gestalt-arch: Architecture for Decentralized Robotics

Daniel Lovell, Faywer Liu, Clare Lin, Haochen Gao
dlovell98@berkeley.edu, faywer_liu@berkeley.edu, 2cyl@berkeley.edu, haochengao@berkeley.edu

EECS 149/249A, December 17, 2021, UC Berkeley

https://github.com/gestalt-arch/gestalt-arch

ABSTRACT

We demonstrate a decentralized robotics control architecture intended for environments in which networks of robots must be robust and tolerant to losses of communication or sensor feedback. Furthermore, we employ this architecture in a logistics network scenario, demonstrating that a fully-connected communication topology is achieved and is trivially extended within our architecture to enable fault and loss tolerance. Incidental to this architecture, we also propose a LIDAR-based localization method which we believe to be well-suited for resource-constrained embedded systems where SLAM is too computationally expensive.

1 INTRODUCTION

With the increasing adoption of logistics robotics in warehouse and construction environments, safe and collaborative robot control architectures are required to increase productivity and reduce downtime.[1][2] Decentralizing the control architecture can increase safety and robustness to robot equipment malfunction, signal loss, and other unforeseen failures in these settings.

To this end, we propose, design, and implement a decentralized robotics control architecture intended for these environments, leveraging a fully-connected network topology for Autonomous Mobile Robots (AMRs) robots operating concurrently in crowded environments.[3] This architecture, which we name *gestalt-arch*, provides a framework for sensor and actuator fusion on-board the AMR, enabling positional and rotational state tracking through time. Perhaps more crucially, it defines the networking implementation to facilitate the core decentralization schema. This allows reconciliation of any arbitrary data within the robot network as well as providing the scaffolding for user-defined loss tolerance rulesets.

In order to explore the benefits of this decentralized control architecture, we additionally demonstrate *gestalt-arch* as the architecture of a logistics network scenario. This scenario involves three robots, based on the iClebo Kobuki platform, navigating arbitrary path-streams within a physical environment. We define this scenario to emulate a busy section of a warehouse floor, where the robots must navigate and avoid each other without a central server/signaling actor. We then extend the scenario to emulate the loss of one of the robots, subsequently demonstrating the capacity of our architecture for arbitrary fault/loss handling rulesets. Upon the disconnection of one of the robots, we show that the other members of the network can acknowledge this loss and collaborate to take over the lost robot's tasks.

To accomplish this demonstration, we utilize the aforementioned iClebo Kobuki platform, the nRF52832 from Nordic Semiconductor, as well as the Berkeley Buckler development board and corresponding software libraries from UC Berkeley's Lab11.[4]

Incidental to our efforts to achieve high-accuracy positional and rotational state tracking, we also explore and propose a LIDAR-based localization method. This method is based on geometric resection and leverages a mechanical spinning LIDAR sensor. We believe it to be well-suited for resource-constrained embedded systems such as the nRF52832 where SLAM can be too computationally expensive to be viable.

[1] Pagliarini, L & Lund, H. *The Future of Robotics Technology*. ICAROB 2017.
[2] Matthews, K. "How Warehouse Robotics Reduce Worker Injuries". EHS Today. Aug. 15, 2019.
[3] *Autonomous Mobile Robots (AMR) Overview: Types and Use Cases*. Intel. (n.d.)

[4] UC Berkeley Lab11. *Lab11/Buckler: Development Board for teaching embedded systems*. GitHub. https://github.com/lab11/buckler

2 ARCHITECTURE

2.1 "Gestalt Architecture"

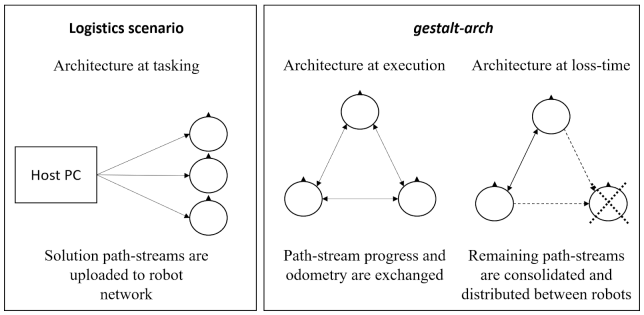

Figure 1. High-level architectural diagram depicting the communication for *gestalt-arch* as well as for the logistics scenario demonstration.

