2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)
August 23-27, 2021. Lyon, France

A ROS Package for Human-In-the-Loop Planning and Control under Linear Temporal Logic Tasks

Robin Baran, Xiao Tan, Peter Varnai, Pian Yu, Sofie Ahlberg, Meng Guo, Wenceslao Shaw Cortez and Dimos V. Dimarogonas

2021 IEEE 17th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-1873-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/CASE49439.2021.9551648

Abstract— **In this paper, we propose a ROS software package for planning and control of robotic systems with a human-in-the-loop focus. The software uses temporal logic specifications, specifically Linear Temporal Logic, for a language-based method to develop correct-by-design high level robot plans. The approach is structured to allow a human to adjust the high-level plan online. A human may also take control of the robot (in a low-level control fashion), but the software prevents the human from implementing dangerous behaviour that would violate the high-level task specification. Finally, the planner is able to learn human-preferred high-level tasks by tracking human low-level control inputs in an inverse learning framework. The proposed approach is demonstrated in a warehouse setting with multiple robot agents to showcase the efficacy of the proposed solution.**

I. Introduction

In recent years, there has been an increased focus in human-robot collaborations [1]. The use of robots and autonomous systems alongside humans has reduced ergonomic injuries in the workplace, while still improving safety and productivity [1], [2]. A review of existing human-robot collaborative methods can be found in [3]. Despite the existing research in human-robot collaborations, there is a need for a planning and control framework with "programming-free robot control" and adaptive/learning abilities that is able to guarantee safety [3]. Programming-free robot control consists of gesture, haptic, verbal, or other such input that does not require skilled personnel to program commands to the robot. Learning and adaptive abilities refer to the ability of the robot system to adapt its plan based on human preference.

Linear Temporal Logic (LTL) offers ways to express complex tasks and planning problems as formulas that can be interpreted as natural language, and can be used for example with speech-type commands [4]–[6].Those existing methods can construct correct-by-design robot motion plans that satisfy LTL specifications (e.g, coverage, safety). Moreover, humans can be further integrated with the autonomous system using recently-developed human-in-the-loop (HIL) features [7]. However the implementation of such temporal logic-based planners requires highly trained specialists to apply the approach to a wide-range of robot applications.

This work was supported by the Swedish Research Council (VR), the Swedish Foundation for Strategic Research (SSF), the Knut and Alice Wallenberg Foundation (KAW) and the H2020-EU Research and Innovation Programme under the GA No. 101016906 (CANOPIES). R. Baran, X. Tan, P. Varnai, P. Yu, S. Ahlberg, W. Shaw Cortez, and D. V. Dimarogonas are with the School of EECS, KTH Royal Institute of Technology, 100 44 Stockholm, Sweden (Email: `rbaran, xiaotan, varnai, piany, sofa, wenscs, dimos@kth.se`). M. Guo is with the Bosch Center for Artificial Intelligence, 71272 Renningen, Germany (Email: `meng.guo2@de.bosch.com`)

There exist few all-encompassing software packages for LTL planning. Most toolboxes only provide partial implementations, as they, for example, only convert an LTL specification into a Büchi automaton [8]. Furthermore, many existing approaches are tailored to certain robot implementations such as robot motion (e.g., move from region A to B) [7], [9]. However task specifications for robots are not solely restricted to robot motion. For example, mobile robots must also consider battery levels for re-charging. Other robots such as robotic manipulators may require plans associated with assembling and unloading/loading packages, which cannot be described by simple motion plans. Adapting those methods to more general implementations would require highly trained personnel and ultimately reduces the usability and flexibility of the approach. With the proposed software package, we aim to bridge the gap between existing theoretical methods in LTL-based planning and their implementation.

We propose an LTL-based ROS (Robot Operating System) software package for human-aware planning and control. The software is highly modular and allows for general robot states and actions, including those in continuous-time (e.g., robot motion) and in discrete-time (e.g., pick/drop, charged/uncharged). Using this software, a human specifies an LTL specification using commands that are close to natural language, and the planner generates a plan to satisfy the human's task. The proposed package contains two main HIL modalities. First, the software provides a mixed-initiative control framework that allows humans low-level remote control of the robots, but prevents violation of the high-level task to enforce safety. Second, an inverse-learning algorithm is provided that allows a human (via the mixed-initiative controller) to alter the high-level plan according to the human's preference in a safe manner. Although there exist many works [10], [11] on robot learning trajectories by human demonstration, the proposed approach is integrated with the LTL specifications and focuses on learning human preference of the high-level plan. The proposed approach is demonstrated in a multi-robot warehouse scenario.

