ETH zürich

Soft Robotics Laboratory

Bachelor Thesis

Model Predictive Contouring Control with Uncertainty Estimation using Deep Evidential Koopman Operators

Luca Lietha

April 2024

Supervision

Prof. Dr. Robert Katzschmann
Minghao Han

Abstract

Effective modeling of complex, nonlinear systems is critical for advancing intelligent machines and robotics. The Deep Evidential Koopman Operator framework presents a novel integration of Koopman Operator Theory with Deep Evidential Regression, facilitating the application of linear control methods to nonlinear systems while quantifying model uncertainty. This thesis expands on this framework to include path tracking capabilities, addressing a vital need for safety in dynamic environments. We incorporate contouring control formulations into the existing controller and expand the simulation framework to test the expanded control framework.

Acknowledgement

Throughout the duration of this thesis, I have received support from a few key people. First, I would like to thank my supervisors Minghao Han and Prof. Dr. Robert Katzschmann for assisting and guiding me along the way of this project. It was an honor to work with you. I would further like to thank the Soft Robotics Laboratory for granting me the opportunity to do this interesting and challenging thesis. Finally, special thanks go to Felix Baumann, Fabrice Bourquin, Michael Lietha and Zhuofan Shi.

Contents

1 Introduction

Modeling complex, nonlinear systems is essential in intelligent machine and robotics development for predicting dynamic system behavior and enabling effective planning and control. The Koopman operator addresses this by enabling the linear propagation of observables in a potentially infinite-dimensional space, allowing the application of linear control and analysis methods to complex nonlinear systems (Han et al., 2022).
Neural networks, renowned for their versatility and effectiveness in learning and adaptation, often surpass handcrafted solutions in performance. This makes them highly promising for tackling complex, high-dimensional robotic challenges, ranging from understanding dynamics to interpreting environmental semantics. Nonetheless, their widespread application in safety-critical or high-risk domains is currently limited (Fan, 2021).
This reluctance can be attributed to several challenges associated with the deployment of Deep Neural Networks (DNNs) in safety-critical applications. DNNs often display overconfidence in their predictions, with a lack of recognition of possible inaccuracies. Additionally, a significant challenge for DNNs, including in Deep Reinforcement Learning (DRL), is their ability to generalize to new instances. Despite impressive performances in standard benchmarks, DNNs are prone to issues with distributional shifts or out-of-distribution (OOD) instances. This means DNNs might produce unreliable outputs when data differs from their training set, potentially leading to risks in OOD situations (Schmoeller da Roza et al., 2023).
Consequently, the capability to estimate model uncertainty emerges as a critical factor for its broader adoption in practical applications. Moreover, accurate uncertainty quantification is instrumental in interpreting model confidence, identifying OOD test samples and conditions under which the model is prone to failure (Amini et al., 2020).
Autonomous robots are engineered to perform an array of tasks, navigating safely through dynamic environments. This often requires the ability to follow a reference path accurately. Several methodologies exist for this purpose, with many approaches leveraging Model Predictive Control (MPC) and DNN for modeling, for example Rokonuzzaman et al. (2021). Therefore, they face the same challenges regarding DNNs and their unawareness of uncertainty as mentioned above. Consider a situation where a robot, utilizing a DNN model for state prediction, follows a predetermined global path. It may encounter OOD inputs, for example atypical lighting conditions, leading to predictions characterized by heightened epistemic uncertainty. In such instances, it is desirable for the robot to have the capability to momentarily alter its course, thereby navigating away from these areas and avoiding any potential risks.

1.1 Goals

The Deep Evidential Koopman Operator framework addresses the problems with DNN's, as it combines the Koopman Operator Theory with Deep Evidential Regression which allows for estimating the epistemic uncertainty of the prediction. The aim of this bachelor thesis is to expand the capabilities of the existing Deep Evidential Koopman framework with a Model Predictive Contouring Control (MPCC) formulation to allow it to track a static, global reference trajectory while locally avoiding regions with high epistemic uncertainty. The following sub-objectives should be realised:

- Implementing a static path generation into the simulation framework that is compatible with the contouring formulation.
- Expanding the evidential Koopman MPC formulation with the contouring formulation.
- Testing and validating the expanded framework.

2 Related Work

The groundwork for this thesis has been established by Minghao Han and the Soft Robotics Lab at ETH Zürich. The Deep Evidential Koopman Operator Minghao Han is currently developing is a combination of a Deep Stochastic Koopman Operator(DeSKO) Han et al. (2022) with an Evidential Deep Neural Networks based Amini et al. (2020) and Meinert and Lavin (2022). We base our approach for path tracking on the contouring control formulation used in Liniger et al. (2015) and Lam et al. (2010).

2.1 Deep Evidential Learning

Neural networks suffer from two distinct types of uncertainty: (1) aleatoric uncertainty, which arises from the stochastic nature of the environment and the data that comes from it and (2) epistemic uncertainty, which stems from the model's limitations in making accurate predictions. While aleatoric uncertainty can be directly deduced from the data and cannot be reduced, estimating epistemic uncertainty requires more sophisticated approaches, such as Bayesian neural networks (BNNs) or ensemble-based methods. However, both approaches face several limitations and Amini et al. (2020) introduced an approach called Deep Evidential Learning which enables non-Bayesian NNs to approximate both aleatoric and epistemic uncertainties. They achieve this by placing evidential priors over the original Gaussian likelihood function and training the NN to infer the hyperparameters of a higher order evidential distribution. The great advantage of this method is, that it does not rely on sampling during inference or OOD examples for training and does not require complex probabilistic models or extensive sampling. Meinert and Lavin (2022) proposed some improvements and expanded the approach to the multivariante case.

2.2 Evidential Koopman Control

Koopman Operator Theory has gained much attention in recent years, because it allows the prediction, estimation, and control of nonlinear systems with standard textbook methods developed for linear systems. It does that by lifting the state to a potentially infinite dimensional observable space where the state can be linearly propagated. (Brunton et al., 2021).
The Deep Stochastic Koopman Operator (DeSKO) proposed by Han et al. (2022), does not require hand-design of Koopman observables or explicit learning of a set of observable functions. Instead, the system state x_t is encoded via a neural network into the parameters $(\mu_\theta(x_t), \sigma_\theta(x_t))$ of a Gaussian distribution of Koopman observables. It is then possible to recursively and linearly propagate the encoded system states to represent the system process, which enables linear control techniques with guarantees for robust stability. It is the first Koopman-based learning control framework with a closed-loop robust stability guarantee for uncertain nonlinear systems.
The Deep Evidential Koopman framework is an extension of the DeSKO framework with an evidential network. Instead of predicting the parameters $(\mu_\theta(x_t), \sigma_\theta(x_t))$ of a Gaussian distribution, the network predicts the parameters $(\bar{\mu}, v, \Psi, \alpha)$ of the evidential distribution as shown in Fig. 1.

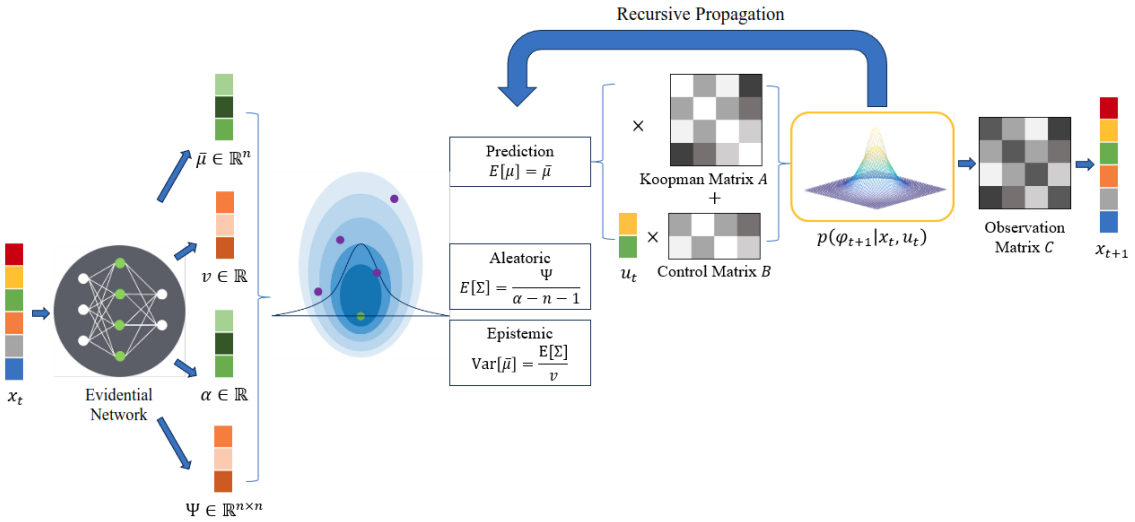


Figure 1: Architecture of Evidential-Koopman framework: With the help of four NNs, we encode the system state x_t into the parameters of an evidential distribution of observables, which are $(\bar{\mu}, v, \Psi, \alpha)$. Then, the learned Koopman matrix A and the control matrix B propagate the distribution of observables to the next time step. This architecture recursively produces a series of distribution predictions over a given time horizon. Finally, the learned observation matrix C maps the observables back into the state space.

