2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

CrazySim: A Software-in-the-Loop Simulator for the Crazyflie Nano Quadrotor

Christian Llanes[1], Zahi Kakish[2], Kyle Williams[2], and Samuel Coogan[1]

***Abstract*— In this work we develop a software-in-the-loop simulator platform for Crazyflie nano quadrotor drone fleets. One of the challenges in maintaining a large fleet of drones is ensuring that the fleet performs its task as expected without collision, and this becomes more challenging as the number of drones scales, possibly into the hundreds. Software-in-the-loop simulation is an important component in verifying that drone fleets operate correctly and can significantly reduce development time. The simulator interface that we develop runs an instance of the Crazyflie flight stack firmware for each individual drone on a commercial, desktop machine along with a sensors and communication plugin on Gazebo Sim. The plugin transmits simulated sensor information to the firmware along with a socket link interface to run external scripts that would be run on a ground station during hardware deployment. The plugin simulates a radio communication delay between the drones and the ground station to test offboard control algorithms and high-level fleet commands. To validate the proposed simulator, we provide a case study of decentralized model predictive control (MPC) that is run on a ground station to command a fleet of sixteen drones to follow a specified trajectory. We first run the controller on the simulator interface to verify performance and robustness of the algorithm before deployment to a Crazyflie hardware experiment in the Georgia Tech Robotarium.**

SUPPLEMENTARY MATERIAL

The code and video that accompanies this work is publicly available through the GaTech FACTS Lab Github: `https://github.com/gtfactslab/Llanes_ICRA2024`.

I. INTRODUCTION

Advancements in micro aerial vehicles (MAVs) have enabled miniaturization of drone hardware platforms into what are now called nano quadcopters [1]. This category includes drones with a weight of approximately 30g and rotor-to-rotor distance of approximately 100mm. Nano quadcopter platforms enable rapid deployment of multi-robot aerial robotics with applications including search and rescue in dense environments [2] where larger platforms may not be able to operate, drone light shows with hundreds of drones [3], and surveying and mapping of large agricultural or forested areas [4]. Moreover, nano quadcopters are generally inexpensive to replace and store, making it cost effective to maintain large fleets of hundreds or more drones.

One of the challenges of controlling a fleet of small drones is the multi-robot scale complexity of verifying that drones will not collide and that they will accomplish their intended objective. Ideally, this verification is performed before hardware deployment and therefore typically takes place in a simulation environment. While simulations never fully capture the real-world physics, communication-constraints, and limits of real hardware, we can approximate the real-world to varying degrees. Closer approximations are achieved with higher fidelity simulations, where there is a trade-off with the complexity of the simulation. The challenge is to design an appropriately high fidelity simulator with a low enough computational complexity to enable multi-robot simulation on consumer computing hardware, with the goal being to reduce or eliminate the iterations between simulation and successful hardware deployment.

In this work, we design a software-in-the-loop (SITL) simulation pipeline that links the vehicle firmware with the simulation environment. This approach enables users to test the firmware code with simulated sensors, communication

