2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

CyPhyHouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination

Ritwika Ghosh[1], Joao P. Jansch-Porto[2], Chiao Hsieh[1], Amelia Gosse[2],
Minghao Jiang[3], Hebron Taylor[2], Peter Du[3], Sayan Mitra[3], Geir Dullerud[2]

***Abstract*— Programming languages, libraries, and development tools have transformed the application development processes for mobile computing and machine learning. This paper introduces *CyPhyHouse*—a toolchain that aims to provide similar programming, debugging, and deployment benefits for distributed mobile robotic applications. Users can develop hardware-agnostic, distributed applications using the high-level, event driven *Koord* programming language, without requiring expertise in controller design or distributed network protocols. The modular, platform-independent *middleware* of *CyPhyHouse* implements these functionalities using standard algorithms for path planning (RRT), control (MPC), mutual exclusion, etc. A high-fidelity, scalable, multi-threaded simulator for *Koord* applications is developed to simulate the same application code for dozens of heterogeneous agents. The same compiled code can also be deployed on heterogeneous mobile platforms. The effectiveness of *CyPhyHouse* in improving the design cycles is explicitly illustrated in a robotic testbed through development, simulation, and deployment of a distributed task allocation application on in-house ground and aerial vehicles.**

I. Introduction

Programming languages like C#, Swift, Python, and development tools like LLVM [1] have helped make millions of people with diverse backgrounds, into mobile application developers. Open source software libraries like Caffe [2], PyTorch [3] and Tensorflow [4] have propelled the surge in machine learning research and development. To a lesser degree, similar efforts are afoot in democratizing robotics. Most prominently, ROS [5] provides hardware abstractions, device drivers, messaging protocols, many common library functions and has become prevalent. Libraries such as PyRobot [6] and PythonRobotics [7] provide hardware-independent implementations of common functions for physical manipulation and navigation of individual robots.

Nevertheless, it requires significant effort and time (of the order of weeks) to develop, simulate, and debug a new application for a single mobile robot—not including the effort to build the robot hardware. The required effort grows quickly for distributed and heterogeneous systems, as none of the existing robotics libraries provide either (a) support for distributed coordination, or (b) easy portability of code across different platforms.

This work is supported by an infrastructure research grant from the National Science Foundation (CNS 1629949). In addition Hsieh and Mitra are also supported in part by the NSF grant CNS 1544901. We acknowledge the support of John M. Hart of the Coordinated Science Laboratory of University of Illinois at Urbana-Champaign for facilitating experiments.

1 Department of Computer Science, UIUC.
2 Department of Mechanical Science and Engineering, UIUC.
3 Department of Electrical and Computer Engineering, UIUC.

With the aim of simplifying application development for distributed and heterogeneous systems, we introduce *CyPhyHouse*[1]—an open source software toolchain for programming, simulating, and deploying mobile robotic applications.

In this work, we target distributed coordination applications such as collaborative mapping [8], surveillance, delivery, formation-flight, etc. with aerial drones and ground vehicles. We believe that for these applications, low-level motion control for the individual robots is standard but tedious, and coordination across distributed (and possibly heterogeneous) robots is particularly difficult and error-prone. This motivates the two key abstractions provided by *CyPhyHouse*: (a) *portability* of high-level coordination code across different platforms; and (b) *shared variable* communication across robots.

The first of the several software components of *CyPhyHouse* is a high-level programming language called *Koord* that enables users to write distributed coordination applications without being encumbered by socket programming, ROS message handling, and thread management. Our *Koord* compiler generates code that can be and has been directly deployed on aerial and ground vehicle platforms as well as simulated with the *CyPhyHouse simulator. Koord* language abstractions for path planning, localization, and shared memory make application programs succinct, portable, and readable (see Section III). We have built the *CyPhyHouse middleware* with a modular structure to make it easy for roboticists to add support for new hardware. In summary, the three main contributions of this paper are as follows.

1) *An end-to-end distributed application for robotic vehicles and drones developed and deployed using CyPhyHouse toolchain* This *Task* application requires that the participating robots mutually exclusively visit a common list of points, while avoiding collisions. Our program written in *Koord* is shorter than 50 lines (see Figure 2). Our compiler generates executables for both the drone and the vehicle platforms, linking the platform independent parts of the application with the platform-specific path planners and controller. We ran more than 100 experiments with a set of tasks running on different combinations of ground and aerial vehicles, all with few edits in the *configuration file* (see Figure 2).

