Social Navigation Using Soar Cognitive architecture as high-level controller for mobile robots

Thesis MSc Robotics by

Badr Essabri

to obtain the degree of Master of Science in Robotics at the Delft University of Technology, to be defended on Tuesday 18 February 2025

Student Number: 5099412

Submission Date: February 10, 2025

Project Duration: January, 2024 – February, 2025

Faculty: Mechanical Engineering (ME)

Department: Cognitive Robotics (CoR)

Supervisors:

Dr. C. Hernandez Corbato, TU Delft

F. Zamani Khalili, MSc, TU Delft

Thesis Committee:

Dr. C. Hernandez Corbato, TU Delft

F.Z Zamani Khalili, MSc, TU Delft

Dr. J. Sijs, TU Delft

Social Navigation Using Soar Cognitive architecture as high-level controller for mobile robots

Thesis MSc Robotics Badr Essabri, Date: February 10, 2025

Badr Essabri, 5099412
Cognitive Robotics
Delft University of Technology
Delft, Netherlands

Forough Zamani Khalili, Supervisor
Cognitive Robotics, Robot Dynamics
Delft University of Technology
Delft, Netherlands

Carlos Hernandez Corbato, Supervisor
Cognitive Robotics, Robot Dynamics
Delft University of Technology
Delft, Netherlands

***Abstract*—This paper presents a Soar-based system for social navigation in mobile robots, where the Soar cognitive architecture serves as a high-level controller to dynamically adapt the navigation behavior of a lower-level motion controller based on environmental and social cues. The navigation behavior configured in this work is the maximum allowed speed, enabling safe and socially appropriate navigation around humans. Soar's symbolic reasoning and procedural logic provide a scalable and flexible framework for high-level control in complex environments.**

The research focuses on human-following within social navigation, with experimental results demonstrating the system's effectiveness and adherence to social norms. However, limitations in navigation performance and simulation realism highlight opportunities for future work to enhance Soar's application in complex, real-world scenarios.

***Index Terms*—Soar, Cognitive Architecture, Social Navigation, High-level control**

I. Introduction

In robotics, there is a growing need for systems that can operate effectively in complex, dynamic environments [1]. These environments require robots to make decisions and adjust their actions in real-time to complete specific tasks. The complexity in these environments typically arises from the varied and dynamic nature of the surroundings, making modeling the environment and decision-making challenging.

One domain in which robots must handle such complex environments is social navigation, which involves moving through human-populated environments while adhering to social norms [1]. The complexity of social navigation stems from the inherent variability of human behavior. Unlike static obstacles, humans exhibit diverse, context-dependent movements influenced by individual goals, cultural norms, and social interactions [1]. Furthermore, these environments are inherently dynamic, with constantly changing interactions and spatial configurations. This combination of variability and rapid change poses significant challenges for robotic systems attempting to navigate effectively while respecting social norms.

To effectively operate in such environments, a robot must possess a decision-making system capable of interpreting complex social cues. It must then adapt its behavior in real-time, balancing task objectives with the need to maintain safety, comfort, and efficiency in human-robot interactions. Without this ability to dynamically model and respond to its surroundings, a robot's actions risk being intrusive, inefficient, or even hazardous in human-populated spaces.

Given the challenges posed by such environments in social navigation, this paper explores the Soar Cognitive Architecture as a potential decision-making system capable of addressing these complexities. The Soar cognitive architecture is a general cognitive framework designed to model human-like decision-making [2]. It aims to model human cognition across various tasks, integrating modules that model different cognitive processes, such as short-term memory, inference, planning, and decision-making [2]. It operates by representing knowledge through symbolic logic and performing reasoning over this knowledge using its decision cycle algorithm, which iteratively updates the system's state and selects the most appropriate actions.

In robotics, one approach to decision-making in complex and dynamic environments is to predefine specific actions and behaviors for various anticipated scenarios within a system. This system is then refined over time, gradually expanding its ability to handle a wider range of situations and select appropriate actions. However, as environments become more complex and unpredictable, managing such a decision-making system can become increasingly difficult. Adding new rules can unexpectedly introduce contradictions, ambiguities, or unintended interactions, leading to failures in reasoning or inconsistencies in behavior. An example of this challenge is the state explosion encountered by finite state machines as they scale [3].

To address these challenges, Soar provides a structured framework for organizing knowledge about both the environment and possible actions [2]. By utilizing symbolic logic in an ontology, Soar represents information about the environment's state and uses procedural rules to model a wide range of potential actions. This approach enhances the management and scalability of predefined behaviors. An example of this

will be shown in the discussion. Soar's structured representation makes it a promising candidate for decision-making in complex and dynamic environments, such as social navigation, where the robot must handle a diverse range of scenarios, as demonstrated in this paper.

Given this potential, the overarching goal of this research is to explore how the Soar cognitive architecture can be applied to social navigation for robots. Specifically, this work addresses the research question: How can the Soar cognitive architecture be utilized to perform social navigation with a mobile robot? To narrow the scope, the focus is on a particular task within social navigation—human-following. The primary objective is to examine how Soar can be leveraged to manage the complexities and dynamics of such a social environment. Moreover, experiments will be conducted to analyze the performance of the proposed system and approach.

This research focuses on selecting appropriate robot speeds for various social contexts in a human-following task. Social navigation scenarios, such as navigating crowded areas or maintaining a specific distance, require varying speeds to meet social expectations [4] [5]. Soar is used as a decision-maker to manage and select the appropriate speed in these situations, organizing relevant information and evaluating actions based on contextual factors. The action selected by Soar adjusts the maximum allowable speed, which is then reconfigured in a lower-level motion controller. This approach is referred to as high-level control.

This paper makes two key contributions. First, it develops a Soar-based system that encodes social norms, specifically socially acceptable velocities, and uses these norms to determine the appropriate behavior for a given situation. Soar acts as a high-level controller, adjusting a lower-level controller to execute the selected behavior. Second, the paper evaluates the system's performance in a simulation environment for a human-following task. The results demonstrate that using Soar as a high-level controller enhances both task performance—by improving the robot's ability to stay on target—and social appropriateness—by adjusting the robot's speed to fit the context—compared to baseline systems that do not utilize Soar.

The paper is structured as follows: the background section explores "high-level control" and relevant Soar literature. This is followed by a discussion of the Soar-based system design and implementation, then the experimental setup and results. Finally, the paper concludes with a reflection on the findings and approach.

II. Background

This section provides background on the Soar cognitive architecture and its role in decision-making for robotics. It outlines Soar's main functionalities, past applications in robot navigation, and its potential use in high-level control for this work.

A. Soar cognitive architecture

The Soar cognitive architecture is a framework for intelligent decision-making, integrating modules such as short-term and long-term memory, learning mechanisms, and interfaces for communication [2]. Its core component, called symbolic working memory, represents the agent's situational awareness as a symbolic graph, capturing objects, properties, and relations. This memory contains perceptual inputs, goals, reasoning outcomes, and interactions with other Soar modules not utilized in this paper.

Knowledge in Soar is represented in working memory and procedural memory. Working memory provides a dynamic, real-time model of the environment, guiding decisions by maintaining the agent's current understanding of its state. Procedural knowledge is encoded as operators in if-then rules, representing actions that can be taken internally (e.g., memory retrieval, goal-setting) or externally (e.g., motor commands). Preferences stored in preferential memory rank operators based on their suitability, enabling the system to select one when multiple operators are applicable.

Soar's reasoning process is governed by its decision cycle, which iteratively updates working memory and selects actions. In each cycle, operators are proposed based on relevance to the agent's goals, evaluated using preferences of operators, and applied to modify the state or perform an external action. This real-time cycle allows Soar to dynamically perform actions based on new information, supporting flexible and adaptive behavior [6] [7].

B. Soar in Robotics

This research seeks to improve upon previous work in which Soar is applied in robotics. A notable example is the development of a robotic system that uses the Soar cognitive architecture to control unmanned vehicles, as described in Hanford et al. [6]. This project spans four papers [6], [8], [9] and [10], along with related work involving a hexapod robot navigating rough terrain [11] [12] and the use of Soar as a controller of a local motion controller [13], [14].

In these papers, the Soar system is integrated into a robot, referred to as the "Cognitive Robotic System" (CRS), which is used to navigate using GPS while avoiding obstacles. Soar acts as the high-level decision-maker, determining actions such as the direction to move based on procedural rules. The CRS obtains information about the environment through "specialized algorithms" (as defined by Kurup et al. [15]) for state estimation and perception, with later enhancements such as improved perception through occupancy grid mapping [8] and additional sensors [9]. Another related project explored implementing Soar on a hexapod robot, providing better mobility for navigating uneven terrain. Here, Soar controlled the robot's gait based on sensor input, enabling it to reach its target while avoiding obstacles [11], [12].

In a similar implementation, Dang et al. [13] [14] introduced a Soar-based system for a human-following robot. The system integrates an obstacle avoidance algorithm, the dynamic window approach (DWA), to control a mobile robot during human-following tasks. Soar analyzes the robot's current situation and decides on movements such as turning left or right, while the DWA computes optimal velocities for the robot's

motors. The system's effectiveness was evaluated in three scenarios: following a target in a corridor, avoiding obstacles while pursuing the target, and adapting to target loss at an intersection.

C. High-level control

Dang et al. [13], [14] present an approach that uses Soar as a high-level controller to manage a lower-level local motion controller, effectively implementing a "control of control" framework. This methodology leverages a specialized algorithm (the local motion controller) while delegating decision-making responsibilities to a higher-level process. Specifically, in Dang et al.'s work, Soar oversees the local motion controller (DWA), determining whether the robot should turn left, right, or move straight based on the current situation.

This concept of high-level control extends beyond robotics and appears in other disciplines. In neuroscience, Eppinger et al. [16] describe a similar mechanism called metacontrol, which involves monitoring and adjusting control parameters based on task goals and internal or external constraints, as illustrated in Figure 1. Likewise, Pler et al. [17] discuss meta-control in autonomous systems, where an additional control layer manipulates and combines regular controllers.

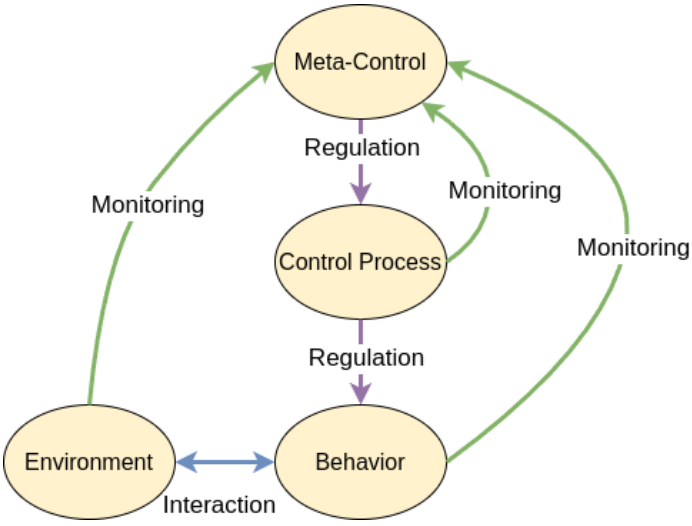

Fig. 1. Components of meta-control. Meta-control involves the monitoring of control parameters, behavioral outcomes and/or environmental features to guide the regulation of control processes (the target of meta-control) in the service of some objective function. A meta-control itself may be guided by the regulation of another meta-control process. Description quoted and figure adapted from Eppinger et al. [16].