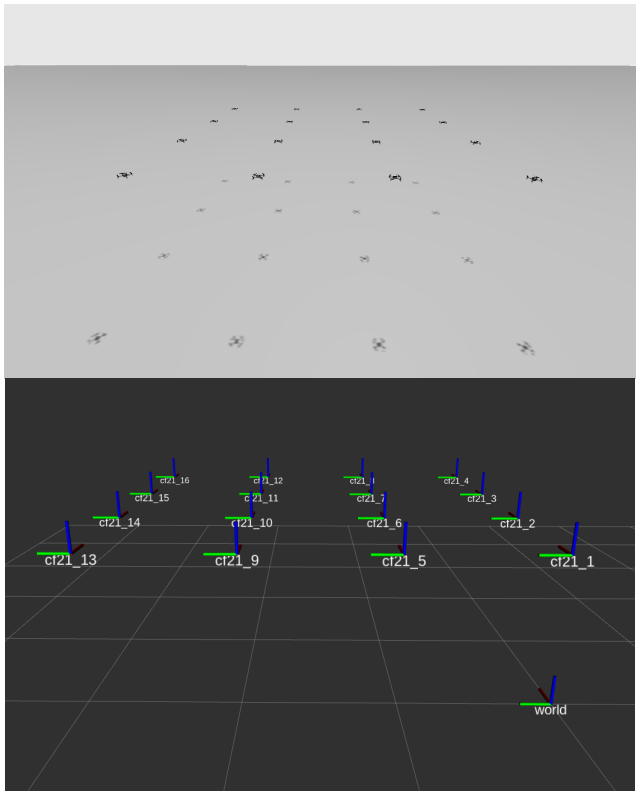

Fig. 1: Sixteen Crazyflie nano quadcopters with our proposed software-in-the-loop simulator interface hovering at a desired altitude in Gazebo Sim and RViz2.

[1]Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. Emails: {christian.llanes, sam.coogan}@gatech.edu
[2]Sandia National Laboratories, Albuquerque, NM 87123, USA. Emails: {zmkakis, kwilli3}@sandia.gov

delay, and external code to command the drones. It also provides a high fidelity framework for simulating nano quadcopter fleets. The platform used in this work is the Bitcraze Crazyflie nano quadcopter. We modify the firmware code to run on a Linux machine with a socket link for communication. We also develop a plugin for Gazebo Sim that provides a link between the simulated sensors, the firmware, and the Crazyflie Python library (CFLib) [5]. CFLib is a tool used to control the Crazyflie from a ground station using Python commands. It is also used as a backend communication library for Crazyswarm2 which provides a ROS 2 interface for operating Crazyflie drone fleets. Having the capability of using CFLib in a simulation environment with the firmware is incredibly useful for verifying that ground station command and control code designed with CFLib works as intended before deployment to hardware with minimal overhead, *e.g.*, changing the universal resource identifier (URI).

The contributions of this paper are as follows. First, we develop a simulator platform that uses the latest Crazyflie firmware and Gazebo version for a software-in-the-loop simulation with the Crazyflie firmware running in the loop on a desktop machine. The simulator platform supports a multi-robot simulation with successful tests of up to sixteen Crazyflies. The simulator platform also supports communication delay for testing radio communication delay effects on control of a multi-robot hardware deployment. To the best of our knowledge, our work is the first to implement a SITL simulator for the Crazyflie that supports running the firmware in the loop along with CFLib. Finally, as a test example that is extensible to many fleet tasks, we provide a decentralized multi-robot model predictive control (MPC) framework for tracking desired trajectories. We demonstrate the efficacy of the simulation platform by providing results for a simulation of sixteen Crazyflies running MPC on ROS 2 using Crazyswarm2 to track a set of arbitrary trajectories of varying speeds.

II. RELATED WORK

In this section, we review several existing open source multirotor simulators while addressing their limitations and how their features inspired our SITL simulator platform.

Crazyswarm2: Crazyswarm2 [6] is a ROS 2 [7] interface for operating a team of Crazyflies. It contains configuration files for setting up the interface and launch files to start the server that communicates with the Crazyflies through the Bitcraze Crazyflie Python library or Cpp Link library. The server also starts the ROS 2 flight command and parameter services, publishers, subscribers, and a simulation layer if desired. The drawback of using the simulation layer is that the dynamics integration technique is the Euler first-order method, thereby resulting in larger integration errors. Moreover, the simulator does not contain a rendering engine which makes adding vision-based sensors require a significant effort. However, this interface contains useful ROS 2 services for multi-robot control which we use in our simulation and hardware section for MPC control of multiple robots.

Webots: Webots [8] is an open source robot simulator with multiple built-in robot models and controllers and, in particular, includes an example application for the Crazyflie with a simple PID controller. This simulator supports vision-based sensors, but the example controller is significantly different from the built-in Crazyflie control pipeline. Notably, there is a Webots controller provided by Bitcraze that uses the Firmware Python bindings. Python bindings are a way to call C and C++ functions through Python; however, Bitcraze only provides the Firmware Python bindings for the controller and high-level command modules. This is a better approach than building a custom external controller to simulate the Crazyflies, but a limitation of this approach is the absence of the estimator modules from the Crazyflie and the real time operating system for the firmware that is deeply embedded in the estimation and control modules.

