Catedra: Calculatoare

**Brat Robotic cu Moveit si ROS-Noetic**

**Nume: Anastasiu Andreea-Valentina**

**Grupa: 30237**

**Nume îndrumător proiect: Mircea Paul Mureșan**

**Data: 8 ianuarie 2025**

**Cuprins**

**1.Rezumat**

**2. Introducere**

**3.Configurare spațiu de lucru**

**3.1. Setup SO**

**3.2. Setup Mediu de Dezvoltare**

**4.Control Brat**

**4.1. Definire unei miscari catre o pozitie specifica**

**4.2. Miscare in Spatiul Articulatiilor**

**4.3. Planificare cu Constrangeri de Orientare**

**4.4. Planificare Carteziana**

**4.5. Planificare cu Obstacole**

**5. Miscare Brat Inverse Kinematics**

**5.1. Ce este Inverse Kinematics?**

**5.2. Explicatie cod inverse kinematic**

**6. Probleme Camera Virtuala**

**7. Bibliografie**

**1.Rezumat**

**În acest proiect urmaresc dezvoltarea unui braț robotic controlat cu Moveit si ROS Noetic. Pe parcursul acestui proiect s-au urmarit configurarea sptiului de lucru, programarea mișcarilor de orientare si planificarea traiectoriilor în prezența unor obstacole.**

**Am urmărit sa acopar diferite functionalitati precum miscarea carteziana, planificarea traiectoriilor si interactiunea cu diferite obiecte virtuale.**

**În aceasta documentatie voi sublinia pasii de dezvoltare a proiectului, problemele intampinate si solutiile aplicate.**

**2.Introducere**

Acest proiect explorează dezvoltarea unui braț robotic controlat utilizând MoveIt și ROS Noetic. Procesul implică configurarea unui mediu de dezvoltare robust, dezvoltarea mișcărilor complexe prin cinematică directă și inversă și integrarea constrângerilor specifice pentru planificarea traiectoriilor.

Proiectul urmărește să pună în aplicare cunoștințele din domeniul roboticii și programării hardware pentru a crea un sistem funcțional. Activitățile includ configurarea inițială a platformei software, dezvoltarea și testarea controlului programatic al brațului.

Realizarea acestui proiect oferă o perspectivă practică asupra utilizării ROS Noetic și MoveIt pentru dezvoltarea roboticii, demonstrând aplicabilitatea acestora în rezolvarea problemelor complexe de control și planificare a mișcărilor.

**3.Configurare spațiu de lucru**

**3.1. Setup SO**

Ubuntu este un sistem de operare open-source foarte popular pentru dezvoltarea software, mai ales în robotică, deoarece ROS este conceput să funcționeze pe el. Versiunea **20.04 LTS** (Long Term Support) este stabilă și recomandată pentru ROS Noetic.

Pasi;

Pasul 1: Crearea unui stick USB bootabil (descarcam Rufus pentru a face stick-ul bootabil, selectam imaginea ISO Ubuntu pe care o descarcam de pe [Ubuntu 20.04](https://releases.ubuntu.com/20.04/))

Pasul 2: Instalarea sistemului de operare

1. Repornim laptopul si accesam meniul de boot
2. Selectam stick-ul USB ca sursa de boot
3. Selectează **Install Ubuntu**.
4. Alegem limba și configurația tastaturii.
5. Configuram partițiile dacă vrei să păstrezi alte sisteme de operare sau alege **Erase disk and install Ubuntu** pentru o instalare curată.
6. Creează un cont de utilizator și setează o parolă.
7. După instalare, repornește sistemul și conectează-te.

Pasul 3: Actualizarea sistemului si folosim urmatoarele comenzi:

sudo apt update # Actualizează lista de pachete disponibile.

sudo apt upgrade -y # Instalează pachetele actualizate.

**3.2. Setup Mediu de Dezvoltare**

ROS (Robot Operating System) este o colecție de biblioteci și instrumente pentru dezvoltarea aplicațiilor robotice. **Noetic** este versiunea recomandată pentru Ubuntu 20.04.

Pasi:

Pasul 1: Configurarea sursei de pachete ROS

Acest Repo contine pachete necesare pentru instalare. Practic noi adaugam un link spre server-ul Ros in lista de surse Ubuntu.

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

Pasul 2: Instalarea cheii GPG

O cheie GPG este o component folosita pentru a securiza comunicatiile si datele. Cu aceasta vom verifica autenticitatea si integritatea fisierelor descarcate dintr-un repo.

sudo apt install curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add –

Pasul 3: Instalarea ROS Noetic desktop-full

Acesta include toate instrumentele necesare, inclusive Rviz cu vizualizare si Gazebo pentru simulare.

sudo apt update

sudo apt install ros-noetic-desktop-full

Pasul 4: Configurarea mediului ROS

Noi va trebui sa activam in fiecare terminal ROS. Dupa ce adaugam aceasta comanda nu va trebui scris in fiecare terminal pentru a active ROS.

echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

Pasul 5: Instalarea și configurarea rosdep

Rosdep gestioneaza toate dependentele ros pentru pachete

sudo apt install python3-rosdep

sudo rosdep init

rosdep update

MoveIt este o platformă pentru controlul mișcării roboților, care funcționează împreună cu ROS.

