Modeling Irregular Time Series with Continuous Recurrent Units

Mona Schirmer 1  Mazin Eltayeb 2  Stefan Lessmann 3  Maja Rudolph 4

Abstract

Recurrent neural networks (RNNs) are a popular choice for modeling sequential data. Modern RNN architectures assume constant time-intervals between observations. However, in many datasets (e.g. medical records) observation times are irregular and can carry important information. To address this challenge, we propose continuous recurrent units (CRUs) – a neural architecture that can naturally handle irregular intervals between observations. The CRU assumes a hidden state, which evolves according to a linear stochastic differential equation and is integrated into an encoder-decoder framework. The recursive computations of the CRU can be derived using the continuous-discrete Kalman filter and are in closed form. The resulting recurrent architecture has temporal continuity between hidden states and a gating mechanism that can optimally integrate noisy observations. We derive an efficient parameterization scheme for the CRU that leads to a fast implementation f-CRU. We empirically study the CRU on a number of challenging datasets and find that it can interpolate irregular time series better than methods based on neural ordinary differential equations.

1. Introduction

Recurrent architectures, such as the long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) or gated recurrent unit (GRU) (Chung et al., 2014) have become a principal machine learning tool for modeling time series. Their modeling power comes from a hidden state, which is recursively updated to integrate new observations, and a gating mechanism to balance the importance of new information with history already encoded in the latent state. Although continuous formulations were frequently considered in early work on recurrent neural networks (RNNs) (Pineda, 1987; Pearlmutter, 1989; 1995), modern RNNs typically assume regular sampling rates (Hochreiter & Schmidhuber, 1997; Chung et al., 2014). Many real world data sets, such as electronic health records or climate data, are irregularly sampled. Measurements of a patient’s health status, for example, are only available when the patient sees a doctor. Hence, the time between observations also carries information about the underlying time series. A lab test not administered for many months could imply that the patient was doing well in the meantime, while frequent visits might indicate that the patient’s health is deteriorating. Discrete RNNs face difficulties modeling such data as they do not reflect the continuity of the underlying temporal processes.

Recently, the work on neural ordinary differential equations (neural ODEs) (Chen et al., 2018) has established an elegant and practical way of modelling irregularly sampled time series. Recurrent architectures based on neural ODEs determine the hidden state between observations by an ordinary differential equation (ODE) and update its hidden state at observation times using standard RNN gating mechanisms (Rubanova et al. 2019; Brouwer et al. 2019; Lechner & Hasani 2020). These methods typically rely on some form of numerical ODE-solver, which can prolong training time significantly (Rubanova et al. 2019; Shukla & Marlin 2020).

We propose continuous recurrent unit (CRU), a probabilistic recurrent architecture for modelling irregularly sampled time series. An encoder maps observations into a latent space, which is governed by a linear stochastic differential equation (SDE). The analytic solution for propagating the latent state between observations and the update equations for integrating new observations are given by the continuous-discrete formulation of the Kalman filter. Employing the linear SDE state space model and the Kalman filter has three advantages. First, a probabilistic state space provides an explicit notion of uncertainty for an uncertainty-driven gating mechanism and for confidence evaluation of predictions. Second, as the Kalman filter is the optimal solution for the linear filtering problem (Kalman, 1960), the gating mechanism is optimal in a locally linear state space. Third, the latent state at any point in time can be resolved analytically, therefore bypassing the need for numerical integration techniques or variational approximations. In summary, our...
contributions are as follows:

- In Sec. 3.3, we develop the CRU, a model that combines the power of neural networks for feature extraction with the advantages of probabilistic state-space models, specifically the continuous-discrete Kalman filter. As a result, the CRU is a powerful neural architecture that can naturally model data with irregular observation times.

- We derive a novel parameterization of the latent state transition matrices via their eigenspaces (Sec. 3.4) leading to a faster implementation we call fast CRU (f-CRU).

- In Sec. 4, we study the CRU on images, electronic health records, and climate data. We find that (i) our method can better interpolate irregular time series than neural ODE-based methods, (ii) the CRU can handle uncertainty arising from both noisy and partially observed inputs, (iii) CRU outperforms both discrete RNN counterparts and neural ODE-based models on image data.

2. Related Work

Stochastic RNNs RNNs, such as LSTMs or GRUs, are powerful sequence models (Hochreiter & Schmidhuber, 1997; Chung et al., 2014), but due to the lack of stochasticity in their internal transitions, they may fail to capture the variability inherent in certain data (Chung et al. 2015). While there are various stochastic RNNs (e.g., Bayer & Osendorfer, 2014; Fraccaro et al., 2016; Goyal et al., 2017; Schmidt & Hofmann, 2018), our work is most closely related to deep probabilistic approaches based on Kalman filters (Kalman, 1960). Variations on deep Kalman filters (Krishnan et al. 2015; Karl et al. 2016; Fraccaro et al. 2017) typically require approximate inference, but Becker et al. (2019) employ a locally linear model in a high-dimensional factorized latent state for which the Kalman updates can be obtained in closed form. By extending this approach with a continuous latent state, the CRU can model sequences with irregular observation times.

RNNs for irregular time series Applying discrete RNNs to irregularly sampled time series requires the discretization of the time line into uniform bins. This often reduces the number of observations, may result in a loss of information, and evokes the need for imputation and aggregation strategies. To avoid such preprocessing, Choi et al. (2018) and Mozer et al. (2017) propose to augment observations with timestamps. Lipton et al. (2016) suggest observation masks. However, such approaches have no notion of dynamics between observations. An alternative approach is to decay the hidden state exponentially between observations according to a trainable decay parameter (Che et al., 2018; Cao et al., 2018). These methods are limited to decaying dynamics whereas the CRU is more expressive.

Continuous-time RNNs Continuous-time RNNs have a long history, dating back to some of the original work on recurrent networks in the field. They are recurrent architectures whose internal units are governed by a system of ODEs with trainable weights (Pearlmutter, 1995). The theory for different gradient-based optimization schemes for their parameters have been developed by Pineda (1987), Pearlmutter (1989), and Sato (1990). Notably, LeCun et al. (1988)’s derivation using the adjoint method provides the theoretical foundation for modern implementations of neural ODEs (Chen et al., 2018).

