Paper:

Trajectory Prediction with a Conditional Variational Autoencoder

Thibault Barbié, Takaki Nishio, and Takeshi Nishida

Department of Mechanical and Control Engineering, Kyushu Institute of Technology
1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804-8550, Japan
E-mail: q595203a@mail.kyutech.jp

[Received December 5, 2018; accepted April 15, 2019]

Conventional motion planners do not rely on previous experience when presented with a new problem. Trajectory prediction algorithms solve this problem using a pre-existing dataset at runtime. We propose instead using a conditional variational autoencoder (CVAE) to learn the distribution of the motion dataset and hence to generate trajectories for use as priors within the traditional motion planning approaches. We demonstrate, through simulations and by using an industrial robot arm with six degrees of freedom, that our trajectory prediction algorithm generates more collision-free trajectories compared to the linear initialization, and reduces the computation time of optimization-based planners.

Keywords: trajectory prediction, machine learning, conditional variational autoencoder

1. Introduction

Motion planning is an essential tool for robots that allows them to move safely in their environment while avoiding obstacles. Most motion planners are solving problems by either using random samples of the configuration space [1–3] or by iteratively improving an initial trajectory (usually a linear trajectory through configuration space) [4–6]. However, even if the robot has encountered the same or a similar situation in the past the motion planners have to recalculate a trajectory from scratch. Furthermore, motion planners are difficult to scale when dealing with high-dimensional problems.

Trajectory prediction seeks to address this challenge by predicting a trajectory given a motion planning problem. The predicted trajectory is then used either as an initial guess or directly as the final trajectory. Starting with an informed prior, this approach reduces the domain to search, which decreases the high dimensional burden. Trajectory prediction algorithms use a motion dataset containing information about previously encountered motion planning problems and the corresponding computed trajectories. When a new planning problem is given, the trajectory prediction algorithms use the dataset to predict a trajectory solution. However, even if the existing methods can provide satisfactory results, the reliance on the dataset at runtime remains problematic. The requirement for substantial memory space and computational cost, both of which increase with the size of the stored dataset, constitute a significant limitation in the case of small robots. As robot technology is evolving toward more interconnectivity and the sharing of experience data between robot [7], it will become increasingly difficult for extremely large datasets to be handled at runtime.

Machine learning has received considerable attention and has achieved notable success over the past decade in various fields, such as object detection [8, 9] or language learning [10]. However, its application to motion planning is yet to achieve similar success. There is currently much interest in conditional variational autoencoders (CVAEs) [11] within the machine learning community for their ability to model the distribution of data in a dataset, and then generating new samples with a high likelihood.

Rather than completely replacing classical motion planners, it can be argued that machine learning can serve as a useful cooperation tool. The present study exploits the trajectory prediction framework of Barbié et al. [12] in conjunction with the CVAE algorithm. The CVAE learns the distribution of the motion dataset and uses it to generate trajectories when confronted with new planning problems. The generated trajectories are not intended to be used directly but rather as initial guesses by a classical motion planner. Our contribution is twofold. Firstly, we provide a fast trajectory prediction algorithm that does not require a dataset at runtime and that works directly in the trajectory space. Secondly, we propose to use an autoencoder to allow working directly in the trajectory space instead of the configuration space.

2. Related Work

Using previous experience to solve novel motion planning problems is not a new idea. Jetchev and Toussaint [13] proposed an algorithm that predicts the index of a trajectory in a precomputed dataset and adapts the trajectory to the new problem. Berenson et al. [14] presented a framework for storing and retrieving trajectories from a dataset while adapting them to new problems.

Hauser [15] proposed a framework for retrieving trajectories from large datasets using a clustering algorithm. While these methods are effective, they require access to the dataset at runtime, a feature that our proposed algorithm seeks to avoid.

The "learning from demonstration" (LfD) paradigm provides an alternative approach to trajectory prediction. A robot first learns from a teacher demonstration how to complete a task and then tries to adapt this acquired knowledge to other situations. Many methods have been proposed to implement this principle [16]. Recently, Duan et al. [17] proposed a one-shot imitation framework, in which a robot learns to repeat a task after a single demonstration. In contrast, our method does not consider the notion of task, and instead involves learning based on big datasets generated entirely by simulations.

