

ITPNet: Initial Trajectory Prediction Network - A deep learning framework for predicting initial trajectories to warm start optimization-based motion planners

Sankaranarayanan Natarajan¹ and Prof. Frank Kirchner^{1,2}

^{1*}Robotics Innovation Center, German Research Center for Artificial Intelligence (DFKI GmbH), Robert-Hooke-Straße 1, 28359, Bremen, Germany.

²AG Robotik, University of Bremen, Robert-Hooke-Straße 1, 28359, Bremen, Germany.

Contributing authors: sankaranarayanan.natarajan@dfki.de; frank.kirchner@dfki.de;

Abstract

Motion planning for robot manipulators in a cluttered environment is one of the most researched areas in the field of robotics. For high degrees of freedom (DOF) robotic systems, optimization-based motion planners are often preferred for trajectory planning, as they are computationally efficient and provide a smooth, locally optimal solution. To converge to an optimal solution, an optimization-based motion planner needs a good initial trajectory. Finding an initial trajectory, which is not far away from the basin of attraction of the optimum is not a trivial task. In this work, we propose an Initial Trajectory Prediction Network (ITPNet), a deep neural network framework for predicting an initial trajectory to warm start optimization-based motion planners. Given a planning task in the form of task and environment features, the ITPNet predicts the best initial trajectory for warm starting an optimization-based motion planner. Two different task features: joint and Cartesian features, and three types of environment features extracted using: Principal Component Analysis (PCA), Variational Autoencoder (VAE), and Signed Spatial Distance (SSD) techniques are compared. The learned models are evaluated on an upper-torso humanoid system in two different scenarios. The results show that the model, using the Cartesian task and the SSD-based environment features, efficiently learns the mapping between the planning tasks and the optimal trajectories. Warm-starting the planner with the predicted initial trajectory, even for an unseen environment results in a higher success rate and requires fewer iterations.

Keywords: Motion planning, optimization-based motion planner, initial trajectory prediction, deep learning

1 Introduction

Motion planning for robotic manipulators stands as one of the cornerstones of robotics research, commanding significant attention and investigation. A motion planning problem can be defined as finding an optimal path for a robotic system to move from its current configuration state

to a target configuration state respecting any given constraints. In low-dimensional planning problems, combinatorial or exact motion planning techniques such as cell decomposition and visibility graphs are commonly employed. These methods are complete in nature[21], implying that the algorithms can either provide a solution or determine within a finite time frame that

a solution does not exist for the given problem. Due to the curse of dimensionality, these combinatorial planning methods are not applicable to higher dimension planning problems. Higher-dimensional motion planning problems for manipulators are primarily solved using sampling-based or optimization-based methods. Sampling-based planners do not explicitly construct traversable and non-traversable regions, but instead, they randomly sample robot state from the Configuration space (C-Space) of the robot. Only valid states respecting any given constraints, such as collision avoidance, kinematic constraints, etc., are considered. These valid states will be used to construct a roadmap, as seen in PRM[16], or a tree, as seen in RRT[20], RRT*[15], BIT*[6], etc. A valid path is then obtained from the valid states of the roadmap or the tree structure. Sampling-based planners offer probabilistic completeness, which means that given enough time and states, the probability of obtaining a solution approaches one. The solutions generated by sampling-based planners are not inherently smooth. Typically, the solution is smoothed before execution on a manipulator. As the dimension of the problem increases, the time required to find a solution also increases. On the other hand, optimization-based solvers such as CHOMP[30], STOMP[13], and TrajOpt[31] provide smoother solutions. They can find a locally optimal trajectory even for higher dimensional planning problems respecting any given constraints. Optimization-based planners formulate the planning problem as an optimal control problem, where its goal is to minimize the sum of state and control costs along the trajectory respecting any given equality and inequality constraints. In the case of STOMP, the state cost helps in avoiding constraints like collision avoidance, while the control cost ensures that the resulting trajectory is smooth.

The optimization-based planners are given with an initial trajectory, which the planners optimize. The initial trajectory is often a linearly interpolated joint space trajectory from a given start to the goal states. The effectiveness of the initial trajectory significantly influences the search for an optimal solution. If the given initial trajectory lies too far away from the basin of attraction, then the planner cannot find an optimal solution or it takes longer time to converge to an optimal solution. The ability of the optimization-based

planner to find an optimal solution lies in the quality of the given initial trajectory. Finding a good initial trajectory for a given problem in a cluttered environment is still an open research problem. The concept of warm-starting an optimization-based planner with a good initial trajectory has been extensively explored in the domains of optimal control and motion planning.

In the motion planning domain, a common strategy involves using a sampling-based planner to obtain an initial solution, which is then employed to warm-start an optimization planner. The dimensionality of the planning problem plays a critical role here, as the dimension of the problem increases, it takes longer for the sampling-based planner to find the solution. Another approach is an analog to Case-Based-Reasoning (CBS) in the field of artificial intelligence, where a solution from a previous problem is utilized to solve a similar problem. Here, an initial trajectory is selected from a stored database using a heuristic approach. In our work, we follow a similar approach, where a trajectory prediction model is learned from a stored database. The method described in this work is based on our previous work [25], employing a Sequence to Sequence (Seq2Seq) learning architecture to predict an initial trajectory to warm start an optimization-based motion planner. In the previous work [25], a model is trained using the combination of joint space as task features and SSD-based environment features. Even though the model performed well for a 6 DOF manipulator, its performance was reduced with the increase in the manipulator DOF. The main contribution of this work is the following.

- ITPNet: A Seq2Seq based learning framework for predicting initial trajectories for higher DOF systems.
- The ITPNet framework can be used as a benchmarking tool for testing various tasks and environmental features. In this work, two different types of task features: joint space and Cartesian space features, and three different types of environment features using: PCA, VAE and SSD are considered and evaluated.
- Finally, we experimentally show that ITPNet using the Cartesian space task features and the SSD-based environment feature predicts

the best initial trajectories even for an unseen planning environment.

The paper is structured as follows: In Section 2, works related to trajectory prediction in the motion planning domain are discussed. Section 3 discusses the proposed ITPNet framework. In Section 4 different types of task and environment features are discussed. Sections 5 and 6 describe the experiments and the results of evaluating the ITPNet model using different features. Finally, Sections 7 and 8 draw the discussion and conclusion of this work.

2 Related Work

The concept of predicting a good initial trajectory to warm-start an optimizer is widely practiced in the field of optimal control and motion planning. Mansard et al.[23] created a memory of motion of optimal trajectories, which is retrieved online to warm-start a model predictive controller. They demonstrated that, instead of learning a control policy, learning good states and control trajectories results in a better convergence rate of the predictive controller. In [35], the authors achieved real-time control of a quadrotor by predicting a good initial trajectory to warm-start an optimal controller.

In recent years, several works have been proposed to improve the performance of optimization-based planners by focusing on finding a good initial trajectory. Most of these approaches follow a similar paradigm, where a database is created by storing optimal trajectories, and an initial trajectory is hierarchically selected from this database. Stolle et al.[33] demonstrated that a global control policy can be established from a library of trajectories. Jetchev et al. [9–11] generated a database of planning features and their corresponding optimal solutions in the form of a frame-invariant task space representation. Subsequently, they predict an index of the task space representation from the database for a similar planning problem. The task space representation undergoes post-processing optimization to find a joint space trajectory from it. This joint space trajectory serves as a warm starter for the optimizer. Dey et al.[4] proposed a contextual sequence prediction approach based on features to optimize

the control library. Rather than predicting a single initial trajectory, they predict multiple initial trajectories. This approach ensures fallback solutions are available in case the predicted trajectory cannot converge to an optimal solution. Dragan et al.[5] showed that instead of predicting complete trajectories from a library, predicting important low-dimensional qualitative attributes of trajectories can be effective. These trajectory attributes can be considered as sub-goals, and the initial trajectory generated using these attributes would result in a low-cost trajectory. In [32], initial trajectories are sampled from a trajectory distribution generated from human demonstration. Following a similar approach proposed in [23], Lembono et al.[22] construct a memory of motion for warm starting a trajectory optimizer. They demonstrated that employing an ensemble method to combine three function approximators: k-Nearest Neighbor, Gaussian Process Regression, and Bayesian Gaussian Mixture Regression results in better performance compared to individual function approximators. Merkt et al.[24] proposed an encoded trajectory library that predicts a good initial trajectory from the trajectory library through classification. Notably, their work is constrained to a known environment. Pan et al.[27] designed a trajectory effectiveness classifier, which quantifies the quality of an initial trajectory. They presented the spatial signed distance (SSD) features, a compact way to encode the environment information along a trajectory.

