Assembly Line Manipulator

PROJECT REPORT

Submitted by

BL.EN. U4AIE19039	Mandiga Sahasra Sai Tarun
BL.EN. U4AIE19055	S Venkatasubramanian
BL.EN. U4AIE19069	VISHAL.C

in partial fulfilment for the award of the degree
0f

BACHELOR OF TECHNOLOGY

IN

"ARTIFICIAL INTELLIGENCE" ENGINEERING

For the subject

19AIE213 - Robotic Operating System and Robot Simulation

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BANGALORE 560 035

May-2021

AMRITA VISHWA VIDYAPEETHAM

PROBLEM STATEMENT

A typical assembly line manipulator are arranged new to each other and works co-ordinately. A manipulator will start its work from the previous robot end task. Design an assembly line of two pick and place manipulator located side by side, with required degree of freedom. The input for the system is pick and place location of robot 1 and place location of robot 2, pick place for robot 2 is same as place location robot 1. Below picture is just to represent the physical structure.

Our project goal is create assembly line manipulator that enables two robot arms to perform simple object manipulation tasks. A typical assembly manipulators are arranged in a such a way with each other that it works co-ordinately. A manipulator will start its work from the previous robot end task. So in our problem statement we were asked design a assembly line of two pick and place manipulator located side by side. The input for the system is pick and place location of robot 1 and place location of robot 2, pick place for robot 2 is same as place location robot 1. We accomplished this goal using ROS package MoveIt. The MoveIt setup assistant tool, helped us to configure our robot arms to work for our application. After configuring the required poses and configuration, we have used move groups ROS node available in Moveit ros package to plan and execute motions.

INTRODUCTION

Since the beginning of the study of robotics, there has been some controversy in the definition of a robot. So long as the evolution of robotics continues, the definition of the robot will change from time to time, depending on the technological advances in its sensory capability and level of intelligence. Robotic manipulators resembling the human arm is known as robotic arms. They are constructed by a structure consisting of structurally robust links coupled by either rotational joints or translating joints. A robotic arm is thus a type of mechanically coupled or joined arm, run by programmable commands, with similar functions to a human arm. It may be the sum total of the mechanism links or may be part of more complex sized robot.

A typical robotic arm has the following components:

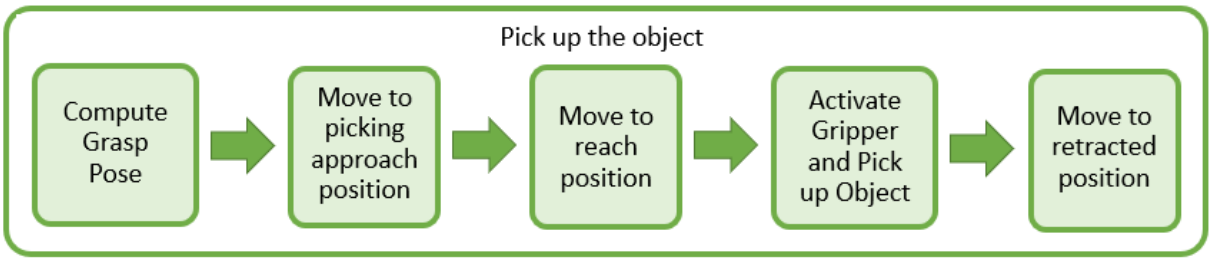
- Links and Joints
- Actuators
- Controller
- End-effector