Machine learning has already been applied to motion planning. Qureshi et al. [18] created a motion planning network that yielded notable results but their network is a motion planner where we aim to only provide an initial guess. The method adopted by Ichter et al. [19] is more similar to our approach but operates in the configuration space, whereas ours operates directly in the trajectory space.

In our previous work [12], we used a multivariate Gaussian distribution to generate the distribution of the motion dataset. This method was capable of generating trajectories without requiring a dataset at runtime, but was limited by the computation speed required to generate a trajectory. On the other hand, our current method, based on CVAEs, involves only matrix multiplication and vector addition, which are amenable to fast computation using existing matrix computation libraries.

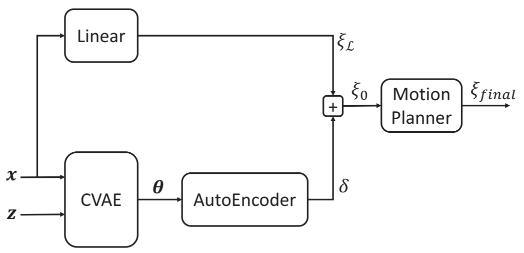

Fig. 1. Overview of the trajectory generation process. A random vector $\boldsymbol{z}$ and a motion planning problem $\boldsymbol{x}$ are given as inputs to the CVAE which then outputs a vector $\boldsymbol{\theta}$ that is then decoded as perturbation trajectory δ. The "Linear" box indicates the creation of a linear trajectory $\xi_{\mathscr{L}}$ from the starting point to the end point specified in the motion planning problem $\boldsymbol{x}$. The predicted trajectory ξ_0 is the sum of the linear trajectory $\xi_{\mathscr{L}}$ and the perturbation trajectory δ: $\xi = \xi_{\mathscr{L}} + \delta$. Finally, the predicted trajectory is used as a prior by a motion planner to compute the final trajectory ξ_{final}.

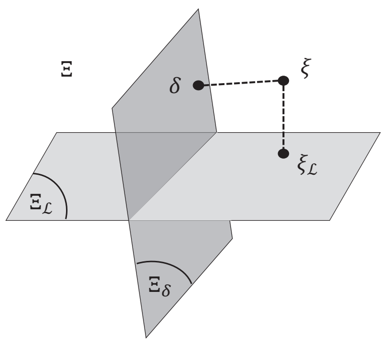

Fig. 2. Decomposition of the trajectory space Ξ into the direct sum of the linear trajectories space $\Xi_{\mathscr{L}}$ and the perturbation trajectories space Ξ_δ : $\Xi = \Xi_{\mathscr{L}} \oplus \Xi_\delta$. Because the linear trajectory component $\xi_{\mathscr{L}}$ is known entirely from the specification of the motion planning problem, only the perturbation component δ needs to be generated.

3. Method

Our proposed method involves using a CVAE to learn how to generate a trajectory conditioned on a given motion planning problem. However, a trajectory ξ is a continuous function mapping time $t \in [0,1]$ to configuration space points $q \in \mathscr{C}$. Unfortunately, the trajectory space Ξ of continuous functions from $[0,1]$ to $\mathscr{C}$ is infinite dimensional whereas the CVAE has a finite dimensional output. To make the problem tractable, we sought to project the trajectory space onto a finite dimensional manifold using an autoencoder [20]. This transformed the problem to one where the CVAE mapped the motion planning problem space X to the perturbation trajectories manifold $\mathscr{M}_\delta$ (see **Fig. 1**).

3.1. Trajectory Manifold

It was first necessary to project the infinite space of trajectories Ξ onto a finite dimensional space. This task was facilitated by a simple observation on such trajectories. Because the starting point and end point in the configuration space are specified within the motion planning problem, the only thing the CVAE has to generate is the path connecting these two points independently of their respective positions. Hence, a given trajectory $\xi \in \Xi$ can be decomposed into a sum of two trajectories $\xi = \xi_{\mathscr{L}} + \delta$ where $\xi_{\mathscr{L}}$ is a linear trajectory through the configuration space (which is known exactly from the motion planning problem specification) and δ is a perturbation trajectory representing the deviation of ξ from $\xi_{\mathscr{L}}$.

$$\xi_{\mathscr{L}}: [0;1] \to \mathscr{C} \qquad t \mapsto (1-t)q_{\text{start}} + tq_{\text{goal}} \quad . \quad . \quad (1)$$

$$\delta = \xi - \xi_{\mathscr{L}} \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (2)$$

