A Graphical Language for LTL Motion and Mission Planning

Shashank Srinivas, Ramtin Kermani, Kangjin Kim, Yoshihiro Kobayashi and Georgios Fainekos

Abstract—Linear Temporal Logic (LTL) has recently become a popular high-level specification language for robotic applications. One of the main reasons for the adoption of LTL is that LTL control synthesis algorithms are scalable while providing sufficient expressive power for a range of applications. However, despite the recent progress, one challenge remains. How can a non-expert robot user, who is not a logician, provide mission and motion plans for multiple robots in LTL? In this paper, we propose a graphical specification environment for LTL specifications that utilizes touchscreen technologies. We demonstrate that the graphical interface can express all the properties of interest that have appeared in the literature.

I. INTRODUCTION

More often than not, emergency personnel and armed forces personnel encounter situations where the use of autonomous robotic vehicles can reduce the risks to human life. A typical such scenario is when firefighters respond to a fire emergency in a big office building. Ideally, on the way to the fire site, the firefighters receive the building blueprints on their tablets and identify the areas of interest. Upon arrival to the site, a number of autonomous Unmanned Aerial Vehicles (UAV) are deployed, detect the fire areas, search for trapped humans and relay the information to the firefighters. The situation awareness will permit the firefighters to efficiently perform their tasks and minimize human casualties. The same scenario almost identically applies to urban counter-terrorism operations or nuclear disaster assessment and recovery operations or, even, to more everyday situations such as robots assisting medical personnel in hospitals.

Linear Temporal Logic (LTL) can capture such mission requirements as well as more complex ones [1]–[6] (for a recent survey see the special issue [7] in the IEEE Robotics and Automation Magazine). LTL has become a popular specification language due its expressive power and scalable control synthesis algorithms which can be tailored to specific applications. Moreover, recently, a number of issues relating to specification debugging and revision [8]–[10], on-line plan revision [3] and partial satisfaction of goals [4] are being studied. The latest developments build upon the LTL control synthesis and planning work to provide a complete set of tools for enabling Human-Robot Interaction (HRI).

However, LTL is a mathematical formalism that requires experts in order to successfully specify the desired goals in the logic. Therefore, we need intuitive human-robot interfaces to utilize for mission planning which can be interfaced with LTL control synthesis methods. In the past, structured or natural human languages have served as possible human-robot interfaces (see for example [11]–[14] and the references therein). Unfortunately, automatic speech recognition algorithms are not yet at the development level that could be deployed in emergency scenarios or where the robot needs additional human guidance. Moreover, natural language interfaces may not be the best way to instruct and receive information from a robot. That is, with the current technology in natural language understanding, it appears that touchscreen interfaces and visual programming tools [15]–[19] offer a better alternative for HRI [20].

The challenge in developing a touchscreen interface is not only on how to create a user-friendly interface, but also on how to capture the full expressivity of the specification language. In this paper, we develop a graphical representation language for LTL called LTLVIS. Our goal is twofold. First, we provide visual templates for commonly occurring specifications. The reasoning behind this is that certain specifications are hard to formalize in LTL even for expert logicians. Second, we design our framework to allow representation of arbitrary LTL specifications. The challenge in accomplishing the two aforementioned goals is that LTL specifications need to capture both topological and timing requirements. We demonstrate that all LTL properties of interest which are published in the literature can be captured by our graphical representation tool.

Related work: The need for software tools that can visualize complex temporal specifications was recognized early in the software verification community. In brief, it was found that the adoption of model checking methods by the industry was hurdled by the inability of the software engineers to understand, write and debug formal requirements in temporal logics. The authors in [21] provide an excellent recent overview of the past research on the topic. In brief, the prior work can be categorized in two main approaches: (1) visual representation of formulas and (2) visual representation of the desired and/or undesired sequences of events and their timing. The closest work to LTLVIS appeared in [22] and [23]. The work in [22] uses acyclic directed graphs to represent formulas of a variant of $\mu$-calculus. In [23], the authors construct a time-line representation of events which is then translated directly into automata. Our work here is fundamentally different from all the previous works because our graphical representation must both capture topological information and, also, it must be tailored to robotic applications for technologically inept users.
II. Problem Description and Solution Overview

In this paper, we are concerned with the problem of developing a visual specification language for human-robot interaction. Our main objectives are two. First and foremost, the visual specifications must be translated into Linear Temporal Logic (LTL) [2] in order to utilize the wealth of domain specific LTL control synthesis methods that have been recently developed. Second, the visual specification language has to be succinct and easy to be understood by non-expert LTL users, i.e., anyone who has not spent 1-2 years studying the subtleties of writing formal specifications in temporal logics.

