Full Stack Swarm Architecture

Kyle Morris, Gabriel Arpino, Sasanka Nagavalli, *Student Member, IEEE*,
Katia Sycara, *Fellow, IEEE*

Abstract— **Robot swarms are homogeneous multi-robot systems that form collective behaviour from decentralized local interactions. Swarms are a favorable choice for solving various problems in robotics as they are robust and fault tolerant in nature, the individual swarm agents themselves being cost effective alternatives to the solution. Developing swarm technology aids in large scale data collection and environmental exploration, package delivery, warehouse management, military reconnaissance, and search and rescue. With the growing interest in developing swarm systems, much researcher and developer time is spent migrating, integrating, and coordinating these various swarm solutions that become deprecated and isolated from the community in frequent cycle. Furthermore, despite the aforementioned swarm architectures provided insights into specific design considerations, there still lacks a generalized architecture that outlines a full swarm pipeline, and is modular enough in design to be readily interchanged with new components as both research and industry advance. We present an architecture for developing full stack swarm systems. Such a system must allow for easy design, deployment, interaction, and evaluation. Using our proposed architecture, we then implemented a framework titled: CMUSWARM, on the ROS platform using Gazebo with irobot create for simulation. We then conduct an evaluation of our architecture by comparing two simple swarm control-laws within the framework.**

I. Introduction

Robot swarms are homogeneous multi-robot systems that form collective behaviour from decentralized local interactions. Swarms are robust and fault tolerant in nature, with individual swarm agents being inexpensive. As individual mobile robots become more robust to real-world conditions [1][6], there is growing interest in swarm robotics [2][5][7]. Developing swarm technology aids the data science industry by allowing for large-scale data collection and environmental exploration [9]. Swarms also show promise in object transportation [10][8], military reconnaissance[15], and search and rescue [13][14].

Multi-robot and swarm algorithms have been developed over the years addressing problems such as navigation [16][19], exploration[20], and coverage [18]. Middleware platforms such as ROS, and Player [30] have an associated community of developers and their software that enables these algorithms to be deployed on a variety of robots [12]; however there is a need for separate bench-marking components in order to evaluate performance. Simulation tools often come prepackaged with performance evaluation tools, and have accelerated the community by allowing multi-robot and swarm systems to be visualized without the complexity and cost of real world experimentation. Platforms include Argos [21], SwarmSimX [22], Menge [23], MATLAB MRSim [26], and Stage [30]. Without middleware such as ROS, these simulators do not allow for deploying swarm algorithms on real robots. The TeleKyb framework merges the ROS middleware with the SwarmSimX simulator using MATLAB/Simulink to provide both the ability to operate a swarm, and to evaluate swarm performance [25]; however this framework is narrow in focus on particular UAVs, and has quickly become deprecated as of ROS fuerte without generalizing the design to other swarms. Recent development of abstraction layers attempt to generalize from specific swarm frameworks to more broad architectures of swarm systems. Current swarm architectures support interaction with a swarm; but are constrained to centralized algorithms [24]. Other designs support both decentralized and centralized swarms but are designed primarily around one aspect of a swarm system such as communication [27], and don't enable interaction with the swarm, or evaluation of performance [31].

This highlights a significant obstacle in the swarm robotics community: the lack of a generalized design pattern (architecture) for creating these swarm systems. Much developer and researcher time is thus spent following a trail of deprecated software and migrating between separate esoteric components in hopes of creating hybrid software for their particular use-case. Furthermore, when a framework is showing great potential in the community [25], the sufficient components in the given framework are not abstracted away to highlight necessary components for any swarm system. As a result, such contributions become isolated for one particular application and deprecated in short time.

We present an architecture that provides a generalized solution for developing full stack swarm systems. A full stack swarm system must allow a developer to **a) design** of a swarm behaviour library by integrating existing centralized or decentralized algorithms, **b) deploy** swarm algorithms on various real robots **c) operate** the swarm through human control and **d) evaluate** algorithm performance through various metrics. Using our proposed architecture, we then implemented a framework titled: CMUSWARM, on the ROS platform using Gazebo with irobot create for simulation. In section II, we outline the architecture design, and then describe our implementation on the ROS platform in section III. Section IV demonstrates the usage of our Framework by conducting an experimental evaluation of two decentralized swarm algorithms We then discuss the various decisions of our work and future research direction.