II. Preliminaries

A. Linear Temporal Logic

An LTL formula over a set of atomic propositions AP is recursively defined as

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2,$$

where $a \in AP$ and $\bigcirc$ (next), U (until). The Boolean connector disjunction $\vee$, and temporal operators $\Diamond$ (eventually) and $\Box$ (always) can be defined as $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\Diamond\varphi := \top U \varphi$ and $\Box\phi := \neg\Diamond\neg\phi$. LTL formulas are interpreted over infinite words made of subsets of AP, i.e., over $(2^{AP})^{\omega}$. The full semantics and syntax of LTL are omitted here due to space limitations, see e.g., [12].

Definition 1. [13] A nondeterministic Büchi automaton (NBA) is a tuple $\mathcal{B} = (S, S_0, 2^{AP}, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- 2^{AP} is the input alphabet,
- $\delta : S \times 2^{AP} \to 2^S$ is the transition function, and
- $F \subseteq S$ is the set of accepting states.

An infinite *run* $\mathbf{s}$ of a NBA is an infinite sequence of states $\mathbf{s} = s_0 s_1 \ldots$ generated by an infinite sequence of input alphabets $\sigma = \sigma_0\sigma_1 \ldots \in (2^{AP})^{\omega}$, where $s_0 \in S_0$ and $s_{k+1} \in \delta(s_k, \sigma_k), \forall k \geq 0$. An infinite run $\mathbf{s}$ is *accepted* by $\mathcal{B}$ if and only if $\text{Inf}(s) \cap F \neq \emptyset$, where $\text{Inf}(s)$ is the set of states that appear in $\mathbf{s}$ infinitely often.

Definition 2 ([9]). A wFTS is a tuple $\mathcal{T}_i = (Q_i, \Sigma_i, \to_i, q_0^i, AP_i, L_i, W_i)$ where $Q_i = \{q_1, ..., q_{N^t}\}$ is the finite set of states; Σ_i is the set of actions, $\to_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the transition function; q_0^i is the set of initial states; AP_i is the set of atomic propositions for $\mathcal{T}_i$; $L_i : Q_i \to 2^{AP_i}$ is the labeling function; $W_i : Q_i \times \Sigma_i \times Q_i \to \mathbb{R}^+$ is the weight function as cost of transition in $\to_i$.

B. Mixed Initiative Controller

The mixed initiative controller (MIC) is a low-level control scheme, which mixes the human input (e.g., from a joystick) with the planner input. One instance of this is:

$$u \triangleq u_r(x) + k(x)u_h(t), \tag{1}$$

where $u_r(x)$ is the planner input at state x, $k(x) \in [0, 1]$ is a smooth function to be designed, and $u_h(t)$ is the human input, which is unknown to the robot. In [7], a MIC is designed for the navigation of single-integrator dynamics.

C. Robot tasks

The robot is assigned a task φ, which is expressed as an LTL formula and is specified with the following structure:

$$\varphi = \varphi^{hard} \wedge \varphi^{soft} \tag{2}$$

where φ^{hard} and φ^{soft} are respectively the hard task and soft task LTL formulas that define the high level task. The hard task φ^{hard} defines specifications that should strictly be satisfied, such as safety requirements. The soft task φ^{soft} could include optional tasks for which satisfaction is less stringent and could be violated if incompatible with the hard task. Human preference can also affect the enforcement of the soft task (see Section III-D).

III. Human-in-the-Loop Planning and Control Software

We propose an automata-based LTL planning package within the ROS (Robot Operating System) framework [14].

A. Software Architecture

ROS consists of a node-based architecture within which the proposed planner is integrated (see Fig. 1). The LTL (core) planner node computes the plans for each robot agent based on the LTL specification, the agent model, and its environment. The agent model is represented as a weighted finite transition system (wFTS) (see Definition 2). The planner node uses the wFTS state from the agent node to either update the plan or output the next action command to be executed. Each agent has a specific node to communicate with the planner node and convert the high level plan to agent-specific low-level commands.

Separating the LTL core planner from the agent-specific node facilitates integration on a wide range of robot platforms, including mobile robots and multi-DOF robot manipulators. The planner relies on a finite Büchi automaton translation of the LTL specification and outputs high-level commands (`string` type) to the agent node. These high-level commands are converted into low-level inputs for the specific agent. State monitors then map the agent physical state to the abstract wFTS state. The associated documentation provides information on how the LTL formula and the agent model wFTS are defined with example code [15].

The wFTS is shared as a ROS parameter and accessible by any ROS node (see Figure 1) [14]. A plugin functionality allows additional code to run within the planner node, providing a modular way of implementing new features, including the HIL features of Section III-D.

