Computational Creativity Via Assisted Variational Synthesis of Mechanisms Using Deep Generative Models

Shrinath Deshpande
Computer-Aided Design and Innovation Lab,
Department of Mechanical Engineering,
Stony Brook University,
Stony Brook, NY 11794-2300
e-mail: shrinath.deshpande@stonybrook.edu

Anurag Purwar[1]
Computer-Aided Design and Innovation Lab,
Department of Mechanical Engineering,
Stony Brook University,
Stony Brook, NY 11794-2300
e-mail: anurag.purwar@stonybrook.edu

Computational methods for kinematic synthesis of mechanisms for motion generation problems require input in the form of precision positions. Given the highly nonlinear nature of the problem, solutions to these methods are overly sensitive to the input—a small perturbation to even a single position of a given motion can change the topology and dimensions of the synthesized mechanisms drastically. Thus, the synthesis becomes a blind iterative process of maneuvering precision positions in the hope of finding good solutions. In this paper, we present a deep-learning-based framework which manages the uncertain user input and provides the user with a higher level control of the design process. The framework also imputes the input with missing information required by the computational algorithms. The approach starts by learning the probability distribution of possible linkage parameters with a deep generative modeling technique, called variational auto encoder (VAE). This facilitates capturing salient features of the user input and relating them with possible linkage parameters. Then, input samples resembling the inferred salient features are generated and fed to the computational methods of kinematic synthesis. The framework postprocesses the solutions and presents the concepts to the user along with a handle to visualize the variants of each concept. We define this approach as variational synthesis of mechanisms. *In addition, we also present an alternate end-to-end deep neural network architecture for variational synthesis of linkages. This end-to-end architecture is a conditional-VAE, which approximates the conditional distribution of linkage parameters with respect to a coupler trajectory distribution. The outcome is a probability distribution of kinematic linkages for an unknown coupler path or motion. This framework functions as a bridge between the current state of the art theoretical and computational kinematic methods and machine learning to enable designers to create practical mechanism design solutions.* [DOI: 10.1115/1.4044396]

Keywords: human machine collaboration, deep generative models, path synthesis, motion synthesis, planar linkage synthesis, machine learning (ML) for user experience, ML for managing uncertainties, ML for computational creativity, deep learning, computational kinematics, conceptual design, creativity and concept generation, mechanism synthesis

1 Introduction

A mechanism is defined to be a collection of rigid bodies connected together by joints such as hinges or sliders in order to generate articulated motions. Kinematic synthesis of the mechanism deals with computing type and dimension of mechanisms that are useful for a particular task. Depending upon the task, synthesis problems have mainly been divided into three categories: (1) function generation, (2) path generation, and (3) motion generation. The function generation task demands a prescribed relationship between the rotation of input and output links of the mechanism. In the path generation task, the aim is to move an object through space along a prescribed path, whereas in motion generation, a rigid body needs to be guided along a prescribed motion. Hereinafter, motion is termed as a continuous sequence of poses, where a pose is a combination of position and orientation. The theory of mechanism synthesis has witnessed vast research in four decades, resulting in various methods for solving the above problems; see Refs. [1–6]. Methods for mechanism synthesis start with taking input in the form of a sequence of path-points, poses, or functional input–output relationship from the user. A large majority of motion and path generation methods take the precision point/pose approach, which optimally minimizes the least squared fitting error between precision points/pose and coupler curve/motion. This approach is highly susceptible to input precision points/poses and in most of the cases results in solutions with circuit or branch defects [7]. The root cause of sensitivity is in the underlying nature of the interpolation problem and highly nonlinear relationship between the input and the output.

To illustrate this chaotic nature, let us consider the following example. Six poses are sampled from a four-bar linkage as shown in the left of Fig. 1. As the poses are already known to lie on the coupler motion, our motion generator algorithm [8,9] obtains the original four-bar as expected. However, even a small change in the orientation of a pose results in an entirely different linkage suffering from branch defect, therefore making it unsuitable to perform the function. Moreover, increasing the number of positions by means of a B-spline interpolation through original poses does not help as shown in the right-most of Fig. 1. It is important to note that the algorithm [8] used for synthesis in the example is a representative of the methods based on the precision point approach. In the real world, when a user inputs a sequence and receives a result not suitable for the application, no informed decisions can be made to rectify the situation. What adds more uncertainty to the problem is that the task is often an approximation of the designer's intended motion. To accommodate this uncertainty, a common approach is to randomly perturb some of the poses within a given

[1]Corresponding author.

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received March 1, 2019; final manuscript received July 19, 2019; published online August 1, 2019. Assoc. Editor: Xiaoping Qian.

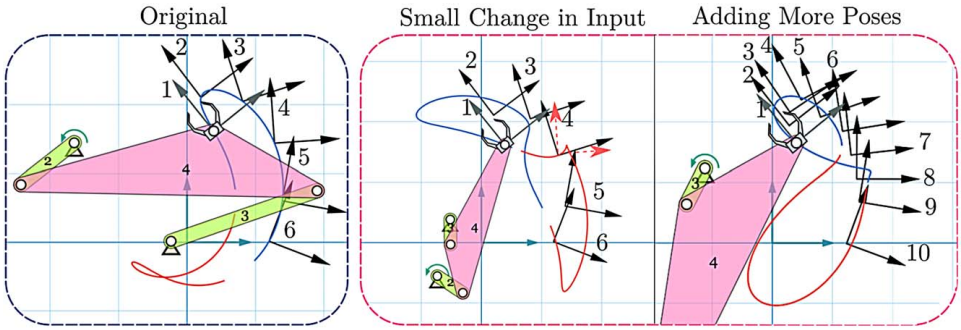

Fig. 1 Slightly different inputs result in entirely different mechanisms. The input on the left generates a defect-free four-bar linkage. The input in the middle is formed by perturbing the orientation of the fourth pose by 15 deg. The dashed frame in the middle input represents the original unaltered pose. Increasing the number of poses does not help either as shown by the mechanism generated on the right.

tolerance. The probability of a random perturbation to find a valid input reduces exponentially with the number of precision positions as well as the tolerance range.

To perform informed and meaningful input modification, it is necessary to possess knowledge about the properties of coupler motions (or paths) of planar linkages. Let us define this knowledge as a *prior probability distribution*. In Bayesian statistical inference, a prior probability distribution, often simply called the *prior*, of an uncertain quantity is the probability distribution that would express one's beliefs about this quantity before some evidence is taken into account.

While the sensitivity to the input has been a problem to finding good solutions, it turns out that we can exploit the susceptibility of synthesis algorithms by providing them with a variety of preconditioned inputs so as to find a *diverse* range of defect-free solutions. By providing tools that can provide the designer with higher level control on input specification, we can help them be more creative as well.

This paper presents such a framework, which acts as an intermediary between designer and computational solvers. The framework captures the salient features in the task given by the user and computes a variety of conditioned inputs for computational solvers. The inputs are conditioned so that the likelihood of getting a good solution is maximized. In addition to the conditioning, the framework also incorporates missing information required by computational solvers. Lastly, the outputs are postprocessed and the designer is presented with a set of distributions of solutions, where each set consists of a concept with different variations. We define this approach, where the input uncertainty is intelligently managed to generate a distribution of solutions as *variational synthesis of mechanisms*. To demonstrate the efficacy of our approach, later on in this paper, we have presented solutions for the path generation problem using our motion generation solver as used in the example presented in Fig. 1. In the absence of using this approach, the solver would mostly produce impractical and defective solutions for general motion problems with a large number of poses. Thus, we have shown the effectiveness of the conditioning and input imputation by constructing valid motions from a crude input of path points. It should be noted that the main idea of the paper is not in solving the path generation problem using motion generation solvers but to introduce the idea of an intermediary that handles users' incomplete and uncertain input and communicates the necessary numerical subtlety to a generic, susceptible computational solver. The paper also presents an end-to-end deep neural network architecture, which approximates the conditional distribution of linkages on task parameters. The task parameters can be a coupler motion, path, or any other desired linkage property.

