Robotics Echolocation Test Platform

Xiaodong Wu, Palmer D'Orazio, Mark Edgington, Miguel Abrahantes
Department of Engineering, Hope College, Holland, Michigan 49423 USA
{edgingtonm, abrahantes}@hope.edu

Abstract— **This work describes a mobile system that has been developed to precisely and reliably carry out echolocation experiments (data collection) for later analysis. A Kobuki robot was used as a base unit, providing mobility and accurate odometry. Custom mounting hardware was designed for the robot to accommodate a laptop for controlling the robot, along with a Microsoft Kinect sensor and ultrasonic transducers for taking experimental measurements. A Python based software package was designed to provide simple control of the robot and its sensors. This software was designed to work within the Robot Operating System (ROS) framework, and includes high-level interfaces for controlling robot movement and for the simultaneous playing and recording of sounds. Each data-collection experiment consists of a sequence of movements and measurements that the robot should perform. The system we have developed will make future data collection simple, allowing for subsequent study and analysis of echo signals.**

I. INTRODUCTION

A bat can identify its position within an environment by using ultrasonic chirps to perform echolocation. The biology of this process has been studied in depth, and engineers have applied ultrasound ranging to the tasks of mapping and object detection. However, most engineered systems do not mimic bats, and there is still much to be understood about how a bat actually processes the echoes it hears. In order to facilitate the investigation of how bats process this information, a mobile robotic system was developed to autonomously perform echolocation experiments and collect data for further analysis. Requirements of such a system include the ability to accurately navigate to different locations, to produce and record high-frequency sound-waves, and to be able to reliably log this information. In this paper, we present the design of this system, including the details of hardware and software development that were required to realize the system.

II. BACKGROUND

Though scientists have been investigating bat migration and homing for more than a century [1], bionic research which combines robotics and advanced analytical mathematics has existed for a comparatively shorter amount of time. Engineering systems based on animal echolocation have been discussed and implemented since the early 1900s, but despite the advances in related fields (biology, computer science, robots, etc.), bio-inspired sonar has yet to produce a "killer app" [2]. Although algorithms have been developed to determine a room's shape using first-order echos [3], these require the use of at least 4 microphones. Bats, however, can apparently accomplish this task with only two "sensors", their ears. In the realm of robotics, there are several approaches to acquire spatial information for use in simultaneous localization and mapping (SLAM) algorithms. In this domain, echolocation has largely been supplanted by systems that use optical sensors. Nonetheless, systems such as the University of Antwerp's BatSLAM have demonstrated the feasibility of high-accuracy large-scale biomimetic echolocation-based mapping [4]. In addition to navigating in large spaces, bats can track and catch insects, and differentiate between plant species [5]. Developing a deeper understanding of the signal-processing performed by bats may allow for higher-resolution mapping, and possibly surface characterization.

This work was supported by Michigan Space Grant Consortium.

III. THE ROBOT

A. Kobuki Platform

Our research team used the iClebo Kobuki [6] as the base for our mobile robotic system. The Kobuki is a wheeled base that provides high-resolution odometry information, allowing for precise control over the robot's position. This makes the base suitable for our experiments, which require detailed logs of the robot's pose. The robot provides a low-level interface that can be controlled via a USB connection from a laptop or embedded device like a Raspberry Pi. Open-source drivers are provided for this purpose. The base includes several power supply ports that can be used to provide power to the connected devices. In order to mount these devices and transducers to the mobile base, we designed a frame that attaches to the top of the Kobuki (see Figure 1). This frame provides sufficient space to transport a typical sized laptop, along with several transducers. The transducers are placed on the highest platform to minimize the extent to which the structure interferes with the produced and recorded sound-waves.

B. Sensors

When considering what kinds of signals would be best to work with for our experiments, we chose to use signals in the ultrasonic frequency range between 20 and 100 kHz. This was motivated by the fact that bat calls are typically in this frequency range. Furthermore, at higher frequencies there tends to be less ambient noise, and it is possible to detect smaller (i. e. higher-resolution) spatial features due to the smaller-wavelengths of the produced and recorded signals.

