Anomaly Detection and Classification to enable Self-Explainability of Autonomous Systems

Florian Ziesche, Verena Klös, Sabine Glesner
Software and Embedded Systems Engineering
Technische Universität Berlin
Berlin, Germany

{verena.kloes, sabine.glesner}@tu-berlin.de

***Abstract*—While the importance of autonomous systems in our daily lives and in the industry increases, we have to ensure that this development is accepted by their users. A crucial factor for a successful cooperation between humans and autonomous systems is a basic understanding that allows users to anticipate the behavior of the systems. Due to their complexity, complete understanding is neither achievable, nor desirable. Instead, we propose self-explainability as a solution. A self-explainable system autonomously explains behavior that differs from anticipated behavior. As a first step towards this vision, we present an approach for detecting anomalous behavior that requires an explanation and for reducing the huge search space of possible reasons for this behavior by classifying it into classes with similar reasons. We envision our approach to be part of an explanation component that can be added to any autonomous system.**

***Index Terms*—**

I. Introduction

Autonomous Systems, like autonomous driving assistants or mobile robots in logistics or health care, perform more and more tasks for us. They become an important part of our daily life and often require a close cooperation between humans and autonomous systems. A prerequisite for the acceptance of such systems and for successful cooperation is that humans are able to anticipate the behavior of the systems. However, autonomous systems are often complex and, thus, there may be situations in which their behavior differs from the anticipated behavior and requires an explanation. Ideally, this explanation is given by the system itself. We call such a system self-explainable. A self-explainable system should detect the need for explanations autonomously and give a reason for its current behavior.

In this paper, we assume that the underlying autonomous system has not been built to be self-explainable. Instead, we propose to add an explanation component to the system that observes the behavior of the system, detects the need for an explanation and produces it accordingly. The explanation component is decoupled from any safety-critical components, e.g. failure detection for hardware components. Thus, it does not need to follow the same strict rules for development and certification. In contrast to model-driven approaches for explainability (e.g., [1], [2]), we do not assume the availability of a behavioral model of the system. Instead, we rely on supervised learning to detect anomalous behavior that may require an explanation. To cope with the inherent complexity of autonomous systems, we combine this detection with a classification of behavior into classes with similar reasons, e.g. sensor failures or safety policies. As a result, we are able to limit the number of potential reasons, and thus, to reduce the search space for explanations.

In this paper, we present our vision of an explanation component that can be added to autonomous systems to achieve self-explainability and show how a recurrent neural network can be used for the detection and classification of anomalous behavior. We evaluate our approach on a simulation of an autonomous racing car that was built in a student project. The simulation enables us to easily generate labeled training data for different anomaly classes.

In Section II, we present our vision of the explanation component. Afterwards, we present our anomaly detection and classification approach in Section III and evaluate its performance in Section IV. We discuss related work in Section V and give a conclusion in Section VI.

II. Vision of the Explanation Component

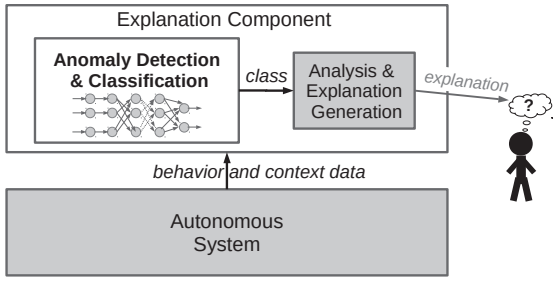

Fig. 1. General Architecture of the Explanation Component

Our vision is an explanation component that can be added to any autonomous system to generate explanations for its behavior on demand. To this end, it observes the behavior and the context of the system, detects behavior that requires an explanation because it differs from the usual behavior, and generates explanations for it. This decoupling has two advantages: the explanation is decoupled from any safety-critical component, and thus, does not need to be tested as thoroughly, and our approach can also be applied to third-party software. The general architecture of this component is

depicted in Figure 1. To reduce the search space of possible reasons for anomalous behavior, we propose to combine the *anomaly detection* with a *classification* that divides the anomalous behavior into classes with similar reasons. These classes guide a subsequent *analysis* to identify the reasons for the observed behavior. In the end, an *explanation generation* produces an understandable explanation for the human that requires it. In this paper, we focus on the anomaly detection and classification. We aim to develop solutions for the challenging task of generating suitable explanations in future work. We describe our approach in the following section.