To accomplish the aforementioned goals, we employ a recently developed deep generative model called variational auto encoder (VAE) [10]. VAE comprises of two neural networks: (1) recognition model (also called, encoder) and (2) generative model (also called, decoder). The recognition model applies the learned prior probability distribution on observed input, thus acting as a posterior inference model. Additional benefits of using VAE are that the learned posterior inference model can also be used for a host of tasks such as denoising, representation, and visualization.

This paper presents two classes of VAEs, which collectively constitute the ML Intermediary. The first class is trained only on the coupler trajectories, which emphasizes representation learning of coupler paths (or motions) for conditioning target task paths (or motions). The second class is trained to learn the distributions of mechanisms in order to recognize their salient features, which is used for concept identification and clustering. This also leads to the generation of a set of diverse mechanism design concepts. An overall view of the approach is depicted in Fig. 2, which will be discussed in detail in later sections.

Another motivation behind this work is to understand the distribution of planar one degree-of-freedom (1-DOF) coupler motions generated by various planar linkages. This will enable us to find connections between different instances of synthesis problems and form a conception of the problem in general. This conception will enable us to answer the following questions: (1) given a path or a motion task, how likely it is that a particular mechanism type can perform the task, (2) for a given task, what is the distribution of linkage parameters with similar coupler motions (or paths), and (3) given a set of linkages, how to cluster the linkages into different design concepts. Although our approach can work with any number and type of linkages, we have implemented the methods for three types of planar mechanisms: (1) four-bar with all revolute joints, (2) slider-crank, and (3) Stephenson six-bar linkages.

The original contributions of the paper are in creation of (1) an end-to-end synthesis framework that accepts raw, high-level input from users and provides them with distinct concept solutions and (2) an approach using state-of-the-art deep generative models that learn the joint probability distribution of various linkage parameters and their interdependence to perform tasks such as input conditioning, imputation, and variational synthesis. In doing so, we leverage the emerging machine learning (ML) techniques to learn meaningful representations and combine them with simultaneous type and dimensional methods of kinematic synthesis [9] to enhance users' computational creativity.

Rest of this paper is organized as follows. Section 2.1 reviews background work and our previously developed motion generation solver [8,9], which is used as a representative of computational solvers and their issues. Section 3 presents the motivation behind using the deep generative models along with the theory behind it. Section 4 presents applications of VAE and conditional-variational auto encoder (C-VAE) to condition the input for the path and motion generation problem by learning the joint probability distribution of planar one DOF trajectories. Two examples are presented

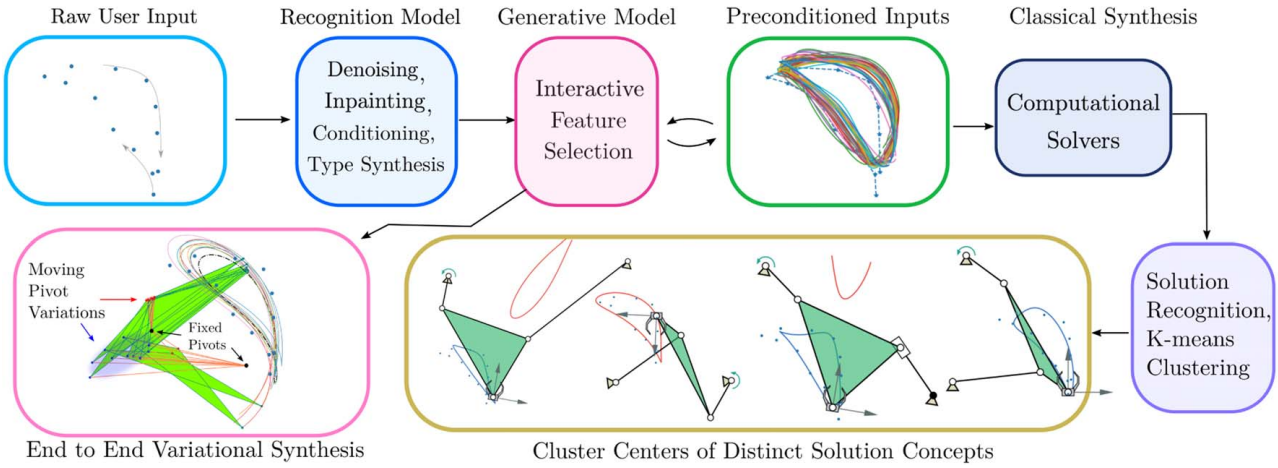

Fig. 2 Crude input from the user is passed through the recognition model which captures the features based on which various further actions are determined. The user has the control to override the feature selection by inspecting reformed variational inputs. These variational inputs are passed to classical synthesis methods like in Refs. [8,9] to generate a multitude of solutions. Solutions are passed through another recognition and clustering module to present the user with distinct distributions of solution concepts. In addition, an end-to-end deep generative model is trained that conditions coupler paths to linkage parameter distributions.

to showcase the recognition and variational generative capabilities of VAE. Section 5 explains how VAE can be used to recognize and cluster similar linkages. This section also presents a generative way of learning end-to-end synthesis for planar linkages using C-VAE. A sample of end-to-end variational synthesis using pure deep learning is depicted. Section 4.5 showcases an example interaction between a user and the intermediary which takes crude user input to return distinct concept solutions. The user can visualize and control which features to pass to the generator for motion generation, in contrast, to complete black-box approaches that use neural networks.

2 Background and Review

Applications of powerful function approximators like neural networks are not new to the domain of mechanism synthesis. Vasiliu and Yannou [11] have used artificial neural network (ANN) to interpolate the map between the path and link lengths for Grashof four-bar linkages with revolute joints. However, they have utilized ANNs mainly to memorize the mapping as a means to replace an atlas. Khan et al. [12] presented an approach where an ANN is used for mapping between Fourier coefficients corresponding to a coupler path and corresponding linkage parameters. Galan-Marin et al. [13] have used a similar approach instead of using Fourier coefficients, and they have used wavelet descriptors to represent the shape of the path. All of the above methods use ANNs just as a mapping tool from a closed coupler path to mechanism link ratios. In contrast to the black-box mapping approach of the above methods, we facilitate user interaction with the network by means of interactive manipulation of latent space. The approach taken by our methods is unsupervised and semi-supervised learning, with an emphasis on representation learning and understanding users' intentions.

Our previous approach [14] to capturing designers' intent by considering the continuity between precision poses results in defect-free linkages; however, this requires expensive database queries at the run time. The approach presented in this paper relies on models trained on those datasets and provides instantaneous response to the user at the run time. Moreover, this paper presents a comprehensive and unified approach to mechanism design solution generation and input rectification using VAEs and C-VAEs.

In contrast to the neural network approach, some researchers have attempted to solve this problem using traditional kinematic methods. For example, in case of the path generation problem, Ullah and Kota [15] and Wu et al. [16] have tried to incorporate the prior on user input by means of finding lower harmonic Fourier descriptors of the path followed by computing link dimensions using optimization methods. Li et al. [17] have developed a Fourier descriptor-based approach for approximate motion generation, which uses the same prior on the coupler path. These methods are relatively robust to spatial variations in input but susceptible to variations in timing information provided by the user. Sharma et al. [18] have addressed this issue to some extent by providing a scheme to compute optimal timing for the input points. Sharma et al. [19] have presented a method for motion synthesis by exploiting the Fourier descriptor relationship between path and orientation data. However, the above methods are only defined for the synthesis of four-bar linkages with revolute joints.

On the other hand, our approach is general enough to condition any type of user input such as precision points, pivot regions, or timing of the precision points and scales to higher-order linkage mechanisms and spatial robots. A key difference in the previous approaches and our approach is that the conditioning on the input is performed by learning the joint probability distribution of input parameters from a database of linkages instead of relying on the fact that four-bar linkages produce curves which have specific harmonic content. In the absence of this knowledge for higher-order linkages, it is difficult to scale those approaches. The end-to-end synthesis framework is based on a conditional-VAE with the coupler curves as predicates. This does not require an existing computational solver.

