2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

World-in-the-Loop Simulation for Autonomous Systems Validation

Carl Hildebrandt[1] and Sebastian Elbaum[2]

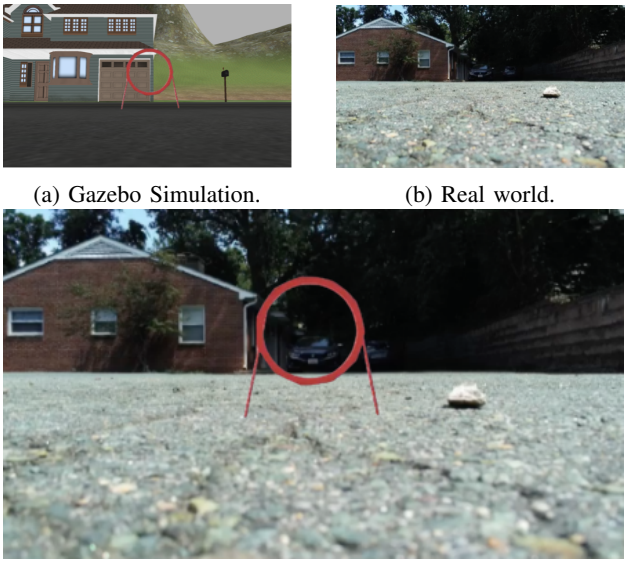
***Abstract*— Simulation is at the core of validating autonomous systems (AS), enabling the detection of faults at a lower cost and earlier in the development life cycle. However, simulation can only produce an approximation of the real world, leading to a gap between simulation and reality where undesirable system behaviors can go unnoticed. To address that gap, we present a novel approach, world-in-the-loop (WIL) simulation, which integrates sensing data from simulation and the real world to provide the AS with a mixed-reality. The approach executes multiple instances of the AS in parallel, one in the real world and at least one in simulation, performs configurable transformations, filtering, and merging operations on the body of sensed data in order to integrate it, and provides the pipelines to distribute the original sensor data and the integrated sensor data back to the executing AS. We present a study on multiple scenarios and two simulators that demonstrates how WIL reduces the simulation-reality gap and increases the chances of exposing failures before deployment.**

I. Introduction

Autonomous systems (AS) are complex entities able to make and execute decisions in the real world to achieve a goal without full or direct human control [1]–[3]. Their increasing prevalence in areas such as transportation means that their failures can have disastrous consequences [4]–[6]. To accelerate their development by detecting faults sooner and lowering the impact of failures, AS are typically validated through a simulation platform. Such simulation platforms create a virtual environment in which the AS can sense and subsequently act on the virtual world [7]. The simulation value proposition, low-cost and rapid system validation, has given rise to many simulators with a wide range of support for different AS [8]–[19].

Despite their broad adoption and cost-effectiveness, the virtual environments provided by simulators are just approximations of the real world [20]. The difference between the real and the simulated worlds is known as the simulation-reality gap [21]. For example, Fig 1b depicts an image returned from a landed drone while Fig 1a depicts a similar situation in a Gazebo simulation [22]. Note how the approximation manifests as differences in the surface's roughness, the background objects, and the lighting. This gap explains, at least in part, why field testing is still the gold standard for AS validation [23].

Developers have three options to reduce the illustrated simulation-reality gap. First, they can increase the simulation's fidelity by tweaking the simulator parameters or objects based on real data [24]–[26]. These practices can narrow the gap but are domain and AS-specific. Second, developers can use multiple and more sophisticated simulators to combine their strengths [27]. This approach can be extremely costly, not just because of the simulators' cost but also because of the cost of setting and running potentially thousands of testing scenarios across multiple simulators. Third, developers can move towards simulators that integrate richer portions of the AS. For example, migrating from a model-based simulation, which uses mathematical equations to simulate both the input and output of the system, to a software-in-the-loop (SIL) simulation, which integrates the AS code into the model allowing higher fidelity output from the system, to hardware-in-the-loop (HIL) simulation, which incorporates the AS actual hardware. These solutions reduce the gap by bringing the AS into the simulation but still require the actual world to be simulated.



(a) Gazebo Simulation. (b) Real world. (c) World-in-the-Loop (WIL).

Fig 1: The camera image sensed from a grounded drone operating in simulation (a), the real world (b), and the mixed reality created by our approach WIL by integrating both (c).

To address this challenge, we propose a new type of simulation called **world-in-the-loop simulation** (WIL). WIL aims to expose the AS to **a mixed-reality that integrates elements from simulation and the real-world.** An example of this is shown in Fig 1c, where the simulated gate sensed by the AS during simulation (Fig 1a) is integrated as part of the sensed reality of the AS (Fig 1b). This approach provides several advantages. First, it narrows the simulation-reality gap by incorporating simulated elements into the real-world. Second, it allows developers to control the information

[1]University of Virginia, USA, *hildebrandt.carl@virginia.edu*
[2]University of Virginia, USA, *selbaum@virginia.edu*

This work is supported in part by the NSF Awards #1924777 and #1853374. We thank Parrot Anafi for making their systems available.

from simulation and reality that are combined, allowing an incremental transition from simulation to reality. Ultimately, it allows developers to place their validation in the real world while reducing the cost of potential failures associated with field tests (i.e., crashing into the virtual gate in Fig 1c does not lead to drone damage).

