FollowMe: A Robust Framework for the Guidance of Sensorless Indoor Mobile Robots

Sanjith Udupa*, Liangkai Liu†, and Weisong Shi‡
*Novi High School
†Department of Computer Science, Wayne State University
‡Department of Computer and Information Sciences, University of Delaware

***Abstract*—In this paper, we present FollowMe, a new system that allows one sensorless robot to autonomously follow another autonomous robot that has multiple sensors. By offloading both the localization and planning computations to the main robot, we are able to maintain a very small hardware requirement on the follower robot (i.e. it only needs to be able to drive). FollowMe works irrespective of what system it is run on and assumes that the main robot has some method of localizing itself and the follower robot, as well as software for navigation (driving itself to a target point). Given this, we can run FollowMe on different kinds of robots, as we have done through our tests on both physical and simulated robots.**

The FollowMe system is comprised of three main components. The first is the *State Machine* that takes input from arbitrary localization sources and coordinates through the *PathSplitter algorithm* to dynamically segment a given path into sequential target positions for each robot. Then, the *Evaluator* adjusts parameters for both following and path splitting depending on the following performance. As such, FollowMe only handles the queuing of new target points given a predefined path and assumes the master robot can handle the actual driving of each robot (follower and main) to the points. In order to make this possible, the *PathSplitter* algorithm ensures the follower robot is in view of the main robot at all times so accurate localization can occur. Finally, the state machine has recovery states as needed.

After running experiments in both a physical and a simulation environment, we determined that FollowMe is effective at accomplishing the task of guiding both robots along a predefined path accurately, but because of the iterative nature of the following process, it is relatively slow. Our results highlight the potential for using existing autonomous driving technology in robot navigation and suggest promising directions for future research in this area, specifically for use in autonomous wheelchairs or in the warehouse industry.

***Index Terms*—autonomous mobile robots, sensorless robots**

I. Introduction

A handful of techniques already exist for the task of following humans [1], objects [2], and other robots (not technically following - just coordinated motion) [3]. Current approaches to robot following exhibit control over only one robot as the other element (what is being followed) are typically not able to be controlled by the robot. Think of a human-following robot, for example, the software on the robot simply cannot adjust the motion of the human. Furthermore, by controlling only one robot, we necessitate having all the facilities for autonomous control existing on only one robot. In general, we have two robots that are both autonomously controlled and therefore can adjust the motion of both as needed. In addition, only one of the robots needs autonomous sensing capabilities so long as it can also localize the other robot. This results in an interesting situation where we have one, relatively standard, autonomous mobile robot and another very barebones robot that together are able to make complex autonomous maneuvers.

In order to maintain a manageable but still broad scope, one of the constraints for this project that we have established was the fact that FollowMe by itself would not be able to perform autonomous driving. At its core, FollowMe is more of a middleware than a specific navigation framework. In order to use it, one must independently perform the localization of both the main and follower robots. This is relatively trivial with the variety of available technologies (as described in Section IV, we used SLAM [4] and computer vision with AprilTags [5] to perform both of these tasks) but it allows users of FollowMe to decide for themselves how this localization will be done. Similarly, FollowMe requires the user to supply at least two instances (one for each robot) of some navigation system capable of routing a robot to a specific goal position. All of these processes will run on the main robot.

Therefore, we define the *robot following problem* with the following four constraints:

- There must be at least one ”main” robot and one ”follower” robot
- The follower robot(s) may not possess/read from any sensors to aid in localization and navigation (other than data logging for evaluation)
- The main robot must have a means of locating the follower robot through some onboard device (i.e. a camera)
- The follower robot must not perform any navigation calculations itself. All processing (except converting a generic velocity command to wheel speeds) must be done on the main robot

To solve the given problem, we have developed the FollowMe system, which is made of three components. At the core is the *state machine coordinator*, which allows the robots to follow a path in a sequential manner, with only one robot moving at a time. Given a full path, this sequence of motions is dynamically determined by the *PathSplitter algorithm*, which breaks the full path into segments that can be tackled one by one. The segments themselves are determined using an iterative approach that ensures the follower robot remains in view of the main robot throughout the entire duration of each

robot's driving. Because of the fact that we do not know exactly how well each robot will follow the path, at the end of each segment an *evaluation state* is reached in which the parameters for the PathSplitter are recalculated with the goal of maintaining a steady following distance. This makes FollowMe an end-to-end system that can accomplish complex maneuvers carefully.

Beyond this application, the concepts can be applied to a more generic question: how can the data collected by sensors on one robot be utilized by another, nearby robot? Say we design a system where one robot can be equipped with all the necessary sensors for complete autonomous control. More than likely, that data can be usable by another robot that doesn't have extra sensors on it. The implications of this could be low-cost robotic fleet management: one mobile sensing unit can inform multiple "dummy" robots. Similarly, this technique can be potentially useful in the future of semi-autonomous motorized wheelchairs. Users of such wheelchairs who are visually impaired or simply wish to be guided can let a more advanced robot guide them while keeping their actual wheelchair itself simple and cheaper. The same can be said about potential search-and-rescue operations. While these reach goals are purely hypothetical, we aimed to design an adaptable framework for this two-sided control of robots with this project where one robot can be equipped with multiple sensors and the other can have virtually none, and the past motion of each robot informs the future motion of each.

