The Thirty-Third International
FLAIRS Conference (FLAIRS-33)

Energy-Aware Path Planning for Autonomous Mobile Robot Navigation

Renan Maidana,* Roger Granada,* Darlan Jurak,* Maurício Magnaguagno*
Felipe Meneguzzi,† Alexandre Amory†
School of Technology - Pontifical Catholic University of Rio Grande do Sul – Brazil
* {renan.maidana, roger.granada, darlan.jurak, mauricio.magnaguagno}@acad.pucrs.br
† {felipe.meneguzzi, alexandre.amory}@pucrs.br

Abstract

Battery life is yet one of the main limiting factors to a robot's total mission time, and efficient energy management is paramount in a robotic application. In this paper, we integrate energy awareness in the path planning of a mobile robot performing autonomous navigation. Our contributions are: 1) The formalization of a planning domain for mobile robot path planning which accounts for energy consumption and integrates energy actions in the generated plans; 2) A proof of concept of automatic path planning that avoids high energy areas in a known environment. We test our approach in simulation, extending an embedded computer's total battery discharge time by approximately 42.8%, and in a real ground mobile robot, achieving a mean energy draw reduction of 52.02%, both compared to conventional path planning.

Introduction

In mobile robotics, state-of-the-art algorithms and methods are not typically concerned with the robot's overall energy consumption, as they require significant computing power and consider unrestricted access to the robot's resources (*e.g.*, sensors, motors, etc). The unconstrained use of these algorithms reduces a battery-powered robot's potential operating time, as not all of the robot's resources are required at all times. For example, in autonomous navigation, performing localization with all sensors always switched on is wasteful, as most robots do not need or use all sensors most of their operating time (*e.g.*, redundant or situational sensors) (Lee and Song 2004).

In this paper, we develop a path planning approach for mobile robots which produces energy-aware plans. Specifically, we define a planning domain for autonomous navigation containing *high energy zones*, where additional resources are required (*e.g.*, additional sensors must be switched on) for the robot to avoid a potential collision. Therefore, our approach accounts for actions that affect energy usage, activating or deactivating sensors according to the robot's position with respect to the energy zones. The resulting plans minimize overall energy use by activating sensors only when necessary.

To evaluate our approach, we implement it in a path planning package for mobile robots, integrated with the Robot Operating System (ROS) (Quigley et al. 2009), and perform autonomous navigation in a simulated world and the real world using a ground robot, equipped with an embedded computer powered through an individual battery. Simulation experiments emulate a sensor connected to the embedded computer by changing its computational power, while measuring the computer battery's energy draw. Real world experiments use a laser rangefinder connected to the embedded computer while executing the approach, once again measuring energy draw. These experiments show that our approach yields an energy draw reduction of 42.80% and 52.02%, for simulation and real world respectively, over the course of the robot's mission, compared to related approaches.

Related Work

Related research deals with energy efficiency by choosing paths that minimize energy consumption through graph search (*i.e.*, A* or Dijkstra), without considering different types of actions such as enabling/disabling hardware devices or computational resources. Thus, planning strategies attempt to save energy by searching for the shortest path (Stentz 1994), or avoiding known regions that require more effort from the robot's motors and consequently higher energy consumption (Datouo et al. 2017; Liu and Sun 2014; Mejri et al. 2017; Niu et al. 2018; Ooi and Schindelhauer 2009). The latter strategy resembles our approach, as we use planning to avoid areas of greater energetic consumption if possible, and act upon enabling/disabling hardware resources.

