Crystal (ball): I Look at Physics and Predict Control Flow! Just-Ahead-Of-Time Controller Recovery

Sriharsha Etigowni
Rutgers University
sriharsha.etigowni@rutgers.edu

Shamina Hossain-McKenzie
University of Illinois at Urbana-Champaign
shaminahossain@gmail.com

Maryam Kazerooni
University of Illinois at Urbana-Champaign
kazerooni.maryam@gmail.com

Katherine Davis
Texas A&M University
katedavis@tamu.edu

Saman Zonouz
Rutgers University
saman.zonouz@rutgers.edu

ABSTRACT

Recent major attacks against unmanned aerial vehicles (UAV) and their controller software necessitate domain-specific cyber-physical security protection. Existing offline formal methods for (untrusted) controller code verification usually face state-explosion. On the other hand, runtime monitors for cyber-physical UAVs often lead to too-late notifications about unsafe states that makes timely safe operation recovery impossible.

We present Crystal, a just-ahead-of-time control flow predictor and proactive recovery for UAVs. Crystal monitors the execution state of the flight controller and predicts the future control flows ahead of time-based on the UAV's physical dynamics. Crystal deploys the operator's countermeasures proactively in case of an upcoming unsafe state. Crystal's just-ahead-of-time model checking explores the future control flows in parallel ahead of the UAV's actual operation by some time margin. The introduced time margin enables Crystal to accommodate operator's feedback latency by the time the actual execution reaches to the identified unsafe state. Crystal periodically queries the controller's execution state. It emulates the UAV physical dynamical model and predicts future sensor measurements (controller inputs) and upcoming feasible controller's execution paths. This drives Crystal's model-checking exploration away from unreachable future states. Crystal's selective model checking saves computational time to stay ahead of execution by concentrating on relevant upcoming control flows only. This eliminates the state-explosion problem in traditional offline formal methods. We evaluated a multi-threaded prototype of Crystal between the control station server and the UAV. Crystal was able to predict upcoming hazardous states caused by the third-party controller program and proactively restored the safe states successfully with minimal overhead.

ACSAC '18, December 3–7, 2018, San Juan, PR, USA

ACM ISBN 978-1-4503-6569-7/18/12…$15.00
https://doi.org/10.1145/3274694.3274724

CCS CONCEPTS

• **Security and privacy** → *Intrusion detection systems*; • **Computer systems organization** → *Embedded and cyber-physical systems*;

KEYWORDS

Unmanned Aerial Vehicle, Just-Ahead-of-Time verification

ACM Reference Format:
Sriharsha Etigowni, Shamina Hossain-McKenzie, Maryam Kazerooni, Katherine Davis, and Saman Zonouz. 2018. Crystal (ball): I Look at Physics and Predict Control Flow! Just-Ahead-Of-Time Controller Recovery. In *2018 Annual Computer Security Applications Conference (ACSAC '18), December 3–7, 2018, San Juan, PR, USA.* ACM, New York, NY, USA, Article 4, 13 pages. https://doi.org/10.1145/3274694.3274724

1 INTRODUCTION

The use of unmanned aerial vehicles (UAVs) has been increasing in many mission-critical settings such as surveillance, delivery systems, and military applications [2]. Consequently, they are becoming attractive targets for malicious penetrations leading to physical damage. Recent attacks took control of the drones remotely [17, 35, 40] and hacked RQ-170 Sentinel built by Lockheed Martin [41]. AnonSec team [45] obtained partial control over the global hawk used by NASA in 2016.

Secure operation of next-generation cyber-physical systems, specifically unmanned aerial vehicles will require effective *scalable formal verification* capabilities. Current static formal verification methods (e.g., TSV [23] and HACMS [30]) analyze the system model in an offline manner, often facing state space explosion, and hence do not scale up to large-scale cyber-physical systems. On the other hand, existing dynamic execution monitoring solutions (e.g., Avatar [48]) notify operators about incidents that have just occurred or are about to occur, and hence do not leave enough of a time buffer for effective manual or automated response and recovery.

To take best of both worlds, we present Crystal that leverages *Just-Ahead-of-Time (JAT)* verification (Figure 1). Crystal stays ahead of the actual system state by an arbitrary time buffer. Crystal is a nonintrusive formal verifier for drone controller programs. It speculatively checks in real-time whether the program execution may drive the drone towards any unsafe state. Drone's get there

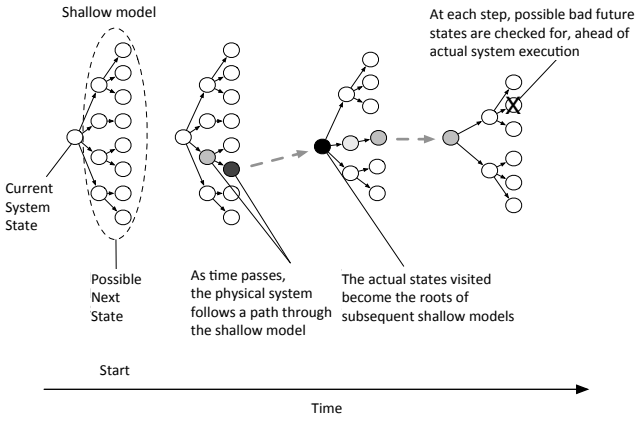


Figure 1: Just-Ahead-Of-Time Verification

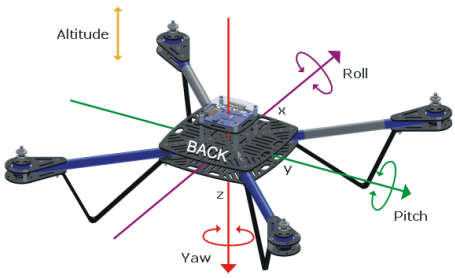


Figure 2: Drone's pitch, roll, and yaw

controller program inputs from the sensors, e.g., inertial measurement unit (IMU) or GPS and sent the output commands to drone's actuators, e.g., propeller motors. Crystal executes the control logic symbolically and calculates possible input-to-output mapping (sensor measurements to actuation commands) of the controller program. Crystal closes the loop on the physical channel by calculating how sensor measurements are determined based on the previous actuation commands and physical drone's dynamic evolution (i.e., actuation commands to sensor measurements). Crystal estimates drone's state (e.g., location and orientation) and deploys a new symbolic execution of the drone's physical dynamics to calculate its input-to-output mappings.

Given the flight control unit and physical system coupling, Crystal *pipes* its findings of IO mappings of the cyber controller and physical dynamics together to create a full closed-loop model of the cyber-physical system. The model captures all the interdependencies between cyber and physical components and is used for Crystal's just-ahead-of-time formal verification. Crystal does not generate the complete system model one-time due to its (very) large size and instead relies on local exploration and model checking of the upcoming future symbolic states up to a finite horizon based on the current system state. This enables Crystal to stay just a few steps ahead of native execution and avoids exhaustively considering all states. Crystal periodically synchronizes the model exploration process with the native execution through communication with the drone's processor and obtaining concrete values of flight controller program variables. The synchronization step allows Crystal to refine the model exploration and not explore the states that will definitely not be reached through the native execution. The model generation process continues to stay ahead of native execution and explore's the states that have not yet been reached.

Crystal inspects each upcoming state and checks its symbolic variable values to determine whether the state *could* be unsafe under a specific concretization. If the drone's native execution is about to enter the unsafe state according to its upcoming concrete input values Crystal will notify the operator about potentially upcoming unsafe state's and requests for recommended recovery response. Crystal expects to receive the operator's response before the native execution catches up. This waiting time denotes how far Crystal's JAT exploration leads the native execution and could be adjusted initially. Crystal deploys the recommended countermeasure if the unsafe state is actually realized, and caches it to not involve the operator again for future similar situations.

The contributions of this paper are as follows:

- We present a scalable formal verification technique, just-ahead-of-time verification, for complex UAV platforms that eliminates state explosion problem and leaves time for the operator to select an appropriate countermeasure strategy.
- We present a cyber-physical symbolic execution framework that leverages programming language analysis and enhanced drone state estimation techniques to create cyber-physical models for formal verification purposes.
- We present a predictive hybrid model with data-driven and system knowledge model of the drone's physical dynamics using a neural network and extended Kalman filter (EKF) that takes actuation commands (flight controller outputs) as inputs and outputs the predicted sensor data (controller's next input).