II. Architecture Design

In this section we cover the architecture components and design decisions. We design by keeping closely in mind the use cases of operators and developers in the swarm robotics community. An operator will be primarily concerned with sending behaviour requests to a swarm and configuring such behaviours, whereas a developer is most concerned with integrating new algorithms and components seamlessly. As such, modularity and simplicity are the fundamental design philosophy.

A. High Level Overview

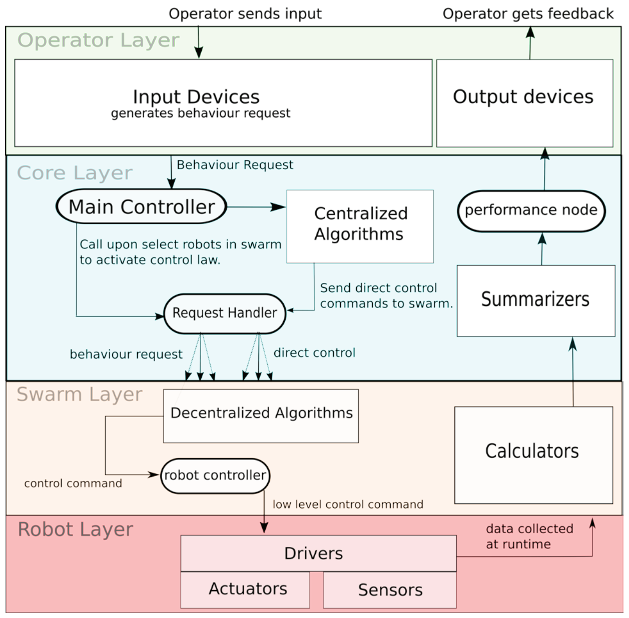

Fig. 1. High level overview of the architecture

The architecture consists of four layers (operator layer, core layer, swarm layer, and robot layer). Both inter-layer and intra-layer information is communicated asynchronously through nodes using a distributed publisher subscriber pattern [4]. This pattern favors the decentralized nature of a swarm in that it provides scalable communication with low coupling. It is important to note that while the swarm algorithms may be decentralized, they are deployed from a centralized core layer. Each layer is also assumed to have a local database in which any node in the layer may access at runtime to gather configuration parameters. The following sections will further outline each layer, it's configuration, and design considerations.

B. Operator Layer

The operator layer allows for interaction with the system. An external input device must act as an adapter to the core layer by generating a behaviour request. A behaviour request is a message consisting of a behaviour name, which corresponds to the algorithm to deploy in the swarm system, along with a list of robot ids, that identify the robots in the swarm to control with this request. Further behaviour specific information may be appended to the request through key-value pairs at run-time. An example of an input device is a joystick. Moving the joystick left may translate to a behaviour request named move_left, a list of all robot ids in the swarm, and additional velocity information with magnitude depending on the joysticks angle of offset. This message will be sent to core layer and received by the main controller.

C. Core Layer

The core layer receives behaviour requests and determines how to toggle the requests upon the swarm.

1) Main Controller: The main controller is a node that first receives the behaviour request. The behaviour name is looked up in a local database for an associated record. If the behaviour name is valid, then static information will be appended to the request from its record. In the example of a joystick generating a move_left_bc request, this will be looked up by the main controller. The barrier certificates portion of the request requires additional information such as robot collision radius, which will be appended from the database to the behaviour request. Each behaviour record in the database must also specify if it is centralized or not. In the case of a centralized behaviour, the updated behaviour request will be remapped to a centralized algorithm node on the topic behaviour_name/control_request. A decentralized behavior request will be delegated to the request handler directly, to be further spread out to the swarm.

2) Request Handler: The request handler receives behaviour requests from the main controller and publishes the request to each robot listed in the request. Alternatively, a centralized node will compute an explicit command for each robot. The request handler will distribute each command to a corresponding robot in the swarm. The request handler also ensures that the swarm robots have disabled active behaviours prior to toggling a new one. In the event that multiple behaviours are requested rapidly, the request handler will schedule them across some time interval.

