UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria della Automazione e Robotica

Report Control LAb project

The tic-tac-toe player CORRRADO

Anno Accademico 2023/2024

Relatore
Ch.mo prof. Raffaele Iervolino

Candidato
Andrea Morghen, **Maria Vittoria Cinquegrani**, **Salvatore Del Peschio**, **Salvatore Oliviero**, **Valentina Giannotti**

GitHub link: https://github.com/saviodp7/Corrado_CL_ros.git

Abstract

This paper deals with the trial of making an anthropomorphic robot with a spherical wrist play tic-tac-toe. A control algorithm has been implemented that allows the manipulator to follow a certain trajectory, which makes it draw an x. An artificial intelligence algorithm, a combination of CNN and computer vision recognizes the current configuration from the camera, furthermore, a min-max algorithm allows us to compute optimal game sequences. All of this is implemented in a ROS environment simultaneously.

Contents

Introduction

The project followed a V-cycle development policy to maintain direct contact between design and physical implementation for the entire project duration. The main development phases are as follows:

1. Create a model of the system (robot) in a development environment (Simulink) and test it (MIL).

2. Exported the model (Simulink), and carried out simulations with control strategies in a simulated environment (MATLAB) (SIL).

3. Create and assemble the system model physically; real simulations were carried out (Arduino), revealing design errors (HIL).

4. Steps 1-3 were repeated until a satisfactory real simulation was achieved.

First, an implementation of the single processes needed for the operation of the robot was carried out, one by one, using mainly Matlab, Simulink, Python, and Arduino hardware. This implementation was limited by delays and hardware capabilities, therefore, to get better

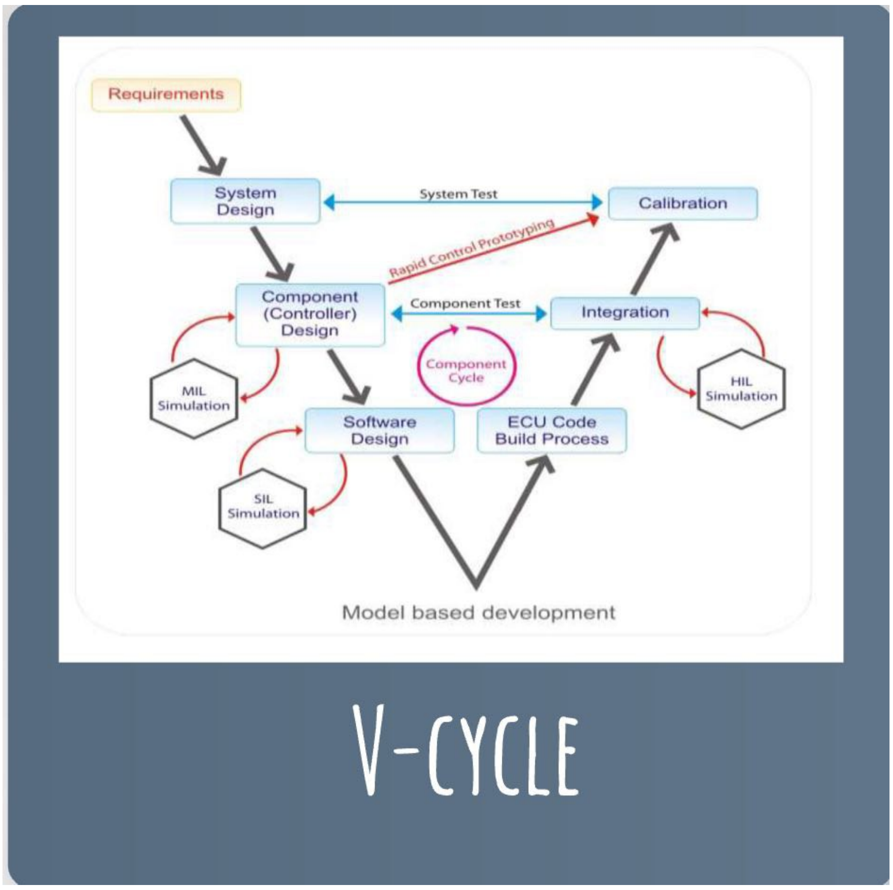


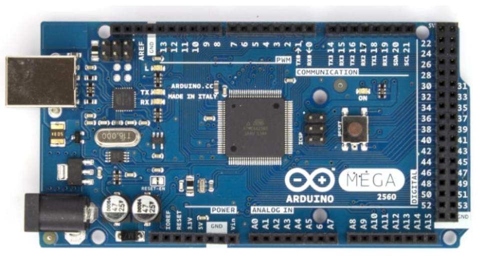
Figure 1: Project strategy

results, we decided to switch from Arduino to Raspberry. All the computations was done on a Ubuntu system, in particular on ROS (Robot Operating System).

This middleware makes it easier to manage the robot's hardware, communicate between the different system components, and simplify the software development and debugging process.

By using ROS, it was possible to take advantage of several libraries, tools, and conventions that facilitated the development of complex robotic applications.

The report is structured in such a way that it takes into account the steps taken during the course of working months, with the last chapters focusing more on the final implementation of the project.


(a) Arduino Mega

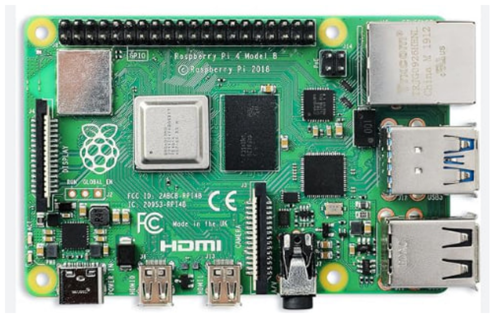

(b) Raspberry Pi 4

Figure 2: Hardware used

Chapter 1

Model

The model is the development process center, from requirements development, through design implementation and testing. The source file of our model has been found online, the links of the robotic arm were exported from the internet and later suitably modified in the CAD environment to accommodate and optimize the project's needs. Afterward, the robot was 3D printed and assembled, and the servo motors were incorporated.