2.1 Review of an Algebraic Fitting Algorithm for the Motion Generation Problem.

In this section, we review a computational solver for the motion generation problem. In Refs. [8,9], we have presented a novel and efficient algorithm for computing type and dimensions of four-bar linkages with revolute and prismatic joints for the motion generation problem. The motion generation problem requires a coupler link to pass through n poses as close as possible. In cases where $n > 5$, the motion can only be approximated in general. According to the problem definition, the aim is to find locations on the coupler link where the ground links of a four-bar could be attached. Depending upon the link type, the moving joint of a ground link of four-bar lies on (1) a circle, (2) a fixed line, (3) a line that is tangent to a fixed circle, or (4) on a line sliding along a fixed line. Thus, the task turns into finding the locations of such points on the rigid body that satisfy one of the four constraints mentioned above. For example, Fig. 3 shows

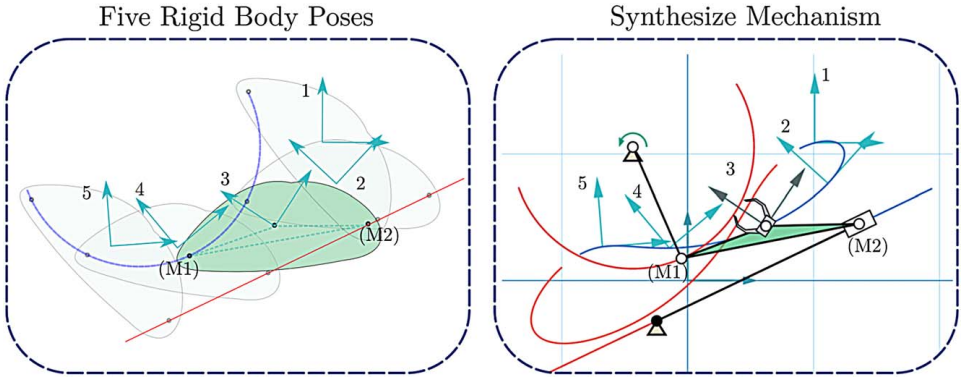

Fig. 3 The figure on the left depicts five poses of the rigid body with two points M1 and M2 on the rigid body tracing a circle and a line. Revolute–revolute (RR) and prismatic–revolute (PR) links are connected to M1 and M2, respectively, to form a slider-crank mechanism.

a rigid body in five precision poses, where points M1 and M2 on the rigid body are calculated to lie on a circle and a line, respectively. Once two such points are found on the rigid body, appropriate ground links can be attached as shown in Fig. 3 to create a four-bar mechanism.

This is done by solving a set of linear equations using singular value decomposition followed by a quartic equation. This method uses a unified formulation of all types of geometric constraints found in planar four-bar mechanisms, which in turn results in a unified approach to simultaneous type and dimensional synthesis of mechanisms. We have also developed a software application that implements this algorithm [20]. We use this application for generating some of the figures for the linkages synthesized in this paper.

3 Generative Model of Linkage Parameters

In this section, we present a generative model of linkage mechanisms, which should have an ability to understand salient aspects of the linkage parameters and to create diverse design concepts for a given task.

A generative model is based on a generative learning principal, which does not just passively observe the events it experiences but constructs its own perceptions about them. For example, training a generative model for coupler curves of the four-bar linkage formulates an understanding of the kind of curves a four-bar linkage can or cannot generate. This understanding is useful in various tasks such as input denoising, modification, or imputation. Here, input imputation is defined as the process of adding the missing information in the input which is necessary for the solver to process. Generative models encapsulate the salient information about the observed data which is essential for tasks involving recognition, representation, and computational creativity. In this work, we have used VAE [10] as our generative modeling framework. The parameters of generative models are much less in number than that of the data it is trained on. Thus, the model is forced to capture salient attributes and their variation in order to generate the data similar to it. This encapsulation of salient features is utilized in tasks that require an understanding of the data. In our case, we use it in providing the user a high-level control on manipulating different aspects of input data and to manage input uncertainties.

3.1 Theory of Variational Auto Encoders. VAE [10] is a neural network architecture that learns to approximate the true distribution of an observed data X. In this work, different models of VAE are trained to learn different observed data, thus X can represent different quantities for different VAEs. Depending upon the quantity that X represents, X can have different dimensions. For example, if a VAE is trained on coupler path dataset, then X represents coupler paths and will be denoted by X_{path}. Whereas, if VAE is trained on coupler motions or entire four-bar entire linkages, then X would represent coupler motions (X_{motion}) and four-bar linkages (X_{fourbar}), respectively. Figure 4 shows a general architecture of VAE. In this architecture, the recognition model encodes the data into probability distribution of latent variables, while the generative model is responsible for generating new data or reproducing trained data. In this figure, there are two hidden layers h_1 and h_2 in the recognition model, while the generative model has one hidden layer h_3. In the middle, there is feature space encoded by the variable z, which seeks to capture the essence of the input data. However, as opposed to an auto encoder architecture in which z is a discrete variable, in the VAE, the z is determined via a probability density function.

Let us assume that the d-dimensional data X is highly structured and occupies a much smaller k dimensional space. We know that a k dimensional unit Gaussian distribution can be mapped into any k-dimensional distribution through a nonlinear mapping. In other words, it can be said that the data are generated by some natural process that maps a k dimensional variable z to d-dimensional variable X. We try to mimic this process with an unknown parametric generative model based on hidden variable z, given that the probability distribution of z is unit Gaussian.

$$\text{pr}(z) = \mathcal{N}(0, 1) \tag{1}$$

$$X = G(z; \theta_g) \tag{2}$$

where θ_g are weight parameters of the neural network that acts as the generative model. The variable z is called latent variable, which contains salient information of the observed variable X. We would like to infer salient attributes z based on observed X, which

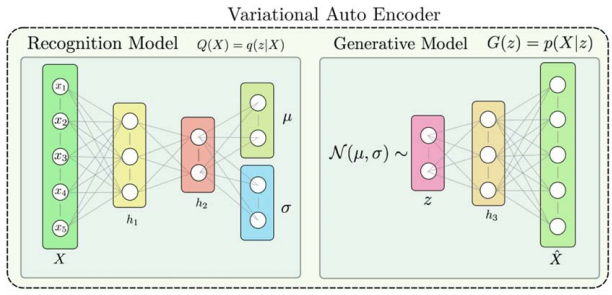

Fig. 4 Recognition model encodes the observed data X into probabilistic latent coding z of dimension much smaller than X. In this case, we assume a multivariate Gaussian distribution for z. Generative model takes samples from this distribution to generate output $\hat{X}$.

can be expressed by conditional probability pr($z|X$)

$$\text{pr}(z|X) = \frac{\text{pr}(X|z)\text{pr}(z)}{\text{pr}(X)} \tag{3}$$

where the abbreviation pr(A) represents the probability of variable A. Unfortunately, computing probability of X (i.e., pr(X)) is usually intractable. It involves computing the integral $\int \text{pr}(X|z)\text{pr}(z)dz$. However, we can apply variational inference [21] to estimate the joint probability distribution pr($z|X$). We approximate pr($z|X$) by a distribution $q(z|X)$, which we define such that it can be computed by a neural network Q.

$$\mu, \sigma = Q(X; \theta_e) \tag{4}$$

$$q(z|X) = \mathcal{N}(\mu, \sigma) \tag{5}$$

Here, $\mathcal{N}(\mu, \sigma)$ is a multivariate Gaussian distribution function with mean μ and variance σ. Now, we want to find parameters of the recognition model $Q(X)$ that predict the distribution $q(z|X)$ such that it is very similar to pr($z|X$). Then, we can use it to perform approximate inference of the intractable distribution.