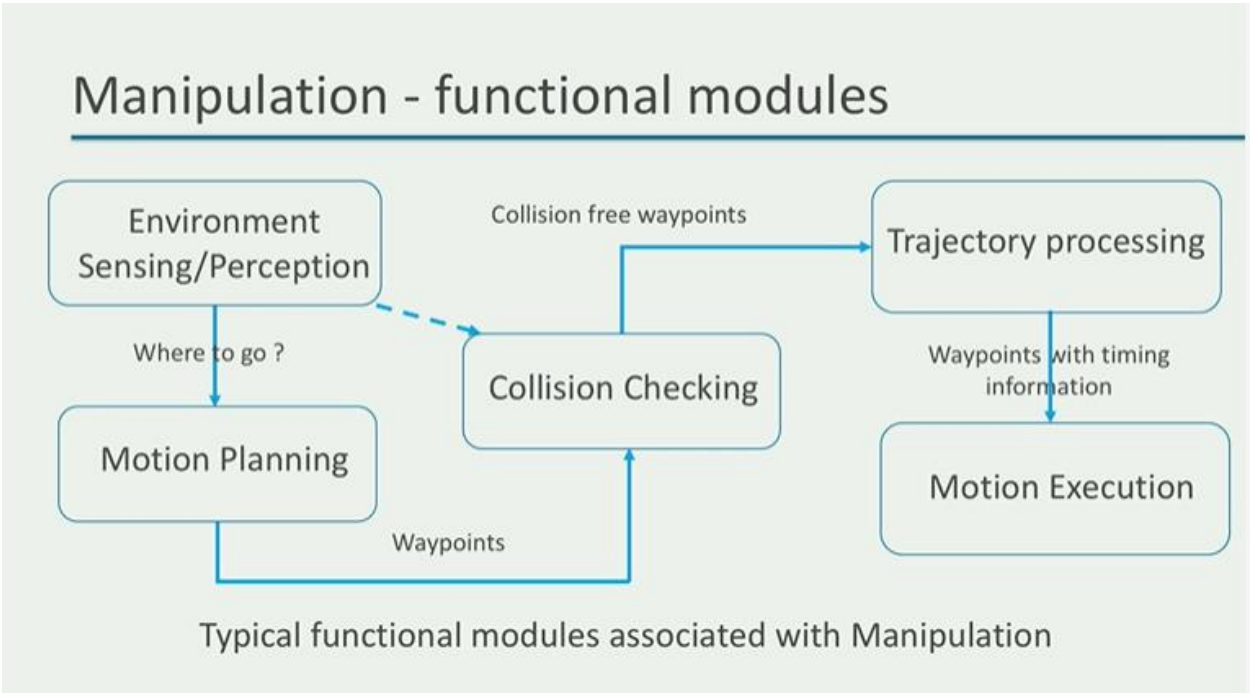
A link is considered as a rigid body that defines the relationship between two corresponding joint axes of a manipulator. Manipulators consist of rigid links, which are connected by joints that allow relative motion of corresponding links. The links move to position with the end-effector. Actuators perform the same role the muscles perform in the human arm – they convert stored energy into movement energy. Actuators used force to move a robot's manipulator joints. The three common types of actuators currently using in contemporary robots are pneumatic, hydraulic, and electrical actuators.

The controller is the main part that processes information and carries out instructions in a robot. It is the robot's 'brain' and controls the robot's movements. It is usually a computer of some type which is used to keep information about the robot and the working process and execute programs which operate the robot. It contains programs, data algorithms, logic analysis and various other processing activities which enable the robot to perform its intended function.

End-effector is a device at the end of a robotic arm, designed to interact with the open world. The exact nature of performance of this device depends on the application of the robot. So, in our project the typical functions of the end-effectors are picking and placing.