WIL tackles two fundamental challenges. First, the AS operations, including their sensing and actuation, need to be synchronized in order for the sensor's values to be consistently integrated during execution. Second, a diversity of sensors (e.g., cameras, LIDAR's, gyroscopes, microphones), each with potentially different data types, rates, and formats, must be supported for integration. To overcome these challenges, our approach works by simultaneously running a single test in both simulation and real-world environments, pushing the AS sensor data from each environment into a framework for transforming, filtering, and mixing the data to form a new sensor-input. This new mixed-reality sensor input is fed back to the executing AS, allowing it to experience mixed-reality. The primary contributions of this work are:

1) A framework to integrate sensor readings from simulation and real-world environments and feed those into simultaneously executing environments. Our approach works using three key configurable functions that first transform the sensed data, then filter it, and finally merge it to form a new mixed-reality before feeding it back to the AS.

2) A study assessing WIL's benefits in reducing the simulation-reality gap between two simulators and a commercial quadrotor. Our findings show that there are test scenarios that pass in simulation, however, fail in reality. In these cases, WIL can detect these failing test cases, indicating a reduction in the simulation-reality gap, and it does so with a significantly lower cost-of-failure than that of real field testing.

II. Background

The validation and verification (V&V) of AS is difficult for two main reasons [17], [28]. First, the systems operate in the real world, which increases the complexity of the input space that must be constructed and checked [29], [30]. Second, the systems can display non-deterministic behavior due to both the sensor and actuator noise, as well as the underlying machine learning techniques commonly used in the perception and control layers [31]. To address these challenges, several methods and tools have been proposed.

Formal verification techniques mathematically reason whether a model of the system complies with a specified property. They have been applied to AS components such as lane change modules to verify that a lane was clear before moving into it [32], safe stop supervisor to verify that a vehicle will only activate the safe stop planner if an error is thrown [33], and a cruise control system to verify that it will never enter a collision mode [34]. They have also been applied to sub-systems that can be abstracted to high-level models [30], [35], [36]. These techniques are effective as long as accurate, and relatively small models are available, and the target properties can be checked in those available models.

Validation techniques do not guarantee that a property is met and instead generate tests that may reveal faults. Their focus is on exploring the input space defined by the systems and the world state. They do it non-exhaustively and thus can scale to full systems. For example, system tests have been generated through rapidly-exploring random trees [37], combining procedural content generation and search-based testing [38], sampling the space of kinematic models to generate stressful trajectories [39], analyzing police reports to generate environments that resemble car crashes [40], approximating the system control model to guide the command generation [41], or sampling traffic models [42].

V&V specific to machine learning components also receive significant attention as their use increases in AS perception and control layers. Validation techniques exist to generate inputs for DNNs to uncover unexpected behavior [43]–[45]. Complementary verification techniques have also emerged that guarantee that machine-learned components are robust against adversarial attacks [46]–[48]. Similar to traditional V&V, validation techniques provide no guarantees but tend to scale and can find property violations in larger systems, while the verification techniques can provide guarantees on some reachability properties but struggle to scale to larger systems [49]–[51].

While validation techniques produce inputs to exercise the AS systems, the execution of those test inputs usually requires some type of world mocking through a simulation platform. Simulators provide a range of fidelities and are used by many AS at different development stages. For example, low-fidelity simulators use mathematical models to approximate the world, and the robot states [52]. These simulators are cheap to run, making them practical for early testing. As fidelity increases, it becomes more common to model the world using either a physics or graphics engine. Such simulators are capable of SIL simulation to increase the accuracy of the robot interactions with the simulated world as devised by the system software, including the learned components. Simulators with high-fidelity physics and low-fidelity graphics [11], [12], [16] are well-suited for testing the physical behavior of new robot designs. Simulators with high-fidelity graphics and simple kinematic models [10] are well-suited for systems with rich sensor input such as cameras and LIDAR's. Other SIL simulators combine high-fidelity graphics and physics engines but tend to be limited to a specific AS or domain and be more expensive to run [8], [9]. HIL simulations, where the hardware (and sometimes the software) are used in conjunction with the simulation [53], [54] can count on a more accurate system's output. However, they tend to be AS specific, with the corresponding cost and scope limitations, and still suffer from the simulation-reality gap as the sensor readings still depend on a simulated world.

Our approach aims to address the simulation-reality gap by integrating portions of the simulated world into the real one. Two efforts are related to this concept. The first uses mixed-reality to overlay robot state and system information

in the real world [55]–[57] to help developers understand and debug systems, but it does not reduce the simulation reality gap. The second line of work explains and exemplifies how the concept of mixed reality could reduce the simulation-reality gap [58], [59], but does not provide a framework and implementation to enable the automated integration of sensed simulation and reality. In addition, our work provides an evaluation to better quantify the potential of mixed-reality to reduce the simulation-reality gap.