In the following, we mainly focus the discussion to motion planning applications. We assume that we are given a finite set of labels Q = {q1, q2, ..., qn} in the workspace of the robot. The labels may represent locations (points-of-interest), static or dynamic obstacles, objects to be manipulated, actions to be performed at a specific location etc. We remark that the framework could be easily extended to support separate atomic propositions for the set of actions and Boolean sensors at the expense of more complex translation algorithms.

The following example briefly introduces the main setting of the targeted applications.

Example 1: The areas q1 – q5 are covered parking lots while the areas q6 – q9 are other points-of-interest (PoI).

Example 2: The graph in Fig. 1 is automatically translated in the LTL specification

\[ G(\neg q_6) \land F(q_5 \land F(q_3 \land F(q_4 \land FGq_1))) \]  

Additional requirements can be added on different screens to reduce overloading with too many symbols a single scene of the operational environment.

![Fig. 1. The user interface, the simple road network and the mission of Example 1. The areas q1 – q5 are covered parking lots while the areas q6 – q9 are other points-of-interest (PoI).](image-url)
III. Syntax and Semantics of LTL\textsubscript{VIS}

When designing a graphical representation language for arbitrary temporal logic formulas the main question is whether the visual formalism should retain the full expressivity of the targeted temporal logic. Clearly, a visual representation that is as expressive as the underlying logic can be developed using as basis the inductive definition of the semantics of the logic [26]. However, as noted in [21], such an approach does not help the non-expert users define correctly the desired specifications.

LTL\textsubscript{VIS} is a graphical representation language that can capture spatio-temporal specifications for robot motion and mission planning or, more general, for reactive supervisory control synthesis. It is a combination and extension of two previous approaches [22] and [23]. Namely, LTL\textsubscript{VIS} utilizes a graphical representation of the structure of the formulas as in [22], but at the same time achieves a time-line representation of events similar to [23].

Formally, an LTL\textsubscript{VIS} specification \(G\) is a directed graph embedded in a geometric space. Namely, \(G\) is a tuple \((V, E, v_0, c, L, L_\sigma, \Lambda, x)\) where

- \(V\) is a set of nodes;
- \(E \subseteq V \times V\) is the set of edges;
- \(v_0 \in V\) is a start node;
- \(c : V \rightarrow \{\text{green}, \text{red}\}\) is a function that colors each node either green or red;
- \(L : V \rightarrow \Phi_\Sigma(Q)\) labels each node with an LTL formula over the set of proposition \(Q\);
- \(\Lambda : E \rightarrow BO_1 \times BO_2 \times TO_1 \times TO_1\) is a function that labels each edge on the graph with one or more Boolean or temporal operators.

In detail,

\[
\begin{align*}
BO_1 & = \{\text{AND, OR}\} \\
BO_2 & = BO_1 \cup \{\varepsilon, \text{IMPLIES}\} \\
TO_1 & = \{\varepsilon, \text{FUTURE, ALWAYS}\} \\
TO_2 & = TO_1 \cup \{\text{NEXT, UNTIL}\}
\end{align*}
\]

where \(\varepsilon\) is the empty string, which is not displayed on the edge of the specification graph; and

- \(x : V \rightarrow \mathbb{R}^2\) is the position of the node on the map or, in general, on the image.

In our current implementation, we restrict the space to be 2D, but there is no theoretical restriction in developing an embedding of the specification graph in a 3D space.

The aforementioned graph definition is quite general and we impose some further restrictions to help the user formulate the desired requirements. The following are some syntactic restrictions on the graph \(G\). Such restrictions are enforced on-the-fly as the user builds the specification graph. Let \((b_1, b_2, t_1, t_2)\) be a triplet that \(\Lambda(u, v)\) maps to, then

- The combinations of Boolean and temporal operators which are allowed are presented in Fig. 2. There are some additional requirements:
  - \(b_2 = \text{AND}\) or \(b_2 = \text{OR}\) only if \(t_1 \neq \varepsilon\).
  - \(t_2 = \text{ALWAYS}\) or \(t_2 = \text{FUTURE}\) only if \(t_1 \neq \varepsilon\).

- Bidirectional edges appear only in Strongly Connected Components (SCC) [27] and the label of each edge of an SCC does not have any temporal operators, i.e., \(t_1 = \varepsilon\), \(t_2 = \varepsilon\).

