ADDITIONAL CASE STUDIES

This section presents extra case studies, which are evaluated using hardware-in-the-loop simulation, to demonstrate how TimeTrap can cause temporal displacements and how they instigate control deviation.

Case Study (4) on Autoware.Auto. Autoware.Auto [69] is an open-source autonomous driving project, which is widely used for research and development. It contains the full-stack software packages required for self-driving cars, including detection, localization, planning, control, etc, and supports different application scenarios. Our experiments were conducted on Autoware.Auto 1.0.0 version. We run the entire software system on AMD Ryzen 7 1700X with 32 GB RAM in compliance with the official recommended platform (8 cores and 32GB). The simulated vehicle model is Lexus 2016 RX Hybrid and the scenario used is the officially released Autonomous Valet Parking [53]. For simulation, we run LGSVL [83] on another PC with a GPU of Geforce RTX 2070S. The communication between the platform being tested and the simulator is via Ethernet.

Attack Result. Figure 12 (b) shows the traveled path of the vehicle under test (VUT) on the map. TimeTrap was launched at the point marked by a star and it was periodically contending for shared resources to cause temporal displacements. On the remaining traveled path, as shown in Figure 12 (d), there exist two segments that incurred obvious temporal displacements by TimeTrap tinted in red. From the first segment, the VUT was driving straight with no obvious control deviation. However, as shown in the second segment, the VUT completely lost control and hit the adjacent building.

Cause Analysis. The reason behind this is the malfunction of the localization module, causing the vehicle to falsely estimate its position. In Autoware.Auto, localization uses the Normal Distributions Transform [84] (NDT) matching algorithm to estimate the pose of the vehicle. Since the optimization problem in NDT is modeled as a maximum a posteriori (MAP) problem in the software, an initial guess value is required to accelerate the solver. We use two code snippets in Figure 12(a) to demonstrate the vulnerability. As shown in Figure 12(a), the estimation of pose (defined as `pose_out`) relies on the initial value, which is defined as `initial_guess`. As to `initial_guess`, its value is inferred upon looking up (line 65) or extrapolating (line 71) the *transformation tree* in the system (in Code 12(a)). This inference is based on the assumption that the transformation graph `tf_graph` is in sync with `target_frame`. As the inference is time-related, a delay on the update of `tf_graph` cause result in a temporal displacement on `target_frame`, producing erroneous results in the critical variable `initial_guess`. Furthermore, the incorrect `initial_guess` causes the solver `register_measurement()` to fall into a local minimum, resulting in erroneous localization result `pose_out`.

By comparing the errors of `pose_out` in two segments that have large temporal displacements, in Figure12(d), it can be observed that although `initial_guess` had errors to the same extent in these two periods, the control was less affected in the first segment. This is because the VUT was driving straight during the first period and more common features persist in two adjacent frames of the point cloud, allowing the NDT to still converge albeit incorrectly `initial_guess`. In Figure 12(c), it can be observed that the localization result never deviated from the reference path in the first segment. On the contrary, the localization results are erroneous in the second segment, where the results were jumping arbitrarily. This indicates that the attack outcome depends on the physical state of the victim. If the vehicle was in a non-vulnerable state, attackers need more CPU budget or higher priority to cause substantial impacts on control performance.

Different Simulation Worlds. We also conducted experiments on different scenarios and the experimental results are reported in Table IV.

TABLE IV: TimeTrap Attack on Different Scenarios

Map	Scene Charac.	Sensing Overhead (%)	Temporal Disp. (s)	Control Dev. (m)	**Success-rate(%)**
Parking Lot	S	0.17%	2.0	3.6	72%
	D	0.21%	2.3	3.9	56%
Borregas Ave.	S	0.18%	2.2	1.8	77%
	D	0.36%	2.3	1.7	61%
GoMent.	S	0.17%	2.3	2.4	65%
	D	0.25%	1.9	2.2	52%
San Fran.	S	0.19%	2.7	4.8	74%
	D	0.34%	2.3	5.4	58%
Shalun	S	0.17%	2.4	2.1	64%
	D	0.23%	2.9	2.6	51%

S: Static Scenario; D: Dynamic Scenario.

