Energy-Aware Path Planning for Autonomous Mobile Robot Navigation

Renan G. Maidana
School of Technology
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, Brazil

Abstract

Battery life is yet one of the main limiting factors to a robot's total mission time, and proper management of energy resources is paramount in a robotic application. In this paper, we propose a solution to integrate energy awareness in the path planning of a mobile robot performing autonomous navigation. Our contributions are: 1) The definition of a planning domain for mobile robot path planning which considers energy consumption and integrates energy actions in the generated plans; 2) The implementation of an automatic path planning solution which avoids high energy areas in a known environment. We test our solution in a simulated embedded mobile robot, extending the robot computer's battery life by approximately 42.8 %, compared to conventional path planning.

1 Introduction

Embedded processors and Systems-on-Chip (SoCs) have gained attention in mobile robotics, as they are attractive substitutes for conventional computers due to their reduced size and weight, and high performance-per-watt[1]. However, many state-of-the-art solutions in mobile robotics require significant computing power. As SoCs become increasingly powerful to meet this requirement, so too increases their energy consumption, and the unconstrained use of these state-of-the-art solutions reduces a battery-powered robot's potential operating time. For example, running a robot's embedded processor always at full capacity is wasteful, as there are situations and environments where the robot does not need all of its computing resources.

In this paper, we propose a path planning solution for mobile robots which produces energy-aware plans. Specifically, we define a STRIPS planning domain with *high energy zones*, where the robot's embedded computer must run at full capacity. The robot's state includes an "energy" variable, the metric to be minimized at each plan step. The domain includes movement and energy actions, the latter which actuate upon the embedded computer's mode of operation (e.g., maximum efficiency or maximum performance). Because of the minimization metric, the planner will give preference to plans with the least amount of energy actions, effectively avoiding the high energy zones.

[1]https://www.insight.tech/health/88-more-performance-per-watt-for-embedded-and-iot

To test this solution, we implement it in a path planning package for mobile robots, integrated to the Robot Operating System (ROS) (Quigley et al. 2009). This package consists on three ROS nodes and one ROS service, tasked with generating STRIPS problems, running a numerical planner, as well as parsing and executing the plan.

To evaluate the package, we experiment by running both it and a conventional path planning solution in a simulated robot, on a battery-powered embedded computer, performing autonomous navigation. We measure and compare the battery voltage over time for both cases, in which our path planning package increased battery life by 42.8 %.

The rest of this paper is organized as follows. The description of the energy-aware STRIPS domain is presented (Section 2), followed by details of the planning package's technical implementation in ROS (Section 3). Afterwards, we detail the experimental setup and the results obtained (Section 4), finishing with the related works (Section 5) and conclusions on our results (Section 6).

2 Energy-Aware Planning Domain

In mobile robotics, the problem of path planning for autonomous navigation is treated directly as a graph search problem in a *Configuration Space* (C-Space). In summary, given the C-Space representation of a known environment, find the shortest possible route between the robot position and its goal, while avoiding obstacles (Siegwart, Nourbakhsh, and Scaramuzza 2011).

This approach fits autonomous navigation, as only movement actions are considered. However, being able to define multiple types of actions is an advantage here, since both movement and energy actions must be considered when planning. Thus, we model a domain sufficiently complex to allow planning in an acceptable time (e.g., less than 30 seconds). The full domain, implemented in PDDL, can be found in the ROS package's GitHub page[2].

We start by modelling the operating environment itself. It consists of an occupancy grid representation (Moravec and Elfes 1985), a 2D grid where each space can be either free or occupied by an obstacle (or a part of one). The

[2]https://www.github.com/rgmaidana/ros_enhsp

world's dimensions are represented by four boundary variables, $(min_x,\ min_y,\ max_x,\ max_y)$. A fifth variable, called *resolution*, represents how many meters there are in any position.

The obstacles are modelled as grid positions (x, y), where the coordinates indicate the obstacle's center in the world. To define zones for obstacle avoidance and high energy, two sets of radii are defined:

$$Obstacle = \begin{cases} \text{Center}:\ C_x, C_y\ \in \mathbb{R}^2 \\ \text{Clearance Radii}:\ Rc_x,\ Rc_y \\ \text{Energy Radii}:\ Re_x, Re_y \end{cases} \tag{1}$$

The clearance radii specify dimensions in x and y to which the robot must keep away, to avoid colliding. They are calculated as the distance from the obstacle's center to its outer border plus some safety margin. The energy radii define the high-energy zones, where the robot is allowed to enter but must operate at full capacity. They are calculated the same way as the clearance radii, but with a larger margin. As such, obstacles in our domain are seen as geometric spaces which must be avoided. If we consider every occupied grid position as an obstacle, the number of objects modelled in our domain blows up for large environments, severely impacting planning time and thus justifying the representation of obstacles as geometric spaces. An example of an obstacle, its zones and sets of radii can be seen in Figure 1.

