A Graphical Language for LTL Motion and Mission Planning

by

Shashank Srinivas

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2013 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Chitta Baral
Winslow Burleson

ARIZONA STATE UNIVERSITY

May 2013
ABSTRACT

Linear Temporal Logic is gaining increasing popularity as a high level specification language for robot motion planning due to its expressive power and scalability of LTL control synthesis algorithms. This formalism, however, requires expert knowledge and makes it inaccessible to non-expert users. This thesis introduces a graphical specification environment to create high level motion plans to control robots in the field by converting a visual representation of the motion/task plan into a Linear Temporal Logic (LTL) specification. The visual interface is built on the Android tablet platform and provides functionality to create task plans through a set of well defined gestures and on screen controls. It uses the notion of waypoints to quickly and efficiently describe the motion plan and enables a variety of complex Linear Temporal Logic specifications to be described succinctly and intuitively by the user without the need for the knowledge and understanding of LTL specification. Thus, it opens avenues for its use by personnel in military, warehouse management, and search and rescue missions. This thesis describes the construction of LTL for various scenarios used for robot navigation using the visual interface developed and leverages the use of existing LTL based motion planners to carry out the task plan by a robot.
DEDICATION

To My Parents
ACKNOWLEDGEMENTS

I would like to express my deepest appreciation and gratitude to my advisor and committee chair, Dr. Georgios Fainekos, for giving me an opportunity to work on this exciting research topic and for providing invaluable amount of ideas, mentoring, support and patience overseeing my research. I would like to thank the committee members, Dr. Chitta Baral and Dr. Winslow Burleson for agreeing to be a part of my committee and providing ideas and feedback. I would like to thank my labmates Adel Donkachi, Bardh Hoxha, Hengyi Yang, Kangjin Kim, Parth Pandya, Ramtin Kermani, Shih-Kai Su and Subrat Kumar Swain for providing support. I would like to thank Parth Pandya, Kangjin Kim and Ramtin Kermani for providing the resources for the experiments conducted. I would also like to thank my friends Hyma Acharya, Tosha Shah, Shamath Kumar, Sailesh Kandula, Nikhil Pratap, Vaibhav Kumar Upadhyaya, Rushil Anirudh, Anu Mercian, Suhas Ranganath and Prasad Ramakrishna for making this experience a pleasant one. Finally, I would like to thank my parents and my sister for always being supportive and encouraging me with their best wishes, without whom this journey would have been very difficult. This work was partially supported by a grant from the NSF Industry/University Cooperative Research Center (I/UCRC) on Embedded Systems at Arizona State University.
# TABLE OF CONTENTS

| LIST OF TABLES                                      | vi      |
| LIST OF FIGURES                                    | vii     |

## CHAPTER

1 INTRODUCTION ...................................................... 1
   1.1 Contribution of thesis .................................. 6
   1.2 Thesis Structure ........................................... 6

2 BACKGROUND .......................................................... 8
   2.1 Linear Temporal Logic ...................................... 8
      Syntax ......................................................... 8
      Semantics ................................................... 9
      Applications ............................................... 11
   2.2 Android ........................................................... 12
      Android Architecture ........................................ 12

3 RELATED WORK .......................................................... 14
   3.1 Sketch based Navigation .................................... 14
   3.2 Natural Language Interface ................................. 15
   3.3 Qualitative studies on Human Robot Interfaces .......... 16
   3.4 Linear Temporal Logic MissOn Planning (LTLMOPT) Toolkit 16
   3.5 Timeline Editor ............................................. 18
   3.6 Visual Specification of Branching Time Temporal Logic .. 19
   3.7 Graphical Interval Logic .................................... 19
   3.8 Property Specification Patterns ............................ 20

4 PROBLEM DESCRIPTION .................................................. 21
   4.1 Problem Overview ............................................ 21
   4.2 Solution Overview ............................................ 23
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Advantages and Disadvantages of various communication mediums</td>
<td>5</td>
</tr>
<tr>
<td>5.1 Graphical elements of LTL\textsubscript{VIS}</td>
<td>30</td>
</tr>
<tr>
<td>5.2 strict ordering visual specification pattern symbol of LTL\textsubscript{VIS}</td>
<td>31</td>
</tr>
<tr>
<td>5.3 Explanation of icons in the pop-up menu in Fig. 4.2</td>
<td>36</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Firefighting in a disaster management scenario using UAV</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Visual specification of firefighting scenario</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>LTL: Semantic intuition</td>
<td>10</td>
</tr>
<tr>
<td>4.1</td>
<td>The user interface, the simple road network and the mission of Example 1</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Pop-up menu of a node</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>The allowed combinations of Boolean and temporal operators over an edge</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>Strict ordering visual specification pattern</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>Software overview</td>
<td>34</td>
</tr>
<tr>
<td>6.1</td>
<td>Periodically visit data upload locations and data gather locations</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>Visit an upload location only after data has been gathered</td>
<td>39</td>
</tr>
<tr>
<td>6.3</td>
<td>Visit an upload location only after data has been gathered (visual specification pattern mode)</td>
<td>40</td>
</tr>
<tr>
<td>6.4</td>
<td>Data gather locations $q_1, q_2, q_3$ must be visited periodically</td>
<td>41</td>
</tr>
<tr>
<td>6.5</td>
<td>Visit an upload location after gathering data</td>
<td>42</td>
</tr>
<tr>
<td>6.6</td>
<td>Visit an upload location after gathering data(visual specification pattern mode)</td>
<td>43</td>
</tr>
<tr>
<td>6.7</td>
<td>Avoid $q_1$ and $q_2$ until $q_3$ is reached</td>
<td>44</td>
</tr>
<tr>
<td>6.8</td>
<td>If $q_3$ is visited, then visit $q_1$ and $q_2$ in that order while enforcing that $q_2$ and $q_3$ are not visited on the way to $q_1$ and that $q_1$ and $q_3$ are not visited again on the way to $q_2$.</td>
<td>45</td>
</tr>
<tr>
<td>6.9</td>
<td>Avoid road connecting $q_6$ and $q_7$.</td>
<td>46</td>
</tr>
<tr>
<td>6.10</td>
<td>Data from location $q_3$ must be uploaded only at location $q_4$.</td>
<td>47</td>
</tr>
<tr>
<td>7.1</td>
<td>Initial location of the robot</td>
<td>48</td>
</tr>
<tr>
<td>7.2</td>
<td>On its way to the first location $q_5$</td>
<td>49</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>7.3</td>
<td>Reached location q5</td>
<td>49</td>
</tr>
<tr>
<td>7.4</td>
<td>On its way to location 3 after making a turn at the intersection near location 1</td>
<td>50</td>
</tr>
<tr>
<td>7.5</td>
<td>Robot almost near location 3</td>
<td>50</td>
</tr>
<tr>
<td>7.6</td>
<td>Robot in location 3</td>
<td>51</td>
</tr>
<tr>
<td>7.7</td>
<td>Robot moving towards location 4</td>
<td>51</td>
</tr>
<tr>
<td>7.8</td>
<td>Robot in location 4</td>
<td>52</td>
</tr>
<tr>
<td>7.9</td>
<td>Robot at its final location 1</td>
<td>52</td>
</tr>
<tr>
<td>7.10</td>
<td>Trajectory followed by the robot</td>
<td>53</td>
</tr>
</tbody>
</table>
Human-Robot Interaction (HRI) will increasingly be a part of everyday life, beginning with scenarios where use of autonomous vehicles reduces the risk to human lives in emergency situations like firefighting, nuclear disaster assessment, urban counter terrorism and other military applications. An ideal scenario would be a situation where firefighters respond to a fire in an office building. The firefighter would load the blueprints of the building on a tablet and deploy multiple Unmanned Aerial Vehicles (UAVs) to search for people trapped in the building, look for areas of fire and relay the information back to the firefighter and is depicted in Fig 1.1. The situation awareness allows firefighters to effectively perform their tasks and minimize human casualties. This will require the firefighters, who are non-experts to be able to communicate with robots effectively without much hassle. We need an efficient way to give instructions to robots to perform tasks in a way which is precise and unambiguous. For the robots to perform these tasks, we require the composition of high-level planning with low level controllers [1, 2]. One of the approaches for motion planning at the highest level involves the specification language called Linear Temporal Logic (LTL) [3]. LTL has been successfully used for path planning of single robots [4] as well as robotic swarms [5] because of its ability to provide a mathematical framework to bridge the gap between natural language and high-level planning algorithms, and high-level planning algorithms and control.

