Lyapunov Stable Neural Network Control with Control Behaviour Optimisation

Manuel R. Wendl
Munich School of Engineering
B.Sc. Engineering Science
TUM - Technical University Munich
Munich, Germany
manuel.wendl@tum.de

***Abstract*—Recent papers show advances in stable neural network controllers for nonlinear dynamical systems. This project paper introduces a method to control a known nonlinear dynamical system with a neural network controller, that satisfies stability with respect to a simultaneously learned Lyapunov function. Additionally controller behaviour regarding convergence and actuator energy shall be incorporated in the stable learning method.**

I. Introduction

Neural networks have emerged as a promising tool for tackling control tasks involving complex nonlinear dynamical systems. These networks are trained using available data and simulations of the system dynamics. However, one of the major challenges encountered in neural network controllers and data-driven approaches is the ability to ensure stability for the controlled system. Several recent studies have proposed a novel learning technique that combines the controller and a concurrently learned Lyapunov function, which provides a guarantee of stability [1] [2] [7]. This project paper demonstrates how such a controller and Lyapunov function can be learned jointly, using the torque-controlled inverted pendulum as a case study. Additionally certain control behaviour is learned regarding the convergence and actuator energy, such that a similar behaviour as of a linear quadratic regulator (LQR) for linearised systems can be achieved. The challenge hereby is to maintain the satisfaction of the Lyapunov stability criterion.

II. Dynamics

The dynamical model of the momentum controlled inverted pendulum exhibits the following nonlinear dynamics:

$$\ddot{\phi} = \frac{mgL\sin(\phi) + u - b\dot{\phi}}{mL^2} \tag{1}$$

The second order ordinary differential equation (ODE) can be transformed into a system of first order ODEs:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \vec{f}(\vec{x}, u) = \begin{bmatrix} x_2 \\ \frac{mgL\sin(x_1)+u-bx_2}{mL^2}\end{bmatrix} \tag{2}$$

The state representation for the ODE system in equation 2 is defined as : $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} := \begin{bmatrix}\phi\\ \dot{\phi}\end{bmatrix}$. In the explicit system under investigation the listed constants and state space boundaries were chosen. The system has an unstable critical point at $\vec{x}^* = [0, 0]^T$, which is also the desired point to stabilise the system on. The uncontrolled system behaviour and the general structure of the system are shown in figure 1.

TABLE I
Dynamical system constraints and domain limits

Parameter	Value	Description
m	0.15	Mass of the pendulum arm.
g	9.81	Gravity constant.
L	0.5	Length of the pendulum arm.
b	0.1	Damping coefficient.
u	$[-5, 5]$	Control input.
ϕ	$[-\pi, \pi]$	Angle of pendulum arm.
$\dot{\phi}$	$[-7, 7]$	Angular velocity.

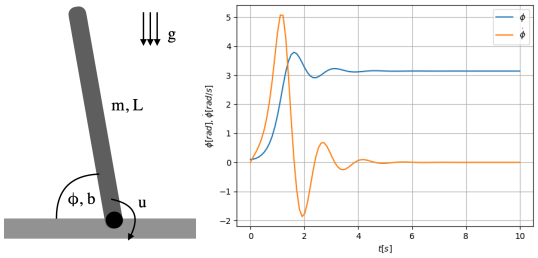

Fig. 1. (left) Structure of the inverted pendulum; (right) Uncontrolled dynamical system behaviour

III. Domain

The state space spanned by the angle and angular velocity, with the according limits from table I, is discretised onto a grid. For better convergence in the region of the critical point $\vec{x}^* = [0, 0]$ the mesh is non uniformly distributed. It has finer grid resolution approaching the critical point. The grid width decreases linearly for both state variables ϕ and $\dot{\phi}$, which can be mathematically formulated as:

$$\mathcal{D} := \left\{ (\phi_i, \dot{\phi}_j) : \phi_i = \text{sign}(i)\pi \sum_{j=1}^{|i|} \left(j \frac{2}{(n+1)n} \right) \wedge \right.$$
$$\left. \dot{\phi}_i = \text{sign}(i)7 \sum_{j=1}^{|i|} \left(j \frac{2}{(n+1)n} \right) \quad | -n \leq i \leq n \right\} \tag{3}$$

The parameter n is the refinement of the mesh. This results in the total number of grid points $N = (2n+1)^2$, including the critical point $\vec{x}^* = [0, 0]$.