Neural ODEs Neural ODEs model the continuous dynamics of a hidden state by an ODE specified by a neural network layer. Chen et al. (2018) propose latent ODE, a generative model whose latent state evolves according to a neural ODE. However, it has no update mechanism to incorporate incoming observations into the latent trajectory. Kidger et al. (2020) and Morrill et al. (2021) extend neural ODEs with concepts from rough analysis, which allow for online learning. ODE-RNN (Rubanova et al., 2019) and ODE-LSTM (Lechner & Hasani, 2020) use standard RNN gates to sequentially update the hidden state at observation times. GRU-ODE-B (Brouwer et al., 2019) and Neural Jump ODE (NJ-ODE) (Herrera et al., 2020) couple ODE dynamics with an Bayesian update step. Neural ODE approaches typically rely on a numerical ODE solver, whereas the forward pass of a CRU is in closed form.

3. Method

The CRU is a RNN for processing sequential data with irregular observation times. It employs a nonlinear mapping (a neural network encoder and decoder) to relate individual observations with a latent state space. In this latent state space, it assumes a continuous latent state whose dynamics evolve according to a linear SDE. The recursive computations of the CRU can be derived using the continuous-discrete Kalman filter (Jazwinski, 1970) and are in closed form. As a result, the CRU has temporal continuity between hidden states and a gating mechanism that can optimally integrate noisy observations at arbitrary observation times.

We first specify the modeling assumptions for the continuous latent state of the CRU as well as the role of the encoder and the decoder in Sec. 3.1. In Sec. 3.2, we derive the recursive internal computations of the CRU with the resulting recurrent architecture summarized in Sec. 3.3. We then develop an efficient CRU parameterization scheme that affects modeling flexibility and run time (Sec. 3.4). Finally, in Sec. 3.5, we describe how to train a CRU.
3.1. Overview of proposed Approach

The CRU addresses the challenge of modeling a time series \( x_T = \{ x_t | t \in T = \{ t_0, t_1, \ldots, t_N \} \) whose observation times \( T = \{ t_0, t_1, \ldots, t_N \} \) can occur at irregular intervals.

Modeling assumptions for the latent state Unlike the discrete hidden state formulation of standard RNNs, the latent state \( z \in \mathbb{R}^M \) of a CRU has continuous dynamics, which are governed by a linear SDE

\[
dz = Az \, dt + Gd\beta, \tag{1}
\]

with time-invariant transition matrix \( A \in \mathbb{R}^{M \times M} \) and diffusion coefficient \( G \in \mathbb{R}^{M \times B} \). The integration variable \( \beta \in \mathbb{R}^{B} \) is a Brownian motion process with diffusion matrix \( Q \in \mathbb{R}^{B \times B} \). The CRU assumes a Gaussian observation model \( H \in \mathbb{R}^{D \times M} \) that generates noisy latent observations

\[
y_t \sim \mathcal{N}(Hz_t, (\sigma_t^{\text{obs}})^2 I), \tag{2}
\]

with observation noise \( \sigma_t^{\text{obs}} \).

Sequential processing At each time point \( t \in T \), the latent observation \( y_t \) and its elementwise latent observation noise \( \sigma_t^{\text{obs}} \) are produced by a neural network encoder \( f_\theta \),

\[
\text{encoder: } [y_t, \sigma_t^{\text{obs}}] = f_\theta(x_t), \tag{3}
\]

applied to the data \( x_t \). At each observation time, we distinguish between a prior and a posterior distribution on \( z_t \).

\[1\]We use the notation \( y_{<t} := \{ y_{t'} \text{ for } t' \in T \text{ s.t. } t' < t \} \) for the set of all latent observations before \( t \) and \( y_{\leq t} := \{ y_{t'} \text{ for } t' \in T \text{ s.t. } t' \leq t \} \) for this set including \( y_t \).

The parameters of the prior, \( \mu_t^- \) and \( \Sigma_t^- \), are computed by propagating the latent state according to Eqn. (1) (we call this the “prediction step”) while the parameters of the posterior, \( \mu_t^+ \) and \( \Sigma_t^+ \), are computed with a Bayesian update (which we call the “update step”). The optimal prediction and update step will be derived in closed form in Sec. 3.2.

Fig. 1 gives an overview of the CRU from a Kalman filtering perspective: observations (green) are mapped by the encoder into a latent observation space (orange). The mean and variance of the latent state is inferred using the predict and update step of the continuous-discrete Kalman filter (red). Finally, a decoder maps the posterior parameters to the desired output space along with elementwise uncertainties.

\[
\text{decoder: } [\alpha_t, \sigma_t^{\text{out}}] = g_\phi(\mu_t^+, \Sigma_t^+). \tag{6}
\]

3.2. Continuous-discrete Kalman filter

The continuous-discrete Kalman filter (Jazwinski, 1970) is the optimal state estimator for a continuous state space model (Eqn. (1)) with a discrete-time Gaussian observation process (Eqn. (2)). This version of the Kalman filter allows modelling observations of a continuous process at potentially arbitrary but discrete observation times. Given the latent observations, the posterior distribution of the latent state (Eqn. (5)) is computed recursively, alternating between a predict and an update step. These steps are derived next.

3.2.1. PREDICTION STEP

Between observation times, the prior density describes the evolution of \( z_t \). It is governed by the SDE in Eqn. (1), which has an analytical solution for linear, time-invariant systems as considered here. To compute the prior at time \( t \), we assume that the posterior parameters \( \mu_{\tau(t)}^- \) and \( \Sigma_{\tau(t)}^- \) at the last observation,

\[
\tau(t) := \max\{ t' \in T \text{ s.t. } t' < t \}, \tag{7}
\]
Modeling Irregular Time Series with Continuous Recurrent Units

Figure 2. The internal hidden states of a CRU cell are the posterior mean and variance \( \mu_t^+ \) and \( \Sigma_t^+ \) of the continuous state variable \( z_t \). They are computed recursively according to Algorithm 1.

We next describe parameterization choices for CRU which lead to more expressive modeling capacity (Sec. 3.4.1) and faster computation (Sec. 3.4.2) of the state equations. Finally, we present in Sec. 3.5 how to train a CRU.

