Learning Based Anomaly Detection for Industrial Arm Applications

Vedanth Narayanan[1] Rakesh Bobba[2]

***Abstract*— Smart Manufacturing (SM) is envisioned to make manufacturing processes more efficient through automation and integration of networked information systems. Robotic arms are integral to this vision. However the benefits of SM, enabled by automation and networking, also come with cyber risks.**

In this work, we propose an anomaly detection framework for robotic arms in a manufacturing pipeline and integrate it into Robot Operating System (ROS), a middleware framework whose variants are being considered for deployment in industrial environments for flexible automation. In particular, we explore whether the repetitive behavior of an industrial arm can be leveraged to detect anomalous behaviour that may indicate an intrusion. Based on a learned model, we classify a robots actions as anomalous or benign. We introduce the notion of a *tolerance envelope* to train a supervised learning model. Our empirical evaluation shows that anomalies that take the robot out of pre-determined tolerance levels can be detected with high accuracy.

I. Introduction

Smart Manufacturing (SM) is an emerging concept where the goal is to integrate and optimize the different stages of a manufacturing pipeline by leveraging advanced manufacturing and information technologies such as robotics, data analytics and Internet-based connectivity [14]. This allows manufacturing systems to be programmable, efficient, precise, and require less human intervention [13]. In short, SM enables flexible manufacturing pipelines that are able to adapt quickly and have less downtime and is therefore gaining traction in the industry. For example, KUKA Systems worked with Microsoft to create an automotive factory that primarily runs with automated robotic arms [24].

Robotic arms play an important role in automation of manufacturing and in SM in general. And the flexibility offered by such systems is in part due to the software that is used to run them. Robot Operating System (ROS) [30] is an open source middleware for robots that runs on Unix platforms. ROS-Industrial (ROS-I) extends ROS for the manufacturing sector [12]. Industrial robot vendors are supporting ROS-I by starting to offer their own software stacks that integrate with ROS-I (e.g., KUKA, Motoman, ABB) [32].

The integration of physical manufacturing systems (e.g., robots, manufacturing pipelines) with network enabled computer control and other information technologies leads to a complex cyber-physical system (CPS) and bring cyber security risks. Attacks on additive manufacturing (AM), or 3D printing processes [37] that can introduce visually undetectable modifications, and attacks on industrial robot controllers [29] that are capable of affecting a whole production line have been discussed in the literature. Belikovetsky et al. presented an in-depth cyber-physical sabotage attack on a 3D printed process that accelerated the fatigue life of a quadcopter propeller [4]. It demonstrates how a sabotaged, yet functioning propeller breaks mid-flight after a short period of operation. Zeltmann et. al also show a unique risk in AM, where the printing orientation is modified to produce parts with less performance potential [43]. Although these examples target AM processes, similar attacks can be envisioned for other SM environments. Pan et al. present a taxonomy classifying cyber-physical attacks in manufacturing processes [28]. Real-world cyber attacks on cyber-physical infrastructure such as the Stuxnet [16] attack, and Ukranian power grid attacks [10] show that the cyber security threats to cyber-physical infrastructure are real and that even air-gapped systems are not immune.

Unlike traditional enterprise computing systems and networks, cyber-physical systems tend to exhibit well-defined patterns and behaviors. Previous work in SCADA systems have shown that probabilistic, Markov model based solutions, and machine learning techniques to be effective in these settings (e.g., [3], [42], [21]). With that in mind, in this work we propose an application level learning-based anomaly detection framework for detecting attacks that aim to modify the manufactured product while avoiding detection (i.e., stealthy attacks). Attacks that aim to destroy the manufacturing line or otherwise create obvious changes to the product or the manufacturing process are out of scope. In particular we focus on robotic arm based manufacturing pipelines where the robots perform well-defined repetitive tasks. We hypothesize that any impactful change to the manufactured product will necessarily be evident through changes in the motion of the robotic arm. The guiding questions in this work are, i) can the repetitive behavior of an industrial arm support detection of anomalies, ii) can such anomalies be detected by monitoring the motion of the robotic arm, and iii) how small a deviation in the manufactured product can be detected using this approach?

The expected behavior of the robotic arm is learned during the commissioning phase. And to overcome the challenge of creating anomalous executions to train the model we introduce the notion of a *tolerance envelope* around the phys-

[1]Master's student in the Department of Electrical Engineering and Computer Science, Oregon State University narayave@oregonstate.edu

[2]Major Advisor in the Department of Electrical Engineering and Computer Science, Oregon State University rakesh.bobba@oregonstate.edu

ical product being manufactured. Our preliminary empirical evaluation using a Yaskawa Motomoan MH5 robotic arm shows that our approach can detect even minute alterations to the robot's predetermined motions during the production phase.

In summary, our contributions are:

- We introduce a learning-based anomaly detection system for manufacturing robotic arms.
- We introduce the notion of a *tolerance envelope* for training the anomaly detection model.
- We present the application level learning-based anomaly detector that we call the ROS Anomaly Detection Module (ADM).
- We integrate the anomaly detection scheme into ROS middleware framework and evaluate it using a real robotic arm, namely, Yaskawa Motoman MH5.

II. Background

A. System Overview

The network architecture of a smart manufacturing environment closely follows the Purdue Reference Model [27]. The Enterprise Network and Industrial Control Systems (ICS) Network are indirectly connected by a DMZ layer, allowing for controlled data sharing. Traditionally, Enterprise networks were independent from the ICS networks, but this has changed to realize SM's vision of integrating and optimizing the different stages of the manufacturing product life cycle. Figure 1 presents the high level architecture. This new connected architecture enables an integrated manufacturing life cycle, along with ease of programming, and maintenance on the factory floor [11].

Figure 3 presents a simplified version of our architecture, and the main components in this work. For our work, we consider our control system to be an Open-Control loop system. Once a task is programmed, before any motion commands are sent to the arm, the entire trajectory of the arm is planned. The arm motions are produced once a path, avoiding preset obstacles, has been determined. The state of the arm is relayed back to the system, however this is only used for visualizing the arm motion on the simulator.

B. ROS Overview

The Robot Operating System (ROS) [30] is a popular open source, middleware for building robot applications. ROS has been used to build mobile robots, UAVs, hand and arm manipulators, and even industrial arms [1]. The operating system functionalities that it provides includes hardware abstraction, low-level device control, a message-passing interface, and package management [39]. ROS provides a range of tools and capabilities that make it a suitable platform for collaborative robotics software development. ROS has even been extended by ROS-Industrial (ROS-I) the manufacturing sector [12].

The different concepts within the ROS are as follows [34]. A single ROS system consists of multiple computational processes, known as nodes, which make up the "graph." Any node wanting to join the ROS network registers itself with the ROS Master. ROS Master also helps locate nodes for communication. ROS nodes communicate over communication channels, known as topics. When communicating, nodes take up either the publisher or subscriber roles. This pattern is known as a publish/subscribe pattern. This pattern was a conscious choice that allows for loose coupling of processes.

There are three other components that complete the ROS architecture that we work with, and they are ROS-Industrial (ROS-I) stack, the MoveIt! framework, and the small controller stack offered and maintained by the vendors of the industrial arm. ROS-I packages offer capabilities for building ROS based applications for industrial systems. When a task is programmed, ROS-I relies on the MoveIt! framework for kinematics and path planning operations. This is accomplished by using industrial arm models provided by the vendor. In addition, the vendor also supplies configuration files and specific ROS libraries that help users interface with their robot controllers and arms. ROS-I passes ROS joint values to vendor libraries which translate them to actuation commands appropriate for the arm. The high level ROS-I architecture is shown in Figure 6.

C. ROS Package File Structure

ROS packages follow a common structure. The following list briefly describes each component.

- **launch/**: Launch files are commonly used by developers and users alike to launch the ROS Master and multiple nodes (likely from different packages) at once. Launch files also help with setting ROS parameters at runtime.

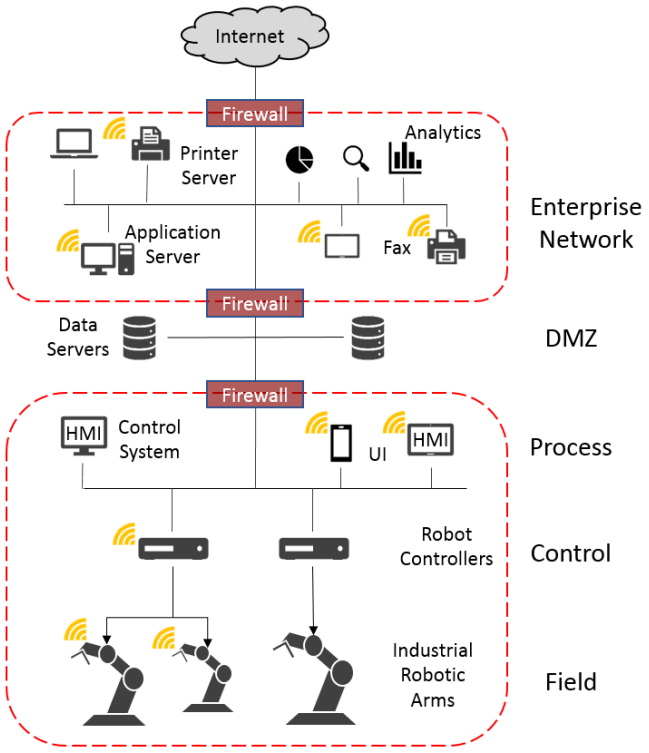


Fig. 1: Smart Manufacturing control network architecture based on the Purdue Reference Model.

