ρbot
Introduction to Robotics final project

University of Trento, Fall 2022

Alberto Zaupa
alberto.zaupa@studenti.unitn.it
Computer Engineering student

Alessandro Mizzaro
alessandro.mizzaro@studenti.unitn.it
Computer Science student

Francesco Maglie
francesco.maglie@studenti.unitn.it
Computer Engineering student

***Abstract*—This robotics project involves the use of a UR5 manipulator to move toy bricks. The UR5 is a 6-degree-of-freedom robotic arm capable of performing complex movements with precision and accuracy.**

The project involves the use of computer vision to identify and locate the toy bricks, and then the UR5 manipulator is programmed to pick up the bricks using a gripper tool and move them to a specified location. The project consists of the development of the control software and the computer vision algorithms.

The project aims to demonstrate the capabilities of the UR5 in a real-world scenario, specifically in a task that requires dexterity and control.

I. Introduction

A. Tasks description

The project is subdivided into 4 different tasks, which are briefly described here:

1) there is just one brick on the stand. The neural network has to identify it (its class and its position), then the robot has to pick it up and move it to a specific spot,
2) the same as task one, but there are more blocks on the stand,
3) the same as task two, but objects of the same class have to be stacked up to form a tower,
4) the objects on the initial stand are those needed to create a composite object with a known design

B. Software layout

The kinematics software is implemented as a Ros Node, while the computer vision algorithm runs as a Ros on-demand service, which is called by the kinematics node at the beginning of each task.

II. Computer vision

A. Introduction

This section presents an approach for object detection using a combination of YOLOv7, PCA analysis, SIFT filter with Torch and OpenCV.

The proposed method begins with object detection and classification using YOLOv7. Then, PCA analysis is applied to reduce the dimensionality of the image features, followed by a SIFT filter to extract salient features of the objects.

B. Yolo v7

Yolo (You-Only-Look-Once) [1] [2] is an object detection algorithm based on CNN (Convolutional Neural Network).

The algorithm extracts features from the input image, and then applies a grid system to divide the image into smaller regions. YOLOv7 predicts the presence of an object and its class for each region.

One of the advantages of YOLOv7 is its fast processing time, making it suitable for real-time applications such as robotics.

1) Dataset: To obtain a dataset for YOLOv7, we took a series of photographs of objects and scenes that we wanted to detect in the *Iot and robotics Lab*. We then used a labeling tool called Roboflow to tag the objects in each image with their corresponding class labels Using Colab, we trained YOLOv7 on our dataset, adjusting the hyperparameters to optimize its performance

2) Object Detection CNN: The first neural network is responsible for classifying objects in an image into different categories such as *X1-Y2-Z1, X1-Y4-Z2*.... This is a standard classification task that many object detection algorithms use.

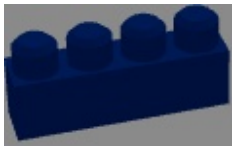

Fig. 1. Yolov7 Object Detection X1-Y4-Z2

3) Up or Down CNN: The second network is designed to detect whether an object is on the ground or in a "natural" position. This network uses same architecture (Yolo) and is trained separately from the object classification network. The reason for having this second network is that in task 3 and task 4, it's important to distinguish between objects that are lying on the ground versus objects that are standing up in their normal position.

C. OpenCV preprocessing

We used OpenCV [3] to preprocess the input image from Yolov7 bounding boxes by applying an HSV mask to remove

shadows and unwanted objects from the scene.

The HSV mask is a filter that extracts pixels in a specific range of hue, saturation, and brightness values. By carefully choosing the values for the mask, we can isolate the objects of interest in the image while removing unwanted background elements such as shadows, reflections, or other objects that may interfere with the object detection algorithm.

We develop a tool for choosing the best HSV mask for each object class.

After the preprocessing step, we used the findContours function to detect the contours of the objects in the image.

Fig. 2. Mask created by HSV values

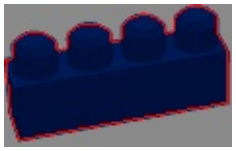

Fig. 3. Contours matching

D. PCA

PCA (Principal-Component-Analysis) [4] allows us to find the direction along which our data varies the most. The function returns two eigenvectors which are the principal components of the data set.

