Online Control Adaptation for Safe and Secure Autonomous Vehicle Operations

Mahmoud Elnaggar[1], Jason D. Hiser[2], Tony X. Lin[3], Anh Nguyen-Tuong[2],
Michele Co[2], Jack W. Davidson[2], and Nicola Bezzo[1,3]
[1]Department of Systems and Information Engineering
[2]Department of Computer Science
[3]Department of Electrical and Computer Engineering
University of Virginia
Email: {mae3tb,hiser,an7s,txl5gd,mc2zk,jwd,nbezzo}@virginia.edu

***Abstract*—Modern cyber-physical systems, like automotive systems and aerial vehicles, are not built with cyber security in mind. Several techniques have been developed recently to overcome cyber-attacks on cyber-physical systems both at the software and the control levels. Adding such cyber security techniques to protect a system against malicious attacks, however, can incur runtime overheads that, in the case of autonomous systems, results in performance degradation and may lead the system into unsafe states. In this paper, we propose a framework for online control performance adaptation for secure and safe navigation of autonomous vehicles. Our approach leverages model predictive control (MPC) and knowledge about the system dynamics and the maximum performance degradation that cyber security techniques can impose at every time step to compute the control input that guarantees a safe operation of the system at all times. We validate the proposed approach both with simulations and experiments for an unmanned ground vehicle (UGV) motion planning case study in a cluttered environment.**

I. Introduction

Nowadays, modern vehicles are becoming autonomous and safer thanks to the large use of embedded computers and sensors. This increased use of automation has however opened the door to multiple cyber-security attacks. In fact, any computing system that accepts inputs that can potentially be manipulated by a malicious adversary is potentially vulnerable to cyber attack. For example, autonomous vehicles can be subject to wireless attacks in which an adversary uses a wireless communication channel to send malicious inputs (e.g., control inputs, map updates, mission updates, etc.) to hijack the system and exploit vulnerabilities in the system [16].

Cyber security researchers have invented a myriad of techniques to protect against cyber attacks. With few exceptions, these techniques add runtime overhead to the system, which increases the time to complete any given task. For example, the widely deployed *address space layout randomization* (ASLR) protection [30], which randomizes the location where code is loaded, can incur significant overheads. Payer reports overheads as much as 26% for some programs [21]. Other techniques such as control-flow integrity [4], *N*-variant systems [9], instruction-set randomization [5] can incur higher overheads—sometimes as much as 100 to 300%, or more. These overheads can be tolerated because they are in a system where resistance to cyber attack is imperative.

When such techniques are deployed on autonomous dynamical systems that interact with their environments (e.g., drones, cars, robots), it is necessary to adapt the control performance to avoid unsafe conditions due to the delays introduced by the security mechanisms. For example, controllers are built to run at a given rate. During normal execution, inputs are applied for a certain duration. A delay may result in an erroneous application of the same input for longer times which can create unsafe conditions. For example, a vehicle may speed up or collide with other obstacles or vehicles.

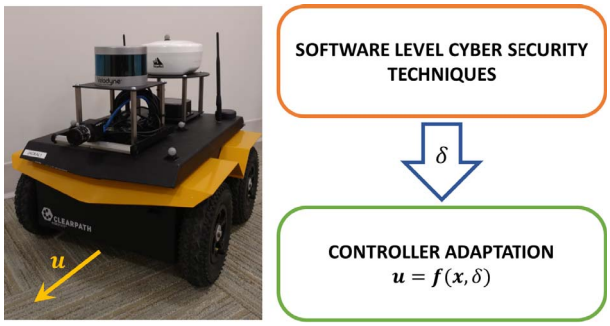


Fig. 1. Motivation for research. Cyber-security techniques affect the performance of an AV because they introduce delays δ that need to be taken into account when designing the control law $\boldsymbol{u} = f(\boldsymbol{x}, \delta)$.

The key insight of this paper is that many cyber security techniques can be used to enhance autonomous vehicle (AV) security, if the vehicle is capable of dealing with the overhead implications. To realize this possibility, we address the implications of applying security techniques on autonomous vehicle control systems by focusing on adapting low-level controllers to maintain the required system safety properties.

We present a method for adapting controllers to tolerate additional runtime overhead by: i) leveraging the dynamical model of the system which we assume to know, or which can be extracted through system identification techniques and ii) using a risk-based model predictive approach to compute the next control input that minimizes the possibility to reach undesired states (safety constraints) while maintaining a certain level of performance.

The rest of the paper is organized as follows: In Section II, we discuss related work about cyber-physical systems security. In Section III we define the problem formulation while in Section IV, we discuss the implications of applying state-of-the-art cyber-security techniques to AVs. In Section V, we

978-1-5386-3439-4/17/$31.00 ©2017 IEEE

present our online adaptation algorithm followed by simulation and experimental results in Section VI. Finally we draw conclusions in Section VII.

II. Related Work

In recent years there has been a growing interest by the research community in the security of AVs. Kim et al. discuss the security vulnerabilities of AVs [16]. Specifically, they note that wireless communication channels provide one means for attack, including gaining full control of the AV by exploiting buffer overflow vulnerabilities [17]. Javaid et al. develop a cyber security threat model for AVs [15]. They discuss integrity attacks that include the use of malicious code or exploiting existing subroutines. In discussing the technical challenges of securing AVs, Wyglinski et al. [29] present remote access vulnerabilities which include, for example, introducing malicious code into a system through an update.