### 3.4. Flexible and Efficient parameterization of the CRU

The linearity assumption of the continuous-discrete Kalman filter is advantageous, as it leads to optimal closed-form computations. However, it also limits the expressiveness of the model. The idea of CRU is that the modelling flexibility of the encoder and decoder mitigates this limitation and that the dimensionality of the state space is large enough for a linearly evolving latent state to lead to expressive relationships between input and output sequences.

On the other hand, the dimensionality of the latent state cannot be too large in practice as it affects the runtime of the matrix inversion in Eqn. (13) and the matrix exponential in Eqn. (8). To address this trade-off between modeling.
flexibility and efficient computation, we make certain parameterization choices for the CRU. In Sec. 3.4.1, we describe a locally linear transition model, which maintains the closed form updates of Sec. 3.2 while making the model more flexible. In Sec. 3.4.2, we develop f-CRU, a version of the CRU with a novel parameterization of the transition matrices in terms of their eigenspaces. The resulting model has less modeling flexibility than the CRU but is significantly faster to train and amenable to larger state spaces.

3.4.1. Locally Linear State Transitions

A locally linear transition model increases the modeling flexibility of CRU while maintaining closed form computation of the predict and update steps in Sec. 3.2. Similar approaches have been used in deep Kalman architectures (Karl et al., 2016; Fraccaro et al., 2017). We employ the parameterization strategy of Becker et al. (2019) and design the transition matrix $A_t$ at time $t$ as a weighted average of $K$ parameterized basis matrices. The weighting coefficients $\alpha_t^{(k)}$ for $k \in \{1...K\}$ are obtained from the current posterior mean $\mu_t^+$ by a neural network $w_\psi$ with softmax output,

$$A_t = \sum_{k=1}^{K} \alpha_t^{(k)} A^{(k)}, \quad \text{with } \alpha_t = w_\psi(\mu_t^+). \quad (15)$$

To reduce the number of parameters, each basis matrix consists of four banded matrices with bandwidth $b$. In addition, we assume a diagonal diffusion matrix $Q$ whose vector of diagonal entries $q$ is a time-invariant learnable parameter. The diffusion coefficient $G$ of the SDE in Eqn. (1) is fixed at the identity matrix, i.e. $G = I$. This is not restrictive as $G$ only occurs in combination with the learnable parameter vector $\mathbf{q}$ (Eqn. (8)), which is unconstrained.

3.4.2. Efficient Implementation

The runtime of the CRU is dominated by two operations: the matrix inversion in the computation of the Kalman gain (Eqn. (13)) and the matrix exponential in the prediction step (Eqn. (8)). As in Becker et al. (2019), there is a trade-off between modeling flexibility and runtime when choosing how to parametrize the model. Becker et al. (2019) use certain factorization assumptions on the state covariance $\Sigma_t$ and the observation model $H$ that increase speed and stability by simplifying the matrix inversion. CRU also benefits from these assumptions, which are detailed in Appendix B. However, the CRU has an additional computational bottleneck, namely the matrix exponential in Eqn. (8).

In this section, we develop fast CRU (f-CRU), a model variant that benefits from an efficient implementation of the prediction step. f-CRU bypasses the computation of the matrix exponential by allowing only commutative and symmetric base matrices $A^{(k)}$ with related eigenspaces. While this limits the modeling flexibility, it reduces the runtime of the matrix exponential from complexity $O(n^3)$ to matrix multiplication and elementwise operations.

To avoid having to compute an eigenvalue decomposition, we directly parameterize the basis matrices $A^{(k)}$ in terms of their eigenvalues and eigenvectors, which enable a change of basis. In the projected space, the state transitions are diagonal and the matrix exponential simplifies to the elementwise exponential function. By allowing only commutative, symmetric basis matrices, we can ensure that the matrix exponential in the projected space is invariant to the order in which the $A^{(k)}$ are summed (Eqn. (15)).

In detail, we assume diagonalizable basis matrices $\{A^{(k)}\}_{k=1...K}$ that share the same orthogonal eigenvectors. That is to say, for all $k \in \{1...K\}$ we have $A^{(k)} = ED^{(k)}E^T$ where $D^{(k)}$ is a diagonal matrix whose $i$-th diagonal entry is the eigenvalue of $A^{(k)}$ corresponding to the eigenvector in the $i$-th column of $E$. When using $E$ to perform a change of basis on the latent state $z_t$, the SDE of the transformed state vector $w$ has diagonal transitions $D$. The prior mean at time $t$ simplifies to,

$$\mu_t^- = E \exp \left( (t - \tau(t)) \sum_{k=1}^{K} \alpha_t^{(k)} D^{(k)} \right) E^T \mu_t^+, \quad (16)$$

where $\exp(\cdot)$ denotes the elementwise exponential function. We follow Rome (1969) to efficiently compute the covariance of the projected state space $\Sigma_t^w$ at time $t$, which is mapped back to the original basis of $z$ to yield the prior covariance at time $t$

$$\Sigma_t^- = E \Sigma_t^w E^T. \quad (17)$$

We provide the thorough definitions and computations of the f-CRU prior computation in Appendix A.1.2. Fig. 3 illustrates the computational gain realized by the parameterization scheme of f-CRU. In Sec. 4, we further study the speed and accuracy trade-off between CRU and f-CRU.

3.5. Training

The trainable parameters of the CRU are the neural network parameters of the encoder and decoder ($\theta$ and $\phi$), and pa-
we study CRU in comparison to other sequence models in

4.3), the task is to learn the underlying dynamics of the

f-CRU. We use the tools by Lezcano-Casado (2019).

4.1. Datasets

We study the CRU and f-CRU in comparison to various

Baseline

We study the CRU and f-CRU in comparison to various

Models

We study the CRU and f-CRU in comparison to various

Parameterizations

We study the CRU and f-CRU in comparison to various

Models

We study the CRU and f-CRU in comparison to various

Baseline

We study the CRU and f-CRU in comparison to various

Models

We study the CRU and f-CRU in comparison to various

Baseline

We study the CRU and f-CRU in comparison to various

Models

We study the CRU and f-CRU in comparison to various

Baseline

We study the CRU and f-CRU in comparison to various

Models

We study the CRU and f-CRU in comparison to various

Baseline

We study the CRU and f-CRU in comparison to various

Models
We first examine the efficacy of CRU in sequence interpolation (Details can be found in Appendix C.).