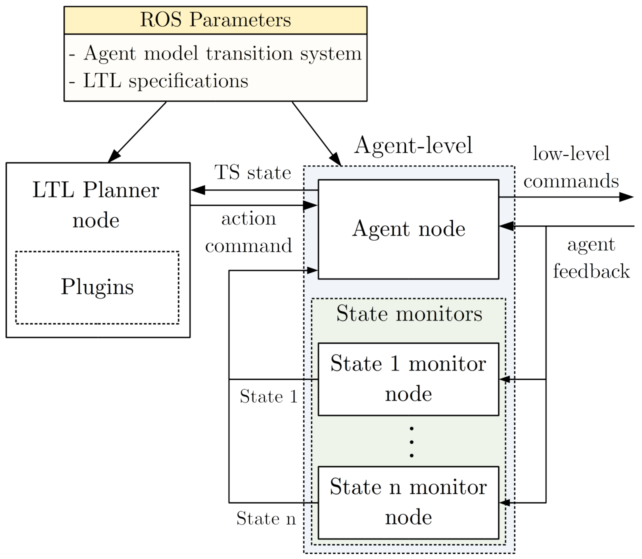

Fig. 1: LTL automaton stack node graph

Example. To illustrate the proposed solution, we introduce the following example. A differential-drive Turtlebot2 robot, simply called turtlebot hereafter, can navigate in a grid-discretized workspace composed of six regions r_j, $j \in \{1, ..., 6\}$, see Fig. 2. It is tasked with "picking up a package at region r_1 and delivering it to region r_5, while avoiding the hazardous region r_4". As an optional requirement, the robot is tasked with "visiting region r_3 once it has picked up the package for a visual inspection before delivery". These tasks are cast as LTL formulas as in the next section. A video of this example can be found at [16].

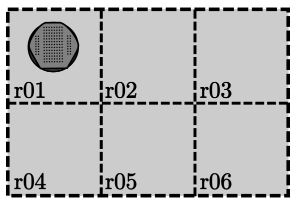

Fig. 2: Turtlebot2 robot (left) on its grid workspace (right)

B. LTL Core & Planner

The core package contains the general planner that is not agent-specific. The planner node takes as input an LTL task φ in the form of (2), as well as an agent model wFTS.

1) LTL to NBA: First, the hard and soft LTL formulas φ^{hard} and φ^{soft} are used to generate the respective NBAs $\mathcal{B}_{\varphi^{hard}}$ and $\mathcal{B}_{\varphi^{soft}}$ via the LTL2BA software [8]. Then, the planner generates a combined Büchi automaton $\mathcal{B}_\varphi := \mathcal{B}_{\varphi^{hard}} \times \mathcal{B}_{\varphi^{soft}} = (S, S_0, 2^{AP}, \delta, F)$ using the safety-ensured product automaton developed in [17]. As part of this safety-ensured product automaton, we implement the design parameter $\beta \in \mathbb{R}_{\geq 0}$, which adds a penalty for violating the soft task. Setting a higher value will help enforcing the soft task. We note that β is dynamic and can be updated online. The plan is published on a ROS topic, which is further described in the documentation [15].

2) Agent Model: The agent model consists of a discretization of the agent's workspace into regions along with allowable transitions between regions, and other types of actions that can be performed (e.g., pick/drop). Recall that agent modelling is based on the wFTS from Definition 2.

We note that while the previous approach [9] provides some modularity by separating the physical workspace from the agent's action map, that approach is still limited to a 2D motion model and a simple action model. We extend that work to address multi-dimensional motion models, and more complex action models for a more general framework. This is accomplished by combining $N \in \mathbb{N}$ motion/action models together via the intersection of wFTSs (see Definition 3). Guard functions G_i are used to identify allowable transitions between each wFTS. This approach allows for region-specific actions. We will refer to the ith motion /action model as the wFTS $\mathcal{T}_i$. The agent model is thus constructed from the intersection of all wFTS $\mathcal{T}_i$, $i \in \{1, .., N\}$.

