LAGR Phase II API

Planner to Perception Interface Additions

Purpose

The purpose of this document is to describe proposed additions to the LAGR API to be used to support plug and play of perception systems with planning and control systems in the Phase II "Best of LAGR" system.

Background

At the beginning of LAGR Phase I, CMU delivered to each team a robot complete with a working software installed. The software consisted of 6 parts:

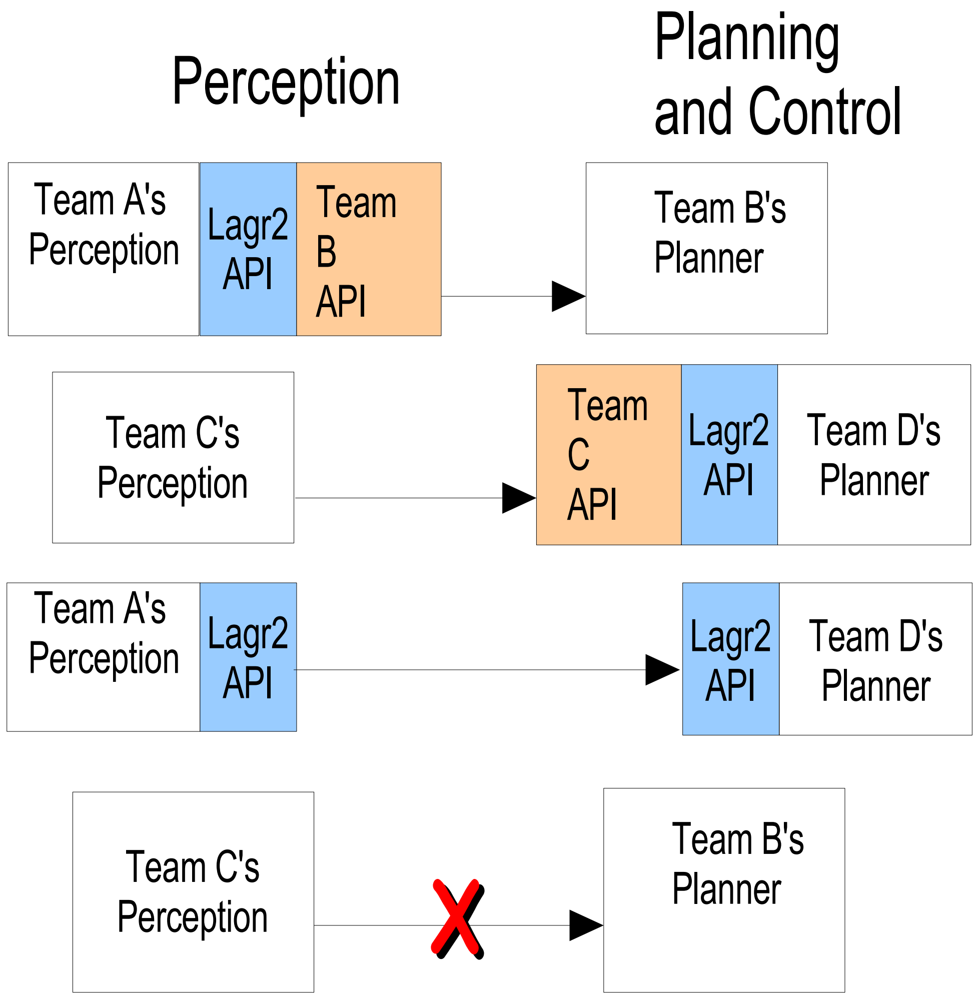
1. RedHat 9 Linux operating system. (in Phase II this was upgraded to SuSE 10.1)
2. Baseline non replaceable low-level control, data acquisition and position estimation software.
3. Replaceable Baseline planner
4. Replaceable Baseline perception software.
5. Replaceable Graphical User Interface to be run remotely across a wireless network.
6. Interface library(liblagr.a) and corresponding header files (lagrInterface.h . . .)

Because some of the baseline components were not replaceable the interfaces were one way. The teams could write software to send some information but not receive it or implement the components that receive it. Other information could be received but teams could not generate it. A consequence of this one way nature of the interfaces was that teams implementing their own planners had no interface to the perception system to work against. The only way to replace the planner was to also replace the perception system and the graphical user interface and define your own custom interfaces between them. Since each of team did this separately, each of these systems is likely incompatible.