III. Anomaly Detection & Classification

The goal of our anomaly detection is to identify anomalous behavior that may be unexpected for humans that interact with the system and, thus, requires an explanation. To detect anomalous behavior, several anomaly detection algorithms exist, as surveyed in [3]. However, a complex autonomous system, like an autonomous car, may have plenty of possible reasons for anomalous behavior (e.g., failures of sensors, changes in the operational environment, and safety critical situations). Because of the huge number of possible reasons, identifying the actual reason for an anomaly is difficult. In order to shrink this search space for each anomaly we propose to divide anomalies into classes with similar reasons and to combine the anomaly detection with a classification. As an example, we group sensor, motor and connection failures into the class *hardware failure*. In a subsequent analysis step, the class information can be used to select the data that has to be evaluated for further information. In case of a hardware failure, for example, only sensor, motor and communication data has to be further evaluated. Thus, in addition to the reduction of the search space, we also get valuable information on the kind of failure that can be used to apply tailored analysis techniques.

In the following, we first motivate the use of recurrent neural networks for the detection and classification, then, we describe how we select the observations and classes. Afterwards, we describe our general architecture for the neural network.

A. Choice of the Classification Algorithm

Our classification algorithm has to fulfill the following requirements:

1) Time Series: To detect and classify anomalous behavior, it is not sufficient to evaluate a single point in time. Instead, the development of the behavior and the operational environment over time have to be evaluated. Thus, the algorithm has to process time series data.

2) Multiple Time Series: For correct classification, we need to analyze the behavior of the system together with the operational environment and other influencing factors like the battery status, which give important clues for locating the reason for the anomalous behavior. Processing each variable separately is not sufficient because anomalies are often caused by a combination of influences. Therefore the classification algorithm has to be able to evaluate multiple time series together.

3) Estimate probability of belonging to a class: In order to allow for further processing based on the probability of an anomaly belonging to each of the classes, the component should output this probability instead of assigning a single class.

4) General applicability: The approach should be applicable to various applications and should not require knowledge of the system implementation or details on the internal decision logic.

Classification tasks can be solved by many approaches. Considering the requirements for our system, we choose a recurrent neural network that is especially suited to process time series data and able to work with large amounts of input variables even if these are time series. We choose a recurrent network with Gated Recurrent Units (GRU) [4]. A GRU consists of a neuron as well as a reset and an update gate. The reset gate learns when to drop or forget information from past steps to reset the value in the neuron with the current input to the neuron. The update gate learns how much the past time step influences the current value. The training process of a neural network forms the necessary connections between the input variables, without the need for a human analyzing the cause-effect chains of the autonomous system. With a *softmax* activation in the output layer, the network outputs the probability of the anomaly belonging to each of the classes.

B. Data Selection

As input for our anomaly detection and classification, we select parameters that describe the behavior of the system as well as additional parameters that describe its context. This selection of parameters is highly application specific. We provide an example with our case study. Although behavioral parameters are important to detect anomalous behavior, they may not suffice to pinpoint its cause. Thus, we need additional information about the cause of anomalous behavior, such as parameters that describe the operational context of the system or other system parameters like the battery status or the stability of a wireless connection. We call these *context parameters*, because they deliver context for the behavior.

As an example, we assume that the behavior of a simple autonomous car can be completely described by its steering angle and its speed. Thus, these are the behavioral variables. If the car stops, we can detect that by noticing that the speed of the car is zero. However, this does not give us any indication on the reason why the car stopped. It might be because a preceding car has stopped, because of a red light or because of a motor defect. With additional context parameters like the distance to any object in front of the car, a traffic light detection and a motor status, we can identify the reason for stopping.