IV. Mathematical Problem Description

The stability of the system shall be proven by the direct Lyapunov method, which is provided below as introduced in [3].

Theorem 1 (Lyapunov Theorem of Asymptotic Stability): Given a continuous map $V : \mathbb{R}^n \to \mathbb{R}$. It is required to be locally positive definite:

1) $V(\vec{w}) = 0$
2) $V(\vec{x} - \vec{w}) > 0, \quad \forall ||\vec{x} - \vec{w}|| > 0$

V is called Lyapunov function if it is differentiable and

$$\forall ||\vec{x}|| > 0 : \dot{V}(\vec{x} - \vec{w}) < 0; \quad \dot{V}(\vec{w}) = 0 \tag{4}$$

For global stability these properties have to hold over the entire domain $\mathcal{D}$.

For the *Control Lyapunov Function* (CLF) additionally exponential stability shall be satisfied [1].

Theorem 2 (Exponential Stability): Given $V(x)$ satisfies the requirements from theorem 1 and $\dot{\vec{x}} = f(\vec{x}, u)$ are the dynamics of the controlled system, the system is exponentially stable if:

$$c_1 ||\vec{x}||_2^2 \leq V(\vec{x}) \leq c_2 ||\vec{x}||_2^2, \quad \dot{V}(\vec{x}) \leq -\alpha V(\vec{x}). \tag{5}$$

for all $\vec{x} \in \mathcal{D} \backslash \{0\}$, with $\alpha, c_1, c_2 \in \mathbb{R}^+$.

The derivative $\dot{V}(\vec{x})$ is the Lie derivative of the continuously differential scalar function $V(\vec{x})$ as introduced in [7].

$$\dot{V}(\vec{x}) = \frac{\partial V}{\partial \vec{x}}(\vec{x}) f(\vec{x}, u) = \nabla V(\vec{x}) f(\vec{x}, u) \tag{6}$$

It is the derivative with respect to the controlled system dynamics $f(\vec{x}, u)$.

V. Neural Network Control Architecture

The basic control architecture of the system is given by a neural network based controller and Lyapunov function. The controller receives the current state $\vec{x}$ as an input and the reference value $\vec{w}$, which is the desired state and in this case the critical point $\vec{x}^* = [0, 0]$. The output of the neural network controller is the control input u for the controlled dynamical system $f(\vec{x}, u)$ of the pendulum. From the current state $\vec{x}$ the Lyapunov function value $V(\vec{x})$ is determined by the Lyapunov neural network and the gradient with respect to the state $\nabla V(\vec{x})$ can be derived. The computed Lyapunov gradient and the controlled system dynamics $f(\vec{x}, u)$ have to satisfy exponential stability as defined above. The control structure is visualised in figure 2.

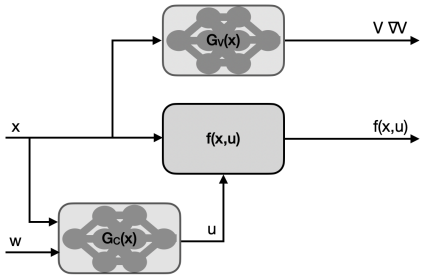

Fig. 2. Neural Network Control Architecture

A. Controller Neural Network

The controller consists of a neural network that receives the current state $\vec{x}$ and the desired state $\vec{w}$ as input values. The inputs are normalised according to zero mean and variance $(\vec{x} - \text{mean}(\vec{x}))/(\sqrt{\text{var}(\vec{x})})$ and subtracted in the first step. The controller is an unconstrained neural network with Rectified Linear Unit (ReLU) activation functions in all hidden layers. The output of the network is then passed through a hyperbolic tangent (Tanh) activation function with value set $[-1, 1]$ and multiplied by the control input bounds, as specified in table I.

$$u = 5 \cdot \tanh(G_C(\vec{x}, \vec{w})) \tag{7}$$

The architecture of the control neural network is depicted in figure 3.

