SE3-Pose-Nets: Structured Deep Dynamics Models for Visuomotor Control

Arunkumar Byravan[1], Felix Leeb[1], Franziska Meier[1,2] and Dieter Fox[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Autonomous Motion Department, MPI-IS, Tübingen, Germany

***Abstract*— In this work, we present an approach to deep visuomotor control using structured deep dynamics models. Our model, a variant of SE3-Nets, learns a low-dimensional pose embedding for visuomotor control via an encoder-decoder structure. Unlike prior work, our model is structured: given an input scene, our network explicitly learns to segment salient parts and predict their pose embedding and motion, modeled as a change in the pose due to the applied actions. We train our model using a pair of point clouds separated by an action and show that given supervision only through point-wise data associations between the frames our network is able to learn a meaningful segmentation of the scene along with consistent poses. We further show that our model can be used for closed-loop control directly in the learned low-dimensional pose space, where the actions are computed by minimizing pose error using gradient-based methods, similar to traditional model-based control. We present results on controlling a Baxter robot from raw depth data in simulation and RGBD data in the real world and compare against two baseline deep networks. We also test the robustness and generalization performance of our controller under changes in camera pose, lighting, occlusion, and motion. Our method is robust, runs in real-time, achieves good prediction of scene dynamics, and outperforms baselines on multiple control runs. Video results can be found at: https://rse-lab.cs.washington.edu/se3-structured-deep-ctrl/**

I. Introduction

Imagine we are receiving observations of a scene from a camera and we would like to control our robot to reach a target scene. Traditional approaches to visual servoing [1] decompose this problem into two parts: data-associating the current scene to the target (usually through the use of features) and modeling the effect of applied actions to changes to the scene, combining these in a tight loop to servo to the target. Recent work on deep learning has looked at learning similar predictive models directly in the space of observations, relating changes in pixels or 3D points directly to the applied actions [2]–[4]. Given a target scene, we can use this predictive model to generate suitable controls to visually servo to the target using model-predictive control [5]. Unfortunately, for this pipeline to work, we need an external system (such as [6], [7]) capable of providing long range data-associations to measure progress.

As we showed in prior work [4], instead of reasoning about raw pixels, we can predict scene dynamics by decomposing the scene into objects and predicting object dynamics instead. While this significantly improves prediction results, it still does not provide a clear solution to the data association problem that we encounter during control - we still lack the capability to explicitly associate objects/parts across scenes. We observe three key points: 1) We can data-associate across scenes by learning to predict a "pose" representation of detected objects/parts in the scene (the pose implicitly provides tracking), 2) We can model the dynamics of an object directly in the learned low-dimensional pose space, and 3) We can predict scene dynamics by combining the dynamics predictions of each detected part.

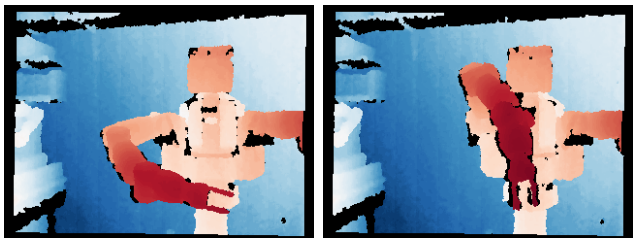

Fig. 1: An example scenario showing the initial (left) and target depth clouds (right). SE3-Pose-Nets can be used to control the robot to reach the target state based on raw depth (and optionally, color) data. Depth images colorized for display purposes only.

We combine these ideas in this work to propose SE3-Pose-Nets, a deep network architecture for efficient visuomotor control that jointly learns to data-associate across long term sequences. We make the following contributions:

- We show how to learn predictive models that detect parts of the scene and jointly learn a consistent pose representation for these parts with minimal supervision.
- We demonstrate how a deep predictive model can be used for reactive visuomotor control using simple gradient backpropagation and a more sophisticated Gauss-Newton optimization, reminiscent of approaches in inverse kinematics [8].
- We present results on real-time reactive control of a Baxter arm using raw depth/color images and velocity control, both in simulation and on real data.
- Finally, we present results on testing the robustness and generalization of the real-world controller under various changes to imaging conditions.

Fig 1 shows an example scenario where our proposed method can be applied to control the robot to reach the target state (right) from the initial state (left).

II. Related work

Modeling scenes and dynamics: Our work builds on top of prior work on learning structured models of scene dynamics [4]. Unlike SE3-Nets, we now explicitly model data associations through a low-dimensional pose embedding that we train to be consistent across long sequences. Similar to Boots et al. [2], our model learns to predict point clouds

based on applied actions, but through a structured intermediate representation that reasons about objects and their motions. Unlike Finn et al. [3], we also use depth data and reason about 3D motion using masks and $\mathbb{SE}(3)$ transforms while using point-wise data association as supervision.

Visuomotor control: Recently, there has been a lot of work on visuomotor control, primarily through the use of deep networks [5], [9]–[13]. These methods either directly regress to controls from visual data [10], [11], generate controls by planning on learned forward dynamics models [5], [9], through inverse dynamics models [12] or by reinforcement learning [13]. Like some of these methods, we generate controls by planning with a learned dynamics model, albeit in a learned low-dimensional latent space. Specifically, work by Finn et al. [5] is closely related, but differs in two main ways: unlike their approach which controls in the observation space through sampled actions (at $\approx$ 5Hz), our controller runs gradient optimization on a learned low-dimensional pose embedding in real-time (> 30 Hz). Also, their approach requires an external tracker to measure progress while we explicitly learn to data-associate across large motions.