2) *A high-fidelity, scalable, and flexible simulator for distributed heterogeneous systems* The simulator executes instances of the application code generated by the *Koord* compiler—one for each robot in the scenario. Within the

[1] https://cyphyhouse.github.io

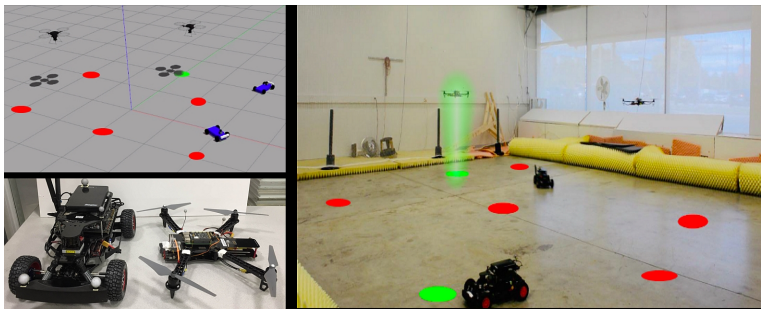


Fig. 1. *Right:* Annotated snapshot of a distributed task allocation application deployed on four cars and drones using *CyPhyHouse* in our test arena. The red tasks are incomplete, and the green are completed. *Left bottom:* different robotic platforms: F1/10 Car and quadcopter. *Left top:* Visualization of the same application running in *CyPhyHouse* simulator which interfaces with *Gazebo*.

simulator, individual robots communicate with each other over a wired or a wireless network and with their own simulated sensors and actuators through ROS topics. For example, a simulation with 16 drones can spawn over 1.4K ROS topics and 1.6K threads, yet, our simulator is engineered to execute and visualize such scenarios in Gazebo running on standard workstations and laptops. In Section V, we present detailed performance analysis of the simulator.

3) *A programming language and middleware for heterogeneous platforms supporting application development, simulation, deployment, as well as verification. Koord* comes with well-defined semantics which makes it possible to reason about the correctness of the distributed applications using formal techniques. [2] *Koord* provides abstractions for distributed applications running on possibly heterogeneous platforms. For example, *Koord* enables easy coordination across robots: a single line of code like

$$x[pid] = (x[pid-1] + x[pid+1])/2$$

assigns to a shared variable $x[pid]$ of a robot with the unique integer identifier pid, the average of the values of $x[pid-1]$ and $x[pid+1]$ which are the values held respectively by robots $pid-1$ and $pid+1$. This makes *Koord* implementations of consensus-like protocols read almost like their textbook counterparts [10]. These statements are implemented using message-passing in the middleware.

II. Related work

Several frameworks and tools address the challenges in development of distributed robotic applications. Table I summarizes a comparision of these works along the following dimensions: (a) whether the framework has been tested with **h**ard**w**are **depl**oyments, (b) availability of support for networked and **dist**ributed robotic **sys**tems, (c) support for **heterogeneous** platforms, (d) availability of specialized **prog**ramming **lang**uage, (e) availability of a **sim**ulator and **comp**iler, and (f) support for formal **v**erification and **v**alidation.

Drona is a framework for multi-robot motion planning and to our knowledge, has so far been deployed only on drones. CyPhyHouse aims to be more general, and multiple applications have been deployed on cars and drones in both simulations and hardware.

Buzz, the programming language used by ROSBuzz [11] doesn't provide abstractions like CyPhyHouse does with *Koord*, for path planning, de-conflicting, and shared variables. Additionally, ROSBuzz specifically requires the Buzz Virtual Machine to be deployed on each robot platform whereas with *CyPhyHouse*, deploying *Koord* only requires standard ROS and Python packages.

It should also be mentioned, that "Correct-by-construction" synthesis from high-level temporal logic specifications have been widely discussed in the context of mobile robotics (see, for example [12], [13], [14], [15], [16]). *CyPhyHouse* differs in the basic assumption that roboticist's (programmer's) creativity and efforts will be necessary well beyond writing high-level specs in solving distributed robotics problems; consequently only the tedious parts of coordination and control are automated and abstracted in the *Koord* language and compiler.

TABLE I

Name	HW Depl.	Dist. Sys.	Hetero- geneous	Sim	Prog. Lang.	Comp.	V&V
ROS [5]	✓		✓	✓	C++/Python/...		
ROSBuzz [11]	✓	✓	✓	✓	Buzz	✓	
PythonRobotics			✓	✓	Python		
PyRobot [6]	✓		✓	✓	Python		
MRPT [17]	✓		✓		C++		
Robotarium [18]	✓	✓	✓	✓	Matlab/Python		
Drona [19]	✓	✓		✓	P [20]	✓	✓
Live [21]	✓		✓		LPR	✓	
CyPhyHouse	✓	✓	✓	✓	*Koord*	✓	✓

Other open and portable languages that raise the level of abstraction for robotic systems include [22], [23], [24]. [3] VeriPhy [22] also has some commonality with *CyPhyHouse*; however, instead of a programming language, the starting point is differential dynamic logic [26].