Launch files are written in an XML format, and the content is placed within the <launch> tag. The <node> tag is used for starting a node, along with the pkg, name, and type arguments. The first argument specifies the package. The 'name' refers to the name of the node. The 'type' argument specifies the executable file. The <arg> tag supports use of variables. Lastly, the file argument within <include> tag assists in importing and launching another roslaunch file within the current scope. Further details can be found in [35], and examples of roslaunch syntax are presented in upcoming sections.

- **scripts/**: Executable scripts placed are placed in this directory.
- **src/<package name>**: Source files for Python libraries are placed here. Any script can import classes from these libraries.
- **package.xml**: This is a package manifest XML-style file that exists in the root of all catkin[1] packages. It is typically generated when a package is created by catkin. The only information the developers need to update are the package descriptions, and details about the package maintainers.
- **CMakeLists.txt**: This is a CMake build file, used by the catkin build system. This is also automatically generated when a catkin project is created.
- **setup.py**: The setup file is placed at the root directory of a package. The modules in the src/ directory can be built when building the entire ROS workspace (specific catkin workspace consisting of ROS package sources).

D. Machine Learning Models

There are two Machine Learning models that we employ in our design. Traditional Support Vector Machines (SVM), and a One-Class SVM for classifying robotic arm executions as benign or anomalous. Note that there exist other classifications methods such as Decision trees, Neural networks, Bayesian networks, and even clustering methods. The advantages of SVMs are explained further on.

Support Vector Machines (SVM): An SVM is a supervised algorithm used for classification problems. From a high level, an SVM transforms input vectors to a higher dimension to be able to linearly classify and assign labels to them [5]. The training data consists of input vector $x_0, ...x_n$, along with the associated output labels, $y_0, ..., y_n \epsilon \{0, 1\}$. In this work, SVMs will be used as a binary classifier, however they are capable of classifying data into multiple classes. Each vector consists of m features. The training data used in this work will be elaborated on in Section V.

An SVM's goal is separating sets of data by drawing a boundary between them. The data points close to the boundary (of any classes) are referred to as the *support vectors*. SVMs aim to draw a boundary that maximizes the distance from the support vectors. By using a small set of data to define the boundary, the model is still generalizable for a larger class of data. This advantage serves as the premise in Section V-C. Figure 2 presents examples of both linear and non-linear SVMs. We pursue a non-linear SVM in this work to better fit our datasets and goals. Specifically, a non-linear SVM allows for building tight boundaries around our datasets. This is better understood in later sections.

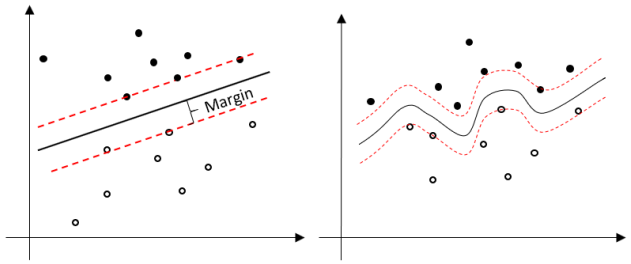


Fig. 2: A Linear SVM example is on the left. A non-linear SVM example is shown to the right. The data points on the margin lines are referred to as the Support Vectors.

One-Class Support Vector Machine (OCSVM): The OCSVM is a special case of the binary SVM, where only one class data is primarily used for training. The distribution of the dataset is learned, and a rough boundary is drawn around it [36]. The data points encountered in the testing phase are checked to see whether they come from the same data distribution as the data encountered during training. Data points are classified as benign if they are indeed from the same distribution. They are classified as anomalous if they are considered to come from a different distribution. The challenge for OCSVM is defining a tight enough boundary around the known class, such that acceptance of outliers is minimized, while maximizing acceptance of true benign samples.

E. Principle Component Analysis (PCA)

Principle Component Analysis (PCA) is well-known technique used for dimensionality reduction. Given a multivariate dataset, PCA can represent it an orthogonal basis of a lower dimension, while holding true to important details of the dataset [2]. In other words, PCA is often used to represent the variation in the original dataset in fewer dimensions, which allows for easier pattern recognition.

From a set of components, the first principle component will inherently carry the largest variance, and the following components will carry variances of smaller amounts. Note that, a certain amount of variance will be lost in the process, albeit it is very small. The majority of the relevant variance will be captured by the initial components.

III. Threat Model

a) Attack Goals: For this work an attacker's goal is to launch an attack that sabotages the manufactured product without being detected. That is, the attacker aims to induce changes to the manufactured product that are subtle and hard to detect yet adversely impact the part's quality and or performance (e.g., attack in [4]). Attacks that aim to destroy

[1]The official ROS build system

the physical system (e.g., Stuxnet attack), or destroy cyber assets (e.g., Ukraine power grid attacks), or otherwise cause obvious defects in manufactured products are out of scope. While our proposed anomaly detection may be able to detect some of the latter attacks oftentimes such attacks don't need sophisticated learning-based approaches to detect them after the fact.

b) Potential Attack Vectors: As seen in Figure 3 an industrial arm is typically controlled using a robot controller that mediates between the robotic arm and the high-level manufacturing logic. An attacker may be able to compromise the control logic after gaining entry into the control system via the control network which may be indirectly connected to the Internet (see Figure 1). In this scenario the attacker needs to understand only the manufacturing logic and need not understand the model or dynamics of the robotic arm. A more sophisticated attacker may also compromise the robot controller to cover their tracks and will need to understand the robotic arm model in addition to the manufacturing logic. In the following we focus on the former attacker and will discuss how to deal with the latter attack scenario with a more sophisticated attack in Section IX.

c) Attacker Goal: The attacker's objective is sabotaging the manufactured product, such that it ultimately fails during end user use. To successfully continue sabotaging, the attacker also aims to evade detection. There are primarily two types of sabotage attacks. The first type of sabotage attack is forcing production of unsatisfactory parts that are simply not functional, or have obvious defects. A second attack, more meticulous in nature, alters the product so it is still functional, but develops quickly develops fatigue. This is what we consider to be a stealthy attack. We assume the former sabotaged products to be intercepted by the QC process, and is not the attacker's focus. However, the latter attack can help accomplish both of the attacker's goals, and has already been demonstrated in prior work [4]. Note that although we focus on the second class of sabotage attacks, we do not distinguish between the attacks when detecting anomalies.

IV. Anomaly Detection System (ADS) Overview

A. Assumptions

Our anomaly detection approach is built on the following primary assumptions. First, industrial arms on a manufacturing assembly line are programmed to execute well-defined repetitive tasks. While a multi-jointed arm is capable of accomplishing a given task following different arm trajectories, typically, a program uses the same arm trajectory unless, unless the task has been perturbed. For our approach, it is critical that the end effector follows the same trajectory repeatedly when performing a given task. It is this consistent behavior that helps with building a white list, and allows for actions not part of the list to result in getting marked anomalous.

Second, we hypothesize that any changes that an adversary wants to induce in the manufactured product will manifest as a change in the motion of the robotic arm[2]. The question we explore in this work is how small a change in the manufactured product can be detected using the change in the robotic arm motion?

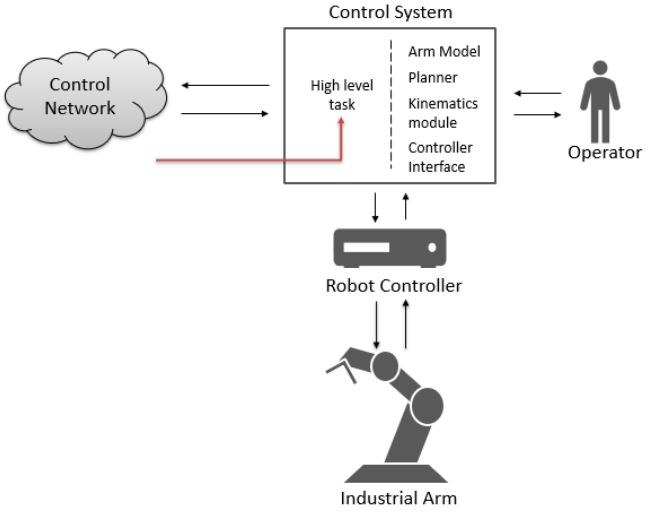


Fig. 3: An attacker originating from the control network targets the high-level task logic in the control system, as depicted with the red arrow. The items to the right of the dashed line may not be directly compromised.

Third, in the process of deploying a robotic arm on the factory floor, we assume that there is an initial commissioning phase. The commissioning phase is when an operator establishes the high level task, and iteratively refines it so the manufactured product aligns with its product specification. Additionally, we assume the commissioning phase occurs in a *secure* and controlled environment where no adversaries are present, and outside interferences or variables do not affect the task establishing process. The robotic arm is not compromised in this phase. These assumptions are not held during the operational phase of the arm, that occurs after the commissioning phase.