To counter attack against systems, security researchers have developed numerous techniques to prevent attacks. Abadi et al. discuss control-flow integrity – a powerful technique to prevent control hijacking attacks [4]. The use of various randomization techniques to introduce artificial diversity has become a standard and widely deployed cyber defense [14]. The use of intrusion detection systems [27] and anti-virus systems [25] are also widely used to prevent various types of intrusions and attacks.

From a control perspective, recent research on cyber-physical systems (CPS) has tackled security problems on modern vehicles and autonomous robotic systems considering sensor, actuator, communication, controller, and physical attacks. Recent incidents that illustrate the susceptibility of CPS to attack include the StuxNet attack on an industrial SCADA system [11] and attacks on automobiles [1], vessels [23], and military drones [3] in which systems may be tampered with, yielding unexpected behavior.

Hence, there is a need to design and develop systems that can tolerate, prevent, and detect attacks and recover and reconfigure to guarantee safety. In recent years we have witnessed an increased academic effort to address such cyber security issues. Most of the literature and current research, however, focuses on how to detect and estimate malicious attacks considering partial or full observability of the internal state of a system [19], [20], [12] and does not deal with the problem of adapting the controller, which is the subject of this paper.

Consequently, significant research effort has been applied to develop control-level techniques that exploit knowledge of system dynamics to address the problem of attack-resilient state estimation, as well as intrusion detection under different types of attacks on sensors, actuators and communication networks while considering robust controllers like PID regulators [7], [26], [20]. Another example is the DARPA High-Assurance Cyber Military Systems (HACMS) program [2], [7], [19], devoted to the design of secure control systems for autonomous vehicles. In HACMS, Bezzo and his colleagues developed techniques for resilient state estimation and resilient sensor fusion that can tolerate various kinds of sensor attacks.

In this work, we propose to go one step further in the research on CPS security by considering controller adaptation.

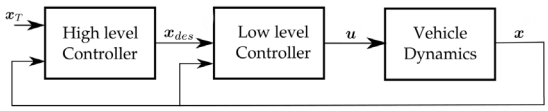


Fig. 2. AV High-level and low-level control architecture

We investigate securing the system using control and software level security techniques that introduce overhead that in turn needs to be taken into account to guarantee that the system is also safe and can maintain a certain level of performance.

Our contribution is twofold: i) we develop a risk-based algorithm to adapt the control inputs provided to the AV when the system is running protection mechanisms that can introduce significant overhead, and ii) we integrate software-level cyber security techniques on a real autonomous vehicle. Finally, we demonstrate the proposed framework and the integration of such techniques on an unmanned ground vehicle tasked to navigate autonomously in a cluttered environment.

III. Problem Statement

Within this work, we are interested in finding a policy to online adapt the low-level control law of an autonomous vehicle subject to time-varying delays to guarantee safety constraints (e.g., avoid collisions with obstacles, entering undesired regions in the environment). We assume that the robot dynamics can be represented as a continuous linear time-invariant (LTI) system of the following form:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{x}(t) \end{aligned} \tag{1}$$

where $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and $\boldsymbol{y}(t)$ are the state vector, control vector, and sensor measurements vector at time t respectively. $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are the system matrix, the input matrix, and the output matrix, respectively.

Usually, an autonomous driving control algorithm consists of a high-level and a low-level controller as depicted in Fig. 2. The high-level (HL) controller is typically used for path planning to generate collision-free trajectories to a goal location $\boldsymbol{x}_T$ while the low-level (LL) controller is used to track the set of points $\boldsymbol{x}_{des}$ along the path computed by the HL controller. The low-level controller is usually implemented as a computer application which runs in a discrete fashion, and thus it needs to be described with a discrete time model. Protecting the controller application by applying cyber security techniques (discussed in more depth in the next section) imposes performance overhead on the controller. This performance overhead can be time varying, depending on the implementation and the complexity of the controller code. From a controller point of view, adding a delay $\delta(k)$ at each iteration in a control loop that is running with sampling time t_s, is equivalent to a control loop that runs at $t_s + \delta(k)$ rate. Therefore, the discrete model of the system, considering time varying delays becomes:

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}'_d(\delta(k))\boldsymbol{x}(k) + \boldsymbol{B}'_d(\delta(k))\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k) \end{aligned} \tag{2}$$

where

$$\boldsymbol{A}'_d(\delta(k)) = e^{\boldsymbol{A}(t_s+\delta(k))}$$
$$\boldsymbol{B}'_d(\delta(k)) = \int_0^{t_s+\delta(k)} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda \quad (3)$$

Designing a robust controller may handle these delays, however if the overhead is large, this can create instabilities and possible performance degradation and unsafe behavior because the input may be applied for longer sampling intervals than the designed rate, driving the system to unexpected and unsafe states (i.e., the AV may run into an obstacle or enter a safety critical region). In this work, we address this issue by computing a control policy that guarantees that at any time the AV is running safely. For the sake of simplicity, we consider obstacle avoidance case studies, however, the proposed framework can be applied to other types of operations involving in general cyber-physical systems.

Formally, in this work we are interested in solving the following problem:

Problem 1. Online control adaptation for safe and secure AV operations under delayed control: *An autonomous vehicle (AV) is tasked to complete a mission over an obstacle populated environment $\mathcal{W} = \mathcal{F} \cup \mathcal{O}$ in which $\mathcal{F}$ represents the obstacle-free region of the environment and vice versa $\mathcal{O}$ is the region occupied by obstacles. The AV is modeled by* (2) *in which delays, δ, are due to security mechanisms running on the controller to protect the system against malicious cyber-attacks. Given these constraints, the current state of the system $\boldsymbol{x}$, the desired input $\boldsymbol{u}$ with no delay, and the maximum delay that we can expect δ_{max}, the objective is to find a control policy $\hat{u} = f(x, u, \delta_{max}, t)$ such that $\boldsymbol{x}(t) \not\subset \mathcal{O}, \forall t \geq 0, \boldsymbol{x}(0) \subset \mathcal{F}$.*

In other words, we are interested in finding a policy to guarantee that a secured AV is safely performing its objective.

