Distributed Indoor Mapping

Haokun Chen
Computer Science
t1k0b
chenmkan777@gmail.com

James Park
Engineering Physics
e2d9
jpark1273@gmail.com

Beichen Zhang
Engineering Physics
j2y8
zhang.beichen@dhs.sg

Howard Zhou
Computer Engineering
q6c9
dvhowardzhou@gmail.com

Divya Ratnasami
Physics & Computer Science
w1v8
divyarat27@gmail.com

ABSTRACT

The goal of this project is to build a distributed system in which peers simultaneously work on mapping out an area. Each node is a mobile robot that consists of a Raspberry Pi 3B, controller(keyboard), and LCD screen. All nodes will communicate their findings with their fellow robots by exchanging their maps only when they come within a certain range of each other. Upon exchange, they will resolve any conflicts between it's local map and the ones received. Then they will coordinate how to explore the undiscovered area. The robots will stop surveying the area when their energy level goes below a certain value, this will resemble real life situations as robots expend energy as they do work.

KEYWORDS

Distributed System, Raspberry Pi, Azure, Wi-Fi Ad hoc, P2P

ACM Reference Format:
Haokun Chen, James Park, Beichen Zhang, Howard Zhou, and Divya Ratnasami. 2018. Distributed Indoor Mapping. In *Proceedings of University of British Columbia.* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The objective of this project is to build a distributed system in which the peers will work together to map out the area they are in. Each node will communicate its finding with its neighbors only when they come within a certain range of other robots. The robots will exchange their maps, resolve any conflicts between these maps and its local one, and coordinate how to explore the undiscovered areas. The robots will stop surveying the area when their energy level goes below a certain value, this will resemble real life situation because a robot would eventually stop.

University of British Columbia, April 2018, Vancouver, British Columbia Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 DESIGN

2.1 High Level Overview

The aim of this project is to implement a distributed mapping system where Raspberry Pi 3B robots (nodes) work together to map out a certain enclosed region. The topology will be time-dependent, that means robots will only be connected if they are within a certain radius of each other. Since the nodes are mobile, this would lead to multiple, different networks simultaneously. Each node will be mapping the area by following a path which will be an array of directions. Trying to emulate actual robots on a mission we introduce the notion of energy that will represent the battery level of the robot. This value will be represented numerically and vary inversely to the steps taken, that is the more steps a robot takes the lower the robots remaining energy will be. The robot will be initialized with an unique ID and its starting coordinate by the user. The user will use a consistent coordinate system therefore each robot will operate in the same coordinate system.

At the beginning of each journey it will start off with some initial energy, an empty map and no current path to explore. Each robot will generate a point to go to called destination point via the task allocation algorithm (explained in detail below) and create a route to reach this point. During its exploration, it will log each coordinate it traversed, the time it traversed, whether it is a wall or free space, and its current energy. Simultaneously, the robot will be broadcasting its IP and position. If during its journey it observes other robots it will follow the introduction protocol; that is, it will stop its motion, connect with these robots only if it is within its communication radius. Then it will send its map to them and also received the maps of its fellow robots.

Upon receiving the maps the robot will merge its local map with each map given by its fellow robots. It will check if there are any conflicts and resolve them. For example, if Robot A recorded that there is a wall at coordinate (2, 7) but Robot B recorded that there was not a wall there, here the latest recording will take precedence as our assumption is that the environment is dynamic. Upon map exchange each robot will compute the next tasks for all the robots via the task creation algorithm (explained in detail below). After this they will follow the task assignment protocol. From these newly computed tasks it will assign tasks for itself and its fellow robots. Each robot can either refuse or accept the task assigned to them. Only after they get a response, they will start to execute the assigned tasks. This eliminates the dependence of one node to assign tasks to the others in the network and is more robust to robot failures during this stage.

Upon task assignment the robots will continue exploring but now with this new task. Since the robots at this time will still be within the communication radius a time-communication threshold is introduced. This prevents robots who were just communicating from connecting so that they can start their new task without unnecessarily repeating the introduction protocol. There will be a timer that will start following task assignment, and if the value of this timer is below the time-communication threshold, the robots cannot communicate even if they are within the communication radius.

