Multiplayer Soccer Game Professor Kaliappa Ravindran

(Robo soccer simulation using Webots)

Members: John Badji, Fherdinand Corcuera

Table Of Contents

I- Introduction:

The aim of this project is to develop a robot football-soccer game simulator that incorporates Artificial Intelligence techniques into each player. By implementing these techniques, the players will be able to demonstrate intelligent gameplay and compete to achieve victory. The project also involves creating programs that enable the robots to perceive their environment, collaborate with teammates, and compete against opponents.

Additionally, we have used "Webots" to build an interactive environment for the multiplayer soccer game. The soccer field features two goalposts, providing a clear objective for the players. To enhance the game experience, we have formed two teams, each comprising three players, who will show their robotic skills in a competitive match. These players are represented by 2-wheel robots that exhibit strategic prowess while navigating the field. The inclusion of a soccer ball robot demands players to utilize a range of techniques in their efforts to score goals. To maintain fair play, an invisible robot referee oversees the match, ensuring adherence to the rules and resolving any conflicts that may arise, thereby guaranteeing a level playing field.

Figure 1: Screenshot of our Webots world

II- Decision Making Models:

Centralized decision-making involves a centralized authority, typically a designated robot or a central control system, making decisions and issuing instructions to individual robots on the field. In this model, a referee robot or a central controller assumes the responsibility of overseeing the game, enforcing the rules, and coordinating the actions of the participating robots. The centralized decision-maker analyzes game events, processes sensor data, and communicates instructions to the individual robots, guiding their movements and actions. This approach ensures a synchronized and coordinated gameplay experience, as all robots follow a unified strategy dictated by the central authority.

On the other hand, ***autonomous decision-making*** places a greater emphasis on individual robot autonomy and decentralized decision-making. Each robot on the field acts as an autonomous agent with its own decision-making capabilities. These robots independently analyze the game situation, process sensor data, and make decisions based on their individual perception of the environment and their assigned objectives. They have the freedom to choose their actions and strategies, adapting to dynamic changes in the game without relying on external commands or centralized control.

We decided to go for a hybrid approach that combines elements of both centralized and autonomous decision-making. A centralized system provides high-level game strategy and coordination instructions to the individual robots while allowing them autonomy in executing lower-level actions and local decision-making.

III- Game Integrity:

In our centralized decision-making model, the robot referee plays a crucial role in maintaining the integrity of the game and ensuring a fair and organized gameplay experience for all participants. Acting as the authoritative figure, the robot referee takes charge of overseeing the match and upholding the established rules and regulations.

One of the primary responsibilities of the robot referee is to accurately announce when a goal is scored. After detecting a successful goal, the referee robot relays this information to the players and spectators, ensuring that the scoring team receives an acknowledgment for their achievement. The referee also takes charge of resetting the ball and players to the kick-off position after a goal has been scored, facilitating a seamless continuation of the game.

The referee is equipped to address irregularities or infractions during the game effectively. For instance, if players become misaligned or deviate from their designated positions, the referee intervenes by repositioning them to ensure fairness and adherence to the game's rules. Also, in cases where the game encounters a lack of progress or a stalemate, the referee robot steps in to resolve the situation, facilitating the game's flow and maintaining a dynamic and engaging experience for all participants.

IV- Robot Communication:

Effective communication plays a vital role in enabling coordination, strategic gameplay, and seamless interaction between the players and the supervisor. Each player robot is equipped with a communication system consisting of at least three receivers and one emitter, facilitating both intra-team and inter-team communication.

- The first receiver serves the purpose of receiving data and instructions from the supervisor or referee. Through this channel, the player robots receive crucial information regarding game events such as kick-offs and goals, allowing them to align their strategies and actions accordingly.
- The second receiver is dedicated to receiving data from the ball. As the central object of the game, the ball communicates its signal strength and direction to both the players and the supervisor. This information provides valuable insights into the ball's location and movement, enabling the player robots to anticipate and react to its trajectory effectively.
- The third receiver, along with the emitter, enables intra-team communication among the player robots. This communication channel allows for the exchange of messages, coordination of actions, and sharing of strategic information within the team. By leveraging

this internal communication, the player robots can collaborate, synchronize their movements, and implement sophisticated team strategies to outmaneuver the opposing team.