In summary, this paper makes the following three contributions:

- Define the robot following problem and observe two technical challenges.
- We propose a modular *FollowMe* system which consists of three components: the state machine coordinator, the PathSplitter, and the evaluation state. This makes FollowMe an end-to-end system that can accomplish complex maneuvers. We have open-sourced the code use for writing and testing FollowMe in this research, including the whole ROS project to control both the physical and simulated robots in Python. [1]
- We evaluate the proposed system in both simulated and real-world environments. The FollowMe system shows stable following performance with low system overhead.

The rest of the paper is organized as follows. Section II presents the observations of technical challenges in *robot following problem*. Section III presents the proposed *FollowMe* System. Section IV presents the implementation of the *FollowMe*, while Section V presents the evaluation . Section VI describes the related work and applications of the research. Section VII concludes the paper.

II. Observations

Due to mechanical issues in the physical robots' designs, we found odometry based on the wheel encoders to be unreliable [6], [7]. As a result, we opted to use HECTOR SLAM'S [8] ScanMatcher utility to do the localization solely based on LIDAR data. With this in mind, the FollowMe system only requires *some* form of localization, which is left up to the implementation. In most cases, odometry will work fine, but for our specific circumstance, this observation allowed us to make the system broader and fit within our goal (Section III).

Another source of issues that we noticed was communication delays. To accommodate for this, the behavior of FollowMe can be modified via several constants. Specifically, `K_SPEED_ADJUST` and `DEFAULT_MAX_LOOKAHEAD` directly influence how the main robot moves ahead relative to the follower robot, therefore being able to compensate for delays. We ran a set of tests to measure the actual network latency on the physical robots (communication via a shared local WiFi network) as a means of determining a good starting point for the aforementioned constants both in the simulated and physical setting.

III. FollowMe Design

A. Overview

FollowMe has been designed to be as modular and platform-independent as possible. That is, with minimal changes, the same FollowMe software should be able to coordinate different pairs of follower and guidance robots. This is reflected by the decisions made in our system design. It is partly inspired by many traditional human following techniques for Autonomous Mobile Robots like [1] and [9]. In these, there is one robot that aims to follow a human by running some form of closed-loop control across a target distance (between the human and the robot) and the speed at that the robot is driving. On top of that, an additional steering controller is employed to keep the robot driving in the correct direction. While these systems are technically autonomous, for all intents and purposes, they are not following paths. Instead, they are simply reacting to real-time changes in the human's motion, as that is the only information the robots are given.

For our application, however, we are in full control of both robots and the main robot (which is guiding the follower robot) is operating autonomously. If a target path is given to the main robot, FollowMe's goal is to use that path to create target checkpoints for both robots sequentially along the given path. We operate on the assumption that the main robot is capable of autonomously driving itself to a target location, and given a constant stream of localization input, it can calculate the driving velocities for the follower robot to do the same. By doing so, the problem is simplified significantly. Now, the goal of FollowMe is to create several new paths for the main robot and the follower robot to each follow and to ensure that the main robot is able to constantly "see" (depending on which method is being used to localize the follower robot) the follower robot.

Because of the unpredictable nature of how the follower robot may act given a specific command, the sequence of paths for the main robot path must be dynamically modified

[1]The code for FollowMe is available at https://github.com/Torreskai0722/FollowMe.

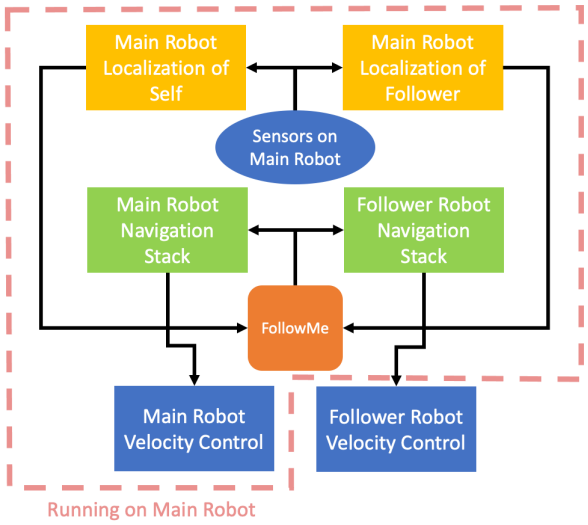

Fig. 1. Overview of the FollowMe system.

to do this. Because of these constraints, we decided that a state machine node will coordinate the localization and navigation goals of both robots. The method of localizing and navigating to the aforementioned goals is depended on the implementation, and as such the system only assumes that it can get such data, not how.

The overview of FollowMe is shown in Fig. 1. It is up to the implementation to localize each robot and to have an open function that can autonomously drive itself to a specific target point. FollowMe is the process that is sitting in the middle that calculates those target points and therefore never has to directly interface with hardware, making it very modular.

B. Methodology

When FollowMe is invoked, it requires the following pieces of data: the desired path for the main robot to travel (with the intention that the follower robot will end up close behind) and then a constant stream of localization data for both robots in world space. It also requires access to functions to send a target point for each robot to drive to. The system which calls these functions is comprised of three components. The first is the **State Machine Coordinator** which handles the automation of each robot and allows for the sequential following nature of FollowMe. In order for the segments of the path to be determined, the **PathSplitter Algorithm** is used to calculate target points for each robot. It uses a technique called FOV Matching to ensure that the follower robot is in view and thus able to be localized continuously during the path. When a new segment is determined, the coordinator sends the following commands to each robot one by one and then enters the **Evaluation State** in which the PathSplitter's parameters are adjusted to best hold a steady following distance. Each component is described in detail in what follows.