Despite technological advancements, a fully autonomous vehicle is still not feasible mainly because of two factors, robot localization and situation awareness. Autonomous agent localization and obstacle avoidance issues will be resolved soon with light to current progress in robotics shown by the DARPA grand challenge [6], which showcased the latest efforts made in perception algorithms and sensors with regard to localization and obstacle avoidance. However, the situation awareness problem has been
Figure 1.1: Firefighting in a disaster management scenario using UAV

hard to solve and remains so for the foreseeable future. An example would be distinguishing between a friend and a foe in a counter terrorism scenario, or a healthy or injured human being in a disaster zone. Such situations require human input and thus requires the robots to be semi autonomous. The latter claim is also supported by the fact the situations mentioned are also highly dynamic. It is very likely that certain mission goals for the robot become infeasible as the mission progresses due to unforeseen situations. For example, a corridor in a nuclear plant might become inaccessible due to debris requiring the mission goals to be revised. Even if the mission objectives can be accomplished, the personnel who are under time pressure in emergency situations might give conflicting mission goals to autonomous agents. The autonomous agents must be capable to providing feedback to the human operator under such situations as well as recommendations on what could be accomplished or revised on the specification.
Therefore intuitive human robot interfaces are needed, to utilize for mission planning, which can be interfaced with LTL without the Human operators needing to know mathematical formalisms such as LTL. We should be able to use natural language and/or visual representations to precisely define the robot tasks. Structured English has been used [7,8] for motion planning which incorporates LTL. Unfortunately, automatic speech recognition algorithms are not advanced enough to be deployed in emergency situations and moreover, natural language interfaces might not be the best way to instruct and receive information to/from the robots. Sketches provide a holistic view of the mission plan compared to other methods like structured English or speech recognition. The user is capable of understanding the plan much quicker and does not need to wade through English sentences to understand what the robot is doing or supposed to do. Also motion planning is inherently visual in nature and the motion plans are easily produced or understood by the user using a mental visual map.

Visual modeling/representations provide another intuitive way to interface with existing LTL planning methods (see section 3.1). Sketches have been used to enable robots to follow trajectories drawn on touch screen interfaces like PDAs which use techniques like Hidden Markov Models (HMM) for gesture recognition [9–12]. This thesis proposes to extend and combine these two areas by providing a mechanism to translate a visual representation of the task plan into a task plan described in LTL. The research concentrates on using a touch based interface, in this case, an Android tablet, to allow a visual description of the robot’s tasks to be translated into LTL. The challenge here is to not only create a user friendly interface, but to also be able to capture the full expressivity of the specification language.

Consider the simple firefighting scenario depicted in 1.1 where the robot must visit particular buildings in sequential order. Lets call the buildings $q_1$, $q_2$, $q_3$ and $q_4$. The LTL specification for visiting the buildings $q_1$, $q_2$, $q_3$ and $q_4$ in that order (without strict
ordering - see section 5.2) would be as follows:

\[ F(q_1 \land F(q_2 \land F(q_3 \land F(q_4)))) \]

As can be seen, even a simple specification like this results in high nesting of temporal operators and non-expert users will not be able to use LTL for specifying motion plans. This thesis provides a visual language to allow non expert users to specify motion plans intuitively without needing to know LTL. The graphical specification for the motion plan is given in the Figure 1.2.

![Figure 1.2: Visual specification of firefighting scenario](image)

The visual specification language is hypothesized to provide an intuitive and expressive interface for providing high level motion plans based on the heuristics developed after doing research on alternative communication strategies and analyzing requirements needed for mission critical scenarios. The advantages and disadvantages of various communication mediums for mission critical scenarios is provided in Table
Table 1.1: Advantages and Disadvantages of various communication mediums

<table>
<thead>
<tr>
<th>Medium</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sketches</td>
<td>Natural to humans, easy to understand</td>
<td>Ambiguity in gesture recognition</td>
</tr>
<tr>
<td>Speech recognition</td>
<td>Natural to humans</td>
<td>Ambiguity in speech, error prone due to noise and variability in human speech, time consuming to modify midway</td>
</tr>
<tr>
<td>Written Structured English</td>
<td>Natural to humans, precise and unambiguous</td>
<td>time consuming on tablet platforms</td>
</tr>
<tr>
<td>Graphical Visualization using LTLVIS</td>
<td>Natural to humans, precise and unambiguous, fast specification, easy to modify midway</td>
<td></td>
</tr>
</tbody>
</table>

This thesis proposes a few heuristic rules that are to be followed for Human Robot Interaction (HRI) using a visual specification language:

- **Heuristic rule 1**: The user interface must be easy to use without a high learning curve.

- **Heuristic rule 2**: Communication of the motion plan should be precise and unambiguous.

- **Heuristic rule 3**: The motion plan must be easy to modify midway quickly to account for dynamic circumstances.

- **Heuristic rule 4**: The interface should be tolerant to environment noise i.e., environmental factors should not cause the motion plan communicated to be lost during user input.
An in-depth research of the qualitative and quantitative effectiveness of these heuristics will be performed in the future as part of usability studies.

1.1 Contribution of thesis

There are two main goals to be achieved. First, to provide visual specification patterns for commonly occurring specifications, the reason being that certain specifications are hard to formalize even for expert logicians. Second, to design the framework allowing the creation of arbitrary LTL specifications.

It consists of a graphical user interface that provides graphical primitives to describe a task plan visually and convert this task plan into LTL which is transmitted to a server in the back end. The tool consists of various primitives in order to construct high level task plans for the purpose of navigation of robots in the field. It involves research on ways to qualitatively improve and intuitively represent various aspects of the formalisms of LTL graphically so that users are able to use the tool without prior knowledge of LTL. It provides a quick way to create task plans and an alternative to structured English or speech. It has to be noted that this tool can be potentially used in an education setting to teach students about Linear Temporal Logic since it helps students visualize the specifications providing better understanding of the intricacies of temporal formalisms.