III. A distributed task allocation application

In this section, we introduce the *distributed task allocation problem (Task)* that we will use throughout the paper to illustrate the capabilities of *CyPhyHouse*.

Given a robot G, and a point x in R^3, we say that G has visited x if the position of G stays within an ϵ_v-ball x for

[2] Formal semantics of the language and the automatic verification tools are not part of this paper. Some of the details of the formal aspects of *Koord* were presented in an earlier workshop paper [9].

[3] For an earlier survey see [25]. Most of these older languages are proprietary and platform-specific.

δ_v amount of time, for some fixed $\epsilon_v > 0$ and $\delta_v > 0$. The distributed task allocation problem requires a set of robots to visit a sequence of points mutually exclusively:

> *Task: Given a set of (possibly heterogeneous) robots, a safety distance $d_s > 0$, and a sequence of points (tasks) list $= x_1, x_2, \ldots \in \mathbb{R}^3$, it is required that: (a) every unvisited x_i in the sequence is* visited *exactly by one robot; and (b) no two robots ever get closer than d_s.*

We view visiting points as an abstraction for location-based objectives like package delivery, mapping, surveillance, or fire-fighting.

The flowchart in Figure 2 shows a simple idea for solving this problem for a single robot: Robot A looks for an unassigned task τ from *list*; if there is a clear path to τ then A assigns itself the task τ. Then A visits τ following the path; once done it repeats. Of course, converting this to a working solution for a distributed system is challenging as it involves combining distributed mutual exclusion ([27], [28]) for assigning a task τ exclusively to a robot A from the *list* (step 1), dynamic conflict-free path planning (step 2), and low-level motion control (step 3).

Our *Koord* language implementation of this flowchart is shown in Figure 2. It has two *events*: *Assign* and *Complete*. The semantics of *Koord* is such that execution of the application programs in the distributed system advances in *rounds* of duration δ[4], and in each round, each robot executes at most one event. A robot can only execute the statements in the event's effect (**eff**) if its precondition (**pre**) is satisfied. If no event is enabled, the robot does nothing. In between the rounds, the robots may continue to move as driven by their local controllers. The *CyPhyHouse* middleware (Section IV) ensures that the robot program executions adhere to this schedule even if local clocks are not precisely synchronized.

In our example, the *Assign* event uses a single *atomic* update to assign a task to robot i from the *shared list* of tasks called *list* in a mutually exclusive fashion.[5] The *route* variable shares paths and positions among all robots, and is used by each robot in computing a collision-free path to an unassigned task.[6] To access variables, e.g., *route*, shared by a certain robot, each robot program also has access to its unique integer identifier *pid* and knows the *pid*s of all participating robots.

The low-level control of the robot platform is abstracted from the programmers in *Koord*, with certain assurances about the controllers from the platform developers (discussed in Section IV and Section VI). The *Task* program uses a controller called *Motion* to drive the robots through a route, as directed by the position value set at its actuator port *Motion.route*. The sensor ports used by the *Task* program are: (a) *Motion.psn*: the robot position in a fixed coordinate system. (b) *Motion.reached*: a flag indicating whether the robot has reached its waypoint.

Here the motion module implements vehicle models for the robots. In the next section, we discuss the *CyPhyHouse* middleware, which implements a modular design of this runtime system to allow a high degree of flexibility concerning these modules in deployment and simulation.

IV. CyPhyHouse Architecture

A system running a *Koord* application has three parts: an application program, a controller, and a plant. At runtime, the *Koord* program executes within the runtime system of a single agent, or a collection of programs execute on different agents that communicate using shared variables. The plant consists of the hardware platforms of the participating agents. The controller receives inputs from the program (through actuator ports), sends outputs back to the program (through sensor ports), and interfaces with the plant. We developed a software-hardware interface (*middleware*) in Python 3.5 to support the three-plane architecture comprising the *Koord* runtime system.