Emulating a real-life situation the robots will eventually stop exploring when their energy goes below a certain value. However, if the user prefers to have the robot keep exploring, it can be configured in the software control by setting a very high initial energy or letting it be independent of steps.

A robot can fail at any point during its journey and our system will be equipped to deal with these. All of the robots necessary information about it's state will be logged onto disk, such as the robot's local map, current task, path remaining to explore, position, energy, and state. So if the robot re-connects it will be able to pick up its journey where it left off by consulting its log. If the log is empty it will know that it is just starting its journey.

2.2 Assumptions and Invariants

The following bullet points are the assumptions/invariants:

- The area will be finite, planar (2D), and objects inside may move around.
- Robot IDs, communication radius, and time communication threshold are set by the user.
- There are no malicious robots.

2.3 Human-Robot Interaction

The robots movement will be facilitated by a human operator. The robot will consist of a Raspberry Pi 3, LCD screen, keyboard, and human operator. Figure 1 displays the schematic diagram of the Raspberry Pi and LCD circuit. The LCD screen displays the direction (North, South, East, West) to go in. Four keys on the keyboard will indicate whether the next step is either a wall or not, and if there is a wall to the right or left. The operator will press the corresponding key. The entire setup is shown in Figure 2. The size of each step is consistent among human operators and will be represented in units in the Raspberry Pi rather than actual SI units. To improve the efficiency of the exploration, we have two more buttons representing the left and right bumper sensors on both side the robot. They are used to detect the presence of obstacles on the side of moving direction. For example, the right bumper button will be pressed when the robot is moving along a wall on its right and the robot is able to record the obstacles immediately to its side.

3 IMPLEMENTATION

3.1 State Machine

A robot can be in the exploring state, joining state, or busy state. Figure 3 displays the robots state machine.

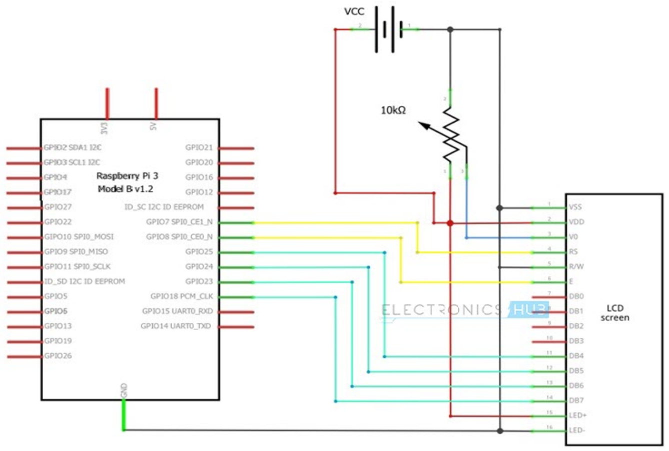


Figure 1: Raspberry Pi and LCD Schematic.Figure Credit: "www.electronicshub.org/interfacing-16x2-lcd-with-raspberry-pi"

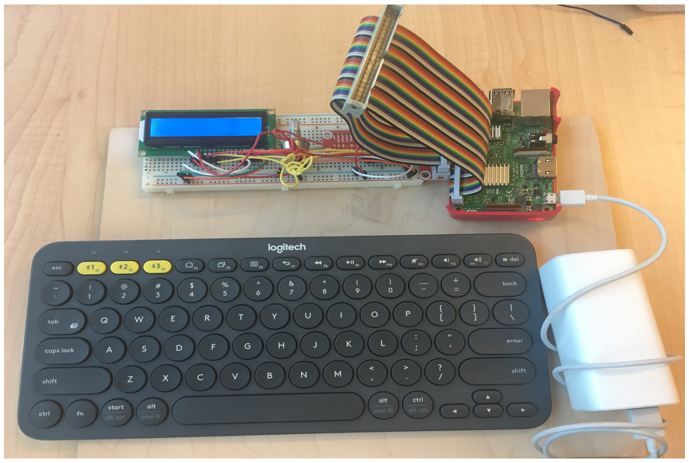


Figure 2: Prototype robot set-up which consists of Pi with Wi-Fi module, LCD circuits, keyboard ,mobile power bank, and a wooden base

3.1.1 Roaming State. If there are no other robots within its communication radius,a configurable constant in the code,the robot will be in Roaming State. In this state, it is mobile and it will be completing its given path. It will simultaneously broadcast its IP address and position and listen to the IP addresses and positions from other robots. If it completes its task or unable to execute the task due to physical limitation (e.g walls are preventing the robots to progress its given task), it will just recompute a new one and start exploring this new task.