1.2 Thesis Structure

This thesis introduces LTL\textit{VIS}, a graphical specification language and the resulting graphical tool used to visually describe the LTL specification for motion planning. The structure of the document is as follows:

- Chapter 1: This chapter introduces the motivation behind the thesis and outlines the overall structure of the document.
- Chapter 2: This chapter discusses the background knowledge required to follow through the rest of the thesis.
• Chapter 3: This chapter discusses the related research that has been done in this area and also discusses similar tools that are being used involving temporal logic.

• Chapter 4: This chapter describes the problem and an overview of the solution.

• Chapter 5: This chapter introduces the graphical specification language LTLVIS used to create visual representations of motion plans, the development of the tool and various features it implements and supports.

• Chapter 6: This chapter discusses various examples that use LTLVIS exhibiting its expressivity and ease of use.

• Chapter 7: This chapter discusses the experiments conducted and demonstrates the specifications running on an actual robot.

• Chapter 8: This chapter discusses the conclusion and possible future work in this area of research.
Chapter 2

BACKGROUND

2.1 Linear Temporal Logic

Temporal logic is used for robot motion planning, among others, since it is closely related to structured English but has the advantage of being a mathematical formalism and hence is precise and unambiguous. Due to this advantage, they are also used in other fields to represent properties and requirements of systems for computer programs.

Syntax

LTL is standard propositional logic along with temporal operators such as always ($G$), eventually ($F$), next ($X$) and until ($U$). With these additional operators, it becomes possible to express the truth values of propositions over time. Using LTL, interesting properties like coverage, sequencing, conditions and reachability can be expressed and is discussed later.

More complex scenarios can be composed using these basic operators. Because of this, LTL is a very convenient way to express the high level task plan for robot navigation. The next section describes LTL and the temporal operators to provide a clear understanding of LTL specifications:

Consider $P$ to be a set of atomic propositions. The set of propositions include regions on a map and other elements like the robot’s sensors. The atomic propositions $\pi \in P$ is used to construct LTL formulas according to the following grammar,

$$\phi ::= true \mid \pi \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid X \phi \mid \phi_1 U \phi_2$$

where $\neg$ denotes negation, $\lor$ denotes disjunction, $X$ denotes next time operator and $U$ is the until operator. From these operators, we can further define conjunction ($\land$) and implication ($\Rightarrow$). The Eventually operator $F$ is define as $F \phi = true U \phi$ and the always operator $G$ as $G\phi = \neg F \neg \phi$ [8]. The temporal operators are informally defined as follows:
• $X\phi$ denotes that $\phi$ is true in the next step (the next time step in the sequence).

• $G\phi$ denotes that $\phi$ is true at every point of time in the sequence

• $F\phi$ denotes that $\phi$ is true at some point in the future (including the present)

• $GF\phi$ denotes that $\phi$ is true infinitely often (always eventually)

• $FG\phi$ denotes that $\phi$ is true eventually and stays true for all future after it becomes true (eventually always)

• $\phi_1 U \phi_2$ denotes that $\phi_1 U \phi_2$ hold at the current moment, if $\phi_2$ hold sometime in the future and $\phi_1$ is true until that future moment.

The properties mentioned earlier can now be discussed in context with LTL as follows:

• Coverage: The formula $F\pi_1 \land F\pi_2 \land \cdots \land F\pi_m$ specifies that the robot should visit the required regions $\pi_1, \pi_2, \ldots, \pi_m$ eventually in no particular order.

• Sequencing: The formula $F(\pi_1 \land F(\pi_2 \land F\pi_3))$ specifies that the robot should visit the regions $\pi_1, \pi_2$ and $\pi_3$ in that order.

• Reachability with avoidance: The formula $\neg(\pi_1 \lor \pi_2 \lor \cdots \lor \pi_n) U \pi_{n+1}$ specifies that the robot should avoid certain location(s) until it reaches the required locations.

More complex specifications can be built by composing the basic specifications using logic operators (see section 6).

Semantics

LTL formulae are properties of an execution path, and in this particular case, the path of the robot. This formula will either be satisfied or falsified. The semantics of an LTL formula
\( \phi \) is defined as a language \( \text{Words}(\phi) \) that contains all infinite words over the alphabet \( 2^{AP} \), where \( AP \) is the set of atomic propositions, that satisfy \( \phi \) \cite{13}. The intuitive semantics of LTL are provided in the figure 2.1.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{ltl_semantics.png}
\caption{LTL: Semantic intuition}
\end{figure}

Let \( \phi \) be an LTL formula over \( AP \). The property induced by \( \phi \) is:

\[
\text{Words}(\phi) = \left\{ \sigma \in (2^{AP})^\omega \mid \sigma \models \phi \right\}
\]

where the satisfaction relation \( \models \subseteq (2^{AP})^\omega \times \text{LTL} \) is the smallest relation with properties of the LTL semantics (satisfaction relation \( \models \)) for infinite words over \( 2^{AP} \) as follows:
\[ \sigma \models \text{true} \]
\[ \sigma \models a \quad \text{iff} \quad a \in A_0 \]
\[ \sigma \models \phi_1 \land \phi_2 \quad \text{iff} \quad \sigma \models \phi_1 \text{ and } \sigma \models \phi_2 \]
\[ \sigma \models \neg \phi \quad \text{iff} \quad \sigma \not\models \phi \]
\[ \sigma \models X\phi \quad \text{iff} \quad \sigma[1...] = A_1A_2A_3... \models \phi \]
\[ \sigma \models \phi_1 U \phi_2 \quad \text{iff} \quad \exists j \geq 0. \sigma[j...] = \phi_2 \text{ and } \sigma[i...] = \phi_1, \text{ for all } 0 \leq i < j \]
\[ \sigma \models F\phi \quad \text{iff} \quad \exists j \geq 0. \sigma[j...] = \phi \]
\[ \sigma \models G\phi \quad \text{iff} \quad \forall j \geq 0. \sigma[j...] = \phi \]
\[ \sigma \models GF\phi \quad \text{iff} \quad \exists^\infty j. \sigma[j...] = \phi \]
\[ \sigma \models FG\phi \quad \text{iff} \quad \forall^\infty j. \sigma[j...] = \phi \]

For \( \sigma = A_0A_1A_2... \in (2^{AP})^\omega \), \( \sigma[j...] = A_jA_{j+1}A_{j+2}... \) is the suffix of \( \sigma \) starting in the \((j + 1)\) symbol \( A_j \)

**Applications**

LTL has been used as a specification language in numerous robotics applications \([14-24]\). In \([14]\), a framework for single mobile robot with second order dynamics is presented. Reactive planning and distributed controller synthesis for multiple robots has been addressed in \([19]\) and control of multiple robots using LTL has been addressed in \([20,22]\). Research in humanoid dancing methods have also used LTL \([24]\).

The next section briefly introduces the Android platform on which the application is built.
2.2 Android

Android is a very popular mobile operating system from Google and is used widely in smartphones and tablets. It is open source and allows anyone to develop applications for its platform free of charge. The decision to develop on this platform was one of convenience and availability of these tablets in the lab. The application can be ported to other mobile operating systems like iOS in the future. The next section gives a brief introduction to the Android architecture.