To avoid confusion with terminology from other fields, this paper adopts the term high-level control to describe the approach used by Dang et al. Here, high-level control is defined as a framework in which a supervisory process oversees and manages a subordinate process. In both Dang et al.'s work and the system presented in this paper, the lower-level process is a local motion controller, while the higher-level control is implemented using Soar. In this work, Soar makes high-level decisions while leveraging its symbolic working memory to maintain situational awareness, enabling effective and contextually appropriate robot behavior.

D. Relevance to Work

This paper builds on the high-level control approach from Dang et al. [14]. While their system uses Soar to issue explicit navigation commands (e.g., turning left or right), this work integrates Nav2, a more elaborate navigation framework. Nav2 provides a flexible environmental representation through costmaps and can plan entire paths executed by a local motion controller [18].

A key limitation of Dang et al.'s approach is its inability to consider past actions or anticipate future outcomes, leading to less fluid movement. By leveraging Nav2's planning and execution capabilities, this work addresses these shortcomings while ensuring navigation aligns with social norms. Instead of using Soar for low-level directional control, it functions as a high-level supervisor, adjusting Nav2's local motion planner to regulate velocity based on social context. Specifically, Soar configures the maximum velocity of Nav2's controller to promote safe and comfortable movement in human environments.

III. Social Navigation using Soar

The high-level control system presented in this paper focuses on context-based navigation, meaning that Soar evaluates the robot's current situation to determine the most appropriate action.

The rationale for adopting a high-level control approach lies in its ability to address limitations in existing social navigation robotic systems [1]. While systems such as motion controllers excel at solving specific problems, they may not be optimally configured for varying social contexts. For instance, a robot's maximum speed should adapt to the environment—slowing down in crowded areas and speeding up in open spaces. High-level control leverages the capabilities of these specialized systems while dynamically adjusting parameters (like maximum speed) based on the situation.

This approach improves upon the method used in Dang et al. [14], where the high-level control system directly decided specific motion actions (e.g., turning left or right). Such direct intervention in the lower-level controller reduced its effectiveness, preventing smooth and continuous movement. In contrast, the high-level control system in this work focuses on adaptive reconfiguration, ensuring that the lower-level system performs optimally across diverse scenarios.

This research focuses on a key aspect of social navigation: selecting appropriate speeds based on social context, which is essential for both safety and perceived comfort [5]. There are various social expectations and norms that influence how robots should move in different contexts. Consider, for example, how humans are often uncomfortable when robots travel at speeds around 1 m/s [4]. Paradoxically, this is also the speed at which humans tend to walk comfortably [19]. In a human-following task, this discomfort can lead to frustration when the robot moves too slowly, forcing the human to either slow down or wait for the robot to catch up. However, even

when the robot can safely increase its speed—such as when maintaining a large following distance—it must still consider other factors in the environment. For instance, navigating through a crowded area or over an intersection might require slower, more cautious movements regardless of distance to the human.

Ultimately, different scenarios demand different speed adaptations. Managing situational information and adjusting behavior accordingly is where Soar can be valuable. In this implementation, Soar's working memory symbolically encodes relevant context, while behaviors and social norms related to speed (e.g., slowing down in crowds or accelerating to reduce distance) are represented as actions that Soar can evaluate and select. The implementation positions Soar as a high-level controller, abstracting social situations and norms into symbolic representations and reconfiguring a lower-level motion controller by dynamically adjusting its maximum velocity based on the selected action to ensure behavior aligns with the context.

This method offers advantages over conventional "baseline" approaches, where a baseline system merely continuously updates the position of a human to follow without accounting for contextual factors.

System Description

The system comprises several components that collectively form the robot's architecture, including perception, decision-making using Soar, and navigation control, as illustrated in Figure 2. This section outlines the system's rules, preferences, possible states, and the integration between Soar and the navigation stack (Nav2).

It is important to note that the system presented here is, in many respects, a conceptual prototype. The specific choices made regarding perception, decision-making rules, output, and state representation are not definitive; other configurations could be equally valid. The approach was designed to reflect commonly observed social norms and to provide a symbolic representation of these norms. However, this paper does not assert that the chosen configuration is the most optimal or effective. Rather, it serves as a proof of concept, demonstrating how the Soar cognitive architecture can be applied to social navigation tasks.

The specific social norms and quantitative values used in this system were derived from social navigation research by Butler et al. [4], Gao et al. [20] and Samarakoon et al. [21]. These values provide a foundation for further exploration but are not intended as final. Developing a truly optimal system would require more extensive research to identify which behaviors and types of information are critical for making decisions in social navigation. For example, further investigation is needed to determine which social norms regarding speed—or other parameters not addressed in this paper—are relevant and under what circumstances they apply. Once identified, these norms could be systematically encoded into Soar, enabling its use as a high-level controller in a more refined and effective manner.

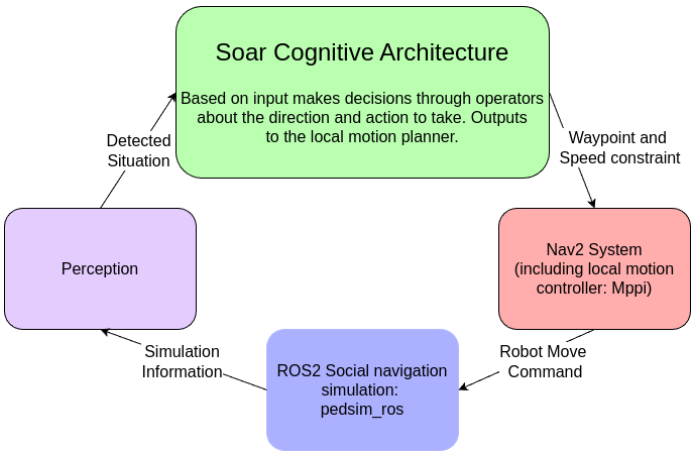

Fig. 2. System Architecture. With the various modules in the boxes, such as the perception module, the Soar system, Nav2 and the robot simulation in gazebo alongside pedsim_ros. The arrows map information and control data, such as the decisions Soar makes, the move commands from Nav2 and simulation and perception information.

A. Perception

The perception system gathers data from the environment and converts it into structured input for Soar, as illustrated in purple in Figure 2. The input is represented using the following key elements, which are put in the state graph, as represented in figure 3.

1) Topology of the Environment (T) The robot's location is classified into discrete categories:

$$T \in \{\text{open space}, \text{corridor}, \text{nearing intersection}, \text{on intersection}\}$$

These regions are predefined (hardcoded) since modeling the spatial layout dynamically is outside the scope of this paper.

2) Density of the Environment (D) The density of humans around the robot is calculated within a radius of $R = 5\,\text{m}$. The density classification is given by:

$$D = \begin{cases} \text{crowded}, & \text{if } n_{\text{humans}} > 2 \\ \text{sparse}, & \text{otherwise} \end{cases}$$

where n_{humans} is the number of humans within the radius R.

3) Agent Information (A_i) Each agent i in the environment is tracked with the following attributes:

$$A_i = \{\text{distance to robot } d_i, \text{interaction type } I_i\}$$

The interaction type I_i is classified as:

$$I_i \in \{\text{following}, \text{arrived}, \text{crossing}\}$$

B. Soar System

As shown in Figure 2, the detection results from the perception module are mapped to the Soar system using the Soar Python interface. The Soar system then determines the appropriate action, which is subsequently sent to Nav2 via the same interface.

1) State Representation: In Soar, perception information is represented symbolically within working memory using a graph-based ontology. This ontology allows for the structured definition and organization of information. In this work, it is used to represent the robot's current state, which is updated during each decision cycle. The system tracks and utilizes multiple types of information, including details about the environment and the agents within it, as illustrated in Figure 3. This information is derived from the perception detections described in the previous section. Using the Soar Python interface (pysoarlib), these detections are mapped into the graph structure, ensuring they are available for decision-making within Soar.

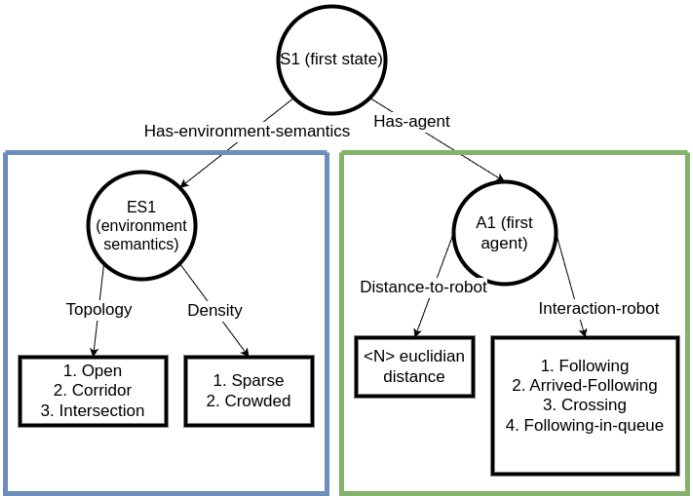

Fig. 3. State representation of Soar system, with options enumerated in the blocks for certain relations, displaying the information present with which decisions are made

- Environment Representation: The environment's state, referring to the real material world the robot is in, is represented through the relation "has-environment-semantics," which links to an object that holds information about the density and topology of the environment. For example, the state represents whether the robot is in a crowded area or in a corridor.
- Agents detected in the environment are tracked using the relation "has-agent." Each agent object contains: 1. Distance to the robot, 2. Interaction type (e.g., following, crossing) and 3. The agent's assigned ID. Even though they are also technically part of the environment, each agent receives a separate node linked to the state root node, as can be seen in figure 3, for a more user-friendly interface for procedural rule creation (next subsection).

2) Procedural memory: In Soar, procedural memory encodes the possible behaviors the robot can exhibit. These behaviors are defined using *if-then* rules, enabling the system to dynamically select appropriate actions based on the current environmental context. This ensures that the robot's behavior aligns with social norms.

Each *if-then* rule specifies an action, or operator, to execute when certain conditions are met. The *if* part evaluates the robot's state, as described in the *State Representation* section, and incorporates general sociability rules [20]. For instance, proxemic distances—such as the boundary between the social and public zones at 4 meters—shape rules about when the robot should adjust its speed.

The *then* part of the rule generates two outputs:

1) Waypoint: The target location for the robot to navigate to.
2) Speed Behavior: The speed the robot should maintain while navigating.

An overview of the possible operators is provided in Table I, alongside which social norm it represents.

This work primarily focuses on speed behavior, with procedural rules based on research on human comfort and proxemics. According to Butler et al. [4], humans perceive a robot moving at 0.4 m/s as comfortable, while speeds exceeding 1 m/s are often considered uncomfortable. However, Giannoulaki et al. [19] notes that humans themselves tend to walk at speeds exceeding 1 m/s. To balance these considerations, the system employs the following speed rules:

- In situations requiring human comfort, the speed is limited to 0.4 m/s.
- In less socially sensitive contexts, the robot is allowed to move at a maximum speed of 1 m/s.