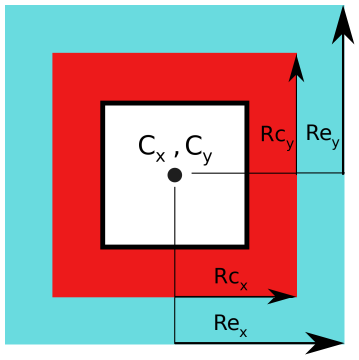

Figure 1: An obstacle, and its clearance and energy zones. The distances from the obstacle center to the boundaries of the red zone, in each dimension, are the clearance radii. The distances to the boundaries of the blue zone are the energy radii.

The robot state is modelled as a grid position and a variable *e*, indicating the robot's energy requirements at each position. If the robot is in a high energy zone, *e* is increased. Formally:

$$Robot = \begin{cases} \text{Position}:\ x, y\ \in \mathbb{R}^2 \\ \text{Energy}:\ e \end{cases} \tag{2}$$

Finally, we obtain the complete planning model by defining the set of possible actions. Here we have 16 movements and 2 energy-related actions, totalling 18 possible actions. Firstly for energy, the two actions simply increase and decrease the robot's *e* variable, without preconditions.

As for movement, we consider eight possible directions: up, down, left, right, and the diagonals in between. The effect of each movement action increases or decreases the robot's coordinates. For example, the "*move up*" action increases the robot's y coordinate, while "*move right*" increases the robot's x coordinate. The preconditions for movement are:

1. For all obstacles, the robot's next position in x or y must be outside of the respective clearance radius, Rc_x or Rc_y;
2. The robot's current euclidean distance to all obstacle centers must be less than all energy radii;
3. The robot's energy variable must be less or equal than zero.

Precondition 1 avoids obstacle collision, while 2 and 3 forbid movement in high energy zones. As the robot must sometimes wander into such zones, we specify an additional set of eight movement actions, where the first precondition holds. Preconditions 2 and 3 are changed to:

2. The robot's current euclidean distance to all obstacle centers must be greater than all energy radii;
3. The robot's energy variable must be greater than 1.

Separating movement into high and low energy sets induces the planner to perform energy actions before entering a high energy zone. As we minimize the energy variable during planning, the planner will choose plans with less actions which increase e, thus avoiding movement actions in the high energy set.

To ensure the robot stays within the bounds of the operating environment, two global constraints are added. The first stops the robot position from straying below the environment's minimum boundaries, and the second stops the robot from going past the maximum boundaries. A third constraint states that the robot's energy $e \geq 0$, to stop planner from performing infinite energy decreasing actions, as we minimize the energy variable at each plan step.

3 Practical Implementation

To test the planning domain in an autonomous navigation application, we implemented a ROS package, called "ros_enhsp", to perform path planning in a known environment. It mainly wraps a numerical planner called *Expressive Numeric Heuristic Search Planner* (ENHSP[3]), which supports the numerical expressions (i.e., euclidean distance), constraints and energy minimization as described in our domain. The package has 3 ROS nodes and 1 service, whose tasks are:

- Problem Interface (node): Obtains the robot's current positions, its goal position and the operating environment's map, calls the problem generator service to create a new STRIPS problem, and publishes the problem in a topic;
- Problem Generator (service): Called by problem interface, generates STRIPS problems;

[3]https://gitlab.com/enricos83/ENHSP-Public

- Planner Interface (node): Gets the domain and newly generated problem, calls the ENHSP planner, obtains the plan of actions and publishes it in a topic;
- Planner Dispatch (node): Parses and executes the plan published by the planner interface;

To run the package and obtain plans, the ENHSP planner must be installed, and at least the first three nodes/services must be executed. The planner dispatch node is optional, as the user may run another node to parse and/or execute the plans (e.g., move_base[4]). Except for ROS itself, and the ENHSP package and its dependencies, *ros_enhsp* has no explicit dependencies. It expects an initial position, a goal position and an occupancy grid map, all provided through ROS topics by any adequate packages.