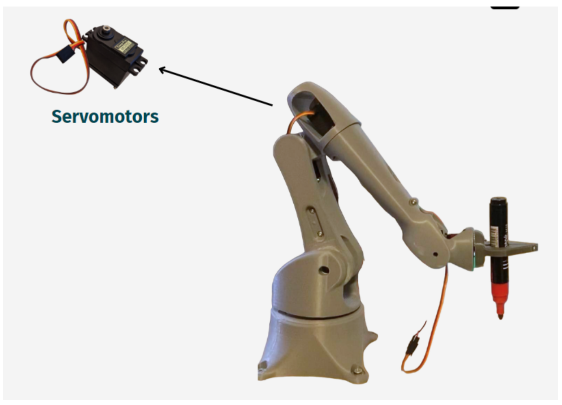


Figure 1.1: Real System

Once we had the real system, we brought it into Simscape which extends Simulink with multi-domain physical system libraries, and it provides a set of fundamental tools for modeling and simulating the robot's behavior in its environment. Hence the CAD models of the links were imported as solid blocks into Simscape and properly linked together using joints, according to the robot design.

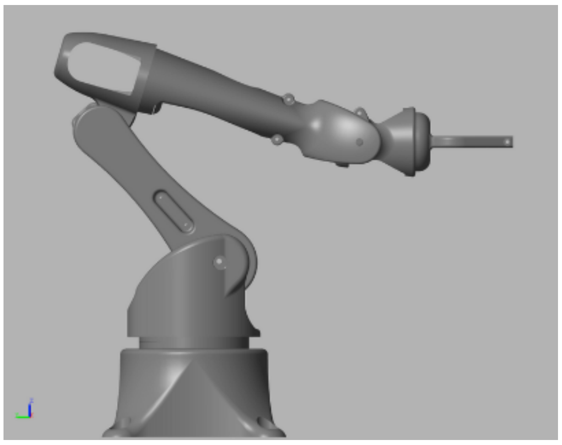


Figure 1.2: Simscape System

As servomotors are employed as joints actuators, in Simulink, the standard servomotor block has been incorporated with other components to model the input signal applied to the joint.

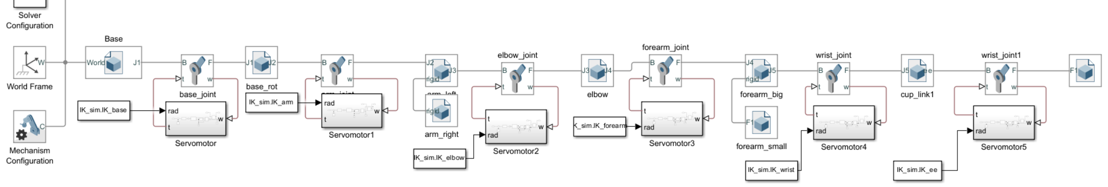


Figure 1.3: Simscape tree

1.1 Model implementation in Simscape

The revised Simulink model of the servomotors takes as input the desired angle in radians and the current angular velocity of the joint, giving as output the actuation torque of the joint.

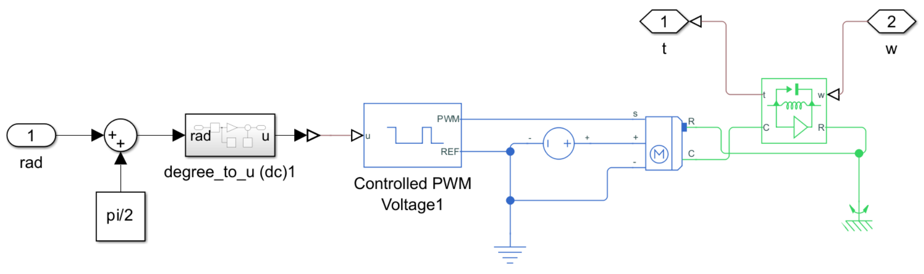


Figure 1.4: Simscape servomotor

Therefore, the servomotor block has inside it:

- Sum with an angle always in radians to compensate for possible misalignment between the expected frame and the actual frame of the coupling

- Conversion block from radians to duty cycle to give the input to the PWM, this is performed with simple calculations related to the type of PWM and servomotor used

- Standard PWM block, that takes the duty cycle signal and converts it into a reference signal for the standard Simulink servomotor block, both of them have been set according to the specifications of the real servomotor

1.2 Model implementation in Ros

In ROS, the robot's structure and kinematics are described through the URDF file (Unified Robot Description Format), providing a detailed representation in terms of links, joints, sensors, and other relevant properties. Actually, the robot's model is implemented in a large number of files (urdf, xacro, trans, stl...) which together contain all the main robot's characteristics as:

- Shape and inertia of links
- Type of links
- Robot chain (ordered links)
- Transmissions between motors and links
- Main frames and collisions

