Remote Interaction with a Nao Humanoid in competitive games
Elective in AI / HRI Report

Flavio Maiorana Valerio Spagnoli Flavio Volpi

September 26, 2024

1 Introduction

In the context of RoboCup games, robots are fully autonomous. However, they surely could benefit from interaction with humans. For example, human soccer players benefit from receiving suggestions and sometimes even explicit instructions on what to do during the match. Similarly, an autonomous soccer robot could receive informed suggestions to enhance its planning

Andrebbe detta meglio sta cosa

. In this project, we aim to make a small step in that direction by developing a system that allows a human operator to remotely control and interact with a Nao Humanoid through a graphical interface.

1.1 Context and Motivation

Receiving instructions from a coach can be a game-changer in a soccer match. And also, receiving feedback from the robot when issuing a command drastically improves the interaction quality. This project was specifically designed to be used in the RoboCup 2024 SPL challenge, where two robots of one team had to compete against two robots of the opponent team, and one of the two robots for each team was controlled by a human operator. Furthermore, the rules of the challenge forced the human operator to turn his back to the field, in order to not directly observe the environment. This constrained us to make use of the directional robot-human communication also for the reconstruction of the world model.

Qua si puó dire di piú

1.2 Objectives

The main objective of the project is to develop a framework that allows the human operator to use the robot as a proxy to interact with the environment, namely the soccer field. To do this, a form of bidirectional interaction between the human operator and the controlled robot is necessary. The human needs to be completely aware of the robot's surroundings, that is, the human can perceive the robot's surroundings only through the robot's sensors. The robot, on the other hand, has to be able to receive commands from the human operator and execute them in real-time. The robot should also be able to send feedback to the human operator in real-time.

Allungare un po' il brodo

Spingerei anche sul fatto che il tutto deve essere utilizzabile in un ambiente competitivo

1.3 Summary of the results

Our system achieved the objective of allowing a human operator to control a Nao by voice and using a graphical interface, and to receive feedback from the robot in response to the commands. The system integrates various modules, such as:

- **Communication**, which manages the communication between the robot and the human operator
- **Mental model**, responsible for representing the field by accessing only the robot's perceptions
- **Interaction**, by answering to the commands before executing them, considering also the feasibility of the execution itself
- **Memory**, the past command is held in memory, in order to resume it in case it is needed

The system was tested in the RoboCup 2024 SPL challenge, where SQPR Team reached the third place, demonstrating the effectiveness of the system in a competitive environment.

2 Related Work

Interpreting human signals has been a challenge for some time now in Robotics. Humans communicate through various modalities, including vision, audio, and motion. This multimodal nature provides rich information that sensory inputs can capture and analyze.

Recent advances in Deep Learning have facilitated the integration of multimodal data, significantly improving the comprehension of relationships within individual modalities, a key factor for precise message interpretation [1] [2].

In RoboCup Soccer, human-robot interaction is predominantly one-way, with human referees conveying game states and events to robots. A significant trend in the RoboCup SPL is the progressive reduction of network communication in favor of human-like signal interpretation, allowing robots to interpret human signals more naturally. [3]

A notable case worth to mention is also [4], where they propose an approach to improve the decision making process through the audience noise by extracting relevant features through MFCC coefficients and applying a reinforcement learning pipeline.

This case could fall into a broader category where the goal is to improve the communication from an ideal coach to the robot in order to improve planning and decision-making. In particular, [5] tackles this problem by designing a system that enriches the planning process with temporal goals and constraints given by human indications.

Our work is inspired by these studies, and the goal is to develop a system that allows a human operator to have a one-to-one interaction with a robot, acting like a coach in a soccer match.

3 Solution

The system can be divided in two main parts: the framework backbone, written in C++, which runs on the robot itself, and the interface, written in Python and Nodejs, responsible for issuing commands to the robot and receiving feedback or sensory information.

