KeyCLD: Learning Constrained Lagrangian Dynamics in Keypoint Coordinates from Images

Rembert Daems^{1,2,3}
Rembert.Daems@UGent.be

Jeroen Taets^{1,3}
Jeroen.Taets@UGent.be

Francis wyffels^{2}
Francis.wyffels@UGent.be

Guillaume Crevecoeur^{1,3}
Guillaume.Crevecoeur@UGent.be

^{1} Ghent University – D2LAB
^{2} IDLab-AIRO – Ghent University – imec, Belgium
^{3} FlandersMake@UGent – corelab MIRO

Abstract

We present KeyCLD, a framework to learn Lagrangian dynamics from images. Learned keypoints represent semantic landmarks in images and can directly represent state dynamics. We show that interpreting this state as Cartesian coordinates, coupled with explicit holonomic constraints, allows expressing the dynamics with a constrained Lagrangian. KeyCLD is trained unsupervised end-to-end on sequences of images. Our method explicitly models the mass matrix, potential energy and the input matrix, thus allowing energy based control. We demonstrate learning of Lagrangian dynamics from images on the `dm_control` pendulum, cartpole and acrobot environments. KeyCLD can be learned on these systems, whether they are unactuated, underactuated or fully actuated. Trained models are able to produce long-term video predictions, showing that the dynamics are accurately learned. We compare with Lag-VAE, Lag-caVAE and HGN, and investigate the benefit of the Lagrangian prior and the constraint function. KeyCLD achieves the highest valid prediction time on all benchmarks. Additionally, a very straightforward energy shaping controller is successfully applied on the fully actuated systems.

1 Introduction and Related Work

Learning dynamical models from data is a crucial aspect while striving towards intelligent agents interacting with the physical world. Understanding the dynamics and being able to predict future states is paramount for controlling autonomous systems or robots interacting with their environment. For many dynamical systems, the equations of motion can be derived from scalar functions such as the Lagrangian or Hamiltonian. This strong physics prior enables more data-efficient learning and holds energy conserving properties [1, 2, 3, 4]. Finzi et al. [5] introduced learning of Lagrangian or Hamiltonian dynamics in Cartesian coordinates, with explicit constraints. Zhong et al. [6] included external input forces and energy dissipation, and introduced energy-based control by leveraging the learned energy models.

Learning dynamics from realistic images is a challenge per Lutter and Peters [7]. Most related work [1, 8, 9, 10, 11] uses VAEs for latent space dynamics. Zhong and Leonard [12] uses interpretable coordinates but requires full knowledge of the kinematic chain. Our approach uses convolutional keypoint estimators for observing states from images. Objects can be represented with one or more keypoints, fully capturing the position and orientation. Keypoints are used for object detection [13],

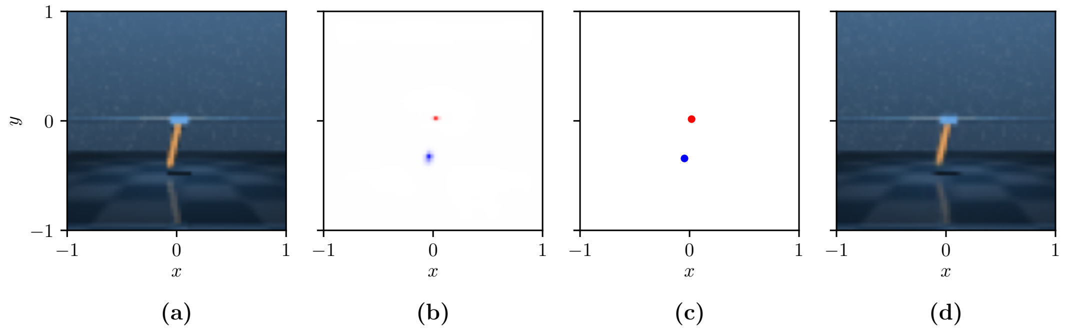


Figure 1: KeyCLD learns Lagrangian dynamics from images. **(a)** An observation of a dynamical system is processed by a keypoint estimator model. **(b)** The model represents the positions of the keypoints with a set of spatial probability heatmaps. **(c)** Cartesian coordinates are extracted using spatial softmax and used as state representations to learn Lagrangian dynamics. **(d)** The information in the keypoint coordinates bottleneck suffices for a learned renderer model to reconstruct the original observation, including background, reflections and shadows. The keypoint estimator model, Lagrangian dynamics models and renderer model are jointly learned unsupervised on sequences of images.

human pose estimation [14], control and robotic manipulation [15, 16], system identification and dynamic modelling [17]. Jakab et al. [18] learn a keypoint representation unsupervised by using it as an information bottleneck for reconstructing images.

Concretely, our work makes the following contributions. **(1)** We introduce KeyCLD, a framework to learn constrained Lagrangian dynamics from images. We are the first to use learned keypoint representations from images to learn Lagrangian dynamics. **(2)** We show how to control constrained Lagrangian dynamics in Cartesian coordinates with energy shaping, where the state is estimated from images. **(3)** KeyCLD is empirically validated on the pendulum, cartpole and acrobot systems from `dm_control` [19]. We compare quantitatively with Lag-caVAE, Lag-VAE [12] and HGN [8], and investigate the benefit of the Lagrangian prior and the constraint function. KeyCLD achieves the highest valid prediction time on all benchmarks (see Table 1).

2 Constrained Lagrangian Dynamics

2.1 Lagrangian Dynamics

For a dynamical system with m degrees of freedom, a set of independent generalized coordinates $\mathbf{q} \in \mathbb{R}^m$ represents all possible kinematic configurations of the system. The time derivatives $\dot{\mathbf{q}} \in \mathbb{R}^m$ are the velocities of the system. If the system is fully deterministic, its dynamics can be described by the equations of motion, a set of second order ordinary differential equations (ODEs):

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}(t), \dot{\mathbf{q}}(t), t, \mathbf{u}(t)) \tag{1}$$

where $\mathbf{u}(t)$ are the external forces acting on the system. From a known initial value $(\mathbf{q}, \dot{\mathbf{q}})$, we can integrate $\mathbf{f}$ through time to predict future states of the system. However, by expressing the dynamics with a Lagrangian we introduce a strong physics prior [3]:

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) \tag{2}$$

where V is the potential energy of the system and T is the kinetic energy:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} \tag{3}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{m \times m}$ is the positive semi-definite mass matrix. It is now possible to describe the dynamics with two neural networks, one for the mass matrix and one for the potential energy. Since both are only in function of $\mathbf{q}$ and not $\dot{\mathbf{q}}$, and expressing the mass matrix and potential energy is more straightforward than expressing the equations of motion. It is generally more simple to learn dynamics with this framework. In other words, adding more physics priors in the form of Lagrangian mechanics, makes learning the dynamics more robust and data-efficient [2, 3, 4, 7].

The Euler-Lagrange equations (4) allow transforming the Lagrangian into the equations of motion by solving for $\ddot{\mathbf{q}}$:

$$\frac{1}{2}\nabla \mathrm{d}\frac{\mathrm{d}}{\mathrm{dt}}\nabla_{\dot{\mathbf{q}}}\mathcal{L} - \nabla_{\mathbf{q}}\mathcal{L} = \nabla_{\mathbf{q}}W = \mathbf{g}(\mathbf{q})\mathbf{u} \tag{4}$$

where W is the external work done on the system, e.g. forces applied for control. The input matrix $\mathbf{g} \in \mathbb{R}^{m\times l}$ allows introducing external forces $\mathbf{u} \in \mathbb{R}^{l}$ for modelling any control-affine system.

2.2 Cartesian coordinates

Finzi et al. [5] showed that expressing Lagrangian mechanics in Cartesian coordinates $\mathbf{x} \in \mathbb{R}^{k}$ instead of independent generalized coordinates $\mathbf{q} \in \mathbb{R}^{m}$ has several advantages:

$$\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\dot{\mathbf{x}}^{\top}\mathbf{M}\dot{\mathbf{x}} - V(\mathbf{x}) \tag{5}$$

The mass matrix $\mathbf{M}$ no longer changes in function of the state, and is thus static. This means that a neural network is no longer required to model the mass matrix, simply the values in the matrix itself are optimized. Also the input matrix $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^{k\times l}$ is now in function of $\mathbf{x}$. Because we are now expressing the system in Cartesian coordinates we additionally need a set of n holonomic constraint functions $\mathbf{\Phi}(\mathbf{x}) : \mathbb{R}^{k} \to \mathbb{R}^{n}$. These guarantee a valid configuration of the system, and a correct number of degrees of freedom: $m = k - n$. Incorporating the constraint function leads to the following equations of motion (see Appendix B for the full derivation):

$$\begin{aligned} \mathbf{f} &= -\nabla_{\mathbf{x}}V + \mathbf{g}\mathbf{u} \\ \ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f} - \mathbf{M}^{-1}D\mathbf{\Phi}^{\top}\left[D\mathbf{\Phi}\mathbf{M}^{-1}D\mathbf{\Phi}^{\top}\right]^{-1}&\left[D\mathbf{\Phi}\mathbf{M}^{-1}\mathbf{f} + \langle D^{2}\mathbf{\Phi}, \dot{\mathbf{x}}\rangle\dot{\mathbf{x}}\right] \end{aligned} \tag{6}$$

with D being the Jacobian operator. Since time derivatives of functions modelled with neural networks are no longer present, equation (6) can be easily implemented (see Appendix F for details).