For ground robots, the main criterion for energy saving is to avoid terrain areas with a high friction coefficient and greater elevation slopes (Datouo et al. 2017; Mejri et al. 2017). Datouo et al. (2017) and Mejri et al. (2017) describe approaches where the priority is choosing a path with less energy consumption, avoiding regions of greater friction, even if the robot has to travel a longer distance relative to the shortest path. In both studies, the planning algorithm runs offline, using previously built maps and previous knowledge of the surface's friction and elevation profile. The approach in (Datouo et al. 2017) achieves a 7.5% reduction in battery energy draw, while (Mejri et al. 2017) achieves an reduction of 46%. Ooi and Schindelhauer (2009) present a graph search approach for path planning, in which the heuristic considers a robot's travel distance as well as energy consumption by

data transmission. The robot travels to a goal while computing the polynomial increase in energy consumption by data transmission, proportional to the distance from the robot to the transmitter. Results in simulation show that choosing paths near the transmitter reduce energy draw up to 22.1%, in comparison to the shortest paths.

All the related research discussed above present results in simulation. Comparable to our approach, only Liu and Sun (2014) and Niu et al. (2018) present real-world experiments for evaluation of the energy saving strategies. Liu and Sun (2014), as well as the studies discussed above, use graph search considering the energy cost of passing through areas of different coefficients of friction. Their results show a reduction of 3.5% in energy consumption, compared to the shortest path. For marine robots, Niu et al. (2018) describe an algorithm based on optimal path planning considering marine current data from an online database. After performing 15 real-world experimental runs around some Singapore islands, the authors measured an energy draw reduction up to 52.84% when compared to the energy consumption when traveling through the shortest path.

In comparison to our work, none of the related studies present energy-aware path planning with different types of actions, using instead graph search with energy minimization heuristics to discover the energy-minimal path.

Energy-Aware Planning Domain

In mobile robotics, the problem of path planning for autonomous navigation is typically treated directly as a graph search problem in a *Configuration Space* (C-Space). In summary, given the C-Space representation of a known environment, the problem is to 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, since only actions that take into account the movement are considered. However, being able to define multiple types of actions is an advantage, as energy-aware plans can be obtained by considering both movement and energy actions. To deal with movement and energy actions, we develop an Energy-Aware Path Planning (hereafter called EAPP) approach that takes into account not only the path the robot must follow but also the energy required to do so. In our approach, we model a sufficiently complex domain to allow planning in less than 30 seconds. The full domain, formalized in *Planning Domain Definition Language* (PDDL) (McDermott et al. 1998), is available for download at our ROS package's GitHub page[1].

Modeling the Environment

We begin developing our domain with the operating environment, which we represent as an occupancy grid (Moravec and Elfes 1985), *i.e.*, a 2D grid where each space can be either free or occupied by an obstacle (or a part of one). Four boundary variables (min_x, min_y, max_x, max_y) represent the world's dimensions in grid cells. Planning is performed in grid space to reduce planning time, and each obstacle is described by its grid position (C_x, C_y), where the coordinates indicate the obstacle's center in the world.

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

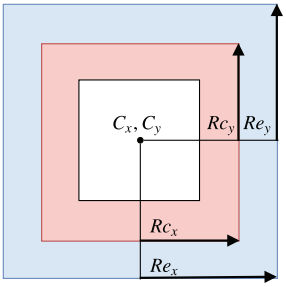


Figure 1: An obstacle, its clearance, and energy zones. The distances from the obstacle center to the boundaries of the red zone are the clearance radii, and the distances to the boundaries of the blue zone are the energy radii.

For each obstacle, two avoidance zones are calculated: the clearance radii and the energy radii. The clearance radii (Rc) specify dimensions in x and y that define an area within which the robot must not enter to avoid a possible collision. They are calculated as the distance from the obstacle's center to its outer perimeter, plus the robot's radius and a safety margin equal to half of such radius. The energy radii (Re) define the high-energy zones in x and y, where the robot is allowed to enter but must operate with additional resources (*e.g.*, with extra sensors on for added localization accuracy). The energy radii 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 to be avoided. This representation is due to computational efficiency, to speed up the planning time: If we consider every occupied grid position as an obstacle, the number of objects modeled in our domain blows up for large environments, severely impacting the planner's efficiency. The separation between clearance and energy zones is essential, as the energy radii define zones in which the robot is allowed to go when no other route is available, albeit at a higher energy cost. Figure 1 illustrates an obstacle, its zones, and sets of radii, where the blue zones represent the energy radii and the red zones represent the clearance radii. Formally, each obstacle is represented as:

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

We model the robot state 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:

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

Modeling Domain Actions

We model the domain actions by defining sets of movement and energy-related actions. Movement actions indicate the direction to which the robot's (x, y) coordinates must be increased or decreased. For example, a *move up* action increases the robot's y coordinate, while a *move right* increases the robot's x coordinate. Energy-related actions activate or deactivate a resource n (*e.g.*, switches the sensor on or off), with the effect of increasing or decreasing the robot's energy variable e by a set amount, without preconditions. As there are situations in which a robot must enter high energy zones (*e.g.*, no other path available, or the goal is within such a zone), we define two sets of actions for movement in low and high energy zones, where the latter contains preconditions specific to traversal within the high energy zones. Separating movement into high and low energy action sets induces the planner to perform energy-related actions before entering a high energy zone, where then it uses actions from the high energy movement set. As we minimize the energy variable e during planning, the planner selects plans with fewer actions that increase e, avoiding high energy movement actions. Our domain's actions are modeled generically as two sets of m movement actions, for movements inside and outside the high energy zones, and $2n$ energy-related actions (*i.e.*, switching the n resources on and off).

The preconditions for the *low energy movement* set are:

1. The robot's next position in x or y must be outside of the clearance radii (Rc_x, Rc_y) for all obstacles;
2. The robot's current position must be outside of the energy radii (Re_x, Re_y) for all obstacles;
3. The robot's energy variable e must be less than or equal to zero;

The precondition 1 avoids obstacle collision, while 2 and 3 do not allow the movement in high energy zones without altering the robot's energy variable. Precondition 3 specifically ensures that an energy action must be performed before moving out of a high energy zone (*i.e.*, the robot's energy must be decreased). As the robot must sometimes enter such zones, we specify the *high energy movement* action set, where the first precondition also holds. Instead of using the preconditions 2 and 3 from low-energy actions, the movement in a high energy zone uses the following preconditions:

2. The robot's current position must be within an energy radius (Re_x, Re_y) for at least one obstacle;
3. The robot's energy variable must be greater than zero;

Two global constraints are added to ensure the robot stays within the bounds of the operating environment:

$$(\text{Robot}_x \geq \min_x) \cup (\text{Robot}_y \geq \min_y) \quad (1)$$

$$(\text{Robot}_x \leq \max_x) \cup (\text{Robot}_y \leq \max_y) \quad (2)$$

A third constraint states that $e \geq 0$, to stop the planner from performing infinite energy decreasing actions, as we minimize the energy variable at each plan step.

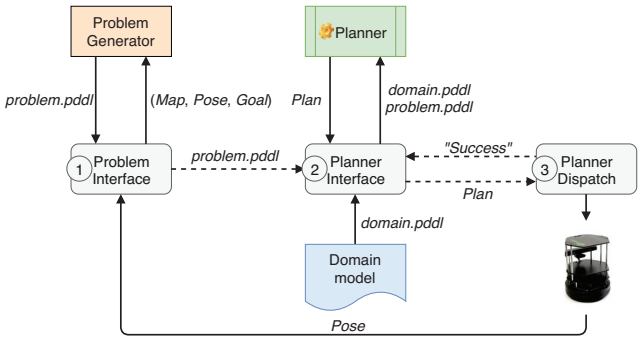


Figure 2: Pipeline of our energy-aware path planning package, where nodes are represented in gray and the service in yellow.