Waypoint selection is guided by proxemic principles [21]. During normal operation, the robot maintains a distance of 1.2 meters behind the human—on the border between personal and social space. This ensures the robot stays close enough to follow effectively without invading personal space.

Special cases include:

- Queueing: The robot follows at a closer distance of 0.8 meters to align with social norms about spacing in queues.
- Arriving: When the human stops, the robot moves to a position in front of the human, allowing it to pause operations.

The system supports four distinct operational modes, as depicted in the actions in Table I:

1) Normal Operations: The waypoint is set 1.2 meters behind the human, and the maximum speed is 1 m/s.
2) Slowed Operations: The waypoint remains 1.2 meters behind the human, but the maximum speed is reduced to 0.4 m/s.
3) Queueing: The waypoint is set 0.8 meters behind the human, with a maximum speed of 0.4 m/s.
4) Paused: The waypoint shifts to a position in front of the human, and the robot moves at 0.4 m/s until it reaches the waypoint, at which point it pauses.

Preferential memory: When multiple actions (operators) are available for a given state, the system must decide which operator to execute. This decision is made by comparing the preferences between operators. In preferential memory, each action is ranked based on predefined priorities to ensure that

TABLE I
OVERVIEW OF SOAR OPERATORS FOR SOCIAL NAVIGATION

Operator Name	Prerequisite (Condition)	Action (Behaviour)	Explanation (Social norm)
pursuit	agent distance $>$ 4; topology = open	Normal operations	Increase speed to close the gap when the human is far away in an open area.
follow	agent distance $<$ 4; topology = open	Slowed operations	Reduce speed when close to the human in an open area.
corridor-pursuit	agent distance $>$ 4; topology = corridor	Normal operations	Increase speed when the human is far away in a corridor.
corridor-follow	agent distance $<$ 4; topology = corridor	Slowed operations	Reduce speed when close to the human in a corridor.
intersection	topology = intersection	Slowed operations	Slow down when approaching an intersection.
intersection-human	topology = in-intersection	Slowed operations	Reduce speed if a human is crossing at an intersection.
arrived	Agent interaction = arrived	Paused	Move slowly toward the human after arrival.
crowded-environment	environment density = crowded	Slowed operations	Reduce speed in a crowded environment.
queueing	Interaction = in queue	Queueing	Line up behind the human in a queue.

This table provides an overview of the possible operators in the Soar system. Each row includes:

- Operator Name: The action the robot can take.
- Prerequisite (if condition): The condition in the current state that must be true for the operator to be considered.
- Action (Behaviour): The output that defines how the robot will behave in response to the condition.

the most suitable behavior is chosen for the robot's context. Each of these preferences must be specified by the creator of the system. The preferences for this system are based on whether an action is more socially appropriate. For example, it is more socially appropriate for a robot to drive slowly in a crowded environment rather than speed up.

In table I all possible actions are compared against one another, with the desired preferences depicted by 1. $>$ meaning it prefers the operator in the first column over the one in the other column 2. $<$ for the other way around 3. $=$ when it is indifferent.

The table also includes numerous X's, indicating pairs of operators that cannot be proposed simultaneously due to the structure of the state representation. As the number of operators grows, manually defining preferences for each possible combination can become labor-intensive. The table highlights how exhaustive this process can be, with the amount of entries that needs be looked at growing with $\sum_{n=1}(n-1)$ for n operators. Which is a limitation of the approach presented in this paper.

C. Nav2

The final component of the system is Nav2, which is responsible for executing the robot's navigation commands. By utilizing the Nav2 Python API, The Soar python interface and ROS2, the robot receives commands from Soar regarding:

- Where to navigate: The target waypoint determined by the Soar system.
- Maximum allowable speed: The speed at which the robot should navigate based on the current context.

Which are then fed to Nav2. Nav2 then handles the rest of the navigation process, such as localization, costmap representation and local motion control. The costmap includes a social costmap plugin, which accounts for the proxemics of nearby people to ensure safe and comfortable navigation around them. The local motion control is managed by the Model Predictive Path Integral Control (MPPI), a variant of model predictive control (MPC). MPPI is particularly advantageous because it optimizes the robot's path by considering future trajectory costs, enabling smoother, more predictive motion that adjusts to dynamic environments [22].

IV. EXPERIMENT AND RESULTS

A. Test Description

The primary objective of the experiments is to evaluate whether using Soar as a high-level controller enhances social navigation. This is investigated through a human-following task conducted in simulation. To measure the system performance, various social navigation metrics will be analyzed, including the robot's ability to maintain proximity to the human and whether it adjusts its velocity appropriately during the task. To provide a basis for comparison, a baseline system is included in the tests. This baseline consists of the default Nav2 system using the dynamic object follower plugin, configured only to receive waypoints leading to the human's location. The baseline operates with a single pre-defined maximum speed, serving as a straightforward alternative to the Soar-controlled system.

Simulation Environment: The Pedestrian Simulator (pedsim_ros) is used as the simulation environment to evaluate the navigation performance of the Soar agent [23]. This simulator provides a variety of realistic scenarios, such as office or café settings, populated by a limited number of pedestrians. The pedestrian behavior is modeled using the Social Force Model, which simulates their movement based on attractive forces guiding them toward their goals and repulsive forces keeping them away from obstacles and other agents [24]. By calculating the net forces acting on each pedestrian, the model determines their movement direction and velocity.

The mobile robot used in the simulation is Tiago, developed by PAL Robotics. Tiago is equipped with a LiDAR sensor, which is utilized for localization tasks. This setup ensures that

TABLE II
DECISION MATRIX FOR ROBOT BEHAVIOR

	Pursuit	**Follow**	**Corridor-Pursuit**	**Corridor-Follow**	**Intersection**	**Intersection-Human**	**Arrived**	**Crowded-Environment**	**Queueing**
Pursuit	-	X	X	X	X	X	X	$<$	X
Follow		-	X	X	X	X	X	=	X
Corridor-Pursuit			-	X	X	X	X	$<$	X
Corridor-Follow				-	X	X	X	=	X
Intersection					-	X	X	=	X
Intersection-Human						-	X	=	X
Arrived							-	$>$	X
Crowded-Environment								-	X

This table illustrates the preferences between operators in Soar's decision-making process. These preferences are necessary when multiple operators are proposed during a decision cycle. For instance, if the robot's current state indicates a large distance and a crowded environment, Soar may propose two operators: "Pursuit" and "Crowded-environment." To resolve this, Soar relies on the table, which specifies that "Pursuit" $<$ (worse than) "Crowded-environment." Consequently, Soar selects the "Crowded-environment" operator.
The table also contains many **"X"** entries, indicating combinations of operators that cannot occur simultaneously (e.g., "Pursuit" implies a large distance to the human, whereas "Follow" implies a small distance, and thus can't be proposed at the same time). Additionally, **"-"** denotes trivial comparisons, such as comparing an operator with itself (e.g., "Pursuit" against "Pursuit").

the simulation provides a realistic testbed for evaluating the robot's social navigation capabilities.

Scenario: This study focuses on a human-following task in an office environment, where the robot must navigate between different areas while interacting with human agents. The goal is to navigate effectively, defined as successfully reaching the destination while adhering to social norms, such as maintaining appropriate speeds based on the context. Figure 4 illustrates the map used in the experiments, depicting the predefined topology of the environment (e.g., intersections and corridors) and the paths taken by both the human agent and the robot. The map is divided into four distinct sections, each representing a specific scenario:

1) Queuing Zone: In the first section, the robot navigates a zone where humans are queuing. The robot must queue behind the human while maintaining a socially appropriate distance.
2) Corridor and Intersection Navigation: In the second section, the robot follows the human through a corridor, navigating intersections along the way.
3) Path Interference: The third section involves the robot following a human whose path intersects with stationary humans. The robot must navigate around these stationary agents while maintaining its path.
4) Open Area: The final section is an open area populated with moving human agents. These agents may cross paths with the robot, walk in the same direction, or approach from the opposite side. Two variations of this scenario are tested:
    a) Scenario 4: Five humans are present in the area.
    b) Scenario 5: Two humans are present in the area.

For each section, the human follows a predefined path (indicated by the blue line on the map in figure 4), while the robot is tasked with following the human. Additionally, an experiment is conducted across the entire map, referred to as Scenario 0, to assess the performance of the systems in a more complex, longer-duration scenario. This setup allows for evaluating the robot's behavior under varying conditions and complexities, providing insights into its social navigation capabilities.

Systems: To evaluate the effectiveness of the Soar system, a baseline comparison is used. The Soar system is designed to dynamically adjust the robot's maximum speed based on the social context, ensuring that the robot maintains appropriate behavior in different situations. In contrast, the baseline systems will operate with a fixed maximum speed configuration, providing a meaningful comparison between dynamic and static maximum speed settings.

Given that the Soar system alternates between two maximum speed configurations—1.0 m/s for faster navigation and 0.4 m/s for socially sensitive navigation—two baseline systems are defined, each using one of these fixed configurations. This results in three systems for the experiments:

1) Nav2 Baseline (Adhering to Social Norms): The robot operates with a fixed maximum speed of 0.4 m/s, prioritizing social sensitivity and human comfort.
2) Nav2 Baseline (High-Speed Configuration): The robot operates with a fixed maximum speed of 1.0 m/s, focusing on effective navigation and maintaining pace with the human.
3) Soar High-Level Control: The robot dynamically adjusts its maximum speed, selecting between 0.4 m/s and 1.0 m/s based on the social situation.

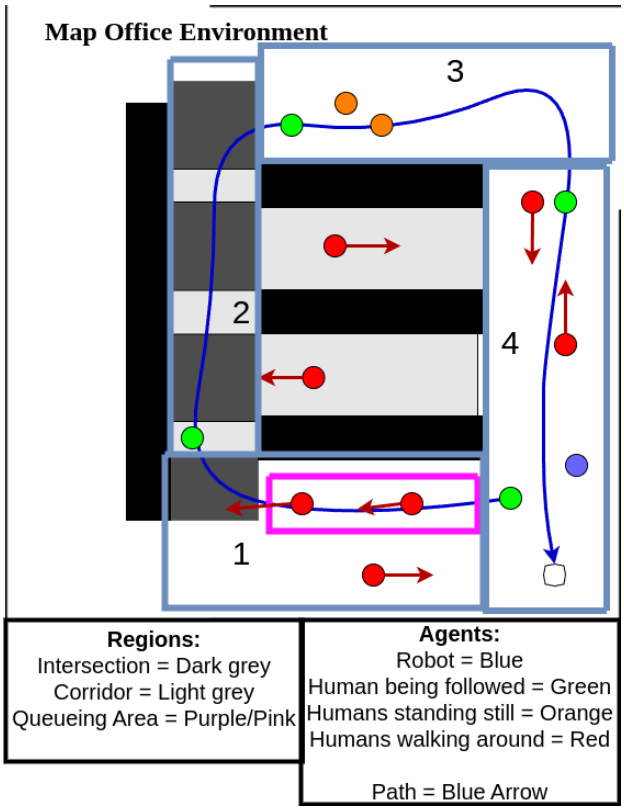

Fig. 4. Map of the office environment, including the scenario with predefined topology and where the agents are moving

Each system will be tested across all scenarios described previously. By analyzing the performance metrics, the differences between the systems can be assessed, providing insights into the benefits and limitations of dynamic speed adjustment versus fixed configurations.