2.3 Model Predictive Contouring Control

Traditionally, path following frameworks treated trajectory planning and control as distinct tasks. The motion planner generates (collision-free) paths, and the motion controller tracked these paths via direct actuator commands. MPCC represents an advanced control framework that integrates local trajectory generation and tracking into a unified optimization problem. This approach facilitates a more cohesive and efficient handling of path following tasks. (Brito et al., 2020).
The MPCC approach requires merely a global, continuously differentiable path (in 2D or 3D) as its reference, thus eliminating the feasibility constraints typical for standard MPC methods. Additionally, it facilitates the incorporation of extra constraints and cost functions for the purpose of obstacle avoidance. A critical feature of the MPCC framework is the decomposition of path-following error into contouring and lag elements. This decomposition enables the system to differentially prioritize between optimizing tracking accuracy and achieving rapid path traversal. (Romero et al., 2022).
Kabzan, Valls, et al. (2019) used MPCC for Optimization-based autonomous racing of 1:43 scale RC cars. Kabzan, Hewing, et al. (2019) used MPCC together with Gaussian processes regression to take residual model uncertainty into account and achieve safe driving behavior for the AMZ Driverless race car "gotthard". Brito et al. (2020) used MPCC for collision avoidance in unstructured dynamic environments and Narkhede et al. (2023) used it for overtaking maneuvers with a bipedal robot. Romero et al. (2022) used MPCC for time-optimal quadrotor flight beating current state-of-the-art methods and a world-class human pilot in terms of lap time achieving speeds of up to 60 km/h.

3 Preliminaries

3.1 Koopman Operator

In recent years, Koopman Operator Theory has become increasingly popular for its ability to predict, estimate, and control nonlinear systems using conventional methods traditionally applied to linear systems. Consider the following nonlinear control system:

$$\mathbf{x_{t+1}} = f(\mathbf{x_t}, u_t) \tag{1}$$

where $x_t \in \mathbb{R}^N, u_t \in \mathbb{R}^M$ are the state and control inputs. The idea of the Koopman Operator Theory is that the system can be lifted to an *observables* space F, where the system dynamics become linear. In this function space, the system's evolution is characterized by the Koopman Operator (KO) $\mathcal{K}$. It holds that:

$$\mathcal{K}\psi(\mathbf{x_t}, u_t) = \psi \circ f(\mathbf{x_t}, u_t) \tag{2}$$

where ψ denotes the observable function. Note that because $\mathcal{K}$ is linear, the system dynamics in the observable become linear. There a two problems with this approach 1) the KO is infinite-dimensional, therefore a finite-dimensional approximation is needed and 2) finding these observable function ψ is non-trivial. For the case of control we can consider the following Koopman equation:

$$\psi(\mathbf{x_{t+1}}) = A\psi(\mathbf{x_t}) + Bu_t \tag{3}$$

where $\Psi(x) = [\psi_1(x), \dots, \psi_n(x)]^T \in \mathbb{R}^{n>>N}$ is the state in the lifted space. The Koopman Matrix $A \in \mathbb{R}^{n\times n}$ and the Control matrix $B \in \mathbb{R}^{n\times M}$ represent the finite dimensional approximations of the KO (Brunton et al., 2021)

3.2 Model Predictive Control

Model Predictive Control solves a finite-time horizon optimal control problem. The optimal control problem is solved at an initial state, and an optimal input sequence is obtained. The first element of the input sequence is applied to the system, and at the next time step, the optimal control problem is solved again for the new state.
The following is a typical MPC formulation for path tracking:

$$\begin{aligned} J(x) &= \min \sum_{k=0}^{N-1} l(\mathbf{x}_k, u_k) \\ \text{s.t.} \quad & x_0 = x \quad \text{(initial condition)} \\ & \mathbf{x}_{k+1} = f(\mathbf{x}_k, u_k) \quad \text{for } k \in [0, N-1] \quad \text{(system's dynamics)} \\ & \mathbf{x}_k \in X \quad \text{for } k \in [0, N-1] \quad \text{(state constraints)} \\ & u_k \in U \quad \text{for } k \in [0, N-1] \quad \text{(input constraints)} \end{aligned} \tag{4}$$

with

$$l(\mathbf{x}_k, u_k) = \|\Delta\mathbf{x}_k\|_Q^2 + \|\Delta u_k\|_R^2 + \|\Delta x_N\|_P^2 \tag{5}$$

where $f(\mathbf{x}, u)$ is the system's dynamics, $\mathbf{x}_k$ is the state of the system, $\Delta\mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{k,ref}$, $\Delta u_k = u_k - u_{k,ref}$, and where $Q \succeq 0$, $R > 0$,$P \succeq 0$ are the state, input and final state weighting matrices. N is the horizon length and X, U are the state and input constraints. The objective is to minimize a quadratic penalty on the error between the predicted states and inputs, and a given dynamically feasible reference $\mathbf{x}_{k,ref}$ and $u_{k,ref}$. This formulation relies on the fact that a feasible, time parameterized reference $\mathbf{x}_{k,ref}(t)$, $u_{k,ref}(t)$ is accessible for the future time horizon N. (Romero et al., 2022)

3.3 Model Predictive Contouring Control

In the case of MPCC, we solve the higher level task of minimizing the Euclidean distance to a reference path while maximizing the speed at which the path is traversed. Therefore combine path generation and path tracking into one problem. We achieve that by decomposing the path following error into a contouring e^c and a lag e^l component, allowing the system to assign different weights to maximizing tracking performance versus fast path traversal. As common in the literature we don't use e^c and e^l and instead use the following approximation:

$$\begin{aligned} e_k^c &\approx \hat{e}_k^c(x_k, y_k, \theta_k) = \sin(\Phi(\theta_k))(x_k - x_{k,ref}(\theta_k)) - \cos(\Phi(\theta_k))(y_k - y_{k,ref}(\theta_k)) \\ e_k^l &\approx \hat{e}_k^l(x_k, y_k, \theta_k) = -\cos(\Phi(\theta_k))(x_k - x_{k,ref}(\theta_k)) - \sin(\Phi(\theta_k))(y_k - y_{k,ref}(\theta_k)) \end{aligned} \tag{6}$$

where θ_k is the arc length, (x_k, y_k) is the position of the robot and $(x_{k,ref}, y_{k,ref})$ is the reference position on the path where the robot should be at time step k and $\Phi(\theta_k)$ is the orientation of the path. The approximate contouring error $\hat{e}^c$ and the approximate lag error $\hat{e}^l$ are defined as the orthogonal and tangential component of the error between the reference position and the position of the robot as can be seen in Figure 2. Let us introduce the following dynamics:

$$\theta_{\mathrm{k+1}} = \theta_k + v_k \tag{7}$$

If we choose $v_k = const$, then we have a fixed relationship between the step number k and the value of θ and the problem is equal to trajectory tracking: when and where the robot should be is fixed. However, we want to enable the robot to decide online how much progress it makes at each step. Most literature define v_k as a virtual input without any physical meaning, which is to be determined by the controller while a minority defines it as the velocity of the robot for example Brito et al. (2020). This leads to the following optimization problem for MPCC:

$$\begin{aligned} J(x) = \min \sum_{k=0}^{N} & \|\hat{e}_k^c(x_k, y_k, \theta_k)\|_{q_c}^2 + \left\|\hat{e}_k^l(x_k, y_k, \theta_k)\right\|_{q_l}^2 + \|\Delta \mathbf{u}_k\|_R^2 - \gamma v_k \\ \text{s.t.} \quad & \theta_0 = \theta_{\text{init}} \\ & \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \\ & \theta_{k+1} = \theta_k + v_k, \quad k = 0, \ldots, N-1 \\ & 0 \leq \theta_k \leq L, \quad k = 1, \ldots, N \\ & 0 \leq v_k, \quad k = 0, \ldots, N \end{aligned} \tag{8}$$

where $\mathbf{x_k} = (x_k, y_k)$ is the state, consisting of the position of the robot at time step k. θ_k is the state of progress at time step k and q_c and q_l are the weights of the contouring and lag error respectively. The term $-\gamma v_k$ incentives the progress along the path where γ is the weight for the progress.