Meanwhile, the supervisor, acting as the central authority, plays a crucial role in managing the game and facilitating communication. The supervisor receives messages from the ball, leveraging the second receiver of the player robots. This communication ensures that the supervisor stays updated with the ball's information, including signal strength and direction, which is essential for making accurate decisions and enforcing the rules of the game.

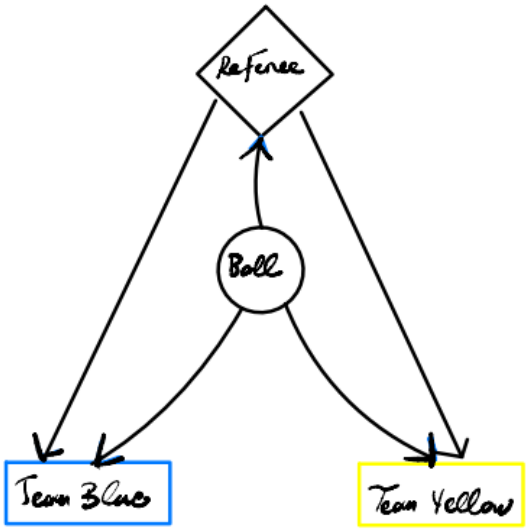


Figure 2:High Level overview of game architecture

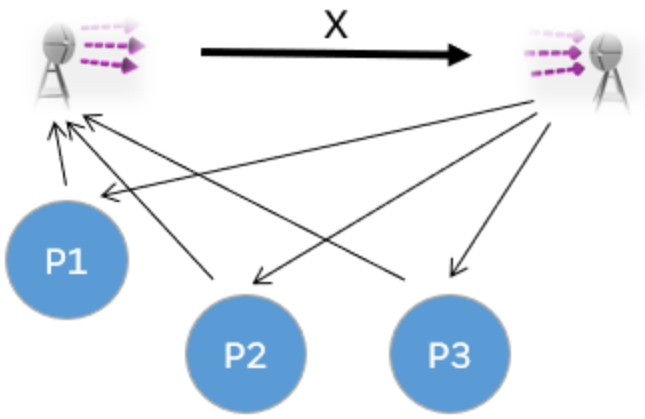


Figure 3: Intra-team communication. (General idea of how communication channels work in Webots)

V- Game Plan and Strategies:

For the game plan, we went with a mostly autonomous decision-making model where each robot can react to the game dynamics independently, making on-the-spot decisions and adapting their strategies based on their local observations. Just as in a live soccer game, each robot player on our team has been given a specific role. With three robot players per team, we've assigned the roles as follows:

- **Player 1**

For Player 1, the role is primarily defensive. This includes chasing the ball within range and attempting to score when possible. However, they must surrender the ball if a teammate is already in possession of it within the offensive zone. Additionally, if the ball is out of sight, Player 1 should quickly return to the defensive zone.

- **Player 2**

In the game, Player 2 plays a pivotal role as the center support. Their primary objective is to chase the ball when it is within range and attempt to score. It is important that they only give up the ball to Player 3 (the offensive player) when they are already in control of the ball. Additionally, Player 2 should position themselves strategically at the center when they are unable to locate the ball.

- **Player 3**

For Player 3, the role of a striker involves several tasks: Firstly, they need to pursue the ball when it's within their range and attempt to score a goal. Secondly, they should only release the ball to their teammate (Player 1) when they are in the defense zone. Lastly, when they are unable to locate the ball, they should position themselves strategically in the opponent's danger zone.