Scope

This document is concerned only with the interface between the perception software andthe planning/control software. Although each of these two large components might be subdivided and additional interfaces defined then defined betweenthose subcomponents we are not yet ready to deal with that level of detail. The term perception is intended to include any team developed software module that generates data compatible with the interface that provides information about which geographic areas should be traversed and which should not including detecting roads, paths, terrain types, the ground plane, obstacles and cover and including any map building, fusion or filtering of this information before being sent to the planner. The planning and control side could also include any software that filters or fuses the updates afterbeing received builds maps and any thing else needed to control the robot such that it reaches the goal in the optimal way.
The standard would require only the use of a single new header (lagr2.h) by each team which wishes to produce a compatible planner or perception system. An implementation of this standard (lagr2Nist.so) will be provided but teams are free to implement their own version of the library. The library itself needs to wrap some form of communication system however the protocols that it uses and the data structures sent across them are internal and unique to each implementation of the library and therefore not defined here.
The new library is not intended to replace the original API liblagr.a or the header files that came with it

but to be used along side it/them.
The new API provides an interface both for the planner and for the perception system however interoperability might require only one to be modified to use the new API as long as a library that communicates with the older non-standard interface is available. Imagine team A has an perception system that uses the new standard APIand team B has a planner that uses a custom team-specific API but can provide a library to team A that wraps their custom API in the standard API. A's perception can be linked with the library and therefore to B's planner. Similarly team C might have an perception system with a custom API and a wrapper library for team D with standard API Planner. Teams A and D could be linked using any version of the library. Unfortunately teams C's perception and teams B's planner both with different incompatible custom interfaces are not going to work together without modifying one or the other. This means ideally both planners and perception modules would use the new API for maximum flexibility but that we will be able to interoperate with some legacy systems most importantly the original baseline planner which cannot be modified to use the standard API. Other combinations are possible including wrapping the new API in some teams custom API.



It is designed to impose as few constraints as possible on the implementations of the planner, the perception system and even on the internal communication system and still allow for interoperability. It should be possible to run multiple perception systems simultaneously connected to a single planner. It is probably not reasonable to run multiple planners at the same time since there is no mechanism described to determine how the robot should behave when the planners have conflicting outputs but the API does not necessarily exclude the possibility either.

Concepts

The lagr2Updater is a pointer to a generic communications object. It might point to a structure that contains a socket file descriptor or a handle used by any underlying communications system, or the structure could even be empty if the library implementor had no need to store anything in it. Applications are expected to call lagr2CreateUpdater() once at startup and lagr2Destroy() once at shutdown. The create function takes as an argument the name of a .ini file which could just be ignored but is expected to contain whatever configuration information is relevant to this implementation. By obtaining the data from the INI file rather than by having the application call functions to set these options the application is kept more independent from the library implementation. The lagr2Updater's primary purpose is to send and receive updates. Each update is conceptually a list of map cells with modified attributes and a vehicle state that tags the time and position at which the raw data was collected from which the attributes were determined. The state might be different from the current state due to delays at any number of places in the system. Also the planner might later correct its internal map based on the knowledge with better estimation of past positions. How these updates are internally stored and whether they are communicated all at once or with a number of messages back and forth is up to the library implementor however the functions lagr2GetUpdate() and lagr2SendUpdate() still need to be called by the user to support libraries that happen to send the data all at once and modifications to the update should not become visible to the user until after a call to lagr2GetUpdate(). Each map cell is associated with geographic position and a number of attributes, it is expected that except for cost very few planners will make use of all of the available attributes and further that very few perception systems would generate all of the attributes, and even for those that do some attributes might be known or relevant only for certain positions and not the entire map. The planner should perform with at least minimally acceptable behavior even if the only attribute set is cost. Developers of perception systems should be aware that when used within the "best of LAGR" system the only attribute essentially certain not to be ignored when they are connected to a compatible planner from another team is cost. This is not to say that is useless to set any other attribute since as the list of attributes is publicized and perception modules available to produce them and assuming the associated information really is valuable in achieving better behavior the likelihood of more planners taking advantage of the attribute increases. The communications library may not necessarily send all of the attributes either based on an assumption perhaps confirmed in the ini file that the planner will not some attributes. The API uses double values for all attributes since this provides the maximum precision and range of values, but the library might very well convert this to a smaller type (char, short , float etc.) and use information in the ini file to confirm that with the given planner and/or perception system this is a safe thing to do that will conserve memory or communications bandwidth.

Some examples of what implementors of perception systems, planners or interface libraries might do was provided to avoid having components be less reusable by making unnecessary assumptions about how the rest of the system will be implemented but developers need to be encouraged to focus on their component and assume the widest possible range of other components. Rather than perception systems becoming highly tuned to the quirks of the planner or comm system they are using or vice-versa.

To be done

An additional document will provide a description of every function within the API. Some examples will be built, both real functioning examples using the NIST, Baseline and SRI planners cross connected to either the NIST or SRI perception systems as well as a more trivial but easier to understand non working example.