Fig. 3. Neural network architecture for controller neural network $G_C(\vec{x}, \vec{w})$

B. Lyapunov Function

The Lyapunov function is derived similar as in [1] from the output of the Lyapunov Neural Network $G_V(\vec{x})$. The required properties of the Lyapunov function are satisfied by the network architecture and formula:

$$V(\vec{x}) = \sigma_{\text{sReLU}}(G_V(\vec{x}) - G_V(\vec{w})) + \frac{1}{2} ||\vec{x} - \vec{w}||_2^2 \tag{8}$$

By choosing the activation function of the neural network and σ_{sReLU} in equation 8 to be smooth ReLU (sReLU) functions, differentiability is guaranteed. The sReLu activation shall be defined as:

$$\sigma_{\text{sReLU}}(x) = \begin{cases} x & \text{if } x > 1 \\ \frac{x^2}{2} & \text{if } 0 < x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

By definition of the Lyapunov function stated in equation 8 $V(\vec{w}) = 0$ is always satisfied, independent from the neural network output $G_V(\vec{x})$. Additionally the Lyapunov function is not allowed to have any non zero extrema. This is guaranteed by using strictly increasing activation functions, as in this case the sReLU activation and non negative weights and biases in

the neural network. This approach is described in [4] as input-convex neural networks.
Since of the requirement of positive weights and biases only, the first input activation function is a parabolic activation function in order to consider the positive and negative domain regions equally.

$$\sigma_{\text{parabolic}}(x) = \frac{x^2}{2} \tag{10}$$

The output of the Lyapunov network is limited by the last activation layer, which is chosen to be a Tanh activation, scaled by 10, such that we obtain the overall output of the network $G_V(\vec{x})$.

$$G_V(\vec{x}) = 10 \cdot \tanh(\tilde{G}_V(\vec{x})) \tag{11}$$

The architecture of the Lyapunov Neural network is depicted in figure 4.

Fig. 4. Neural network architecture for Lyapunov neural network $G_V(\vec{x})$

VI. Joint Learning of the Neural Network Controller and Lyapunov Function

The goal is to learn the Neural Network controller and the Lyapunov function jointly, such that global, exponential stability is satisfied as described in section IV. For the joint learning an approach with one single loss function is chosen.

A. Loss Function ensuring Stability

The used loss function ensures that the Lyapunov function is monotonically decreasing along the path with the minimal required gradient, defined by the exponential stability theorem.

$$\text{loss} = \sigma_{ReLU}(\nabla V(\vec{x})^T f(\vec{x}, u) + \alpha V(\vec{x})) \tag{12}$$

The custom training loop implements the loss function by computing:

- the Lyapunov function $V(\vec{x})$ of the initial state.
- the gradient of the Lyapunov function $\nabla V(\vec{x})$ with respect to the input state $\vec{x}$.
- the control parameter $u(\vec{x}) = G_C(\vec{x})$ and the resulting controlled system dynamics $f(\vec{x}, u)$.
- the loss function according to equation 12 by plugging in the individual terms.

This joint learning method guarantees global, exponential stability $\iff \forall \vec{x}_i \in \mathcal{D} : \text{loss}(\vec{x}_i) = 0$. Which is therefore the necessary termination criterion of the learning process.

1) Computation of Lyapunov Function: The Lyapunov function is computed by evaluating the results of $G_V(\vec{x})$ of the input state $\vec{x}$ and $G_V(\vec{w})$ the target $\vec{w}$. The Lyapunov function is then computed as stated in equation 8.