3 Learning Lagrangian dynamics from images

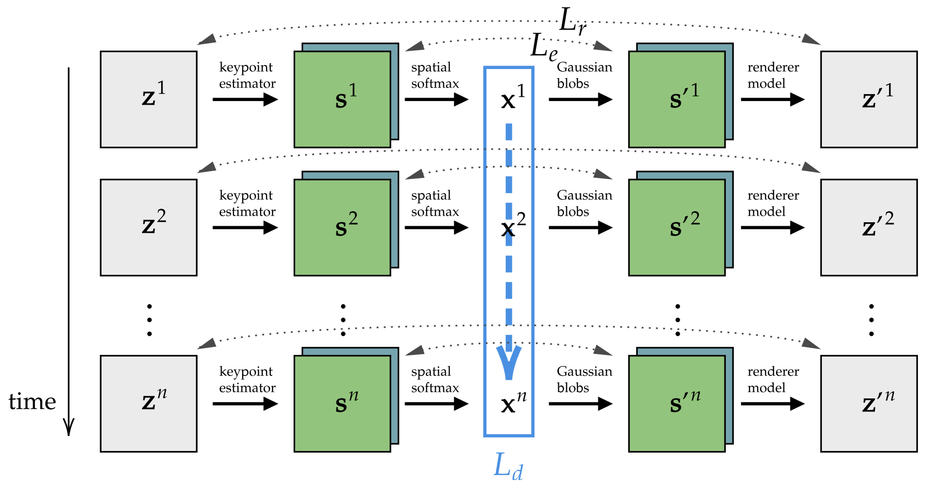


Figure 2: Schematic overview of training KeyCLD. A sequence of n images $\{\mathbf{z}^i\}, i \in \{1, \ldots, n\}$, is processed by the keypoint estimator model, returning heatmaps $\{\mathbf{s}^i\}$ representing spatial probabilities of the keypoints. $\mathbf{s}^i$ consists of m heatmaps $\mathbf{s}^i_k$, one for every keypoint $\mathbf{x}^i_k, k \in \{1, \ldots, m\}$. Spatial softmax is used to extract the Cartesian coordinates of the keypoints, and all keypoints are concatenated in the state vector $\mathbf{x}^i$. $\mathbf{x}^i$ is transformed back to a spatial representation $\mathbf{s}'^i$ using Gaussian blobs. This prior is encouraged on the keypoint estimator model by a binary cross-entropy loss L_e between $\mathbf{s}^i$ and $\mathbf{s}'^i$. The renderer model reconstructs images $\mathbf{z}'^i$ based on $\mathbf{s}'^i$, with reconstruction loss L_r. The dynamics loss L_d is calculated on the sequence of state vectors $\mathbf{x}^i$. Keypoint estimator model, renderer model and the dynamics models (mass matrix, potential energy and input matrix) are trained jointly with a weighted sum of the losses $L = L_r + L_e + \lambda L_d$.

We introduce the use of keypoints to learn Lagrangian dynamics from images. KeyCLD is trained unsupervised on sequences of n images $\{\mathbf{z}^i\}, i \in \{1, \dots, n\}$ and a constant input vector $\mathbf{u}$ (Fig. 2). Appendix G offers a more detailed explanation of the keypoint estimator and renderer models.

The sequence $\{\mathbf{x}^i\}$, corresponding to the sequence of given images $\{\mathbf{z}^i\}$, and the constant input $\mathbf{u}$ is used to calculate the dynamics loss. We use a central first order finite difference estimation, and project the estimated velocity on the constraints, so that the constraints are not violated:

$$\dot{\mathbf{x}}^i = \left[\boldsymbol{I} - D\boldsymbol{\Phi}(\mathbf{x}^i)^+ D\boldsymbol{\Phi}(\mathbf{x}^i)\right] \frac{\mathbf{x}^{i+1} - \mathbf{x}^{i-1}}{2h} \qquad , i \in \{2, \dots, n-1\} \tag{7}$$

where $(\cdot)^+$ signifies the Moore-Penrose pseudo-inverse and h the timestep. The equations of motion (6) are solved starting from all initial values in parallel, for ν timesteps This maximizes the learning signal obtained to learn the dynamics and leads to overlapping sequences of length ν: $\{\hat{\mathbf{x}}^{i+1}, \dots, \hat{\mathbf{x}}^{i+\nu}\}, i \in \{2, \dots, n-\nu\}$. Thus, $\hat{\mathbf{x}}^{i+j}$ is obtained by integrating j timesteps forward in time, starting from initial value $\mathbf{x}^i$, which was derived by the keypoint estimator model. All $\{\hat{\mathbf{x}}^{i+j}\}$ in all sequences are compared with their corresponding keypoint states $\{\mathbf{x}^{i+j}\}$ in an L_2 loss:

$$L_d = \sum_{i=2}^{n-\nu} \sum_{j=1}^{\nu} \left\| \mathbf{x}^{i+j} - \hat{\mathbf{x}}^{i+j} \right\|^2 \tag{8}$$

The total loss is the weighted sum of L_r, L_e and L_d, with a weighing hyperparameter λ: $L = L_r + L_e + \lambda L_d$. To conclude, the keypoint estimator model, renderer model and dynamics models (mass matrix, potential energy and input matrix) are jointly trained end-to-end on sequences of images $\{\mathbf{z}^i\}$ and constant inputs $\mathbf{u}$ with stochastic gradient descent.

4 Experiments

We adapted the pendulum, cartpole and acrobot environments from `dm_control` [19, 20] for our experiments (see Appendix O). As a metric we use the valid prediction time (VPT) score [11, 21] which measures how long the predicted images $\{\mathbf{z}'^i\}$ stay close to the groundtruth images $\{\mathbf{z}^i\}$:

$$\text{VPT} = \text{argmin}_i[\text{MSE}(\mathbf{z}'^i, \mathbf{z}^i) > \epsilon] \tag{9}$$

We present evaluations with the following ablations and baselines. **KeyCLD:** the full framework as described in Sections 2 and 3. **KeyLD:** The constraint function is omitted. **KeyODE2:** A second order neural ODE modelling the acceleration is used instead of the Lagrangian prior. The keypoint estimator and renderer model are identical to KeyCLD. **Lag-caVAE:** The model presented by Zhong and Leonard [12]. We adapted the model to the higher resolution and color images. **Lag-VAE:** The model presented by Zhong and Leonard [12]. We adapted the model to the higher resolution and color images. **HGN:** Hamiltonian Generative Network presented by Toth et al. [8].

We generate predictions of 50 frames, given the first 3 frames of the ground truth sequences to estimate the initial velocity according to equation (7). The VPT metric is calculated for the validation set and averaged. See Table 1 for an overview of results.

5 Conclusion and Limitations

We introduce the use of keypoints to learn Lagrangian dynamics from images. Learned keypoint representations derived from images are directly used as positional state vector for learning constrained Lagrangian dynamics.

The main limitations of our work are that we only consider 2D systems, where the plane of the system is parallel with the camera plane. Furthermore we do not model energy dissipation and the constraint function is given.

Acknowledgments and Disclosure of Funding

The authors thank Jonas Degrave, Peter De Roovere and Tom Lefebvre for many insightful discussions. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. Furthermore it was supported by Flanders Make under the SBO projects MultiSysLeCo and CADAIVISION.

Table 1: Valid prediction time (higher is better, equation (9)) in number of predicted frames (mean $\pm$ std) for the different models evaluated on the 50 sequences in the validation set. Lag-caVAE and Lag-VAE are only reported on the pendulum environment, since they are unable to model more than one moving body without segmented images. HGN is only reported on non-actuated systems, since it is incapable of modelling external forces and torques. KeyCLD achieves the best results on all benchmarks.

	# actuators	**KeyCLD**	KeyLD	KeyODE2	Lag-caVAE	Lag-VAE	HGN
Pendulum	0 (Fig. 14)	$\mathbf{43.1 \pm 9.7}$	16.4 ± 11.3	19.1 ± 6.2	0.0 ± 0.0	10.8 ± 13.8	0.2 ± 1.4
	1 (Fig. 15)	$\mathbf{39.3 \pm 9.8}$	14.9 ± 7.9	12.0 ± 4.1	0.0 ± 0.1	8.0 ± 10.2	-
Cartpole	0 (Fig. 16)	$\mathbf{39.9 \pm 7.4}$	29.8 ± 11.2	29.5 ± 9.5	-	-	0.0 ± 0.0
	1 (Fig. 17)	$\mathbf{38.4 \pm 8.7}$	28.0 ± 9.7	24.4 ± 7.9	-	-	-
	2 (Fig. 18)	$\mathbf{30.2 \pm 10.7}$	23.9 ± 9.6	17.7 ± 8.2	-	-	-
Acrobot	0 (Fig. 19)	$\mathbf{47.0 \pm 6.0}$	40.0 ± 7.9	34.3 ± 9.5	-	-	2.2 ± 6.9
	1 (Fig. 20)	$\mathbf{46.8 \pm 4.6}$	29.5 ± 6.3	33.0 ± 7.4	-	-	-
	2 (Fig. 21)	$\mathbf{47.0 \pm 3.5}$	39.1 ± 9.9	30.8 ± 9.3	-	-	-

References

[1] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

[2] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2018.

[3] M Lutter, K Listmann, and J Peters. Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)*, pages 7718–7725. IEEE, 2019.

[4] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[5] Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. *Advances in neural information processing systems*, 33:13880–13889, 2020.

[6] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[7] Michael Lutter and Jan Peters. Combining physics and deep learning to learn continuous-time dynamics models. *arXiv e-prints*, pages arXiv–2110, 2021.

[8] Peter Toth, Danilo J Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2020.

[9] Steindor Saemundsson, Alexander Terenin, Katja Hofmann, and Marc Deisenroth. Variational integrator networks for physically structured embeddings. In *International Conference on Artificial Intelligence and Statistics*, pages 3078–3087. PMLR, 2020.

[10] Christine Allen-Blanchette, Sushant Veer, Anirudha Majumdar, and Naomi Ehrich Leonard. LagNetVip: A lagrangian neural network for video prediction. *arXiv preprint arXiv:2010.12932*, 2020.