3.1.2 Joining State. A robot can be in this state if there are robots within the communication radius (CR). They will remain in this state for the duration of the joining period, a time that is set by the user, which allows a robot in this state to continue connecting with

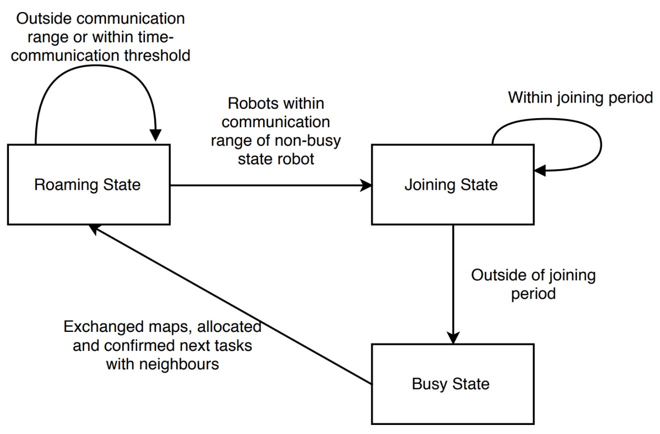

Outside communication range or within time-communication threshold
Robots within communication range of non-busy state robot
Within joining period
Roaming State
Joining State
Outside of joining period
Exchanged maps, allocated and confirmed next tasks with neighbours
Busy State

Figure 3: Robot State Machine

other robots. The definition of *within the communication radius* is when the absolute distance between the robots are off by a natural number unit defined by the user in the code. When they are within CR, the robots will make a connections; thus forming a network. In this state, the robot will stop its motion (therefore its task). It will exchange its IP's with the robots in its proximity to form a network. When the two or more robots are in this state, a constant timer (called *Joining Period* defined by the user will start. Within the joining period, other robots (whose current state is roaming) will have the opportunity to join the network formed by the robots (whose state is in the joining state).

3.1.3 Busy State. After the joining state, no other robots can join the network formed by robots in the joining state. In this state, each robot in the network will do the following:

(1) Each robot will send its current map to all of its neighbor
(2) Each robot will follow the map merging protocol (which will be discussed be in section 3.3).
(3) Once each robot has finished merging the map using its neighbors' maps, it will calculate all the tasks for itself and the other robots.
(4) Each robot will allocate the task according to the protocol defined in section 3.4.

After each robot receives a task, all the robots will go back to the roaming state to execute the new task. To avoid the robots from re-forming the same network that was established in the busy state, a timer will start (defined by the user) and within that timer, the robots will not communicate with another robots.

3.2 Failure Handling

We identified and handled failures under all three states of the robot with arbitrary number of robots failed. Below is the detailed failure description and corresponding handling.

(1) **Single Robot Crash and Recovery:**
Whenever the robot updates its current location or its task, it will log its location, map, task and energy level onto the disk immediately. If the robot crashes at any moment for reasons such as power outage, when the program restarts, it will first try to read from log stored on the disk and resume the previous state if any. If no log is found, it will generate a new task and start exploring.
(2) **Disconnected Neighbor During Robot "Meeting":**
When the robots form a local Ad hoc network and proceed to exchange maps and tasks, one or more of them could disconnect at any moment. Initially, we experimented with heartbeats but it quickly lead to memory shortage due to Raspberry Pi's limited RAM. We therefore decided to ping each robot prior to starting any inter-robot communications. We also maintained an attribute "rNeighbours" within the "Robot" struct which holds a list of real-time online neighbors.
Due to the nature of space exploration , previously transmitted information could still be used even if the original sender has disconnected. Therefore, if one robot disconnects while it's merging it's map, its neighbors will delete it from their own "rNeighbour" lists; if the robot disconnects during the task allocation, all its neighbors will stop receiving and responding tasks from the robot. In the rare case where the robot who's task was accepted by the network fails, the remaining robots alive in the group will still perform the dead robots task as each task created is unique (given that it assigned tasks prior to it's death). For the area that is supposed to be explored by the disconnected robot, it will be covered during the next "meeting" because the task generation algorithm will detect the undiscovered area and allocate another robot to fill the blank.
(3) **Recovery From Disconnected State:**
If the robot previously disconnects from an Ad hoc network "meeting" and then recovers, when it broadcasts joining-neighbor signals, the receiving robot from the original "meeting" network will reject as if the network is out of the range(non-visible) from the reconnecting robot. Therefore, the reconnecting robot would not interrupt any unfinished "meeting" and will proceed on its own task until a new group of robots are visible.