ROS Package

In order to integrate our energy awareness planner with a mobile robot to perform autonomous navigation in a known environment, we developed a package using the Robot Operating System (ROS) robotics framework. The package wraps a numerical planner called *Expressive Numeric Heuristic Search Planner* (ENHSP[2]) (Scala et al. 2016), which supports numerical expressions (*e.g.*, Euclidean distance), constraints and energy minimization as described in our domain. The package contains 3 ROS nodes and one service, whose tasks are:

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

Our package implements the pipeline illustrated in Figure 2, where dashed lines represent published topics and the solid lines represent data obtained through files (*e.g.*, PDDL domain) or passed directly between the nodes. Initially, the *Problem Interface* (1) reads the map, the pose of the robot and a goal position informed by the user or an external node, and calls the *Problem Generator* service, passing these three parameters. The *Problem Generator* builds a PDDL problem (*problem.pddl*) based on its inputs and publishes this problem in a topic. As soon as the problem is published, the *Planner Interface* (2) reads it and executes the ENHSP planner as a subprocess, which uses the domain description (*domain.pddl*) and the problem description (*problem.pddl*) to compute an action plan for the robot. The *Planner Interface* publishes this plan as a topic, which is read by the *Planner Dispatch* (3). This node reads the plan, parses the movement

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

actions into a path, based on the robot's current position, and executes it in the robot. When the robot reaches the goal position, a message of "success" is published as a topic. The *Planner Interface* reads the "success" message and may either stop the process or set a new goal for the robot. For this package, we formalized a PDDL domain with 8 movement actions (up, left, right, down and the diagonals in-between), in each of the two movement sets, and 2 energy actions for turning a sensor on and off, totaling 18 actions.

Experiments and Results

In order to measure the efficiency of our energy-aware path planning approach (*i.e.*, the reduction in energy draw), we perform experiments in a simulated world and the real world using a Turtlebot 2 mobile base[3] (with a radius of 0.2 meters) and an NVIDIA Jetson TX2 embedded computer[4], powered by an individual 11.1 Volts 2300 miliampere-hour Lithium-Polymer (LiPo) battery. The Jetson TX2 was chosen because it contains built-in power sensors connected to the board's power supply. Experiments in the simulated world allow us to ensure that our domain and package are correctly designed, as well as to measure the efficiency of our EAPP approach. In the real world experiments, we compare the energy draw from the Jetson's battery with a laser sensor connected to the embedded computer when using our package and a conventional path planning package. The energy draw E (in Joules) is calculated by measuring and logging the battery's instant power draw through the Jetson TX2's internal power sensor, and by integrating the measurements over time. Formally, the energy can be described in continuous and discrete-time.

$$E = \int_0^T P(t)\, dt = \sum_{i=0,k,2k\ldots}^{T} P[i] \qquad (3)$$

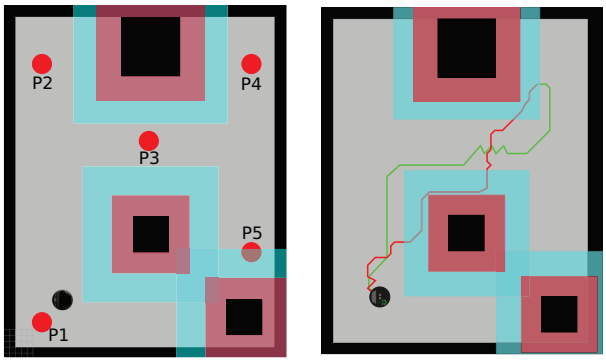
where $P(t)$ and $P[i]$ are the instant power in Watts, at continuous and discrete-time respectively, T is the total experiment time, and k is the discrete sampling time. As a Watt equals to Joules per second (*i.e.*, $W = J/s$), integrating the power eliminates the seconds and gives us the total energy in Joules over a time period T.