Pasul 1: Crearea unui workspace pentru MoveIt

Workspace-ul este un loc separat unde se vor afla toate fișierele și pachetele ROS.

mkdir -p ~/ws\_moveit/src

cd ~/ws\_moveit/src

git clone <https://github.com/ros-planning/moveit.git>

Pasul 2: Instalarea dependențelor

Dependentele sunt biblioteci externe necesare pentru ca Moveit sa functioneze correct

cd ~/ws\_moveit

rosdep install -y --from-paths src --ignore-src --rosdistro noetic

Pasul 3: Construirea codului MoveIt

Construirea (compilarea) transformă codul sursă în fișiere executabile.

catkin\_make

source devel/setup.bash

Testarea:

**Testarea ROS**

Rulează roscore pentru a verifica dacă ROS funcționează

Iar intr-un alt terminal vom lansa un nod simplu: rosrun turtlesim turtlesim\_node

**Testarea MoveIt**

Rulează demo-ul Panda inclus în MoveIt:

roslaunch panda\_moveit\_config demo.launch

În RViz, verifică dacă brațul robotic Panda este afișat și răspunde la comenzi.

**4.Control Brat**

**4.1. Definire unei miscari catre o pozitie specifica**

Mișcarea către o poziție specifică implică setarea poziției și orientării capătului brațului robotic folosind coordonate carteziene.

Definirea poziției țintă:

**geometry\_msgs::Pose target\_pose1;**

**target\_pose1.orientation.w = 1.0; // Orientarea fără rotație**

**target\_pose1.position.x = 0.28; // Coordonata X**

**target\_pose1.position.y = -0.2; // Coordonata Y**

**target\_pose1.position.z = 0.5; // Coordonata Z**

 geometry\_msgs::Pose este utilizat pentru a defini poziția 3D și orientarea.

 Orientarea este exprimată printr-un quaternion (orientation.w indică rotație zero).

Setarea țintei:

**move\_group\_interface.setPoseTarget(target\_pose1);**

Aceasta setează poziția dorită a capătului brațului robotic.

Planificarea traiectoriei:

**moveit::planning\_interface::MoveGroupInterface::Plan my\_plan;**

**bool success = (move\_group\_interface.plan(my\_plan) == moveit::core::MoveItErrorCode::SUCCESS);**

Planificatorul MoveIt calculează o traiectorie pentru a ajunge la target\_pose1

Executarea mișcării:

**if (success) {**

**move\_group\_interface.execute(my\_plan);**

**} else {**

**ROS\_WARN\_NAMED("tutorial", "Planificarea a eșuat, nu executăm traiectoria");**

**}**

**4.2. Miscare in Spatiul Articulatiilor**

În acest caz, poziția brațului robotic este setată direct prin valorile articulațiilor (unghiurile acestora).

Obținerea stării curente a articulațiilor:

**moveit::core::RobotStatePtr current\_state = move\_group\_interface.getCurrentState();**

**std::vector<double> joint\_group\_positions;**

**current\_state->copyJointGroupPositions(joint\_model\_group, joint\_group\_positions);**

 current\_state conține informații despre poziția curentă a articulațiilor.

 Valorile articulațiilor sunt copiate într-un vector joint\_group\_positions.

Setarea unei poziții noi:

**joint\_group\_positions[0] = -tau / 6; // -60 de grade**

**move\_group\_interface.setJointValueTarget(joint\_group\_positions);**

Modificăm poziția primei articulații, iar setJointValueTarget setează ținta.

Limitarea vitezei și accelerației:

**move\_group\_interface.setMaxVelocityScalingFactor(0.05);**

**move\_group\_interface.setMaxAccelerationScalingFactor(0.05);**

Aceste funcții ajustează viteza și accelerația maximă pentru o mișcare mai lină.

**4.3. Planificare cu Constrangeri de Orientare**

Constrângerile de orientare sunt utilizate pentru a forța brațul robotic să păstreze o anumită orientare în timpul mișcării.

Definirea constrângerii de orientare:

**moveit\_msgs::OrientationConstraint ocm;**

**ocm.link\_name = "panda\_link7";**

**ocm.header.frame\_id = "panda\_link0";**

**ocm.orientation.w = 1.0; // Orientare dorită**

**ocm.absolute\_x\_axis\_tolerance = 0.1;**

**ocm.absolute\_y\_axis\_tolerance = 0.1;**

**ocm.absolute\_z\_axis\_tolerance = 0.1;**

**ocm.weight = 1.0;**

OrientationConstraint specifică link-ul pentru care aplicăm constrângerea și toleranțele axiale.

Adăugarea constrângerii:

**moveit\_msgs::Constraints test\_constraints;**

**test\_constraints.orientation\_constraints.push\_back(ocm);**

**move\_group\_interface.setPathConstraints(test\_constraints);**

Ștergerea constrângerilor:

**move\_group\_interface.clearPathConstraints();**

**4.4. Planificare Carteziana**

Planificarea carteziană permite robotului să urmeze o traiectorie formată din puncte intermediare.

Definirea punctelor intermediare:

**std::vector<geometry\_msgs::Pose> waypoints;**

**geometry\_msgs::Pose start\_pose = move\_group\_interface.getCurrentPose().pose;**

**waypoints.push\_back(start\_pose);**

**start\_pose.position.z -= 0.2; // Coborâm**

**waypoints.push\_back(start\_pose);**

**start\_pose.position.y -= 0.2; // Mergem la dreapta**

**waypoints.push\_back(start\_pose);**

Calcularea traiectoriei:

**moveit\_msgs::RobotTrajectory trajectory;**

**double fraction = move\_group\_interface.computeCartesianPath(waypoints, 0.01, 0.0, trajectory);**

**ROS\_INFO("Cartesian Path Achieved: %.2f%%", fraction \* 100.0);**

Executarea traiectoriei:

**move\_group\_interface.execute(trajectory);**

**4.5. Planificare cu Obstacole**

Aici adăugăm obstacole în scenă și planificăm mișcări care le evită.