IV. Software Cyber Security Techniques

In this section, we discuss the implications of applying state-of-the-art cyber-security techniques to AVs in terms of system performance overheads. There are numerous software techniques that have been developed to protect cyber systems from attacks. These techniques range the spectrum from approaches that monitor the entire system for undesired behaviors or intrusions (e.g., malicious changes to system parameters, corruption of key files, etc.) to specific defenses applied to an application that prevent a particular attack or attack class (e.g., control-flow integrity, instruction-set randomization, etc.). The following sections review these approaches and their potential impact on the control software of an AV.

A. System-level Software Security

With system-level software security techniques, such as anti-virus and intrusion-detection systems, the entire system is monitored for indications of compromise. For example, an intrusion detection system may monitor network traffic for anomalous traffic, monitor system calls for suspicious patterns, or other unusual patterns of behavior. An anti-virus system may periodically scan the filesystem to determine if a virus has been introduced into the system. Modern anti-virus systems may also monitor network connections and scan for incoming malicious software. As noted in the introduction, these cyber security systems can significantly impact system performance.

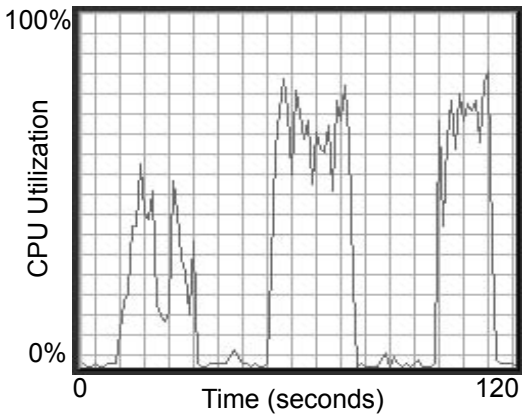


Fig. 3. CPU utilization with periodic virus scanning

For example, Fig. 3 shows, over a 120s interval, the CPU utilization of a system with an anti-virus system installed and configured to scan periodically. The measured system was running Windows 7 Enterprise on a Intel Core-I7-3820QM with 16GB of RAM, and a 1TB disk drive. The anti-virus system deployed was Symantec Endpoint Protection, Enterprise Edition.

The graph shows that CPU utilization increases significantly (with a high of 82%) when the anti-virus system is active. Such high levels of CPU utilization could significantly affect the response time of the low-level controller resulting in unsafe operations. As evidenced by the graph in Fig. 3, anti-virus systems often exhibit periodic behavior—scans are initiated at preset intervals. It should be noted, however, that many anti-virus systems also do on-demand scans that are initiated by particular events such as a file being opened, the execution of a program, modifications of configuration data, etc. Consequently, it is often not possible to predict when and for how long a system-level, software-based defense may impact CPU utilization.

B. Application-level Software Security

In contrast to system-level software security, application-level security protects a single application. One such application-level approach is control-flow integrity (CFI) [4].

The basic idea is to extract a specification of a program's intended control flow. Using this specification, instrumentation is added to the program to enforce that specification at run time. If an adversary attempts to subvert control flow through an attack (e.g., a return-to-libc attack [22] or return-oriented programming attack [24]), the added instrumentation detects the attack and blocks it.

Using the SPEC CPU2006 benchmark suite, an industry standard set of programs used to compare CPU performance [13], we measured the overhead of a state-of-the-art

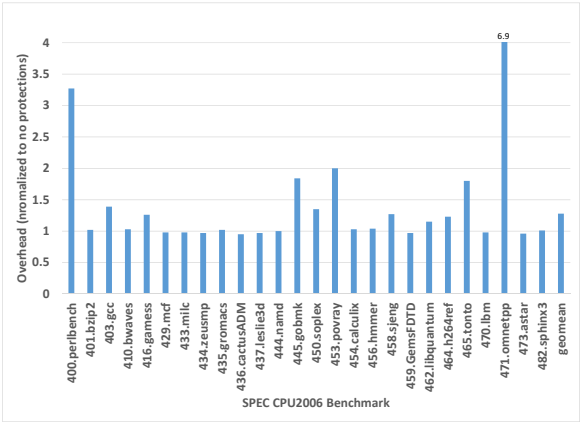


Fig. 4. CFI performance overhead on selected benchmarks

implementation of CFI. Fig. 4 graphs the overhead of CFI normalized to non-protected execution.

The graph shows that the overhead of CFI varies significantly based on the application. For example, the benchmark `perlbmk` (a perl interpreter) incurs more than a 3X overhead. The application `omnetpp` (a discrete simulation of a large Ethernet network) incurs almost a 7X overhead. Furthermore, the overhead of CFI on an application can vary depending on the inputs it processes. Like the system-level security techniques, it is very difficult to predict the overhead of CFI at any particular point in a program.

Another promising technique for protecting applications is the *N*-variant system approach [10], [9]. This approach employs the systematic application of artificial diversity to prevent large classes of attacks. One of the major benefits of the *N*-variant system approach is that it is possible to provide formal proofs that classes of attacks are not possible.