Even though predicting a good initial trajectory for warm starting has been predominantly studied, very few works [10], [32] demonstrate how to effectively represent environmental features from sensor data for the learning model. The representation of the environment plays a significant role in learning the initial trajectory prediction model. In comparison to the aforementioned works, in this work, three different types of environment representation and two different types of task representation are examined and evaluated. The presented work introduces a deep learning framework for predicting initial trajectories in cluttered environments. We demonstrate that the combination of Cartesian space task features and SSD-based environment features outperforms other types of representations. The

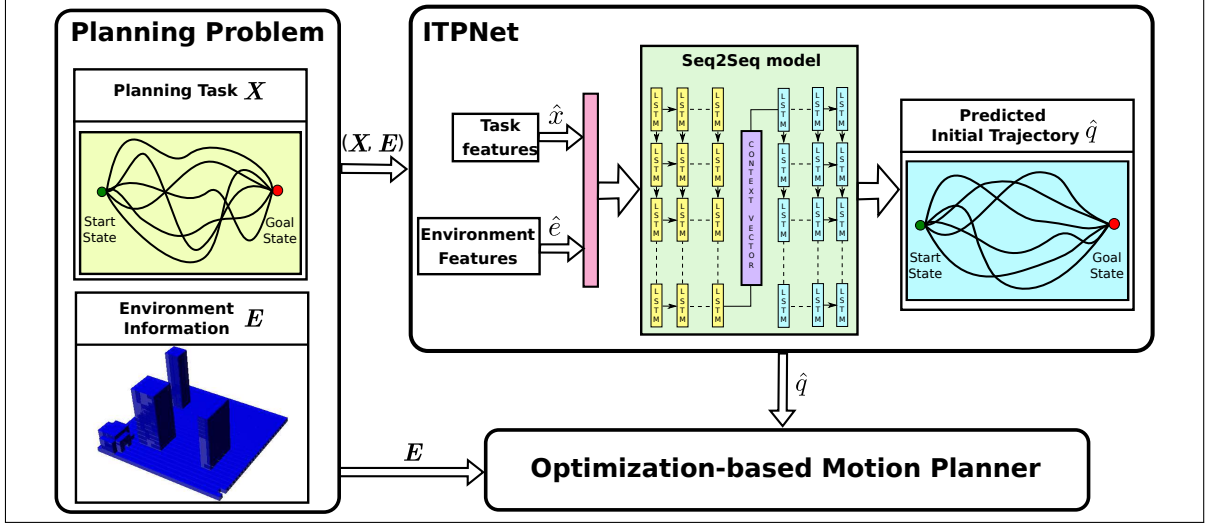


Fig. 1 ITPNet framework architecture

use of SSD collision features provides better generalization compared to alternative environment representations.

3 Architecture For Trajectory Prediction

An overall architectural view of the proposed Initial Trajectory Prediction Network (ITPNet) is shown in Figure 1. The planning problem consists of a planning task X and the current environment information E . The planning task X represents the start and goal robot configurations. It is generated by naively interpolating the start and goal joint states of the planning problem, which is often used as the default initial trajectory in many optimization-based motion planners. The constraint considered by the motion planner in this work is collision avoidance. The environment information E describes the distribution of obstacles in the robot’s workspace in the form of a voxel grid. A simple depth image camera or a lidar system can create a pointcloud representation of the captured environment, which can later be converted into a voxel grid representation. The planning task X and its environment information E are converted into task $\hat{x}$ and environment $\hat{e}$ features. These feature representations play a crucial role in learning the trajectory prediction model. In this work, two different types of task features and three different types of environment features

are considered and evaluated. A detailed description of these feature representations is provided in the subsequent sections. Given the task $\hat{x}$ and environment features $\hat{e}$, the ITPNet maps these features to an optimal trajectory q^* .

$$g : (\hat{x}, \hat{e}) \mapsto q^* \quad (1)$$

The ITPNet uses the Seq2Seq architecture [34] for learning the initial trajectory prediction model. The Seq2Seq learning methods are predominantly used for natural language translation [14], [3], [34], due to their ability to convert an input sequence from one domain into an output sequence in an another domain. In [36], the authors use a Seq2Seq model to predict captions for videos. In [17], [28], deep learning using a Seq2Seq model to predict the trajectory of the surrounding vehicle is proposed.

The Seq2Seq model follows an encoder-decoder paradigm as shown in Figure 2 using a Recurrent Neural Network (RNN). The goal of an encoder is to map an input sequence $X = (x_1, x_2, \dots, x_i)$ to a fixed length context vector c . This context vector c is then passed on to a decoder, which sequentially predicts the output data y_j based on the context vector c and the previously predicted data $y_1, y_2, \dots, y_{j-1}$. The Long Short Term Memory (LSTM) cell, a variant of the Recurrent Neural Network (RNN), is used in the Seq2Seq architecture. The LSTM is advantageous over the vanilla version of the RNN due to its ability to address the

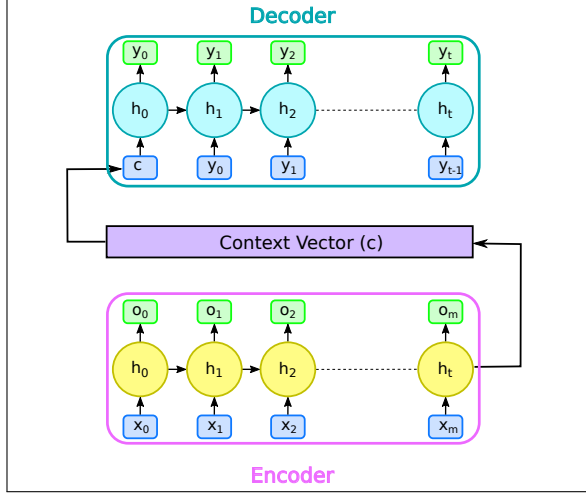


Fig. 2 Seq2Seq Architecture

vanishing gradient problem. The task $\hat{x}$ and environment $\hat{e}$ features are jointly fed to the encoder LSTM layer at each time step. The context vector c is the last hidden state of the encoder LSTM layers at the final time step. The decoder LSTM layer is initialized using the context vector c . The output of the decoder at the previous time step is fed as input to the current time step. The output of the decoder LSTM layer at each time step is concatenated to form an initial trajectory $\hat{q}$. This initial trajectory $\hat{q}$ from ITPNet is then used to warm start an optimization-based motion planner.

4 Features

In this work, two types of features are used: task $\hat{x}$ and environment $\hat{e}$ features. The task features $\hat{x}$ represent the actual planning problem such as the start and goal state of the manipulator. The environment features $\hat{e}$ represent the obstacles distribution of the planning scene. To effectively learn the trajectory prediction model, the features should be an abstract and compact representation of the planning problem. The size of the features is a bottleneck in the learning process of the trajectory prediction model. In particular, the environmental features should not only be rich enough to capture most of the environmental information but also be compact enough to allow the learning algorithm to learn the model effectively. In the following subsection, different types of task and environment features are explained.