2) Computation of Lyapunov Gradient: From the result of the Lyapunov function $V(\vec{x})$ the gradient is calculated with respect to the input state $\vec{x}$.

$$\nabla V(\vec{x}) = 2\gamma(\vec{x} - \vec{w}) + \sigma'_{sReLU}(G_V(\vec{x}) - G_V(\vec{w}))\nabla G_V(\vec{x})$$
$$\sigma'_{sReLU}(z) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x < 1 \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

3) Computation of System Dynamics: The control input u is taken from $G_C(\vec{x})$ as stated in eq. 7. The system dynamics are given according to the definition of the system dynamics 2. Optionally for systems with unknown dynamics also learned dynamics can be used in this step.
Using the parallels to numerics, one can optimise the convergence of the controller by using implicit Euler dynamics instead of the given system dynamics. The controller tends no longer to instantly minimise $\dot{\phi}$ and learns a smoother converging trajectory to $\vec{w}$. The implicit Euler step is approximated in form of the Heun method:

$$f_h(\vec{x}_i, u) = \frac{1}{2}(f(\vec{x}_i, u(\vec{x}_i)) + f(\vec{x}_{i+1}, u(\vec{x}_{i+1})))$$
$$\vec{x}_{i+1} = \vec{x}_i + \tilde{\delta t} f(\vec{x}_i, u(\vec{x}_i)) \tag{14}$$

For time discrete controllers, only updating the control input $u(\vec{x})$ after each control time increment δt, the choice of the Heun time increment $\tilde{\delta t}$ is essential.
The dynamics in the Heun step propose a convergence of order $\mathcal{O}(\tilde{\delta t}^2)$, whilst the original dynamics only have convergence $\mathcal{O}(\delta t)$. This requires the Heun time increment $\tilde{\delta t}$ in the system dynamics to have the following relation to the control time increment δt:

$$\delta t = (\tilde{\delta t})^2 \tag{15}$$

to ensure stable convergence to the desired position.

B. Discussion of Joint Learning Results

The controller and Lyapunov function were jointly learned according to the method proposed so far in section VI. The system constants and domain limits were chosen as stated in table I. The domain was discretised with refinement $n = 50$ according to the definition in equation 3. The networks were learned with an exponentially decreasing learning rate, with an initial value of $lr = 1 \cdot 10^{-3}$ decreasing by the factor of 0.75 every second epoch. The exponential stability was enforced with $\alpha = 1$.
The resulting learning history of the loss defined by equation 12 is given in figure 5. The learning process was terminated when the termination criterion previously described had been reached. The trajectories can be seen in figure 6 for a single initial state and multiple initial states. Additionally the Lyapunov function for the respective trajectories are shown. All trajectories reach the desired target $\vec{w} = [0, 0]^T$. These were simulated using a time discrete controller with $\delta t = 0.01$, such that a Heun time increment of $\tilde{\delta t} = 0.1$ has been sufficient for training, according to the derived policy. The

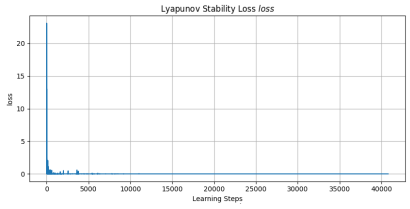

Fig. 5. Learning History of joint learning loss

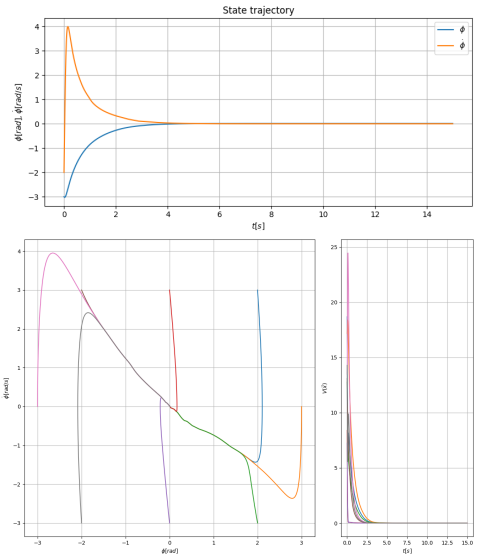

Fig. 6. (top) Single trajectory with initial value $\vec{x}_0 = [-3, -2]$. (bottom) Trajectories with multiple initial values and according Lyapunov function values.