We track the development over time of each selected parameter by saving a certain number of previous values. These time series form the input for our neural network. The number of saved time steps for each parameter depends on the available computational power of the classification component, as well as the time available for training the network. The time span between the samples has to be chosen with care to achieve a suitable snapshot of the current development. A time frame of a few milliseconds, for example, might not encompass the

bigger picture of the current circumstance. In contrast, a time series covering a sensible amount of time but having only few points might not include all of the data that is necessary for precise classification. We propose to use a time frame of a few seconds with a period length of a few hundred milliseconds as a suitable trade-off between these two influences.

C. Selection of Classes

We require the classes to reflect the reason or a group of reasons for an anomalous behavior. Reasons from different classes may have the same observable effect. In the case of a car, for example, the car might stop because it detects congestion ahead, or because it detects a flat tire. Both of these anomalies have the same effect, namely the car stopping, but they can belong to two different classes, e.g., *Traffic* and *Hardware Failure*. The number of classes influences the complexity of the task. With more classes, i.e. more precise class definitions, we get more information for the generation of explanations. At the same time, however, increasingly complex connections have to be learned by the network. Thus, it is important to choose a small number of clearly separated classes to reduce the training time, risk of overfitting and necessary amount of training data.

The choice of classes also depends on the available data. We can only classify an anomaly into a class if we have access to relevant context data. For example, a traffic class that contains anomalies due to the influence of traffic, such as diverting from a route because of a traffic congestion, is only sensible when the autonomous car has data on the traffic conditions. Thus, the definition of classes depends on the complexity of the autonomous system, the available data, available training data and the required class-resolution.

D. Neural Network Architecture

In this section, we present a general architecture for a recurrent neural network that is easily adjustable to the specific number of classes and input variables.

The input is given in form of time series data. We propose to normalize each input time series to be in the interval $[0, 1]$. This is possible because the possible ranges of sensors values are usually known. We define the number of input GRUs for our proposed network as $\#input\ neurons = \#input\ parameters \cdot time\ series\ length$. The network architecture is split into two parts, as illustrated in Figure 2. It consists of a recurrent layer, suitable for detecting local or time-invariant patterns in the series, followed by a multilayer classifier. To prevent overfitting, we add a dropout layer between them. During training, it randomly disables a certain fraction of the connections between the recurrent layer and the subsequent dense layer during the training process. This forces the dense layers to classify robustly. The number and width of the dense layers in the classifier are hyperparameters of this approach, allowing wider or deeper networks for more complex classification problems. The output layer has as many neurons as the number of classes and uses softmax activation, so that each output can be interpreted as the probability that the current behavior belongs to the corresponding anomaly class.

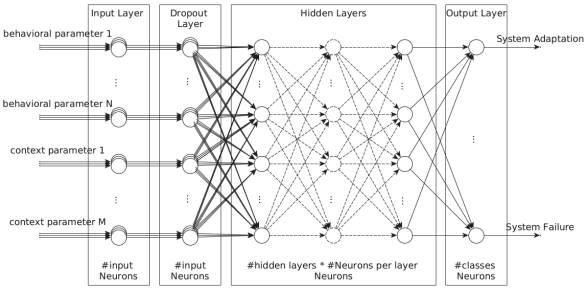

Fig. 2. General Architecture of the Classification Network

IV. Evaluation

We evaluate our approach by applying it on simulated data of an autonomous racing car that has been developed in a student project. In the following, we first introduce the autonomous racing car and instantiate our anomaly detection and classification for this case study. Subsequently, we evaluate the performance of our neural network.