This in turns leads to a decomposition of Ξ into a direct sum of two trajectory subspaces, i.e., $\Xi = \Xi_{\mathscr{L}} \oplus \Xi_\delta$, with $\Xi_{\mathscr{L}} = \{\xi_{\mathscr{L}} \in \Xi \mid \exists q_1, q_2 \in \mathscr{C}\ \xi_{\mathscr{L}}(t) = (1-t)q_1 + tq_2\}$ the space of linear trajectories, and $\Xi_\delta = \{\delta \in \Xi \mid \delta(0) = \delta(1) = 0\}$, the space of perturbation trajectories (see **Fig. 2**).

We then hypothesized that the subspace Ξ_δ is a manifold with few dimensions, $\mathscr{M}_\delta$. Using an autoencoder, a perturbation trajectory $\delta \in \Xi_\delta$ can be projected onto a finite dimensional vector $\boldsymbol{\theta} \in \mathscr{M}_\delta$. Because the linear part of ξ is already known, every vector $\boldsymbol{\theta} \in \mathscr{M}_\delta$ represents a full trajectory $\xi \in \Xi$.

3.2. Conditional Generation of Trajectories

The above decomposition reduces the problem to finding a mapping between X (the space of planning problems) and $\mathscr{M}_\delta$ (the manifold of perturbation trajectories). To solve this problem in practice, we used the CVAE to learn how to map X onto $\mathscr{M}_\delta$. The CVAE must train on a dataset $\mathscr{D} = \{\boldsymbol{x}_i, \boldsymbol{\theta}_i\}_{i=1,\dots,N}$ containing N motion planning problems and their associated trajectory solution projected onto $\mathscr{M}_\delta$.

A CVAE consist of a generative model $p_\psi(\boldsymbol{\theta}, \boldsymbol{z}|\boldsymbol{x}) = p_\psi(\boldsymbol{\theta}|\boldsymbol{z},\boldsymbol{x}) p_\psi(\boldsymbol{z}|\boldsymbol{x})$ and an inference model $q_\phi(\boldsymbol{z}|\boldsymbol{\theta},\boldsymbol{x})$. The vector $\boldsymbol{z}$ is a random vector sampled from a normal distribution $\mathscr{N}(\boldsymbol{0}, \boldsymbol{I}_K)$, with K representing the dimension of the noise vector. When a new motion planning problem $\boldsymbol{x}$ is specified, the CVAE aims to generate a vector $\boldsymbol{\theta}$ with a high likelihood. However, the direct optimization of the likelihood is not a tractable problem. In Kingma and Welling paper [11] a lower bound $\mathscr{L}$ of the likelihood $p_\psi(\boldsymbol{\theta}|\boldsymbol{x})$ is derived and maximized to optimize each models.

$$\log p_\psi(\boldsymbol{\theta}) \geq E_{q_\phi(\boldsymbol{z}|\boldsymbol{\theta},\boldsymbol{x})}\left[\frac{p_\psi(\boldsymbol{\theta},\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{\theta},\boldsymbol{x})}\right] = \mathscr{L}(\psi,\phi;\boldsymbol{\theta}) \quad (3)$$

After undergoing the training procedure, the CVAE can generate new samples $\boldsymbol{\theta}_{gen}$ from a random vector $\boldsymbol{z}$ and a motion planning problem $\boldsymbol{x}$ using only its generative model.

4. Experiments

We performed three experiments to assess the performance of our method. First, we used a simulated problem in a two-dimensional configuration space and projected the trajectories into manifolds of different dimensions to observe the loss incurred by the autoencoder reconstruction. Second, in the same environment, we compared the number of collision-free trajectories between those generated by the linear initialization and the one from the CVAE. We call "linear initialization" the standard initialization for optimization based motion planners consisting of using the linear trajectory between the starting and end points in the configuration space. The third experiment used a real-life industrial problem involving a robot arm and showed that our method accelerated the process of trajectory generation and increased its success rate. The autoencoder and the CVAE were both trained using the Chainer framework, version 4.3.1.