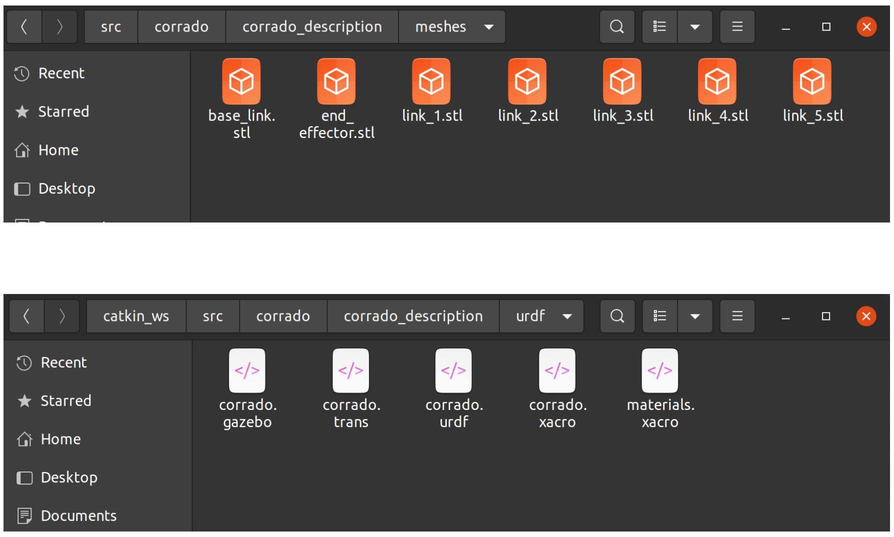


Figure 1.5: Ros file

Chapter 2

Inverse kinematic

Inverse kinematics is a pivotal aspect of robotic control. It handles determining the joint configurations, relevant for motion control, that let the end-effector achieve a desired position and orientation, described by the trajectory.

The following section of this chapter will address the inverse kinematics analysis of a 6-degrees-of-freedom (6-DOF) robot manipulator. The study employs MATLAB to model and analyze the inverse kinematics of the robotic system. The solution was also implemented using the ROS language.

2.1 Inverse kinematic Algorithms

There are three main algorithms for the inverse kinematic, each of them presents advantages and disadvantages. First of all, it is important to remember that the analytical link between joint space and operational space is made by the Jacobian matrix that relates joint velocity and end-effector velocity. Then the integrator makes the transition from velocity to position. The Jacobian is the main component of kinematic inversion CLICK algorithms [3]; three different types of CLICK algorithms are thus shown:

- Jacobian Pseudo-inverse: it works as a feedback linearization control to keep the end-effector position error to zero. Inversion, however, is not always possible and there may be problems of computational complexity.

$$\dot{q} = J^{\dagger}\left(\dot{x}_d + Ke\right) \tag{2.1}$$

- Jacobian Transpose: solves computational complexity and singularity issues but stability analysis is linked to the proportional gain value of the error, which cannot exceed a certain threshold if we want to achieve stability. This leads to not achieving a small error at steady state

$$\dot{q} = J_A^T(q)Ke \tag{2.2}$$

- Second-order Algorithms: The joint accelerations and the relative error are also included int the control law, thus the dynamics of the system are also taken into account. With these algorithms, the system is asymptotically stable but it is more computationally complex.

$$\ddot{x}_e = J_A(q)\ddot{q} + \dot{J}_A(q,\dot{q})\dot{q} \tag{2.3}$$

$$\ddot{q} = J_A^{-1}(q)\left(\ddot{x}_d + k_D\dot{e} + K_P - \dot{J}_A(q,\dot{q})\dot{q}\right) \tag{2.4}$$

The problem of inverse kinematics can be solved more efficiently without using the previously mentioned algorithms, which are often complex. Dedicated tools are available, such as blocks in Simulink and packages in ROS, like MoveIt, which significantly simplify the process.

2.2 Simulink Inverse Kinematics

In Simulink, we have implemented inverse kinematics equations using MATLAB Function blocks and other Simulink blocks. This integration efficiently solves inverse equations in the Simulink model, expanding the available options for addressing the inverse kinematics problem.

2.3 ROS Inverse Kinematics

In ROS, we have configured a working environment using MoveIt. This platform streamlines motion planning and kinematics management, allowing users to interact through intuitive APIs, regardless of

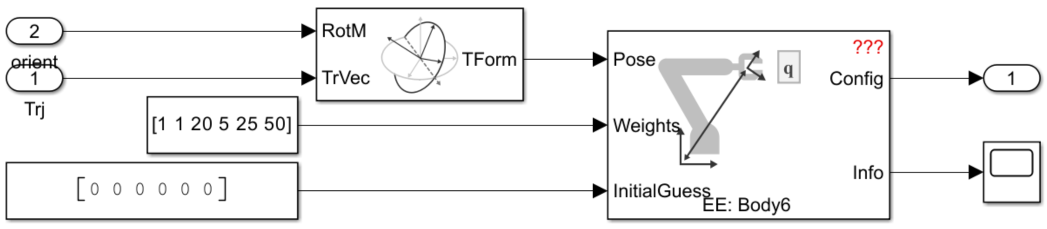
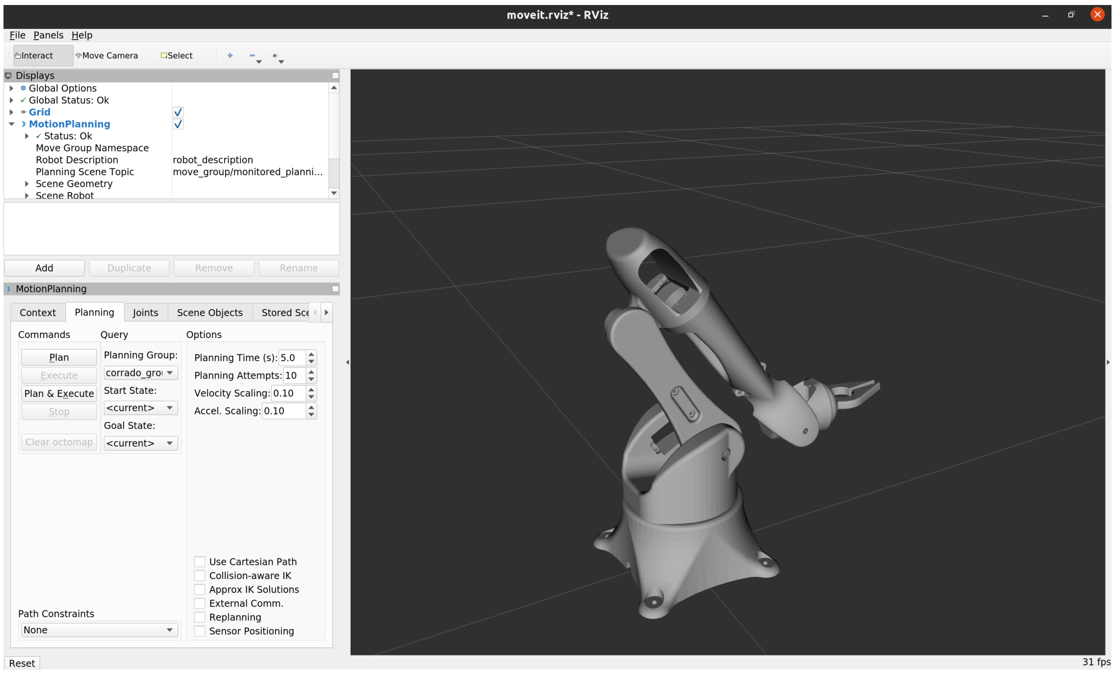