3.3 Map Merging

Our aim is to obtain an uniform map of the robots in proximity of each other. A robot will send a map payload to all robots in the network. A map payload will consist of the robot's map and its ID. Upon exchanging maps each robot will check for any conflicts between its current map and the one it received. A conflict happens when the same coordinate gets labels as an "obstacle" and "free space" by to different robots. To resolve conflict, each robot will take the latest measurement as the correct one.

3.4 Task Creation

The goal of this system is to map the entire area. The task creation algorithm takes in the current map stored in the robot and generates N tasks for N robots in the network. The output will be identical among robots as long as the input is the same. As shown in Figure 4, the algorithm firstly calculates the center of current explored map by finding the extrema of the map (Xmin, Xmax, Ymin, Ymax)

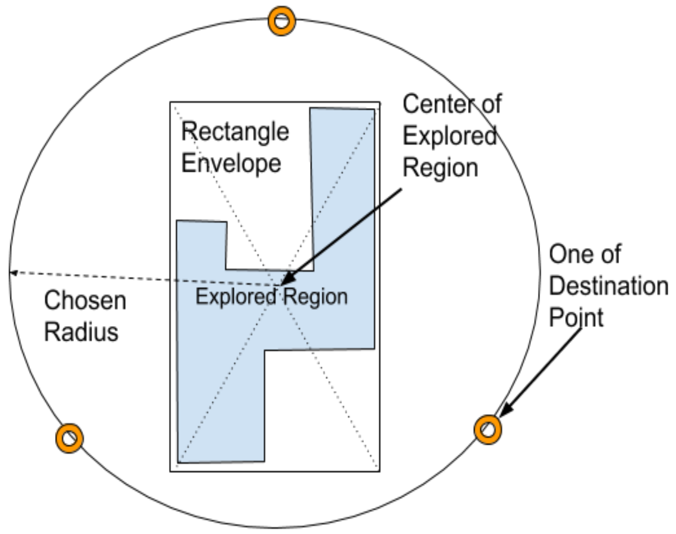


Figure 4: Diagram of the Task Creation logic

of the explored region and modeled it as a rectangle envelope to calculate its center. The algorithm then picks N destination points with same radius spreading evenly on the 360 degree angular range. Then the algorithm estimates the distance from its location to the destination by calculating the sum of differences between in x and y coordinate, as if the robot will be traversing a " L " shape path which is the likely layout of a rectangle room / corridor. A task is a destination point, and this algorithm will essentially generate a list of them.

3.5 Task Allocation

This protocol will take place following the task creation algorithm. From the results of the task creation algorithm the robot will choose the task that is closest to itself and choose tasks for the the other robots in the network at random. Then the robot will send its fellow robots the task it created for them along with its ID. The robot will wait for a response from its peers (whom are are alive), only when it receives all these responses will it continue to start exploring its new task. A robot can choose to either accept or refuse a task that it is assigned by another robot. They will decide to accept the task assigned to them by the robot with the lowest ID and therefore refuse the tasks assigned by robots with higher IDs including itself. The time sequence diagram displayed in Figure 5 describes the basis of this protocol where they are all in the busy state.