To ensure a fair comparison, we follow the setup of Becker et al. (2019) combines a continuous-time version of the decoder are trained jointly. More information and other implementation details can be found in Appendix C.7. 

For processing pendulum images, the CRU encoder and latent ODE in Rubanova et al. (2019)’s Physionet set-up. Our method can make efficient use of batching and unlike some state-of-the-art methods.

Our method can make efficient use of batching and unlike some neural ODE-based architectures, CRU does not require solving the state propagation in sync for the union of all timestamps in a minibatch. Thus, the complexity of CRU does not scale with the heterogeneity of timestamps in a minibatch. On data where timestamps vary widely across sequences (e.g. Physionet), the number of required update steps can be up to B-times more for recurrent neural ODE-architectures than for CRU, where B denotesbatchsize.

ODE-based models We also test CRU against three ODE-based models that can naturally deal with irregularly sampled time series: (1) ODE-RNN (Rubanova et al., 2019) alternates between continuous hidden state dynamics defined by an ODE and classical RNN updates at observation times. (2) Latent ODE (Chen et al. (2018), Rubanova et al. (2019)) is a generative model that uses ODE-RNN as recognition network to infer the initial value of its latent state and models the state evolution with an ODE. (3) GRU-ODE-B (Brouwer et al., 2019) combines a continuous-time version of GRUs with a discrete update network.

Implementation details For a fair comparison, we use the same latent state dimension for all approaches (M = 30 for pendulum, M = 20 for physionet and M = 10 for USHCN), except for GRU, where we increase the latent state size such that the number of parameters is comparable to CRU. For the Physionet and USHCN experiments, the encoder and decoder architecture of the CRU mimic that of the latent ODE in Rubanova et al. (2019)’s Physionet set-up. (Details can be found in Appendix C.)

For processing pendulum images, the CRU encoder and decoder are a convolutional architecture (see Appendix C.7). To ensure a fair comparison, we follow the setup of Becker et al. (2019) and use the same encoder and decoder architecture as for RKN, CRU and f-CRU to give the same feature extraction capabilities to the other baseline models: the baseline models are applied to the latent observations that the encoder produces from the inputs and the decoder maps the baseline outputs to the target output. In this augmented framework, the encoder, the baseline model, and the decoder are trained jointly. More information and other implementation details can be found in Appendix C.

4.3. Results on Sequence Interpolation

We first examine the efficacy of CRU in sequence interpolation on pendulum images, USHCN and Physionet. The task consists of inferring the entire sequence \( s_T = x_T \) based on a subset of observations \( x_S \) where \( S \subseteq T \). In the pendulum interpolation task, \( S \) contains half of the total time points sampled at random. For USHCN and Physionet, we follow the Physionet set-up in Rubanova et al. (2019), where reconstruction is based on the entire time series, i.e. \( S = T \).

Table 1. Test MSE (mean ± std) and runtime (average seconds/epoch) for interpolation on USHCN and Physionet.

| Model       | USHCN   | Physionet | USHCN   | Physionet | USHCN   | Physionet |
|-------------|---------|-----------|---------|-----------|---------|-----------|
| Interpolation MSE (×10⁻²) |         |           |         |           |         |           |
| GRU-D       | 0.944 ± 0.011 | 0.338* ± 0.027 | 1.718 ± 0.015 | 0.873 ± 0.071 | 292 | 5736 |
| Latent ODE  | 1.798 ± 0.009 | 0.212* ± 0.027 | 2.034 ± 0.005 | 0.725 ± 0.072 | 110 | 791 |
| ODE-RNN     | 0.831 ± 0.008 | 0.236* ± 0.009 | 1.955 ± 0.466 | 0.467 ± 0.006 | 81 | 299 |
| GRU-ODE-B   | 0.841 ± 0.142 | 0.521 ± 0.038 | 5.437 ± 1.020 | 0.798 ± 0.071 | 389 | 90 |
| f-CRU (ours) | 0.013 ± 0.004 | 0.194 ± 0.007 | 1.569 ± 0.321 | 0.714 ± 0.036 | 61 | 62 |
| CRU (ours)  | 0.016 ± 0.006 | 0.182 ± 0.091 | 1.273 ± 0.066 | 0.629 ± 0.093 | 122 | 114 |

* Results from Rubanova et al. (2019).

Table 2. Test MSE × 10⁻³ (mean ± std) and runtime (average sec/epoch) on pendulum interpolation and regression.

| Model       | Interpolation | R.time | Regression |
|-------------|---------------|--------|------------|
| GRU         | 5.086 ± 0.028 | 12     | 9.435 ± 0.998 |
| GRU-Δt      | 2.073 ± 0.185 | 12     | 5.439 ± 0.988 |
| RKN         | 5.200 ± 0.051 | 20     | 8.433 ± 0.610 |
| RKN-Δt      | 1.903 ± 0.137 | 20     | 5.092 ± 0.395 |
| Latent ODE  | 15.060 ± 0.138 | 52    | 15.700 ± 2.848 |
| ODE-RNN     | 2.830 ± 0.137 | 37     | 7.256 ± 0.406 |
| GRU-ODE-B   | 9.144 ± 0.020 | 60     | 9.783 ± 3.403 |
| f-CRU       | 1.386 ± 0.162 | 29     | 6.155 ± 0.881 |
| CRU         | 0.996 ± 0.052 | 36     | 4.626 ± 1.072 |
Modeling Irregular Time Series with Continuous Recurrent Units

4.4. Results on Pendulum Angle Prediction from Images

Next, we consider a regression task on irregularly sampled pendulum image sequences. Here, each observed image $x_t$ is mapped to a continuous target variable $s_t$ representing the pendulum angle at each time step. To assess the noise robustness of the methods, the image sequences are corrupted by a correlated noise process as in (Becker et al., 2019).

