Runtime-Switchable Heuristics in A* for Autonomous Robots

Ignacio Pastore Benaim, David Padilla Orenga

Abstract— **This document presents a practical exploration of integrating a global planner based on the A* algorithm with runtime-switchable heuristics for ROS-based autonomous navigation. Traditional A* implementations utilize a fixed heuristic, which can limit adaptability across diverse environments. Our approach introduces a flexible framework that allows dynamic heuristic selection (e.g., Manhattan, Euclidean, Chebyshev) during execution without requiring navigation stack restarts.**

We demonstrate the methodology, implementation, and experimental results obtained across various map scenarios. Experiments highlight how adaptive heuristic switching can improve planning times and path quality by tailoring the heuristic to specific map topologies. This innovation ensures robust performance in heterogeneous environments, making it a valuable tool for autonomous robotic navigation.

I. Introduction

In autonomous robotics, global path planning is critical for navigating complex environments. Algorithms such as Dijkstra, A*, and RRT* have become foundational tools, with A* standing out due to its balance between optimality and computational efficiency. Introduced by Hart, Nilsson, and Raphael in their seminal work [?], the A* algorithm combines the advantages of Dijkstra's algorithm and greedy best-first search by using a heuristic function to estimate the cost to the goal, ensuring both optimality and completeness when the heuristic is admissible.

Despite its effectiveness, traditional A* implementations rely on a fixed heuristic function, such as Manhattan (L1 norm) or Euclidean (L2 norm), which can constrain performance in dynamic or heterogeneous environments. Different heuristics have varying strengths; for example, Manhattan is well-suited for grid-based maps with axis-aligned obstacles, while Euclidean provides better results in open spaces. However, a static heuristic limits the algorithm's ability to adapt to changing conditions.

This project addresses these limitations by introducing a runtime-switchable A* planner that enables dynamic heuristic selection during navigation. This feature enhances adaptability and efficiency without requiring the navigation stack to restart, making it particularly suitable for ROS-based systems.

A. Project Motivation and Scope

The primary motivation is to demonstrate the impact of heuristic selection on planning speed and path quality in grid-based navigation. The scope of this work includes developing a ROS plugin that integrates with the `move_base` framework and supports dynamic heuristic switching. Key components include:

- The ROS Navigation Stack for costmap generation and path execution,
- A custom global planner plugin implementing A*,
- Dynamic Reconfigure for runtime heuristic adjustments.

B. Literature Review

Recent studies underscore the importance of heuristic selection in grid-based path planning. For instance, Manhattan (L1 norm) excels in structured, corridor-like environments, while Euclidean (L2 norm) is more effective in open spaces [?], [?]. Advanced methods such as RRT* [?] provide smooth paths but at a higher computational cost. This work builds on the foundational insights of Hart et al. [?], emphasizing the flexibility of A* through heuristic switching to address these challenges.

C. Outline of Approach

The remainder of this paper is organized as follows:

- Section **??** details the classical A* algorithm and our modifications for heuristic switching.
- Section **??** describes the ROS plugin architecture and implementation.
- Section **??** presents experimental results on various map types.
- Section **??** concludes with key findings and future research directions.

II. Method Description

A* [?] is a graph-based path planning algorithm that expands nodes from a start position until reaching the goal. The priority of exploration is determined by:

$$f(n) = g(n) + h(n), \tag{1}$$

where $g(n)$ is the cost from the start to node n and $h(n)$ is a heuristic function estimating the cost from n to the goal.

Common heuristic functions in 2D grids:

- **Manhattan:** $h(n) = |x_{goal} - x_n| + |y_{goal} - y_n|$,
- **Euclidean:** $h(n) = \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2}$,
- **Chebyshev:** $h(n) = \max(|x_{goal} - x_n|,\ |y_{goal} - y_n|)$.

A. Proposed Improvement: Runtime-Switchable Heuristic

Rather than fix one heuristic, we introduce an interface that can *toggle* the heuristic type (Manhattan, Euclidean, Chebyshev) at runtime. This is accomplished through a Dynamic Reconfigure server in ROS, enabling quick adaptation to different map characteristics.

III. Implementation

A. System Overview

Our system consists of:

- **Global Planner Node (Custom):** Implements A* with a dynamic parameter for the heuristic.
- **Costmap2DROS:** Provides obstacle/costmap data from sensor inputs.
- **Dynamic Reconfigure Server:** Exposes the parameter `/GlobalPlanner/heuristic_type` for at-runtime switching.

IV. Implementation

A. System Overview

Our approach provides a plugin that implements the `nav_core::BaseGlobalPlanner` interface, making it compatible with the standard ROS Navigation Stack (`move_base`). Specifically, this plugin is declared as:

Hence, from the perspective of the navigation stack, our planner is loaded dynamically at runtime as `heuristics4astar/GlobalPlanner`.

- **Heuristics4AStar (Plugin):** A library that extends `nav_core::BaseGlobalPlanner` and implements A* with runtime-switchable heuristics, along with a Dijkstra fallback.
- **Costmap2DROS:** Provides obstacle/costmap data from sensor inputs; used by the planner to determine feasible paths.
- **Dynamic Reconfigure Server:** Exposes the parameter `/GlobalPlanner/heuristic_type` for at-runtime switching among Manhattan, Euclidean, or Chebyshev heuristics.