III. Approach

Given a world W and a goal G, the AS builds an understanding of W through sensors S, and acts on W through behaviors B to achieve G. When validating the AS, a simulation uses a virtual world W_v, which is only an approximation of W. The difference of that approximation from the real world defines the simulation-reality $Gap = diff(W, W_v)$. A key element in that gap is the semantics of the sensor readings. In the simulated world W_v, the sensor input at each time step $S_v = \{s_{v1}, s_{v2}, \dots s_{vn}\}$ is computer-generated, while in W the sensor input $S = \{s_1, s_2, \dots s_m\}$ is read from the AS sensors capturing the real world. This implies that S_v is also an approximation of S. Since different sensed values can lead to different behaviors, $B(S_v)$ and $B(S)$, ensuring that a set of desirable behavioral properties (i.e., no crashes) is met by B_v (in simulation) does not necessarily mean that B (in reality) will meet those properties.

WIL's goal is to narrow that sensing gap by creating a new mixed-reality that combines elements from S and S_v to generate S_m such that $diff(B(S), B(S_m)) < diff(B(S), B(S_v))$. WIL is depicted in Fig 2. Given a goal G, the approach simultaneously runs two instances of AS, one in W and one in W_v, collecting sensors values S from the real world and S_v from simulation, combining them to generate a mixed sensor value set S_m, and feeding that mixed reality back to both AS (we depicted just one S_m but different S_m can be sent to each execution environment).

A. Key Components

Algorithm 1 summarizes WIL. The inputs are a real and simulated AS (AS, AS_v), a real and a simulated world (W, W_v), a goal (G), and a recipe file. First, WIL subscribes to both AS. The subscriber directs all sensor readings to the $sensor\_callback$ function described in line 5. In lines 3 and 4, the real and simulated AS are started with G. The $sensor\_callback$ function receives sensor data S and S_v, from AS and AS_v respectively. It then calls a transform, a filter, and a merge function to produce S_m, the mixed reality sensor data. Finally, in line 9, S_m is published and used as input to both AS, which will then act on their worlds and sense new data to invoke the callback repeatedly. We now describe each of the functions in more detail.

1) Transforming Sensor Readings: The transformation function removes structural discrepancies between the sensor readings obtained in W and W_v. Our built-in support focuses on dimensions-units, shape, and frame-of-reference, which

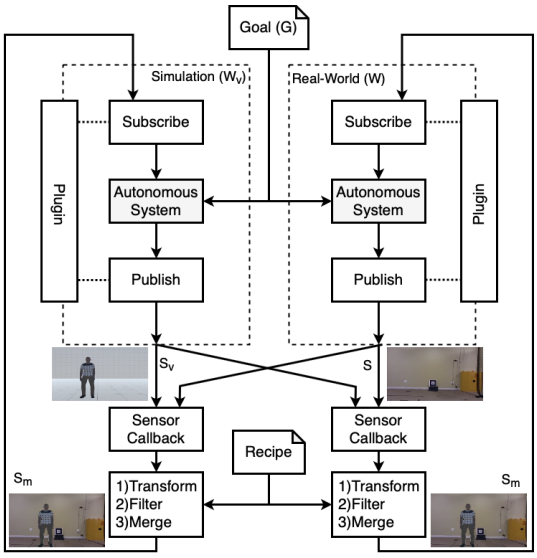


Fig 2: An overview of our approach, showing examples of S, S_v and S_m.

Algorithm 1: WIL

1 **Given** *AS, AS_v, W, W_v, G, recipe*
2 subscribe(AS, AS_v, S_m)
3 start(AS, W, G)
4 start(AS_v, W_v, G)
5 **Function** *sensor_callback(S, S_v)*
6 S_t = **transform**(S, S_v, recipe)
7 S_f = **filter**(S, S_v, recipe)
8 S_m = **merge**(S, S_v, recipe)
9 publish(S_m)

we have identified as common sources of dissonance among execution environments. In terms of units and dimensions, we found that it is common to describe the same quantity in different ways. For example, when working with GPS, some systems sensors will use the full GPS data while others return positions in terms of a local frame using X, Y, and Z. When referring to rotations, we encounter quaternions, radians, and Euler angles [60]. Even when the units are comparable, they might not use the same frame-of-reference [61]. For example, one might work using a North East Down (NED) frame and another in an East North Up (ENU) frame. In terms of shape dissonance, it is common to find simulation environments that use a different resolution to build on an existing dated component or use a lower resolution to improve performance. The transformation function enables us to overcome such discrepancies in sensor data.

