A Dataset Generation Tool for Deep learning-based Motion Planning in Complex Environments

Muhammad Usman Sarwar
Dept. of Computer Science
COMSATS University
Islamabad – Lahore Campus, Pakistan
ORCID: 0000-0001-6252-0034

Moman Sohail
Dept. of Computer Science
COMSATS University
Islamabad – Lahore Campus, Pakistan
ORCID: 0000-0002-3275-3774

Muhayy Ud Din
Inst. of Industrial and Control Eng.
Universitat Politècnica de Catalunya
Barcelona, Spain
ORCID: 0000-0001-6214-1077

Jan Rosell*
Inst. of Industrial and Control Eng.
Universitat Politècnica de Catalunya
Barcelona, Spain
ORCID:0000-0003-4854-2370

Wajahat M Qazi
Dept. of Computer Science
COMSATS University
Islamabad – Lahore Campus, Pakistan
ORCID:0000-0002-6970-8328

***Abstract*—Learning-based approaches for motion planning have gained much attention. These approaches show better performance in terms of computational efficiency as compared to the tradition approaches, such as sampling-based motion planning. One of the key challenges for learning-based approaches is to generate training datasets. This work-in-progress proposes a generalized dataset generation approach for motion planning that allows to generate simulation data for motion planning under geometric, kinodynamic and physics-based constraints. The generated dataset can easily be encoded in any required format for training. This tool will be used at the Institute of Industrial and Control Engineering (IOC-UPC) to test learning-based motion planning strategies for dual arm mobile manipulators under physics-based constraints. As a first step, in this study the developed approach is tested by generating datasets for mobile robots. The generated datasets are used to train *motion planning networks*, an approach for deep learning-based motion planning.**

***Index Terms*—Motion planning, Deep neural networks, dataset generation.**

I. Introduction

Motion Planning is an essential component to solve manipulation planning problems for robots, ranging from simple mobile robots to high dimensional robotic systems, such as dual-arm manipulators with anthropomorphic hands. The main objective of motion planning is to compute collision-free paths from a start to a goal state in the configuration space. Over the last decades, various sampling-based planning approaches have been proposed, such RRT [1] along with their variants like the bidirectional RRT [2] or the optimization variant RRT* [3]. These planning algorithms work efficiently for low-dimensional spaces but, for high dimensions, the planning time may take too long for some applications. To overcome this challenging issue, deep learning-based motion planning approaches have emerged recently.

The deep learning-based approaches for motion planning use sample datasets of valid motion paths generated for random environments. These sample datasets are used to train deep convolutional neural networks that, once trained, are able to compute very fast the path from a start to a goal state in a given environment. Learning-based approaches have shown significant improvements in terms of computational time over the classical sampling-based approaches, although the accuracy is highly dependent on the training data, i.e. accurate and large datasets leads to a better performance of the deep learning-based approaches. The problem is that the generation of these training datasets is one of the most critical and computationally intensive tasks in deep learning-based approaches. Various techniques have been used for dataset generation, such as human demonstration or the execution of the task to be learned in the real system. These have very interesting features, although may be very time consuming and difficult to prepare when a reasonable amount of data is required. Alternatively, the approach most commonly used among learning based-approaches, is to simulate the real system and generate data in an automated way. However, in this case the challenging issue to be handled is the precise modeling of the real world scenarios and the automation of the data generation. In the case of motion planning, the generation of the paths also implies the implementation of the motion planning algorithms whose behavior is to be learned.

This study proposes a software tool for automated motion planning data generation for learning-based approaches. It allows to generate datasets under geometric, kinodynamic and physics-based constraints, and also allows to easily incorporate the required enhancements in the motion planner core to generate data under certain motion planning constraints. Moreover, it provides a data encoder which converts the generated motion planning data into any desired format.

After this introduction, the rest of the paper is structured as follows. Sec. II presents some related previous works, Sec. III

*This work was partially supported by the Spanish Government through the project PID2020-114819GB-I00. Email: jan.rosell@upc.edu.