learned controller and Lyapunov function are visualised in surface plots 7. Further the scalar product $\nabla V(\vec{x}) f(\vec{x}, u)$ is depicted and the controlled system dynamics are visualised as a vector field plot. In case the required relation between

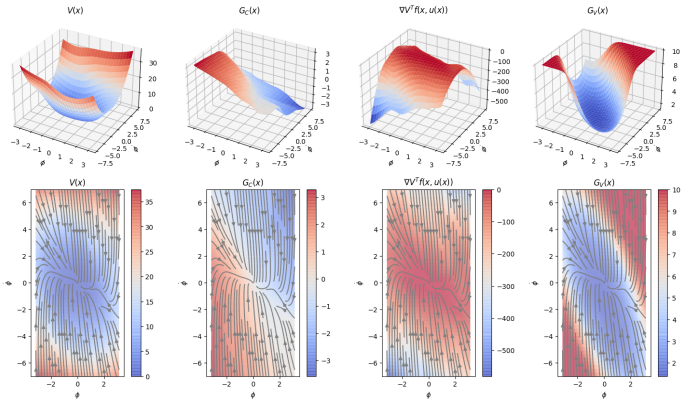

Fig. 7. (top) Surface plots. (bottom) Heat maps with including controlled system dynamics as vector field

control time increment δt and Heun time increment $\tilde{\delta t}$ is not satisfied, the trajectory is showing minor instabilities in figure 8 in form of a slight oscillations.

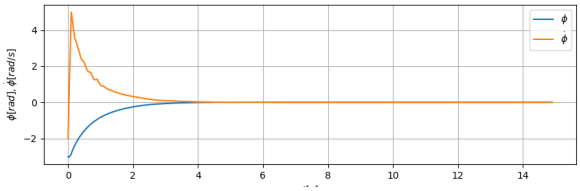

Fig. 8. Unstable trajectory for violated relation between δt and $\tilde{\delta t}$

C. Comparison Utilising Heun-Dynamics vs. Pure System Dynamics

In order to demonstrate the advantage of the proposed usage of the Heun dynamics, identical learning parameters were used learning the system with the pure dynamics. The trajectory converges faster to $\dot{\phi} = 0$ as postulated before, which results in a rather slow convergence to $\vec{x}^*$. The Lyapunov function also decreases slower, especially for operation points with small $\dot{\phi}$, such that the exponential stability criterion is much more difficult to learn to the control architecture using the pure system dynamics. The trajectories resulting from such a controller are shown in figure 9. The trained controller is

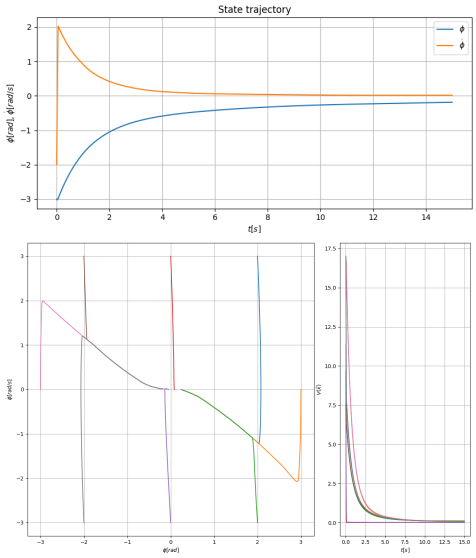

Fig. 9. (top) Single trajectory with initial value $\vec{x}_0 = [-3, -2]$ trained with pure system dynamics (bottom) Trajectories with multiple initial values and according Lyapunov function values converging comparably slower.

visualised as a surface plot in figure 10.

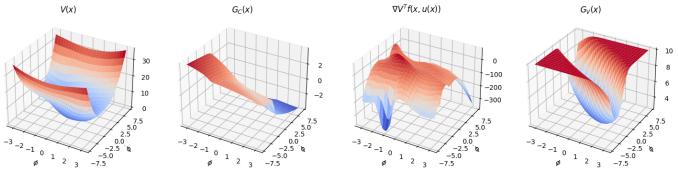

Fig. 10. Surface plot of the Controller and Lyapunov function trained with pure system dynamics.