It is common for assembly lines to be occupied by multiple industrial arms, often programmed to work collaboratively on a single product. In this work, we consider a single industrial arm. However, the ideas presented here can be extended to scenarios with more than a single industrial arm.

B. Challenges

In employing a learning based anomaly detection approach there are three related challenges that need to be solved.

a) Challenge 1: The first challenge for any machine learning algorithm is selecting the right dataset to work with. This is no different for our work. The dataset needed for accomplishing our goal should describe the arm's motions when executing a task. For a successful implementation, a continuous stream of the arm's motions is also required. This

[2]We focus on robotic arms here but this applies to other manufacturing equipment e.g., additive manufacturing equipment such as 3D printers.

assists in tracking trajectories at a steady rate. The selected features also pose an important aspect of the dataset.

b) Challenge 2: The second challenge is selecting an appropriate learning algorithm that offers the best results in detecting changes in the arm's trajectory. The typical choices for learning methods are unsupervised or supervised. An unsupervised learning approach has the benefit of not requiring labeled data. Since there can be potentially innumerable ways of sabotaging a manufactured product and corresponding deviations in robotic arm motion, it is impractical to gather labeled data for all anomalous behavior. Therefore the better option here would be unsupervised learning method. However, supervised methods are known to do really well with categorical classification and we would like to explore supervised methods as well.

c) Challenge 3: As discussed previously, in order to explore any supervised methods, we need labeled data. That is, not only should the dataset contain benign data which is easy to obtain, but also labeled anomalous data. While one can generate labeled anomalous data by running sample anomalous executions, the challenge is in ensuring that it is representative of attack executions that an adversary might launch.

V. ADS Design

A. Learning Dataset Selection

Typical mobile robot possess various components and sensors, including LIDAR systems, compass bearings, GPS or IPS systems, and more. Data from these components can be leveraged for building a dataset to learn their motion or trajectories. There is not an array of options when it comes to stationary, industrial robotic arms. An industrial arm does in fact have internal sensors associated with its joints. In order to address challenge 1, we build a dataset of the joint states of the industrial arm during the execution of the programmed task. The sensor readings from the physical processes are translated to joint states and relayed back to the control system. The features in our dataset are the individual joint values from a single joint states reading. In this work, we will focus only on joint state values. Approaches that leverage other data and the associated challenges will be revisited in future works, as discussed in Section XI.

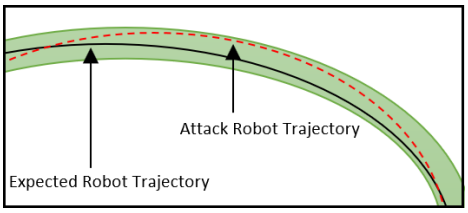
B. Learning Model Selection

To reiterate, one of the challenges of building a classifier is being unable to foresee an attacker's modified task and resulting arm trajectories so a model can be trained. However, learning the benign task's movements is feasible. Intuitively, we can learn the benign trajectory and classify everything else as anomalous, that is, a whitelisting approach. The OCSVM (discussed in Section II-D) lends itself to novelty detection, and aligns well with the whitelisting approach.

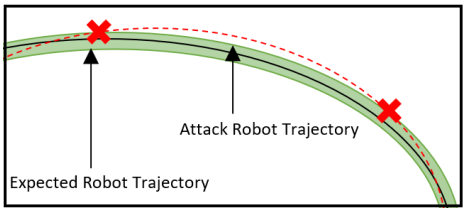
The OCSVM learns the data distribution of the expected data points from the dataset that it is passed. Accordingly, the dataset we supply for training we for OCSVM encompasses the joint state values of the arm during *benign* task executions. We also built validation and testing datasets, that include some anomalous runs, to verify the accuracy of the learned model. We noticed that anomalous runs with large deviations were rightfully getting classified as anomalies. However, results deteriorated with anomalous runs that had smaller deviations. That is, the boundaries learned by the OCSVM for the benign task were not tight enough. This led us to training a traditional SVM as discussed next.

The supervised learning model that we chose to train was RBFSVM (discussed in Section II-D), which is an SVM classifier using the RBF kernel. This classifier was chosen to emulate what the OCSVM does, but with labeled data. As was previously discussed, it is challenging to supply a representative dataset of all possible anomalous examples. Thus, our training data includes the benign data, and select anomalous samples produced by our "tolerance envelope" approach that is discussed in Section V-C.

The original data has n features, corresponding to the robotic arm's n joints. The arm can be in a large number of states when considering the numerical range that each joint can lie in. However the possible states the arm is in during the execution of the programmed task is relatively small. Classification might be harder due to the sparsity of our dataset in the joint space. To address this, we explore dimensionality reduction using PCA, as part of the data prepocessing step that occurs before supplying the data to the learning models for training. For a single learning model, we experiment with transformed datasets consisting of $n-1$ to 2 principle components as features. This ultimately helps decide how many dimensions should be reduced to produce optimal results.



(a) Large trajectory boundary



(b) Small trajectory boundary

Fig. 4: The black lines in the green region are the trajectories of benign runs. The green regions symbolize tolerance boundaries for the task. 4a presents a more lenient boundary than in 4b. The red, anomalous trajectories should be marked in both situations, but this only occurs in 4b.

C. Anomalous Data Generation using Tolerance Envelope

For supervised learning all collected data points need a benign or anomalous label. Collecting benign data is straightforward. It involves executing the benign task multiple times and labeling the collected data points as benign. Collecting representative anomalous data is challenging however as noted in the previous section. For successful use of a supervised model, it would be best if all possible anomalous situations are represented in the training data. However, it is not practical to collect data points for all possible anomalous executions.

We address this challenge by taking advantage of the physical aspect of the system. In particular, we define a *tolerance envelope* around the physical product being manufactured. While this notion can be applied to both additive and subtractive manufacturing, in this paper we use a subtractive example of metal cutting. The data points within the *tolerance envelope* are meant to be classified as benign, and everything outside the envelope as anomalous. The tolerance envelope will vary from product to product and should be defined carefully as this will impact the false positive and false negative rates of the anomaly detection (see Figure 4).

We define the tolerance envelope by taking into account the tolerance limits of the specific product. For instance, suppose that for the example we use in this paper of cutting a triangular metal piece, the physical tolerance limit is $2mm$. That is, the triangular pieces cannot be more or less by $2mm$ than the specified size. We then create executions of the robotic arm that produce triangular pieces that is uniformly $2mm$ smaller, and uniformly $2mm$ larger than the specified size and label the collected data as anomalous (see Figure 11b).

An abstract illustration of the envelope created using anomalous tasks at the edge of physical tolerance limits is shown in Figure 5. The intuition is that, no matter the trajectory that an adversary's modifications induce, we want to raise an alarm if the trajectory of the robot might mean the product will be outside of physical tolerance limits. Consequently, this also means that if an adversary's actions do not cause the product to go out of physical tolerance limits then such actions may remain undetected. But this is a trade-off we make in our design.

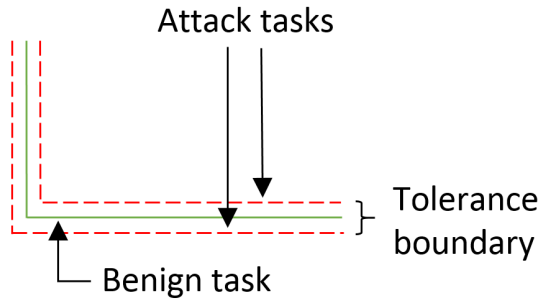


Fig. 5: This abstract illustration shows how an envelope around the benign task is created. This involves introducing attack tasks that produce a product just outside of the expected tolerance limits.

VI. ADS Workflow

To evaluate our anomaly detection methodology we considered an industrial arm programmed and controlled with ROS. We developed the ROS Anomaly Detector Module (ADM)[3], designed to execute alongside industrial robotic arm tasks to detect unintended deviations at the application level. The process has been made efficient by building the ADM as a ROS package that aptly fits in the ROS ecosystem. Figure 6 presents the ROS-I architecture. The red box around ROS ADM in the ROS Layer shows where the ADM exists within the ecosystem.

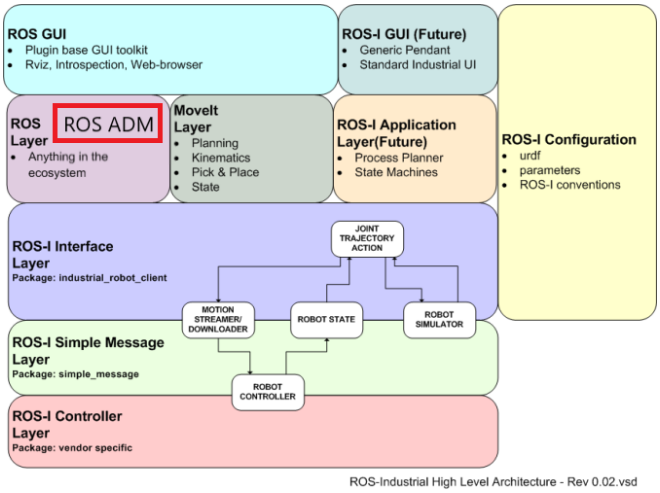


Fig. 6: ROS Industrial Architecture [33] showing where the ROS ADM fits in the ecosystem.