Definirea unui obstacol:

**moveit\_msgs::CollisionObject collision\_object;**

**collision\_object.header.frame\_id = move\_group\_interface.getPlanningFrame();**

**collision\_object.id = "box1";**

**shape\_msgs::SolidPrimitive primitive;**

**primitive.type = primitive.BOX;**

**primitive.dimensions = {0.1, 1.5, 0.5};**

**geometry\_msgs::Pose box\_pose;**

**box\_pose.orientation.w = 1.0;**

**box\_pose.position.x = 0.5;**

**box\_pose.position.y = 0.0;**

**box\_pose.position.z = 0.25;**

**collision\_object.primitives.push\_back(primitive);**

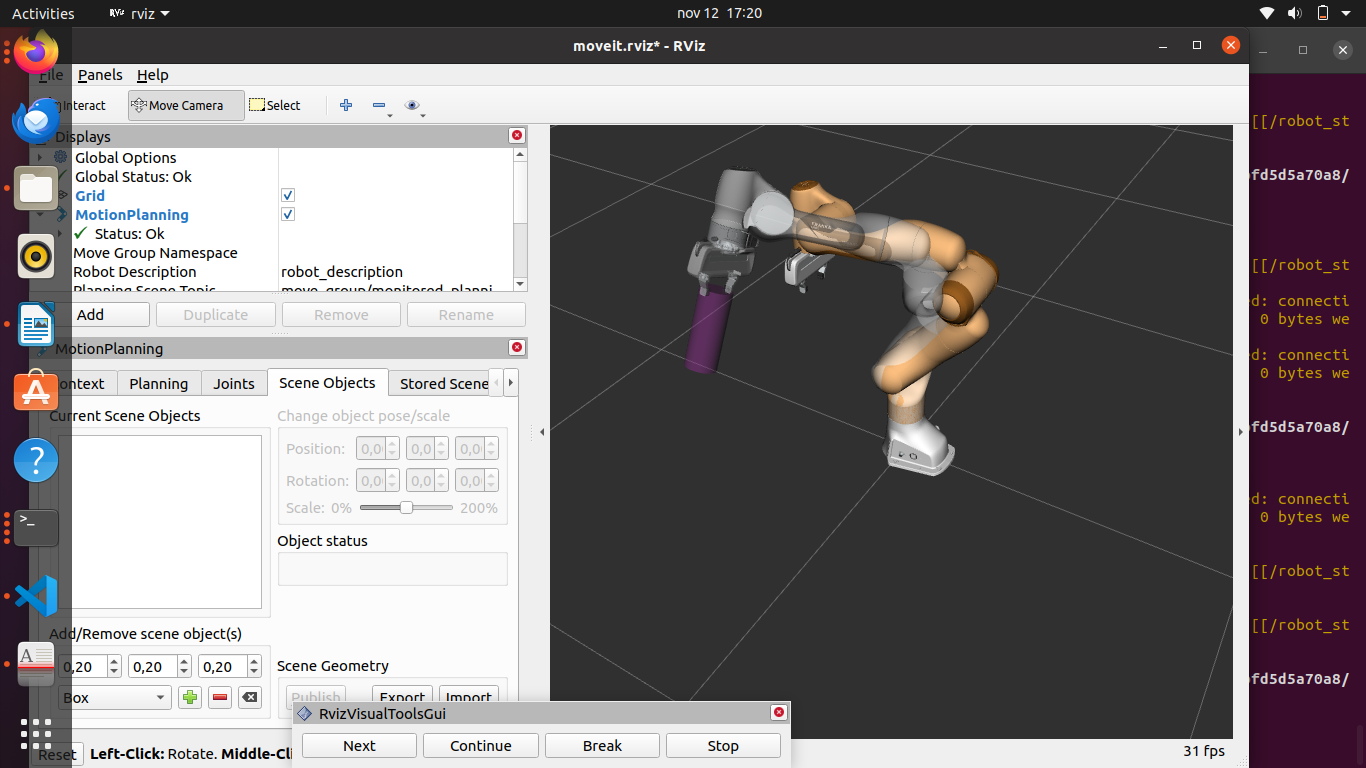
**collision\_object.primitive\_poses.push\_back(box\_pose);**

