MUSSE: A Designed Multi-Ultrasonic-Sensor System for Echolocation on Multiple Robots

Xiaodong Wu, Miguel Abrahantes, Mark Edgington
Department of Engineering
Hope College
Holland, Michigan 49423, USA
e-mail: contact@xavierwu.com; {abrahantes, edgingtonm}@hope.edu

Abstract—**This paper presents a multi-ultrasonic-sensor system designed for Simultaneous Localization and Mapping (SLAM) on the open source robot Kobuki Turtlebot. The operating system used in this research project, Robot Operating System (ROS), is also an open source project initially released in 2007. Our system has reduced response time and lower cost than traditional laser sensors. It consists of 8 HY-SRF05 rangers that can work sequentially on an embedded system such as a Raspberry Pi. The communication between the embedded system and the robot is achieved with a Python package that communicates via local TCP/IP. Our multi-sensor structure can be applied to a multi-agent system, composed of 3 identical robots. Our results from preliminary experiments show the possibility of conducting SLAM using our designed system. By implementing this improved design on a multi-agent system, the SLAM process can be conducted more efficiently and accurately than a single robot with a laser sensor.**

Keywords-multi-robots; mapping; echolocation; SLAM; ultrasonic sensing

I. INTRODUCTION

SLAM is considered as a fundamental prerequisite of fully autonomous robots [1]. Building a map and localizing a robot in an unknown environment is the main purpose of SLAM. A robot needs to make a series of decisions on path planning based on its perception to surrounding objects. An autonomous exploration of a robot is usually the beginning of the study of SLAM [2]. A robot collects necessary information including distance and orientation via a series of sensors and records speed and angle to estimate the pose and position of itself while constructing the map. There are several options for the proximity sensors, which are essential to the research of SLAM. A typical choice is laser sensors, which are accurate, fast, but generally costly. Though laser radars usually have high speed of data acquisition, they also have certain limitations such as sensitivity to strong light and noticeable errors at highly reflective surfaces like glass. [3] Instead of using laser sensors, our research team developed a robust and low-cost Multi-Ultrasonic-Sensor System for Echolocation (MUSSE). The idea of echolocation comes from the inspiration of bats, who use sound waves to navigate. Multiple sensors can reduce the time of data collection notably. To further improve the efficiency of SLAM, MUSSE was implemented on a multi-agent robot system based on ROS.

II. SLAM

A. Background

This study is the continuation of a previous research we conducted focusing on ultrasonic echolocation using one transducer [4]. SLAM has been a question since the moment when people try to localize and map simultaneously. Most similar studies using SLAM rely on laser range finders for accuracy and speed, however we have built an alternative low-cost solution using multiple ultrasonic sensors. Unlike pure echolocation, SLAM is essentially about mapping and localizing in an unknown environment. Therefore, measuring the distance to surroundings and recording the path are two fundamental prerequisites for SLAM. A typical method of recording path is odometry, which uses the data from motion sensors or rotation sensors to estimate a robot's trajectory of spatial displacement [5]. For a robot to localize and map on its own, it has to be able to build a reference system of coordinates and landmarks. The problem of modeling the environment and sensing the location of a moving robot simultaneously has attracted interest of researchers in the realm of robotics.

B. Challenges

A robot's path and the map can be estimated by the probability density function $p(x_{1:t}, m|z_{1:t}, u_{1:t})$ where observations (z) and controls (u) are known [6]. A map is needed for localization while a pose estimate is needed for mapping. SLAM can usually be processed through approximation. There are many different algorithms developed for SLAM. ROS comes with several packages for localization and automatic path optimization. A map can be considered as a collection of facts and data about the world, whereas constructing a robot's own reference in an unstructured environment is the real challenge for robot [7]. Also, the development of ROS for multiple robots is still in the early stage. Thus, a number of packages do not natively support multiple robots, which is a big challenge for our project.

III. SENSORS

We designed a multi-sensor system based on eight HY-SRF05 transducers and an embedded system (Raspberry Pi) which is capable of signal processing, data storage and

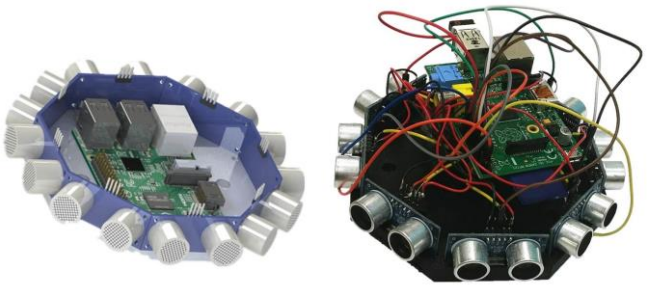
978-1-5090-1362-3/16/$31.00 ©2016 IEEE

transmission, and wireless communication on a local network.