CrazyS: CrazyS [9] is a ROS 1 [10] simulator platform that uses Gazebo for the Crazyflie nano quadcopter. This software package is an extension to the well known RotorS [11] micro aerial vehicle (MAV) ROS 1 and Gazebo simulator. This simulator uses sensor information from Gazebo in a complementary filter to perform state estimation. However, the controller is similar to Webots as a standalone controller without dependency on the Crazyflie firmware. The interface also uses Gazebo Classic and ROS 1 middleware which are nearing end-of-development support as they are superseded by Gazebo Sim and ROS 2.

sim_cf: This repository [12] is an implementation of a software-in-the-loop and hardware-in-the-loop (HITL) simulator platform. The main feature of this interface is that it runs the Crazyflie firmware on Linux with a socket interface for communication with a custom Gazebo-classic plugin. The plugin also enables ROS 1 services for multi-robot control. This interface enables users to develop their own custom modules on the Crazyflie firmware and test them on a simulator platform before hardware deployment. The use of Gazebo also enables vision-based sensors such as optical flow and distance sensors for testing obstacle avoidance algorithms. This makes this simulator platform versatile for various simulation testing purposes. In addition, its high fidelity allows users to test algorithms on the same real time operating system that runs on the microcontroller. The HITL feature is also advantageous for testing algorithms on the microcontroller before a flight. The limitation of this interface is that it uses ROS 1 with Gazebo Classic and the Crazyflie firmware provided is approximately five years old. It also does not support interfacing with CFLib provided by Bitcraze.

PX4: The PX4 project [13] is a well known open source autopilot project for drones. It is actively supported by a large community of developers and used in hobby, academic, and industry applications. Users have extended PX4's capabilities by porting it onto Crazyflies. Additionally, PX4 has a SITL feature for running its firmware within multiple simulator platforms, namely, Gazebo Classic, Gazebo Sim, FlighGear,