D. Swarm Layer

The swarm layer is decentralized and thus refers to each robot separately in the swarm. This layer contains a node for each supported algorithm (ie behaviour) that the robot can perform. When each robot in the swarm performs the same desired algorithm, the global behaviour emerges. Each robot falls into its own topic namespace, and as such receives commands to its local node on /{robot_id}/{behaviour_name}/control_requests. Each algorithm node in this layer will publish the computed velocities to a robot controller, which further delegates to the robot layer. In the case of a centralized algorithm, a direct command is propagated to this robot controller.

E. Robot Layer

The robot layer contains the drivers and other on-board software for operating the swarm robot. In the case of a simulation, the robot layer may redirect output commands

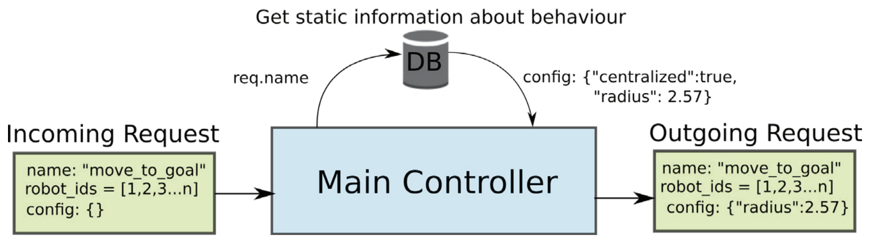

Fig. 2. The process of gaining information about incoming messages to the main controller.

to a separate program which will then simulate the robot's trajectory.

F. *Performance Evaluation*

This performance component is spread across the robot, swarm, and core layers of the architecture using a series of what we denote as calculators, and summarizers. To outline how performance metrics are calculated by propagating information through the system, we will use the example of a work summarizer.

1) Calculators: Each robot may contain a collection of calculators, located in the swarm layer. Each calculator reads information gathered during a trial. For example, a work summarizer may use odometry to track the distance a robot has travelled. The calculator will then publish results to a corresponding summarizer. It is important to note that in a real-world evaluation, the onboard sensors of a robot will be inaccurate, and thus the calculators would publish inaccurate data. We acknowledge this; and made this decision choice despite it. During a real-world evaluation, if there exists some supervisory process monitoring each robot in the swarm, it may act as the calculator and communicate whatever precise information it has captured. There exists inaccuracy in any real-world sensor, thus we are not concerned with this matter in our performance evaluation. Furthermore, tasking each robot with providing performance calculations allows for effective debugging of a real-world system, as inconsistencies may be found between measured, and predicted data.

2) Summarizers: Summarizer nodes are centralized in the core layer and receive calculations from each active robot in the swarm. The summarizer then evaluates this information and organizes it. In the case of a work summarizer, it would collect each robots indivdiual distance information, and compute the sum and average to give insights about the whole swarms travel. Each summarizer must have access to termination conditions stored in the core layers local database. These termination conditions ensure the summarizer will either halt after a maximum execution time, or when some other termination condition is met.

3) Performance Node: A single performance node subscribes to each summarizer, and will save performance summaries in a readable format. This may be in a database, excel spreadsheet, etc. Furthermore, results can be mapped to an output device and shown to the operator.

III. Framework Implementation

We present a framework titled: CMUSWARM, which implements the aforementioned architecture. The framework was implemented on the ROS platform and tested with both indigo and kinetic. ROS provides the publisher and subscriber pattern required along with a parameter server that acts as the database for the system. Gazebo was used as a simulation platform with iRobot create sdf modelled robots, however the framework may be used on real robots supported by ROS compatible drivers.

Three performance metrics have been implemented.
work: track the overall distance travelled by all robots in swarm
coverage: using the gridmap_ros package [28], this will track how much of the map was covered by the swarm. Each robot covers the area created by a disk with the radius of the robot.
collisions: tracks the number of collisions a robot has with nearby obstacles, assuming it can read in the environmental information.
Experimental trials are run by having a supervisory process fork a roslaunch instance of the framework. The launch file is provided with arguments generated dynamically by the supervisory program, which specifies where robots and obstacles are located, as well as the termination conditions and any other arguments required for an instance of the framework. Once the framework is launched, a mock_publisher node will sent a behaviour request to activate the algorithm in question. The performance is then evaluated by the work, coverage, and collision calculators until their corresponding summarizers terminate (either by termination condition, or by max_execution time). The results are saved and the supervisory monitor will kill the child framework instance and create a new one.