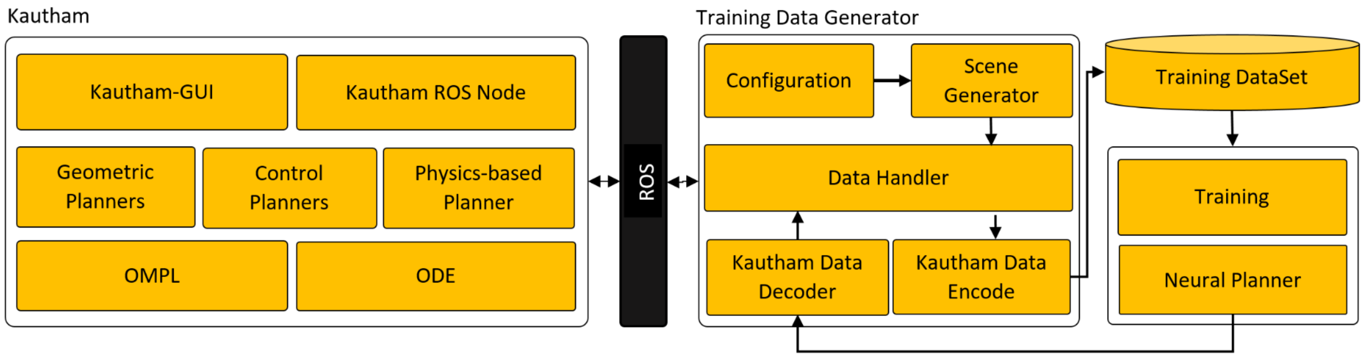


Fig. 1. Framework for training dataset generation.

details the proposal, that includes the framework and the data generation procedures and GUI, and Sec. IV some preliminary results and discussion. Finally, Sec. V sketches the conclusions and future work.

II. Related Work

Deep learning-based approaches for motion planning have recently gained much attention. Various approaches are proposed to address key motion planning issues. For instance, Motion Planning Networks (MPNet, [4]) approach is proposed to address the issue of computational complexity for high-dimensional configuration spaces. MPNet consists of two Neural Network models: Encoder Network (ENet) and Planner Network (PNet). ENet has three linear layers and an output layer. ENet takes vector of point clouds as an input and encodes them into a N-dimensional space. PNet consists of nine layers, it takes the current state, goal state of the robot and encoded space from ENet as an input and plan the next state for the robot that lies in collision-free space. To train the network 110 different environments are generated for each different problem type considered, and 5000 collision-free paths are computed in each environment using RRT*. The training dataset is generated using simulation.

A fully convolutional network is proposed in [5] that learns path planning by demonstration. Once training is performed, it is combined with RRT*. The integration of the predicted path with RRT* guarentees and optimal and feasible path. One-shot motion planning approach for multi-agents in 2D and 3D environments is proposed in [6], which is also based on convolutional neural networks. In this work, instead of generating paths iteratively, the complete path in one shot is given. The data is generated as a grid map or binary image, where 0 represents free space and 1 represents obstacle space. These grid maps are used to train the convolutional neural network.

All these approaches are required to have a dataset that is used for training. Usually the approaches implement one of the sampling-based motion planners and generate the data or some existing motion planning libraries are used. However, to generate data with multiple planners for comparison purposes, or to incorporate additional constrains such as physics-based constrains in the planning process, may require extra work or may not be possible. This study proposes a generalized dataset generation approach for learning-based motion planning, which allows to compute the sample data under geometric, kinodynamic or physics-based constrains. Moreover, it also allows to generate sample data for manipulation planning that includes a sequence of move, pick and place actions.

III. Proposed Approach

The proposed framework is depicted in Fig. 1. It consists of three main modules that are explained below.