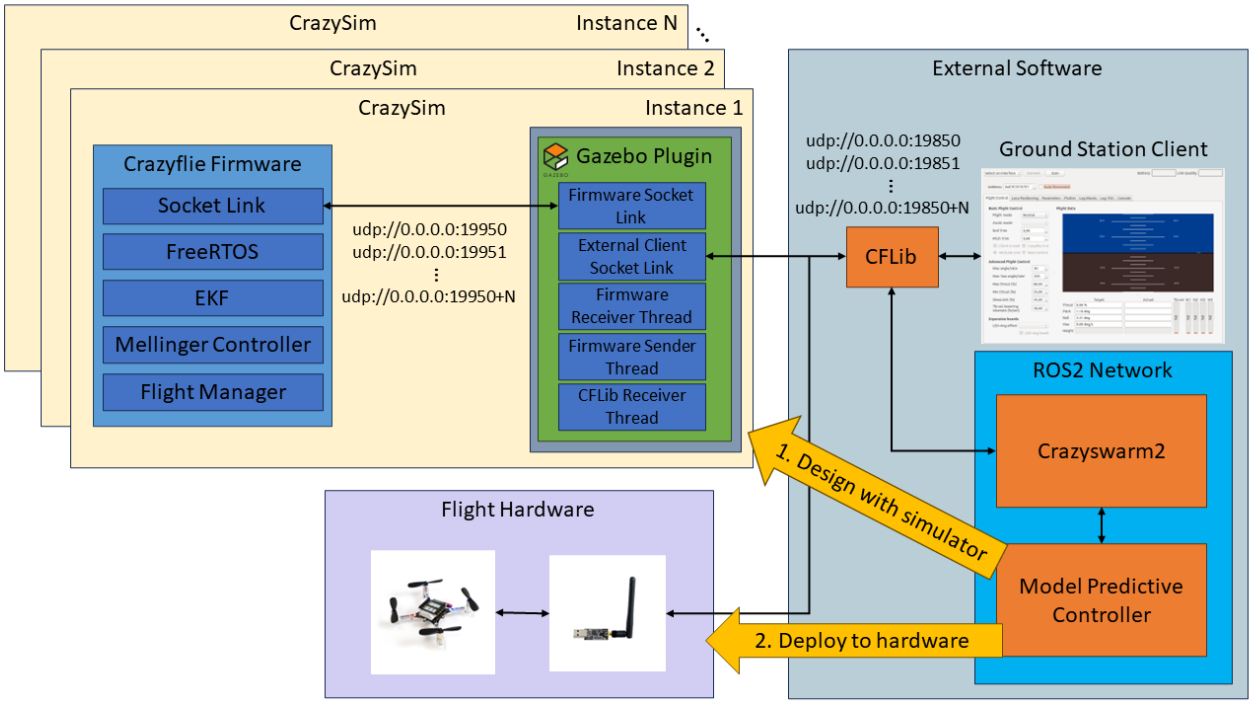

Fig. 2: Block diagram for our proposed Crazyflie testing pipeline. Our CrazySim simulator interface connects to external software such as Crazyswarm2 or the Crazyflie ground station client through the Crazyflie Python library (CFLib). Users test their control algorithms in the external software using the simulator interface before deploying to real flight hardware.

jMAVSim, and JSBSim. The SITL feature also provides support for spawning multiple robots and a bridge interface for ROS 2 support. However, the simulator uses a custom firmware rather than the official Bitcraze Crazyflie firmware and there is no support for CFLib or the Cpp link version provided by Bitcraze.

Our simulator combines the best features from each of the simulators discussed above. It is most closely an extension of *sim_cf* where we modify the latest Crazyflie firmware to remove microcontroller hardware dependencies to enable it to execute on a 64-bit Linux computer. The novelty of our work is a Gazebo Sim plugin without ROS 2 dependencies that also allows users to command the simulated drones using CFLib which is officially supported by Bitcraze. No other simulator supports connecting a SITL instance of the firmware to CFLib. This enables users to design scripts for hardware deployment and test them in simulation directly with minimal overhead requirements. Furthermore, we added a communication delay feature to simulate radio delay in hardware deployment allowing users to assess the robustness of their control scripts due to variable communication delay settings which was inspired by [14]. With our CFLib support feature, users also have the option to use *Crazyswarm2* to provide a ROS 2 interface for multi-robot control.

III. INTERFACE DESCRIPTION

In this section we discuss the simulator architecture used for our CrazySim simulator interface and implementation details on the multi-robot MPC case study.

A. Architecture

1) Crazyflie Firmware: Flight software development is typically designed around simulator wrappers that work directly with the flight software that would otherwise run onboard flight hardware. This requires software components to deal with different platforms (*e.g.* 32-bit microcontrollers or 64-bit computer processors), computing architectures (*e.g.* x64 or ARM), and sensor packages (hardware or software). The Crazyflie firmware was designed to be built for an ARM microcontroller such as the STM32F405 on the Crazyflie 2.1. There is trade-off in designing cross-platform flight software between software complexity, development time, and testing capabilities. Designing a single flight stack to run on multiple platforms may have higher software complexity, but it generally has lower development time if multiple single-target software packages are developed simultaneously for multiple platforms. Flight software with multi-platform support that is capable of running in SITL and HITL enables testing and debugging new development features before hardware deployment.

We begin with the latest Crazyflie firmware and extend the approach of *sim_cf* for a new build system for SITL. We use preprocessor macros to define when certain code in a source file should be built for SITL or hardware deployment. We initially attempted to use the Crazyflie platform build feature with *Makefiles*, however, the Crazyflie *Makefiles* are

intended for ARM microcontroller platforms. Therefore, we use a *CMake* file strictly for SITL building on a desktop machine with Ubuntu 22.04.

2) Gazebo Plugin: Gazebo Sim is chosen as the simulator engine with sensor simulation. Gazebo is a well known open-source robotics simulator with multiple sensor types such as inertial, Lidar, and camera, and it has capabilities for further user-developed sensor packages. The main requirements we need for a simulator are that it contains an inertial measurement unit (IMU), laser distance sensor, optical flow sensor, and a physics engine with efficient multi-robot support. We do not currently support optical flow or laser distance capabilities at this time, but the simulator does support these sensor types and we plan to implement these features for simulations that require them.