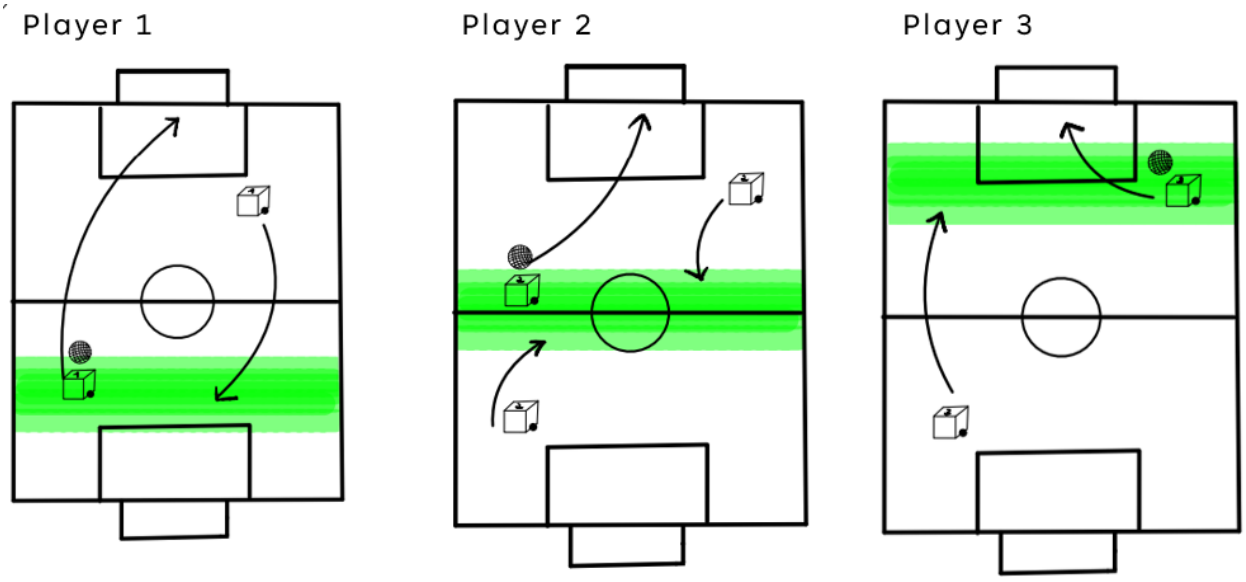
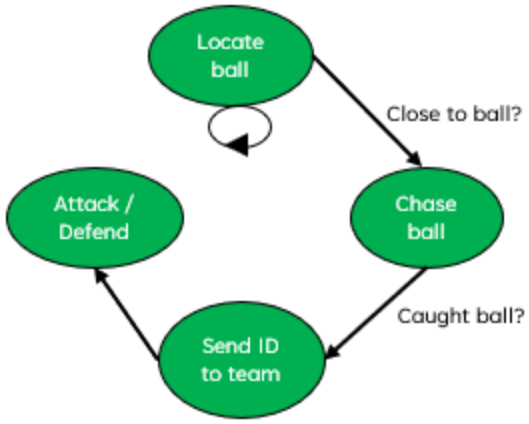


Figure 4: Players' role depiction

Although each player has a distinct role, they all adhere to a shared strategy that can be succinctly depicted as the following state machine.



The game's fundamental logic is augmented by incorporating new components such as the robot's location, orientation (which denotes the direction in which the robot is pointing), identification and location of teammates, and sonar readings to detect objects. This has resulted in a more natural and uninterrupted gameplay experience, with fewer occurrences of team members being trapped in a stalemate.

- **Strategies formations:**

Using unique roles assigned to each player, we came up with 3 strategies that team can choose to play when starting a game.

- **Defensive Strategy**: During defensive play, both Player 1 and Player 2 assume a defensive stance. Should either player lose possession of the ball, they both promptly retreat to their team's defensive zone.

- **Offensive Strategy**: During offensive play, both Player 2 and Player 3 assume an offensive stance. If either player loses the ball, they strategically position themselves in the opponent's team offensive zone.
- **Collaborative Strategy**: During collaborative play, all players assume their assigned roles. All sections of the field are covered.