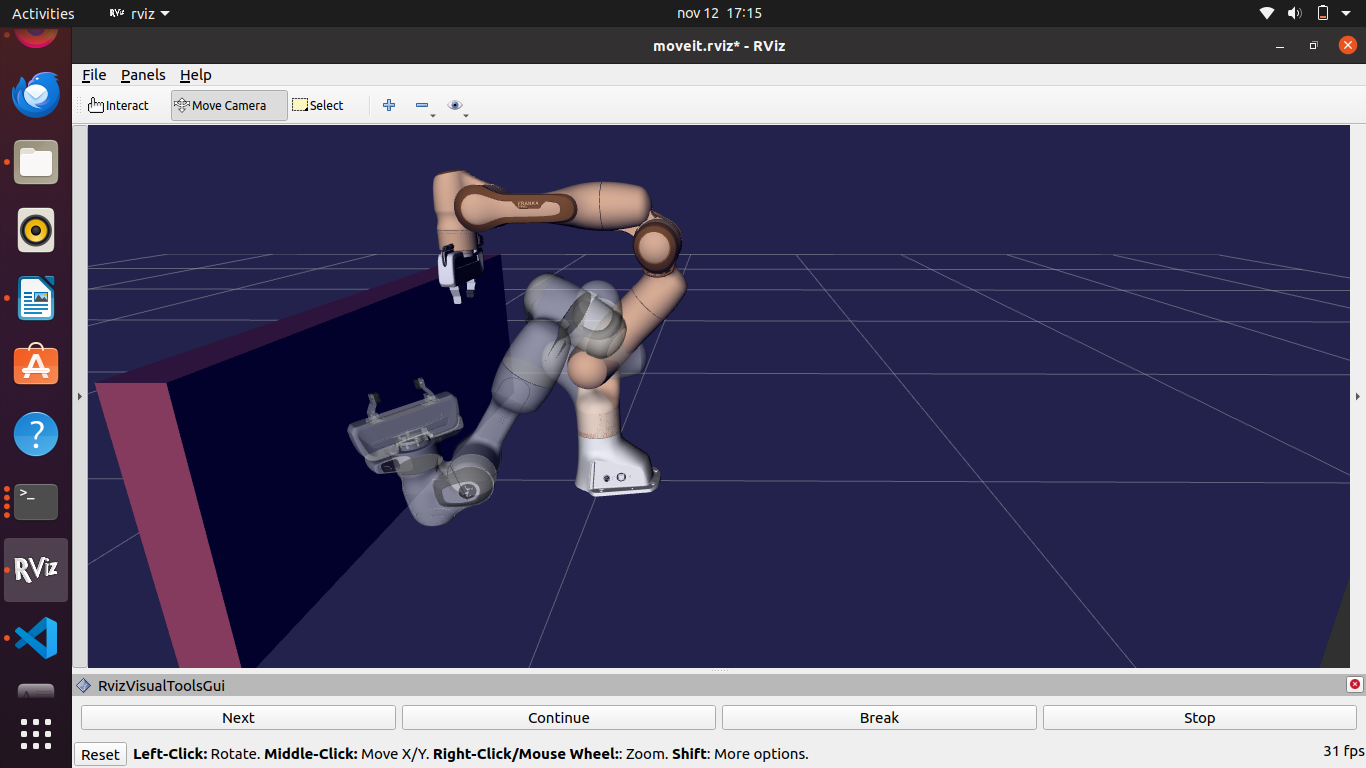
**collision\_object.operation = collision\_object.ADD;**

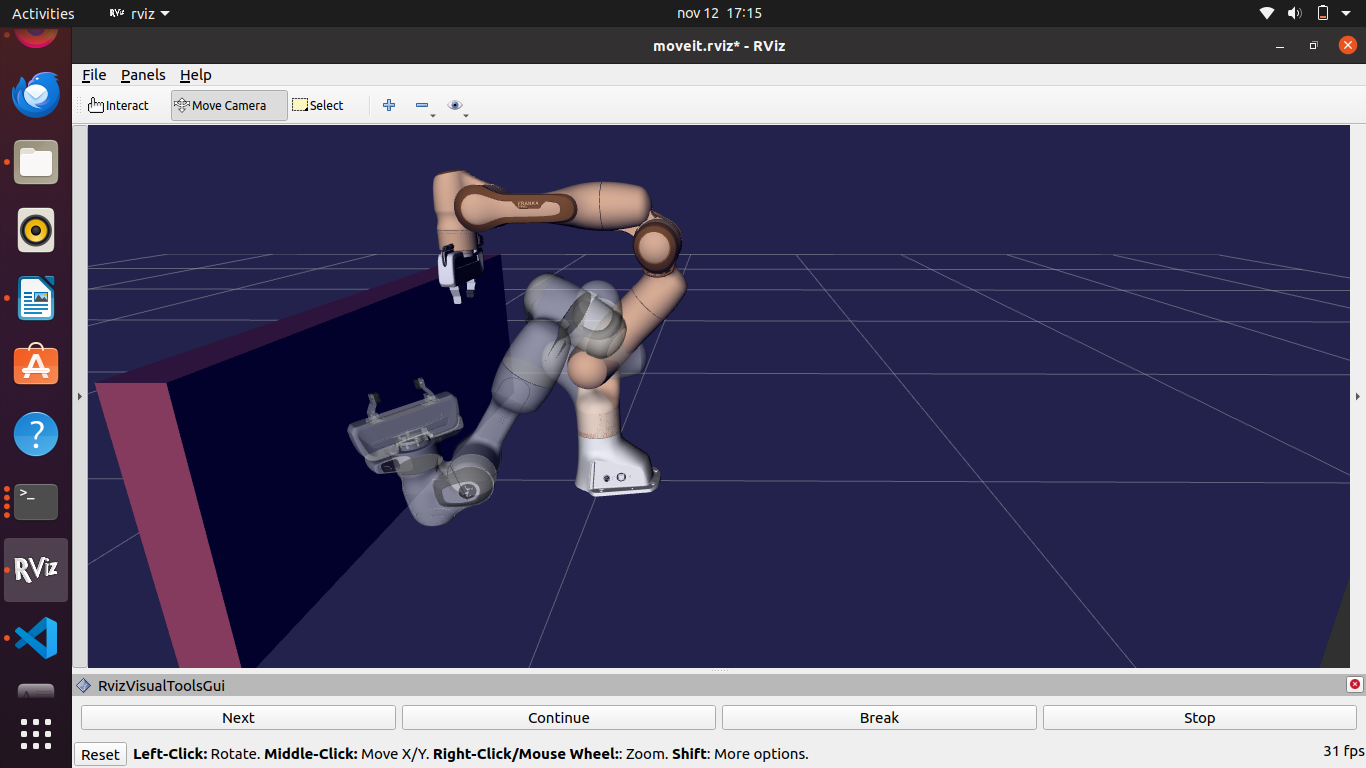
Adăugarea obstacolului în scenă:

**std::vector<moveit\_msgs::CollisionObject> collision\_objects = {collision\_object};**

**planning\_scene\_interface.addCollisionObjects(collision\_objects);**

****

****

****

Mi-am initailizat sa am si o interfata grafica( e un un pachet de instalat) pentru butoane de genul next, continue, break, stop ca sa stie ce comanda sa imi dea.

