Fondamenti di Robotica

Group: Christian Sassi, Luca Pedercini, Pietro Bologna

The goal of this project is to use a UR5 robotic manipulator to pick some lego blocks on a table and move them in certain positions.
The blocks are recognized by a 3D stereo-camera, the ZED 2 by Stereolabs, and classified by using computer vision techniques like yolov5 and OpenCV. The robot used in simulation is the 5th version of the 6 degree of freedom arm by Universal Robots, with some development environments such as Gazebo and Rviz.

A. Assignments

There are four different assignments with increasing difficulty. In each of them, the 3D camera has to recognize the blocks on the table, in a set of 11 different classes of objects, retrieve the position and the orientation of them. They have to be taken by the robot arm and moved in a certain position (drop area).

Assignment 1
There is only one object in the initial stand, positioned with its base “naturally” in contact with the ground.

Assignment 2
Multiple objects on the initial stand, one for each class. The base is “naturally” in contact with the ground.

Assignment 3
Multiple objects on the initial stand that can be rotated and lying on one of their lateral sides. They can’t be one on each other.

Assignment 4
The objects on the initial stand are those needed to create a composite object like a castle.

B. Environment

The basic software used to allow the communication between the different parties of the project is the middleware ROS (Robot Operating System). Topics and services are used to share messages between the nodes, such as the robot, the vision node and the motion plan node.
The project is divided into two big areas:

- the **computer vision** part, to recognize the objects and retrieve their position, developed entirely in Python.
- the **motion planning** part, taking care of the kinematics and the movement of the robot, mainly developed in C++.

C. Computer Vision

The main components of the vision are:

- **main**
- **detection**
- **image processing**
- **object processing**
- **communication with the robot**

Main

In the main, the script subscribes to the ros topic where the ZED camera publishes the captured image and once done, it invokes a custom callback function.

Detection

In the detection phase, it's possible to choose between two types of detection: live detection and background detection. The main difference between these two is that the first one shows the processed image previously captured by the ZED camera.

Image processing

In the image processing phase, the yolo model is firstly loaded and then used to detect any objects inside the captured image. Additionally, a threshold is used in order to discard uncertain values. Once done, a dictionary of the detected objects is ready to be processed in the next phase.

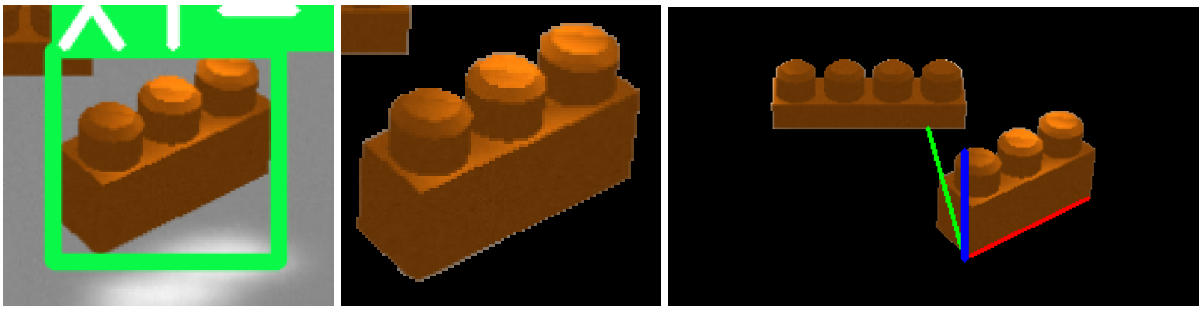
Object processing

During this phase, each object is processed using different algorithms and logics in order to understand its position and orientation. Firstly, the box image detected by yolo is filled of black except for the pixels representing the object. Even the pixels of other unwanted objects that can be inside the box are removed. This last step is done by calculating the contours of each object inside the box and then by keeping the larger one. Finally, all the black pixels are removed. Once done, all the 2D points are converted to their corresponding 3D points using the topic where the ZED camera publishes the 3D coordinates of the captured image. The characterization of the object consists of these parts:

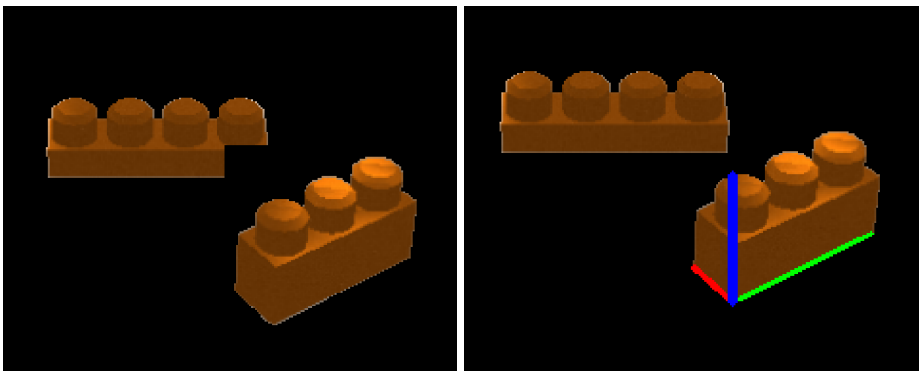
- **Gripper point**: To calculate a valid point for the gripper to pick the object, we first retrieve the maximum and minimum x and y 3D coordinates. We then use these coordinates to calculate a box composed of four points and determine the center of the resulting rectangle. Finally, we calculate the maximum z-coordinate. The desired point will be determined by finding the real 3D point that is most similar to the calculated one. However, we cannot use the calculated point directly since it is derived and not retrieved from the list of real 3D points.
- **Bottom side**: To calculate the bottom side of the object, which refers to the part of the body that rests on the table, we firstly extract all the pixels from the image and organize them into rows. We check which from the last, second last and third last rows is the bigger one and associate it to the bottom side of our object. This is because objects may have dummy pixels around them, even in the bottom part and, by doing this, we aim to discard such pixels and obtain a more accurate representation of the bottom side.

- **Left and right side points**: The method used to calculate the left side is the same as the one used for the right side. The approach involves calculating all the points that have an x-coordinate lower than the leftmost point of the bottom side. If no points are found, we assume that the left point is equal to the leftmost point of the bottom side. If any points are found, we calculate the first *n* points that have the most similar z-coordinate to the reference point of the bottom line, because these two points should be at the same level. To determine the most similar coordinate, we increase the z-coordinate of the leftmost point of the bottom side by a factor of 0.01., in order to compensate for potential inaccuracies in the ZED camera's measurements, ensuring a closer match to the desired point. Once we find at least one point, we calculate the mean of the pixel distance between each point and the leftmost point of the bottom. The calculated mean is used to keep all the first *n* points that are below it and discard the others. Finally, we determine which point has the maximum x-coordinate, as it will be farthest from the bottom side. If the retrieved pixel has a 2D x-coordinate similar to that of the leftmost point of the bottom side, we align the point with the reference point of the bottom line.
- **Orientation**: The method used to calculate the orientation of the object involves different techniques:
    - **Z-axis**: We take the left side point and the leftmost point of the bottom side as reference. Once these points are detected, the angle is calculated by taking the arctangent of the difference between the y-coordinates of the left side point and the leftmost bottom side point, divided by the difference between their x-coordinates. If the two points have the same x-coordinate, we skip the previous calculation and set the angle to 0. Additionally, if the left side of the object, compared to the right one, corresponds to the shorter side or, alternatively, if the bottom side is equal to the shorter side, we add 90 degrees to the calculated angle. This adjustment is made because we consider a rotation of 0 degrees when the left side of the object aligns with the longer side.

In this scenario, it is evident that the top left corner is being interfered with by an unwanted object. This interference has the potential to cause errors during the characterization of the current block.



To address this issue, as mentioned earlier, we calculate the contours of all the colored figures within the box and then keep only the largest one. By doing so, we can observe in the image that the small interfering piece of the object is replaced with black pixels, effectively deleting any interference during the object processing phase.



Above all, thanks to this correction and the previously described methods, we were able to approximate the angle of the highlighted object to be 32 degrees, compared to the actual angle of 34 degrees.

Communication with the robot

Finally, the detected and processed objects are published over a custom topic in order to be received by the robot that will execute the tasks. To facilitate this, we use a custom message format that encloses all the objects including the position, orientation, and object classification into a single message.

D. Robot Motion

The main components of the robot motion are:

- **kinematics**, in particular direct kinematics and inverse kinematics
- **p2pMotionPlan**, to compute a trajectory from a point to another point in the operational space.
- **threep2p**, to compute an entire movement of the robot, performing three p2p motions.
- **check functions**, to perform some checks about the computed positions, in order to avoid collisions.
- **main**

Kinematics

The kinematics describes the motion of points and bodies (objects).
The **direct kinematics** takes as input the set of the 6 values of the robot joints, computes the transformation matrix of each joint and link, multiplies the matrices and returns the transformation matrix (composed by position and rotation matrix) of the end effector.
The **inverse kinematics** takes as input a 3-dimension vector representing the final position of the end effector, and calculates eight possible combinations of values for the six joints.