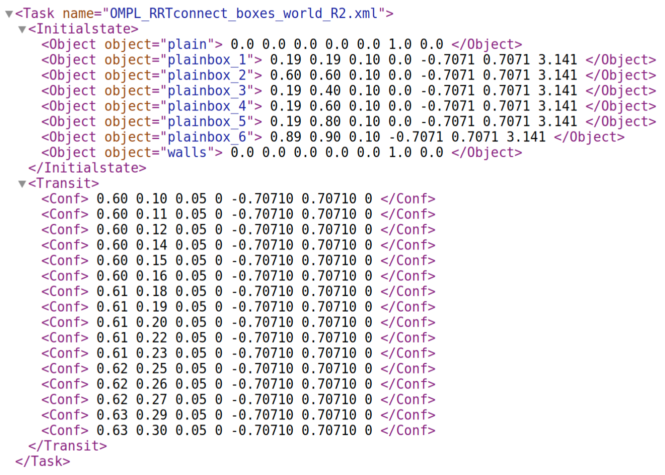
A. *The Kautham Project*

The Kautham Project [7] is a C++ based open source tool for motion planning (sir.upc.edu/projects/kautham/). It provides the ability to plan motions under geometric, kinodynamic and physics-based constraints. The Open Motion Planning Library (OMPL, [8]) is used for motion planning and physics simulations are handled using open dynamic engine (ODE). The Kautham Project also provides the flexibility to generate data for manipulation planning problems in which the task execution is explained via a sequence of actions such as move, pick and place [9].

The robots and the obstacles are modeled using URDF (wiki.ros.org/urdf), whereas the workspace is defined with a problem file with XML format. This file contains the position of the robot, the obstacles, the planner to be used along with its planning parameters and the query. Besides the GUI, The Kautham Project includes an interface implemented in ROS [10] that is used to communicate with other modules by means of services, and also a Python interface that wraps those ROS services, easing its integration in other projects.

Some of the utilities offered by Kautham as ROS services are:

- *OpenProblem / CloseProblem*
- *SetPlannerByName / SetQuery / GetPath*
- *SetRobotsConfig / GetObstaclePos / SetObstaclePos*
- *AttachObstacle2RobotLink / DetachObstacle*
- *SetRobControls*



```
▼<Task name="OMPL_RRTconnect_boxes_world_R2.xml">
  ▼<Initialstate>
     <Object object="plain"> 0.0 0.0 0.0 0.0 0.0 1.0 0.0 </Object>
     <Object object="plainbox_1"> 0.19 0.19 0.10 0.0 -0.7071 0.7071 3.141 </Object>
     <Object object="plainbox_2"> 0.60 0.60 0.10 0.0 -0.7071 0.7071 3.141 </Object>
     <Object object="plainbox_3"> 0.19 0.40 0.10 0.0 -0.7071 0.7071 3.141 </Object>
     <Object object="plainbox_4"> 0.19 0.60 0.10 0.0 -0.7071 0.7071 3.141 </Object>
     <Object object="plainbox_5"> 0.19 0.80 0.10 0.0 -0.7071 0.7071 3.141 </Object>
     <Object object="plainbox_6"> 0.89 0.90 0.10 -0.7071 0.7071 3.141 </Object>
     <Object object="walls"> 0.0 0.0 0.0 0.0 0.0 1.0 0.0 </Object>
   </Initialstate>
  ▼<Transit>
     <Conf> 0.60 0.10 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.60 0.11 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.60 0.12 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.60 0.14 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.60 0.15 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.60 0.16 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.61 0.18 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.61 0.19 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.61 0.20 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.61 0.22 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.61 0.23 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.62 0.25 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.62 0.26 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.62 0.27 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.63 0.29 0.05 0 -0.70710 0.70710 0 </Conf>
     <Conf> 0.63 0.30 0.05 0 -0.70710 0.70710 0 </Conf>
   </Transit>
 </Task>
```

Fig. 2. Kautham Taskfile.

B. Training Data Generator

The training dataset generator is a Python-based Kautham client which generates the dataset needed to train the neural network. This module randomly samples the obstacles positions in the envionment along with the collision-free start and goal states. This information is used to set the planning scene in Kautham via ROS. The computed motion planning path is then saved along with the environment details (such as obstacle configurations) in a *taskfile* with XML format, as that shown as an example in Fig. 2.

The main parameters that are required to generate training data are specified in a configuration file with *json* format. These parameters include the dimensions of the configuration space, the number of environments to be generated for the training data, the number of paths for each environment and the required data encoder format.