3.6 Inter-Robot Communication

As stated before, a Raspberry Pi 3B will represent the brain of the robot. As opposed to its predecessors (e.g Raspberry Pi 2B and the original Raspberry Pi) and other variants (e.g Raspberry Pi Zero), the Raspberry Pi 3B has a built-in Wi-Fi and Bluetooth chips, which allow the Pi to make networks calls with other machines.

With this feature, we have multiple choices of inter-robot communications, namely NFC, Bluetooth, Ad hoc Wi-Fi and infrastructure Wi-Fi. Ultimately, we have decided to use Ad hoc for the

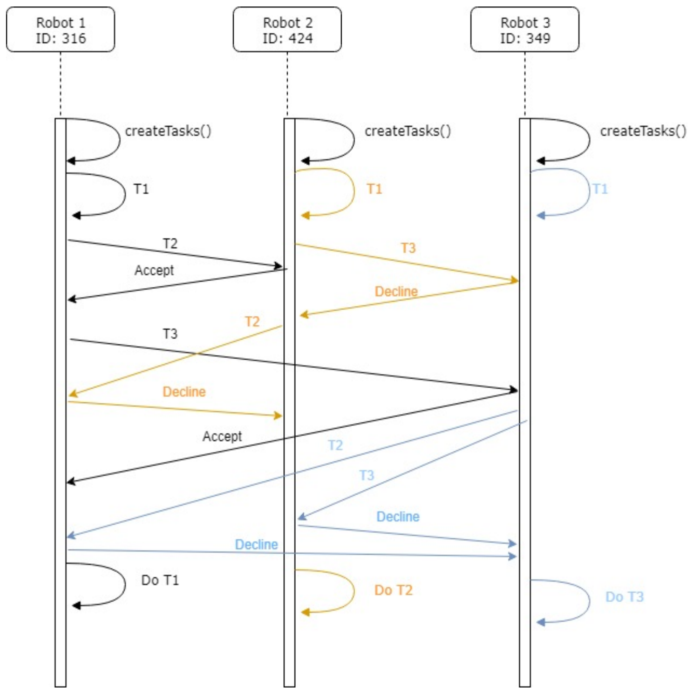


Figure 5: Time Sequence Diagram of Task Allocation Logic

robots to converse each other because we would like the intercommunication to be independent from any infrastructure hardware such as Wi-Fi router while still be able to utilize powerful Internet stack functionalities unlike NFC and Bluetooth.

3.7 Azure Integration & Web app

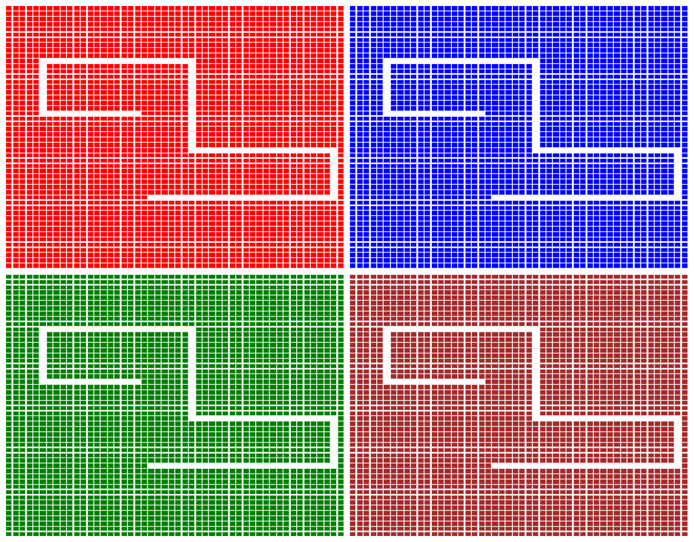


Figure 6: AzureMap Output

(1) **Web App:**
The front end of our system will be built with React.JS, the app will display all four robots' maps on the same page. The web app will fetch for maps from robots periodically, once a payload is received on the web app, the rendered map (see Figure 6) will be updated. A map will be presented as a heat-map. If a grid inside the heat-map is blank, it means the grid is an empty space, else if a grid is labeled a color, that means the grid is either unexplored or a wall.