Figure 4 shows how the gating mechanism of CRUs works under varying degrees of observation noise. The norm of the Kalman gain mirrors the noise process of the sample sequence. At times of high observation noise, the norm of the Kalman gain is small and consequently the state update is dominated by the history encoded in the latent state prior. In contrast, when the pendulum is clearly observed, the Kalman gain attributes high weight to the new observation. This principled handling of noise is one of the factors that can help explain the success of CRU in pendulum angle prediction. Results in terms of MSE are shown in Tab. 2. CRU outperforms existing baselines.

4.5. Results on Sequence Extrapolation

Finally, we study the performance of CRU on extrapolating sequences far beyond the observable time frame. We split the timeline into two halves $T_1 = \{t_0, ... t_k\}$ and $T_2 = \{t_{k+1}, ... t_N\}$. Models are tasked to predict all time points of the sequence $T = T_1 \cup T_2$ based on time points in $T_1$ only. In the Physionet experiment, the input consists of the first 24 hours and the target output of the first 48 hours of patient measurements. For USHCN we split the timeline into two parts of equal length $t_k = N/2$. To guide the training process, the entire observation sequence is used as target during training, except for GRU-ODE-B, where we used the training strategy proposed by the authors.

Tab. 1 reports test errors on the extrapolated test sequences $s_T$. On the Physionet data the ODE-RNN achieves the lowest errors. On climate data, CRU and f-CRU produce better results than the baselines with the flexible transition model of the CRU reaching the highest performance.

5. Conclusion

We have developed CRU, a RNN that models temporal data with non-uniform time intervals in a principled manner. It incorporates a continuous-discrete Kalman filter into an encoder-decoder structure thereby introducing temporal continuity into the hidden state and a notion of uncertainty into the network’s gating mechanism. Our empirical study finds that the gating mechanism of the CRU weights noisy and partially observed input data accurately. Our method outperforms established recurrent sequence models such as GRU on irregular sampled image data and achieves better interpolation accuracy than neural ODE-based models on challenging datasets from various domains.
References

Axelsson, P. and Gustafsson, F. Discrete-time solutions to the continuous-time differential lyapunov equation with applications to kalman filtering. IEEE Transactions on Automatic Control, 60(3):632–643, 2014.

Bayer, J. and Osendarf, C. Learning stochastic recurrent networks. arXiv preprint arXiv:1411.7610, 2014.

Becker, P., Pandya, H., Gebhardt, G., Zhao, C., Taylor, C. J., and Neumann, G. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In International Conference on Machine Learning, pp. 544–552. PMLR, 2019.

Brouwer, E. D., Simm, J., Arany, A., and Moreau, Y. Gru-ode-bayes: continuous modeling of sporadically-observed time series. In Advances in Neural Information Processing Systems, pp. 7379–7390, 2019.

Cao, W., Wang, D., Li, J., Zhou, H., Li, L., and Li, Y. Brits: Bidirectional recurrent imputation for time series. Advances in Neural Information Processing Systems, 31:6775–6785, 2018.

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. Scientific reports, 8(1):1–12, 2018.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In Advances in Neural Information Processing Systems, pp. 6572–6583, 2018.

Choi, E., Xiao, C., Stewart, W. F., and Sun, J. Mime: Multi-level medical embedding of electronic health records for predictive healthcare. arXiv preprint arXiv:1810.09593, 2018.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. Advances in Neural Information Processing Systems, 28:2980–2988, 2015.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. arXiv preprint arXiv:1605.07571, 2016.

Fraccaro, M., Kamronn, S. D., Paquet, U., and Winther, O. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In Advances in Neural Information Processing Systems, 2017.

Goyal, A., Sordoni, A., Côté, M.-A., Ke, N. R., and Bengio, Y. Z-forcing: Training stochastic recurrent networks. arXiv preprint arXiv:1711.05411, 2017.

Herrera, C., Krach, F., and Teichmann, J. Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering. In International Conference on Learning Representations, 2020.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

Jazwinski, A. H. Stochastic Processes and Filtering Theory. Academic Press, 1970.

Kalman, R. E. A new approach to linear filtering and prediction problems. 1960.

Karl, M., Soelch, M., Bayer, J., and Van der Smagt, P. Deep variational bayes filters: Unsupervised learning of state space models from raw data. arXiv preprint arXiv:1605.06432, 2016.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. Neural controlled differential equations for irregular time series. arXiv preprint arXiv:2005.08926, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Krishnan, R. G., Shalit, U., and Sontag, D. Deep kalman filters. arXiv preprint arXiv:1511.05121, 2015.

Lechner, M. and Hasani, R. Learning long-term dependencies in irregularly-sampled time series. Advances in Neural Information Processing Systems, 33, 2020.

LeCun, Y., Touresky, D., Hinton, G., and Sejnowski, T. A theoretical framework for back-propagation. In Proceedings of the 1988 connectionist models summer school, volume 1, pp. 21–28, 1988.

Lezcano-Casado, M. Trivializations for gradient-based optimization on manifolds. In Advances in Neural Information Processing Systems, pp. 9154–9164, 2019.

Lipton, Z. C., Kale, D., and Wetzel, R. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In Machine learning for healthcare conference, pp. 253–270. PMLR, 2016.

Menne, M., Williams Jr, C., and Vose, R. Long-term daily climate records from stations across the contiguous united states, 2015.

Morrill, J., Salvi, C., Kidger, P., and Foster, J. Neural rough differential equations for long time series. In International Conference on Machine Learning, pp. 7829–7838. PMLR, 2021.
Mozer, M. C., Kazakov, D., and Lindsey, R. V. Discrete event, continuous time rns. *arXiv preprint arXiv:1710.04110*, 2017.

Pearlmutter, B. A. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

Pearlmutter, B. A. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural networks*, 6(5):1212–1228, 1995.

Pineda, F. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, pp. 602–611, 1987.

Rome, H. A direct solution to the linear variance equation of a time-invariant linear system. *IEEE Transactions on Automatic Control*, 14(5):592–593, 1969.

Rubanova, Y., Chen, R. T., and Duvenaud, D. Latent odes for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pp. 5320–5330, 2019.

Sato, M.-a. A learning algorithm to teach spatiotemporal patterns to recurrent neural networks. *Biological Cybernetics*, 62(3):259–263, 1990.

