A Model of a Malware Infected Automated Guided Vehicle for Experimental Cyber-Physical Security

Richard French, Viktoriya Degeler and Kevin Jones
Airbus Group Innovations
Newport, UK
richard.french@airbus.external.com, viktoriya.degeler@airbus.com and kevin.jones@airbus.com

***Abstract*—As part of a factory's manufacturing cycle, materials need to move through a sequence of operations provided by work-cells, eventually culminating in the finished product. To facilitate this, the collection and delivery of materials around a factory environment is often provided by a fleet of Automated Guided Vehicles and results in increased reliability and reduced operating costs. However, if malware is able to get into the system via a deliberate cyber attack on the site or indeed by way of an infected portable data storage device and human error, then the AGV system is potentially at risk of disruption. As part of the exploration of this growing problem space new tools are needed to assist with developing counter strategies towards blocking such cyber-borne industrial sabotage. This article describes one such qualitative research tool we have developed as part of ongoing research into protecting industrial processes from cyber attack.**

I. Introduction

As part of a factory's manufacturing cycle, materials need to move through a sequence of operations provided by work-cells, eventually culminating in the finished product. To facilitate this, the collection and delivery of materials around a factory environment is often provided by a fleet of industrial robots, known as Automated Guided Vehicles (AGVs), and results in increased reliability and reduced operating costs [1]. Significantly, as with all distributed industrial control systems, this equipment is networked to allow it to operate with other subsystems such as work-cells and automated warehousing. However, if malware is able to get into the system via a deliberate cyber attack on the site or indeed by way of an infected portable data storage device and human error (e.g., [2]), then the AGV system is potentially at risk of disruption.

Undoubtedly, with greater automation of industrial processes come great benefits in terms of efficiency and flexibility in an increasingly competitive business world. However, with these benefits also come great risks, and as part of the exploration of this growing problem space new tools are needed to assist with developing counter strategies towards blocking such cyber-borne industrial sabotage. This article describes one such qualitative research tool we have developed as part of ongoing research into protecting industrial processes from cyber attack.

This paper proceed as follows: Firstly an overview of recent cyber attacks on production facilities is given in order to emphasize the importance of this problem. Next, the rationale for choosing the AGV as the focus of this work is given, together with a hypothetical scenario. Following on, modelling in simulation versus the utility of physically grounded systems is discussed, explaining how this approach results in a novel research tool for malware affected cyber-physical systems. Consequently the implementation of an AGV model is described. Next, the model malware is considered, explaining its type, trigger, payload and implementation. This is followed with an example describing how the malware affects the model AGV behaviour. Finally, this paper concludes with a summary of achievements and a discussion of how this work can assist with development of counter-measures to attacks on cyber-physical systems.

II. Cyber Attacks on Production Facilities

The Repository of Industrial Security Incidents database[1] holds details of malware attacks that have affected industrial control systems at production facilities. The following is a small representative set of examples that aim to illustrate the importance of this ongoing problem:

- In 2005, an internet worm affected assembly lines at 13 vehicle manufacturing plants. Assembly line workers were sent home and vehicle production was stopped for almost an hour.
- In 2008 a system controlling production line operations was infected by a virus, causing a reduction of capacity.
- In 2010 an Iranian mill, probably for wheat flour, was affected by the Stuxnet virus[2]. A 350-ton capacity multi-purpose mill was shut down. Also, and famously, in this year, Stuxnet malware affected centrifuge spinning speed causing Iranian uranium enrichment to be disrupted for at least one week.
- In 2011 the Conficker worm[3] spread throughout a steel plant's power automation network. This caused instability in the communications between PLCs and supervisory stations and freezing most of the supervisory systems.

[1] http://www.risidata.com/Database
[2] http://home.mcafee.com/virusinfo/virusprofile.aspx?key=268468#none
[3] http://home.mcafee.com/virusinfo/virusprofile.aspx?key=153464

- In 2012 oil facilities in Iran were affected by a malware attack. Consequently equipment on Kharg island and at other Iranian oil plants were disconnected from the network as a precaution.

Intuitively, with the increased usage of automation (e.g., [3]), the likelihood of attack is only going to increase, and with it, the loss of revenue.