The ultrasonic sensors used in this study are similar to those used in proximity sensors. In contrast to typical proximity

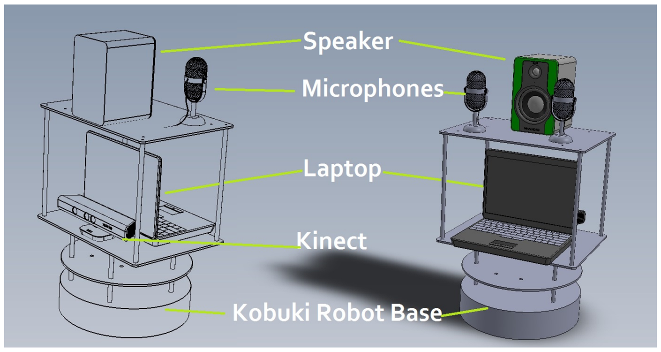

Fig. 1. Rendered Design Model of Test Platform

sensors, which provide only a "distance signal", the sensors we use provide us with a raw measurement of the detected sound waves. Most inexpensive off-the-shelf sensors have a fairly narrow bandwidth of around 4-5 kHz, with a center frequency of 40 kHz. Because the signals we wish to use are often swept sine-waves that have a frequency ranging from 50 to 100 kHz, the off-the-shelf sensors are not ideal for our experiments. Though our first design iteration makes use of these sensors, they will be replaced with wider-bandwidth devices in the future. To house our transducers, we designed and 3D-printed a structure that allows the placement of multiple transducers in a variety of different positions and angles (see at the top of Figure 2).

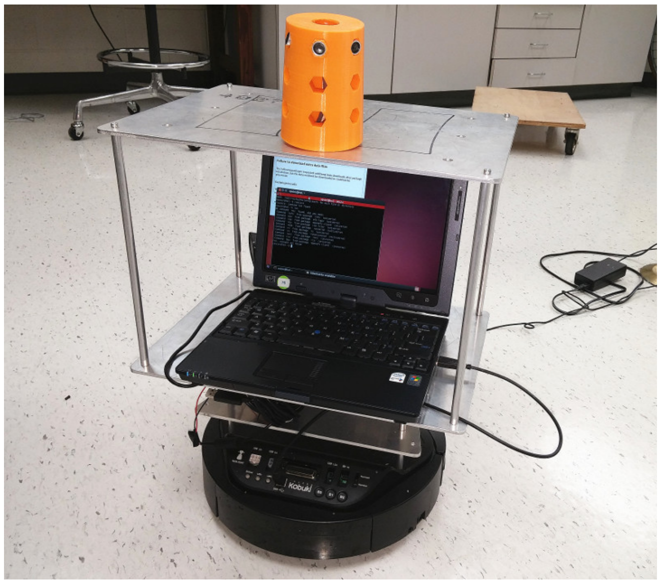

Fig. 2. Implemented Test Platform

A Microsoft Kinect sensor was also integrated in the system, primarily for odometry calibration and real-time visualization during experiments. The Kinect's depth sensor, similar to ultrasonic proximity sensors, projects an infrared laser pattern onto objects, and its internal CMOS sensor acts like a receiver [7]. Because it uses a laser, lighting conditions do not affect its functionality. Thus, the Kinect sensor is similar to ultrasonic proximity sensors, except that it is able to provide distances to multiple points simultaneously, whereas an ultrasonic proximity sensor only provides a single distance.

C. Signal Processing Hardware

In order to produce and measure the sound waves, it is necessary to have hardware capable of generating and recording electrical signals at the desired frequencies. We investigated two different means of doing this. We first investigated the use of an instrument called the Red Pitaya [8]. The Red Pitaya uses a "partially" open-hardware design, in which all software, including Verilog Hardware Description Language (HDL) files, are made available for modification and extension. The hardware design (e. g. PCB schematics), however, is not published.

In order to replicate the chirp signals used by a large number of bat species, we needed the ability to generate signals that span a range of frequencies from 30-80 kHz, and have a total duration of around 2.5 ms. With a desired minimum sample rate of at least 5 samples/cycle, at least 1000 samples must be stored (or generated) in order to produce the desired signal. Though the maximum frequency at which the Red Pitaya can generate signals is higher than 80 kHz, using its default hardware design does not allow for storing this many samples, nor is it possible to smoothly (i. e. so that there is minimal jitter between the samples) generate the samples without in-depth modification of the Red Pitaya's architecture / hardware API.