- If the out-degree of node \(u\) is greater than 1, then all the outgoing edges \((u, v)\) must have the same label \(b_1\). That is, an LTL\textsubscript{VIS} graph resembles an and-or tree.

- If the label of the start node \(v_0\) is not modified by the user, then \(L(v_0) = \text{true}\).

The semantics of an LTL\textsubscript{VIS} specification are given through the translation algorithm to LTL formulas. The algorithm is presented in [24]. In the following, we give the intuition behind the syntax and semantics of specification graphs. Table I presents the main visual elements of the graph.

**Example 3:** Consider the following subgraph with labels \(L(1) = q_1\) and \(L(2) = q_2\).

![Example 3](image)

Intuitively, the left node must not hold now since it is red while the right node must hold in the future since it is green and it is preceded by a FUTURE label. The AND label combines the two aforementioned requirements to form the specification: \(\neg q_1 \land F q_2\).

**Example 4:** Consider the following subgraph with labels \(L(1) = q_1\), \(L(2) = q_2\) and \(L(3) = q_3\).

![Example 4](image)

The right subgraph converts to the LTL formula as in Example 3. The left subgraph on the other hand translates to \((\neg q_1) U q_3\). Since the outgoing edges from node 1 are dashed, the two subformulas are connected through a disjunction. Thus, the resulting specification is: \((\neg q_1 \land F q_2) \lor (\neg q_1) U q_3\).

**Example 5:** Consider the following subgraph with labels \(L(1) = q_1\) and \(L(2) = q_2\).

![Example 5](image)

The subgraph that includes nodes 1 and 2 translates to \(q_1 \Rightarrow X((\neg q_1) U q_2)\) according to the template. Thus, the resulting specification is: \(G(q_1) \Rightarrow X((\neg q_1) U q_2)\).

**Remark 1:** An LTL\textsubscript{VIS} graph can actually represent any arbitrary LTL formula since the labels of any node \(u\) on the graph can be any well formed LTL formula. This strikes a balance between the needs of expert users and the needs of
The green node indicates a location that must be visited. More generally, a green node indicates a Boolean sensor event that must be true or an action to be performed.

The red node indicates a location that must be avoided. More generally, a red node indicates a Boolean sensor event that must be false or an action to be stopped.

Solid edges between nodes indicate conjunctive requirements among the nodes that the edges connect.

Dashed edges between nodes indicate disjunctive requirements among the nodes that the edges connect.

When the AND label appears over an edge, it indicates a conjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.

When the OR label appears over an edge, it indicates a disjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.

When the IMPLIES label appears over an edge, it indicates that satisfaction of the SCC on the originating node of the edge implies satisfaction of the subgraph where the edge points to.

When the NEXT label appears over an edge, it indicates that the subgraph that the edge points to should be satisfied at the next step (when discrete time semantics are considered).

When the FUTURE label appears over an edge, it indicates that the subgraph that the edge points to should be satisfied some time in the future.

When the ALWAYS label appears over an edge, it indicates that the subgraph that the edge points to should always be satisfied in the future.

When the UNTIL label appears over an edge, it indicates that the SCC on the originating node should be satisfied until the point in time when the subgraph that the edge points to is satisfied.

non-expert users. The latter class of users can use only the graph specifications while the former can mix arbitrary graph and textual specifications.

