Networking Swarm

Suesin Chong, Sarah Hung, James Lam, Zhilong Liu

***Abstract*—The project aims to coordinate a swarm of moving robots to explore and send exploration data using RSSI while navigating into an unknown territory.**

***Index Terms*—Swarm, networking, RSSI, XBee, collaboration, communication.**

I. Introduction

THE objective of this project is to coordinate a swarm of moving robot via low-powered radio - Xbee and Received Signal Strength Indicator (RSSI). The goal of the swarm movement is to maximize the area of signal coverage of the low-powered radio on the robots.

We envision the swarm behavior to be the following. Robots will travel into an unknown territory one at a time. Each robot will attempt to follow the trace of pioneer robots by sensing their signal strengths. Once they reach the end of the trace, they will travel further into the unknown territory until they can maintain minimal connection to the rest of the group, before turning into one of the pioneers itself.

Once chained, every bots will follow the signal trace to reach the last bot until all the robots in the chain will accumulate near the farthest bot from the point of origin. Finally, the robots will spread out to maximize signal coverage.

II. Architecture

This section discusses the hardware architecture of Zumy, software stack and our calibration efforts.

A. *Hardware Architecture*

Zumy robots are prototype bots of Biomimetic Millisystems Laboratory of UC Berkeley. We have assembled four new Zumy's for stability and urability. Figure 2 shows the hardware architecture of a Zumy. Listed below are the components:

- **Board:** Hardkernel ODROID-U3 with 1.7GHz Quad-Core processor, running XUbuntu 13.10
- **Micro-controller:** mbed LPC1768 & Pololu DRV8833 Dual Motor Driver Carrier
- **Communication:** 2.4GHz XBee Series 1 Radio & Wifi module
- **Power:** 2-cell lithum-polymer battery
- **Frame:** Zumo Robot Kit [1]
- **Others:** Pololu Voltage Regulator (D15V70F5S3)

To setup XBee Series 1 module, we flashed a firmware which adopts *XBEE 802.15.4* protocol with the official tool - X-CTU. We turned on the API mode and set the baudrate 57600, which top speed for stable two-way communication. The board communicates with the Xbee via a serial port - */dev/ttyUSB0* with the baudrate is set to 57600, consistent with the XBee firmware.

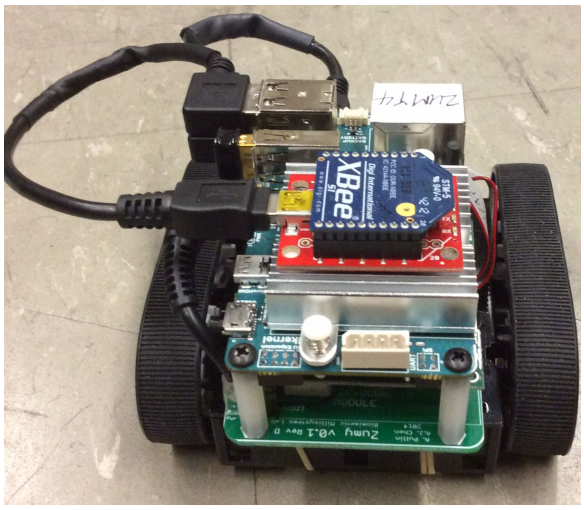

Fig. 1. Fully Assembled Zumy

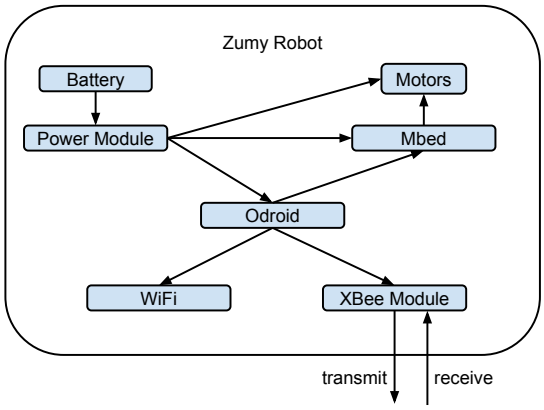

Fig. 2. Hardware Architecture

B. *Software Implementation*

Software architecture consists of 3 independent modules, all in Python. Detailed diagram of software architecture can be found in appendix.