The objective is to find parameters θ_g and θ_e of $G(z)$ and $Q(X)$, respectively, such that our model generates samples as close as the true observed distribution and distribution $q(z|X)$ is as close as the true distribution $p(z)$. This is achieved by training the neural network models for maximizing the lower bound of marginal likelihood, which is given by

$$\mathcal{L}^{(X^i)} = \mathbb{E}_{Q(z|X^i;\theta_e)}(\log(p(X^i|z))) - D_{\text{KL}}(Q(z|X^i; \theta_e)\|p(z)) \tag{6}$$

Here, the first term on RHS represents reconstruction likelihood and the second term is called Kullback–Leibler divergence (KL divergence) [22] which ensures that our learned distribution $Q(z|X; \theta_e)$ is similar to the true prior distribution $p(z)$. Since we assume that $p(z)$ is a Gaussian distribution, the lower bound of marginal likelihood becomes

$$\mathcal{L}^{(X^i)} = -(\hat{X}^i - X^i)^2 - \left(\sum_i^k {\sigma_i}^2 + {\mu_i}^2 - \log(\sigma_i) - 1\right) \tag{7}$$

The training objective is given by

$$\arg\min_{\theta_e,\theta_g}(-\mathcal{L}^{(X^i)}) \tag{8}$$

For further details, please see Ref. [10].

Once entire VAE is trained, the recognition model and the generative model can be used separately or together depending upon the application. In what follows, we describe the details of the recognition and the generator models.

3.2 Recognition Model. The architecture shown in Fig. 4 consists of a recognition model, which is an ANN. The inputs are passed through dropout [23], which randomly skips a connection between the input and the first hidden layer with a probability of 0.1. This small amount of uncertainty in the input helps in learning robust patterns present in the input. The recognition model is composed of hidden layers, which finally produce two z_{dim} dimensional vectors representing mean and variance of latent attribute z. The hidden layers can be convolutional layers or fully connected layers depending upon the nature of observed data. Convolutional layers are often the preferred choice when dealing with images. This is due to the nature of convolution operation which connects only the neighboring neurons to capture the local pattern. This works well in the case of images since pixels far away from each other may not be correlated. However, no such assumption is being made in the case of vectors representing kinematic quantities. Thus, this work uses fully connected layers. Each hidden layer is passed through a rectified linear (ReLU) activation function. ReLu function is given by

$$\text{ReLU}(x) = \max(0, x) \tag{9}$$

Output of the recognition model is a multivariate probability distribution of latent variable, which captures the salient attributes of the data. Random samples drawn from this distribution are passed to a generator network. In the case of training via back-propagation algorithm, gradients are passed from the generator to the recognition model by means of reparameterization trick [10]. The recognition model effectively captures the approximate posterior inference (pr($z|X$)) of the input data and thus can be used for tasks such as recognition, denoising, representation, and visualization purposes. In Sec. 4.2, we showcase various tasks performed using recognition.

3.3 Generator Model. Parameters of this neural network are learned to map a latent vector to a reconstruction, which would exhibit similar latent attributes if passed through the recognition network. Thus, the generative model can generate data whose probability distribution is similar to that of training data. The architecture of this model starts with an input layer which receives latent vector and passes through single or multiple layers of neurons culminating into the original size of observed data. The middle layers can be deconvolutional or fully connected layers which upscale the input they receive from the previous layer. Instead of ReLu, we apply a leaky ReLU activation function which prevents the gradient from becoming zero which can hinder training and is known to work better for generative networks [24]. Leaky ReLu is given by

$$\text{ReLU}(x) = \max(\alpha x, x) \tag{10}$$

where α is a small constant, which we take to be 0.001.

3.4 Variants of Variational Auto Encoder

3.4.1 Denoising Variational Auto Encoder. VAE networks can be used to remove noise from input data. This is done by adding noise to the input data while training. However, the generator is forced to produce original noise-free samples by defining the reconstruction loss between reconstructed image and original noise-free image. This forces the recognition network to encode robust features from the data.

3.4.2 Conditional Variational Auto Encoder. Until now, we have seen VAE with unsupervised learning architectures, i.e., which requires unlabeled data for training. C-VAEs are trained to learn the conditional distribution of an observed variable with respect to an explicitly observed property (or label) Y. This is achieved by small modification in vanilla VAE. Instead of only providing with observed data, the recognition network accepts concatenated input of X and Y. Moreover, the generator also receives a concatenated input of z and Y. This grants VAE additional information for the variational inference task. This can be used to generate samples which are highly associated with the given Y at the time of generation.

4 Variational Auto Encoder for Coupler Trajectories

In this section, we present the implementation of VAE for coupler path and motion databases. First, consider a database of coupler paths formed by planar linkages. Each coupler path is an ordered collection of m points sampled uniformly throughout the entire rotation of the crank, where each point has x and y coordinates. Thus, total observation dataset D of N data points is given by $D = \{X_i\}_{i=1}^N$, where each X_i is $\{x_j, y_j\}_{j=1}^m$ for path and $\{x_j, y_j, \theta_j\}_{j=1}^m$ for motion data.

4.1 Variational Auto Encoder Training. First, each coupler trajectory is normalized with respect to scale and position. This is done by subtracting the means $(\bar{x}, \bar{y})$ and dividing by the root

mean squared variance in Cartesian X- and Y-directions. Next, the trajectory is rotated such that its principal component axes are aligned with the coordinate axes. The principal component axes are the eigenvectors of the Covariance matrix of a point cloud data that define the trajectory. The covariance matrix C of a point cloud of m 2D points $\{x_j, y_j\}_{j=1}^{m}$ is given by

$$C = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{yx} & C_{yy} \end{bmatrix} \tag{11}$$

where

$$C_{xx} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})(x_i - \bar{x}) \tag{12}$$

$$C_{yy} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \bar{y})(y_i - \bar{y}) \tag{13}$$

$$C_{xy} = C_{yx} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y}) \tag{14}$$

Figure 5 shows the normalization process.

Various datasets of linkages are used to train various architectures of VAE with varying number of latent dimensional size. To create a dataset of a particular linkage type, a new linkage is constructed from a set of linkage parameters. These parameters are obtained by sampling from a uniform distribution of the practical range of linkage parameters. The obtained linkage is added to the dataset if it is sufficiently dissimilar from all the linkages previously added to the dataset. Here, the measure of dissimilarity between two linkages is given by a sum of $L2$ norms between the normalized trajectories traced by all moving points of two linkages. Table 1 presents the details on various datasets used for training.

Figure 6 depicts training losses for two such architectures with latent dimension (i.e., the dimension of z vector) 2 and 3. As we can see in Fig. 6, the model with three-dimensional z vector space achieves lower reconstruction error with slightly higher KL divergence loss. This result is expected because with an increase in latent size, the room to capture variation in the data increases. However, the increase in latent dimension also increases the complexity in visualization, interpretation, and manipulation of latent attributes in recognition tasks. It is interesting to notice that the two losses somewhat compete with each other in the training process. The reconstruction loss pushes the model to capture the diversity in the dataset, by forcing the generator to be able to reconstruct every type of data in the dataset. Whereas, KL divergence forces the latent space to occupy a restrictive distribution and thereby demanding coherence in the generation.

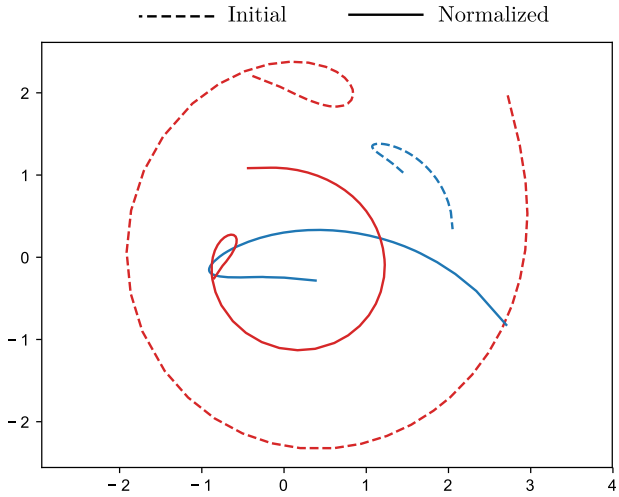

Fig. 5 Coupler trajectories are normalized with respect to position, orientation, and scale. Here, the orientation refers to the orientation of the path.