### IV. IMPLEMENTATION AND USER INTERFACE

The software application is built on the Android platform using Java programming language and it can be downloaded from [25]. It has been tested on the Samsung galaxy tab 10.1 which runs Android Honeycomb 3.1. It has also been tested on an Android emulator based on JellyBean 4.1 with WXGA screen resolution. It uses the built-in AndroidGestureListener API to check for touch events like tap, doubletap, long press and scroll actions. It implements the DropBox Sync API to upload/update the LTL specification to a Dropbox folder on the cloud.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>TABLE II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRAPHICAL ELEMENTS OF LTL</strong>&lt;sub&gt;VIS&lt;/sub&gt;.</td>
<td><strong>TEMPLATE SYMBOLS OF LTL</strong>&lt;sub&gt;VIS&lt;/sub&gt;.</td>
</tr>
<tr>
<td>The green node indicates a location that must be visited. More generally, a green node indicates a Boolean sensor event that must be true or an action to be performed.</td>
<td>This indicates a special template that represents properties of the form ( q_1 \implies X((\neg q_1)Uq_2) ). It is activated by enabling the labels IMPLIES and NEXT over an edge when the template mode is active.</td>
</tr>
<tr>
<td>The red node indicates a location that must be avoided. More generally, a red node indicates a Boolean sensor event that must be false or an action to be stopped.</td>
<td></td>
</tr>
<tr>
<td>Solid edges between nodes indicate conjunctive requirements among the nodes that the edges connect.</td>
<td></td>
</tr>
<tr>
<td>Dashed edges between nodes indicate disjunctive requirements among the nodes that the edges connect.</td>
<td></td>
</tr>
<tr>
<td>When the AND label appears over an edge, it indicates a conjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.</td>
<td></td>
</tr>
<tr>
<td>When the OR label appears over an edge, it indicates a disjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.</td>
<td></td>
</tr>
<tr>
<td>When the IMPLIES label appears over an edge, it indicates that satisfaction of the SCC on the originating node of the edge implies satisfaction of the subgraph where the edge points to.</td>
<td></td>
</tr>
<tr>
<td>When the NEXT label appears over an edge, it indicates that the subgraph that the edge points to should be satisfied at the next step (when discrete time semantics are considered).</td>
<td></td>
</tr>
<tr>
<td>When the FUTURE label appears over an edge, it indicates that the subgraph that the edge points to should be satisfied some time in the future.</td>
<td></td>
</tr>
<tr>
<td>When the ALWAYS label appears over an edge, it indicates that the subgraph that the edge points to should always be satisfied in the future.</td>
<td></td>
</tr>
<tr>
<td>When the UNTIL label appears over an edge, it indicates that the SCC on the originating node should be satisfied until the point in time when the subgraph that the edge points to is satisfied.</td>
<td></td>
</tr>
</tbody>
</table>

### A. The User Interface

The User interface (see Fig. 1) consists of buttons to select a Map, to add Missions, to delete Missions and to Upload the motion plan to Dropbox on the top row. It contains checkboxes to select waypoints, which may also include actions and sensors, for the current mission. A waypoint represents a location on a map and each location contains an atomic propositions used to construct the LTL specification. The waypoint represents a location by the virtue of its ID and not by its current coordinates on the map. Future versions of the toolbox will have tighter integration with the map by also designing the regions that the waypoints correspond to. The UI consists of a text view on the left side that shows relevant context which include the missions that are currently added and changes to the final specification that is uploaded when the upload button is pressed. Next to it, it is main area for creating and editing missions graphically. On the right side, the user will find a list of robots that can be selected to upload the plan to a specific robot. A preview of the LTL specification is shown at the bottom of the interface. The interface also pops up a context menu (see [24]) on a log press touch event on waypoints which allows the user to select more options for creating the motion plan.

### B. Creating a motion plan

The first thing to do is to load a map by pressing the *Select Map* button (top left on the user interface Fig. 1). This redirects the user to the gallery application present on Android allowing the user to select the desired map. The maps are generic images with waypoint information representing each location integrated into the map. The next step is to start building the motion plan by selecting the waypoints required for the current mission. The waypoints appear on the top of the draw area in the ascending order of their IDs. The user is now free to drag the waypoints around to their respective locations on the map. It has to be noted that this step is not a requirement and only helps the user visualize the motion plan better. In this prototype version, the position of the nodes is not forwarded to the LTL planner, but rather it is assumed that the positions of the points-of-interest are prespecified. The main reason for this choice is that there is no common format for all the planners being developed by the different research groups. One of the future implementation goals will be to develop such a format by reviewing toolboxes like [1], [13].

There are a number of gestures and options to create a motion plan which the users should be able to familiarize themselves with little or no effort. Each of the gestures available in the UI is explained below:
TABLE III
EXPLANATION OF SOME OF THE ICONS IN THE POP-UP MENU IN [24]

| SET LABEL | Change the label of a node on the graph. |
| VISIT/AVOID | This option tells the robot whether to visit the location or to avoid it. |
| ALWAYS/ONCE | It specifies whether the robot should perform the specified action once or always (or infinitely often depending on the subgraph). |
| TOGGLE GF/FG | Toggle between ALWAYS EVENTUALLY and EVENTUALLY ALWAYS. |
| TOGGLE IMPLIES | Change between IMPLIES, AND and OR of an edge. |
| TOGGLE FUTURE | Enable or disable a FUTURE label on the edge. |
| TOGGLE UNTIL | Enable or disable an UNTIL label on the edge. |
| TOGGLE UNTIL PICKUP OBJECT | Enable of disable a NEXT label on the edge. |
| DROP OBJECT | This option tells the robot to drop the object at the waypoint specified (application specific). |

