A Single Integrator Distribution Integration

For a single integrator, we define the state to be the position vector $\mathbf{s} = \mathbf{p} = [x, y]^T$, the control to be the velocity vector $\mathbf{u} = \dot{\mathbf{p}} = [\dot{x}, \dot{y}]^T$, and write the linear discrete-time dynamics as

$$\mathbf{p}^{(t+1)} = I_{2\times 2}\mathbf{p}^{(t)} + \Delta t I_{2\times 2}\dot{\mathbf{p}}^{(t)}. \tag{6}$$

At each timestep, and for a specific latent value z, *Trajectron++* produces a Gaussian distribution over control actions $\mathcal{N}(\mu_{\mathbf{u}}, \mathbf{\Sigma}_{\mathbf{u}})$. Specifically, it outputs

$$\mu_{\mathbf{u}} = \begin{bmatrix} \mu_{\dot{x}} \\ \mu_{\dot{y}} \end{bmatrix} \qquad \mathbf{\Sigma}_{\mathbf{u}} = \begin{bmatrix} \sigma_{\dot{x}}^2 & \rho_{\dot{x}\dot{y}}\sigma_{\dot{x}}\sigma_{\dot{y}} \\ \rho_{\dot{x}\dot{y}}\sigma_{\dot{x}}\sigma_{\dot{y}} & \sigma_{\dot{y}}^2 \end{bmatrix}, \tag{7}$$

where $\mu_{\dot{x}}$ and $\mu_{\dot{y}}$ are the respective mean velocities in the agent's longitudinal and lateral directions, $\sigma_{\dot{x}}$ and $\sigma_{\dot{y}}$ are the respectivte longitudinal and lateral velocity standard deviations, and $\rho_{\dot{x}\dot{y}}$ is the correlation between $\dot{x}$ and $\dot{y}$. Since $\mathbf{\Sigma}_{\mathbf{u}}$ is the only source of uncertainty in the prediction model, Equation (6) is a linear Gaussian system.

A.1 Mean Derivation[2]

Following the sum of Gaussian random variables [25], the output mean positions are obtained by Equation (6). Thus, at test time, *Trajectron++* produces $\mu_{\dot{\mathbf{p}}}^{(t)}$ which is passed through Equation (6) alongside the current agent position $\mu_{\mathbf{p}}^{(t)}$ to produce the predicted position mean $\mu_{\mathbf{p}}^{(t+1)}$.

A.2 Covariance Derivation[2]

The position covariance is obtained via the covariance of a sum of Gaussian random variables [25]

$$\begin{aligned} \mathbf{\Sigma}_{\mathbf{p}}^{(t+1)} &= I_{2\times 2}\mathbf{\Sigma}_{\mathbf{p}}^{(t)} I_{2\times 2}^T + \Delta t I_{2\times 2}\mathbf{\Sigma}_{\mathbf{u}}^{(t)} \Delta t I_{2\times 2}^T \\ &= \mathbf{\Sigma}_{\mathbf{p}}^{(t)} + (\Delta t)^2 \mathbf{\Sigma}_{\mathbf{u}}^{(t)}. \end{aligned} \tag{8}$$

B Dynamically-Extended Unicycle Distribution Integration

Usually, unicycle models have velocity and heading rate as control inputs [30,35]. However, vehicles in the real world are controlled by accelerator pedals and so we instead adopt the dynamically-extended unicycle model which instead uses acceleration a and heading rate ω as control inputs [29]. The dynamically-extended unicycle model has the following nonlinear continuous-time dynamics

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\phi) \\ v\sin(\phi) \\ \omega \\ a \end{bmatrix}, \tag{9}$$

[2] These equations are also found in the Kalman Filter prediction step [25].