1) State Machine Coordinator: Fig. 2 shows the full State Diagram for the FollowMe system including the transitions between the states. FollowMe starts idle, waiting for it to receive the full path. Once it is received, the PathSplitter is used to determine the first segment for the main robot, and the main robot is commanded to that point. Once it is reached, the follower robot is commanded to the previous position of the main robot. After this, the evaluation state is entered in which the PathSplitter parameters are adjusted and then used again to determine the next segment. This sequential process continues until the main robot reaches its goal position. If the evaluation state determines that the progress is beyond the point of return/unable to be fixed, the recovery state is resorted to. The recovery state is equivalent to the **naive solution** which is outlined in the next section about the PathSplitter algorithm.

2) The PathSplitter Algorithm: FollowMe takes in an initial path to an end goal that the Main robot's navigation layer calculates. Before allowing the robot to follow this path, FollowMe sequentially splits up the path at various intervals and commands the main robot to one location and the follower robot to the previous location of the master robot. While doing so, FollowMe ensures that the Follower Robot will remain in the FOV of the Main robot so that continuous localization can occur because the Follower robot's location is only known when it is seen by the main robot's sensors (be it a camera or anything else). After each iteration, FollowMe evaluates how well the follower maintained a `PREFERRED_FOLLOWING_DISTANCE` and adjusts the following gains accordingly. This results in an adaptable solution to the following problem. If despite these adjustments, the robots remain underperforming, a recovery mode is activated in which the robots resort to the following "naive solution."

The Naive Solution: We began by experimenting with the most intuitive approach: allow the main robot to calculate a path for itself and have the follower robot copy the same velocity commands that are calculated for the main robot. This is accomplished quite easily through the form of subscribing to the same Twist2D ROS Topic (this is obviously dependent on the implementation, but the former option is consistent with what is described in Section IV).

In practice, this method had several issues. Initially, we thought there would be inconsistencies between movements of the two robots. However, because the dimensions and wheelbases were roughly the same, and because the motors were running accurate PID Loops, the robots were more or less equivalent in terms of how they executed a given velocity command. The actual issues actually came in the form of a fundamental issue with this method. Even if the follower robot was able to perfectly replicate the behavior of the main robot, if the two robots don't start in the same position they will not follow the same path.

Throughout the testing of the naive solution, when we varied the starting locations even slightly, we would have issues with either a) the main robot losing sight of the follower robot, or b) the follower robot executing the same path at an odd angle, ultimately ending in the wrong place.

Fig. 3 demonstrates both of these issues in ideal situations. That is, with the assumption that the follower robot is in view

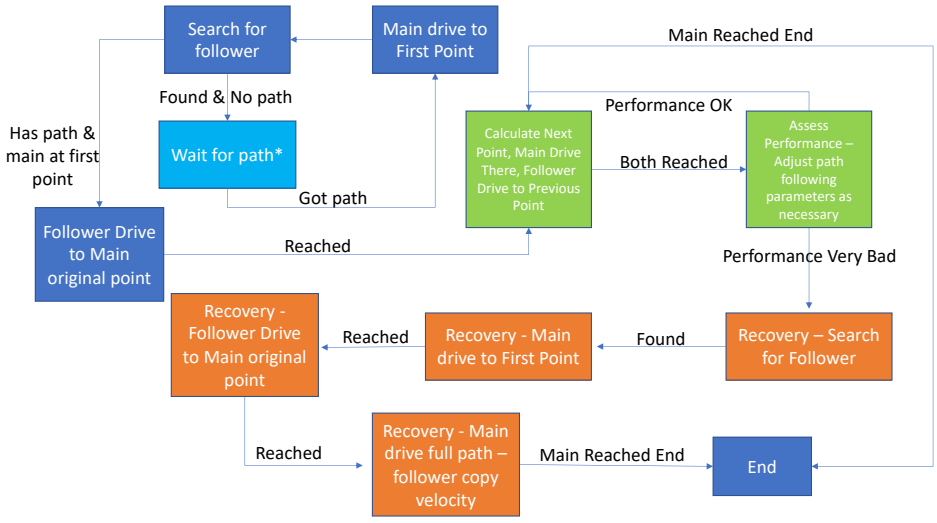

Fig. 2. State Diagram for FollowMe. The * denotes the initial state.

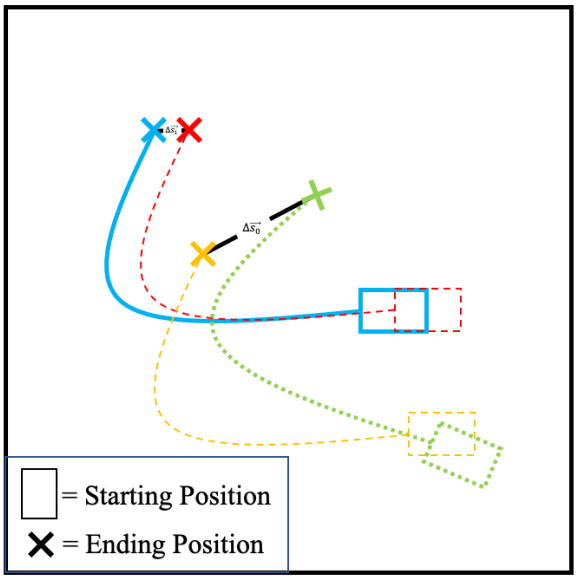

Fig. 3. Two examples of master/follower robot pairs that demonstrate the variability in ending position presented by minor changes in starting position by the naive solution

of the main robot's cameras at all times and the follower robot can perfectly replicate any action the main robot does (both of which are unrealistic assumptions). When the follower robot begins at a different pose, perhaps at a slight angular offset, than the main robot (a very reasonable assumption), the endpoints of each path are vastly different. Both of these two outcomes are demonstrated by the Δx displacement values.