A. *Algorithms and Dynamics*

In order to demonstrate and verify the functionality of the framework and implementation, a number of swarm control algorithms were implemented and some benchmarked on the machine. They are also used as a ground-truth comparison for any future work we do using CMUswarm. The algorithms currently implemented are:

- Move to goal
- Flocking
- Rendezvous
- Formation
- Voronoi static coverage

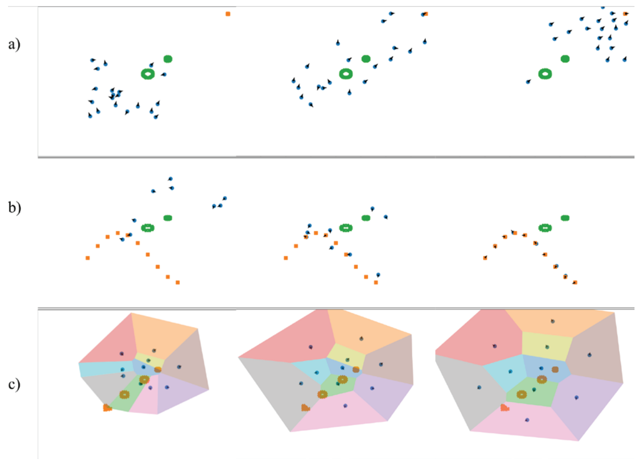

Fig. 3. Our independent script implementation of common control laws outside of the Framework. a) The move_to_goal behaviour with barrier certificates, b) formation control with barrier certificates c) voronoi static coverage

The details and guarantees for the algorithms are detailed below.

The robot model used for all algorithms was the following:

$$\begin{bmatrix}\dot{x}_i\\ \dot{y}_i\\ \dot{\theta}_i\end{bmatrix} = \begin{bmatrix}cos(\theta_i) & 0\\ sin(\theta_i) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_{i,v}\\ u_{i,w}\end{bmatrix}$$

where v and w are the control linear and angular velocities, respectively. Following the definition in [3], we represent the ith robot by its position vector $\boldsymbol{p}_i \in \mathbb{R}^2$ and heading vector in the direction of θ_i, $\boldsymbol{b}_i \in \mathbb{R}^2 : \|\boldsymbol{b}_i\|_2 = 1$.

Remark: The framework is not restricted to this robot model, each algorithm is free to utilize any desired robot model.

Let S represent the set of robots used in simulation. We calculate the change in heading and update u_v and u_w depending on the desired velocity pointed towards the target, $\dot{\boldsymbol{p}}$:

$$u_{i,v} = K_v * (\boldsymbol{b}_i)^T \dot{\boldsymbol{p}}$$

$$u_{i,w} = K_w * w$$

The algorithms differ in their calculation of $\dot{\boldsymbol{p}}$ and w. The velocity relations are discretized into difference equations, outputting a $\boldsymbol{dv}$ vector from an input of $\boldsymbol{p}_k$, $k \geq 0$.

The robot dynamics are updated according to $\boldsymbol{v_{k+1}}$:

$$\boldsymbol{v_{k+1}} = \boldsymbol{v_k} + \boldsymbol{dv}$$

$$d\theta = atan2(\frac{\boldsymbol{v_{k+1}}[1]}{\boldsymbol{v_{k+1}}[0]}) - \theta_k$$

$$w = atan2(\frac{\sin(d\theta)}{\cos(d\theta)})$$

$$\boldsymbol{b} = \begin{bmatrix}\cos(\theta_k)\\ \sin(\theta_k)\end{bmatrix}$$

The linear and angular velocities are capped at a maximum:

$$u_v = \max(-U_{v_{max}}, \min(U_{v_{max}}, u_v))$$

$$u_w = \max(-U_{w_{max}}, \min(U_{w_{max}}, u_w))$$

The positions of the robots are then updated to follow the dynamics:

$$\boldsymbol{p_{k+1}} = \boldsymbol{p_k} + \begin{bmatrix}u_v \cos\theta dt\\ u_v \sin\theta dt\end{bmatrix}$$

$$\theta_{k+1} = wraptopi(\theta_k + u_w dt)$$

where the $atan2$ function is a four-quadrant inverse tangent, and $wraptopi$ is a function used to wrap angles in radians to $[-\pi, \pi]$.