Metrics: The evaluation of the systems will involve three categories of metrics: effectiveness, "socialness", and efficiency [25]. Each category focuses on a specific aspect of the robot's behavior and performance, as outlined below:

1) Effectiveness measures how well the robot performs its task of following the human to their destination. This will be assessed using the following metrics:
   - Distance to Target: The average distance at which the robot maintains its position relative to the human throughout the task. A smaller, consistent distance indicates better performance.
   - Time to Complete the Run: The total time taken for the human to reach their target and for the robot to settle into its final position in front of the human. Faster completion times indicate better navigation efficiency, provided that social norms are respected.
2) "Socialness" evaluates whether the robot adheres to socially acceptable behaviors, particularly speed norms. These metrics include:
   - *Magnitude of Speed Violations*: Measures the extent to which the robot's speed exceeds or falls below the socially appropriate speed. For example, if the expected speed is 0.4 m/s and the robot drives at an average speed of 0.6 m/s, the violation magnitude is 0.2 m/s.
   - *Relative Duration of Speed Violations*: The percentage of time the robot spends violating the speed norms relative to the total duration during which it is expected to adhere to those norms.
3) Efficiency examines the smoothness and control effort of the robot's movements, focusing on its acceleration and jerk (rate of change of acceleration). The metrics are as follows:
   - *Energy of the Signal*: Calculated as the squared sum of acceleration and jerk over the duration of the task. This metric reflects the intensity of the robot's control effort and movement smoothness. Lower energy values indicate smoother, more efficient movements.

   The rationale for using energy rather than averages is to emphasize the impact of sudden, large changes in movement (jerkiness). Excessive jerk not only implies inefficiency but also negatively affects perceived socialness, as humans tend to find unpredictable movements uncomfortable.

Results

To begin, examples of the baseline experiments are presented to provide an initial understanding of how these systems perform. Later, the quantified metrics will be analyzed in detail. Figures 5, 6, 7, and 8 illustrate the performance of the baseline systems for the scenario in which the robot navigates through all sections of the map (i.e., the entire environment).

In Figure 5, the robot's distance to the human and its speed are shown for the baseline system configured with a socially sensitive maximum speed. The graph shows that the robot consistently maintains a velocity below the maximum speed. This system exhibits "social" behavior, as the robot adheres to a speed considered comfortable by humans. Similarly, the acceleration and jerk of the robot are relatively low and smooth, as demonstrated in Figure 6, contributing to a predictable and less erratic movement.

However, the distance-to-human graph reveals a significant limitation: the robot struggles to keep up with the human. In a real-world scenario, this would likely result in the robot losing track of the human, or the human having to frequently stop and wait for the robot to catch up. Notably, the human is already walking at a relatively slow speed of 0.3 m/s—a pace that most humans find inconveniently slow for sustained walking, with a preferred walking speed closer to 1 m/s.

The second baseline system allows the robot to drive at a maximum speed of 1 m/s. Figure 7 illustrates a different type of performance compared to the socially sensitive baseline. The graph shows that Nav2 is relatively effective in maintaining a close distance to the human, with an average distance of approximately 3 m ± 1.0 m, as verified by two additional runs with similar results. This distance keeps the robot within

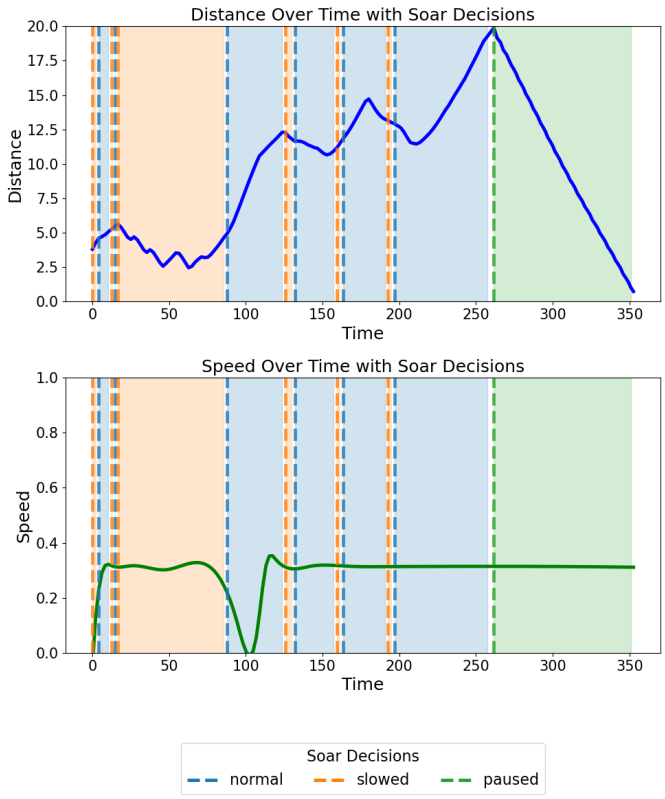

Fig. 5. Baseline run in Simulation with low maximum speed of robot (0.4 m/s), showing the robot's speed and the distance to the human

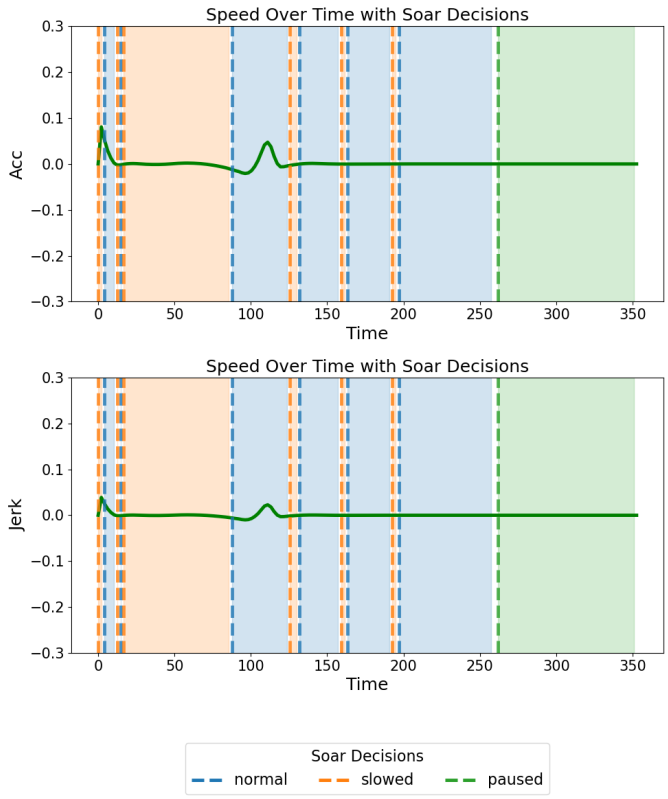

Fig. 6. Baseline run in Simulation with low maximum speed of robot (0.4 m/s), showing the robot's acceleration and jerk

the "social zone" as defined by proxemics (1.2 m $<$ distance $<$ 3.6 m), which is considered an appropriate and comfortable range for interaction [26].

However, this configuration sacrifices adherence to social speed norms, as the robot's speed often exceeds the socially appropriate maximum of 0.4 m/s. Additionally, Figure 8 reveals that the robot's acceleration and jerk are more erratic, indicating less smooth and predictable movement compared to the socially sensitive baseline.

In all the figures, the desired speed configurations as determined by Soar are also shown. While these configurations do not influence the baseline Nav2 systems, they serve as a reference to compare how Soar would have acted under the same circumstances. The color coding corresponds to specific speed configurations: "Normal" for a high maximum speed of 1 m/s, "Slowed" for a low maximum speed of 0.4 m/s, and "Paused" for the scenario where the human has arrived, and the robot is navigating to a waypoint in front of the human.

Figures 9 and 10 illustrate runs where Soar's decisions actively influence the system. In these figures, the regions where Soar makes decisions are highlighted, showing its adjustments through reconfiguration of the maximum speed. These adjustments are reflected in the graphs: the robot's speed decreases in response to situations requiring safer behavior. For instance, the robot slows down when it gets too close to the human ("follow" decision) or when it approaches an intersection with poor visibility ("intersection" decision). Soar translates these decisions into reconfiguration actions, selecting the "slowed" configuration, which enforces a lower maximum speed of 0.4 m/s.

One noticeable difference in the Soar-controlled system is that the distance between the robot and the human tends to be greater compared to the high-speed baseline system. In the high-speed baseline, Nav2 operates without speed constraints, allowing it to close the distance to the human more effectively. Conversely, in the Soar system, the robot occasionally closes the distance, but only when the situation permits. This trade-off highlights the Soar system's ability to adhere to socially appropriate speeds when required, even though it sacrifices some effectiveness in maintaining proximity.

Another key observation is the abrupt speed changes seen in the Soar system. Within a single decision cycle, the robot's speed can increase from around 0.4 m/s to 0.8 m/s when Soar permits a higher speed. This behavior results in significant peaks in acceleration and jerk, which are undesirable from both an efficiency and socialness perspective. High acceleration and jerky movements not only demand greater energy but also make the robot's behavior less predictable and comfortable for humans interacting with it.

Metrics Results: In Table III, the performance metrics for each scenario and system are presented. Scenario 0 refers to the complete run over the entire map, while Scenarios 1 to

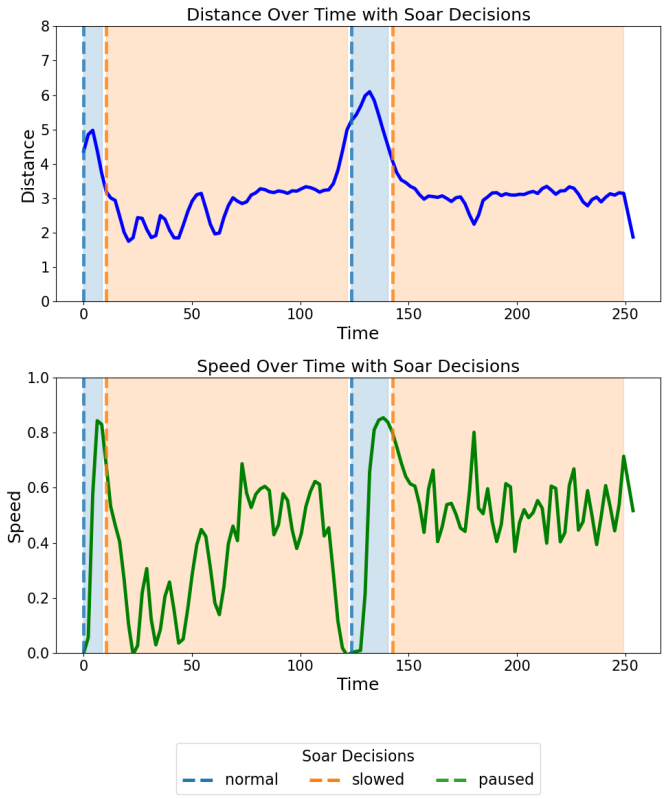

Fig. 7. Baseline run in Simulation with high maximum speed of robot (1.0 m/s), showing the robot's speed and the distance to the human