The crux of anomaly detection within this module relies on a three step process. The steps includes creating datasets out of the messages published within ROS, training learning models from those datasets, and deploying it in production. Appropriately, the three modes that the ROS ADM can be executed in are Collect, Learn, and Operate. Figure 7 presents the intended workflow of the ADM at a high level. The Collect and Learn modes of the ADM run during the commissioning phase of the robot.

It is convenient to expect the Collect and Learn modes to run consecutively, and as a single unit. However, the separation of these was a conscious design decision. The ADM is semi-automated, and human intervention is expected between executions of modes to build appropriate learning models for anomaly detection. The details are elaborated on in the following sections.

To start the ROS ADM, the 'module.launch' file needs to be included in the project's launch files. Code listing 1 presents the ADM's 'module.launch' file. Note that the three portions of the code is associated with the three modes that it can be run in. To initiate a mode, the 'mode_arg' tag should

[3]https://github.com/narayave/mh5_anomaly_detector

be supplied with one of the following values: '-collect,' '-learn,' or '-operate.' The other required arguments for each mode are presented further on.

A. Collect Mode

The ADM is first run in the Collect mode after the task has been programmed and assessed in the commissioning phase of the robot. This mode runs as a node by subscribing to the '/joint_states' topic, of message type 'sensor_msgs/JointState,' within ROS. The message fields corresponding to this type of a message are presented in code listing 2. The Header field is common to all ROS messages, consisting of a sequence ID, time stamp, and a frame ID. The name array identifies the unique joints of a robot. Accordingly, the data in other arrays correspond to the identified joints.

```
# sensor_msgs/JointState
  Header header
  string[] name
  float64[] position
  float64[] velocity
  float64[] effort
```

Code Listing 2: The sensor_msgs/JointState data type holds messages describing the state of a set of torque controlled joints.

The data that is collected in this mode is the position array. There are other available message fields on top of the position field, however the caveat is that they need to be populated by the publisher, which in this case are vendor ROS nodes. For example, the vendor of the MH5 industrial robotic arm that we use later on is Motoman. Motoman publisher nodes populate the position array, but not the velocity and effort arrays. Thus, the position array is the only data field we collect. These positions will be the input for the learning models, and need appropriate output labels. When running a benign task (the intended task), the output supplied to the mode is a benign label. Appropriately, an anomaly label is awarded when running an anomalous task. This is one again achieved by employing a "tolerance envelope" around the product that is to be manufactured.

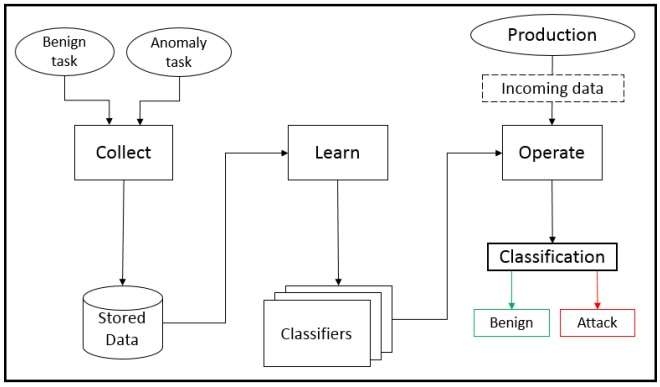


Fig. 7: The intended workflow of the ROS ADM follows collection of data, generating learning based models, and using learned models during production.

The data (input array and the output label) is outputted to a comma-separated value (CSV) file[4], which is a common practice when working with data for machine learning techniques. In the process of creating datasets for learning models, there is some minor manual tasks that the user needs to complete. The user defines the number of cycles a task will run. Typical datasets, particularly for supervised learning models, consists of both classes (benign and anomalous) of data in the same dataset. However, the ADM is capable of only awarding a single output label when running a task. Thus, the outputted CSVs for benign tasks and anomalous tasks are independent. The separated CSVs need to be concatenated together to create a single dataset that can be used for learning. This step has not been automated, and is accomplished manually.

Code listing 3 presents what is included in a roslaunch include tag to run the ADM in the Collect mode. The 'module' launch file is included for running the ADM. The 'mode_arg' specifies the mode that the user wants to run the ADM in. The 'other_args' argument tag first specifies the topic, which is followed by the path (including filename) of the output file. And lastly the output label awarded for the collected data is specified.

B. Learn Mode

The goal of the Learn mode is submitting the user with trained classifiers. In order to accomplish this the user is depended on for not only supplying the training datasets, but also the classifiers of interest. These are supplied to an internal library via a script. The internal library semi-automates the learning process. Once the script is supplied, the internal library pre-processes the data (reduces dimensionality if specified), and the set of classifiers are all trained with their appropriate datasets. After training, the classifiers are saved in storage. An output matrix is presented to the user consisting of the following values: True Positive, False Positive, True Negative, False Negative, Accuracy, Recall, Precision, and an F1 score. The specifics of this are explained in Section VIII. These results are for assisting the user in selecting the classifier that is most optimal for the Operate mode. Figure 8 shows the design of the different components of the library, and how they interact with each other. Each component is further dissected in the Section VII.

In the process of training classifiers, tuning the associated hyperparameters is an important step. Hyperparameter tuning is the procedure of finding the appropriate combination of parameters that allows the classifiers to return the most optimal results. This combination is not known beforehand, and the user can find it by iteratively performing experiments to determine it. Specifically, the user instantiates a set of classifiers each with different combinations of parameters, and they are all trained on the same training set, and the results are compared. An example of initiating the classifiers

[4]https://tools.ietf.org/html/rfc4180

```
<launch>
  <arg name="mode_arg" />
  <arg name="other_args" />
  <arg name="file_name" />
  <arg name="pkg_name" />

  <group if="$(eval arg('mode_arg') == '-collect')">
    <node name="data_collector" pkg="ros_anomaly_detector" type="data_collect.py"
      output="screen" cwd='node' args="$(arg other_args)" />
  </group>

  <group if="$(eval arg('mode_arg') == '-train')">
    <node name="train" pkg='$(arg pkg_name)' type='$(arg other_args)'
      args="$(arg file_name)" cwd='node' output="screen" />
  </group>

  <group if="$(eval arg('mode_arg') == '-operate')">
    <node name="monitor" pkg="ros_anomaly_detector" type="monitor.py"
      output="screen" cwd='node' args="$(arg other_args)" />
  </group>
</launch>
```

Code Listing 1: The ADM's module.launch file that is to be included to other launch files.

```
<include file="$(find ros_anomaly_detector)/launch/module.launch">
  <arg name="mode_arg" value="-collect" />
  <arg name="other_args" value="'/joint_states'
     '$(find ros_anomaly_detector)/data/test.csv' 1" />
</include>
```

Code Listing 3: Initiating the ADM to run in the Collect mode.

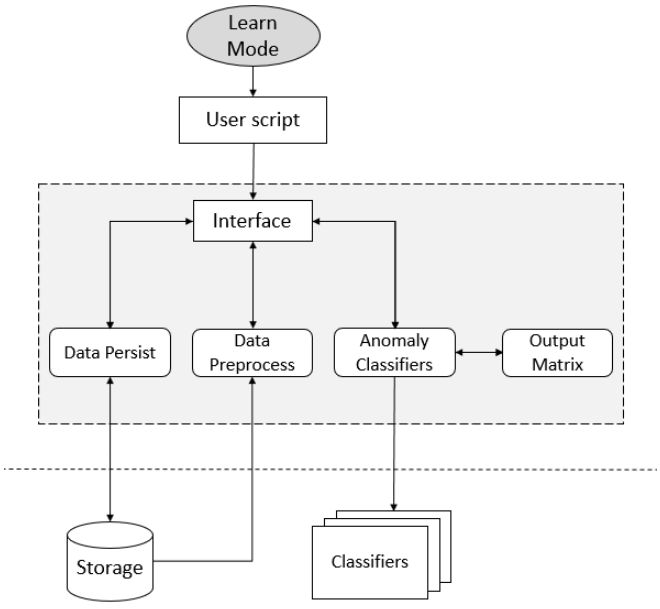


Fig. 8: The high level architecture of the Learning Library.

can be observed in code listing 6's step 2. The methods for hyperparameter selection is out of scope of this report, although a common method of accomplishing this is running a Grid Search. This exhaustive search method involves selecting a wide range of potential parameter values for all parameters, and validating all combinations of them. These parameters need to be further fine-tuned after selection.

Parameter selection effectively helps determine the appropriate classifier for the given task. Thus far the datasets we used had output labels, but this won't be the case moving forward. When running the ADM in Operate Mode, the goal is to raise an alarm when a certain number of joint values are classified as anomalous as part of a task. We define an anomaly score as the ratio of data points classified as anomalous out of all the points associated with a task. To set the threshold value, we test against new representative anomaly tasks that the user builds. As a guideline, these new anomaly tasks should affect the shape, size, and orientation of the product. The lowest anomaly score of these tasks will be the threshold, so everything beyond it will be considered anomalous. Another thing to keep in mind is that benign data points can also be classified anomalous (false positives), and the threshold should be larger than the amount of false

positives that is observed.