III. Scenario

Arguably, attacking the element that underpins the flow through an automated manufacturing process would yield the greatest damage. Beyond the control network itself, a potential candidate for this target in terms of a cyber-physical system is the AGV. Thus, by disrupting this service, the entire production line will gradually grind to a halt as materials go uncollected from automated warehousing and subassemblies do not progress through to the next stage of production.

Hence this scenario takes place in a hypothetical modern production line. AGVs diligently collect material from the warehouse and deliver it to work-cells, keeping the whole line running smoothly. However, unbeknownst to the operators, this factory has been subject to a cyber attack and some of the vehicles have been infected with malware. This malicious code has never been seen before and thus avoids detection by traditional signature-based techniques. Hiding in a robot's memory, the malware silently activates. The robot is now rogue, possibly just stopping and blocking its guide-path, moving to an incorrect location causing chaos, or even writing random data directly to its motor control chips and causing an accident. Clearly this situation has the potential to become much more than an inconvenience to production line efficiency.

IV. Modelling

In order to model the robotic aspect of this scenario a representation of AGVs in their environment is needed. What is the best representation for this exercise?

A. To Simulate, or not to Simulate?

There are good reasons for modelling to occur in simulation. For example, working purely in a virtual world eliminates many inconvenient aspects of real-world robotics. Thus, the requirement for actual robot hardware, its maintenance, the space needed to carry out experiments, as well as degradation through wear and tear on mechanical systems, can all be ignored. Also as pointed out by Nehmzow [4]:

> To conduct experiments with mobile robots can be very time consuming, expensive, and difficult. ... being mechanical and electronic machines, do not perform identically in every experiment. Their behaviour sometimes changes dramatically, as some parameter changes. Such hardware-related issues make simulation an attractive alternative.

However, using such an abstraction also means moving away from the unforeseen influences that its real-world counterpart must be robust to:

> ... there are also good reasons not to simulate. In some circumstances, to obtain an accurate model can be far more work than to run a robot. And to run the robot will give you true answers, whilst running a simulation won't.

And, as Brooks [5] puts it:

> ... because the world is its own best model (as usual). When running a physically grounded system in the real world, one can see at a glance how it is interacting. It is right before your eyes. There are no layers of abstraction to obfuscate the dynamics of the interactions between the system and the world. This is an elegant aspect of physically grounded systems.

Further, a simulation lacks the physical presence that is arguably very valuable when demonstrating a real-world concept to an audience when trying to get the point across. Therefore an aim of this work is to create a realistic and physically demonstrable, small form-factor, model that reacts in real-time to changing conditions in order to present our ongoing research in the clearest way possible.

B. A New Model?

For the scenario to be realised a model of the malware is needed too. Thus the end point of this work is a physically grounded model of an AGV which also incorporates configurable malware for experimental investigation.

Is this a novel approach? As observed by Le-Anh and Koster (2004) [6], there ...

> ... are few review papers on AGV systems. However, they concentrate on only limited parts of the problem ....

Unfortunately we have also found this to be the case, making comparison of our work with that of others difficult.

From the point of view of a physically grounded model, although there are examples of full-size prototypes (e.g., [7]), to the best of our knowledge there are no current publications describing small robotic models of AGVs for experimental work. Likewise, although malware propagation has been simulated (e.g., [8] [9]) and an investigation has been carried out into cyber attacks on automated vehicles [10], we are not aware of a study in which a robotic vehicle has been developed with the explicit purpose of studying the impact of malware with a view to developing counter-measures.

Hence, we believe this work results in a new kind of AGV model. It is a physically grounded implementation that offers complete control over the infection, enabling the resultant behaviour of a model production facility during cyber attack scenarios to be studied.

V. Model Implementation

Sharma [11] decomposes an AGV system into supervisory and subordinate controllers, and this is the approach taken here. In order to maintain focus on the goal of this work and thus avoid designing a full AGV System, supervisory and subordinate control functions are simplified whilst also aiming for a high degree of realism in terms of vehicle behaviour. Hence, greater emphasis is placed on development and operation of the subordinate, rather than the supervisory, in the remainder of this paper.

A. Supervisory Control

The AGV Controller (AGVC) is the supervisory level control, responsible for AGV assignment to a particular job together with route planning and control of vehicle interactions. In this work, the AGVC simply assigns the nearest free AGV to the collection point, creates a route for it and sends the robot on its way.