In fact, in the figure, the final distance between the follower and main robots is nearly double that of before the motion. As a result, some more sophisticated logic is necessary to coordinate the motion.

Those hypothetical examples demonstrate that the naive solution does work in those select situations when the robots start near each other. Instead, what if we first move the follower and master robot to be next to each other, and then just follow the path naively?

By this same logic, we don't need to use the naive solution at all. If we just sequentially move the robots to positions along the path where they are near each other, then we could have the most accurate following process. We can choose a point along the path of the main robot, drive it there, and then command the follower robot to the original starting pose of the main robot. The local routing of each robot would use the navigation function based on the implementation.

Once again, though, this has some issues in practice. Namely, almost all navigation stacks assume that a constant stream of localization data will be available. While the navigation stack for the main robot will have this (either via SLAM [4], odometry, or any other method as set up by the specific implementation), the navigation stack for the follower robot may not. This is because when the main robot will be moving we can't ensure that the follower robot will remain in view of the main robot throughout the duration of the path.

To solve this problem, we have the first element of the PathSplitting algorithm: **FOV Matching**. This is visualized in Fig. 4. You can see the application of the two main functions in the PathSplitter: `angle_to_midpoint` and `fit_in_fov`. Used together, these functions return a new point for the master robot to target where the entire path of the follower robot will remain in the FOV. First, we seek a certain "Lookahead Distance" into the path, based on the acceptable distance between the main and follower robots for accurate localization. Then, we calculate the angle that the robot would need to face at that point if the sensor were to point directly

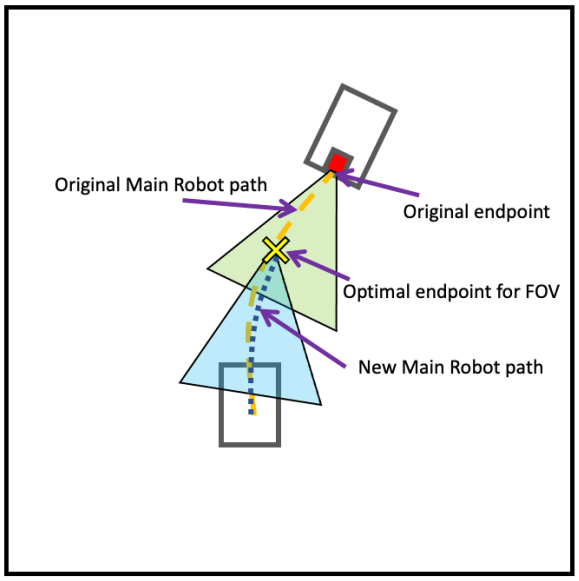

Fig. 4. Visualization of the FOV Matching Routine

at the midpoint between the main robot's original pose and the follower robot's original (also current) pose. This means that the main robot would end up in a position where it can equally see where the follower robot starts and where it aims to end up (the starting position of the main robot). We can do a check to ensure that neither the main robot nor the follower robot will be out of the FOV (both will be at equal angles from the center of the sensing zone) and if it is not, we will recursively search again with a larger lookahead distance.

This approach allows us to systematically find the shortest viable distance that the master robot can go ahead into the path where the follower robot can be tracked throughout its entire journey to the main robot's original position.

After commanding each robot to its next points, we arrive at the final component of FollowMe: the evaluation state.

3) Evaluation State: Throughout the duration of the path, the absolute distance between the main robot and the follower robot's poses is measured. A constant `PREFERRED_FOLLOWING_DISTANCE` is defined before running FollowMe. In the evaluation State, the percent error between the true average distance and the target distance is calculated, and the `FOLLOWING_SPEED` (a parameter for the actual navigation node - not for FollowMe) and `LOOKAHEAD_DISTANCE` parameters for the PathSplitting function are linearly scaled. This results in self-correcting motion as the path continues to be followed.

IV. Implementation

We implemented the FollowMe system on two systems to test its usability. One was a set of physical robots running ROS and another was a simulated environment via the Gazebo Simulator for ROS.

Component	Main Robot	Follower Robot
Computing Device	NVIDIA Jetson Xavier NX	Raspberry Pi 4
OS & Software Version	Ubuntu 18.0.4 - ROS Melodic	Ubuntu 18.0.4 - ROS Melodic
Drivetrain	Dual DC Motor FWD	Dual DC Motor FWD
Localization of Self	RPLidar S1	–
Localization of Other	Generic USB Camera	–

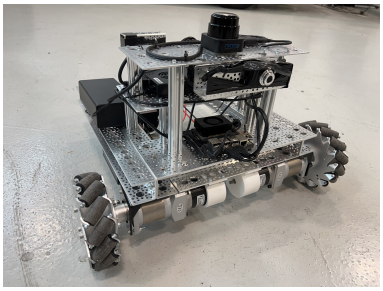

Fig. 5. Front of physical Main Robot

A. Physical Testing

For this project, we built from scratch two modified versions of the HydraOne Computing Platform [6], one for the Main robot and one for the Follower robot. Each is a similar differential drive robot. The following is an overview of the differences between the robots.

1) Main Robot: The main robot is very similar to the original HydraOne platform, apart from a few minor differences. It is running a Jetson Xavier NX on Ubuntu 18.0.4 with Robot Operating System(ROS) [10] Melodic. Installed on it are an RPLidar S1 [11], two USB webcams (oriented 180 degrees apart), and 2 brushless DC motors and powered by Chip Robotics motor controllers [12]. The front two wheels are direct-driven Mecanum wheels and the back wheel is a free caster wheel. See Fig. 5 and Fig. 6.