A. Case Study: Autonomous Racing Car

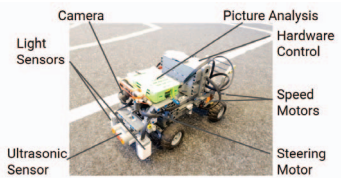

Fig. 3. LEGO Robot with Raspberry Pi

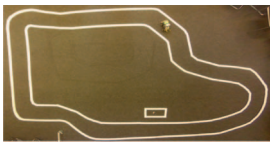

Fig. 4. Track built with white tape

As a case study, we use an autonomous racing car that has been developed in a student project (cf. Fig. 3). It consists of a LEGO Mindstorm robot, an additional Raspberry Pi wide angle camera and a Raspberry Pi for image processing and neural network calculations. The aim of the project was to build a robot that is able to autonomously drive on a racing track without crossing the borders or driving into static obstacles. The track is marked with white tape on a black floor (cf. Fig. 4). In the developed robot, the camera takes pictures of the track in front of the robot and the neural network calculates a steering angle and a speed for each picture. Steering angle and speed are sent via Bluetooth to the driving system hat is located on the LEGO mindstorm brick and controls the motors. The robot is equipped with three motors: a steering motor and two propulsion motors for the rear wheels. The steering motor changes the angle of the steering axle to drive turns. The other two motors each move one of the rear wheels to move the robot with the desired speed.

The neural network has been trained to drive safely within the track. Additionally, a safety system has been implemented that stops the robot if it crosses the track borders, if an obstacle is in its path, or if the Bluetooth connection between the neural network and the driving system is lost. For the safety system, the robot is equipped with multiple sensors. Two brightness

sensors in the front for detecting the track borders, and an ultrasonic sensor to measure the distance between the robot and any obstacle in front of it. The loss of the Bluetooth connection is noticed if the robot does not receive any instructions from the neural network for a specific amount of time.

For our evaluation, we concentrate on the driving system and use an abstract simulation of the system to generate training data. Using simulated data allows the construction of much larger and more balanced datasets, since the simulation can run much faster than real time. Obtaining enough data about anomalous behavior in a real robot can be challenging since it is, by definition, much less common. In future work, we want to investigate how the resulting ML model can be adapted for real data from the robot with a much smaller dataset, in a process called *transfer learning* [5]. For faithful enough simulations, this transfer may even be possible without any new training [6]. In the following, we describe the instantiation steps for our anomaly detection and classification approach.

1) Data Selection: We monitor all inputs and outputs of the driving system, namely the desired steering angle and speed given via Bluetooth to the robot, the status of the Bluetooth connection, the speed that was calculated for the propulsion motors, the current angle of the steering axle, the brightness values for both brightness sensors and the distance to any obstacle provided by the ultrasonic sensor. These values are used to detect and classify anomalous behavior.

2) Classes: We divide the anomalies into three classes: *System Adaptation*, *Emergency Halt* and *System Failure*. Each class groups a family of anomalies with similar causes.
The *System Failure* class encompasses any anomaly which is caused by failures of system components. Specifically any anomaly caused by failures of one or both of the light sensors, or the ultrasonic sensor is part of this class. When the robot detects these failures, it stops until the failure is repaired. Although, the robot has a failure detection system, we have to detect these failures again, due to the decoupling of system and explanation component.
Emergency Halt is the class of anomalies which are caused by the safety features of the robot that stop the robot when driving without direction commands provided via the Bluetooth connection, driving over the track border, or if the robot is about to crash into an obstacle in front of the robot.
The *System Adaptation* class encompasses any anomaly that is caused by an adjustment of the system to the current circumstances. In our racing car scenario, we assume that any uncommon driving behavior, such as driving a zigzag course, falls into this class.

Note that in this example the classification approach with a neural network is not necessary as these anomalies could be detected with simpler means, e.g. failure logs. However, we require our approach to be able to classify anomalous behavior of any black box system. The small robot example is well suited to illustrate how our approach can be applied.

3) Generation of training data: In order to train the neural network, we have generated labeled training and test data with a simulation of the robot. This approach enables us to generate a sufficiently large dataset with an equal distribution of anomaly classes. Data from real executions is usually unbalanced as anomalous behavior can be observed less often than normal behavior. To cope with unbalanced datasets several methods have been proposed. See [7] for a review.

a) Simulation Environment: To generate training and test data, we have implemented a simulation that resembles the behavior of the real robot. The simulation receives a steering angle and realistic sensor values as input and computes the motor speed that would be calculated by the robot controller for this input. The main difference between simulation and real robot is, that we we do not consider friction in our simulation. We use a sample time of 200 ms and save the data as a collection of time series in a file which is later used to train and evaluate the classification network. We have chosen to save time series with a length of 15 elements, which corresponds to a history of the last three seconds. The length of the time series is a trade-off between precision and computation effort for training the network. Longer time series contain more information but also lead to larger networks.