Pentru a modifica codul trebuie salvate modificarile in editorul de text sau eu folosesc vs code apoi ii dau catkin\_make ca sa imi reactulizeze spatiul in care lucrez, compilez pe scurt, pentru a actualiza environmental ros voi scrie source devel/setup.bash si apoi intr-un terminal voi rula roslaunch panda\_moveit\_config demo.launch si deschid un alt terminal unde voi rula codul meu rosrun moveit\_tutorials move\_group\_interface\_tutorial

**5.Miscare Brat Inverse Kinematics**

**5.1. Ce este Inverse Kinematics?**

**Inverse Kinematics (IK)** este procesul de calcul al valorilor unghiulare ale articulațiilor unui robot astfel încât capătul efector (end-effector) să atingă o poziție și orientare specificate într-un spațiu tridimensional.

**Scop:** De exemplu, dacă dorim ca un braț robotic să prindă un obiect aflat la o anumită poziție, IK calculează cum trebuie să fie rotite articulațiile pentru ca efectorul să ajungă în poziția și orientarea dorite.

**5.3. Explicatie cod inverse kinematic**

**Datorita faptului ca este un cod foarte complex o sa explic aici doar partile importante si unde am folosit mai exact inverse kinematic.**

**Partile importante din cod sunt:**

**1.Initializarea ROS si Moveit**

**ros::init(argc, argv, "move\_group\_interface\_tutorial");**

**ros::NodeHandle node\_handle;**

**ros::AsyncSpinner spinner(1);**

**spinner.start();**

**2.** Crearea grupului de mișcare

**static const std::string PLANNING\_GROUP = "panda\_arm";**

**moveit::planning\_interface::MoveGroupInterface move\_group(PLANNING\_GROUP);**

PLANNING\_GROUP este un nume care se referă la brațul robotic (definit în fișierul MoveIt! config).

move\_group este un obiect care permite controlul articulațiilor și planificarea mișcărilor.

**3.** Calculul cinematicei inverse și planificarea

move\_group.setPoseTarget(target\_pose1);

**moveit::planning\_interface::MoveGroupInterface::Plan my\_plan;**

**bool success = (move\_group.plan(my\_plan) == moveit::planning\_interface::MoveItErrorCode::SUCCESS);**

Funcția setPoseTarget calculează automat **cinematica inversă**, transformând poziția și orientarea dorite în unghiuri ale articulațiilor.

plan generează o traiectorie care duce brațul din poziția curentă în poziția țintă.

Apoi se executa.

**Funcția pickWithInverseKinematics**

**1. Configurare poziție de prindere**

Cod:

grasps[0].grasp\_pose.header.frame\_id = "panda\_link0";

tf2::Quaternion orientation;

orientation.setRPY(-tau/4, -tau/8, -tau/4);

grasps[0].grasp\_pose.pose.orientation = tf2::toMsg(orientation);

grasps[0].grasp\_pose.pose.position.x = 0.415;

grasps[0].grasp\_pose.pose.position.y = 0;

grasps[0].grasp\_pose.pose.position.z = 0.5;

* **Orientare:** Folosește un quaternion pentru rotații 3D, convertit în format ROS cu tf2::toMsg.
* **Poziție:** Specifică coordonatele X, Y, Z ale punctului țintă pentru prindere.

**2. Configurare traiectorie de apropiere**

Cod:

grasps[0].pre\_grasp\_approach.direction.header.frame\_id = "panda\_link0";

grasps[0].pre\_grasp\_approach.direction.vector.y = -1.0;

grasps[0].pre\_grasp\_approach.min\_distance = 0.095;

grasps[0].pre\_grasp\_approach.desired\_distance = 0.115;

* **Direcție:** Brațul se deplasează în direcția -Y față de cadrul de referință "panda\_link0".
* **Distanță:** min\_distance definește distanța minimă de apropiere, iar desired\_distance specifică distanța finală.

**3. Configurare retragere după prindere**

Cod:

grasps[0].post\_grasp\_retreat.direction.header.frame\_id = "panda\_link0";

grasps[0].post\_grasp\_retreat.direction.vector.z = 1.0;

grasps[0].post\_grasp\_retreat.min\_distance = 0.1;

grasps[0].post\_grasp\_retreat.desired\_distance = 0.25;

* **Direcție:** Brațul se retrage pe axa Z.
* **Distanțe:** Setează o retragere lentă și controlată.

**4. Configurare postură gripper**

Cod:

openGripper(grasps[0].pre\_grasp\_posture);

closedGripper(grasps[0].grasp\_posture);

* **Postură deschisă:** Setează gripper-ul pentru a prinde obiectul.
* **Postură închisă:** Închide gripper-ul după prindere.

**5. Execuție mișcare de prindere**

Cod:

move\_group.setSupportSurfaceName("table1");

move\_group.pick("object", grasps);

* **Suprafață suport:** Definirea suprafeței pe care se află obiectul.
* **Execuție:** move\_group.pick calculează traiectoria folosind IK și o execută.