This paper is organized as follows. Section 2 gives an introduction to drones dynamics and a previous related work [23]. Section 3 lays out our assumed threat model and gives an overview of Crystal's architecture. Section 4 explains the predictive model based on a hybrid approach of a neural network and EKF model to formulate the drone physical dynamics. Section 5 describes how the drone's controller code and physical dynamical formulations are integrated into a unified cyber-physical model. Section 6 explains the real-time formal verification and recovery using the cyber-physical models. Section 7 describes our prototype implementation and evaluation results. Section 8 covers the related work, and Section 9 concludes the paper.

2 BACKGROUND

2.1 Drone Flight Dynamics

Since the motion of UAV is in three-dimensional space, it can be controlled along three axes. The altitude of the drone is proportional to and controlled by the thrust produced by the propellers from the four motors. For instance, the thrust on all the motors should be the same to move the drone just in the z-axis. In order for the drone to hover (stay at a fixed location in air), all the motors should produce a thrust to neutralize gravity. To balance the rotational torque produced by the motors and to increase the stability of the

drone, motors on the opposite direction rotate in the clockwise direction and the ones adjacent to these rotate in the counterclockwise direction. The rotation of the drone along the x-axis is called *roll*, along the y-axis is called *pitch*, and along the z-axis is called *yaw* (Figure 2).

The drone's left-right motion can be controlled by changing its roll. If the drone has to move towards left (right), the thrust on the right (left) motors is increased. Similarly, the drone's front-back motion can be controlled by changing the pitch and changing the thrust on front or back motors. Drone's direction along the z-axis is controlled by changing the yaw. If the drone has to rotate clockwise, the thrust on the motors rotating counterclockwise has to be reduced. The aforementioned parameters allow defining the drone's state notion as a six-entry vector of its location (x,y,z), and orientation (roll, pitch, yaw). Given the drone's current state and the latest issued actuation commands to the motors, the drone's next state can be calculated based on its physical dynamic models. In this paper, we used a combination of extended Kalman filter and a data-driven model of the drone's physical dynamics to predict its future states.

2.2 Offline Controller Code Verification

Due to model error and random disturbances, drones obtain the aid of flight control unit to constantly monitor and regulate the flight operations. The controller repeatedly takes measurements of the process state and feeds these to its software *control logic* to determine what changes need to be made to keep the process on course. This sense-execute-actuate loop is called the *scan cycle* and may occur many times per second.

In most commercial-grade flight control units, the control logic can be modified remotely, exposing the threat of malicious logic injection. Authentication required for modifying flight control logic is often weak or does not exist, e.g., no authentication for Bitcraze [5]. Hardcoded backdoors are a common industry practice [37]. Modification of a flight control logic grants the attacker complete control of the physical drone operation. As demonstrated by the well-known Stuxnet attack [9], the malicious control logic can also forge sensor data reported to human operators, thus hiding the first signals of malicious behavior.

We review the most related recent work on offline controller code verification [23]. The trusted safety verifier (TSV) [23] is interposed between controllers (i.e., programmable logic controllers - PLCs) and the control network (i.e., supervisory control and data acquisition - SCADA). Any controller-bound code must be formally verified against a set of safety properties. The safety properties are stated in linear temporal logic (LTL) that allows for the description of temporal properties such as causal relationships, and guarantees of eventual progress [11]. An LTL property contains a set of atomic propositions that are non-temporal property typically stated in propositional logic, e.g., `landed` = (`altitude` = 0) $\wedge$ (`drone_velocity` = 0). The LTL property then combines these atomic propositions using temporal connectors to describe relationships over time, e.g., `safe_landing` = $\neg$`motors_off` until `landed`.

Cyber-physical system controllers typically follow synchronous execution paradigm as a sequence of fixed-time sense-process-actuate epochs called *scan cycles*. TSV performs a mixed concrete and symbolic execution of an ILIL program to produce a *symbolic scan cycle*. A symbolic scan cycle represents every possible execution of a single scan cycle of the controller program. It is a mapping from path constraints over controller input variables (sensor measurements) to symbolic values for PLC output variables (actuation commands). For a single entry in the mapping, if the input variables satisfy the path constraint, then the output variables will have the corresponding symbolic value.

To model the PLC execution's subsequent scan cycles, TSV combines successive symbolic scan cycles into the Temporal Execution Graph (TEG). The TEG is a tree that represents a nondeterministic execution of a controller program for some fixed number of scan cycles. For example, a TEG of depth three would represent all possible executions of the program for three successive scan cycles. The TEG is later checked against the safety requirements to determine if the controller code could ever produce unsafe outputs.

2.3 Limitation of Existing Solutions

By design, offline verification solutions [18, 23, 33] should complete before the code is allowed to run on the controller. Due to the exhaustive exploration of *all* possible states and too many possibilities, existing formal solutions face state-space explosion and do not scale up to real-world complex cyber-physical systems [23].

Additionally, TSV's analysis of subsequence scan cycles considers all controller inputs (sensor measurements) as free variables that can take on *any* value. This results in too-pessimistic outcomes (i.e., the code rarely satisfies all safety properties) and deteriorates the solution scalability. The physical dynamics of the drone determines the sensor values given its recent actuation commands. For instance, the actuation command "increase the propellers rotation speed" results in an increase in altitude sensor value. As discussed later, the models of the physical dynamics along with the controller code models can be used to predict and investigate the drone's future behavior.

3 OVERVIEW

3.1 Threat Model

The threat to the UAVs can be from one or more of the following attack vectors [27]: physical (attack on sensors, actuators), cyber (firmware, controller software, guidance and navigation algorithms) and communication (radio link to ground control station). One of the most prominent causes for recent drone security failures is remote cyber attacks [17, 35] that Crystal aims to protect against. We assume that the underlying software stack (e.g., operating system or firmware) and hardware are trusted, while the control logic, guidance and navigation algorithms on the drone's flight control processor can be malicious. By ahead-of-time verifying that control logic will not violate safety properties, Crystal protects against arbitrary control logic injection on the controller. Crystal does not defend against physical attacks since they are arbitrary. Crystal does not defend against sensor channel attacks, where sensor data is forged [21, 43]. In such a case, the control logic may behave exactly as intended, but on false sensor data. Such attacks are outside the scope of this paper and must be addressed by improved state estimation techniques [21].

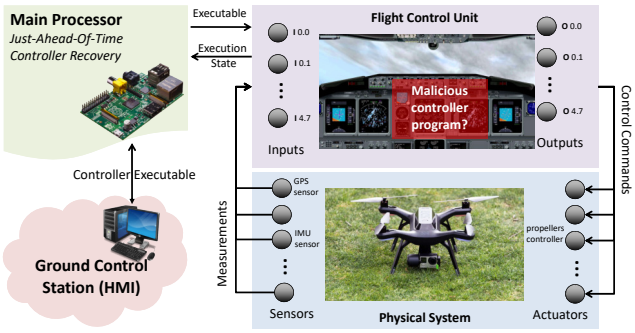


Figure 3: Crystal's High-Level Architecture

3.2 Crystal Architecture

Figure 3 shows Crystal's context and its interactions with other control assets. Drones typically follow a dual processor architecture. The flight control unit processor takes care of the real-time sense-process-actuate executions; it receives sensor measurements, runs its control logic, and sends the output values to the motors. The main processor interacts with other peripheral devices, e.g., to communicate with the ground control station. All the control logic updates to the flight control unit from the external world go through the main processor. Crystal runs on the main processor to monitor each control logic update and its execution on the flight control unit processor.

3.3 Safety Requirement Definition

Before Crystal's setup, the drone operator needs to define a set of high-level safety properties for the underlying physical dynamics. The safety requirements do not have to correspond to specific actuator outputs and could be defined for global system parameters. For instance, if the controller has to maintain the propeller speeds around a fixed set-point, the safety requirements can be defined to limit the whole drone's acceleration. Crystal's consideration of the physical system dynamics enables automated correlation of global system parameters to individual actuation commands.

Previous protection solutions (e.g., TSV [23]) fully ignore the physical dynamics of the underlying system, and hence pose strong requirements for the operators: *i)* flight dynamics expertise: all safety requirements should be defined for only the target flight control unit's output values; hence, in the example above, the operators have to analyze the flight dynamics to determine what flight control unit's output values (propeller set-points) would cause unacceptable accelerations of the quadcopter; *ii)* tedious human involvement: the operators should redefine/update the safety requirements every time the parameters of the components change due to external environmental factors, which occurs often in practice.