4.1. Manifold Projection

In order to work directly in the trajectory space, an autoencoder was used to project Ξ_δ onto $\mathscr{M}_\delta$. The smaller the dimension of the manifold is, the easier it would be for

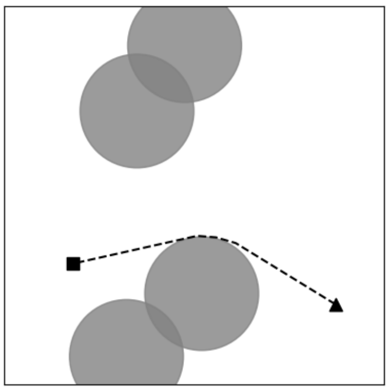

Fig. 3. Example of a planning problem and its solution from the dataset used for the first experiment. Four obstacles of equal radius (gray circles) are uniformly randomly placed. The positions of the starting point (black triangle) and the end point (black square) positions were sampled uniformly randomly (outside of the obstacles). The dashed line represents the computed trajectory solution.

the CVAE to learn. However, on the other hand, it is more difficult for the autoencoder to find a representation that approximate correctly the perturbation trajectories. There is thus an important trade-off between the accuracy of the trajectory representation and the CVAE ease of learning. We want our representation to be as expressive as possible (to approximate all trajectories and generalize well) while keeping it small to accelerate the CVAE training. The aim of this experiment was therefore to project the trajectories onto manifolds of different dimensions and to assess the reconstruction accuracy.

The motion planning problem considered was set in a two-dimensional configuration space that contained four uniformly randomly placed circular obstacles, with the starting point and end point also uniformly randomly positioned outside the obstacles, as depicted **Fig. 3**. The dataset was constituted by using the rapidly-exploring random trees motion planner (RRT) from the OMPL library. When training the autoencoder to perform the projection of Ξ_δ onto $\mathscr{M}_\xi$, it was important to provide the autoencoder with many different examples of possible trajectories. In practice, however, most trajectory solutions to randomly generated motion planning problems are simple linear trajectories in the configuration space (i.e, equivalent to a perturbation $\delta = 0$). To avoid this situation, we biased the dataset by retaining only 1% of these linear trajectories. The dataset was thus constituted of 500,000 problems and their corresponding trajectory solutions.

All the autoencoders we used had the same architecture that differed only in their output layer, for which the number of neurons corresponded to the number of dimension of the manifold $|\mathscr{M}_\delta|$ (see **Fig. 4**). The Adam optimizer [21] was used with $\alpha = 0.001$ and $\beta = 0.9$. We compared the reconstruction loss of the different autoencoders as the dimension of the manifold varied from 1 to 10 (see **Figs. 5** and **6**). There is no substantial improve-

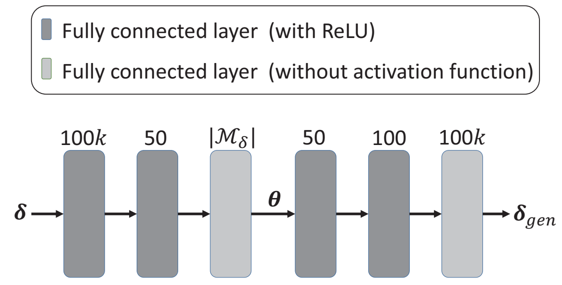
Fully connected layer (with ReLU)
Fully connected layer (without activation function)
100k
50
$|\mathcal{M}_\delta|$
50
100
100k
δ
θ
δ_{gen}

Fig. 4. Overview of the autoencoder architecture. A perturbation trajectory δ is projected onto a small vector $\boldsymbol{\theta}$ and then reconstructed as $\delta_{\text{reconstructed}}$. $|\mathscr{M}_\delta|$ denotes the dimension of the perturbation trajectories manifold $\mathscr{M}_\delta$. The numbers above the layers indicate the number of neurons. The last layer outputs a perturbation trajectory δ_{gen} of 100 waypoints, for a robot with k degrees of freedom.

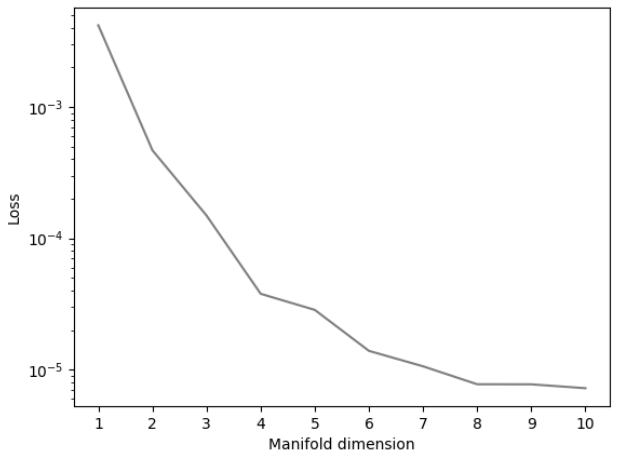

Fig. 6. Reconstruction loss of the autoencoder as a function of the manifold dimension. The reconstruction loss decreases with the increasing dimensionality.