B. Subordinate Control

Subordinate control is carried out on each AGV and is responsible for the robot's navigation and behaviour in pursuance of its allocated job:

1) AGV Navigation: The pathway an AGV travels between collection and delivery points is often described by a buried cable, magnetic strip or optically detected surface. Additionally, an AGV can be augmented with varying degrees of navigational aids. These may range from index check-points to counter wheel-encoder slippage during dead-reckoning, through to machine-readable reference points such as barcodes and radio-frequency identification tags positioned along its route (e.g., [12], [13]).

In this work an optically detected navigation grid is chosen for ease of development. As such, it is plotted on a paper roll 610mm wide, resulting in an environment (including perimeter space) of approximately 610mm by 949mm. The grid lines are black on a white background and so can be sensed by reflected infrared light, as this is a common sensing ability among mobile robots. The grid is illustrated in Figure 1. Thus a robot using a line-following approach can maintain its course and, by sensing the junctions and keeping track of its orientation, simply increments, or decrements coordinate variables as it traverses the grid. This then is our navigation scheme using nothing more complex than a small array of optical sensors and a few bytes of local memory.

2) Behaviour: The model AGV's behaviour is generated by four layers of control. This is illustrated in Figure 2. From the point of view of its hierarchy, that is from the lowest level of sensory-motor control and reflexive behaviour through to its top-level task description, this architecture may be regarded as inspired by the subsumption architecture of Brooks [14]. However, an important distinction is that the levels of our control scheme are not competing

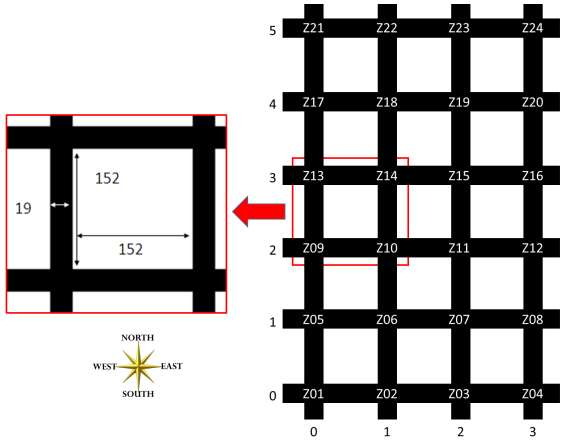

Figure 1. The navigation grid layout for this work, showing track spacing and width in mm, together with zone identifers Z01 to Z24. Zone identifier positions relative to points of the compass are also shown.

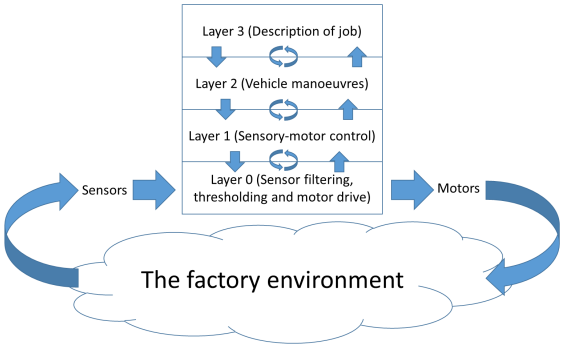

Figure 2. The layered control of our model AGV.

behaviours running asynchronously with their own sensor-actuator loops, but are complimentary to the top-most task, with sensory-motor links at the basal level.

C. Platform

The platform chosen for building the AGV model is the very low-cost desktop Pololu 3Pi robot[4]. It is based upon the Atmel 328P microcontroller and is equipped with optical sensors and onboard battery power supply as standard.

1) Robot Model Interfacing Considerations: In order to turn this base unit into the model, a low-power BlueTooth *BlueSMiRF*[5] modem is added for communication, together with a Force Sensitive Resistor (FSR)[6] for detecting the presence of a load, and a normally-closed (NC) microswitch for detecting collisions. As customising a small mobile robot with limited onboard I/O capacity for this kind of application can be problematic, a little lateral thinking was needed in the approach to interfacing. A brief description of the solution

[4]https://www.pololu.com/docs/pdf/0J21/3pi.pdf
[5]https://www.sparkfun.com/products/12577
[6]http://www.digikey.co.uk/product-search/en?mpart=30-81794vendor=1027

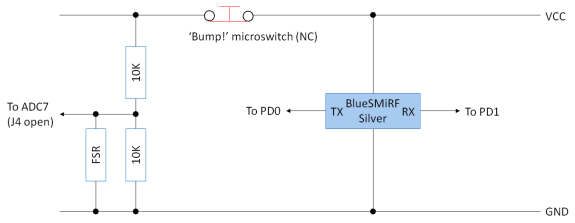

Figure 3. Hardware addition to the Pololu 3Pi for the AGV model.

is offered here in the event it is helpful for replication of this work.