The data handler module is used to control the data generation process. It is responsible to invoke the required data encoder and processing back the data received from data decoder. The Kautham Data Encoder (KDE) is the main module that is responsible to encode the computed motion planning data into the required training dataset format. Different deep learning-based approaches for motion planning may follow slightly different data format. This module provides the flexibility to encode data into any arbitrary format by implementing the data encoder method. Kautham Data Decoder (KDD) is used to visualize the computed path by the neural network in Kautham or send to the real robotics platform. Similar to data encoder, the data decoder also provides the flexibility to implement the decoder method for decoding any arbitrary data format.

For the current demonstration we used *motion planning networks* that is one of the best deep learning-based approaches for motion planning. The functionality of the MPNet is summarized in the related work. KDE encodes the environment into obstacle cloud (*ObsC*). It contains the point cloud of the obstacles, the positions of the points in the cloud are defined with respect to the world frame. All the generated

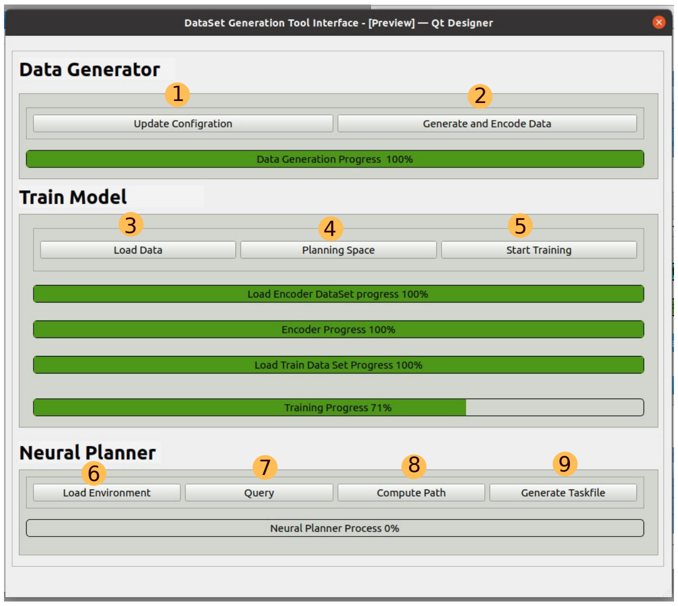


Fig. 3. User interface of training dataset generator tool.

environments are encoded using KDE and saved to the disk as training dataset. A set of valid paths $\{p_1^{e_i}, \ldots, p_n^{e_i}\}$ for each environments is also saved along with the obstacle cloud. A path $p_i^{e_i}$ consists of the set of robot configurations $\{q_1, \ldots, q_n\}$ valid for the particular environment e_i, represented with obstacle cloud $ObsC_i$. These files are later used by the encoder network of MPNet for training.

Once training is performed, the generated path by MPNet neural planner along with the environment (represented by obstacle cloud) can be decoded back to kautham *taskfile* using Kautham data decoder. The KDD will extract the positions of the obstacles from the obstacle cloud along with the path configurations (generated as an output of the MPNet). The data handler will save them into the *taskfile* with appropriate XML tags. The *taskfile* then can be used to visulize the scene in Kautham using Kautham-Gui or can be send to the real robotic platform.

C. User Interface

The user interface over the dataset generation framework of Fig. 1 is depicted in Fig. 3. Below is the description of its main functionalities for generating the datasets (steps 1 and 2), training the networks (steps 3 to 5) and playing the neural planner (steps 6 to 9).

1) *Update configuration* allows to modify the training configurations represented in *json* format.
2) *Generate and encode data* invokes the scene generator that will generate the environments and forward to data handler. The data handler will set the planning queries to generate the data and invoke the required data encoder.
3) *Load data* will select the location of the dataset.
4) *Planning space* is used to set the dimension of the workspace such as SE2, SE3 or R^n.