Schmidt, F. and Hofmann, T. Deep state space models for unconditional word generation. *arXiv preprint arXiv:1806.04550*, 2018.

Shukla, S. N. and Marlin, B. M. A survey on principles, models and methods for learning from irregularly sampled time series. *arXiv preprint arXiv:2012.00168*, 2020.

Silva, I., Moody, G., Scott, D. J., Celi, L. A., and Mark, R. G. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pp. 245–248. IEEE, 2012.
A. Prior computation

A.1. f-CRU

For an efficient implementation, we assume locally linear transitions with symmetric basis matrices \( \{A^{(k)}\}_{k=1}^{K} \) that share the same eigenvectors,

\[
A_{t} = \sum_{k=1}^{K} \alpha_{t}^{(k)} A^{(k)}, \quad \text{with} \quad A^{(k)} = ED^{(k)}E^{T}. \tag{20}
\]

A.1.1. Prior mean

We can simplify the matrix exponential in Eqn. (8) to the elementwise exponential function:

\[
\mu_{t}^{-} = \exp\left( A_{t}(t - \tau(t)) \right) \mu_{\tau(t)}^{+}
= \exp\left( \sum_{k=1}^{K} \alpha^{(k)} A^{(k)}(t - \tau(t)) \right) \mu_{\tau(t)}^{+}
= \exp\left( \sum_{k=1}^{K} \alpha^{(k)} ED^{(k)}E^{T}(t - \tau(t)) \right) \mu_{\tau(t)}^{+}
= E \exp\left( \sum_{i=1}^{K} \alpha^{(k)} D^{(k)}(t - \tau(t)) \right) E^{T} \mu_{\tau(t)}^{+} \tag{21}
\]

In the last step, we exploit a property of the matrix exponential. The exponential of diagonalizable matrices can be obtained by exponentiating each entry on the main diagonal of the matrix of eigenvalues.

A.1.2. Prior covariance

For diagonalizable transition matrices of a linear time-invariant system Rome (1969) proposes an analytical solution for the computation of the prior covariance based on the eigendecomposition of the transition matrix. Precisely, we can define a new state vector \( \tilde{w} \) with covariance \( \Sigma^{w}_{\tau(t)} \)

\[
\tilde{w}_{\tau(t)} = E^{T} z_{\tau(t)} = E^{T} \Sigma^{+}_{\tau(t)} E
\]

We further define \( \tilde{D} \) as the matrix whose \( ij \)-th element is the sum of the \( i \)-th and \( j \)-th diagonal entry of \( \sum_{k=1}^{K} D^{(k)} \),

\[
\tilde{D}_{ij} = \sum_{k=1}^{K} D_{ii}^{(k)} + D_{jj}^{(k)} \tag{23}
\]

Let \( S \) be the transformed noise component,

\[
S = E^{T} GQG^{T} E, \tag{24}
\]

then we can compute the covariance of \( \tilde{w} \) at time \( t \) with

\[
\Sigma_{t}^{w} = (S \odot \exp(\tilde{D}(t - \tau(t))) - S) \odot \tilde{D} + \Sigma_{\tau(t)}^{w} \odot \exp(\tilde{D}(t - \tau(t))) \tag{25}
\]

Mapping back to the space of \( z \), we obtain for the prior covariance of \( z \) at time \( t \)

\[
\Sigma_{t}^{-} = E \Sigma_{t}^{w} E^{T} \tag{26}
\]
A.2. CRU

A.2.1. Prior Covariance

The integral in the prior covariance (Eqn. (8)) can be resolved analytically using matrix fraction decomposition (Axelsson & Gustafsson, 2014). The idea of the method consists in computing the integral by solving a matrix valued ODE. We compute the matrix exponential of the matrix

\[
B = \begin{pmatrix}
A & GQG^T \\
0 & -A^T
\end{pmatrix}
\]

where \(M_1, M_2, M_3\) and \(0\) are of dimension \(M \times M\). The prior covariance matrix at time \(t\) is then given by

\[
\Sigma_t = \exp\left(A(t - \tau(t))\right) \Sigma_t^{\tau(t)} \exp\left(A(t - \tau(t))\right)^T + M_2 \exp\left(A(t - \tau(t))\right)^T
\]

B. Approximations

In their work on RKNs, Becker et al. (2019) exploit assumptions on the structure of components of the Kalman filter to simplify computation. In particular, they reduce the matrix inversion in the Kalman gain to elementwise operations. We also exploit these approximations the CRU. The following summarizes the approximations by Becker et al. (2019) and the resulting simplified update equations. We refer to Becker et al. (2019) for detailed derivations.

Observation model The dimension of the latent state \(M\) is twice of the dimension of the latent observation space \(D\), i.e. \(M = 2D\). The observation model links both spaces and is fixed at \(H = [I_D, 0_D]\) where \(I_D\) denotes the \(D\)-dimensional identity matrix and \(0_D\) the \(D\)-dimensional zero matrix. The idea is to split the latent state into an observed part, which extracts information directly from the observations and a memory part, which encodes features inferred over time.

State covariance The covariance matrix of the latent state \(\Sigma_t\) is built of diagonal blocks \(\Sigma_t^u, \Sigma_t^l, \Sigma_t^s\), whose vectors of diagonal entries are denoted by \(\sigma_t^u, \sigma_t^l, \sigma_t^s\), respectively.

\[
\Sigma_t = \begin{pmatrix}
\Sigma_t^u & \Sigma_t^s \\
\Sigma_t^s & \Sigma_t^l
\end{pmatrix}
\]

The observation part of the latent state is thus only correlated with the corresponding memory part. The argument behind this strong assumption is that the free parameters in the neural encoder and decoder suffice to find a representation where the above limitations hold.