3.4 Predictive Flight Modeling

Crystal enables hybrid cyber-physical symbolic execution by complementing flight control code analysis [23] through its predictive modeling of the underlying flight physical dynamics. The drone's physical state (i.e., location and orientation) is estimated and predicted using a hybrid approach of neural network and EKF that is already trained and configured to formulate the flight dynamics. The predictive trained hybrid model will estimate the values of the sensors in the future given the upcoming actuation commands. The upcoming actuation commands are calculated by Crystal's analysis of the drone's controller code. During the drone's operation, the hybrid predictive model adaptively corrects itself using a sliding window on the most recent sensor values. Later in the paper, we extend to multiple parallel actuation commands (e.g., power control of the drone's four propeller motors), where the actuation is modeled as a symbolic vector (section 4).

3.5 Just-Ahead-of-Time Verification

Crystal is deployed on the drone's main processor and intercepts every control logic updates bound to the flight control unit processor. Crystal disassembles the binary and starts the dynamic conversion of the resulting source to its corresponding finite state automaton. During the conversion (model exploration), Crystal implements on-the-fly formal model checking in parallel to ensure that the reachable states are safe and do not violate the safety requirements.

In practical situations, the conversion does not complete within a reasonable time due to the state explosion problem. After a predefined waiting time (so-called *time margin* t_m which could be varied) and most likely before it completes its formal verification, Crystal uploads the control logic on the flight control unit. The control logic starts its native on-device execution, while Crystal, in parallel, is exploring and verifying its future states, called just-ahead-of-time (JAT) verification. The future states being explored are initially ahead of the native execution by t_m. Higher the value of t_m lower is the accuracy.

Assuming that JAT's speed is not lower than the flight control unit's native execution, Crystal maintains the time margin. That is any state $\mathcal{S}$ visited by JAT at time $\mathcal{T}$ will not be visited by the native execution sooner than $\mathcal{T} + t_m$. Otherwise, if Crystal gets behind the actual execution, Crystal's outputs would be useless, because the operators would get notified about the *actual* adversarial consequences of the malicious control logic before Crystal could analyze them symbolically. It is noteworthy that JAT exercises all the paths, while the native execution goes through a single path only. Therefore, the native execution may never visit that specific state $\mathcal{S}$, because it may take a different execution trace.

The time margin between the symbolic model exploration and the flight control unit's actual execution enables the operators or automated response systems to take the time deciding upon an appropriate recovery strategy in case an unsafe state is visited during the future state speculations. We will not focus on the automated response selection, and consider it outside the scope of this paper.

4 DRONE PHYSICS MODELING

To ensure effective JAT drone safety monitoring, Crystal must be able to quickly model and reason about its flight's physical dynamics, it's current and future behaviors when the upcoming actuation command sequences are given. The model outcomes are later merged with flight control code analysis to analyze the controller code execution impact on the flight dynamical operation and drone safety. Crystal uses the hybrid model for its formal JAT

verification to determine if the controller execution can drive the drone towards unsafe situations.

4.1 Normal Operation Mode Physical Modeling

The estimation of the future sensor values during normal operation mode is computed using an extended Kalman filter (EKF). The extended Kalman filter is a nonlinear state estimation algorithm that uses data containing noise and inaccuracies collected from a series of observations over time and estimates the unknown states. The states of the system are the physical parameters (acceleration, altitude, pitch, roll, yaw) which are obtained by inertial measurement unit sensors on the drone. The observations are the values given to the actuators like the PWM signal to the motors of the drones. The EKF is one the popular technique used for future sensor data estimation for a nonlinear system like drones. The EKF linearizes the flight dynamics equations at each step and applies Kalman filter technique on the linear system. The system equations which are required for the algorithm are derived from the physical dynamics hence in order to use the EKF algorithm for state estimation, the system dynamics (flight dynamics) has to be known. Detailed equations of the model are described in Appendix B. The estimation of sensor data K steps ahead of time is given by Equation 1

$$\hat{x}(n+k|n+k-1) = f(\hat{x}(n+k-1|n+k-1)), u(n+k-1)) \quad (1)$$

The values for (K - m) step are evaluated based on (K - m - 2) estimation. The EKF predicts the sensor values K steps ahead of time and then updates it if the error is large after every scan of the sensor data and updates equations to minimize the error from the actual sensor data.

4.2 Failure Mode Data-Driven Modeling

Crystal leverages a data-driven physical flight dynamics modeling using neural networks for the estimation of sensor values during abnormal conditions such as failure modes. The trained model's inputs are the drone's current state (sensor measurements) and upcoming sequence of actuation commands calculated by the flight controller code analysis. The neural network outputs the next sequence of drone's flight dynamical state (location and orientation), and hence the upcoming sequence of sensor measurements.

Since the sensor data is collected from the physical system, the data is continuous and has some fixed slew rate with respect to the actuator actions. The sensor data varies continuously and does not change abruptly although it is collected in a discrete manner. For instance, during the takeoff, the drone's altitude increases from 0 to some $\mathcal{H} \gg 0$ gradually and does not jump $0 \rightarrow \mathcal{H}$ instantaneously. The change rate (e.g., $\mathcal{T}$ per second) is limited by the drone's physical limitations, e.g., maximum thrust by its motors. Since the sensor data is continuous, Crystal leverages the spatial features of the measurements by using convolutional neural networks. The model uses convolutional layer, pooling layer, flattening layer, and a dense layer. The applications of different layers are explained below.

Convolutional layer. It is the core of the neural network. Crystal obtains the sensor data from the flight controller memory and uses them as inputs to the convolutional layer through an input layer. Each convolutional layer has four learnable filters and the window size is fifty. In our experiments, we empirically found these numbers to optimize the trade-off between hard-to-train accurate huge networks and inaccurate easy-to-train small models. Each filter performs a convolution (dot product) between the drone input sensor data and the values of the filter data.

The filters are learned to activate when they observe specific features on input sensor data such as increasing, decreasing or fluctuating between values based on the time and state of the drone dynamics. These dot product from a filter and input data produces outputs called activation map. Since Crystal uses four filters, it produces four activation maps. These maps are stacked to produce output volume that feeds the next layer. The rectifier linear units are used along with the convolutional layers as the activation function.

Pooling layer. It is periodically inserted in between the convolutional layers reduce the over-fitting by control reduction of the spatial size. In convolutional layers, the window slides through all the drone sensor data in the strides of one. So, it has a lot of spatial data which leads to large spatial size and high computation apart from the overfitting. Crystal uses pooling layers to reduce the spatial size and hence over-fitting. It makes use of one-dimensional pooling layer after each convolutional layers.

Flatten layer. Crystal uses this later to flatten the activation maps and get a single dimensional data. Flatten layer is used to merge all the four activation maps to a single dimensional activation map.

Dense layer. These layers are connected to all the activations from the flattening layer to the output layer. The dense layer is a conventional neural network with weights and biases.

The drone sensor data has a lot of spatial locality in nature due to the continuity of the underlying flight physical dynamics and how the drone's physical state evolves over time. Therefore, the neural network just uses previous finite $\mathcal{N}$ sensor data samples to predict the future sensor data i.e it does not depend on the whole flight history. Similar to the EKF even the neural network model predicts K steps ahead of time, based on the (K - 1) prediction. During the runtime, the predicted (K - 1) steps are continuously compared against the actual values from the sensors and updates the values.

4.3 Full Flight Operation mode

The system knowledge is essential for the prediction of sensor data by using EKF. In most of the scenario's the system model is not complete due to missing few modeling parameters due to lack of complete knowledge, approximating few parameters during modeling or due extreme nonlinearities. The system model parameters will also not hold true over the lifetime of a system due to the changes caused to the parameters by wear and tear or damage, replacement of old or damaged part with newer once. Apart from the difficulties of modeling of the system, EKF also takes few iterations to converge from the initial estimated state values by correcting itself based on the noise parameters. If the change on sensor values are random and sporadic then EKF takes many iterations to correct itself whereas the neural network takes only a single pass. Due to the linearization of the system equations in each step, if the system is modeled incorrectly or if the initial state estimation is wrong then the algorithm quickly diverges. On the other hand, the neural network does not know the system dynamics initially and

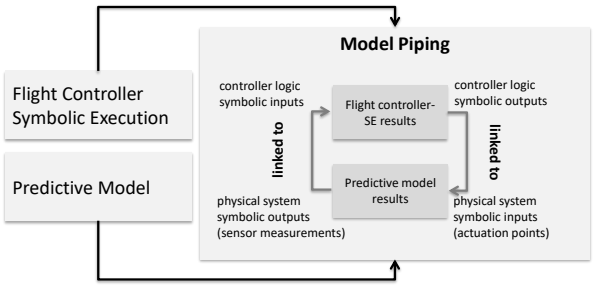


Figure 4: Hybrid Cyber-Physical Symbolic Execution

has to be trained and re-designed using a large set of data considering different scenarios. The neural network's model performs well if the scenario's which are trained but not so well on a newer strange scenario. Since the neural networks lack the physical system knowledge, they are weaker in self-adaptability compared to the EKF. Hence a hybrid approach is used leveraging the advantages [42] of both the models depending on the sensor data.