Figure 2.1: Inverse Kinematics Simulink

the specific algorithm used. The MoveIt libraries can be used to perform inverse kinematics calculations and robot motion planning, handling both analytical and numerical solutions using various kinematic solvers.



Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description robot_description
Planning Scene Topic move_group/monitored_planni...
Scene Geometry
Scene Robot

Planning Group: corrado_gro
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Path Constraints
None

Figure 2.2: Inverse Kinematics Ros

Chapter 3

Trajectory planner

As previously introduced, the implemented robotic manipulator is designed with the specific goal of playing a game of tic-tac-toe. To achieve this, it is essential to implement a trajectory planning algorithm, a process of determining a sequence of points the end-effector must follow over time that enables the robotic arm to draw an "X" at predetermined position on the game grid.

3.1 Matlab implementation

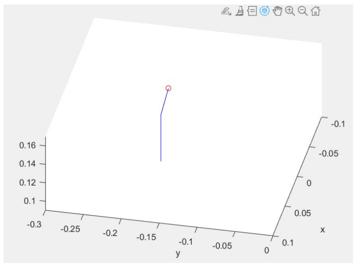
Trajectory planning was initially implemented in MATLAB. The employed approach defines the trajectory key points in the operational space, then they are interpolated together using segments. The curvilinear abscissa s(t) defines the law of motion and it is characterized by a trapezoidal profile to ensure smooth and controlled motion of the robot. Once the curvilinear abscissa was defined, the accelerations and cruising velocities were calculated as follows:

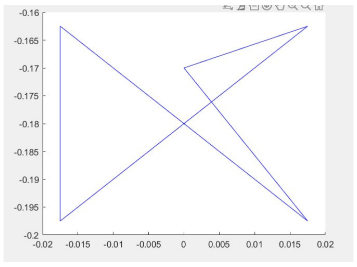
$$\dot{s}_c = \frac{1.5 \cdot |s_f - s_i|}{\delta_t}; \tag{3.1}$$

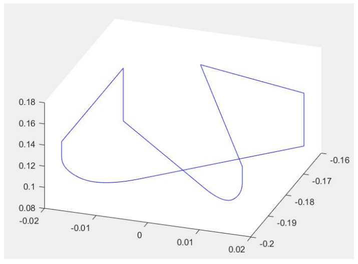
$$t_c = \frac{s_i - s_f + \dot{s}_c \cdot \delta_t}{\dot{s}_c}; \tag{3.2}$$

$$\ddot{s}_c = \frac{\dot{s}_c}{t_c}; \tag{3.3}$$

The implemented trajectories are three-dimensional to allow the pen to be lifted off the paper. The 'X' trajectory follows a closed path defined by ten path points, two of which are waypoints introduced to avoid any smudging. Similarly, the 'POINT' trajectory is made up of four waypoints.


(a) Point Trajectory


(b) X trajectory top view


(c) X trajectory

3.2 ROS implementation

The trajectory planning in Ros was structured via a class we created in python. The class allows us to define:

- Home position
- Grid centring
- Grid cell width
- X/O radius

The homing position is the position in space that the end-effector reaches when the system is switched on and subsequently at the end of each executed trajectory. The grid centre position together with the grid width makes it possible to calculate all nine cell centre positions in which to write. The trajectory to be executed is then calculated in the functions draw_x() (draw_circle() was created but will not be used in this project) and homing(). These trajectories are generated as sequences of points taking into account the center of the grid the x should be written and the radious desired, the idea of work is the same as previously developed in matlab.

Chapter 4

Computer Vision

To develop a robotic manipulator designed to play tic-tac-toe, a crucial aspect is its integration with computer vision. Computer vision, an artificial intelligence (AI) field, focuses on providing the robotic manipulator with the capability to visually perceive and interpret images and videos, akin to human visual processing. In the tic-tac-toe game context, the robotic manipulator uses a camera to capture images of the game grid and, based on this visual information, makes decisions about the moves to be performed during the game.

The following steps were followed for its implementation:

- A camera installed near the robotic manipulator captures images of the game grid. These images are transmitted to the computer vision algorithm.
- The computer vision algorithm processes the received images by recognizing the grid and letters to identify the current position

of the signs (X and O) on the grid.