4.1 Task Features

The task features embody the essential components of the planning task. In the manipulation domain, the task involves maneuvering a manipulator from a start configuration to a goal configuration. Task features are often represented in joint space, such as manipulator joint angles, or Cartesian space like the manipulator end-effector pose. These task features are extracted from the given planning task, which is in the form of a joint space trajectory. Each of these task features is described in the following sub-sections.

4.1.1 Joint Space Features

An intuitive way of representing the task features is to directly use the given planning task. Instead of using the interpolated joint angle directly, the sine and cosine of the joint angle are used $q = [\sin(q), \cos(q)]$. The choice of using the sine and cosine values of a joint angle provides a continuous and smooth representation, facilitating neural networks in learning and generalization. These linearly interpolated joint state representations are concatenated to form a joint space trajectory. The joint space task features have the form:

$$\hat{x} = \begin{bmatrix} [(s_{11}, c_{11}), (s_{21}, c_{21}), \dots, (s_{m1}, c_{m1})] \\ [(s_{12}, c_{12}), (s_{22}, c_{22}), \dots, (s_{m2}, c_{m2})] \\ \vdots \\ [(s_{1n}, c_{1n}), (s_{2n}, c_{2n}), \dots, (s_{mn}, c_{mn})] \end{bmatrix} \quad (2)$$

Here (s_{mn}, c_{mn}) represent the sine and cosine joint angle of the m^{th} joint of the n^{th} waypoint. The loss function for using the joint space features is of the form:

$$L = \sum_{m=1}^M \sum_{n=1}^N \|\hat{q}_{m,n} - q_{m,n}\|^2 \quad (3)$$

where $\hat{q}$ and q represent the predicted and target joint trajectories for N number of waypoints. Here M represents number of joints in the manipulator.

4.1.2 Cartesian Space Features

The Cartesian space features are generated by calculating the manipulator's end-effector pose along

the given planning task. The Cartesian space features are of the form:

$$\hat{x} = \begin{bmatrix} (p_1, o_1) \\ (p_2, o_2) \\ \vdots \\ (p_n, o_n) \end{bmatrix} \quad (4)$$

Here (p_n, o_n) represents the position and orientation of the manipulator end-effector at n^{th} way-point. Learning an orientation representation in Cartesian space is not a trivial task. The orientation representation must satisfy the orthogonality constraint and needs to be continuous for the neural network to learn the orientation representation. An obvious choice is to represent the orientation representation in $\mathcal{SO}(3)$ matrix form $\mathcal{R}$. This representation is continuous but an orthogonality constraint must be enforced. Rotation parameterization techniques such as Euler-angle, Angle-axis, and quaternions are not continuous in the Euclidean topology [37]. This discontinuity in the representation makes it difficult for the neural network to learn them. Moreover, additional work needs to be done to hold the orthogonality constraint. We use a continuous 6D orientation representation $\hat{x}_r$ proposed in [37] to help the neural network learn the orientation. An orientation representation space $\hat{x}_r$ is considered continuous if there exists a continuous mapping from the original space $\mathcal{R}$ to the orientation representation space $\hat{x}_r$, i.e. $f(g(\mathcal{R})) = \mathcal{R}$

$$\hat{x}_r = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix} \xrightleftharpoons[g: \mathcal{R} \mapsto \hat{x}_r]{f: \hat{x}_r \mapsto \mathcal{R}} \begin{bmatrix} | & | & | \\ r_1 & r_2 & r_3 \\ | & | & | \end{bmatrix} = \mathcal{R}$$

The Forward mapping (f) will map the 6D orientation representation space $\hat{x}_r$ to the original $\mathcal{SO}(3)$ representation $\mathcal{R}$ by following Gram-Schmidt procedure.

$$f \left(\begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix} \right) = \begin{bmatrix} | & | & | \\ r_1 & r_2 & r_3 \\ | & | & | \end{bmatrix} \quad (5)$$

The $\mathcal{SO}(3)$ representation in the matrix form $\mathcal{R}$ is calculated using the below formulas[37]:

$$r_1 = \hat{a}_1 = \frac{a_1}{|a_1|} \quad (6a)$$

$$r_2 = \hat{b}, b = a_2 - (r_1 \cdot a_2)r_1 \quad (6b)$$

$$r_3 = r_1 \times r_2 \quad (6c)$$

The inverse mapping (g) will map the $\mathcal{SO}(3)$ representation $\mathcal{R}$ to the 6D orientation representation space $\hat{x}_r$. The inverse mapping $g: \mathcal{R} \mapsto \hat{x}_r$ is done by dropping the last column vector of the $\mathcal{R}$ as shown in equation 7

$$g \left(\begin{bmatrix} | & | & | \\ r_1 & r_2 & r_3 \\ | & | & | \end{bmatrix} \right) = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix} \quad (7)$$

The task features using the Cartesian space are of the form $\hat{x}$:

$$\hat{x} = \underbrace{\{p_x, p_y, p_z\}}_{\text{3D position features } \hat{x}_p}, \underbrace{\{a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}\}}_{\text{6D orientation features } \hat{x}_r} \quad (8)$$

In this 9D feature vector, the first 3D features $\hat{x}_p = \{p_x, p_y, p_z\}$ represent the position features and the remaining 6D features $\hat{x}_r = \{a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}\}$ represent the orientation features $\hat{x}_r$.

The loss function using Cartesian space is of the form:

$$L = (\alpha \times pos_err) + ((1 - \alpha) \times ori_err) \quad (9)$$

where pos_err and ori_err hold the position and orientation errors between the predicted and the desired end-effector pose. The term α is the scaling factor. The position error pos_err is calculated as below:

$$pos_err = \underbrace{\frac{1}{N} \sum_{n=1}^N (x_{p_n} - \hat{x}_{p_n})^2}_{\text{position cost}} + \underbrace{\alpha \left(\frac{1}{N} \sum_{n=2}^N (x_{p_n} - x_{p_{n-1}})^2 \right)}_{\text{smooth cost}} \quad (10)$$

Here, the first term is a mean squared error of the predicted position $\hat{x}_{p_n}$ and the desired position x_{p_n} . The second term is added to penalize jumps in the consecutive position. Here, N represents the number of waypoints. The orientation error loss ori_err is calculated by calculating the geodesic loss between the two rotation matrices $\mathcal{R}_p$ and $\mathcal{R}_d$.

$$ori_err = \frac{1}{N} \sum_{n=1}^N \arccos \left(\frac{\text{trace}(R_n) - 1}{2} \right) \quad (11)$$

where,

$$R = \mathcal{R}_p \mathcal{R}_d^t \quad (12)$$

Here, $\mathcal{R}_p$ is the rotation matrix calculated from the predicted $6D$ orientation representation $\hat{x}_r$ using the forward mapping function f as shown in equations (6) and $\mathcal{R}_d$ is the desired rotation matrix.

4.2 Environment Features

In the field of motion planning for manipulators, it is common to employ laser scanners or depth image cameras to capture the environment information of the manipulator’s workspace. These sensors generate a significant amount of data. Due to their size, it is not practical to use this data directly as planning features in learning procedures. To address this challenge, linear dimension reduction methods such as PCA, Independent Components Analysis (ICA), Linear Discriminant Analysis (LDA), etc., or non-linear dimension reduction techniques like Auto-Encoders could be employed to obtain a reduced dimensionality of the planning environment. The key essence of these dimensionality reduction methods is to reduce the size of environment information but keep the variance in the data. For instance, Jetchev et al.[9] utilized PCA to reduce the environment information for learning initial trajectories. In [27] the authors propose a novel collision feature method called Signed Spatial Distance (SSD). The SSD offers a compact and yet a rich representation of the obstacle distribution around an initial trajectory. In this work, the planning scene is similar to the scene shown in Figure 3. A robot is placed in front of a table with 4 primitive objects randomly placed on it. Depth image sensors are placed around the table to capture the

environment information as a pointcloud, which in turn is converted into a voxel grid E . The environment information E is converted to environment features $\hat{e}$ before feeding to the ITPNet. In the following subsection, three different types of environment features are discussed.

