Autonomous Navigation in Unknown Environments via Language Grounding

Koushik (kbhavani)

Aditya (avmandal)

Sanjay (svnaraya)

Mentor
Jean Oh

Introduction

As robots become an integral part of various domains ranging from households to military, they are expected to be intelligent systems that can support the human counterparts by taking advantage of new modalities for control, sensing and reasoning capabilities. This will require robots to possess cognitive abilities; understand natural language, perceive semantics of the environment around them, and perform high-level reasoning once given a task. This level of understanding would enable humans to easily cooperate with complex robots without requiring specialized interfaces, protocols or training.

This project aims to develop a robot that can understand natural language commands and plan to navigate through known or unknown environments to achieve the specified goal. With this high level project goal in place, the team aims to focus specifically on the language grounding and planning aspects under the assumption that perception through vision is a solved problem.

The project platform is a part of Robotics Collaborative Technologies Alliance (RCTA).

Problem Specification

Given

1. User Input: Commands as voice or text in structured (natural) language
2. Perception: A predetermined map of obstacles

Therefore it has been assumed that the robot is capable of perceiving its environments and localizing the obstacles or goal locations in its environment. In such scenario, given the user command in a high level natural language, the following goals have been set forth.

Goals

1. NLP: Ground commands from structured language to action space
2. Motion Planning: Plan for the grounded commands

The Robot

For the purpose of this project, the robot used to implement the ideas was the Clearpath Husky, a ground vehicle with the following specifications:

1. 4-wheeled, 4-motor skid-steered robot
2. 4 Mac Minis for processing:

(a) am1: Local planner (with roscore)
(b) am2: Perception
(c) am3: ROS Master
(d) am4: Global Planner

3. Combination of NML and ROS communication pipelines

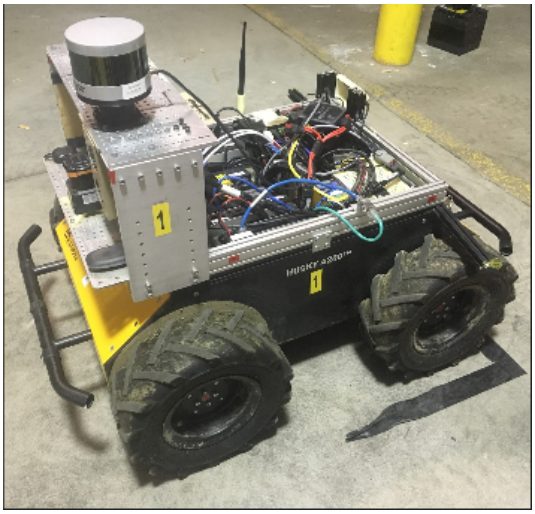

Figure 1: Clearpath Husky

Implementation

This section can be divided into 3 major sections, each one talking about one of the following; i) the language grounding module, ii) the communication module, and iii) the motion planning module.

Language Grounding

The language parser implemented in this project is quite simplistic. Importance was given to the recognition of objects and constraints, and grammar and semantic understanding took a backseat. Separate dictionaries were defined to hold the world coordinates of various objects (table, chair, desk, etc.) and the definitions of constraints (beside, behind, left of, etc.). A search for these keywords in the user's command, followed by correlating the objects found with the constraints (either specified by the user or assumed by default), results in grounded commands in the action space of the robot (in terms of world coordinates). The planning module then takes these 'checkpoints' as input, and generates an optimal path of waypoints (and replans as it goes, in the case of RTAA*) for the robot to follow.

Some of the capabilities of the simple language parser are demonstrated below. These examples assume that the left-right constraints are along the world x-axis, the front-behind constraints are along the world y-axis, and the radius of an object is denoted by r_{object}.

Example 1

Command: Go to the left of the table
Inference:
Goal = $(x_{table}, y_{table}) - (r_{table}, 0)$

Example 2

Command: Go to the chair, and then to the left of the desk
Inference:

Goal 1 = $(x_{chair}, y_{chair}) - (0, r_{chair})$
Goal 2 = $(x_{desk}, y_{desk}) - (r_{desk}, 0)$

Example 3

Command: Go to the left of the desk via the chair
Inference:
Goal 1 = $(x_{chair}, y_{chair}) - (0, r_{chair})$
Goal 2 = $(x_{desk}, y_{desk}) - (r_{desk}, 0)$

Example 1 exhibits a basic command being grounded to the (x, y) position of the target with an appropriate offset so that the specified constraint is satisfied. Examples 2 and 3 show more complex commands that involve multiple targets. In both examples, a pair of checkpoints are recognized and generated. However, the order of the checkpoints remain the same in both cases, although the intention of the user's command in Example 3 is to go in the opposite order. These examples demonstrate the capability of the parser to recognize goals and constraints, and the shortcoming of not realizing the semantics of the command.