**Funcția placeWithInverseKinematics**

**1. Configurare poziție de plasare**

Cod:

place\_location[0].place\_pose.header.frame\_id = "panda\_link0";

tf2::Quaternion orientation;

orientation.setRPY(0, 0, tau / 4);

place\_location[0].place\_pose.pose.orientation = tf2::toMsg(orientation);

place\_location[0].place\_pose.pose.position.x = 0;

place\_location[0].place\_pose.pose.position.y = 0.5;

place\_location[0].place\_pose.pose.position.z = 0.5;

* **Orientare:** Se rotește pe axa Z pentru plasare precisă.
* **Poziție:** Setează coordonatele țintă pentru plasarea obiectului.

**2. Configurare traiectorie de apropiere**

Cod:

place\_location[0].pre\_place\_approach.direction.header.frame\_id = "panda\_link0";

place\_location[0].pre\_place\_approach.direction.vector.z = -1.0;

place\_location[0].pre\_place\_approach.min\_distance = 0.095;

place\_location[0].pre\_place\_approach.desired\_distance = 0.115;

* **Direcție:** Se deplasează pe axa Z în jos, spre poziția de plasare.
* **Distanțe:** Setează parametrii pentru o apropiere controlată.

**3. Configurare retragere după plasare**

Cod:

place\_location[0].post\_place\_retreat.direction.header.frame\_id = "panda\_link0";

place\_location[0].post\_place\_retreat.direction.vector.x = -1.0;

place\_location[0].post\_place\_retreat.min\_distance = 0.1;

place\_location[0].post\_place\_retreat.desired\_distance = 0.25;

* **Direcție:** Retragerea se face pe axa X.

**4. Configurare postură gripper**

Cod:

openGripper(place\_location[0].post\_place\_posture);

* **Postură deschisă:** Gripper-ul este deschis pentru a elibera obiectul.