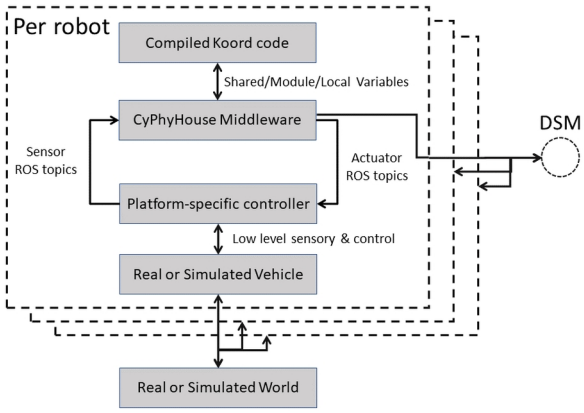


Fig. 3. Each compiled *Koord* program interacts with *CyPhyHouse* middleware simply via variables. The middleware implements distributed shared memory(DSM) across agents and the language abstractions over platform-specific controllers through actuator ROS topics, and obtain (real or simulated) information such as device positions through sensor ROS topics.

A. Compilation

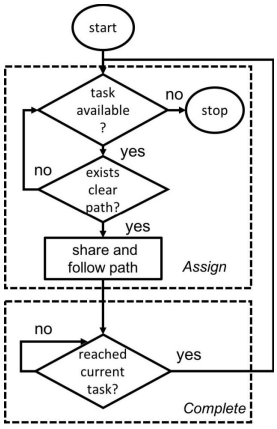
The *Koord* compiler included with *CyPhyHouse* generates Python code for the application using all the supported libraries, such as the implementation of distributed shared variables using message passing over WiFi, motion automata of the robots, high-level collision and obstacle avoidance strategies, etc. The application then runs with the Python *middleware* for *CyPhyHouse*. The *Koord* compiler is written using Antlr (Antlr 4.7.2) in Java [29].[7] We use ROS to handle the low-level interfaces with hardware. To communicate between the high-level programs and low-level controllers, we use Rospy, a Python client library for ROS which enables the (Python) middleware to interface with ROS Topics and Services used for deployment or simulation.

[4] δ is a parameter set by the user, with a default value of 0.1 second.

[5] We provide several library functions associated with abstract data types (assign, allAssigned) and path planners (findPath, pathIsClear). Users can also write functions permitted by *Koord* syntax.

[6] Platform specific path-planners can ensure that ground vehicles do not find paths to points above the plane, and aerial vehicles do not find paths to points on the ground.

[7] Details of the grammar, AST, and IR design of the *Koord* compiler are beyond the scope of this paper; however, description of the language and its full grammar are provided in [30] for the interested reader.




```
using Motion:
 actuators:
  pos[] path
 sensors:
  pos psn
  boolean reached
local:
 boolean isOnTask
 Task myTask
 pos[] tstRoute
 int i

allread:
 pos[] route
allwrite:
 Task[] tasks

Assign:
 pre: isOnTask == false
 eff: if allAsgned(tasks):
        stop
 else: atomic:
  for i = 0, length(tasks):
   if task.assigned==false:
    myTask=tasks[i]
    tstRoute=\
      findPath(Motion.psn, myTask.loc)
    if pathIsClear(route, tstRoute, pid):
      isOnTask=true
      assign(myTask, pid)
      tasks[i]=myTask
      route[pid]=tstRoute
      Motion.path=tstRoute
    else:
      isOnTask=false
      route[pid]=[Motion.psn]

Complete:
 pre: isOnTask and Motion.reached
 eff: isOnTask=false
    route[pid]=[Motion.psn]
```

```
robot:
    pid: 0
    on_device: hotdec_car
    motion_automaton: MoatTestCar
    ...
device:
    bot_name: hotdec_car
    bot_type: CAR
    planner: RRT_CAR
    positioning_topic:
        topic: vrpn_client_node/
        type: PoseStamped
    reached_topic:
        topic: reached
        type: String
    waypoint_topic:
        topic: waypoint
        type: String
    default_leader_pid = 0
    ...
num_robots: 3
```

Fig. 2. *Left* shows the flowchart for a simple solution to *Task* application. *Middle* shows the *Task* program implemented in *Koord* language for robots with identifier *pid* to solve distributed task allocation problem. *Right* shows snippet of a sample configuration. It includes platform-agnostic settings for the robot, e.g., robot id (*pid*), device to run on (*on_device*), and the number of robots (*num_robots*), as well as platform specific settings, e.g., path planners (*planner*) and position systems (*positioning_topic*).

B. Shared memory and Communication

At a high level, updates to a shared variable by one agent are propagated by the *CyPhyHouse* middleware, and become visible to other agents in the next round. The correctness of a program relies on agents having consistent values of shared variables. When an agent updates a shared variable, the middleware uses message passing to inform the other agents of the change. These changes should occur before the next round of computations.