We implement a Gazebo system plugin to bridge the simulator and firmware together through a socket link connection. When the Gazebo server calls the plugin it initializes three threads: a Crazyflie firmware receiver thread, a firmware sender thread, and a CFLib receiver thread. The firmware receiver thread waits for a handshake message from the firmware on port $19950 + N$, where N is the robot instance index starting at zero. When the handshake message is received then the address of the receiver is stored and used in the sender thread. When the connection is established the receiver thread continues to parse packets from the firmware. One of the issues we discovered when modifying the Crazyflie firmware is that there can only be a single communication link at a time. Therefore, if we set up a socket link interface between the firmware and simulator, then we cannot use CFLib or Crazyswarm2 to communicate with the fleet to provide offboard control or high-level commands. The solution we found was to set up a relay system within the plugin so that packets that contain motor messages are directly applied to the motor model while packets that are directed to an external library such as CFLib are first driven through the communication delay feature to simulate radio communication delay. The communication delay parameters can be modified in the Crazyflie SDF model file. Message handling for a multi-producer and single-consumer queue is done through a concurrent queue [15]. For example, new sensor messages are packed and inserted into a message queue along with messages from the CFLib receiver thread. The messages are then dequeued in the firmware sender thread and sent through the socket link.

3) ROS 2 & External Software: A novel feature of our simulator interface is that it supports connecting external client software to the firmware in a simulator environment. This allows users to design scripts to control a fleet of Crazyflies using software such as the Bitcraze CFLib API, Crazyswarm2, or the Bitcraze Crazyflie C++ link. Users can test and verify their scripts directly in a simulator environment before a hardware deployment. Prior research development with the above software tools involved designing scripts and testing directly on hardware. This often leads to unnecessary crashing of drones during experimental testing of new scripts. This is especially pertinent for researchers that are new to drone research, Crazyflie software tools, or designing complex offboard controllers that are robust to communication delay. A feature in our simulator interface is that we do not have a direct dependency on ROS 2. This was intended for users that are not familiar with the ROS 2 interface, but are familiar with the CFLib API. For users that want to add ROS 2 support for their drone fleet, then we recommend using the Crazyswarm2 software package because it supports CFLib as the backend communications interface. This also adds RViz2 support for visualization of your drone fleet in simulation or during hardware deployment. The SITL firmware fully supports parameter and logging features and those can be modified in the Crazyswarm2 configuration files.

B. Model Predictive Controller

In order to demonstrate the capabilities of our simulator interface we provide a case study of a trajectory tracking example that uses model predictive control (MPC) to track a set of temporally parameterized functions of position and velocity. A main component of MPC is choosing a model that mathematically characterizes a physical description of the dynamics of the real system. This involves a trade-off between complexity and fidelity of the model. Complex models are typically difficult to solve in real time and are even more challenging to use when we need to meet specific time constraints such as running the controller at 100Hz. Therefore, [16], [17] uses a reduced-order model with assumptions about the higher-order dynamics. One example is time-scale separation with slower dynamics being used for the reduced-order model and allowing low-level controllers to handle the faster dynamics. Quadrotor dynamics are typically modeled using twelve states and a control input for each motor. Controlling motors directly from a ground station with communication delay would be challenging and is also not officially supported by Bitcraze firmware and client software tools. Our proposed solution is to allow an onboard inner-loop PID controller to track desired attitude angles. We assume the realized rotational dynamics for the inner-loop system are a first-order linear system with time constant τ and an angle command that is applied to the inner-loop controller. Define

$$
\begin{aligned}
\dot{p}_x &= v_x \\
\dot{p}_y &= v_y \\
\dot{p}_z &= v_z \\
\dot{v}_x &= [\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi]u_T \\
\dot{v}_y &= [\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi]u_T \\
\dot{v}_z &= [\cos\phi\cos\theta]u_T - g \\
\dot{\phi} &= \frac{1}{\tau_\phi}(\phi_{\text{comm}} - \phi) \\
\dot{\theta} &= \frac{1}{\tau_\theta}(\theta_{\text{comm}} - \theta) \\
\dot{\psi} &= \frac{1}{\tau_\psi}(\psi_{\text{comm}} - \psi)
\end{aligned}
\tag{1}
$$