An important distinction from the other two modes is that the Learn mode is independent of ROS. It's possible for the classifiers to be trained off-line and still be used during operation. However, there are two reasons for this design choice. The first is in trying to keep a consistent location for retrieval and storage of data, which is achieved when running the Learn mode as part of the ADM. The second reason is that it is possible to enhance this mode and make it ROS dependent in the future, perhaps even support online learning[5].

Code listing 4 shows how to the execute the ADM in the Learn mode, and supply the necessary scripts and data. The python script is supplied for the 'other_args' arguments. The two new arguments, 'pkg_name' and 'file_name,' are also required when initiating the ADM in the Learn mode. The first argument helps specify the package that the python script exists in. The second file specifies the path for the input file for the script. The details of use are explained in the following section.

C. Operate Mode

The final mode that the ADM runs in is the Operate mode. The classifier that was previously determined the best and the preprocessing models (for data normalization and PCA reduction) are put into effect as the robot is deployed to execute its task. It is worth noting that the task the robot executes is the same one that was initially programmed in the commissioning phase. Similar to the Collect mode, the Operate mode runs as a node and subscribes to the '/joint_states' topic. The incoming data is appropriately preprocessed and fed to the determined classifier for a prediction. The user is warned of anomalous data points, and is encouraged to take action when the anomaly score of the task surpasses the set threshold.

Code listing 5 shows the method of invoking the ADM in the Operate mode. The 'other_args' argument requires the user to specify the topic to subscribe to, the name of the stored classifier, and the threshold that should not be passed.

VII. Learning Library Implementation

The Learning Library works on top of the Scikit-learn machine learning library. Scikit offers a variety of classifier types and capabilities. The Learning Library works as a wrapper around the Scikit library, and helps abstract away the complexity of training and building classifiers. In addition, the Learning Library's objective is to also help automate the learning process as much as possible. Internally, it is modularly designed so users can efficiently modify and utilize parts of the source code based on the project and its goals. Figure 8 shows the architecture of the Library.

The Learning Library consists of five classes, or components, that help with its functionality. The five classes are Interface, DataProcess, DataPersist, AnomalyDetector, and the OutputMatrix. The Interface class is similar to a software API, which is what the user primarily interacts with. The Interface further manages multi-class operations. The DataProcess class assists in preprocessing datasets before they are used for training. The preprocessed data can be stored when passed to the DataPersist class. When necessary, the saved data can be retrieved. The AnomalyDetector class operates with the classifiers to have them trained and return predictions for different datasets. Lastly, the OutputMatrix presents evaluation metrics to the user, allowing them choose an optimal classifier to be used during production.

At a high level, the user's responsibilities includes writing a simple script, and supplying the names of datasets in a structured manner. From a small set of functions calls to the Interface class, the classifiers are quickly trained and are efficiently prepared for deployment. There are four key, programmatic steps to accomplish in order to get training predictions and its associated accuracies. First, the Interface class needs to be instantiated by passing the YAML[6] input file as an argument. YAML is a human-friendly data serialization method, that is interoperable between programming languages and systems. Next, the user creates instances of the classifiers they want trained. The user then passes off the classifiers to the Interface class for training. Lastly, testing predictions can be gathered for all classifiers. Code listing 6 presents an example of a script utilizing the learning library. The rest of this section goes over the details of different classes in the learning library.

```
if __name__ =="__main__":
  unsupervised_models =[]
  supervised_models =[]

  # Step 1
  input_file =sys.argv[1] # YAML input
  interface =Interface(input_file)

  # Step 2
  ocsvm =svm.OneClassSVM(nu=0.5, kernel="rbf",
     gamma=1000)
  unsupervised_models.append(('ocsvm', clf_ocsvm))

  svm1 =svm.SVC(kernel='rbf', gamma=5, C=10)
  supervised_models.append(('rbfsvm1', svm1))
  svm2 =svm.SVC(kernel='rbf', gamma=50, C=100)
  supervised_models.append(('rbfsvm2', svm2))

  # Step 3
  interface.genmodel_train(unsupervised_models,
     supervised_models)

  # Step 4
  unsup, sup =interface.get_testing_predictions()
```

Code Listing 6: 'Example of script'

A. YAML Input

The ".yaml" extension determines a YAML input file. The YAML file the user presents provides relevant filenames of input data in a structured manner allowing for easy parsing. Code listing 7 presents a template with the required syntax

[5]A classifier continues to be trained even after the initial training phase, allowing the training model to adapt to it data.

[6]http://yaml.org/

```
<include file="$(find ros_anomaly_detector)/launch/module.launch">
  <arg name="mode_arg" value="-train" />
  <arg name="other_args" value="run.py" />
  <arg name="pkg_name" value="pkg" />
  <arg name="file_name" value="'$(find pkg)/scripts/learning_script.yaml'" />
</include>
```

Code Listing 4: Initiating the ADM to run in the Learn mode.

```
<include file="$(find ros_anomaly_detector)/launch/module.launch">
  <arg name="mode_arg" value="-operate" />
  <arg name="other_args" value="`/joint_states' chosen_clf.pkl 0.02" />
</include>
```

Code Listing 5: Initating the ADM to run in the Operate mode.

for the YAML input file. YAML syntax significantly revolves around the list and dictionary data structures.

```
Files:
  fit_file:
    name: 'data/file_1.csv'
    input_col: [0, 1, 2, 3, 4, 5]
    output_col: [6]
  unsupervised_train:
    name: 'data/file_1.csv'
    input_col: [0, 1, 2, 3, 4, 5]
    output_col: [6]
  supervised_train:
    name: Null #'data/file_2.csv'
    input_col: [0, 1, 2, 3, 4, 5]
    output_col: [6]
  testing:
  - name: 'data/file_3.csv'
    input_col: [0, 1, 2, 3, 4, 5]
    output_col: [6]
  - name: 'data/file_4.csv'
    input_col: [0, 1, 2, 3, 4, 5]
    output_col: [6]

Operations:
  reduction: 3
```

Code Listing 7: YAML syntax

Specific to the ADM, the high level keys are 'Files' and 'Operations.' The associated values for the 'Files' key is another list of keys referring to the files that will be used in training. The three components for every specified file is as follows: name, input columns, and output column. The 'name' key refers to the name of the dataset file. The files that are getting consumed are CSV files, that not only have the input data for the learning models, but also the associated outputs. The corresponding columns are determined as values for the input and output columns. The 'Operations' key tag's value is another key, 'reduction.' If dimensionality of the data needs to be reduced, then a value specifying a number of PCA components can be supplied here. If dimensionality does not need to be reduced, an integer zero must be passed.

B. Interface

The Interface class provides the user an interface to the underlying library that assists in automating a majority of the learning process. The Interface class also serves as the intermediary between different classes, as visually presented in Figure 8. When an instance of this class is created, the sequence of events that follow are as follows. The YAML input file that is supplied is parsed, and the dataset filenames are stored. Following that, the fit file data is used for setting up the data preprocessing models. This involves splitting the input and output columns, normalizing the data, and reducing dimensionality if specified. This data then gets persisted into storage, helping prevent redundant actions. Once the data processing models have been set up, a similar sequence of events occur for the training dataset.

While the training data is prepared for training, it does not get initiated until the `genmodel_train(self, unsupervised_models, supervised_models)` function has been called. The classifiers for training are passed in as arguments to the function. The classifiers that are supplied are then trained with the processed training datasets. When the `get_testing_predictions(self)` function call is made in the script, predictions and output can be gathered for the testing datasets.

C. Data Preprocess

Instantiated from the Interface class, this class aims to preprocess data and prepare it for training. The first function it offers is `split(self, in_file, x_cols, y_cols)`. The input dataset is read and split into input columns and output label column. The Interface class makes a call to fit

the data scaling mode. The fitted model is saved in storage. If the dimensions need to be reduced, then a reduction model is also fitted. With the fitted models in place, the training datasets follow a similar routine to be ready for training.

D. Data Persist

The purpose for the DataPersist class is to save data in storage. Following the data preprocessing in the previous step, the modified data is passed to this class, so it can be persisted. The other important functionality that this class offers is retrieving saved data.

E. Anomaly Detector

The training and retrieving of predictions for datasets occur in the AnomalyDetector class. It is instantiated with both unsupervised and supervised models passed in as arguments.

Independent functions exist for fitting supervised and unsupervised classifiers, as the dataset used for training the particular type of classifier is different. Specifically, unsupervised classifiers do not require output labels, however supervised classifiers do. From the interface class, appropriate function calls are made to train the classifiers. The user is allowed to supply multiple classifiers, and they are iteratively trained with their respective datasets (labeled or unlabeled).

When the `classify_unsupervised(self, data)` and `classify_supervised(self, data)` functions are called when predictions are needed for validation and testing datasets. Similar to training, these functions iterate over all trained classifiers, and return predictions collectively. Before presenting them to the user, this class also calls on the OutputMatrix class for evaluating the predictions and their true labels.