Simulation

In order to test our energy aware planner in an autonomous navigation application, we conducted two experiments using a simulator. We use the Stage (Vaughan 2008) robot simulator, and run the Turtlebot 2 packages in the Jetson TX2 and Stage in a separate networked computer, emulating the setup of a real Turtlebot 2 driven by the embedded board. Figure 3a illustrates the simulated world, where the black circle represents the robot, black squares represent obstacles and red circles represent the waypoints for autonomous navigation ($P_{1..5}$), which are followed sequentially (*i.e.*, P_1 to P_5 in a cycle). As described in Figure 1 the red zone represents the clearance radii and the blue zones represent the energy radii.

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

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



(a) (b)

Figure 3: (a) Map used in simulation, where the red circles indicate the five waypoints. (b) Path planning considering the energy metric (green path) and ignoring the energy metric (red path). The simulated world consists of obstacles (black squares), the robot (black circle), 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 first simulation checks if the energy minimization metric significantly changes the plan produced by ENHSP. Using the robot's initial position in the map and an arbitrary goal position inside a high energy zone, we execute the planner with and without the metric. The plans' movement actions are translated into paths between these points, as shown in Figure 3b. Both plans avoid obstacles correctly, staying away from the clearance zones (red areas). As EAPP without the minimization metric is not concerned with the overall energy draw, the plan (red path) cuts into an obstacle's high energy zone (blue area). On the other hand, when we set EAPP to minimize the energy draw, it steers clear of such zones (green path). Thus, energy minimization serves its intended purpose, inducing the planner to choose a low-energy path when possible. We can see this result by verifying 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 activate its resources to enter it. Once it reaches the goal, the second energy action is performed, as the robot is stopped and its resources can be freed or deactivated. The red path contains four resource transitions: One at the goal, plus three when going in or out of high energy zones.

In the second simulation, we execute an autonomous navigation task, first with EAPP and then with the standard ROS navigation stack[5], measuring the computer's battery voltage and power at 1-second intervals with the Jetson's internal sensors. We power the Jetson with its individual LiPo battery, and execute the navigation task continuously until the battery dies, cycling between the waypoints in Figure 3a, for the ROS navigation stack and our package separately. The objectives are to compare the energy draw for both approaches over one hour (with a sampling time of 1

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

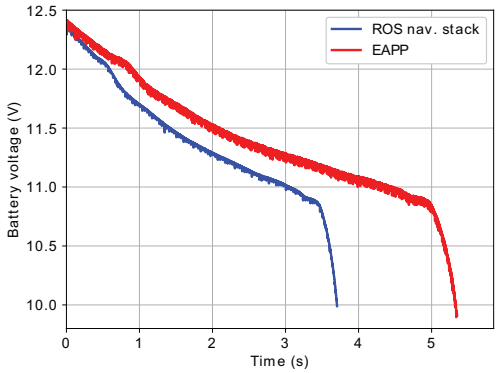


Figure 4: Battery discharge for the ROS navigation stack and the EAPP approach.

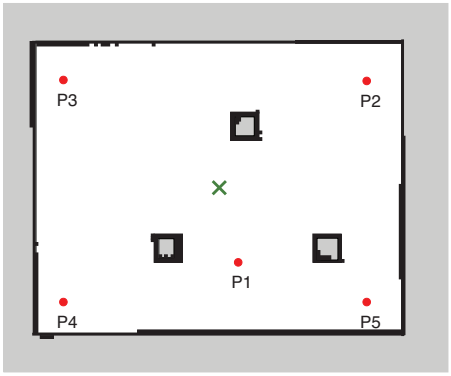


Figure 5: Map of the real world experiment environment, created with the SLAM package. The squares with black borders represent obstacles, red circles represent waypoints, and the green cross represents the center of the map, where the robot is initially positioned.

second), and how long the Jetson's battery life is extended by using our EAPP approach. As in simulation we do not have a real laser rangefinder connected to the board, we set the operating mode of the Jetson TX2 to emulate the battery consumption of a real sensor. In our experiments, we set two modes of operation: *high performance* (also called *Max-N* by NVIDIA) and *high efficiency* (also called *Max-Q* by NVIDIA). In *high performance* mode, the Jetson board uses two dual-core Denver 2 processor and four quad-core ARM Cortex-A57, both operating at 2.0 GHz frequency. In *high efficiency* mode, the board uses only the four quad-core ARM Cortex-A57 operating at 1.2 GHz. When the experiment starts, the Jetson TX2 computer is set to the *high efficiency* mode.