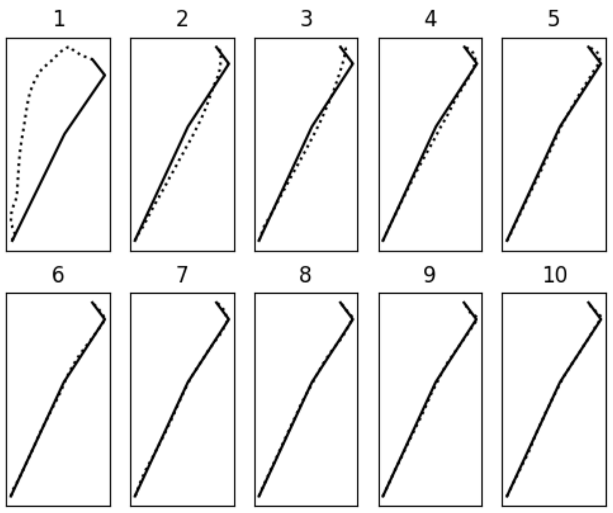

Fig. 5. Projection of a trajectory onto low dimensional manifolds by an autoencoder. The reference trajectories (solid lines) are approximated by their projections onto $\mathscr{M}_\delta$ (dashed lines). The different figures show the projection onto manifold of different dimensions. By increasing the number of dimension of the manifold the autoencoder can approximate the trajectory more easily.

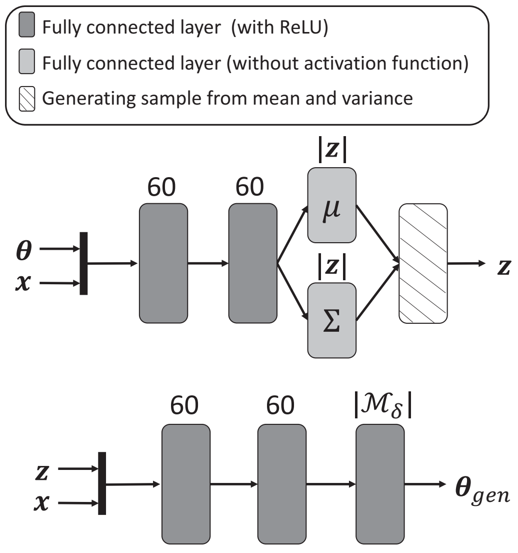
Fully connected layer (with ReLU)
Fully connected layer (without activation function)
Generating sample from mean and variance
60
60
$|z|$
μ
$|z|$
Σ
θ
x
z
60
60
$|\mathcal{M}_\delta|$
z
x
θ_{gen}

Fig. 7. Overview of the CVAE architecture. The upper and lower models perform the encoding and decoding tasks, respectively. The blocks labeled μ and Σ block denote the mean and variance of the distribution from which the random vector $\boldsymbol{z}$ will be sampled. $|\mathscr{M}_\delta|$ denotes the dimension of the manifold $\mathscr{M}_\delta$. The numbers written above the layers indicate the number of neurons.

ment in the reconstruction when the dimension increases beyond 7. We therefore reduced the dimension from 200 (two-dimensional configuration space and 100 waypoints) to 7. This significantly facilitated the task of the CVAE.

4.2. Collision-Free Trajectories

The second experiment aimed to compare how many collision-free trajectories were generated by the CVAE compared to the case of a linear initialization traditionally used by optimization based motion planners. For this purpose, we defined a simple metric:

$$f(\xi) = \begin{cases} 1 & \text{if } \xi \text{ is colliding with an obstacle} \\ 0 & \text{otherwise} \end{cases}. \quad (4)$$

This experiment used the same setting as the previous one. It should be noted that the RRT planner usually finds trajectories with few clearance to minimize the length of the trajectory. However, because the CVAE is generating trajectories approximatively, it was preferable to compute a dataset with trajectory examples with better clearance. To simulate this, we increased the obstacle size in the dataset relative to those used for the test set.