B. Plugin Architecture

Because the plugin inherits from `nav_core::BaseGlobalPlanner`, it integrates seamlessly with `move_base`. Internally, we adapt functionality from the standard `global_planner` package (e.g., expansions, cost computations) but add our own code to handle heuristic switching via Dynamic Reconfigure. This design allows the `move_base` node to recognize our planner as a valid global planner plugin and load it within the navigation configuration showed in Figure **??**.

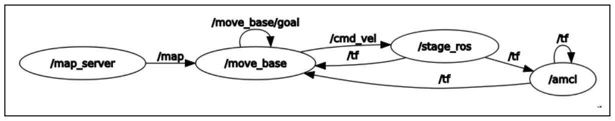

Fig. 1. Conceptual Node Diagram: the custom global planner node with connections to costmap and move_base.

C. Key Modifications

Motivation: Different map structures benefit from different heuristics. **Deviation from Standard A***: We add a callback in the A* plugin that updates the "heuristic_type" string (or integer) at runtime. **Performance Impact:** If a user chooses a suboptimal heuristic for a large open map, planning might be slower. But if they switch to a more direct Euclidean measure, path length and timing might improve.

Algorithm 1 Pseudo-code for Switchable A* Heuristic

```
Require: heuristic_type in {Manhattan, Euclidean, Chebyshev}
1: if heuristic_type == Manhattan then
2:     $h(n) \leftarrow |x_{goal} - x_n| + |y_{goal} - y_n|$
3: else if heuristic_type == Euclidean then
4:     $h(n) \leftarrow \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2}$
5: else if heuristic_type == Chebyshev then
6:     $h(n) \leftarrow \max(|x_{goal} - x_n|, |y_{goal} - y_n|)$
7: end if
8: return $f(n) = g(n) + h(n)$
```

V. Experimental Results

A. Setup

We evaluated the planner in ROS Noetic, using:

- **Simulator:** Gazebo/Stage
- **Maps:**
  - **maze_corridor:** A corridor-like environment with turns
  - **blank:** Very open space, minimal obstacles
  - **corridor:** Another partially blocked map with line-shaped obstacles
- **Metrics:** Planning time, path length, sum of turns
- **Heuristics Tested:** Manhattan, Euclidean, Chebyshev

B. Data and Observations

We collected multiple runs for each map and heuristic, then aggregated the results to produce average (mean) values of time, path length, and turning cost. Table **??** shows a summary of the computed averages for each (Map, Heuristic) combination:

1) Qualitative Analysis per Map:

- **Maze-like map (maze_corridor)**:
  - Paths bend around walls. Manhattan often takes a more "grid-aligned" route, while Euclidean and

TABLE I
AGGREGATED AVERAGES OVER RUNS (TIME IN SECONDS, LENGTH IN METERS, TURNS IN SUM OF TURN ANGLES).

Map	Heuristic	AvgTime	AvgLength	AvgSumTurns
maze_corridor	manhattan	0.0571	37.5759	324.41
maze_corridor	euclidean	0.0504	35.0652	57.1588
maze_corridor	chebyshev	0.0520	35.5563	56.8049
blank	manhattan	0.00454	17.2552	3.52737
blank	euclidean	0.05271	17.1446	189.8077
blank	chebyshev	0.07646	17.3730	195.2520
corridor	manhattan	0.00578	21.5848	24.8970
corridor	euclidean	0.03303	20.7376	31.9885
corridor	chebyshev	0.04731	21.2848	64.44

Chebyshev produce slightly smoother curves yet similar planning times.
  - Comparing results: Manhattan is *slower in planning time* and yields a *much higher turn sum* than Euclidean or Chebyshev. Euclidean and Chebyshev are quite close in planning time, but Chebyshev's path is slightly longer or differs in turn cost.
- **Blank map**:
  - The path is basically a straight line from start to goal under any heuristic, but *Manhattan expansions are extremely fast* (about 0.0045 s). Path length is around 17.25, with only 3.53 turns.
  - Euclidean and Chebyshev times are higher, and they produce significantly bigger turn sums—likely because the expansions in open space allow diagonal expansions (or full expansions) that lead to more angle changes or expansions in the search.
- **Corridor map (partially blocked, lined obstacles)**:
  - Manhattan often yields a step-like or more direct route, while Euclidean/Chebyshev can find diagonal segments. Obstacles force detours, causing the path to weave among blocks.
  - Manhattan is very quick again, with moderate turn sum (24.90). Euclidean is a bit slower (0.0330 s) with 31.99 turns. Chebyshev is intermediate in time but has a large turn sum (64.44), suggesting more zig-zag expansions.

2) Visual Examples: Figure **??** showcases the path in a maze-type environment, while Figure **??** shows the nearly straight diagonal route in an open map, and Figure **??** illustrates the corridor or lined-obstacle scenario. The green lines represent the global path from the start (bottom-left) to the goal (top-right).