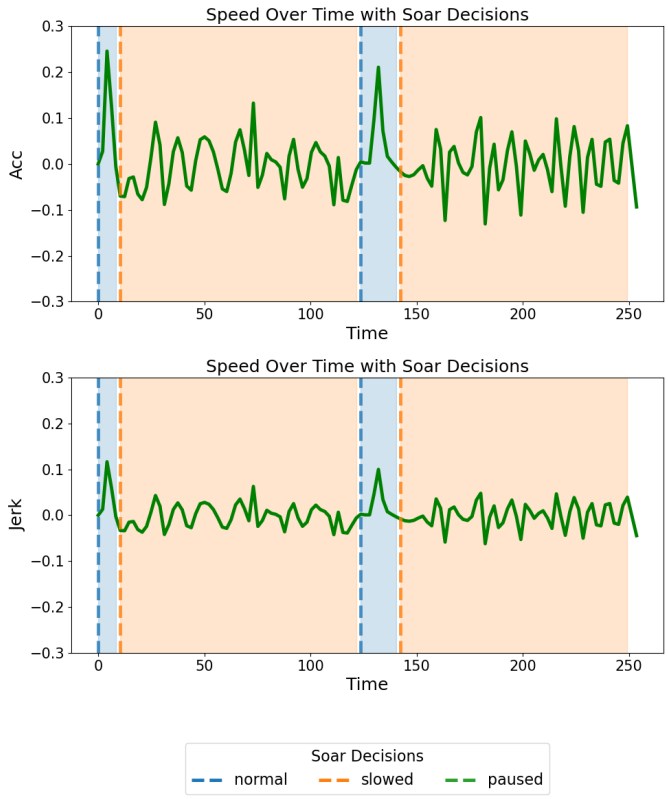

Fig. 8. Baseline run in Simulation with high maximum speed of robot (1.0 m/s), showing the robot's acceleration and jerk

4 correspond to the sections of the map described earlier, as shown in Figure 4. The last scenario (Scenario 5) also takes place in Section 4 but involves fewer people for the robot to navigate around. The experiments are conducted with three different systems: *baseline_slow*, *baseline_fast*, and *Soar*. Each experiment consists of three runs, and the reported metrics represent the average performance over those runs.

1) Distance on Target: This metric measures the robot's distance to the human. Both the *Soar* and *baseline_fast* systems perform similarly, maintaining a distance of approximately 2.5 to 3.5 meters from the human. This distance falls within the "social zone" as defined by proxemics [26]. The *baseline_slow* system, however, struggles to keep up with the human, particularly in longer runs such as Scenario 0. This trend is consistent across all scenarios, highlighting the difficulty of the *baseline_slow* configuration in maintaining proximity.
2) Task Completion Time: The time required to complete tasks is significantly longer for the *baseline_slow* configuration due to its restricted speed. In contrast, both the *baseline_fast* and *Soar* systems complete tasks more quickly, reducing the time the human needs to wait for the robot. The completion times of these two configurations are approximately the same, demonstrating their efficiency compared to *baseline_slow*.
3) Speed Violation: This metric measures how much the robot exceeds the "ideal social speed limit." The *baseline_slow* system does not violate speed limits, as its configuration prevents it from exceeding them. However, the *baseline_fast* system violates the limit by an average of 0.2 m/s, with peaks reaching up to 0.4 m/s, as shown in Figure 7. The *Soar* system exhibits similar average violations, primarily due to transitional periods when it adjusts its speed after reconfiguring the maximum allowed speed. These adjustments are also visible in Figure 7.
4) Duration of Speed Violations: This metric, expressed as a percentage of the total duration, indicates how long the robot exceeds the speed limit. The *Soar* system shows minimal or no speed violations, with violations occurring only during short transitional periods. In contrast, the *baseline_fast* system violates the speed limit for the majority of the duration (over 60%), except in Scenario 1, where the robot is in a queue, reducing the need for high speeds.
5) Acceleration Energy: This metric quantifies the intensity of the robot's accelerations. As shown in Table III, the *Soar* and *baseline_fast* systems exhibit similar levels of acceleration energy, though this varies slightly across specific scenarios. However, the *Soar* system has a lower

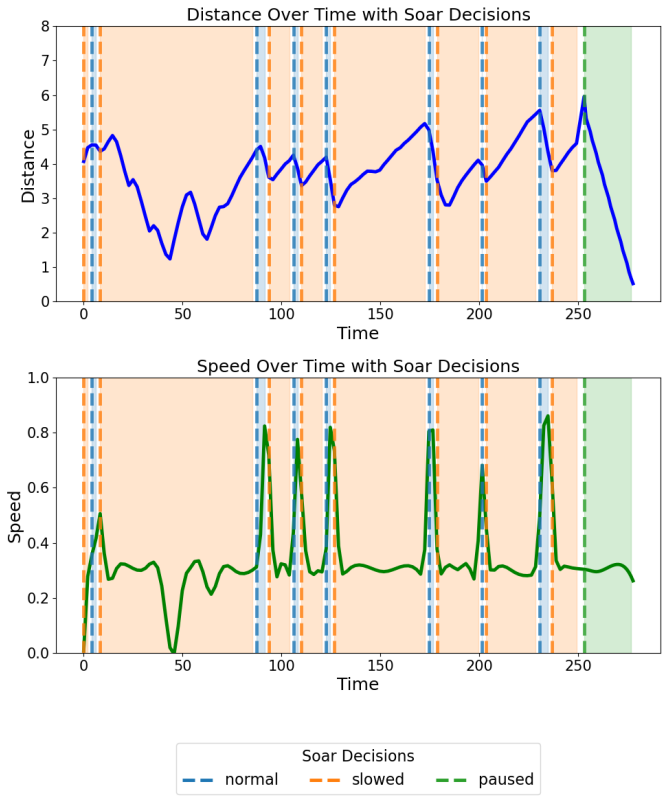

Fig. 9. Run where Soar reconfigures the maximum allowed speed, showing the robot's speed and the distance to the human

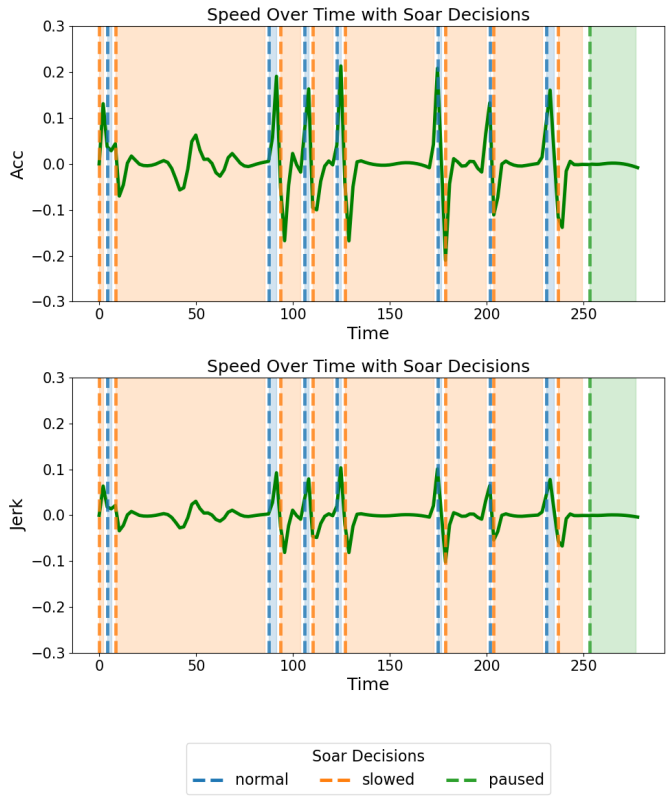

Fig. 10. Run where Soar reconfigures the maximum allowed speed, showing the robot's acceleration and jerk

average velocity, indicating that despite similar energy levels, the sources of this energy differ. The *Soar* system experiences high acceleration energy due to intense acceleration peaks during configuration changes, whereas the *baseline_fast* system maintains higher acceleration energy because of its consistently higher velocity. This suggests that the *Soar* system is likely less efficient than the *baseline_fast* system.

6) Jerk Energy: The jerk energy metric, which measures the abruptness of motion, exhibits trends similar to the acceleration energy. Figures 8 and 10 show that both the *baseline_fast* and *Soar* systems have moments of abrupt and non-smooth motion. These jerky movements can negatively impact the robot's social acceptability by making its behavior less comfortable for humans nearby.

V. Discussion

A. Reflection Results

The results demonstrate that the *Soar* system operates as intended, with *Soar* making decisions about the maximum allowed speed and the robot adjusting its behavior accordingly. This approach uses high-level control to dynamically reconfigure the robot's maximum speed based on contextual input, unlike conventional navigation systems such as the baseline system, where a single maximum allowed speed is predefined and remains unchanged.

As can be seen in the metrics results, the *Soar* system addresses a gap that neither baseline system can resolve alone. The *baseline_slow* system, with the maximum allowed speed set to a low value (0.4 m/s), is able to drive socially, restricting its speed to a level that is considered comfortable by humans. However, this system often struggles to keep up with the human, requiring the human to either wait for the robot or for the robot to get lost. On the other hand, the *baseline_fast* system, with a higher maximum speed (1 m/s), is better at closing the distance more quickly. However, it does not take the social aspect into account, mainly disregarding whether humans are comfortable with the robot's speed.

The *Soar* system, utilizing high-level control, combines the advantages of both baseline systems. It allows for socially appropriate speeds while also maintaining the ability to effectively close the distance when necessary. However, there are limitations. While the *Soar* system keeps up with the target and finishes tasks more quickly compared to the *baseline_slow* system, it still lags slightly behind the *baseline_fast* system in terms of maintaining distance and completion time, which is a trade-off for ensuring socially appropriate behavior.

Furthermore, there are limitations regarding the acceleration and jerk, which tend to be quite intense when the maximum speed configuration is abruptly changed. This erratic and intense behavior is uncomfortable for humans, as it makes the robot seem unpredictable, potentially endangering them.

TABLE III
PERFORMANCE RESULTS FOR EXPERIMENTS

experiment	distance to target (m)	duration run (s)	speed violation (m/s)	duration violation $(\%)$	acceleration energy $(m/s^2)^2$	jerk energy $(m/s^3)^2$
scenario 0, system baseline_slow	8.8	346	0.0	0	0.024	0.005
scenario 0, system baseline_fast	3.6	265	0.184	64	0.588	0.136
scenario 0, system soar	3.4	272	0.141	4	0.434	0.102
scenario 1, system baseline_slow	3.3	86	0.0	0	0.005	0.001
scenario 1, system baseline_fast	2.7	77	0.162	25	0.17	0.039
scenario 1, system soar	2.2	92	0.0	1	0.116	0.028
scenario 2, system baseline_slow	3.9	83	0.0	0	0.003	0.001
scenario 2, system baseline_fast	2.7	55	0.18	82	0.071	0.016
scenario 2, system soar	2.9	64	0.0	6	0.046	0.011
scenario 3, system baseline_slow	4.0	84	0.0	0	0.001	0
scenario 3, system baseline_fast	2.8	58	0.161	87	0.071	0.016
scenario 3, system soar	3.3	72	0.129	4	0.042	0.01
scenario 4, system baseline_slow	5.7	119	0.0	0	0.001	0
scenario 4, experiment baseline_fast	2.9	77	0.148	92	0.074	0.017
scenario 4, experiment soar	3.3	88	0.129	9	0.117	0.027
scenario 5, experiment baseline_slow	4.6	104	0.0	0	0.005	0.001
scenario 5, experiment baseline_fast	2.7	77	0.077	60	0.064	0.015
scenario 5, experiment soar	3.2	87	0.183	5	0.091	0.021

Additionally, this type of behavior is energy inefficient, as high acceleration peaks tend to lead to wasted energy. This inefficiency could accumulate in more complex environments where configuration changes occur more frequently.