The 3Pi robot has a single unused Analogue to Digital Converter (ADC) input port designated ADC7[7]; the remaining I/O being occupied with the robot's reflective infrared sensors, battery voltage sensing, user option switches, motor drive and LCD display. As both load and collision sensors are required for this application, it is necessary to perform both functions using this single free analogue input. The circuit diagram of the solution together with communication module connections is given in Figure 3.

Under normal circumstances the microswitch is closed, dropping VCC (5V) across the potential divider formed by the resistors and the FSR. However, if during its movement across the navigation grid the AGV runs into an obstruction, the microswitch is operated, opening the contacts and thus the voltage presented at ADC7 is by way of the FSR and parallel 10K, acting as a pull-down resistor, giving 0V. Importantly, the resistance value of 10K is chosen to be in line with Atmel's requirement that the ... *ADC module is optimized for analog signals with an output impedance of 10k or less. If such a source is used then the sampling time will be negligible*. [8](Page 244)

With no loading the resistance of the FSR is so high that the potential divider effectively presents 2.5V to ADC7. However, under load conditions the FSR value can fall to almost 1K. This means ADC7 will be presented with a voltage range of around 0.5V to 2.5V for varying loads on the AGV. With the analogue to digital converter set to 10-bit operation, a zero load naturally results in a decimal value of around 512. Initial tests with a plastic weight resulted in a value of around 412, whereas a heavier metallic weight gave a value of around 319. Operating the microswitch drops the value significantly below that for any realistic load the AGV may carry, approaching zero. Thus a software routine sensitive to this range of values can determine load type or the presence of an obstacle. The AGV model is shown in Figure 4.

2) Control layers: With reference to Figure 2, layer zero is the lowest level of control in the AGV model. Its functions

[7]https://www.pololu.com/file/0J119/3pi-schematic.pdf

[8]http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf

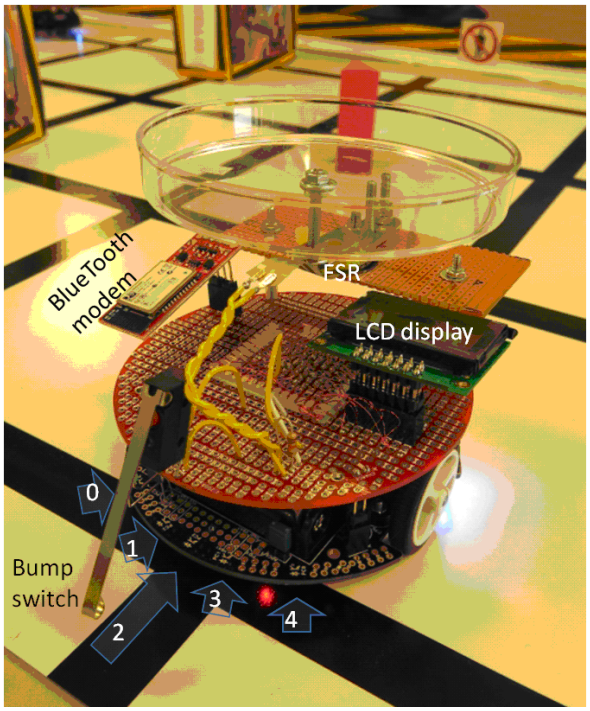

Figure 4. Automatic Guided Vehicle model based upon the Pololu 3Pi robot. The positions of reflective infrared line sensors 0-4 are shown, together with bump switch, BlueTooth modem, Force Sensitive Resistor and LCD display.

