Software Impacts 10 (2021) 100159

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

SBMPO: Sampling Based Model Predictive Optimization for robot trajectory planning

Mario Harper [a,*], Camilo Ordonez [b], Emmanuel Collins [c]

[a] Utah State University, Logan, UT, United States of America
[b] Center for Intelligent Systems, Controls and Robotics (CISCOR), FAMU-FSU College of Engineering, Tallahassee, FL, 32310, United States of America
[c] Louisville Automation and Robotics Research Institute (LARRI), University of Louisville, Lousville, KY 40292, United States of America

ARTICLE INFO

Keywords:
Motion planning
Model Predictive Control
Optimization
Kinodynamic planning

ABSTRACT

Sampling-Based Model Predictive Control (SBMPO) is a novel nonlinear MPC (NMPC) approach that enables motion planning with dynamic models. This tool is also well suited to solve traditional MPC problems and has been tested in various situations ranging from robotics, task scheduling, resource management, combustion processes, and general optimization.

Code metadata

Current code version	v2.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2021-108
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/6106440/tree/v1
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	C++, some python and Matlab scripts are also included for visualization
Compilation requirements, operating environments & dependencies	See README and install guide found at https://github.com/DIRECTLab/SBMPO/blob/main/README.md and https://github.com/DIRECTLab/SBMPO/blob/main/QuickStart.md
If available Link to developer documentation/manual	For example: https://github.com/DIRECTLab/SBMPO/blob/main/README.md/
Support email for questions	mario.harper@usu.edu

1. Introduction

Sampling-Based Model Predictive Optimization (SBMPO) generates a cost-optimal trajectory of feasible control inputs, which, if executed, enables mobile robots, manipulators, aerial platforms, or general dynamical systems to operate intelligently in complex environments for challenging tasks. This trajectory directly accounts for situation-specific input and output constraints, system dynamics, and efficient sampling for effective scaling in high dimensions. In addition, the cost functions employed can involve physically motivated quantities such as distance, time, or energy.

This algorithm has successfully demonstrated planning capabilities on a variety of mobile robots such as autonomous ground vehicles (AGVs), autonomous air vehicles (AAVs), autonomous underwater vehicles (AUVs), and legged robotic platforms operating under complex environmental constraints [1–7].

The SBMPO algorithm is a tree-based sampling algorithm that combines aspects from classical A* with sampling methods to generate efficient tree structures that can be optimized. SBMPO utilizes an implicit grid to reduce computation. This implicit grid is created through a user defined grid resolution and grid axis (generally a robot pose of x, y, θ). Upon expansion, nodes are checked within the same grid for other occupants, if another node exists, the node with the higher cost is pruned. This additional layer of check reduces the memory footprint of the algorithm and improves scalability as SBMPO does not conduct a grid-based search (which increases expansions by $N = k^d$ where

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail address: mario.harper@usu.edu (M. Harper).

https://doi.org/10.1016/j.simpa.2021.100159
Received 24 August 2021; Received in revised form 26 September 2021; Accepted 7 October 2021

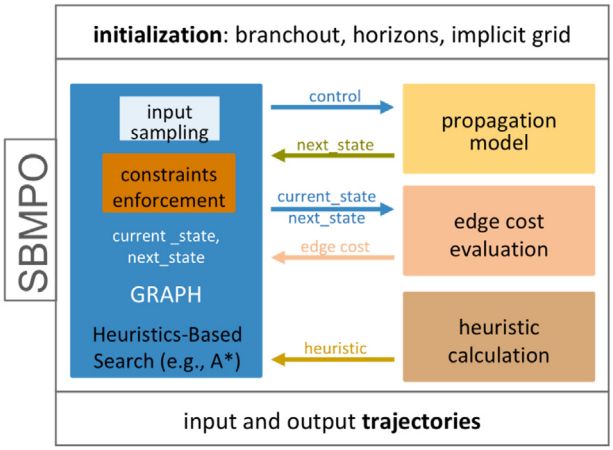

Fig. 1. SBMPO computes the change in system states from sampled control inputs, and uses a cost function to prioritize exploration regions. Sampling in the control space ensures that the generated trajectories are admissible, and SBMPO returns the trajectory of control commands directly, rather than relying on an inverse-kinematic solver to compute the controls.

k is the fixed points per axis, and d is the dimensions) [8]. Due to the tree-based nature, SBMPO scales well to higher dimensions while maintaining convergence speeds and performance guarantees.

This algorithm also benefits from being sampling-based. Sampling based approaches are often "anytime" in nature, meaning the algorithm can be terminated at any time and still return a feasible, if not optimal, solution. These algorithms are also more capable of accounting for complexities due to reduced memory requirement. While there are many merits to sampling-based approaches, this family of algorithms suffers from sub-optimality in practice. They rely on an asymptotically optimal guarantee that requires long-iteration times. SBMPO gives optimal guarantees at the expense of the "anytime" nature.