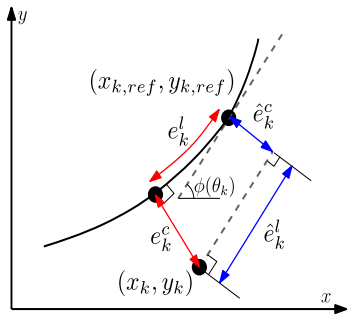


Figure 2: Illustration of the contouring error e_k^c and lag error e_k^l and its approximation $\hat{e}_k^c$ and $\hat{e}_k^l$, where (x_k, y_k) is the position of the robot and $(x_{k,ref}, y_{k,ref})$ is the reference position on the path.

4 Methodology

4.1 Reference path

The global reference path $p = (x_{\text{ref}}(\theta), y_{\text{ref}}(\theta))$, is parameterized by its arc length $\theta \in [0, L]$, where L is the total length of the path. We use splines which are fitted between each two points of a series of waypoints. We get the orientation of the path for a given θ as:

$$\Phi(\theta) = \arctan2\left(\partial y_{\text{ref}}(\theta), \partial x_{\text{ref}}(\theta)\right) \tag{9}$$

Arc length parameterization of general curves is a nontrivial problem, and finding a closed-form solution for an arc length parameterized curve given another arbitrary parameterization is, in general, not possible. Romero et al. (2022) We therefore use numerical methods to create a lookup table, which relates the parameterization parameter t (do not confuse with time) of the spline to the arc length θ. For more detail see Figure 3 and section 4.4.1
Note: The MPCC formulation only needs an arc length parametrized, continuously differentiable 2D path to track. The path does not need to be parametrized by time nor does it need to be feasible.

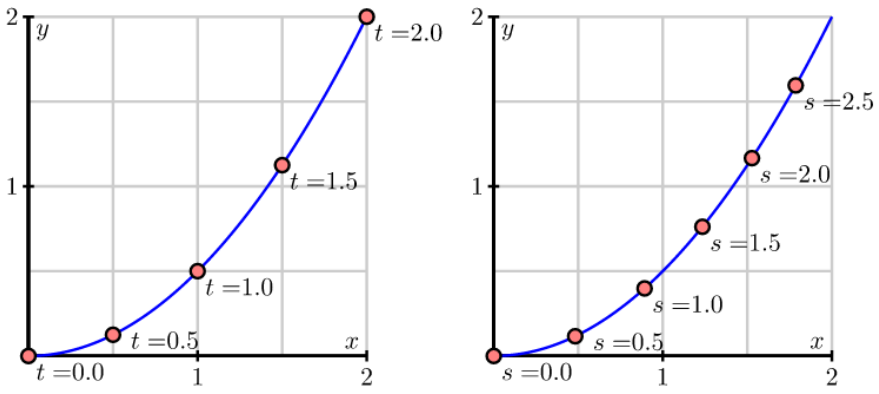


Figure 3: Normal parameterization vs arc length parameterization: As one can see a value of $t = 1$ (left side) does not correspond to a arc length of $s = 1$ (right side). There does not exist a general solution for the mapping from t to s or vice versa, because it would require to solve the Integral $s(t) = \int_0^t \|(p)'(t)\| dt$ for which in general no analytical solution exists.

4.2 Evidential Koopman Contouring Control

Consider the Evidential Net $f_\theta(\mathbf{x_t})$. We use it to encode the system state $\mathbf{x_t}$ into the parameters of a Evidential distribution of observable:

$$(\bar{\mu}_t, v_t, \Psi_t, \alpha_t) = f_\theta(\mathbf{x_t}) \tag{10}$$

We can then use them to compute the prediction of the Evidential Net and the corresponding epistemic uncertainty in the observable space with:

$$\mathbb{E}[\mu] = \bar{\mu} \quad \text{(prediction)} \tag{11}$$

$$\text{Var}[\mu] = \frac{1}{v(\alpha - n - 1)}\Psi \quad \text{(epistemic uncertainty)} \tag{12}$$

where n is the dimension of the observable space. Based on equation(3) we can use the Koopman Matrix A and the Control Matrix B to propagate the prediction to the next time step:

$$\bar{\mu}_{t+k+1} = A\bar{\mu}_{t+k} + Bu_{t+k} \tag{13}$$

We can used the learned Observation Matrix C to retrieve the state of the system:

$$\mathbf{x_t} = C\bar{\mu}_t \tag{14}$$

We can combine equation (8) with equations (10) - (14) to receive the final formulation for the Evidential Koopman Contouring Control:

$$\begin{aligned}
J(x) = \min \sum_{k=0}^{N} & \|\hat{e}_k^c(x_k, y_k, \theta_k)\|_{q_c}^2 + \left\|\hat{e}_k^l(x_k, y_k, \theta_k)\right\|_{q_l}^2 - \gamma v_k T_s + \|\mathrm{Var}[\mu_k]\|_{q_v}^2 \\
\text{s.t.} \quad & \theta_0 = \theta_{\text{init}}, \\
& \mathbf{x_0} = \mathbf{x_{init}} \\
& (\bar{\mu_0}, v_0, \Psi_0, \alpha_0) = f_\theta(\mathbf{x_0}), \\
& \theta_{k+1} = \theta_k + v_k T_s, \\
& \bar{\mu}_{k+1} = A\bar{\mu}_k + B_t u_k, \\
& \mathbf{x_k} = C\bar{\mu}_k, \\
& 0 \leq \theta_k \leq L, \\
& 0 \leq v_k, \\
& \mathrm{Var}[\mu_k] \leq \tau_1
\end{aligned} \tag{15}$$

where τ_1 is the threshold for the epistemic uncertainty and $\mathbf{x_t}$ is the state of the system containing the position (x_t, y_t) of the robot.

4.3 Cross-Entropy Method

To find the optimum of our cost function we use the Cross-Entropy Method (CEM). CEM is an optimisation technique that aims to minimise or maximise an objective function through an iterative process. In our case of MPCC, the basic idea is to generate a random sample of actions according to a normal distribution, evaluate the cost function for these actions and then use the results to update the parameters of the distribution to generate a better sample in the next iteration. This process is repeated until convergence is achieved or a certain number of iteration is exceeded. This methods also allows to incorporate constrains and is a common optimisation method for MPC and Reinforcement Learning used for example by Chua et al. (2018) or Liu et al. (2021).

4.4 Practical Implementation

4.4.1 Path Generation

The path generation and the framework in general were implemented using `Python`. To optimize path generation for maximum efficiency and ensure compatibility with the existing framework, we tried to leverage `TensorFlow` functions wherever possible. In cases where `TensorFlow` was unsuitable, we carefully used the `tf.numpy_function()` and limited its use to the beginning of the program, to avoid performance degradation.
Before the simulation and control loop, the path is defined by a series of waypoints. Between each of these waypoints, either a linear or cubic spline is fitted using the `linear_interpolation()` or the `CubicSpline()` method from the `SciPy` library. The spline is then evaluated for a discrete set of values t and the corresponding x and y values together with the orientation of the path (see Equation 9) are saved into a tensor. To achieve the mapping from the arc length to the parameterization t, the arc length is calculated for each value of t using numerical integration based on the trapezoidal rule. For more detial on the implementation see Appendix A.1

4.4.2 Evidential Koopman Contouring Control

Befor the controller can be deployed, the Evidential Network needs to be trained and the Koopman Matrix A, Control Matrix B and Observation Matrix C are learned using a NN. How this is achieved is not in the scope of this thesis and the reader my refer to Amini et al. (2020), Meinert and Lavin (2022) and Han et al. (2023) . For the scope of this thesis, it is important to note that during training the robot was randomly initialized and generated random trajectories built up from fundamental building blocks, within the regions of the environment defined as 'safe'. The robot was no exposed to the unsafe region during training, leading the Evidential Net to predict high epistemic uncertainty for this region.
Also for practical reasons, related to the runtime of the optimization procedure used, we define v_k from Equation 7 as the velocity of the robot giving the controller a indirect control over the progress.