Communication

The Husky used in the project came with an internal communication pipeline between its various machines. The part that is relevant to the navigation aspects of the robot is visualized below in Fig(2).

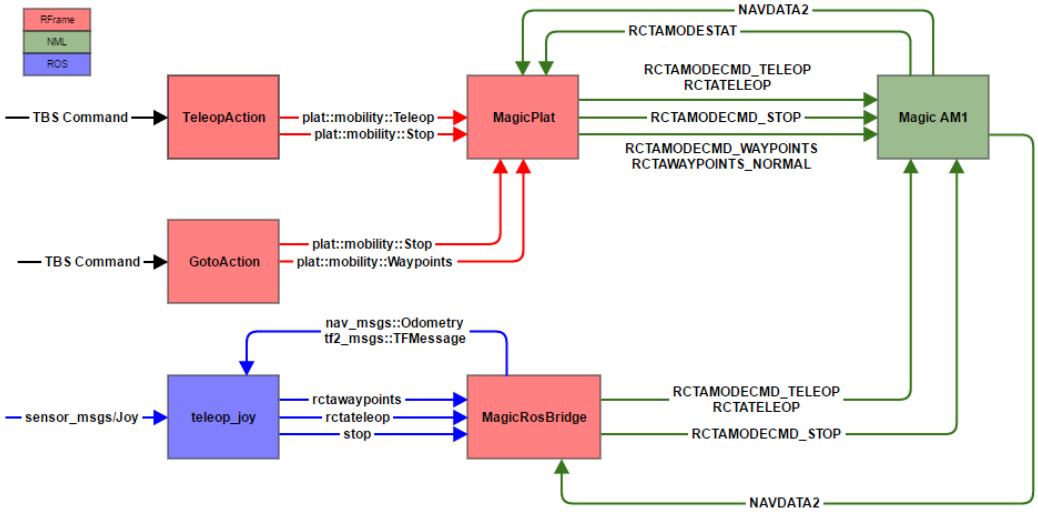

Figure 2: Communication Network

As mentioned earlier in the section, there are two ROS cores that power the system. One of the ROS masters is located in *am3*, and is the hub for the language parsing, perception, and global planning modules, which are divided between three computers - *am2, am3*, and *am4*. The other ROS core runs on *am1*, which houses the local planner for the Husky. The *MagicRosBridge* node (visualized in pink at the bottom of the figure) and the *Magic AM1* node (shown in green on the top right of the figure) are responsible for facilitating proper communication between the two ROS masters running on the robot. The *MagicROSBridge* node expects waypoints in a specified structural format to be published to a ROS topic called *rctawaypoints*, converts these waypoints to a format that the local planner can process, and communicates them to *Magic AM1*. The global planner nodes implemented in this project publish waypoints in the expected format to the *rctawaypoints* topic. This sums up a brief description of the communication pipeline associated with the motion planning and navigation of the robot.

Motion Planning

The robot Husky is equipped with low level controllers which ensure trajectory following if a global planner provides it with a set of waypoints along a trajectory.

The environment of the robot has been modelled as a grid-world which allow for implementation of graph-search algorithms to search for optimal paths to the goal. Multiple state-of-the-art algorithms (some of which find applications in real-time motion planning) were reviewed, such as A*[1], RTAA*[2], LRTA*[3] and LSS-LRTA*[4].

A* Algorithm

A* is an algorithm for finding cost-minimal paths in state space (graphs). For every state s, encountered during the search, A* computes the minimal cost-to-come $g(s)$ and maintains a heuristic for the state $h(s)$ to subsequently compute $f(s) = g(s) + h(s)$. Note that the heuristic needs to be consistent for A* to be optimal. The algorithm maintains a priority queue called OPEN list which initially consists just of the start state (node). The priority function for the queue is the f-value of a given state. The state s with the smallest f-value is popped from the OPEN list. If it is the goal state, the algorithm terminates. Otherwise, the cost-to-come of the state is updated if necessary and the successors of s are all inserted into the OPEN list. This process repeats until the goal state is popped from the OPEN list terminating with a success or the OPEN list is empty in which case the algorithm terminates with failure.