Android Architecture

The Android system is based on a linux kernel (version 2.6) serving as the hardware abstraction layer and provides memory management, file system management, process management, networking and security. A bunch of software libraries sit on top of it and provide support for the user application. They include SQLite for storage and database management, WebKit for rendering webpages, OpenGL ES for graphics, media codecs and many other libraries. This helps application programmers create rich applications with ease.

Android programs are mainly written in Java programming language and the java files produced are compiled to a Dalvik executable and run on the Dalvik Java Virtual machine. Android provides an application framework to be used by application developers and provides features for package installation, telephony management, resource management, location services and many others.

There are four main components in an Android application:

1. Activities: These are the main UI components in the application. Each screen of the application is a separate activity.

2. Intents: These specify some action that needs to be performed like retrieving an
image and describes what action has to be performed. Applications or activities registered to process such intents can then act on these intents to perform actions on behalf of the activity which produced the intent.

3. Services: These are background tasks that need to be performed and do not have a UI attached to them. An example would be playing music files even when the music player is not in view.

4. Content providers: These enable applications to share data like the address book which allows users to query an application to retrieve and update its data.

The application developed mainly uses the Android graphics library for drawing graphics primitives and displaying text, and gesture APIs to listen to touch events which are used to manipulate the graphical primitives. It also connects to the Drobox cloud service using their DropBox Sync API. The software application has been developed using the Eclipse IDE which has plugins for Android development making the process of Android application development seamless.
Chapter 3

RELATED WORK

This thesis presents the idea of describing a motion plan with a set of connected graphical primitives and the subsequent conversion of it into Linear Temporal Logic. The following section discusses research on interfaces that have been used to control robots and the use of Linear Temporal Logic for motion planning.

3.1 Sketch based Navigation

Sketch based interfaces essentially use a touchscreen to input a hand drawn sketch of the actions to be performed by a robot or a group of robots based on the context.

Sketch based interfaces have been implemented in [9,11,12,25,26] for the purpose of robot navigation. In [12], they describe a sketch interface where users use stroke gestures on a computer screen showing a live camera view of the room in order to control an iRobot Roomba, a vacuuming robot. They use ARToolkit to recognize objects in the room and use a stroke recognition algorithm to recognize the following gestures:

- A line starting from the robot to move the robot along a path.
- A lasso to vacuum a certain area.
- Other gestures for auxiliary commands to pause, resume, stop and go home.

In [11], they use a similar overhead camera, stroke recognition and ARDev, an augmented reality toolkit. They also include a 3D model of the robot rendered in OpenGL which can be manipulated to reflect the changes on the real robot. They also discuss highlighting objects in the camera view by drawing a circle around objects for the robot to perform some action. They avoid using maps that are sketched and instead go with live camera views in order to avoid calibration issues between the real world and the map.
In [9], they describe work to provide basic multi-robot formations such as follow the leader and march side by side which use Hidden Markov Models to recognize gestures from sketches which are drawn on PDAs and tablet PCs. The basic procedure is to:

- Capture the pixel information (x-y coordinates of the sketch).
- Perform feature extraction.
- Apply Hidden Markov Model (HMM) and perform post-processing to classify the symbol belonging to a finite set of symbols.

This research uses hand drawn gestures to indicate various actions to be performed by the robots and presents the idea of using graphical primitives to perform various robot actions which is one of the foundations of this thesis. It also presents the idea of using cellphones and PDAs as modes of input for the sketches which have now evolved into today’s powerful smartphones and tablets allowing greater programming flexibility and computing power.

Sketches have also been modeled as Variable Duration Hidden Markov Models (VDHMM) in [25] supporting flexible and multi-stroke gestures and the ability to extend the work to incorporate multimodal communication such as verbal cues. It has to be noted however, that all the approaches above use some kind of probabilistic modeling to recognize freeform sketches which might not be suitable in emergency situations where there is no room for false gesture recognition. There is also a need for the users to learn the various gestures which becomes increasingly difficult as the motion plans get more complex and the robots acquire more capabilities.

3.2 Natural Language Interface

Natural Language Interfaces have been used for the purpose of Human Robot Interaction [7][8][27][28]. A natural language interface uses linguistic aspects such as verbs, clauses
and phrases to describe syntax or commands to a computer. One approach is the use of structured English which is a subset of the English language and imbibes the syntax of structured programming. It combines the benefits of both programming languages and natural language, and hence has the benefit of being unambiguous and easy to understand. A structured English Language has been built which maps directly into LTL [6] [7] in an effort to build a natural language interface for LTL planning methods. They do this by using a simple grammar which results in sentences in the given structured language. They then provide semantics for the sentences with respect to LTL formulas. This research provides the idea of mapping structured English primitives to graphical primitives and/or options as part of the Graphical User Interface (GUI) for this thesis.

3.3 Qualitative studies on Human Robot Interfaces

Research has been done examining the issues related to robotic operator performance and different user interface solutions have been reviewed in regard to its use in military and civilian applications. A particular section of [29] examines the user interface designs for the control of semi autonomous robots using cell phones and PDAs. They also examine sketch interfaces described above and state the limitations of using such interfaces that the system needs to consistently interpret the stylus inputs of the user, which vary from person to person and also on different occasions by the same user. Users use spatial reasoning techniques to create course-of-action diagrams. Their study also encompasses the use of multimodal interfaces such as natural language and gestures and the use of vibrotactile displays. All these studies steers the research towards a more robust and simple approach of using the notion of waypoints and other pre-determined graphical primitives built into the software that the user can use to construct motion plans.

3.4 Linear Temporal Logic MissiOn Planning (LTLMOP) Toolkit

This is a software package that is used to build hybrid robot controllers from structured English and LTL specifications. It has been explained in detail in [28]. This tool has three
separate stages:

1. Parsing structured English to LTL which has been discussed in a previous section.

2. Synthesizing an automaton from LTL.

3. Automaton to hybrid controller which generates continuous control commands using the transitions between states of the automaton.

The GUI of the LTLMOP toolkit is made modular and consists of the following components:

- A specification editor, for editing, compiling and executing structured English specifications.
- A Region editor, for creating named polygonal regions in the map and to generate topological connectivity graphs.
- A structured English parser that converts structured English into an equivalent LTL specification.
- A control synthesizer that checks for reliability and generates a control automaton from the LTL specification.
- A calibration tool that maps the real world coordinates to map coordinates.
- A controller executor that executes the controller in a simulation or real world environment.
- A simulation monitor that provides real time information about the controller’s execution.
The ultimate goal of the current research is to provide a similar set of tools along with the graphical specification language replacing the structured English parser.

3.5 Timeline Editor

The Timeline Editor is a graphical tool used to input formal specifications for logic model checking in the form of a timeline as expressing complex requirements in logic is quite challenging [30]. Logic model checking is mainly used in scenarios involving debugging control logic and communication errors. Logic model checking is performed by running model checking on a model of the source and a model of the requirements, if there is one, and evaluating error traces to determine the errors, if any, is due to the source code, requirements, or both, and making corrections as needed.