Definition 3. The wFTS intersection is defined by:

$$\mathcal{D} = \mathcal{T}_1 \times \mathcal{T}_2 \times ... \times \mathcal{T}_N = (Q^P, \Sigma^P, \rightarrow_P, Q_0, AP, L_P, W_P)$$

where $Q^P = Q_1 \times Q_2 \times ... \times Q_N$ is the set of states; $\Sigma^P = \bigcup_{i=1}^{N} \Sigma_i$, $Q_0 = q_{1,0} \times q_{2,0} \times ... \times q_{N,0}$ is the set of initial states; where $AP = \bigcup_{k=1}^{N} AP_k$; $L_P : Q_1 \times Q_2 \times \ldots \times Q_N \rightarrow 2^{AP}$, $L^P(q_1, \ldots, q_N) = \bigcup_{i=1}^{N} L_i(q_i)$ is the labelling function, W_P is the weight function, and $\rightarrow_P \subseteq Q^P \times \Sigma^P \times Q^P$ is the transition function where $((q_1, q_2, ..., q_N), \sigma_a, (q'_1, q'_2, ..., q'_N)) \in \rightarrow_P$ if and only if

- $\exists i \in [1, 2, ..., N]$ s.t. $(q_i, \sigma_a, q'_i) \in \rightarrow_i$,
- $\forall j \neq i$, $q_j = q'_j$, and
- $G_i(q_1, q_2, ..., q_N, \sigma_a)$ is true, i.e. the guard associated with the transition is satisfied at state $q = (q_1, q_2, ..., q_N)$. Here, the guard function is defined as:

$$G_i : Q_1 \times Q_2 \times \ldots \times Q_N \times \Sigma_i \rightarrow \{true, false\}.$$

The ability to combine any type of discretizable state-space in the agent model (e.g., 3D motion, 6-DoF manipulator configuration, battery state, pick/drop state) allows the planner to be used with any type of agent as long as the state-space can be described as a finite transition system. It is important to note that whatever the combination of $\mathcal{T}_i$, the LTL planner yields a sequence of actions such that each action is performed one-at-a-time. Also, by executing one action, we assume that only one dimensional state changes. This is not restrictive in general as different $\mathcal{T}_i$ describe different aspects of the robot state (e.g., pose, battery, loaded/unloaded).

Example (Continued)**.** In the turtlebot example, the turtlebot model wFTS in Fig. 3 is a combination of two individual wFTS: the "region" wFTS with a grid-discretized 2D workspace and the "load" wFTS. A guard on the "pick" and "drop" actions restricts these actions to specific regions and therefore defines a picking station and a delivery station without the need to include it in the LTL specifications. Indeed on the combined graph the "pick" transition only exists in "$r1$" and the "drop" transition only exists in "$r5$". We note that here we use two individual wFTSs to build our wFTS intersection for readability. However, our framework accomodates an arbitrary number of wFTSs.

Thanks to this guard function, the task can be succinctly expressed as the following hard and soft task LTL specifications: $\varphi^{hard} = (\Box\Diamond loaded) \wedge (\Box\Diamond unloaded) \wedge (\Box\neg r_4)$, $\varphi^{soft} = \Box\Diamond(loaded \wedge r_3)$ In the hard task, the turtlebot agent must infinitely often (always eventually) be loaded (i.e. pick up the package), infinitely often (always eventually) be unloaded (i.e. deliver the package) and always avoid (always not) region r_4. The soft task requires the turtlebot to infinitely often visit region r_3, while loaded with the package to exemplify a package inspection point for quality control that is optional in the high level task.

3) Product Automaton & Plan Generation: The planner node uses the previously obtained NBA $\mathcal{B}_\varphi$ and the wFTS intersection $\mathcal{D}$ to generate a discrete plan. The code for plan generation is based on previous work [18]. A product Büchi automaton (PBA) $\mathcal{A}_\mathcal{P}$ is defined as a tuple:

$$\mathcal{A}_\mathcal{P} = \mathcal{B}_\varphi \otimes \mathcal{D} = (\mathcal{S}_\mathcal{P}, \delta_\mathcal{P}, \mathcal{S}_{\mathcal{P},0}, \mathcal{F}_\mathcal{P}, \mathcal{W}_\mathcal{P})$$

where $\mathcal{S}_\mathcal{P} = S \times Q^P$; transition relation $(\langle s, q\rangle, \langle s', q'\rangle) \in \delta_\mathcal{P}$ iff $\exists \sigma \in \delta, s' \in \delta(s, \sigma)$ and $\exists \sigma_a \in \Sigma^P, (q, \sigma_a, q') \in \rightarrow_P$; the set of initial states $\mathcal{S}_{\mathcal{P},0} = S_0 \times Q_0$; the set of accepting states $\mathcal{F}_\mathcal{P} = (F \times Q^P)$; the weight function $\mathcal{W}_\mathcal{P} : \delta_\mathcal{P} \rightarrow$