F. Output Matrix

The OutputMatrix class is only accessed from the AnomalyDetector class. The confusion matrix is set up when the `output_matrix(self, true_y, pred_y)` function is called. The confusion matrix consists of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). A positive label refer to an anomaly prediction. A negative label refers to a benign prediction. The confusion matrix is further used to set the accuracy, recall, precision, and the F1 measure. Accuracy is the percentage of true positives and negatives of the whole dataset. Recall is the proportion of actual positives that are properly identified as such. Of the positive predictions, precision is the percentage that had true positive labels. The F1 score is a weighted average of both the recall and precision. When using labeled data, the F1 score is the easiest and best way to determine how good a classifier is.

VIII. Evaluation

A. Experiment Setup

The industrial arm we use is to evaluate the ADS is the Yaskawa Motoman MH5, a stationary arm. Equipped with six axes, allowing the arm six Degrees-of-Freedom (DOF). Figure 9 presents a visual of the industrial arm. The arm is

Fig. 9: Yaskawa Motoman MH5

controlled by the FS100 robot controller. An Ubuntu machine running the ROS Kinetic distribution communicates with the robot controller. The tasks for the arm are defined within ROS, and the generated commands are passed on the robot controller. Motoman, the industrial arm vendor, provides an interface driver to translate ROS commands to low level actuation commands to the arm.

1) Industrial Arm Task: The sample task we set the industrial arm for evaluation is cutting a triangle from a piece of sheet metal but without a real laser at the end effector. Figure 11a presents a simulation of the model of the arm performing the said task. The green trace presents the path taken by the end effector in the process of accomplishing the task. The arm starts from its home position, bows down to follow the cutting motion, and then returns home. Figure 10 shows the joint state values in 3D space (dimensionality is reduced with PCA from 6 to 3). Note that this is how the data from Figure 11a's motion is represented in 3D space.

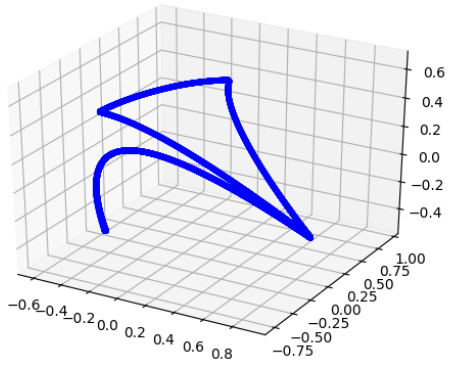
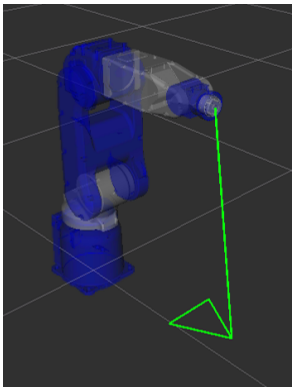


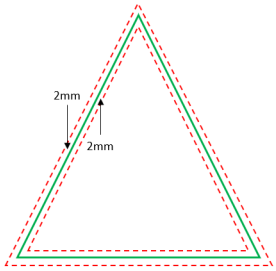
Fig. 10: 3D representation of the industrial arm's joint state values during benign task execution.

2) Collecting Dataset: The attack training data we collect builds a tolerance envelope of size 4mm, that is, 2mm on each side of the expected product. The associated product outlines for our training data is presented in Figure 11b.

Similarly, we created other labeled, datasets for validating and testing the classifiers. Table I presents some statistics on the datasets that we used for training and testing the classifiers. The "Total Data Points" column refers to the total number of data points present within the dataset. The 'Benign' and 'Anomalous' columns specify the number of benign and anomalous points within the dataset. Task counts indicate the number of task runs used for collecting the data points. While the training sets were specific to the classifiers, the validation and testing sets were used for both.



(a) Industrial arm, and end effector trajectory to accomplish the triangle cutting task.



(b) The red, dashed lines present the outline of sabotaged product. The benign product outline is bounded by 2mm on either side of the border.

Fig. 11: Industrial Arm Tasks

B. Evaluation metrics

A confusion matrix [7] is typically used to evaluate the performance of binary prediction models. A confusion matrix captures the rates of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). A positive label refers to an anomaly prediction. A negative label refers to a benign prediction. Data from the confusion matrix is used to compute the accuracy, recall, precision, and the F1 measure of the detection model. Accuracy is the percentage of true positives and true negatives w.r.t. the whole dataset (i.e., accuracy = (TP + TN)/(TP+FP+TN+FN)). Recall is the proportion of true positives that are properly identified as such (recall = TP/(TP+FN)). Of the positive predictions, precision is the percentage that had true positive labels (precision = TP/(TP+FP)). The **F1** score is a harmonic average of both the recall and precision (f1 = 2*precision*recall)/(precision+recall)). We evaluate our ADS classifiers by comparing the F1 score, as it takes into account both recall and precision.

C. Classifier Comparison

As previously mentioned we used grid search for finding the the best combination of model parameters for the most optimal results. This is also an exploratory search, so a wide range of parameters needed to be chosen and experimented with. This is done for both the OCSVM and RBFSVM classifiers. In addition, the classifiers are also trained with lower dimensionality data. The optimal OCSVM classifier was trained on the original dataset consisting of 6 features. The RBFSVM did better when trained with a reduced amount of features, particularly 3.

The results of the classifiers were initially compared against predictions for labeled data of the joint states. However, before production we assess the detection rate of attack modifications. In the process of training and validating the classifiers we find a threshold such that if the anomaly score is higher than it, the entire task execution is considered to be an anomalous execution. We tune the threshold so False Positives are low, but true positives are high. In our experiments we found that an appropriate threshold for the OCSVM classifier was 0.22. We set the threshold for RBFSVM to be 0.01, meaning if more than 1% of the joint states of a task are anomalous, then the entire task execution is considered to be anomalous execution. The performance of the classifiers with their respective detection thresholds are compared in Table II.

When comparing the detection rate of the models on training data, we find that the RBFSVM is consistently better. Not only does it provide high accuracy with joint state prediction, but also only requires a low threshold for high accuracy detection. In order to further evaluate RBFSVM we evaluate it against very subtle attacks introduced in Section VIII-D, and verify that they can be detected using the RBFSVM.

D. Attack Scenarios

Note that the RBFSVM was trained using tolerance envelope anomalous executions and not with any specific attacks. To see how effective the anomaly detection model is, we evaluate it using particular types of attacks representing the larger set of possible attacks. There are two dimensions to the attacks: the type of modification and its magnitude. Figure 12 presents seven different types of product modifications. We refer to these modifications as attacks, and address particular ones by their alphabet listing. Attacks a through d affect two sides of the product. They are evaluated with modification sizes of 1mm, 2mm, and 3mm. Attacks e through g are considered intricate, by affecting a portion of a single side of the product. These are evaluated by the change in slope they induce to the product.

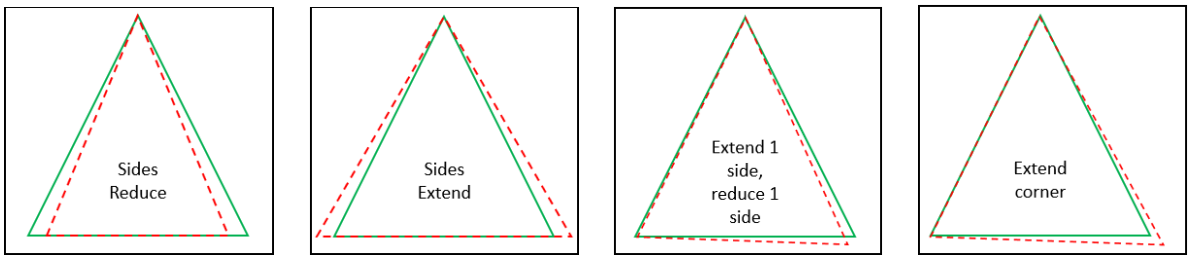
E. Performance Against Attacks

We now validate the performance of the RBFSVM classifier against more realistic attacks. Specifically, we first evaluate attacks a-d, with different attack magnitudes of them. There are no existing labels for these attacks, however the anomaly scores can be gathered and used for detection of anomalous tasks. Table III presents the scores. With the threshold of 0.01, RBFSVM is able to detect all the attack tasks including those with very small deviations (around $1mm$).

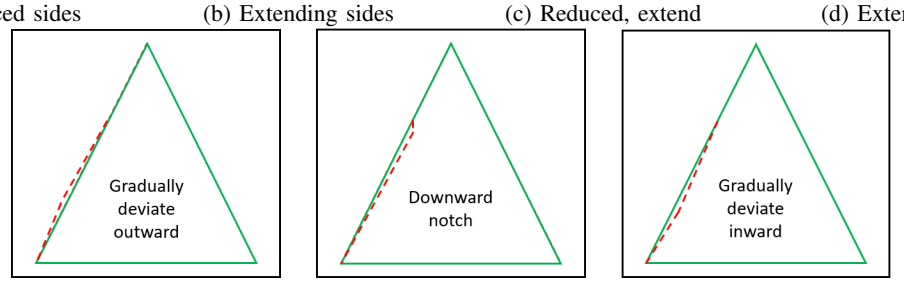
[7] A simple two-by-two matrix used to check the performance of a classifier.