Real-Time Algorithms

Although the shortest path is obtained, offline search algorithms like A* are not ideal for dynamic environments. It does not account for moving obstacles and in a large graph search problem planning time might be too high before the robot starts to execute the plan. This necessitates the development of real-time planning algorithms where search happens in real-time.

Real-time heuristic search methods interleave planning and execution. They find only the beginning of a trajectory from the agent's current position to the final goal position. Their search is hence limited to the local region in the state space around the agent which can be reached within a small number of actions (or edges in the graph). Once a local trajectory has been determined, the agent executes appropriate actions to move along the planned trajectory. This process is repeated until the agent reaches its goal position.

Since the strategy involves only local searches the total planning time is generally lower but at the cost of optimality. However the major advantage with real-time algorithms is that they can satisfy dynamic constraints like dynamic obstacles in the environment. In this project two versions of A* algorithm that are competitive to the much conventional D* algorithm which has already been implemented on the robot.

1. RTAA* Algorithm

Real-Time Adaptive A* algorithm is a real-time search algorithm that chooses its local search spaces smartly after partial execution of local trajectories. The *adaptive* nature of the algorithm helps in reducing the planning time of subsequent A* searches when performing multiple A* searches from the current state of the agent by appropriately updating the heuristic costs of the states based on history of traversal. Hence each of the A* search is more informed than the previous A* algorithm.

Assume that s is a state that was expanded during an A* search. We can obtain an (updated) admissible estimate of its goal distance $h(s)$:

The distance from the current start state s_{curr} to any goal state via state s is equal to sum of the distance from the start state s_{curr} to state s and the goal distance $h(s)$. It clearly cannot be smaller than the goal distance $h(s_{\mathrm{curr}})$. Therefore we have:

Algorithm 1 `RTAA* Algorithm`

```
1: while s_curr ≠ GOAL do
2:   AStar(lookahead)                                   ▷ Local Planning
3:   if s̄ = FAILURE then
4:     return FAILURE
5:   for all s ∈ CLOSED do
6:     h(s) = g(s̄) + g(s̄) − g(s)
7:   movements := any desired integer greater than zero
8:   while s_curr ≠ AND movements > 0 do
9:     s_curr = succ(s_curr) along obtained local trajectory   ▷ Plan Execution: 1 Step
10:    movements = movements - 1
11:    increase cost of edges in graph (dynamic obstacles)
12:    if cost of current cost-minimal path has increased then
13:      break
```

$$
\begin{aligned}
g(s) + h(s) &\geq h(s_{\text{curr}}) \\
h(s) &\geq h(s_{\text{curr}}) - g(s) \\
h(s) &\geq f(\bar{s}) - g(s)
\end{aligned}
$$

The algorithm also has a concept of *lookahead* and *movement* which define the current start state s_{curr} and $\bar{s}$ more precisely. Lookahead defines how far or how local a particular A* search is and movement defines how much along the obtained local path the agent moves. Generally, higher the lookahead, less sub-optimal the obtained path. When lookahead is infinte, the algorithm degenerates to a normal A* algorithm. Movement allows for replanning between A* searches. In a highly dynamic environment, lower movement is preferable. The algorithm has been described in Alg.(1). The notation followed in the algorithm is as follows:

The *lookahead* is the number of states, at the most, to be expanded by A*.

The parameter *movement* is the number of steps to move along the plan given by A* algorithm.

The current state of the agent is s_{curr}.

The state that A* was just about to expand when it terminated is represented by $\bar{s}$.

2. LRTA* Algorithm

LRTA* and RTAA* are similar algorithms with the strategy for update of heuristics different.

LRTA* replaces the heuristic of each expanded state with the sum of the distance from the state to a generated but unexpanded state s and the heuristic of state s, minimized over all generated but unexpanded states s. The heuristics of the other states are not changed as their cost-to-come need not be optimal when A* terminated. If h' denotes the heuristic after all the updates, the heuristics of LRTA* satisfy the following for all expanded states s:

$$
h'(s) = \min_{a \in A(s)} (c(s,a)) + h'(succ(s,a)) \tag{1}
$$

where a is the action considered from the action space A.

LRTA* and RTAA* perform similarly for smaller lookaheads but for larger lookaheads LRTA* updates heuristics to be more informed than RTAA*. However since the update step of LRTA* is more computational(minimal over all possibilities), it takes longer planning time for LRTA than RTAA*. This can be attributed to the fact that LRTA* needs to perform two searches, one to search in the local state space and another search to appropriately update the heuristics.