Table 1 Datasets used for training VAE and CVAE models

Data type	Size
Four-bar linkage (Grashof only)	480
Four-bar linkages	2188
Slider-crank linkages (Grashof only)	466
Six-bar Stephenson IIIa linkage (Grashof only)	937
Six-bar Stephenson IIIa linkage	3902

4.2 Recognition and Generation of Coupler Trajectories. For training or inference, each data point X_{path} is passed through a recognition model which computes the parameters for multivariate Gaussian distribution of latent vector. In inference, the aim is to recognize the salient features of the input point cloud. To showcase the inference functionality, let us input two one-dimensional point datasets as shown in Fig. 7. Each of the datasets represent an approximate target path. Now, we would like to infer their salient features and generate plausible coupler paths having similar salient features. We take a trained VAE with two-dimensional z vector trained on closed coupler paths. Each of the two datasets is passed through a recognition network, which predicts a 2D Gaussian distribution of latent features for each of the input. Now, for each case, random samples drawn from the distribution are passed to the generator network of the VAE, which generates samples with closed paths that resemble the original input path.

4.3 Interactive Shape Modification With Variational Auto Encoder. Let us assume a scenario where the user needs to specify a closed loop target path. Since the synthesis of linkages is chaotic, it is always fruitful to condition the inputs such that the probability of finding good linkages is maximized. Moreover, it is desirable to have a higher level of control on the overall shape of the target path. We use VAE to accomplish both of the above tasks. First, raw user input X is passed through the recognition model as shown in Fig. 7. The recognition model captures the shape user intends to draw and returns it in the form of latent feature distribution $q(z|X)$ as shown in Fig. 7. The VAE generator takes a sample feature vector z_s from $q(z|X)$ and generates a path $\hat{X}$. It should be noted that $\hat{X}$ is a sample from the distribution $p(X|z)$ and has more probability of being a path drawn from a four-bar. As we can see in Fig 7, generated paths follow the user-defined points and also resemble paths generated by Grashof four-bar linkages. Moreover, the user can select or modify the z_s interactively based on the variation of its corresponding generated

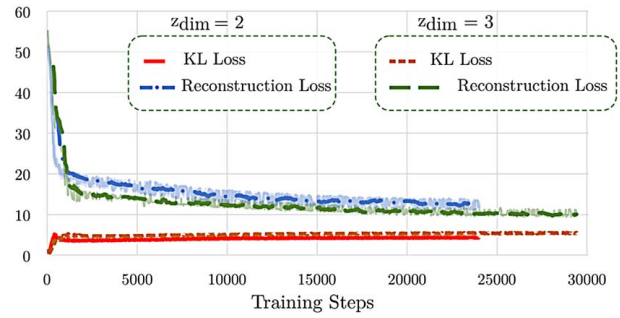

Fig. 6 Reconstruction and KL divergence losses for two architectures with z dimension 2 and 3. It can be seen that higher z-dimension enables capturing more variation in the database, which results in lower reconstruction losses.

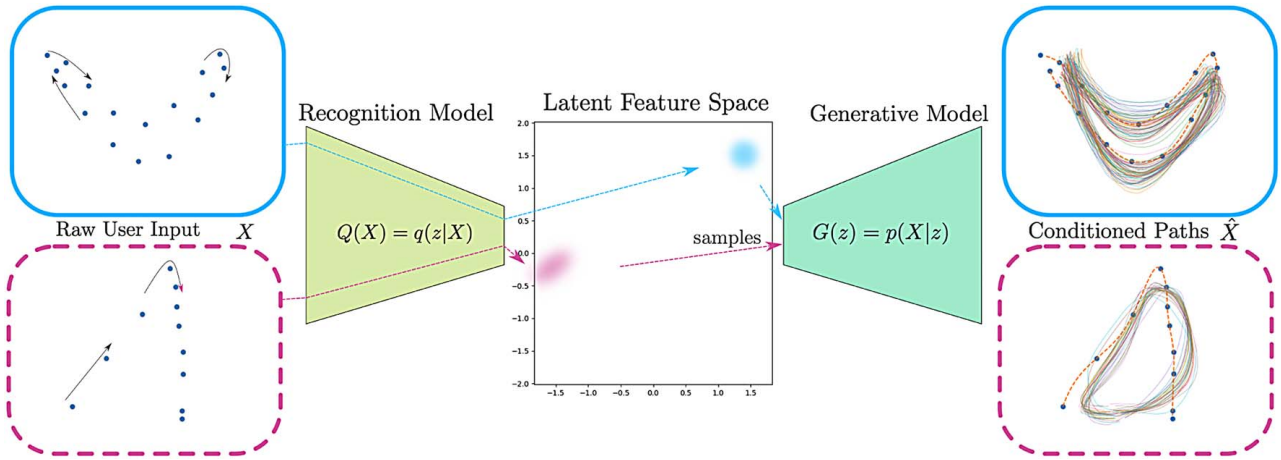

Fig. 7 The raw user input *X* is passed through a recognition network, which captures the salient information in the form of multivariate distribution of latent features. Random samples from this distribution are fed to the generator to generate paths with a high likelihood of producing good solutions. Moreover, users can manipulate the sample location in latent space, which gives them a low-dimensional and higher level of control on modifying the shape of the path.

path. Hence, the user has a higher level control on the shape of the path which improves the experience rather than modifying the input path point by point. This can be achieved by creating a user interface where the user can see the effects of changing feature vector z in all possible directions. Figure 8 presents samples created by the generator for different latent vectors. Here, the user can visualize where the given curve lies and change its shape by moving across the plane in 2D. This approach is amenable if the dimensionality of feature vector z is kept below 4, which indeed is the case for VAEs trained on coupler paths and motions.

4.4 Type Synthesis Via Recognition. Several VAEs with different architectures are trained on the dataset that comprises of coupler paths, motions from four-bars, slider-cranks, and Stephenson-type six-bars. Since the linkages above have a different topology, their coupler motions should possess some features that are affected by it. We use the features predicted by the recognition model of a VAE and train simple linear classifiers to identify their linkage type. Figure 9 depicts 2D embedding of closed coupler paths from four-bar (all revolute joints), slider-crank, and Stephenson six-bar mechanisms. It can be clearly seen that six-bar coupler curves cover a wider variety of shapes compared to four-bar linkages. Variety covered by slider-crank linkages is the lowest and is mostly overlapped by a four-bar linkage. The reasoning behind this observation can be given by the fact that the slider-crank linkages are a special case of 4R linkages when the length of the rocker link and its fixed pivot approach infinity. A classifier trained on such data can predict the probabilities of a given task being fulfilled by the corresponding linkage type.

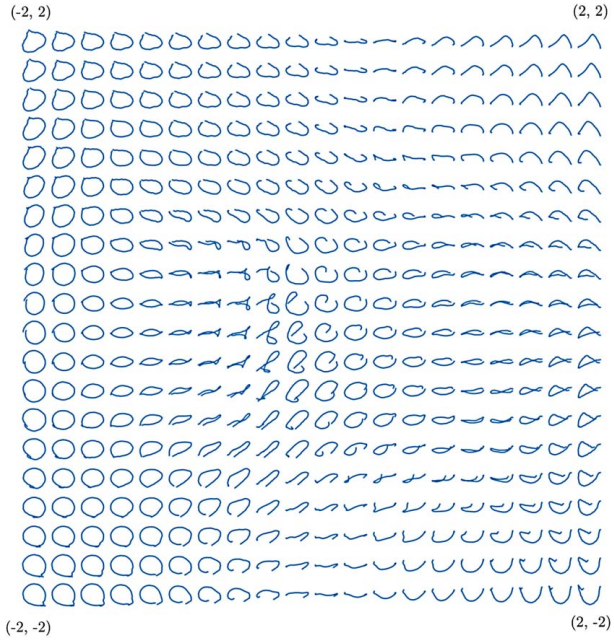

Fig. 8 Samples generated from a generator from a two-dimensional latent vector which is a vertex in the two-dimensional uniform grid with limits shown in corners. It can be seen that samples generated from neighboring vertices are highly correlated, which gives away the indication that recognition network learns the salient information about the shape of coupler curves.