4.2.1 Environment Features using PCA

PCA is a statistical approach widely used as a dimensionality reduction tool. It is an unsupervised linear transformation technique, which projects the data as a linear combinations of principal components. PCA is often used for dimensionality reduction, but it is also used for noise reduction, visualization, etc. The goal of the PCA method is to capture the key low dimensional variance structure from the high dimensional, correlated data without losing valuable information. This is done by calculating the eigenvectors and eigenvalues from the covariance matrix of the data. The principal components are formed by sorting the eigenvectors based on their eigenvalues.

Using PCA to reduce the environment information is often used in learning models for predicting trajectories [12]. The reasons for using PCA to extract environment features are that it is easier to implement and it is computationally less expensive compared to other dimensionality reduction techniques. For analyzing PCA, environment information similar to the one shown in Figure 3 is used. Altogether 4000 planning scenes are collected. In each scene, the obstacles are randomly placed. The voxel grid is of size $33 \times 67 \times 32$, with each voxel of size 3cm. So altogether we have a total of 70752 features. Here, we reduce it to 200 features by retaining the first 200 PCA components. Figure 4 shows the reconstructed voxel grid from the 200 PCA components. The feature size of 200 is empirically chosen. An increase in the features needs a deeper neural network layer and we need more samples to learn the trajectory prediction model. As shown in Figure 5 with 200 principal components 60% of the variance in the data can be represented. These 200 features will be used as the environment features $\hat{e}$.

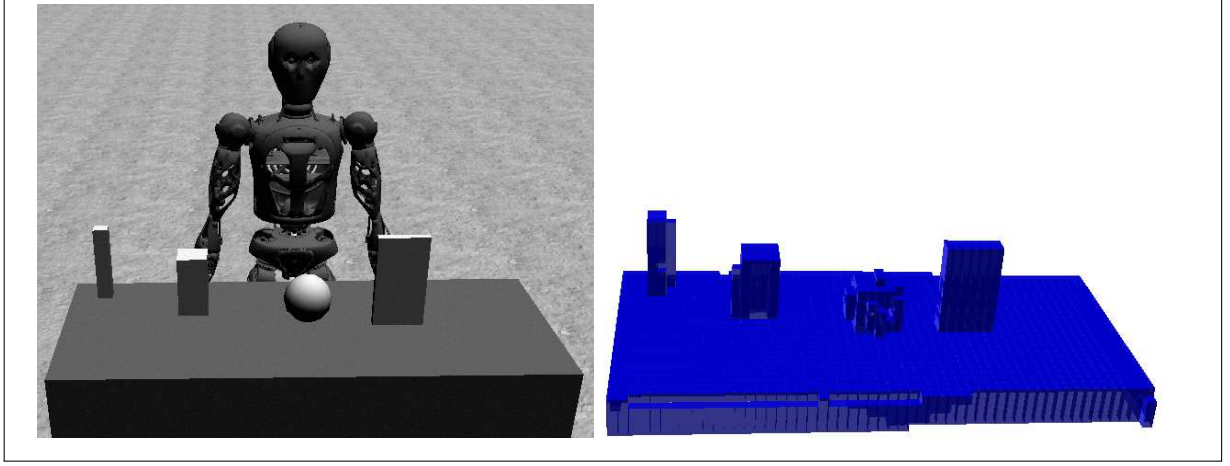


Fig. 3 Left: A planning scene in which a robot is standing in front of a table with four obstacles on it. Right: Environment information as a voxel grid.

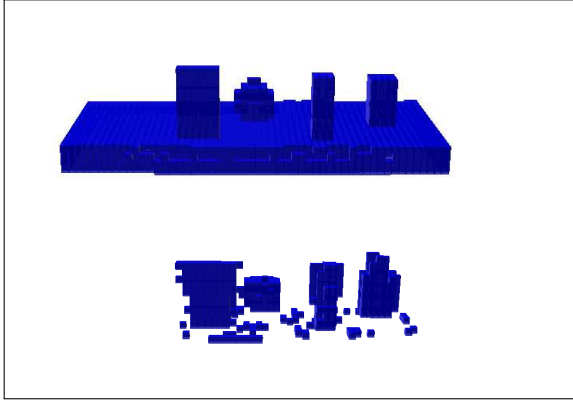


Fig. 4 Result of reconstructed voxel grid using PCA. The top voxel grid is the input and the below voxel grid is the reconstructed one.

4.2.2 Environment Features using Variational Autoencoder

Autoencoder is an unsupervised artificial neural technique to encode the input data into a lower dimension latent space and reconstruct the input data from the latent space. Similar to PCA, autoencoder is often used for noise reduction and dimensionality reduction. PCA performs a linear transformation, whereas an autoencoder in combination with a non-linear activation function can model a non-linear function. Autoencoder consists of two neural network structures: encoder and decoder. An encoder compresses the input features E to latent space $\hat{e}$ which is lower in dimension compared to the input features E . The decoder decompresses the latent space $\hat{e}$ to features $\hat{E}$

which should be similar to the input features E . Autoencoders perform well in data when there are correlations between the input features. The task of an encoder is to encode maximum important information from input features to a latent space $\hat{E}$ and the goal of the decoder is to decode the latent space and regenerate the input features by minimizing the reconstruction loss.

In recent years, different model variations of autoencoder architecture such as Variational Autoencoder (VAE) [18], Conditional Variational Autoencoder (CVAE), β Variational Autoencoder (β VAE), etc have been proposed. In our work, we use a VAE to reduce the dimensionality of the planning scene. VAE is a generative model with an architecture similar to an autoencoder. The key advantage of using a VAE is that the VAE offers

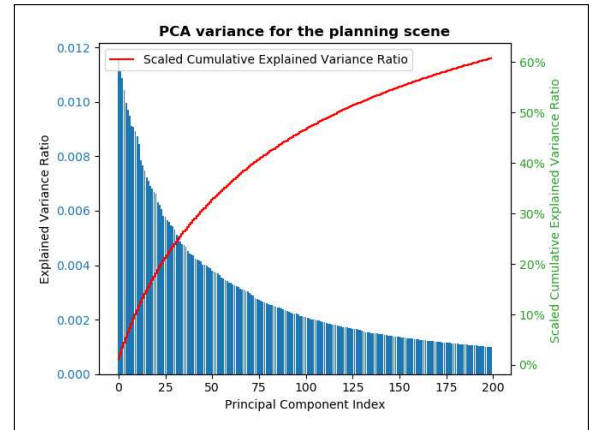


Fig. 5 PCA Explained Variance Ratio

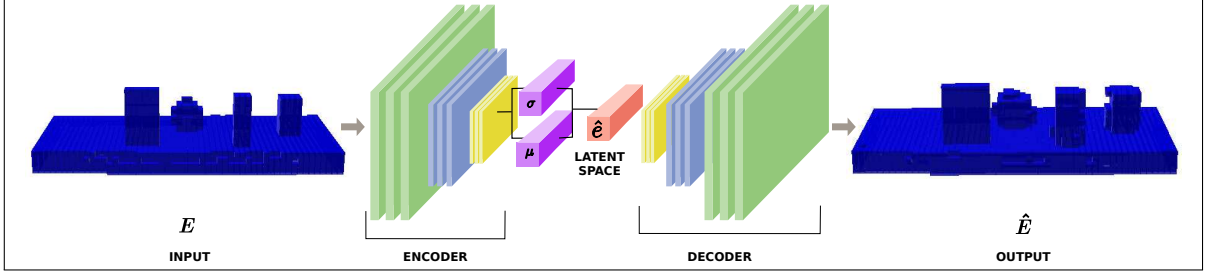


Fig. 6 VAE architecture

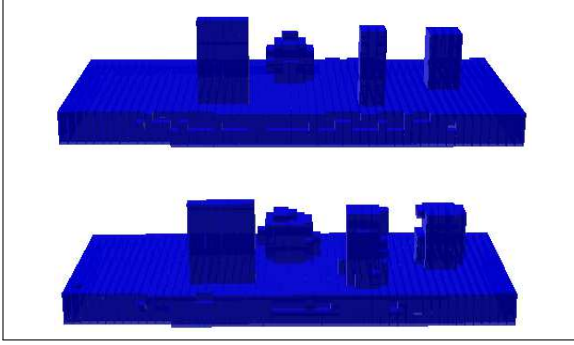


Fig. 7 Result of reconstructed voxel grid using VAE. The top voxel grid is the input and the below voxel grid is the reconstructed one.

a smooth latent space due to the Gaussian noise infused in its latent space.

