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

Flavio Maiorana Valerio Spagnoli Flavio Volpi

October 12, 2024

1 Introduction

In the RoboCup games, robots are fully autonomous, yet there is potential for improvement through human interaction. Just as human soccer players benefit from receiving real-time suggestions or explicit instructions during a match, autonomous soccer robots could also enhance their performance by incorporating informed guidance. In this project, we aim to take a step in this direction by developing a system that enables a human operator to interact with a Nao humanoid robot via a graphical interface and voice commands. The goal is to enhance the robot's performance and provide real-time strategic suggestions, similar to the role of a football coach.

1.1 Context and Motivation

1.1.1 RoboCup SPL Challenge

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.

1.1.2 Possible Extensions

The system developed in this project could be adapted to various other contexts. For instance, it could be applied to the RoboCup SPL main competition, where a human operator for each team might be allowed to provide real-time instructions to the robots via voice commands, using the graphical interface to view the reconstructed world model of the entire team.

Another potential extension involves modifying the system to enable robots to play alongside human players. In this scenario, the robot would need to interpret human commands and execute them in real-time, allowing for mixed teams of humans and robots in a soccer game.

1.2 Objectives

In the context of the RoboCup SPL challenge, 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.

1.2.1 Bidirectional Communication

Receiving instructions from a coach can significantly impact the outcome of a soccer match. Likewise, getting feedback from the robot when issuing commands greatly enhances the quality of the interaction. The human acting as a coach must be completely aware of the robot's surroundings, while the robot must be able to interpret the human's commands and respond accordingly.

- **Robot-to-Human Communication**: The robot provides the human operator with all the necessary information to reconstruct the world model. These data are transmitted over the network to the operator's computer, where are filtered and displayed on the graphical interface. Moreover, the robot responds with vocal feedback to confirm received commands or to report any issues (e.g. the impossibility to execute the command).
- **Human-to-Robot Communication**:
  - **Coach-to-Robot Communication**: The human operator, acting as a coach, can analyze the reconstructed world model using the graphical interface and decide to send commands to the robot. These commands are transmitted through the network or via voice commands.
  - **Referee-to-Robot Communication**: In the context of the RoboCup SPL challenge, another human, acting as a referee, can send commands to the robot using whistle signals. The robot must be capable of interpreting these signals and responding accordingly.

1.2.2 Real-time Interaction

In the context of the RoboCup SPL challenge, the system must process commands in real-time, given the competitive nature of the scenario. The robot needs to promptly interpret and execute the human operator's commands while providing immediate feedback on the execution status. Any delay in processing

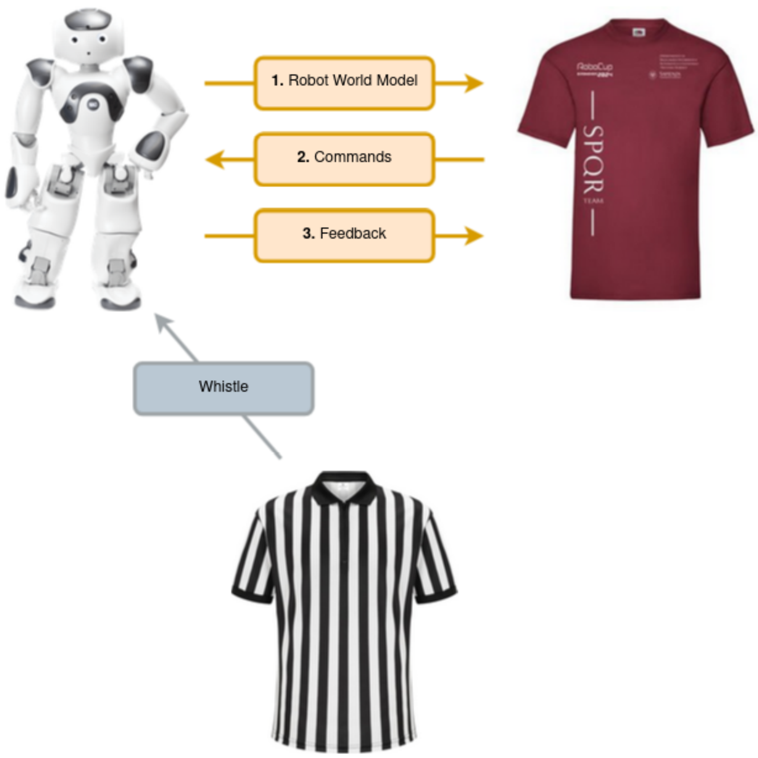

Figure 1: Bidirectional communication scheme

could adversely affect the robot's performance and the outcome of the match. Even in other applications, real-time interaction is critical to maintain the robot's responsiveness to human input and ensure accurate reconstruction of the robot's world model.