VII. Control Behaviour Optimisation

With the method described in VI it is now possible to train an arbitrary controller, that satisfies the exponential stability criterion. Further it is desired to be able to learn the controller a certain control behaviour, regarding convergence speed and actuator energy. It shall produce comparable results as a linear quadratic regulator (LQR) for linearised systems. The LQR loss function definition according to [5] is given by:

$$J(\vec{x}_0) = \int_0^\infty [\vec{x}(t)^T Q \vec{x}(t) + \vec{u}(t)^T R \vec{u}(t)] dt, \\ x_0 = x(0), Q = Q^T > 0, R = R^T > 0 \tag{16}$$

The matrix Q penalises the state loss and optimises faster convergence, whilst R penalises the actuator energy used. A similar approach shall be developed for the already existing joint learning method. The challenge is to still ensure global, exponential stability.

A. Convergence Behaviour

In the so far introduced method exponential stability was defined utilising a parameter α, which was arbitrary chosen to be 1 for the exemplarily learned controllers presented in VI-B. Due to the fact that the learned Lyapunov function has a limited value range $[0, \lambda||\vec{x}||_2^2 + 10]$, a lower bound for the convergence speed can be defined by choosing α. For larger α the system will be forced to converge faster than for smaller α. This means that we can easily introduce the lower bound of the convergence in the presented method.
Different controllers were learned for $\alpha = \{0.5, 1, 3\}$, the results are presented in figure 12. These controllers satisfy a lower bound of convergence, but within this bound they are still arbitrarily learned, since the loss is zero, based on the joint learning loss function from equation 12, as soon as the exponential stability is satisfied.

B. Actuator Energy Behaviour

The aim is now to optimise the controllers for actuator energy. The exponential stability lower bound shall not be violated by the additional learning goal.

1) Actuator Energy Loss: The actuator energy loss function is defined as:

$$\text{loss}_u = \lambda_1 \cdot \frac{1}{2} \left(\frac{u}{5}\right)^2 w \tag{17}$$

λ_1 is a weighting factor, u the control output $G_C(\vec{x})$ and w an additional weighting factor ensuring stable learning.

2) Forcing Exponential Stability: Stability of the controller is ensured by only applying the loss function to the training-batch if all samples satisfy the exponential stability lower bound. This is checked for each individual sample with the inequality $\nabla V(\vec{x}) f_h(\vec{x}, u) + \alpha V(\vec{x}) <= 0$.
Additionally a weighting factor w has been introduced in the actuator energy loss equation. The idea is to penalise the actuator energy more for operating points, which are converging even faster than the required lower bound and less for operating points being near to the bound.
The weighting factor is multiplied to each individual sample of the batch and is defined by:

$$w = \sigma_{ReLU}(\tanh(-\lambda_2 \cdot (\nabla V(\vec{x}) f_h(\vec{x}, u) + \alpha V(\vec{x})))) \tag{18}$$

The term $\nabla V(\vec{x}) f_h(\vec{x}, u) + \alpha V(\vec{x})$ is negative when computing loss_u, since exponential stability is already ensured. By multiplying with a weighting factor $-\lambda_2$ with $\lambda_2 \in \mathbb{R}^+$ the result is scaled to a positive number. For fast convergence (large negative term) the outcome will be a large positive number. The outcome is then passed to a $\tanh$ function, which will give a weighting value between $[0, 1]$ (because of satisfied stability the lower bound is zero rather than negative one). Ensuring this property a ReLU function may be added, which is not explicitly necessary to add.

3) Learning Results Discussion: The previously learned controllers for $\alpha = \{0.5, 1, 3\}$ were relearned with the additional loss function for the actuator energy with $\lambda_1 = 1$ and $\lambda_2 = 0.1$. In the resulting plots in figure 13 one can observe in the controller heat map plot, that the control output is drastically reduced at certain operation points, which already satisfy the lower bound for small control outputs.