Relevant to its namesake, the Gestalt Architecture (*gestalt-arch*) aims to create a fully-connected topology for a network of independent and autonomous robots whereby the behaviors achievable by the network exceeds that which each robot could accomplish individually.

The architecture specifies this fully-connected network topology to host a decentralized network of robots, where each robot combines several independent sensors to track its position and rotation through time. The robots in the network then share their state, odometry, and progress along a given task with each other. Nominally, there does not exist a master/central actor in this network. Every member of the network tracks a record of the other member's information and the overall tasks of the network as a whole. This last point is critical to the loss-tolerance features of *gestalt-arch*. Because every robot has knowledge of the tasks of the others, if it determines the loss of another member of the network, it can apply any arbitrary loss-tolerance ruleset.

For our logistics scenario demonstration, we leverage the ability of the robots to share odometry to avoid collisions before any contact is allowed to happen. Notably, this is accomplished without any externally-facing sensors (e.g. cameras or ultrasonic sensors). Next, we leverage the loss-tolerance features and apply our own arbitrary loss-handling ruleset to ensure that the robots can complete all the prescribed tasks in the event one or more bots is catastrophically disconnected from the network.

Now, we fully define the fine-grain application of *gestalt-arch* to the demonstration in two stages: *Tasking/Host PC* and *Execution/Robot Network.*

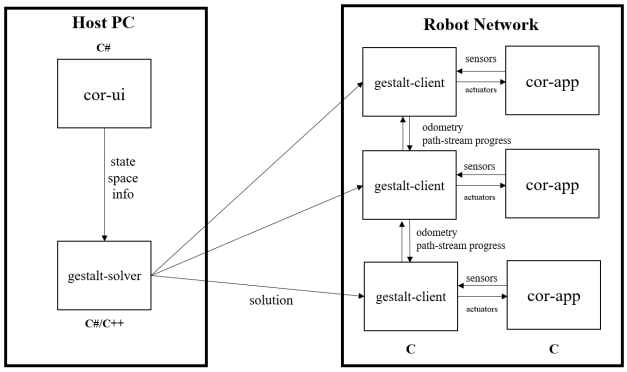

Figure 2. A finer-grain architectural overview which details the submodules of the *gestalt-arch* and how they are connected for the logistics scenario demonstration.

Stage 1 - Tasking/Host PC
The first stage of the architecture is executed on a remote, host PC running Windows or Ubuntu Linux. The first actor, *cor-ui*, is a Unity-based program that provides the user interface to the architecture, allowing the initial and final state spaces to be defined as well as facilitating interaction with the *gestalt-solver*. The *gestalt-solver* is a library written in both C# and C++ that solves and generates path-streams for each robot based upon the initial and final state spaces. The navigation tasks, designed to emulate the delivery of items from one location to another, are specified within the *cor-ui* interface by the user at tasking-time.

Stage 2 - Execution/Robot Network
The second stage of the architecture begins once all robots within the network have received the path-stream solution over BLE. This BLE communication is implemented within the *gestalt-client* C library linked to the *cor-app* firmware application. In addition to implementing the specifics BLE communication, *gestalt-client* stores the path-stream and updates *cor-app*'s present goal navigation destination based on sensor data that is integrated and combined within the *gestalt-client*. More explicitly, the *gestalt-client* fuses all available sensors on the robot, including LIDAR, accelerometer, gyroscope, and motor/wheel encoder sensor data to determine its own position and the position of other robots. In our final implementation, due to challenges with integration of the LIDAR, the *gestalt-client* relied exclusively on inward-facing sensors, pertaining to its own odometry, to resolve its physical state through time.

3 DESIGN AND IMPLEMENTATION

3.1 Fully-connected BLE network

A decentralized communication scheme between all the robots within the *gestalt-arch* network at runtime is achieved by utilizing the nRF52832's BLE functionality. Specifically, we implement BLE's Generic Access Profile (GAP) to encode and advertise (broadcast) a robot's odometry and path stream progress. Each robot then switches states, scanning for other robot's advertised odometry.

This particular approach to achieve a fully-connected network was chosen with several advantages and disadvantages in mind. In typical BLE implementations, communication is implemented using the core BLE data connection profiles: GAP and the Generic Attribute Profile (GATT)[5]. Summarily, GATT provides a one-to-many connection from central to peripheral actors and allows larger quantities of data to be exchanged. However, in the GATT scheme, once a device's role as a peripheral device has been established it nominally cannot host any other connections. This precluded our use of GATT as our architecture specified a fully-connected topology.