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 features, 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.

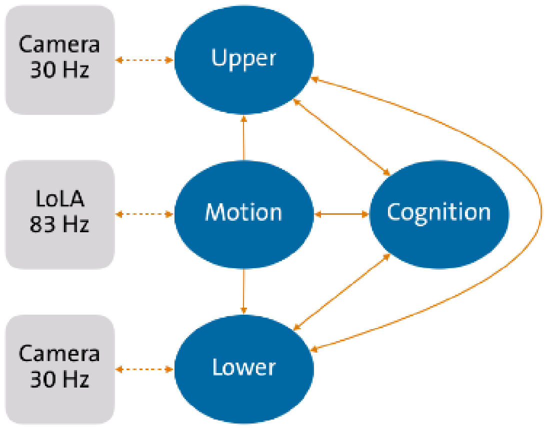

Figure 2: Threads

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 Node.js, 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 four threads:

- **Upper camera thread**: it deals with upper camera of the robot, positioned on its forehead;
- **Lower camera thread**: it deals with lower camera of the robot, placed on its chin;
- **Cognition**: it is responsible for collecting all informations from the environment through cameras and sensors; also, using some informations as input, it 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 control of the 25 joints of the robot;

Both camera threads capture images from the cameras of the robot, and work on them to retrieve informations to describe the world.

3.1.1 Representations and Modules

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 the right execution order of the modules according to cyclic dependencies between representations. For example, if a module M1 updates the representation R1, and this representation is required from module M2 to modify its representation, the module M2 cannot be executed before the module M1, so the scheduler will execute in order M1 and M2.

aggiungere immagine di due moduli che richiedono rappresentazioni

Every update function has an execution frequency of 83 Hz, which practically allows an information refresh in real time.

3.1.2 Behaviors and Skills&Card

Behaviors represent the real actions the robot will take during the game. They are modeled as finite automata, according to CABSL standard (*C-based Agent Behavior Specification Language*) []. This is based on the concepts of options, states, transitions and actions: the options are finite-state machines that describe a determined way of acting; transitions define the conditions to move between two states of the graph; when the latter arrives in a state, the specified actions are executed. Actions are expressed as *Skills*, which are the leaves of the graph and the lowest abstraction elements. They make calls directly to the motion engine of the robot.

Option idea matches with the concept of Card: each card represents one option, and it is the interpretation of an higher abstraction level for the actions. To enter in a card, a robot has to satisfy its specified preconditions; to exit from a card, preconditions are not satisfied anymore or its postconditions are met instead. Cards are the fundamental elements that compose the Decks, which are literally ordered collections of cards. The order establishes the priority of the cards: the ones on the top have higher priority than the ones on the bottom.

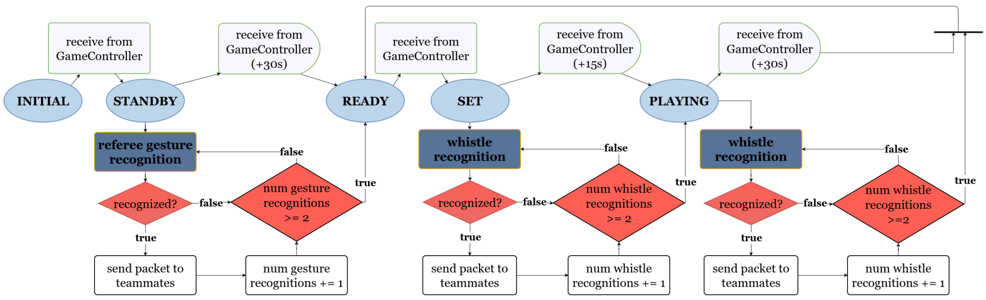

Figure 3: Complete pipeline of the game states

Starting from the first card, the agent enters in it if its preconditions are satisfied, otherwise he discards the card and checks the second one, and so on. So it is very important to establish the right disposition, according to own puproses.

This hierarchy of the system allows to link the decks to the roles of the players in the field. In this way it is possible to associate just the cards relative to the corresponding role, avoiding possible undesired behaviors.