Most state-of-the-art planning methods typically account for complexities using two sub-planners: deliberative and reactive. The deliberative planner generally assumes simplified motion models to plan the overall path generation and a steering function that computes the necessary velocities to follow the path. A reactive planner then operates to track the path and constantly replans paths to account for complexities not considered in the deliberative planner, such as dynamic behaviors. SBMPO directly considers the impact of non-trivial complexities throughout the overall optimization by directly using a dynamic model in the planning process, alleviating the cumbersome necessity of needing sub-planners.

SBMPO has successfully conducted navigation tasks in the face of complexities without the need for sub planning. Examples include terrains that drastically alter UGV motion: grass, vegetation, and steep hills [1,2]. Similarly, underactuated manipulators have lifted heavy objects using knowledge of system dynamics, even when the weight exceeded the forces motors could provide [9]. While complexities differ based on application, several typical situational constraints are included in our software and documentation on adding new modules is found in the README file at: https://github.com/DIRECTLab/SBMPO/.

2. Architecture

SBMPO is comprised of model specific components (propagation model, edge cost evaluation, heuristic calculations), and the optimization engine (see Fig. 1). As long as a model is defined with these model specific components, SBMPO is capable of optimizing a plan based on minimizing a defined cost function .

2.1. Adding new models

A new model may be added by creating a new .cpp file and associated header file. This new model file must contain the model specific components as independent methods. Each of these three methods must be declared with certain inputs as shown below:

```
State model::[new_model]::next_state(const State& state,
     const Control& control,
     float integ_step_size)

float model::[new_model]::cost(const State& current,
     const State& next,
     const Control& control)

float model::[new_model]::heuristic(const State& current,
     const State& goal,
     const Control& control)
```

These new models will be placed in the /lib/src/model (for .cpp files) and /lib/src/include (for .h files). The make file is then updated to include the new models following the same pattern as for the example cases (Section 3).

2.2. New models example

We illustrate a simple example (called ACplanner) where it is desired to minimize dollar costs of running an A/C unit. This system has one control (control[0]) that determines if the A/C system is on or off. The state contains time, current temperature of the system, and outside temperature. The state and control parameters will be initialized in the config.json file (see README for details), and the .cpp file is built as follows:

```
State model::ACplanner::next_state(const State& state,
    const Control& control,
    float integ_step_size) {

   State newState;
   State states = state;
   newState.resize(state.size());
   float coolRate = 1.5; //How quickly things cool
   float heatRate = 0.125; //How quickly things heat
   int timeStep = state[0];
   float oTemp = 39; //Assume fixed temperature outside
   float price = 0.17; //Assume fixed price per kilowatt

   newState[0] = timeStep + 1; //next state time
   newState[1] = state[1] - coolRate*control[0] +
     heatRate*(oTemp - state[1]); //next state temp
   newState[2] = price; //logging price
   return newState;
}


float model::ACplanner::cost(const State& current,
   const State& next,
   const Control& control) {

   double cost; float Power = 3.5/4;
   cost = Power*control[0]*next[2]; //Returns dollar cost
   return cost;
}


float model::ACplanner::heuristic(const State& current,
   const State& goal,
   const Control& control) {
   double h;
   h = 0.0; //No heuristic, this will do an exhaustive search
   return h;
}
```

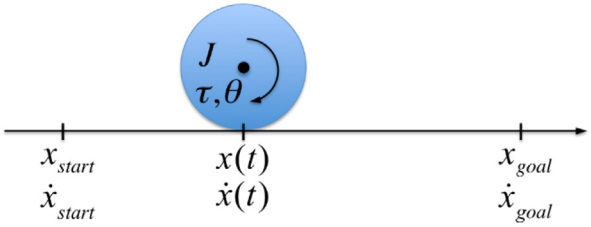

Fig. 2. 1D Motion planning example. A rolling wheel on a level surface.

This creates a simple dollar minimizing A/C temperature planner. The propagation (next state) method computes a new temperature of the system in one time step based on fixed cooling and heating efficiencies, and if the A/C unit is on. The cost function multiplies dollar cost of power with a set power draw if the system is on. There is no heuristic calculated, illustrating that without a heuristic the optimal solution will still be found. Additional constraints (such as maximum or minimum allowed system temperature) can be added into the config.json file as described in the README.

3. Usage and examples

Three example implementations come integrated with the software for easy prototyping and development. These examples are accessed via a command-line argument 'example=[1D kinematic]', which specifies the example to run. Specific instructions on running the examples and interpreting or visualizing results can be found in the software documentation (https://github.com/DIRECTLab/SBMPO/blob/main/README.md).