The CVAE architecture consisted entirely of fully connected layers (see **Fig. 7**). The perturbation trajectory manifold dimension was set to $|\mathscr{M}_\delta| = 7$ according to the previous result. The Adam optimizer was used with

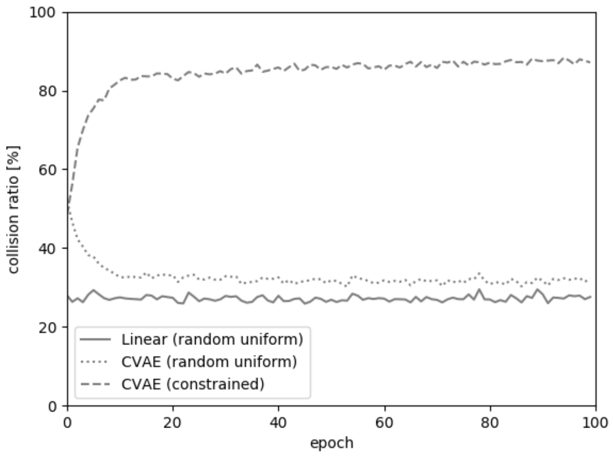

Fig. 8. Collision ratio of the linear and CVAE initializations with respect to the epoch for the random uniform problem and the constrained problem. The linear initialization always collided in the constrained motion planning problems.

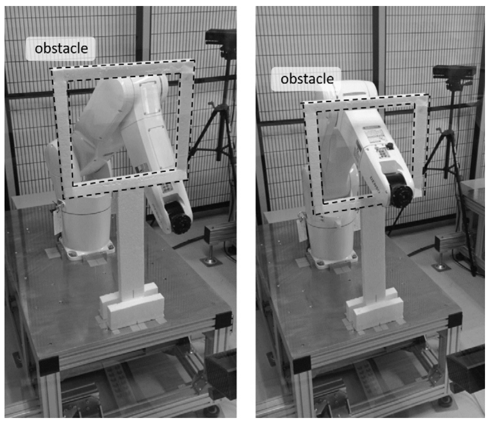

Fig. 9. Motion planning problem with an industrial arm. Left: the robot starts from a randomly chosen position below the obstacle (on the bottom-right side in the example shown). Right: the target position is inside the obstacle. This is a difficult problem for a motion planner to solve, owing to the tunnel-like shape of this obstacle.

$\alpha = 0.01$ and $\beta = 0.5$. We set the ratio coefficient between the reconstruction loss and the KL divergence loss to $\lambda = 0.01$ and the noise dimension to $K = 7$.

For each epoch during the CVAE training, we performed two tests on the model. First, we randomly created 10,000 problems (with the same distribution as that of the dataset) and measured the collision ratio of the linear initializations and the CVAE initializations. Second, we created 10,000 problems where the obstacles were randomly placed within the center of the space with the starting and end points placed in opposite position (with central symmetry). We performed these two tests because, in practice, most problems were easily solved by a straight line trajectory (for which the linear initialization was perfect). To demonstrate the superior performance of our method compared to the linear case, more complex scenarios were required.

The results are shown **Fig. 8**. In the case of the first test, our method performed only slightly worse than the linear case. This shows that, on average, the CVAE is almost as good as the linear initialization when the scenario requires a simple solution. However, the second test clearly demonstrates the superior performance achieved when using the CVAE, compared to the case of a linear initialization, when tackling a more complex scenario. This suggests that our method is superior to the linear initialization for generating collision-free trajectories.

4.3. Industrial Robot Arm

The last experiment employed the VS087 industrial robot arm with six degrees of freedom. It was important to demonstrate that our trajectory prediction algorithm was efficient when applied to a real life problem, and that it can be easily integrated. The experiment consisted of making the robot traverse a "tunnel-like obstacle" (see **Fig. 9**), starting from a randomly chosen position below the obstacle. The purpose was to test the performance of our method using a scenario considered to be difficult for classical motion planners.