The key idea is to execute a carefully created set of diversified versions of an application on the same inputs, and monitor the variants to detect deviations in their behavior (see Fig. 5). If the variants are diversified so that the exploitation sets of the variants are disjoint, deviations in behavior are guaranteed. As a simple example, consider two variants whose code space are disjoint. An attack that depends on a code location (e.g., a return-oriented programming attack) cannot succeed in both variants simultaneously—one of the variants will crash.

A second major benefit of the *N*-variant system approach is that it is possible to recover from an attack and continue operation. This property is extremely valuable for AVs where a fail-stop reaction could have serious consequences. However, the recovery operation is not without cost. During the recovery process, normal operation of the system is suspended for some period of time.

Because of the unpredictable nature of the overhead of different cyber security techniques, it is vital to develop online controller adaptation techniques that can tolerate overheads and delays imposed by software cyber security techniques.

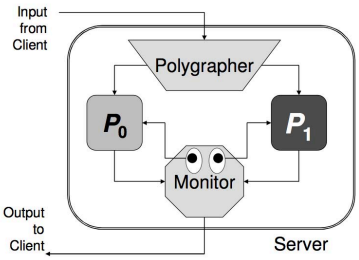


Fig. 5. N-Variant Framework (from original USENIX paper [10])

V. Online Controller Adaptation

In this section, we describe our framework for online control adaptation of AVs to solve Problem 1. As described in the previous section, cyber security protection techniques add variable runtime performance overheads to protected software. In our problem, the protected software is the controller node on the AV. Our approach aims to guarantee the safety of the AV while maintaining a certain level of performance.

The framework that we propose consists of a series of steps in which we first predict the future inputs and states of the system and then re-plan these inputs to avoid the possibility to reach unsafe states. To achieve this behavior we consider both an estimate of the current delay $\delta_e(k)$ and an upper bound on the maximum possible delay δ_{max}. Because $\delta(k)$ is not known a priori, a conservative approach would be to compute $\boldsymbol{u}$ based on δ_{max}. The generated input will drive the AV always to a safe state, however, the performance of the system will be greatly reduced because the system will run unnecessarily slower. To maintain system's performance we propose the following adaptive procedure in which the input is selected considering the current state of the system in relation to the unsafe states to avoid.

The first step in our approach consists of using the following Model Predictive Control (MPC) [6], to predict the evolution of the AV states over a finite horizon h.

$$J(\boldsymbol{x}(k), \boldsymbol{u}(k)) = \min_{\boldsymbol{u}(k)} \sum_{k=0}^{h-1} \boldsymbol{e}_x^T(k)\boldsymbol{Q}\boldsymbol{e}_x(k) + \boldsymbol{e}_u^T(k)\boldsymbol{R}\boldsymbol{e}_u(k)$$
$$\text{subject to } \boldsymbol{x}(k+1) = \boldsymbol{A}'_d(\delta(k))\boldsymbol{x}(k) + \boldsymbol{B}'_d(\delta(k))\boldsymbol{u}(k) \tag{4}$$

The result of (4) is a series of inputs $[\boldsymbol{u}(0), \boldsymbol{u}(1), \ldots, \boldsymbol{u}(k), \ldots, \boldsymbol{u}(h-1)]$ from the current time to the h time horizon where $\boldsymbol{e}_x(k)$ and $\boldsymbol{e}_u(k)$ are the errors between the current state $\boldsymbol{x}_c$ and the desired state $\boldsymbol{x}_{des}$ and the error between consecutive inputs, respectively. $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the weight matrices and $\boldsymbol{A}'_d$ and $\boldsymbol{B}'_d$ are calculated according to (3). One of the advantages of using MPC controller is the ability to define constraints on system states and the control inputs which are necessary to guarantee the safety of the system.

To determine the correct input to apply to the underlying system, one approach is to solve the MPC optimization problem at each time step k by changing $\boldsymbol{A}'_d$ and $\boldsymbol{B}'_d$ according to the measured delay $\delta(k-1)$. However, this approach cannot be realized as the runtime overhead introduced by the cyber

security techniques is time varying and not known a priori before implementing the control law. Instead, we consider the exponential weighted moving average algorithm (EWMA) which is widely used to monitor processes over time [18] to compute an estimated value of the delay at time k based on the previous estimates and measured delays as follows:

$$\delta_e(k) = (1-\alpha)\delta_e(k-1) + \alpha\delta(k-1) \tag{5}$$

where $\delta_e(k)$ is the estimated delay at time step k, $\delta(k-1)$ is the actual delay measured at time step $k-1$, and $0 \leq \alpha \leq 1$. We can determine an operating envelope by calculating the maximum delay δ_{max} which can occur at any time step using a variety of worst-case execution time (WCET) and profiling techniques as outlined in [28].

Once we have an estimate of the delay δ_e, we solve the MPC optimization at each time step k subject to the following constraint:

$$\begin{aligned} \boldsymbol{A}'_d(\delta_e(k)) &= e^{\boldsymbol{A}(t_s+\delta_e(k))} \\ \boldsymbol{B}'_d(\delta_e(k)) &= \int_0^{t_s+\delta_e(k)} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda \end{aligned} \tag{6}$$

The control input $\boldsymbol{u}_e(k)$ that we obtain by solving (4) subject to (6) is then used to predict the state $\boldsymbol{x}_e(k+1)$ reached by the AV in one time step. By applying $\boldsymbol{u}_e(k)$, one of the following three cases can occur:

1) $0 < \delta(k) < \delta_e(k)$: The actual delay is less than the expected delay, so the system will move to the next state but with slower performance than the one computed by the MPC.
2) $\delta(k) = \delta_e(k)$: The actual delay equals the estimated delay, so the system will move to the next state with the optimal performance computed by the MPC.
3) $\delta_e(k) < \delta(k) < \delta_{max}$: The actual delay is greater than the estimated delay, which means that the system states may evolve to undesired and unsafe states.