TABLE I: Generated Datasets

	Benign Data				Anomalous Data		
	Total Data Points	Benign Points	Task Count	% Total Points	Anom. Points	Task Count	% Total Points
OC Train	15528	15528	30	100	0	0	0
RBF Train	20221	15527	30	76.79	4692	20	23.20
Validate	13953	9304	20	66.68	4649	20	33.3
Test	7767	5182	10	66.72	2585	10	33.28



(a) Reduced sides (b) Extending sides (c) Reduced, extend (d) Extend corner



(e) Side moving out (f) Quick jerk down (g) Side moving in

Fig. 12: These are seven types of attack modifications that we evaluate. Attacks a-d are general attacks. Attacks e-f are more intricate.

TABLE II: Comparing the performance of detecting attacks with OCSVM and RBFSVM. The OCSVM's threshold is set to be 0.22, while RBFSVM's is 0.01.

	Training		Validation		Test	
	OC	RBF	OC	RBF	OC	RBF
Dimension	6	3	6	3	6	3
TP	0.4	0.4	0.5	0.5	0.5	0.5
FP	0.04	0.0	0.5	0.0	0.0	0.0
TN	0.56	0.6	0.0	0.5	0.5	0.5
FN	0.0	0.0	0.0	0.0	0.0	0.0
Acc	0.96	1.0	0.5	1.0	1.0	1.0
Rec	1.0	1.0	1.0	1.0	1.0	1.0
Prec	0.91	1.0	0.5	1.0	1.0	1.0
F1	0.95	1.0	0.67	1.0	1.0	1.0

We further consider intricate attacks, e, f, and g, which affect a portion of a single side of the manufactured product. The results in Table IV are presented as changes in the slope of the product's side. The slope of the benign part is 2. As is seen from the table except for one attack where the change in slope is very small (=0.01) all attacks executions' anomaly scores are higher than 0.01. Figure 13 shows the joint state values in 3D space (dimensionality is reduced with PCA from 6 to 3) for an anomalous run of reduced sides attack of magnitude $2mm$.

IX. Limitations

While our learning-based anomaly detection approach is promising it has the following limitations. First, if an attacker introduces modifications falling within the tolerance boundary, they may be able to evade detection. Like with any anomaly detection systems an attack may stay under the detection threshold.

Second, while the learning-based approach saves the effort

TABLE III: Anomaly scores for each of the attack tasks when using RBFSVM. RBFSVM threshold was set to be 0.01, and thus all attacks shown were classified as anomalous.

	Attack a	Attack b	Attack c	Attack d
1mm	0.0858	0.0271	0.1459	0.0814
2mm	0.1543	0.1553	0.1578	0.0854
3mm	0.198	0.1934	0.1709	0.1362

TABLE IV: Anomaly scores for intricate attacks using RBFSVM. With a threshold of 0.01, all of the attacks except one will be classified as anomalous.

Slope	Anomaly Score
4	0.0912
3	0.0716
2.75	0.0603
2.17	0.0605
2.1	0.0463
2	0.0019
1.99	0.0037
1.97	0.0304
1.91	0.0324
1.86	0.0605
1	0.0788

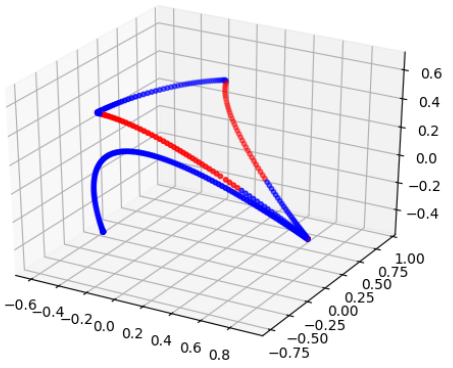


Fig. 13: 3D representation of the industrial arm's joint state values during anomalous task execution. Reduced sides attack of magnitude $2mm$ is executed. Blue points are benign points, and red points are anomalous points as classified by RBFSVM.

or building and maintaining a detailed model of the robot as is required for model-based anomaly detection(e.g., [17]), the classification model and the associated tolerance envelope will need to be retrained and tuned whenever the manufacturing task is modified. While any changes to the manufacturing task will go through a commissioning/testing phase that can be used for re-training the models, this overhead can become significant if the tasks are changed frequently. Aging and mechanical wear of the robot may mean that the learned model may need to be updated over time even if the task remains the same. These issues will be studied in future work.

Third and more significant limitation is that our ADS relies on the joint state values provided by the robot controller. If an adversary is able to compromise the robot controller and provide false joint state values (similar to how false sensor data was reported in Stuxnet attack) then the attacks may go undetected. To thwart such more sophisticated attacks, we need to be able to authenticate or corroborate the joint state values by using other available sensor data if any. In fact, having an independent (of the robot controller) way to verify joint state values will become critical. Fortunately, appropriately deployed wireless motion sensors such as accelerometers and gyroscopes attached to the robotic arm may be leveraged to independently corroborate the joint state values. This will be explored in our future work.

Fourth, if an attacker has infiltrated the ROS system, an ADM module within ROS may also be susceptible. Fortunately, ROS is capable of functioning in a distributed manner and thus our ADM module can be executed from an independent hardware with a one directional information flow in a similar manner to trusted logging approaches.

Finally, while our approach is designed for detecting anomalous behavior induced by malicious attackers, it may also detect deviations caused due to wear and tear of the robotic arm (e.g., aging or failing robotic arm parts) and this may lead to false alarms if not addressed. These issues will be considered in our future work.

X. Related Works

In recent years, security issues for both smart manufacturing and robots have received some attention. Work on robot security broadly falls into security of the ROS framework and security of robots and robotic vehicles such as automated cars. Work on smart manufacturing security covers general security issues with smart manufacturing and with additive manufacturing systems specifically.

a) ROS Security: ROS was a community project from the robotics academic community and security was not a priority in the beginning. McClean et al. were the first to empirically present the security concerns within ROS [23]. Among the range of vulnerabilities in ROS, the high stake ones are plain-text communication, unprotected ports, and unencrypted data storage. They essentially show that compromising CIA (Confidentiality, Integrity, and Availability) is relatively easy. Following this, a steady growth of papers have tried to address these concerns.

One of the earlier works, ROSRV, tackled safety concerns within ROS [18]. This Runtime Verification framework allows for users to specify safety constraints. Constraints are enforced by introducing access control policies into ROS' publish-subscribe model. An intervening node monitors the specified topic for messages violating safety constraints. Our work resembles ROSRV in that we also monitor a topic in our detection method, but we do not censor any messages from reaching their target node. ROSRV requires users to specify the safety constraints manually. This can be seen as akin to specifying the tolerance envelope in our work.

In 2016, Lera et al. [20] focused on ROS message security and proposed a framework where plaintext messages are republished as encrypted messages. While it is a viable solution, the overall performance is reduced in the system. Dieber et al. [15] presented a security architecture for message authentication and encryption. This is realized by an authorization server that only allows authorized nodes to communicate within the system. During registration, each node is required to provide a certificate to be accepted into the system. The same group extended the work by presenting a peer-to-peer way to secure communication between two nodes, providing authenticity and confidentiality [8]. Our framework is complementary to these approaches and can in fact benefit from the security mechanisms proposed in these.

The Open Source Robotics Foundation (OSRF), the organization behind ROS, recently initiated the SROS project which aims to secure the ROS ecosystem [41], [9]. Along with TLS/SSL support for socket level communication, SROS also makes use of AppArmor[8] profiles for isolating ROS processes. SROS's security is at the Transport layer.

[8]A linux security module that implements mandatory access control

Secure ROS [38] has a similar goal for secure communication, but accomplishes this with IPSec in transport mode, at the network level. These previous works on security work within ecosystem focused on preventive controls. Our work on the other hand focuses on a detection framework for ROS applications.

b) Robot Security: Both model-based and learning-based anomaly detection approaches have been studied in the context of mobile robots(e.g., [7], [17]). A behaviour-based detection mechanism for robotic vehicle is presented in [7]. In the learning phase, the normal range of the robot is learned. During operation, features are tracked, classified, and the system is determined to be under attack or not. A model-based anomaly detection method for mobile robots is discussed in [17]. In our case we have the advantage of dealing with a fixed robot which performs repetitive tasks, while on the other hand mobile robots typically have more sensor data that can be leveraged for anomaly detection.

HoneyBot a honeypot for networked robots was introduced in [19]. Attacks on industrial robot controllers have been discussed in [29]. The attack scenario we considered in this work is inspired by the attacks discussed in [29].

c) Smart Manufacturing Security: Pan et al. presented a taxonomy classifying cyber-physical attacks in manufacturing processes [28]. Attacks on additive manufacturing (AM) or 3D printing processes that can introduce visually undetectable modifications and lead to performance failures have been demonstrated [4], [37], [43]. To defend against such sabotage attacks, power consumption monitoring [26] and digital audio signature-based [22] approaches have been previously proposed for detecting anomalies in additive manufacturing processes. Our robot arm trajectory monitoring approach is complementary to those approaches.

d) ADS in Other CPS: Both learning-based and specification or model-based anomaly or intrusion detection schemes have been proposed for other CPS environments such as power grids (e.g., [25], [6], [3], [31], [42], [21], [40]). Our work is inspired by such approaches and leverages the unique characteristics of manufacturing robotic arms and ROS to defend against stealthy product sabotage attacks.