3) Overall Conclusions from Data:

- No single heuristic dominates across all maps.
- *Manhattan* tends to be extremely fast in open spaces (`blank` map), but can produce more corners (hence a higher turn sum) in corridor-like scenarios.
- *Euclidean* and *Chebyshev* often generate more diagonal or smoother expansions, especially in partially open areas, but can show slightly longer times in simple open spaces if the expansions are large.

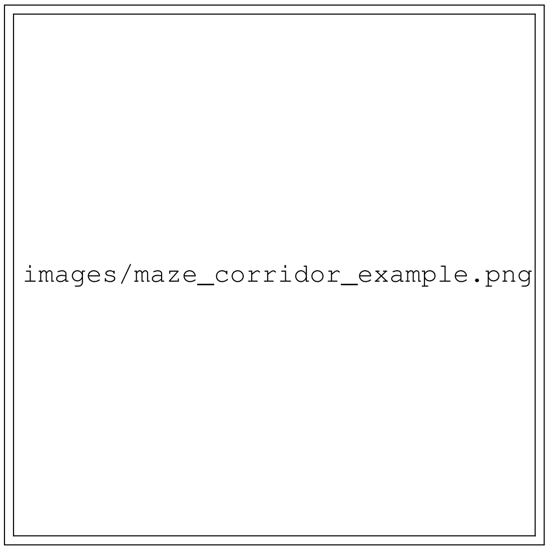

Fig. 2. Maze corridor example. Path must navigate multiple turns.

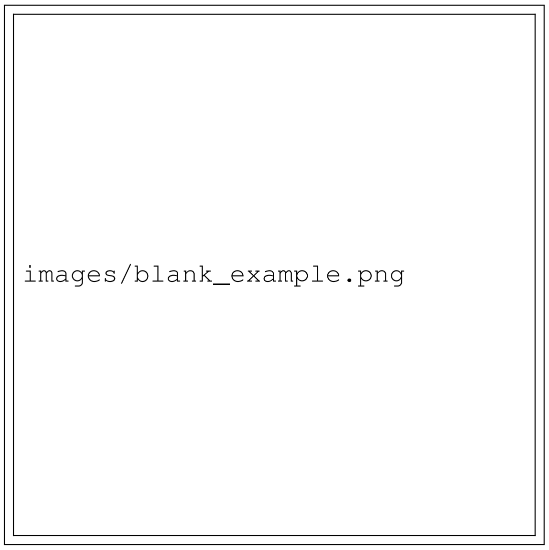

Fig. 3. Blank map example. Almost a direct diagonal path.

- Computing standard deviations or min/max could further reveal run-to-run variability.

VI. CONCLUSION AND FUTURE WORK

We introduced a runtime-switchable heuristic A* planner within the ROS Navigation Stack, enabling dynamic changes among Manhattan, Euclidean, and Chebyshev heuristics at runtime.

Key Observations:

- **Maze-like environments**: Manhattan gave higher turn counts and was slower in some runs, while Euclidean/Chebyshev remained relatively efficient with fewer sharp angles.
- **Open spaces**: Manhattan expansions finished extremely quickly, though path length and sum of turns were very

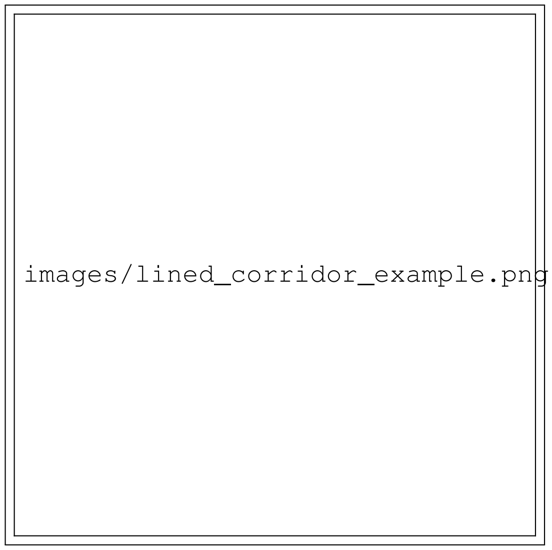

Fig. 4. Lined obstacles corridor example. The path weaves around blocks.

low. Euclidean/Chebyshev had more expansions and angle changes, raising turn metrics.

- **Corridor or lined obstacles**: Manhattan found step-like paths with moderate turn sums, while Euclidean or Chebyshev sometimes introduced diagonal arcs but also occasionally higher expansions and turn sums.

Limitations: We tested in simulation with static obstacles; real-world trials may differ due to sensor noise and dynamic obstacles. We also did not integrate advanced local planners or post-processing for smoothing.

Future Work:

- Evaluate the plugin on a physical robot to assess real-time performance and sensor impacts.
- Investigate automatic heuristic selection based on live map analysis (e.g., detecting corridor vs. open areas).
- Integrate advanced local planners (like TEB) or smoothing to reduce sharp path corners.

References

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, ”A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, MIT Press, 2005.

[3] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[4] S. Karaman and E. Frazzoli, ”Sampling-based Algorithms for Optimal Motion Planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.