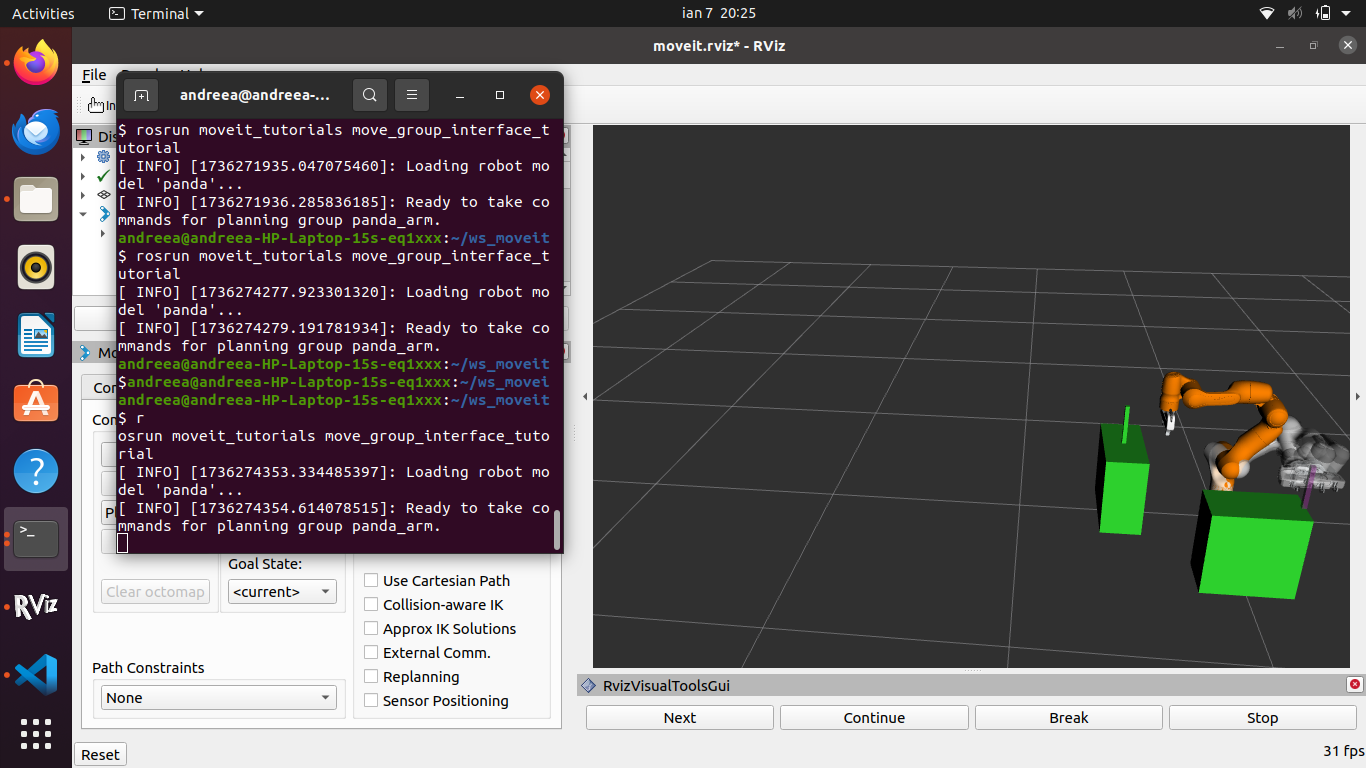
**5. Execuție mișcare de plasare**

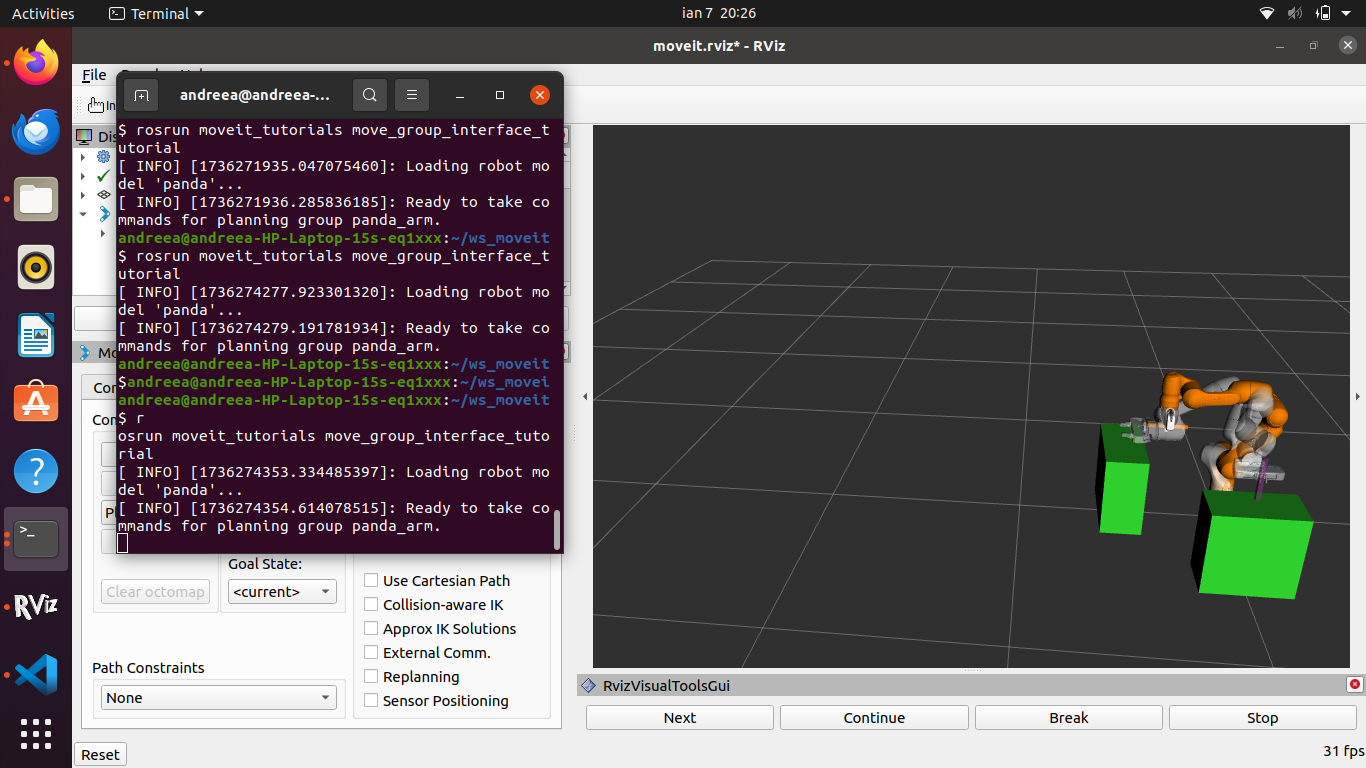
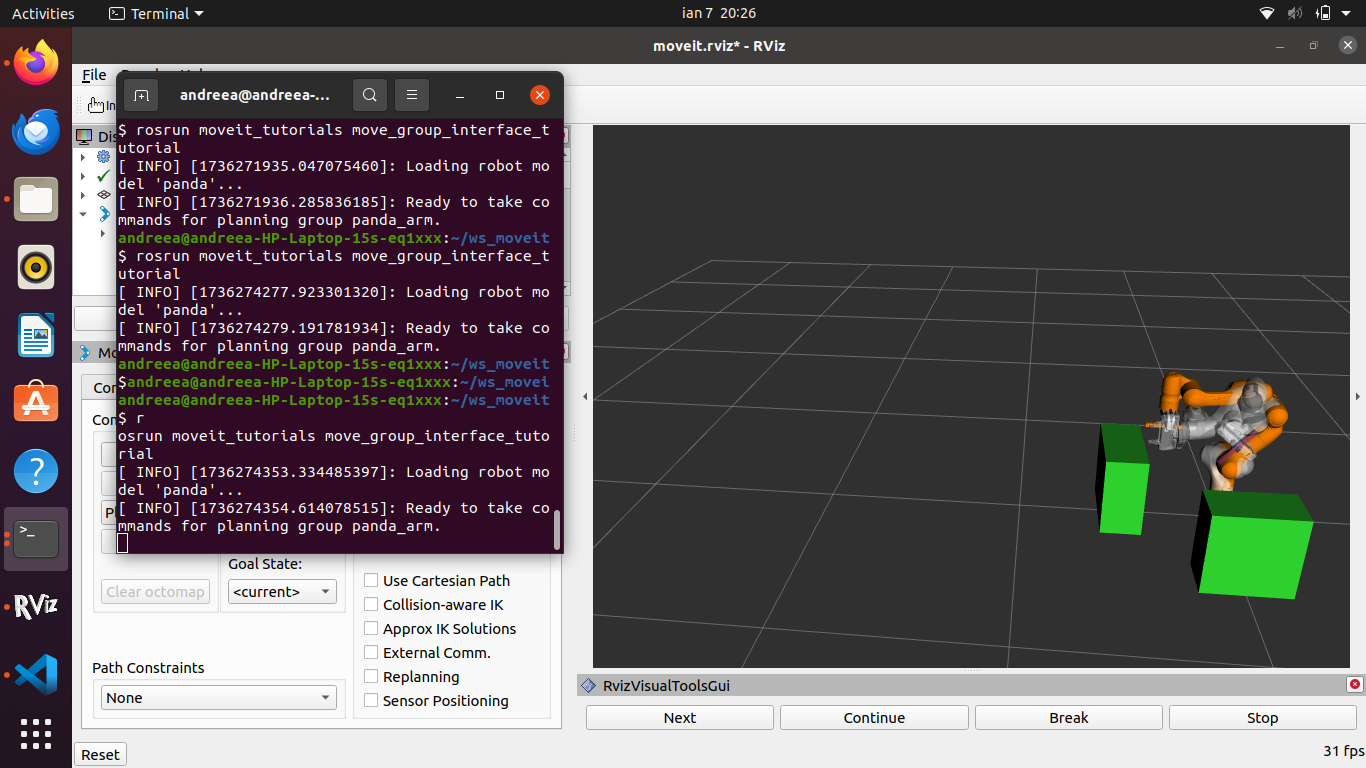
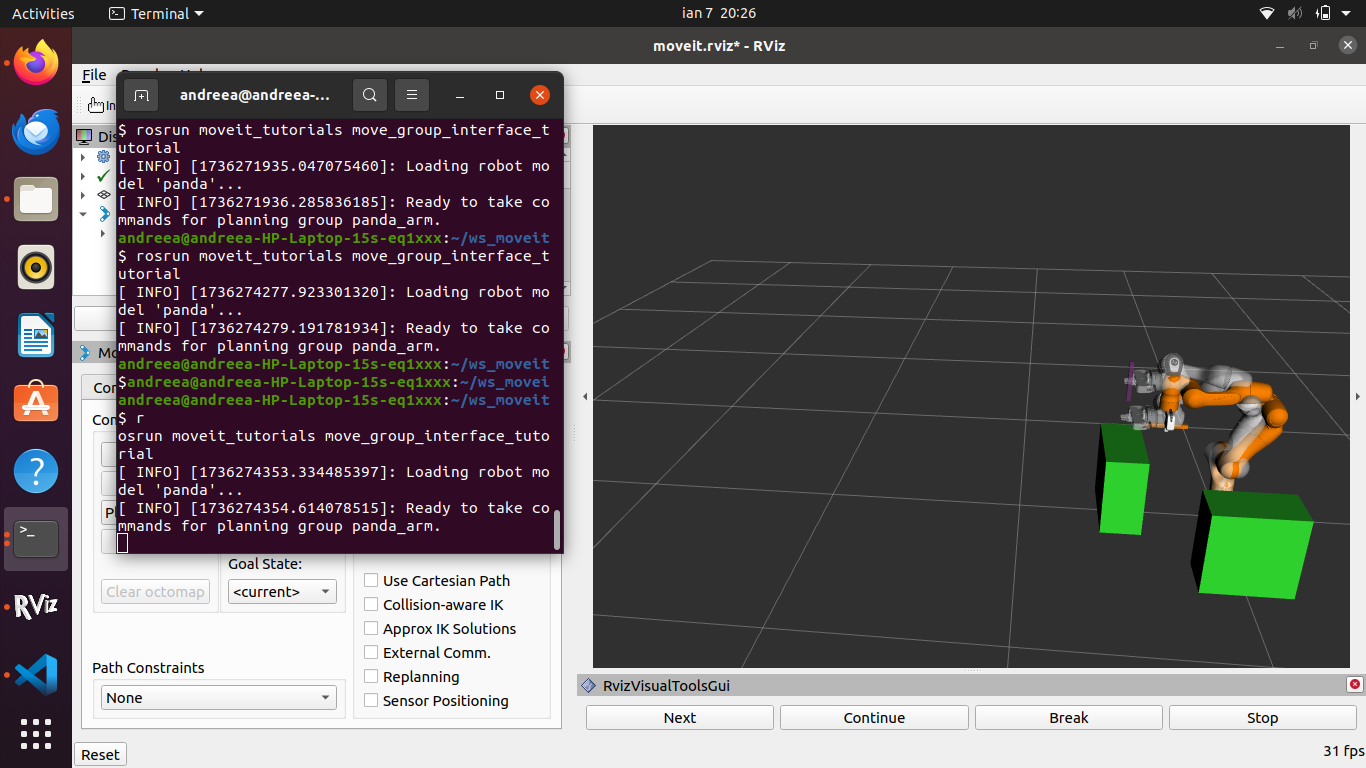
Cod:

move\_group.setSupportSurfaceName("table2");

move\_group.place("object", place\_location);

* **Suprafață suport:** Definirea suprafeței unde se va plasa obiectul.
* **Execuție:** move\_group.place utilizează IK pentru plasare.

****

****

**6.Probleme Camera Virtuala**

În timpul implementării unei camere virtuale în mediul ROS Noetic, am întâmpinat dificultăți majore în ceea ce privește vizibilitatea nodurilor ROS. Sistemul nu reușea să detecteze nodurile asociate camerei virtuale, împiedicând astfel funcționarea corectă.

* Nodurile ROS asociate cu camera virtuală nu erau vizibile în sistem
* Comenzile rosnode list și rostopic list nu afișau nodurile așteptate
* Problemă de comunicare între componentele sistemului

**Solutia ar fi sa instalez de la 0 un alt sistem de operare mai performant si sa imi refac tot setup-ul de la 0 cu alta versiune de moveit si ros.**

**7. Bibliografie**

<https://www.youtube.com/watch?v=9GeyCgNqoDw>

<https://github.com/LearnRoboticsWROS/cobot_IK/blob/main/src/ik_test.cpp>

<https://www.youtube.com/watch?v=ggOROufX0tE&t=32s>

<https://moveit.github.io/moveit_tutorials/doc/move_group_interface/move_group_interface_tutorial.html>

<https://wiki.ros.org/noetic/Installation/Ubuntu>

<https://moveit.ai/install/source/>

<https://moveit.github.io/moveit_tutorials/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html>