XI. Future Work

To create a more robust classifier, data published by other ROS processes can be incorporated as features for the learning models being trained. However, the primary challenge, as in any distributed system, is consensus between the processes. The challenge here is two-fold. The processes should not only be publishing at a similar rate, but their clocks must be synchronized so all collected features describe the arm at the same time step. This challenge should be revisited in the future works.

Another limitation to address moving forward is scaling the solution, so the new tasks can be introduced with less commissioning phase overhead. A trivial solution is building multiple classifiers for multiple tasks, and adding a 'detection' step before classification. Online learning methods can also be explored, where model continues to train, while adapting to the data it is fed.

XII. Conclusion

As smart manufacturing gains momentum, cyber attack risks emerging from the integration of advanced automation and information technologies need to be addressed. In this work we explored a learning-based anomaly detection approach to deal with stealthy product sabotage attacks on robotic arms in a manufacturing pipeline. We show that the challenge of creating representative labeled data can be overcome by introducing the concept of a tolerance envelope, which leverages the tolerance specifications of a product. A preliminary evaluation on a real robotic arm showed that the approach is promising. However, model training and tuning in practice, and authenticity of the robotic motion information used by the anomaly detector are practical concerns that need to be addressed and will be tackled in future work.

ACKNOWLEDGMENTS

I am grateful for my advisor, Dr. Rakesh Bobba, for his guidance every step of the way. I want to thank my committee members, Prof. Bill Smart, Dr. Jesse Walker, Prof. Yeongjin Jang. I also extend my gratitude to my research group. I am thankful they were always willing to hear me out. I am incredibly grateful for each and every one of them.

References

[1] Featured robots. http://robots.ros.org/. Accessed: 2018-02-23.

[2] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] Muhammad Qasim Ali and Ehab Al-Shaer. Probabilistic model checking for ami intrusion detection. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, pages 468–473. IEEE, 2013.

[4] Sofia Belikovetsky, Mark Yampolskiy, Jinghui Toh, Jacob Gatlin, and Yuval Elovici. dr0wned – cyber-physical attack with additive manufacturing. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX Association.

[5] Jason Bell. Support vector machines. *Machine Learning: Hands-On for Developers and Technical Professionals*, pages 139–160, 2014.

[6] R. Berthier and W. H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *Pacific Rim International Symposium on Dependable Computing, IEEE(PRDC)*, volume 00, pages 184–193, 12 2011.

[7] Anatolij Bezemskij, George Loukas, Richard J Anthony, and Diane Gan. Behaviour-based anomaly detection of cyber-physical attacks on a robotic vehicle. In *Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), International Conference on*, pages 61–68. IEEE, 2016.

[8] Benjamin Breiling, Bernhard Dieber, and Peter Schartner. Secure communication for the robot operating system. In *Systems Conference (SysCon), 2017 Annual IEEE International*, pages 1–6. IEEE, 2017.

[9] Gianluca Caiazza. Security enhancements of robot operating systems. B.S. thesis, Università Ca'Foscari Venezia, 2017.

[10] Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 2016.

[11] Microsoft News Center. Industrial robots improved by robotic technology. https://enterprise.microsoft.com/en-us/customer-story/industries/discrete-manufacturing/microsoft-and-kuka-present-intelligent-future-generation-of-robotics/, 2015. Accessed: 2018-06-09.

[12] ROS-Industrial Consortium. Ros-industrial. https://rosindustrial.org/, 2018. Accessed: 2018-02-23.

[13] Laurence Cruz. Digitization and iot reduce production downtime. https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1764957, 2016.

[14] Jim Davis, Thomas Edgar, James Porter, John Bernaden, and Michael Sarli. Smart manufacturing, manufacturing intelligence and demand-dynamic performance. *Computers & Chemical Engineering*, 47:145–156, 2012.

[15] Bernhard Dieber, Severin Kacianka, Stefan Rass, and Peter Schartner. Application-level security for ros-based applications. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4477–4482. IEEE, 2016.

[16] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[17] Pinyao Guo, Hunmin Kim, Nurali Virani, Jun Xu, Minghui Zhu, and Peng Liu. Roboads: Anomaly detection against sensor and actuator misbehaviors in mobile robots. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 574–585, June 2018.

[18] Jeff Huang, Cansu Erdogan, Yi Zhang, Brandon Moore, Qingzhou Luo, Aravind Sundaresan, and Grigore Rosu. Rosrv: Runtime verification for robots. In *International Conference on Runtime Verification*, pages 247–254. Springer, 2014.

[19] C. Irvene, D. Formby, S. Litchfield, and R. Beyah. Honeybot: A honeypot for robotic systems. *Proceedings of the IEEE*, 106(1):61–70, Jan 2018.

[20] Francisco Javier Rodrıguez Lera, Jesús Balsa, Fernando Casado, Camino Fernández, Francisco Martın Rico, and Vicente Matellán. Cybersecurity in autonomous systems: Evaluating the performance of hardening ros. *Málaga, Spain*, page 47, 2016.

[21] Leandros A Maglaras and Jianmin Jiang. Intrusion detection in scada systems using machine learning techniques. In *Science and Information Conference (SAI), 2014*, pages 626–631. IEEE, 2014.

[22] Thomas Mativo, Colleen Fritz, and Ismail Fidan. Cyber acoustic analysis of additively manufactured objects. *The International Journal of Advanced Manufacturing Technology*, 96(1-4):581–586, 2018.

[23] Jarrod McClean, Christopher Stull, Charles Farrar, and David Mascareñas. A preliminary cyber-physical security assessment of the robot operating system (ros). In *Unmanned Systems Technology XV*, volume 8741, page 874110. International Society for Optics and Photonics, 2013.

[24] Microsoft. Kuka systems creating a connected factory. https://www.youtube.com/watch?v=yPNpbEgs42w, 2015.

[25] Y. Mo and B. Sinopoli. Secure control against replay attacks. In *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 911–918, Sept 2009.

[26] Samuel B Moore, Jacob Gatlin, Sofia Belikovetsky, Mark Yampolskiy, Wayne E King, and Yuval Elovici. Power consumption-based detection of sabotage attacks in additive manufacturing. *arXiv preprint arXiv:1709.01822*, 2017.

[27] L Obregon. Secure architecture for industrial control systems. *SANS Institute InfoSec Reading Room*, 2015.

[33] Matthew Robinson. Ros-industrial overview. https://wiki.ros.org/Industrial/#ROS-Industrial_Overview. Accessed: 2018-03-13.

[28] Yao Pan, Jules White, Douglas C Schmidt, Ahmad Elhabashy, Logan Sturm, Jaime Camelio, and Christopher Williams. Taxonomies for reasoning about cyber-physical attacks in iot-based manufacturing systems. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(3), 2017.

[29] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. An experimental security analysis of an industrial robot controller. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 268–286. IEEE, 2017.

[30] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[31] Mohammad Ashiqur Rahman, Ehab Al-Shaer, and Rakesh B. Bobba. Moving target defense for hardening the security of the power system state estimation. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 59–68, New York, NY, USA, 2014. ACM.

[32] Matthew Robinson. Industrial support hardware. https://wiki.ros.org/Industrial\#Supported_hardware. Accessed: 2018-04-04.

[34] Aaron Martinez Romero. Ros concepts. https://wiki.ros.org/ROS/Concepts, 2014.

[35] Isaac Saito. roslaunch. https://wiki.ros.org/roslaunch, 2018. Accessed: 2018-05-24.

[36] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[37] Logan D Sturm, Christopher B Williams, Jamie A Camelio, Jules White, and Robert Parker. Cyber-physical vulnerabilities in additive manufacturing systems: A case study attack on the. stl file with human subjects. *Journal of Manufacturing Systems*, 44:154–164, 2017.

[38] Aravind Sundaresan, Leonard Gerard, and Minyoung Kim. Secure ros. https://sri-csl.github.io/secure_ros/. Accessed: 2018-02-23.

[39] Dirk Thomas. Ros introduction. https://wiki.ros.org/ROS/Introduction, 2014. Accessed: 2017-02-23.

[40] David I. Urbina, Jairo A. Giraldo, Alvaro A. Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1092–1105, New York, NY, USA, 2016. ACM.

[41] Ruffin White, Dr Christensen, I Henrik, Dr Quigley, et al. Sros: Securing ros over the wire, in the graph, and through the kernel. *arXiv preprint arXiv:1611.07060*, 2016.

[42] Man-Ki Yoon and Gabriela F Ciocarlie. Communication pattern monitoring: Improving the utility of anomaly detection for industrial control systems. In *NDSS Workshop on Security of Emerging Networking Technologies*, 2014.

[43] Steven Eric Zeltmann, Nikhil Gupta, Nektarios Georgios Tsoutsos, Michail Maniatakos, Jeyavijayan Rajendran, and Ramesh Karri. Manufacturing and security challenges in 3d printing. *Jom*, 68(7):1872–1881, 2016.