The hybrid estimation model is used at runtime for Crystal's JAT verification to predict future drone states (location, orientation) over next several scan cycles and sensor data that feed the controller's future executions. The time margin between the current concrete execution and the predicted sensor data allows for Crystal's ahead-of-time analysis and timely notification of the operators about potentially upcoming unsafe states. Crystal collects upcoming actual sensor measurements (coming from actual execution) and use them to correct the predicted data. This feedback loop enables Crystal to update the hybrid model about the drone's actual physical dynamical evolution, and improve the next predicted sensor measurements.

5 CYBER-PHYSICAL SECURITY MODELING

Crystal combines the outcomes of the controller code symbolic execution and the flight physics models (i.e., neural network). The resulting hybrid symbolic model captures the dynamics of both the cyber flight controller and the underlying physical flight dynamics. This allows Crystal to analyze how the (malicious) controller code execution affects the drone's physical operation and safety status.

The flight controller and physical dynamics are interconnected through sensing and actuation channels (Figure 4). The flight control unit's outputs feed actuators on the drone, and the flight control unit's inputs are fed by the drone's sensors. Our calculated symbolic hybrid model formulates how the flight control unit's outputs and inputs are interdependent based on the flight dynamics. The flight controller code models encode how actuation commands are calculated based on sensor measurements. On the other hand, physics models (EKF and neural network) close the loop and determine the next sensor measurements based on the actuation commands.

Crystal generates the hybrid models through the following four phases. First, Crystal translates the drone global safety requirements to the flight control unit's actuation output constraints automatically. Second, Crystal analyzes the flight control unit's code to determine the primary constraints on its inputs that guarantee the above-mentioned output constraints if the code executes. Third, Crystal emulates the flight dynamics symbolically using the above-mentioned flight control unit's output constraints and calculates a secondary set of constraints on the flight control unit's inputs. Finally, Crystal uses formal theorem provers to prove the drone safety; Crystal verifies that the flight control unit's input value space defined by the secondary constraints is a subset of the input space defined by the primary constraints. Otherwise, there will be flight control unit's inputs that violate its output constraints and hence the drone's safety requirements. In this section, we explain these four steps in more details using a running example.

drone requirements→controller output constraints. Crystal requires the *global* safety requirements for the drone's physical dynamics. The global requirements impose constraints on the parameters that are often different from those directly controlled by the flight control unit's outputs. For instance, the global requirements may state "the drone should not accelerate on z-axis by more than Xms^{-2} " while the flight control unit's output controls the amount of power fed to the propeller motors. The z-axis acceleration depends on the propellers power feed according to the flight dynamics of the drone. Crystal uses its drone physics models (section 4) to calculate the flight control unit's output constraints given the drone's global safety requirements. Crystal uses the predictive model along with the hybrid symbolic execution to calculate the range of the flight control unit's output values that ensure the global requirements are satisfied.

Flight controller output constraints→controller's primary input constraints. Flight controller code produces actuation outputs given its sensor data inputs. Crystal calculates the ranges of the controller inputs that ensure output values satisfying the constraints calculated in step a. Crystal executes the control logic symbolically and computes the corresponding input constraints and symbolic output values for feasible execution paths. Crystal calculates the controller input constraints as the logical conjunction of the path condition predicates and the calculated controller output constraints (step a). Consequently, Crystal obtains input predicates for feasible control logic execution. The calculated input constraints represent the permissible drone sensor value ranges to ensure that the controller's actuation outputs will not cause the drone's global safety property violations.

Controller output constraints→secondary input constraints. In step b, we discussed how Crystal transforms the controller output constraints to its input constraints by analyzing the cyber channel (i.e., the controller code). In this step, Crystal performs the same transformation (the controller output to input constraints) but considering the physical channel (i.e., flight dynamics). The controller's actuation outputs lead to the drone's physical state change (dynamical evolution) indicated later by the updated sensor measurements back to the controller. Crystal assumes the controller outputs comply with the constraints (calculated in step a) and emulates the flight dynamics and its evolution over time using the trained neural networks. Crystal calculates the resulting constraints on the upcoming sensor measurements. We call the calculated controller input constraints *secondary* constraints to differentiate them from the constraints calculated in step b.

Formal proof of the drone safety. So far, we have calculated two sets of flight controller input constraints: *i)* the primary constraints for the controller code that ensure the flight controller execution (cyber dynamics) will generate outputs that do not violate the drone safety, and *ii)* the secondary constraints that, if violated, indicate the controller outputs must have taken on values (based on the flight dynamics) that do not satisfy the global safety requirements. To guarantee the compliance with the global requirements, the value space defined by the secondary constraints should be a subset of the space defined by the primary constraints; otherwise, the controller code execution could possibly result in outputs that violate the global safety requirements. Consequently, Crystal checks for the above-mentioned relationship. Crystal negates the primary constraint for each execution path and calculates it's conjunction with the secondary constraint predicate. Crystal checks whether the ultimate predicate is satisfiable. If not, the plant is marked as verified. Otherwise, the global requirements could be violated, because there would be a concrete value set that satisfies the negated primary constraint (violates the primary constraint) and satisfies the secondary constraint. This would be equivalent to the primary constraint being a subset of the second constraint; therefore, the cyber-physical flight control unit is not verified (is either buggy or malicious) and its execution can lead to unsafe drone states such as a physical ground crash.

6 JAT VERIFICATION AND RECOVERY

In the previous section, we described how Crystal develops a hybrid verification of the drone operation. However, the above-mentioned analysis does not consider the drone operation across subsequent sense-process-actuate scan cycles (subsection 2.2). The flight control code symbolic execution (section 5: step 2) goes through individual execution paths of the control logic just once, whereas the flight controller's actual execution immediately starts the next scan cycle based on the updated sensor inputs right after the current one finishes. The control logic often leverages stateful variables such as timers and counters that retain their values across successive cycles; this causes inter-cycle output value dependencies.

Just-ahead-of-time analysis. To address subsequent cycles, Crystal explores possible symbolic hybrid states of the drone and creates the corresponding state-based finite state automaton. Each symbolic hybrid state captures the symbolic values of the controller code as well as the flight dynamics parameters. Each state transition in the automaton represents a new cycle. The first state represents the drone right after it is turned on. The increasing depths within the automaton represent subsequent scan cycles, and the number of outgoing transitions from each state (branching factor) represents the number of feasible execution paths in the flight controller code. As the result, Crystal considers individual control logic executions and the corresponding flight dynamical evolution and creating the corresponding automaton state. Crystal performs the formal verification steps (section 5: step 4) for that state specifically to ensure the drone safety in that state.

The main problem with the algorithm above is scalability due to the exponential growth of the automaton size on each depth. The traditional alternative to offline formal analysis is the runtime methods to detect the unsafe states as the system enters those states dynamically. In cyber-physical real-time drone operations, runtime solutions are not sufficient generally, because they often report safety violations too late for a timely response and recovery countermeasure. For instance, a runtime monitor for a drone may report "the drone is about to hit a pedestrian". That is too late for the operator or automated decision maker to spend time picking a response and carrying it out while the drone (physical world) has inertia and hence will keep moving. Crystal implements just-ahead-of-time (JAT) verification.