For the software, we have written our own differential drive controller which takes a ROS `Twist` message and commands the Chip controllers library [12]. We implement Hector Mapping [8] using the RPLidar ROS SDK for localization, which is fed into `move_base` [13] as the navigation layer for the main robot. We run a duplicate navigation layer for the follower robot on the main robot computer ensuring that the only processes running on the follower only pertain to the individual motor controls of that respective robot.

The computing device on the main robot is an NVIDIA Jetson Xavier NX Developer Kit running Ubuntu 18.0.4 and ROS Melodic.

2) Follower Robot: The follower robot is similar to the main robot but simply without the sensors. Here, the drivebase is identical, with two brushless DC motors powered by the same Chip Robotics controllers [12] in the front with Mecanum wheels attached and a free-spinning back caster wheel. This robot does not have any of the external sensors that the main robot has, but we have attached 4 AprilTags [5] to each of the 4 sides (90 degrees apart) to the robot. See Fig. 7.

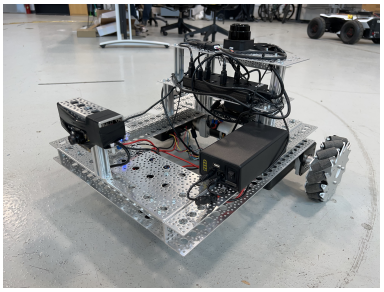

Fig. 6. Back of physical Main Robot

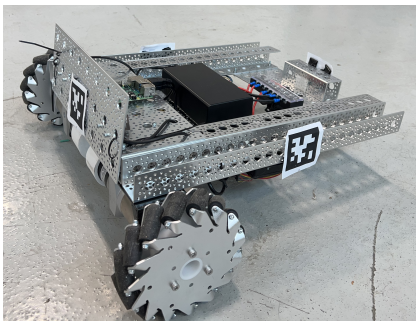

Fig. 7. One side of physical Follower Robot

The computing device is a Raspberry Pi 4. The software of the follower robot has the same differential drive controller as the main robot and nothing else. For networking, we used ROS over a local WiFi network. The master node was run on the Main robot.

In order to localize the follower robot, the main robot is running the ROS AprilTag node, which publishes a ROS Transform (TF) [14] transformation between the camera and the detected tag (see Fig. 9), is used in conjunction with our TF Tree (see Fig. 10) which establishes static transforms between each of the tags and the center of the follower robot as well as between the camera and the main robot to accurately determine where the follower robot is. Fig. 11 demonstrates this result with `base_link` as the center of the main robot and `follower_link` as the center of the follower robot. Note that we have two cameras on the main robot but a single `camera_switcher.py` script we wrote handles the switching of both TF [14] transforms and topics for the AprilTag detection system.

Combined with the localization data given from the Main robot's SLAM [4] process, we get a result with a map and two localized robots like that of Fig. 12. Note that every part of this localization code for both robots is running only on the master robot's computer.

The computer on the follower robot is a Raspberry Pi 4B running Ubuntu 18.0.4 and ROS Melodic.

3) Course/Setup: We set up a course as seen in figure 5 for testing the FollowMe system on our physical robots. This was assembled as an arbitrary arrangement of cardboard boxes to simulate a simple bounded region with protrusions to make the path-planning process slightly rigorous.