where $\mathbf{p} = [x, y]^T$ defines the position, v the speed, and ϕ the heading. As mentioned above, the control inputs are $\mathbf{u} = [\omega, a]^T$. To discretize this, we assume a zero-order hold on the controls between each sampling step (i.e. control actions are piece-wise constant). This yields the following zero-order hold discrete equivalent dynamics

$$\begin{bmatrix} x^{(t+1)} \\ y^{(t+1)} \\ \phi^{(t+1)} \\ v^{(t+1)} \end{bmatrix} = \begin{bmatrix} x^{(t)} \\ y^{(t)} \\ \phi^{(t)} \\ v^{(t)} \end{bmatrix} + \begin{bmatrix} v^{(t)} \cdot D_S^{(t)} + \frac{a^{(t)} \sin(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} + \frac{a^{(t)}}{\omega^{(t)}} \cdot D_C^{(t)} \\ -v^{(t)} \cdot D_C^{(t)} - \frac{a^{(t)} \cos(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} + \frac{a^{(t)}}{\omega^{(t)}} \cdot D_S^{(t)} \\ \omega^{(t)} \Delta t \\ a^{(t)} \Delta t \end{bmatrix},$$
$$\text{where } D_S^{(t)} = \frac{\sin(\phi^{(t)} + \omega^{(t)} \Delta t) - \sin(\phi^{(t)})}{\omega^{(t)}}$$
$$D_C^{(t)} = \frac{\cos(\phi^{(t)} + \omega^{(t)} \Delta t) - \cos(\phi^{(t)})}{\omega^{(t)}}. \tag{10}$$

We will refer to these dynamics in short with $\mathbf{s}^{(t+1)} = \mathbf{f}(\mathbf{s}^{(t)}, \mathbf{u}^{(t)})$. We adopt a slightly different set of dynamics when $|\omega| \leq \epsilon = 10^{-3}$ to avoid singularities in Equation (10). With a small ω, we instead use the following dynamics, obtained by evaluating the limit as $\omega \to 0$.

$$\begin{bmatrix} x^{(t+1)} \\ y^{(t+1)} \\ \phi^{(t+1)} \\ v^{(t+1)} \end{bmatrix} = \begin{bmatrix} x^{(t)} \\ y^{(t)} \\ \phi^{(t)} \\ v^{(t)} \end{bmatrix} + \begin{bmatrix} v^{(t)} \cos(\phi^{(t)}) \Delta t + 0.5 a^{(t)} \cos(\phi^{(t)}) (\Delta t)^2 \\ v^{(t)} \sin(\phi^{(t)}) \Delta t + 0.5 a^{(t)} \sin(\phi^{(t)}) (\Delta t)^2 \\ 0 \\ a^{(t)} \Delta t \end{bmatrix}. \tag{11}$$

Thus, the full discrete-time dynamics are

$$\begin{bmatrix} x^{(t+1)} \\ y^{(t+1)} \\ \phi^{(t+1)} \\ v^{(t+1)} \end{bmatrix} = \begin{cases} \text{Equation (10)} & \text{if } |\omega| > \epsilon \\ \text{Equation (11)} & \text{otherwise} \end{cases}. \tag{12}$$

At each timestep, and for a specific latent value z, *Trajectron++* produces a Gaussian distribution over control actions $\mathcal{N}(\mu_{\mathbf{u}}, \mathbf{\Sigma}_{\mathbf{u}})$. Specifically, it outputs

$$\mu_{\mathbf{u}} = \begin{bmatrix} \mu_\omega \\ \mu_a \end{bmatrix} \qquad \mathbf{\Sigma}_{\mathbf{u}} = \begin{bmatrix} \sigma_\omega^2 & \rho_{\omega a} \sigma_\omega \sigma_a \\ \rho_{\omega a} \sigma_\omega \sigma_a & \sigma_a^2 \end{bmatrix}, \tag{13}$$

where μ_ω is the mean rate of change of the agent's heading, μ_a is the mean acceleration in the agent's heading direction, σ_ω is the standard deviation of the heading rate of change, σ_a is the acceleration standard deviation, and $\rho_{\omega a}$ is the correlation between ω and a. The controls $\mu_{\mathbf{u}}$ and uncertainties $\mathbf{\Sigma}_{\mathbf{u}}$ are then integrated through the dynamics to obtain the following mean and covariance integration equations [46].

B.1 Mean Derivation[3]

The output mean positions are obtained by applying the mean control actions to Equation (12) [46].

