University of Trento

Via Sommarive 9
38123,Trento (TN)

Members: Lorenzo Pieropan, Federico Buzzini, Andrea Gravili, Andrea Pedrini

Fundamentals of Robotics

September 11th 2023 - February 14th 2024

Table of Contents

Overview

The goal of this undertaking is to use a UR5 robot manipulator for the retrieval and repositioning of Lego blocks on a table. A 3-D stereo-digital camera, especially the ZED 2 by Stereolabs, is applied for block reputation, using pc vision techniques such as YOLOv5 and OpenCV for category. The simulation entails using the Universal Robots' fifth model of the 6-degree-of-freedom arm, and it's implemented within development environments like Gazebo and Rviz.

Simulation .world

In the world of Gazebo simulation there are four main objects:

- The Lego blocks of different classes
- The table where the Lego blocks are “naturally” laid
- The Ur5 manipulator
- The 3D camera

The 3D camera has to recognize the blocks on the table, retrieve the position and orientation of them. Then, they have to be grasped by the robot’s end effector and located in a precise position reserved for each different class of blocks.

Environment

The main software that facilitates communication between the project components is the ROS middleware (Robot Operating System). Nodes including robots, vision nodes, and motion planning nodes use objects and services to exchange messages.

The project consists of two main parts:

- All of the computer vision components focusing on object recognition and spatial retrieval were developed in Python.
- The motion planning part that controls robot kinematics and kinematics is developed mainly in C++.

Computer Vision

Dataset

To enable YOLOv8 to detect the Lego blocks, we had to create a training dataset consisting of images annotated with bounding box coordinates outlining the Lego blocks. To do so we utilized Roboflow software, we took images from the gazebo camera and we created the labels of our images. In this setup, the camera remained static, and the images captured varied only in terms of the arrangement of blocks on the table and their randomized attributes such as position, rotation, color, and class.

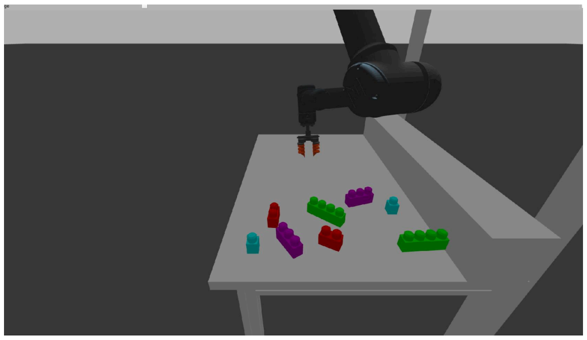
YOLO V8 Detection and Classification

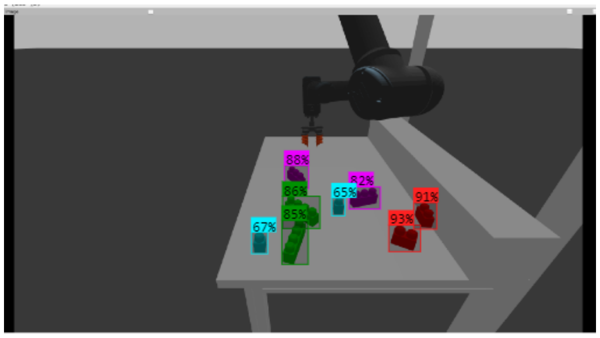
Those blocks with the corresponding bounding boxes are our training examples and their corresponding labels.

We used our training set with YOLOv8, which trained his neural network to detect the blocks and their corresponding bounding boxes, giving in output the weights of the trained neural network. We will use these weights to detect the blocks during the simulation.

Position determination using PointCloud data

Given our weights, now we can start our simulation and activate our ZED camera, the camera will take a photo of our table and detect the blocks on the table.





Observing the second image there is a confidence percentage, it indicates how confident the neural network is in recognising blocks.

Because each block has a bounding box, we just need to compute the mean of the four angles of each bounding box to obtain the center point of each block with respect to the image.

Finally we take the corresponding points in the pointcloud received from the ZED2. Once we have done it, we need to transform the frame system from the camera one, to the world one.

Communication with nodes

After we have computed the position of the blocks, we'll publish on the "/motion/pos" topic the coordinates of our block, waiting for an acknowledgement signal given from the "/vision/ack" topic which will

say if the motion has stopped the previous movement and it's prepared to have a new block position.

Task manager

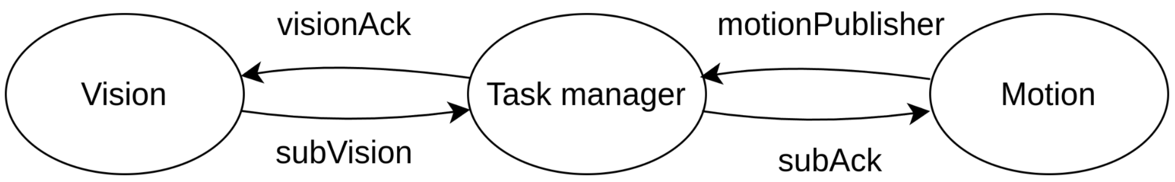
Objective

The task manager serves as an intermediary node between vision and motion nodes. Its purpose is to coordinate message exchange and enhance the efficiency of communication between vision and motion nodes

How it works

The task manager typically performs the following activities:

- Listens on the "vision" topic, awaiting the appearance of a block.
- Upon detecting a block, the task manager receives a message containing the coordinates of the detected block.
- Adjusts the height and waits for the motion to be ready to move, listening on the motion topic to determine when the manipulator can initiate a new movement.
- Sends a message to the motion node; upon completion, receives an acknowledgment (ack) from the motion node and sends an acknowledgment to the vision node to signal readiness for a new message transmission.