- Based on the information obtained from image processing, an artificial intelligence algorithm analyzes the current game situation, identifying possible moves and evaluating the best strategy for the robotic player using a min-max algorithm.
- Once the game situation is understood, the robot system plans the next move. This involves using trajectory planning algorithms for the robotic manipulator to position the robot arm in the correct position to execute the 'X' move on the grid.
- The robotic manipulator physically executes the move on the grid by placing the mark (X or O) in the desired cell.

The process is performed each turn, allowing the computer vision system to monitor the progress of the game and make decisions based on the new visual information. The next sections will focus on the first three points, not yet covered in the previous chapters.

4.1 Algorithms for grid extraction and evaluation

For the recognition of the game id grid, a number of special functions were created. The main function is "find_game_grid", which takes as input a frame of the sheet image, a threshold value, and the kernel size for erosion. The process starts by extracting a region of interest (ROI) from the frame and converting it to greyscale. Next, a threshold is applied to convert the image to black and white, then to detect the contours and edges of the game grid the find_corners and sort_corners functions are used.

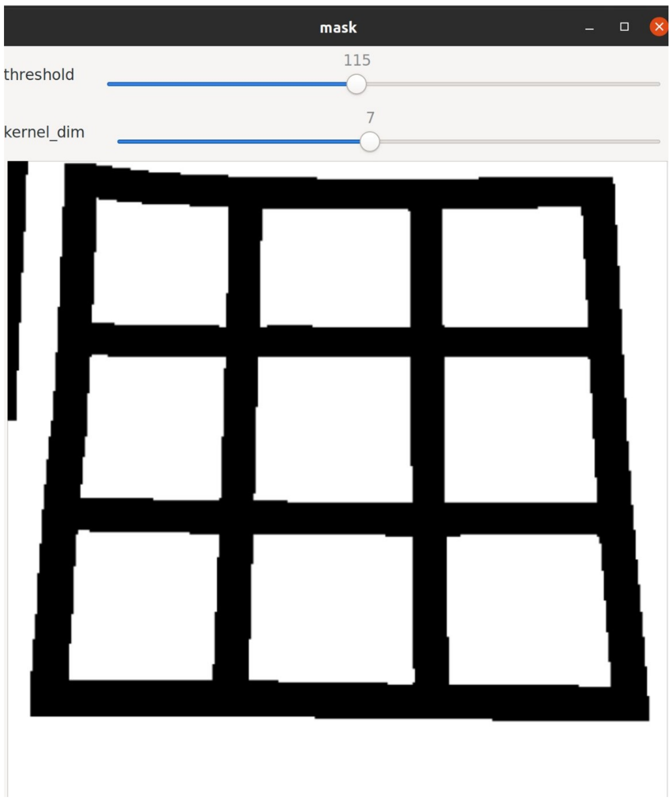


Figure 4.1: Black and white grid

The find_corners function extracts the corners from the found contours, using bounding rectangles and ignoring contours with children.

The sort_corners function organises the corners in a 3x3 grid by first sorting them by y-coordinate and then by x-coordinate (top left to bottom right).

Finally, other functions are included, such as find_color, which determines whether any colour beside white or black is present in a cell, which allows us to distinguish the cells on which to apply the neural network for X or O recognition from empty cells (to reduce the computational cost). Finally, print_board prints the current state of the grid, and debug_image displays the grid mask for debugging purposes.

Changing the threshold parameters can help achieve a more effective mask. By changing the threshold value, the distinction between light and dark pixels can be adjusted. Lowering the threshold makes dark pixels more visible, while increasing it improves the discrimination between grey tones and darker pixels. Increasing the kernel size influences the merging of neighbouring dark pixels. As lighting conditions can vary, it is necessary to experiment with different configurations to find the best combination of these parameters and optimise the recognition of the desired shapes.

4.2 Letter recognition

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed for image pattern recognition. The project's goal revolves around utilizing a CNN to classify handwritten letters, specifically focusing on the recognition of 'x' and 'o' letters. The initial steps involve data preparation, including the loading and concatenation of handwritten letter datasets from various CSV files [1](Comma-

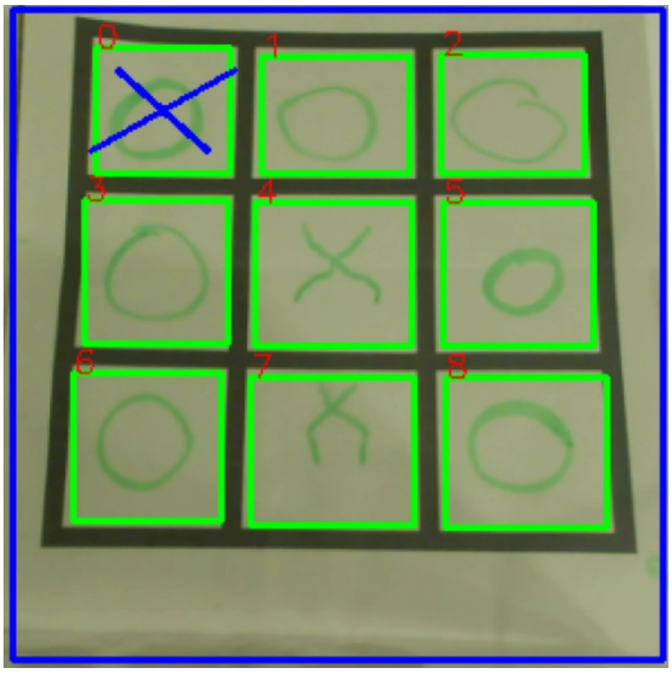


Figure 4.2: Printout of the detected grid

Separated Values file is a text file format used to represent tabular data). Only the letters of interest were extracted from this dataset. One-hot encoding is applied to represent the letters and background categories. The prepossessing phase encompasses reshaping, filtering, and normalizing the data.