For example, to produce Fig 1c, the sensor data was transformed in multiple ways. First, the quadrotor's simulated position was transformed from a local frame to a global frame that used GPS coordinates. Second, the quadrotor orientation in simulation was transformed using an offset so that the quadrotors heading in the real-world matched those in simulation. Third, the simulated camera image needed to be reshaped to match the real world camera's resolution. Finally, both cameras' sensor publishing rates needed to be matched, which we achieved by storing the latest sensor data

from both simulation and reality, allowing the merge function to access the latest data regardless of rate.

2) Filtering Sensor Readings: Filtering aims to retain the sensor readings or parts of readings that will be integrated by the merge function. It can include diverse filters, from dropping a range of values or noise from the LIDAR, to removing sets of colors from an image, similar to the techniques used when using a greenscreen. Other examples use DNN's that perform object detection and isolation [62], or background subtraction techniques to remove camera images' backgrounds. Other sensors, such as microphones, could have bandpass filters applied to isolate a range of frequencies, for example, those typical to human speech. Thus, filtering functions for the sensed values enables WIL to isolate parts of the sensor data required for merging while discarding data to reduce excess throughput and later speed up merging.

As an example, to produce Fig 1c, the camera data from the simulation was passed through a color isolation function that identified the color orange. This removed all parts of the image except for the gate which was orange.

3) Merging Sensor Readings: The merge function creates a mixed-reality set of sensor readings S_m. For example, given simulated and real-world camera data, a simulated obstacle can be overlayed over the real camera data, giving the AS the impression that an obstacle exists in the real world (as per Figure 1c). The function starts by creating an empty set of sensor readings S_m that is then populated with the mixed reality readings by applying the corresponding combination function to each sensor. The sensor data can be combined using a number of general mechanisms, including: 1) sensor prioritization, where sensor data from S_v and S is layered according to some predefined priority, 2) sensor replacement, where sensor data is replaced according to some rule, for example, the source world, and 3) sensor aggregation, where the sensor data is combined by performing some operation like average, minimum, or maximum sensor reading.

Going back to our original example in Fig 1c, we note how some of the real world camera pixels are replaced by the virtual gate pixels, allowing the AS in the real world to perceive virtual obstacles.

B. Implementation

Our implementation provides support for the collection and distribution of sensor values, which is built on top of ROS publish and subscribe architecture [63]. To be integrated with existing simulators and systems, WIL only requires the completion of a plugin to set the pipelines to distribute AS sensor values through specific message types underlying the pub-sub model. By building on ROS we also leverage its standard message types that already support a wide range of sensors such as cameras, LIDARs, or IMU's [64]. Our approach also took advantage of ROS launch files to quickly activate or deactivate the appropriate subsystems to easily switch between simulation, mixed-reality, and reality.

Another piece of the implementation worth mentioning is the support for processing $recipe$ files, which resemble launch files but with distinct tags. An example of a $recipe$ file is shown in Listing 1. The first tag is **AS sensors**, which lists all the AS sensors. When a sensor is listed, it is given a unique ID. The id allows WIL to differentiate between two sensors of the same type, for example, two cameras. Another major tag is the **combine** tag, which is linked to a sensor reading using the unique ID allocated in **AS sensors**. The combine tag links a specific file that should be used to perform each stage of our approach. For example, in Listing 1, we show that the 'color_isolation.py' file performs the filter function. This general approach allows developers to implement these functions and quickly switch different functions to create various mixed-realities for the AS.

Listing 1: An example recipe file

```
<recipe file >
    <AS sensors>
        <Camera id="camera1"/>
        <Camera id="camera2"/>
        <Lidar id="lidar1"/>
        < .../ >
    </AS sensors>
    <combine id="camera1">
        <transform> camera_transform.py </transform>
        < filter > color_isolation .py </ filter >
        <merge> prioritize_overlay .py </merge>
    </combine>
    ...
    <combine id="lidar1">
        ...
    </combine>
</recipe file >
```

The infrastructure is available at: `/git@github.com:hildebrandt-carl/MixedRealityTesting.git`.

C. Limitations

WIL makes a few assumptions that may limit its applicability and efficacy. First, WIL assumes that it is possible to match sensor specifications between simulation and reality. In recent years simulation has become more sophisticated and robust, making this a reasonable assumption for many scenarios. In our implementation, we used the simulator [12] created by the drone developers so the sensors were closely matched. Second, WIL assumes the latency it introduces through the sensor data manipulation does not affect the test results. Our implementation mitigates this risk by using low-latency communication channels combined with optimized data manipulation libraries. For example, our implementation generates mixed-reality sensor readings at 17Hz. Although slower than the drone's 60Hz camera [65], it was faster than the perception layers control loop, which operated between 5-15Hz. Third, it assumes that it has access to precise AS pose information. Inaccurate pose information could lead to a misalignment of the AS in simulation and reality, resulting in mismatched sensor information. Our implementation mitigates this risk through the use of Vicon [66], an industry-grade motion capture system with mm accuracy.

IV. Evaluation

The evaluation aims to assess the potential of WIL to reduce the simulation-reality gap and uncover the implications such as failure detection before real-world deployment.