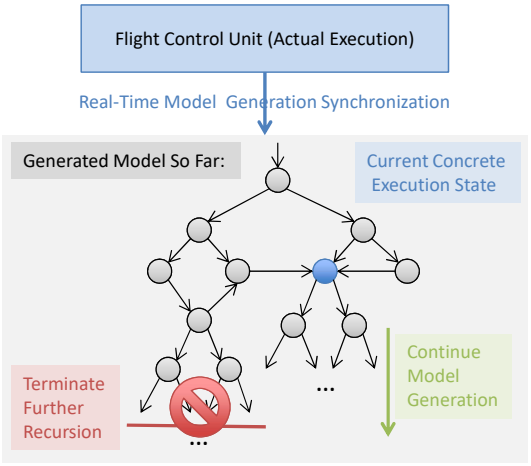


Figure 5: Model Generation, Refinement, and Checking

Crystal starts symbolic exploration and safety verification of the automaton states as discussed above. After a predefined time margin t_m, Crystal launches the flight control unit's actual execution in parallel, while the depth exploration is maintained at (at least) the same speed as the concrete execution, and hence it stays t_m time units ahead of the actual execution permanently. Figure 5 shows JAT in operation, where the automaton has been explored and created two depths ahead of the current system state. Crystal avoids exploration of the states that are not reachable from the system's current concrete state.

Human-assisted drone safety recovery. Crystal enables human-assisted system recovery through its fast enough JAT verification to maintain the time margin t_m between JAT and the flight controller native execution. If JAT encounters a potentially unsafe future state, Crystal asks for the operator's recommendation. A potentially unsafe state represents the existence of a feasible execution path from the flight controller's current native execution state to a future point where the drone safety requirements are violated. The violation would be caused by the flight controller actuation outputs that change the critical flight parameters.

The violation may be because of a malicious or buggy flight controller program. Crystal facilitates such a hybrid cyber-physical reasoning about the drone operation through its model piping (Figure 4). Due to the model's symbolic state variable values, Crystal uses a formal SMT solver to determine the possibility of safety requirement violation in every encountered future state through one of its concretizations. Crystal expects the operator's feedback

within t_m, otherwise, the native execution catches up and may enter the same unsafe state, while Crystal is not yet given with the optimal response action. In those cases, the controller takes a default non-optimal safe response action that is set initially before the controller program execution launch time.

The operator's recommendation within t_m could be any sequence of the following: *i)* to modify or read any control logic variable value on the flight control unit dynamically; *ii)* to inject an instruction (e.g., call a recovery subroutine) on the running control logic; *iii)* to upload a new control logic (possibly a safe controller following the Simplex paradigm [39]); and *iv)* to halt the flight controller's dynamic maneuvers so that the drone enters the hover mode. Crystal implements the operator's recommendation immediately if the native execution enters the target symbolic state *and* the state's current *concrete* variable values actually violates the safety requirements. Regardless, Crystal stores the operator's recommendation to avoid inquiring the operator again later for similar scenarios, where JAT encounters the same or a symbolically equivalent state.

Optimization for practical feasibility. To ensure that JAT will keep up with the native execution speed, we leverage various drone architectural features and implement several optimization techniques: *i)* symbolic execution. Before the runtime setup, Crystal implements an offline symbolic execution of the flight controller code to avoid exploring individual concrete traces with similar outcomes in terms of the compliance with the safety requirements. Additionally, the offline symbolic execution enables Crystal runtime model exploration to consider only feasible control logic execution paths and not waste its online analysis time on unreachable (infeasible) states. The use of symbolic execution significantly reduces JAT's search space and improves its runtime performance. Crystal then pipes the cyber- and flight physics-side analysis results (input-to-output mappings) to use a full cyber-physical system model for JAT verification (Figure 4). *ii)* runtime model pruning. Any time the native execution takes (or does not take) a branch at time t, a subset of the future states at time $t + t_m$ that JAT is about to explore may become unreachable. To maximize its time utilization, Crystal periodically investigates the native execution state and prunes the unreachable model states accordingly. Crystal's runtime model pruning gives an exponential speedup and eliminates the exponential recursive branching of the model states during the exploration. Such exponential branching leads to the impracticality of the previous formal method solutions [23]. It leads to a finite state sub-space within JAT's forward-sliding t_m window over the complete model's state space (Figure 1). *iii)* parallel JAT. Given the exponential speed up using the runtime model pruning, Crystal's exploration is accelerated further through its multi-threaded implementation theoretically up to the point that JAT can complete verification of every model depth within an execution-cycle interval. In our implementations, Crystal achieves this objective for complex controller programs using a quad-core machine. *iv)* physics-aware flight dynamics prediction. The neural network created is trained offline and the model is used for the prediction purposes to improve the analysis performed. Crystal predicts the drone's physical state and hence limits the upcoming possible sensor input values to the control logic. The introduced input constraints mark many originally-feasible control logic execution paths as infeasible saving analysis time.

7 EVALUATIONS

We have implemented and evaluated Crystal on two commercial products *i)* 3DR Solo Quadcopter [1] and *ii)* advanced Siemens S7-300 PLC controller [9]. We designed the experiments to evaluate the following questions: *i)* How accurately and fast can Crystal intercept and reverse engineer the flight controller-bound machine codes between the ground base station and the drone's flight control unit processor? *ii)* How efficiently do the proposed symbolic hybrid cyber-physical system analysis techniques calculate the formal sensing/actuation constraints? and *iii)* Can Crystal perform JAT verification efficiently for real controller programs?

We have taken three steps to optimize our implementations to ensure Crystal maintains its exploration ahead of the native execution: *i)* symbolic execution enables Crystal to consider similar multiple concrete states through a single symbolic state analysis; *ii)* periodic acquisition of native execution state and the predictive modeling of the drone's physical dynamics using a hybrid approach of convolutional neural network and EKF enables Crystal to avoid wasting verification time on unreachable future states; *iii)* multi-threaded future state exploration and verification gives Crystal's performance a linear boost. Our experimental results are promising and show that Crystal can detect violation of drone safety properties and recover the safe operation successfully.

7.1 Evaluation on 3DR Solo Quadcopter

Implementation. We implemented Crystal on a Raspberry Pi 3 embedded computer running Linux kernel 4.4. We used Keras with TensorFlow as a backend to implement the predictive neural network. The extended Kalman filter was implemented in C/C++ code. The Z3 theorem prover is used by Crystal for checking the feasibility of paths and simplifying the symbolic outputs obtained during symbolic execution. The sensor data consists of flight data collected in stabilize, acro, drift, altitude hold, position hold and loiter modes in various scenario's including a couple of crash scenarios.

We implemented Crystal on a 3DR Solo quadcopter. The 3DR Solo has a flight control unit which is isolated from the external world for programming and an i.MX6 solo processor from Freescale which runs Linux based operating system 3DR Poky (based on Yocto project reference Distro) and has connections with the external world as well as the flight control unit. The Raspberry Pi can communicate to the flight control unit through UDP. The Raspberry Pi sends MAVLink [24] packets to request the sensor data from the quadcopter. This sensor data is feed to the predictive model to obtain the predicted sensor data. This predicted sensor data is used to prune the TEG and determine if there is any unsafe state in the near future and inform the operator about it. The flight control unit has a relatively less powerful processor (168 MHz / 252 MIPS Cortex-M4F) compared to i.MX6 Solo (ARM Cortex A9 1Ghz, 1 CPU core with VPU and GPU) and Raspberry Pi 3 (1.2GHz 64-bit quad-core ARMv8 CPU), Hence the implementation for JAT verification will not lag behind the flight control unit's actual execution.

Case Studies: 3DR Solo Quadcopter Control Attack.

- **Control attack on Motor and servo control**

`Safety requirements`: The angular velocity of motors on the quadcopter should be within safety limits. $\{P\}C\{Q < \gamma\}$ where P is the current angular velocity, C is the next command and Q is the resulting angular velocity which must be less than γ boundary condition for safe operation. This limit changes with respect to the mode of operation. eg.., the rate will be a bit relaxed when the quadcopter is in acro mode compared to stabilize mode.
`Safety violation`: We implemented a malware (Maldrone) to crash the quadcopter while landing. The Maldrone we developed will decrease the thrust on the quadcopter significantly while landing leading to crashing the quadcopter. These kind of attacks are difficult to detect by humans if the quadcopter is flying at a distance away from the operator.
`Violation detection`: The predictive model predicts the significant reduction of the thrust on the quadcopter ahead of time, based on the current and past reduction rate of the thrust. Crystal estimates the possible unsafe state in future and informs the operator about the unsafe state. We were able to receive the information about the unsafe state well in advance by using Crystal. Figure 6 shows the predicted value before the crash which could not be detected until the crash had occurred without Crystal.