We used the PCA analysis with contours points set for finding the yaw rotation of the objects

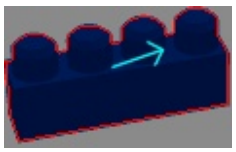

Fig. 4. PCA analysis computation

E. SIFT

The SIFT (Scale-Invariant Feature Transform) [5] is a widely used computer vision algorithm that allows for the detection and description of local features in images. We used SIFT filter for object skeletonization matching pin of megabloks and calculating the position of the object

F. Result

The results of Detection service include: 1) Center of object; 2) Center of Pins; 3) Yaw rotation of blocks;

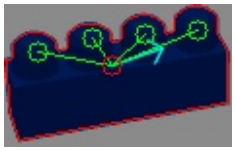

Fig. 5. Skeletonization by sift

```
Class:  8
Ground:  False
Center:
        X:  -0.16364871660619984
        Y:  -0.2076784536242485
        Z:  -0.14632263857871297
Piolini:
        X:  -0.17899328539595016
        Y:  -0.20371715128421783
        Z:  -0.15330091534182433
Rotation:
        Yaw:  0.3064527679965148
```

Fig. 6. Detection Service response. Coordinates refer to Robot Frame

III. KINEMATICS

A. Problem statement

The first important problem to address is knowing where the robot is and, as consequence, how its position can be retrieved. To resolve this, we initially have to identify the number of variables needed to describe our system; in our case, this value is six because we are operating with an *UR5* robot, a 6-degree-of-freedom manipulator. Then we can define the set of equations that let us model the robot motion as a function of its joints (**direct kinematics**) within the so-called *C-space*, a n-dimensional space containing all possible configurations of the robot.

Given a position and an orientation in space, the following step is to be able to determine which joint configuration lets the robot reach that desired position; to solve this, we implemented the **inverse kinematics** of the robot. The solution to such problem is a multi-variables differential equation, that we were able to solve using a numerical approach. Moreover, we needed a solution to control and plan the robot's movement within the workspace. To deal with this task, in this work we came up with two different possible implementations, as described in sections III-C and III-D.

B. Inverse kinematics: the Levenberg-Marquardt method

Inverse kinematics aim is to determine the joint configuration given some end-effector pose. To solve such differential equations, there are two approaches: analytical and numerical. Analytical formulas in some cases may not exist and generally cannot be optimized to take into account other important constraints, such as joint limits. For this reason, we adopted a numerical approach, which is iterative and can be easily combined with key constraints.

The objective is the computation of the following position, knowing the destination and the current configuration of the robot. To do so, we need to specify an **error**, which is a vector

representing the translation and the rotation of the end-effector from the current pose to the desired one.

$$\boldsymbol{e} = \begin{pmatrix} \tau(\boldsymbol{T_{e*}}) - \tau(\boldsymbol{T_e}) \\ \alpha\left(\rho(\boldsymbol{T_{e*}})\rho(\boldsymbol{T_e})^T\right) \end{pmatrix} \in \mathbb{R}^6$$

where $\boldsymbol{T}$ represents the forward kinematics (*SE(3)-space homogeneous matrix*) of current (e) and desired ($e*$) pose of the end effector, the top three rows correspond to the translational error and the bottom three to the rotational one. $\alpha(\cdot)$ is a conversion from an orientation matrix to the equivalent angle-axis vector.

A possible implementation to solve the inverse kinematics problem is the *Newton-Raphson* method, which strives to minimize an error function E to find the joint configuration that enables the pose expressed by $\boldsymbol{T_{e*}}$. The error function E is the following

$$E = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{W_e}\boldsymbol{e} = \frac{1}{2}\|\boldsymbol{e}\|^2$$

since we set $\boldsymbol{W_e}$ to be the identity matrix.

To improve the solvability, due to the frequent failure of this method to converge, *Gauss-Netwon* approach would be preferable. However, when the jacobian matrix ($\boldsymbol{J}$) is singular, which means that the determinant of such matrix is 0, both methods have to face up with that problem. To overcome this, we need to adopt another technique, which is the *Levenberg-Marquardt* method:

$$\begin{gathered} \boldsymbol{q}_{k+1} = \boldsymbol{q}_k + (\boldsymbol{A}_k)^{-1}\boldsymbol{g}_k \\ \boldsymbol{A}_k = \boldsymbol{J}_k^T\boldsymbol{W}_e\boldsymbol{J}_k + \boldsymbol{W}_n \\ \boldsymbol{g}_k = \boldsymbol{J}_k^T\boldsymbol{W}_e\boldsymbol{e}_k \end{gathered}$$

where $\boldsymbol{W}_n = \text{ diag}(w_n)(w_n \in \mathbb{R}^n_{>0})$, which is needed to ensure that $\boldsymbol{W}_n$ is non-singular and positive definite. In order to define $\boldsymbol{W}_n$, we chose the approach proposed in [8], where it is suggested

$$\boldsymbol{W}_n = \lambda E_k \boldsymbol{1}_n$$

It's important to point out that this method, as the one described above, is subject to local minima and may fail to converge.