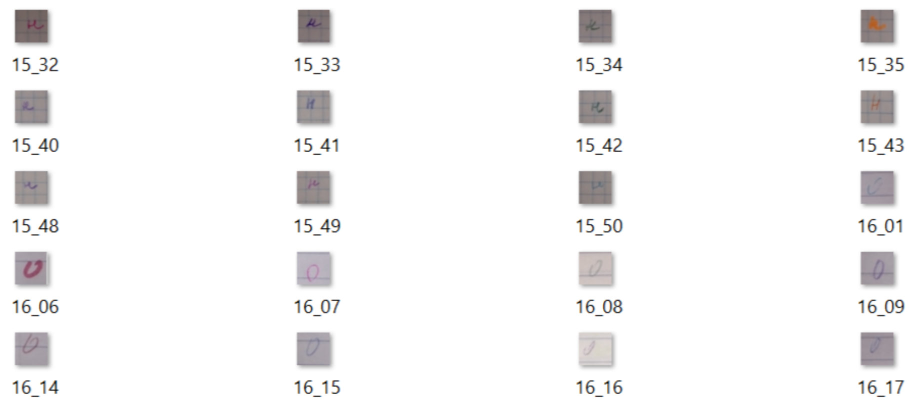


Figure 4.3: Examples of images from the dataset.

The training phase is described below :

- Data Preparation: Data is split into two sets: the training set and the validation set. The training set contains input examples (e.g., images of handwritten letters) correct labeled with the corresponding letters. The goal is to teach the model patterns in the training data to make accurate predictions.
- Model Construction: The model architecture is defined by specifying the number and type of neural layers, activation functions, input and output dimensions, etc. For a CNN recognizing letters, this may include convolutional layers, pooling layers, fully connected layers, and others.
- Model Compilation: The loss function to minimize during training, the optimizer to use, and the metrics to monitor (e.g., accuracy) are specified. The loss function takes into account the mismatch between model predictions and desired target values.
- Model Training: The model is trained iteratively on the training data. During each iteration (epoch), the model makes predictions on the training inputs and compares them with the correct labels. The optimizer adjusts the model weights to minimize the loss function. This process continues until the model effectively represents patterns in the training data.

After training, Testing (Evaluation) Phase is performed:

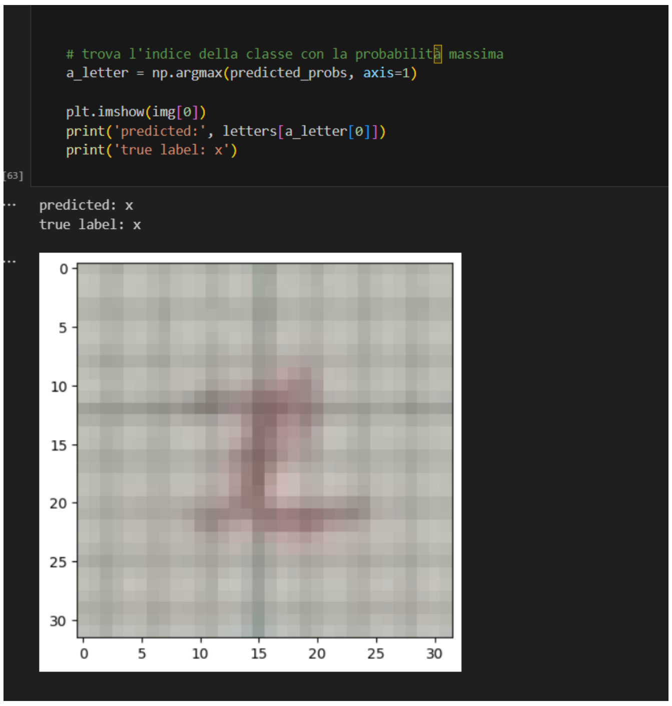
- Test Data Preparation: The test set is separated from the training and validation data and is used to evaluate the model's performance on unseen data.

- Model Evaluation: The model is evaluated on the test set using the metrics specified during compilation (e.g., accuracy). During evaluation, the model makes predictions on the test data and compares them with the correct labels.

- Results Analysis: Evaluation metrics are analyzed to understand the model's performance. This step is crucial to assess whether the model generalizes well to new, unseen data.

- Return to Training (if necessary): If the model exhibits unsatisfactory performance during testing, a return to the training phase may be necessary. Possible approaches include adding more training data, modifying the model architecture, or optimizing hyperparameters.

The training and testing cycle is iterated until the model achieves satisfactory performance on the test data. This process is essential to ensure that the model generalizes well to unseen data and avoids overfitting to the training data. The trained neural network and its weights can be saved in an HDF5 file. This allows the network to be used without having to perform the training and testing phase each time. In an external file, indeed, it is necessary to create a network model with the same characteristics and load the correct parameters from the HDF5 file.



```
# trova l'indice della classe con la probabilità massima
a_letter = np.argmax(predicted_probs, axis=1)

plt.imshow(img[0])
print('predicted:', letters[a_letter[0]])
print('true label: x')
```

```
predicted: x
true label: x
```

Figure 4.4: Test of the CNN

Chapter 5

AI algorithms

5.1 Minmax algorithm

The Minimax artificial intelligence algorithm was used to calculate the next move to make the robot win. Minimax is designed for two-player zero-sum games because of the mathematical interpretation of the game's outcome: one player wins (+1) and the other loses (-1), or neither player wins (0). The algorithm takes into account the current game state and the available moves, recording each valid move and switching between player Min and player Max until reaching a terminal state (win, draw, or loss). The algorithm aims to maximise the positive outcome for player Max and minimise it for player Min at each level of the game tree. The game's outcome modelled by the heuristic function, an AI concept that we will not explore in this paper. This process continues until a final outcome is determined.