3.1 Framework backbone

The framework on which everything stands on is derived from that one of the German team B-Human [6], University of Bremen. At low level, the robot is controlled by three threads:

- Cognition: it is responsible for collecting all informations from the environment through cameras and sensors; also, it takes images and informations as input, and returns high level commands about the actions the robot has to execute;
- Motion: it converts high level commands of the thread Cognition in effective motion of the robot;
- Debug: it allows a communication between the robot and the host pc for the transmission of debugging information.

aggiungere immagine dei processi

The two main components for the collection and storage of information from the environment are called Representations and Modules. Representations store informations at the current instant, and they represent the actual overview of the world. Each representation is a derived class of the class *Streamable*, which allows a direct connection of all attributes and functions of the given representation with all other representations and modules. Furthermore, each representation has a single module provider, which is the only capable of modifying its attributes.

Instead, the task of the modules is to make computations which require specific inputs and returns determined output. In details, a module can specify the representations it needs in input through the macros REQUIRES and USES, and must specify the representations it is going to modify through the macro PROVIDES. A module must specify a function named *update*, that has the scope to perform the real updates of the informations inside the provided representations

The correct usage of REQUIRES and USES is established by a scheduler inside the software, which decides which is the right execution order of the modules according to cyclic dependencies between representations. For example,

@FLAVIO V. : inserire esempio di moduli/rappresentazioni

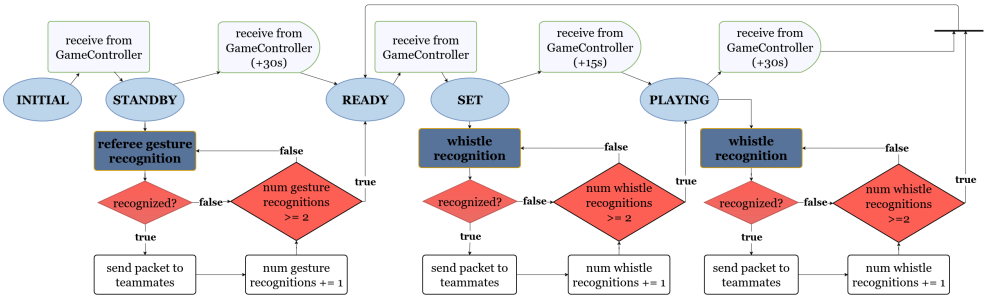

Figure 1: Complete pipeline of the game states

3.1.1 Memory

Boh

3.1.2 Social Reasoning

An important aspect of RoboCup SPL games are the game rules. These are conceived from year to year to shift the games to be more realistic. If we want, they could be considered as a form of social rules which the robots need to obey to while playing. Depending on the game state, some actions are not permitted. For instance, in SET the robots are not allowed to move. Similarly, one could impose a rule that avoids the robot to score in its own goal, even if it is requested to. Of course, in order to to some adequate social reasoning, it is necessary for the robot to have a sufficiently detailed model of the game state and the field.

3.1.3 Mental Modeling

Come rappresenta il mondo il robot? Stato del gioco e del campo fisico

3.2 Interface

We opted for a solution that prioritizes high-level commands and audio-visual feedback. The graphical interface is made by a bunch of buttons, that allow the human operator to send commands to the robot, and a 2D representation of the field that shows the robot's position and its recosntruction of the world.