An ultrasonic signal typically lies in the range of 20-100 kHz. There is little noise or interference in this range. A higher frequency than 20kHz increases the resolution of ranging by decreasing the wavelength of signals. Thus, more features can be recorded as signal data. The sound frequency of HY-SRF05 sensors is 40kHz, which is adequate for our experiments. HY-SRF05 sensors have a valid distance range of 2cm to 450 cm and are set to be triggered every 50mS at least to avoid a false echo on the next range measurement [8]. The distance is calculated based on the time interval and the speed of sound (340.1m/s in dry air at 22°C [9]).

A. *Implementation*

Fig. 1 shows the design of MUSSE. Eight sensors can scan the plane and return the localization information in milliseconds. The low-power Raspberry Pi makes it possible to power the system using a portable 5V battery or from the power ports of Kobuki robots.



(a) 3D Rendering of MUSSE **(b) Implemented Prototype**

Figure 1. MUSSE model

A challenge of this multi-sensor structure is that each sensor must avoid operating concurrently with other sensors due to acoustic interference and lag in the embedded system and the signal transmission process. They must operate sequentially, and the response time needs to be reduced as much as possible to minimize errors in the localization process. Furthermore, ultrasonic signals can be distorted on sound absorbing materials or sloping surfaces, which are common in daily environments. Thus, we need to figure out how to filter out these kind of unwanted measurements.

B. *Measurement Script*

Listing 1 shows the Python script we wrote to perform sequential measurements using eight ultrasonic sensors. The core "measure()" function calculates the distance between a sensor and an object based on the pulse lag of triggered signals. To conserve GPIO pins on the embedded system without causing any interference, only one GPIO port (#8) is used as trigger, and 8 GPIO ports are used as echo. Those eight echo ports are contained in a created list, "GPIO_EchoList". A for-loop is made to take measurements while iterating over all eight echo ports in the "GPIO_EchoList".

This script was initially designed for debugging purpose so that all the measurements can be displayed on our master computer in real time. Changes were made to it when Measure.py was integrated to the MUSSE package (See Section IV.B) so that the measurement data is projected to related nodes for SLAM. Then further analysis and approximation can be done with the measurement results.

```
import time, RPi.GPIO as GPIO

def measure():
    GPIO.output(GPIO_TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)
           start = time.time()

    while GPIO.input(GPIO_ECHO) == 0:
        start = time.time()
    while GPIO.input(GPIO_ECHO) == 1:
        stop = time.time()

    pulse_interval = stop-start
    distance = (pulse_interval * 34010)/2
    return distance

GPIO.setmode(GPIO.BOARD)
GPIO_TRIGGER = 8
GPIO_EchoList = [7,11,12,13,15,16,18,22]

print("Ultrasonic Measurement")

GPIO.setup(GPIO_TRIGGER,GPIO.OUT)
for i in GPIO_EchoList:
    GPIO.output(GPIO_TRIGGER, False)
    GPIO.setup(i,GPIO.IN)
    GPIO_ECHO = i
    distance = measure()
    if distance > 450.0:
        print("Pin#%d: Object too far" % i)
    elif distance < 2.0:
        print("Pin#%d: Object too close" % i)
    else:
        print("Pin#%d Distance : %.1f cm" % (i, dist
ance))
    time.sleep(0.01)
GPIO.cleanup()

#Part of this code was adapted and used with permiss
ion from Matt Hawkins
```

Listing 1. Measure.py script taking measurements via 8 transducers

C. *Algorithm*

A well-designed algorithm needs to be simple enough considering the real-time execution and the limited computing power of a low-cost system [10]. Acquiring more distance measurements in any one location will increase the resolution of mapping. Thus, the robot is set to rotate by a small amount after finishing the first measurement, with the location being unchanged. Although more rotations will increase the resolution, one rotation should be enough for the MUSSE system because it has eight transducers, which are sufficient for the measurements. Fig. 2 shows two measurements, one before and one after rotating 22.5°. All distance measurements are stored in a matrix, D_m, in which X stands for the number of rotations. For each measurement, MUSSE records 8 readings. The matrix D_m is then analyzed and processed using Algorithm 1.

$$D_m = \begin{pmatrix} A_1 & A_2 & \cdots & A_8 \\ B_1 & B_2 & \cdots & B_8 \\ \vdots & \vdots & \ddots & \vdots \\ X_1 & X_2 & \cdots & X_8 \end{pmatrix}$$