From the results, we can observe that the sensing had larger errors in dynamic scenarios. This is because the randomly generated traffic participants can cause uncertain events such as slowdowns or stops. Those events make the sensing more challenging for TimeTrap. Moreover, to handle the unexpected events, our adversarial task took more time to filter the disturbance. Overall, the sensing part of TimeTrap achieved effective performance. The maximum overhead in the experiments is 0.36% CPU usage. As to the control deviation, the VUT running on *Borregas Avenue* was the less affected because its map of *Borregas Avenue* is relatively simpler. Even for the least control deviation $1.7m$, TimeTrap was still able to lead the vehicle to hit the curb.

Case Study (5) on Humanoid ROBOTIS OP3. ROBOTIS OP3 [67] is a miniature humanoid robot, which consists of 20 movable joints each equipped with a sensor module and a control module. The sensor module perceives the states of the robot (such as the velocity at any joint). The control module is designed based on a PID controller to generate the torque at each joint. Under the cooperation of the two modules, ROBOTIS OP3 can finish some complicated interactive tasks such as shaking hands, squatting down, standing up walking,

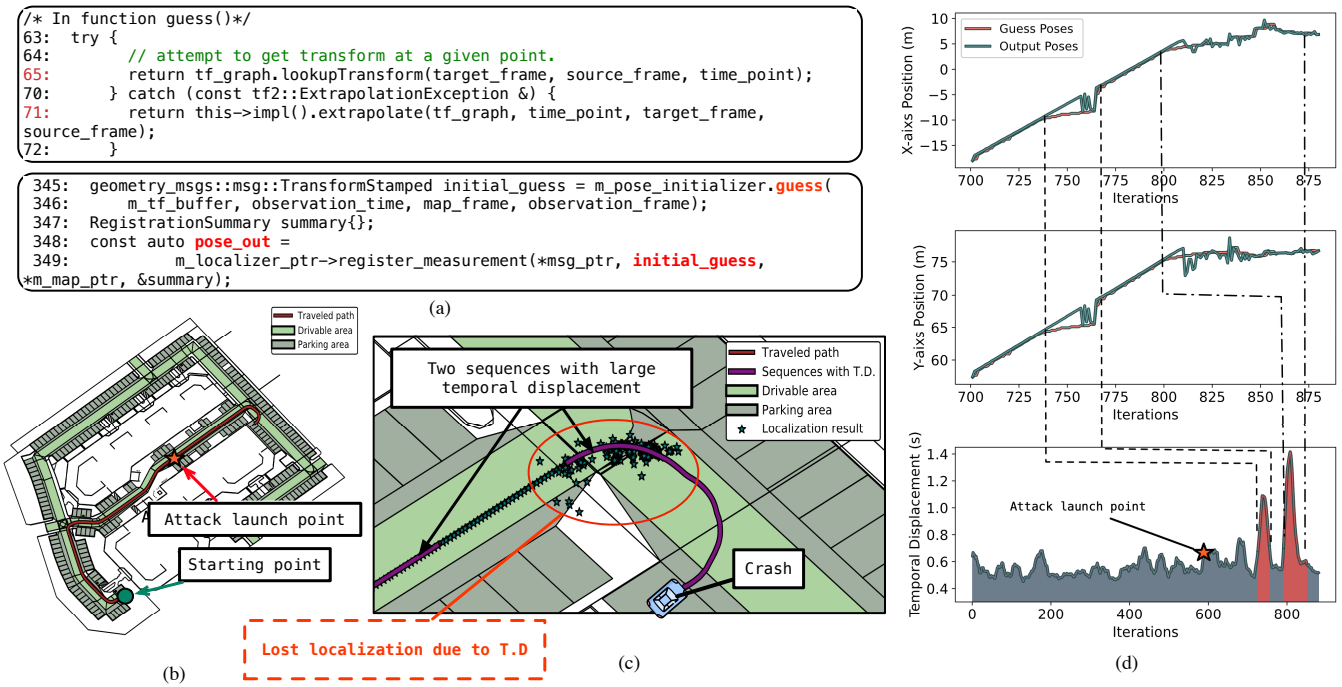
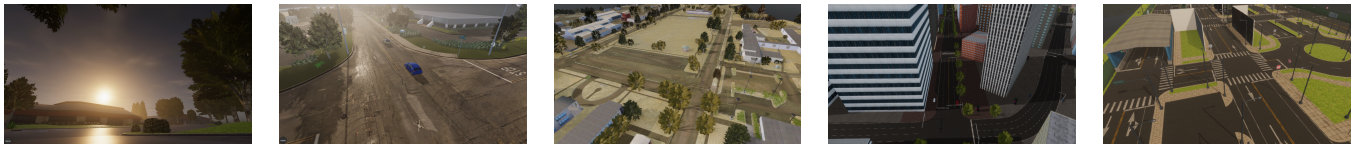