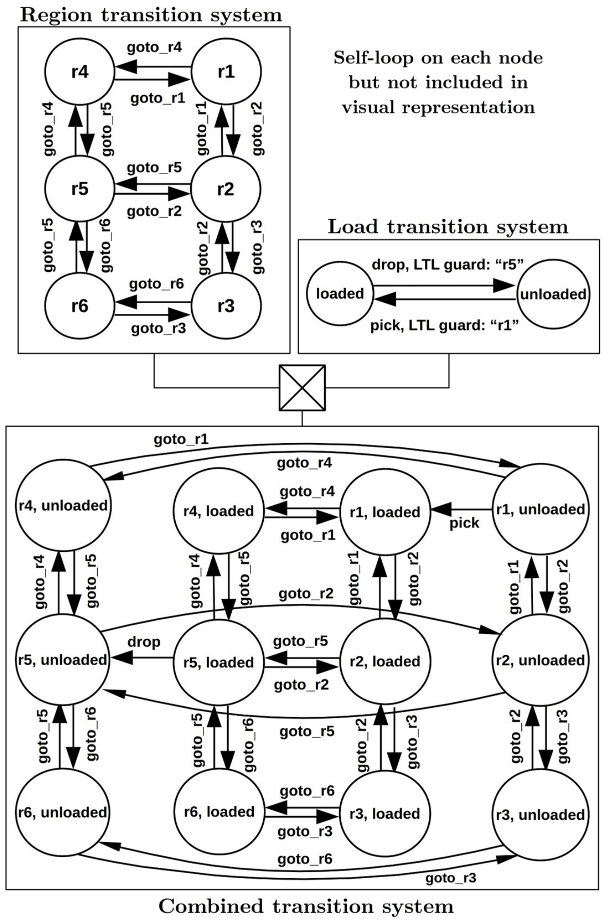

Fig. 3: Turtlebot combined transition system

$\mathbb{R}^+, W_{\mathcal{P}}(\langle s,q\rangle, \langle s',q'\rangle) = W_P(q,q')$. From this PBA, we can find an optimal run and project it back to the wFTS intersection $\mathcal{D}$ using model-checking methods [19].

Accepting runs have the following *prefix-suffix* structure: $r_{\mathcal{P}} = p_0, p_1 \cdots p_k(p_{k+1} \cdot \cdots \cdot p_n p_k)^\omega$, where $p_0 \in \mathcal{S}_{\mathcal{P},0}$ and $p_k \in \mathcal{F}_{\mathcal{P}}$. The output word has two distinct parts: a finite *prefix* part that is executed only once from the initial state p_0 to an accepting state p_k and a *suffix* part that is repeated infinitely from the accepting state p_k to itself [19]. Additionally, the optimal accepting run minimizes a cost function based on transition weights. A design parameter $\gamma \in \mathbb{R}_{\geq 0}$ adds weight on the cost of the suffix. A higher value for γ will therefore favor accepting runs with shorter suffixes. We refer to [17] for more details.

The optimal accepting run has an associated input word, i.e., an action sequence, for the agent to execute to satisfy its specifications. This plan is also in the *prefix-suffix* structure. The planner node keeps track of the execution of the plan and outputs the next action command to the agent node.

Example (Continued)**.** Following with our turtlebot example (see [16] for reference), for the parameters $\beta = 10$, $\gamma = 10$ and initial condition $x_{\mathcal{P},0} = (r1, unloaded)$, the planner outputs the following plan: prefix: $[pick, goto\_r2]$, suffix: $[goto\_r5, drop, goto\_r2, goto\_r1, pick, goto\_r2]$. The turtlebot will execute the prefix once, which consists of picking up the package (while in r_1) and moving to region r_2. It will then repeat the suffix in an infinite loop, i.e., go to region r_3, drop the package, go to region r_2, go to region r_1, pick up the package, go to region r_2, and so forth. Notice in [16] that the low β value causes the soft task not to be fulfilled. Indeed region r_3 is never visited in this action sequence. Increasing β sufficiently (e.g., we set $\beta = 1000$) will enforce the soft task and result in the following plan: prefix: $[pick, goto\_r2]$, suffix: $[goto\_r3$, $goto\_r2$, $goto\_r5$, $drop$, $goto\_r2$, $goto\_r1$, $pick$, $goto\_r2]$. Now region r_3 is visited while loaded, satisfying the soft task.

C. Agent-level Software

The agent-level nodes (see Fig. 1) are agent-specific (or even scenario-specific). The agent node is in charge of interpreting the action command (of `string` type) and sending the appropriate low-level commands to the physical system. We denote by the low-level control $u_{r_i} \in \mathbb{R}^{m_i}$ the action to be implemented in the transition $\rightarrow_i$ for $m_i \in \mathbb{N}$ and $i \in \{1, ..., N\}$. Additionally, state monitors (working as independent nodes or integrated within the agent node) track the physical system configurations and relate them to states in the respective $\mathcal{T}_i$. The agent node then aggregates these states to a TS state and publishes it to the planner node.