Algorithm 1 Measurements Optimization

Initialize the Measure.py Script
$i \leftarrow$ Pin Index Number
$A_i, B_i \ldots X_i \leftarrow$ Measurements of *Distance*
$T \leftarrow$ Number of Rotations
while T $\leq$ Planned Number of Rotation **do**
  for Each pin i **do**
    Get Distance Reading
    Update the position of the robot
    Store readings into D_m
    T + 1
  end for
end while
for All pins i **do**
  if number of $(A_i \leq X_i) < 4$ **then**
    Keep Vector $[A_1, A_2 \ldots A_8]$
  else
    Keep Vector $[X_1, X_2 \ldots X_8]$
    Repeat for all vectors in D_m till the one before X
  end if
end for
return Final Vector (Array of 8 Measurements)

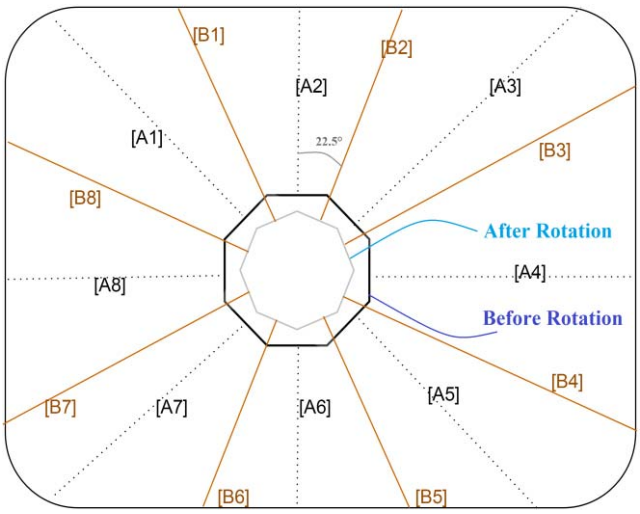


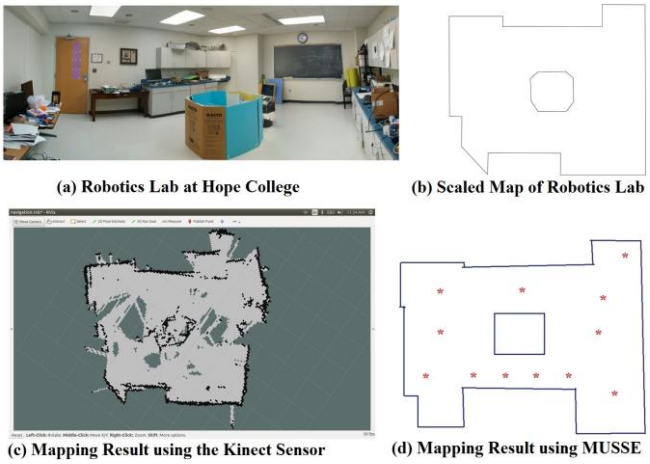
Figure 2. Measurements

D. Data Transmissions

The data transmission module is realized via a local TCP/IP network. ROS supports a TCP/IP-based message transport, known as TCPROS [11]. A Python script was written and integrated to the MUSSE package to send distance data to the computer on the robot in real time. All computers in this research, including the Raspberry Pi boards, have Avahi implemented to allow mutual communications and controls on a local network. Avahi is an open-source decentralized zero-configuration networking implementation that enables multi-cast DNS resolution, address allocation and service publication locally [12]. To access SSH, using Avahi is much more convenient than looking up the host names individually, especially when the number of computers is relatively large.

E. Results

The results of mapping using a Kinect sensor and using MUSSE are compared in Fig. 3, in which (d) shows the sketch from our preliminary experiments where multiple points were selected with known coordinates. The distance data was then analyzed and visualized as the sketched map. The map based on the Kinect sensor (Fig. 3(c)) was obtained by using the "turtlebot_navigation" package integrated in ROS. It has better resolution to corners and curves, and more details in a complex environment, while the map based on MUSSE has a more accurate big picture of the entire environment. Occasionally there is some unexpected fluctuation of measurements because of the limitations of ultrasonic sensors: the ultrasonic waves diverge within a given beam angle in which the detected object reflects the wave. Small delays of signal also occur. In addition, specular reflection and diffuse reflection happens in a real environment that affects the accuracy of an ultrasonic sensor's reading [13].