[11] Aleksandar Botev, Andrew Jaegle, Peter Wirnsberger, Daniel Hennes, and Irina Higgins. Which priors matter? benchmarking models for learning latent dynamics. *arXiv e-prints*, pages arXiv–2111, 2021.

[12] Yaofeng Desmond Zhong and Naomi Leonard. Unsupervised learning of lagrangian dynamics from images for prediction and control. *Advances in Neural Information Processing Systems*, 33, 2020.

[13] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[14] Ce Zheng, Wenhan Wu, Taojiannan Yang, Sijie Zhu, Chen Chen, Ruixu Liu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. Deep learning-based human pose estimation: A survey. *arXiv preprint arXiv:2012.13392*, 2020.

[15] Boyuan Chen, Pieter Abbeel, and Deepak Pathak. Unsupervised learning of visual 3d keypoints for control. In *International Conference on Machine Learning*, pages 1539–1549. PMLR, 2021.

[16] Mel Vecerik, Jean-Baptiste Regli, Oleg Sushkov, David Barker, Rugile Pevceviciute, Thomas Rothörl, Raia Hadsell, Lourdes Agapito, and Jonathan Scholz. S3k: Self-supervised semantic keypoints for robotic manipulation via multi-view consistency. In *Conference on Robot Learning*, pages 449–460. PMLR, 2021.

[17] Miguel Jaques, Martin Asenov, Michael Burke, and Timothy Hospedales. Vision-based system identification and 3d keypoint discovery using dynamics constraints. *arXiv preprint arXiv:2109.05928*, 2021.

[18] Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks through conditional image generation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4020–4031, 2018.

[19] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. `dm_control`: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[21] Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis. Sympnets: Intrinsic structure-preserving symplectic networks for identifying hamiltonian systems. *Neural Networks*, 132:166–179, 2020.

[22] Cornelius Lanczos. The variational principles of mechanics. In *The Variational Principles of Mechanics*. University of Toronto press, 2020.

[23] Peter Betsch. The discrete null space method for the energy consistent integration of constrained mechanical systems: Part I: Holonomic constraints. *Computer Methods in Applied Mechanics and Engineering*, 194(50-52):5159–5190, 2005.

[24] Boyuan Chen, Kuang Huang, Sunand Raghupathi, Ishaan Chandratreya, Qiang Du, and Hod Lipson. Automated discovery of fundamental variables hidden in experimental data. *Nature Computational Science*, 2(7):433–442, 2022.

[25] Elizaveta Levina and Peter Bickel. Maximum likelihood estimation of intrinsic dimension. *Advances in neural information processing systems*, 17, 2004.

[26] L. P. Laus and J. M. Selig. Rigid body dynamics using equimomental systems of point-masses. *Acta Mechanica*, 231(1):221–236, Jan 2020. ISSN 0001-5970, 1619-6937. doi: 10.1007/s00707-019-02543-3.

[27] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[28] Zaiwang Gu, Jun Cheng, Huazhu Fu, Kang Zhou, Huaying Hao, Yitian Zhao, Tianyang Zhang, Shenghua Gao, and Jiang Liu. Ce-net: Context encoder network for 2d medical image segmentation. *IEEE transactions on medical imaging*, 38(10):2281–2292, 2019.

[29] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[30] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Holo-GAN: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7588–7597, 2019.

[31] Romeo Ortega, Arjan J Van Der Schaft, Iven Mareels, and Bernhard Maschke. Putting energy back in control. *IEEE Control Systems Magazine*, 21(2):18–33, 2001.

[32] Fabio Gomez-Estern, Romeo Ortega, Francisco R Rubio, and Javier Aracil. Stabilization of a class of underactuated mechanical systems via total energy shaping. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)*, volume 2, pages 1137–1143. IEEE, 2001.

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[34] Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in jax!, 2020. URL http://github.com/deepmind/optax.

[35] Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.

A Broader impact

A tenacious divide exists between control engineering researchers and computer science researchers working on control. Where the first would use known equations of motion for a specific class of systems and investigate system identification, the latter would strive for the most general method with no prior knowledge. We believe this is a spectrum worth exploring, and as such use strong physics priors as Lagrangian mechanics, but still model e.g. the input matrix and the potential energy with arbitrary neural networks. The broad field of model-based reinforcement learning could benefit from decades of theory and practice in classic control theory and system identification. We hope this paper could help bridge both worlds.

Using images as input is, in a broad sense, very powerful. Since camera sensors are consistently becoming cheaper and more powerful due to advancements in technology and scaling opportunities, we can leverage these rich information sources for a deeper understanding of the world our intelligent agents are acting in. Image sensors can replace and enhance multiple other sensor modalities, at a lower cost.

To conclude, this work demonstrates the ability to efficiently model and control dynamical systems that are captured by cameras, with no supervision and minimal prior knowledge. We want to stress that we have shown it is possible to learn both the Lagrangian dynamics and state estimator model from images in one end-to-end process. The complex interplay between both, often makes them the most labour intensive parts in system identification. We believe this is a gateway step in achieving reliable end-to-end learned control from pixels.

B Full derivation of the constrained Euler-Lagrange equations

The constrained Euler-Lagrange equations are expressed with a vector $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$ containing Lagrange multipliers for the constraints [5, 22]:

$$\frac{\mathrm{d}}{\mathrm{dt}}\nabla_{\dot{\mathbf{x}}}\mathcal{L}(\mathbf{x},\dot{\mathbf{x}}) - \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\dot{\mathbf{x}}) = \mathbf{g}(\mathbf{x})\mathbf{u}(t) + D\mathbf{\Phi}(\mathbf{x})^\top\boldsymbol{\lambda}(t) \tag{10}$$

with D being the Jacobian operator. Because the mass matrix is static [1], this is simplified to:

$$\mathbf{M}\ddot{\mathbf{x}} + \nabla_{\mathbf{x}}V(\mathbf{x}) = \mathbf{g}(\mathbf{x})\mathbf{u}(t) + D\mathbf{\Phi}(\mathbf{x})^\top\boldsymbol{\lambda}(t) \tag{11}$$

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f} + \mathbf{M}^{-1}D\mathbf{\Phi}(\mathbf{x})^\top\boldsymbol{\lambda}(t), \quad \mathbf{f} = -\nabla_{\mathbf{x}}V(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}(t) \tag{12}$$

Calculating twice the time derivative of the constraint conditions yields:

$$\begin{aligned} \mathbf{0} &\equiv \mathbf{\Phi}(\mathbf{x}) \\ \mathbf{0} &= \dot{\mathbf{\Phi}}(\mathbf{x}) \\ \mathbf{0} &= D\mathbf{\Phi}(\mathbf{x})\dot{\mathbf{x}} \\ \mathbf{0} &= D\dot{\mathbf{\Phi}}(\mathbf{x})\dot{\mathbf{x}} + D\mathbf{\Phi}(\mathbf{x})\ddot{\mathbf{x}} \end{aligned} \tag{13}$$

The Lagrange multipliers $\boldsymbol{\lambda}(t)$ are solved by substituting $\ddot{\mathbf{x}}$ from equation (12) in equation (13):

$$\begin{aligned} -D\dot{\mathbf{\Phi}}(\mathbf{x})\dot{\mathbf{x}} &= D\mathbf{\Phi}(\mathbf{x})\mathbf{M}^{-1}\mathbf{f} + D\mathbf{\Phi}(\mathbf{x})\mathbf{M}^{-1}D\mathbf{\Phi}(\mathbf{x})^\top\boldsymbol{\lambda}(t) \\ \boldsymbol{\lambda}(t) &= \left[D\mathbf{\Phi}(\mathbf{x})\mathbf{M}^{-1}D\mathbf{\Phi}(\mathbf{x})^\top\right]^{-1}\left[D\mathbf{\Phi}(\mathbf{x})\mathbf{M}^{-1}\mathbf{f} + D\dot{\mathbf{\Phi}}(\mathbf{x})\dot{\mathbf{x}}\right] \end{aligned} \tag{14}$$

We use the chain rule a second time to get rid of the time derivative of $D\mathbf{\Phi}(\mathbf{x})$:

$$D\dot{\mathbf{\Phi}}(\mathbf{x})\dot{\mathbf{x}} = \langle D^2\mathbf{\Phi}, \dot{\mathbf{x}}\rangle\dot{\mathbf{x}} \tag{15}$$

Substituting $\boldsymbol{\lambda}(t)$ in (12) we finally arrive at:

$$\begin{aligned} \mathbf{f} &= -\nabla_{\mathbf{x}}V + \mathbf{g}\mathbf{u} \\ \ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f} - \mathbf{M}^{-1}D\mathbf{\Phi}^\top\left[D\mathbf{\Phi}\mathbf{M}^{-1}D\mathbf{\Phi}^\top\right]^{-1}&\left[D\mathbf{\Phi}\mathbf{M}^{-1}\mathbf{f} + \langle D^2\mathbf{\Phi}, \dot{\mathbf{x}}\rangle\dot{\mathbf{x}}\right] \end{aligned} \tag{16}$$

Note that in equation (16) only the Jacobian of $\mathbf{\Phi}(\mathbf{x})$ is present. This means that there is no need to learn explicit constants in $\mathbf{\Phi}(\mathbf{x})$, such as lengths or distances between points. Rather that constant distances and lengths through time are enforced by $D\mathbf{\Phi}(\mathbf{x})\dot{\mathbf{x}} = \mathbf{0}$. We use this property to our advantage since this simplifies the learning process.

[1] In other words, the centrifugal and Coriolis forces are zero because $\dot{\mathbf{M}} = \mathbf{0}$ and $\nabla_{\mathbf{x}}\mathbf{M} = \mathbf{0}$.