One of the main features of our path planning solution is the minimization of the energy variable at each plan step. This is implemented in the problem generator node, as it specifies the robot's energy as the metric to be minimized by ENHSP, in the end of all generated STRIPS problems. To see the effects of this metric in the plans, the user can disable this feature by setting the "*/problem_generator/ignore_metric*" ROS parameter to "true". By default, the energy value at the STRIPS problem's goal state is zero, to ensure that the mode of operation is set to maximum efficiency when the robot is stopped at the goal.

4 Experiments and Results

To evaluate the planning domain and the implemented package, we perform two experiments. They are both set in a simulated world, with a simulated Turtlebot 2 robot[5], running in a battery-powered NVIDIA Jetson TX2 embedded computer[6]. The simulated world and robot can be seen in Figure 2.

The first experiment aims to check if the energy minimization metric significantly changes the plan produced by ENHSP. Using the same initial and goal positions, we run the planner with and without the metric, by toggling the "*/problem_generator/ignore_metric*" parameter discussed in Section 3. The plans' movement actions are translated into paths, shown in Figure 2. Both plans avoid obstacles correctly, staying away from the clearance zones (red areas). However, the plan without metric (red path) cuts into an obstacle's high energy zone (blue area), while the plan which minimizes the metric (green path) steers clear of it. Thus, energy minimization serves its intended purpose, inducing the planner to choose a low-energy path when possible. This result can also be verified by the number of energy actions in each plan. There are two energy actions in the green path: As the goal position lies within a high energy zone, the robot must go into maximum performance mode to enter it. Once it stops, the mode can be switched back to maximum efficiency. The red path contains five transitions between performance and efficiency, as it switches when going in or out of high energy zones.

[4]http://wiki.ros.org/move_base

[5]https://www.turtlebot.com/turtlebot2/

[6]https://developer.nvidia.com/embedded/buy/jetson-tx2

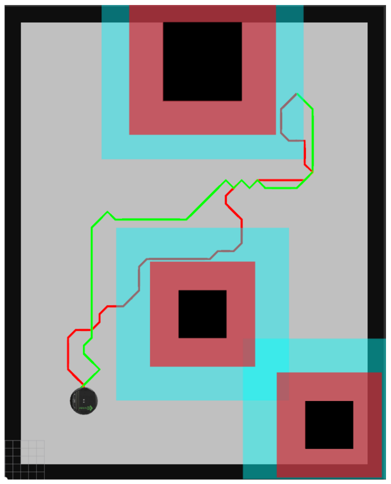

Figure 2: Path planning considering and ignoring the energy metric. The simulated world consists of obstacles (black squares), the robot (black sphere), and each obstacle's clearance and high energy zones (red and blue areas respectively). The walls also have these zones, omitted here for clarity. The red path is the plan generated by ignoring the energy metric, while the green path is the opposite.

In the second experiment, we measure how longer the robot's battery life is extended by using ros_enhsp in an autonomous navigation task, compared to the standard ROS navigation stack[7]. Specifically, we define four waypoints that the robot must visit autonomously. We power the Jetson TX2 with a 11.1 Volts, 1300 milliampere-hour Lithium-Polymer (LiPo) battery, and run the navigation task continuously until the battery dies, for the ROS navigation stack and for our package separately. We measure the battery's voltage during execution with the Jetson's internal power sensors (NVIDIA Corporation 2017). The robot starts at the same position as in Figure 2, in maximum efficiency mode.

Figure 3 shows the results for the second experiment. The battery voltage ranges from 12.4 Volts to 10.8 Volts, at which point it is considered to be discharged. The ros_enhsp package runs for approximately 5 hours, while the ROS navigation stack runs for 3.5 hours. Thus, compared to the standard solution, our path planner extends battery life by 1.5 hours, or 42.8 %.

5 Related Work

Energy efficiency is often a bonus of optimal path planning (Stentz 1994)(Kruger et al. 2007), and few works consider energy consumption as a key aspect in their planning domains (Mei et al. 2006)(Cabreira et al. 2018). An example is (Ooi and Schindelhauer 2009), where the authors propose a path planner which minimizes energy consumption for both mobility and communication, considering a robot's total distance to a goal and the transmission power required for communication from the robot's position to a fixed base station. The authors in (Plonski, Tokekar, and Isler 2013)

