Fleet2D: A fast and light simulator for Home Robotics

Apaar Sadhwani*
Amazon Lab126
Sunnyvale, CA, USA
apaar@amazon.com

Hamid Badiozamani*
Amazon Lab126
Sunnyvale, CA, USA
badiozam@amazon.com

Tushar Agarwal
Amazon Lab126
Sunnyvale, CA, USA
tagarw@amazon.com

Saraswathi Marthandam
Amazon Lab126
Sunnyvale, CA, USA
marthsar@amazon.com

Amin Atrash
Amazon Lab126
Sunnyvale, CA, USA
amatrash@amazon.com

Jing Zhu
Amazon Lab126
Sunnyvale, CA, USA
jngzh@amazon.com

Aarthi Raveendran
Amazon Lab126
Sunnyvale, CA, USA
raveendr@amazon.com

William D. Smart†
Oregon State University
Corvallis, OR, USA
smartw@oregonstate.edu

***Abstract*—Home robots operate in diverse and dynamic environments, delivering a range of functions that enhance utility. Many of these functions span extended periods, from weeks to months, typically improving through observations and interactions. Efficient development and validation of these functions necessitate simulations that can run faster than real time. However, many current robot simulators focus on high-fidelity physics simulation that limits their speed to a small multiple of real time. While these are tailored for critical low-level functions, they aren't optimized for simulating higher-level functions such as learning human behaviors and interacting with them. In this work, we introduce Fleet2D, a fast, lightweight simulator designed for long-term human-robot interactions in the home. By abstracting away low-level physics, aggressively caching compute-intensive operations, and operating in a simplified two-dimensional world, we are able to perform realistic simulations of robot behavior at more than 10,000 times real time. We present the design, development, and validation of Fleet2D, showcase its effectiveness on a number of use cases in home robotics, and discuss how it has accelerated the development cycle for a home robot. Finally, we make Fleet2D open source for the research community at `github.com/amazon-science/fleet2d`.**

***Index Terms*—Simulation, Robotics, HRI**

I. Introduction

The burgeoning field of home robotics represents a departure from traditional, industrial robotics in significant ways. Unlike industrial robots, which predominantly function in controlled, well-structured environments, home robots are engineered to operate in a wide variety of unstructured, dynamic environments. Home robots, such as Astro (Fig. 1) [1] and Jibo [2], are expected to adapt to an ever-changing range of circumstances through learned behaviors and to interact meaningfully with human beings. This intrinsic complexity presents unique challenges for feature development in home robots: simulating a *diverse* set of scenarios over *long time horizons*. From initial prototyping to model fine-tuning, testing and quality assurance, simulation is an indispensable tool to ensure that robots behave safely and effectively in these nuanced environments.

Fig. 1. Astro [1] is a household robot for entertaining users and home monitoring.

Most contemporary simulators have been developed with an emphasis on intricate, short-timescale details. They prioritize capabilities such as high-fidelity physics engines and precise sensor simulations. This level of detail is undoubtedly required for critical low-level functions such as sensing and manipulation, much of which directly interacts with raw sensor data. Yet, when it comes to broader functionalities inherent to home robots—like estimating where in the home users might be—these simulators often become limiting. They tend to be slow, computationally intensive, and necessitate the specification of low-level details, which can be cumbersome in practice.

Consider a high-level function such as long-term human presence [3] that estimates the rooms users frequent. It is a key ingredient to create opportunities for human-robot interaction (HRI) [4]. At a low level, a perception model identifies users whenever they are in the robot's field of view. The presence layer then aggregates perception signals over weeks and months to develop a robust estimate. To develop human presence algorithms, the high-fidelity simulation of perception is not critical and efficient heuristics should suffice. Essen-

*The authors contributed equally to this work.
† Work done as a visiting scholar at Amazon Lab126.

tially, developing such high-level functions in home robotics requires long-term simulations, where the details are abstracted without compromising the essence of the scenarios. *Fleet2D*, a Fast lightweight environment for experimentation in 2D, fills this gap.

The primary contributions of this work include:

- We delve into the design and architecture of Fleet2D. This also serves as a template for the design of HRI simulators.
- We compare Fleet2D with existing simulators and find it to be extremely fast and light.
- We employ Fleet2D in the development of a home robot and showcase its versatility across multiple use cases.
- We make Fleet2D open source for the research community at `github.com/amazon-science/fleet2d`.

II. Related Work

The use of simulation in robotics has become an indispensable tool for researchers and developers, providing a controlled environment to prototype, test, and refine algorithms before deploying them on physical robots [5]. In the realm of home robotics, where robots are expected to operate in diverse and often unpredictable environments, simulation plays a pivotal role. It helps recreate, both, a large variety of scenarios and identified edge cases where the robot must remain performant. For HRI applications, simulation is used to refine long-term robot behavior to better align it with human expectations [6].

Before embarking on building Fleet2D, we surveyed a number of existing robot simulators including Gazebo and Unreal. Robotic simulators can be roughly categorized into two groups: three-dimensional (3D) and two-dimensional (2D). The 3D simulators typically offer higher fidelity and broader applicability, making them a more popular choice. We look at these next and close this section by noting general trends in the area of long-term simulation.