Algorithm 1 Evidential Koopman Contouring Control with CEM

Input: Distribution $\Theta = (\mu, \sigma^2)$, num. samples N, num. elites k, state x_k, epi. threshold τ_1, progress along path θ, path p
Output: optimal action a^*

Initialize the sampling distribution and the last optimal action a_{last}
calculate $v_{projected}$ and update θ
while $<$ maximum iteration steps or $\sigma > \epsilon_1$ **do**
    Sample action sequences $a_0, a_1, \ldots, a_N \sim \mathcal{N}(\mu, \sigma^2)$
    Rollout of the Evidential model to estimate future states $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N$ and epistemic uncertainty $\text{Var}[\mu_k]$
    Estimate cost $c(x_t, \theta$,p) with CalculateCost(x_t, θ, p)
    Select the feasible set $\Omega \leftarrow \{a_j\}$ based on $\text{Var}[\mu_k] < \tau_1$
    if $|\Omega| < k$ **then**
        Pick k elites $\varepsilon \leftarrow \{a_j\}$ by lowest $\text{Var}[\mu_k]$
    else
        Then pick k elites $\varepsilon \leftarrow \Omega$ by lowest c
        Update (μ, σ^2) by weighted average based on ε
    end if
end while
Pick a^* as the first action of the top x_0 in ε

procedure CalculateCost($\mathbf{x}, \theta$, p)
    Calculate $(v_{projected,0}, \ldots, v_{projected,k})$ and $(\theta_0, \ldots, \theta_k)$
    Map arc length θ to t
    Calculate $(x_{\text{ref}}, y_{\text{ref}})$ and Φ
    Calculate $\hat{e}_k^c$ and $\hat{e}_k^l$ based on p
    return Total cost
end procedure

5 Results and Discussion

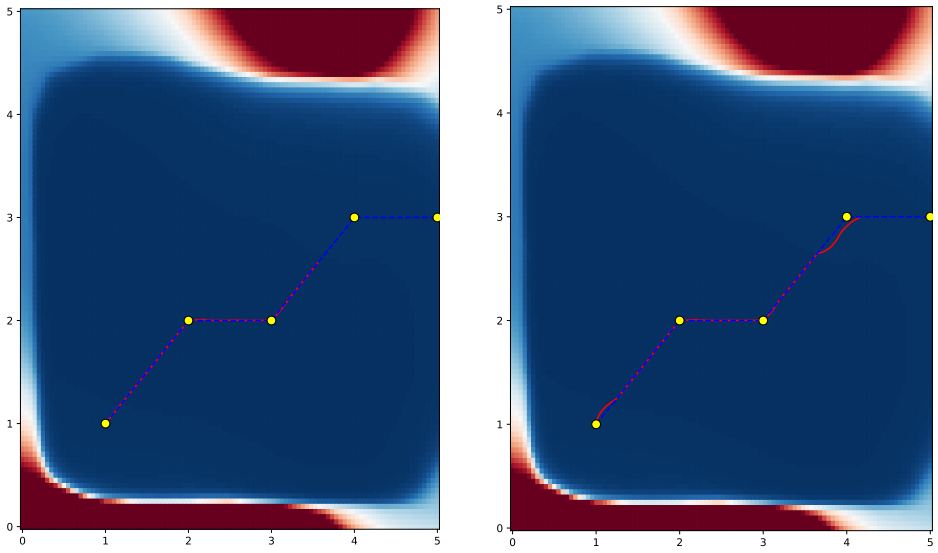
5.1 Environments and Tasks

The Evidential Koopman Contouring Control framework is tested using a TurtleBot in a Pybullet Simulation Environment. The robot's state space is 8-dimensional, consisting of the cosine and sine of the angle $[cos(\theta), sin(\theta)]$, the position $[x, y]$, the rate of change of cosine and sine $[cos'(\theta), sin'(\theta)]$, and the linear velocity $[v_x, v_y]$. An unsafe region is defined, and a global reference path is generated that passes through that region. The robot should initially follow the path and subsequently deviate from it to avoid the unsafe region before returning to the path. To test the framework's general ability to avoid obstacles, constraints were first placed on the robot's position, followed by constraints on the epistemic uncertainty.
All the data generation, training of the model and simulation were done on a GeForce GTX 1660 with CUDA 11.4. For each of these simulation the sampling time on the controller was $T_s = 0.2s$

5.2 Results

5.2.1 Path Tracking

First, the framework's ability to follow a reference path was tested. A path with directional changes was created. The robot was initialized at the beginning of the path with no constraints present and a high weight was placed on the contouring error compared to the progress weight. This was done to ensure that the framework would achieve an accurate traversal of the path rather than a time-optimal one. The framework ran around a 1000 simulation steps, which often wasn't enough to reach the end of the path, but is sufficient to show the ability to follow a path. As can be seen in Figure 4, the framework was able to follow the path with high accuracy, given the correct tuning of the cost function weights. For trials (a) a prediction horizon $N = 80$, contouring error weight $q_c = 10$, error weight $q_l = 1$ and progress weight $\gamma = 4$ were chosen.



(a) (b)

Figure 4: Path following for different paths and weights: Accurate path following is possible, as can be seen in (a). Suboptimal choices can lead to less accurate results, as shown in (b). The key factor for accurate path following is high weight on the contouring error compared to the progress weight.
Note: The background colour coding has no meaning as it represents the predicted epistemic uncertainty, which is not utilised for these tests.

5.2.2 Path Tracking with Position Constrains

Next, the implemented control framework's ability to perform obstacle avoidance in a general case was tested. Constraints were placed on the robot's position to prevent it from crossing a rectangular region. Because no information about the epistemic uncertainty were used in these tests, the term $\|\text{Var}[\mu_k]\|^2_{q_v}$ was replace with

$$J_{\text{repulsive}} = \left(\frac{1}{(\Delta x_k)^2 + (\Delta y_k)^2 + \alpha} \right)$$

where (Δx_k^2) and (Δy_k^2) is the distance of the robot to the obstacle. The framework was tested with two regions of different size. Different sets of parameters for the controller were tested. We chose a prediction horizon $N = 60$, contouring error weight $q_c = 6$, lag error weight $q_l = 3$, progress weight $\gamma = 50$ and $\alpha = 0.055$. As can be seen in Figure 5 the framework was able to avoid the smaller region but not the larger one. It may be possible to find a set of parameters that allow for the avoidance of the larger area. However, due to the long run time of the simulation framework and the limited time frame of this thesis, none were found.