A few "standard" state monitors (2D pose regions, 6-DoF joint space regions) are included in the software package for convenience. Code for different types of agents is also provided (holonomic Nexus robot, differential-drive Turtlebot2, HEBI 6-DoF manipulator).

Example (Continued)**.** The turtlebot can navigate in a 2D plane with discretized regions and pick/drop a package. The navigation action *goto_region* is converted to a velocity command by the agent node and the default ROS turtlebot navigation stack, and the robot position is mapped to a region r_j of the workspace by the *2D region state monitor*. In a similar manner, the *pick* and *drop* actions are themselves converted by the agent-node to more complex low-level action sequences involving all the steps necessary for transporting the package. We note that the use of the ROS navigation stack to convert the action to low-level velocity command is specific to the turtlebot and other agent types may use different nodes. The proposed software applies to general systems and is agnostic to the low-level implementation.

D. Human-In-the-Loop and Inverse Reinforcement Learning

Our software implementation includes two HIL features: MIC and inverse reinforcement learning (IRL) [7]. The MIC mixes the human input and planner input without violating the hard task specifications. The IRL feature allows the agent to learn the human preference by satisfying or neglecting the soft task according to the run executed by the human.

1) MIC: The MIC ensures that the hard task φ_{hard} is being respected at all time. The violation of the hard task occurs if the agent enters a PBA state, called a *trap state*, that prevents the hard task from being fulfilled. Trap states are PBA states from which the Büchi acceptance condition cannot be fulfilled, i.e., states that cannot reach accepting states that appear infinitely often. The set of all trap states is denoted $\mathcal{O}_t$. In general, a distance metric is required to track the proximity of the current state to the trap state set $\mathcal{O}_t$. The definition of such a distance metric and the associated mixed-initiative controller is system- and $\mathcal{T}_i$-specific. For instance, the mixed initiative navigation controller proposed in [7] guarantees that the single-integrator system never reaches a trap state no matter what human input is provided.

In addition to the mixed initiative navigation controller in [7], we further define a MIC for the wFTSs (e.g., the load transition system in Fig. 3). The MIC is given by:

$$u_i^k(x_i^k, p, \mathcal{O}_t) \triangleq \begin{cases} u_h & \text{if } (p, p') \in \delta_{\mathcal{P}} \text{ and } p' \notin \mathcal{O}_t \\ u_{r_i}^k & \text{otherwise,} \end{cases} \quad (3)$$

where $k \in \mathbb{N}$ denotes the discrete time step and $i \in \mathbb{N}$ corresponds to $\mathcal{T}_i$ of $\mathcal{D}$, p, p' are states in the PBA $\mathcal{A}_{\mathcal{P}}$, u_h is the human input and $u_{r_i}^k$ is the autonomous control command at time step k.

This MIC feature is implemented in the software (see Fig. 4) through an additional mixed-initiative node that takes in the low-level commands and human inputs, and outputs the mixed commands. According to a $\mathcal{T}_i$-specific distance metric, the monitoring node provides both the "closest" state and the distance to this state. In addition, the planner node offers a ROS service to check if a transition system state is a trap state via its HIL plugin. This modular approach facilitates integration of other mix-initiative controllers based on different dynamics by simply updating the mixed-initiative node.

Example (Continued)**.** Both types of mixed-initiative controllers ((1) and (3)) can be used with the turtlebot. The high-level *goto_region* command is converted to a velocity command via the default ROS navigation stack. The human controls the turtlebot by sending velocity commands via joystick. These two control commands are mixed in a similar manner in [7]. As a result, the human will have shared control with the planner unless the robot approaches the hazardous region r_4. In addition, the human can command the *pick* and *drop* actions at the push of a button, but this command is only valid when the robot is in the regions r_1 or r_5, as the pick/drop actions are restricted to these regions (see Fig. 3). We note that the commands are sent through a joystick in this example but the proposed software accepts any inputs that can be converted to a ROS topic.

2) IRL: The IRL plugin allows the planner to learn the human preference regarding the satisfaction of the soft task. The IRL algorithm adapts to human preference by adjusting β (the weight determining the priority of satisfying the soft constraints). The adaptation of β is dependent on the resulting path the robot takes as a result of the low-level human input implemented via the mixed-initiative controller. The combination of the mixed-initiative control law and the IRL algorithm allows for safe learning of human preference. Furthermore, the IRL algorithm is only dependent on the robot path, and not on the robot dynamics, making the approach applicable to a wide-range of systems. The details of the IRL algorithm can be found in [7].