b) Simulated Scenarios: To train the network with a supervised approach, we need correct class labels for all data points. To achieve this, we use scenarios to generate input data together with correct labels. Each scenario produces anomalous behavior that belongs to a certain anomaly class. To generate our data we use six different scenarios.

i. *Standard Scenario*: In the standard scenario the robot drives unhindered, i.e. we assume the absence of obstacles, that the robot does not cross any lines and that the neural network generating the steering and speed commands is connected to the robot for the time of the scenario. Therefore we generate randomized values in the appropriate ranges for the light sensors and the ultrasonic sensor detects no obstacles. Data points generated with this scenario are non-anomalies and show the system in normal operation.
ii. *Steering Scenario:* In the steering scenario the robot drives a zig-zag course, changing its steering angle with a constant rate from one extreme to the other and back. In this scenario, we keep the normal behavior of sensors, motors and Bluetooth connection, and only change the steering angle. Thus, we generate values such that the robot does not cross any line and detects no obstacles on the track. Data points generated from this scenario are anomalies of the *System Adaptation* class.
iii. *Hardware Failure Scenario:* In the hardware failure scenario some of the sensors the robot is equipped with break and report wrong values. To achieve this, we model a normal behavior up to a randomly determined point, which we call the break point. From this point on, one or more randomly determined sensors report wrong values. The robot then detects these wrong values and stops driving. Anomalies generated with this scenario fall under the *System Failure* class of anomalies, that describes anomalies due to the failure of the system.
iv. *Safety Feature Scenarios:* The robot has multiple safety

features. To generate anomalies that were caused by them, we use three different scenarios each geared towards one of the aspects of the safety features:

- Bluetooth failure: In this scenario, the robot receives commands from the neural network and data from the sensors as described in the standard scenario until a connection failure occurs. As soon as the robot notices the connection failure, it will stop driving.
- Passing the track border: In this scenario, the robot drives over one of the lines bordering the track. To train the network for many of the possible angles the distance in time between the left and right sensor detecting the line is randomized. When both sensors have crossed the track border the robot will stop.
- Obstacle: In the last scenario, the robot detects and approaches an obstacle in front of it. This is modeled by decreasing values of the distance sensor that depend on the speed of the robot. When the robot reaches a certain minimum distance to the obstacle it stops to prevent crashing.

These anomalies belong to the *Emergency Halt* class.

Except for the steering scenario, these scenarios represent anomalous behavior that can be observed in real executions of the robot. The steering scenario, however, describes artificial behavior that is highly unlikely to happen in real executions. We have chosen this scenario to generate several time series with anomalous changes in the steering angle.

B. Performance Evaluation

In this section, we evaluate the classification performance of different instances of our general network architecture using a 10-fold cross-validation experiment. Afterwards, we have a closer look at the accuracy over the different anomaly classes for the best performing network architecture. In the following, we first introduce our datasets and the different network architectures before presenting our experimental results.

1) Datasets: To test our networks we generate two datasets: one for training and one for testing. Each of these datasets consists of 43200 data points generated by our simulation environment. The datasets contains an equal amount of data for each class to avoid a bias of the overall accuracy towards one of the classes. Therefore each of the four classes (*No Anomaly*, *System Adaptation*, *Emergency Halt*, and *System Failure*) has exactly 10800 data points in each dataset. For each of the classes in a dataset we use all the possible scenarios in equal fractions. For the *System Adaptation* class that means that half of the 10800 data points are generated with the *Steering Scenario* and the other half with the *Standard Scenario*. The *Emergency Halt* and the *System Failure* classes also have equal fractions of all their associated scenarios.