C Relationship between Lagrangian and Hamiltonian

Both Lagrangian and Hamiltonian mechanics ultimately express the dynamics in terms of kinetic and potential energy. The Hamiltonian expresses the total energy of the system $H(\mathbf{q}, \mathbf{p}) = T(\mathbf{q}, \mathbf{p}) + V(\mathbf{q})$ [1, 8]. It is expressed in the position and the generalized momenta $(\mathbf{q}, \mathbf{p})$, instead of generalized velocities. Using the Legendre transformation it is possible to transform L into H or back. We focus in our work on Lagrangian mechanics because it is more general [4] and observing the momenta $\mathbf{p}$ is impossible from images. See also Botev et al. [11] for a short discussion on the differences.

D Constraints as prior knowledge

The given constraint function $\mathbf{\Phi}(\mathbf{x})$ adds extra prior information to our model. Alternatively, we could use a mapping function $\mathbf{x} = \mathbf{F}(\mathbf{q})$. This leads directly to an expression of the Lagrangian in Cartesian coordinates using $\dot{\mathbf{x}} = D\mathbf{F}(\mathbf{q})\dot{\mathbf{q}}$:

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top D\mathbf{F}(\mathbf{q})^\top \mathbf{M} D\mathbf{F}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{F}(\mathbf{q})) \tag{17}$$

from which the equations of motion can be derived using the Euler-Lagrange equations, similar to equation (6). In terms of explicit knowledge about the system, the mapping $\mathbf{x} = \mathbf{F}(\mathbf{q})$ is equivalent to the kinematic chain as required for the method of Zhong and Leonard [12]. Using the constraint function is however more general. Some systems, such as systems with closed loop kinematics, can not be expressed in generalized coordinates $\mathbf{q}$, and thus have no mapping function [23]. Furthermore, learning the constraint manifold $\mathbf{0} = \mathbf{\Phi}(\mathbf{x})$ from data with geometric manifold learning algorithms [24, 25] could be a future research direction. We therefore argue that adopting the constraint function $\mathbf{\Phi}(\mathbf{x})$ is more general and requires less explicit knowledge injected in the model.

E Rigid bodies as rigid sets of point masses

By interpreting a set of keypoints as a set of point masses, we can represent any rigid body and its corresponding kinetic and potential energy. Additional constraints are added for the pairwise distances between keypoints representing a single rigid body [5]. For 3D systems, at least four keypoints are required to represent any rigid body [26]. We focus in our work on 2D systems in a plane parallel to the camera plane. 2D rigid bodies can be expressed with a set of 2 point masses, which can further be reduced depending on the constraints and connections between bodies (see Appendix H for more detail and proof). In our framework, the keypoint model is free to choose the relative placement of keypoints on the different moving parts of the dynamic system, enabling the choice of distinct landmarks that also express the state accurately, e.g. the endpoint of a beam.

The interpretation of rigid bodies as sets of point masses allows expressing the kinetic energy as the sum of the kinetic energies of the point masses. Corresponding to equation (3), the mass matrix for a 2D system is defined as a diagonal matrix with masses m_k for every keypoint $\mathbf{x}_k$:

$$T(\dot{\mathbf{x}}) = \frac{1}{2}\dot{\mathbf{x}}^\top \mathbf{M}\dot{\mathbf{x}} = \frac{1}{2}\begin{bmatrix}\dot{\mathbf{x}}_1 & \dots & \dot{\mathbf{x}}_n\end{bmatrix}\begin{bmatrix} m_1 & 0 & \dots & 0 & 0 \\ 0 & m_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & m_n & 0 \\ 0 & 0 & \dots & 0 & m_n \end{bmatrix}\begin{bmatrix}\dot{\mathbf{x}}_1 \\ \vdots \\ \dot{\mathbf{x}}_n\end{bmatrix} \tag{18}$$

To enforce positive values, the masses are parameterized by their square root and squared.

F Implementation of constrained Euler-Lagrange equations in JAX

It could seem a daunting task to implement the derivation of the constrained Euler-Lagrange equations (6) in an autograd library. As an example, we provide an implementation in JAX [27].

```
import jax
import jax.numpy as jnp


def constraint_fn(x):
   # function that returns a vector with constraint values
   c = jnp.array([
     ...,
   ])
   return c


def mass_matrix(params, x):
   # function that returns the mass matrix
   ...
   return m


def potential_energy(params, x):
   # function that returns the potential energy
   ...
   return V


def input_matrix(params, x):
   # function that returns the input matrix
   ...
   return g


def euler_lagrange(params, x, x_t, action):
   m_inv = jnp.linalg.pinv(mass_matrix(params, x))
   f = - jax.grad(potential_energy, 1)(params, x) + input_matrix(params, x) @ action

   Dphi = jax.jacobian(constraint_fn)(x)
   DDphi = jax.jacobian(jax.jacobian(constraint_fn))(x)

   # Lagrange multiplicators:
   l = jnp.linalg.pinv(Dphi @ m_inv @ Dphi.T) @ (Dphi @ m_inv @ f + DDphi @ x_t @ x_t)
   x_tt = m_inv @ (f - Dphi.T @ l)

   return x_tt
```

G Keypoint estimation and renderer model

All images $\mathbf{z}^i$ in the sequence are processed by the keypoint estimator model, returning each a set of heatmaps $\mathbf{s}^i$ representing the spatial probabilities of keypoint positions. $\mathbf{s}^i$ consists of m heatmaps $\mathbf{s}^i_k$, one for every keypoint $\mathbf{x}^i_k, k \in \{1, \ldots, m\}$. The keypoint estimator model is a fully convolutional neural network, maintaining a spatial representation from input to output (see Fig. 3 for the detailed architecture). This contrasts with a model ending in fully connected layers regressing to the coordinates directly, where the spatial representation is lost [8, 12]. Because a fully convolutional model is equivariant to translation, it can better generalize to unseen states that are translations of seen states. Another advantage is the possibility of augmenting $\mathbf{z}$ with random transformations of the D_4 dihedral group to increase robustness and data efficiency. Because $\mathbf{s}$ can be transformed back with the inverse transformation, this augmentation is confined to the keypoint estimator model and has no effect on the dynamics.

To distill keypoint coordinates from the heatmaps, we define a Cartesian coordinate system in the image (see for example Fig. 1). Based on this definition, every pixel $\mathbf{p}$ corresponds to a point $\mathbf{x}_\mathbf{p}$ in the Cartesian space. The choice of the Cartesian coordinate system is arbitrary but is equal to the space of the dynamics $\ddot{\mathbf{x}}(\dot{\mathbf{x}}, \mathbf{x}, t, \mathbf{u})$ and the constraint function $\mathbf{\Phi}(\mathbf{x})$ (see Section 2). We use spatial softmax over all pixels $\mathbf{p} \in \mathcal{P}$ to distill the coordinates of keypoint $\mathbf{x}_k$ from its probability heatmap:

$$\mathbf{x}_k = \frac{\sum_{\mathbf{p}\in\mathcal{P}} \mathbf{x}_\mathbf{p} e^{\mathbf{s}_k(\mathbf{p})}}{\sum_{\mathbf{p}\in\mathcal{P}} e^{\mathbf{s}_k(\mathbf{p})}} \tag{19}$$

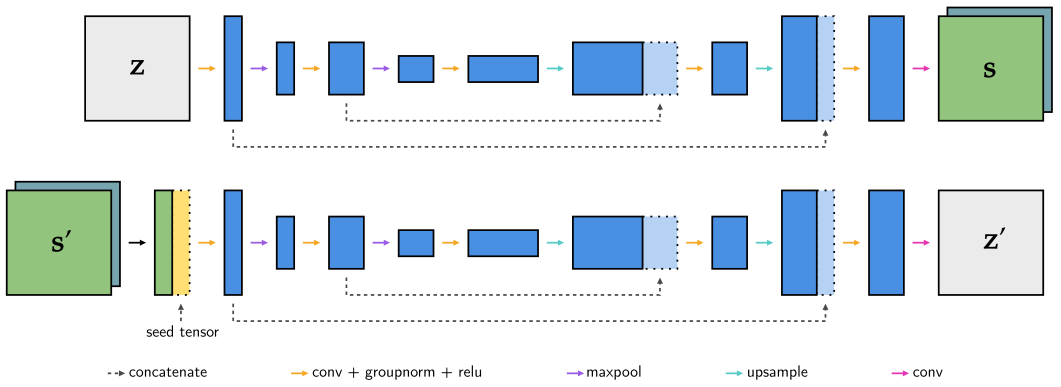


Figure 3: Visualization of the keypoint estimator (top) and renderer (bottom) model architectures. The keypoint estimator model and renderer model have similar architectures, utilizing down- and upsampling and skip connections which help increasing the receptive field [28, 29]. The renderer model learns a constant feature tensor that is concatenated with the input s'. The feature tensor provides positional information since the fully-convolutional model is translation equivariant.

Spatial softmax is differentiable, and the loss will backpropagate through the whole heatmap since $\mathbf{x}_k$ depends on all the pixels. Cartesian coordinates $\mathbf{x}_k$ of the different keypoints are concatenated in vector $\mathbf{x}$ which serves as the state representation of the system. *This compelling connection between image keypoints and Cartesian coordinates forms the basis of this work.* The keypoint estimator model serves directly as state estimator to learn constrained Lagrangian dynamics from images.