In the first (and default) example, a 1D planning task is presented. In this scenario, a simple double integrator corresponding to the rolling wheel of Fig. 2 is considered. The control input corresponds to the linear acceleration of the wheel.

We provide options to alter parameters within a config file (https://github.com/DIRECTLab/SBMPO/blob/main/resources/kinematic/config.json) , Table 1 lists some of these common parameters. There are also choices of three optimization criteria: distance, velocity, and time. Distance optimal planning will optimize to minimize the total distance traveled by looking at the simple difference in the desired position from the current position. The velocity flag will optimize to minimize total distance while being aware of acceleration constraints. Finally, the time-optimal paradigm will minimize time used to get to the goal location. This example showcases how simple constraints and models can be implemented with custom definitions of optimization objectives.

The second example is a skid-steered UGV controlled by left and right wheel velocities while performing a typical motion planning task in an obstacle field. The motion planning tasks minimizes the distance needed to move from a start to goal location. A start pose (coordinates in [x,y] and orientation [theta]), goal location, and obstacle information can be easily changed by the user. Obstacles are default defined in the obstacles.txt file with three markers: x position, y position, and radius. Circular obstacles are assumed for ease of integrating with point-cloud sensors such as lidar or depth perception cameras.

The final example runs the same skid-steered robot but uses an energy model instead. (For details on the energy models for the skid-steered, refer to [1]). This energy model change allows optimization to minimize energy based on an analytical function. The final trajectory is feasible and optimal, honoring the vehicle motor-power and body geometry constraints.

Table 1
Key parameters required for all motion planning tasks.

Parameter Variable	Description
Start, Goal	Define initial and final states
Max Iterations	Maximum number of expansions allowed before terminating search
Branchout	Number of children generation during expansion
Samples	Sampling method (e.g., fixed samples, halton samples, random samples, etc.)
Sampling Time	Time resolution used to propagate system model during expansions (shorter is more accurate at the cost of increased computing time. User should ensure that Max Iterations is increased when the sampling time is reduced.
Constraints	Defines constraints on control inputs. This can be a simple constraint such as minimum and maximum torque (see $config.json$ file in the 1D example) or expanded for multiple inputs such as turn radius and commanded velocity ($config.json$ in kinematic example).

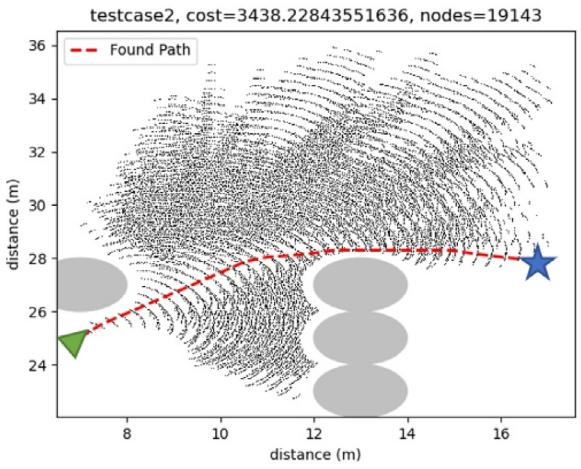

Fig. 3. Expanded Nodes, obstacle, and optimized trajectory for an energy-efficient path on a Clearpath Husky skid-steered UGV. Default visualizations only plot obstacles and the final path.

In both UGV examples, different states and other variables can be logged. The default logged parameters are x coordinate, y coordinate, heading angle, right wheel angular velocity, left wheel angular velocity, turn radius. The obstacle file can be altered by adding (or removing) rows. Each row contains three columns corresponding to the location [x,y] and radius of the obstacle. This obstacle file can read in point clouds if the user wishes to use sensor feedback from hardware.

After successful execution of the algorithm, resulting trajectories are generated as a JSON file. This JSON file is organized with a control and state vector at each time interval. A helpful plotting and visualization script has been provided in both python and Matlab to parse and display the results Fig. 3. Detailed descriptions of the outputs and their meanings can be found in the README (Interpreting Results subsection) accompanying this software.

4. Impact

This software has successfully optimized motion plans for several different platforms, including autonomous underwater vehicles, skid-steered vehicles, robotic arms, and spacecraft. Additionally, as this

planning paradigm enables planning for highly non-holonomic systems, it is used as the algorithm of choice for several dynamic and unique mobility systems, including the first intelligent climbing robot [10].