The TimeLine Editor is mainly used to express requirements which have a preamble (a pattern matching an execution sequence) and a response. It is represented graphically as a horizontal bar with time progressing from left to right. It has vertical bars representing regular events, that are optional events identifying the execution sequence, required events, that are required to occur if a particular execution sequence occurs, and failed events, that are to never occur after a particular execution sequence occurs. In addition to these events, the timeline also consists of black horizontal lines below the timeline bar which indicates that certain events are of no interest for particular intervals. This timeline is then converted into a test automaton for the purpose of model checking. The TimeLine editor makes it easy to specify and read requirements especially when they are highly nested, which is also true for the current thesis topic where the mission plans can get highly nested. The TimeLine editor can express a fragment of LTL and is geared towards software engineering. This thesis looks at visualizing LTL for robot mission and motion planning.
3.6 Visual Specification of Branching Time Temporal Logic

This work introduces a way to visualize the formalism of BTTL by recursively defining graphical primitives in a 3D virtual environment \cite{31}. They use it for specifying the ordering relationships of execution sequence of time varying systems. BTTL is another flavor of temporal logic where time is not Linear as in LTL but rather branches off into multiple execution sequences. It includes operators like the Universal Until($\forall[\phi_1 U \phi_2]$), where the condition $\phi_1$ has to be met for all possible execution sequences until a condition $\phi_2$ is met, and the Existential Until($\exists[\phi_1 U \phi_2]$), where the condition $\phi_1$ has to be met for at least one execution sequence until condition $\phi_2$. Here, their aim is to effectively visualize the temporal logic specifications by providing a one-one mapping between the textual representation and graphical representation, whereas this thesis concentrates on the creation of intuitive motion plans with topological information for robots using LTL, the goal being that users need not even be aware of the existence of the LTL formalism.

3.7 Graphical Interval Logic

Temporal logic can be represented graphically resembling timing diagrams similar to the ones used by hardware and software engineers and includes model theoretic semantics providing a way to reason rigorously about temporal properties of concurrent systems \cite{32}. They use a variant of temporal logic called Propositional Temporal Logic (PTL) which includes the Until operator but not the Next operator which is present in LTL. It consists of 2D graphical formulas read from top to bottom and from left to right where the horizontal dimension shows the progression of time and the vertical dimension depicts composition of formulas from subformulae. This research tries to make it easier for engineers to employ temporal formalisms.
3.8 Property Specification Patterns

Methods like model checking using LTL in system verification provides a powerful way to detect errors that are difficult to reproduce and subtle. Pattern based approaches are presented to reuse LTL property specifications to overcome some of the barriers of using formal methods due to lack of tool support, good training materials and process support for formal methods [33–35]. The challenge arises due to the fact that it is hard to write correct specifications and specifications patterns provide a solution for specifying common design requirements. This work demonstrates how property specification patterns can capture the experience base and enable the transfer of this experience across practitioners similar to the way design and coding patterns are used today. This thesis incorporates the idea behind specification patterns to simplify commonly used aspects of motion plans like sequencing.
Chapter 4

PROBLEM DESCRIPTION

4.1 Problem Overview

The objectives of this thesis are to:

1. Design a visual specification language for Human Robot Interaction (HRI)

2. Develop an easy to use User Interface for use with the visual specification language

This section describes the visual specification language and the User Interface is described in the implementation section.

The two important objectives in designing this visual specification language are:

1. To translate the visual specification into LTL, enabling the plethora of domain specific LTL control synthesis algorithms to be used for motion and mission planning.

2. To design the visual specification language such that it is intuitive, and can be used by non-experts who do not have any knowledge whatsoever of LTL, while still maintaining the full expressivity of complex LTL specifications.

The following section describes the specification language with respect to motion planning applications. There are a finite number of labels $Q = \{q_1, q_2, \ldots, q_n\}$ in the robot workspace. These labels represent waypoints (points-of-interest), obstacles that could be static or dynamic, objects to manipulate, actions to be performed etc. These labels can also represent subformulas composed of waypoints, obstacles and actions. The current framework, intended for producing motion plans, can be easily extended to support atomic propositions for other set of actions and Boolean sensors.
The following example provides a brief introduction of the main setting of the targeted applications.

Figure 4.1: The user interface, the simple road network and the mission of Example 1. The areas $q_1 - q_5$ are covered parking lots while the areas $q_6 - q_9$ are other points-of-interest (PoI).

**Example 1** Consider a search and rescue mission where an earthquake has hit an apartment complex. A rescue team dispatches an autonomous Unmanned Air Vehicle (UAV) to verify the structural integrity of garage structures. A member of the rescue team uses a tablet interface to download the map of the affected area and proceeds to provide a plan for the UAV.

The goal of the UAV is to:

1. Visit the parking lots $q_5$, $q_3$, $q_4$ and $q_1$ in that order.

2. The UAV should stay in the parking lot $q_1$ at the end of the mission.
3. $q_8$ should be avoided at all times.

Such scenarios can be easily captured using Linear Temporal Logic (LTL).

A major hurdle that has been identified by the model checking community is in writing formal specifications. The problem is made worse by the fact that the users of the robots will not have programming skills or the knowledge of mathematical formalisms.

**Problem 1** Develop a visual specification language for robotic applications that provides automatic translation to LTL so that existing control synthesis algorithms can be used. The visual language should provide templates for commonly occurring LTL specifications so that specification errors are minimized and also provide a generalized framework to create arbitrary LTL specifications for power users.

### 4.2 Solution Overview

To address the problem of creating LTL formulas, an Android app providing a touchscreen interface has been developed (see Fig. 4.1) to create motion plans that are intuitive to the user and, most importantly, making sure that the resulting specification indeed captures the desired user intention.

The user drops nodes on the screen that can be moved around. These waypoints can be connected to each other via directed edges (arrows), indicating temporal relationships, or undirected edges (lines) that indicate boolean relationships. There are, however, restrictions on how the nodes can be connected, and arbitrary directed and undirected edges are not allowed. The color of these nodes indicate desired or undesired behavior. More options for creating the motion plan can be accessed by touching the nodes for approximately 2 seconds which pops a menu as shown in Fig. 4.2.

**Example 2** The graph in Fig. 4.1 is automatically translated in the LTL specification

$$G(\neg q_8) \land F(q_5 \land F(q_3 \land F(q_4 \land FGq_1))).$$ (4.1)
More requirements can be composed conjunctively by creating specifications on different screens to reduce overloading a single scene with too many symbols.
The main concern when designing the graphical representation language is whether or not to retain the expressivity provided by the underlying formalism. This can be clearly achieved using the inductive definition of semantics of logic \cite{31}. However, this interface needs to serve non expert users and such an approach would be of little use, as noted in \cite{36}.

\LTLVIS\ is a graphical representation language used for robot motion and mission planning and is capable of capturing spatio-temporal specifications. In general, it can also be used for reactive supervisory control synthesis \cite{19}. It basically combines the work in \cite{37} and \cite{30}. It uses the graphical representation of formulas as in \cite{37} and achieves a timeline ordering of events similar to \cite{30}.