Since the experiments involved a small state space (owing to the environment constraints and resolution of graph) RTAA* was preferred over LRTA*.

Implementation of Algorithms

Two of the above motion planning algorithms were implemented and tested on the robot. The first one was the classic A*, which works very well (for its notorious simplicity) in static environments. The second one was the Real-Time Adaptive A* algorithm, which can outperform the A*, and also be adapted to dynamic (constantly changing) environments.

The A* algorithm was run on the robot in a number of predefined worlds. The objects (obstacles and targets) were placed at known world coordinates, and a discretized cost map of the world was generated and passed to the algorithm. Then, various commands, like the ones discussed in section *Language Grounding*, were given to the robot were successfully executed. Video links demonstrating the same can be found in the *Results and Media* section.

The RTAA* algorithm was tested both in simulation as well as on the robot. Some of the results from simulation:

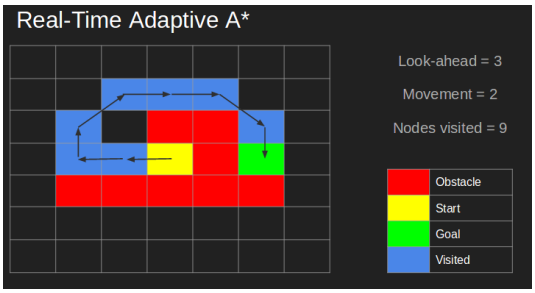

Figure 3: Small Lookahead. Low optimality and low planning time.

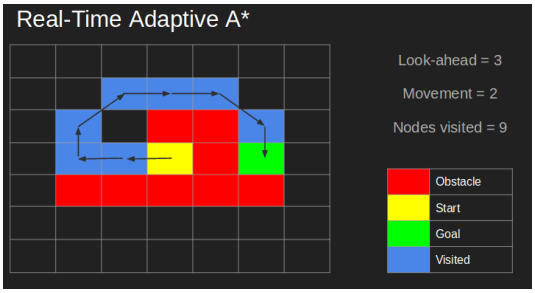

Figure 4: Balances optimality and planning time

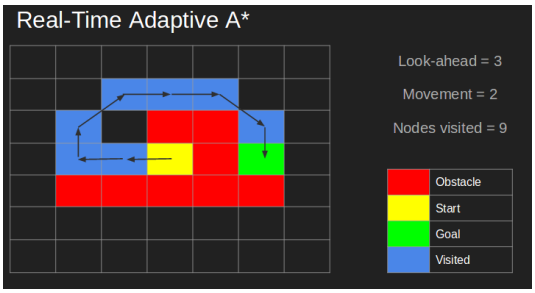

Figure 5: Infinite Lookahead: Equivalent to A*. Optimal but higher planning time.

The parameter lookahead determines the weighing of planning time and optimality of planning. Fig.(3 - 5) show the dependence of optimality on lookahead. The importance of movement can be witnessed in the videos attached along with the report.

The algorithm was implemented on the robot. Similar to the implementation of the A*, objects were placed at known world locations, and a discretized cost map of the world was generated and passed

to the algorithm. Additionally, predefined changes to the world environment were specified at certain timestamps. For completeness, these changes were emulated in the real world. A video demonstrating the performance of this algorithm in dynamic environments is included in the *Results and Media* section.

Conclusion

The project was largely successful, and most of the set targets were achieved. The implementation of different algorithms helped demonstrate the real-world applications of these methods in the context of this project. Further work in integrating an active perception system using the robot's onboard sensors and expanding the functionality of the current language processing system will enable more robust experiments to be performed on the system.

Work Division

The team believed that division of work might reduce the speed of the project progress as the components of the project pipeline we were focussing on were closely connected. Hence we decided to work together on all aspects of the project. Each one of us believes we have contributed equally.

Results and Media

A* planner, Demo 1

A* planner, Demo 2

Real-Time Adaptive A* planner with *lookahead* of 4 and *movement* 2

References

1. Peter E. Hart, Nils J. Nilsson, Bertram Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions of Systems Science and Cybernetics*, July 1968
2. Sven Koenig, Maxim Likhachev, Real-Time Adaptive A*, *In Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, May 2006
3. Sven Koenig, A Comparison of Fast Search Methods for Real-Time Situated Agents, *In Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, 2009
4. Sven Koenig, Xiaoxun Sun, Comparing Real-Time and Incremental Heuristic Search for Real-Time Situated Agents, *In Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, 2009