The provided examples and framework can be tailored to solve difficult planning problems such as snow/ice surface navigation, moving on slopes, planning for autonomous boats/submarines, as well as other complex dynamic vehicles. Increased success in many complex planning applications will be possible due to the unique treatment of our algorithm of: 1. Flexible model structures: This software can optimize many types of complex input–output relationships by allowing flexibility to dictate models. This is demonstrated in simulation of a dynamic-legged robot where whole-body motions are difficult to determine analytically. In this case, a machine learning algorithm was trained and implemented as the SBMPO internal model. Users can define models however they wish without reconfiguring or changing the optimizer. 2. Optimization criteria: The user-defined cost functions determine optimizations. Several cost functions are pre-configured (energy, time, and distance) for convenience in the integrated examples of our software. In the case of energy-based planning, the cost function is written in terms of energy consumption. Final trajectories minimize the overall energy required to attain the desired goal criteria. 3. Goal conditions: The overall objective of this planner is generally to get to a spatial coordinate. However, the goal criteria are flexible and can be adjusted by the user to be different (ex: move for a given time and cover a particular area). Application-specific success criteria can be defined easily by the user. By having the flexibility to define application-specific models and optimization criteria, this tool can leverage the user's subject matter understanding and provides a powerful optimizer tailored for specific tasks in robotics, controls, general optimization, and reinforcement learning

SBMPO uses a model of the robotic system to generate a graph like the one shown in Fig. 3. This algorithm also can optimize for unique and challenging cost functions. These are often based on complex dynamic models, as in the case of energy-efficient planning [1].

5. Ongoing development

Currently, this algorithm uses the LPA* (Lifelong Planning A*) optimizer, which operates on the generated tree. Additional optimization techniques such as MHA* (Multi-Heuristic A*) are planned to be incorporated to allow faster optimization in complex domains where a heuristic function may be hard to define (as in complex mobility robots). While C++ allows for faster convergence, a python and Matlab version of the software will be created. This development undertaking will be done to embed machine learning libraries and allow for faster adoption and development. Direct ROS integration is also planned. Another additional improvement is the ability to work beyond simple 2D maps. More complex map objects will be used to add interesting motion constraints experienced while traversing vegetation, hills, and soft terrains.

CRediT authorship contribution statement

Mario Harper: Software, Writing – original draft, Investigation. **Camilo Ordonez:** Investigation, Formal analysis. **Emmanuel Collins:** Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the collaborative participation in the Robotics Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD 19-01-2-0012. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation thereon.

References

[1] N. Gupta, C. Ordonez, E.G. Collins, Dynamically feasible, energy efficient motion planning for skid-steered vehicles, Auton. Robots 41 (2) (2016) 453–471, http://dx.doi.org/10.1007/s10514-016-9550-8.

[2] C. Ordonez, N. Gupta, O. Chuy, E.G. Collins, Momentum based traversal of mobility challenges for autonomous ground vehicles, in: 2013 IEEE International Conference on Robotics and Automation, 2013, http://dx.doi.org/10.1109/icra.2013.6630657.

[3] C. Ordonez, N. Gupta, E.G. Collins, J.E. Clark, A.M. Johnson, Power modeling of the Xrl hexapedal robot and its application to energy efficient motion planning, Adapt. Mob. Robot. (2012) 689–696, http://dx.doi.org/10.1142/9789814415958_0088.

[4] G. Francis, E. Collins, O. Chuy, A. Sharma, Sampling-based trajectory generation for autonomous spacecraft rendezvous and docking, in: AIAA Guidance, Navigation, and Control (GNC) Conference, arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2013-4549. URL https://arc.aiaa.org/doi/abs/10.2514/6.2013-4549.

[5] M. Harper, J. Pace, N. Gupta, C. Ordonez, E.G. Collins, Kinematic modeling of a RHex-type robot using a neural network, SPIE, Unmanned Syst. Technol. XIX 10195 (2017) http://dx.doi.org/10.1117/12.2262894.

[6] M.P. Austin, M.Y. Harper, J.M. Brown, E.G. Collins, J.E. Clark, Navigation for legged mobility: Dynamic climbing, IEEE Trans. Robot. 36 (2) (2020) 537–544, http://dx.doi.org/10.1109/TRO.2019.2958207.

[7] C.V. Caldwell, D.D. Dunlap, E.G. Collins, Motion planning for an autonomous underwater vehicle via sampling based model predictive control, in: OCEANS 2010 MTS/IEEE SEATTLE, 2010, pp. 1–6, http://dx.doi.org/10.1109/OCEANS.2010.5664470.

[8] S.M. LaValle, Planning Algorithms, Cambridge University Press, USA, 2006.

[9] O. Chuy, J. E.G. Collins, A. Sharma, R. Kopinsky, Using dynamics to consider torque constraints in manipulators planning with heavy loads, Trans. ASME, J. Dyn. Syst. Meas. Control (2016).

[10] M.Y. Harper, J.V. Nicholson, E.G. Collins, J. Pusey, J.E. Clark, Energy efficient navigation for running legged robots, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 6770–6776.