3D simulators: Numerous multiagent robotics simulators exist today, emphasizing high fidelity in dynamics, kinematics, inverse kinematics, and collision detection across all major areas of robotics. Notably, Gazebo [7] stands out as a fully-featured 3D open-source simulator integrated with ROS 2, while Webots [8], another comprehensive 3D open-source simulator, is distinctive for its refined ODE physics engine. On the other hand, PyBullet [9] is an open-source simulator optimized for sim-to-real transfer, running on the Bullet Physics engine. Additionally, CoppeliaSim [10], formerly known as V-REP, is a versatile closed-source simulator that supports multiple physics engines and integrates seamlessly with ROS. The overarching aim of these simulators is to maintain precise physical models to smoothly transition algorithms from the simulation to real-world robotic applications. In the realm of HRI, there are several niche solutions, such as simDRLSR [11], which is geared towards learning reinforcement learning policies. It is focused on vision signals and short-term interactions, making it too computationally heavy for our purpose.

[12] makes a comparison across multiple 3D simulators. Common to all these is the goal of simulating with high fidelity. We are neither concerned with this level of fidelity, nor in directly transferring a motion or perception algorithm to the robot. Instead, we wish to study long-running HRI algorithms by simulating our robot's interactions with humans over long periods of time. Thus, our simulator is concerned more with behavioral contact points (or lack thereof) rather than physical ones.

2D simulators: Given the high-level behaviors we seek, 2D simulators are likely to offer a better balance of speed and fidelity. Stage [13] is a 2D multi-robot simulator that's part of the Player/Stage project. Stage provides collision detection, odometry, and sensing of range-finders among other capabilities. Unfortunately, it is no longer maintained. The Robotics Toolbox for MATLAB [10] provides a collection tools for robotics and automation, primarily intended for robot kinematics, dynamics and motion planning. For a more encompassing experience, Microsoft Robotics Developer Studio (MRDS) [14] stands out as a 3D simulator with rich tooling that also handles 2D robotic simulations. Nevertheless, it's worth noting that these simulators aren't specifically tailored for home robotics, and their interfaces necessitate specifying intricate low-level details, making adoption more challenging.

Long-term simulation: We note that simulation capabilities can be looked at through the lens of an operating point on the Pareto frontier trading off between speed and fidelity. The crux of Fleet2D is the desire to speed up simulation drastically while sacrificing the fidelity of low-level functions that are not critical for our long-term scenarios. We observe a similar trend in other areas, such as in LimSim [15], a long-term multi-scenario traffic simulator for urban road networks that contrasts with other fine-grained simulators in its domain. Like Fleet2D, LimSim emphasizes both diverse scenarios and long-running simulations.

III. Design

The design of Fleet2D is motivated by several use-cases, which are discussed next. We work backwards from these to identify guiding principles and capabilities for the simulator. Finally, we document key design decisions that are baked into the simulator to maintain alignment with these principles.

There are two reasons to document this process. First, it provides the reader with a deeper understanding of both the capabilities and the limitations of Fleet2D. Second, more generally, this process provides a template for designing HRI simulators which model the world at an appropriate level of complexity. This is often cited as a major barrier for the use of simulation in robotics [5].

A. Use-cases

We outline four categories of use-cases that motivated the design of Fleet2D. While there may be several use-cases within each category, we use examples related to human presence (defined below) as a common thread through the paper.

Prototyping: Developing new algorithms and fine-tuning existing ones require simulating diverse scenarios and the ability to conduct large-scale experimentation over these.

Consider the problem of estimating human presence. *Long-term presence* (LTP) [3] estimates spaces inside the home where users typically reside. It is concerned with the long-term averages. LTP has a burn-in period of weeks to months, over which its estimate improves by aggregating the sparse human observations the robot collects daily. It is a key driver of HRI, for example, by allowing the robot to hang out in non-intrusive spots easily accessible to its users [4]. Prototyping LTP algorithms requires simulation over 30-60 days, and generating scenarios with a gamut of floorplans, human activity schedules and robot behaviors. Fine-tuning LTP models adds an outer loop to search over the model (hyper)parameters. In sum, this requires long-term simulations and large-scale experimentation. Other variants of presence, such as *Real-time Presence* (RTP) [16], operate on shorter time horizons but still require experimentation over a wide variety of scenarios and model parameters.

Cross-functional interaction: Many features of the robot are developed in parallel. Even so, they must ultimately function seamlessly when integrated. This integration can lead to *interaction effects*. For example, LTP suffers from poor performance in the burn-in period. Another feature, *Find* [17], is responsible for finding a user and leverages LTP. This dependency implies that the burn-in performance of Find must also be tested over the same horizon as LTP.

In other cases, interactions can manifest in unexpected ways. For HRI functions in particular, this can lead to feedback loops that are critical to identify and mitigate early. Consider *Hangout* [4], which seeks to place the robot in the frequently-visited spaces provided by LTP. This action, in turn, accelerates the learning of LTP – a feedback loop. To leverage it, however, LTP updates must now be carefully designed; specifically, LTP approaches that assume independent human-robot motions are prone to catastrophic failures. To mitigate this, LTP and Hangout must be tested together early in prototyping. [5] cites facilitating the understanding of HRIs an underutilized opportunity for robotic simulation.