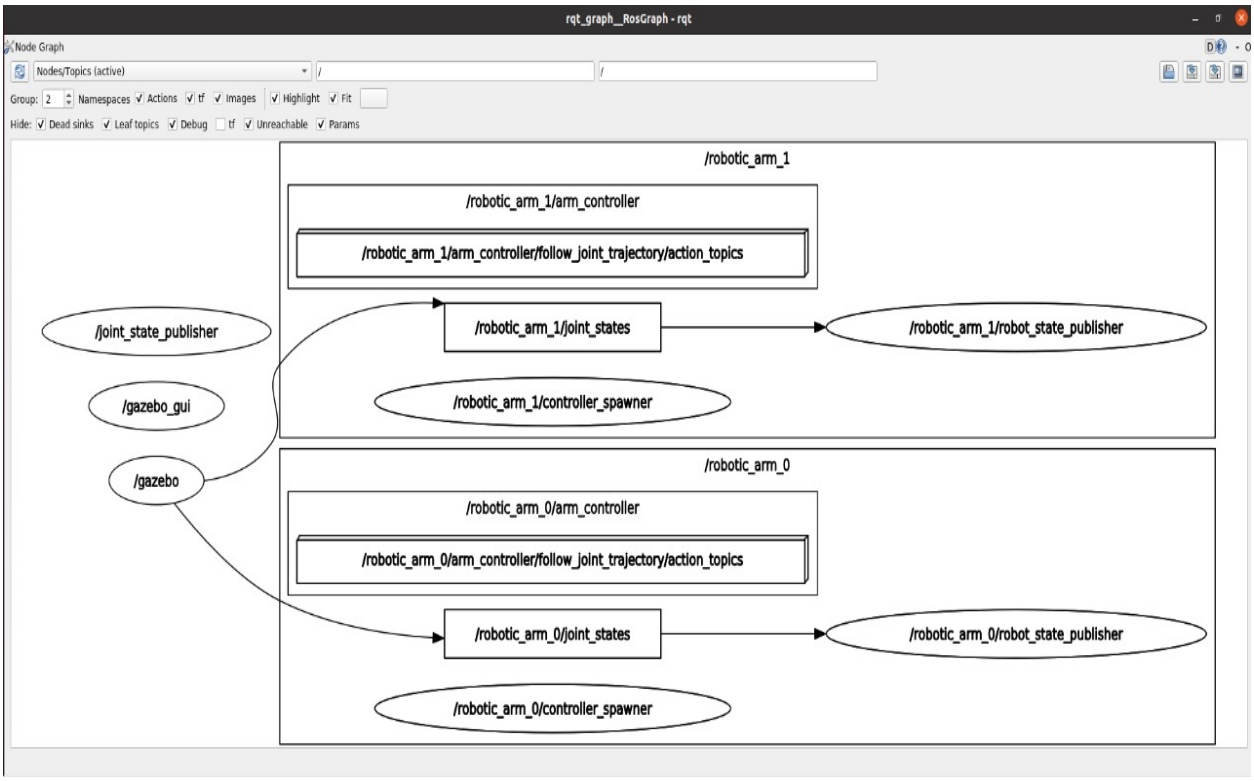
FLOW CHART:





In a general sense, all these steps can be viewed as different functional modules associated with manipulation. We need a module that senses or perceives the environment and that can find objects of interest in our environment. This way, we can obtain information on where to go. This information is passed on to the motion planning module that can provide a set of waypoints to wherever we want to go in the environment. But, we have to make sure that these waypoints are safe for the robot to go through so that it does not hit any previously known obstacles and any newly detected obstacles by the sensing module. Once the waypoints are guaranteed to be collision free, they can be passed on to the trajectory processing module. This module adds timing information to the different waypoints such that we are not only aware of where we should be but also at what times should we be at each waypoint. In essence, velocity and acceleration information is appended to the waypoints so that, the desired motions can be executed on a robot.

ROS NODE GRAPH:



The above figure is the ros node graph of our simulation. We have used
The Move Group interface is a collection of APIs to access the various capabilities of the move group ROS node of MoveIt. For our project we have two robotic arm groups named robotic_arm_0 and robotic_arm_1.
Trajectories are executed on simulated robots or real hardware, and they are controlled by *controllers.* Gazebo uses *controllers* to control the motion and to know the poses and status of the robot. MoveIt! plans movements and send those movements to the *controllers* in Gazebo. Gazebo uses the JointStateController to publish current joint values and JointTrajectoryController to control the execution of trajectories. . Communication between all these components happens over ROS topics and action servers. Trajectory controllers contain a name, a type, a list of associated joints, the gains, their constraints, and some other information to do with tolerances. Controllers are provided as ROS action servers and they need to be powered by nodes. We can launch them using launch files.

Note : The end effector used in our project is a vacuum gripper. The vacuum gripper plugin is used to grab the blue object.