Formally, an \LTLVIS\ specification $G$ consists of a directed acyclic graph in some geometric formation, that is, $G$ is a tuple $(V, E, v_0, c, L, \Lambda, x)$ where

- $V$ is the set of nodes;
- $E \subseteq V \times V$ is the set of edges;
- $v_0 \in V$ is the start node;
- $c : V \rightarrow \{\text{green, red}\}$ is a function that colors each node either green or red, which corresponds to visiting or avoiding a node;
- $L : V \rightarrow \Phi_B(\mathcal{Q})$ labels each node with an LTL formula over the set of proposition $\Pi$;
- $\Lambda : E \rightarrow BO_1 \times BO_2 \times TO_2 \times TO_1$ is a function that labels each edge on the graph with one or more Boolean or temporal operators. In detail,
- \( BO_1 = \{ \text{AND}, \text{OR} \} \)

- \( BO_2 = BO_1 \cup \{ \varepsilon, \text{IMPLIES} \} \)

- \( TO_1 = \{ \varepsilon, \text{FUTURE}, \text{ALWAYS} \} \)

- \( TO_2 = TO_1 \cup \{ \text{NEXT}, \text{UNTIL} \} \)

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

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

The graphical representation of the graph in the current implementation is restricted to a 2D space but there is no obvious theoretical restriction for visualizing the graphs in 3D space.

![Figure 5.1: The allowed combinations of Boolean and temporal operators over an edge.](image)

Figure 5.1: The allowed combinations of Boolean and temporal operators over an edge.

The graph mentioned is generalized and necessitates the addition of further constraints in order to get meaningful specifications that the user desires. The following are some of the constraints imposed:

Let \( (b_1, b_2, t_1, t_2) \) be a quadruplet that \( \Lambda(u, v) \) maps to, then

- The combinations of Boolean and temporal operators which are allowed are presented in Fig. 5.1. The additional requirements are mentioned below:
- \( b_2 = \text{AND} \) or \( b_2 = \text{OR} \) only if \( t_1 \neq \epsilon \).
- \( t_2 = \text{ALWAYS} \) or \( t_2 = \text{FUTURE} \) only if \( t_1 \neq \epsilon \).

- Bidirectional edges or lines appear only in Strongly Connected Components (SCC) \([38]\) and the label of each edge of an SCC does not have any temporal operators, i.e., \( t_1 = \epsilon, t_2 = \epsilon \).

- If the out-degree of node \( u \) exceeds 1, then all the outgoing edges \((u, v)\) must have the same label \( b_1 \) resembling an and-or tree.

- If the label of the start node \( v_0 \) is not modified by the user, then \( L(v_0) = \text{true} \). In the current software implementation, the default behavior is to assign the ID of the node as a label.

This graphical representation is then translated to the appropriate LTL formula using the semantics of LTL\(_{\text{VIS}}\). The algorithm used for translation is presented next.

Algorithm 1 initializes the graph \( G' \) for translation to LTL. In line 2 of Alg. 1, the Strongly Connected Components (SCC) of the graph are computed. In line 3 of Alg. 1 each SCC \( C \in \mathcal{C} \) is replaced by a new node \( v_C \) where \( \mathcal{C} \) is the set of all SCCs. Since the edges in all the SCC are labeled only with Boolean connectives, the label on each new node \( v_C \) is going to be a Boolean combination of the labels of all the nodes of the SCC. Note that any constraints on the graph construction should be applied during the user input phase (see Sec. 5.1). Thus, when Alg. 1 calls algorithm Visit in line 4 then the graph \( G' \) is guaranteed to be a tree.

Remark 1 It has to be noted that it is possible to construct an automaton directly that accepts all traces satisfying the corresponding graph specification and could be useful in planning frameworks where the LTL formula is converted into an automaton \([14]\). See \([30,39]\) for more details.
Algorithm 1 Graph2LTL

Inputs: a graph $G = (V, E, v_0, c, L, \Lambda)$.
Output: an LTL formula $\phi$

1: procedure GRAPHL2TL($G$)
2: $C \leftarrow SCC(G)$
3: $G' \leftarrow CollapseSCC(G, C)$
4: $\phi \leftarrow Visit(G', v_s)$
5: end procedure

Algorithm 2 works by recursively visiting each node in the subtree, generating sub formulas in the process. The termination of the algorithm is guaranteed since it does not contain any cycles and since each node is visited only once, the running time of the algorithm is linear on the input size which is equal to the number of nodes in the tree.

Lines 2-6 check whether a negation symbol is needed in front of the label of the node that is being currently processed. Line 8 returns the label of the node if the node turns out to be a leaf node of the tree. Algorithm 2 is recursively called on each subtree in line 12. The formula is constructed depending on whether the temporal operator is UNTIL. This is because there is no boolean operator present when there is an UNTIL temporal operator. This special case of the algorithm is handled from lines 14-19 followed by the general case till line 26.

The table 5.1 describes each of the graphical elements of the graph in detail.

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

It can be seen that the red node should not hold at the current time and that the green node should hold at a later point in time which is indicated by the label FUTURE.
Algorithm 2 Visit

Inputs: a graph $G = (V, E, v_0, c, L, \Lambda)$ and a node $u \in V$.
Output: an LTL formula $\phi$

1: procedure VISIT($G, u$)
2: if $c(u) == \text{red}$ then
3: neg $\leftarrow \neg$
4: else
5: neg $\leftarrow \varepsilon$
6: end if
7: if $\text{Adj}[u] = \emptyset$ then
8: return $\phi = \text{neg}(L(u))$
9: else
10: $\phi \leftarrow \varepsilon$
11: for $v \in \text{Adj}[u]$ do
12: $\psi \leftarrow \text{Visit}(G, v)$
13: $(b_1, b_2, t_1, t_2) \leftarrow \Lambda(u, v)$
14: if $t_1 == \text{Until}$ then
15: if $\phi == \varepsilon$ then
16: $\phi \leftarrow ((\text{neg}(L(u))) t_1 t_2 (\psi))$
17: else
18: $\phi \leftarrow \phi b_1 ((\text{neg}(L(u))) t_1 t_2 (\psi))$
19: end if
20: else
21: if $\phi == \varepsilon$ then
22: $\phi \leftarrow (\text{neg}(L(u)) b_2 t_1 t_2 (\psi))$
23: else
24: $\phi \leftarrow \phi b_1 (\text{neg}(L(u)) b_2 t_1 t_2 (\psi))$
25: end if
26: end if
27: end for
28: end if
29: end procedure

where $\text{Adj}[u] = \{v \in V \mid (u, v) \in E\}$.
Table 5.1: Graphical elements of LTL-VIS.