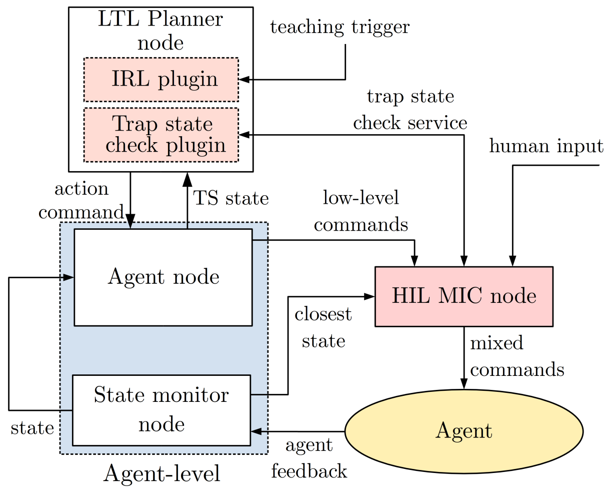

Fig. 4: HIL with Inverse Reinforcement Learning node graph

The IRL feature is implemented in a plugin of the LTL planner as shown in Fig. 4. An external input triggers the teaching phase, wherein the agent runs are recorded. The trigger signal is sent using a ROS topic and can therefore be linked to any interface. In the teaching phase, the human may demonstrate his/her preference by sending low-level commands to the agents, in a MIC manner (Section III-D.1). The recorded runs are then used for updating β, with which a new plan is constructed to satisfy/negate the soft task.

Example (Continued)**.** Let's consider a case where the initial β is too small for the soft task to be in the plan, i.e, $\beta = 10$: the turtlebot does not go to r_3 after picking. If, during the run, the human drives the turtlebot to r_3 while being loaded, the soft task is executed. In this case, the resulting robot path is used by the IRL algorithm to update the value of β. With a sufficient large β, re-planning yields a plan with the soft task being satisfied. Similarly, the human could negate the soft task by using the mixed-initiative controller to drive the turtlebot away from r_3 while loaded.

IV. Multi-Agent Hardware Demonstration

In this section we present a demonstration of the software in a multi-agent setup. The multi-agent demonstration exemplifies a factory packaging station where packages are assembled and transported for shipping. Humans assemble the packages, while the robots, i.e. the nexusbots (holonomic mobile robots), Hebi arm (a 6-DOF robot manipulator), and turtlebot, transport the package to a shipping station.

The demonstration setup is shown in Figure 5. For brevity, we refer to the documentation [20] for the exact definitions of each agent's wFTS model. The nexusbots are tasked with transporting assembled packages to the delivery region, while remaining inside the workspace. The nexusbots cannot leave the delivery region until the Hebi arm has confirmed that it has picked up the package. The nexusbots must also await confirmation from the human that a package is ready for delivery. We note that low level collision avoidance is implemented to prevent collisions between the nexusbots. The Hebi arm is tasked with picking up a package from a nexusbot only when a nexusbot is in the delivery region. Once the Hebi arm (via an electromagnet) picks up the package, it will transport the package to the drop region only if the turtebot is awaiting delivery. When no pick-up/drop-off task is being executed, the Hebi arm waits in a stand-by configuration. Finally, the turtlebot is tasked with transporting the package from the Hebi arm only once the Hebi arm has confirmed that the package has been loaded onto the turtlebot. Once loaded, the turtlebot must avoid obstacles, while transporting the package to the human. Once the human confirms the package is delivered, the robot will return to the pick-up region and wait for a new package from the Hebi arm. Finally, the turtlebot is tasked with re-charging whenever its battery levels are low. When low battery is detected, the turtlebot must return to the charging station until its batteries are charged, and then resume the transport task. We note that the wFTS model and agent tasks are constructed to prevent collisions between all heterogeneous agents and each agent does not require knowledge of the other agents to complete the task.

In [21], a time-lapse of the entire scenario is shown including the packages being assembled and transported. At time 07:58, a human operator needs to change the high level plan to inspect the turtlebot package. The human activates the teaching trigger via joystick and uses the MIC to navigate the turtlebot to satisfy the soft task. The turtlebot learns that the human wishes to satisfy the soft task via the IRL and the new plan ensures the package enters the inspection site.