C. Redundant Control

An effective motion plan not only strives to reach the set destination, but it also tries to do that without violating a given set of constraints. Often one of such constraints is to move the robot through a set of configurations not too close to the limits of the joints' range of movement.

When a robot is redundant, it is often possible to exploit such characteristic to achieve the given constraints, because the dimensionalities of the task space and of the configuration space are such that for most points in the task space there exists an infinite set of solutions to the related inverse kinematics problem.

Of course, though such redundancy-based strategies could not be directly applied to our setting, as an UR5 robot is not redundant in general.

1) Planar movement: While experimenting with different strategies for planning the movement of the UR5, we found that moving the robotic arm on a plane and then descending to pick up or put down the bricks is a simple, yet effective one.

Moving the robot in such a way ensures that it will not hit the bricks when approaching them, because when picking them up the end-effector simply travels downwards perpendicularly with respect to the surface of the table. Also, because such strategy breaks up the whole trajectory into two simpler movements, we found that the robot is more likely to avoid undesirable joint configurations.

Finally though, the main advantage of this approach is that we can perform the first portion of the trajectory without taking into consideration the rotation of the end effector, allowing us to exploit the introduced redundancy in the control loop.

2) Exploiting redundancy: The main idea of a redundant controller is that of biasing the solution of the differential inverse kinematics problem towards a joint configuration $\mathbf{q}_0$ that we know does not violate the given constraints, which in our case were to keep the joint rotations close to the middle of their range of movement.

In order to do that we defined a secondary objective function:

$$\hat{L}(\mathbf{q}) = -\frac{1}{2N}\|(\mathbf{q} - \bar{\mathbf{q}}) \oslash (\mathbf{q}_M - \mathbf{q}_m)\|^2$$

Where $\oslash$ is the element wise division operator, $\bar{\mathbf{q}}$ is the vector of median values of the joint ranges, $\mathbf{q}_M$ is the vector of maximum values, $\mathbf{q}_m$ is the vector of minimum values, and N is the number of joints.

We then differentiate such function to get $\frac{\partial \hat{L}}{\partial q}$, a vector which indicates the direction we should move towards in order to achieve a joint configuration where each joint coordinate is close to its median value.

$$\frac{\partial \hat{L}}{\partial q} = \dot{q}_0 = -\frac{1}{N}(\mathbf{q} - \bar{\mathbf{q}}) \oslash (\mathbf{q}_M - \mathbf{q}_m)^2$$

Finally, we use such result in a control loop, which updates the joints velocities in the following way:

$$\dot{q}_{k+1} = \mathbf{J}^\dagger(\mathbf{K}\mathbf{e} + \mathbf{v}_d) + (\mathbf{I}_6 - \mathbf{J}^\dagger\mathbf{J})\dot{q}_0$$

$\mathbf{J} \in R^{3\times 6}$ is not actually the full manipulator jacobian, but only its translational component, while $\mathbf{J}^\dagger$ is its pseudoinverse. $\mathbf{K}$ is a damping diagonal matrix, $\mathbf{I}_6$ is the 6×6 identity matrix, e is the vector difference between the desired position of the end effector and its current position, while v_d is the desired translational velocity.

D. Vector field controller

Some areas of the workspace of an UR5, which are generally referenced to as *singularities*, should be avoided because they lead to the robot being unable to move effectively.

The most relevant of such areas, given the task at hand, is the one close to the axis centered on the robot's base. When the robot is moving close to its base frame, even small

movements of the end effector correspond to large joints rotations, and therefore the robotic arm might unexpectedly perform dangerous movements.

In order to deal with such problem, we simulated a vector field centered on the z axis of the base frame. The vector field acts as a vortex that pushes the motion of the robot away from the singularity.

1) Control loop: The main control loop of the inverse kinematics solver is set up as follows:

$$\mathbf{v}_d = \lambda \mathbf{e}(\mathbf{q}),$$
$$\dot{q} = (\mathbf{J}^T\mathbf{J} + \mathbf{W}_d)^{-1}\mathbf{J}^T\hat{v}(\mathbf{r}, \mathbf{v}_d)$$

$\mathbf{e}(\mathbf{q})$ is an error vector, defined as in our implementation of the *Levenberg-Marquardt* method, and it is used to compute $\mathbf{v}_d$, the desired translational and rotational velocity to be applied to the end effector.