We next discuss a broader category of use-cases that emerged outside of the prototype-develop cycle. One employs the simulator for testing at the product stage (post-development) and the other for generating training data for ML models (pre-prototyping).

Manual Testing: Aside from testing at the algorithm development stage, Quality Assurance (QA) teams perform thorough product testing. This requires identifying unit tests that may be run automatically instead of in a time-consuming, manual fashion on the robot [18]. Enabling this requires fine-grained control over the scenario, long-term simulation and reproducibility for features such as LTP.

Data Generation: Several HRI capabilities employ learning-based methods. Machine learning (ML) models typically require large training datasets that are expensive to acquire manually. An alternative route is to train using vast amounts of synthetically generated data and to then fine-tune on a small amount of manually collected high quality data. [5] cites the use of robotic simulation for generating data for ML models as a key opportunity. As a specific example, we wish to employ Fleet2D for generating training and evaluation data to develop RTP models. Such large-scale data generation requires long-term simulation, diverse scenarios, and reproducibility of the evaluation dataset.

B. Capabilities

We organize the common capabilities across different use-cases to prioritize these in the simulator design. These are summarized in Table I.

Long-term simulation is a capability that many features require, forming the cornerstone of our simulator. Specifically, it refers to simulations spanning multiple months. We caution that this may *not* generalize to the robot functionality (or even HRI features) in another context.

Scenario diversity is crucial for training and fine-tuning models. It helps identify gaps early through exhaustive coverage of the domain space. For model training, it ensures ML models assimilate a broad spectrum of priors. In our case, this encompasses variations in three vital components of Fleet2D: floorplan, human activity schedule, and robot behavior, along with algorithm-specific parameters to be fine-tuned.

Reproducibility is an essential feature for HRI research and will serve as a guiding principle in the simulator design.

Manual adjustment pertains to the capability to manually configure a scenario based on high-level semantics. A key ingredient for enabling this is providing a simulator interface at the appropriate level of abstraction.

TABLE I
LIST OF CAPABILITIES FOR A SUBSET OF GENERAL HRI FEATURES.

	Capabilities			
Feature	Long-term simulation	Diverse scenarios	Reproducibility	Manual adjustment
Prototyping				
LTP [3]	✔	✔	✓	
RTP [16]		✔	✓	
Cross-functional interaction				
Find [17]	✓	✓	✔	✓
Hangout [4]	✔	✓	✔	✓
Manual testing	✔		✔	✔
Data generation	✔	✔	✔	

✔= Requirement ✓= Desirable feature

C. Guiding principles

Based on the capabilities we seek, we prioritize the following characteristics in the simulator architecture:

Fast and light: High-speed simulation is a top priority as many use-cases need *both* long-term simulation and diverse scenarios. Roughly speaking, we seek a speedup on the order of $10,000$x to allow performing 30-day simulations in under 5 minutes on a PC without specialized hardware acceleration. This is highly desirable from the standpoint of quick and iterative prototyping.

Consider a large-scale evaluation with 10 variants each for the floorplan, robot and human motion. This would result in $1,000$ variants overall, requiring about 3.5 days of compute that may be executed in parallel. Fine-tuning models would incur additional expense due to the model parameters. For example, a minimal search over 10 hyperparameters would require over 35 days. We note in passing that this precludes the use of ROS-based simulators that primarily target distribution of a single (compute-heavy) experiment over multiple machines. There the (de)serialization of messages alone would slow the simulator down to significantly below the $10,000$x threshold.

Reproducibility: We seek to bake reproducibility into the design of Fleet2D. Seeded experiments must be reproducible. Further, even when randomness is desired (and no seed is set), the design ensures that traceability is maintained to reproduce any run.

Modularity: We seek a modular design that is easy to follow, extend, and stitch with external modules for developers. Further, we seek to reflect this modularity in configuration files to enable a facile interface with high-level semantics. We make an effort to maintain this simplicity - a benefit to all users - over expanding functionality to narrow use-cases.

D. Key Decisions

We end this section by identifying key design decisions baked into Fleet2D.

Abstracting low-level functionality: is an important mechanism through which we increase speed, lower compute and reduce complexity. At a high level, our strategy has been to significantly simplify the compute-heavy operations using, say, data-driven heuristics.

- **Mimic perception signals**: The fine-grained details of human detection models do not matter to the higher-level robot functions we described in Table I. Rather, what matters is if the user was detected, identified and their high-level attributes (e.g. location, pose, activity). We mimic this process probabilistically, by randomly creating false positive/negative detection/identification as a function of the relative distance and orientation. We match these rates to the reported on-device error profile. Similarly, we mimic the robot's visual perception in both its angular and distance range (min and max).
- **2D world**: Once the above heuristics are employed, the value of operating in a 3D simulation world quickly diminishes, especially given the significant compute it takes.

Extensive caching: Even after abstracting low-level complexity using heuristics, there are important compute-heavy operations such as FOV computations (in 2D) that still remain. An option in Fleet2D allows precomputing the visibility polygons [19] for a given floorplan (see Fig. 2). The experiments show an order of magnitude speedup by leveraging this cache. Note that visibility polygons depend only on the floorplan, and crucially, not on robot characterisitcs. This permits scalable experimentation over robot parameters.