Table I
LAYER ZERO CONTROL

Layer 0 control	Description
Collision-detected	True if $ADC7\ Value < 200$
At-line	True if IR sensor 2 detects line
At-junction	True if IR sensors 0 and 4 detect line
Loaded	True if $500 > ADC7\ Value > 200$
Stop	Stops motors
Run-motors	Enables motors if no collision
Turn, Forwards, Backwards	Run-motor commands

(Table I) remove transients in sensor readings and thresholds the result, detects a grid-line and junction, determines the load status, detects an obstacle and operates the motors. Layer one functions (Table II) use sensory-motor feedback through the robot's environment using layer zero functions to perform fine-positioning of the vehicle relative to the grid lines and junctions. Layer two functions (Table III) control vehicle manoeuvres. These are locked to the navigational grid junctions and are formed from sequences of layer one and layer two functions.

Finally, layer three is the highest level of control and holds a description of the current job assigned to the robot. As such it is a Finite State Machine (FSM) (e.g., [15]) description whose states are associated with layer two functions. FSM State transitions are made on successful completion of layer 2 functions resulting in meeting a waypoint in the environment or collection/delivery of a load. This is

Table II
LAYER ONE CONTROL

Layer 1 control	Description
Turn-off-line	Turn until At-line False
Turn-onto-line	Turn until At-line True
Move-off-junction	Forwards until At-junction False
Move-on-junction	Forwards until At-junction True
Backup-from-junction	Backwards until At-junction False
Line-follow	Line-follow forwards
Reverse-line-follow	Line-follow backwards
Pick-up-load	Wait until Loaded True
Drop-off-load	Wait until Loaded False

Table III
LAYER TWO CONTROL

Layer 2 control	Description
Move-to-line	1) Move-off-junction 2) Line-follow 3) Move-on-junction 4) Update-XY
Backup-to-previous-junction	1) Backup-from-junction 2) Reverse-line-follow 3) Update-XY
Turn-to-line	1) Turn-off-line 2) Turn-onto-line
Grid-right-turn	1) Turn-to-line (skid right) 2) Update-direction 3) Move-to-line
Grid-left-turn	1) Turn-to-line (skid left) 2) Update-direction 3) Move-to-line
Grid-about-face	1) Backup-from-junction 2) Turn-to-line (on-axis 180^o) 3) Update-direction 4) Move-to-line
Pick-up	1) Stop 2) Pick-up-load
Drop-off	1) Stop 2) Drop-off-load

illustrated in Figure 5. Thus by defining the FSM's states and transitions, a route can be defined for the AGV across the factory floor in pursuance of its allocated job. Importantly, on successful completion of each vehicle manoeuvre, the robot sends a message to supervisory control. In the event it collides with an object in its environment, the robot stops what it is doing and executes a reflexive behaviour of reversing back to the preceding junction on the navigational grid and stopping. This manoeuvre occurs in layer two of the control hierarchy. A message reporting this is also

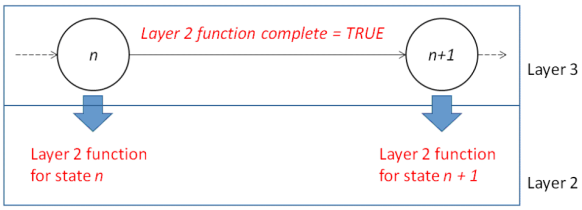

Figure 5. Transition to state n+1 on completion of layer 2 function for state n.

sent across the communication link with supervisory control signalling a problem has occurred.

VI. MALWARE

Morris and Gao [16] point out that a cyber criminal ...

> ... can use command injection attacks to overwrite ladder logic, C code which in turn can result in ... interruption [of] process control [and] interruption of device communications ...

Assuming a vulnerability in a target system is available to leverage, several questions need to be answered prior to creating the malware. For the scenario in this work these are: What type of weapon to create, how to trigger the attack, what payload to deploy and how to implement it?

A. Type

What kind of malware is useful for attacking a cyber-physical system? Unless the attack is an act of terrorism where the perpetrator's identity is needed to be known, it is desirable to have a weapon that stays hidden and then activates some time after insertion into the target is complete. Importantly, this delay also allows the attacker to infect more physical devices before the attack occurs. Hence, this approach minimises any trace of who the aggressor is, and can also be the ideal weapon of choice for a disgruntled employee servicing industrial control systems.

Arguably, the scenario in this work is suitable for the application of a *logic bomb*, which is ...