Fig. 8. Physical course setup with Follower and Main robots

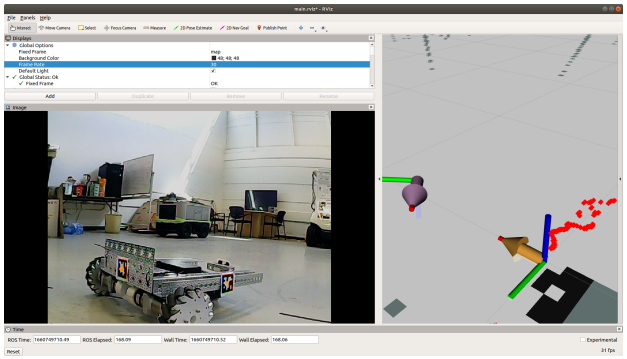

Fig. 9. AprilTag detection Transform output

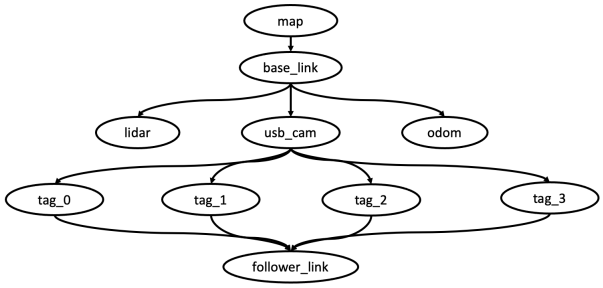

Fig. 10. Transform Hierarchy for Both Physical and Simulated Robots

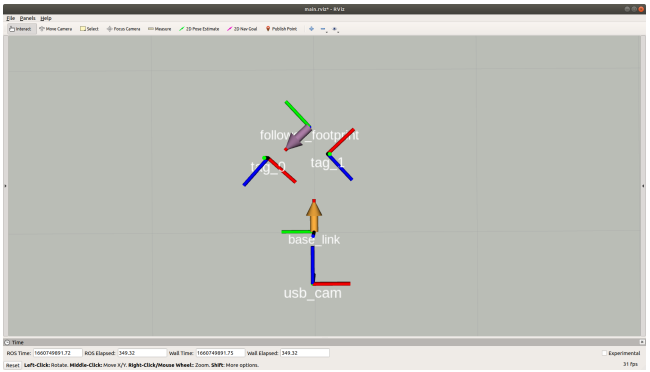

Fig. 11. Localization of Follower Robot via AprilTags

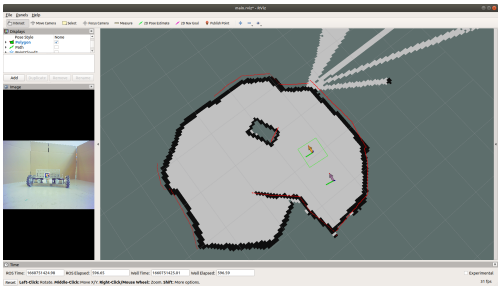

Fig. 12. Full localization of both Follower and Main Robots

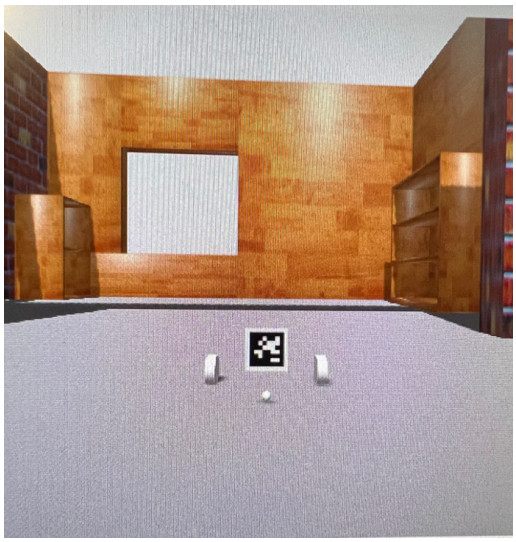

Fig. 13. Follower Robot through the view of the camera on the back of the Main Robot

B. Simulation Testing

The goal of our simulation environment was to mimic as accurately as possible the real environment that was created in the lab. We did this by running a combination of ROS melodic with the Gazebo [15] ROS simulation environment which handled physics and rendering. Because this was still using ROS, most of the software written for the physical robots could be reused. This is in part due to FollowMe's modular nature, but also because we designed the simulation to be essentially a recreation of the robots built in the lab.

We built two robots in the simulation using two identical differential drive models and chose one to be the main robot and the other to be the follower. The main robot had a simulated LIDAR placed atop it and a back-facing camera at the rear. Fig. 13 shows what the view from that camera looks like, showing the Follower Robot.

The second robot had four AprilTag models (with 0 mass) attached rigidly to the sides of the robot in the same fashion as the physical follower robot.

The course we used was the sample TurtleBot [16] "House" world although nothing else in the simulation environment was from the TurtleBot ecosystem. This allowed us to have a relatively complex environment for the mapping to happen in. Fig.

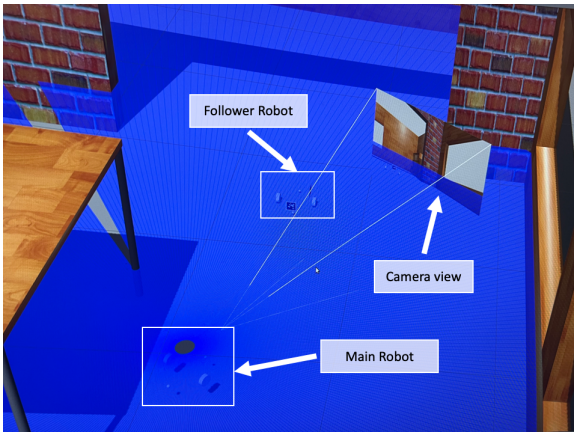

Fig. 14. The Main and Follower robots together in the Gazebo environment

– shows both robots together in the simulation environment and the localization of both robots being performed.

Nearly everything else was identical to the physical robot setup, with two instances of the ros_navigation/move_base node running, one for each robot, and hector_slam performing the mapping.

V. Evaluation

For evaluating FollowMe, we established the following metrics: Path Completion, Distance Between Robots, and System Utilization. Together, these test the usefulness and efficiency of the framework.

A. Path Completion

Path Completion is straightforward - just a qualitative way of measuring how well robots using FollowMe completed their paths. As described previously, the first (physical robot) implementation (described in Section IV) struggled with progressing through the typical following states and thus had to resort to the "recovery mode" each time. This did tell us, however, that the sensitivity of when to transition to the recovery mode was adequate and the recovery mode (see Section III - Naive Solution) worked as expected. We determined that this was an issue with the physical constraints of the robot and therefore the same configuration worked quite well when simulated.

In the simulation environment (also described in Section IV), we were more successful in replicating the preferred behavior of the FollowMe state machine. Each state progressed in the expected fashion and the robots were successful in following each other across multiple different paths. Fig. 15 is an example of a simple path being followed successfully.