[7]http://wiki.ros.org/navigation

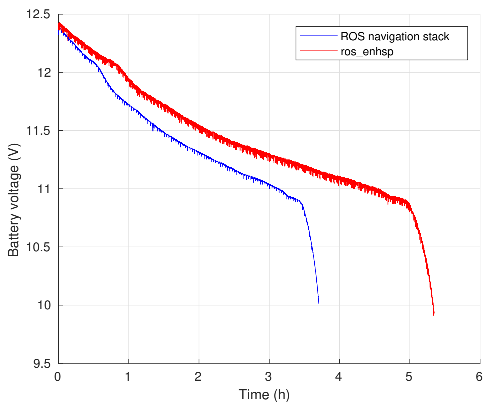

Figure 3: Battery discharge for the ROS navigation stack and the ros_enhsp package.

use dynamic programming to find energy-minimal paths for a solar-powered ground robot, based on a model for power consumption and a "solar map" of the environment. Finally, (Franco and Buttazzo 2015) explore energy-aware coverage path planning, where an energy model, derived from real measurements, is used in the planning algorithm to minimize energy consumption while satisfying requirements of coverage and resolution.

6 Conclusion

In this paper, we proposed a path planning solution which accounts for high energy zones and integrates energy-changing actions in the generated plans. We model a STRIPS planning domain to represent obstacles as 2D grid positions plus clearance zones, to which the robot must keep away to avoid collision, and high energy zones, which are avoided by choosing actions which minimize an energy variable in the robot's model, at each plan step.

We integrate the domain in a ROS package called *ros_enhsp* and test the package in two experiments. First, we verify that the planner indeed avoids high energy zones by choosing actions which minimize the energy at each step. Second, we run an autonomous navigation task continuously, with the ROS navigation stack and then with ros_enhsp, comparing the battery discharge times and concluding that ros_enhsp extends battery life by approximately 1.5 hours (42.8 %).

Our proposed solution has three main limitations. Although we achieve an extended battery life, the ROS navigation stack outperforms our package in terms of planning time. As stated in Section 2, the navigation stack runs much faster because it treats path planning as a problem of graph search, considering only movement actions. Another feature of the ROS navigation stack is dynamic obstacle avoidance by using a local planner, currently impossible in ros_enhsp.

Secondly, we model the high energy zones assuming that the robot's response time is related to the operating mode, and that the robot can avoid straying into a clearance zone by having a faster response time. As future work, those assumptions must be rigorously tested in a real environment.

Finally, the ros_enhsp package is currently specific to the Jetson TX2 embedded computer, as the energy actions directly switch the processor's operating modes. However, the energy actions in our domain are general in the sense that they only change the robot's energy variable. Thus, as future work, we can specify other instructions to save energy, such as momentarily cutting power to an unused or redundant sensor. Furthermore, the operating environment's boundaries, resolution and obstacles are currently hard-coded in the package's problem generator node, to represent the environment used in our experiments. Another future work is to build an automatic extraction tool, which receives an occupancy grid map and generates the obstacle representations for our domain, based on safety and energy margins specified by the user.

References

Cabreira, T. M.; Franco, C. D.; Ferreira, P. R.; and Buttazzo, G. C. 2018. Energy-aware spiral coverage path planning for UAV photogrammetric applications. *IEEE Robotics and Automation Letters* 3(4):3662–3668.

Franco, C. D., and Buttazzo, G. 2015. Energy-aware coverage path planning of UAVs. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, 111–117.

Kruger, D.; Stolkin, R.; Blum, A.; and Briganti, J. 2007. Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *IEEE International Conference on Robotics and Automation*, 4265–4270.

Mei, Y.; Lu, Y.-H.; Lee, C. S. G.; and Hu, Y. C. 2006. Energy-efficient mobile robot exploration. In *IEEE International Conference on Robotics and Automation*, 505–511.

Moravec, H., and Elfes, A. 1985. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, volume 2, 116–121.

NVIDIA Corporation. 2017. Tegra linux driver package R27.1. `https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-271/pdf/Tegra_Linux_Driver_Package_Release_Notes_R27.1.pdf`. Accessed on: 14-11-2018.

Ooi, C. C., and Schindelhauer, C. 2009. Minimal energy path planning for wireless robots. *Mobile Networks and Applications* 14(3):309–321.

Plonski, P. A.; Tokekar, P.; and Isler, V. 2013. Energy-efficient path planning for solar-powered mobile robots. *Journal of Field Robotics* 30(4):583–601.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Siegwart, R.; Nourbakhsh, I. R.; and Scaramuzza, D. 2011. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*, volume 4, 3310–3317.