Similar to Jakab et al. [18], $\mathbf{x}$ acts as an information bottleneck, through which only the Cartesian coordinates of the keypoints flow to reconstruct the image with the renderer model. First, all $\mathbf{x}_k$ are transformed back to spatial representations $\mathbf{s}'_k$ using unnormalized Gaussian blobs, parameterized by a hyperparameter σ.

$$\mathbf{s}'_k = \exp\left(-\frac{\|\mathbf{x}_\mathbf{p} - \mathbf{x}_k\|^2}{2\sigma^2}\right) \tag{20}$$

A binary cross-entropy loss L_e is formulated over s and s' to encourage this Gaussian prior. The renderer model can more easily interpret the state in this spatial representation, as it lies closer to its semantic meaning of keypoints as semantic landmarks in the reconstructed image. The renderer model learns a constant feature tensor (inspired by Nguyen-Phuoc et al. [30]), which provides it with positional information. Since the model itself is translation equivariant, it needs positional information to reconstruct background information or specific appearances that depend on the positions of objects. See Fig. 3 for the detailed architecture.

Finally, a reconstruction loss is formulated over the reconstructed images $\mathbf{z}'^i$ and original images $\mathbf{z}^i$:

$$L_r = \sum_{i=1}^{n} \|\mathbf{z}'^i - \mathbf{z}^i\|^2 \tag{21}$$

H Rigid bodies as sets of point masses

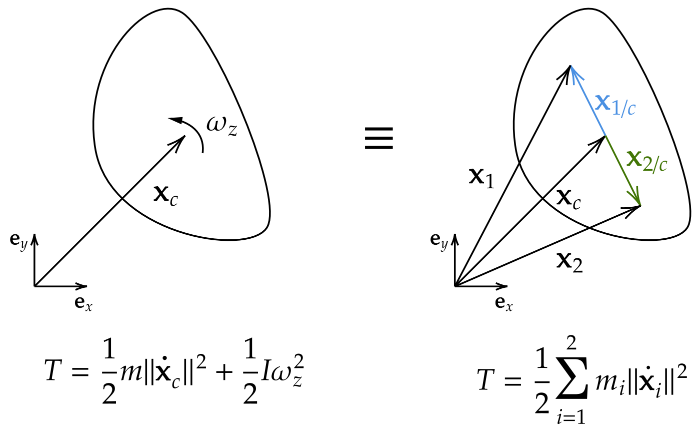


Figure 4: Any 2D rigid body with mass m and rotational inertia I is equivalent to a set of two point masses $\mathbf{x}_1$ and $\mathbf{x}_2$ with masses m_1 and m_2. The kinetic energy of the rigid body, expressed in a translational part and a rotational part, is equal to the sum of the kinetic energies of the point masses.

The position of a rigid body in 2D is fully described by the position of its center of mass $\mathbf{x}_c$ and orientation θ. Potential energy only depends on the position, thus if we want to describe the potential energy with an equivalent rigid set of point masses, two points are sufficient to fully determine $\mathbf{x}_c$ and θ. For the kinetic energy, we provide the following Theorem and proof:

Theorem 1. *For any 2D rigid body, described by its center of mass* $\mathbf{c}$*, mass* m *and rotational inertia* I*, there exists an equivalent rigid set of two point masses* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *with masses* m_1 *and* m_2*.*

Proof. To find conditions such that the kinetic energy expressed in two point masses should be equal to the rigid body representation, we start by expressing general 3D-movement:

$$\mathbf{x}_i = \mathbf{x}_c + \mathbf{x}_{i/c} \qquad , i \in \{1, 2\} \tag{22}$$

Where the vector $\mathbf{x}_c$ are the coordinates of the center of mass and the vector $\mathbf{x}_{i/c}$ is the position of the point mass relative to the center of mass. Since this relative position $\mathbf{x}_{i/c}$ has fixed length, only a rotation is possible and hence the equation of the velocity is:

$$\dot{\mathbf{x}}_i = \dot{\mathbf{x}}_c + \boldsymbol{\omega} \times \mathbf{x}_{i/c} \qquad , i \in \{1, 2\} \tag{23}$$

where $\boldsymbol{\omega}$ is the rotational velocity of the body. Substituting this in the kinetic energy of the point masses, we get:

$$\begin{aligned} T &= \frac{1}{2}\sum_{i=1}^{2} m_i \|\dot{\mathbf{x}}_c + \boldsymbol{\omega} \times \mathbf{x}_{i/c}\|^2 \\ &= \frac{1}{2}\sum_{i=1}^{2} m_i \Big(\|\dot{\mathbf{x}}_c\|^2 + \|\boldsymbol{\omega} \times \mathbf{x}_{i/c}\|^2 + 2\mathbf{x}_{i/c} \cdot \big(\dot{\mathbf{x}}_c \times \boldsymbol{\omega}\big)\Big) \end{aligned} \tag{24}$$

Where we calculated the square and used the circular shift property of the triple product on the last term.

For movement in the 2D-plane (i.e. $\boldsymbol{\omega} = \vec{\mathbf{e}}_z\omega_z$ and $\mathbf{x}_i = \vec{\mathbf{e}}_x\mathbf{x}_{i,x} + \vec{\mathbf{e}}_y\mathbf{x}_{i,y}$), this becomes:

$$\begin{aligned} T &= \frac{1}{2}\sum_{i=1}^{2} m_i \Big(\|\dot{\mathbf{x}}_c\|^2 + \|\mathbf{x}_{i/c}\|^2\omega_z^2 + 2\mathbf{x}_{i/c} \cdot \big(\dot{\mathbf{x}}_c \times \boldsymbol{\omega}\big)\Big) \\ &= \frac{1}{2}\big(m_1 + m_2\big)\|\dot{\mathbf{x}}_c\|^2 + \frac{1}{2}\big(m_1\|\mathbf{x}_{1/c}\|^2 + m_2\|\mathbf{x}_{2/c}\|^2\big)\omega_z^2 + \big(m_1\mathbf{x}_{1/c} + m_2\mathbf{x}_{2/c}\big) \cdot \big(\dot{\mathbf{x}}_c \times \boldsymbol{\omega}\big) \end{aligned} \tag{25}$$

Matching the kinetic energy of the 2 point masses (equation (25)) with that of the rigid body representation (left hand side of Fig. 4), we get following conditions:

$$\begin{cases} m = m_1 + m_2 \\ I = m_1\|\mathbf{x}_{1/c}\|^2 + m_2\|\mathbf{x}_{2/c}\|^2 \\ \mathbf{0} = m_1\mathbf{x}_{1/c} + m_2\mathbf{x}_{2/c} \end{cases} \tag{26}$$

Since the last equation is a vector equation, this gives us four equations in six unknowns (m_1,m_2,$\mathbf{x}_{1,x}$,$\mathbf{x}_{1,y}$,$\mathbf{x}_{2,x}$,$\mathbf{x}_{2,y}$), which leaves us the freedom to choose two. □

It follows from the third condition of (26) that points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_c$ should be collinear. To conclude, we can freely choose the positions of the point masses (as long as $\mathbf{x}_c$ is on the line between them), and will be able to model the rigid body as a set of two point masses. In practice, KeyCLD will freely choose the keypoint positions to be able to model the dynamics. Depending on the constraints in the system, it is possible to further reduce the number of necessary keypoints. See Appendix O for examples.

I Energy shaping control

A major argument in favor of expressing dynamics in terms of a mass matrix and potential energy is the straightforward control design via passivity based control and energy shaping [31].

Recent works of Zhong et al. [6, 12] use energy shaping in generalized coordinates. In Cartesian coordinates, energy shaping can still be used. This is easily seen from the fact that for the holonomic constraints $\mathbf{\Phi}(\mathbf{x}) \equiv 0$, we have the derivative $D\mathbf{\Phi}(\mathbf{x})\dot{\mathbf{x}} = \mathbf{0}$, which means that the constraint forces in equation (6) are perpendicular to the path and hence do no work nor influence the energy [22].

Energy shaping control makes sure that the controlled system behaves according to a potential energy $V_d(\mathbf{x})$ instead of $V(\mathbf{x})$:

$$\mathbf{u} = (\mathbf{g}^\top\mathbf{g})^{-1}\mathbf{g}^\top\left(\nabla_{\mathbf{x}}V - \nabla_{\mathbf{x}}V_d\right) - y_{\text{passive}} \tag{27}$$

where y_{passive} can be any passive output, the easiest choice being $y_{\text{passive}} = k_d\mathbf{g}^\top\dot{\mathbf{x}}$, where k_d is a tuneable control parameter. The proposed potential energy V_d should be such that:

$$\begin{aligned} \mathbf{x}^* &= \text{argmin} V_d(\mathbf{x}) \\ \mathbf{0} &= \mathbf{g}^\perp\left(\nabla_{\mathbf{x}}V - \nabla_{\mathbf{x}}V_d\right) \end{aligned} \tag{28}$$

Where $\mathbf{g}^\perp$ is the left-annihilator of $\mathbf{g}$, meaning that $\mathbf{g}^\perp\mathbf{g} = \mathbf{0}$. For fully actuated systems, the first condition of equation (28) is always met and the easiest choice is:

$$V_d(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^*)^\top k_p(\mathbf{x} - \mathbf{x}^*) \tag{29}$$

where k_p is a tuneable control parameter. The desired equilibrium position $\mathbf{x}^*$ is obtained by processing an image of the desired position with the keypoint estimator model. Finally, the passivity-based controller that is used is:

$$\mathbf{u} = (\mathbf{g}^\top\mathbf{g})^{-1}\mathbf{g}^\top\left[\nabla_{\mathbf{x}}V - k_p(\mathbf{x} - \mathbf{x}^*)\right] - k_d\mathbf{g}^\top\dot{\mathbf{x}} \tag{30}$$

Changing the behavior of the kinetic energy is also possible [32], but is left for future work. Many model-based reinforcement learning algorithms require the learning of a full neural network as controller. Whilst in this work, due to knowledge of the potential energy, we only need to tune two parameters k_p and k_d.