*1) **XBee Library**:* We utilized the open source XBee library - *python-xbee*, available on [2]. The core library provides basic transmit, receive and packet decoding functionailities for Xbee radio. On top of which, we wrote a wrapper library to add threading and RSSI specific functionalities for Xbee communication.

The Xbee source address(MY) is used to identify individual bots, whereas the channel(CH) and PAN-ID(ID) are consistent across the entire network. RSSI specific post-processing functions includes obtaining a list of 30 consecutive readings from a particular sender and maximum, minimum, mean and

median of arbitary list of RSSI.

*2) **Motor Library**:* This library is a modification of a default library [3], which uses Lightweight Communications and Marshalling (LCM) framework [4] to communicate with the hardware motors. Using this library, we developed a set of functions that order the bots to drive forward or backward and turn in 90 degree. Also, driving distances are varied by drive time instead of voltage change under this framework.

C. Motor Calibration

Hardware differences and environmental factor like texture of floor often affect motors in unexpected ways. Hence, we have to find the optimal speed and drive time parameters for each movement of the Zumies, so that they are able to perform precise movements, such as turn 90 degree and drive specific distance as intended in the algorithm.

*1) **Hardware Difference**:* The motor strength varies across different robots, hence we need to calibrate each Zumy independently for every movement. In particular, difference in Zumy bots right and left motors were problematic since the bot will turn when expected to drive straight.

*2) **Battery Strength**:* It was observed that a single Zumy bot could exhibit varying motor strength based on the battery level. Zumy running on low battery would run slower even with same parameter. Hence we always ensure the moving bot is high on battery.

We compromise by observing the bot on the run, and decide whether the movements are within an acceptable range with the understanding that achieving perfect turn and drive is almost impossible in non-ideal environments.

III. Signal Indicator

To achieve our objective, it is necessary to have an adequate signal indicator that could correlate with distance between bots. The Received Signal Strength Indication (RSSI) is an indication of the RF energy at the antenna port in dBm (Decibel-milliwatts) which should correlates with distance from transmitter in ideal conditions.

However, RSSI is generally considered too noisy and seldom used as a strict indicator of distance by itself. Hence, we have to first investigate whether RSSI is a good indicator of relative distance. To this end , we performed numerous experiments to investigate relationship between RSSI and bot distance under various conditions.

A. Experimental Setup

Our experimental setup involves 1 transmitter and 1 receiver communicating continuously. RSSI value was recorded for each packet received while incrementing or decrementing the distance between the transmiter and the receiver by an fix amount each time. Experiments took place in two locations: Davis Hall 605 and Cory Hall 309, which are drastically different environments. Performing experiments in the different environments with different signal interferences proves that the observations were reliable and applicable in non-ideal environments.

B. Observations & Findings

*1) **Moving vs Stationary**:* RSSI values fluctuate greatly when the bots were in motion. Hence, we decide to only process RSSI values obtained when bots were stationary since the amplitude of fluctuation is much smaller under this situation. We were also adviced to collect multiple RSSI samples and compare among their maximum, minimum, median, and average to further reduce the effect of fluctuation on our algorithms.

*2) **Constructive & Destructive Node**:* Although the standard deviation was large when one of the Zumies entered constructive or destructive nodes, it was observed that using maximum instead of average RSSI readings at each distance provided approximately a linear correlation with the distance within the range of 1.5 meters. This observation has been verified in non-ideal environment.

*3) **Orientation**:* 3D Orientation of the Xbee devices have drastic impact on RSSI values. There was no observable trend when the orientations of Xbee devices are not controlled. In particular, RSSI readings are most strongly correlated with relative distance when the two Xbees are oriented in parallel to each other at the same height. Any movement and reorientation of Xbee devices in the perpendicular axis only introduce random and undesirable fluctuation. Thus, we standardize the orientation of Xbee devices by taping them onto the Zumies.

C. Plots on RSSI versus Distance

Figure 3 shows one of the best results. Distance was decremented by 0.2 meters each time. From the figure, we observe that the RSSI value is approximately linear with distance, hence using maximum RSSI as an indicator of distance within this ranfe is feasible. This observation is consistent across both ideal and non-ideal environments.

D. Nearest Neighbor on Signal Trends

We analyzed the features of overall signal trend of a approaching or leaving bot. We characterize data set at each fixed location with meta data like maximum, minimum, mean, median, mode, and standard deviation. Feature of entire signal trend include meta-data at each stop and differentials between stops. For a sufficiently long signal trend, we were able to classify the path as towards, away or noise by calculating sum squared differences and covariance of the new path with the data set.