CyPhyHouse implements the shared memory between robots through UDP messaging over Wi-Fi. Any shared memory update translates to an update message which the agent broadcasts over WiFi.[8] The agents running a single distributed *Koord* application are assumed to be running on a single network node, with little to no packet loss. However, the communication component of the middleware can be easily extended to support multi-hop networks as well.

C. Dynamics

If an application requires the agents to move, each agent uses an abstract class, *Motion automaton*, which must be implemented for each hardware model (either in deployment or simulation). This automaton subscribes to the required ROS Topics for positioning information of an agent, updates the *reached* flag of the motion module, and publishes to ROS topics for motion-related commands, such as waypoint or path following. It also provides the user the ability to use different path planning modules as long as they support the interface functions. Figure 4 shows two agents executing the *same application* using different path planners.

D. Portability

Apart from the dynamics, all aforementioned components of the *CyPhyHouse* middleware are platform-agnostic. Our implementation allows any agent or system simulating or deploying a *Koord* program to use a configuration file (as shown in Figure 2) to specify the system configuration, and the runtime modules for each agent, including the dynamics-related modules, while using the same application code.

[8]The interested reader is referred to [30] for more details on the shared memory model, and its formal semantics.

V. CyPhyHouse multi-robot simulator

We have built a high-fidelity simulator for testing distributed *Koord* applications with large number of heterogeneous robots in different scenarios. Our middleware design allows us to separate the simulation of *Koord* applications and communications from the physical models for different platforms. Consequently, the compiled *Koord* applications together with the communication modules can run directly in the simulator—one instance for each participating robot, and only the physical dynamics and the robot sensors are replaced by their simulated counterparts. This flexibility enables users to test their *Koord* applications under different scenarios and with various robot hardware platforms. Simpler physical models can be used for early debugging; and the same code can be used later with more accurate models. The simulator can be used to test different scenarios, with different numbers of (possibly heterogeneous) robots, with no modifications to the application code itself, rather simply modifying a configuration file as shown in Figure 5. To our knowledge, this is the only simulator for distributed robotics providing such fidelity and flexibility.

A. Simulator Design

Simulating Koord and communication: To faithfully simulate the communication, our simulator spawns a process for each robot which encompasses all middleware threads. The communication handling threads in these processes can then send messages to each other through broadcasts within the local network. To simulate robots on a single machine, we support specifying distinct network ports for robots in the configuration file. Since the communication is through actual network interfaces, our work can be extended to simulate different network conditions with existing tools in the future.

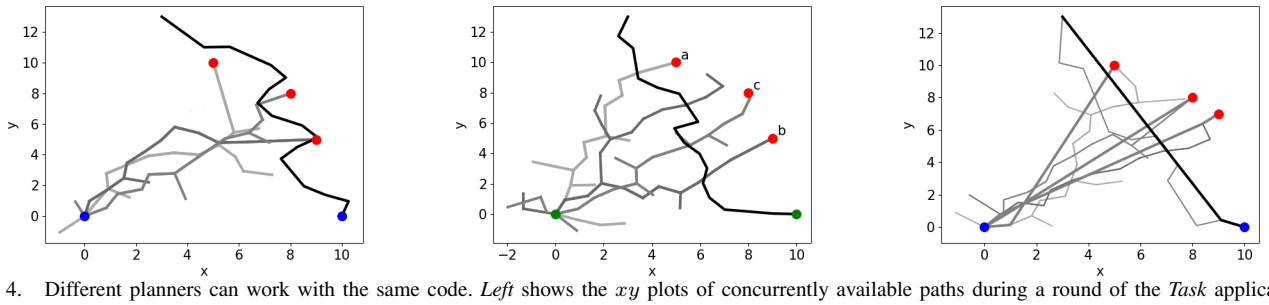


Fig. 4. Different planners can work with the same code. *Left* shows the xy plots of concurrently available paths during a round of the *Task* application using an RRT planner for two quadcopters. *Middle* shows the same configuration, where paths computed are not viable to be traversed concurrently. The green markers are current quadcopter positions, The black path is a fixed path, and the red points are unassigned task locations. *Right* shows the same scenarios under which paths cannot be traversed concurrently, except that a different RRT-based planner (with path smoothing) is used.