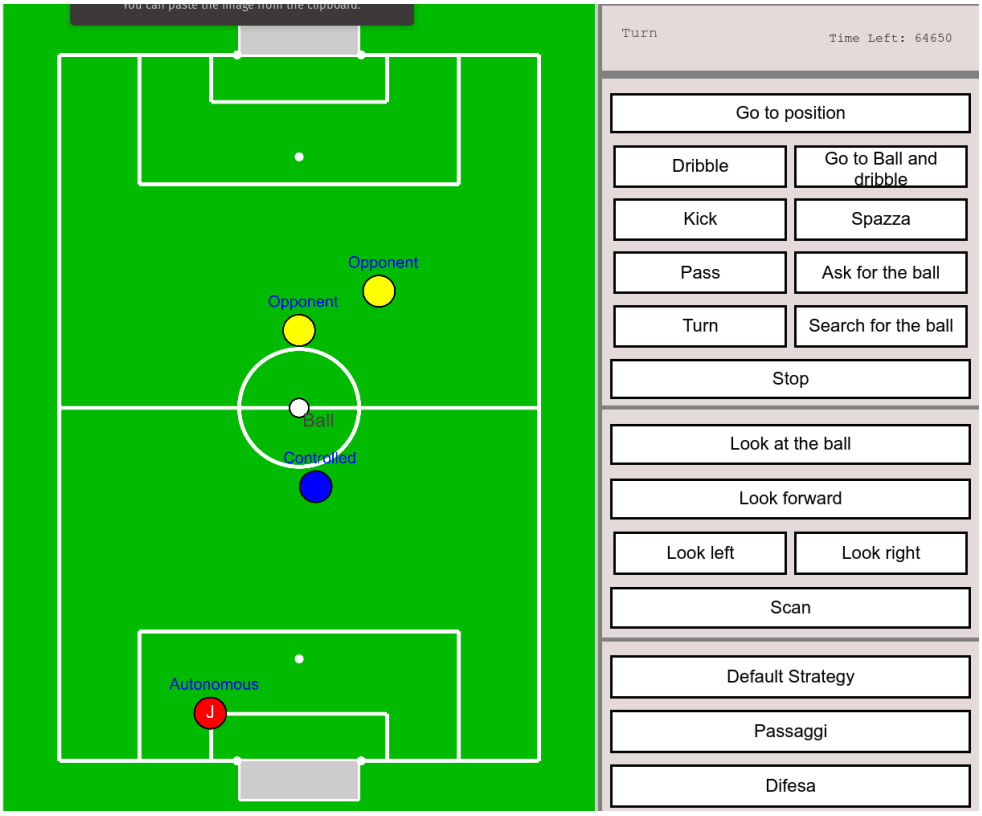

Figure 2: The graphical interface

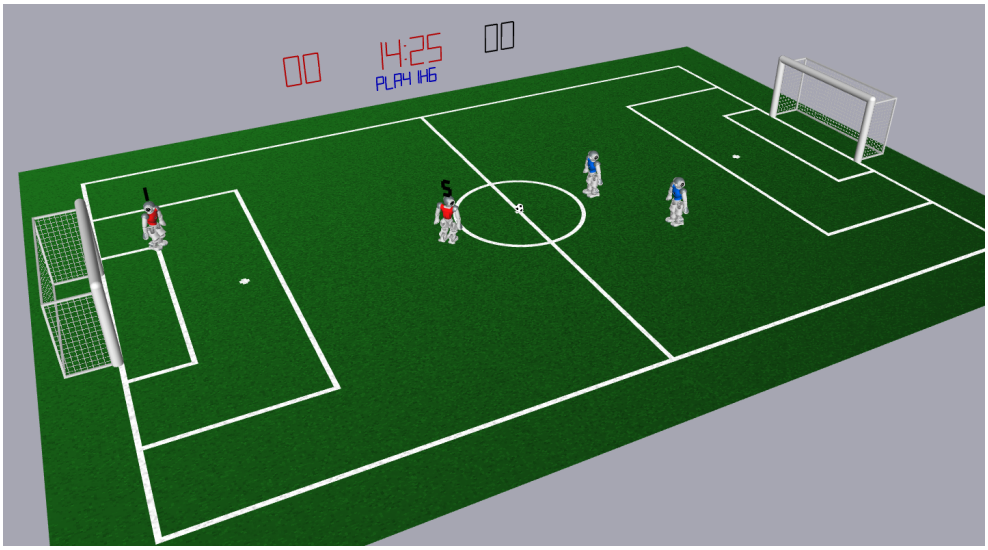

Figure 3: The Corresponding Field Configuration

4 Implementation

From an implementative point of view, the system can be divided in four modules:

- **Communication Layer**: exchanges messages between the robot and the human operator
- **Framework**: the backbone of the system, which runs on the robot
- **GUI**: the graphical interface that allows the human operator to interact with the robot
- **Speech-to-Text Module**: the module that converts the human operator's voice commands to text

These modules are wrapped around the two main data structs that encapsulate the exchanged messages, namely the *HumanCommand* struct and the *DebugInfo* struct. The former is used to send commands from the human operator to the robot. It has the following structure:

```
1B: Command body
1B: Command head
1B: Strategy
8B: [pos_x, pos_y]
command_format: "<BBBff"
```

Basically, it is formatted as a tuple of 5 elements, where the first three are Bytes and the last two are floats. The last two elements are the coordinates of the target position, which are indicated on the 2D field. This precise structure is needed to be able to send the command through the socket and parse it on the C++ side.