Simplified update step The Kalman gain simplifies to a structure with two diagonal blocks of size \(D \times D\), i.e. \(K_t = [K_t^u, K_t^l]^T\). The vector of diagonal entries \(k_t^u, k_t^l\) can be computed with element wise division (\(\oslash\))

\[
k_t^u = \sigma_t^{u,-} \oslash (\sigma_t^{u,-} + \sigma_t^{obs}) \quad k_t^l = \sigma_t^{s,-} \oslash (\sigma_t^{u,-} + \sigma_t^{obs})
\]

The posterior mean update then simplifies to

\[
\mu_t^{+} = \mu_t^{-} + \begin{bmatrix} k_t^u \\ k_t^l \end{bmatrix} \oslash \begin{bmatrix} y_t - \mu_t^{u,-} \\ y_t - \mu_t^{l,-} \end{bmatrix}
\]

where \(\mu_t^{u}\) and \(\mu_t^{l}\) denote the upper and lower part of the prior mean respectively, i.e. \(\mu_t^{-} = [\mu_t^{u,-}, \mu_t^{l,-}]^T\). The update of the posterior covariance reduces to

\[
\sigma_t^{u,+} = (I_m - k_t^u) \oslash \sigma_t^{u,-}
\]
\[ \sigma_{t}^{s,+} = (1_m - k^s_t) \odot \sigma_{t}^{s,-} \quad (33) \]

\[ \sigma_{t}^{l,+} = \sigma_{t}^{l,-} - k^l_t \odot \sigma_{t}^{s,-} \quad (34) \]

where \( \odot \) denotes elementwise multiplication.

### C. Implementation details

**Training** In all experiments, we train each model for 100 epochs using the Adam optimizer (Kingma & Ba, 2014). Reported MSE results are averages over 5 runs. We used a batchsize of 50 for the pendulum and USHCN data and a batchsize of 100 for Physionet. For USHCN and Physionet, we split the data into 80% train and 20% test and used 25% of the train set for validation. For the pendulum experiments, we generated 2,000 training sequences, 1,000 validation sequences and report results on a hold-out set of 1,000 sequences. The folds are reused for each compared model.

**Hyperparameters and architecture** Here, we summarize hyperparameter choices made in the empirical study with details provided in Appendices C.1 to C.7. We used the following procedure to select latent state size, number of layers, number of hidden units, and training parameters: For GRU-D, latent ODE and ODE-RNN, we use the choices optimized by Rubanova et al. (2019) in their Physionet setup. We then designed the encoder and decoder of the CRU such that the CRU has roughly the same number of parameters as the latent ODE model. We proceed analogously for hyperparameters of GRU-ODE-B: We employ the hyperparameter settings optimized by the authors on their USHCN experiment. We keep the baseline architectures fixed across experiments, except for the latent state size, which we chose separately for each experiment according to the dimensionality of the input and previous work. See Appendix C.7 for the latent state size of each experiment, which is shared across architectures. For the pendulum image set, encoder, decoder and latent space sizes of CRU follow the RKN (Becker et al., 2019) baseline on this set. To scale the methods based on neural ODEs to images, we also embedded ODE-RNN, latent ODE and GRU-ODE-B into the same encoder decoder structure as CRU. Similarly, the GRU baseline is augmented with the same encoder and decoder. This was also done in the pendulum experiments of Becker et al. (2019) and more details and justification can be found there. We found GRU-ODE-B unstable when trained jointly with an encoder and decoder on the image interpolation task and therefore trained it directly on the raw images.

**C.1. GRU-D**

We use a hidden state size of \( M = 30 \) for pendulum, \( M = 20 \) for Physionet, and \( M = 10 \) for USHCN. The other parameters are fixed across experiments. As described previously, we use hyperparameter and architecture choices by Rubanova et al. (2019), which result in 100 hidden units, a learning rate of 0.01 with a decay factor of 0.999.

**C.2. Latent ODE**

We use a latent state size of \( M = 30 \) for pendulum, \( M = 20 \) for Physionet, and \( M = 10 \) for USHCN. The other parameters are fixed across experiments. We use the latent ODE with an ODE-RNN recognition model. The recognition model has a hidden state of 40 dimensions, an ODE function with 3 layers and 50 hidden units and a GRU update with 50 hidden units. The ODE function of the generative model has 3 layers with 50 hidden units. We train the method with a learning rate of 0.01 and a decay rate of 0.999.

**C.3. ODE-RNN**

We use a hidden state size of \( M = 30 \) for pendulum, \( M = 20 \) for Physionet and \( M = 10 \) for USHCN. The other parameters are fixed across experiments. We use the hyperparameter choices proposed by Rubanova et al. (2019). Notably, we use 100 hidden units and a learning rate of 0.01 with decay rate of 0.999.

**C.4. GRU-ODE-B**

We use a latent state size of \( M = 30 \) for pendulum, \( M = 20 \) for Physionet and \( M = 10 \) for USHCN. The other parameters are fixed across experiments. As outlined above, we keep the hyperparameters determined by the authors in their USHCN
experiment. That is to say, we use 50 hidden units, a learning rate of 0.001, weight decay of 0.0001, and a dropout rate of 0.2. The \( f_{\text{reg}} \) function of the GRU-Bayes component has 10 hidden units, the \( f_{\text{obs}} \) mapping has 25 hidden units.

C.5. RKN

We use a hidden state size of \( M = 30 \) for pendulum experiments. We use the architecture and hyperparameter choices proposed by Becker et al. (2019), which we also employ for CRU in the pendulum experiments. We refer to Appendix C.7 for details. For the time-aware variant, RKN-\( \Delta_t \), we feed the time gaps as additional input to the transition net that learns weights \( \alpha_{\tau(t)} \) for the basis matrices (Eqn. (16)), i.e. \( \alpha_{\tau(t)} = w_{\psi}([t - \tau(t), \mu_{\tau(t)}^+]) \).

C.6. GRU

To test GRU on image data, we embed a GRU cell in the encoder-decoder architecture employed for RKN and CRU as in Becker et al. (2019). To make parameter sizes comparable, we increased the hidden state size to 75 for GRU and GRU-\( \Delta_t \).

For GRU-\( \Delta_t \), the time gap between observations is concatenated to the input of the GRU cell.