GAP provides the preceding conditions to a GATT connection, whereby devices advertise details of their state by broadcasting an advertisement "packet". Of critical note, GAP allows for the inclusion of a small amount (up to 26 bytes) of manufacturer's data in an advertisement. We chose to leverage this user-specified data to create our decentralized BLE network.

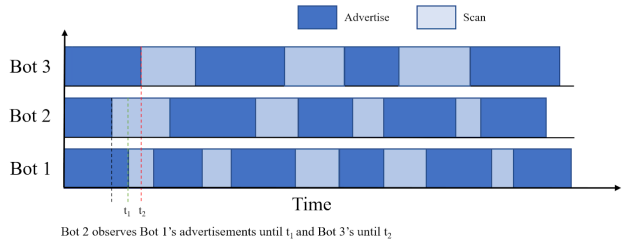

Figure 3. Sample timing diagram for the random scheduling algorithm used to achieve a fully-connected network with BLE's Generic Access Profile (GAP).

With a GAP advertising-based BLE design selected, we evaluated methods for its implementation on our hardware. Due to limitations of BLE's feature-set and hardware on the nRF52832, scanning and advertising cannot be performed simultaneously. To minimize the possibility of missed broadcasts or aliasing between robots where scanning or advertising states are approximately in-phase, we derive a state switching paradigm based on randomized advertising/scanning intervals. This random scheduling scheme effectively prevents aliasing and guarantees successful exchange of information with a high degree of probability. A visualization of our random scheduling algorithm is shown in Figure 3.

We successfully demonstrated this scheduling scheme in practice on the nRF52832, and the transmission and receipt of encoded data within advertisements was both stable and sufficiently frequent for our application. This demonstration involved two and three robots communicating according to this protocol, and we are confident that it can be extended to several more. However, a more formal analysis of the limitations of this timing scheme with specific constants relevant to the specifications of BLE GAP advertising and scanning intervals would quantify this scheme's efficacy. We considered, but did not explore, probabilistic analysis as discussed by Liu, et. al. in *Random Coverage with Guaranteed Connectivity: Joint Scheduling for Wireless Sensor Networks.*[6] Regardless, we expect that the same scheduling and connectivity benefits found by Liu, et. al. extend to our paradigm, including adherence to coverage constraints and robustness to time asynchrony.

3.3 Host-Network BLE Communication

GATT was used to implement host-to-network BLE communication for its two-way communication capabilities. Upon testing our implementation, we discovered that the path-stream data to be sent to the robot network exceeded BLE packet size limits. This required for the path-stream data to be broken into multiple packets. Upon reception of the first packet, the robot would update its acknowledgement channel, signalling the server to send the following packets. Upon reception of all packets, the robot would re-serialize the complete path-stream and end the bluetooth communication, allowing for the server to make contact with the next robot and the robot to prepare itself for GAP based network communication.

3.2 State tracking: discrete-time sensor fusion

The *gestalt-client* module is responsible for providing accurate positional and rotation state tracking throughout operation of

[5] Stewart, Scott. "What is BLE?" Cardinal Peak. https://www.cardinalpeak.com/blog/what-is-ble-and-how-do-its-related-gap-and-gatt-profiles-work

[6] Liu, et. al. *Random Coverage with Guaranteed Connectivity: Joint Scheduling for Wireless Sensor Networks*. IEEE Transactions on Parallel and Distributed Systems. June 2006.

the robot. State tracking is critical to correct operation of the architecture as navigation, collision avoidance, and task completion is completely dependent on information about the robot's odometry. While the robot operates autonomously and without a centralized actor to corroborate its state, it becomes uniquely responsible for calculating and tracking its state.

In order to accomplish accurate positional tracking, we combine multiple sensors as available on the iClebo Kobuki, nRF52832, and Berkeley Buckler. This included two motor/wheel encoders, two accelerometers, and two gyroscopes. These sensors are inherently sampled in discrete-time by the firmware, however state tracking necessitated interpolation to a continuous-time model.