The *DebugInfo* struct, on the other hand, is used to send feedback from the robot to the human operator. It has the following structure:

```
1)      4B header
2)      1B version
3)      1B player_num
4)      1B team_num
5)      1B fallen
6-9)    3f (12B) [pos_x, pos_y, theta]
9)      1f (4B) ball_age
10-12)  2f (8B) [ball_pos_x, ball_pos_y]
12)     1H (2B) num_of_data_bytes
13)     1B player_role
14)     1B current_obs_size
15-35)  20B obstacle_types
35-55)  20f (80B) obs_center_x
```

```
55-75)  20f (80B) obs_center_y
75-95)  20f (80B) obs_last_seen
95)     1H (2B) message_budget
96)     1H (2B) secs_remaining
97-99)  2B arm_contact
99-101) 2B arm_push_direction
101-103)2f (8B) arm_time_of_last_contact
103)    2f (8B) padding (whistle)
104-106) 2f (8B) teamball
106-108) 2f (8B) teamballvel
108)    12B padding

data_format: "<d4sBBBB3ff2fHBB20B20f20f20fHH2B2B2f2f2f2f12B"
```

This data packet contains all the information that is needed to have a graphical representation of the robot's mental model of the field.

Spiegare un po'

4.1 Communication Layer

The communication layer is responsible for the exchange of messages between the robot and the human operator. It is written in Python and based on UDP sockets.

```
def send_command_to_cpp(self, command: int, strategy: int, x: int, y: int):
    if command_number > self.config.command_offset:
        command_body_number = Command.Null.value
        command_head_number = command_number - self.config.command_offset
    else:
        command_body_number = command_number
        command_head_number = Command.Null.value
    encoded_data = struct.pack(
        self.config.command_format,
        command_body_number,
        command_head_number,
        strategy_number,
        x_position,
        y_position
    )
    try:
        client_socket.sendto(encoded_data, (robot_ip, command_send_port))
        print(f"Sending message to C++: {encoded_data}")
    except Exception as e:
        print(f"Error in send_message: {e}")
```

This function sends a command that is received from the GUI to the robot. The receive function from nodejs is the following:

```
def receive_command_from_js(self) -> tuple[int, int, int, int]:
    try:
        data, addr = self.server_socket.recvfrom(1024)
    except Exception as e:
        print(f"Error in receiving the message: {e}")
    try:
        message = data.decode()
    except UnicodeDecodeError:
        print(f"Received non-UTF-8 message from JavaScript: {data} from {addr}")
    message_split = message.split('|')[1]
    content_message = message_split.split(',')
    task_type = content_message[2]
    command_number = Command[task_type].value
    strategy_number = int(content_message[5])
    task_label = content_message[4]
    selection = content_message[1]
    print(f"Task Label: {task_label}")
    if selection == "selection":
        x_position = int(content_message[6])
        y_position = int(content_message[7])
        print(f"X Position: {x_position}")
        print(f"Y Position: {y_position}")
    else:
        x_position = 0
        y_position = 0
    return command_number, strategy_number, x_position, y_position
```

Basically, the python scripts receives a command in form of a string from the GUI, unpacks it, puts it into a serialized format and sends it to the robot.

4.2 Framework

The framework is the backbone of the system. The two main functions are the update of the CommandReceiver and the update of the DebugMessageHandler (which sends sensory information). The CommandReceiver is responsible for updating the representation HumanCommand, while giving some audio feedback to the human about the received command. In the following function, the command is parsed, converted to enums and stored in the HumanCommand representation.

```
void CommandReceiver::update(HumanCommand& command) {

  if (theRobotInfo.number != RobotInfo::RoleNumber::controlled) return;
```

```
  char buffer[BUFFER_SIZE];
  int n = socket_read.read(buffer, BUFFER_SIZE);

  // First field (command_body): unsigned char
  HumanCommand::CommandBody received_command_body =
      static_cast<HumanCommand::CommandBody>(buffer[0]);

  // Second field (command_head): unsigned char
  HumanCommand::CommandHead received_command_head =
      static_cast<HumanCommand::CommandHead>(buffer[1]);

  // Third field (strategy): unsigned char
  HumanCommand::Strategy strategy =
      static_cast<HumanCommand::Strategy>(buffer[2]);

  // Fourth field (x_pos): int (4 bytes)
  float x_pos;
  std::memcpy(&x_pos, buffer + 3, sizeof(x_pos));

  // Fifth field (y_pos): int (4 bytes)
  float y_pos;
  std::memcpy(&y_pos, buffer + 7, sizeof(y_pos));

  if (n > 0) {
      SystemCall::say(HumanCommand::CommandBody2String(received_command_body));

      if(received_command_body != HumanCommand::CommandBody::BaseCommandBody){
          command.commandBody = received_command_body;
          command.x = x_pos;
          command.y = y_pos;
      }
      if(received_command_head != HumanCommand::CommandHead::BaseCommandHead)
          command.commandHead = received_command_head;

      command.strategy = strategy;
  }
  return;
}
```