It is important to note that every algorithm performed inter-robot and obstacle collision avoidance in the form of barrier certificates [11], a obstacle avoidance method chosen due to its available collision avoidance guarantees on holonomic agents and its minimally invasive nature which allows for obstacle avoidance only when absolutely necessary. It is achieved using a dynamic programming approach outlined in [11]. Every algorithm presented therefore possesses an underlying control in the form of barrier certificates that only intervenes when it is absolutely necessary in order to avoid a collision, and it does so while minimizing the difference between the avoidance trajectory and the original intended trajectory. Due to our non-holonomic robot model, the barrier certificates algorithm still allowed for collisions to happen in simulation, though less than in the no obstacle avoidance case (see Figure 4).

1) Move to goal: The move to goal algorithm involves a simple proportional controller that leads the robot towards the desired position. The algorithm consists of a discrete loop of the following differential equation:

$$\dot{\boldsymbol{p}} = -K_i(\boldsymbol{p} - \boldsymbol{E})$$

where d is the distance of the robot to the goal position E, $K_i > 0$. The solution

$$\boldsymbol{p}(t) = \boldsymbol{p}(0)e^{-K_i t} + \boldsymbol{E}$$

guarantees convergence to the desired point $\boldsymbol{E}$ since K_i is positive definite. This is approximated using a difference equation:

$$\boldsymbol{dv} = -K_i(\boldsymbol{p_{i,k}} - \boldsymbol{E})$$

The dynamics are then updated according to the robot model. An independent script implementation of this algorithm is seen in Figure 3 a).

2) Flocking: The flocking algorithm is inspired by the known boids algorithm [17]e and [3], using the three steers of separation, cohesion, and alignment. Specifically, the following control laws were utilized for each steer:

Remark: The distance vector from agent i to agent j is noted as $\boldsymbol{d} = \boldsymbol{p}_{ij} = \boldsymbol{p}_j - \boldsymbol{p}_i$.

Separation

$$\boldsymbol{dv} = -\frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_{\boldsymbol{2}}}$$

$$d\theta = atan2(\frac{\boldsymbol{dv}[1]}{\boldsymbol{dv}[0]}) - \theta_i$$

Alignment

$$\boldsymbol{dv} = \boldsymbol{0}$$

$$d\theta = \theta_j - \theta_i$$

Cohesion

$$\boldsymbol{dv} = \boldsymbol{d}$$

$$d\theta = atan2(\frac{\boldsymbol{dv}[1]}{\boldsymbol{dv}[0]}) - \theta_i$$

The robot positions are then updated according to the previously stated dynamics, and each control law is activated once $\boldsymbol{d}$ reaches certain physical thresholds of separation, alignment, and cohesion defined by the user, termed $repulsion\_radius, alignment\_radius, attraction\_radius$ in our implementation.

3) Rendezvous: This control law is described in [3] and controls the robots such that they meet at a common point. Define $\boldsymbol{d}$ as before: $\boldsymbol{d_{ij}} = \boldsymbol{p}_{ij} = \boldsymbol{p}_j - \boldsymbol{p}_i$. The following update is provided to $\boldsymbol{v}$ in order to produce the rendezvous behaviour:

$$\boldsymbol{dv_i} = \boldsymbol{dv_i} + \boldsymbol{d_{ij}}, \forall j \neq i \in S$$

$\boldsymbol{v_i}$ is then normalized by $\|S\|$ and the dynamics follow from above.

4) Formation: In the formation control algorithm, each agent is provided with a target position E_i, and they follow a simple proportional controller towards this goal. Recall obstacle avoidance and inter-robot collision avoidance are handled using barrier certificates.

$$\boldsymbol{dv} = -K_i(\boldsymbol{p_{i,k}} - \boldsymbol{E_i})$$

And the dynamics follow. An independent script implementation of this algorithm is displayed in Figure 3 b).