Figure 6 illustrates the architecture of the VAE for learning the environment features. The encoder consists of three layers of 3D convolution kernels, which compress the input planning environment to a latent space of size 200. This latent space is then passed through a decoder comprising four layers of 3D convolution kernels to reconstruct the input planning environment. During the training phase, the VAE is regularised to avoid overfitting. The loss function of the VAE is calculated by combining two loss functions: the reconstruction loss function and the Kullback-Leibler (KL) divergence loss function. The reconstruction loss increases the likelihood of the reconstruction. The KL divergence loss makes sure that we penalize the learned distribution if it differs from the input distribution. We use a specialized form of Binary Cross-Entropy (BCE) proposed by Brock et.al [2] as reconstruction loss for the VAE model.

$$R_L = -\gamma E \log(\hat{E}) - (1 - \gamma)(1 - E) \log(1 - \hat{E})$$

Where γ is a hyperparameter to weigh the false positive against false negatives. E is the input

voxel and $\hat{E}$ is the reconstructed voxel. For training a VAE model, environment information similar to the one shown in Figure 3 is used. Altogether 4000 planning scenes are collected. The planning scenes are the same as the one used in evaluating the PCA model. The VAE model reduces the voxel grid of size $33 \times 67 \times 32$ to 200 features. Figure 7 shows the reconstructed voxel grid from 200 features using a VAE model. Once the VAE model is learned for the planning scenes, the encoder part of the VAE model will be used for generating the environment features $\hat{e}$.

4.2.3 Environment Features using SSD

For predicting a collision free trajectory, knowledge about an obstacle distribution around the manipulator is crucial. Obstacles situated far from the manipulator’s workspace are not relevant in the determination of a collision-free trajectory. Pan et.al [27] propose the spatial signed distance (SSD) vectors to efficiently embed collision information locally around a robot trajectory. The SSD vectors hold the signed distance information between the environment and robot collision links in a more compact form. Another key advantage of the SSD vectors is their size. The size of the SSD vectors is independent of the sensor data size. This plays a significant role in the learning procedure, as it takes fewer features to represent the obstacle distribution. Instead of using the entire environment information, the environment features are extracted only around the initial trajectory using the SSD features.

The collision body is modeled as a sphere, which is discretized into equal surface area pixels. Each surface area pixel will hold two attributes: a surface normal vector b and a signed distance sd value. The signed distance corresponds to the nearest obstacle signed distance in the direction of

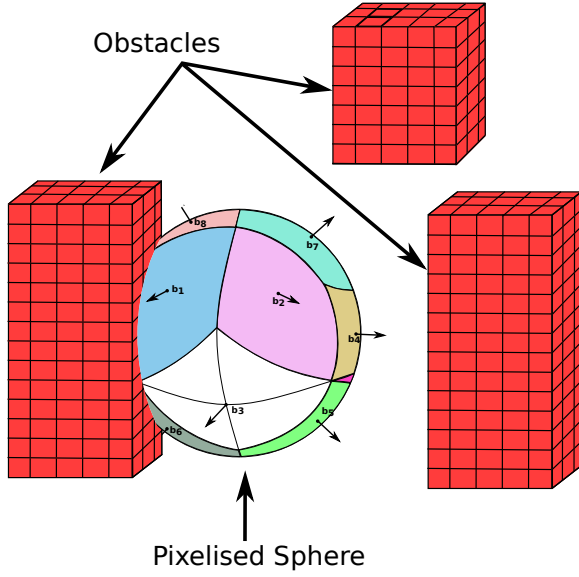


Fig. 8 A simple example to demonstrate SSD sphere.

its surface normal vector. Figure 8 shows a simple example of a single spherical collision body modeled as a SSD sphere. The SSD sphere’s pixels are initialized with the sphere radius as their default signed distance value. In the example scenario shown in Figure 8, the pixels 1, 6, and 8 are in collision with an obstacle. The pixels that are in collision with the obstacle will be updated with the penetration depth as its signed distance information. This information plays a significant role in defining in which direction and how far a robot can move to avoid a collision. In the case of a robot manipulator, each robot link is modeled as a SSD sphere. The diameter of the SSD sphere is equal to the robot link length.

Figure 9 shows a visualization of SSD spheres created for a manipulator. In this work, we discretize the sphere into 12 equal pixels, and two types of SSD features: ssd_1 and ssd_2 are collected. The number of pixels is arbitrarily chosen. The ssd_1 features embed the obstacle distribution around the given initial trajectory. ssd_2 will tell how the ssd_1 features change along the initial trajectory.

Given the robot URDF model R_{urdf} , planning scene E , and initial trajectory X , the collision features ssd_1 are generated according to the steps outlined in Algorithm 1. First, the collision models $C = \{c_1, c_2, \dots, c_m\}$ of the manipulator links are extracted from the provided URDF model. The

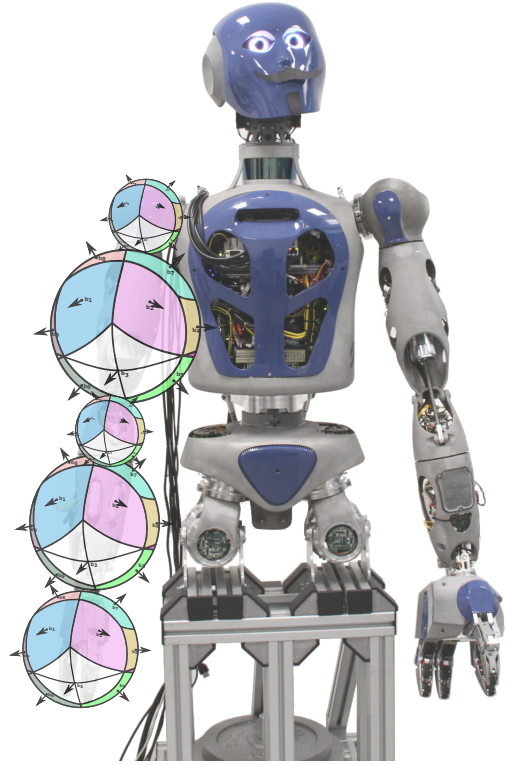


Fig. 9 Pixelised spheres for the right arm of the robot Manus.

following procedure is followed to calculate the ssd_1 features for each waypoint x_i of the given initial trajectory $X = \{x_1, x_2, \dots, x_n\}$. SSD spheres are created based on the number of collision links C in the manipulator. The diameter of each SSD sphere matches the corresponding collision link’s length of the manipulator. Each pixel of the SSD sphere is initialized with the radius of the SSD sphere. At each waypoint x_i , the poses of the SSD spheres are updated and the signed distance and its corresponding normal vector to the nearest collision objects are calculated. The signed distance indicates how far the nearest collision object is and, in the case of a collision, how deeply the obstacle has penetrated the SSD sphere. The normal vector will show the direction in which the signed distance is measured. As the SSD sphere is discretized into pixels, the nearest pixel where this signed distance information is collected should be identified. This is done, by taking the dot product between the normal vector of the signed distance and all the normal vectors of a SSD sphere’s pixel. The one with the largest dot product value is the

nearest pixel to the normal vector of the signed distance. The SSD sphere’s pixel value is now updated as: If the signed distance value is negative and the pixel value is negative, a collision has already occurred in this region. To penalize the movement of the robot in this direction, the pixel value is increased by the penetration depth. In other cases, the pixel is updated by taking the minimum value between the current pixel value and the signed distance value. The above mentioned calculation is performed for all the SSD spheres of the manipulator along the waypoints. The resulting pixels are concatenated to form the ssd_1 features. For an initial trajectory of n waypoints, with m collision links and l pixels, the size of ssd_1 features will be $(n \times m \times l)$. As you can see, the SSD features offer a more compact obstacle distribution independent of the size of the planning environment E . In addition to the ssd_1 features, an extra feature ssd_2 is computed to capture the changes in the ssd_1 over the waypoints. This is achieved by taking a dot product between the ssd_1 features along the trajectory waypoints and along the adjacent collision links.