As a result, we settled on using a standard PC sound-card, capable of producing and recording signals with a sample-rate of 192 kS/sec. Due to this relatively low sample-rate (for our purposes), unless we relax the 5 samples/cycle constraint, the maximum frequency we can produce or record with sufficient resolution is 38.4 kHz. During our initial tests, it appeared that the sound card had a low-pass filter (LPF) in place with a cutoff frequency around 20 kHz, but we later came to realize that this was a *software imposed* limitation, associated with the PulseAudio software layer commonly used in Linux distributions [9]. By avoiding the use of PulseAudio, and directly interfacing with the sound-card through the low-level API exposed via the Advanced Linux Sound Architecture (ALSA), we were able to both generate and record signals at the maximum sample-rate (i. e. 192 kS/sec). The relationship between PulseAudio, ALSA, and a sound card is illustrated in Figure 3.

D. Communication and Control

Typically, two computers are used when running an experiment: one which connects directly to the robot and exposes it as a ROS node, and one which runs the ROS "core" framework, and issues commands. For the robot-mounted computer, a Raspberry Pi was initially considered, but it lacked the processing power required to fully make use of the Kinect sensor. We therefore chose to use a mid-sized laptop. Both computers were connected to the same local network.

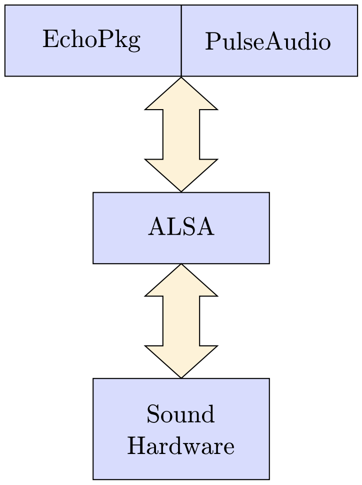

Fig. 3. Typical Audio System Architecture in Linux: ALSA allows a single process to connect to it, which could be either PulseAudio (which exposes an audio interface that multiple processes can use simultaneously), or, for example, our software package. ALSA, in turn, is responsible for communicating directly with the underlying hardware.

IV. Software Framework

In addition to our development and construction of the robot's hardware platform, we also created a system of software by which experiments can be designed that control the robot to perform movements and produce and record sound signals. The software package is available in the form of an online software repository [10].

A. Robot Operating System

Most commercial robots run on proprietary software, developed by the manufacturer. These systems were immediately ruled out due to cost concerns and the need for low-level system control. Several open-source software ecosystems are available for robot control. We selected Robot Operating System (ROS) because of its compatibility with the Kobuki, its growing community, and its flexible nature[11, 12]. ROS represents processes as *nodes*, which may communicate with each other over a network using system-agnostic *messages*. Any node may listen to any message, so it is simple to simultaneously control and robot and simulate its projected path. Though ROS has a somewhat steep learning curve, we found [13] and [14] to be excellent resources that aided our ROS installation and setup process.

ROS provides native APIs for C++ and Python. We chose to build our control software on the Python API, *rospy*, since it allows for fast project iteration and compatibility with many existing libraries related to other components of the system [15]. Since ROS runs on unix-based operating-systems, we also wrote shell scripts to set up environment variables and start all necessary nodes for our system.

Several nodes are used to run each experiment. A desktop computer runs the ROS core node, a blank "map" node, a simple robot simulation node, and a node for the visualization package *RViz* [16]. The robot's laptop (connected to the same network as the desktop) runs a node which turns pose messages into actual motor commands, and returns odometry data.

The Movement API is a software layer we developed for controlling the motion of the robot (discussed in more detail in Section IV-B). This layer starts a node behind-the-scenes to broadcast messages based on each experiment script's instructions. Finally, *RViz* is able to plot the goals and the ideal simulated trajectory of the robot alongside the actual path followed by the real robot.

B. Audio and Motion Libraries

Using Python, we created two custom APIs to simplify the design of experiments via the writing of simple scripts. The first API, the *Movement API*, handles lower-level interactions with the Kobuki robot through the rospy interface. It can create waypoints, convert them to ROS goals, and broadcast messages on the network. It also listens for odometry messages from the robot. An overview of this API's structure is shown in Figure 4. Figure 5 shows a partial ROS graph of the Movement API in action. The API's ROS node communicates with a node called *move_base*, which coordinates goals, map data, and feedback from the robot's sensors.