5) Voronoi Static Coverage: The voronoi static coverage algorithm addresses the multi-robot problem of maximizing static coverage over a certain area. The solution implemented follows that of [Lloyd's algorithm] where a controller moves the robots to the centroid of their respective voronoi regions (denoted as $\boldsymbol{C_i}$). The voronoi regions were constructed for each robot, and the following controller was then applied:

$$\boldsymbol{dv} = -K_i(\boldsymbol{C_i} - \boldsymbol{p_{i,k}})$$

followed by the dynamics update detailed above. The voronoi static coverage method outlined by Lloyd's Algorithm guarantees convergence to the optimal static coverage configuration for holonomic robots. An independent script implementation of this algorithm is displayed in Figure 3 c).

The algorithms were successfully implemented in our framework and they cover different important areas of multi-robot algorithms: coverage, formation, and navigation, all with obstacle avoidance. It is shown in simulation that the framework is well-adept for rapid porting and automated benchmarking of such multi-robot algorithms. Figure 3 highlights a simple independent prototype implementation of three of the above algorithms. We initially wrote the controllers independently from the framework, as seen in Figure 3, and we later integrated them into the framework to evaluate the ease of integration.

IV. Experimental Evaluation

We evaluate the performance component of the framework using gazebo_ros on the ROS indigo and kinetic. Two navigation control laws, namely move to goal, and move_to_goal_bc are compared across four scenarios using 4, 8, 16, and 32 robots for 20 trials each totalling 640 trials. The purpose was to highlight the use of the framework for evaluating swarm algorithms.

A. Setup

The swarm is homogeneous and each robot represented as an irobot create sdf file in gazebo. Obstacles are static and modelled as 1x1x1 boxes. A swarm robot has a sensing radius of 3 meters, as read from gazebo model positions, and is assumed to have perfect sensing ability to obtain local information. The robots max velocity is 4 meters/sec, and maximum angular velocity is pi/4 radians. Four environment scenarios are used, each a 20x20 meter world generated in gazebo. The swarm robots are spawned in randomly generated locations with some specified region, and must navigate to a standard goal region within 1 minute.

Scenario 1: Empty World: This world is empty, with robots spawning in the upper plane defined as $R1 = (x, y)$ such that $x \in [1, 6], y \in [1, 19]$. This is a minimal Scenario to serve as easy comparison.

Scenario 2: Empty Dense World This world is also empty, with a smaller spawning region for robots defined as $R2 = (x, y)$ such that $x \in [1, 6], y \in [1, 6]$. The goal region G is a disk of radius 4 centered at $(17, 17)$. This scenario explores algorithm performance when swarm robots begin within close proximity of each other. All other scenarios use the larger radius except this one. The reason, is that if an algorithm struggles when initialized in a dense environment, it wont matter what other obstacles are elsewhere, thus replicating this density on any other world would be extraneous.

Scenario 3: Uniform World. Robots are spawned in the region $R3 = (x, y)$ such that $x \in [1, 6], y \in [1, 19]$. Obstacles are spawned across the region $O = (x, y)$ such that $x \in [6, 16], y \in [1, 18]$. The goal region G is a disk of radius 4 centered at $(17, 17)$. This scenario evaluates basic collision avoidance ability.

Scenario 4: Concave World. This world uses a fixed map with no randomly generated obstacles. Robots are spawned in the region $R4 = (x, y)$ such that $x \in [1, 6], y \in [1, 19]$. The goal region G is a disk of radius 4 centered at $(18, 18)$. This scenario evaluates complex collision avoidance in a non-convex environment.

Additional environments have been provided with the framework, such as a corridor environment with a long passageway. We dont evaluate proportional control laws on such an environment, as the effort would be wasted. A planning algorithm would be needed instead.

B. Comparison Results

The benchmarking capabilities of the framework allowed for the sample comparison of $move\_to\_goal$ and $move\_to\_goal\_bc$ algorithms (proportional controllers with and without obstacle avoidance, respectively). Sample data is displayed for the Concave Map and the Uniform Map configurations (see Figures 4 and 5). The data confirm our predictions that control algorithms that did not involve Barrier Certificates (no avoidance) yielded a larger number of robot collisions in simulation, both for the Concave and Uniform Map scenarios. This test was performed in order to evaluate the benchmarking capabilities of the framework, confirmed by the output of logically coherent results.