B. Distance Between Robots

In order to quantitatively measure the performance of FollowMe, we track the distance between the follower and main robot poses as a function of time. In the graph in Fig. 16 it is shown that through nearly the full following process the distance between the robots stays very steady (at around 0.55 meters). Only near the end of the path do the robots

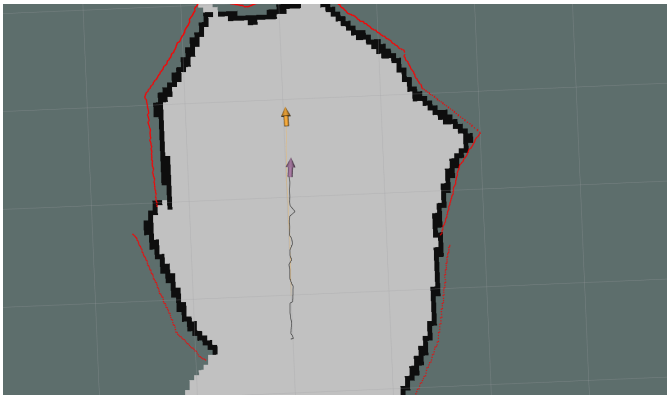

Fig. 15. Result of a Basic Following Path

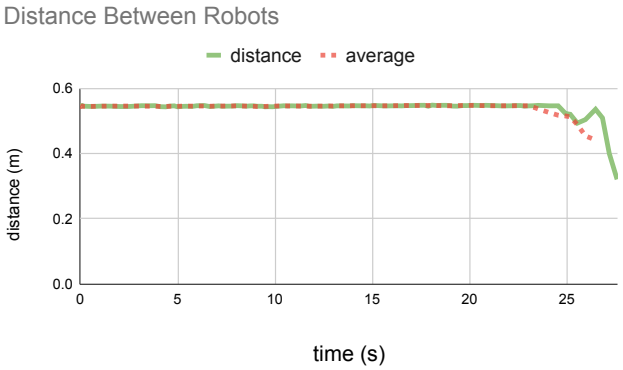

Fig. 16. Plot of the distance between robots in meters vs time in seconds into the FollowMe process. Various measures are included to demonstrate the relative stability of the distance measurement.

come closer together and this is likely a result of the main robot reaching its target but the follower robot still has a part of its path remaining. This is reflected by the distance only decreasing and not increasing past the 0.55 meters point after the main robot stopped. This data was recorded on the physical robots performing a path similar to the one in Fig. 12.

C. System Utilization

System Utilization measures both the average CPU usage and the memory usage of the *FollowMe* process during simulation. These metrics were measured using the psutil [17]. The baseline was the Gazebo simulation running by itself along with all the necessary components for robot function and navigation (i.e. two navigation layers, SLAM, odometer, etc). We then measured the usage during an instance of running the *FollowMe* scripts on top of the baseline. The usage vs time plot is in Fig. 17. From the graph, we have concluded that *FollowMe* is reasonably performant but for the sake of scaling, it may be necessary for further performance improvements to be made. Generally, the program only used around 10-20% of the resources available, which considering the fact that *FollowMe* cannot be run alone (it needs external navigation nodes to be running) we can conclude that we want

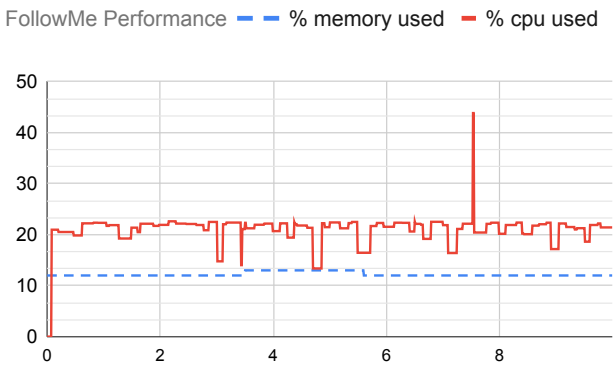

Fig. 17. Plot of CPU and memory usage percentages in a full FollowMe Routine.

to minimize the usage even further. A likely cause for the spike seen at around 7.5 seconds is a repeated failed lookup with the *PathSplitter* as the path curved at that point and the function had to be recursively called multiple times.

D. Evaluation Summary and Limitations

In summary, *FollowMe* proved to be effective at completing the original goal of effectively guiding the follower robot along the path of the master robot. Similarly, the method of updating the path following parameters proved effective to make the following distance be consistent throughout the following process. Although the CPU utilization was relatively high at times, it is important to note that the project was defined with the assumption that the main robot will have a reasonably capable controller. This is due in part to the fact that it will have to run multiple navigation instances, one for itself and for the follower. Because of this, we have determined the performance to be acceptable for our tests. This tradeoff is revisited in the following two sections.

VI. Related Work and Applications

A. Related Works

The problem we have defined, where one robot outfitted with multiple sensors guides another robot with no sensors, does not have any direct parallels with existing research that we could find. There do exist a handful of approaches to similar problems that can be grouped into two types. While neither is exactly like FollowMe's problem, elements of our final system can be found in current techniques for both forms of the following problem.

One form is where there is only *one* robot in the entire system that acts as the "follower" which is set to follow one external object which it has no control over. Examples of this include methods of following people ([1] and [9]) and, more generically, objects [2]. The techniques in these papers mainly focus on the central idea of maintaining a target distance between the robot and the target object, whatever that object may be. The means for measuring this distance are abstracted but generally, a PID [18] controller is employed to

maintain that distance. Another is used to maintain a steering angle. FollowMe's `PREFERRED_FOLLOWING_DISTANCE` functionality in the **Evaluation State** is similar to this.