2) Neural Network Architectures: We evaluate different neural network architectures that only differ in the size of the hidden layers. They have one input neuron for each time step for all variables, i.e. for our example we get $\#input\ variables \cdot \#time\ steps = 9 \cdot 15 = 135$ neurons. The recurrent input layer is followed by a dropout layer with a 10% dropout probability. After these, we employ a variable number of hidden layers with a variable number of neurons. These layers are fully connected, i.e. each neuron is connected to each neuron of the previous layer. We evaluate networks with h hidden layers, where $h \in \{1, 2, 3, 4, 5, 6\}$ and p neurons in each layer with $p \in \{15, 30, 45, 60, 75, 90\}$. The neurons use a hyperbolic tangent activation function. Each network architecture has 4 output neurons, one for each class, and use the *softmax* function for probability output values in the interval $[0, 1]$.

3) Experimental Results:

a) Performance of different Network Architectures: We have evaluated the described network architectures using a 10-fold cross validation. Each of the architectures achieve an accuracy of $\sim 99\%$. This seams to indicate that the GRU input layer is doing the heavy lifting and the size of the hidden layers does not matter.

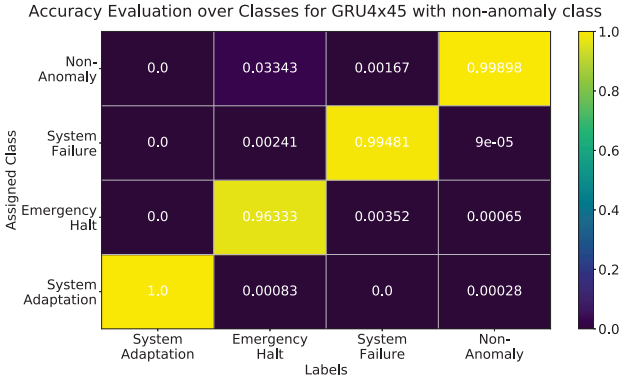

Fig. 5. Accuracy Evaluation over Classes for GRU4x45

b) Classification Accuracy over Classes: To evaluate the classification accuracy for each class, we have performed a more detailed analysis of the classification results on our separate test set that has not been used for training. The results for the network with 4 hidden layers of 45 Neurons each are depicted in Figure 5. We can see that the accuracy is very good for all classes (above 96%). We have $\sim 3.5\%$ false negatives, i.e. anomalies that were classified as non-anomalies, and hardly any false positives, i.e. non-anomalies that were classified as anomalies (below 0.1%). The false negatives mainly belong to the *Emergency Halt* class.

c) Explanations for Classifications: Our experimental results show that we can successfully detect and classify anomalous behavior such as sudden stops due to hardware failures or safety policies, or anomalous driving in a zigzag course by observing the system behavior and relevant context parameters. Our anomaly classes already enable us to provide an explanation for the observed behavior. For example, consider a situation, where the robot stops because it lost the Bluetooth connection. In this case, our approach identifies an *Emergency Halt* as reason for the sudden stop. Without any further evaluation of the input data, we can already generate the following explanation: *The robot has stopped because of an Emergency Halt. Possible reasons are: the robot has lost its Bluetooth*

connection to its visual guiding system, the robot has detected that it has crossed the track border, the robot has detected an obstacle in front of it. This is a very broad explanation and would not be very useful. However, it shows that we have already curtailed the possible reasons for the observed behavior. In a subsequent step, we can use this knowledge to identify the input parameters that have to be analyzed to identify the actual reason. In our example, such an analysis could check whether the observed values of relevant parameters, i.e. the Bluetooth connection status, the light sensors and the distance sensor, differ from predefined ranges of *normal* values. With that, we can identify the Bluetooth connection failure. To create the coarse explanations for each class, predefined sentences can be used. For more precise explanations, textual explanation patterns can be used and filled with the identified reasons.

In more complex scenarios, more sophisticated analyses are required. In future work, we aim to develop suitable analysis techniques for more complex case studies.