Our work borrows several ideas from prior work by Watter et al. [9] which learns a latent low-dimensional embedding for fast reactive control from pairs of images related by an action. Unlike their work, we use a structured latent representation (object poses), predict object masks, and use a physically grounded 3D loss that only models changes in observations as opposed to a restrictive image reconstruction loss. Lastly, our losses are physically motivated, similar to those proposed for training position-velocity encoders [13], but our learned pose embedding is significantly more structured and we train our networks end-to-end directly for control.

Data association: Related work in the computer vision literature has looked at tackling the data association problem, primarily by matching visual descriptors, either hand-tuned [14], or more recently, learned using deep networks [15], [16]. In prior work, Schmidt et al. [15] learn robust visual descriptors for long-range associations using correspondences over short training sequences. We however only use correspondences between pairs of frames to learn a consistent pose space that lets us data-associate across long sequences.

Pose estimation: There has also been a lot of recent work on object and camera pose estimation from RGB/D data using learning methods [17]–[19], most of which use some form of explicit supervision. Unlike these methods, our method learns a low-dimensional pose representation purely through self-supervised consistency losses – this resulting pose space can be used for robust control but does not directly correspond to the canonical pose of objects in the scene.

Visual servoing: Finally, there have been multiple approaches to visual servoing over the years [1], [20], [21], including some newer methods that use deep learned features and reinforcement learning [22]. While these methods depend on an external system for data association or on pre-specified features, our system is trained end-to-end and can control directly from raw visual data.

III. SE3-POSE-NETS

Our deep dynamics model SE3-POSE-NETS decomposes the problem of modeling scene dynamics into three sub-problems: a) modeling scene structure by identifying parts of the scene that move distinctly and by encoding their latent state as a 6D pose, b) modeling the dynamics of individual parts under the effect of the applied actions as a change in the latent pose space (parameterized as an $\mathbb{SE}(3)$ transform), and finally c) combining these local pose changes to model the dynamics of the entire scene. Each sub-problem is modeled by a separate component of the SE3-POSE-NET:

- **Modeling scene structure:** An ***encoder*** (h_{enc}) that decomposes the input point cloud ($\mathbf{x}$[1]) into a set of K rigid parts, predicting per part a 6D pose (p^k, $k = 1 \dots K$) and a dense segmentation mask (m^k) that highlights points belonging to that part
- **Modeling part dynamics:** A ***pose transition*** network (h_{trans}) that models dynamics in the pose space, taking in the current poses ($\mathbf{p}_t$) and action ($\mathbf{u}_t$) to predict the change in poses ($\Delta\mathbf{p}_t$)
- **Predicting scene dynamics:** A ***transform*** layer (h_{tfm}) that generates the next point cloud ($\hat{\mathbf{x}}_{t+1}$) given the current point cloud ($\mathbf{x}_t$), predicted object masks ($\mathbf{m}_t$) and the predicted pose deltas ($\Delta\mathbf{p}_t$) by explicitly applying 3D rigid body $\mathbb{SE}(3)$ transforms on the input point cloud.

Fig. 2 shows the network architecture of the SE3-POSE-NET. Next, we present the details of the three sub-components and outline a training procedure for training the SE3-POSE-NET end-to-end with minimal supervision.

A. Modeling scene structure

Given a 3D point cloud $\mathbf{x}$ from an RGBD sensor (represented as a 3-channel H x W image), the **encoder** (blue block in Fig. 2, top half) segments the scene into distinctly moving parts ($\mathbf{m}$) and predicts a 6D pose ($\mathbf{p}$) per part:

$$(\mathbf{p}, \mathbf{m}) = h_{enc}(\mathbf{x}) \tag{1}$$

The encoder has three components, the first is a convolutional network that generates a latent representation of the input point cloud ($\mathbf{x}$). This network has five convolutional layers, each followed by a max pooling layer. The latent representation is further used as input for the mask and pose predictions.

Object masks: We use a fully-convolutional network with five de-convolutional layers and a skip-add architecture (similar to prior work [4]) to predict a dense pixel-wise segmentation of the scene into its constituent parts ($\mathbf{m}$). The masks predicted by this network are at full resolution with K channels (K x H x W), where K is a pre-specified hyper-parameter that is greater than or equal to the number of moving parts in the scene (including background). The predicted segmentation mask learns to attend to parts of the scene that move together, representing areas of the scene that can move independently as different parts. As in prior work [4], we formalize mask prediction as soft-classification where