> ... a routine or set of routines that are activated when a particular set of conditions is met (for example, the nth time the program is executed), and may be a component of a virus or Trojan. A logic bomb might also be inserted into a legitimate program as a precursor to blackmail, or pre-emptive revenge in anticipation of dismissal, or with some sort of backdoor functionality.[17](Page 100)

B. Trigger

As the malware is to be inserted into the control system of a mobile robot, there are various alternatives available to the author when choosing the trigger. For example, the charge level in the battery, the size of the job loaded in the memory, the location of the robot in its environment or indeed the

step currently being executed. For this work a combination of the last two of these possibilities has been chosen as the basis of a trigger: On entering a location within the factory environment, identify a forthcoming step in the current job to deploy the payload.

C. Payload

With the trigger decided upon, how much trouble should this malware cause? This is the function of the payload. For this work the payload needs to be capable of demonstrating a significant level of disruption it would cause in a manufacturing plant, while also being convenient to work with from an experimental perspective. Hence, for repeating experiments it is useful for the AGV model to largely be able to recover from payload deployment, whilst also requiring action from supervisory control to get the production line back in service.

The payload's actions decided upon for this work are thus as follows:

- Stop moving.
- Erase job memory.
- Break communication with supervisory control for a fixed period of time.

D. Implementation

How can all this be implemented? In order to simulate injection of malware into the model AGV, an appropriate level of abstraction is used by always having it present in the AGV source code. In this work, the logic-bomb is located within layer 3 of the control system.

1) Placing the Malware: The malware is positioned in the code used to send acknowledgement of a successful layer two manoeuvre to supervisory control. For example, within the switch block of the C source code (e.g., [18]) used for selecting the control function *move_to_line()*, there is a call to the *display_data()* routine that outputs AGV coordinates and orientation, together with job state to LCD display as well as sending messages to supervisory control:

```
move_ok_flag=move_to_line();
if(move_ok_flag)
 display_data();
 break;
```

Thus, on successful completion of *move_to_line()*, the *display_data()* routine is called to inform supervisory control. This in turn calls *isolate()*, and is the 'malware'.

2) Trigger: Within *isolate()* is a test that monitors the AGV navigation variables *world_X* and *world_Y* that give the robot's position within the factory navigation grid. Once these match a given pair of values, a random delay *states_to_fault* is calculated. This is simply an offset relative to the current state of the FSM for malware payload activation.

```
if(world_X==fault_X && world_Y==fault_Y){
 if(fault_flag && !fault_triggered){
  range=num_states-system_state;
  if(range>1){
   states_to_fault = 1+rand()%(range-1);
   fault_triggered=_TRUE_;
  }
 }
}
```

And so, once the malware has been triggered, *states_to_fault* counts down with every state transition. On reaching zero, the payload is activated.

```
if(fault_triggered && states_to_fault == 0){

 //Malware Payload
 :
 :
}
```

3) Payload: The malware payload carries out three disruptive acts: stopping the AGV motors, erasing the job being executed and breaking communications with supervisory control.

```
 //Malware Payload
 :
 all_stop();              //stop drives
 system_state=0;          //lose FSM state
 clear_program();         //erase job
 :
 delay(26000);            //break comms
 :
 states_to_fault = -1;//ready next attack
 fault_triggered=_FALSE_;
 :
```

VII. Evaluation with an Example

To illustrate this with an example, the layout shown in Figure 1 is used. A job is sent to AGVC and is to collect goods from Z24 (stores) and deliver to Z17 (a work cell's materials input). An AGV is free for work, and is parked at Z21 (0,5), facing West. A route Z21, Z22, Z23, Z24, Z20, Z19, Z18, Z17 is generated to carry out the job. Using this route together with the vehicle starting position and orientation, an intermediate representation of a new state machine description is generated. For sake of clarity this describes a step-by-step traversal of the route as opposed to a more efficient machine that eliminates states 2, 5 and 7, and is shown in Table IV. This description is then sent to the AGV over BlueTooth, where it is evaluated as the FSM, shown in Figure 6.

A. Stepping through the Scenario

For this example, the trigger point set to Z20 (3,4). The new FSM program is loaded, the AGV starts its FSM and control is passed to state 0, causing the vehicle to turn about its axis and move to Z22 (1,5). On completion of this action control is passed to state 1.

Table IV
THE INTERMEDIATE STATE MACHINE DESCRIPTION GENERATED BY THE AGVC.