4.5 Example User-Machine Learning Interaction. In this section, we showcase the application of VAE to solve a path generation problem using motion generation problem solver. As we have stated earlier, the ML Intermediary can take crude input from the user and provide all the necessary information required by the available solver with computational subtleties. These tasks require VAE

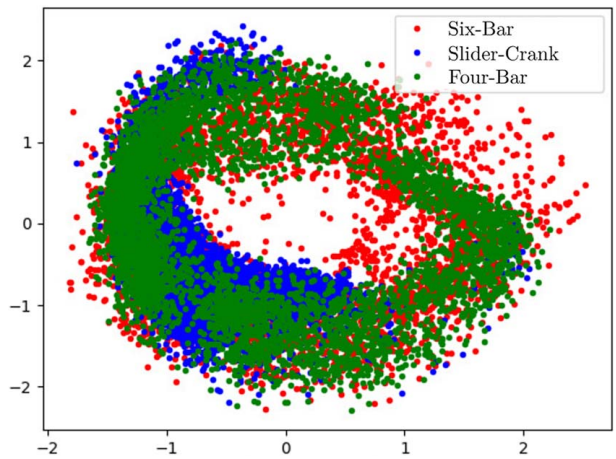

Fig. 9 Visualization of 2D feature embedding obtained by training a VAE on closed coupler paths of various planar linkages. The variety in features by means of spread over the space directly relates to the variety in shape of respective linkage type.

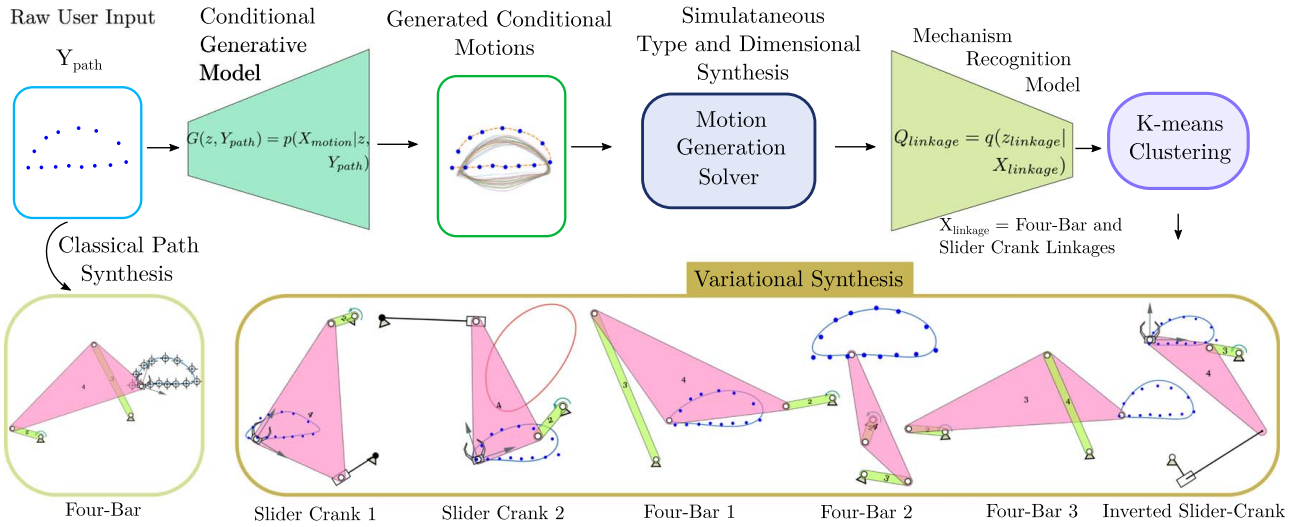

Fig. 10 C-VAE generates coupler motions corresponding to the path input. The orientation information of the generated motion is not shown for cleaner illustrations. The generated motions are fed as inputs for the motion generation problem. The solutions solved by Ref. [8] are passed through a linkage recognition model which predicts the latent distribution for each solution. This mixture of distributions is clustered to form distinctive concept distributions. The four-bar linkage in the bottom left corner depicts the optimal path synthesis solution using Ref. [18].

to capture primitive information provided by raw user input and generate a set of plausible coupler motions. It is important to note that since the orientation information required by the motion generation solver is not provided by the user, it should be computed by the ML Intermediary. This is done using a C-VAE that is trained to generate coupler motions (X_{motion}) data given its path (Y_{path}). The architecture and training scheme for the C-VAE follows the discussion presented in Sec. 3.4. We use our computational methods [8,9] for solving the motion generation problem in real time. The solutions returned by the solver are fed to the linkage recognition model of a VAE trained on entire four-bar linkages. This linkage recognition model computes a compact feature representation of each linkage on which we apply the K-means [25] clustering algorithm. The details of linkage recognition and clustering are presented in Sec. 5. It should be noted that the feature distributions of linkages produced by a recognition model are in ten-dimensional space. Thus, the depiction of the latent distribution of linkages and following clusters as shown in Fig. 10 are only for illustration purposes. The cluster centers are distinct solution concepts. Figure 10 depicts the entire procedure from start to finish. The solutions obtained using the proposed approach yield diverse mechanisms with various linkage topologies. In order to juxtapose our approach with classical approaches, Fig. 10 also depicts the result obtained using a classical path synthesis method [18] based on Fourier descriptors. It can be seen that the classical result is similar to one of the concept solution obtained using variational synthesis.

5 Variational Auto Encoder for the Entire Linkage

In Sec. 4, we presented applications of VAE for conditioning the task based on the learned prior from linkage trajectory dataset. This section presents generative modeling for the entire linkage mechanism which can be used as an alternative approach to the classical synthesis solvers. Since each linkage type has a different set of parameters, network architecture would be different for each of the linkage type. The objective is to train a model that learns conditional probability distribution of linkages given a coupler curve or motion, which is given by

$$\text{pr}(X_{\text{linkage}}|Y, z) = G_{\text{linkage}}(z, Y; \theta_g) \tag{15}$$

$$z \sim p(z|X_{\text{linkage}}, Y), \quad \text{and} \tag{16}$$

$$\text{pr}(z|X_{\text{linkage}}, Y) \approx Q_{\text{linkage}}(X_{\text{linkage}}, Y; \theta_e) \tag{17}$$

Here, G and Q are generator and recognition models from Eqs. (2) and (4), which also take an additional input Y. Here, X_{linkage} represents linkage state vector and is discussed next.

5.1 State of the Linkage. C-VAE discussed in Sec. 3.4 requires a tuple (X, Y) to train, where X is an observed variable (in this case, the entire linkage) and Y is an observed property or condition (in this case, the coupler curve). In theory, this formulation should work for any such tuple which has a strong correlation. We note that simply using the dimensional parameters of the mechanism, which describe link length and the fixed pivot locations of a mechanism, would not be a suitable choice for the vector X as they would merely describe a set of unrelated and discrete parameters with no possible meaning associated with them. This demonstrates that divorcing kinematic knowledge from the ML will not give us meaningful answers. Instead, we formulate the observed variable X from the linkage parameters such that it should contain the information of the entire simulation of linkage in its current configuration. First, we orient and scale a linkage such that one of its fixed links has magnitude 1 and is parallel to the Cartesian x-axis. We uniformly sample locations of all the points of interest for m crank orientations sampled uniformly throughout the possible range. Next, we represent these locations in polar coordinates with origin at fixed pivot corresponding to the crank. Then, these coordinates are stacked together for all of the m orientations. In the case of four-bar linkage, we have three points of interest P_1, P_2, and P_3 as shown in Fig. 11. Thus, the state tensor for the four-bar linkage is given by

$$X_{\text{state}} = \{r_{P1}, \theta_{P1}, r_{P2}, \theta_{P2}, r_{P3}, \theta_{P3}\}_{i=1}^{m} \tag{18}$$

where r_{Pj} and θ_{Pj} are the radial and angular coordinates of point P_j.