Algorithm 1: SSD Feature: ssd_1

Input: Robot URDF model R_{urdf} ,
planning scene as voxel grid E ,
initial trajectory
 $X = \{x_1, x_2, \dots, x_n\}$
Output: Collision feature $ssd_{t \times m \times n}$

```

1  $C = \{c_1, c_2, \dots, c_m\} =$ 
  GetCollisionLinks( $R_{urdf}$ );
2 for  $n := 1$  to  $length(X)$  do
3    $ssd = \text{InitializeSSDSpheres}(C)$ ;
4    $\text{UpdateSSDSphere}(x_n, ssd)$ ;
5    $sd = \text{GetSignedDistance}(E, ssd)$ ;
6   for  $m := 1$  to  $length(ssd)$  do
7      $k = \text{NearestSSDPixel}(ssd(m), sd(m))$ ;
8     if  $sd(m) < 0$  AND  $ssd(m_k) < 0$ 
9       then
10       $ssd(m_k) \leftarrow ssd(m_k) + sd(m)$ ;
11    else
12       $ssd(m_k) \leftarrow$ 
13       $\min(ssd(m_k), sd(m))$ ;
14    end
15  end
16 end

```

5 experiments

5.1 Implementation

The ITPNet model using different task and environment features are evaluated on a 22 DOF upper torso humanoid system “RH5 Manus” [1]. RH5 Manus has two 7 DOF manipulators with a two-finger gripper attached to its left manipulator and a three-finger gripper attached to its right manipulator. A 3 DOF torso connects a fixed platform and the two manipulators. For evaluating the ITPNet, we use the 7 DOF right manipulator along with the 3 DOF torso. Altogether the ITPNet predicts an initial trajectory for the 10 joints of the RH5 Manus system. The ITPNet models are implemented in C++ using the libtorch framework [29], a C++ frontend of the pytorch. For training the model, we use an Intel Core i9-9820X CPU with 128GB RAM and Nvidia GeForce RTX 2080 Ti graphic card. For evaluation, a stochastic optimization-based motion planner (STOMP [13]) is used. STOMP uses by default a linearly interpolated joint space trajectory between the given start and goal state as its default initial trajectory. During the evaluation, instead of using the default linear interpolated initial trajectory, the planner will be warm started with the predicted initial trajectories. For pixelization of the SSD spheres, the Hierarchical Hierarchical Equal Area isoLatitude Pixelization (HEALPIX) [7] framework is used. The Flexible Collision Library (FCL) [26] is used for both calculating signed distance in SSD features extraction and for performing collision checking during motion planning.

5.2 Data Collection

A simulation framework [19] is used to collect the data for training the ITPNet model. Figure 3 shows the RH5 Manus robot and the planning environment in the simulation. The RH5 Manus robot is placed in front of a table where four different primitives shaped collision objects are randomly distributed on the table. To capture the planning environment, five depth image cameras are used. One camera is placed on the RH5 Manus torso, and the other four cameras are strategically placed around the table. These cameras are optimally positioned to collect the depth information of all the obstacles including the tabletop in

Model Name	Task Feat.		Env. Feat.		Overall Feat. Size
	Type	Size	Type	Size	
ITPNet_JS_PCA	JS	20	PCA	200	62x220
ITPNet_JS_VAE	JS	20	VAE	200	62x220
ITPNet_JS_SSD	JS	20	SSD	385	62x405
ITPNet_CS_PCA	CS	9	PCA	200	62x209
ITPNet_CS_VAE	CS	9	VAE	200	62x209
ITPNet_CS_SSD	CS	9	SSD	385	62x394

Table 1 ITPNet Models

the planning environment. A total of 4000 planning scenes are simulated, and their corresponding environment information is saved in the form of a voxel grid [8]. In each planning scene, the obstacles are randomly placed on the table. To avoid obstacles clustering together, a minimum distance of 20 cm is maintained between the obstacles. Figure 3 illustrates a sample planning scene. Random start and goal poses are generated for the right arm of the RH5 Manus in each planning scene. The start and goal poses should have a minimum Euclidean distance of 30 cm to avoid a simple planning problem. Given a randomly generated planning environment, start, and goal poses, the STOMP planner is used to find a collision free optimal trajectory for this planning problem. The planning problem is considered valid, only if it satisfies the following criteria:

- The start and goal poses should be collision free.
- Number of iterations to find an optimal solution should be more than 10 iterations. This condition makes sure that the planning problem is complex enough.

In total, 20 valid planning problems are collected for each planning scene. This results in a dataset of 80000 instances of the planning problem used for training the ITPNet model.

5.3 ITPNet Models

To evaluate the different task and environment features, six different ITPNet models are trained. Table 1 shows an overview of these models. The abbreviations JS and CS in the table denote joint space and Cartesian space task features, respectively. Joint space task features consist of 20 features, representing the 10 joints of the RH5

Manus right arm. Cartesian space task features encompass 9 features, with the first 3 features representing the position and the next 6 features representing the orientation as mentioned in the subsection 4.1.2. The collision features for PCA and VAE are of size 200, while the collision features for SSD have size 385. The dimensions for PCA and VAE collision features are chosen empirically. The SSD features are passed through a 2-layer multilayer perceptron (MLP) to reduce their size before feeding them to the LSTM layers. All the models are trained for a trajectory consisting of arbitrarily chosen 60 waypoints.

6 Results

This section describes the evaluation results of the different ITPNet models in two different scenarios. The planning scenes in the first scenario are similar to the trained planning scenes. The RH5 Manus robot placed in front of a table with obstacles are distributed randomly on the table. In the second scenario, an unseen planning scene is used. Here, the obstacles are located over the table. For benchmarking, the proposed trajectory prediction models, we use the optimization-based motion planner STOMP, which is warm-started with different initial trajectories. By default, a linearly interpolated joint space trajectory between the start and goal configuration is used. Such a naive default initial trajectory is commonly used in many optimization-based motion planners and often converges to a locally optimal solution. These default initial trajectories are compared with the predicted initial trajectories using the ITPNET models using different task and environment features.

6.1 Similar Planning Environment

In this scenario, four primitive-shaped obstacles are randomly distributed over the table. The obstacles are similar to the ones used during the training of the models. For this evaluation, 25 different planning scenes are generated in simulation. For each of these planning scenes, 20 random start and goal poses are generated. Altogether, a total of 500 planning problems are collected. The only criterion for the selection of the start and goal poses is that they should not be in collision with

Warm starting opt. based motion planner using	Success	Planning Time (s)		No. of Iterations	
		Avg.	SD	Avg.	SD
Default initial trajectory	22.2%	15.6	5.6	9.6	3.4
ITPNet_JS_PCA	47.0%	9.8	6.4	6.0	3.9
ITPNet_JS_VAE	67.0%	8.2	6.1	5.0	3.7
ITPNet_JS_SSD	72.0%	7.9	5.9	4.8	3.6
ITPNet_CS_PCA	47.0%	8.7	6.3	5.3	3.8
ITPNet_CS_VAE	81.8%	5.6	4.3	3.4	2.6
ITPNet_CS_SSD	90.6%	6.1	4.5	3.7	2.8

Table 2 Similar planning environment benchmark result: success rate, planning time and number of iterations for the seven different initial trajectories.