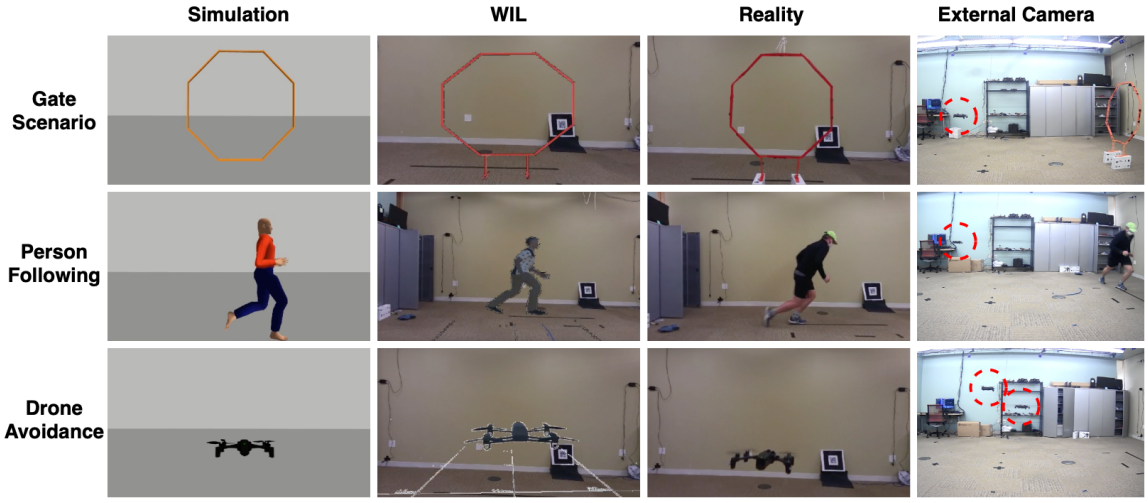


Fig 3: The camera sensor data that is fed into the AS software during simulation, mixed-reality, and reality for all three test scenarios. The final column shows an external camera with the drone highlighted in dashed lines.

A. Study Setup

For the evaluation, we are using the Parrot Anafi quadrotor [65], which weighs $0.5kg$, has a width of $0.3m$, and is equipped with a stabilized front-facing camera. We use two simulation platforms. The first one is Sphinx, a simulator developed and maintained by Parrot and built upon Gazebo [22]. Since it was developed by Parrot's engineers [12], it serves as a good baseline to characterize the simulation-reality gap. However, Sphinx does not enable read/write access to the whole sensor space, a requirement to integrate it with WIL. So as a second platform, we use a simulator built on the Unity framework [67], which has already been used for drones in the past [10], focusing mainly on the graphical aspects as the camera readings given a drone's pose. This second simulator's data is mixed with real sensor data.

We designed three distinct scenarios and goals for the Parrot to achieve. The real-world tests were run in an indoor flight cage of $6m \times 6m \times 2.5m$ instrumented with a Vicon infrared motion capture system [66] that allowed precise tracking of the drone to obtain further data for some of the scenarios. A description of each scenario is given below.

Scenario 1: The first scenario had the drone fly through a gate [68], as seen in the top row of Fig 3. To do this, we built a subsystem that uses visual cues from the camera to navigate through the gate before stopping [69]. The subsystem works by identifying the gate and navigates the quadrotor towards the center of the gate's mass using a PID controller. Orange gates with a diameter of $1m$, and $0.5m$ were selected to allow for both an easy and challenging scenario. In the challenging scenario, the quadrotor has only 10cm between the propellers and the gate's sides if it is centered correctly. The gate is placed in front of the quadrotor $3m$ away. A failure occurs if the quadrotor touches any part of the gate at any time.

Scenario 2: The second scenario was person following [70], as seen in the second row of Fig 3. We built another subsystem that used object detection based on the camera to track and follow the person. An existing object tracking algorithm, YOLO [62], generated a bounding box around the person object. The subsystem uses the bounding box's center to align the quadrotor with the person while using the bounding box's area to keep the person a set distance away. In this scenario, a person started $3.5m$ away from the drone and either walked or ran between the starting point and another point perpendicular to the quadrotor $1.5m$ away. When the drone moves outside a predefined area while attempting to follow the person, we assert a failure as it risks colliding with external obstacles or the person. The area was set such that the quadrotor was allowed to overshoot by at most $1m$ horizontally and needed to maintain between $2.25m$ and $4.25m$ away from the person at all times.

Scenario 3: The final scenario was obstacle avoidance [71], shown in the final row of Fig 3. We developed a third subsystem using camera based object detection to avoid incoming obstacles. We extended the object tracking implementation from the previous scenario and measured the bounding boxes of objects to judge whether an item was moving towards the quadrotor. If that is the case, the quadrotor will attempt to avoid it by moving upwards. In our scenario, we placed two drones $2.5m$ apart. The first drone's goal was to avoid the incoming drone. The second drone would take off after a set time, and once at the same height as the first drone, fly towards it. We developed two test cases. The first had the incoming drone reach a velocity of $0.5m/s$. The second had a velocity of $1m/s$. For this scenario, we considered the two drones colliding a failure.