The other similar problem is where there are multiple robots but they aren't necessarily following each other but moving in a coordinated fashion: swarm movement in other words. Critically, each robot is outfitted with sensors, whereas the key goal of FollowMe is for one of the robots to have none. A prime example is research on the coordination of spacecraft formations. [3]. In this case, each is essentially following its own path with its own navigation logic. The coordinated nature comes in the fact that when an obstruction or unexpected occurrence appears, the main details of the path (i.e. waypoints) are all decided by one of the robots, and the rest (i.e. the intermediate points along the new trajectories) is recalculated by each ship's individual navigation system. This is somewhat how the PathSplitter algorithm in FollowMe works, in which the general details (new target for each robot) are calculated by the main robot but the navigation firmware for the follower robot is *also* being run on the main robot with FollowMe.

Existing research on similar problems has aided in the development of FollowMe, but the problem our novel system seeks to solve is itself a novel one. The following applications for it are reasoning of why this problem is relevant and important for mobility.

B. Applications

The motivation behind this project was the potential for real-world applications of this system in Mobility for Wheelchairs as well as Industry scenarios.

1) Mobility for Wheelchairs: Originally, the goal of the FollowMe Framework was for use to assist those with electric wheelchairs both in low and high-stakes scenarios. However, due to the nature of the relatively slow iterative following process, low-stakes scenarios seem to be the most productive and safe use case.

A low-stakes scenario could be one where someone in a cost-effective wheelchair, one that does not have complex sensors or powerful computers, could employ high-quality autonomous navigation software using a guidance robot. Trivial tasks like guiding a visually impaired person through a park can be accomplished through this.

Electric wheelchairs are actually quite commonplace in today's society. A quick search on ElectricWheelchairsUSA for options of such devices found [19], [20], and [21] with prices of $4209, $3699, $6549, respectively. Autonomous wheelchairs are not yet commercially available, although researchers at MIT's CSAIL [22] did create one that, while worked, was comprised of several components whose costs far exceeded those of existing electric wheelchairs.

A hypothetical high-stakes scenario could be one of search-and-rescue. Imagine a situation in which there is a person inside a burning building in an electric wheelchair. This wheelchair is equipped with controls to drive and turn but has no way of performing its own perception. The processor on the wheelchair similarly is not necessarily capable of performing the autonomous tasks to escape the building. Instead, we can use FollowMe, which only requires a method of communication with the wheelchair. A robot with FollowMe is sent into the building and is able to perceive its own surroundings and adaptively guide the wheelchair to safety, all while firefighters can be focused on fighting the fire and helping other victims.

2) Industry: In a warehouse setting, FollowMe could be implemented for autonomous fleet management The experiments in this paper have proven that it is indeed possible to guide *at least* one robot through localization from another through FollowMe with minor additional CPU overhead. Given this, we can reasonably assume that FollowMe is scalable enough such that we could use it for more complex navigation tasks with more than one follower robot. Because the implementation is not dependent on what method of navigation is being used, the path-following approach is not necessary, and instead, FollowMe's localization methods could be used

VII. Conclusion

In conclusion, the FollowMe project met the goals we established for it. Our framework is lightweight yet effective at following complex paths on multiple robots. It is capable of maintaining a stable following distance on those complex paths. While there is room for improvement in terms of performance, both speed and actual computational efficiency, the concept behind FollowMe and the software in its current state both have very real applications for mobility.

References

[1] M. Bakar and M. F. Mohamad Amran, "A study on techniques of person following robot," *International Journal of Computer Applications*, vol. 125, pp. 27–30, 09 2015.

[2] A. Al-Omary, "Design and implementation of object seeking robot," *International Journal of Computer and Electrical Engineering*, pp. 816–820, 01 2012.

[3] W. Ren and R. Beard, "Formation feedback control for multiple spacecraft via virtual structures," *Control Theory and Applications, IEE Proceedings* -, vol. 151, pp. 357 – 368, 06 2004.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[6] Y. Wang, L. Liu, X. Zhang, and W. Shi, "{HydraOne}: An indoor experimental research and education platform for {CAVs}," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[7] L. Liu, J. Chen, M. Brocanelli, and W. Shi, "E2m: an energy-efficient middleware for computer vision applications on autonomous mobile robots," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 59–73.

[8] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[9] S. Hassan, M. Khan, and A. Khan, "Design and development of human following robot," 07 2015.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[11] " RPLidar S1 ," https://www.slamtec.com/en/Lidar/S1.

[12] " Chip Robotics BLDC Motor Controller ," https://chiprobotics.gitbook.io/chip-robotics/bldc-motor-controller/single-brushless-dc-motor-controller.
[13] " ROS Navigation Package ," http://wiki.ros.org/navigation.
[14] " ROS Transform Library ," http://wiki.ros.org/tf.
[15] 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, 2004, pp. 2149–2154 vol.3.
[16] R. Amsters and P. Slaets, *Turtlebot 3 as a Robotics Education Platform*, 01 2020, pp. 170–181.
[17] " PSUtil Python Library ," https://github.com/giampaolo/psutil.
[18] R. Paz, "The design of the pid controller," 01 2001.
[19] "Drive Medical Cirrus Plus EC Folding Power Electric Wheelchair," https://www.electricwheelchairsusa.com/products/drive-medical-cirrus-plus-ec-folding-power-electric-wheelchair?variant=6833674682403.
[20] "Drive Medical Titan AXS Mid-Wheel Drive Electric Wheelchair," https://www.electricwheelchairsusa.com/products/drive-medical-titan-axs-mid-wheel-drive-electric-wheelchair.
[21] "Merits Health P312 FWD/RWD Dualer Power Chair," https://www.electricwheelchairsusa.com/products/merits-p312-fwd-rwd-dual-power-chair.
[22] " Autonomous Wheelchair - MIT CSAIL ," https://www.csail.mit.edu/node/5962.