleman, Ioan A. Șucan, Sachin Chitta, Nikolaus Correll, Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study, *Journal of Software Engineering for Robotics*, 5(1):3–16, May 2014. doi: 10.6092/JOSER_2014_05_01_p3.
[5] Panda robot https://www.franka.de/technology (2021. 02. 03.)
[6] UR robot https://www.universal-robots.com/hu/ (2021. 02. 03.)
[7] RTDE https://www.universal-robots.com/articles/ur/real-time-data-exchange-rtde-guide/ (2021. 02. 03.)
[8] Node-Red https://nodered.org/ (2021. 02. 03.)
[9] MATLAB Faranka Emika Robots https://ch.mathworks.com/products/connections/product_detail/franka-emika-robots.html (2021. 02. 18.)