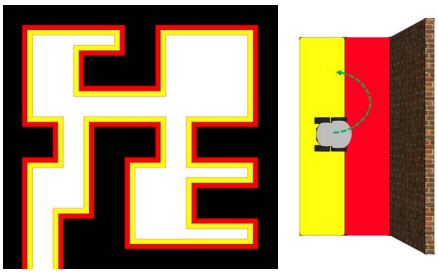
Thus, we propose an algorithm that adapts the control input to guarantee the safety of the AV and at the same time minimizes the degradation in its performance.

Given the shape, dynamics of the AV, and δ_{max} we can inflate the unsafe regions or obstacles to construct an inflated set $\mathcal{S}$ that satisfies the following condition:

$$\forall \boldsymbol{x}(k) \subset \mathcal{S}, \exists\, \boldsymbol{u}_c \in \boldsymbol{U} \text{ such that } \boldsymbol{x}_{max}(k+1) \not\subset \mathcal{S} \tag{7}$$

where $\boldsymbol{x}_{max}(k+1) = \boldsymbol{A}'_d(\delta_{max})\boldsymbol{x}(k) + \boldsymbol{B}'_d(\delta_{max})\boldsymbol{u}_c$, $\boldsymbol{U} = \{\boldsymbol{u} | \boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}\}$, with $\boldsymbol{u}_{min}$ and $\boldsymbol{u}_{max}$ the minimum and maximum controller inputs, respectively.

Fig. 6 shows an example of inflated obstacles. The figure shows a 2-D view of the configuration space of an AV. The yellow shaded regions indicate the points inside the set $\mathcal{S}$. It is clear that $\forall \boldsymbol{x}(k) \not\subset \mathcal{S} \implies \boldsymbol{x}(k) \not\subset \mathcal{O}$. The width d of the yellow region is calculated as $d = v_{max}(t_s + \delta_{max})$,where t_s is the sampling time and v_{max} is the maximum speed of the AV. Therefore, an AV that lies inside a safe region cannot cross this region in a single controller time step even if the maximum delay occurs. The width of the red shaded regions is calculated based on the shape and the turning radius of the AV to guarantee that it is able to apply a steering input that drives it outside the unsafe region without hitting an obstacle, Fig. 6(b).



(a) (b)

Fig. 6. Obstacles inflated regions. (a) Configuration space after inflating obstacles. (b) The red regions are constructed according to the AV's shape and maximum turning radius

By constructing the set $\mathcal{S}$, we can develop an adaptation algorithm that outputs a control input $\boldsymbol{u}_c$ only if the AV is inside the inflated obstacles set $\mathcal{S}$. To compute $\boldsymbol{u}_c$, we add $\boldsymbol{x}(k+1) \not\subset \mathcal{S}$ as a constraint in the MPC optimization problem. For ease of discussion we denote the MPC containing this extra constraint as $MPC_{Cons}(\delta_{max})$.

The online controller adaptation Algorithm 1 checks, at each time step k, if the state of the AV lies inside the inflated region $\mathcal{S}$. If this is true, it outputs a conservative control input $\hat{\boldsymbol{u}}(k) = MPC_{Cons}(\delta_{max})$ which guarantees that in the next time step $(k+1)$, the AV state will be outside the inflated obstacle set $\mathcal{S}$ even if maximum delay δ_{max} occurs. If the state of the AV at time step k is outside the inflated region, the algorithm calculates a controller input considering a delay $\delta_e(k)$. The adaptation of the controller comes into play by using a risk-based approach to compute the controller input when the AV is outside the inflated region, as follows:

1) MPC is used to compute a controller input $\boldsymbol{u}_{max}$ considering maximum delay δ_{max}.
2) A risk factor r that indicates the accuracy of the last estimated delay is calculated as follows:

$$r = \left\| \frac{\delta(k-1) - \delta_e(k-1)}{\delta_{max}} \right\| \tag{8}$$

3) Finally, the adapted controller input $\hat{\boldsymbol{u}}(k)$ which will be applied to the AV is computed as follows:

$$\hat{\boldsymbol{u}}(k) = \boldsymbol{u}_{max} + r\Delta\boldsymbol{u} \tag{9}$$

where $\Delta\boldsymbol{u} = \boldsymbol{u}_e - \boldsymbol{u}_{max}$

The algorithm generates a more conservative input as the risk factor r increases from 0 to 1. If $r = 1$, the algorithm generates the most conservative input $\boldsymbol{u}_{max}$ while if $r = 0$, that means that there is no risk and the delay estimate is very accurate, then algorithm generates a controller input $\boldsymbol{u}_e$.