$\mathbf{J}$ is the robot's jacobian, $\mathbf{W}$ is a damping matrix, and $\mathbf{r}$ is the vector of (x, y) coordinates that represents the position of the end effector with respect to the robot's frame.

$\hat{v}(\mathbf{r})$ is the actual rotational and translational velocity used to compute $\dot{q}$. $\hat{v}(\mathbf{r})$ is obtained by applying to $\mathbf{v}_d$ a simulated vector field $\overrightarrow{\mathbf{V}}$.

2) Field equations: When the end effector is moving towards the z axis of the robot frame, the simulated vector field $\overrightarrow{\mathbf{V}}$ influences its translational velocity $\mathbf{v}_\tau = \tau(\mathbf{v}_d)$ along the (x, y) coordinates.

$$\overrightarrow{\mathbf{V}}(\mathbf{v}_\tau, \mathbf{r}) = \begin{cases} \mathbf{0}_3, & \text{if } \|\mathbf{r}\| \geq r_0 \text{ or } \mathbf{v}_\tau \times \mathbf{r}_\perp < 0 \\ \frac{\mathbf{r}_\perp}{\|\mathbf{r}_\perp\|}, & \text{otherwise} \end{cases}$$

Where r_0 is some constant representing the maximum range of the vector field and $\mathbf{r}_\perp$ is a vector perpendicular to $\mathbf{r}$.

Let $\overrightarrow{\mathbf{v}_\tau} = \frac{\mathbf{v}_\tau}{\|\mathbf{v}_\tau\|}$. Then $\hat{v} = (\hat{v}_\tau, \hat{v}_\rho) \in R^6$, the actual velocity used in the inverse kinematics controller, is:

$$\hat{v}_\tau = \|\mathbf{v}_\tau\| \frac{\overrightarrow{\mathbf{v}_\tau} + \overrightarrow{\mathbf{V}}(\mathbf{v}_\tau, \mathbf{r})}{\|\overrightarrow{\mathbf{v}_\tau} + \overrightarrow{\mathbf{V}}(\mathbf{v}_\tau, \mathbf{r})\|}$$

$$\hat{v}_\rho = \rho(\mathbf{v}_d)$$

3) Visualization: We provide a simple visualization that exemplifies the way in which the vector field influences the motion of the end effector.

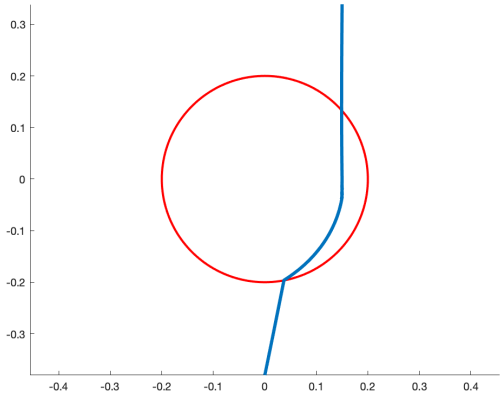

Fig. 7. The trajectory of what would be the end effector is depicted in blue, while the red circle represents the range of the vector field. A straight-line trajectory to reach the desired destination would come very close to the singularity, at the center of the red circle. Instead, the vector field smoothly deviates the trajectory, making sure that the movement of the robot is safe.

E. The general workflow

The general workflow is as follow:

1) the computer vision service locates all blocks and sends all the information about position and orientation to the kinematics Ros node.
2) the manipulator reaches the *home position* (defined ahead of time).
3) for each brick:
   a) a planar motion (with redundant or potential field controller) towards a position above the target brick is performed.
   b) the robot descends to pick up the brick.
   c) the robot grasps (or releases) the brick.
   d) the robot travels towards the desired destination, which is given by the current task.
   e) repeat the same process to reach the next pose.

REFERENCES

[1] Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*
[2] WongKinYiu, *GithubRepo*
[3] OpenCV Lib https://opencv.org/
[4] PCA analysis docs
[5] SIFT Filter docs
[6] Kevin M. Lynch, Frank C. Park *Modern Robotics: Mechanics, Planning, and Control*, Cambridge University Press, 2017, website
[7] Jesse Haviland, Peter Corke, *Manipulator Differential Kinematics Part I: Kinematics, Velocity, and Applications*, arXiv, 2022, website
[8] S. K. Chan, P. D. Lawrence, *General inverse kinematics with the error damped pseudoinverse*, in *Proceedings*. 1988 IEEE international conference on robotics and automation, IEEE, 1988, pp. 834–839.