Commands to run the simulation:

- roslaunch team21 gazebo.launch (for launching gazebo simulator)
- rosrun team21 arm_controller.py (to run the python ros node)

POSES OF THE MANIPULATOR

POSE 1:

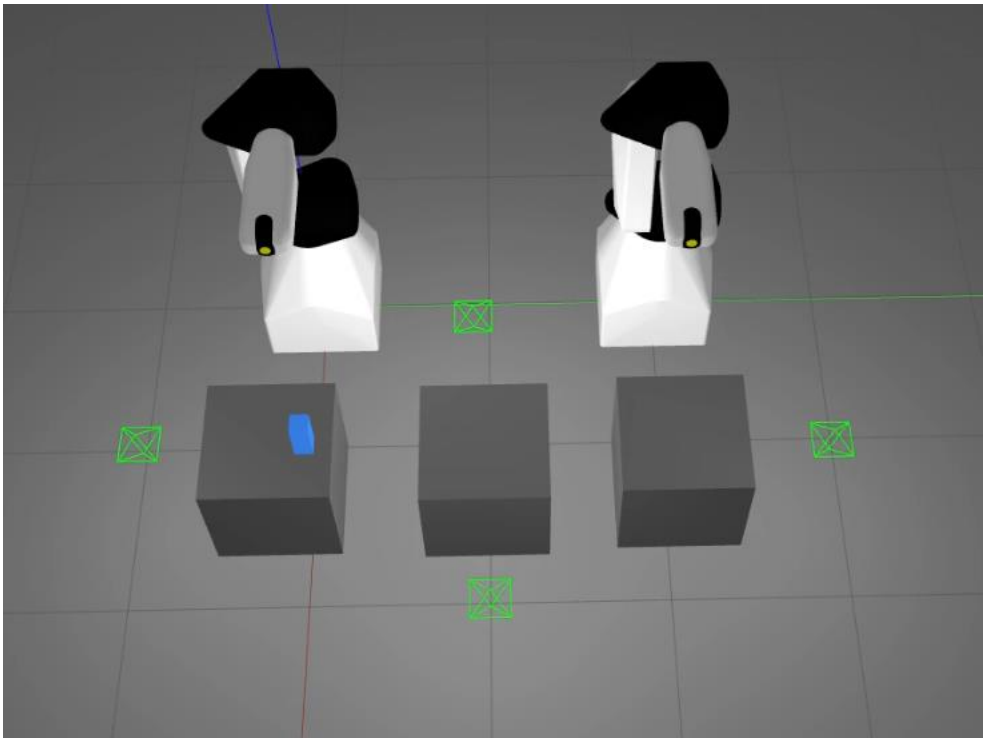


Fig (1)

The typical assembly manipulators are arranged in a such a way with each other that it works co-ordinately. In this the other manipulator will start its work from the previous robot's end task. So, in our problem statement we were asked to design an assembly line of two pick and place manipulator located side by side.

The above figure 1 shows the idle state of the manipulators before performing the task of picking and placing the object(blue block) .

POSE 2:

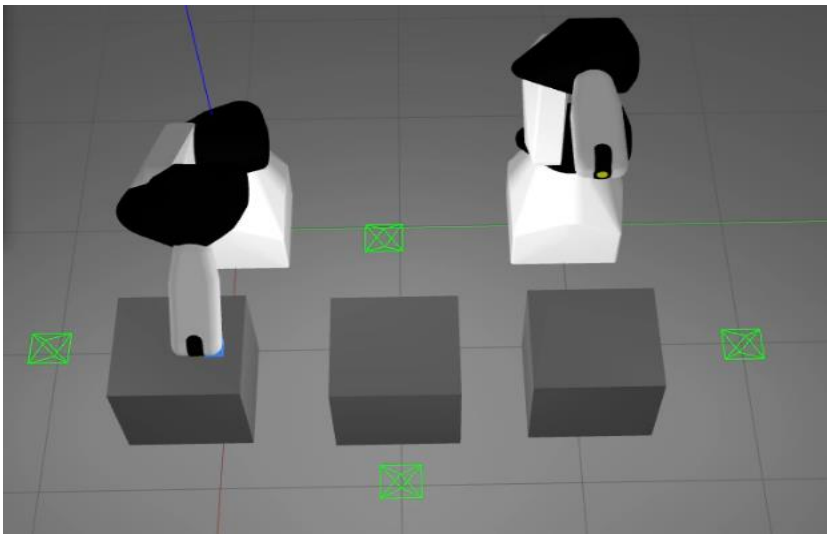


Fig (2)

The input for the system is pick and place location of robot 1 and place location of robot 2, pick place for robot 2 is same as place location robot 1. The above figure shows the pose executed by 1st robotic arm in order to pick the object.