Lemma V.1. *The input $\hat{\boldsymbol{u}}(k)$ generated by the adaptive Algorithm 1 is guaranteed to always drive the system toward safe states $\boldsymbol{x}(k+1)$ iff the delay is bounded between 0 and δ_{max}.*

Proof. Algorithm 1 generates $\hat{\boldsymbol{u}}(k)$ at each time step k to drive the system to the next state $\boldsymbol{x}(k+1)$. If $\boldsymbol{x}(k) \not\subset \mathcal{S}$, the state $\boldsymbol{x}(k+1)$ can only be in one of the following cases:

Algorithm 1 Online Controller Adaptation

Input: $\delta_e(k), \delta(k-1), \delta_e(k-1)\delta_{max}, \boldsymbol{x}(k), \mathcal{S}$
Output: $\hat{\boldsymbol{u}}(k)$
  if $\boldsymbol{x}(k) \not\subset \mathcal{S}$ **then**
    $\boldsymbol{u}_e \leftarrow MPC(\delta_e)$
    $\boldsymbol{u}_{max} \leftarrow MPC(\delta_{max})$
    $r \leftarrow \left\| \frac{\delta(k-1) - \delta_e(k-1)}{\delta_{max}} \right\|$
    $\Delta\boldsymbol{u} \leftarrow \boldsymbol{u}_e - \boldsymbol{u}_{max}$
    $\hat{\boldsymbol{u}}(k) \leftarrow \boldsymbol{u}_{max} + r\Delta\boldsymbol{u}$
  else
    $\hat{\boldsymbol{u}}(k) \leftarrow MPC_{Cons}(\delta_{max})$
  end if

1) $\boldsymbol{x}(k+1) \not\subset \mathcal{S} \implies \boldsymbol{x}(k+1) \not\subset \mathcal{O}$, then, the next state is safe and is outside the inflated regions as well.
2) $\boldsymbol{x}(k+1) \subset \mathcal{S}$. By the definition of the inflated regions, the system state cannot cross an inflated region in one single time step k even if $\delta(k) = \delta_{max}$. Hence, $\boldsymbol{x}(k+1) \not\subset \mathcal{O}$ and the algorithm will generate an input to drive the system outside $\mathcal{S}$.

If $\boldsymbol{x}(k) \subset \mathcal{S}$, Algorithm 1 drives the AV outside the inflated region. From (7), the MPC can always find $\boldsymbol{u}_c$ such that $\boldsymbol{x}(k+1) \not\subset \mathcal{S} \implies \boldsymbol{x}(k+1) \not\subset \mathcal{O}$ even if $\delta(k) = \delta_{max}$. Given that $0 \leq \delta(k) \leq \delta_{max}$, and $\boldsymbol{x}(0) \not\subset \mathcal{O} \implies \boldsymbol{x}(k) \not\subset \mathcal{O}, \forall k > 0$.

Let's consider now the case in which $\delta(k) > \delta_{max}$; i.e., $\hat{\delta} = \delta_{max} + \epsilon$, $\epsilon > 0$. We can rewrite (2) based on (6) as follows:

$$\begin{aligned}
\hat{\boldsymbol{x}}(k+1) &= e^{\boldsymbol{A}(t_s+\hat{\delta})}\boldsymbol{x}(k) + \int_0^{t_s+\hat{\delta}} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda \\
&= e^{\boldsymbol{A}(t_s+\delta_{max})}\boldsymbol{x}(k) + \int_0^{t_s+\delta_{max}} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda \\
&+ e^{\boldsymbol{A}\epsilon}\boldsymbol{x}(k) + \int_{t_s+\delta_{max}}^{t_s+\delta_{max}+\epsilon} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda \\
&= \boldsymbol{x}_{max}(k+1) + e^{\boldsymbol{A}\epsilon}\boldsymbol{x}(k) + \int_{t_s+\delta_{max}}^{t_s+\delta_{max}+\epsilon} e^{\boldsymbol{A}\lambda}\boldsymbol{B}d\lambda
\end{aligned}$$

which shows, as expected, that a monotonic increase in the delay implies a monotonic increase in the state of the system which may lead to a violation of the safety constraints if $\delta > \delta_{max}$, concluding the proof. □

VI. Simulation and Experimental Results

In this section, we present simulations and experimental results to evaluate the proposed online controller adaptation algorithm.

A. UGV Model

In this work we consider that the dynamics of the AV can be described by the following non-holonomic differential drive skid steering model that reflects the dynamics of our ground robot used during experiments:

$$\begin{aligned}
\dot{x} &= \frac{R}{2}(v_r + v_l)\cos\theta \\
\dot{y} &= \frac{R}{2}(v_r + v_l)\sin\theta \\
\dot{\theta} &= \frac{R}{2}(v_r - v_l)
\end{aligned} \tag{10}$$

where $\dot{x}$ and $\dot{y}$ are the $x-y$ velocity components of the vehicle. v_r and v_l are the linear velocity of the right and left side wheels of the robot, respectively, and R is the radius of the wheel. θ is the angle of the vehicle with respect to a global frame and $\dot{\theta}$ is its angular velocity.