[1]Bold fonts denote collections of items

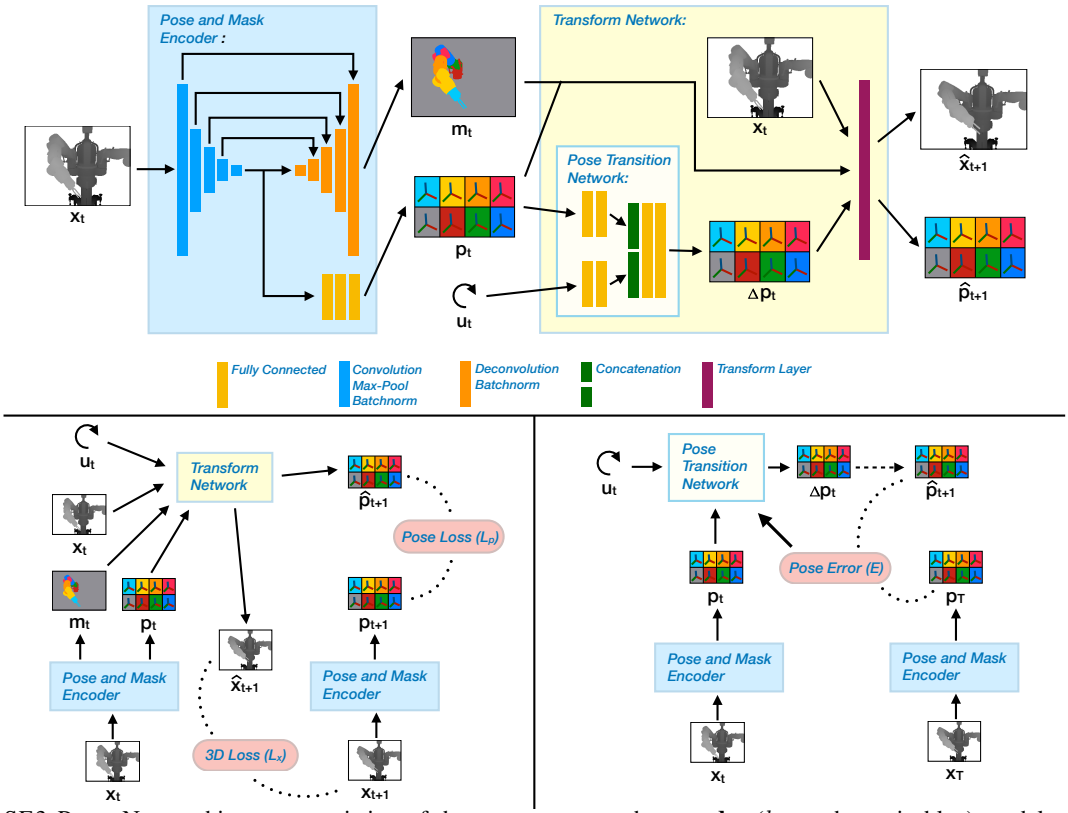

Fig. 2: ***Top:*** SE3-POSE-NET architecture consisting of three components: the **encoder** (h_{enc}, shown in blue) models scene structure (explained in Sec. III-A), the **pose transition** net (h_{trans}) models object/part dynamics (Sec. III-B) and the **transform layer** (h_{tfm}) which transforms point clouds based on the predicted object dynamics (Sec. III-C). ***Bottom Left:*** Schematic showing the training procedure for SE3-POSE-NET (Sec. III-D). ***Bottom Right:*** Procedure for closed loop control using the SE3-POSE-NET (Sec. IV)

the network outputs a k-length probability distribution which we sharpen to get a binary segmentation mask.

Object poses: Given the encoded latent representation, we use a three layer fully-connected network to predict the 6D pose p^k of each of the K segmented parts. We represent each pose by 6 numbers: a 3D position ($y \in \mathbb{R}^3$) and an orientation ($R \in \mathbb{SO}(3)$), represented as a 3-parameter axis-angle vector. As we show later, our pose network learns to predict consistent poses which can be used to data-associate observations over long sequences of motions.

At a high level, the **encoder** implicitly learns the structure of observed scenes by persistently identifying parts, and by predicting consistent poses for parts across multiple scenes.

B. Modeling part dynamics

Next, we reason about the effect of applied actions on the identified parts. We model this notion of "part dynamics" through a fully-connected **pose transition** network that takes predicted poses from the encoder ($\mathbf{p}$) and applied actions ($\mathbf{u}$) as input to predict the change in pose ($\Delta\mathbf{p}$) for all K segmented parts:

$$\Delta\mathbf{p} = h_{\text{trans}}(\mathbf{p}, \mathbf{u}) \tag{2}$$

where $\Delta\mathbf{p} = [\mathbf{R}, \mathbf{T}]$ is represented as an $\mathbb{SE}(3)$ transform per part (parameterized similar to the poses). Fig. 2 (top half) shows the architecture of the transition network. As we show later in Sec. IV we rely on good predictions of pose-deltas through the pose-transition network for efficient control.

C. Predicting scene dynamics

Finally, given the predicted segmentation ($\mathbf{m}_t$) and the change in poses ($\Delta\mathbf{p}_t$), we model the dynamics of the input scene ($\mathbf{x}_t$) due to the applied action ($\mathbf{u}_t$). We do this through the **Transform** layer (h_{tfm}) which applies the predicted rigid rotations ($\mathbf{R}_t$) and translations ($\mathbf{T}_t$) to the input depth cloud, weighted by the predicted mask probabilities ($\mathbf{m}_t$). We predict the transformed point cloud as:

$$\hat{x}^j_{t+1} = \sum_{k=1}^{K} m^{kj}_t \left(R^k_t \; x^j_t + T^k_t \right) \tag{3}$$

where $\hat{x}^j_{t+1}$ is the 3D output corresponding to input point x^j_t. In effect, we apply the kth rotation and translation ($\Delta p^k = [R^k, T^k]$) to all points x^j that belong to the corresponding object as indicated by the kth mask channel m^k (assuming binary masks) to predict the transformed points $\hat{x}^j$ for that object. Repeating this for all objects gives us the transformed output point cloud ($\hat{\mathbf{x}}$). Note that this part has no trainable parameters. For more details, please refer to prior work [4].