POSE 3:

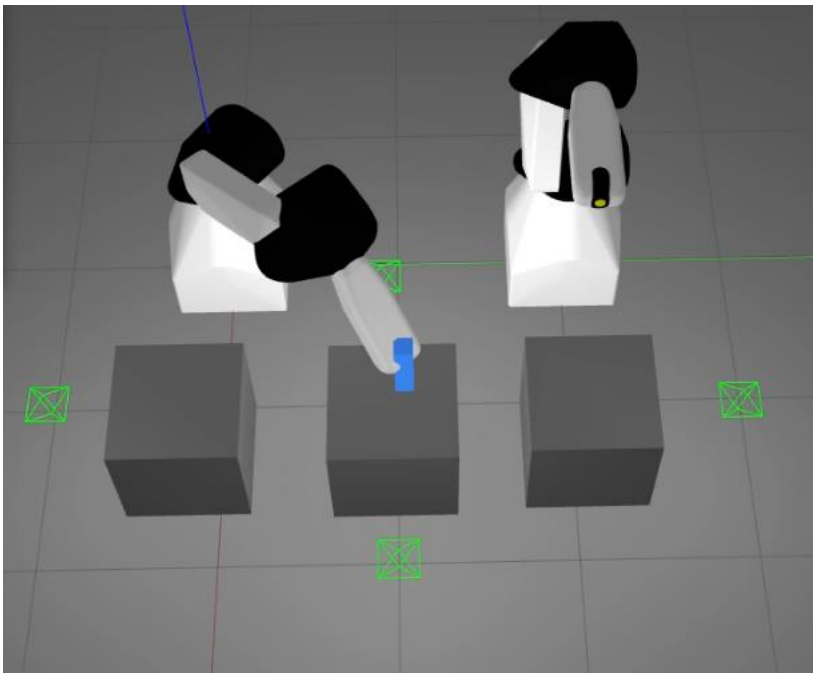


Fig (3)

The above figure 3 shows the pose executed by the first robotic arm to place the object on the centre box. After the blue block has been placed successfully, the 1st robotic arm returns to it's original pose.

POSE 4:

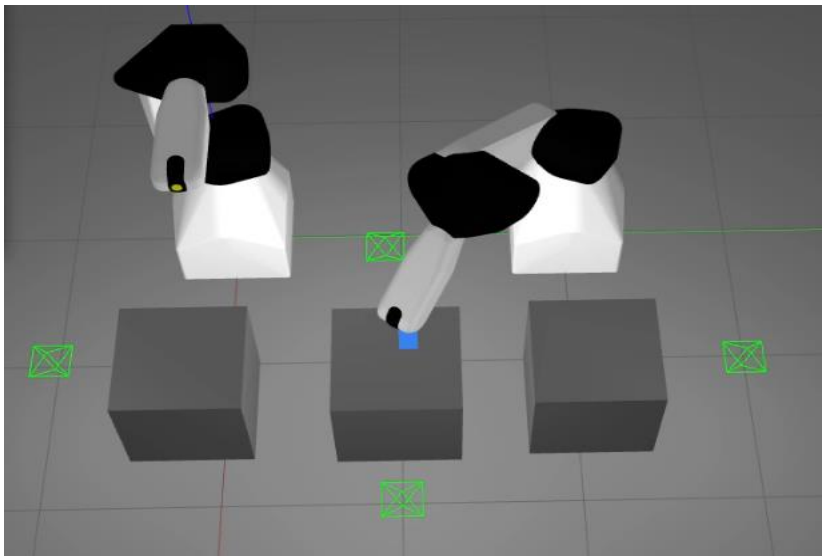


Fig (4)

The above figure 4 shows the pose executed by 2nd robotic arm to pick the object from the centre box.