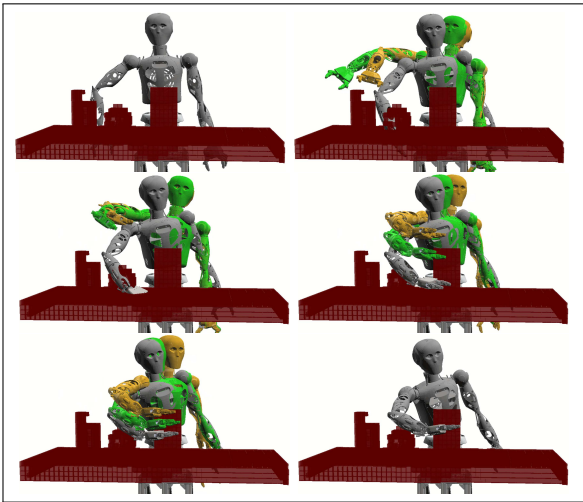


Fig. 10 Similar planning environment result: default initial trajectory(grey), predicted initial trajectory(amber), optimal trajectory(green).

the robot. Table 2 shows the result of this experiment. For the 500 planning problems, by warm starting the motion planner using the default initial trajectory, 22.2 % of the planning problems were solved successfully. The default initial trajectories take an average of 15.6 seconds to find an optimal solution using an average of 9.6 iterations. Warm starting the motion planner using the predicted trajectories from the ITPNet model ITPNet_CS_SSD, which utilizes Cartesian space task features and the SSD-based environment features yields good results compared to all the other combinations of task space and environment features. ITPNet models using PCA as environment features show similar results independent of using

joint or Cartesian space task features. The VAE and SSD environment features based models show better results when combined with the Cartesian task space. In this benchmark, the models using Cartesian space task features show better performance than those using joint space task features. The reason is that the joint space task feature has a higher dimensionality (20) compared to the Cartesian space task features. The ITPNet using the SSD features as its collision features show better performance when compared to using the collision features from PCA or VAE.

6.2 Unseen Planning Environment

In this unseen planning environment benchmark, a planning scene as shown in Figure 11 is used. This planning environment is unknown to any of the ITPNet models. Here 9 obstacles are placed

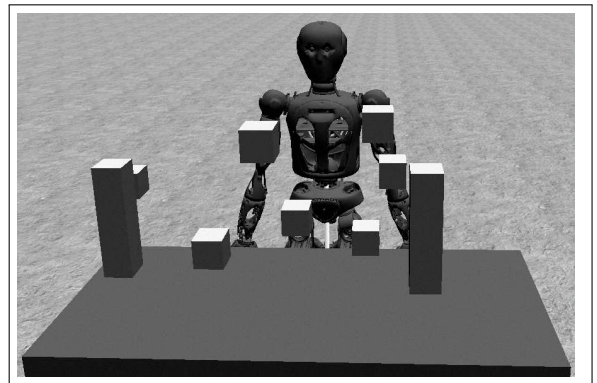


Fig. 11 Unseen planning environment: Obstacles are located above the table

Warm starting opt. based motion planner using	Success	Planning Time (s)		No. of Iterations	
		Avg.	SD	Avg.	SD
Default initial trajectory	43.0%	14.7	5.2	9.3	3.3
ITPNet_JS_SSD	58.0%	11.7	6.4	7.3	4.0
ITPNet_CS_SSD	80.0%	7.7	5.3	4.9	3.4

Table 3 Unseen planning environment benchmark result: success rate, planning time and number of iterations for the three different initial trajectories.

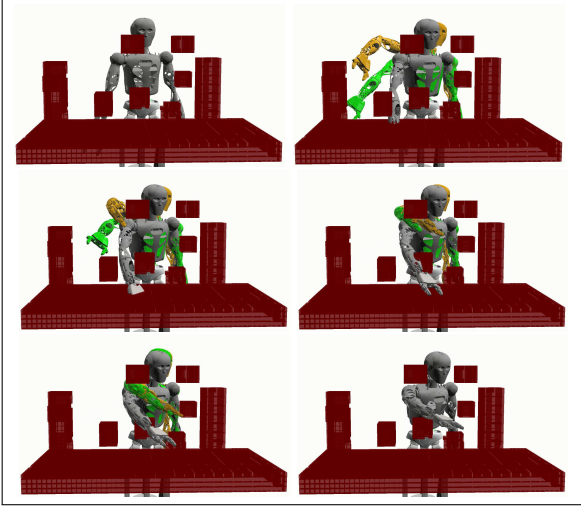


Fig. 12 Unseen planning environment result: default initial trajectory(grey), predicted initial trajectory(amber), optimal trajectory(green).

randomly in front of the robot. All the obstacles are floating over the table. For this planning scene, 100 random start and goal poses are generated. A total of 100 planning problems are collected for this benchmark. Similar to the previous benchmark, only collision free start and goal poses are taken into account. The choice of this scenario is to evaluate the performance of the ITPNet models (ITPNet_JS_SSD and ITPNet_CS_SSD) on the out-of-distribution planning environment. The ITPNet models utilizing PCA and VAE for environment features are excluded from this benchmark, as the planning scene falls outside the distribution they were trained on. Table 3 shows the result of this benchmark. Using the default initial trajectories, 43% of the planning problem was successfully solved. The default initial trajectories take an average of 14.7 seconds to find an optimal solution using an average of 9.3 iterations. For the ITPNet model utilizing

joint space as task features and SSD as environment features, a success rate of 58% was achieved. The average planning time for this model is 11.7 seconds, with an average of 7.3 iterations. Similar to the previous benchmark, the ITPNet model using the Cartesian space as task features and SSD as environment features achieves a higher success rate of 80%, taking an average of 7.7 seconds for planning time and using an average of 7.3 iterations. This outcome is consistent with the findings in the previous benchmark.

7 Discussion

In the previous section, the performance of the ITPNet models predicted initial trajectories are compared with the default initial trajectories in two different scenarios. Warm starting the motion planner using ITPNet predicted trajectory enhances the planner’s success rate and it requires fewer iterations compared to using the default initial trajectory. The performance of different ITPNet models on those different benchmark scenarios has shown different characteristics. When the planning environment is similar to the trained planning environment, ITPNet models using Cartesian space task features outperform the joint space task features. This is due to the number of features, the model needs to predict. In the case of Cartesian space as task features, the model predicts only 9 features that represent the end-effector pose. But in the case of joint space task features models, it needs to predict 20 features. Notably, ITPNet models using SSD environment features outperform the other two types of approaches in both benchmark scenarios. This is due to the fact that the SSD environment features encapsulate obstacles distribution around a given initial trajectory. The model trained with

SSD features learns to navigate around obstacles in close proximity. However, it is important to note that since SSD environment features do not capture obstacle distributions far from the initial trajectory, the model’s capacity to guarantee a globally optimal trajectory is limited. Using PCA and VAE to obtain the environment features is more convenient, but the ITPNet models employing those environment features are limited to predicting initial trajectories for planning environments within the distribution they were trained on.

8 Conclusion

Warm starting an optimization-based motion planner with a good initial trajectory is crucial in converging to an optimal solution. In this work, we presented an initial trajectory prediction framework, which predicts a good initial trajectory based on the planning environment to warm start an optimization-based motion planner. The network efficiently learns the mapping between the task and environment features to the corresponding optimal robot path. Two different types of task features: Cartesian and joint space and three different types of environment features: PCA, VAE, and SSD are analyzed and evaluated in two different planning environments using an upper-torso humanoid system. The ITPNet using the combination of Cartesian space task features and SSD based environment features offers better performance when compared to other ITPNet models. Using SSD environment features helps the models to predict good initial trajectories even in an unseen planning environment.