Integrating the current draw over one hour (3600 seconds) using Equation 3, we see that autonomous navigation with EAPP draws 2053.65 Joules from the battery, a 44.18% reduction compared to the 3678.87 Joules drawn when using the ROS navigation stack. Figure 4 shows the battery discharge when executing the ROS navigation stack (blue line) and our approach (red line). The battery voltage ranges from 12.4 Volts to 10.8 Volts, at which point it is considered to be discharged. Our package executes the path planning for approximately 5 hours, while the ROS navigation stack executes for 3.5 hours. Thus, our EAPP approach extends battery life by 1.5 hours or 42.8% compared to the standard approach.

Real World

After performing simulations using our approach and the ROS navigation stack, we perform experiments in the real world using a Turtlebot 2, to measure energy draw and to check if the EAPP approach is capable of successful autonomous navigation in a real environment. In this experiment, we demonstrate the management of a different resource, controlling the activation of a Hokuyo URG-04LX-UG01[6] scanning laser rangefinder. In the EAPP approach, the laser is activated when the robot enters in energy zones or stops at the goal position, in which case the laser scans the environment to update its current position and avoid odometry drift over time. To physically activate and deactivate the laser, we use a modified USB hub whose USB ports' power lines are controlled by a 5-channel relay driver circuit, which is controlled by the Jetson TX2's GPIO ports. When the laser needs to be activated, a GPIO port's state is set to "*high*", and vice-versa.

Before performing path planning, a map of the environment must be created using the laser scanner. Using the *gmapping* ROS slam package[7], we create a map of the field containing three obstacles, as shown in Figure 5. With the map, we set five waypoints ($P_{1...5}$) to which the robot navigates autonomously. The robot navigates sequentially from waypoints P_1 to P_5 and then cycles back to P_1. With fully charged batteries, we set the robot at the center of the map (indicated as a green cross in Figure 5) and start the path planning task. We execute the EAPP and ROS navigation approaches five times each, once again measuring the power draw at 1-second intervals over one hour, and later obtaining the energy draw with Equation 3.

Figure 6 presents the energy draw for the conventional path planner and our EAPP approach, for each of the five experimental executions, along with their standard deviations represented as error bars. The mean energy draw for EAPP is 6243.49 Joules in one hour, a reduction of 52.02% relative to the mean 13011.69 Joules from the conventional path planner. The EAPP's slightly larger standard deviation is due to variations in the obstacle coordinates, depending on the quality of the map produced by the gmapping SLAM package. For example, in execution 3, the obstacles centers are slightly offset because of variations in the generated map. As some of the obstacles' high energy zones overlap, the planner needs to turn on the laser sensor when crossing between them, slightly increasing the overall energy draw. The opposite happens in execution 4, and the overall energy draw is decreased. With EAPP, the laser sensor was active for 40.45% of the one hour experiment time (*i.e.*, spent 59.55%

[6]https://www.hokuyo-aut.jp/search/single.php?serial=166

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

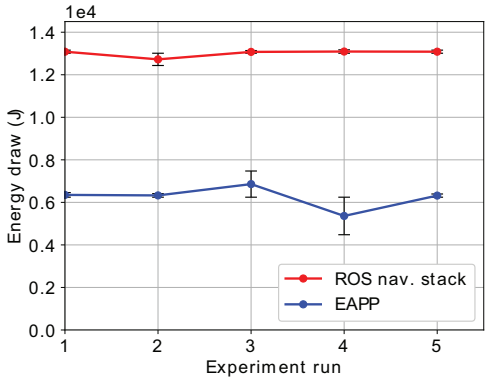


Figure 6: Energy draw with standard deviation for five experimental executions of the ROS navigation stack and the EAPP approach respectively, performing autonomous navigation for one hour in each execution.

of the time deactivated). Finally, in regards to successful autonomous navigation, EAPP is capable of providing plans to safely navigate through the five waypoints in Figure 5.