POSE 5:

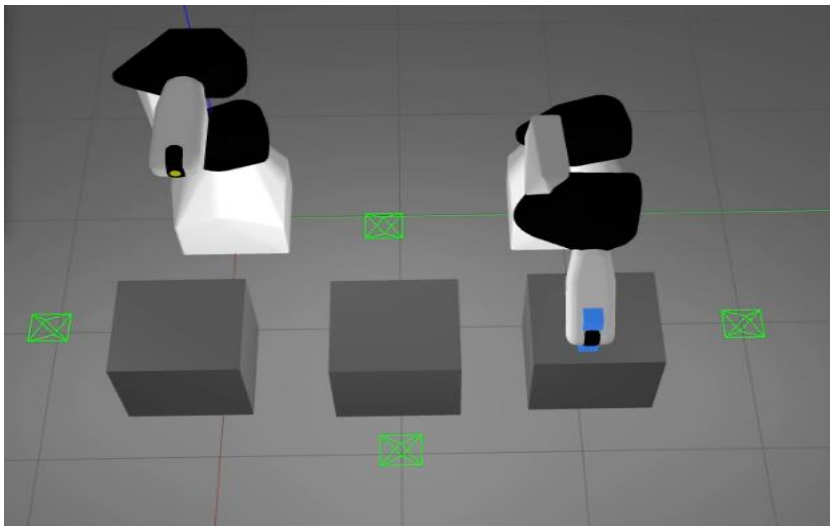


Fig (5)

The above figure 5 shows the pose executed by 2nd robotic arm to place the object in front of it.

POSE 6:

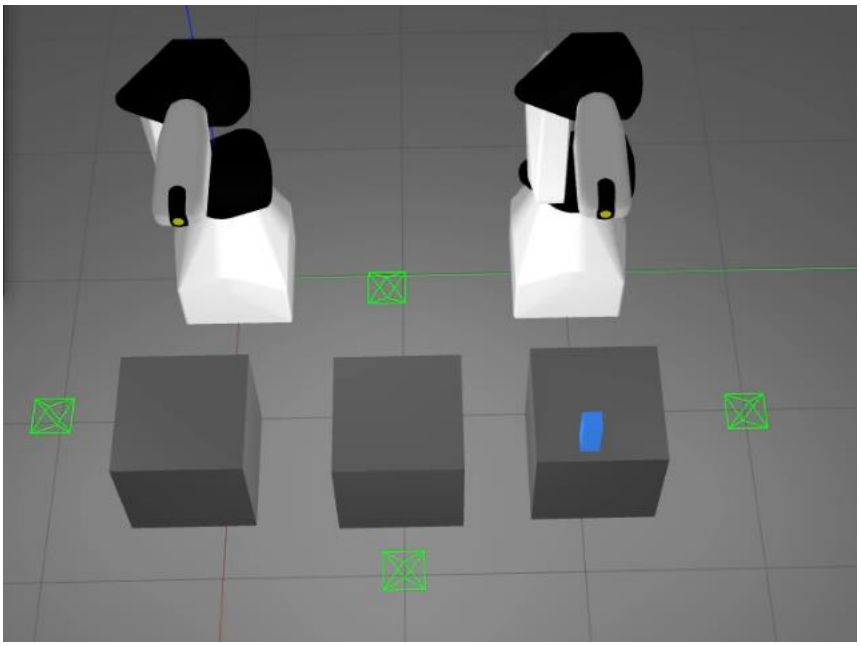


Fig (6)

The above figure 6 shows the pose executed by 2nd robotic arm to return to it's original pose.

Model: https://github.com/ros-industrial/fanuc/tree/indigo-devel

References:

http://jderobot.github.io/RoboticsAcademy/exercises/IndustrialRobots/pick_place

https://learning.edx.org/course/course-v1:DelftX+ROS1x+1T2020/home

https://ros-planning.github.io/moveit_tutorials/

https://github.com/ros-industrial/universal_robot

https://www.youtube.com/watch?v=Yj5DEocFa48

https://www.youtube.com/watch?v=BPOLWBsOnOQ