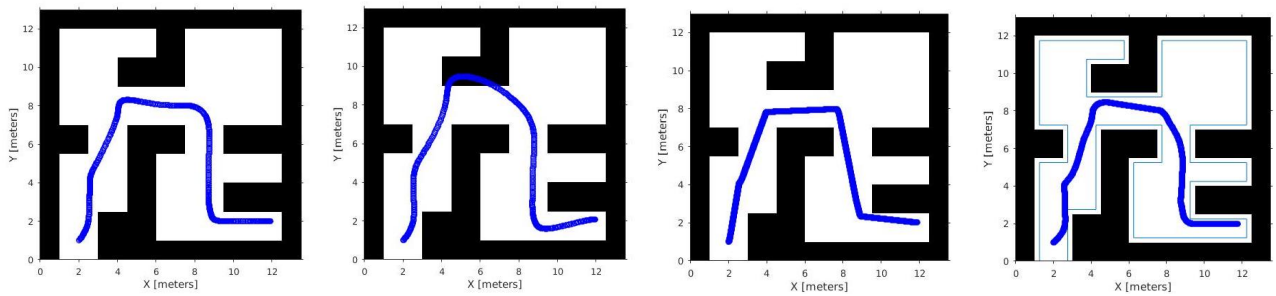
B. Simulations

In order to show the effect of variable overhead delays applied to a low-level controller of an AV, we consider a scenario in which an unmanned ground vehicle (UGV) navigates in a cluttered environment toward a desired goal. The task of the UGV is to go from a starting position of coordinate (2,1)m to a goal located at (12,2)m. We use a *Probabilistic Roadmap algorithm* (RPM) implemented inside the Matlab Robotics toolbox to generate a collision-free path from the start to the end points. This path consists of tuples of waypoints. The task of the low-level controller is to visit all the waypoints along the path until it reaches the goal. The controller outputs the acceleration of the vehicle in x and y directions. The controller is designed to run with a sampling time $t_s = 0.01$s.

Fig. 7(a) shows the scenario in which the vehicle completes the task successfully in 6.3s without incorporating any delays in the controller. The effect of cyber security protection techniques is simulated in Fig. 7(b-d) by imposing a cyclic overhead delay that varies between 0 and 10x overhead delay. We use this specific overhead function because it has a similar behavior of the CPU utilization for virus scanning described in section IV. Without adapting the controller, the vehicle starts to drift away from the collision-free path and runs into obstacles as shown in Fig. 7(b). In Fig. 7(c) we show the case in which a conservative controller is used assuming always maximum delay. The vehicle remains inside the safe regions at all times however the simulation ends up running slower taking 43.4s. Finally, in Fig. 7(d) we show the case in which the online adaptation algorithm is operating considering α = 0.7. The vehicle is able to reach the goal without hitting obstacles in 24.7s. Table I summarizes and compare the different execution times obtained in this simulation scenarios.

TABLE I
Comparison between navigation task execution times in different scenarios.

Scenario	Task Execution Time(s)
No overhead	6.3
Overhead + conservative controller	43.4
Overhead + online adaptation	18.1



(a) (b) (c) (d)

Fig. 7. Simulation Results. (a) No Overhead Delay (b) No adaptation (c) Conservative navigation (d) Adaptive navigation

C. Experiments

The experiments were run on a Clearpath Jackal UGV [8] equipped with a core-i7 CPU, lidar, camera, GPS, IMU, and wheels encoders. Experiments were performed inside our Vicon Motion Capture arena that allows precise tracking of the position and the orientation of the UGV. Our motion planning control algorithms were developed under the Robot Operating System (ROS) framework which consists of x86 applications running on top of a Linux operating system.

1) Integrating software level cyber security techniques on a real autonomous vehicle: The first experimental result that we present is the implementation of a *N*-variant system (see section IV(B)), called *Double Helix* [9]. Double Helix is used to protect our ROS nodes for mapping and navigation. The goal of this experiment is to provide an example of how we can protect a controller software implemented on a real robot and evaluate its performance in terms of overhead. Fig. 8 summarizes the process that we followed to implement Double Helix on our robot. The robot is tasked to go from an initial position to a goal while avoiding an obstacle along its path. Double Helix generates different variants of the ROS source nodes, and finally, each variant is deployed on the Jackal. Each variant is protected with different techniques picked from the Diversity Transformation Library in Double Helix. In each run (i.e., for every variant) we recorded the performance overhead imposed by the protected controllers nodes (*slam_gmapping* and *move_base* nodes) measuring delays up to 30%. Table II shows the average performance overhead imposed by the security techniques on the two nodes used for navigation operations.

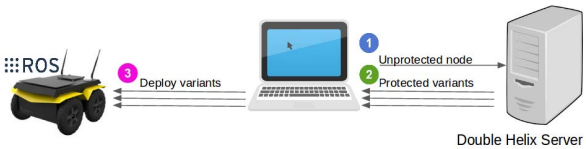


Fig. 8. Generating and deploying Double Helix variants into our ROS based UGV

2) Online Adaptation Control with Unknown Overhead: In this section we show the results obtained by running our online adaptation algorithm when large random delays are applied to our controller nodes. Note that here we introduce large artificial delays into the controllers, instead of implementing directly the techniques introduced in the previous section, because the overall overhead effects are more visible by inflating the delays to 10x, typical of other cyber security techniques introduced in Section IV. Specifically here we study the effects of these delays on our platform by running several trials with increasingly higher delays until the system starts to misbehave (i.e., it hits the obstacle).

TABLE II
AVERAGE RUNTIME OVERHEADS FOR TWO ROS NODES PROTECTED BY DOUBLE HELIX [9]

Type of Overhead	*slam_gmapping*	*move_base*
Overhead in CPU load	17.8%	12.5%
Overhead in Memory consumption	3.0%	2.56%

Without any delay, the UGV accomplishes its task in 34s. When we increase the overhead delay above 10x the UGV hits the obstacle, as shown in Fig. 9(a).

When we apply the conservative controller designed for δ_{max}, The UGV completes its task in 50s. Finally, in Fig. 9(b) we show a snapshot of the same operation running the online adaptation algorithm described in Section V with a square wave cyclic runtime overhead pattern with a maximum delay of 10x. The UGV was able to navigate safely avoiding the obstacle in 42s.

VII. CONCLUSIONS & FUTURE WORK

In this paper, we have investigated the problem of using software-level cyber security techniques to protect AV operations from malicious attacks. Specifically, we have developed and demonstrated a risk-based online adaptation algorithm to deal with overheads introduced by these security techniques. Both simulations and experimental results show that we can achieve a good performance in terms of time while maintaining safety and security constraints. Our future plan is to integrate more advanced cyber security techniques that can create overheads up to 400% on real AVs. Finally, we plan to use the proposed framework on aerial vehicles where dynamics are faster and thus adaptation is even more needed than on UGV operations.