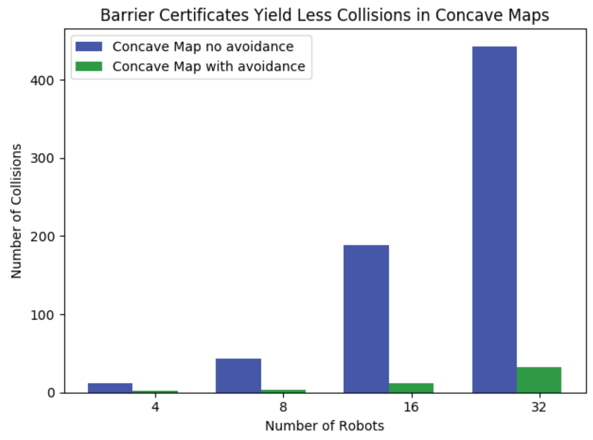

Fig. 4. Data on collisions in Concave Map.

V. Discussion and Conclusion

We presented an architecture that provides a design pattern for developing full stack swarm systems. This entails 4 primary layers: design (through building a swarm behaviour library), deployment (on various systems), interaction (through operator control) and evaluation(through benchmarking) of the swarm. Using our proposed architecture, we then implemented a framework titled: CMUSWARM, on the ROS platform using Gazebo with irobot create for simulation. We then conduct an evaluation of our architecture by comparing two simple swarm control-laws within the CMUSwarm framework. Our work generalizes existing swarm architectures by focusing not on one particular subsystem, or existing framework; but rather the challenge of designing a full stack system that is platform independent. Such design patterns are required to allow for rapid integration and evaluation of new methods and technologies. There remains much additional work we will focus on in the future that will better highlight the usage of our proposed architecture. Using our implemented framework CMUSWARM, we wish to address the following issues: **a)** Providing a wider span of algorithms, input devices, and evaluation data for the framework. **b)** Extending the architecture to support heterogeneous swarms **c)** Performing human-swarm interaction experiments using CMUSWARM **d)** Alternative extension of our framework using a DDS vendor for publisher-subscriber architecture on the ROS2 platform.

Our work in this paper presents the architecture and brief overview, but the effectiveness of our design will best be demonstrated through the proposed future work. We hope that our proposed architecture will be used as a standard for developing further swarm systems. By using a common overarching design, the swarm research community will become less decentralized and able to focus on moving forward.

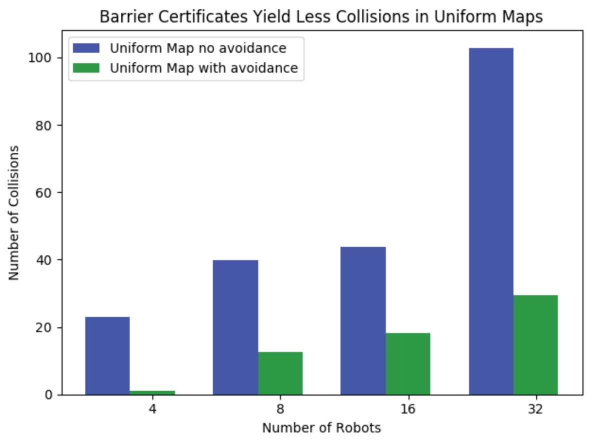

Fig. 5. Data on collisions in Uniform Map.

References

[1] R. Gerndt, D. Seifert, J. Baltes, S. Sadeghnejad, and S. Behnke, "Humanoid robots in soccer: Robots versus humans in robocup 2050," *IEEE Robotics and Automation Magazine*, vol. 22, no. 3, pp. 147–154, 9 2015.

[2] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm*

Intelligence, vol. 7, no. 1, pp. 1–41, Mar 2013. [Online]. Available: https://doi.org/10.1007/s11721-012-0075-2

[3] S. Nagavalli, "Behavior composition in human interaction with robotic swarms," PhD Thesis Proposal, Carnegie Mellon University, 2017.

[4] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 123–138, Nov. 1987. [Online]. Available: http://doi.acm.org/10.1145/37499.37515

[5] E. Şahin, *Swarm Robotics: From Sources of Inspiration to Domains of Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 10–20. [Online]. Available: https://doi.org/10.1007/978-3-540-30552-1_2

[6] J. Bjerknes and A. Winfield, "On fault tolerance and scalability of swarm robotic systems," *Distributed autonomous robotic systems*, pp. 431–444, 2013.