Through our testing, we determined that the encoders provided the highest degree of accuracy for positional changes between discrete time-steps of program execution cycles. To extrapolate the encoder measurements of each wheel to describe the overall trajectory of the robot, we define two equations which apply in two mutually exclusive scenarios. Let d_{left} and d_{right} describe wheel travel distance in a given time-step:

Scenario 1: Wheels move in opposite directions

$$\theta_{\Delta t} = 360 * \frac{\frac{|d_{left}| + |d_{right}|}{2}}{wheelbase * \pi}$$

Scenario 2: Wheels move in same direction

$$\theta_{\Delta t} = 360 * \frac{d_{left} - d_{right}}{wheelbase * \pi}$$

Using these two definitions, we are able to fully describe the trajectory of the robot as a cumulative sum of encoder measurements through discrete time.

However, this method alone was not enough to provide sufficient description of trajectory, as the encoders were prone to erroneous measurements at random. We then turned to the gyroscopes, of which we had two, to reconcile our encoder's measurements and provide a substitute in the event of erroneous/outlying values. Let $\frac{d\Theta}{dt}$ describe the average of both gyroscope measurements, we then integrate this measurement using a Δt provided by sampling a 1 MHz hardware timer:

$$\theta_{\Delta t} = \frac{d\Theta}{dt}\Delta t$$

Then, in order to smooth the typically noisy gyro readings, we emulate a simple single delay infinite impulse response (IIR) filter where K is a smoothing/scaling factor:

$$\theta_{smooth} = K\theta_{smooth} + (1 - K)\theta_{\Delta t}$$

The final sensor integration included weighted linear combinations of the aforementioned terms, and successfully tracked the robot's position through time for times >5-7 minutes. After which, drift between internal state representation and true localized position became noticeable within the scope of our demonstration. Our further exploration of localization for state tracking using externally-facing sensors (LIDAR) is discussed in section 4.

3.4 **Navigation - Modeling & Verification**

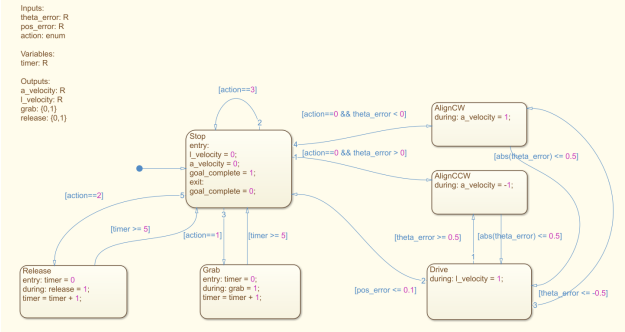

Figure 4. A MATLAB Stateflow® model of the robot waypoint navigation finite state machine.

The ability to accurately navigate between the positional waypoints prescribed by each robot's path-stream is also critical to the implementation of the architecture. This navigation behavior is implemented as a finite state machine (FSM) on the robots' firmware, and is designed to have as minimal states as possible while offloading the capacity for dynamic behavior to the architecture (specifically the *gestalt-client* which drives the FSM's inputs). In an effort to explore unwanted behavior and edge-cases, as well as verify the accuracy of our design, we modeled the Kobuki's navigation FSM using MATLAB's Statecharts modeling tool, Stateflow. Figure 4 shows this Stateflow model, and the four states relevant to navigation: *Stop, AlignCW, AlignCCW, and Drive*, as well as two additional states - *Grab* and *Release* - which were included to model expanded complexity of our architecture. The FSM is modeled as a Moore machine, with output defined based on the present state of the machine.

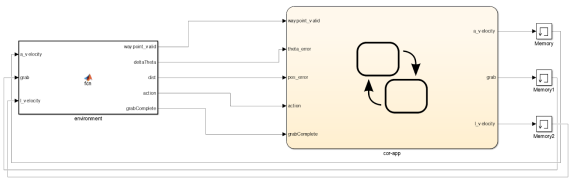

Figure 5. The MATLAB Simulink® model of the *cor-app* navigation FSM connected to the environment simulation for verification.

To verify our navigation FSM over simulated path-streams, we created a Simulink model to model the environment and *gestalt-client* implementation. Within the environment model, non-linearities in motors and drift in sensors were modeled at a high level of abstraction. This Stateflow model is not particularly granular, and was intended to demonstrate feasibility and accuracy of our minimalized waypoint navigation system. Extra effort in modeling was instead directed toward the environment model, and ensuring noise and sensor drift is corrected adequately with state transitions from the *Drive* state back to *Align* states. This error-based feedback driving inputs to our FSM creates basic closed loop control to keep the Kobuki on course within a margin of rotational error.