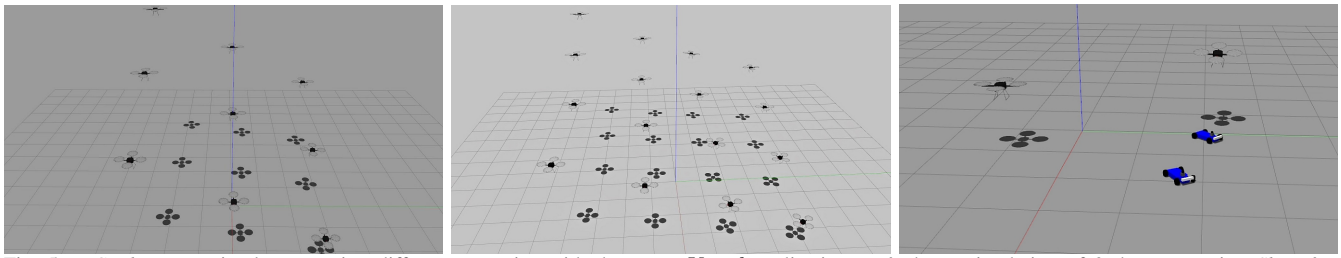


Fig. 5. *CyPhyHouse* simulator running different scenarios with the same *Koord* application. *Left* shows simulation of 9 drones running *Shapeform* application, *Middle* shows the *Shapeform* application on 16 drones. Different scenarios are specified by changing the configuration file.*Right* shows a simulation of *Task* on heteterogenous robots.

Physical Models and Simulated World: Our simulated physical world is developed based on Gazebo [31] and we provide a simulated positioning system to relay positions of simulated devices from Gazebo to the *CyPhyHouse* middleware. We integrate two Gazebo robot models from the Gazebo and ROS community, the car from the MIT RACECAR project [32] and the quadcopter from the hector quadrotor project [33]. Further, we implement a simplified version of position controller by modifying the provided default model. Users can choose between simplified models for faster simulation or original models for accuracy.

We also develop Gazebo plugins for trace visualization. These can be used to plot robot movement for real-time monitoring during experiments, or for post-experiment visualization and analysis with Gazebo.

B. Simulator performance analysis experiments and results.

Large scale simulations play an important role in testing robotic applications and also in training machine learning modules for perception and control. Therefore, we perform a large set of experiments to measure the performance and scalability of the *CyPhyHouse* simulator and experiment with various scenarios (such as different application *Koord* programs, increasing numbers of devices, or mixed device types). We then collect the usages of different resources and the amount of messages in each scenario. Finally, we compare resource usages and communications to study how our simulator can scale across different scenarios.

In our experiments, we use three *Koord* programs including the example *Task* in Figure 2, a line formation program *Lineform*, and a program forming a square *Shapeform*. For *Task*, we simulate with both cars and quadcopters to showcase the coordination between heterogeneous devices. For *Lineform* and *Shapeform*, we use only quadcopters to evaluate the impact of increasing numbers of robots on these statistics. For each experiment scenario with a timeout of 120 seconds, we collect the total message packets and packet length received by all robots and sample the following resource usages periodically: Real Time Factor (RT Factor, the ratio between simulated clock vs wall clock), CPU percentage, the memory percentage and the number of threads. All experiments are run on a workstation with 32 Intel Xeon Silver 4110 2.10GHz CPU cores and 32 GB main memory.

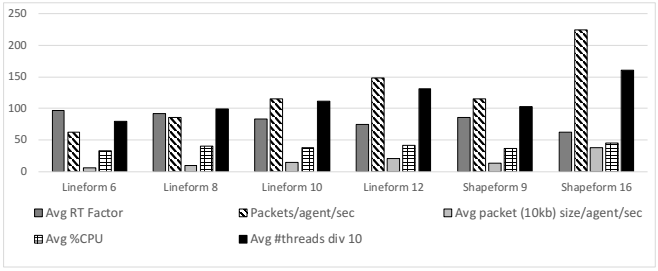


Fig. 6. Resource usages and communications for *Shapeform* and *Lineform*.

In Figure 6, we show the average of each collected metric. For *Lineform* and *Shapeform*, RT factor drops while all resource usages scale linearly with the number of robots. Average number and size of packets received per second for each robot grows linearly; hence, the message communication complexity for all robots is quadratic in the number of robots. One can improve the communication complexity with a more advanced distributed shared memory design.

VI. Deployment Setup

A. Vehicles

The *CyPhyHouse* toolchain was developed with heterogeneous robotics platforms in mind. In order to demonstrate

such capabilities, we have built both a car and a quadcopter.