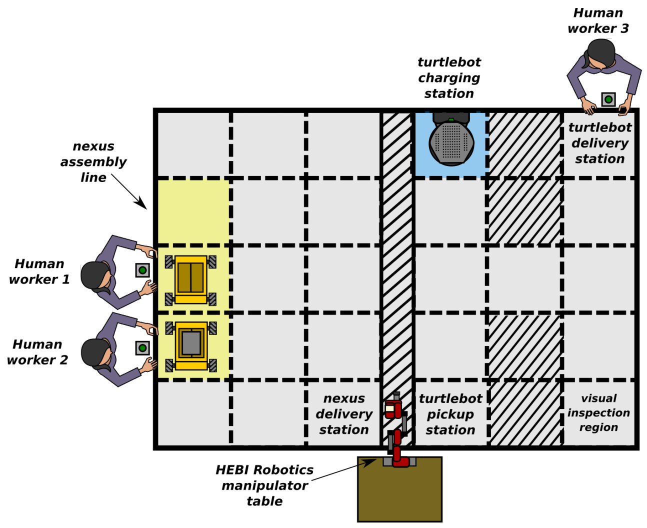

Fig. 5: Factory setting demonstration

V. Conclusion

In this paper, we presented a ROS software package for robot planning and control with HIL features. The software allows humans to define a high-level task (composed of a hard and soft task) in the form of LTL specifications. The software allows for general descriptions of a robot's motion and action models and provides a correct-by-design plan to satisfy the high level tasks. The HIL features include mixed-initiative control to allow a human low-level control of the robot, while always respecting the hard task. Furthermore, an inverse reinforcement learning plugin is provided so that the robot can adapt the high level plan to satisfy human-preference. Future work consider multi-modal human inputs and multiple human-preferred soft task specifications.

References

[1] A. Cherubini, R. Passama, A. Crosnier, A. Lasnier, and P. Fraisse, "Collaborative manufacturing with physical human–robot interaction," *Robotics and Computer-Integrated Manufacturing*, vol. 40, pp. 1–13, 2016.

[2] J. Krüger, T. K. Lien, and A. Verl, "Cooperation of human and machines in assembly lines," *CIRP Annals*, vol. 58, no. 2, pp. 628–646, 2009.

[3] L. Wang, R. Gao, J. Váncza, J. Krüger, X. V. Wang, S. Makris, and G. Chryssolouris, "Symbiotic human-robot collaborative assembly," *CIRP Annals*, vol. 68, no. 2, pp. 701–726, 2019.

[4] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "From structured english to robot motion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2717–2722.

[5] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the darpa urban challenge," in *IEEE International Conference on Automation Science and Engineering*, 2008, pp. 766–771.

[6] P. Schillinger, S. García, A. Makris, K. Roditakis, M. Logothetis, K. Alevizos, W. Rei, P. Tajvar, P. Pelliccione, A. Argyros, K. J. Kyriakopoulos, and D. V. Dimarogonas, "Adaptive heterogeneous multi-robot collaboration from formal task specifications," 2021, to be published.

[7] M. Guo, S. Andersson, and D. V. Dimarogonas, "Human-in-the-loop mixed-initiative control under temporal tasks," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 6395–6400.

[8] D. Oddoux and P. Gastin, "Ltl 2 ba : fast translation from ltl formulae to büchi automata." [Online]. Available: http://www.lsv.fr/~gastin/ltl2ba/

[9] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Motion and action planning under ltl specifications using navigation functions and action description language," in *IEEE/RSJ International Conference on Intelligent Robots and Systems:*, 2013, pp. 240–245.

[10] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 365–371.

[11] I. Havoutis and S. Calinon, "Learning assistive teleoperation behaviors from demonstration," 10 2016, pp. 258–263.

[12] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[13] J. R. Büchi, "On a decision method in restricted second order arithmetic," in *The Collected Works of J. Richard Büchi*. Springer, 1990, pp. 425–435.

[14] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[15] R. Baran, M. Guo, X. Tan, P. Varnai, and W. S. Cortez, "ltl˙automaton˙core." [Online]. Available: https://github.com/KTH-DHSG/ltl_automaton_core

[16] R. Baran, X. Tan, P. Varnai, P. Yu, S. Ahlberg, M. Guo, W. Shaw Cortez, and D. V. Dimarogonas. A human-in-the-loop, ltl planning and control ros package - turtlebot example demonstration. Youtube. [Online]. Available: https://youtu.be/9juAhYtq7aw

[17] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[18] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[19] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer-Verlag GmbH, Mar. 2017.

[20] R. Baran, M. Guo, X. Tan, P. Varnai, and W. S. Cortez, "Coin ms4 factory demonstration." [Online]. Available: https://github.com/KTH-DHSG/coin_ms4_demo

[21] R. Baran, X. Tan, P. Varnai, P. Yu, S. Ahlberg, M. Guo, W. Shaw Cortez, and D. V. Dimarogonas. A human-in-the-loop, ltl planning and control ros package - factory setting demonstration. Youtube. [Online]. Available: https://youtu.be/Gd-rqX04OdU