State	X	Y	Layer 2 function
00	0	5	Grid-about-face
01	1	5	Move-to-line
02	2	5	Move-to-line
03	3	5	Pick-up
04	3	5	Grid-right-turn
05	3	4	Grid-right-turn
06	2	4	Move-to-line
07	1	4	Move-to-line
08	0	4	Drop-off
09	0	4	End

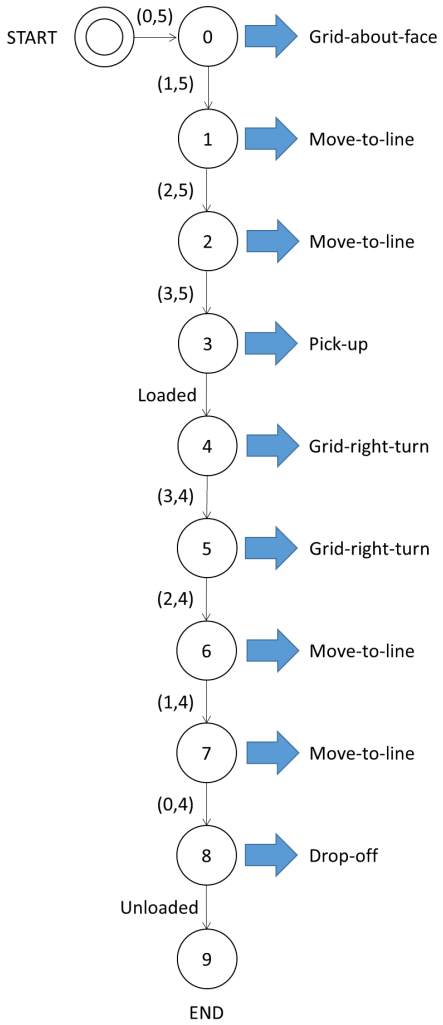

Figure 6. An example FSM.

State 1 causes the vehicle to move forward to Z23 (2,5). This progression through the states with completion of Layer 2 functions continue until state 3 is reached. The AGV waits for an object to be presented on its platform. This deforms the FSR, is detected by software at Layer 0 and, via Layer 1, completes the Pick-up function in Layer 2.

Eventually, state 4 goes active and causes the vehicle to turn right. On successful completion of this manoeuvre, the AGV sends a message to supervisory control via the *display_data()* routine. Hence the malware is called and checks the trigger point against the robot's location. There is a match and so a random *states_to_fault* value of 1 is generated.

Control is passed to state 5 and the AGV turns right with its load. On completion of this manoeuvre the robot is at Z19 (2,4) and *states_to_fault* is decremented to zero. Control is passed to state 6.

The robot moves forward and a message is sent to supervisory control indicating a successful manoeuvre. The malware detects the countdown has hit zero and the payload is deployed.

The robot halts in its tracks, has its FSM erased and enters a long wait state and so cannot service communication requests. This AGV is now isolated from supervisory control, has lost its task description, is carrying a load it cannot deliver and is blocking a track in the factory. Chaos.

This is the starting point in the scenario for research of counter strategies, intended as part of supervisory control, that attempt automatic recovery of the lost AGV, thus delivering its cargo and keeping the plant running without human intervention. As such, our early work in this area is given in [19].

In the context of triggerable malware such as a logic bomb, it is important that any such counter strategy learn through experience, eventually preventing further attack. This knowledge ("immunisation") can then be given to other such facilities, preventing them from suffering similar disruption.

B. How does this compare with other models of AGV failure?

Davies (1986) [20] observes ...

> Breakdowns of any resource in a simulation model, whether it is a machine, worker, or material handling device, are most often treated by creating submodels which schedule breakdown events to occur according to random distribution. In the AGV model, breakdowns are accomplished by halting the vehicle at the end of its transport.

However, the *reason* for AGV failure is non-specific.

Importantly, had this work been confined to simulation alone, then additional events such as malfunction due to premature battery discharge or wearing out of the AGV guide-path (the navigation grid), would not necessarily have

been considered for inclusion in the model. Indeed, these two events are mentioned as they occurred after running the AGV model robots at length and were never originally planned as part of a set of scenarios! Thus, such real-world reasons for a robot failing to complete its job, over and above any malware influence, would possibly have been missed in designing a simulation. This is significant because development of supervisory control systems that create counter strategies for dealing with AGV problems must be able to differentiate between malicious attacks and more conventional operational situations, without confusing the two.