3.1.3 Memory

Boh

Suggestion: The robot's perception is not limited to the current state of the environment; it also utilizes past information to inform its decisions. The robot maintains a memory of past events and when they occurred. For instance, it can remember the last time it saw the ball or when it last encountered an obstacle. This historical data is crucial for the human operator to accurately reconstruct the world model and make informed decisions. Additionally, the robot can recall the last command it received and resume it if necessary.

3.1.4 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.

3.1.5 Mental Modeling

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

Every robot has two different mental models, and they represent the way the robot describes the world: a local world model and a global world model. The former is reconstructed starting just from the preceptions the robot itself has, so the generated mental model will be partial. Furthermore, the model is expressed with respect to the robot itself. It contains:

- relative positions of the perceived obstacles (also teammates and opponents)
- relative positions of the perceived landmarks
- relative position of the ball
- relative velocity of the ball
- time when ball was seen last time
- self localization position in the field

The latter instead is reconstructed using the local world model and the information received from the teammates through the network. Because the local perceptions of the single robot can be not so accurate, due to some additional noise that affects the sensors, some filters are applied to the single attributes that constitute the world model. This is done to result in a more accurate probabilistic estimate of the values of those attributes. It is composed by:

- global positions of the obstacles (teammates and opponents)
- global positions of the landmarks
- global position of the ball
- global velocity of the ball
- time when ball was seen last time from some teammates

Cohexistance of both the mental models is very important. The global one takes advantage of the intrinsic properties of a distributed system, e.g. the capability to capture simultaneously different aspects and glimpses of the environment; this is very helpful to have a wider knowledge of the world, but it is affected by possible noisy informations coming from robots that have not so precise informations. On the other hand, the local model is the key of the interaction between the robot and the environment, because it is a more accurate image of its surroundings. Indeed, it gives a smaller but deeper representation of the world.

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 reconstruction of the world.