Quadcopter: The quadcopter was assembled from off-the-shelf hardware, with a 40cm $\times$ 40cm footprint. The main computing unit consists of a Raspberry Pi 3 B+ along with a Navio2 deck for sensing and motor control. Stabilization and reference tracking are handled by Ardupilot [34]. Between the *CyPhyHouse* middleware and Ardupilot we include a hardware abstraction layer to convert setpoint messages from the high-level language into MAVLINK using the mavROS library ([35]), so Ardupilot can parse them. Since the autopilot was originally meant to use a GPS module, we also convert the current quadcopter position into the Geographic Coordinate System before sending it to the controller.

Car: Similarly, the car platform uses off-the-shelf hardware based on the open-source MIT RACECAR project [32]. The computing unit consists of an NVIDIA TX2 board. In the car platform, instead of using Ardupilot to handle the waypoint following, we wrote a custom ROS node uses the current position and desired waypoints to compute the input speed and steering angle using a Model Predictive Controller (MPC). The car has an electronic speed controller that handles low-level hardware control.

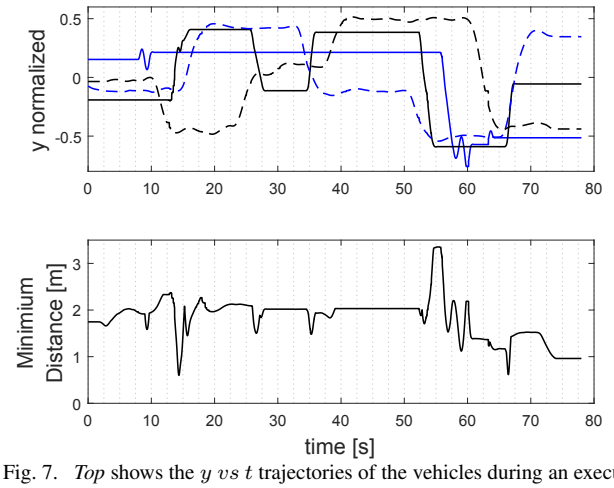


Fig. 7. *Top* shows the y *vs* t trajectories of the vehicles during an execution of the task, and *bottom* shows the minimum distance between all robots. The vehicle positions in the *top* graph were normalized to improve visualization. We can see concurrent movement when it is safe (for example at 13, 36, and 52 seconds), and only one robot moving or no robot moving when trying to compute a safe and collision-free path to a task.

B. Test arena and localization

We performed our experiments in a 7m $\times$ 8m $\times$ 3m arena equipped with 8 Vicon cameras. The Vicon system allows us to track the position of multiple robots with sub-millimeter accuracy, however, we note that the position data can come from any source (for example GPS, ultrawide-band, LIDAR), as long as all robots share the same coordinate system. While the motion capture system transmits all the data from a central computer, each vehicle only subscribes to its own position information. This was done to simplify experiments, as the goal of the paper is not to present new positioning systems. All coordination and de-conflicting across agents is performed based on position information shared explicitly through shared variables in the *Koord* application.

C. Interface with middleware

As mentioned in Section IV, the same application can be deployed using different path planners, which are associated with the platform-specific *motion automaton* through interfaces defined by the *CyPhyHouse* middleware. Both vehicles use RRT-based path planners [36] to compute a path to the next task. The car planner uses a bicycle model to compute the feasible paths, while the quadcopter planner assumes it can move in a straight line between points. The path generated is then forwarded to the robot via a ROS topic. The ROS topics required for positioning and setting waypoints of the vehicles were specified in the configuration. Each vehicle updates the *reached* topic when they reach a predefined ball around the destination. The car has nonholonomic constraints, while the quadcopter has uncertain dynamics, so in other standard settings, a roboticist would have to develop a separate application for each platform.

D. Experiments with Task on upto four vehicles

The *Task* application of Section III was run in over 100 experiments with different combinations of cars and quadcopters. Figure 7 shows the (x, y)-trajectories of the vehicles in one specific trial run, in which two quadcopters and two cars were deployed. Careful examination of the figure shows that all the performance requirements of *Task* are achieved, with concurrent movement when different robots have clear paths to tasks, safe separation at all times, and agents getting blocked when there is no safe path found. In our experiments with up to 4 vehicles, we found that with fewer agents, there are fewer blocked paths, so each robot spends less time idling, but this non-blocking effect is superseded by the parallelism gains obtained from having multiple robots. For example, three agents (2 quadcopters and 1 car, or 1 quadcopter and 2 cars) show an average runtime of about 110 seconds for 20 tasks. The average runtime for the same with 4 agents across 70 runs was about 90 seconds. We experience zero failures, provided the wireless network conditions satisfy the assumptions stated in Section IV.