IV. Algorithm

The path tracing and spreading of the robots as discussed in Introduction is achieved by gradient ascend and gradient descend algorithm. To find the closest pioneer robot, each Zumy will perform gradient ascend on the pioneer robot's RSSI. Similarly, to distance itself from all the neighbor robots, each Zumy will perform gradient descend on the RSSI of its neighbors collectively. This section gives an overview of the two algorithms.

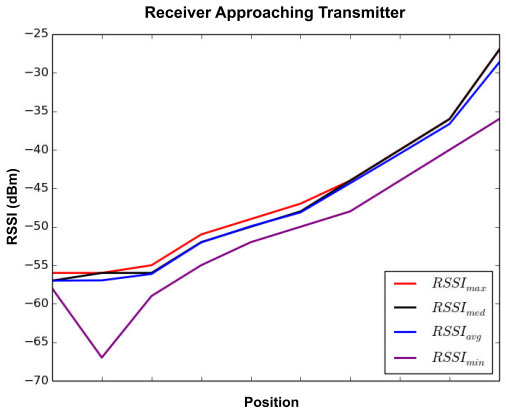

Fig. 3. RSSI signals as a function of distance

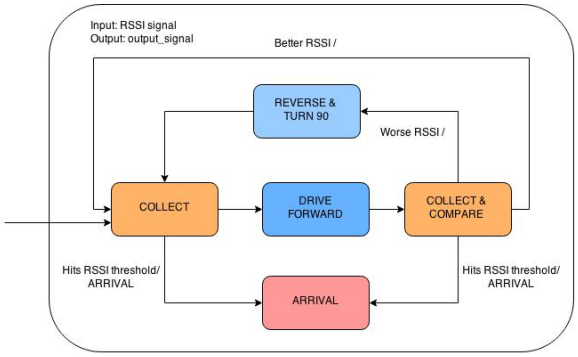

Fig. 4. State machine of the gradient ascend algorithm.

A. *Gradient Ascend Algorithm*

In the gradient ascend algorithm, the robot undergoes a routine of RSSI comparison between two different locations. The Zumy robot will first calculate the maximum RSSI based on 30 samples collected on its original location. It will then travels for a carefully calculated distance in the direction that it is currently facing, in order to collect another thirty samples of RSSI values. Comparing the two maximums, if the signal strength of the new location was better than that of the original location, the robot will repeat the routine. If contrary, the robot will return to its original location and turn for 90 degrees then repeat the routine. The algorithm terminates when the observed RSSI was above a certain threshold.

*1) **RSSI-related Drive Distance**:* The distance traveled is calculated based on the observed RSSI. If a high RSSI value is observed, this indicates that the robot is near the signal transmitter. Therefore, the robot will be given a smaller traveling distance. Likewise, a greater travel distance will be given to the robot if the observed RSSI was small. This design decision reduces the execution time of this algorithm in general in that it reduce the likelihood for close range errors.

*2) **Navigation Stage**:* In case when the robot cannot determine a better location after it has investigate in all four directions, it will enter the next navigation stage in the same location. Navigation distance is increased by 10cm for every subsequent navigation stage. The robot in navigation stage n will travel n*10cm in each direction.

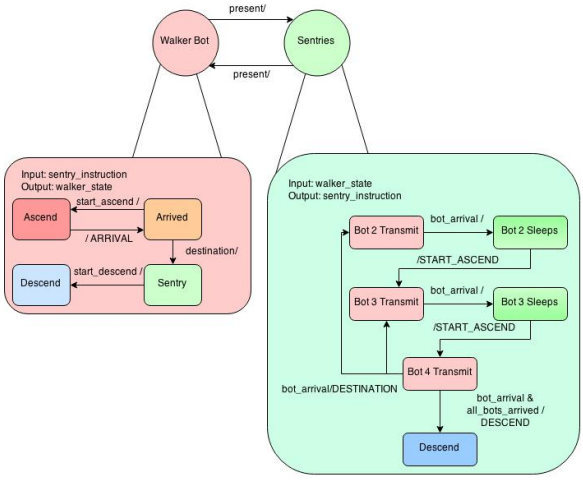

Fig. 5. State machine of the overall algorithm.

The bot has navigation distances of 10cm multiples because 10cm is the wavelength of the Xbee signal. This will ensure we are comparing signal strengths between peaks, troughs or signal strength in the same phase.