(a) Robotics Lab at Hope College (b) Scaled Map of Robotics Lab

(c) Mapping Result using the Kinect Sensor (d) Mapping Result using MUSSE

Figure 3. Comparison of mapping

IV. SOFTWARE

Python is essential to this project. It is a simple but powerful programming language that is used in ROS, and in our developed software packages and APIs. Several ROS packages, such as gmapping and sensor_msgs, were used for experiments, simulations and comparison of results.

A. ROS

ROS (ros.org) is an open-source software framework developed and maintained by the Open Source Robotics Foundation. As one of the most popular project for robotics, ROS can run on various Unix-based operating systems including Linux distributions and Mac OS X. It contains a number of libraries, packages, and algorithms. ROS has not only a stable package management system but also an advanced message passing service framework which enhances the interoperability of its packages [14].

B. *MUSSE Package*

Packages in ROS are similar to software in an operating system. They contain libraries, nodes, APIs, configuration files, parameters and other necessary components to realize some functions. The communication architecture is based on topics and nodes. Topics, which are "named buses" [11] for unidirectional communication, contain nodes inside. Nodes do not communicate with each other directly; instead, they publish messages and data to topics to which other nodes subscribe. All scripts in the MUSSE package are written in Python, because ROS has native low-level support for Python. Nodes and topics can be used in a Python script using `rospy`, a Python client library and API. Thus, data can be transferred among different modules and packages.

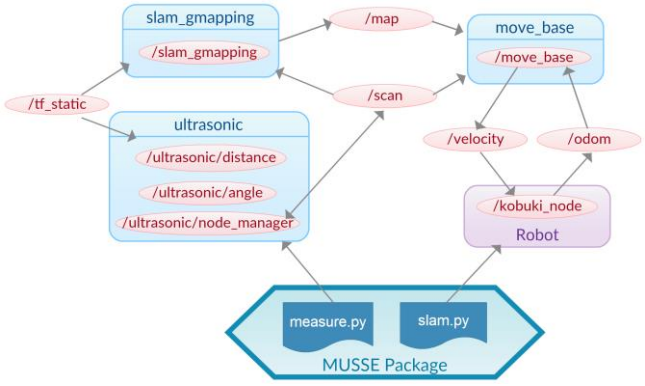


Figure 4. Overview of the MUSSE package ROS graph

ROS graph is a visualization tool that shows all nodes and topics currently running. Figure 4 shows an overview of the ROS graph that contains the essential nodes and topics of MUSSE. The MUSSE software package has two fundamental modules: "measure" and "slam". The "measure" module sends commands to the "ultrasonic" topic to collect the distance data, and then the collected information is transferred to the "slam_gmapping" topic via the "scan" node. At the same time, the "slam" module commands the robot to move via "kobuki_node". Next "slam_gmapping" completes mapping and collects odometry information to localize the robot. The "ultrasonic" topic publishes a distance message at a constant frequency, and "scan" subscribes to both "ultrasonic" and "slam_gmapping" topics at the same time. "tf_static" saves the coordinate frame information of a robot. Since the "tf" package also works in a distributed system, "tf_static" is significant to transport information about the coordinate frames among multiple robots. The communication architecture of ROS is like a giant web of nodes, clusters, and lines that make information transmission reliable and efficient.

C. *MUSSE on Multi-Agent Robot Systems*

Using multiple robots can greatly reduce the time to build a global map since each robot gathers information independently and sends the processed data to the master robot. Distributed systems that are commonly used in mobile robot exploration employ multiple robots to map independently, reducing the influence of single point of failure [15]. The communication in a distributed multi-agent system is not always reliable because all actions are separated. However, the challenge is that the difficulty of system control will increase as the number of robots increases in a centralized system. Also, modeling the map will also be more difficult because the complex system can get more unpredictable and noisy. In contrast, a centralized robot system with a master robot and several slaves was used for SLAM by us. Unlike the Swarm robot system which has a large number of simple robots, a centralized robot system works more efficiently and generates more optimal plans since the master robot can make decisions for each robot based on the information of each robot. Since we aim to design a low-cost solution for SLAM, centralized architecture is used because slave robots do not need as much computing power as the master does.