V. Related Work

Explainability of autonomous systems became a prominent research area due to research projects on *Explainable AI*, which focus on explaining machine learning results. Other approaches focus on *explainable planning* to explain decisions that are not based on ML but classical planning. Examples are [8], [9], [10], and [11]. Explanations are often provided in form of contrastive explanations that explain why an action A has been chosen instead of an action B. In contrast to our work, these approaches focus on how to generate explanations for planning decisions and neither provide a framework for identifying situations that need to be explained nor consider anomalous behavior due to hardware failures. In [2], the authors sketch first steps towards a conceptual framework for self-explaining CPS. They propose to add a layer for self-explanation that includes an abstract model of the system and to use this model to construct cause-effect chains for observable actions. Users can access these chains to understand the cause of actions. In contrast, we propose a model-free approach that is based on supervised learning and classification of anomalous behavior. Another model-based approach is presented in [12]. The authors propose to use a feedback loop to identify situations where user feedback would be valuable. They compare the user behavior with a goal model and ask for feedback when users achieve sub-goals or when they deviate from an expected sub-goal.

As we use sensor data to determine the reason for anomalous behavior, our approach is in line with research on diagnosability. In [13], an approach for assessing the degree of diagnosability, i.e. how many faults can be discriminated by the available set of sensors, is proposed. A high degree of diagnosability is a prerequisite for detailed explanations.

VI. Conclusion

We have presented our vision of a self-explainability component that autonomously detects and explains anomalous system behavior in order to enable a successful cooperation between humans and autonomous systems. As a first step towards this vision, we have present an approach for detecting and classifying anomalous behavior. By classifying the behavior into classes with similar reasons, we can already generate coarse explanations that can be connected to the classes. Furthermore, we can use the classes to reduce the huge search space of possible reasons for an anomalous behavior, enabling a subsequent analysis to identify this reason. We have shown the feasibility of our approach by classifying anomalous behavior of an autonomous robot simulation. In future work, we want to optimize our approach and evaluate it on real executions with fault injection. Classification with neural networks is not always the appropriate method. Thus, we want to investigate other diagnosis approaches and develop a guideline for choosing the right detection and classification methods. Furthermore, we aim to work on analysis techniques to generate precise explanations for classified behavior and techniques for generating individual explanations for different users. Furthermore, we would like to investigate the application of networks that are pre-trained with simulated data on a deployed system using techniques from transfer learning.

References

[1] M. Blumreiter, J. Greenyer, F. J. Chiyah Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, and A. Wortmann, "Towards self-explainable cyber-physical systems," in *22nd Int. MODELS-C, The 14th Int. Workshop on Models@run.time*, pp. 543–548, IEEE, Sep. 2019.

[2] R. Drechsler, C. Lüth, G. Fey, and T. Güneysu, "Towards self-explaining digital systems: A design methodology for the next generation," in *3rd Int. Verification and Security Workshop (IVSW)*, pp. 1–6, IEEE, 2018.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[5] S. Barrett, M. E. Taylor, and P. Stone, "Transfer learning for reinforcement learning on a physical robot," in *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, vol. 1, 2010.

[6] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. A. Malikopoulos, and A. Bayen, "Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles," in *10th ACM/IEEE International Conference on Cyber-Physical Systems*, pp. 291–300, 2019.

[7] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[8] X. Fan, "On generating explainable plans with assumption-based argumentation," in *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 344–361, Springer, 2018.

[9] E. Zhao and R. Sukkerd, "Interactive explanation for planning-based systems," in *10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2019)*, 2019.

[10] R. Sukkerd, R. Simmons, and D. Garlan, "Towards explainable multi-objective probabilistic planning," in *4th Int. Workshop on Software Engineering for Smart Cyber-Physical Systems*, pp. 19–25, ACM, 2018.

[11] S. Sreedharan, S. Srivastava, D. Smith, and S. Kambhampati, "Why can't you do that hal? explaining unsolvability of planning tasks," in *International Joint Conference on Artificial Intelligence*, 2019.

[12] D. Wüest, F. Fotrousi, and S. Fricker, "Combining monitoring and autonomous feedback requests to elicit actionable knowledge of system use," in *Requirements Engineering: Foundation for Software Quality* (E. Knauss and M. Goedicke, eds.), pp. 209–225, Springer, 2019.

[13] L. Trave-Massuyes, T. Escobet, and X. Olive, "Diagnosability analysis based on component-supported analytical redundancy relations," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 36, no. 6, pp. 1146–1160, 2006.