D. Training

We now outline a procedure to train the SE3-POSE-NET end-to-end, using supervision in the form of point-wise data associations between a pair of point clouds ($\mathbf{x}_t$, $\mathbf{x}_{t+1}$), related by an action ($\mathbf{u}_t$) i.e. for each input point (x^i_t), we know its corresponding point (x^i_{t+1}) if it is visible. No other supervision is given for learning the masks, poses, and the

change in poses. Fig. 2 (bottom left) shows a schematic of this procedure. Given two point clouds $\mathbf{x}_t, \mathbf{x}_{t+1}$, we use the **encoder** to predict the corresponding masks and poses:

$$\mathbf{p}_t, \mathbf{m}_t = h_{enc}(\mathbf{x}_t) \; ; \; \mathbf{p}_{t+1}, \mathbf{m}_{t+1} = h_{enc}(\mathbf{x}_{t+1}) \quad (4)$$

Next, the predicted pose ($\mathbf{p}_t$) and control ($\mathbf{u}_t$) are used as input to the **pose transition** net to predict the change in pose from t to $t+1$:

$$\Delta\mathbf{p}_t = h_{trans}(\mathbf{p}_t, \mathbf{u}_t) \quad (5)$$

Finally, we predict the next point cloud using the **transform** layer (3):

$$\hat{\mathbf{x}}_{t+1} = h_{tfm}(\mathbf{x}_t, \mathbf{m}_t, \Delta\mathbf{p}_t) \quad (6)$$

The predicted mask ($\mathbf{m}_{t+1}$) at time $t+1$ is discarded. We use two losses to train the entire pipeline end to end:

- A 3D loss (L_x) that penalizes the error between the predicted point cloud ($\hat{\mathbf{x}}_{t+1}$) and the data associated target point cloud ($\tilde{\mathbf{x}}_{t+1}$). We use a normalized version of the mean-squared error (MSE) that scales based on the target magnitude:

$$L_x = \frac{1}{N}\sum_{i=1}^{HW}\frac{(\hat{x}^i_{t+1} - \tilde{x}^i_{t+1})^2}{\alpha\tilde{f}^i + \beta} \quad (7)$$

  where ($\tilde{f}^i = \tilde{x}^i_{t+1} - x^i_t$) denotes the ground truth motion for point i relative to the input point cloud $\mathbf{x}_t$, N is the number of points that actually move between t and $t+1$ and α & β are hyper-parameters ($\alpha = 0.5, \beta = 1e-3$ in all our experiments). This loss is aimed to tackle two main issues with a standard MSE loss: a) By normalizing the loss by a separate scalar per dimension ($\tilde{f}^i$) that depends on the target magnitude we make the loss scale invariant, allowing us to treat equally parts that move less (such as the end-effector when only the wrist rotates) as those that have large motion (e.g. the elbow), and b) By dividing the total error by the number of points (N) that move in the scene, we treat scenes where very few points move equally as those where large parts move.
- A pose consistency loss (L_p) that encourages consistency between the poses predicted by the encoder ($\mathbf{p}_t, \mathbf{p}_{t+1}$) and the change in pose predicted by the pose transition network ($\Delta\mathbf{p}_t$):

$$\hat{\mathbf{p}}_{t+1} = \mathbf{p}_t \oplus \Delta\mathbf{p}_t \; ; \; L_p = \frac{1}{I}\sum_{i=1}^{I}(\hat{p}^i_{t+1} - p^i_{t+1})^2 \quad (8)$$

  where $\oplus$ refers to composition in $\mathbb{SE}(3)$ pose space, $\hat{\mathbf{p}}_{t+1}$ is the expected pose at $t+1$ from composing the current pose ($\mathbf{p}_t$) and the predicted pose change from the transition model ($\Delta\mathbf{p}_t$) and I is the cardinality of $\mathbf{p}_t$. In essence, this loss constrains the encoder to predict poses that are consistent with the pose-deltas predicted by the transition model. This loss encourages global consistency in the pose space by enforcing local consistency over pairs of frames and is crucial for learning a pose space that is consistent across long term motions.

The total loss for training $L = L_x + \gamma L_p$, where γ controls the relative strengths of the two losses. We set $\gamma = 10$ in all our experiments. A key point to note is that we do not provide any explicit supervision to learn the pose space. While the consistency loss ensures that the poses are more or less globally consistent, it does not anchor them to a specific reference frame such as the object's center and its principal axes. As such, the poses learned by the network need not correspond directly to the canonical 6D pose of the parts making direct comparisons to pose estimation networks hard. Providing more constraints to physically ground the pose space is an interesting area for future work.

IV. Closed-Loop Visuomotor Control

We now show how an SE3-Pose-Net can be used for closed-loop visuomotor control to reach a target specified as a depth image, essentially performing visual servoing [1]. A crucial component of every visual servoing system is to perform data association between the current image and the target image, which can then be used to generate controls that reduce the corresponding offsets. SE3-Pose-Nets solve this problem by making use of the learned, low-dimensional latent pose space. By enforcing frame-to-frame consistency in the pose space through the consistency loss (Eqn. 8), the pose space becomes consistent, that is, our encoder network learns to data-associate observations to unique poses which are consistent under the effect of actions. Importantly, these data associations are generated at the mask, or object level, resulting in an ability akin to object detection in computer vision. Unlike prior work [4], [23] which is restricted to operate in the observation space, we can now directly minimize error between the poses $\mathbf{p}_0$ and $\mathbf{p}_T$, which are automatically extracted from the initial and the target observations, to recover the sequence of actions that takes the robot from $\mathbf{p}_0$ to $\mathbf{p}_T$. Additionally, unlike prior work [23], we do not need an external tracking system to measure progress toward the goal as our learned **encoder** implicitly tracks in the pose space.

A. Reactive control

Algorithm 1 presents a simple algorithm for reactive control using SE3-Pose-Nets that efficiently computes a closed-loop sequence of controls that takes the robot from any initial state $\mathbf{x}_0$ to the specified target $\mathbf{x}_T$ (Fig. 2, bottom right). Given a target point cloud, $\mathbf{x}_T$, the algorithm uses the learned encoder to predict the poses of the constituent parts $\mathbf{p}_T = h_{enc}(\mathbf{x}_T)$. This becomes the target to the controller.