- **Control attack on AHRS**
  `Safety requirements`: The prediction value on the flight control should not deviate from the actual sensor value by more than γ (The safe operation range).
  `Safety violation`: The synthetic malware which disrupts the estimation of the EKF was introduced into the control algorithm. This malware modifies the EKF algorithm so that the estimation values do not follow the safety requirements.
  `Violation detection`: Crystal predicts the upcoming unsafe conditions and informs the operator about the situation. Figure 7 shows that the Crystal could predict and inform the operator.
- **Drifting due control attack on PID control**
  `Safety requirements`: The quadcopter should not drift more than a certain tolerance which depends on the accuracy of the sensors and the external environmental factors. The drift considered over a cycle as well as over an accumulated period of time.
  `Safety violation`: The PID parameters of the motor are changed due to which the adaption of the change in the motor speed was not accurate. Hence a lot of drift was introduced into the system.
  `Violation detection`: The predictive model detects this drift caused due to change in PID parameters and informs the operator about the violation.
- **Control Attack on altitude control**
  `Safety requirements`: The quadcopter should not change its altitude by more than the tolerance range. The tolerance is due to external environmental factors like wind and/or due to the accuracy of the sensors.
  `Safety violation`: The synthetic malware was introduced into the system which changes the speed of rotation of the motors (angular velocity) and changes the altitude of the quadcopter. The malware injects false data into the control algorithm.
  `Violation detection`: The predictive model detects this change in altitude by more than the tolerance value in altitude hold mode and informs the operator about the unsafe conditions. Since the firmware is trusted Crystal gets its sensor values from the firmware, hence false data injection on the control algorithm could be easily detectable by the Crystal.
- **Control Attack on position control**
  `Safety requirements`: The quadcopter should hold its position and not change the position by more than tolerance. Again the tolerance might be due to external environmental factors such as wind.
  `Safety violation`: The synthetic malware was introduced to change the position of the drone by more than the tolerance range. The malware can inject false data of either GPS and /or IMU of the sensor data to the control algorithm.
  `Violation detection`: The predictive model detects the change in the position greater than the tolerance range and informs the operator about the unsafe condition.

Table 1: Average mean absolute error (MAE) for extended Kalman filter (EKF) and neural network (NN) model during minimal and heavy transitions

Sensor Data	NN MAE	EKF MAE
Roll during minimal transition	1.3242	0.1136
Roll during heavy transition	0.1713	0.8268
Yaw during minimal transition	1.9644	1.6359
Yaw during heavy transition	3.5643	18.5647

Accuracy. In Figure 8a and Figure 8b the estimation of sensor data is better by using EKF for gradual and smooth transitions of the sensor data since EKF is aware of the physical dynamics much more accurately than neural network. However, in case of heavy and violent transitions, the neural network is better than EKF as it just takes one pass whereas EKF takes few iterations to converge to the actual data based on the noise parameters used. Table 1 shows the results for the accuracy during smooth transitions and violent transitions. The error is represented using mean absolute error. Mean absolute error (MAE) is the average error between the predicted value and the absolute value.

$$Mean\ Absolute\ Error(MAE) = \frac{1}{n}\sum_{t=1}^{n}|e_t| \tag{2}$$

Figure 9 shows the MAE for different iterations (epochs) of learning the sensor data. The learning data converges between 10 to 15 iterations (epochs). Hence Crystal's neural network uses 12 epochs for the experiments. The false positive rates due to the predicted sensor data are shown in Figure 10.

Performance. The performance of the EKF and predictive neural network running on a Raspberry Pi 3 is shown in Table 2. The analysis is performed for predicting the sensor data for five seconds and ten seconds ahead of time. The time taken for EKF prediction is almost equal to the scan rate of the sensor data, while the time for NN is around four times of it. The performance of the neural

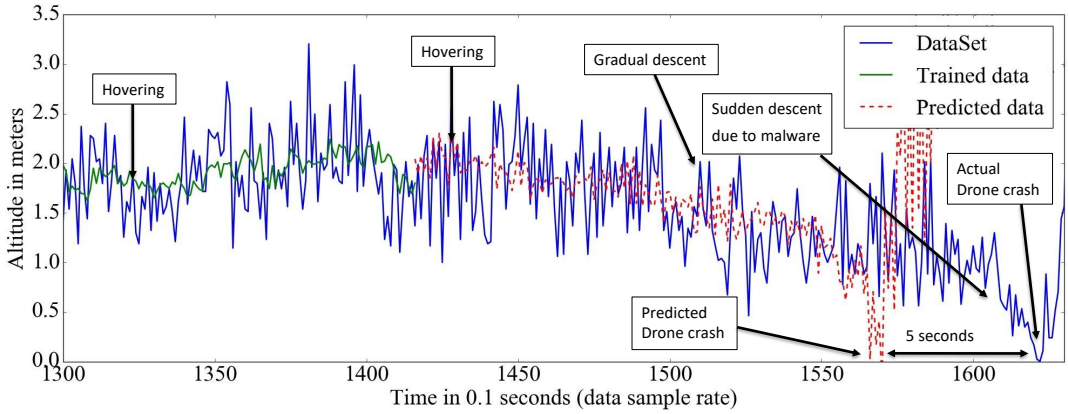


Figure 6: Crystal predicting the crash before the actual crash occurred

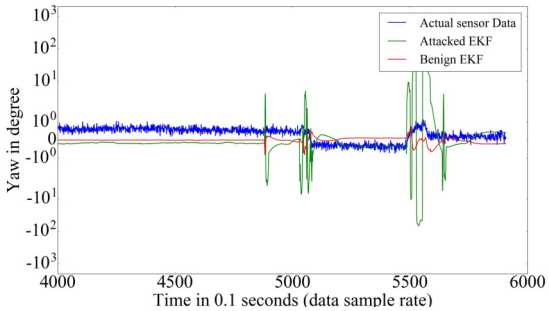
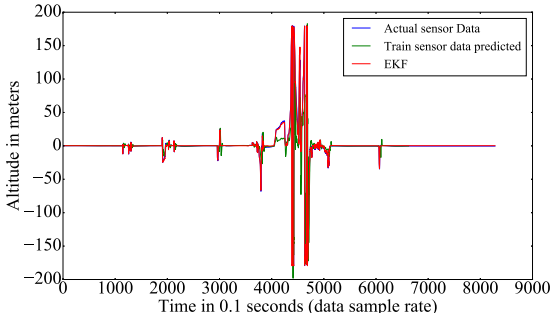


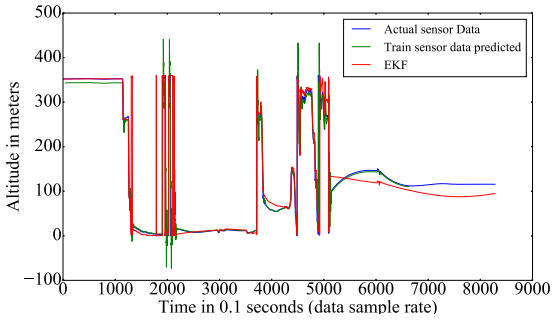
Figure 7: Crystal predicting the attack on attitude and heading reference system (AHRS)

Table 2: Latency in milliseconds for predicting sensor data with data points accumulating to 5 and 10 seconds

Estimation methods	Min	Avg	Max	Mdev
EKF (5 seconds)	10	56	120	26.64
EKF (10 seconds)	60	104.5	140	23.5
NN (5 seconds)	290	389.5	480	54.72
NN (10 seconds)	330	417.5	530	59.9



(a) Roll sensor data



(b) Yaw sensor data

Figure 8: Two figures showing that extended Kalman filter (EKF) is better in estimating with smoother transitions than neural network and during violent transitions, neural network is better than EKF

network can be further improved by using minimal lightweight tools written in native code compared to heavy tools like Keras and TensorFlow.

8 RELATED WORK

Control system security. The related work to protect the control systems' trusted computing base (TCB) are insufficient as software patches are often applied only months after release [34], and new vulnerabilities are discovered on a regular basis [3, 32]. The traditional perimeter-security tries to keep adversaries out of the protected control system entirely. Attempts include regulatory compliance approaches such as the NERC CIP requirements [46] and access control [10]. Despite the promise of information-security approaches, thirty years of precedence have shown the near impossibility of keeping adversaries out of critical systems [14] and less than promising results for the prospect of addressing the security problem from the perimeter [19, 20, 26]. Embedded controllers from most major vendors [19, 47] and popular HMIs [26] have been shown to have fundamental security flaws.