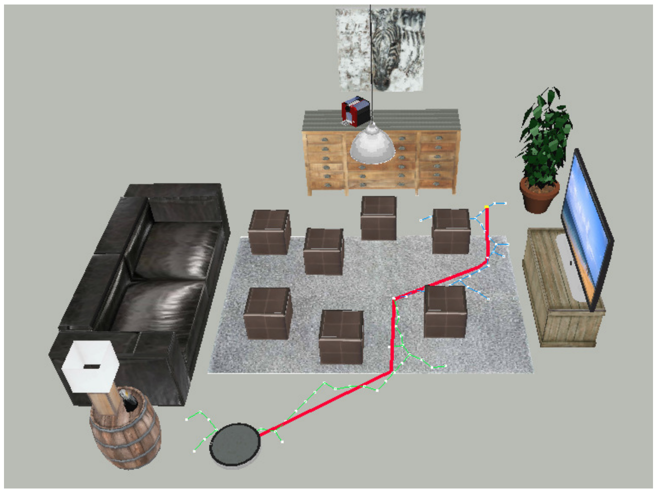


Fig. 4. Example scene for training dataset.

5) *Start training* will check the planning space and start training accordingly for the given data set. The progress of each step can be views visually on the progress bars.
6) *Load environment* sets the planning environment for the neural planner.
7) *Query* allows to set any arbitrary start and goal state for the neural planner.
8) *Compute path* will call the neural planner and solve the planning query.
9) *Generate taskfile* invokes the Kautham data decoder that will decode the path to visualize it in Kautham.

IV. Results and Discussions

The scenario selected as a test to generate training dataset consists of a cleaning robot moving in an environment that is obstructed with several floor seats, as shown in Fig. 4. We have generated three training datasets by changing the motion planners to RRT, RRT* and KPIECE. The training dataset consists of 140 environments where 1000 paths with random start and goal states were generated for each (examples of some randomly generated environments are shown in Fig. 5). MPNet is trained for each dataset. Once training is performed, the same scenes are provided as input to the three trained models and used to generate output paths available for comparison purposes. The generate datasets, trained weights and the code are avalible online[1].

The reason to generate three datasets is to highlight the flexibility and effectiveness of the proposed training data generation tool. In this work-in-progress a simple demonstration is performed by changing different geometric planners. Similarly, the dataset could also have been generated using the planners that incorporate kinodyanmic or physics-based constraints.

V. Conclusions and Future Work

This work-in-progress proposes a generalized dataset generation tool for deep learning-based motion planning in complex

[1]https://gitioc.upc.edu/muhayyuddin.gillani/mp_dataset_generation.git

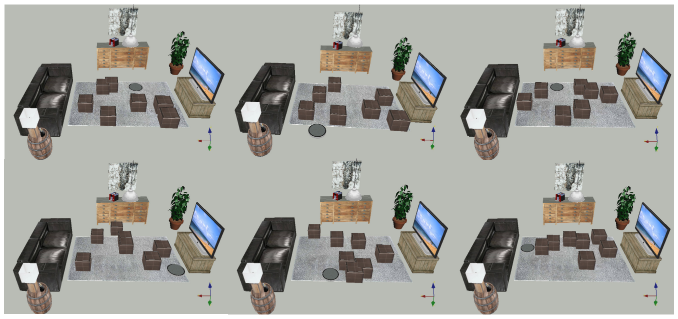


Fig. 5. Example of some random environments generated by scene generator.

environments. The generated data could easily be encoded in any required format for providing input to the neural network and decoded back to visualize with the Kautham GUI.

This approach will further be used to capture datasets for mobile manipulators under various kinodynamic and physics-based constraints. These datasets will be used to train deep learning based approaches for motion planning to investigate the effect of kinodynamic and physics-based constraints over the learning.

Moreover the approach will be enhanced to generate the datasets for task planning, that will consists of sequences of pick and place actions.

References

[1] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[2] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robotics Research*, vol. 30, no. 7, pp. 846–894, Junio 2011.

[4] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[5] N. Pérez-Higueras, F. Caballero, and L. Merino, "Learning human-aware path planning with fully convolutional networks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5897–5902.

[6] T. Kulvicius, S. Herzog, T. Lüddecke, M. Tamosiunaite, and F. Wörgötter, "One-shot multi-path planning for robotic applications using fully convolutional networks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1460–1466.

[7] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García, "The Kautham Project: A teaching and research tool for robot motion planning," in *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2014.

[8] I. Sucan, M. Moll, L. E. Kavraki *et al.*, "The open motion planning library," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 72–82, 2012.

[9] S. Saoji and J. Rosell, "Flexibly configuring task and motion planning problems for mobile manipulators*," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1285–1288.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, 2009.