B.2 Covariance Derivation[3]

Since $\mathbf{\Sigma_u}$ is the only source of uncertainty in the prediction model, Equation (12) can be made a linear Gaussian system by linearizing about a specific state and control. For instance, the Jacobians $\mathbf{F}$ and $\mathbf{G}$ of the system dynamics in Equation (10) are

$$\mathbf{F}^{(t)} = \frac{\partial \mathbf{f}}{\partial \mu_{\mathbf{s}}^{(t)}} = \begin{bmatrix} 1 & 0 & v^{(t)} D_C^{(t)} - \frac{a^{(t)} D_S^{(t)}}{\omega^{(t)}} + \frac{a^{(t)} \cos(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} & D_S^{(t)} \\ 0 & 1 & v^{(t)} D_S^{(t)} + \frac{a^{(t)} D_C^{(t)}}{\omega^{(t)}} + \frac{a^{(t)} \sin(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} & -D_C^{(t)} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G}^{(t)} = \frac{\partial \mathbf{f}}{\partial \mu_{\mathbf{u}}^{(t)}} = \begin{bmatrix} G_{11}^{(t)} & \frac{D_C^{(t)}}{\omega^{(t)}} + \frac{\sin(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} \\ G_{21}^{(t)} & \frac{D_S^{(t)}}{\omega^{(t)}} - \frac{\cos(\phi^{(t)} + \omega^{(t)} \Delta t) \Delta t}{\omega^{(t)}} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix},$$

$$\begin{aligned} \text{where } G_{11}^{(t)} &= \frac{v \cos(\phi + \omega \Delta t) \Delta t}{\omega} - \frac{v D_S}{\omega} - \frac{2a \sin(\phi + \omega \Delta t) \Delta t}{\omega^2} - \frac{2a D_C}{\omega^2} \\ &\quad + \frac{a \cos(\phi + \omega \Delta t)(\Delta t)^2}{\omega} \\ G_{21}^{(t)} &= \frac{v \sin(\phi + \omega \Delta t) \Delta t}{\omega} + \frac{v D_C}{\omega} + \frac{2a \cos(\phi + \omega \Delta t) \Delta t}{\omega^2} - \frac{2a D_S}{\omega^2} \\ &\quad + \frac{a \sin(\phi + \omega \Delta t)(\Delta t)^2}{\omega}. \end{aligned} \tag{14}$$

Then, applying the equations for the covariance of a sum of Gaussian random variables [46] yields

$$\mathbf{\Sigma}_{\mathbf{p},\theta,v}^{(t+1)} = \mathbf{F}^{(t)} \mathbf{\Sigma}_{\mathbf{p},\theta,v}^{(t)} \mathbf{F}^{(t)^T} + \mathbf{G}^{(t)} \mathbf{\Sigma}_{\mathbf{u}}^{(t)} \mathbf{G}^{(t)^T}. \tag{15}$$

C Average and Final Displacement Error Evaluation

While ADE and FDE are important metrics for deterministic, single-trajectory methods, any deeper probabilistic information available from generative methods is destroyed when taking the mean over the dataset. Instead, in the main body of the paper we focus on evaluation methods which maintain such information. However, we can somewhat directly compare deterministic and generative methods using ADE and FDE by directly plotting the full error distributions for any generative methods, as in [20]. This provides an idea as to how close and

[3] These equations are also found in the Extended Kalman Filter prediction step [46].

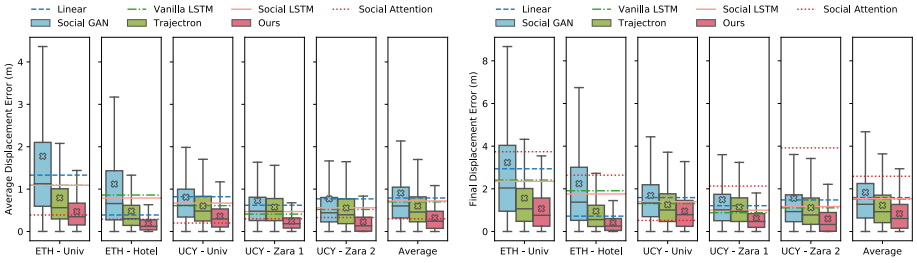

Fig. 5. Left: ADE results of all methods per dataset, as well as their average performance. Boxplots are shown for all generative models since they produce distributions of trajectories. 2000 trajectories were sampled per model at each prediction timestep, with each samples ADE included in the boxplots. Our approach with dynamics integration is compared here, specifically its z_{mode} output configuration. X markers indicate the mean ADE. Mean ADE from deterministic baselines are visualized as horizontal lines. **Right**: The same analysis for FDE.