A caveat to our approach to FSM-based runtime course-correction is that localization must be achieved to determine absolute positions and rotations within the global, physicalized operating space. Recognizing this limitation, our approach toward localization is discussed in the later section on resource-constrained localization using LIDAR.

3.5 **Unity Interface: *cor-ui***

We created a Unity-based user-interface program, *cor-ui*, to interact with our architecture. The program implements the *gestalt-solver* library as well as buttons and UI to set positions of each robot's initial positions, and each waypoints' initial and target positions.

In the Unity program, we place the Kobukis into their starting position, reflecting their positions in the physical setup. The cubes act as markers for arbitrary tasking waypoints, which are passed to the solver for optimal distribution and scheduling between the Kobukis within the network.

3.6 **Tasking-time solver: *gestalt-solver***

The *gestalt-solver* library is implemented in both C++ and C# and linked to our Unity project. The solver interprets the initial and final state space vectors of the selected waypoints, which it uses to generate a tasking solution. This tasking solution is generated to advance the initial state vector to the final state vector through discrete "solution steps" in time. The current implementation is a naive solution for assigning tasks based on optimal distance traveled, but more sophisticated algorithms were attempted. For this naive solution, we assign a solution configuration where the sum of all robot-to-initial waypoint distances is minimized. Next we construct a solution path-stream. Each path-stream in the final solution contains a robot's initial position, assigned waypoint's initial position and final position, and it's initial position again. The solution also contains an action stream, where values at each position node correspond to actions such as move, stop, or any other arbitrary action that can be assigned by the user.

Other solver implementations we attempted but were unable to complete include using a modified A* search algorithm. This algorithm would run traditional A* at each movement step of specified size and consider other Kobukis as obstacles (need to consider Kobuki travel speed). This theoretically would create a collision free solution. We also considered using Unity's NavMesh library where we specify traversable areas as well as agents and obstacles, and later output the AI generated NavMeshPath and insert it into our path stream solution.

4 **RESOURCE CONSTRAINED LOCALIZATION USING LIDAR**

Positional and rotational state tracking through the integration of inward-facing sensors (accelerometers, gyroscopes, and encoders) introduces positional and rotational drift away from global/physical absolute truth. While our sensor-fusion based state tracking implementation is precise, over time it becomes less accurate. We believed that this issue could be solved by using another sensor for localization such as LIDAR. However, due to resource considerations for our chosen hardware, the nRF52832, industry-ubiquitous SLAM implementations were not viable. Instead, we derived a simpler method based on physical localization references placed within the navigation space. Opposed to SLAM, this LIDAR system would require less computation and memory.

Before localization can occur, data must be collected from the LIDAR. The LIDAR sends packets that can be parsed into arrays of angles and corresponding distances over UART. Shown in Figure 6 is an example of what the LIDAR scans and in Figure 7 is the corresponding data collected.

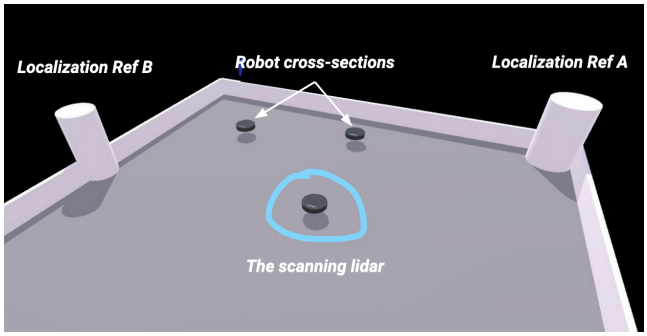

Figure 6. Simulated situation of an example LIDAR scan. The desired objects to be seen are the two other LIDARs and the two reference poles which have different diameters.

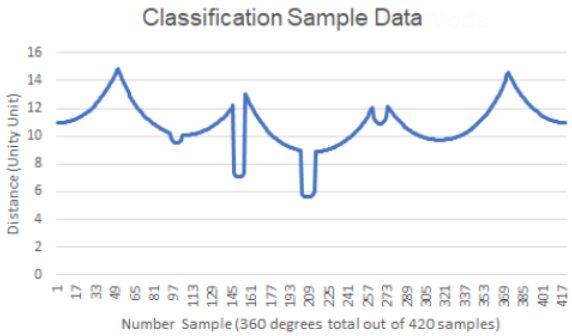

Figure 7. Simulated LIDAR data. The positive peaks are the walls while the negative divots are the objects. The smaller pivots correspond to the two reference poles while the larger pivots are the other two LIDARs.