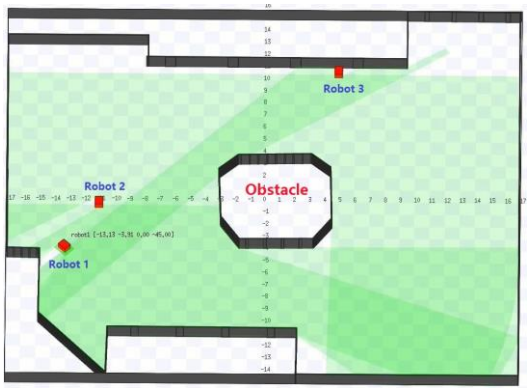


Figure 5. Simulation of multi-robot

Choosing and designing algorithms depends on how the expected map must be presented. MUSSE is designed for grid-based maps. As a classic method of environment representation, grid-based maps are easier to construct and control in comparison to others like topological or feature-based maps. No extra filter is needed and thus the stability and speed of constructing a map are desirable [16]. Also, ultrasonic sensors have the best performance when they are sending signals perpendicular to a surface. Because of the uncertainties of ultrasonic measurements, the errors of mapping can be decreased when an area is divided into small blocks [17]. Thus, grid-based mapping makes more sense in this case.

V. CONCLUSION

We have presented a possible solution for robust SLAM on multiple robots using low-cost ultrasonic sensors and computing equipment. This work demonstrates that ultrasonic sensors can be used as an alternative to laser rangers and other proximity sensors. They are usually inexpensive and easy to implement, and can generate good results in relatively ideal environments. Due to the limitation of computing power and cost of the sensors, the accuracy of MUSSE can be further improved by trying different algorithms and calibrating the transducers through a number of measurements. Overall, the main goals are achieved with possibility of future improvements.

A. Future Work

We need to keep testing and revising the design of our multi-sensor system to make it work seamlessly with ROS. Currently we managed to scan the environment using a Kinect sensor, and we will modify the software package to replace the Kinect sensor with our ultrasonic sensor system completely. We will also improve the algorithm for distance measurements.

ACKNOWLEDGMENT

We would like to thank Dave Daugherty for his assistance in fabricating and modeling our robots, and Matt Hawkins for allowing us to use the "measure()" function he wrote in our Measure.py script.

REFERENCES

[1] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," 2002, pp. 593–598.

[2] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed multirobot exploration and mapping," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1325–1339, 2006.

[3] S. D. Tumbo, M. Salyani, J. D. Whitney, T. A. Wheaton, and W. M. Miller, "Investigation of laser and ultrasonic ranging sensors for measurements of citrus canopy volume," *Applied Engineering in Agriculture*, vol. 18, no. 3, p. 372, 2002.

[4] X. Wu, P. D'Orazio, M. Edgington, and M. Abrahantes, "Robotics echolocation test platform," *2015 IEEE International Conference on Electro/Information Technology (EIT)*, p. 558, 2015.

[5] F. Pasila, Y. Tanoto, R. Lim, M. Santoso, and N. D. Pah, Eds., *Proceedings of Second International Conference on Electrical Systems, Technology and Information 2015 (ICESTI 2015)*. Springer Singapore, 2016.

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

[7] T. Barfoot, "On the representation and estimation of spatial uncertainty," University of Toronto Institute for Aerospace Studies, Tech. Rep., 2010.

[8] Srf05 technical documentation - robot electronics. [Online]. Available: http://www.robot electronics.co.uk/htm/srf05tech.htm

[9] P. Papacosta and N. Linscheid, "An inexpensive and versatile version of kundt's tube for measuring the speed of sound in air," *The Physics Teacher*, vol. 54, no. 1, pp. 50–51, 2016.

[10] P. Hoppenot and E. Colle, "Real-time mobile robot localisation with poor ultrasonic data," *3rd IFAC Symposium on Intelligent Component and Instrument for Control Application*, pp. 135–140, 1997.

[11] D. Forouher. (2014) Topics - ros wiki. Open Source Robotics Foundation. [Online]. Available: http://wiki.ros.org/Topics

[12] Avahi (software). [Online]. Available: https://en.wikipedia.org/ wiki/Avahi (software)

[13] C. H. Kim and J. Y. Lee, "Feature extraction method for a robot map using neural networks," *Artificial Life and Robotics*, vol. 7, no. 3, pp. 86–90, 2003.

[14] J. O'Kane, *A Gentle Introduction to ROS*, 2nd ed. Columbia, SC: O'Kane, 2014.

[15] C. M. Gifford, R. Webb, J. Bley, D. Leung, M. Calnon, J. Makarewicz, B. Banz, and A. Agah, "Low-cost multi-robot exploration and mapping." IEEE, 2008, pp. 74–79

[16] I. Rekleits, "Exploration tutorial," Tech. Rep., 2010

[17] V. Santos, J. Goncalves, and F. Vaz, "Perception maps for the local navigation of a mobile robot: A neural network approach," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, vol. 3, p. 2193, 1994.