We flatten this tensor to form a 600-dimensional vector X_{fourbar} for a total of 100 orientations of the crank. For a six-bar Stephenson mechanism, we have a total of six moving pivot joints that we track.

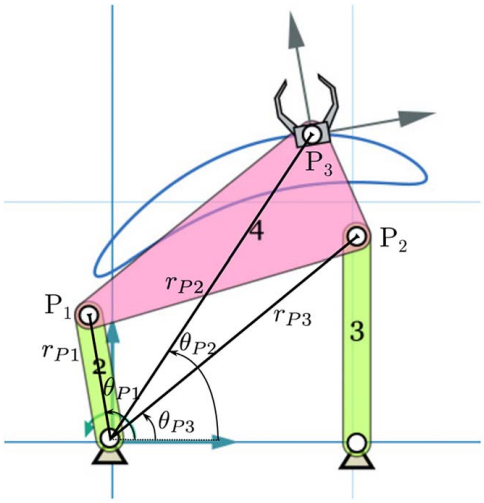

Fig. 11 A four-bar linkage with the fixed link of unit magnitude and colinear with *X*-axis. The polar coordinates of points P_1, P_2, and P_3 stacked together for *m* crank orientations constitute the state representation of the four-bar.

In that case, the state tensor for the linkage is given by

$$X_{\text{state}} = \{r_{P1}, \theta_{P1}, r_{P2}, \theta_{P2}, r_{P3}, \theta_{P3}, r_{P4}, \theta_{P4}, r_{P5}, \theta_{P5}, r_{P6}, \theta_{P6}\}_{i=1}^{m} \tag{19}$$

The dimension of the state vector X_{linkage} of a linkage is equal to $2 \times m \times \text{PoI}$, where PoI is the number of points of interest for that linkage.

5.2 Conditional-Variational Auto Encoder Training. As discussed in Sec. 5.1, X_{linkage} has dimensions $2 \times m \times \text{PoI}$. Whereas, Y is taken as the corresponding coupler path, which has dimensions $2m$. In order to train C-VAE, we pass a batch of X and Y to the network and compute gradients and losses. The training results have a similar pattern as presented in Fig. 6. The model architectures and their training results are tabulated in Table 2, where P. and M. signify path and motion, respectively. This table also shows the details of each architecture, such as the number of neurons in each layer, the number of hidden layers, and the losses.

5.3 Linkage Recognition and Clustering. Consider a set of M linkages represented by a set of vectors, $\{X_{\text{linkage}_i}\}_{i=1}^{M}$. Now, it is useful to find K representative linkages that cover the variety of obtained solutions, where K is a user-selected parameter. To find such K representative linkages, we perform K-means [25] clustering on the set of M linkages, where $K << M$. Before performing clustering, we first pass the set through the recognition module to obtain a compressed representation for the set $\{(\mu_{\text{linkage}_i}, \sigma_{\text{linkage}_i})\}_{i=1}^{M}$ given by

$$\mu_{\text{linkage}_i}, \sigma_{\text{linkage}_i} = Q_{\text{linkage}}(X_{\text{linkage}_i}; \theta_e) \tag{20}$$

where Q_{linkage} is the recognition model of the VAE trained on the corresponding linkage dataset.

Now, for computing the pairwise distance between two linkages X_{linkage_i} and X_{linkage_j}, we take the $L2$-norm between their corresponding mean latent vectors μ_i and μ_j. Using the $L2$-norm on latent representations instead of $L2$-norm on the original vectors has shown to yield better clustering for high dimensional data [26]. This results in the identification of K clusters and along with the corresponding cluster centers. Figure 2 shows cluster centers for the recognition and clustering task for a set of 100 four-bar linkages with $K = 4$.

5.4 End-to-End Variational Synthesis of Planar Linkages. C-VAE is trained to map the probability distribution of linkages to the shape of their corresponding coupler paths. C-VAE-10 takes a 200-dimensional vector Y and a sample from Gaussian distribution as an input and returns 600-dimensional vector $\hat{X}_{\text{linkage}}$ as output. From this 600 dimensional vector, we take the average location of each point of interest and construct the mechanism. This ensures that each generated sample results in a valid linkage.

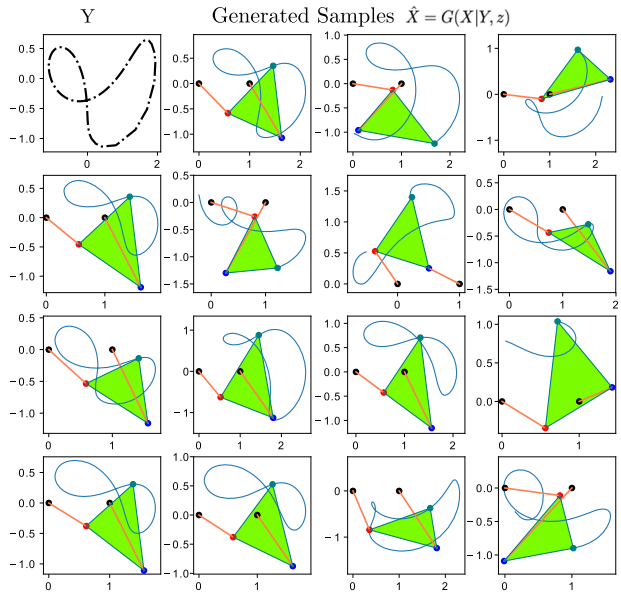

Fig. 12 Sample linkages generated by C-VAE-LF10 when it is supplied with the conditional coupler curve Y and 10-dimensional Gaussian multivariate *z*. Architecture of this C-VAE is presented in Table 2.

Table 2 VAE and C-VAE model architectures (FB = four-bar, SC = slider-crank, SB = six-bar)

Observed data (X)	X Dim	Name	Encoder Arch.	Latent (z) dim	Decoder Arch.	Y	Reconstruction loss	KL loss
FB P.	200	VAE-P2	(20)	2	(20)	—	11.46	4.33
FB P.	200	VAE-P3	(20)	3	(20)	—	9.95	5.48
FB M.	300	VAE-M2	(40)	2	(40)	—	21.31	4.23
FB M.	300	VAE-M3	(40)	3	(40)	—	13.95	5.81
FB, SC, SB P.	200	VAE-P3	(20)	3	(20)	—	16.12	5.34
FB, SC, SB M.	300	VAE-M3	(40)	3	(40)	—	18.43	6.45
FB M.	300	C-VAE-M3	(30)	3	(30)	6 points on P.	7.21	2.32
FB, SC, SB M.	300	C-VAE-M3	(30)	3	(30)	10 points on P.	10.13	3.42
FB linkages	600	C-VAE-LF10	(300, 100)	10	(100, 300)	P.	11.04	13.20
SB linkages	1200	C-VAE-LS15	(600, 300)	15	(300, 600)	P.	13.02	17.34