Fig. 5 shows results of successful swing-up of the pendulum, cartpole and acrobot system. The same control parameters $k_p = 5.0$ and $k_d = 2.0$ are used for all systems, demonstrating the generality of the control method.

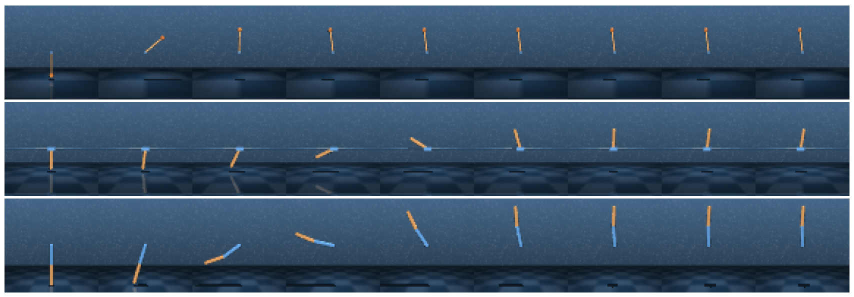


Figure 5: KeyCLD allows using energy shaping control because the learned potential energy model is available. Based on a swing-up target image $\mathbf{z}^*$, the target state $\mathbf{x}^*$ is determined by the keypoint detector model. The sequences show that all three systems can achieve the target state. The control parameters $k_p = 5.0$ and $k_d = 2.0$ are the same for all systems, demonstrating the generality of the control method.

J Learned potential energy models

Since the potential energy V is explicitly modelled, we can plot values throughout sequences of the state space. A sequence of images is processed by the learned keypoint estimator model, and the states are then used to calculate the potential energy with the learned potential energy model. Absolute values of the potential energy are irrelevant, since the potential is relative, but we gain insights by moving parts of the system separately. See Fig. 6 for results for the pendulum, Fig. 7 and 8 for the cartpole and Fig. 9 and 10 for the acrobot.

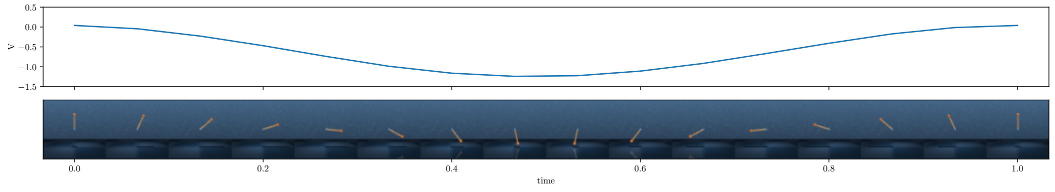


Figure 6: Potential energy of the trained KeyCLD model of the pendulum environment. The pendulum makes a full rotation. As expected, the potential energy follows a smooth sinusoidal path throughout this sequence. The maximum value is reached when the pendulum is upright, and the minimum value is reached when the pendulum is down.

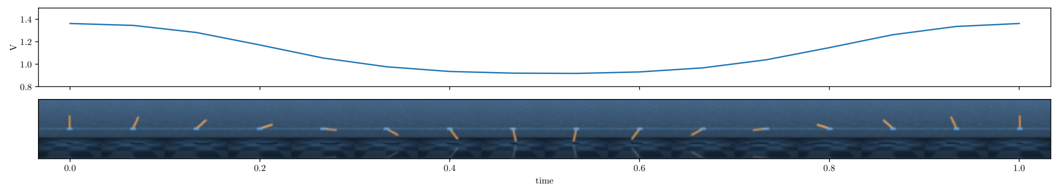


Figure 7: Potential energy of the trained KeyCLD model of the cartpole environment. The position of the cart is fixed, and the pole makes a full rotation. As expected, the potential energy follows a smooth sinusoidal path throughout this sequence.

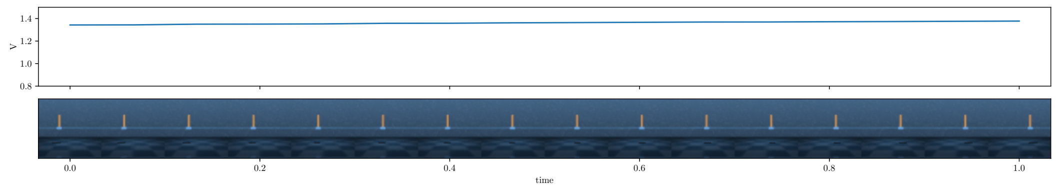


Figure 8: Potential energy of the trained KeyCLD model of the cartpole environment. The pole is fixed and the cart moves from left to right. As expected, the change in potential energy in this sequence is very low (compare to Fig. 7 with the same axis). A horizontal movement has no impact on the gravity potential.

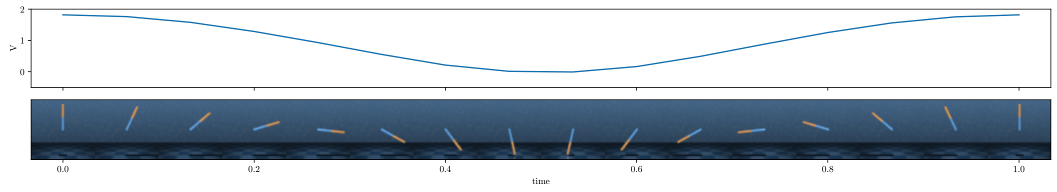


Figure 9: Potential energy of the trained KeyCLD model of the acrobot environment. The first link makes a full rotation, the second link is fixed relative to the first link. As expected, the potential energy follows a smooth sinusoidal path throughout this sequence.

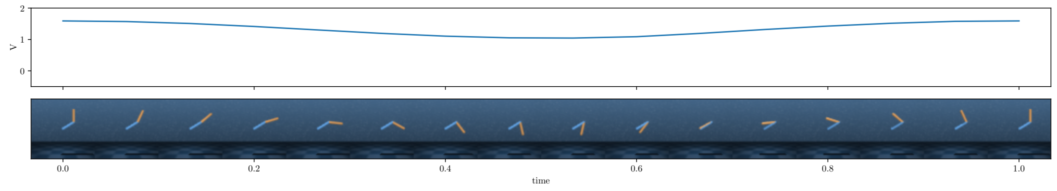


Figure 10: Potential energy of the trained KeyCLD model of the acrobot environment. The first link is fixed and the second link makes a full rotation. Again, the potential energy follows a smooth sinusoidal path throughout this sequence. Please compare with Fig. 9, where both links are moving. Here the potential energy changes less, because the first link is not moving.

K Learned input matrix models

Learning the input matrix $\mathbf{g}(\mathbf{x})$ is crucial for learning dynamics models with external inputs $\mathbf{u}$. We can visualize the vector basis that is represented by the input matrix, by drawing the vectors originating on their respective keypoints. See Fig. 11, 12 and 13 for the input matrices that are learned with our model. Each input corresponds to a vector base, visualized in different colors. The vectors multiplied by their respective input, can be interpreted as forces acting on the keypoints. These qualitative results allow further insight in our method.

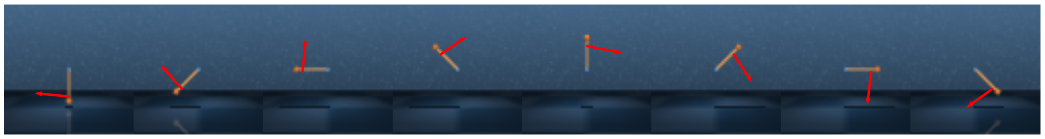


Figure 11: Visualization of the input matrix of the trained KeyCLD model of the pendulum environment. The input of this environment is a torque acting on the pendulum. In the KeyCLD framework this is correctly modelled with a force acting perpendicular on the pendulum.

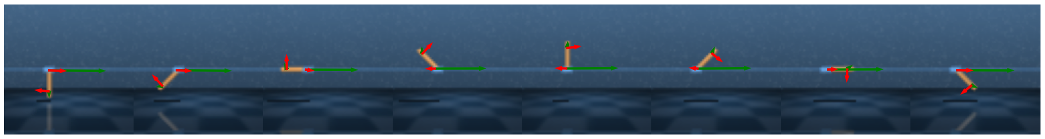


Figure 12: Visualization of the input matrix of the trained KeyCLD model of the cartpole environment. This environment has two inputs, a horizontal force acting on the cart, and a torque acting on the pole. The horizontal force corresponds to the green vectors. The first vector acting on the cart keypoint stays constant, and the second vector is negligibly small, since the horizontal force does not act on the pole. The torque corresponds to the red vectors, it is modelled with forces acting on the pole in opposite directions, such that the residual force can be zero.

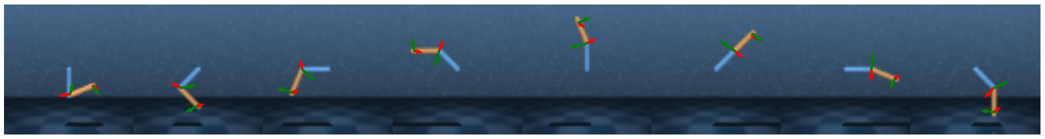


Figure 13: Visualization of the input matrix of the trained KeyCLD model of the acrobot environment. This environment has two inputs, two torques acting on each pole. The torques are modelled with opposite forces on each end of the poles, such that the residual force can be zero.

L Qualitative results

One sequence of each experiment is visualized in the paper. Please compare these qualitative results for the unactuated and actuated pendulum environment (Fig. 14, 15), unactuated, underactuated and fully actuated cartpole environment (Fig. 16, 17, 18) and unactuated, underactuated and fully actuated acrobot environment (Fig. 19, 20, 21). Every third frame of the sequence is shown.