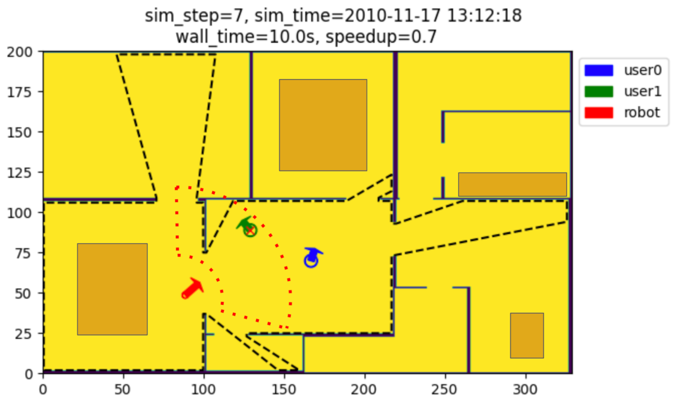


Fig. 2. An illustrative example of the 2D world in Fleet2D, here with 1 robot, 2 users and a synthetically-generated floorplan. The (black) dashed line corresponds to the visibility polygon from the robot's x-y location, which is agnostic to the robot's orientation. The (red) dotted line shows the heuristic used to approximate the range of the robot's visual perception. An entity must be within both these regions to be observable by the robot.

Hierarchical components: We adopt a hierarchical component-based design across all components (to be described shortly). Adding new components entails creating new classes under the relevant component. Modifying behaviors entails subclassing components. Each component is separately configured from its specific section within the configuration file (Fig. 4).

IV. Architecture

A. Organization

The high-level framework of Fleet2D is shown in Fig. 3. A single N-day trajectory is defined by the scenario (*inputs*), runs in the simulation engine (*processing*), and generates artifacts (*outputs*).

Scenario: represent the long-running scene to simulate. Its core components such as the floorplan, users, robots and their behaviors, are discussed in Section IV-B. A scenario is defined via the configuration file (Fig. 4) that provides a high level interface suitable for the HRI functions we study. For example, we often use `simulation.duration.days=30` and HouseExpo [20] floorplans by setting `floorplan.type="he"` and selecting one of the many floorplans. Finer options, such as robot speed and FOV parameters, can be controlled from deeper within its nested structure.

Simulation engine: is the workhorse of Fleet2D and executes a discrete-time simulation representing the scenario. It is fast, light and reproducible. The execution sequence within a single simulation step is described in detail in Fig. 5. The simulation parameters, such as the time granularity (default 1sec) and a seed for pseudorandomness, are set in the configuration file.

Artifacts: may be generated by each component of Fleet2D, although the metrics are tasked solely with that. These artifacts are organized and stored for downstream analysis.

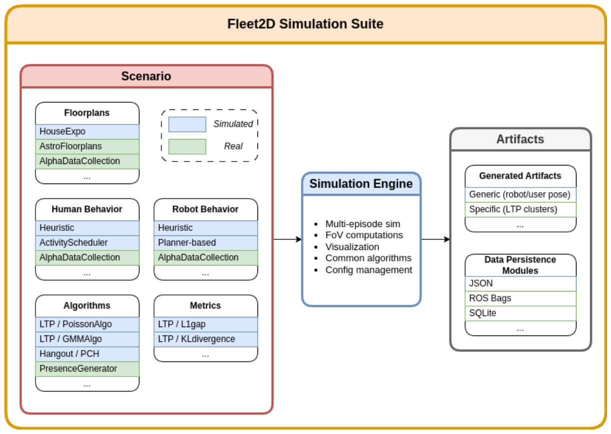
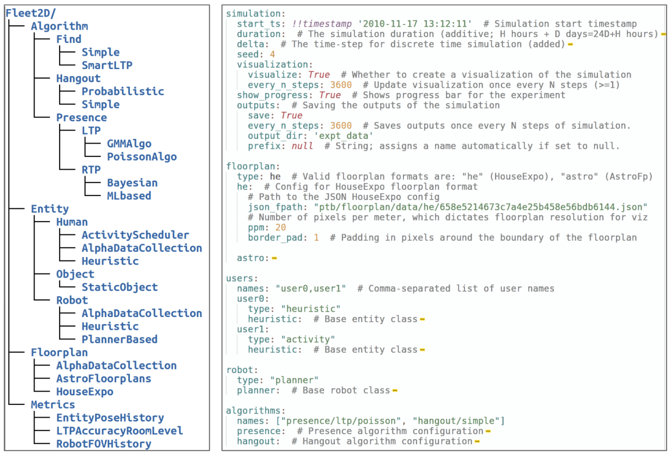