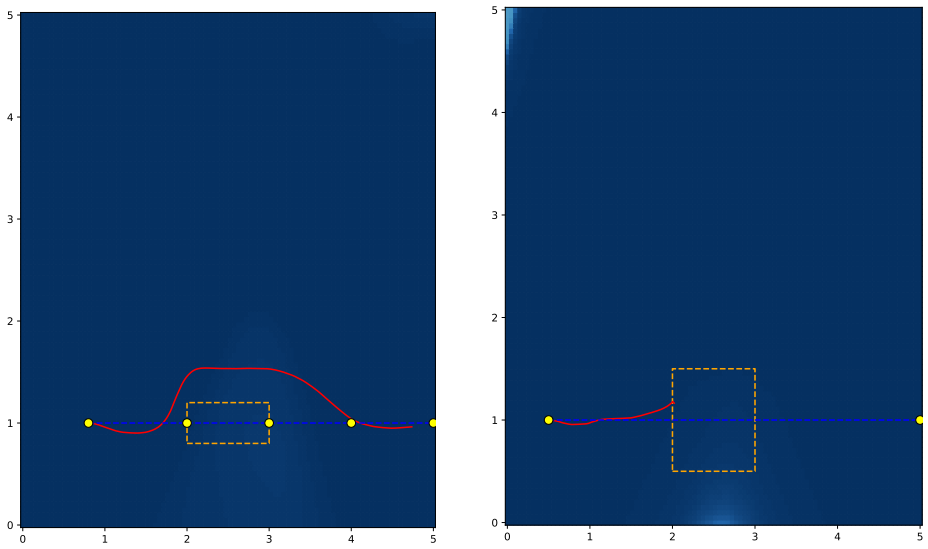
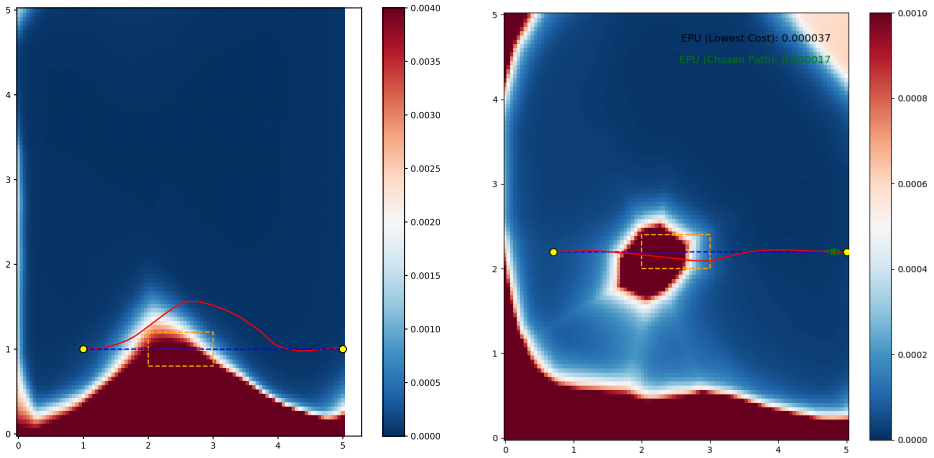


Figure 5: Path tracking with obstacle defined byputting constrains on the position

5.2.3 Path Tracking with Epistemic Uncertainty Constrain

Next, the control framework based on Equation 15 was tested. For the trials in Figure 6 we chose a prediction horizon $N = 60$, contouring error weight $q_c = 6$, lag error weight $q_l = 3$, progress weight $\gamma = 50$, epistemic uncertainty weight $q_v = 1000$ and a epistemic uncertainty threshold $\tau_1 = 0.0001$. Figure 6a shows that the framework was able to follow the path while avoiding the unsafe region. As can be seen in Figure 7, the robot never enters a state with an epistemic uncertainty higher than the τ_1, with the exception of a single outlier at the very end of the control process. However, in Figure 6b, the same set of parameters used on a different dataset with a region of the same size did not yield a successful result. The difference in epistemic uncertainty values across different datasets is the most likely cause of the problem, along with insufficient characterisation of the uncertain area by increased epistemic uncertainty. For more detail see Appendix A.3



(a) (b)

Figure 6: Path tracking with constrain on epistemic uncertainty: Trial (a) shows the controller can deviate from it path to avoid the unsafe region, and later comes back onto the path. Trial (b) shows the inability of the controller to be used on a different dataset without changes to the control.
Note: The heat map in the background gives an approximation of the predicted epistemic uncertainty.

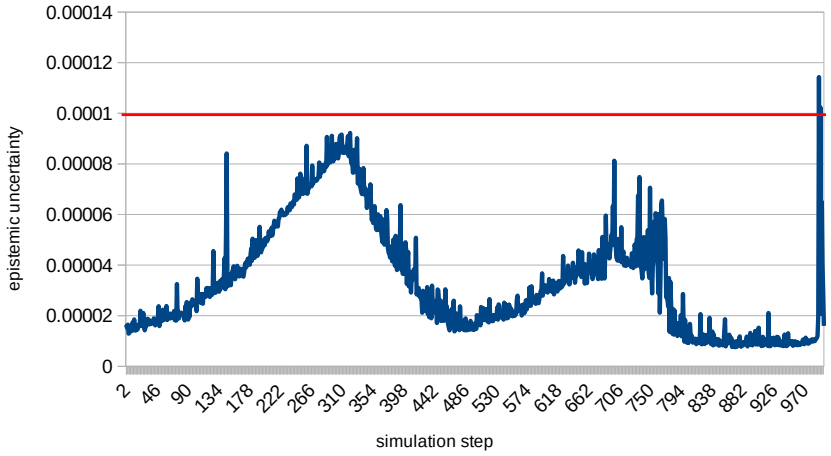


Figure 7: Epistemic uncertainty of the state at each simulation step for Trial (a)

6 Conclusion

In this thesis, we expanded an existing Deep Evidential Koopman Control framework by incorporating a MPCC formulation for path tracking. We leveraged information about the epistemic uncertainty to avoid regions that the robot was not exposed to during training. We created a code base to efficiently generate a global reference path suitable for the MPCC formulation. The framework was tested and its ability to accurately follow a path, handle position constraints, and avoid unsafe regions was shown. Limitations of the current implementation were discussed and possible improvements are suggested.

7 Future Work

Based on the current limitations of the Deep Evidential Koopman framework and its application within the Evidential Koopman Contouring Control framework, the following improvements are proposed for the future.

Improvement of the optimisation procedure

The current implementation of the Evidential Koopman Contouring Control has a high runtime, making tuning of controller parameters slow and difficult and rendering its application on real-time systems impossible. This is mainly due to the use of CEM for the optimization procedure, which suffers from a high computational cost and consequently slow run time, making its basic form unsuitable for real-world applications (Pinneri et al., 2020). Thus, the implementation of an improved CEM or an entirely different optimization scheme is proposed.

Improvement of the data generation

The current implementation of the data generation scheme is suboptimal. Its limitation stems from the limited data size that can be generated and from the samples not being evenly distributed across the entire state space. This leads to an insufficient differentiation between safe and unsafe areas based on the predicted epistemic uncertainty. See Appendix A.3 for more detail on this problem.

Improvement and additional testing of the control framework

While the current Evidential Koopman Contouring Control framework is functional, its performance is limited, highly affected by quality of data and dependent on manual fine-tuning of controller parameters. Therefore, additional research is required to improve the framework's performance and robustness and additional testing and comparison to comparable control frameworks are needed. It would be interesting to expand the framework to more complex system and leverage uncertainty information about other states, beside the position.

References

Amini, A., Schwarting, W., Soleimany, A., & Rus, D. (2020, November). Deep Evidential Regression [arXiv:1910.02600 [cs, stat]]. https://doi.org/10.48550/arXiv.1910.02600 (cit. on pp. 1, 2, 8).

Brito, B., Floor, B., Ferranti, L., & Alonso-Mora, J. (2020, October). Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments [arXiv:2010.10190 [cs] version: 1]. https://doi.org/10.48550/arXiv.2010.10190 (cit. on pp. 3, 5).

Brunton, S. L., Budišić, M., Kaiser, E., & Kutz, J. N. (2021, October). Modern Koopman Theory for Dynamical Systems [arXiv:2102.12086 [cs, eess, math]]. https://doi.org/10.48550/arXiv.2102.12086 (cit. on pp. 2, 4).

Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018, November). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models [arXiv:1805.12114 [cs, stat]]. https://doi.org/10.48550/arXiv.1805.12114 (cit. on p. 7).

Fan, D. D. (2021). Safe Robot Planning and Control Using Uncertainty-Aware Deep Learning [Publisher: Georgia Institute of Technology]. Retrieved October 9, 2023, from http://hdl.handle.net/1853/65047 (cit. on p. 1).

Han, M., Euler-Rolle, J., & Katzschmann, R. K. (2022). DESKO: STABILITY-ASSURED ROBUST CONTROL OF NONLINEAR SYSTEMS WITH A DEEP STOCHASTIC KOOPMAN OPERATOR (cit. on pp. 1, 2).

Han, M., Wong, K., Euler-Rolle, J., Zhang, L., & Katzschmann, R. K. (2023). Robust Learning-Based Control for Uncertain Nonlinear Systems With Validation on a Soft Robot. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15. https://doi.org/10.1109/TNNLS.2023.3328643 (cit. on p. 8).

Kabzan, J., Hewing, L., Liniger, A., & Zeilinger, M. N. (2019). Learning-Based Model Predictive Control for Autonomous Racing. *IEEE Robotics and Automation Letters*, *4*(4), 3363–3370. https://doi.org/10.1109/LRA.2019.2926677 (cit. on p. 3).