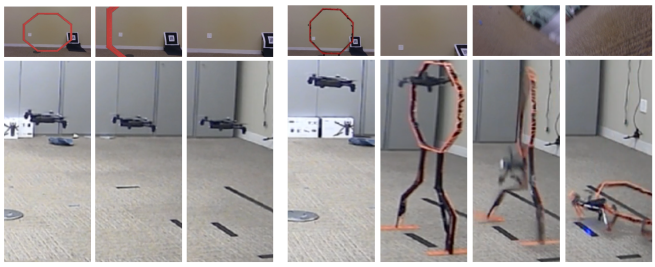
Each scenario was developed so that the quadrotor could reliably complete each of the tasks in simulation. This represents a typical development process where an AS is first perfected in simulation before real-world tests begin. After the quadrotor passed each simulation scenario, it was run in mixed-reality and then in the real world (when feasible).

B. Results

Table I summarizes the results across the three scenarios. Overall, 5 tests were run for each of the 2 variants of the 3 scenarios, resulting in a total of 30 tests. The number of passing (P) test cases and failing (F) test cases were recorded

TABLE I: Results from each scenario

Scenario	Test Case	Simulation	WIL	Reality
Gate Navigation	Large	P 5 0 F	P 5 0 F	P 5 0 F
	Small	P 5 0 F	P 1 4 F	P 1 1 F
Person Following	Walk	P 5 0 F	P 4 1 F	P 0 5 F
	Run	P 5 0 F	P 4 1 F	P 1 4 F
Obstacle Avoidance	Slow	P 5 0 F	P 5 0 F	P 5 0 F
	Fast	P 5 0 F	P 2 3 F	Too Costly



(a) Mixed reality (b) Real world.

Fig 4: Gate navigation failure in mixed reality and reality.

for each scenario. Table I shows that **although all tests pass in simulation, WIL found failing test cases. Moreover, if WIL found a failing test case, there was always a failed test case in reality. Similarly, if WIL found no failing test cases, there were no failed tests in reality.**

When considering each scenario in isolation, for example, gate navigation, we notice that the drone can always navigate through the gates without any failures in simulation. Using WIL, we find that the drone can successfully navigate through the large gate. However, for the small gate, the drone crashes 80% of the time. To further assert that the mixed reality results represented how the drone would behave in reality, the tests were repeated in the real world. We can see that for the large gate, the drone can successfully navigate through it without failure. However, when running the drone through a small gate in reality, it successfully passed once and then failed on the next attempt. After the failed test, testing was stopped due to the cost of damaging the drone.

The person following scenario produces results similar to that of the gate navigation. Moving from simulation to mixed reality and then reality, we notice cases where the drone fails. We also observe that there are more failure cases in reality than in mixed-reality. We believe this is partly due to the person's movement's variation not being modeled accurately in simulation. Consider the velocity of the person while walking. The average standard deviation in the velocity of the walking person in simulation was $1.51m/s$, in mixed-reality it was $1.32m/s$, and in reality it was $2.27m/s$. This additional variation caused the quadrotor to move outside the stipulated area to track the person in reality. Regardless of this variation, WIL still identified at least one failing instance in mixed-reality, reducing the simulation-reality gap.

The obstacle avoidance scenario shows similar trends. Failure cases start to appear in the fast drone scenario during mixed reality. However, due to the expense of failure in the real world, where a failure would likely destroy two drones, the fast test case was never attempted in the real world.

C. Implications

We found that the quadrotor's simulated behavior does not always reflect its real behavior due to the simulation-reality gap, but WIL can reduce that gap producing results more closely aligned with that of tests performed in the real world.

An expected but still worthwhile experience to mention is WIL's reduction in the cost of failures compared to field testing. Consider the scenario of navigating through the small gate. During WIL testing, the collisions are detected when overlaying the Unity simulator data on reality, where the failure is simply a boolean flag indicating a collision; this leaves the drone unharmed while flying in an empty real-world scenario, as shown in Fig 4a. This figure shows three frames with views from both the onboard camera (top) and external cameras (bottom). The quadrotor flies through the mixed-reality gate with no consequences in the real world. On the other hand, a failure in reality, shown in Fig 4b, causes the drone to physically connect with the gate in the second frame, and the propeller's momentum flips the drone causing it to land upside down. This sequence of events broke the gate's leg and damaged the drone propellers and camera.

Similar arguments can be made for the other two scenarios. For the person following, the scenario is designed conservatively to minimize the risk to humans, which would not be as necessary if using WIL. Similarly, in the incoming drone avoidance scenario, running the fast test in the real world implies two drones' potential destruction, which we deemed unworthy given WIL's results.