as the reduced-order model for the Crazyflie that is used

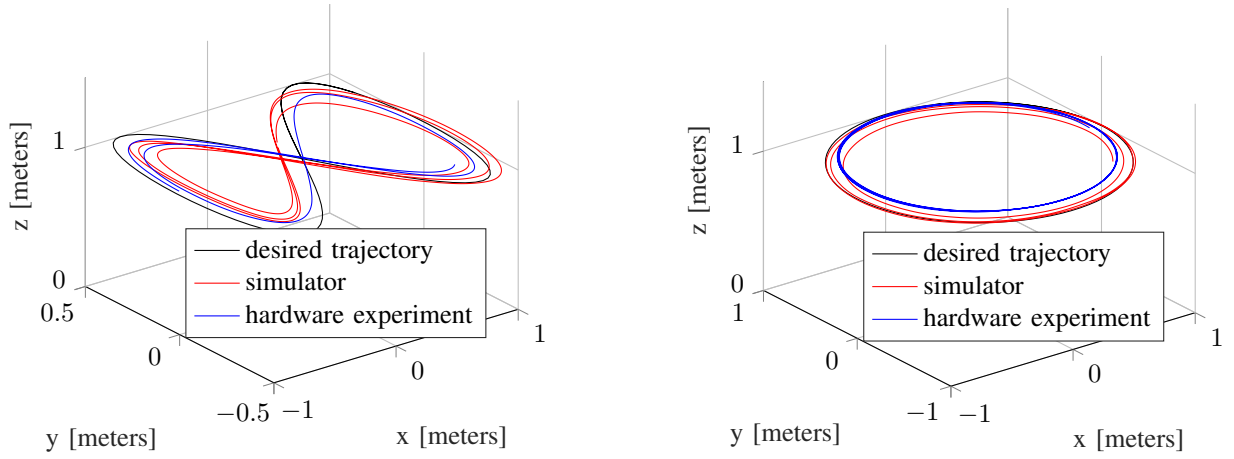

Fig. 3: A comparison of our simulator to a real flight hardware experiment on a Crazyflie 2.1 nano quadcopter for a desired (a) lemniscate trajectory and (b) circular trajectory.

for the MPC formulation with position and velocity states p and v respectively in the three dimensions. To describe the orientation of the quadrotor we use Euler angles defined as ϕ, θ, and ψ for roll, pitch, and yaw respectively. The control inputs to this system are the commanded attitude angles ϕ_{comm}, θ_{comm}, and ψ_{comm} to the inner-loop controller and mass-normalized collective thrust u_T. For the MPC problem we seek to minimize the linear least squares objective function

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N-1} \|y_i - y_{i_{\text{ref}}}\|_W^2 + \frac{1}{2}\|y_N - y_{N_{\text{ref}}}\|_{W_N}^2 \tag{2}$$

with $y = [p_x, p_y, p_z, v_x, v_y, v_z]$, reference $y_{\text{ref}} \in \mathbb{R}^{n_y}$, weighing matrix $W \in \mathbb{R}^{n_y \times n_y}$ from the weighted L2-norm, i.e. $\|x\|_W^2 = x^T W x$, and terms with N denote parameters in the terminal cost.

IV. SIMULATION AND HARDWARE DEPLOYMENT

In this section, we provide a case study for our simulator interface that compares simulation to hardware flight tests for a Bitcraze Crazyflie 2.1 nano quadcopter.

A. Simulation & Hardware Results

To test our simulator interface we use a lemniscate reference trajectory defined as

$$\begin{aligned} x(t) &= \sin(\omega t) \\ y(t) &= \sin(\omega t)\cos(\omega t) \\ v_x(t) &= \omega\cos(\omega t) \\ v_y(t) &= \omega\cos(2\omega t) \end{aligned} \tag{3}$$

and a circular reference trajectory, both with time t and angular rate ω. Then, we let the MPC compute attitude and thrust commands to the Crazyflie at a rate of 100Hz. For both trajectories we start the angular rate parameter at zero and increment slowly to a desired steady-state value using the function, $\omega = 0.75\tanh(at)$ with tuning parameter a. This approach reduces the error in the desired and current velocity at the beginning of the test when the vehicle is hovering with zero velocity. For our tests we use [18] to solve the MPC problem with a prediction horizon of $T = 3\,\text{s}$ and $N = 20$.