Our localization algorithm operates in three steps: segmentation, classification, localization. To segment this data, we compute the derivative across the LIDAR sensor data stream, and if the magnitude of that derivative at a specific index (θ) is above a threshold, we consider it to be a unique object. Then to classify, we must resolve the widths of each of these segmented objects. Using the arrays of angles and distances provided by the LIDAR, we can compute the width of each object by applying the Law of Cosines. The known values are d_1, d_2, θ_1, *and* θ_2 as shown below in Figure 8. We solve $\theta = \theta_2 - \theta_1$ and then applying the Law of Cosines to get

$$w = \sqrt{d_1^2 + d_2^2 - 2d_1 d_2 \left|cos(\theta_2 - \theta_1)\right|}.$$

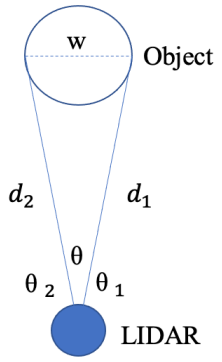

Figure 8. Using the Law of Cosines, we can find the width of each object.

Based on these widths, these objects can be classified based on how close they are to the a priori known widths of the reference poles and LIDARs. The most important objects to be classified are the two localization reference objects whose cross-sections are critically shaped as cylinders to prevent variable widths at different viewing perspectives.

Once the reference poles have been classified, localization based on geometric resection can be calculated as the angles, distances, and predetermined positions of the poles are known as shown in Figure 9.

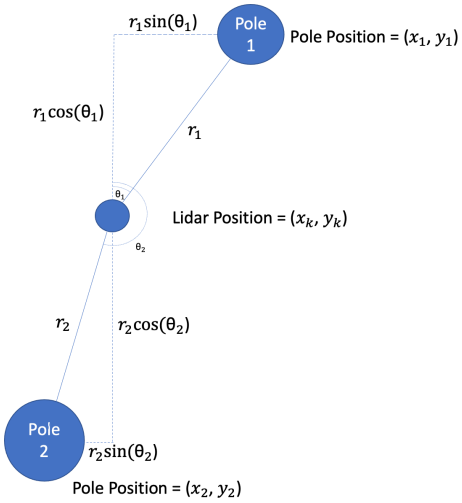

Figure 9. Using trigonometry, we see that $\Delta x_n = r_n sin(\theta_n)$ and $\Delta y_n = r_n cos(\theta_n)$.

Our goal is to calculate the position of the LIDAR, which is nominally centered on top of the robot: (x_k, y_k). To perform this calculation, we first define a system of linear equations extrapolated from the two rays emitted from the robot's center. Then we apply Cramer's Rule to solve for the geometric

intersection of these rays, which fully defines the absolute localized position of the robot in physicalized space. The derivation of this proposed localization method is provided below.

Application of Cramer's Rule specifies a system of linear equations in the standard form:

$$A_n x_k + B_n y_k = C_n \quad (1)$$

We begin with a linear equation in point-slope form.

$$y_k - y_n = m_n(x_k - x_n) \quad (2)$$

Then rearranging into the standard form:

$$m_n x_n - y_n = m_n x_k - y_k \quad (3)$$

Referencing Figure 9, we get:

$$m_n = \frac{\Delta y_n}{\Delta x_n} = \frac{r_n cos(\theta_n)}{r_n sin(\theta_n)} = cot(\theta_n) \quad (5)$$

Then we have

$$A_n = m_n = cot(\theta_n)$$
$$B_n = -1$$
$$C_n = cot(\theta_n)\ x_n - y_n \quad (6)$$

Finally, we apply Cramer's Rule to solve the system for the desired nominal center position.

$$x_k = \frac{B_2 C_1 - B_1 C_2}{A_1 B_2 - A_2 B_1} \qquad y_k = \frac{A_1 C_2 - A_2 C_1}{A_1 B_2 - A_2 B_1} \quad (7)$$

Though we evaluated this LIDAR localization system in simulation, integration of the LIDAR proved to be difficult due to several factors. During development, we experienced delays in procuring our LIDARs while we evaluated various models vs budget considerations. One of the bigger challenges we faced was that the nRF52832 only supports one UART connection which is already used by the Kobuki. To remedy this, we planned on using I2C to UART bridges; however, they were severely delayed in shipping. The final solution that we decided to move forward with was to add another nRF52832 which connects to the LIDAR via UART and interfaces with the other nRF52832 over I2C TWIM/TWIS. Using two nRF52832s has the benefit of allocating the LIDAR computations to a single nRF which could reduce computational and memory strain on the other nRF that controls the Kobuki. During integration, we had trouble debugging UART outputs and parsed LIDAR packets. When we used printing over RTT, we would routinely get nonsensical values. A possible explanation could be that since our LIDAR operates at 115200 baud, our print statements could be interrupted and corrupted. Toward the end of development, we were able to measure some accurate LIDAR readings as shown in Figure 10.