*3) **Right Turn Refinement**:* Since the Zumy robot will often travel in all four directions in a navigation routine, it is beneficial for the robot to keep track of the RSSI values from all locations of four different directions. Spefically, if the RSSI values were both less than that of the original location and that of the location by the previous turn, the Zumy robot will turn rightwards instead of leftwards (which is the default setting). This design choice was based on the speculation that an inaccurate RSSI measurement has occured. Adopting this design helped reduce the execution time of the algorithm.

B. *Gradient Descend Algorithm*

It is similar to *Gradient ascend algorithm*, except it seraches for a smaller RSSI value instead a greater RSSI value. The ending RSSI threshold is -50dB.

*1) **Multiple RSSI measurement**:* The main difference between *Gradient descend algorithm* and *Gradient ascend algorithm* is that the descending robot has to listen to multiple transimitter bots, and navigate to a loaction far away from all transimitters. Currently, the walker bot measures RSSI values from all transimitter robots to obtain the average RSSI value, and compares this value with the end RSSI threshold value.

C. *Swarm Algorithm*

The swarm algorithm is a combination of *Gradient ascend algorithm* and *Gradient descend algorithm*. We initialize a bot chain, where all bots except Zumy1 is in sentry mode, Zumy1 is in gradient ascend mode. Zumy1 executes *Gradient ascend algorithm* to navigate to Zumy2 and Zumy3, until it reaches the end of the chain, and it switches into *Sentry mode*. After that, the bot at the beginning of the chain, Zumy2, exits *Sentry mode*, and navigates to the end of the chain also via *Gradient ascend algorithm*. Eventually, when all bots reach the end of the chain, they execute *Gradient descend algorithm* one by one to expand the network coverage area.

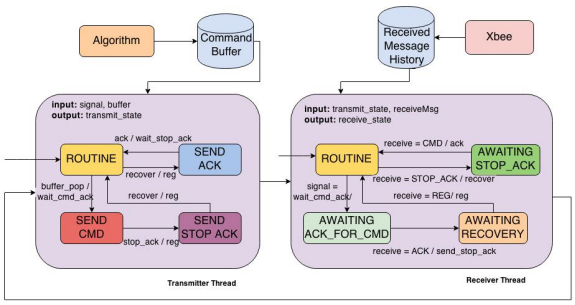

Fig. 6. Synchronous Reactive model for Communication Thread.

D. State Machine Analysis

The hierarchical state machine for overall algorithm is a asynchronous composition which has history transition, but not pre-emptive transition. Asynchronous because the walker and sentry bot are independent actors. The input and output signals of each machines are command packets traffic between the walker and sentry bot. Each robot keeps track of its state in the swarm algorithm with a variable, refined states correspond to variables in the robot's memory.

However, the signals are neither instanteneous nor simultaneous between the 2 major states. This model captures the interaction of one walker bot with the entire bot chain, but does not capture the individual transition from walker to sentry bot.

V. Communication Protocol

The coordination among the robots relies heavily upon network communication. Hence, we need to define a set of signals to order the robots to enter gradient ascent or descent at appropriate times in order to achieve the envisioned swarm movement. Since the control of swarm robots is decentralized in nature, a concurrent network communication protocol was implemented and discussed in this section.

A. Concurrency

Each bot always runs both the transmission and receiving thread, regardless of the state it is in. When the bot is not transmitting, transmitter thread is still on, but do not write to Xbee and sleep for a short time when it is executed. The receiver thread is always running and updates the received message buffer. Below is a synchronous state machine that models our thread organization.

The algorithm feeds next commands into the command buffer, which is popped into the transmitter once the previous command acknowledgement cycle is completed. Each command is stopped only when the entire command acknowledgement cycle is completed. In both the transmitter and receiver state machine, there are 2 entirely independent cycles, the send-ack and send-command cycle in transmitter & await-ack-for-command & await-stop-ack cycle in receiver. Each of the cycle can only be pushed back to routine when specific received message is read, this pre-empts any bots from being simultaneously involved in more than 1 command-acknowledgement cycle. This ensures the atomicity and sequence of commands sent is preserved.

*1) **State Machine Analysis**:* A Synchronous Reactive model dictates that all signals are absent at all times except at global tick of the clock. Input and output of each machine corresponds to shared variables within the code, hence it is simultaneously available to both machines.