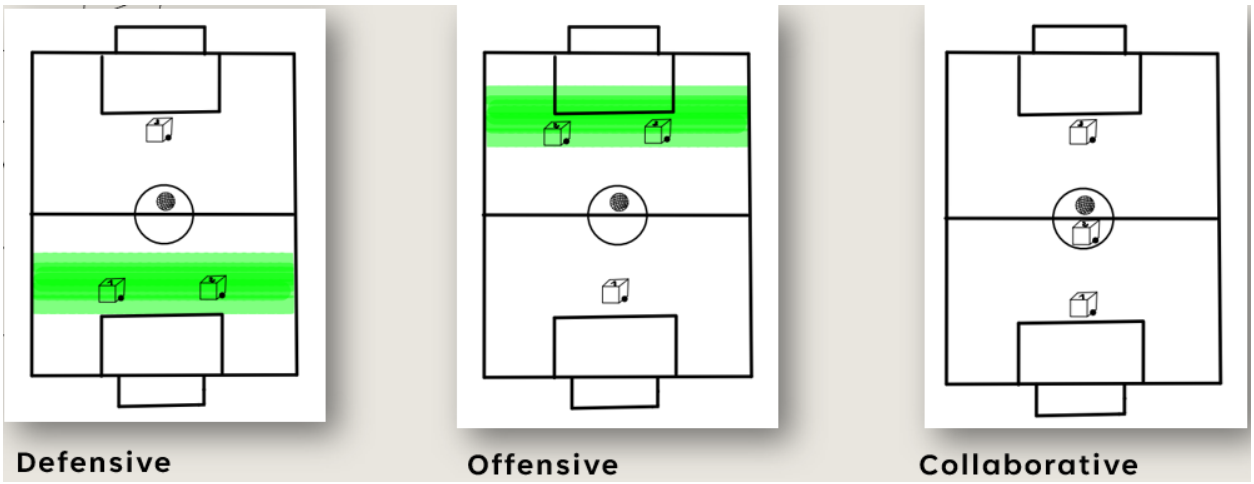


Figure 5: Depiction of game plans and strategies formations

- ***Choosing a strategy***

At first, we aimed to function as a coach and modify tactics during gameplay through keyboard input. However, we faced time limitations that prevented us from implementing this feature.

Our approach is simple. One member, designated as the captain, randomly chooses the team's gameplay strategy. Afterward, this captain shares the strategy with the rest of the team.

Robot players wearing jerseys with number 1 have an extra emitter, the ***strategy emitter***, enabling them to communicate the strategy with the other two team members. The other two

players have a corresponding receiver for receiving the strategy message. The strategy channel operates independently of other communication channels to avoid confusion.

VI- Handling Highlights Events:

Our team has developed a set of functions dedicated to displaying messages on the world tab. These functions serve various purposes and are designed to enhance the user experience during the game.

Show “GOAL!” message:

```
def draw_event_messages(self, messages: List[str]):
    """Visualize (draw) the event messages from queue

    Args:
        messages: List of string messages to be drawn
    """
    if messages:
        self.setLabel(
            LabelIDs.EVENT_MESSAGES.value,
            "\n".join(messages),
            0.01,
            0.95 - ((len(messages) - 1) * 0.025),
            0.05,
            0xFFFFFF,
            0.0,
            "Tahoma",
        )

def draw_goal_sign(self, transparency: float = 0.0):
    """Visualize (draw) a GOAL! sign after goal gets scored.

    Args:
        transparency (float): the transparecny of the text, with 0 meaning
            no transparency and 1 meaning total transparency (the text will
            not be visible).
    """
    team_goal = None
    team_kickoff = None

    ball_translation = self.get_ball_translation()
    ball_x, ball_y = ball_translation[0], ball_translation[1]

    if is_in_blue_goal(ball_x, ball_y):
        self.setLabel(
            LabelIDs.GOAL.value,
            "GOAL!",
            0.30,
            0.40,
            0.4,
            0xf8fc7c,
            transparency,
            "Impact",
        )
    elif is_in_yellow_goal(ball_x, ball_y):
```