Fig. 12: Case study of Autoware.Auto. (a) Code snippets where temporal displacements happened. (b) Overall map and traveled path. (c) Zoomed traveled path and sequences with high temporal displacements that are in purple. (d) The values of two critical variables, `initial_guess` and `pose_out`, and their relations with temporal displacements.



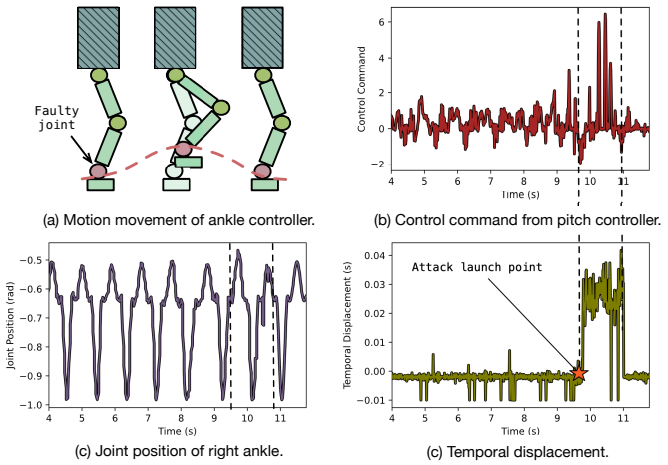
(a) Parking Lot (b) Borregas Avenue (c) GoMentum Station (d) San Francisco (e) Shalun

Fig. 13: Digital twins for testing Autoware.Auto in hardware-in-the-loop simulation environments. All those environments are simulated via modeling the real-world.

etc. We deploy the software stack of OP3 in Nvidia Jetson Nano. The simulation of OP3 is running on another PC in Gazebo. We generate a series of different tasks in advance and send them to OP3 one by one. OP3 receives the task and executes the action demanded. If it is already in the process of executing a task, any request will be ignored. During the testing, TimeTrap was randomly launched at different tasks.

Attack Result. In most cases, our attack cannot affect the stability of OP3. However, for some specific actions, such as walking, TimeTrap can effectively corrupt it, making the robot fall. Figure 14(a) shows the movement of joints in OP3. We observe that the movements of joints are highly similar and periodic with a specific time interval. However, some joints start to produce abnormal movements once TimeTrap is launched. This minor error breaches the coordination between joints, causing the robot to fall down. Once the CPU budget is beyond 5%, adversary can cause an attack outcome as shown in Figure 14.

Cause Analysis. At the control module of each joint in OP3, there are two concatenated controllers. The first one is the action controller that computes the target position a joint should reach in each control loop. The second one is the effort controller that receives the target position from the action controller and then computes the effort the servo should perform to reach the target position. Besides the input from the action controller, the effort controllers also take the sensor input of the Inertial Measurement Units (IMUs) in each joint to estimate the current position. Since they take inputs from two different sources: higher-level controllers and inertial sensors, the temporal displacement may happen within the effort controllers. From Figure 14(d), we can observe that the temporal displacement was relatively stable (around 0 ms) when it is not under attack. In this case, different joints coordinate well and move steadily. However, once the attack is launched, the temporal displacement could increase to 30ms to a point that the command calculated by the effort controller starts to oscillate (The interval marked by the dotted line in



(a) Motion movement of ankle controller. (b) Control command from pitch controller.

(c) Joint position of right ankle. (c) Temporal displacement.