Kabzan, J., Valls, M. d. l. I., Reijgwart, V., Hendrikx, H. F. C., Ehmke, C., Prajapat, M., Bühler, A., Gosala, N., Gupta, M., Sivanesan, R., Dhall, A., Chisari, E., Karnchanachari, N., Brits, S., Dangel, M., Sa, I., Dubé, R., Gawel, A., Pfeiffer, M., … Siegwart, R. (2019, May). AMZ Driverless: The Full Autonomous Racing System [arXiv:1905.05150 [cs]]. https://doi.org/10.48550/arXiv.1905.05150 (cit. on p. 3).

Lam, D., Manzie, C., & Good, M. (2010). Model predictive contouring control [ISSN: 0191-2216]. *49th IEEE Conference on Decision and Control (CDC)*, 6137–6142. https://doi.org/10.1109/CDC.2010.5717042 (cit. on p. 2).

Liniger, A., Domahidi, A., & Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale RC cars [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/oca.2123]. *Optimal Control Applications and Methods*, *36*(5), 628–647. https://doi.org/10.1002/oca.2123 (cit. on p. 2).

Liu, Z., Zhou, H., Chen, B., Zhong, S., Hebert, M., & Zhao, D. (2021, March). Constrained Model-based Reinforcement Learning with Robust Cross-Entropy Method [arXiv:2010.07968 [cs]]. https://doi.org/10.48550/arXiv.2010.07968 (cit. on p. 7).

Meinert, N., & Lavin, A. (2022, February). Multivariate Deep Evidential Regression [arXiv:2104.06135 [cs, stat]]. https://doi.org/10.48550/arXiv.2104.06135 (cit. on pp. 2, 8).

Narkhede, K. S., Thanki, D. A., Kulkarni, A. M., & Poulakakis, I. (2023, July). Overtaking Moving Obstacles with Digit: Path Following for Bipedal Robots via Model Predictive Contouring Control [arXiv:2308.00119 [cs] version: 1]. https://doi.org/10.48550/arXiv.2308.00119 (cit. on p. 3).

Pinneri, C., Sawant, S., Blaes, S., Achterhold, J., Stueckler, J., Rolinek, M., & Martius, G. (2020, August). Sample-efficient Cross-Entropy Method for Real-time Planning [arXiv:2008.06389 [cs, stat]]. Retrieved April 14, 2024, from http://arxiv.org/abs/2008.06389 (cit. on p. 12).

Rokonuzzaman, M., Mohajer, N., Nahavandi, S., & Mohamed, S. (2021). Model Predictive Control With Learned Vehicle Dynamics for Autonomous Vehicle Path Tracking [Conference Name: IEEE Access]. *IEEE Access*, *9*, 128233–128249. https://doi.org/10.1109/ACCESS.2021.3112560 (cit. on p. 1).

Romero, A., Sun, S., Foehn, P., & Scaramuzza, D. (2022, May). Model Predictive Contouring Control for Time-Optimal Quadrotor Flight [arXiv:2108.13205 [cs] version: 4]. https://doi.org/10.48550/arXiv.2108.13205 (cit. on pp. 3, 4, 6).

Schmoeller da Roza, F., Hadwiger, S., Thorn, I., & Roscher, K. (2023). Towards Safety Assurance of Uncertainty-Aware Reinforcement Learning Agents. Retrieved September 22, 2023, from https://publica.fraunhofer.de/handle/publica/441238 (cit. on p. 1).

A Appendix

A.1 Supplementary Code path generation

In the following we want to present some of the implemented code for the path generation. For practical reasons, only some portions of the code are provided.

```python
def create_periodic_spline_tf(waypoints):
    """
    Creates a periodic cubic spline interpolation for a set of 2D waypoints.

    :param waypoints: A list of 2D points (each point is a tuple or list of two elements).
    :return: A TensorFlow-compatible spline function.
    """
    # Validate waypoints
    if len(waypoints) < 2:
        raise ValueError("Waypoints should be a list of at least 2 points.")

    waypoints = np.asarray(waypoints)
    if waypoints.ndim != 2 or waypoints.shape[1] != 2:
        raise ValueError("Waypoints should be a 2D array with shape (n_points, 2).")

    # Extend waypoints periodically
    extended_waypoints = np.vstack([waypoints, waypoints[0]])

    n = len(waypoints)  # Number of waypoints
    t = np.arange(n + 1)  # Create an array of indices for the waypoints

    # Create periodic cubic spline
    spline = scipy.interpolate.CubicSpline(t, extended_waypoints, bc_type='periodic')

    # Wrap in a TensorFlow function for compatibility
    def tf_spline(x):
        # Ensure x is a 2D tensor
        original_shape = tf.shape(x)
        x = tf.cast(x, tf.float32)  # Cast x to float32 for compatibility
        x_flattened = tf.reshape(x, [-1])  # Flatten x to apply spline

        # Use numpy_function to apply the spline function over each element of x_flattened
        result = tf.numpy_function(spline, [x_flattened], tf.float32)

        # Reshape result to 3D tensor of shape (m, n, 2)
        result = tf.reshape(result, tf.concat([original_shape, [2]], axis=0))

        return result

    return tf_spline

    def generate_path(number_of_steps, waypoints):
        # generate spline
        spline = create_periodic_spline_tf(waypoints)
        n = len(waypoints)
        # Generate t values
        t_values = tf.linspace(0.0, n - 1, number_of_steps)
        # Assign the first row with t values
        lookup_table = tf.Variable(tf.zeros((5, number_of_steps), dtype=tf.float32))
        lookup_table[0].assign(tf.cast(t_values, tf.float32))

        # Apply the spline function to each t value to get x and y coordinates
        coordinates = spline(tf.reshape(t_values, (1, -1)))

        # Fill the second and third rows with x and y coordinates respectively
        lookup_table[1].assign(tf.cast(coordinates[0, :, 0], tf.float32))  # X coordinates
```

```
        lookup_table[2].assign(tf.cast(coordinates[0, :, 1], tf.float32))  # Y coordinate

        # Assuming delta_x and delta_y need to be calculated without tf.diff
        delta_x = lookup_table[1, 1:] - lookup_table[1, :-1]
        delta_y = lookup_table[2, 1:] - lookup_table[2, :-1]

        angles = tf.atan2(delta_y, delta_x)
        angles = tf.concat([[0], angles],
                           axis=0)

        # Assign the calculated angles to the last row of the lookup table
        lookup_table[3].assign(tf.cast(angles, tf.float32))
        delta_t = (n-1)/number_of_steps

        arclenghts = compute_arc_length_tf(spline, t_values, delta_t)

        lookup_table[4].assign(tf.cast(arclenghts, tf.float32))

        return lookup_table

    def compute_arc_length_tf(tf_spline, t_values, delta_t):
        # Using the trapezoidal rule for numerical integration
        velocity = compute_derivatives(tf_spline, t_values, delta_t)
        velocity = velocity * delta_t
        arc_lengths = tf.cumsum(velocity)
        return arc_lengths

        def extract_closest_value(input_tensor, lookup_table,typ):
        # Expand dims of input_tensor and t_values for broadcasting
        expanded_input = tf.expand_dims(input_tensor, -1)  # Shape becomes (m, n, 1)
        t_values = tf.expand_dims(lookup_table[0], 0)  # Shape becomes (1, number_of_steps)

        # Calculate absolute differences with broadcasting
        abs_differences = tf.abs(expanded_input - t_values)

        # Find the indices of the minimum differences
        closest_indices = tf.argmin(abs_differences, axis=-1)

        # Extract the x values for the closest indices
        value = tf.gather(lookup_table[typ], closest_indices)

        return value

    def extract_x_from_path(input_tensor, lookup_table):
        return extract_closest_value(input_tensor, lookup_table,1)
    def extract_y_from_path(input_tensor, lookup_table):
        return extract_closest_value(input_tensor, lookup_table,2)

    def get_projected_velocity(progress_along_path, vx, vy, lookup_table):

        t = extract_t_from_arc(progress_along_path, lookup_table)

        angle = extract_angle_from_path(t, lookup_table)

        # Calculate the unit vector components of the path
        ux = tf.cast(tf.cos(angle), tf.float32)
        uy = tf.cast(tf.sin(angle), tf.float32)

        vx = tf.cast(vx, tf.float32)
        vy = tf.cast(vy, tf.float32)

        # Project vx and vy onto the path
        pv = ux * vx + uy * vy
        pv = tf.abs(pv)
        return pv
```