To start the simulation we spawn the desired number of Crazyflies on Gazebo, launch Crazyswarm2 to provide the ROS 2 interface, and then launch our ROS 2 decentralized MPC nodes. With everything set up we finally publish a takeoff and start trajectory message to the MPC node and start collecting data. In our tests we command a fleet of sixteen drones to track a tilted circular reference trajectory. In Fig. 5 we provide a snapshot of the test in the RViz2 interface with the $T = 3\,\text{s}$ horizon look ahead path computed by MPC for each drone in green. We also provide a plot with the trajectory of each drone.

We then conduct a real flight hardware demonstration from the Robotarium at Georgia Tech as pictured in Fig. 4 with the MPC controller for the two reference trajectories and provide plots in Fig. 3. The simulation and real hardware flight demonstration are both capable of tracking the reference trajectory. Note that the main component of this paper is the simulator interface and its capability to test control algorithms and obtain similar results to hardware deployment. Therefore, we do not make a deep dive into comparing the performance of MPC.

B. Computation Time

A drone fleet simulation requires running an instance of the Crazyflie firmware and Gazebo Sim plugin for each robot in the fleet. To study how drone fleet sizes affects the simulator performance we provide a bar graph in Fig. 6 that displays the real time factor from Gazebo versus the drone fleet size. This test is conducted using a 16-core AMD Ryzen 9 5950x desktop processor. In the current stage of the interface the Gazebo simulator side runs at an independent

Fig. 4: A hardware demonstration of a Crazyflie nano quadcopter following a circular trajectory using our model predictive controller on ROS 2.

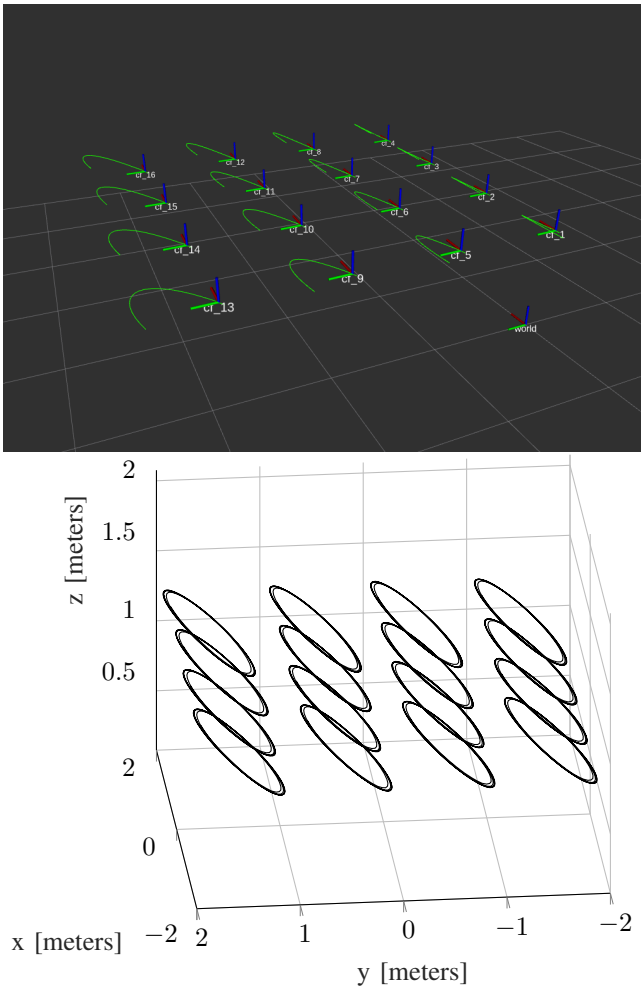

Fig. 5: Sixteen Crazyflie nano quadcopters with our proposed software-in-the-loop simulator interface commanded to track a tilted circular trajectory using MPC.