[7] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "Nasa's swarm missions: The challenge of building autonomous software," *IT professional*, vol. 6, no. 5, pp. 47–52, 2004.

[8] M. H. M. Alkilabi, A. Narayan, and E. Tuci, "Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies," *Swarm Intelligence*, pp. 1–25.

[9] M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. M. Marques, S. M. Oliveira, and A. L. Christensen, "Application of swarm robotics systems to marine environmental monitoring," in *OCEANS 2016 - Shanghai*, April 2016, pp. 1–8.

[10] M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, and R. Nagpal, "Collective transport of complex objects by simple robots: theory and experiments," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 47–54.

[11] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Trans. Robotics*, vol. 33, no. 3, pp. 661–674, 2017. [Online]. Available: https://doi.org/10.1109/TRO.2017.2659727

[12] 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.

[13] M. Bakhshipour, M. J. Ghadi, and F. Namdari, "Swarm robotics search & rescue: A novel artificial intelligence-inspired optimization approach," *Applied Soft Computing*, 2017.

[14] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2016.

[15] R. Bogue, "The role of robots in the battlefields of the future," *Industrial Robot: An International Journal*, vol. 43, no. 4, pp. 354–359, 2016. [Online]. Available: https://doi.org/10.1108/IR-03-2016-0104

[16] L. He, J. Pan, W. Wang, and D. Manocha, "Proxemic group behaviors using reciprocal multi-agent navigation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 292–297.

[17] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/37401.37406

[18] G. M. Atin, D. M. Stipanovi, P. G. Voulgaris, and M. Karkoub, "Swarm-based dynamic coverage control," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 6963–6968.

[19] W. Luo, N. Chakraborty, and K. Sycara, "Distributed dynamic priority assignment and motion planning for multiple mobile robots with kinodynamic constraints," in *American Control Conference (ACC)*, July 2016.

[20] A. Viseras, T. Wiedemann, C. Manss, L. Magel, J. Mueller, D. Shutin, and L. Merino, "Decentralized multi-agent exploration with online-learning of gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 4222–4229.

[21] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[22] J. Lächele, A. Franchi, H. Bülthoff, and P. Robuffo Giordano, "Swarmsimx: Real-time simulation environment for multi-robot systems," *Simulation, modeling, and programming for autonomous robots*, pp. 375–387, 2012.

[23] S. Curtis, A. Best, and D. Manocha, "Menge: A modular framework for simulating crowd movement," *Collective Dynamics*, vol. 1, no. 0, pp. 1–40, 2016.

[24] F. Aznar, M. Sempere, F. Mora, P. Arques, J. Puchol, M. Pujol, and R. Rizo, "Agents for swarm robotics: Architecture and implementation," in *Highlights in Practical Applications of Agents and Multiagent Systems*. Springer, 2011, pp. 117–124.

[25] V. Grabe, M. Riedel, H. H. Bulthoff, P. R. Giordano, and A. Franchi, "The telekyb framework for a modular and extendible ros-based quadrotor control," in *Mobile Robots (ECMR), 2013 European Conference on*. IEEE, 2013, pp. 19–25.

[26] MATLAB. (2012) Mrsim - multi-robot simulator. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/38409-mrsim-multi-robot-simulator–v1-0-

[27] J. Vain, T. Tammet, A. Kuusik, and E. Reilent, "Software architecture for swarm mobile robots," in *Electronics Conference, 2008. BEC 2008. 11th International Biennial Baltic*. IEEE, 2008, pp. 231–234.

[28] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS) The Complete Reference (Volume 1)*, A. Koubaa, Ed. Springer, 2016, ch. 5. [Online]. Available: http://www.springer.com/de/book/9783319260525

[29] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "Buzz: An extensible programming language for self-organizing heterogeneous robot swarms," *arXiv preprint arXiv:1507.05946*, 2015.

[30] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems."

[31] K. Hawick, H. James, J. Story, and R. Shepherd, "An architecture for swarm robots," *Computer Science Division, School of Informatics University of Wales, North Wales, UK*, 2002.