C. Comparison to LQR

Finally the derived learning method is compared to a full state feedback LQR controller tuned for the matrices $Q = I$ and $R = 1$ on the linearised system. The trajectory and the controller are visualised in figure 11. Compared to the LQR the derived method converges drastically faster. Analysing the heat maps of the controllers, one can observe that the general pattern of the control output with respect to the state space is similar. For future work it might also be interesting to compare with nonlinear state-dependent LQR as mentioned in [6].

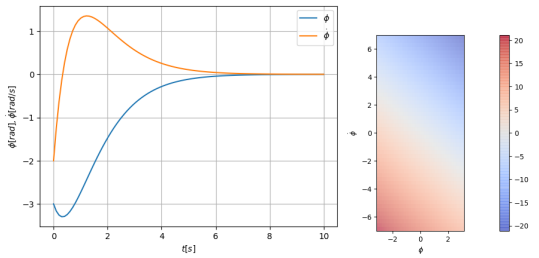

Fig. 11. (left) Trajectory of initial state $\vec{x}_0 = [-3, -2]$ and (right) controller heat map of state feedback LQR controller.

References

[1] Y. Min, S. M. Richards, N. Azizan , "Data-Driven Control with Inherent Lyapunov Stability", 2023.
[2] H. Dai, B. Landry, L. Yang, M. Pavone, R. Tedrake, "Lyapunov-stable neural-network control", 2022
[3] H.K. Khalil, "Nonlinear Systems," Prentice Hall, 1996.
[4] G. Manek, J.Z. Kolter, "Learning stable deep dynamics models," in 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019.
[5] R. Tedrake, "Underactuated Robotics, Algorithms for Walking, Running, Swimming, Flying and Manipulation", Course Notes for MIT 6.832, 2023
[6] S. Richards, J.-J. Slotine, N. Azizan, M. Pavone, "Learning Control-Oriented Dynamical Structure from Data", 2023
[7] R. Zhou, T. Quartz, H. de Sterck, J. Liu, "Neural Lyapunov Control of Unknown Nonlinear Systems with Stability Guarantees", 2022