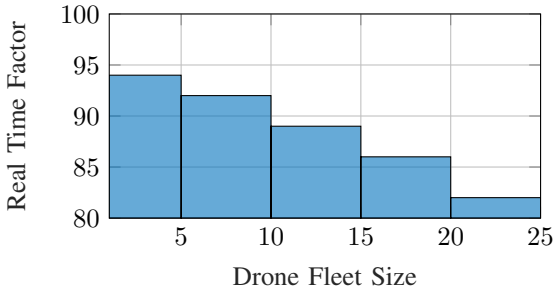

Fig. 6: A bar graph displaying the real time factor from Gazebo as a percentage for a number of drone fleet sizes using a 16-core AMD Ryzen 9 5950x desktop processor. A percentage of 100 indicates that the simulation clock is at real time speed.

time as the firmware. This means that there is no lock-stepping to synchronize the firmware rate with the Gazebo sensors. We recommend running at a real time factor above 80% for stability of the estimator and control module.

V. CONCLUSIONS & DISCUSSIONS

This work presents a software-in-the-loop simulator interface for a fleet of Crazyflie nano quadrotors. We derived our work by analyzing the limitation and important features of other available simulators. Our work differs in that it runs the firmware in a desktop machine with simulated sensors, runs a Gazebo Sim plugin interface with simulated communication delay, and the interface is capable of communicating with external client software for the Crazyflies using CFLib. The goal is to allow researchers that use the Crazyflie to test their code in a simulated environment before a hardware deployment to a drone fleet. This ensures that simple bugs can be caught early without crashing drone hardware. This also enables users to do development in the firmware such as developing a new estimator or controller and testing the development code in a simulator before deploying to hardware.

VI. ACKNOWLEDGEMENTS

We thank the Georgia Tech Robotarium for allowing us to use the space to conduct the real flight hardware experiments. This work was supported in part by the National Science Foundation under awards #1749357 and #1924978, by the NASA University Leadership Initiative under grant #80NSSC20M0161, and by Sandia National Laboratories. This article solely reflects the opinions and conclusions of its authors and not any NASA entity or Sandia National Labs. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, pp. 37–42.

[2] K. N. McGuire, C. D. Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, p. eaaw9710, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aaw9710

[3] J. Huang, G. Tian, J. Zhang, and Y. Chen, "On unmanned aerial vehicles light show systems: Algorithms, software and hardware," *Applied Sciences*, vol. 11, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7687

[4] S. Karam, F. Nex, O. Karlsson, J. Rydell, E. Bilock, M. Tulldahl, M. Holmberg, and N. Kerle, "Micro and macro quadcopter drones for indoor mapping to support disaster management," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, pp. 203–210, 2022.

[5] "cflib: Crazyflie python library," https://github.com/bitcraze/crazyflie-lib-python, 2023.

[6] "Crazyswarm2," https://github.com/IMRCLab/crazyswarm2, 2023.

[7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[8] "Webots," https://github.com/cyberbotics/webots, 2023.

[9] G. Silano and L. Iannelli, *Robot Operating System (ROS): The Complete Reference (Volume 4)*. Springer International Publishing, 2020, ch. CrazyS: a software-in-the-loop simulation platform for the Crazyflie 2.0 nano-quadcopter, pp. 81–115.

[10] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[11] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23

[12] "sim_cf," https://github.com/wuwushrek/sim_cf, 2018.

[13] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[14] B. Barros Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo, "An efficient real-time nmpc for quadrotor position control under communication time-delay," 12 2020, pp. 982–989.

[15] "Concurrentqueue," https://github.com/cameron314/concurrentqueue, 2023.

[16] M. Faessler, D. Falanga, and D. Scaramuzza, "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 476–482, 2017.

[17] P. Foehn and D. Scaramuzza, "Onboard state dependent lqr for agile quadrotors," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6566–6572.

[18] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021. [Online]. Available: https://doi.org/10.1007/s12532-021-00208-8