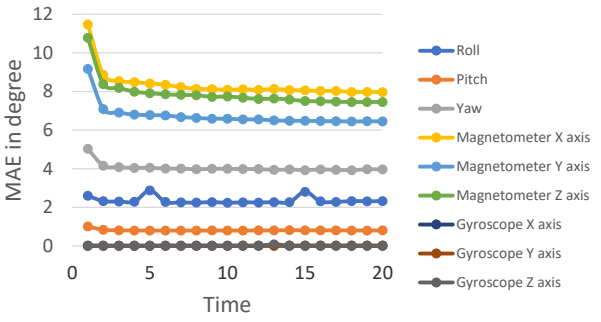


Figure 9: Mean Absolute Error vs time

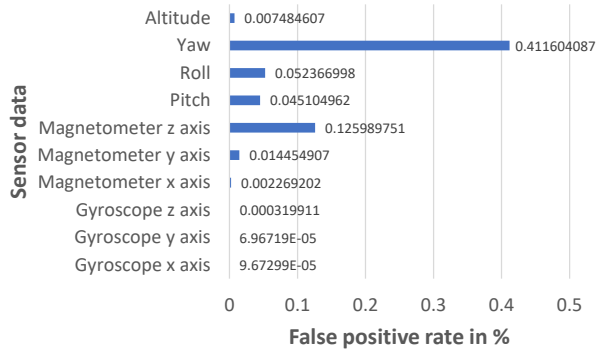


Figure 10: False positive rate due to sensor prediction

Controller program analysis. Basic static program analysis approaches use SAT-based model checking through Boolean logic [12, 22, 31] that could analyze sequence-based control systems with timers, but those are only narrowly applicable. Unlike [6], the two theorem-proving based approaches [13, 29] handle numerical instructions but do not implement rules for overflow checks or mixed bit vector and integer arithmetic. Almost all static analysis techniques [23] fall short in either checking for all program details or scaling up to large-scale critical infrastructures. To improve dynamic SCADA infrastructure monitoring techniques [44], PLC-based approaches have been suggested [4, 15] for dynamic physical plant monitoring. Dynamic plant behavior safety monitors [25] and mathematical intrusion detectors [8] are also related. In addition to being intrusive and causing performance overhead, dynamic monitoring solutions such as WeaselBoard [28] focus mainly on accidental failures, ignoring malicious actions, and/or leave an insufficient time buffer for an effective response and recovery in case of an attack or failure.

Drone security and safety. There have been several recent efforts on offline and runtime formal verification of drone platforms [36]. Javaid et al. [16] investigates potential threats against UAV platforms and how existing cybersecurity techniques fall short in defense due to the lack of consideration of the physical dynamics. Chan et al. [7] present an overview of formalizing stability properties of cyber-physical systems and drone platforms using the Coq proof assistant. The proof procedures introduced require fairly tedious human involvement. R2U2 [38] proposes a runtime formal verification for monitoring of security properties and diagnosing of security incidents. R2U2 continuously monitors inputs from various sources such as the GPS and the ground control station and identifies anomalous behaviors once they occur. R2U2 relies on the models of the controller code that are assumed given by the operators. Additionally, as discussed earlier, R2U2's runtime verification failure alerts often result in too-late notifications for a timely recovery strategy selection and deployment.

9 CONCLUSION

We presented Crystal, a just-ahead-of-time formal verification and controller recovery solution for cyber-physical system and evaluated our solution over 3DR Solo quadcopter. Crystal's just-ahead-of-time analysis eliminates the state explosion problem and gives the operators a time gap to choose recovery actions. Additionally, unlike traditional online monitoring solutions, Crystal leaves the operators with an arbitrarily-adjustable time gap to decide upon how to recover the system normal operation mode in case of an unsafe state. Our experimental results show that Crystal can proactively detect unsafe states, and recover the system with a negligible performance overhead.

A GLOBAL SAFETY CONDITIONS

The class of attacks and the relevant global safety requirements for the attacks on the quadcopter are shown in Table 3

B NORMAL OPERATION MODE PHYSICAL MODELING

The nonlinear discrete-time system obtained from the flight dynamics equations can be written by the state equations as

$$x(n+1) = f(x(n), u(n)) + w(n) \tag{3}$$

$$y(n) = h(x(n)) + v(n) \tag{4}$$

where x is the (i * 1) state vector of sensor data, y is the (j * 1) observation vector of actuator data, f is the state transition model, h is the observation model. Both f and h are nonlinear functions. w and v are the processes and observation noises which are zero-mean Gaussian noises with known covariance q and r respectively. u is the control vector. n is the current sample of the sensor data and n + 1 is the future sample of the sensor data. For the system equations in Equation 3 and Equation 4 the EKF solution for state estimation of one step ahead of time are given by Equations 3-9

$$\hat{x}(n+1|n) = f(\hat{x}(n|n)), u(n)) \tag{5}$$

Equation 5 is the predicted sensor data values for one step ahead of time.

$$\hat{x}(n|n) = \hat{x}(n|n-1) + K(n)[y(n) - h(\hat{x}(n|n-1))] \tag{6}$$

Equation 6 is the sensor values estimation update.

$$K(n) = P(n|n-1)H(n)^T[H(n)P(n|n-1)H(n)^T + R(n)]^{-1} \tag{7}$$

K(n) is (i * j) Kalman gain matrix

$$P(n+1|n) = F(n)P(n|n)F(t)^T + Q(n) \tag{8}$$

Equation 8 is the predicted covariance estimate.

$$P(n|n) = P(n|n-1) - K(n)H(n)P(n|n-1) \tag{9}$$

Table 3: Evaluation attacks and descriptions

Class	Description	Example attack	Attack consequence	Example safety requirements
Motor and server control	Motors or the servo control, controls the movement of the quadcopter	Switching off the motor	Crash to the ground due to the lack of thrust	$Thrust \geq \gamma, \forall altitude > 0$
		Increasing the speed of a motor	Fly high due to increase in thrust	$\frac{Thrust}{dt} < \lvert\gamma\rvert, \forall altitude > 0$
		Increasing the adjacent motors	Move towards the direction away from the motors	$\frac{Thrust\_on\_adjacent\_Motors}{dt} < \lvert\gamma\rvert, \forall altitude > 0$
		Increasing the opposite motors	Rotates along its axis	$\frac{Thrust\_on\_opposite\_Motors}{dt} < \lvert\gamma\rvert, \forall altitude > 0$
AHRD	Attitude and heading reference systems (AHRS) provides heading and attitude information based on magnetometer, accelerometer and gyroscope	IMU sensor data modification on control algorithm	Heading towards undesired directions	$IMU\_Data\_on\_Drone - IMU\_data\_predicted < \lvert\gamma\rvert$
		Timing attack by delaying the sensor data	Delays the control response leading to undesired motion of the quadcopter	$Time\_on\_Drone = Time\_on\_Crystal$
		Modified control algorithm	Inaccurate control commands given to quadcopter	$State\_on\_Drone = State\_on\_Crystal$
Position hold	Holds the quadcopter in a position by using GPS data	IMU sensor data modification on control algorithm	Movement of the quadcopter	$IMU\_Data\_on\_Drone - IMU\_data\_predicted < \lvert\gamma\rvert$
		GPS sensor data modification on control algorithm	Change in position of quadcopter	$GPS\_Data\_on\_Drone - GPS\_data\_on\_Crystal < \lvert\gamma\rvert$
		Modified control algorithm	Inaccurate control commands given to quadcopter	$State\_on\_Drone = State\_on\_Crystal$
Altitude hold	Holds the altitude of the quadcopter by using barometer	Barometer sensor data modification on control algorithm	Change of altitude	$Barometer\_Data\_on\_Drone - Barometer\_data\_predicted < \lvert\gamma\rvert$
		Modified control algorithm	Inaccurate control commands given to quadcopter	$State\_on\_Drone = State\_on\_Crystal$
Drift	Drifts due to external environment or sensor accuracy	Sensor data modification on control algorithm	Slight drifting motions of the quadcopter	$SensorData\_on\_Drone - Sensordata\_predicted < \lvert\gamma\rvert$
		Modified control algorithm	Inaccurate control commands given to quadcopter	$State\_on\_Drone = State\_on\_Crystal$

Equation 9 is the update of the covariance estimate

$$F(n) = \frac{\partial f}{\partial x}_{x=\hat{x}(n|n)} \tag{10}$$

$$H(n) = \frac{\partial h}{\partial x}_{x=\hat{x}(n|n-1)} \tag{11}$$

Equation 10 and Equation 11 are Jacobian matrices.