However, messages and commands popped from buffers might be stale, thus some inputs might not be instanteneous at tick of global clock.

This is a synchronous composition of interleaving semantics type 3 with history transitions. The processor chooses either the transmitter or receiver will react.

B. Commands

We have defined a series commands for the Zumy bots for collaboration to achieve our algorithm's goal. Below are some of commands Zumy bots recognizes:

1) *ASCEND_START* - Start gradient ascend
2) *DESCEND_START* - Start gradient descend
3) *TRANSMIT_START* - Start transmission of bot
4) *ARRIVAL* - Ascending bot signals arrival to sentry bot
5) *ACK* - Acknowledges receipt of above commands
6) *STOP_ACK* - Stops acknowledgement packet

Acknowledgement packets acknowledges specific commands and will only be processed by an intended recipient.

VI. Analysis

In this section, we analyze the performance of the algorithms and communication protocol. We determine the bottleneck of each algorithm by identifying the critical path in graphs of tasks of various algorithms.

A. Gradient Ascend Performance

On average, gradient ascend of a distance of 1m takes from around 3 to 15 minutes.

The greatest obstacle is a large area of constructive interference that is far from the transmitter, wrongly identified as a better location. The bot eventually gets out of a constructive node due to the navigation distance increase, but this usually causes long delay since the constructive interference might span a large area. Performance between a walk that fell into a constructive node versus another which did not is directly correlated with the area of the constructive interference. Destructive nodes can usually be identified by high standard deviation, our algorithm avoids trusting such readings.

Noise in RSSI measurement in distances where RSSI signals do not relate linearly with relative distance also causes delay. The random fluctuation in RSSI often lead the walker bot into random location. Different Xbee transmiter and receiver pairs also have different correlation between relative distances and RSSI. Distances that correspond to the same RSSI can vary from 5cm to 10cm from pair to pair.

B. Gradient Descend Performance

Most time is spent on collecting data from specific Xbees for gradient descend. Especially from a marginally connecting bot since the receiving thread is often loaded by packets from

Xbees with better connection. Simultaneous gradient descend often results in collision, An IR sensor would be able to prevent collision, but our current hardware architecture lack the component.

C. *Communication delay*

To ensure the command-acknowledgement cycle is atomic, all commands from another bot is ignored until the transmitter is free from the previous cycle. Our implementation parses the received message buffer for messages that will allow for transition out of a state. Time wasted on searching for the parsing process increases as size of buffer and amount of irrelevant messages increases.

VII. Trials & Future Work

Here is the list of things we tried but either did not have time to complete, or did not work.

- Communicating across different channel with different frequency.
- Multi-stop gradient ascend with expectimax tree

VIII. Conclusion

Throughout the project, we have utilized concepts and techniques taught in lectures and lab to design, model and analyze the task at hand scientifically. We have modelled our system with hical and Synchronous Reactive state machine, and found the abstraction to be extremely useful for conceptualization and debugging. And designed a concurrent communication model. We now understood the significance and gained experience dealing with concurrency and multitasking issues covered in class. Finally, we assessed the performance our system by analyzing the the observations and trials. The systematic approach made the design and analysis scalable

The biggest obstacle was to determine the validity in the RSSI values we collect in our experiments, due to various concurrency and buffer issues. We re-executed many experiments for multiple times for accurate data sets.

All in all, it has truly been a wonderful learning journey. We have gained invaluable experience with embedded systems completing the project.

Acknowledgment

The team would like to thank the Biomimetic Millisystems Lab for providing the hardware components. We offer our sincere gratitute to Matt Weber for mentoring over the course of this project and providing many insightful advices. We would also like to thank Professor Edward Lee for inspiring us to look into the possibilities of analyzing and using RSSI as distance indicator. Special thanks also goes to Ben Zhang for guiding us through as we stumple upon signal collecting. Lastly, we would also like to thank Austin Buchan for providing the initial draft of the idea.

References

[1] Zumo robot kit. [Online]. Available: http://www.pololu.com/product/2509
[2] G. R. Paul Malmsten and Brian. (2012) Xbee python library. [Online]. Available: https://github.com/blalor/python-xbee
[3] A. Chen. (2014) Zumy library. [Online]. Available: https://github.com/andrewjchen/zumy
[4] A. Huang, E. Olson, and D. Moore, "LCM: Lightweight communications and marshalling," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.