At every time step, the algorithm computes the pose embedding $\mathbf{p}_t$ of the current observation $\mathbf{x}_t$. We would like to find controls that move these poses closer to the target poses. To do this, the algorithm makes a prediction through the learned **pose transition** model using the current poses ($\mathbf{p}_t$) and an initial guess for the controls (here we use $\mathbf{u}_t = 0$), resulting in a predicted change in poses ($\Delta\mathbf{p}_t$) and the corresponding predicted next pose ($\hat{\mathbf{p}}_{t+1}$) [2]. We measure the mean-squared error (E) between the predicted poses ($\hat{\mathbf{p}}_{t+1}$) and the target poses ($\hat{\mathbf{p}}_T$), compute its gradient with respect to the control inputs (g) and use it to generate the next control. We propose two ways of computing this update:

[2]Even when using a zero control initialization, this forward pass through the network is necessary to get the correct gradients for the backward pass.

Algorithm 1: Reactive visuomotor control

Given: Target point cloud ($\mathbf{x}_T$)
Given: Pre-trained encoder (h_{enc}) and transition model (h_{trans})
Given: Control magnitude: u_{max}
Compute target pose: $\mathbf{p}_T = h_{enc}(\mathbf{x}_T)$
while Pose Error (E) > ϵ **do**
  Receive current observation ($\mathbf{x}_t$)
  Predict current pose: $\mathbf{p}_t = h_{enc}(\mathbf{x}_t)$
  Initialize control to all zeros: $\mathbf{u}_t = 0$
  Predict change in pose: $\Delta\mathbf{p}_t = h_{trans}(\mathbf{p}_t, \mathbf{u}_t)$
  Predict next pose: $\hat{\mathbf{p}}_{t+1} = \mathbf{p}_t \oplus \Delta\mathbf{p}_t$
  Compute pose error: $E = \frac{1}{I}\sum_{i=1}^{I}(\hat{p}^i_{t+1} - p^i_T)^2$
  Compute gradient of error w.r.t. control: $g = \frac{dE}{dU_t}$
  Compute control: $\mathbf{u}_t = -u_{max} \times \frac{g}{||g||}$
  Execute control $\mathbf{u}_t$ on the robot

- **Backpropagation:** A simple approach to compute the update is to backpropagate the gradients of the pose error E through the transition model. Unlike backpropagation during training, where we compute gradients w.r.t. the network weights, here we fix the weights and compute gradients over the input controls. The resulting control scheme is analogous to the Jacobian Transpose method from inverse kinematics [8].
- **Gauss-Newton:** A better approach is to compute the Gauss-Newton update:

$$g = (J^T J + \lambda I)^{-1} J^T \frac{dE}{d\hat{\mathbf{p}}_{t+1}} \tag{9}$$

  where J is the Jacobian of the transition model, and $\frac{dE}{d\hat{\mathbf{p}}_{t+1}}$ is the gradient of the pose error (E) w.r.t the predicted poses $\hat{\mathbf{p}}_{t+1}$. This update conditions the pose error gradient through the Jacobian's pseudo-inverse, where λ controls the strength of the conditioning (set to 1e-4 in all our experiments). In practice, this leads to significantly faster convergence with little to no additional overhead in computation compared to the backpropagation method as the Jacobian can be computed efficiently through finite differencing. We do this by running a single forward propagation with perturbed control inputs (perturbation set to 1e-3) stacked along the batch dimension to take advantage of GPU parallelism. Eqn. 9 is analogous to the Damped Least Squares technique from inverse kinematics [8].

Finally, the algorithm computes the unit-vector in the direction of the computed update and scales this by a pre-specified control magnitude u_{max} (1 radian in all our experiments) to get the next control $\mathbf{u}_t$. We execute this control on the robot and repeat in a closed-loop either until **convergence**, measured by reaching a small error in the pose space ($E < \epsilon$) or a maximum number of iterations, whichever comes first.

V. Evaluation

We first evaluate SE3-Pose-Nets on predicting the dynamics of a scene where a Baxter robot moves its right arm in front of the RGBD camera, both in simulation and in the real world. We also present results on control performance where the task is to control the joints of the Baxter's right arm to reach a specified target observation.

A. Task and Data collection

We first provide details on the task setting in simulation. Our simulator uses OpenGL to render depth images from a camera pointed towards the robot (see Fig. 3) and is kinematic with little to no dynamics in the motion and no depth noise. We use this as a test bed to parse the effectiveness of the proposed algorithm and compare it to various baselines. We collected around 800K training images (from a single viewpoint) in the simulator where the robot moves all joints on its right arm. Around half of the examples are whole arm motions where the robot plans a trajectory to reach a target end-effector position sampled randomly in front of the robot. The remaining motions are perturbations to individual joints from various initial configurations sampled to be within the viewpoint of the camera. These additional motions help de-correlate kinematic dependencies, improving performance especially on joints lower down the kinematic chain.

Additionally, we collected (RGBD) data from the real robot where the Baxter moves its right arm in front of an ASUS Xtion Pro camera placed around 2.5 meters from the robot. Data associations, ground truth masks, and ground truth flows are determined via the DART tracker [6]. We collected around 7 hours of training data on the real robot with a 1:1 mix of whole arm and single joint motions (specifically of the lower joints of the arm). This data has some (unintended) variations in the background and minor changes in the camera pose and lighting. Unlike the simulated data, the depth data in the real world is quite noisy and there are significant physical and dynamic effects. For both the simulated and real world settings, our controls are joint velocities ($\mathbf{u}$).