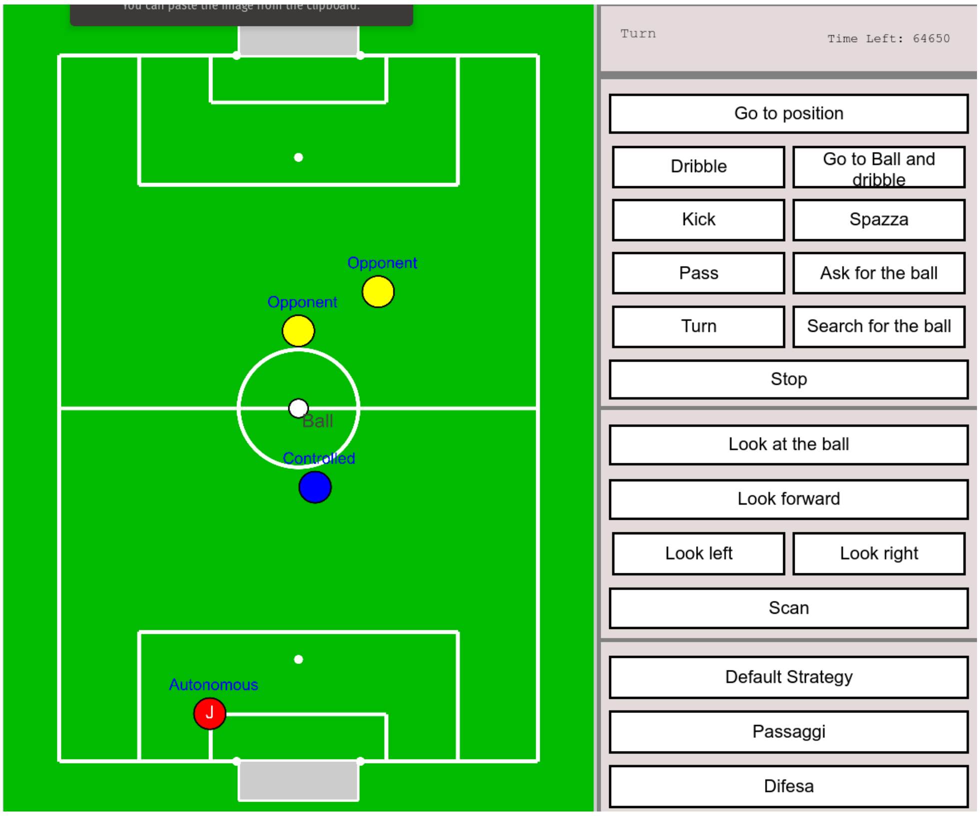

Figure 4: The graphical interface

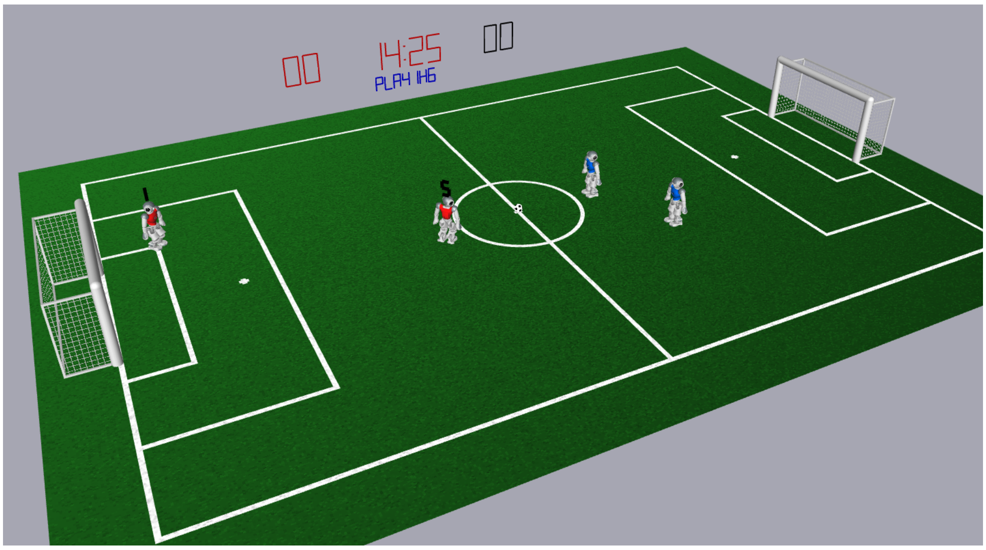

Figure 5: The Corresponding Field Configuration

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.

The most important fields are:

- `fallen`: a boolean that indicates if the robot is fallen,
- `pos_x, pos_y, theta`: the position of the robot in the field,
- `ball_age`: the age of the ball that indicates how many cycles have passed since the robot saw the ball,
- `ball_pos_x, ball_pos_y`: the position of the ball in the field,
- `obstacle_types`: an array that indicates the type of the obstacles in the field (e.g. teammate, opponent),
- `obs_center_x, obs_center_y`: the position of the obstacles in the field,
- `obs_last_seen`: the time when the obstacles were last seen,
- `secs_remaining`: the time remaining in the game,
- `arm_contact`: a boolean that indicates if the robot is in contact with an obstacle,
- `arm_push_direction`: the direction in which the robot arm is pushed,
- `arm_time_of_last_contact`: the time when the robot was last in contact with an obstacle,
- `teamball, teamballvel`: the position and velocity of the ball in the field.

Using these informations, the human operator can have a detailed view of the field through the robot's perceptions.

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 Interface

4.4 Speech-to-Text Module

As a complementary feature of the interface, we integrate a Speech-to-Text module to enable voice interaction with the robot. This integration enhances the communication between the coach and the robot, making interactions faster and more efficient. In a dynamic environment like soccer, quick information exchange is crucial, and voice commands streamline the process.

5 Results

5.1 Usability

The system was tested exclusively by team members, and the feedback was positive across various aspects:

- **Reliability**: The system underwent extensive testing before the RoboCup SPL challenge and was used during the competition without any significant issues. Communication between the robot and the human operator remained stable, with the robot accurately executing commands and providing the expected feedback.

- **Validity**: The robot successfully interpreted the human operator's commands and executed them in real-time. Additionally, it provided the operator with a comprehensive view of the field through its sensors and offered feedback on the execution of commands.

- **Sensitivity**: The system is sensitive to network delays, which can impact the real-time interaction between the robot and the human operator.

The graphical interface was designed to be user-friendly and intuitive, ensuring that the human operator can interact with the robot quickly, which is essential in the competitive RoboCup SPL challenge, where each match lasts only 3 minutes.

5.2 Adversarial Environment - RoboCup 2024 SPL Challenge

The system was tested in the highly competitive and adversarial setting of the RoboCup 2024 SPL challenge. Several factors in this environment could impact the system's performance, such as:

- **Network Delays**: The system's real-time interaction can be affected by network delays, posing challenges to command execution.
- **Game Rules**: The robot must be capable of interpreting and adhering to game rules in real-time.
- **Opponent's Strategy**: Opponents aim to exploit any weaknesses in the system, requiring the robot and the human-operator to adapt to their strategies dynamically.