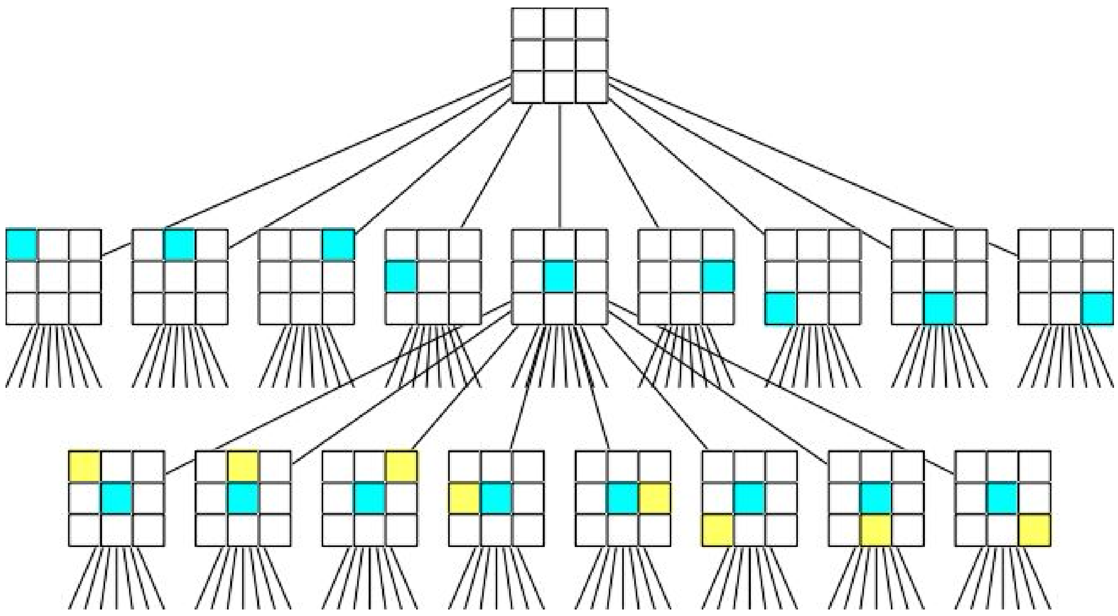


Figure 5.1: Decision tree

The game tree's nodes represent different game states, with levels corresponding to each player's turn. The 'root' node stands for the initial game state (empty grid). At the second level, nodes are created for possible states resulting from the first player's moves (X or O), referred to as 'sons' of the root. Each node in the second level has successors representing game configurations resulting from the opposing player's moves. This progression continues until reaching end-game states, where tic-tac-toe concludes with either a player winning or a tie when the board is full.

Using MinMax's algorithm, the robot system analyzes the current game situation, thanks to computer vision algorithm, identifying possible moves and evaluating the best strategy for the robotic player. Once the move is determined it is sent to the trajectory planner.

Chapter 6

ROS Implementation

This chapter given a quick introduction to ROS and its used tools, followed by a specific explanation of its role in the Corrado project. Then the roles of the different nodes and how they interact are explained. Finally, the setup and game phases are described.

6.1 ROS introduction

The Robot Operating System, commonly known as ROS, is an open-source middleware framework specifically designed for developing robotic software. ROS provides a set of libraries, tools, and conventions to simplify the development of complex and robust robotic systems. ROS supports a modular architecture that allows the integration of different components such as hardware drivers, perception algorithms, motion planning and control systems. Besides ROS, two other fundamental tools for robotic system development were used; RViz to facilitate visualizing and debugging the intricate details of robotic systems, and MoveIt that provides a software interface to implement motion planning, both explained in more detail below.

6.1.1 MoveIt

MoveIt is a powerful ROS library that focuses on motion planning for robotic systems. It enable robots to navigate their environments, avoid obstacles, and manipulate objects with precision. In this project it assist in the planning and control of the movement of the robotic arm. MoveIt also take in into account collision detection and inverse kinematics, critical for trajectory execution.

6.1.2 RViz

RViz, short for ROS Visualization, is a visualization tool integrated with ROS to aid in the debugging and visualization of robotic systems. RViz allows users to visualize sensor data, robot models, and other relevant information in a 3D space. It plays a crucial role in the development and testing of robotic applications by providing a real-time representation of the robot's perception and state. With RViz, users can inspect the robot's sensor readings, verify the correctness of motion plans generated by MoveIt, and gain insights into the overall behavior of the robotic system.

6.1.3 ROS Communication

In ROS, communication occurs through a node and topic based system, later explained, which allows the various components of the robotic system to exchange information asynchronously. The units of code organization in ROS are packages, which enclose nodes, libraries, launch files, data, and resources needed for a specific application or module. Packages simplify the distribution, installation and management of ROS code. Nodes, which are independent processes that perform specific tasks, can be written in various programming languages, such as Python or C++. They communicate with each other by publishing and subscribing messages through communication channels named 'topics'.

6.2 ROS Corrado project

6.2.1 ROS-Raspberry-Pi comunication

In addition, communication with the servomotors is established during this process. To control the servomotors, a ServoDriver class was implemented.
This class uses the `adafruit_motor` and `adafruit_pca9685` libraries on a Raspberry Pi board with a PCA9685 driver. The class is designed to control multiple servo motors, in this case six. It provides methods for writing the position of the servomotors and executing a

trajectory.