B. Baselines

We compare against five different baselines:

- **SE3-Pose-Nets + Joint Angles:** Our proposed network with the robot's joint angles given as an extra input to the **encoder**. This is a strong baseline that uses significant extra information to inform pose prediction.
- **SE3-Nets:** Prior work from [4] which directly predicts masks and change in poses given input point clouds and control. As there is no explicit pose space, we control in the point cloud space with this network.
- **SE3-Nets + Joint Angles:** SE3-Nets that additionally take in joint angles as inputs.
- **Flow Net:** Baseline flow model from prior work [4]. This network directly regresses to a per-point 3D flow without any $\mathbb{SE}(3)$ transforms or masks. Similar to SE3-Nets we control in point cloud space with this network.
- **Flow Net + Joint Angles:** Baseline flow network that additionally takes in joint angles as input.

All baseline networks are trained on the same data as the SE3-Pose-Nets using the 3D normalized loss (L_x).

C. Training details

We implemented our networks in PyTorch using the RMSprop/ADAM optimizers for training with a learning rate of 1e-4 (baselines work best with ADAM). We set the maximum number of moving objects $K = 8$ for all our

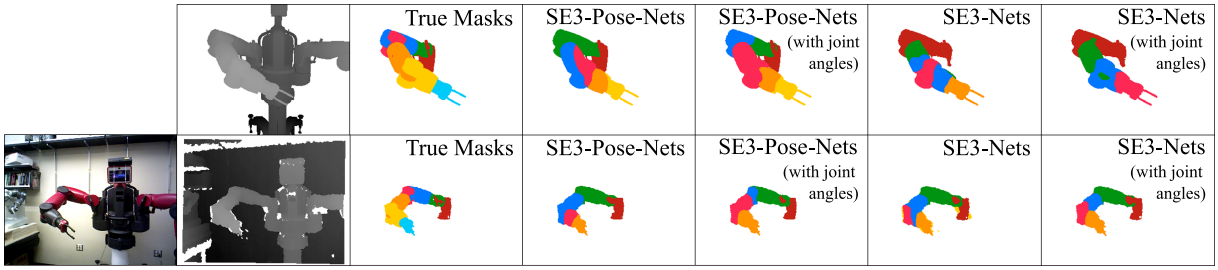

Fig. 3: Masks generated by different networks on simulated (top) and real data (bottom, with additional color input shown). Both SE3-NETS and SE3-POSE-NETS segment the arm into multiple "physically consistent" parts without any explicit supervision. *Note:* Colors are for display only. Predicted mask colors **do not** need to match the colors of ground truth masks.

Setting	SE3-POSE-NETS	SE3-POSE-NETS + Joint Angles	SE3-NETS	SE3-NETS + Joint Angles	Flow	Flow + Joint Angles
Simulated	0.027	0.017	0.022	**0.012**	0.037	0.021
Real	0.209	0.200	0.182	**0.170**	0.201	0.193

TABLE I: Average per-point flow RMSE (cm) across tasks and networks, normalized by the number of points M that have ground truth motion > 1mm. Our network achieves results comparable/better than baseline networks on simulated data and performs slightly worse on real data. However, it is also solving additional tasks necessary for control.

experiments (7 joints + background). We train each network for 100K iterations, and use the one with the least 3D loss (L_x) on a separate validation set for our results. Additionally for the real-world experiments, we use RGB images as an extra input to the pose-mask encoder (concatenated with the depth image) for training all networks including baselines.

D. Results on modeling scene dynamics

First, we present results on the prediction task used for training all networks. Table I shows the average per-point flow RMSE (cm) across all baselines on both simulated and real data. SE3-NETS achieve the best results while the baseline flow network performs slightly worse. Unsurprisingly, networks that have access to the joint angles do better than those which do not, as they have strictly more information that is highly correlated with the sensor data. To our initial surprise, SE3-POSE-NETS had the largest prediction errors among all baseline models (on real data). However, this makes sense given the following considerations: a) SE3-POSE-NETS are trained to explicitly embed the observations in a pose space from which they predict the scene dynamics, rather than using the input point cloud directly. While this provides more structure and is necessary for the control task, it also restricts the prediction to go through an information bottleneck which generally makes training hard. b) SE3-POSE-NETS additionally have to optimize the consistency loss, which enforces constraints that are different from those of the prediction problem evaluated in this experiment.
Fig. 3 visualizes the masks predicted by SE3-POSE-NETS and the baseline SE3-NET on an example each from the simulated and real data along with the ground truth masks. Even without any supervision, SE3-POSE-NETS and SE3-NETS learn a detailed segmentation of the arm into multiple salient parts, most of which are consistent with ground truth segments on both the simulated and real data.

E. Control performance

Next, we test the performance of the networks on controlling the first six joints of the Baxter's right arm to reach a target configuration, specified as a point cloud $\mathbf{x}_T$. We test both the gradient update schemes from Sec. IV, comparing their performance on a set of 11 distinct servoing tasks (each with an average initial error of ~25 degrees per joint).

Control with baseline models: As most of our baseline models operate directly in the observation space (unlike the learned pose space in SE3-POSE-NETS), they **require external data associations** to be able to do any control at all. For the simulation experiments, we provide the baselines with ground-truth associations and use the procedure outlined in Alg. 1 using the MSE between the predicted point cloud $\hat{\mathbf{x}}_{t+1}$ and the target $\mathbf{x}_T$ as the error to be minimized for generating controls. It is important to keep in mind that the baseline models have an advantage over SE3-POSE-NETS for the control task as they get strictly more information in the form of ground-truth data associations.

Metric and Task specification: We use the mean absolute error in the joint angles as the metric for measuring control performance. We run all models to convergence (based on the pose error for SE3-POSE-NETS and 3D point/flow error for the baseline models) or for a maximum of 200/500 iterations (sim/real). We integrate joint velocities forward to generate position commands both in simulation and the real world.