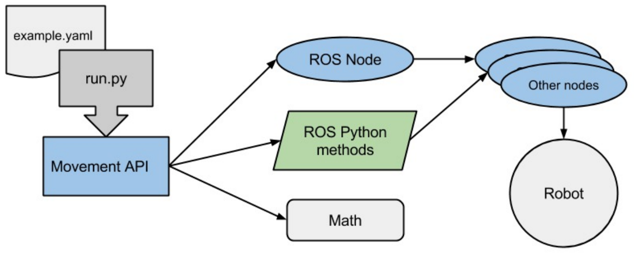

Fig. 4. Movement API Overview

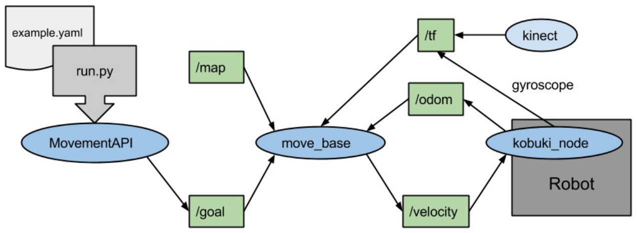

Fig. 5. Movement API ROS graph

The *Sound API* can simultaneously play and record WAV files on hardware connected to a computer's sound card. It relies on NumPy and PyAudio [17, 18], and may in the future support using other types of hardware (such as the Red Pitaya). An overview of this API's structure is shown in Figure 6.

C. Experiment Runner

A single Python script, `run.py`, coordinates all of the other software packages of our system. It reads a YAML file [19], determines which actions the robot must perform, and uses the Sound and Movement APIs to execute the instructions.

Listing 1 shows an example YAML experiment profile. The file has a name, and list of movement and audio-related tasks to perform. The first two sound experiments, *sweep1* and *sweep2*, simply play a specified WAV file and record simultaneously. The last sound experiment generates and outputs a swept sine-wave (i. e. chirp) signal. The robot's initial pose is given in (x, y, direction) format. The action list tells the robot where to move, and when to perform sound experiments. Each action is executed in sequence. Figure 7 shows an experiment in progress using the robot's point-of-view Kinect camera, an external camera, and the RViz simulator view.

```
# Example experiment profile definition

name: experiment_profile_1

sound_experiments:
    sweep1:
        record_duration: 2
        play_filename:   sweep1.wav

    sweep2:
        record_duration: 3
        play_filename:   sweep2.wav

    sweep3:  # a generated sweep
        record_duration:     1      # seconds
        start_freq:          1000   # hertz
        end_freq:            60000
        signal_duration:     2      # seconds
        play_device_index:   0
        record_device_index: 1

# poses are specified as [x, y, angle_in_degrees]
initial_pose: [0,0,0]

# by default, pose x and y values are in meters,
# but these can be scaled with this value
scale_factor: 1

actions_list:
    - {action: move,          pose: [1,0,0]   }
    - {action: snd_expt,      name: sweep1    }
    - {action: move,          pose: [1,1,180] }
    - {action: snd_expt,      name: sweep1    }
    - {action: move,          pose: [0,1,270] }
    - {action: snd_expt,      name: sweep1    }
    - {action: move,          pose: [0,0,0]   }
    - {action: snd_expt,      name: sweep1    }
    - {action: gen_rec_sound, name: sweep3    }
```

Listing 1: Example YAML file defining an experiment

V. Conclusions

In this paper, we have presented the design for an echolocation test platform prototype. Both the hardware and software aspects of this system have been described. In particular, a hardware structure has been built to enable control devices and transducers to be connected with a Kobuki mobile base, and a software package has been created that integrates ROS, Python tools, and audio hardware. Our work