Acknowledgments. This work was performed as part of the project TransFIT funded by the German Aerospace Center (DLR) with federal funds from the Federal Ministry for Economic Affairs and Climate Action (BMWK) (Grant Nos. FKZ 50RA1701, FKZ 50RA1702 and FKZ 50RA1703).

References

- [1] Boukheddimi M, Kumar S, Peters H, et al (2022) Introducing rh5 manus: A powerful humanoid upper body design for dynamic movements. In: IEEE International Conference on Robotics and Automation (ICRA)
- [2] Brock A, Lim T, Ritchie JM, et al (2016) Generative and discriminative voxel modeling with convolutional neural networks. CoRR abs/1608.04236. URL <http://arxiv.org/abs/1608.04236>, 1608.04236
- [3] Cho K, van Merriënboer B, Gülçehre Ç, et al (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar. ACL, pp 1724–1734
- [4] Dey D, Liu TY, Hebert M, et al (2012) Contextual sequence prediction with application to control library optimization. In: Roy N, Newman P, Srinivasa SS (eds) Robotics: Science and Systems VIII, University of Sydney, Sydney, NSW, Australia, July 9-13, 2012, <https://doi.org/10.15607/RSS.2012.VIII.007>, URL <http://www.roboticsproceedings.org/rss08/p07.html>
- [5] Dragan AD, Gordon GJ, Srinivasa SS (2011) Learning from experience in manipulation planning: Setting the right goals. In: Robotics Research - The 15th International Symposium ISRR, 9-12 December 2011, Flagstaff, Arizona, USA, Springer Tracts in Advanced Robotics, vol 100. Springer, pp 309–326, https://doi.org/10.1007/978-3-319-29363-9_18, URL https://doi.org/10.1007/978-3-319-29363-9_18
- [6] Gammell J, Srinivasa S, Barfoot T (2015) Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. Proceedings - IEEE International Conference on Robotics and Automation 2015:3067–3074. <https://doi.org/10.1109/ICRA.2015.7139620>
- [7] Górski KM, Hivon E, Banday AJ, et al (2005) HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. The Astrophysical Journal 622(2):759–771. <https://doi.org/10.1086/4267660>

- 1086/427976, arXiv:astro-ph/0409513 [astro-ph]
- [8] Hornung A, Wurm KM, Bennewitz M, et al (2013) OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots* <https://doi.org/10.1007/s10514-012-9321-0>, URL <http://octomap.github.com>, software available at <http://octomap.github.com>
 - [9] Jetchev N, Toussaint M (2009) Trajectory prediction: Learning to map situations to robot trajectories. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, New York, NY, USA, ICML '09, pp 449–456, <https://doi.org/10.1145/1553374.1553433>
 - [10] Jetchev N, Toussaint M (2010) Trajectory prediction in cluttered voxel environments. In: *2010 IEEE International Conference on Robotics and Automation*, pp 2523–2528
 - [11] Jetchev N, Toussaint M (2013) Fast motion planning from experience: Trajectory prediction for speeding up movement generation. *Autonomous Robots* 34:111–127
 - [12] Jetchev NN (2012) Learning representations from motion trajectories: Analysis and applications to robot planning and control. Dissertation, Freie Universitaet Berlin, Berlin, URL <http://dx.doi.org/10.17169/refubium-13239>
 - [13] Kalakrishnan M, Chitta S, Theodorou E, et al (2011) Stomp: Stochastic trajectory optimization for motion planning. In: *International Conference on Robotics and Automation*
 - [14] Kalchbrenner N, Blunsom P (2013) Recurrent continuous translation models. *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* 3:1700–1709
 - [15] Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7):846–894. <https://doi.org/10.1177/0278364911406761>
 - [16] Kavraki LE, Svestka P, Latombe J, et al (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580. <https://doi.org/10.1109/70.508439>
 - [17] Khosroshahi A, Ohn-Bar E, Trivedi MM (2016) Surround vehicles trajectory analysis with recurrent neural networks. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp 2267–2272, <https://doi.org/10.1109/ITSC.2016.7795922>
 - [18] Kingma DP, Welling M (2014) Auto-Encoding Variational Bayes. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014*, <http://arxiv.org/abs/1312.6114v10>
 - [19] Langosz M, Rohn M, von Szadkowski KA (????) Mars (machina arte robotum simulans): A cross-platform simulation and visualisation tool. <https://github.com/rock-simulation/mars/wiki>, accessed: 2020-04-15
 - [20] LaValle S (1998) Rapidly-exploring random trees : a new tool for path planning. *The annual research report*
 - [21] LaValle SM (2006) *Planning Algorithms*. Cambridge University Press, New York, NY, USA
 - [22] Lembono TS, Paolillo A, Pignat E, et al (2020) Memory of motion for warm-starting trajectory optimization. *IEEE Robotics and Automation Letters* 5(2):2594–2601
 - [23] Mansard N, Del Prete A, Geisert M, et al (2018) Using a memory of motion to efficiently warm-start a nonlinear predictive controller. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, <https://doi.org/10.1109/ICRA.2018.8463154>, URL <https://hal.archives-ouvertes.fr/hal-01591373>

- [24] Merkt W, Ivan V, Vijayakumar S (2018) Leveraging precomputation with problem encoding for warm-starting trajectory optimization in complex environments. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 5877–5884
- [25] Natarajan S (2021) Learning initial trajectory using sequence-to-sequence approach to warm start an optimization-based motion planner. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 9430–9436, <https://doi.org/10.1109/IROS51168.2021.9636456>
- [26] Pan J, Chitta S, Manocha D (2012) Fcl: A general purpose library for collision and proximity queries. In: 2012 IEEE International Conference on Robotics and Automation, pp 3859–3866, <https://doi.org/10.1109/ICRA.2012.6225337>
- [27] Pan J, Chen Z, Abbeel P (2014) Predicting initialization effectiveness for trajectory optimization. In: Proceedings - IEEE International Conference on Robotics and Automation, pp 5183–5190, <https://doi.org/10.1109/ICRA.2014.6907620>
- [28] Park SH, Kim B, Kang CM, et al (2018) Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp 1672–1678
- [29] Paszke A, Gross S, Massa F, et al (2019) Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., p 8024–8035
- [30] Ratliff N, Zucker M, Bagnell JA, et al (2009) Chomp: Gradient optimization techniques for efficient motion planning. In: 2009 IEEE International Conference on Robotics and Automation, pp 489–494, <https://doi.org/10.1109/ROBOT.2009.5152817>
- [31] Schulman J, Duan Y, Ho J, et al (2014) Motion planning with sequential convex optimization and convex collision checking. The International Journal of Robotics Research 33:1251–1270. <https://doi.org/10.1177/0278364914528132>
- [32] Shyam R, Lightbody P, Das G, et al (2019) Improving local trajectory optimisation using probabilistic movement primitives. pp 2666–2671, <https://doi.org/10.1109/IROS40897.2019.8967980>
- [33] Stolle M, Atkeson CG (2006) Policies based on trajectory libraries. In: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pp 3344–3349
- [34] Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. MIT Press, Cambridge, MA, USA, NIPS’14, p 3104–3112
- [35] Tang G, Sun W, Hauser K (2018) Learning trajectories for real-time optimal control of quadrotors. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 3620–3625, <https://doi.org/10.1109/IROS.2018.8593536>
- [36] Venugopalan S, Rohrbach M, Donahue J, et al (2015) Sequence to sequence – video to text. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp 4534–4542
- [37] Zhou Y, Barnes C, Jingwan L, et al (2019) On the continuity of rotation representations in neural networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)