Simulation results: Fig. 4 plots the error in joint angles as a function of the number of control iterations. The first two plots (from the left) show results in simulation, both for networks that do not use joint angles (leftmost) and otherwise (middle). In general, SE3-POSE-NETS achieve excellent performance compared to the baseline models, converging quickly to an almost zero error even **in the absence any external data associations**. We highlight a few key results: 1) For all networks, Gauss-Newton based optimization (GN) leads to faster convergence than backprop. This is to be expected as Gauss-Newton conditions the gradient based on pseudo-second order information. 2) Baseline SE3-NET models perform worse given joint angles than without. This is due to an issue of credit assignment during gradient computation - the networks learn erroneous causations (when there are only correlations) between the input joint angles

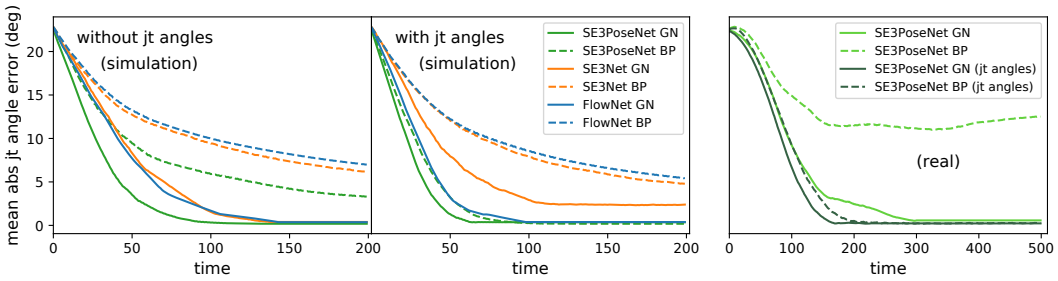

Fig. 4: Convergence of mean absolute joint angle error (averaged across 11 different examples, 6 joint control) for simulated Baxter control tasks (left - w/o joint angle input & center - with joint angle input) and real robot control (right). Dotted lines use backprop gradient updates (BP), solid lines use Gauss-Newton updates (GN). SE3-POSE-NETS perform as well or better than baseline methods even though baseline models have additional information in the form of ground truth-associations. All plots share the y-axis.

Fig. 5: Examples of different generalization tests. Clockwise from top left: 1) Low light setting, 2) Motion in the background (person on the chair), 3) Fixed distractor(s) in the foreground and 4) Change in camera pose (rotated ~10 degrees to the right).

and the predicted flows which diminishes the control's contribution to the prediction problem and subsequently affects the gradient. Additionally, SE3-NETS performance is affected by the lack of a good control initialization (needed to ensure that they get a good segmentation) - given zero controls the SE3-NET can choose not to segment the arm at all 3) All models struggle with the motion of the final wrist joint due to increasing correlations along the kinematic chain that result in a small contribution of the joint's own motion to the full movement of the wrist. SE3-POSE-NETS (and other baselines) do overcome this problem given input joint angles in a separate experiment, achieving < 1 deg error across all 11 examples when controlling the entire right arm. This provides encouraging proof that adding in the joint state supplements information that is hard to parse directly from vision, establishing the case for including them systematically, and finally 4) Good performance of SE3-POSE-NETS indicate that the learned pose space is consistent across large motions and can be used for fast, robust, and reactive control.

Real robot results: We further test the control performance of SE3-POSE-NETS in the real world (RGBD input) on the same set of configurations. We do not compare to any baselines as they need an explicit external data association system to be feasible. Fig. 4 shows the results - SE3-POSE-NETS converge very quickly to nearly zero error on all examples indicating that our network can control robustly even in the presence of sensor noise and unmodeled dynamics. Once again, Gauss-Newton significantly outperforms the backprop update which fails to converge in the absence of joint angles. Additionally, adding joint angles does allow us to control the wrist, albeit not as robustly as in simulation.

Result	Lighting	Camera	Occlusion	Motion
Average Error	3.8	1.4	1.9	2.5
Failures	0/11	1/11	1/11	2/11

TABLE II: Average final joint angle error (degrees) and number of failures (divergent examples) out of 11 tasks across different perturbations. Reported errors averaged over non-divergent examples.

Generalization/Robustness results: Finally, we tested the generalization performance and robustness of real-robot control using SE3-POSE-NETS (GN, no joint angles) to novel perturbations of the scene. We tested control performance across all 11 servoing tasks for the following four variations: *1) Change in lighting:* We considered a low-light setup, significantly darker than the training set (Fig. 5, top left), *2) Varying camera pose:* We considered three variations to the camera pose: moving forward by 10 cm and rotating by 10 degrees to the left & right (Fig. 5, bottom right), *3) Occlusion:* We tested three settings where we added multiple fixed occluders to the robot: cloth attached to the robot torso only, additionally occluding the base joint and adding occluders to the torso, base joint and end-effector (Fig. 5, bottom left) and finally, *4) Motion:* We tested four different settings with moving distractors (people, books) in the foreground (in front of the robot) and background Fig. 5, top right).

Table II summarizes the control results under perturbations. In general, SE3-POSE-NETS can control robustly even in the presence of scene changes, achieving low joint angle error at the end of the control optimization. There is a caveat though: we observed divergent behavior for joints lower down in the kinematic chain (primarily joints 5,6) for 1-2 servoing examples under some perturbations (Table II, last row). While improvements in robustness and a more thorough evaluation (also in simulation) are necessary, we believe that these results serve as a good proof of concept of the strengths of SE3-POSE-NETS. Also, these results highlight some of the advantages of learning methods as opposed to more traditional model-based tracking methods (such as DART [6]) which need significant additional work to handle such variations. A video showing real-robot control (with and without scene variations) can be found here.