(a)

(b)

Fig. 9. Experimental Results. A sequence of snapshots for (a) a UGV hitting obstacles due to improper input with large delay, (b) a UGV avoiding obstacles while applying the online adaptation algorithm presented in this paper.

ACKNOWLEDGMENT

This work is based on research sponsored by DARPA under agreement numbers FA8750-12-2-0247 and FA8750-15-C-0118 and ONR under agreement number N000141712012. The authors also wish to acknowledge the financial support of the University of Virginia School of Engineering and Applied Science under grant 152544.FASST.FA00250.

REFERENCES

[1] "Hackers remotely kill a jeep on the highway - with me in it," https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[2] "High-assurance cyber military systems," http://www.darpa.mil/Our_Work/I2O/Programs/High-Assurance_Cyber_Military_Systems_(HACMS).aspx.

[3] "Iran-u.s. rq-170 incident," https://en.wikipedia.org/wiki/Iran-U.S._RQ-170_incident.

[4] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 4:1–4:40, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1609956.1609960

[5] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanović, "Randomized instruction set emulation," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 1, pp. 3–40, Feb. 2005. [Online]. Available: http://doi.acm.org/10.1145/1053283.1053286

[6] N. Bezzo, K. Mohta, C. Nowzari, I. Lee, V. Kumar, and G. Pappas, "Online planning for energy-efficient and disturbance-aware uav operations," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 5027–5033.

[7] N. Bezzo, J. Weimer, M. Pajic, O. Sokolsky, G. J. Pappas, and I. Lee, "Attack resilient state estimation for autonomous robotic systems," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 3692–3698.

[8] Clearpath. Jackal (unmanned ground vehicle). [Online]. Available: https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle

[9] M. Co, J. W. Davidson, J. D. Hiser, J. C. Knight, A. Nguyen-Tuong, W. Weimer, J. Burket, G. L. Frazier, T. M. Frazier, B. Dutertre, I. Mason, N. Shankar, and S. Forrest, "Double helix and raven: A system for cyber fault tolerance and recovery," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, ser. CISRC '16. New York, NY, USA: ACM, 2016, pp. 17:1–17:4. [Online]. Available: http://doi.acm.org/10.1145/2897795.2897805

[10] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: A secretless framework for security through diversity," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267336.1267344

[11] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011.

[12] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1454–1467, 2014.

[13] J. L. Henning, "SPEC cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[14] J. D. Hiser, A. Nguyen-Tuong, M. Co, J. W. Davidson, and M. Hall, "ILR: Where'd my gadgets go?" *IEEE Symposium on Security & Privacy*, pp. 571–585, May 2012.

[15] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," in *2012 IEEE Conference on Technologies for Homeland Security (HST)*, Nov 2012, pp. 585–590.

[16] A. Kim, B. Wampler, J. Goppert, I. Hwang, and H. Aldridge, "Cyber attack vulnerabilities analysis for unmanned aerial vehicles," in *Infotech@ Aerospace 2012*, 2012, p. 2438.

[17] B. A. Kuperman, C. E. Brodley, H. Ozdoganoglu, T. N. Vijaykumar, and A. Jalote, "Detection and prevention of stack buffer overflow attacks," *Commun. ACM*, vol. 48, no. 11, pp. 50–56, Nov. 2005. [Online]. Available: http://doi.acm.org/10.1145/1096000.1096004

[18] J. F. Kurose, *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[19] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. Pappas, "Robustness of attack-resilient state estimators," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, 2014, pp. 163–174.

[20] F. Pasqualetti, F. Dorfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions onAutomatic Control*, vol. 58, no. 11, pp. 2715–2729, 2013.

[21] M. Payer, "Too much pie is bad for performance," ETH Zurich, Department of Computer Science, Tech. Rep., 2012.

[22] J. Pincus and B. Baker, "Beyond stack smashing: Recent advances in exploiting buffer overruns," *IEEE Security & Privacy*, vol. 2, no. 4, pp. 20–27, 2004.

[23] A. H. Rutkin, "Spoofers Use Fake GPS Signals to Knock a Yacht Off Course." 2014, mIT Technology Review, http://www.technologyreview.com/news/517686/spoofers-use-fake-gps-signals-to-knock-a-yacht-off-course.

[24] H. Shacham, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 552–561. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315313

[25] O. Sukwong, H. Kim, and J. Hoe, "Commercial antivirus software effectiveness: An empirical study," *Computer*, vol. 44, no. 3, pp. 63–70, March 2011.

[26] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson, "Attack models and scenarios for networked control systems," in *Proceedings of the 1st international conference on High Confidence Networked Systems*, ser. HiCoNS '12, 2012, pp. 55–64.

[27] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Computer Communications*, vol. 25, no. 15, pp. 1356 – 1365, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366402000373

[28] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problemoverview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008.

[29] A. M. Wyglinski, X. Huang, T. Padir, L. Lai, T. R. Eisenbarth, and K. Venkatasubramanian, "Security of autonomous systems employing embedded computing and sensors," *IEEE Micro*, vol. 33, no. 1, pp. 80–86, Jan 2013.

[30] J. Xu, Z. Kalbarczyk, and R. Iyer, "Transparent runtime randomization for security," in *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, Oct. 2003, pp. 260–269.