Conclusion

In this paper, we developed a path planning approach that accounts for high energy zones and integrates energy-changing actions in the generated plans. Our approach could be applied in disaster scenarios, in which using paths with less battery discharge represents more mission time. In simulation, we saw that our approach indeed avoids high energy zones by choosing actions which minimize the energy at each step. When compared with the ROS navigation stack, our EAPP approach extends battery discharge time by approximately 1.5 hours (42.8%) and reduced the consumption by 44.18% over one hour. In real-world experiments, we execute EAPP with a laser sensor connected to a ground robot. Comparing our results with the ROS navigation stack, we observed a reduction of 52.02% in energy draw.

Our approach has three main limitations. First, although we achieve a lower energy draw and extended battery discharge time, the ROS navigation stack outperforms our package in terms of planning time. Second, the ROS navigation stack performs dynamic obstacle avoidance by using a local planner, currently not a concern in the EAPP package. Third, our domain does not model the energy cost of movement (as there is an individual power supply for the robot computer), and thus the planner does not consider such information. The advantage in modeling this cost is, for example, if the energy-minimal path is a significant detour to a goal due to high energy zones, and it may be overall less costly to choose the shortest path, regardless of the high energy zones.

As future work, we intend to improve the physical hardware and software for activating and deactivating sensors to avoid or reduce failures related to sensor shutdown. Secondly, we will power the Jetson TX2 and its sensors through the Turtlebot 2's power supply, allowing us to measure and model the cost of movement, which will then be integrated in our planning domain. Finally, we plan on testing EAPP in different platforms (*e.g.*, Unmanned Surface Vehicles or Unmanned Aerial Vehicles) with different energy actions, to verify if the energy draw reduction holds.

Acknowledgments

The authors acknowledge support from CNPq under project numbers 407058/2018-4 and 305969/2016-1, FAPERGS process number 18/2551-0000500-2 and CAPES under the project 88887.115590/2015-01, Pro-Alertas program. We also would like to thank the NVIDIA Corporation for the donation of the Jetson TX2 embedded computer used in this work.

References

Datouo, R.; Motto, F. B.; Zobo, B. E.; Melingui, A.; Bensekrane, I.; and Merzouki, R. 2017. Optimal motion planning for minimizing energy consumption of wheeled mobile robots. In *ROSBIO 2017*, 2179–2184.

Lee, S., and Song, J.-B. 2004. Robust mobile robot localization using optical flow sensors and encoders. In *ICRA 2004*, 1039–1044.

Liu, S., and Sun, D. 2014. Minimizing energy consumption of wheeled mobile robots via optimal motion planning. *IEEE/ASME Transactions on Mechatronics* 19(2):401–411.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language. *AIPS-98 Competition Committee*.

Mejri, E.; Kelouwani, S.; Dube, Y.; Trigui, O.; and Agbossou, K. 2017. Energy efficient path planning for low speed autonomous electric vehicle. In *IEEE-VPPC 2017*, 1–6.

Moravec, H., and Elfes, A. 1985. High resolution maps from wide angle sonar. In *ICRA 1985*, 116–121.

Niu, H.; Lu, Y.; Savvaris, A.; and Tsourdos, A. 2018. An energy-efficient path planning algorithm for unmanned surface vehicles. *Ocean Engineering* 161:308–321.

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

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*.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *ECAI 2016*, 655–663.

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 *ICRA 1994*, 3310–3317.

Vaughan, R. 2008. Massively multi-robot simulation in stage. *Swarm Intelligence* 2(2):189–208.