Future work should focus on addressing these issues, either by refining the *Soar* high-level controller to account for past and future events to make smoother decisions or by improving the local motion controller to constrain the acceleration, resulting in more predictable and energy-efficient motion.

Moreover, there are a few other limitations to consider. One major limitation is the reliance on the simulation. In these tests, the robot uses ground truth data to detect agents, directly extracting their actual positions from the simulation. This ensures perfect observability but is unrealistic for real-world scenarios. For future work, it will be essential to implement and evaluate the Soar system using a real detection system to assess its performance when dealing with detection failures or inaccuracies.

Additionally, the agent behavior in the simulation is less dynamic than what might be expected in a real-world environment. In practice, a human might slow down in crowded or sensitive areas, and might also be more mindful of staying close to the robot. In contrast, the simulated agent moves at a constant pace, which do not reflect realistic interactions. Therefore, another key area for future work will involve validating this system in real-life settings. This will allow for a thorough examination of the complexities of human-robot interaction, providing insights into how the system performs when interacting with real humans in more unpredictable and dynamic environments.

Lastly, future work could refine the specific social norms and contextual factors encoded into Soar. Determining which aspects of social behavior are most relevant for high-level control remains an open question. Identifying additional parameters that could be meaningfully controlled—such as following distance in specific scenarios, acceleration rates, etc.—would help improve adherence to human expectations. Addressing these considerations would require deeper study of human-robot interaction literature to ensure that the system aligns with realistic social dynamics and enhances the robot's ability

to operate effectively in human environments.

B. Reflection on System

Now that the high-level control concept and its implementation using *Soar* have been explained, along with results evaluating the system's performance, this section reflects on the system as a whole and explores the reasoning behind its design and implementation choices. Specifically, why was *Soar* chosen for high-level control? How does *Soar* compare to other systems, such as Finite State Machines (FSMs), in this context?

Various systems can handle reconfiguration and decision-making, but *Soar* distinguishes itself in one critical aspect: its ability to store and organize the environmental information and desired high-level control behaviors.

To understand these aspects better, let's compare Soar with a more traditional system like an FSM. While it may seem unfair to compare a cognitive architecture like Soar with an FSM, which is a simpler decision-making tool, this comparison is useful because FSMs are a common choice for robotic decision-making.

Knowledge Representation: Soar's procedural rules offer a flexible and scalable means of encoding human-like decision-making logic. These rules can vary from highly specific (for narrowly defined situations) to general (applicable across different contexts), allowing for an expansive rule set that remains organized and scalable. For instance, in Jones et al. [27] Soar was used to develop an intelligent pilot for combat flight simulation, using over 5,200 production rules and 140 operators for managing complex aircraft maneuvers. This example illustrated Soar's ability to manage vast amounts of knowledge while maintaining flexibility.

In contrast, an FSM is far more rigid in its structure. States and transitions must be predefined, and as the system grows in complexity, the FSM tends to suffer from state explosion, where the number of states and transitions grows exponentially [3]. Each new scenario or context-sensitive decision requires additional states or more complex transitions, making the system difficult to manage.

Moreover, FSMs lack a standardized way to store knowledge. Their "knowledge" is implicit in the state transitions, without an explicit mechanism for rules governing transitions. Soar, on the other hand, separates knowledge (encoded in the state representation and production rules) from the state transitions (constrained by decision cycle). This allows Soar to dynamically adapt to new contexts without adding more states, giving it much more flexibility in decision-making.

When implementing high-level control using an FSM, its lack of flexibility becomes evident. Context-sensitive decisions require the FSM to introduce additional states and transitions for every possible scenario. This results in a bloated state diagram that is cumbersome to manage and prone to errors. In contrast, *Soar* handles such reconfigurations more efficiently by allowing the addition or modification of rules without requiring a complete redefinition of the state structure.

An example of this is illustrated in Figure 11, which compares the high-level control concept implemented in *Soar* with its equivalent finite state machine (FSM). On the left, the state representation and procedural rules of *Soar* are shown in simplified form, utilizing only two rules. On the right, the corresponding FSM is displayed, where transitions correspond to the "if" conditions in the production rules of *Soar*, and the states represent the "then" actions.

The FSM also incorporates numbered priorities for statements that can be true simultaneously, with lower numbers (starting at 0) indicating higher priority. Conversely, *Soar* employs preferential memory to define preferences, enabling deliberation over the best decision. The figure further highlights what happens when a new rule is added: in *Soar*, the new rule is seamlessly integrated, whereas the FSM requires additional transitions and possibly states, leading to significant growth.

In the worst-case scenario, for n procedural rules, the number of transitions in the FSM tends to grow quadratically (n^2), while the number of states grows linearly (n). This demonstrates *Soar*'s scalability compared to the FSM, which quickly becomes unwieldy due to the need to account for transitions between all states. However, it is worth noting that certain scenarios may still favor implementing the high-level control concept in an FSM, depending on the specific requirements.

In conclusion, Using Soar for high-level control in this system enables scalable and flexible knowledge representation. Its ability to handle dynamic, context-sensitive decisions makes it particularly well-suited for complex robotic applications.

VI. Conclusion

This paper presented a Soar-based system for performing a social navigation task, utilizing Soar as a high-level controller. It demonstrated how high-level control can enhance social navigation by dynamically reconfiguring a lower-level motion controller to optimize behavior across varying contexts. The optimized behavior focused on adjusting the robot's speed to ensure safe and socially appropriate interactions in different scenarios. Soar facilitated this process by evaluating symbolic inputs and applying knowledge of preferred actions to select suitable configurations for maximum speed.

The results demonstrated that the system could effectively control a robot navigating with the Nav2 stack. By adapting configurations based on context, the system successfully achieved effective and socially compliant navigation. However, some limitations were identified, such as inefficient acceleration, non-smooth movements, and the simplicity of the simulation environment. Addressing these limitations in future work can help enhance Soar's effectiveness as a high-level controller in social navigation tasks. Furthermore, a reflection on the rationale for using Soar was provided, comparing it to a finite state machine. This comparison highlighted the advantages of Soar in its ability to organize environmental information and desired behaviors, such as social norms, in a more flexible and scalable manner.

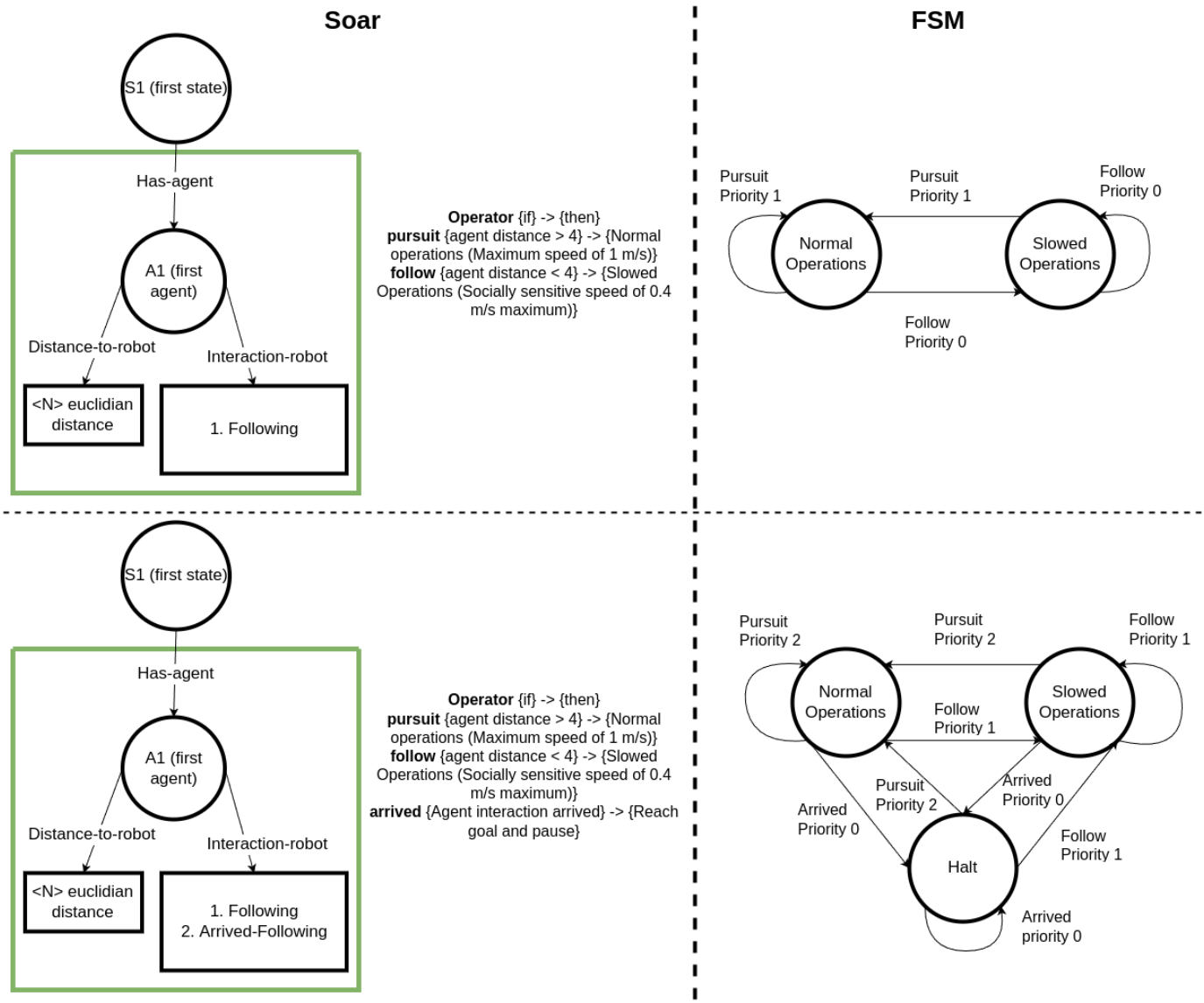

Fig. 11. Example of high-level control concept implemented in Soar and an equavalent finite state machine

ACKNOWLEDGMENT

I would like to give thanks to my supervisors Forough Zamani Khalili and Carlos Hernandez Corbato for all the support they provided throughout the thesis. And would also like to give thanks to the members of the KAS laboratory at the TU Delft for their help and feedback for my thesis.