VII. Conclusions and Future Work

In this paper we presented the *CyPhyHouse* software development and deployment toolchain for distributed robotic applications. It interfaces with and complements existing tools commonly used by roboticists, such as ROS, providing easy integration with popular platforms by almost any user. Our experience suggests that *Koord* and the *Koord* compiler cann enable users to develop and run distributed robotics applications in a hardware independent fashion; *Koord* programs can be ported across platforms automatically with minimal effort from the application developer; and our high fidelity simulation can provide a valuable testing and debugging environment. In the future, we plan to perform a user study to gain a better understanding of the robot app developer's experience and efficiency with *CyPhyHouse*.

REFERENCES

[1] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, ser. CGO '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 75–. [Online]. Available: http://dl.acm.org/citation.cfm?id=977395.977673

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[3] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[5] M. Quigley, K. Conley, B. P. 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*, 2009.

[6] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.

[7] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Pythonrobotics: a python code collection of robotics algorithms," *CoRR*, vol. abs/1808.10703, 2018.

[8] A. Cunningham, M. Paluri, and F. Dellaert, "Ddf-sam: Fully distributed slam using constrained factor graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 3025–3030.

[9] R. Ghosh, S. Misailovic, and S. Mitra, "Language semantics driven design and formal analysis for distributed cyber-physical systems: [extended abstract]," in *ApPLIED@PODC*, 2018.

[10] M. Mesbahi and M. Egerstedt, *Graph-theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[11] D. St-Onge, V. S. Varadharajan, G. Li, I. Svogor, and G. Beltrame, "ROS and buzz: consensus-based behaviors for heterogeneous teams," *CoRR*, vol. abs/1710.08843, 2017. [Online]. Available: http://arxiv.org/abs/1710.08843

[12] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[13] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, 2010, pp. 101–110.

[15] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 2011, pp. 313–314.

[16] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[17] J. L. B. Claraco and I. F. Steps, "Development of scientific applications with the mobile robot programming toolkit the mrpt reference book."

[18] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1699–1706.

[19] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia, "Drona: A framework for safe distributed mobile robotics," in *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2017, pp. 239–248.

[20] A. Desai, V. Gupta, E. Jackson, S. Qadeer, S. Rajamani, and D. Zufferey, "P: Safe asynchronous event-driven programming," *SIGPLAN Not.*, vol. 48, no. 6, pp. 321–332, June 2013. [Online]. Available: http://doi.acm.org/10.1145/2499370.2462184

[21] M. Campusano and J. Fabry, "Live robot programming: The language, its implementation, and robot API independence," *Science of Computer Programming*, vol. 133, pp. 1 – 19, 2017.

[22] B. Bohrer, Y. K. Tan, S. Mitsch, M. O. Myreen, and A. Platzer, "Veriphy: Verified controller executables from verified cyber-physical system models," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: ACM, 2018, pp. 617–630. [Online]. Available: http://doi.acm.org/10.1145/3192366.3192406

[23] D. Zufferey, "The REACT language for robotics," 2017.

[24] B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliott, "Model-based programming of intelligent embedded systems and robotic space explorers," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 212–237, 2003.

[25] A. Nordmann, N. Hochgeschwender, and S. Wrede, *A Survey on Domain-Specific Languages in Robotics*. Cham: Springer International Publishing, 2014, pp. 195–206. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11900-7_17

[26] B. Bohrer, V. Rahli, I. Vukotic, M. Völp, and A. Platzer, "Formally verified differential dynamic logic," in *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, ser. CPP 2017. New York, NY, USA: ACM, 2017, pp. 208–221. [Online]. Available: http://doi.acm.org/10.1145/3018610.3018616

[27] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996.

[28] S. Ghosh, *Distributed systems: an algorithmic approach*. Chapman and Hall/CRC, 2014.

[29] T. Parr, *The Definitive ANTLR 4 Reference*, 2nd ed. Pragmatic Bookshelf, 2013.

[30] Koord Language Development Team, "A design and analysis framework for distributed cps." [Online]. Available: https://cyphyhouse.github.io/papers/koord.pdf

[31] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2149–2154 vol.3.

[32] S. Karaman, A. Anders, M. Boulet, J. Connor, K. Gregson, W. Guerra, O. Guldner, M. Mohamoud, B. Plancher, R. Shin, and J. Vivilecchia, "Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit," in *2017 IEEE Integrated STEM Education Conference (ISEC)*, March 2017, pp. 195–203.

[33] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots*, I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 400–411.

[34] ArduPilot Development Team, "Ardupilot."

[35] Vladimir Ermakov, "mavros."

[36] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.