Fig. 3. *High-level framework of Fleet2D.* A scenario defines a long-running scene of interest using its different components. It is executed by the simulation engine, which runs a multi-entity long-term simulation and generates artifacts stored for downstream analysis.



```
Fleet2D/
├─ Algorithm
│  ├─ Find
│  │  ├─ Simple
│  │  └─ SmartLTP
│  ├─ Hangout
│  │  ├─ Probabilistic
│  │  └─ Simple
│  └─ Presence
│     ├─ LTP
│     │  ├─ GMMAlgo
│     │  └─ PoissonAlgo
│     └─ RTP
│        ├─ Bayesian
│        └─ MLbased
├─ Entity
│  ├─ Human
│  │  ├─ ActivityScheduler
│  │  ├─ AlphaDataCollection
│  │  └─ Heuristic
│  ├─ Object
│  │  └─ StaticObject
│  └─ Robot
│     ├─ AlphaDataCollection
│     ├─ Heuristic
│     └─ PlannerBased
├─ Floorplan
│  ├─ AlphaDataCollection
│  ├─ AstroFloorplans
│  └─ HouseExpo
└─ Metrics
   ├─ EntityPoseHistory
   ├─ LTPAccuracyRoomLevel
   └─ RobotFOVHistory
```

```
simulation:
  start_ts: !!timestamp '2010-11-17 13:12:11'  # Simulation start timestamp
  duration:  # The simulation duration (additive; H hours + D days=24D+H hours)⋯
  delta:  # The time-step for discrete time simulation (added)⋯
  seed: 4
  visualization:
    visualize: True  # Whether to create a visualization of the simulation
    every_n_steps: 3600  # Update visualization once every N steps (>=1)
  show_progress: True  # Shows progress bar for the experiment
  outputs:  # Saving the outputs of the simulation
    save: True
    every_n_steps: 3600  # Saves outputs once every N steps of simulation.
    output_dir: 'expt_data'
    prefix: null  # String; assigns a name automatically if set to null.

floorplan:
  type: he  # Valid floorplan formats are: "he" (HouseExpo), "astro" (AstroFp)
  he:  # Config for HouseExpo floorplan format
    # Path to the JSON HouseExpo config
    json_fpath: "ptb/floorplan/data/he/658e5214673c7a4e25b458e56bdb6144.json"
    # Number of pixels per meter, which dictates floorplan resolution for viz
    ppm: 20
    border_pad: 1  # Padding in pixels around the boundary of the floorplan

  astro:⋯

users:
  names: "user0,user1"  # Comma-separated list of user names
  user0:
    type: "heuristic"
    heuristic:  # Base entity class⋯
  user1:
    type: "activity"
    heuristic:  # Base entity class⋯

robot:
  type: "planner"
  planner:  # Base robot class⋯

algorithms:
  names: ["presence/ltp/poisson", "hangout/simple"]
  presence:  # Presence algorithm configuration⋯
  hangout:  # Hangout algorithm configuration⋯
```

Fig. 4. (*left*) The hierarchical code structure of the scenario components in Fleet2D. It is extensible, and new functionality continues to be integrated continually. (*right*) The configuration file offers a high-level interface to define this scenario (details folded to conserve space). It mirrors the code components for ease of use and development.

B. Core Components

Fleet2D has four core components: floorplan, entities (users, robots, objects), algorithms, and metrics. While the simulator functionality is continually expanding, this section emphasizes the significance of each component and identifies key aspects facilitating speed, reproducibility and modularity in Fleet2D.

Floorplan: defines the structural layout of a home and captures the semantics of rooms (e.g. kitchen) and objects (e.g. bed, door) to support HRI use-cases. Fleet2D offers both real and synthetic (e.g. HouseExpo [20]) floorplans that can be configured from the configuration file (Fig. 4). Typically, a floorplan is meant to be static, although our design allows for dynamic non-structural elements (e.g. updating room semantics). `Floorplan` enables entities and algorithms in Fleet2D to create customizable layers, which is a general mechanism to support several use-cases. For example, the robot may use an explored layer to incrementally map its environment and a costmap layer for path planning.

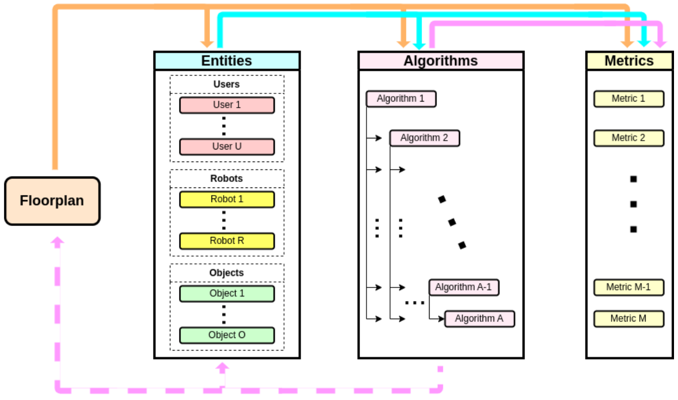


Fig. 5. *Execution within a single simulation step in Fleet2D.* As shown by the top arrows, each component updates sequentially from left to right and has access to the components updated prior to it (e.g. the entities may use the updated floorplan when updating). Upon completion, the algorithm updates feed back into the entities and the floorplan for consumption in the next step, as indicated by the bottom dashed arrows. These are crucial for capturing interactions between entities (e.g. HRI). Components within a bucket update independently (e.g. metrics), except for the algorithms (discussed in Section IV-B). All updates are executed in lock-step for reproducibility.