REFERENCES

[1] A. Francis, C. Pérez-D'Arpino, C. Li, F. Xia, A. Alahi, R. Alami, A. Bera, A. Biswas, J. Biswas, R. Chandra, H.-T. L. Chiang, M. Everett, S. Ha, J. Hart, J. P. How, H. Karnan, T.-W. E. Lee, L. J. Manso, R. Mirksy, S. Pirk, P. T. Singamaneni, P. Stone, A. V. Taylor, P. Trautman, N. Tsoi, M. Vázquez, X. Xiao, P. Xu, N. Yokoyama, A. Toshev, and R. Martín-Martín, "Principles and guidelines for evaluating social robot navigation algorithms," 2023. [Online]. Available: https://arxiv.org/abs/2306.16740

[2] J. E. Laird, "Introduction to soar," 2022. [Online]. Available: https://arxiv.org/abs/2205.03854

[3] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Lecture Notes in Computer Science*, ser. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–30.

[4] J. T. Butler and A. Agah, *Autonomous Robots*, vol. 10, no. 2, p. 185–202, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1008986004181

[5] P. T. Singamaneni, P. Bachiller-Burgos, L. J. Manso, A. Garrell, A. Sanfeliu, A. Spalanzani, and R. Alami, "A survey on socially aware robot navigation: Taxonomy and future challenges," *The International Journal of Robotics Research*, vol. 43, no. 10, p. 1533–1572, Feb. 2024. [Online]. Available: http://dx.doi.org/10.1177/02783649241230562

[6] S. D. Hanford, O. Janrathitikarn, and L. N. Long, "Control of mobile robots using the soar cognitive architecture," *Journal of Aerospace Computing, Information, and Communication*, vol. 6, no. 2, p. 69–91, Feb. 2009. [Online]. Available: http://dx.doi.org/10.2514/1.37056

[7] J. E. Laird, "Toward cognitive robotics," in *Unmanned Systems Technology XI*, G. R. Gerhart, D. W. Gage, and C. M. Shoemaker, Eds., vol. 7332. SPIE, May 2009, p. 73320Z. [Online]. Available: http://dx.doi.org/10.1117/12.818701

[8] S. Hanford and L. Long, "Integration of maps into the cognitive robotic system," in *AIAA Infotech@Aerospace 2010*. American Institute of Aeronautics and Astronautics, Apr. 2010. [Online]. Available:

http://dx.doi.org/10.2514/6.2010-3350

[9] L. N. Long, S. D. Hanford, and O. Janrathitikarn, "Cognitive robotics using vision and mapping systems with soar," in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2010*, J. J. Braun, Ed., vol. 7710. SPIE, Apr. 2010, p. 77100J. [Online]. Available: http://dx.doi.org/10.1117/12.853684

[10] S. D. Hanford and L. N. Long, "Development of a mobile robot system based on the soar cognitive architecture," *Journal of Aerospace Information Systems*, vol. 11, no. 10, p. 714–725, Oct. 2014. [Online]. Available: http://dx.doi.org/10.2514/1.I010191

[11] O. Janrathitikarn and L. N. Long, "Gait control of a six-legged robot on unlevel terrain using a cognitive architecture," in *2008 IEEE Aerospace Conference*. IEEE, Mar. 2008, p. 1–9. [Online]. Available: http://dx.doi.org/10.1109/AERO.2008.4526240

[12] ——, "Gait control of a six-legged robot on unlevel terrain using a cognitive architecture," in *2008 IEEE Aerospace Conference*. IEEE, Mar. 2008, p. 1–9. [Online]. Available: http://dx.doi.org/10.1109/AERO.2008.4526240

[13] C. Van Dang, H. Ahn, H. C. Seo, and S. C. Lee, *A Cognitive Robotic System for a Human-Following Robot*. IOS Press, Nov. 2020. [Online]. Available: http://dx.doi.org/10.3233/FAIA200737

[14] C. V. Dang, H. Ahn, J.-W. Kim, and S. C. Lee, "Collision-free navigation in human-following task using a cognitive robotic system on differential drive vehicles," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 15, no. 1, p. 78–87, Mar. 2023. [Online]. Available: http://dx.doi.org/10.1109/TCDS.2022.3145915

[15] U. Kurup and C. Lebiere, "What can cognitive architectures do for robotics?" *Biologically Inspired Cognitive Architectures*, vol. 2, p. 88–99, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.bica.2012.07.004

[16] B. Eppinger, T. Goschke, and S. Musslick, "Meta-control: From psychology to computational neuroscience," *Cognitive, Affective, amp; Behavioral Neuroscience*, vol. 21, no. 3, p. 447–452, Jun. 2021. [Online]. Available: http://dx.doi.org/10.3758/s13415-021-00919-4

[17] J. Päßler, E. Aguado, G. R. Silva, S. L. T. Tarifa, C. H. Corbato, and E. B. Johnsen, *A Formal Model of Metacontrol in Maude*. Springer International Publishing, 2022, p. 575–596. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-19849-6_32

[18] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2020, p. 2718–2725. [Online]. Available: http://dx.doi.org/10.1109/IROS45743.2020.9341207

[19] M. Giannoulaki and Z. Christoforou, "Pedestrian walking speed analysis: A systematic review," *Sustainability*, vol. 16, no. 11, p. 4813, Jun. 2024. [Online]. Available: http://dx.doi.org/10.3390/su16114813

[20] Y. Gao and C.-M. Huang, "Evaluation of socially-aware robot navigation," *Frontiers in Robotics and AI*, vol. 8, Jan. 2022. [Online]. Available: http://dx.doi.org/10.3389/frobt.2021.721317

[21] S. M. B. P. Samarakoon, M. A. V. J. Muthugala, and A. G. B. P. Jayasekara, "A review on human–robot proxemics," *Electronics*, vol. 11, no. 16, p. 2490, Aug. 2022. [Online]. Available: http://dx.doi.org/10.3390/electronics11162490

[22] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control using covariance variable importance sampling," 2015. [Online]. Available: https://arxiv.org/abs/1509.01149

[23] P. Kaur, Z. Liu, and W. Shi, "Simulators for mobile social robots:state-of-the-art and challenges," 2022. [Online]. Available: https://arxiv.org/abs/2202.03582

[24] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, p. 4282–4286, May 1995. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.51.4282

[25] N. Pérez-Higueras, R. Otero, F. Caballero, and L. Merino, "Hunavsim: A ros 2 human navigation simulator for benchmarking human-aware robot navigation," 2023. [Online]. Available: https://arxiv.org/abs/2305.01303

[26] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From proxemics theory to socially-aware navigation: A survey," *International Journal of Social Robotics*, vol. 7, no. 2, p. 137–153, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1007/s12369-014-0251-1

[27] R. Jones, J. Laird, P. Nielsen, K. Coulter, P. Kenny, and F. Koss, "Automated intelligent pilots for combat flight simulation," *AI Magazine*, vol. 20, 08 1999.

APPENDIX

A. Soar cognitive architecture

The Soar cognitive architecture, illustrated in Figure 12, comprises a collection of task-independent modules that work together to enable intelligent decision-making. These include short-term and long-term memory, processing modules, learning mechanisms, and various interfaces that facilitate communication among components. The central feature of Soar is its working memory, which maintains an agent's current state in the form of a symbolic graph, representing objects, properties, and relations. This working memory reflects the agent's situational awareness, encompassing perceptual input, intermediate reasoning outcomes, active goals, and interactions with systems like semantic memory, episodic memory, the spatial-visual system (SVS), and the motor system [2]. This section explores the architecture's Knowledge Representation and Reasoning components in more depth.

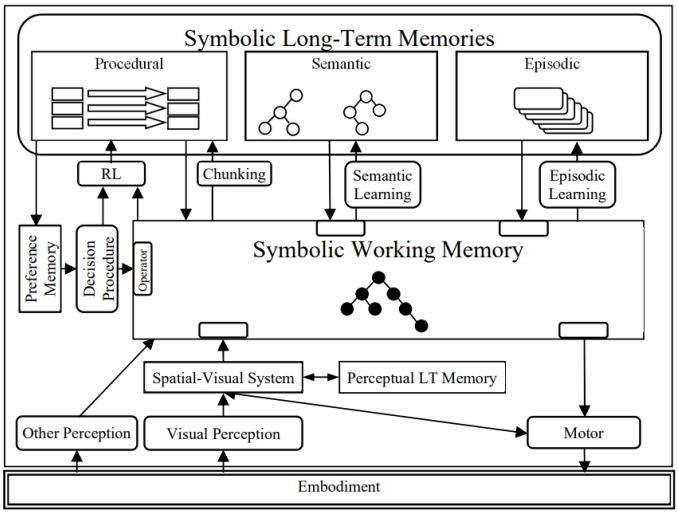

Fig. 12. Soar Cognitive Architecture [2]

Knowledge representation, Working memory: Soar's knowledge representation is centered around working memory, which holds information about the agent's environment and ongoing tasks. Knowledge in Soar is represented in two primary ways: the symbolic working memory and procedural knowledge. Additionally, preferences play a crucial role in guiding decision-making by evaluating different options.

Soar uses a symbolic working memory structured as a graph to represent the current situation. This graph-based structure allows for the flexible representation of objects, their properties, and their relationships. The contents of working memory include real-time perceptual inputs, hypothetical future states, knowledge retrieved from long-term memory, active goals, and motor control outputs.

Working memory is a key part of the agent's reasoning process as it continuously updates to reflect the agent's current understanding of its environment. The situational awareness encoded in working memory is crucial for guiding decision-making, as it determines which actions (or operators) should be considered for execution.

Knowledge representation, Procedural Knowledge: In Soar, procedural knowledge is encoded in the form of operators, which are stored as if-then rules in procedural memory. These operators are responsible for defining actions the agent can take. Operators can be of two types:

- Internal Operators: These handle actions like memory retrieval, goal-setting, or generating hypothetical states for reasoning.
- External Operators: These execute observable actions, such as motor commands for a robot or communication with external systems.

Unlike in traditional rule-based systems where only one rule "fires" per decision cycle, Soar allows multiple operators to be considered and processed in parallel. Each operator represents a unit of knowledge that can be triggered based on the current contents of working memory, allowing for more flexible and dynamic behavior.

Knowledge representation, Preferential memory: Preferences provide a mechanism for evaluating and selecting between multiple operators. Preferences determine how operators are ranked and chosen for execution. They are stored in preferential memory, a specialized part of Soar that allows operators to be compared either explicitly (e.g., operator A is preferred over operator B) or through numeric evaluation.

Preferences help the system decide which operator is best suited for the current situation. For example, multiple operators may be proposed for a given state, but preferences guide the decision-making process by comparing them and assigning higher priority to the most appropriate operator. The preferences can incorporate both general knowledge and specific situational information, such as perceptual input or data retrieved from long-term memory.