Crystal predicts K steps ahead of time instead of estimating one step at a time. The estimation of sensor data K steps ahead of time is given by Equation 12

$$\hat{x}(n+k|n+k-1) = f(\hat{x}(n+k-1|n+k-1)), u(n+k-1)) \tag{12}$$

ACKNOWLEDGMENTS

This work is supported in part by the US National Science Foundation under grant numbers CNS-1446471 and CNS-1453046.

REFERENCES

[1] 2017. 3DR Solo Quadcopter, available at https://3dr.com/solo-drone/. (2017).
[2] Dane Bamburry. 2015. Drones: Designed for Product Delivery. *Design Management Review* 26, 1 (2015), 40–48.
[3] Dillon Beresford. 2011. Exploiting Siemens Simatic S7 PLCs. In *Black Hat USA*.
[4] Hans Berger. 2013. *Automating with SIMATIC S7-1200: Configuring, Programming and Testing with STEP 7 Basic*. Vol. 11. John Wiley & Sons.
[5] AB Bitcraze. 2016. Crazyflie 2.0. (2016).
[6] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and P. Schnoebelen. 2000. Towards the Automatic Verification of PLC Programs Written in Instruction List. In *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4. 2449–2454.
[7] Matthew Chan, Daniel Ricketts, Sorin Lerner, and Gregory Malecha. 2016. Formal Verification of Stability Properties of Cyber-physical Systems. (2016).
[8] Steve Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. 2007. Using Model-based Intrusion Detection for SCADA Networks. In *Proceedings of the SCADA Security Scientific Symposium*.
[9] Nicolas Falliere, Liam O. Murchu, and Eric Chien. 2010. *W32.Stuxnet Dossier*. Technical Report. Symantic Security Response.
[10] David Formby, Preethi Srinivasan, Andrew Leonard, Jonathan Rogers, and Raheem Beyah. 2016. Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems. In *NDSS*.
[11] Rob Gerth, Doron Peled, Moshe Y Vardi, and Pierre Wolper. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *International Symposium on Protocol Specification, Testing and Verification*. IFIP.
[12] J.F. Groote, S.F.M. van Vlijmen, and J.W.C. Koorn. 1995. The Safety Guaranteeing System at Station Hoorn-Kersenboogerd. In *Tenth Annual Conference on Systems Integrity, Software Safety and Process Security*. 57–68.
[13] Ralf Huuck. 2005. Semantics and Analysis of Instruction List Programs. *Electronic Notes in Theoretical Computer Science* 115 (2005), 3–18.
[14] Vinay M Igure, Sean A Laughter, and Ronald D Williams. 2006. Security issues in SCADA networks. *Computers & Security* 25, 7 (2006), 498–506.
[15] Maria G Ioannides. 2004. Design and implementation of PLC-based monitoring control system for induction motor. *Energy Conversion, IEEE Transactions on* 19, 3 (2004), 469–476.
[16] Ahmad Y Javaid, Weiqing Sun, Vijay K Devabhaktuni, and Mansoor Alam. 2012. Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*. IEEE, 585–590.
[17] Samy Kamkar. 2013. SkyJack: autonomous drone hacking. *Online* (2013). http://samy.pl/skyjack
[18] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, and others. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 207–220.
[19] Egor Vladimirovich Kuz'min and Valery Anatolievich Sokolov. 2012. On Construction and Verification of PLC-Programs. *Modelirovanie i Analiz Informatsionnykh Sistem [Modeling and Analysis of Information Systems]* 19, 4 (2012), 25–36.
[20] Ted G Lewis. 2006. *Critical infrastructure protection in homeland security: defending a networked nation*. John Wiley & Sons.
[21] Yao Liu, Peng Ning, and Michael K Reiter. 2011. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 13.
[22] Stephen McLaughlin and Patrick McDaniel. 2012. SABOT: specification-based payload generation for programmable logic controllers. In *Proceedings of the 2012 ACM conference on Computer and Communications Security*. 439–449.
[23] Stephen McLaughlin, Saman Zonouz, Devin Pohly, and Patrick McDaniel. 2014. A Trusted Safety Verifier for Controller Code. In *Network and Distributed System Security Symposium*.
[24] Lorenz Meier, J Camacho, B Godbolt, J Goppert, L Heng, M Lizarraga, and others. 2013. Mavlink: Micro air vehicle communication protocol. *Online]. Tillgänglig: http://qgroundcontrol. org/mavlink/start.[Hämtad 2014-05-22]* (2013).
[25] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. 2012. S3A: Secure System Simplex Architecture for Enhanced Security of Cyber-Physical Systems. http://arxiv.org. (2012).
[26] Thomas H Morris, Anurag K Srivastava, Bradley Reaves, Kalyan Pavurapu, Sherif Abdelwahed, Rayford Vaughn, Wesley McGrew, and Yoginder Dandass. 2009. Engineering future cyber-physical energy systems: Challenges, research needs, and roadmap. In *North American Power Symposium (NAPS), 2009*. IEEE, 1–6.
[27] Devaprakash Muniraj and Mazen Farhood. 2017. A framework for detection of sensor attacks on small unmanned aircraft systems. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*. IEEE, 1189–1198.
[28] Department of Homeland Security. 2014. Cyber Security Division Transition to Practice Technology Guide. (2014).
[29] Sidi Ould Biha. 2011. A Formal Semantics of PLC Programs in Coq. In *IEEE 35th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 118–127.
[30] P Paganini. 2014. Hack-proof drones possible with HACMS technology. *http://resources. infosecinstitute. com/hack-proof-drones-possiblehacms-technology* (2014).
[31] Taeshin Park and Paul I Barton. 2000. Formal Verification of Sequence Controllers. *Computers & Chemical Engineering* 23, 11 (2000), 1783–1793.
[32] Dale G. Peterson. 2012. Project Basecamp at S4. http://www.digitalbond.com/2012/01/19/project-basecamp-at-s4/. (January 2012).
[33] André Platzer. 2011. Logic and compositional verification of hybrid systems. In *Computer Aided Verification*. Springer, 28–43.
[34] Jonathan Pollet. 2010. Electricity for Free? The Dirty Underbelly of SCADA and Smart Meters. In *Proceedings of Black Hat USA 2010*.
[35] Michael Robinson. 2015. Knocking my neighbor's kid's cruddy drone offline. In *Defcon*.
[36] Ann Rogers and John Hill. 2014. *Unmanned: Drone warfare and global security*. Between the Lines.
[37] Ruben Santamarta. 2012. Here be backdoors: A journey into the secrets of industrial firmware. *Black Hat USA* (2012).
[38] Johann Schumann, Patrick Moosbrugger, and Kristin Y Rozier. 2015. R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In *Runtime Verification*. Springer, 233–249.
[39] Lui Sha. 2001. Using simplicity to control complexity. *IEEE Software* 4 (2001), 20–28.
[40] Noah Shachtman. 2011. Computer virus hits US drone fleet. *CNN. com* (2011).
[41] Scott Shane and David E Sanger. 2011. Drone crash in Iran reveals secret US surveillance effort. *The New York Times* 7 (2011).
[42] Ali Shareef, Yifeng Zhu, Mohamad Musavi, and Bingxin Shen. 2007. Comparison of MLP neural network and kalman filter for localization in wireless sensor networks. In *Proceedings of the 19th IASTED International conference on parallel and distributed computing and systems*. ACTA Press, 323–330.
[43] Yunmok Son, Hocheol Shin, Dongkwan Kim, Young-Seok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, Yongdae Kim, and others. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors.
[44] Keith A Stouffer, Joseph A Falco, and Karen A Scarfone. 2011. SP 800-82. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC). (2011).
[45] uasvision. 2016. NASA Drone Hack Revealed. *http://www.uasvision.com/2016/02/02/nasa-drone-hack-revealed/* (2016).
[46] U.S. Department of Energy Office of Electricity Delivery and Energy Reliability. 2015. North American Electric Reliability Corporation Critical Infrastructure Protection (NERC-CIP) Standards; available at http://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx. (2015).
[47] Sidney E Valentine. 2013. *PLC code vulnerabilities through SCADA systems*. Ph.D. Dissertation. University of South Carolina.
[48] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. 2014. AVATAR: A framework to support dynamic security analysis of embedded systems firmwares. In *Symposium on Network and Distributed System Security (NDSS)*.