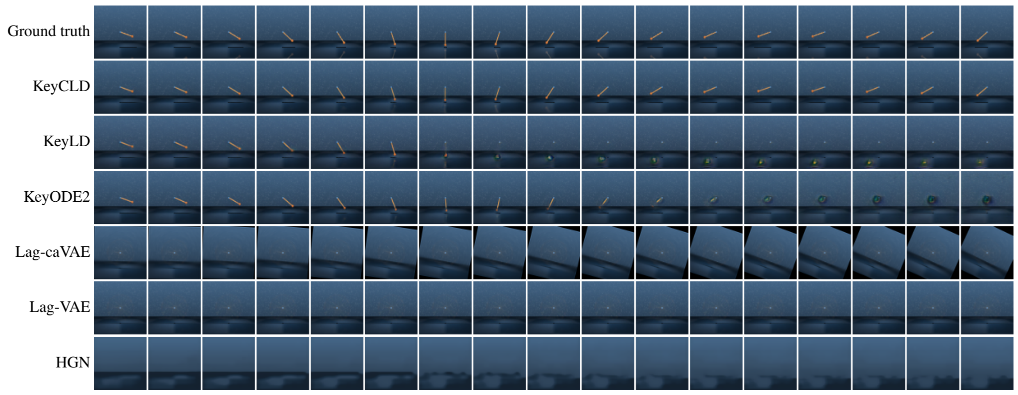


Figure 14: Future frame predictions of the unactuated pendulum. These correspond to the first row in Table 1. 50 frames are predicted based on the first three frames of the ground truth sequence to estimate the velocity. Every third frame of every sequence is shown. KeyCLD is capable of making accurate long-term predictions with minimal drift of the dynamics. Without constraint function, KeyLD is not capable of making long-term predictions. Similarly, KeyODE2 is unable of making long-term predictions. Lag-caVAE is fundamentally incapable of modelling data with background information, since the reconstructed images are explicitly rotated. Lag-VAE does not succeed in modelling moving parts in the data, and simply learns to predict static images. HGN also does not capture the dynamics and only learns the background.

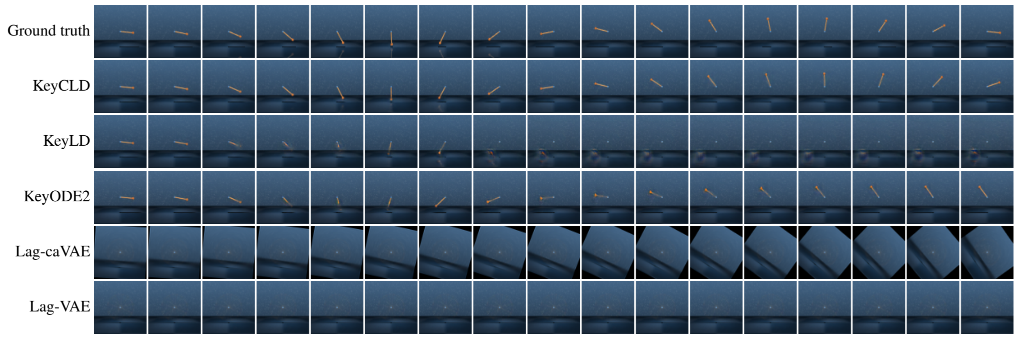


Figure 15: Actuated pendulum.

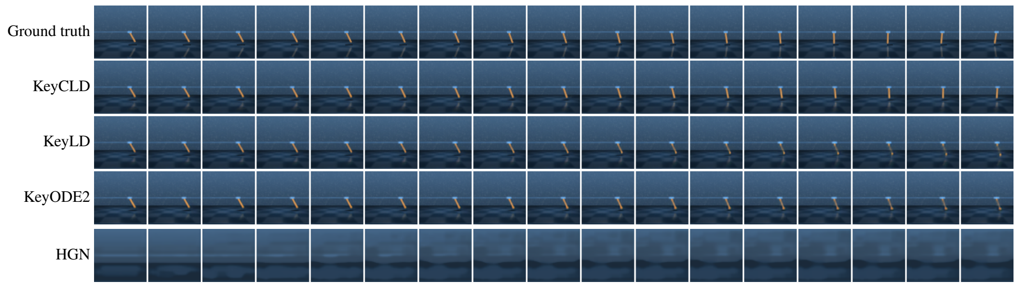


Figure 16: Unactuated cartpole.

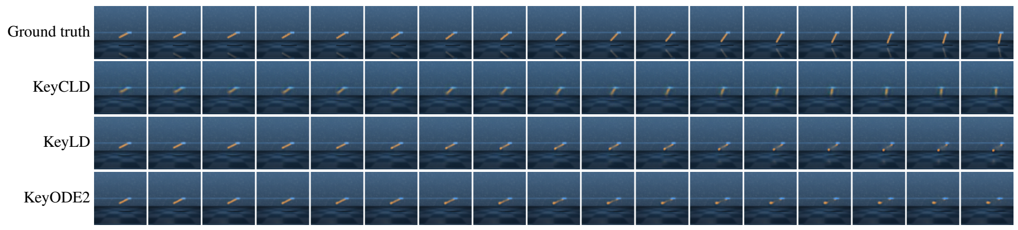


Figure 17: Underactuated cartpole.

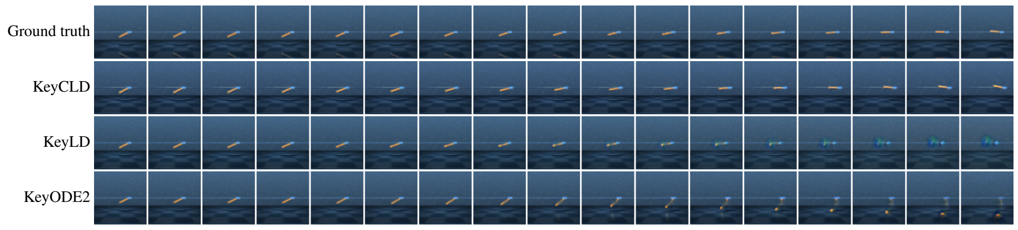


Figure 18: Fully actuated cartpole.

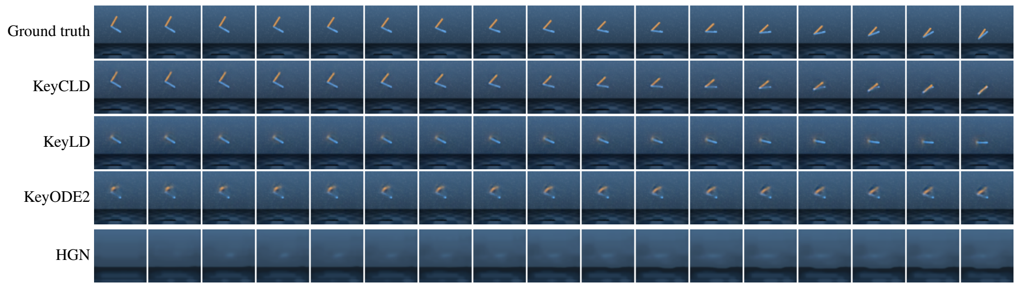


Figure 19: Unactuated acrobot.

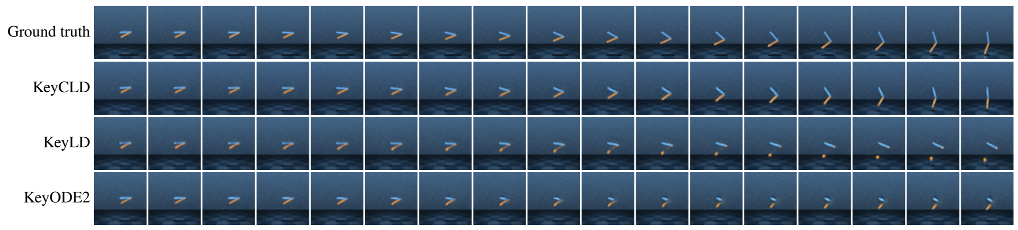


Figure 20: Underactuated acrobot.

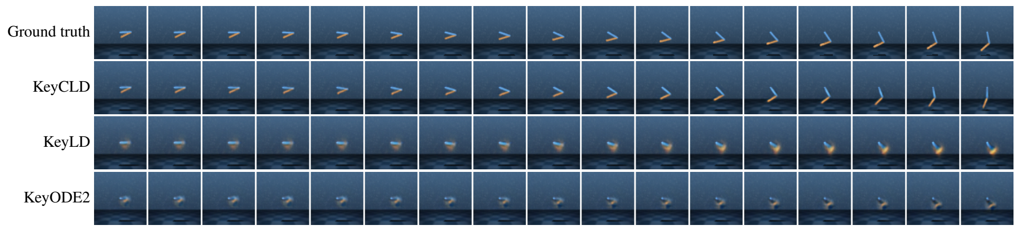


Figure 21: Fully actuated acrobot.

M Failure cases

A possible failure case is that the model learns a faulty keypoint representation that does not correspond to the given constraint function. This results in a failed model, and the training is stuck in this local minima. The encoder model will keep focussing on this erronous representation and is unable to switch to the correct keypoints. Figure 22 shows an example of this failure case. We observed this failure in a minority of the experiments. This can be mitigated by retraining the model.

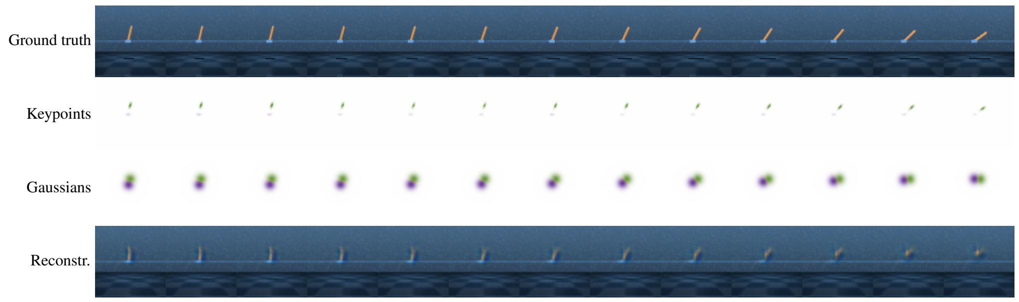


Figure 22: Failure case of KeyCLD on the cartpole environment. Keypoints are indicated in green and purple. The model erronously assigned the green keypoint to the pole. Since the given constraint function dictates that the green keypoint can only move horizontally, this results in faulty model.