Fig. 14: Case study on right ankle pitch controller of OP3. The period under attacking is fenced using dotted lines.

Figure 14(d)). As a result, the position of the right ankle joint turns out to be abnormal, which breaks the coordination between different joints such that the robot fell down.

Case Study (6) on Jackal UGV. Jackal UGV [68] is an unmanned guided vehicle that can operate in both indoor and outdoor environments. We set up the simulation world in Gazebo and run the software stack on Intel i3-8100. Simulated hardware includes the four-wheel vehicle itself and a SICK LMS111 laser. The software stack is composed of a localization module (google Cartographer [42], [85]) and a control module is implemented based on the pure pursuit algorithm [86]. We pre-defined several long mission paths in an office environment (shown in Figure 15). The Jackal UGV navigates to track the reference path.

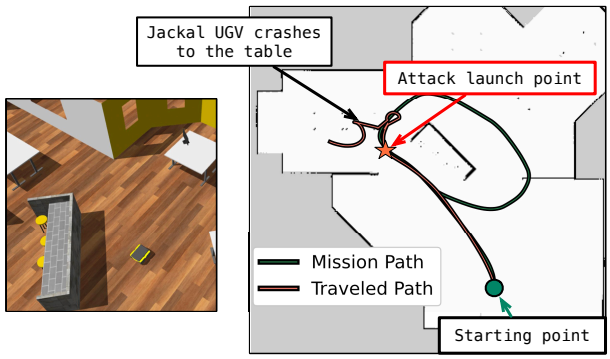


Fig. 15: Case study on Jackal UGV.

Attack Result and Cause Analysis. From Figure 15, we can observe that Jackal UGV failed to track the path and hits the table once TimeTrap is launched. In this case study, the crash was caused by erroneous location results. In Jack UGV's localization module, Cartographer, there is a sporadic task running in the background, responsible for optimizing accumulated drifts during the navigation. The optimization is supposed to release in every ten laser scans processed by the system. Since the optimization aims to mitigate the accumulated drifts, its finish will lead to a shift of localization value. The direct consequence is the vehicle's position will shift for a short distance. Such shift is negligible and within the tolerance of the controller if Cartographer can keep the task running within the implicitly (no deadline specified for this task) expected frequency. However, TimeTrap breaks the implicit deadline by constantly delaying the optimization task. Consequently, the task cannot finish in time and its workload keeps accumulating which increases the execution time in turn. After that, the vehicle's position shifts more significantly such that the controller can no longer keep tracking the reference path, leading to the crash on the table.

Case Study (7) on Baidu Apollo. Baidu Apollo [43] is an open-source full-stack self-driving project supported by Baidu. Similar to Autoware.Auto, Baidu Apollo supports multiple self-driving scenarios, including taxi, valet parking, minibus, etc. We run the entire software stack on AMD Ryzen 7 1700X with 32 GB RAM and Geforce RTX 2070s. All modules are deployed under the soft real-time scheduling framework `Cyber-RT`. The simulated vehicle model is the Lincoln 2017 MKZ. The simulation is running on another powerful PC with an AMD Ryzen 3900 CPU and a Geforce RTX 2070s GPU.

Attack Result and Cause Analysis. As a complex autonomous system, Apollo shares many similar timing problems that persist on Autoware.Auto. For example, SLAM-based localization is prone to failure. One case is that, in our testing scenario of *Borregas Avenue* (Figure 13(b)), TimeTrap can disturb the control performance when Apollo is changing lanes and then crashed on another vehicle from the adjacent lane. The crash is induced by the malfunction of the motion planning module, where unsmooth motion trajectories are generated. In Apollo, the MPC controller selects a point (called head point) from the motion path as the control input according to the vehicle's position and the current time. Consequently, large jitters can negatively impact the selection of the head point, which is the input of the control module. Under the attack of TimeTrap, the motion path was delayed periodically, causing larger jitters. Consequently, the vehicle control selected the head points with varying horizons which it cannot track. In this case, the vehicle failed to follow the current lane and its control variables started to oscillate constantly. Should there exists a vehicle passing by on the adjacent lanes, the victim's vehicle is prone to crash on it.