We conducted the experiment using the software ROS (version kinetic) with the *Moveit!* and *industrial moveit* packages. We used the RRT planner to create the dataset, but used the CHOMP [4] and the STOMP planners [5] instead for testing. The RRT planner was required to cope with the slow creation of the dataset, as the CHOMP and STOMP planners had a low success rate when solving this problem. We generated 60,000 examples for our dataset. We used the same CVAE and autoencoder settings as in the previous experiment, except that the noise dimension of the CVAE was set to $K = 12$ and the perturbation trajectory manifold dimension was set to $|\mathscr{M}_{\delta}| = 10$. The change was due to the increase of dimensionality of the problem. For both the linear initialization and the CVAE initialization, 1,000 tests were carried out. For each test, the computation time and the number of iterations of the STOMP planner and the CHOMP planner were recorded for each solution found. The planner settings were unchanged except to make them accept trajectories with 100 waypoints (to conform with our autoencoder architecture).

The success rate of the STOMP planner for this problem was 36.1%, when using the linear initialization. When the CVAE was used to initialize the planner, the success rate increased to 43.1%. The CHOMP planner was unable to find a solution when using the linear initialization. However, when using the CVAE initialization, its success rate was 56.7%. We also observed that solutions reached successfully with the STOMP planner required less computation time (see **Fig. 10**), owing to the lower number of iterations necessary for optimizing the initial trajectory (see **Fig. 11**). It took 0.13 ms to generate one trajectory on average, which is negligible compared to the STOMP and CHOMP optimizations. This indicates that our trajectory prediction algorithm has almost

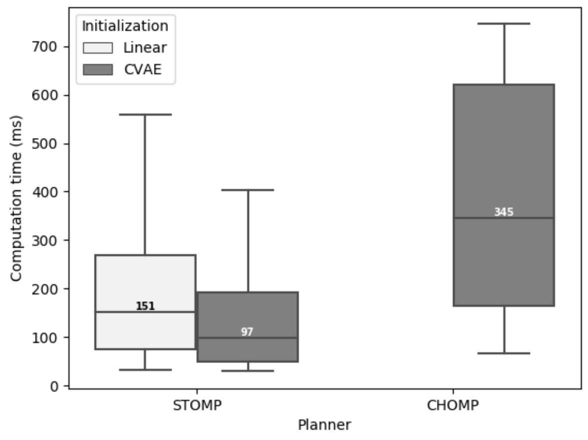

Fig. 10. Boxplot of the computation time for 5,000 tests conducted by the STOMP planner with a linear initialization or a CVAE initializations. The median of each boxplot is written above the central horizontal line. The computation time for optimizing the final trajectory is shorter when using the CVAE initialization than the linear initialization.

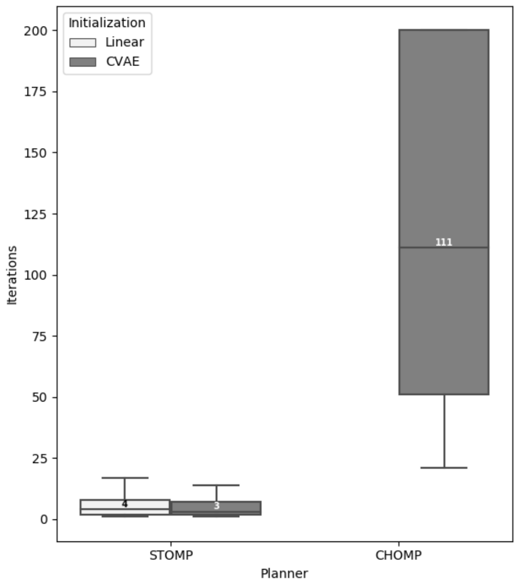

Fig. 11. Boxplot of the number of iterations of the STOMP planner over 5,000 tests, conducted with a linear initialization or a CVAE initialization. The median of each boxplot is written above the central horizontal line. A CVAE initialization requires fewer iterations than a linear initialization to yield optimized final trajectories.

no overhead compared to the linear initialization and can thus significantly boost the performance of the trajectory generation.

5. Conclusions

We proposed using an autoencoder and a CVAE to perform trajectory predictions. The CVAE initially learns the distribution of a previously computed motion dataset and can then generate trajectory solutions for motion planning problems that are similar but not identical to those already featuring in the dataset. We demonstrate that this method can generate initial trajectories that significantly reduce the computation time and increase the success rate of optimization based planners compared to those of more traditional motion planners that use a linear initialization. Because we are using a learning-based approach, a limitation of the method is when the dataset lacks data. However, the CVAE is able to extract important features from the dataset, which mitigates the problem. A future development of our work will contain the analysis of these extracted features.