N Ablating L_e

A binary cross-entropy loss L_e is formulated over $\mathbf{s}$ and $\mathbf{s}'$ to encourage the Gaussian prior. When L_e is omitted, the model can get stuck in a local minima where the encoder does not learn to predict keypoints, but rather larger regions or static values. Image 23 shows an example of this failure case.

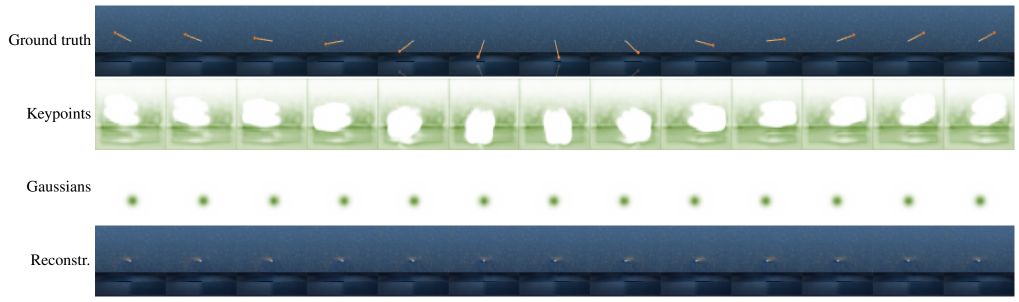


Figure 23: Omitting L_e can result in poor learning of keypoints. The encoder model does not predict distinct keypoints, but other shapes. This is effectly a local minima in the learning process, since the model is uncapable of switching to a correct representation.

O Details about the `dm_control` environments and data generation

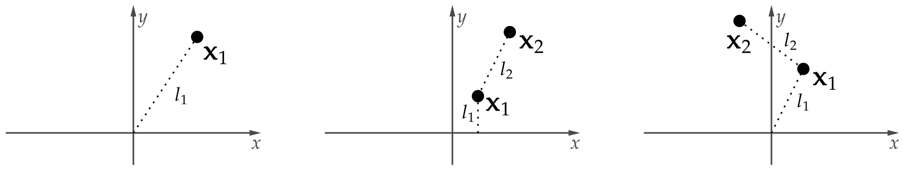
We adapted the pendulum, cartpole and acrobot environments from `dm_control` [19] implemented in MuJoCo [20]. Both are released under the Apache-2.0 license. Following changes were made to the environments to adapt them to our use-case:

Pendulum The camera was repositioned so that it is in a parallel plane to the system. Friction was removed. Torque limits of the motor are increased.

Cartpole The camera was moved further away from the system to enable a wider view, the two rails are made longer and the floor lowered so that they are not cut-off with the wider view. All friction is removed. The pole is made twice as thick, the color of the cart is changed. Torque limits are increased and actuation is added to the cart to make full actuation possible.

Acrobot The camera and system are moved a little bit upwards. The two poles are made twice as thick, and one is changed in color. Torque limits are increased and actuation is added to the upper part to make full actuation possible.

Data generation For every environment, 500 runs of 50 timesteps are generated with a 10% validation split. The initial state for every sequence is at a random position with small random velocity. The control inputs $\mathbf{u}$ are constant throughout a sequence, and uniform randomly chosen between the force and torque limits of the input. We set $\mathbf{u} = \mathbf{0}$ for 20% of the sequences. We found this helps the model to learn the dynamics better, discouraging confusion of the energy models with external actions.



$$\Phi(\mathbf{x}) = \begin{bmatrix} \|\mathbf{x}_1\|^2 - l_1 \end{bmatrix} \qquad \Phi(\mathbf{x}) = \begin{bmatrix} \mathbf{e}_y \mathbf{x}_1 - l_1 \\ \|\mathbf{x}_2 - \mathbf{x}_1\|^2 - l_2 \end{bmatrix} \qquad \Phi(\mathbf{x}) = \begin{bmatrix} \|\mathbf{x}_1\|^2 - l_1 \\ \|\mathbf{x}_2 - \mathbf{x}_1\|^2 - l_2 \end{bmatrix}$$

Figure 24: From left to right the pendulum, cartpole and acrobot `dm_control` environments. The respective constraint functions are given below each schematic.

Constraint functions The constraint function for each of the environments are given in Fig. 24. As explained in Appendix H, every rigid body needs to be represented by two keypoints. But due to the constraints it is possible to omit certain keypoints, because they do not move or coincide with other keypoints. As experimentally validated, we can thus model all three systems with a lower number of keypoints, where the number of keypoints equals the number of bodies.

Pendulum One keypoint is used to model the pendulum. The second keypoint of this rigid body can be omitted because it can be assumed to be at the origin. Due to the constraint function, this point will provide no kinetic energy since it will not move. Since the other keypoints position and mass is freely chosen, any pendulum can be modelled. The constraint function expresses that the distance l_1 from the origin to $\mathbf{x_1}$ is fixed. The value of l_1 in the implementation is irrelevant because it vanishes when taking the Jacobian.

Cartpole Two keypoints are used to model the cartpole. The constraint function expresses that $\mathbf{x}_1$ does not move in the vertical direction and the distance l_1 between $\mathbf{x}_1$ and $\mathbf{x}_2$ is constant. Again, the values of l_1 and l_2 in the implementation are irrelevant.

Acrobot Two keypoints are used to model the acrobot. The constraint function expresses that lengths l_1 and l_2 are constant through time. Again, the values are irrelevant in the implementation.

P Training hyperparameters and details

All models were trained on one NVIDIA RTX 2080 Ti GPU.

KeyCLD, KeyLD and KeyODE2 We use the Adam optimizer [33], implemented in Optax [34] with a learning rate of 3×10^{-4}. We use the exact same hyperparameters for all the environments and did not tune them individually. Dynamics loss weight $\lambda = 1$, $\sigma = 0.1$ for the Gaussian blobs in $\mathbf{s}'$. The hidden layers in the keypoint estimator and renderer model have at the first block 32 features, this increases to respectively 64 and 128 after every maxpool operation. All convolutions have kernel size 3×3, and maxpool operations scale down with factor 2 with a kernel size of 2×2. See Tables 4 and 5 for the number of parameters.

The potential energy is modelled with an MLP with two hidden layers with 32 neurons and celu activation functions [35]. The weights are initialized with a normal distribution with standard deviation 0.01. See Table 2 for the number of parameters. Likewise, the input matrix is modelled with an MLP similar to the potential energy. The outputs of this MLP are reshaped in the shape of the input matrix. See Table 3 for the number of parameters.

The KeyODE2 dynamics model is an MLP with three hidden layers with each 64 neurons. We chose a higher number of layers and neurons, to allow this model more expressivity compared to the potential energy and input matrix models of KeyCLD.

Table 2: Number of parameters of the potential energy model

	Pendulum	Cartpole	Acrobot
`Dense_0`	96	160	160
`Dense_1`	1056	1056	1056
`Dense_2`	33	33	33
Total	1185	1249	1249

Table 3: Number of parameters of the input matrix model

	Pendulum	Cartpole	Acrobot
`Dense_0`	96	160	160
`Dense_1`	1056	1056	1056
`Dense_2`	66	264	264
Total	1218	1480	1480

Table 4: Number of parameters of the keypoint encoder model

	Pendulum	Cartpole	Acrobot
`Block_0/Conv_0`	896	896	896
`Block_0/GroupNorm_0`	64	64	64
`Block_1/Conv_0`	18496	18496	18496
`Block_1/GroupNorm_0`	128	128	128
`Block_2/Conv_0`	73856	73856	73856
`Block_2/GroupNorm_0`	256	256	256
`Block_3/Conv_0`	110656	110656	110656
`Block_3/GroupNorm_0`	128	128	128
`Block_4/Conv_0`	27680	27680	27680
`Block_4/GroupNorm_0`	64	64	64
`Conv_0`	289	578	578
Total	232513	232802	232802

Lag-caVAE, Lag-VAE and HGN For the Lag-caVAE and Lag-VAE baselines, the official public codebase was used [12]. We adapted the implementation to work with the higher input resolution of 64 by 64 (instead of 32 by 32), and 3 color channels (instead of 1).

For the HGN baseline, we used the implementation that was also released by Zhong and Leonard [12]. The architecture was adapted to work with the higher input resolution of 64 by 64 (instead of 32 by 32) by adding an extra upscale layer in the decoder, and a maxpool layer and one extra convolutional layer in the encoder.

Table 5: Number of parameters of the renderer model

	Pendulum	Cartpole	Acrobot
`Seed Tensor`	126976	126976	122880
`Block_0/Conv_0`	9248	9248	9248
`Block_0/GroupNorm_0`	64	64	64
`Block_1/Conv_0`	18496	18496	18496
`Block_1/GroupNorm_0`	128	128	128
`Block_2/Conv_0`	73856	73856	73856
`Block_2/GroupNorm_0`	256	256	256
`Block_3/Conv_0`	110656	110656	110656
`Block_3/GroupNorm_0`	128	128	128
`Block_4/Conv_0`	27680	27680	27680
`Block_4/GroupNorm_0`	64	64	64
`Conv_0`	867	867	867
Total	368419	364323	364323