(2) **Azure usage:**
The web app mentioned above will be hosted on Microsoft Azure Cloud. Due to the limitation of Ad hoc network, Azure will be inaccessible, however, a workaround can be implemented. Each robot will be connected to a laptop through Ad hoc network, and the laptop will be listening to the robot for its map. Once a map is send to the laptop, it will disconnect from Ad hoc network and connect to Wi-Fi, then relay the payload to a listener hosted on Azure.

4 RESULTS AND EVALUATION

Our system works as we described with connections and disconnections. From robots forming networks to merging maps, allocating tasks for each other, and waiting to receive a task and response from each other was tested and validated.

The task coordination procedure where each robot assigns the other robots in the network tasks and replies to them is shown via vector clocks with the help of Shiviz in Figure 7. Robot recovery was tested by disconnecting robot and observing the network's behavior. The expected behavior of the connected nodes is to continue what ever they are doing as if the deceased robot never existed. This is what was observed, for example, following task allocation all robots wait for a response from each of its peers before starting on its new task, when a robot was disconnected during this period the robots whom are still alive are aware of this event and proceed with out waiting on the dead robots' task - which will definitely never arrive.

To evaluate the accuracy of our distributed system we initially placed four robots at different locations in the room and allowed them to roam. We viewed their map as they updated them which they periodically sent to a laptop for display. The the accuracy of their map was viewed this way. The robots' LCD screen displayed the direction to take the next step in if it was in the roaming state, and just its state if it was either in the busy or joining state. The robot recognized when other robots were in its communication radius and connected only with these ones. It followed the expected behavior of merging maps, creating tasks, assigning tasks, and waiting for everyones response prior to doing the new task.

5 LIMITATION

(1) **Ad hoc:**
An Ad hoc network can be established between Raspberry pis and laptops, however, the network is only meant for the nodes to connected locally, accessing the Internet is impossible. It is proven troublesome when it comes to Azure, since Azure requires Internet connection. A workaround is provided, please check section 3.7 for more detail.

(2) **Pinpointing Physical Location:**
Civil GPS provides poor accuracy, therefore utilizing GPS to provide Robot's location can yield erroneous data and cause unexpected behavior in our system. A workaround is to use location defined by software for each robot in the system.

(3) **Physical Robots:**
Due to a constraint time line, the integration of wheels and motors on each Raspberry pi is unachievable at the moment.

6 FUTURE WORK

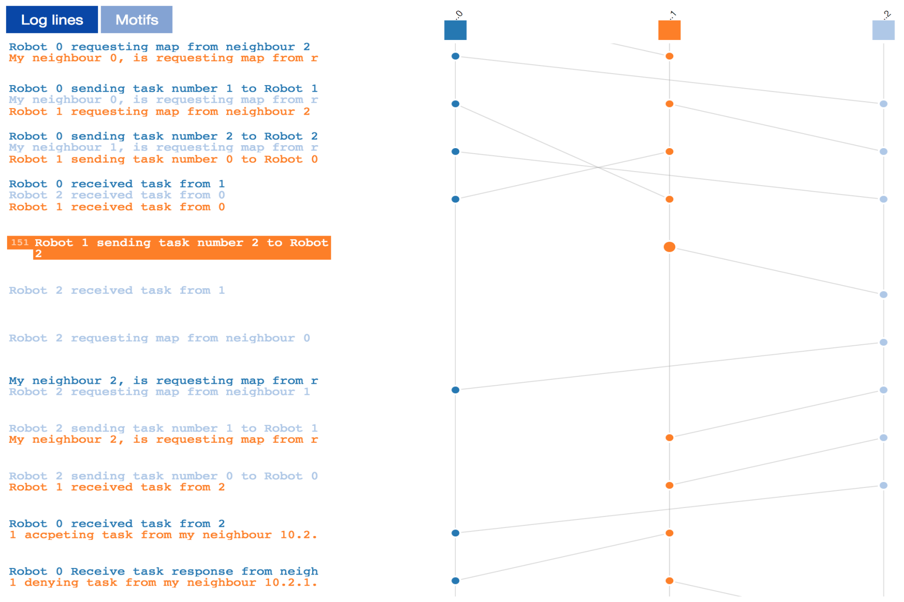
This system can be enhanced by adding new features which will make the system more robust and effective. The robot could be made to be fully automated and therefore eliminating the need of a human operator. For example, the robot could have wheels, sensors, and a GPS to facilitate it's own movement and navigation. Another feature is to allow the robot to regain energy it expended during its exploration by introducing a gas station. This spot will allow the robot to refuel it's energy when it is running low so it can maximize it's exploration. Additionally, robots goal could be expanded to not only map the area but also to find a specific object in the room with OpenCV and a camera.