Internally, a floorplan is modeled using a grid of cells with configurable cell sizes (e.g. 5cm x 5cm), with additional layers for room semantics. It optimizes for speed by streamlining common operations. The most important amongst these is the FOV computation, where it precomputes a cache that is employed for downstream FOV operations by an entity (robot or human). This results in a speed gain of several orders of magnitude. Users can configure Fleet2D to use Visibility Polygons or the Flashlight algorithm to leverage this (Section IV-C discusses this further).

Entities: represent *users*, *robots*, and *objects* in the simulation environment. Relative to floorplan elements, they are more dynamic and often interact with each other. Each entity type has a set of attributes (e.g. location, orientation, moving) constituting its current *state* and comes with default *behaviors* that can be configured easily. These behaviors are responsible for both updating/expanding the state and for interactions across entities.

Fleet2D provides several default behaviors, primarily centered around presence-related HRI use-cases, as discussed in Section III-A.

- The robot can move around based on set schedules, it can proactively find users and hang out with them, or it can opportunistically interact with them when they are seen. We prototype these on-device behaviors in Fleet2D. We also provide the alternative of consuming actual data collected from a device operating in a home. Finally, as mentioned before, the robot's FOV parameters and errors in visual perception are matched to reality using data-driven heuristics (and available as configurable defaults).
- The users in Fleet2D can move across rooms based on either set or randomly generated schedules. To facilitate

a large variety of user behaviors, we allow randomly sampled activity schedules, room selection and pose selection within a room. The extensible user class also allows consuming data collected from a real-time location system (RTLS) and a human activity simulator [21].
- Objects are distinct from users and robots in that they do not natively possess an FOV and cannot interact or take actions. They might still possess sensors and expose additional state attributes (e.g. to mimic a smart home device).

Algorithms and Metrics: `Algorithm` is a function that consumes the entire simulation state (with its history) to produce an output feeds into the entities and the floorplan; this is shown in Fig. 5. This broad construct supports a variety of use-cases (e.g. to mimic ROS messages for testing robot software). However, it is typically used in narrower settings to enable key functionalities in Fleet2D. In the simplest case, an algorithm simply consumes a current state and generates summary statistics that are stored (e.g. accuracy); these are simply called `Metrics`. More complex algorithms influence an entity; they provide robot behaviors like Hangout (discussed in Section III-A) and user behaviors like activity schedules. Further, algorithms can influence other algorithms (e.g. LTP feeds into Hangout) and Fleet2D support such chaining (see Fig. 5). Algorithms enable even more complex behaviors, such as social activities (e.g., dinner) influencing multiple users.

We briefly note two important design aspects that greatly affect the utility of Fleet2D for downstream tasks, such as integration with other simulators/components and for scientific analyses. First, the storage of artifacts produced by the algorithms and the metrics is provided by the data persistence modules, as illustrated in Fig. 3. Second, the algorithms have access to the entire simulation state and thus provide a customizable mechanism to create bindings for downstream components.

C. Salient aspects

We mention some salient aspects of Fleet2D that make it a versatile tool in the development of a home robot:

Mixed Reality Simulation: Fleet2D's modular architecture permits individual components, such as the floorplan, to be either synthetically generated or sourced from real-world datasets. Consequently, real data from a specific component, like robot-obtained floorplans, can be seamlessly combined with simulated data from other elements. This amplifies the utility of even modest data collections. Fig. 6 depicts a robot-collected floorplan where a wide variety of simulated robot and user behaviors can be studied.

Methods for Calculating FoV: We offer two distinct methods for FOV computations. The primary distinction between these methods is their approach to speed optimization. (Fig. 6)

- **Visibility Polygon**: This method prioritizes precomputation of a persistent shared cache proportional to the floorplan's size (Fig. 2). Once formulated, it facilitates conducting multiple experiments on an identical floorplan while varying all other parameters. This is the fastest approach for a single floorplan.
- **Flashlight**: precomputes a cache that grows with the size of an entity's pose sequence as opposed to the size of the floorplan. This method is optimal when we wish to study performance *across* floorplans such that only a few experiments are run on any one floorplan.

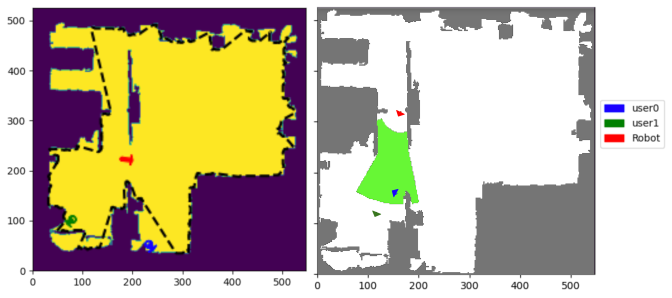


Fig. 6. A real-world captured floorplan built from a robot-guided exploration, with simulated robot and human in Fleet2D. FoV computations using (*left*) VisibilityPolygon in dashed line, and (*right*) Flashlight algorithm shaded bright green.