<table>
<thead>
<tr>
<th>Green Node</th>
<th>Red Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must be visited or action performed or Boolean sensor event true.</td>
<td>Must be avoided or action stopped or Boolean sensor event false.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solid Edges</th>
<th>Dashed Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunctive requirements on nodes that the edges connect to.</td>
<td>Disjunctive requirements on nodes that the edges connect to.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AND Label</th>
<th>OR Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.</td>
<td>Disjunctive requirement between the previous node on the graph and the specification that corresponds to the subgraph where the edge points to.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMPLIES Label</th>
<th>NEXT Label</th>
<th>ALWAYS Label</th>
<th>UNTIL Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfaction of the SCC on the originating node of the edge implies satisfaction of the subgraph where the edge points to.</td>
<td>Next label indicates that the subgraph that the edge points to should be satisfied at the next step (when discrete time semantics are considered).</td>
<td>Always label indicates that the subgraph that the edge points to should always be satisfied in the future.</td>
<td>Until label 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>
</tr>
</tbody>
</table>
Table 5.2: strict ordering visual specification pattern symbol of $\text{LTL}_{VIS}$.

This indicates a special visual specification pattern that represents properties of the form $q_1 \implies X((\neg q_1) U q_2)$. It is activated by enabling the labels IMPLIES and NEXT over an edge when the visual specification pattern mode is active.

on the directed edge from the red node to the green node. The AND label combines the two requirements conjunctively to form the specification: $\neg q_1 \land Fq_2$.

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

![Diagram](image)

The right subgraph is the same as in the previous Example 3 and thus has the same LTL formula. The left subgraph on the other hand translates to $(\neg q_1) U q_3$. The red node should not hold UNTIL the green node 3 in the left subgraph holds. Since the outgoing edges from node 1 are dashed, the two subformulas are combined with a disjunction operator resulting the specification $(\neg q_1 \land Fq_2) \lor (\neg q_1) U q_3$.

**Remark 2** An $\text{LTL}_{VIS}$ 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 feature, however, is to be used with care since the current implementation does not check whether the formula is well formed, and is recommended only for expert users. This strikes a balance between the needs of expert users and the needs of non-expert users. The latter class of users can use only the graph specifications while the former can mix arbitrary graph and textual specifications.
5.2 Visual Specification Patterns

The idea of visual specification patterns stems from the use of property specification patterns in the area of formal verification and is described in [34] as “A generalized description of a commonly occurring requirement on the permissible state/event sequences in a finite state model of a system.” In robotic applications, the permissible state/event sequences depend on the LTL synthesis method used. This thesis introduces a visual specification pattern called “strict ordering” which is described as follows:

**Pattern:** Strict Ordering

**Intent:** To impose a strict sequential ordering between waypoints that are sequentially connected as a chain of waypoints.

**Example mapping:** consider an LTLVIS graph $G$ where each SCC in the sequence is collapsed and each node $v$ in the graph $G$ is labelled by $R_i$ where each $R_i$ is either a disjunction or a conjunction of atomic propositions, i.e.,

$$\forall_i R_i = \land_j q_j \text{ or } R_i = \lor_j q_j$$

The pattern for strict ordering visualized in the Figure 5.2 is given as:

$$G(R_1 \implies X((\neg \land_{j=1,j\neq 2}^m R_j) U (R_2 \land X((\neg \land_{j=1,j\neq 3}^m R_j) U (R_3 \land \ldots X((\neg \land_{j=1}^{m-1} R_j) U (R_m) \ldots)$$

Note that the LTL expression $F(R_1 \land F(R_2 \land \ldots F(R_m)))$ does not imply strict ordering. Hence the strict ordering pattern allows non-experts and experts alike to conveniently specify strict ordering in motion plans which is a very common occurrence.

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

![Figure 5.2: Strict ordering visual specification pattern](image)
The subgraph that includes nodes 1 and 2 translates to $q_1 \implies X((\neg q_1)Uq_2)$ according to the template resulting in the specification $G(q_1 \implies X((\neg q_1)Uq_2))$.

5.3 Implementation and User Interface

The software application is built on the Android platform using Java programming language and it can be downloaded from:

[https://github.com/cpslab/LTLvis](https://github.com/cpslab/LTLvis)

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. The software application overview is shown in Fig. 5.3.

The User Interface

The User interface (see Fig. 4.1) 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 proposition used to construct the LTL specification currently. The waypoint represents a location by the virtue of its label 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, is the 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 Fig. 4.2) on a long press touch event on waypoints which allows the user to select more options for creating the motion plan.

**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. 4.1). 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, with default labels 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 pre-specified. 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 [22, 28].

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:

- Each circle (green/red) is a waypoint and represents a particular location on a map regardless of where it is placed on the map.

- By default, enabling a location puts a waypoint on the map indicating that the robot should eventually visit the location.

- Enabling multiple waypoints tells the robot to go to each location with no particular order.

- The interface supports the following gestures:
  
  - 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)
Table 5.3: Explanation of icons in the pop-up menu in Fig. \[4.2\]

<table>
<thead>
<tr>
<th>SET LABEL</th>
<th>Change the label of a node on the graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISIT / AVOID</td>
<td>This option tells the robot whether to visit the location or to avoid it.</td>
</tr>
<tr>
<td>Change operator $b_1$</td>
<td>Cycle through the boolean operator $t_1$ between AND and OR.</td>
</tr>
<tr>
<td>Change operator $b_2$</td>
<td>Cycle through the boolean operator $b_2$ between AND, OR and IMPLIES.</td>
</tr>
<tr>
<td>Change operator $t_1$</td>
<td>Cycle through the temporal operator $t_1$ between ALWAYS, EVENTUALLY, NEXT and UNTIL.</td>
</tr>
<tr>
<td>Change operator $t_2$</td>
<td>Cycle through the boolean operator $t_2$ between ALWAYS and EVENTUALLY.</td>
</tr>
</tbody>
</table>

- Double tap: Double tapping two consecutive waypoints produces sequential ordering from the first point to the next.

- Long press: Long pressing a waypoint results in a menu being displayed (see Fig. \[4.2\]) which gives further options as indicated in Table \[5.3\].

- 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.
Chapter 6

EXAMPLES

The following section describes a few cases based on the work by Smith et. al. [40]. The work in [40] 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 \((q_1, q_2, q_3)\) and data upload locations \((q_4, q_5)\) on a road network. Further details on the examples can be found in [40].

**Case A:** Repeatedly visit data collect locations \((q_1, q_2, q_3)\) and repeatedly visit data upload locations \((q_4, q_5)\). The requirements can be specified using the LTL graph in Fig. 6.1. 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:

\[
GF(q_1 \lor q_2 \lor q_3) \land GF(q_4 \lor q_5) \tag{6.1}
\]

**Case B:** To avoid visiting two upload locations consecutively, a robot must visit an upload location only if it has just gathered data. In order to graphically represent this requirement with the algorithm 1, two graph specifications are required; one for

\[
\psi_1 = X(\neg(q_3 \lor q_4)U(q_1 \lor q_2 \lor q_3)) \quad (6.2)
\]

and one for

\[
G((q_3 \lor q_4) \implies \psi_1) \quad (6.3)
\]

The specification can also be entered in a single screen as shown in Fig. 6.2 to yield the LTL formula

\[
G((q_3 \lor q_4) \implies X(\neg(q_3 \lor q_4)U(q_1 \lor q_2 \lor q_3))) \quad (6.4)
\]

To enable the user enter such specifications on a single screen in a much simpler way, a visual specification pattern can be used as in Fig. 6.3. In order to use the visual
specification pattern the edge must be labeled with IMPLIES followed by NEXT and the visual specification pattern mode must be activated by checking the visual specification pattern mode checkbox in the user interface. Thus, the graph as it appears in Fig. 6.3 captures the requirement:

\[ G((q_3 \lor q_4) \implies X(\neg(q_3 \lor q_4)U(q_1 \lor q_2 \lor q_3))) \]  

(6.5)

**Case C:** In order to specify that the data gather locations \(q_1, q_2\) and \(q_3\) must be visited periodically, the specification can be visualized with the graph given in Fig. 6.4 resulting in the LTL formula:

\[ GFq_1 \land GFq_2 \land GFq_3 \land GF(q_4 \lor q_5) \]  

(6.6)

**Case D:** In order to specify that the robot must visit an upload location after gathering data, a graph representation as in Fig. 6.5 can be used. However, it can be
seen that the graph gets cluttered with too many nodes. A visual specification pattern can be used to resolve this issue as shown in Fig. 6.6.