Listing 1: Code of path generation

A.2 Supplementary Code MPCC

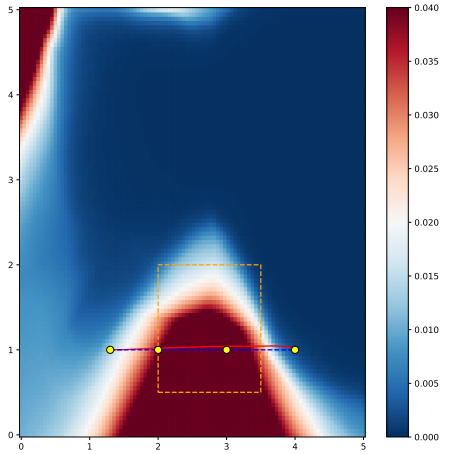
In the following we want to present some of the implemented code for controller. For practical reasons, only some portions of the code are provided.

```

    def stage_cost(self, x, action, path, epi_norms):

        x_temp = (x * self.scale) + self.shift

        velocity = tf.sqrt(tf.square(x_temp[:, :, 6]) + tf.square(x_temp[:, :, 7]))

        # calculate progress
        progress = velocity * 0.2
        theta = find_closest_point_arc_length_v2(x_temp[:, :, 2], x_temp[:, :, 3],
    path)

        # mapping from arc length theta to parameter t
        t = extract_t_from_arc(theta, path)

        # calculate ref position on path
        x_ref = extract_x_from_path(t, path)
        y_ref = extract_y_from_path(t, path)

        # calculate angle/orientation of path
        angle = extract_angle_from_path(t, path)

        ce = self.calc_contouring_error(x_temp, x_ref, y_ref, angle)
        le = self.calc_lag_error(x_temp, x_ref, y_ref, angle)

        delta_theta = theta - self.progress_along_path

        cost = (6 * tf.math.square(ce) + 3 * tf.math.square(le) -  50* progress +
    10 * epi_norms)


        cost = tf.concat([cost[:, :-1], (cost[:, -1] * self.end_weight)[:, tf.
    newaxis]], axis=1)

        return cost

    def calc_contouring_error(self, x, x_ref, y_ref, angle):
        x_position = x[:, :, 2]
        y_position = x[:, :, 3]
        ce = tf.sin(angle) * (x_position - x_ref) - tf.cos(angle) * (y_position -
    y_ref)
        return ce

    def calc_lag_error(self, x, x_ref, y_ref, angle):
        x_position = x[:, :, 2]
        y_position = x[:, :, 3]
        le = -tf.cos(angle) * (x_position - x_ref) - tf.sin(angle) * (y_position -
    y_ref)
        return le
```

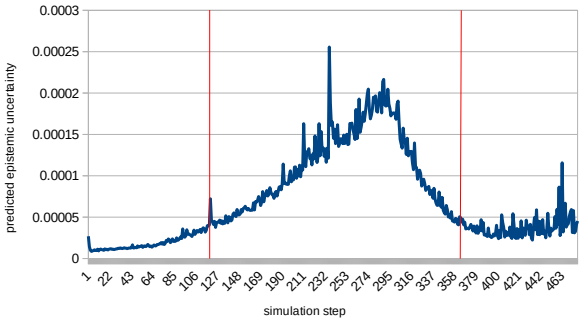
Listing 2: Code of the controller

A.3 Additional Data

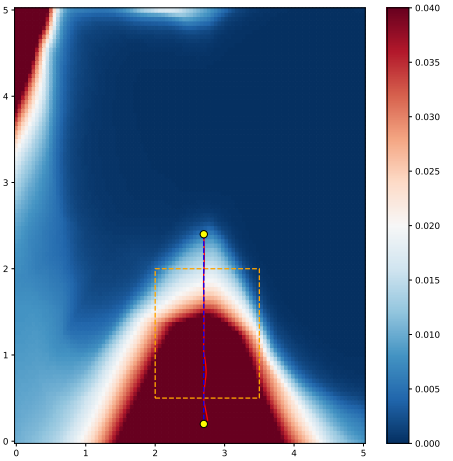
The purpose of this section is to present additional data on the predicted epistemic uncertainty within the unsafe region. For each of these datasets, 250,000 samples were generated and a evidential net was trained with 300 epochs and a batch size of 256. To obtain the predicted epistemic uncertainty of the unsafe region, the robot was steered through the unsafe region and at each time step the current state of the system was passed through the evidential net to obtain the predicted epistemic uncertainty. As can be seen, it is not always the case that the inside of the unsafe region is characterized by a higher epistemic uncertainty compared to the surrounding area. The area's size and position within the environment appear to have an effect. The reason for this is probably due to two shortcomings in the current data generation implementation. During training not the entire area is evenly explored and the data size is limited. During data generation, the robot is initialized at a random position and moves until it exceeds a certain number of actions or enters the unsafe region, then the robot is reinitialized at a new position. If the unsafe region is located at the edge of the environment, it covers the area behind it, which the robot can only reach if it is initialized there. Leading to less the data for this area and a higher epistemic uncertainty. This effect can be seen when comparing Figure 9 with Figure 10 and Figure 11 Figure 12. This effect could be reduced with a larger dataset. However, the current implementation on the used hardware causes memory issues when attempting to generate a larger dataset.



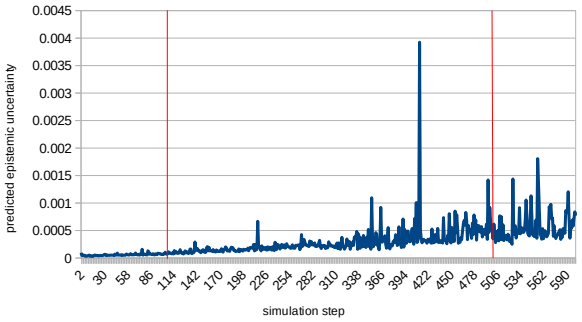
(a)



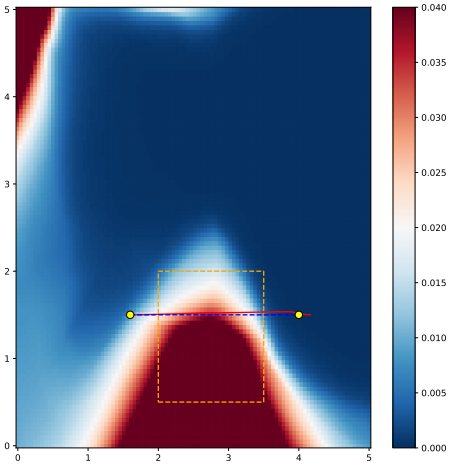
(b)



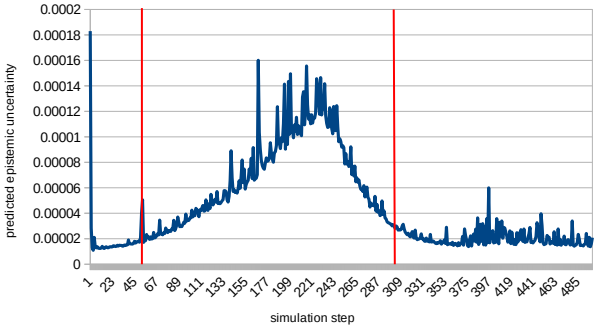
(c)



(d)