1) Each circle (green/red) is a waypoint and represents a particular location on a map regardless of where it is placed on the map.
2) By default, enabling a location puts a waypoint on the map indicating that the robot should eventually visit the location.
3) Enabling multiple waypoints tells the robot to go to each location with no particular order.
4) The interface supports the following gestures:
   a) Single tap: a single tap on two consecutive waypoints changes the predicate from AND to an OR (E.g. Visit A AND B now becomes visit A OR B)
   b) Double tap: Double tapping two consecutive waypoints produces sequential ordering from the first point to the next.
   c) Long press: Long pressing a waypoint results in a menu being displayed (see [24]) which gives further options as indicated in Table III.
5) There are several other buttons to select a Map, to add/delete missions and to upload the missions created, to a server that computes the trajectory for the robot from the uploaded LTL specification.

V. EXAMPLES

The following section describes a few cases based on the work by Smith et. al. [28]. The work in [28] was selected due to complex requirements that it includes. The scenario considered is a collect, transfer and upload mission. In brief, one or more robots need to visit a number of data collect (q1, q2, q3) and data upload locations (q4, q5) on a road network. Further details on the examples can be found in [28].

**Case A:** Repeatedly visit one of the data collect locations q1, q2 or q3 and repeatedly visit one of the data upload locations q4 or q5. The requirements can be specified using the LTL\_VIS graph in Fig. 3. Note that graph nodes 1, 2 and 3 and nodes 4 and 5 are connected with undirected dashed edges which indicate disjunction between the labels of the nodes. The resulting formula is:

\[
GF(q1 \lor q2 \lor q3) \land GF(q4 \lor q5)
\]

**Case B:** In addition to Case A, avoid visiting two upload locations without collecting data in between. In order to graphically represent this requirement with the algorithm presented in [24], two graph specifications are required; one for

\[
\psi_1 = X(\neg (q4 \lor q5) U (q1 \lor q2 \lor q3))
\]

and one for

\[
G((q4 \lor q5) \implies \psi_1)
\]

To enable the user enter such specifications on a single screen we can use the template as in Fig. 4. In order to use the template the edge must be labeled with IMPLIES and NEXT and the template mode must be activated. Thus, the graph as it appears in Fig. 4 captures the requirement:

\[
G((q4 \lor q5) \implies X(\neg (q4 \lor q5) U (q1 \lor q2 \lor q3)))
\]

**Case C:** In order to specify that the data gather locations q1, q2 and q3 must be visited periodically, we can visualize the specification with the graph as given in Fig. 5 resulting in the LTL formula:

\[
GFq1 \land GFq2 \land GFq3 \land GF(q4 \lor q5)
\]

**Remark 2:** The experienced LTL user may notice that taking the conjunction of many different requirements, as
Moreover, with LTL expert users who do not need guidance in building the desired primitives and shapes for capturing locations, objects etc, robot. Furthermore, LTL entering high-level plans in the immediate workspace of the robot. Thus, allowing for entering high-level plans in the immediate workspace of the robot. Furthermore, LTL\textsubscript{VIS} can immediately be utilized over live video feed using kinect and, thus, allowing for entering high-level plans in the immediate workspace of the robot. Furthermore, LTL\textsubscript{VIS} can be extended with geometric primitives and shapes for capturing locations, objects etc, as well as with sketches for more immediate robot control. Finally, in the future, we plan to contact usability studies to determine possible improvements to the user interface.

VI. CONCLUSIONS

We have presented LTL\textsubscript{VIS} which is a graphical representation tool for specifying Linear Temporal Logic (LTL) formulas. LTL\textsubscript{VIS} can represent arbitrary LTL formulas for expert users who do not need guidance in building the desired specifications. Moreover, with LTL\textsubscript{VIS} the technologically inept user can now utilize specific spatio-temporal graphs and templates in order to construct the desired specification. Even though in this paper we present the basic toolbox with a basic functionality, the extensions of this work can be numerous. First and foremost, LTL\textsubscript{VIS} can immediately be utilized over live video feed using kinect and, thus, allowing for entering high-level plans in the immediate workspace of the robot. Furthermore, LTL\textsubscript{VIS} can be extended with geometric primitives and shapes for capturing locations, objects etc, as well as with sketches for more immediate robot control. Finally, in the future, we plan to contact usability studies to determine possible improvements to the user interface.

ACKNOWLEDGEMENTS

The authors would like to thank Parth Pandya for his help with the experiments.

REFERENCES