We plan to extend the application of the method to multiple robots and different scenarios. We also aim to assess its performances when it is applied to a new set of motion planning problems (transfer learning). A further extension of this work will involve a robot continuously gathering information and learning from data while moving. Finally, because motion planning is a very broad field, our framework could be extended to autonomous cars navigation problems and mobile robots easily. Moving obstacles could also be handled which we plan to study in future works.

References:

[1] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," 1998.

[2] J. J. Kuffner and S. M. LaValle, "Rrt-connect: an efficient approach to single-query path planning," Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA'00), Vol.2, pp. 995-1001, 2000.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The Int. J. of Robotics Research, Vol.30, No.7, pp. 846-894, 2011.

[4] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning," Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA'09), pp. 489-494, 2009.

[5] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 4569-4574, 2011.

[6] Z. Marinho, B. Boots, A. Dragan, A. Byravan, G. J. Gordon, and S. Srinivasa, "Functional gradient motion planning in reproducing kernel hilbert spaces," Proc. of Robotics: Science and Systems, Ann Arbor, Michigan, June 2016.

[7] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," IEEE Trans. Automation Science and Engineering, Vol.12, No.2, pp. 398-409, 2015.

[8] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, pp. 7263-7271, 2017.

[9] Y. Konishi, K. Shigematsu, T. Tsubouchi, and A. Ohya, "Detection of target persons using deep learning and training data generation for Tsukuba challenge," J. Robot. Mechatron., Vol.30, No.4, pp. 513-522, 2018.

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Advances in Neural Information Processing Systems, pp. 3111-3119, 2013.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," arXiv:1312.6114, 2013.

[12] T. Barbié, R. Kabutan, R. Tanaka, and T. Nishida, "Gaussian mixture spline trajectory: learning from a dataset, generating trajectories without one," Advanced Robotics, Vol.32, No.10, pp. 547-558, 2018.

[13] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," Autonomous Robots, Vol.34, Nos.1-2, pp. 111-127, 2013.

[14] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," 2012 IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 3671-3678, 2012.

[15] K. Hauser, "Large Motion Libraries: Toward a "Google" for Robot Motions," Proc. of Robotics Challenges and Vision (RCV2013), 2014.

[16] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," Robotics and Autonomous Systems, Vol.57, No.5, pp. 469-483, 2009.

[17] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," Advances in Neural Information Processing Systems (NIPS 2017), pp. 1087-1098, 2017.

[18] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion planning networks," arXiv:1806.05767, 2018.

[19] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," 2018 IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 7087-7094, 2018.

[20] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, Vol.313, No.5786, pp. 504-507, 2006.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980, 2014.

Name:
Thibault Barbié

Affiliation:
Department of Mechanical and Control Engineering, Kyushu Institute of Technology

Address:
1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804-8550, Japan

Brief Biographical History:
2016 Received M.Sc. in Aerospace Engineering from ISAE-Supaero
2016 Received M.Sc. in Artificial Intelligence from Université Paul Sabatier
2016- Doctoral Student, Kyushu Institute of Technology

Main Works:
• motion planning, trajectory prediction, and machine learning

Name:
Takaki Nishio

Affiliation:
Department of Mechanical and Control Engineering, Kyushu Institute of Technology

Address:
1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804-8550, Japan

Brief Biographical History:
2015 Graduated from National Institute of Technology, Maizuru College
2017-2019 Master Course Student, Kyushu Institute of Technology

Main Works:
• convolutional neural networks, robotic control by machine learning, and teaching systems for industrial robots

Name:
Takeshi Nishida

Affiliation:
Department of Mechanical and Control Engineering, Kyushu Institute of Technology

Address:
1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804-8550, Japan

Brief Biographical History:
1999 Received M.Sc. from Kyushu Institute of Technology
2002 Received Ph.D. from Kyushu Institute of Technology
2002-2012 Assistant Professor, Kyushu Institute of Technology
2013- Associate Professor, Kyushu Institute of Technology

Main Works:
• control engineering, artificial intelligence, and robotics

Membership in Academic Societies:
• The Society of Instruments and Control Engineering (SICE)
• The Robotics Society of Japan (RSJ)