C.7. CRU and f-CRU

We trained CRU with an Adam optimizer (Kingma & Ba, 2014) on the Gaussian negative log likelihood (Eqn. (18)) for Physionet, USHCN and pendulum angle prediction and on the loss of Eqn. (19) for the pendulum interpolation task. On the validation set in the Pendulum interpolation experiment, we found a learning rate of 0.001 to work best for CRU and a slightly higher learning rate of 0.005 for f-CRU. We kept this choice for all datasets throughout all other experiments. (Through hyperparameter optimization on each dataset separately, the experimental results might improve further.) Gradient clipping was used on USHCN and Physionet.

The initial conditions for the latent state are set to \( \mu_{t_0}^- = 0 \) and \( \Sigma_{t_0}^- = 10 \cdot I \). We found an initialization of the transitions such that the prior mean is close to the posterior mean of the previous time step crucial for performance and stability. Thus, we initialized the basis matrices \( \{A^{(k)}\}_{k=1}^K \) filled with zeros to start off with a prediction step of \( \mu_{t_0}^+ = I \mu_{t_0}^- \). Equivalently, we chose \( E = I \) and \( D^{(k)} = 1e^{-5} \cdot I, \forall k = 1...K \) for the f-CRU initialization. Missing features are zero-encoded and masked out in the negative loglikelihood (NLL) and MSE computation. For all experiments, we used a transition net \( w_{\psi} \) with one linear layer and softmax output.

The CRU architecture used in each experiment is explicitly summarized next.

C.7.1. Pendulum Interpolation

**Continuous-discrete Kalman filter**

- Latent observation dimension: 15
- Latent state dimension: 30
- Number of basis matrices: 15
- Bandwidth (for CRU): 3

**Encoder:** 2 convolution, 1 fully connected, linear output

- Convolution, 12 channels, 5 \( \times \) 5 kernel, padding 2, ReLU, max pooling with 2 \( \times \) 2 kernel and 2 \( \times \) 2 stride
- Convolution, 12 channels, 3 \( \times \) 3 kernel, padding 1, 2 \( \times \) 2 stride, ReLU, max pooling with 2 \( \times \) 2 kernel and 2 \( \times \) 2 stride
- Fully-connected, 30 neurons, ReLU
- Linear output for latent observation; linear output, elu+1 activation for latent observation variance

**Decoder output sequence \( o_T \):** 1 fully-connected, 3 Transposed convolution

- Fully connected, 144 neurons, ReLU
- Transposed convolution, 16 channels, 5 \( \times \) 5 kernel, padding 2, 4 \( \times \) 4 stride, ReLU
• Transposed convolution, 12 channels, 3 × 3 kernel, padding 1, 2 × 2 stride, ReLU
• Transposed convolution, 1 channel, 2 × 2 kernel, padding 5, 2 × 2 stride, sigmoid activation

C.7.2. PENDULUM REGRESSION
We used the same encoder and Kalman filter architecture as in the pendulum interpolation task.

Decoder output sequence $o_T$: 1 fully-connected, linear output

• Fully connected, 30 neurons, Tanh
• Linear output

Decoder output variance $\sigma_{oT}^2$: 1 fully-connected, linear output

• Fully connected, 30 neurons, Tanh
• Linear output, elu+1 activation

C.7.3. USHCN

Continuous-discrete Kalman filter

• Latent observation dimension: 5
• Latent state dimension: 10
• Number of basis matrices: 15
• Bandwidth (for CRU): 3

Encoder: 3 fully connected, linear output

• Fully connected, 50 neurons, ReLU, layer normalization
• Fully connected, 50 neurons, ReLU, layer normalization
• Fully connected, 50 neurons, ReLU, layer normalization
• Linear output for latent observation; linear output, square activation for latent observation variance

Decoder output sequence $o_T$: 3 fully connected, linear output

• Fully connected, 50 neurons, ReLU, layer normalization
• Fully connected, 50 neurons, ReLU, layer normalization
• Fully connected, 50 neurons, ReLU, layer normalization
• Linear output for latent observation; linear output, square activation for latent observation variance

Decoder output variance $\sigma_{oT}^2$: 1 fully-connected, linear output

• Fully connected, 50 neurons, ReLU, layer normalization
• Linear output, square activation

C.7.4. PHYSIONET
We used the same encoder and decoder architecture as in the USHCN experiment.
Continuous-discrete Kalman filter

- Latent observation dimension: 10
- Latent state dimension: 20
- Number of basis matrices: 20
- Bandwidth (for CRU): 10

D. Data preprocessing

D.1. USHCN

Daily weather records can be downloaded at https://cdiac.ess-dive.lbl.gov/ftp/ushcn_daily/. We remove observations with a bad quality flag as in Brouwer et al. (2019). However, unlike Brouwer et al. (2019) we are interested in long term extrapolation and thus, select a different time window of four years from 1990 to 1993. We keep only centers that start reporting before 1990 and end reporting after 1993. We split the remaining 1168 centers into 60% train 20% validation and 20% test set. For each set, we remove measurements that are more than four standard deviations away from the set mean and normalize each feature to be in the [0,1] interval individually per set. For the baselines, we apply the time scaling strategies of previous work: we scale timestamps to be in the [0,1] for ODE-RNN, latent ODE and GRU-D as in Rubanova et al. (2019) and feed time points unprocessed to GRU-ODE-B. For CRU, we scale the timestamps, which unit is days, by a factor of 0.3.

D.2. Physionet

The data is publicly available for download at https://physionet.org/content/challenge-2012/1.0.0/. We preprocess the data as in Rubanova et al. (2019): We discard four general descriptors reported once at admission (age, gender, height, ICU-type) and keep only the remaining set of 37 time-variant features. Time points are rounded to 6 minutes steps. We split the patients into 60% train, 20% validation and 20% test set. Lastly, we normalize each feature to be in the [0,1] interval separately per set. To mimic the training routine proposed by the authors, we scale the timestamps to the [0,1] interval for GRU-D, ODE-RNN and latent ODE, leave the timescale unchanged for GRU-ODE-B, and multiply timestamps by 0.2 for CRU.

E. Computing infrastructure

Models were trained on one Nivida TU102GL Quatro RTX 6000/8000 with 40 physical Intel Xeon Gold 6242R CPU.

F. Sample trajectory

Figure 6. f-CRU

Figure 7. Sample trajectory for the pendulum interpolation task: The f-CRU predicts images precisely despite irregular intervals between image frames.