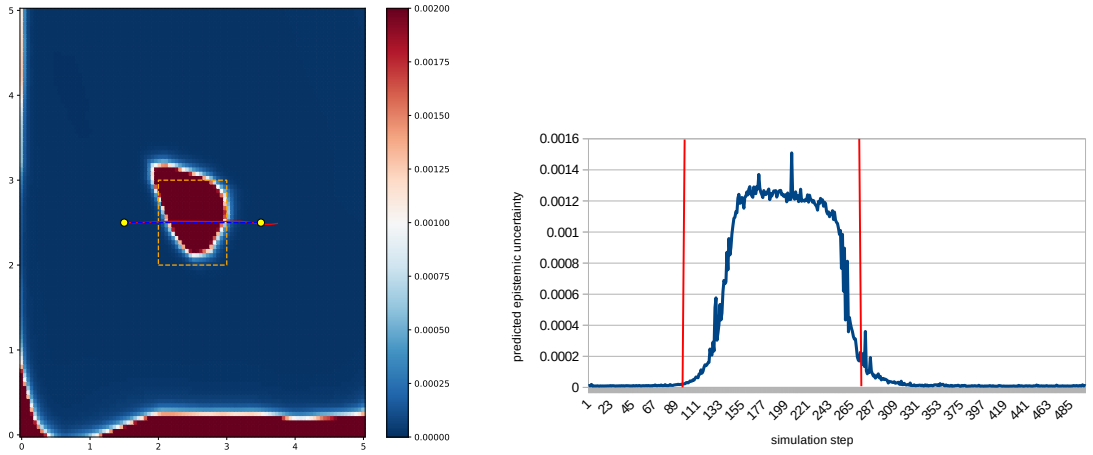


(e)



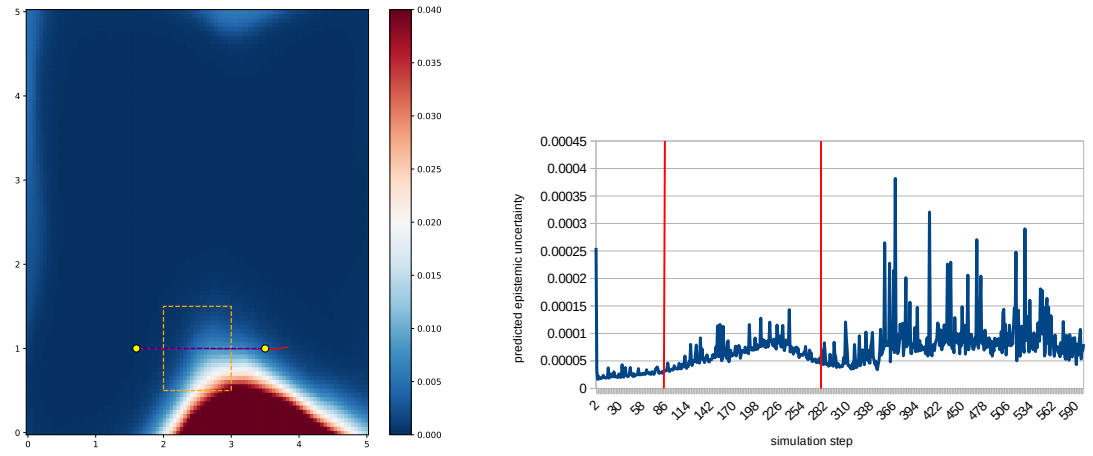
(f)

Figure 8: Data set 1

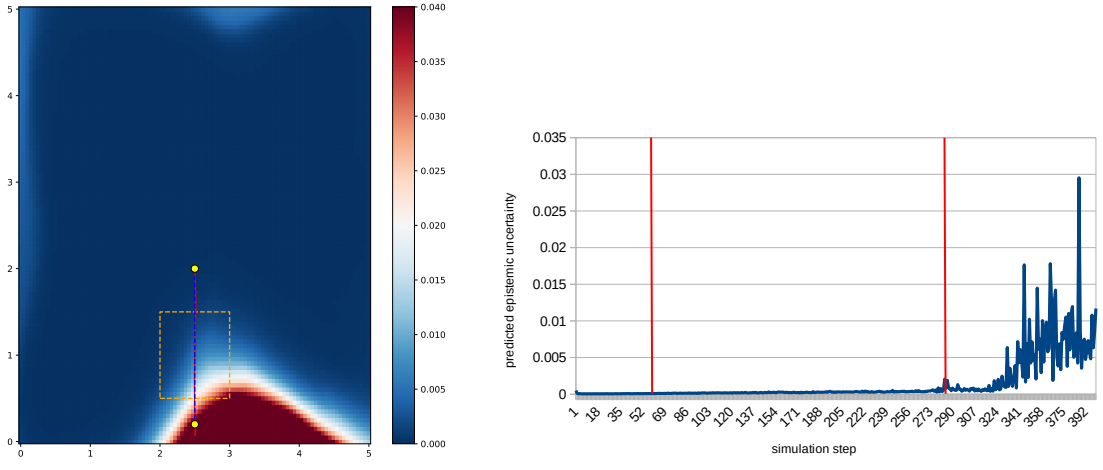


(a) (b)

Figure 9: Data set 2

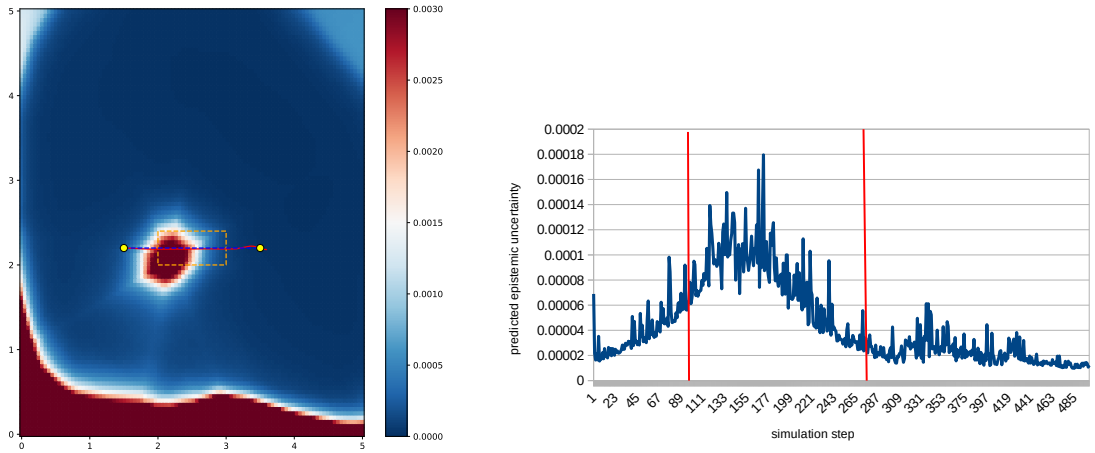


(a) (b)



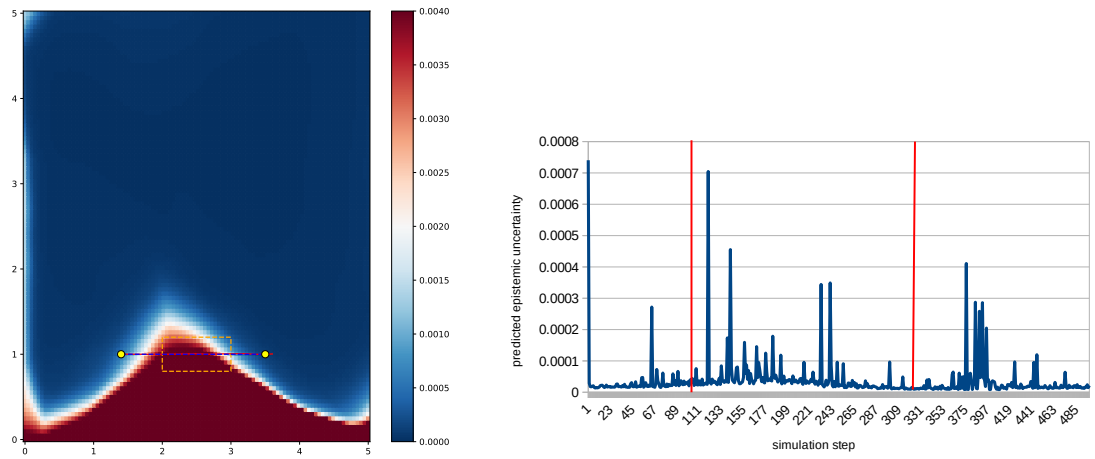
(c) (d)

Figure 10: Data set 3



(a) (b)

Figure 11: Data set 4



(a) (b)

Figure 12: Data set 5

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Title of work:

Model Predictive Contouring Control with Uncertainty Estimation using Deep Evidential Koopman Operators

Thesis type:

Bachelor Thesis

Student:

Name:	Luca Lietha
E-mail:	llietha@student.ethz.ch
Student ID:	19-939-289

Supervisors:

Prof. Dr. Robert Katzschmann
Minghao Han

Declaration of originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Signature: Zürich, April 15, 2024