VIII. Conclusions

This paper has described the development of a novel, physically grounded model of an automated guided vehicle for investigation of cyber attack scenarios. As such, the malware model resides in the top-most level of a layered subordinate control system. Importantly, being a configurable part of the AGV source code offers the experimenter full control over the underlying dynamics of infected AGVs, and thus allows any emergent phenomena in the behaviour of the system as a whole to be tied back to this source.

This work is intended to be used in two ways: Firstly, further development of the AGV scenario will be explored to aid research of adaptive supervisory control systems. These systems will monitor multiple and heterogeneous subordinate control, aiming for automated resolution of disruption caused by a cyber attack across a model production site. Secondly, this will lead into an investigation into the application of these techniques to real-world plant and machinery.

References

[1] L. Schulze, S. Behling, and S. Buhrs, "Automated guided vehicle systems: a driver for increased business performance," in *Proceedings of International Multi Conference of Engineers and Computer Scientists 2008 (IMECS 2008*, 2008, pp. 1275–1280.

[2] E. Gent, "Successful hacks and cyber attacks commonly result of human error," In Engineering and Technology Magazine, April 2015. [Online]. Available: http://eandt.theiet.org/news/2015/apr/threat-reports.cfm

[3] E. Ackerman, "Chinese unmanned factory replaces 600 humans with 60 robots," In IEEE Spectrum Magazine, August 2015. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/industrial-robots/chinese-unmanned-factory-replaces-humans-with-robots

[4] U. Nehmzow, *Mobile robotics: a practical introduction*. Springer, 2003.

[5] R. A. Brooks, "Elephants don't play chess," *Robotics and Autonomous Systems*, vol. 6, pp. 3–15, 1990.

[6] T. Le-Anh and M. D. Koster, "A review of design and control of automated guided vehicle systems," *"European Journal of Operational Research"*, vol. 171, no. 1, pp. 1 – 23, 2006.

[7] R. G. Rosandich, R. R. Lindeke, and J. Berg, "Developing an automatic guided vehicle for small to medium sized enterprises," *"Progress in Material Handling Research"*, pp. 461 – 470, 2002.

[8] Z. Chen, S. Member, and C. Ji, "Spatial-temporal modeling of malware propagation in networks," *IEEE Transactions on Neural Networks*, pp. 1291–1303, 2005.

[9] M. Garetto and W. Gong, "Modeling malware spreading dynamics," in *In Proceedings of IEEE INFOCOM*, 2003, pp. 1869–1879.

[10] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 546 – 556, 2015.

[11] M. Sharma, "Control classification of automated vehicle systems," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 2, 2012.

[12] M. Schneier, R. Bostelman, N. I. of Standards, and T. U. E. L. I. S. Division. (2015) Literature review of mobile robots for manufacturing. [Online]. Available: https://books.google.co.uk/books?id=_BgFjwEACAAJ

[13] S. Yaghoubi, S. Khalili, R. M. Nezhad, M. R. Kazemi, and M. Sakhaiifar", "Designing and methodology of automated guided vehicle robots/ self guided vehicles systems, future trends," *International Journal of Research and Reviews in Applied Sciences*, vol. 13, 2012.

[14] R. A. Brooks, "A robust layered control system for a mobile robot," *"IEEE Journal of Robotics and Automation"*, pp. 14 – 23, 1986.

[15] A. K. Dewdney, *The New Turing Omnibus*. Palgrave Macmillan, 2003.

[16] T. H. Morris and W. Gao, "Industrial control system cyber attacks," in *Proceedings of the 1st International Symposium for ICS SCADA Cyber Security Research*, 2013.

[17] D. Harley, R. Slade, and U. E. Gattiker, *Viruses Revealed*. Osbourne/McGraw-Hill, 2001.

[18] B. W. Kernighan and D. M. Ritchie, *The C Programming Language 2nd Edition*. Prentice Hall, 1988.

[19] V. Degeler, R. French, and K. Jones, "Combined danger signal and anomaly-based threat detection in cyber-physical systems," in *2nd EAI International Conference on Safety and Security in Internet of Things (SaSeIoT)*. EAI, 2015.

[20] J. Wilson, J. Henriksen, and S. Roberts, Eds., *Modeling AGV Systems*, 1986.