The `draw_event_messages` function takes a list of string messages as input and visualizes them by setting a label. The function checks if the `messages` list is not empty. If it is not empty, it uses the `setLabel` method to display the messages as a concatenated string, with each message on

a new line. The position and style of the label are determined by the provided parameters. The `draw_goal_sign` function is responsible for visualizing a "GOAL!" sign when a goal is scored. It takes an optional parameter `transparency` to control the transparency of the text. The function retrieves the translation (coordinates) of the ball using the `get_ball_translation` method. It then extracts the x and y coordinates from the translation. Next, it checks if the ball is in the blue goal using the `is_in_blue_goal` function with the ball's coordinates. If it is, the `setLabel` method is called to display the "GOAL!" sign in blue color (0x2377fc) at a specific position with the provided transparency and font style. If the ball is not in the blue goal, it checks if it is in the yellow goal using the `is_in_yellow_goal` function. If it is, the `setLabel` method is called again, but this time the "GOAL!" sign is displayed in yellow color (0xf8fc7c) at a specific position with the provided transparency and font style.

"GOAL" Message after a team scored.

In order to display the names of the teams, the robot's name initiating the kick-off, socres, as well as the start and end times, dedicated functions were implemented. These functions share a fundamental similarity with the `draw_event_messages` functions. By utilizing these functions, the relevant information can be visually presented in a manner consistent with how event messages are displayed. This allows for a cohesive and unified approach to visualizing different types of game-related information within the system.

Function for calling the message formatter based on the progress of the game:

Understanding the progress of the game is crucial for presenting appropriate messages at the right time. To accomplish this, we have devised a function that evaluates the game's progress and triggers the message formatter accordingly. By dynamically adjusting the message content based on the game's progression, we deliver real-time updates and relevant information to the users. This message formatter function plays a fundamental role in ensuring that the displayed messages are

visually appealing and well-structured. It takes the raw information related to game events, such as goals scored, kick-off initiations, and game start/end times, and formats it into a user-friendly message format. Once the message is properly formatted, it is rendered on the screen, providing clear and concise information to the users.

Progress Tracker for the game:

```
    def track(self, position: List[float]):
        if not self.prev_position:
            self.prev_position = position
            return

        prev_position = self.prev_position

        delta = math.sqrt(
            (prev_position[0] - position[0]) ** 2
            + (prev_position[1] - position[1]) ** 2
        )

        # store the currently computed delta sample
        self.samples[self.iterator % self.steps] = delta
        self.iterator += 1

        self.prev_position = position

    def is_progress(self) -> bool:

        s = sum(self.samples)
        if s > self.threshold:
        if self.iterator < self.steps:
            return True

        return s >= self.threshold
```

The implementation of a progress checker was crucial in ensuring the accurate display of all event messages. By incorporating the progress checker, we could monitor the game's progress and determine when specific events should be triggered. To enable the progress checker to respond to new positions, we introduced a function called "track." This function allowed the progress checker to update its calculations based on the latest position data.

The track function employed a distance formula as its key method to calculate the distance between the current position and previous positions. This distance measurement enabled the progress checker to assess the magnitude of movement and determine if an object had undergone a significant displacement. To maintain precise progress tracking, the calculated data was stored

within the system, enabling efficient access to past position information during movement calculations.

Through these techniques, we established a robust mechanism for tracking game progress and displaying event messages accurately based on calculated movement and positional data. This approach ensured the reliability and effectiveness of the system's progress-tracking capabilities.

Referee roles:

The referee function, which encompasses multiple definitions and roles, has been implemented as the central decision-making component in the system. It incorporates several key functionalities:

Goal Scoring: A function has been defined to determine whether a goal has been scored. It evaluates various conditions and returns true or false based on the outcome.

Initial Position Tracking: To facilitate accurate progress tracking, an initial position definition has been created. It provides the starting position of the ball, ensuring precise tracking throughout the game.

Event Display: An event subscriber has been integrated into the system, enabling the display of relevant messages on the world tab. This feature ensures real-time updates and effective dissemination of information.

Event Message Processing: A comprehensive lineup of event messages has been defined. The function processes incoming messages, identifying their relevance to specific events, and determines which message should be displayed for each event.

Position Reset: A function has been implemented to reset the positions of both the robots and the ball to their initial positions. This functionality is essential for maintaining the integrity of the game and enabling consistent gameplay.