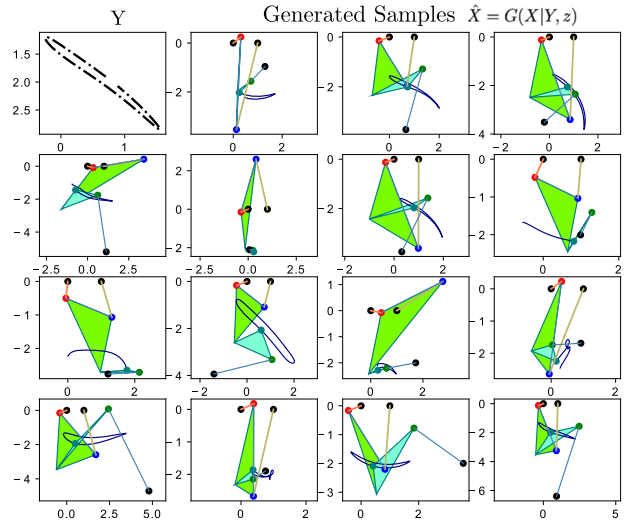

Fig. 13 Stephenson six-bars generated by C-VAE-LS15 (see Table 2) conditioned for the coupler curve *Y* and 15-dimensional Gaussian multivariate *z*

There is no guarantee that generated linkage should resemble the path Y, but C-VAE is trained to maximize that likelihood. Figures 12 and 13 depict some examples generated by C-VAE when a unseen coupler curve is passed as Y along with a multivariate Gaussian with zero mean and unit variance as z in Eq. (15). It is interesting to see the variations in the linkage parameters generated by C-VAE.

Combining this model with VAE for coupler trajectory can yield useful planar linkages with variations, making it an end-to-end deep learning model for the variational synthesis of planar linkages.

6 Conclusion

In this paper, we have presented a holistic machine learning-driven approach to the path and motion generation problems. This approach derives from the existing kinematic knowledge to create a new framework for mechanism synthesis, which solves problems that have had no good theoretical underpinning, such as defect-free generation, conditioning of the input, and contextual concept generation. Deep learning was used to learn the meaningful representations of linkage parameters and used in a novel way to enhance the users' design experience. Instead of discarding years of research that lead to the development of state-of-the-art synthesis algorithms, this framework combines them with deep learning to intelligently manage uncertainties and provide highly accurate distinct design solutions. A novel idea of an ML intermediary was introduced, which communicates between the user and computational algorithms. The intermediary intelligently captures the user's intention while managing the input for synthesis algorithms. Finally, it interprets numerous solutions returned by the solver and provides the user with a distinct distribution of concept solutions.

Acknowledgment

This work has been financially supported by the National Science Foundation (NSF) under a research grant to Stony Brook University (A. Purwar and Q. J. Ge; Grant No. CMMI-1563413). All findings and results presented in this paper are those of the authors and do not represent those of the funding agencies.

References

[1] McCarthy, J. M., and Soh, G. S., 2010, *Geometric Design of Linkages*, Vol. 11, Springer, Berlin.
[2] Sandor, G. N., and Erdman, A. G., 1997, *Advanced Mechanism Design: Analysis and Synthesis*, Vol. 2, Prentice-Hall, Englewood Cliffs, NJ.
[3] Hunt, K., 1978, *Kinematic Geometry of Mechanisms*, Clarendon Press, Oxford.
[4] Hartenberg, R. S., and Denavit, J., 1964, *Kinematic Synthesis of Linkages*, McGraw-Hill, New York.
[5] Suh, C. H., and Radcliffe, C. W., 1978, *Kinematics and Mechanism Design*, John Wiley and Sons, New York.
[6] Lohse, P., 2013, *Getriebesynthese: Bewegungsablufe ebener Koppelmechanismen*, Springer-Verlag, Berlin.
[7] Chase, T., and Mirth, J., 1993, "Circuits and Branches of Single-Degree-of-Freedom Planar Linkages," ASME J. Mech. Des., **115**(2), pp. 223–230.
[8] Ge, Q. J., Purwar, A., Zhao, P., and Deshpande, S., 2016, "A Task Driven Approach to Unified Synthesis of Planar Four-Bar Linkages Using Algebraic Fitting of a Pencil of g-Manifolds," ASME J. Comput. Inf. Sci. Eng., **17**(3), p. 031011.
[9] Deshpande, S., and Purwar, A., 2017, "A Task-Driven Approach to Optimal Synthesis of Planar Four-Bar Linkages for Extended Burmester Problem," ASME J. Mech. Robot., **9**(6), p. 061005.
[10] Kingma, D. P., and Welling, M., 2014, "Auto-Encoding Variational Bayes," Comput. Res. Repository, e-print arXiv:1312.6114.
[11] Vasiliu, A., and Yannou, B., 2001, "Dimensional Synthesis of Planar Mechanisms Using Neural Networks: Application to Path Generator Linkages," Mech. Mach. Theory, **36**(2), pp. 299–310.
[12] Khan, N., Ullah, I., and Al-Grafi, M., 2015, "Dimensional Synthesis of Mechanical Linkages Using Artificial Neural Networks and Fourier Descriptors," Mech. Sci., **6**(1), pp. 29–34.
[13] Galan-Marin, G., Alonso, F. J., and Del Castillo, J. M., 2009, "Shape Optimization for Path Synthesis of Crank-Rocker Mechanisms Using a Wavelet-Based Neural Network," Mech. Mach. Theory, **44**(6), pp. 1132–1143.
[14] Deshpande, S., and Purwar, A., 2019, "A Machine Learning Approach to Kinematic Synthesis of Defect-Free Planar Four-Bar Linkages," ASME J. Comput. Inf. Sci. Eng., **19**(2), p. 021004.
[15] Ullah, I., and Kota, S., 1997, "Optimal Synthesis of Mechanisms for Path Generation Using Fourier Descriptors and Global Search Methods," ASME J. Mech. Des., **119**(4), pp. 504–510.
[16] Wu, J., Ge, Q. J., Gao, F., and Guo, W. Z., 2011, "On the Extension of a Fourier Descriptor Based Method for Planar Four-Bar Linkage Synthesis for Generation of Open and Closed Paths," J. Mech. Robot. Trans. ASME, **3**(3), p. 031002.
[17] Li, X., Wu, J., and Ge, Q. J., 2016, "A Fourier Descriptor-Based Approach to Design Space Decomposition for Planar Motion Approximation," ASME J. Mech. Rob., **8**(6), p. 064501.
[18] Sharma, S., Purwar, A., and Ge, Q. J., 2019, "An Optimal Parametrization Scheme for Path Generation Using Fourier Descriptors for Four-Bar Mechanism Synthesis," ASME J. Comput. Inf. Sci. Eng., **19**(1), p. 014501.
[19] Sharma, S., Purwar, A., and Ge, Q. J., 2019, "A Motion Synthesis Approach to Solving Alt-burmester Problem by Exploiting Fourier Descriptor Relationship Between Path and Orientation Data," ASME J. Mech. Robot., **11**(1), p. 011016.
[20] Purwar, A., Deshpande, S., and Ge, Q. J., 2017, "Motiongen: Interactive Design and Editing of Planar Four-Bar Motions Via a Unified Framework for Generating Pose- and Geometric-Constraints," ASME J. Mech. Robot., **9**(2), p. 024504.
[21] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D., 2017, "Variational Inference: A Review for Statisticians," J. Am. Stat. Assoc., **112**(518), pp. 859–877.
[22] Kullback, S., and Leibler, R. A., 1951, "On Information and Sufficiency," Ann. Math. Stat., **22**(1), pp. 79–86.
[23] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., 2014, "Dropout: A Simple Way to Prevent Neural Networks From Overfitting," J. Mach. Learn. Res., **15**, pp. 1929–1958.
[24] Radford, A., Metz, L., and Chintala, S., 2016, "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks," Comput. Res. Repository, e-print arXiv:1511.06434.
[25] Lloyd, S., 1982, "Least Squares Quantization in PCM," IEEE Trans. Inform. Theory, **28**(2), pp. 129–137.
[26] Song, C., Liu, F., Huang, Y., Wang, L., and Tan, T., 2013, "Auto-Encoder Based Data Clustering," *Iberoamerican Congress on Pattern Recognition*, Springer, Berlin, pp. 117–124.