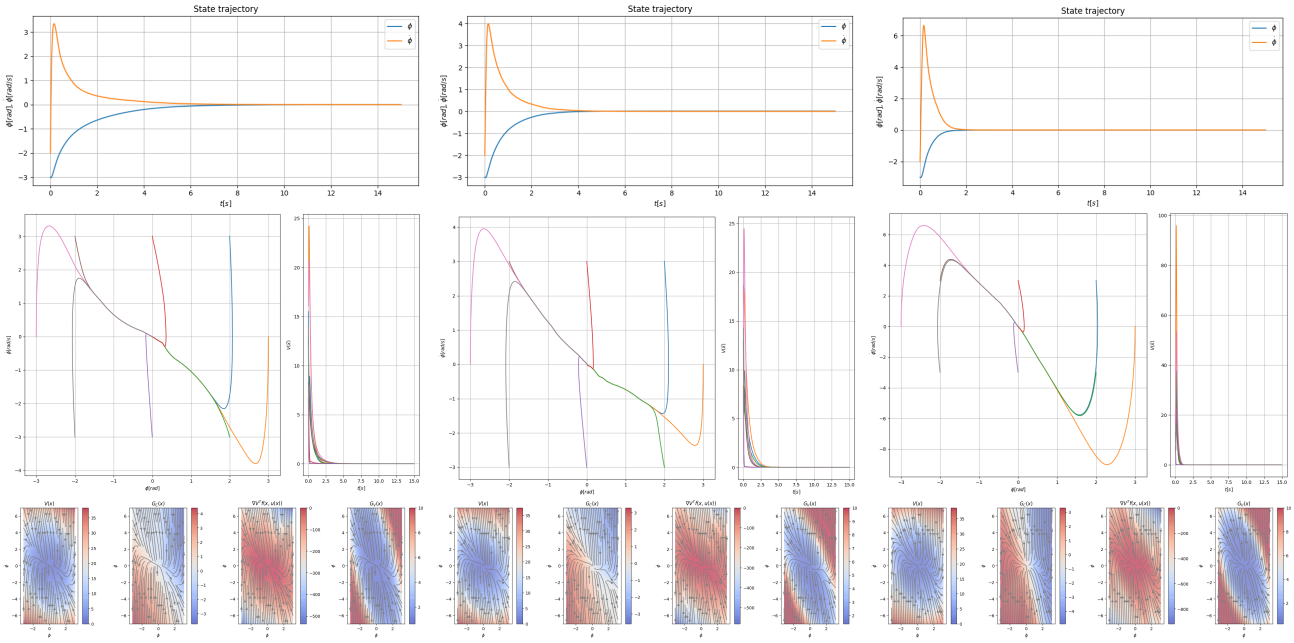

Fig. 12. (top) single trajectory with initial value $\vec{x}_0 = [-3, -2]$. (middle) multiple trajectory with different initial values. (bottom) controller visualisation with Lyapunov function and controlled dynamics. Different α values for (left) $\alpha = 0.5$, (middle) $\alpha = 1$ and (right) $\alpha = 3$.

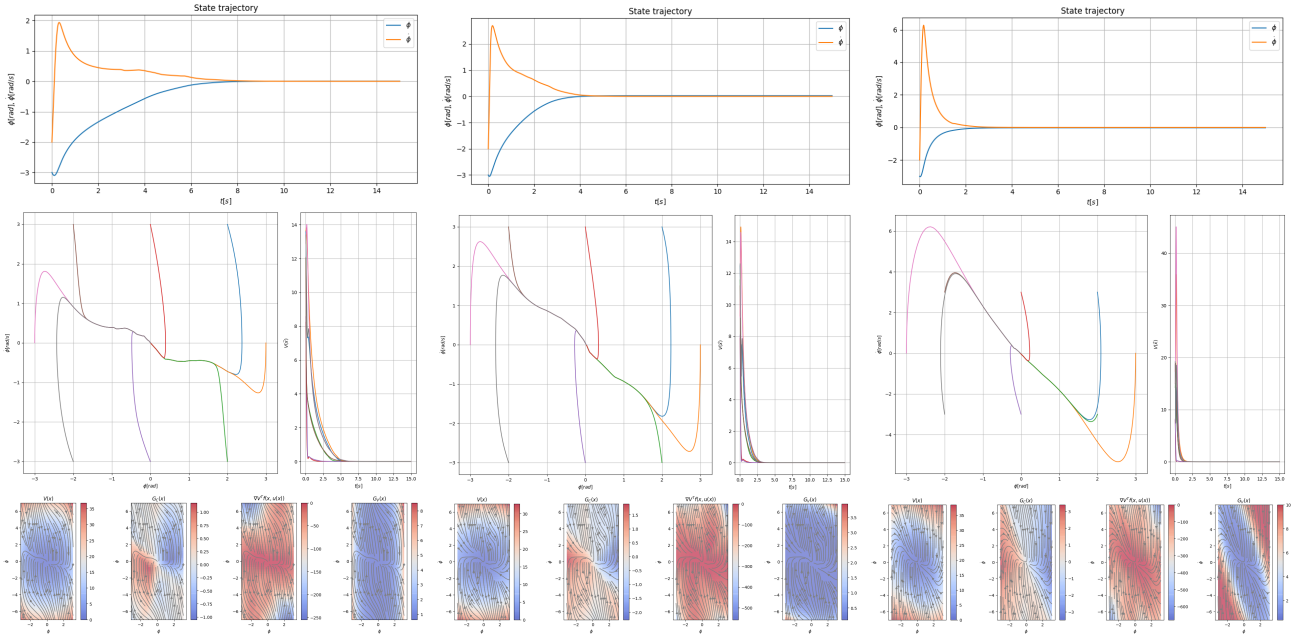

Fig. 13. (top) single trajectory with initial value $\vec{x}_0 = [-3, -2]$. (middle) multiple trajectory with different initial values. (bottom) controller visualisation with Lyapunov function and controlled dynamics. Different α values for (left) $\alpha = 0.5$, (middle) $\alpha = 1$ and (right) $\alpha = 3$ and additionally applied actuator energy loss.