p2pMotionPlan

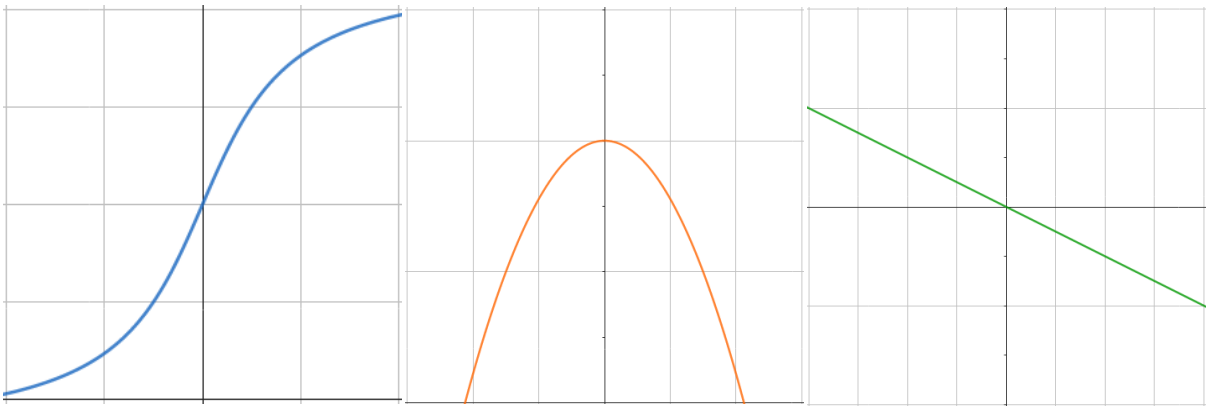
The p2pMotionPlan function computes a path of points between a starting point and a final point.
The motion is assigned in the operational space, using kinematic inversion to find the joint space configurations.

The function takes as input the initial configuration of the joints (the current values of the robot joints), and the 3D position to reach.
Initially, the function computes 8 possible sets of joints for the final pose, using the inverse kinematics, that are sorted (with a “bestInverse” function) using the euclidean distance between the initial and the final values of each joint.

Later on, for each joint, it is build a sequence of joint configurations from **qEs** to **qEf,** synthesizing a function q(t) such that:

- $q(t_s) = qEs$
- $q(t_f) = qEf$
- $v(t_s) = q'(t_s) = 0$
- $v(t_f) = q'(t_f) = 0$

This is made using a cubic polynomial $q(t) = a_3 t_3 + a_2 t_2 + a_1 t + a_0$
imposing constraints on velocity and on position building a system of linear equations that produces as a solution the coefficients of the cubic polynomial.
Each configuration computed is checked using a “check_angles” function, to avoid collisions and not allowed values.



position *velocity* *acceleration*

threep2p

“threep2p” is the function used to build an entire movement from a starting point to a final point. It joins three “p2pMotionPlan” functions, in order to reach two different intermediate points along his path to the final position.
For example, to perform a movement from a starting point xEs = (0.4, 0.2, -0.7) to a final point xEf = (-0.4, 0.3, -0.7) the following moves are performed:

- p2pMotionPlan from xEs to xE1 = (0.4, 0.2, -0.5), a slightly higher point than xEs
- p2pMotionPlan from xE1 to xE2 = (-0.4, 0.3, -0.5)
- p2pMotionPlan from xE2 to xEf

The result is a fluid movement from xEs to xEf, passing through the two intermediate positions, almost stopping in such points.

Rotate

“Rotate” is a function that allows to perform rotations to the objects, taking as input a rotation index, to understand the euler angles of the desired pose.
Rotations are divided into elementar rotations, in which it is needed only one step, and composite rotations, in which are needed two steps, performed using a sort of recursive function.

Check functions

There are three main function that performs some checks:

- **check_point,** checks if the inserted position is reachable by the robot, by comparing the result of the inverse kinematics and the result of the direct kinematics applied to the joints returned by the inverse.
- **check_angles,** checks if the computed joint by the inverse kinematics and the p2pMotionPlan are reachable by the ur5 robot. The joint values are checked to be within a range of allowed values.
- **check_collision,** checks if the final pose of every single link of the robot is over some space limits, such as the table height and the backward wall.

Main

In the main function of the motionPlan node:

1. A starting procedure is performed, to take the robot in a well known position.
2. The node enters a while loop in which:
    1. Waits for the position to reach, sent by the vision node.
    2. Checks if the inserted position is reachable by the robot.
    3. Checks if a rotation is needed.
    4. Compute the movement by managing the opening and closing of the gripper
    5. Publishes the computed joint configurations in a topic.

E. KPI Table

KPI 1-1: Time to detect the position of the object	**1.47 sec**
KPI 1-2: Time to move the object between its initial and final positions, counting from the instant in which both of them have been identified	**16 sec**
KPI 2-1: Total time to move all the objects from their initial to their final positions.	**99 sec**

F. Results Achieved

All the application has been developed only in the simulation environment, using the Gazebo simulator.
The first and the second assignment have been achieved with good success.
The third assignment has been completed successfully in the matter of the robot motion, with the rotations that are performed by the ur5 manipulator robot, while the Zed camera is not able to sufficiently recognize the objects when they are rotated on the table.

The main difficulties encountered are:

- Understand the objects' orientation with the zed camera, due to perspective and rotations.
- Manage the movement of the robot avoiding singularities and obstacles during his trajectories and rotations.
- Difficulties given by the problems related to gripping the objects in the gazebo simulator.