concentrated the predictions are around the ground truth. Figure 5 shows both generative and deterministic methods' ADE and FDE performance. In both metrics, our method's error distribution is lower and more concentrated than other generative approaches, even outperforming state-of-the-art deterministic methods.

D Additional Training Information

D.1 Choosing α, β in Equation (5)

As shown in [17], the β parameter weighting the KL penalty term is important to disentangle the latent space and encourage multimodality. A good value for this hyperparameter varies with the the size of input $\mathbf{y}$, condition $\mathbf{x}$, and latent space z. Therefore, we adjust β depending on the size of the encoder's output $e_{\mathbf{x}}$. For example, we increase the value of β when encoding map information in the condition. Additionally, β is annealed following an increasing sigmoid [4]. Thus, a low β factor is used during early training iterations so that the model learns to encode as much information in z as possible. As training continues, β is gradually increased to shift the role of information encoding from $q_\phi(z \mid \mathbf{x}, \mathbf{y})$ to $p_\theta(z \mid \mathbf{x})$. For α, we found that a constant value of 1.0 works well.

D.2 Separate Map Encoder Learning Rate

When used, we train the map encoding CNN with a smaller learning rate compared to the rest of the model, to prevent large gradients in early training iterations. We use leaky ReLU activation functions with $\alpha = 0.2$ to prevent saturation during early training iterations (when the CNN does not provide useful encodings to the rest of the model). We found that regular ReLU, sigmoid, and tanh activation functions saturate during early training and fail to recover.

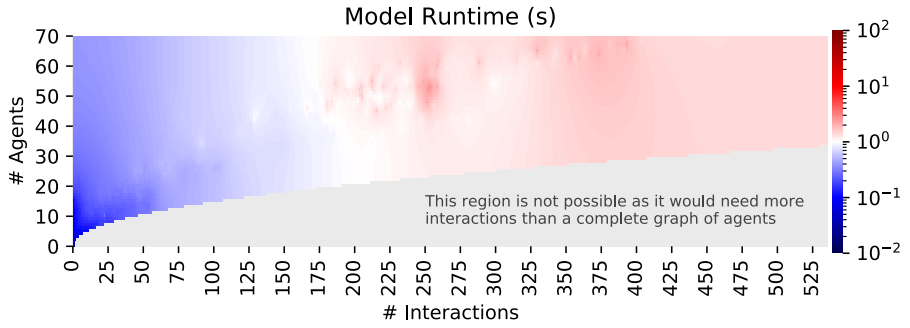

Fig. 6. Mean time for *Trajectron++* to generate future trajectory distributions on a laptop with a 2.7 GHz Intel Core i5 (Broadwell) CPU and 8 GB of RAM. None of the runtimes exceed $1.2s$ (with most at or below $1s$), enabling the model to run $3-5\times$ per prediction horizon for all tested datasets.

E Online Runtime

Figure 6 illustrates how *Trajectron++*'s runtime scales with respect to problem size. In particular, a heatmap is used as there are two major factors that affect the model's runtime: number of agents and amount of interactions. For points with insufficient data, e.g., the rare case of 50 agents in a scene with only 3 interactions, we impute values using an optimization-based scheme [15]. To achieve this real-time performance, we leverage the stateful representation that spatiotemporal graphs provide. Specifically, *Trajectron++* is updated online with new information without fully executing a forward pass. This is possible due to our method's use of LSTMs, as only the last LSTM cells in the encoder need to be fed the newly-observed data. The rest of the model can then be executed using the updated encoder representation.