V. Conclusion

We have introduced a novel approach, world-in-the-loop simulation (WIL), to narrow the simulation-reality gap by integrating sensing data from simulation and the real world to provide an AS with a mixed-reality. As defined, WIL provides a framework to enable world-in-the-loop validation, and the study shows how it can assist in exposing behaviors more closely aligned to those present during field-testing, including potential costly failures. These promising findings open several avenues for future work. We will explore the generation of richer mixed-realities that incorporate more sensor types (e.g., laser scans, points clouds, microphone arrays) and sensors from multiple simulators (e.g., Carla [8], Airsim [9], FlightGoggles [10], BeamNG [19]) leveraging their diverse set of strengths. We will also grow the API to provide support for common operators and conduct studies on other systems to better characterize the potential of WIL.

REFERENCES

[1] J. Connelly, W. Hong, R. Mahoney Jr, and D. Sparrow, "Challenges in autonomous system development," *PERFORMANCEMETRICS*, p. 220, 2006.

[2] P. LeBeau, "Waymo starts commercial ride-share service," *URL: https://www. cnbc. com/2018/12/05/waymo-starts-commercial-ride-share-service. html*, 2018.

[3] M. Dikmen and C. M. Burns, "Autonomous driving in the real world: Experiences with tesla autopilot and summon," in *Proceedings of the 8th international conference on automotive user interfaces and interactive vehicular applications*, pp. 225–228, 2016.

[4] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 331–342, IEEE, 2018.

[5] Jackie Wattles, "CNN Business - Tesla on Autopilot crashed when the driver's hands were not detected on the wheel." `/https://www.cnn.com/2019/05/16/cars/tesla-autopilot-crash/index.html`, 2019. [Online; accessed 5-November-2019].

[6] Brian Garrett-Glaser, "Avionics - Drone Delivery Crash in Switzerland Raises Safety Concerns As UPS Forms Subsidiary." `/https://www.aviationtoday.com/2019/08/08/drone-delivery-crash-in-switzerland-raises-safety-concerns/`, 2019. [Online; accessed 5-November-2019].

[7] A. Harris and J. M. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *2011 Proceedings of IEEE Southeastcon*, pp. 243–249, IEEE, 2011.

[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[10] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," *arXiv preprint arXiv:1905.11377*, 2019.

[11] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Sendai, Japan), pp. 2149–2154, Sep 2004.

[12] Parrot, "Parrot-Sphinx." `/https://developer.parrot.com/docs/sphinx/whatissphinx.html`, 2019. [Online; accessed 22-August-2019].

[13] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.

[14] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, *et al.*, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *arXiv preprint arXiv:1911.01562*, 2019.

[15] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497–1504, IEEE, 2017.

[16] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS)*, pp. 595–625, Springer, 2016.

[17] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1443–1450, IEEE, 2017.

[18] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek, "Generating adversarial driving scenarios in high-fidelity simulators," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8271–8277, IEEE, 2019.

[19] BeamNG, "Research." `/https://beamng.gmbh/research/`, 2019. [Online; accessed 26-January-2020].

[20] J. C. Zagal and J. Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 19–39, 2007.

[21] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*, pp. 704–720, Springer, 1995.

[22] Gazebo, "Robot simulation made easy." `/http://gazebosim.org`, 2014. [Online; accessed 28-October-2020].

[23] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 586–597, IEEE, 2018.

[24] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.

[25] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[26] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4243–4250, IEEE, 2018.

[27] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, IEEE, 2017.

[28] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020.

[29] P. Helle, W. Schamai, and C. Strobel, "Testing of autonomous systems–challenges and current state-of-the-art," in *INCOSE International Symposium*, vol. 26, pp. 571–584, Wiley Online Library, 2016.

[30] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pp. 504–513, Springer, 2012.

[31] G. E. Mullins, A. G. Dress, P. G. Stankiewicz, J. D. Appler, and S. K. Gupta, "Accelerated testing and evaluation of autonomous vehicles via imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–7, IEEE, 2018.

[32] A. Zita, S. Mohajerani, and M. Fabian, "Application of formal verification to the lane change module of an autonomous vehicle," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pp. 932–937, IEEE, 2017.

[33] J. Krook, L. Svensson, Y. Li, L. Feng, and M. Fabian, "Design and formal verification of a safe stop supervisor for an automated vehicle," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 5607–5613, IEEE, 2019.

[34] O. Stursberg, A. Fehnker, Z. Han, and B. H. Krogh, "Verification of a cruise control system using counterexample-guided search," *Control Engineering Practice*, vol. 12, no. 10, pp. 1269–1278, 2004.

[35] G. Horányi, Z. Micskei, and I. Majzik, "Scenario-based automated evaluation of test traces of autonomous systems," 2013.

[36] Y. Lu, Y. Guan, X. Li, R. Wang, and J. Zhang, "A framework of model checking guided test vector generation for the 6dof manipulator," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4262–4267, IEEE, 2014.

[37] C. E. Tuncali and G. Fainekos, "Rapidly-exploring random trees for testing automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 661–666, IEEE, 2019.