Details

The node does not process any frames but merely passes the height of the table at our location. Additionally, for any issues, a stop function can be called on the "taskManager" topic, instructing to halt the entire procedure in case of errors.

All control and waiting for the transmission of each message will not be handled in this portion of the code. Instead, any scenarios involving the arrival of external messages that do not affect the node's operation will be addressed in practice.

Messages

The message exchange will be defined by "motion/pos", and each message will contain the following information:

- int32 Class_id: the type of block being received (the class number indicates a specific block of dimensions NxM)
- float64 x: coordinate on the x-axis
- float64 y: coordinate on the y-axis
- float64 z: coordinate on the z-axis
- float64 roll: rotation angle along the x-axis
- float64 pitch: rotation angle along the y-axis
- float64 yaw: rotation angle along the z-axis

Robot Motion

The motion planner part is composed by these components:

- **kinematics**, both direct and inverse
- **motionPlanner.cpp**, compute the trajectory of the robot from a point to a point in the operational space

Kinematics

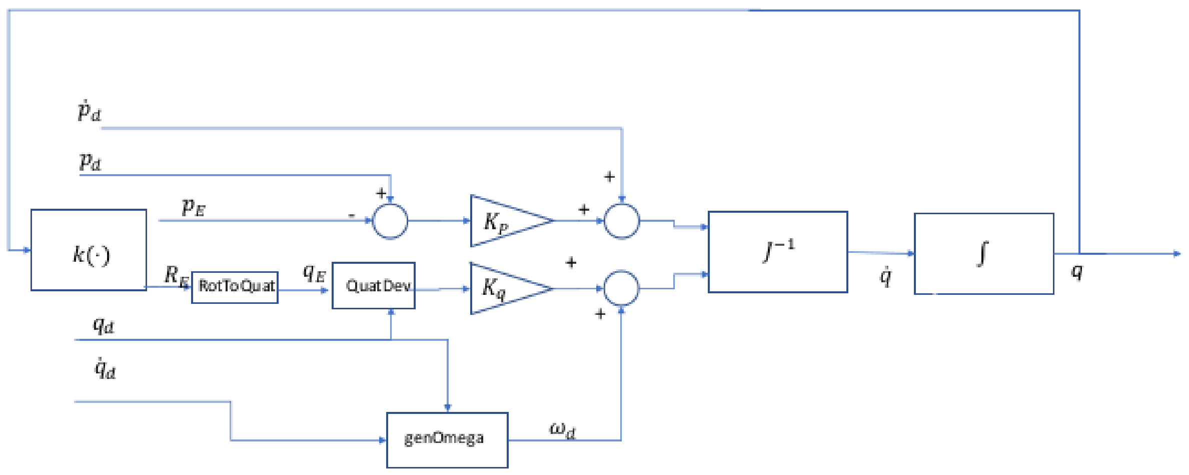
The direct kinematics describe the motion of points and bodies with transformation matrices. The function directKin(VectorXf th) get as input the set of six values of the joints and compute the transformation matrix for each joint and link, then it multiplies all the matrices to get the transformation matrix of the end-effector.

The function invKin(frame &frame) takes as input a three dimensional vector representing the final position of the end-effector and returns eight possible combinations of the joints value.

motionPlanner.cpp

Objective

The motion planner receives the coordinates of the final point and the orientation of the end-effector by the task manager and computes the joint states using the inverse kinematics. This is the control scheme:



The scheme produces exponential convergence toward the desired end-effector position and orientation: the position error $\mathbf{p_e}$ is asymptotically stable, and collapses to zero;
The orientation error $\mathbf{q_e}$, expressed through a quaternion, drops to zero as expressed by Lyapunov stability. Spherical Linear Interpolation is used to improve the movement of the actuator.

How it works

- Listen on the motion/pos subscriber and go on the starting position
- When arrives a message from the task manager plan a movement trajectory to the desired position of the block received
- Execute the movement based on the type of the block and move it in a determined position

- When the movement has finished return to the starting position and send an ack signal to the tasDetermine object's orientation using a zed camera with perspective and rotation.k planner node and restart to wait

Accomplishments Obtained

The main problems faced (and not solved) are:

- Determine objects orientation using a zed camera with perspective and rotation.
- Manage the robot's movement and remove singularities and obstacles in its trajectories and rotations.
- Troubleshoot grip issues in the gazebo simulator and troubleshoot problems that arise.

But even if those problems weren't able to be solved, we accomplished many other tasks that were difficult like those problems or much more. Some examples can be:

- Using quaternions for rotations in the space
- Detecting and using images of blocks for getting their coordinates
- Correct the simulation errors for the x-axis positioning based on the blocks.
- Manage the communication between the nodes in order to not have conflicts.
- Creating the dataset from 0

And much more....