The resulting LTL formula is:

\[
G((q_1 \lor q_2 \lor q_3) \implies X(\neg(q_1 \lor q_2 \lor q_3)U(q_4 \lor q_5)))
\]  \hspace{1cm} (6.7)

**Case E:** The next specification is that the robot should visit \(q_3, q_1, q_2\), in that order, and return to \(q_3\) with the additional requirements that (1) \(q_1\) and \(q_2\) are not visited on the way to \(q_3\), (2) \(q_1\) and \(q_3\) are not visited on the way to \(q_2\), and (3) \(q_2\) and \(q_3\) are not visited on the way to \(q_1\).

In this case, the specification is divided into 2 parts:

1. Go to \(q_3\) for the first time and
2. Repeat the sequence $q_3, q_1, q_2, q_3$ infinitely often.

The first part is presented in Fig. 6.7. If the basic algorithm discussed is used for the second part, then a series of graph specifications would need to be constructed since a single specification, even though possible would end up being unreadable. The series of specifications would then be $\psi_1 = \neg(q_1 \lor q_2)Uq_3$, $\psi_2 = \neg(q_1 \lor q_3)U(q_2 \land X\psi_1)$, $\psi_3 = \neg(q_2 \lor q_3)U(q_1 \land X\psi_2)$ and, finally, $G(q_3 \implies X\psi_3)$. However, the visual specification pattern can be used so that the whole specification can be captured in one screen as shown in Fig. 6.8.

Note that the visual specification pattern symbol appears on each transition and it means that all the other highlighted locations on the map should not be visited until the
Figure 6.4: Data gather locations $q_1$, $q_2$, $q_3$ must be visited periodically.

A node that the edge points to is visited. The resulting specification is:

$$\neg(q_1 \lor q_2)Uq_3 \land$$

$$\land G(q_3 \Rightarrow X(\neg(q_2 \lor q_3)U(q_1 \land$$

$$\land X(\neg(q_1 \lor q_3)U(q_2 \land X(\neg(q_1 \lor q_2)Uq_3)))))))) \quad (6.8)$$

**Case F:** Safety constraints for robots such as avoiding certain locations at all times can also be specified. A specification that the robot should avoid the road between intersections $q_6$ and $q_7$ is shown in Fig. 6.9 resulting in the following LTL formula:

$$G((\neg q_6) \lor X(\neg q_7)) \quad (6.9)$$

**Case G:** Adding a constraint that data from location $q_3$ must be uploaded at location $q_4$ can be visualized as in Fig. 6.10 resulting in the following LTL formula:

$$G(q_3 \Rightarrow ((\neg q_5)Uq_4)) \quad (6.10)$$
Figure 6.5: Visit an upload location after gathering data.

The final LTL specification will be a conjunction of all the desired cases discussed above.

**Remark 3** It has to be noted that taking conjunctions of so many requirements as in the above example will sometimes lead to unrealizable specifications and hence, an implementation which provides visual feedback to the user about such issues is needed and is planned for the future [41–43].
Figure 6.6: Visit an upload location after gathering data (visual specification pattern mode).
Figure 6.7: Avoid $q_1$ and $q_2$ until $q_3$ is reached.
Figure 6.8: If $q_3$ is visited, then visit $q_1$ and $q_2$ in that order while enforcing that $q_2$ and $q_3$ are not visited on the way to $q_1$ and that $q_1$ and $q_3$ are not visited again on the way to $q_2$. 
Figure 6.9: Avoid road connecting $q_6$ and $q_7$. 
Figure 6.10: Data from location $q_3$ must be uploaded only at location $q_4$. 
Chapter 7

EXPERIMENTS

A Map of the road network described in the examples was created in the lab and experiments were conducted on the Turtlebot platform which uses an iRobot Create as its base. A stargazer was used for the purpose of indoor localization. ROS packages were used to program and control the iRobot create and to retrieve sensor localization data from the stargazer module. The LTL specification sent to dropbox is given to a planning algorithm which takes as input, the LTL formula and the environment specification, generates a Buchi automaton and outputs a series of states the robot must go through to satisfy the LTL formula. These series of states are then converted to real world coordinates on the map and input to the lower planner on the robot which uses this information to traverse the map. A few pictures of the robot traversing the motion plan described in Fig 4.1 follows:

Figure 7.1: Initial location of the robot
Figure 7.2: On its way to the first location q5

Figure 7.3: Reached location q5
Figure 7.4: On its way to location 3 after making a turn at the intersection near location 1

Figure 7.5: Robot almost near location 3
Figure 7.6: Robot in location 3

Figure 7.7: Robot moving towards location 4
Figure 7.8: Robot in location 4

Figure 7.9: Robot at its final location 1
The trajectory followed by the robot during the course of execution of the motion plan was recorded and overlayed on top of the map as shown in Fig. 7.10. It is to be noted that the markers on the floor seen in the pictures were used to record the stargazer coordinates initially and the markers themselves are not used during the robot runs.
A graphical specification language $\text{LTL}_{VIS}$ has been presented and an application that demonstrates its use for robot motion and mission planning has been built on the Android tablet platform. The creation of arbitrary LTL specifications which is useful for expert users who understand the mathematical formalism thoroughly, and the use of templates, for non-expert users who do not have any knowledge of LTL formalism has been demonstrated through various examples. This tool facilitates educating students in temporal formalisms by visualizing the LTL specifications which is much easier to understand.

This application toolbox provides the basic functionality for creating motion and mission plans and will be extended with other useful functionalities in the future. Some of the extensions to this tool include:

- Provision to display unrealizable specifications to the user by highlighting certain nodes and edges in the graph.
- Extending the implementation to have a region editor similar to the one in LTLMOP where users can specify regions on the map. The users can then drag the nodes to the appropriate regions to give them appropriate labels.
- To use it on a live video feed streamed from a camera like the Microsoft Kinect instead of a map, or augmenting the map, allowing users to specify motion plans relating to picking up and dropping objects in view by selecting and highlighting parts of the camera view, which converts them into nodes and can be used similar to the examples discussed.
- $\text{LTL}_{VIS}$ can be extended with more geometric shapes in addition to the circles...
present currently to represent objects, obstacles and other useful information.

There are also plans to conduct usability studies to determine possible improvements to the user interface.
REFERENCES