Speed: SE3-POSE-NETS optimize errors directly in the low-dimensional pose space for control. This leads to significant speedups: while both the flow and SE3-NETS can operate at around 10Hz (excluding data association), SE3-POSE-NETS run in real-time (30Hz) including pose detection.

VI. Discussion

This paper presents SE3-Pose-Nets, a framework for learning predictive models that enable control of objects in a scene. In the context of a robot manipulator, we showed how to solve this problem by learning a predictive model for the individual parts of the manipulator, as in prior work [4]. Additionally, SE3-Pose-Nets learn a consistent *pose space* for these parts, essentially learning to *detect* the 6D poses of manipulator parts in the raw depth images. This detection capability enables SE3-Pose-Nets to solve the data association problem that is crucial for relating the current observation of the manipulator to a desired target observation. The difference between these poses can be used to generate control signals to move the manipulator to its target pose, similar to visual servoing applied to an image of the manipulator. We also showed how the learned network can be used to determine the gradients needed for the control signals. Our experiments show that SE3-Pose-Nets generate control superior to representations learned by previous techniques, even when these are provided with external data associations. Furthermore, in addition to providing data associations, SE3-Pose-Nets allow us to compute controls directly in the low dimensional pose space, enabling far more efficient control than techniques that operate in the raw perception space. Additionally, control using SE3-Pose-Nets is robust to some perturbations to imaging conditions. Crucially, all these abilities are learned in a single framework based on raw data traces solely annotated with frame-to-frame point cloud correspondences.

Overall, the control performance shown by our SE3-Pose-Nets is extremely encouraging and provides strong evidence that such networks can learn a consistent pose space that provides long-range correspondences for fast reactive control. While this provides reason to rejoice, there are multiple areas for improvement: 1) As shown in the generalization results, SE3-Pose-Nets can exhibit non-convergent behavior under some perturbations - this is primarily for joints lower down the kinematic chain (joints 5–7). There are potentially multiple ways to tackle this problem, including curriculum and active learning, physical grounding of the pose space and through the use of domain randomization [24] techniques to facilitate sim-to-real transfer and generalization to novel scenarios. 2) A key area for future work is in extending our system to interact with and manipulate external objects. Here, a consistent pose space for objects will help the robot to plan its motion toward the objects, enabling smooth interactions. 3) SE3-Pose-Nets decompose the visuomotor control problem into state estimation and dynamics learning, resulting in a modular architecture reminiscent of traditional model-based methods. We plan to exploit this modularity by pre-training parts of our system on sub-problems followed by fine-tuning, reducing data needs for generalization to novel settings. 4) Finally, while we have shown that SE3-Pose-Nets can be used for single-step reactive control, we would like to do long-term planning using model based techniques such as iLQG [25] to leverage the full strength of the latent pose space, i.e., fast real-time rollouts directly in the pose space.

Acknowledgments

This work was funded in part by the National Science Foundation under contract no. NSF-NRI-1637479 and STTR no. 1622958 awarded to LULA Robotics. We thank NVIDIA for providing a DGX via the UW NVIDIA AI Lab (NVAIL) and Aaron Walsman for help in designing the visualizations.

References

[1] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE transactions on robotics and automation*, vol. 12, no. 5, pp. 651–670, 1996.

[2] B. Boots, A. Byravan, and D. Fox, "Learning predictive models of a depth camera & manipulator from raw execution traces," in *ICRA*. IEEE, 2014, pp. 4021–4028.

[3] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *NIPS*, 2016.

[4] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *ICRA*. IEEE, 2017, pp. 173–180.

[5] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *ICRA*. IEEE, 2017, pp. 2786–2793.

[6] T. Schmidt, R. A. Newcombe, and D. Fox, "Dart: Dense articulated real-time tracking." in *RSS*, 2014.

[7] R. Anderson, D. Gallup, J. T. Barron, J. Kontkanen, N. Snavely, C. Hernández, S. Agarwal, and S. M. Seitz, "Jump: virtual reality video," *ACM Transactions on Graphics (TOG)*, 2016.

[8] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods."

[9] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *NIPS*, 2015, pp. 2728–2736.

[10] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015.

[11] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, vol. 17, no. 39, pp. 1–40, 2016.

[12] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *arXiv preprint arXiv:1606.07419*, 2016.

[13] R. Jonschkowski, R. Hafner, J. Scholz, and M. Riedmiller, "Pves: Position-velocity encoders for unsupervised learning of structured state representations," *arXiv preprint arXiv:1705.09805*, 2017.

[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.

[15] T. Schmidt, R. Newcombe, and D. Fox, "Self-supervised visual descriptor learning for dense correspondence," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 420–427, 2017.

[16] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *ICCV*, 2015, pp. 2794–2802.

[17] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-dof object pose from semantic keypoints," in *ICRA*. IEEE, 2017, pp. 2011–2018.

[18] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *ICCV*. IEEE, 2015, pp. 2938–2946.

[19] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d object pose estimation using 3d object coordinates," in *ECCV*. Springer, 2014, pp. 536–551.

[20] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *Geometric reasoning for perception and action*, pp. 106–136, 1993.

[21] F. Chaumette, S. Hutchinson, and P. Corke, "Visual servoing," in *Springer Handbook of Robotics*, 2016, pp. 841–866.

[22] A. X. Lee, S. Levine, and P. Abbeel, "Learning visual servoing with deep features and fitted q-iteration," *ICLR*, 2017.

[23] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," *arXiv preprint arXiv:1610.00696*, 2016.

[24] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS*. IEEE, 2017, pp. 23–30.

[25] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *ACC*. IEEE, 2005, pp. 300–306.