6.2.2 Launch files and nodes

Launch files, written in XML format, coordinate the startup and configuration of multiple nodes, topics and other components in a consistent way. These files make it possible to specify nodes to run, parameters to set, and other configuration details.

Two launch files were created, the `tris_main.launch` file is in charge of the game AI and computer vision side while the `demo.launch` file is responsible for the planning and simulation of the movement.

- The `tris_main.launch` file runs the following nodes together:
  - `corrado_camera_node` initializes the webcam using the OpenCV, captures a frame from the webcam every second and processes it. Subsequently, it identifies the tic-tac-toe grid in the frame using the `find_game_grid` function. It extracts the image and uses the LetterRecognition class to recognize whether it contains an 'X' or an 'O'. The current configuration is passed to the MinMaxSolver class and the best move computed, after that the move is published through the BestMove message to the `"/best_move"` topic, taking into consideration the timestamp and accuracy of the game configuration. Finally, the computed move is visualized on the screen.

  - `tris_main_node` is responsible for managing the game phases. It provides an interface between the computer vision module and the trajectory planner. After it receives the `best_move` information from the `corrado_camera_node`, it waits for user input to start the execution of the move by publishing the move on the `/cmd_move` topic.
  - `corrado_main_node` acts as an interface between the commands sent through the `/cmd_move` topic and the actual control of the robotic system trajectory, managed by the `CorradoTrajectoryController` class.

- `demo.launch` file starts the necessary nodes to simulate and plan the robot's movements using MoveIt, allowing the status to be displayed in Rviz. The `demo.launch` file is crucial for setting up and starting a motion planning system using the MoveIt framework. The demo.launch file manages the launch of the `move_group` node, a key component of MoveIt that offers a simplified interface for interacting with the planning system. This node serves as the core of MoveIt, being responsible for motion planning, controlling, and communicating with other ROS nodes. It allows for the configuration of various parameters related to trajectory execution, planning, and more. In our project the `move_group` node was used to compute cartesian path between the waypoints we established to draw the 'x' symbol. The planning algorithm was used with a path resolution of 3mm, a good compromise between processing time and accuracy.

6.2.3 Get ready to start

Accurate camera setup is crucial before using the robot in a tic-tac-toe game, as it ensures precise letters recognition. The first step involves adjusting the mask parameters to clearly distinguish the grid, as already explained in the 4.1 section. First, the `demo.launch` file was launched to establish communication with MoveIt, followed by the launch of the `tris_main.launch` file, which starts the game and the other necessary nodes, already covered in the section 6.2.2.

6.2.4 Let's play the game

Once the main program has been started, the robot's camera communicates the best move visually on a screen and to the node responsible for planning the trajectory. During the robot's turn, the user has to press "Enter" to allow the robot to play. This cycle of communication and interaction between the camera, the planning node and the user is repeated until one of the two players (robot or user) wins the game or the game grid is completely full.

Chapter 7

Conclusion

The project as a whole therefore combined technical, mechanical and computer skills, knowledge of control applied to the robotic system and a good dose of trial and error. The feedback loop closed by computer vision creates a system capable of receiving a move as input and responding to that move (controller) in an appropriate manner by pursuing a goal (desired output), all in autonomy.

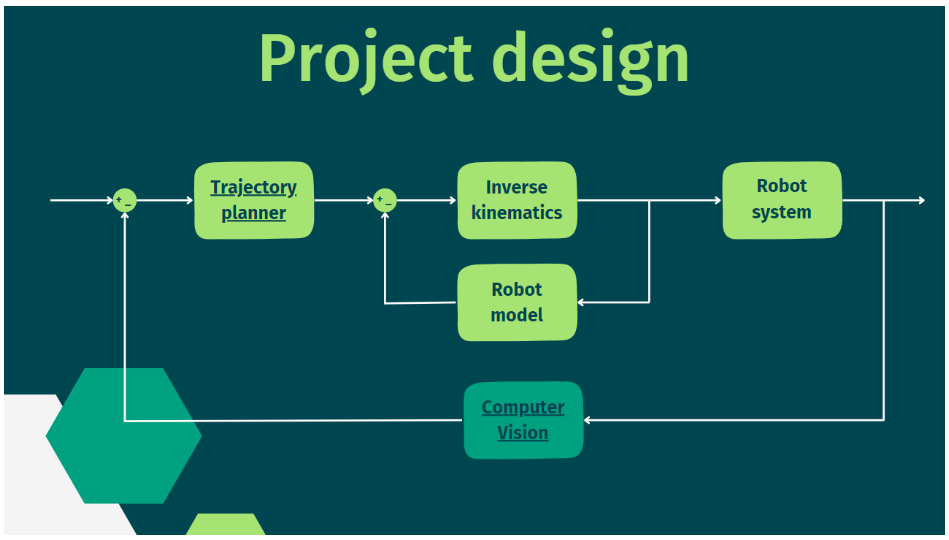


Figure 7.1: Project design

Going through different ideals, languages and programming methods has led us to develop a greater awareness for projects like this. Each tool has its strengths and weaknesses and may be more or less effective depending on the problem to be addressed.

There are definitely possible implementations that we have not explored in depth, but overall our robot CORRADO meets its goals. The final code can be found in the Github Repository [2]

Bibliography

[1] Olga Belitskaya. Classification of handwritten letters, 2021. `https://www.kaggle.com/datasets/olgabelitskaya/classification-of-handwritten-letters`.

[2] Federico II. Corrado ros, 2024. `https://https://github.com/saviodp7/Corrado_CL_ros.git`.

[3] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control.* Springer Publishing Company, Incorporated, 2010.