Once the command is stored in the HumanCommand representation, it is used by the specific behavior we designed for the interacting robot. The update function iterates over the possible commands (defined as an enumeration) in the start state with a switch, executing the corresponding skill for each command. For instance:

```
state(pass)
{
    transition
    {
    if (theHumanCommand.commandBody != HumanCommand::CommandBody::Pass)
        goto start;
    }

    action
    {
    int num = theTeamData.numberOfActiveTeammates;
    int size = theTeamData.teammates.size();
    if(num > 0){
        theSaySkill("Pass the ball");
        Vector2f target = theTeamData.teammates[0].theRobotPose.translation;
        KickInfo::KickType kickType = theLibPass.getKickType(target);
        float distance = theLibMisc.distance(target, theRobotPose);
        theGoToBallAndKickSkill(calcAngleToTarget(target),
            kickType, true, distance);
    }
    else{
        theSaySkill("Go to ball and dribble");
        theGoToBallAndDribbleSkill(calcAngleToTarget(
            theLibStriker.strikerMovementPoint()));
    }
    }
}
```

Whenever a new command is received, the robot answers with an audio feedback and reiterates over the command list.

4.3 Speech-to-Text Module

Blablabla

4.4 Interface

5 Results

Blocco 6 delle slide di iocchi

6 Experimental Evaluation

Blocco 6 delle slide di iocchi anche qui

7 Conclusion

Amen

References

[1] Hongyi Liu, Tongtong Fang, Tianyu Zhou, Yuquan Wang, and Lihui Wang. Deep learning-based multimodal control interface for human-robot collaboration. *Procedia CIRP*, 72:3–8, 2018. 51st CIRP Conference on Manufacturing Systems.

[2] Hang Su, Wen Qi, Jiahao Chen, Chenguang Yang, Juan Sandoval, and Med Amine Laribi. Recent advancements in multimodal human–robot interaction. *Frontiers in Neurorobotics*, 17:1084000, 2023.

[3] Valerio Di Giambattista, Mulham Fawakherji, Vincenzo Suriani, Domenico D. Bloisi, and Daniele Nardi. On field gesture-based robot-to-robot communication with nao soccer players. In Stephan Chalup, Tim Niemueller, Jackrit Suthakorn, and Mary-Anne Williams, editors, *RoboCup 2019: Robot World Cup XXIII*, pages 367–375, Cham, 2019. Springer International Publishing.

[4] Emanuele Antonioni, Vincenzo Suriani, Filippo Solimando, Daniele Nardi, and Domenico D. Bloisi. Learning from the crowd: Improving the decision making process in robot soccer using the audience noise. In Rachid Alami, Joydeep Biswas, Maya Cakmak, and Oliver Obst, editors, *RoboCup 2021: Robot World Cup XXIV*, pages 153–164, Cham, 2022. Springer International Publishing.

[5] Emanuele Musumeci, Vincenzo Suriani, Emanuele Antonioni, Daniele Nardi, and Domenico D. Bloisi. Adaptive team behavior planning using human coach commands. In Amy Eguchi, Nuno Lau, Maike Paetzel-Prüsmann, and Thanapat Wanichanon, editors, *RoboCup 2022: Robot World Cup XXV*, pages 112–123, Cham, 2023. Springer International Publishing.

[6] Thomas Röfer, Tim Laue, Fynn Böse, Arne Hasselbring, Jo Lienhoop, Lukas Malte Monnerjahn, Philip Reichenberg, and Sina Schreiber. B-Human code release documentation 2023, 2023. Only available online: `https://docs.b-human.de/coderelease2023/`.