Game Progress Check: The referee function takes charge of monitoring the progress of the game. It performs various checks, such as evaluating whether a goal has been scored or if the ball is outside the field boundaries. Additionally, it handles the setup of kickoffs by placing the kicking-off robot near the center point.

With its ability to oversee various aspects of the game and make crucial decisions based on the defined rules and conditions, the referee function plays a pivotal role in maintaining the integrity and smooth operation of the game. By serving as a central authority, it upholds fairness and impartiality, contributing to the smooth progress of the game for all players.

VII- Future Works:

In addition to its core responsibilities, further enhancements can be made to the program by integrating a function that acts as a "coach", allowing for strategy changes mid-game using keyboard input. This feature enables real-time adjustments and allows for adaptive gameplay based on the user's decisions, enhancing the overall performance of the team.

Moreover, efforts can be made to improve communication between the robot and other robots on the playing field. By establishing more efficient and reliable communication channels, such as through enhanced wireless protocols or advanced messaging systems, robots can exchange critical information, share strategies, and coordinate their actions more effectively. This increased

level of communication promotes teamwork and collaboration, ultimately leading to more synchronized and successful gameplay.

Once the communication between the robots has been improved, it opens up opportunities for advanced programming techniques that facilitate coordinated gameplay. For example, algorithms can be developed to coordinate the movements and actions of multiple robots simultaneously, allowing them to work together strategically and execute complex game plans. This coordination can involve tasks such as passing the ball, setting up plays, and executing defensive or offensive maneuvers in a synchronized manner, maximizing the team's overall performance, and increasing their chances of success.

VIII- Conclusion:

As part of our senior project design 1, we accomplished several future works. Firstly, we addressed and resolved issues related to the player's controller that were causing undesirable outcomes. This led to improved coordination and communication among the robots, enabling them to execute more effective strategies. The enhanced information exchange among the robots allowed for better coordination of various activities, expanding the player's controller to encompass more complex actions, such as determining which robot should locate the ball and which one should return to the initial position. To enhance gameplay further, we implemented a separate game plan specifically designed for collaborative, offensive, and defensive plays. Additionally, we successfully implemented auto-localization, which enabled the robots to accurately determine their positions on the field. This information played a crucial role in planning effective moves, especially in cases where the striker failed to drop below half-court, or

the defender did not advance over half-court. This setup ensured efficient and reliable communication during the game.

Through the development of an algorithm integrating artificial intelligence, significant progress was made in enabling autonomous participation of robots in a soccer game. The project successfully achieved its objectives, focusing on the development of algorithms for communication and movement control, both vital aspects of effective gameplay. With an array of sensors and other advanced capabilities, the robot was well-equipped to navigate the challenges presented within the simulator environment. The project's standout accomplishment was enabling a single robot, operating within the virtual world, to autonomously execute a sequence of independent movements, efficiently locating the ball and directing it towards the goal. This achievement showcases the potential for AI-powered robotics to emulate human-like decision-making and motor control, laying the foundation for further advancements in robotic sports and related applications.

*How does Webots handle the rendering and refresh rate?

Webots does not directly control the refresh rate of the simulation display, as it is primarily determined by the display device and operating system settings of the computer. The scene refresh rate in Webots is typically set to 60 frames per second, but the actual refresh rate can be calculated based on the basicTimeStep and desired Frames Per Second (FPS). By dividing 100 by the basicTimeStep, the number of simulation steps within 100 milliseconds can be determined. Multiplying the basicTimeStep by the FPS yields the time required to render one frame. The

number of frames that can be rendered within 100 milliseconds is then obtained by dividing 100 by the frame render time and rounding up. Taking the reciprocal of this value and ensuring it does not exceed 1 provides a rate to synchronize simulation steps and rendering frames. This rate can be used in the simulation control loop to regulate the frequency of calling the Webots steps function. It is important to note that while increasing the basicTimeStep can accelerate the simulation, it may sacrifice accuracy and stability, especially in physics calculations and collision detection.