V. Results

A. Fleet2D Performance

Setup: We discuss the performance of Fleet2D for our use-cases from Section III. Our experiments ran 30-day simulations with a single robot and a single simulated user on a 2000 sq-ft HouseExpo floorplan, at a granularity of 1 second over time and 5cm x 5cm over space. The robot relied on two key algorithms: LTP to determine human presence, and Hangout which used LTP to proactively approach the user periodically. The user is modeled with varying degrees of fidelity - meandering using 2-D Brownian motion, room-level sampling given the LTP distribution, and randomly sampled activity schedules employing room semantics - that do not affect the run time significantly. The base configuration used Visibility Polygons (VP) for FOV computation with the (default) robot parameters, ran without visualization (headless mode) and saved artifacts every hour of simulation time. We tried variations of this base setup with a combination of: turning off robot algorithms, introducing multiple users, employing Flashlight, using real-world floorplans, and turning visualization on. These results are summarized next.

Results: The base experiment took 194.4 ± 4.6s, where loading a precomputed FOV cache took 45.1 ± 2.5s and the simulation 127.2 ± 5.3s (the rest being overhead). This corresponds to a speedup of $\sim 13,000$x over real time. Turning off the robot algorithms (and using a meandering robot motion) reduced the duration to 140.8s (28% faster, speedup of $\sim 18,500$x). Increasing the number of users in the simulation resulted in linear growth of the experiment duration, taking ~ 28 minutes for a 50-user experiment. More users could not be introduced due to memory limitations. Visualization significantly slowed down the simulator, resulting in a reduced $8 - 12$x speedup.

This is not particularly limiting since visualization is usually restricted to debugging scenarios.

VP versus Flashlight: Flashlight is ~ 8x slower than VP on HouseExpo floorplans and ~ 1.25x on real-world ones. The speedup from VP is most noticeable in smooth-walled environments like HouseExpo where visibility polygons are sparse. FlashLight is suited for real-world floorplans with jagged edges. Both algorithms scale linearly with the simulation duration (say, from 30 to 60 days). Flashlight scales better with the floorplan size (constant vs linear time). While VP requires computing a cache once per floorplan (~ 1hr for HouseExpo and ~ 6hrs for real-world), Flashlight does not. We recommend using Flashlight for real-world floorplans and for large-scale experiments where studying the floorplan variation is important. VP should be reserved for experimenting over user and robot behaviors with a few synthetically-generated floorplans.

B. Comparing Simulators

We consider the viability of employing existing simulators for our use-cases and compare their performance with Fleet2D. Recall that our primary motivation is to simulate long-term scenarios for long running robot functions and to scale these up for large-scale experimentation (e.g for model fine-tuning).

Setup: In a series of experiments we tested two well-established simulators, Gazebo and WeBots. While it is hard to replicate exactly the setup across all simulators, we note that our experimental setup gives an edge to other simulators in every comparison (e.g. by reducing complexity, employing GPU) and minimizes the use of fine-grained computations that are not required for our application. For Gazebo, we used ROS2 Humble and Gazebo 11, and placed a small two-wheeled robot equipped with a single LiDAR sensor (per [22]) in a small house world environment provided by AWS Robotics [23]. For WeBots, we placed a small two-wheeled robot with a LiDAR sensor (sourced from `rosbot` [24]) in a small apartment provided by WeBots for the iRobot Create demo. All experiments ran in headless mode and employed the NVIDIA GeForce RTX 2080 GPU that was required for optimal performance; no GPU was employed by Fleet2D.

TABLE II
SPEED COMPARISON ACROSS SIMULATORS

Simulator	Conditions			
	Sim Time	Speedup	Ease of Setup	GPU
Gazebo	8 hours	12.07×	★★☆	✓
WeBots	8 hours	15.9×	★★★	✓
Gazebo	32 hours	12.58×	★★☆	✓
WeBots	32 hours	15.1×	★★☆	✓
Fleet2D	30 days	13, 343×	★☆☆	○

[a] Experiments conducted on $500\,m^2$ floorplan with a single robot.
[b] We acknowledge that the ease of setup is subjective.

Results: Table II summarizes the results. Based on our findings above, for simulations up to 48 hours, the costs of setting up, configuring, and running 3D simulators such as Gazebo or WeBots are tractable. However, for scenarios spanning time horizons over 48 hours, Fleet2D is strongly preferred. From a prototyping standpoint, we recommend using Fleet2D, reserving 3-D simulators such as WeBots and Gazebo, for scenarios requiring higher fidelity (e.g. end-to-end testing).

Note that with both Gazebo and WeBots, the GPU was utilized extensively even in headless mode; this complicates containerization as the GPU becomes a contentious resource. For example, consider a set of experiments varying the robot behavior (e.g. exploration policy). With Fleet2D, these can be run in parallel (on CPU) on a single machine. In contrast, spawning these in separate Gazebo or WeBots instances causes GPU contention and leads to defaulting to the much slower CPU-only mode.

C. Versatility across Use-cases

Fleet2D was employed across a suite of use-cases described in Section III-A. We briefly summarize the results here.

- **Long-Term Presence**: Fleet2D was used to conduct large-scale experimentation to characterize our LTP algorithm. We tested across a diverse set of user and robot behaviors on a single floorplan. The main takeaway was that our LTP algorithm worked well and gradually approached the true long-term probabilities of the user. Fig. 7 shows the gap between the actual and the estimated probabilities over the rooms averaged over a set of 100 randomly sampled robot behaviors (for a fixed user behavior and floorplan). A similar trend was observed across different user behaviors.
- **ML-based Real-Term Presence**: We used Fleet2D for data generation and successfully used that to train ML-based RTP models (work under review). These models predict where inside the home a user is from their sparse observation history. They are trained on a large set of simulated user and robot motion data. Fig. 7 indicates the superiority of our model over the baseline in scenarios where the user was not seen for several minutes (large last seen gaps).