[38] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 318–328, 2019.

[39] C. Hildebrandt, S. Elbaum, N. Bezzo, and M. B. Dwyer, "Feasible and stressful trajectory generation for mobile robots," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 349–362, 2020.

[40] A. Gambi, T. Huynh, and G. Fraser, "Generating effective test cases for self-driving cars from police reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 257–267, ACM, 2019.

[41] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "Rvfuzzer: finding input validation bugs in robotic vehicles through control-guided testing," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 425–442, 2019.

[42] A. Sarkar and K. Czarnecki, "A behavior driven approach for sampling rare event situations for autonomous vehicles," *CoRR*, vol. abs/1903.01539, 2019.

[43] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, pp. 303–314, 2018.

[44] H. Zhou, W. Li, Y. Zhu, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "Deepbillboard: Systematic physical-world testing of autonomous driving systems," *arXiv preprint arXiv:1812.10812*, 2018.

[45] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

[46] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[47] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018.

[48] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[49] D. Xu, D. Shriver, M. B. Dwyer, and S. Elbaum, "Systematic generation of diverse benchmarks for dnn verification," in *International Conference on Computer Aided Verification*, pp. 97–121, Springer, 2020.

[50] D. Shriver, S. Elbaum, and M. Dwyer, "Reducing dnn properties to enable falsification with adversarial attacks," in *2021 IEEE/ACM 43nd International Conference on Software Engineering (ICSE)*, IEEE, 2021.

[51] D. Shriver, D. Xu, S. Elbaum, and M. B. Dwyer, "Refactoring neural networks for verification," *arXiv preprint arXiv:1908.08026*, 2019.

[52] Y. Shaoqiang, L. Zhong, and L. Xingshan, "Modeling and simulation of robot based on matlab/simmechanics," in *2008 27th Chinese Control Conference*, pp. 161–165, IEEE, 2008.

[53] K. D. Nguyen and C. Ha, "Development of hardware-in-the-loop simulation based on gazebo and pixhawk for unmanned aerial vehicles," *International Journal of Aeronautical and Space Sciences*, vol. 19, no. 1, pp. 238–249, 2018.

[54] M. Odelga, P. Stegagno, H. H. Bülthoff, and A. Ahmad, "A setup for multi-uav hardware-in-the-loop simulations," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pp. 204–210, IEEE, 2015.

[55] T. H. J. Collett and B. A. Macdonald, "An augmented reality debugging system for mobile robot software engineers," 2010.

[56] F. Ghiringhelli, J. Guzzi, G. A. Di Caro, V. Caglioti, L. M. Gambardella, and A. Giusti, "Interactive augmented reality for understanding and analyzing multi-robot systems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1195–1201, IEEE, 2014.

[57] A. G. Millard, R. Redpath, A. M. Jewers, C. Arndt, R. Joyce, J. A. Hilder, L. J. McDaid, and D. M. Halliday, "Ardebug: an augmented reality tool for analysing and debugging swarm robotic systems," *Frontiers in Robotics and AI*, vol. 5, p. 87, 2018.

[58] J. Leathrum, Y. Shen, R. Mielke, and N. Gonda, "Integrating virtual and augmented reality based testing into the development of autonomous vehicles," *Proceedings of ModSim World 2018*, pp. 24–26, 2018.

[59] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5382–5387, IEEE, 2015.

[60] J.-P. Ore, C. Detweiler, and S. Elbaum, "Phriky-units: a lightweight, annotation-free physical unit inconsistency detection tool," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 352–355, 2017.

[61] ROS, "ROS transform library." `/http://wiki.ros.org/tf`, 2017. [Online; accessed 20-September-2020].

[62] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[63] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[64] ROS, "ROS common messages." `/http://wiki.ros.org/common_msgs`, 2017. [Online; accessed 20-September-2020].

[65] Parrot, "Anafi." `/https://www.parrot.com/us/drones/anafi`, 2019. [Online; accessed 11-November-2019].

[66] VICON, "Motion capture system." `/https://www.vicon.com`, 2020. [Online; accessed 20-September-2020].

[67] U. G. Engine, "Unity game engine-official site," *Online][Cited: October 9, 2008.] http://unity3d. com*, pp. 1534–4320, 2008.

[68] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[69] J.-J. Hernandez-Lopez, A.-L. Quintanilla-Olvera, J.-L. López-Ramírez, F.-J. Rangel-Butanda, M.-A. Ibarra-Manzano, and D.-L. Almanza-Ojeda, "Detecting objects using color and depth segmentation with kinect sensor," *Procedia Technology*, vol. 3, pp. 196–204, 2012.

[70] Q. Shen, L. Jiang, and H. Xiong, "Person tracking and frontal face capture with uav," in *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, pp. 1412–1416, IEEE, 2018.

[71] A. C. Woods and H. M. La, "Dynamic target tracking and obstacle avoidance using a drone," in *International Symposium on Visual Computing*, pp. 857–866, Springer, 2015.