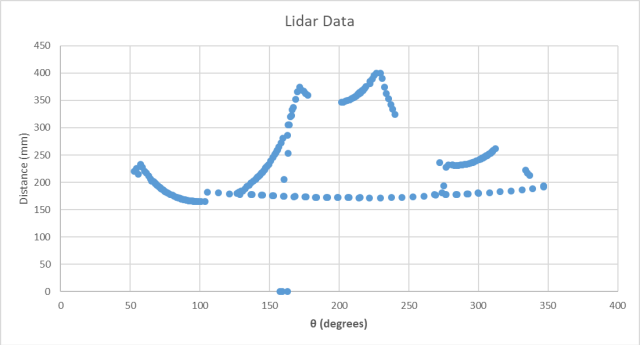

Figure 10. Data from the LIDAR placed inside a box without corrupted prints. The peaks correspond to the walls of the box while the dots in the middle may be wires. The dots at d = 0 could be explained by the LIDAR being too close to the wall of the box.

5 RESULTS AND CONCLUSION

Our final demonstration of the logistics network scenario applied the *gestalt-arch* decentralized control paradigm to a robot network which was then able to accomplish an arbitrary and dynamic series of tasks autonomously. This was done without a central server or handler, and all dynamic pathing decisions such as runtime collision avoidance and subsequent repathing, were made independently by the robots. Notably, this was accomplished through sensor integration, internal state space tracking, and BLE communication alone. That is, while being effectively blind to their surroundings, the robot network accomplished their prescribed tasks in a crowded environment while making little-to-no physical contact with each other.

Furthermore, and most significantly, we were able to demonstrate the capacity for *gestalt-arch* to enable arbitrary loss-tolerance rulesets in the event that a robot was catastrophically dropped from the network. All remaining members of the network accurately recognized the loss of a network member, then applied a retasking procedure to absorb the lost robot's remaining tasks into the functional network.

Finally, despite being unable to complete the final integration of our LIDAR-based localization algorithm, we had positive results when applying the algorithm to Unity-generated LIDAR simulation data.

Video Demonstration:

https://drive.google.com/file/d/1ejuQQTmcGH3xbadZbX4kaOf7KAXtNSFE/view?usp=sharing

6 REFERENCES

[1] Pagliarini, L & Lund, H. *The Future of Robotics Technology*. ICAROB 2017. https://www.researchgate.net/profile/Luigi-Pagliarini/publication/333472283_The_future_of_Robotics_Technology/links/6070536f92851c8a7bb396c4/The-future-of-Robotics-Technology.pdf

[2] Matthews, K. "How Warehouse Robotics Reduce Worker Injuries". EHS Today. Aug. 15, 2019 https://www.ehstoday.com/safety-technology/article/21920298/how-warehouse-robotics-reduce-worker-injuries

[3] *Autonomous Mobile Robot (AMR) overview: Types and use cases*. Intel. (n.d.). Retrieved December 17, 2021, from https://www.intel.com/content/www/us/en/robotics/autonomous-mobile-robots/overview.html

[4] UC Berkeley Lab11. *Lab11/Buckler: Development Board for teaching embedded systems*. (n.d.). GitHub. Retrieved December 18, 2021, from https://github.com/lab11/buckler

[5] *What are Bluetooth Low Energy, gap and GATT?* Contract Engineering, Product Design & Development Company - Cardinal Peak. (2021, November 24). Retrieved December 18, 2021, from https://www.cardinalpeak.com/blog/what-is-ble-and-how-do-its-related-gap-and-gatt-profiles-work

[6] Liu, et. al. *Random Coverage with Guaranteed Connectivity: Joint Scheduling for Wireless Sensor Networks*. IEEE Transactions on Parallel and Distributed Systems. June 2006.