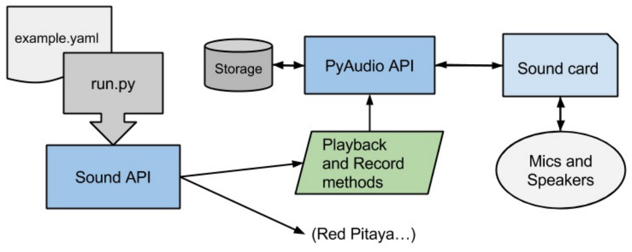

Fig. 6. Sound API

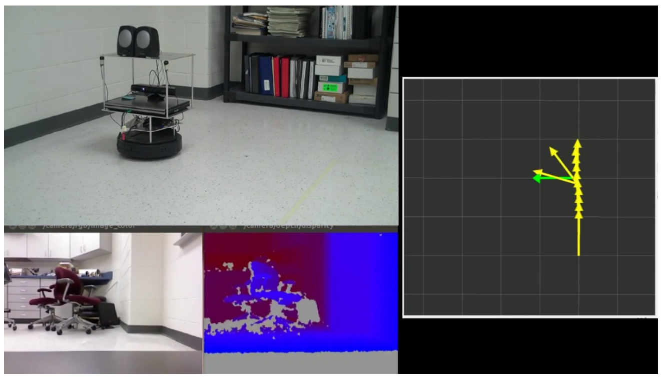

Fig. 7. External camera, onboard Kinect, and RViz tracking

shows the feasibility of building a relatively low-cost robotic system to effectively support echolocation research.

A. Future Work

In the future, the platform's ultrasonic hardware will be improved. Wide-band transducers and microphones are necessary to produce and capture the calls of a wider variety of bat species. In addition to monolithic wideband transducers, it may be possible to create a wide-band system by combining an array of low-cost transducers and microphones.

Acknowledgements

We would like to thank Dave Daugherty for his help in the fabrication of the shelving system. We also appreciate Andrew Bunnell's help related to equipment and material acquisition.

References

[1] D. R. Griffin, "The early history of research on echolocation," English, in *Animal Sonar Systems*, ser. NATO Advanced Study Institutes Series, R.-G. Busnel and J. F. Fish, Eds., vol. 28, Springer US, 1980, pp. 1–8.
[2] R. Müller and R. Kuc, "Biosonar-inspired technology: goals, challenges and insights.," *Bioinspir Biomim*, vol. 2, S146–S161, Dec. 2007.
[3] I. Dokmanic, R. Parhizkar, A. Walther, Y. M. Lu, and M. Vetterli, "Acoustic echoes reveal room shape.," *Proc Natl Acad Sci U S A*, vol. 110, pp. 12 186–12 191, Jul. 2013.

[4] J. Steckel and H. Peremans, "Batslam: simultaneous localization and mapping using biomimetic sonar.," *PLoS One*, vol. 8, e54076, Jan. 2013.

[5] Y. Yovel, B. Falk, C. F. Moss, and N. Ulanovsky, "Active control of acoustic field-of-view in a biosonar system.," *PLoS Biol*, vol. 9, e1001150, Sep. 2011.

[6] *Kobuki::About*. [Online]. Available: http://kobuki.yujinrobot.com/home-en/about.

[7] H. Hexmoor, "Essential principles for autonomous robotics," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 7, no. 2, pp. 1–155, 2013.

[8] *Red Pitaya*. [Online]. Available: http://redpitaya.com.

[9] *PulseAudio*. [Online]. Available: http://www.freedesktop.org/wiki/Software/PulseAudio (visited on Feb. 20, 2015).

[10] *EchoPkg Software Repository*. [Online]. Available: https://bitbucket.org/palmtree94/echopkg.

[11] *About ROS*. [Online]. Available: http://www.ros.org/about-ros.

[12] *Core Components*. [Online]. Available: http://www.ros.org/core-components.

[13] R. P. Goebel, *ROS by Example: A Do-It-Yourself Guide to the Robot Operating System*. Raleigh, NC: Lulu.com, 2014.

[14] J. M. O'Kane, *A Gentle Introduction to ROS*. University of Southern Carolina, 2014.

[15] *Rospy homepage*. [Online]. Available: http://wiki.ros.org/rospy.

[16] *Rviz*. [Online]. Available: http://wiki.ros.org/rviz.

[17] *NumPy*. [Online]. Available: http://www.numpy.org/.

[18] *PyAudio: PortAudio v19 Python Bindings*. [Online]. Available: http://people.csail.mit.edu/hubert/pyaudio.

[19] *The Official YAML Web Site*. [Online]. Available: http://yaml.org.