Despite these challenges, the system performed effectively, and the team secured third place in the competition.

6 Experimental Evaluation

In this section we make some considerations about the efficacy of our pipeline by validating it experimentally. We focus on the Nao robot's performance in the RoboCup SPL 2024 competition and its effectiveness as an assistive player guided by a human coach. The goal of the evaluation is to assess how well the robot responds to both autonomous decision-making and human-provided instructions, and how these interactions impact its overall gameplay and decision-making quality during the tournament.

The primary research questions are:

1. How accurately does the robot convey its mental model to the human operator?
2. To what extent does the combination of autonomous behavior and human coaching enhance the robot's performance in the competitive environment of RoboCup?
3. How effectively does the human-robot interaction framework support timely and efficient communication under the constraints of real-time soccer gameplay?

We have formulated the following hypotheses to address these questions. Firstly, we hypothesize that the robot is able to summarize well enough its perceptions by sending them to the 2D interface. We also hypothesize that the robot can correctly interpret and execute human commands in over 90% of the given situations during the game. Thirdly, we hypothesize that the collaborative control system will enhance the robot's overall performance, leading to more effective gameplay and strategy adaptation compared to fully autonomous play. Lastly, we hypothesize that the communication framework will facilitate smooth, real-time interaction between the human coach and the robot, enabling responsive and coordinated action during the game.

6.1 Experimental Variables

To test these hypotheses, we have defined several key variables. The independent variables include:

- **Control mode:** whether the robot operates autonomously or under human-assisted control.
- **Type of tactical scenario:** predictable (routine situations) vs. unpredictable (dynamic, novel situations).
- **Prior knowledge of the human coach:** whether the coach has a background in football (experienced) or no prior football coaching experience (inexperienced).

The dependent variables include:

- **Decision-making accuracy:** measured by the robot's ability to correctly interpret and act on human commands in real-time.
- **Tactical performance:** measured by the robot's ability to adapt its strategy based on human input, especially in response to changing game conditions.
- **Overall game performance:** measured by key metrics such as goals scored, positioning, and game results.

By evaluating these variables, we aim to understand how the integration of human coaching impacts the robot's performance in a competitive, high-pressure environment like RoboCup.

6.2 Outcome

By testing the pipeline directly in the RC24 Challenge we gained the following insights:

- Given the commands are light-weight, they proved to be very effective and efficient, obtaining high accuracy and winning over other teams' more sophisticated approaches like using a portable console as interface, or also a visore. Also, the voice commands are especially useful in emergency situations , like the robot walking out of the field, where the GUI may be not the ideal choice.

- The second and third indicator are somehow intertwined, since the second influences the third. We noticed that we were able to seamlessly integrate the human coached robot as if it were an automomous one on steroids. We were able to complete passages and score goals in a very reduced timespan, demonstrating once more how important the high-level decision-making layer is in the context of RoboCup.

The first and third independent variables are, of course, important factors to be considered in designing such a system and are somehow related. The most important question is how much autonomy to leave to the robot, since less knowledge of the human would require more autonomy by the robot. Our choice of leaving some degree of autonomy to the robot just for low level actions and world-modeling proved itself successful.

7 Conclusion

In this project, we developed a system that enables real-time human-robot interaction in competitive soccer games, specifically targeting the RoboCup SPL 2024 challenge. By allowing a human operator to interact with a Nao humanoid robot through both voice commands and a graphical interface, the system demonstrated the potential for enhancing robotic performance in dynamic, real-world environments. The bidirectional communication framework, combined with the robot's mental modeling and real-time feedback capabilities, enabled the operator to act as a coach, offering strategic insights that improved the robot's decision-making on the field.

The success of our system in the RoboCup 2024 SPL challenge, where our team reached third place, validates the effectiveness of integrating human input into autonomous robot systems in high-pressure, time-sensitive scenarios. The project also highlights the importance of balancing local and global world models in collaborative robotics, ensuring accurate perception and effective execution of commands.

Future work could expand on this system by further refining the interaction between the robot and human, improving the robot's ability to interpret and prioritize human input in more complex scenarios and also with different modalities (e.g. gestures). Related to the latter, integrating more advanced social reasoning and context-aware behaviors could enhance the robot's autonomy, allowing for seamless transitions between human-provided instructions and self-driven actions.

Ultimately, this project represents a step forward in developing robust frameworks for human-robot collaboration in competitive environments, with potential applications extending beyond soccer games into broader fields where real-time human-robot communication is essential.

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/`.