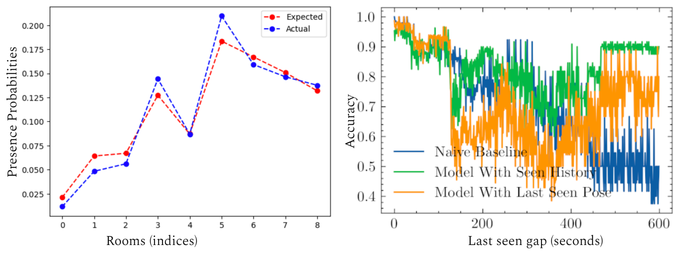


Fig. 7. *(Left)* Long-term Presence analysis using Fleet2D, showing the LTP distribution over rooms. *(Right)* The comparison of RTP accuracy with a naive baseline for different parameter value called "last seen gap" that denotes how long ago the user was observed.

VI. Conclusion

The rapidly evolving domain of home robotics presents unique challenges especially over long time horizons. With this new challenge, there is a new need for effective simulators that can facilitate the development and validation of long time horizon robot functions. In this work, we introduced Fleet2D, a novel simulator specifically designed for long-term human-robot interactions within home environments.

Fleet2D strategically sidesteps the limitations of existing simulators in this domain. By abstracting the low-level physics, caching compute-intensive tasks, and simplifying the operational world to 2-D, Fleet2D offers an environment that is both computationally efficient and adequate for the simulation of high-level functions. Our findings show that Fleet2D can operate at speeds exceeding $10,000$x times real time, a benchmark that underscores its potential in significantly accelerating the home robot development cycle. Our simulator's architecture is flexible and can be adapted to a wide array of robots and tasks, making it a valuable tool for the broader research community.

References

[1] Amazon, "Introducing amazon astro, household robot for home monitoring." [Online]. Available: https://https://amazon.com/astro

[2] P. Rane, V. Mhatre, and L. Kurup, "Study of a home robot: Jibo," *International journal of engineering research and technology*, vol. 3, no. 10, pp. 490–493, 2014.

[3] T. Vintr, Z. Yan, T. Duckett, and T. Krajník, "Spatio-temporal representation for long-term anticipation of human presence in service robotics," 2019.

[4] J. J. Lee, A. Atrash, D. F. Glas, and H. Fu, "Developing autonomous behaviors for a consumer robot to be near people in the home," 2023.

[5] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve *et al.*, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *PNAS*, 2021.

[6] C. Breazeal, *Designing sociable robots*. MIT press, 2004.

[7] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS 2004*. IEEE, 2004.

[8] O. Michel, "Cyberbotics webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, 2004.

[9] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[10] I. Tursynbek and A. Shintemirov, "Modeling and simulation of spherical parallel manipulators in coppeliasim (v-rep) robot simulator," in *NIR 2020*. IEEE, 2020.

[11] J. P. R. Belo and R. A. Romero, "A social human-robot interaction simulator for reinforcement learning systems," in *ICAR 2021*. IEEE.

[12] A. Farley, J. Wang, and J. A. Marshall, "How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion," *Simulation Modelling Practice and Theory*, 2022.

[13] B. Gerkey, R. T. Vaughan, A. Howard *et al.*, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *ICRA 2003*.

[14] K. Johns and T. Taylor, *Professional microsoft robotics developer studio*. John Wiley & Sons, 2009.

[15] L. Wen, D. Fu, S. Mao, P. Cai, M. Dou, Y. Li, and Y. Qiao, "Limsim: A long-term interactive multi-scenario traffic simulator," 2023.

[16] F. Jovan, M. Tomy, N. Hawes, and J. Wyatt, "Efficiently exploring for human robot interaction: partially observable poisson processes," *Autonomous Robots*, 2023.

[17] E. A. Sisbot, L. F. Marin, and R. Alami, "Spatial reasoning for human robot interaction," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 2281–2287.

[18] A. Bihlmaier and H. Wörn, "Robot unit testing," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 255–266.

[19] F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller, "Efficient computation of visibility polygons," *arXiv preprint arXiv:1403.3905*, 2014.

[20] L. Tingguang, H. Danny, L. Chenming, Z. Delong, W. Chaoqun, and M. Q.-H. Meng, "Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots," *arXiv preprint arXiv:1903.09845*, 2019.

[21] I. Idrees, S. Singh, K. Xu, and D. F. Glas, "A framework for realistic simulation of daily human activity," 2023.

[22] A. Sears-Collins. How to load a robot model (sdf format) into gazebo – ros 2. [Online]. Available: https://automaticaddison.com/page/12/

[23] A. Robotics. Aws robomaker small house world ros package. [Online]. Available: https://github.com/aws-robotics/aws-robomaker-small-house-world

[24] Cyberbotics. Webots. [Online]. Available: https://github.com/cyberbotics/webots/