The diagram above is a visualization of task allocation protocol generated by Shiviz. Shiviz is a tool used to visualize events happening in a distributed system across multiple nodes. As shown in the diagram,

A THIRD-PARTY LIBRARIES

We have used the following software libraries in this project:

- **Set**
  - Source: https://github.com/fatih/set
  - Usage: when looping through the possible neighbors array, we want to make sure not to make the same RPC call to the same neighbor.
- **go-hd44780**
  - Source: https://github.com/KarolBedkowski/go-hd44780
  - Usage: this library will interface with Hitachi HD44780 LCD Display using Golang. The current existing library was modified by Beichen Zhang because the original library unnecessarily clear LCD's register and reinitialized LCD each time it displays.
- **gpio**
  - Source: https://github.com/brian-armstrong/gpio
  - Usage: this library will interface with the general pin input and output (hence GPIO). The GPIO will be used to create interaction with the LED lights and LCD screen. The current existing library was modified by Beichen Zhang because original library failed to consider Linux serial port operation time resulting in random mistakes during the access.
- **govector**
  - Source: https://github.com/DistributedClocks/GoVector
  - Usage: It is a tool for analyzing vector clocks of each node of a distributed system.



```
Robot 0 requesting map from neighbour 2
My neighbour 0, is requesting map from r

Robot 0 sending task number 1 to Robot 1
My neighbour 0, is requesting map from r
Robot 1 requesting map from neighbour 2

Robot 0 sending task number 2 to Robot 2
My neighbour 1, is requesting map from r
Robot 1 sending task number 0 to Robot 0

Robot 0 received task from 1
Robot 2 received task from 0
Robot 1 received task from 0

151 Robot 1 sending task number 2 to Robot
    2

Robot 2 received task from 1

Robot 2 requesting map from neighbour 0

My neighbour 2, is requesting map from r
Robot 2 requesting map from neighbour 1

Robot 2 sending task number 1 to Robot 1
My neighbour 2, is requesting map from r

Robot 2 sending task number 0 to Robot 0
Robot 1 received task from 2

Robot 0 received task from 2
1 accpeting task from my neighbour 10.2.

Robot 0 Receive task response from neigh
1 denying task from my neighbour 10.2.1.
```

Figure 7: Shiviz Diagram of a Task Allocation Logic and Task Response

B CHANGES FROM THE PROPOSAL

While there is no major functional change, the modifications we made are mainly about implementation details.

1. **Replace Waiting Stage with Waiting Routine**
   During the development, We noticed that waiting stage is redundant because the robot requires a timing-out service both during joining stage and after the busy stage. Therefore, we removed the waiting stage and adding two timers for joining and leaving the "meeting".
2. **Replace Heart-Beat with Need-Based Pinging**
   Heart-Beat was initially implemented but it soon depletes the limited memory of Raspberry Pi. We therefore separate inter-robot communication into multiple stages and ping the receiver only when needed.
3. **Replace LED/Button with LCD/Keyboard**
   To display more information and make debug/demonstration more interactive, we decided to use a 16*2 LCD as an output device and keyboard as the input device. Due to the headless operation nature of our demonstration, we will turn the Pi's into Command Line Interface (CLI) mode as opposed to the Desktop mode. We wrote a start-up script to automatically run the program and listen to keyboard input without any additional setup.

C GLOSSARY

Robot: Raspberry Pi 3, LCD screen, Keyboard
Communication Radius: A predefined threshold distance set by user that robots relative distance needs to be within in order to connect with each other.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Ivan Beschastnikh and the rest of the teaching staff of CPSC 416 for providing us the opportunity to work on this complicated, yet interesting project.

The authors would also like to thank the Engineering Physics Project Lab for providing us with the space and the resources to create the necessary hardwares for the robots.