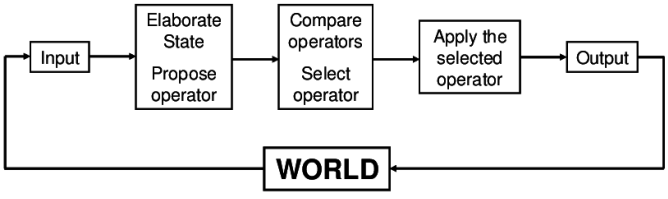

Fig. 13. Decision cycle of Soar [6]

Reasoning, Soar Decision cycle: The process of reasoning in Soar is governed by the decision cycle (Figure 13). This cycle describes how Soar evaluates and selects operators to perform actions. The decision cycle can be broken down into three main phases:

- Proposing Operators: In the first phase, Soar evaluates the current situation in working memory and proposes operators that are applicable. These operators are selected based on their relevance to the agent's current goals and the state of the environment. For example, if the agent is tasked with moving toward a target, a "move" operator might be proposed if the target is within range.
  Operators are proposed based on production rules stored in procedural memory, which define when a particular operator should be considered. This phase generates a set of potential actions that the agent could take.
- Evaluating Operators: Once operators are proposed, Soar enters the evaluation phase. This involves comparing the proposed operators using preferences to determine which one is most suitable for the current situation. Preferences can be either: Qualitative, specifying which operator is better (e.g., operator A $>$ operator B), or Quantitative, assigning a numerical score to operators.
- Applying the Selected Operator: After evaluating the proposed operators, Soar selects the best operator based on the preference evaluation and applies it. The selected operator modifies the current state in working memory, either by changing the agent's internal goals, retrieving knowledge, or executing an action in the environment (e.g., moving a robot or interacting with external systems).

The decision cycle then repeats, continuously updating working memory and refining the agent's actions as new information is perceived or as goals change. This cycle allows Soar to engage in real-time decision-making, where actions are dynamically adjusted based on the agent's evolving understanding of its environment. [7]

B. Example Soar Decision-making

In Figure 14, an example of the Soar decision cycle is illustrated, showing how the state is represented in working memory as the system progresses through three decision cycles. The table is read from left to right, with each row corresponding to a single decision cycle and the columns representing different components of the decision cycle.

In the first cycle, Soar is initialized, and the state begins empty. The "initialize-social-navigation" operator (described in Table I) is proposed, evaluated (without contest since it is the only operator), and applied. This operator prepares Soar to initialize the navigation graph using input from the next cycle.

In the second cycle, perception input defines the graph by specifying detected elements in the environment. For instance, it detects a human, who is the robot's target to follow. A new operator is proposed and evaluated. This time, the output is external and sent to Nav2, instructing it to navigate to a waypoint directly behind the human at normal speed.

In the third and final cycle, the input updates to reflect a change in the environment—specifically, the density of humans around the robot is now classified as "crowded." As a result, two operators are proposed. Soar evaluates these options based on preferential memory, which specifies that "Pursuit" $<$ "Crowded-environment". In natural language, this means the "Crowded-environment" behavior is preferred over "Pursuit." Consequently, the "Crowded-environment" operator is selected and applied, leading Nav2 to receive a command to navigate to the waypoint behind the human but at a slower speed to account for the increased density.

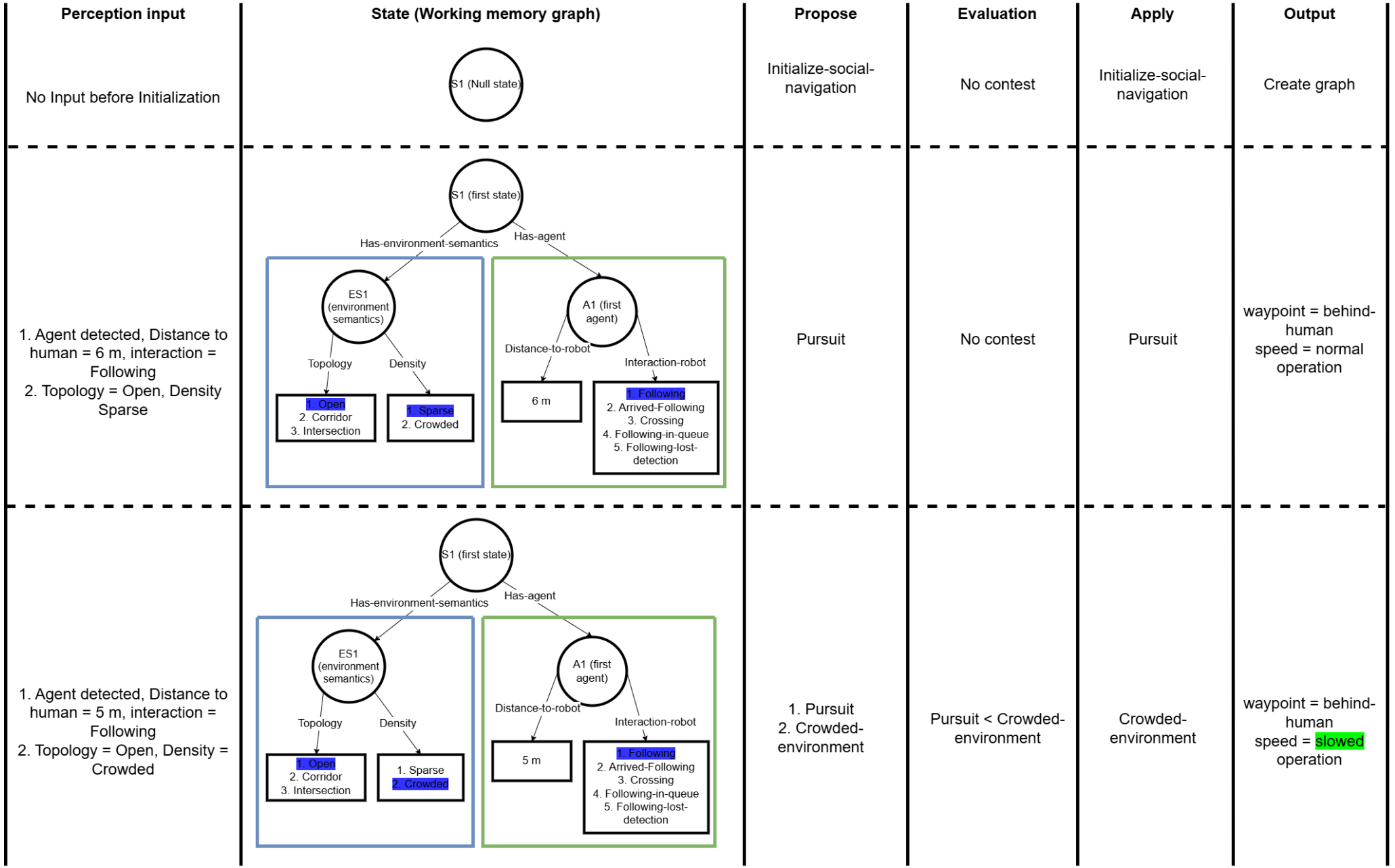

Fig. 14. Example of Soar decision-making

C. Simulation Description

1) Tiago: The robot used in the simulation is the Tiago, developed by PAL Robotics, as shown in Figure 15. Tiago features a differential drive mobile base, which supports the robot's body, an RGB-D camera, and a manipulator. The mobile platform also includes a 2D LiDAR sensor for environment mapping and localization. PAL Robotics provides the ROS 2 stack, which is utilized in the simulation to interface with the robot's hardware and manage its operations.

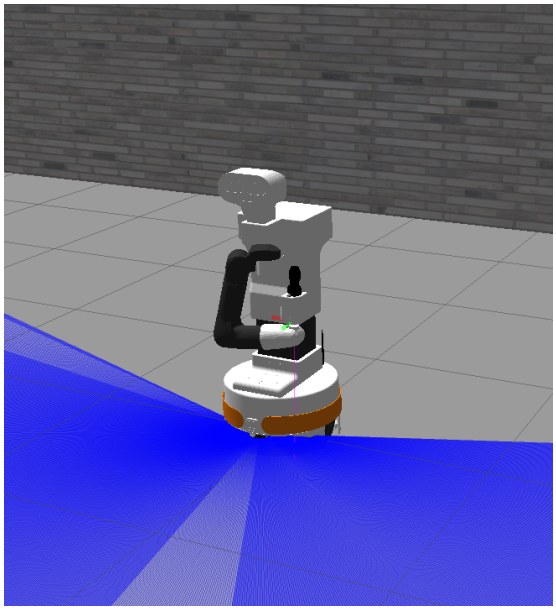

Fig. 15. Tiago robot used in simulation

2) pedsim_ros: The simulation is conducted in Gazebo Classic, utilizing the pedsim_ros package to simulate human agents following the social force model [24]. Each agent is assigned a route to traverse and maintains an average speed of approximately 0.3 m/s. While most agents follow conventional navigation patterns, focusing on reaching their goals, some agents are designed to pass through queueing zones, as shown in figure 4. The humans and robot navigate through an office environment, which is free of objects but contains corridors and intersections, as was depicted in figure 4. This environment is designed to model typical indoor spaces, providing a controlled setting to assess the robot's social navigation performance.

Figure 16 provides an overhead view of the simulation environment, where only the mobile robot base of Tiago is visible, and the number of human agents is fewer than in the full simulation scenario. In Figure 17, a closer view is shown, with Tiago positioned at the top of the image. The blue rays represent LiDAR scans, indicating the robot's sensing range within the environment.

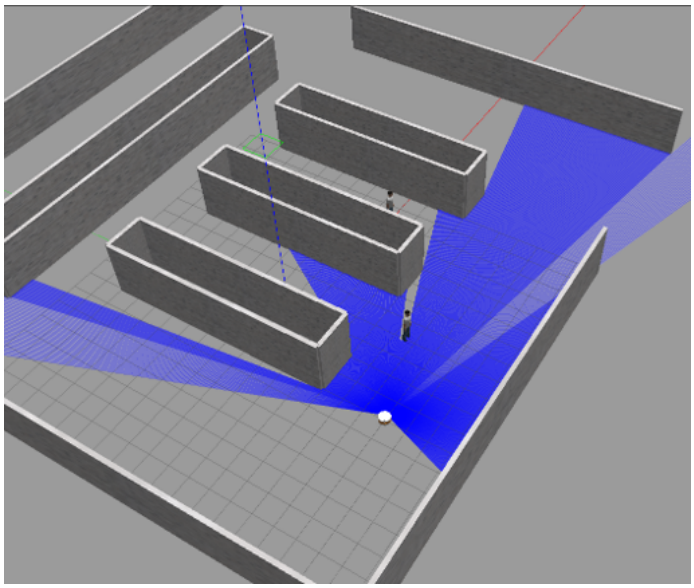

Fig. 16. Simulation environment from above

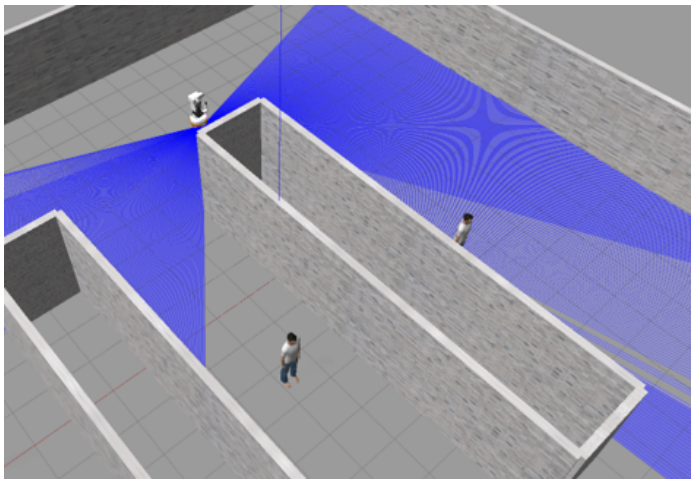

Fig. 17. Simulation environment with Tiago