SmartGuide - A Navigation Support for Visually-Impaired People

Prachi Shukla, Aditya Narayan, Furkan Eris
prachis, adityan, fe

Abstract

We propose Smart Guide, an embedded system leveraging the mobility and low energy gains of smaller chips to create a better lifestyle for people that are visually impaired. We have prototyped the concept of using gumstix boards in order to use echolocation in unison with bluetooth technology to help blind people "hear" their surroundings. We have used a master and slave gumstix board with communication to a bluetooth headset or speaker in unison with one sensor on the slave board and multiple sensors on the master board. This has proven the usage of real time sensor buffering with both multi sensors on single board and multi boards capable of cross communication.

I. Introduction

In the modern world, embedded computing have been integrating humans into the digital world at an exponential rate. The advancement of such technologies have also started catering to the less-fortunate people. Our project aims at targeting visually-impaired people and develops wearable technology to assist them about their surroundings. We call our product SmartGuide, which is a navigation system to assist blind people by tracking obstacles and notifying them via a bluetooth headset.
To establish this goal we set about proving the concept in three main ways:

- We wished to establish communication between multiple gumstix boards with one of them a master terminal that would be in communication with a bluetooth headset or speaker and buffer messages incoming from the slave terminals. We were able to prove this was possible with two gumstix boards. The master gumstix is setup as a head sensor, while the slave gumstix is set up as a foot sensor.
- We wished to show that it was possible to receive multiple messages from ultrasonic sensors to a single board and buffer real time messages for communication. We were able to do this for two sensors thus proving it possible to scale up the number of sensors on a single terminal.
- We established that a range of distance values is separable and can be parsed to output different messages which we proceeded to do with the messages "Safe" , "Warning" and "Danger"

The sensors receive messages every 200 milliseconds which are than buffered by the master terminal every 1500 milliseconds. It is possible to increase the speed or change the rate of communication but this requires further calibration as well as increases the noise with increasing speed. We found this value to give fast and reliable results.

II. Design Overview

- **Smart Cap:** Carries two sensors - one for detecting obstacle in front of the user, while the other for detecting obstacle behind the user. Master Gumstix resides in the smart cap.
- **Smart Boot:** Carries one sensor for to detect a step in front of the user. Slave gumstix resides in the smart boot
- **Speaker/Headphones:** Smart Cap sends appropriate audio file to the speaker/headset to inform the user of the obstacle distance.
- **Components Used:**
  - 2 Gumstix Verdex boards which is based on ARM architecture.
    - Speed: 400 MHz

  - ∗ RAM: 64 MB
  - ∗ Flash: 16 MB
  - ∗ Linux 2.6.27 or higher
- Bluetooth modules for inter-gumstix & audio communication
- 3x HC-SR04 ultrasonic sensors [1]
  - ∗ Operating Voltage: 5V DC
  - ∗ Operating Current: 15mA
  - ∗ Measure Angle: 15^o
  - ∗ Ranging Distance: 2cm - 4m
- JBL speakers

III. Design Flow

- Slave kernel module (Prachi)
- Slave user-level application (Aditya)
- Master kernel modules (Prachi)
- Master user-level application (Prachi, Aditya, Furkan)
- Gumstix to gumstix communication over bluetooth (Aditya, Prachi)
- Gumstix to headset/speaker communication over bluetooth (Aditya)

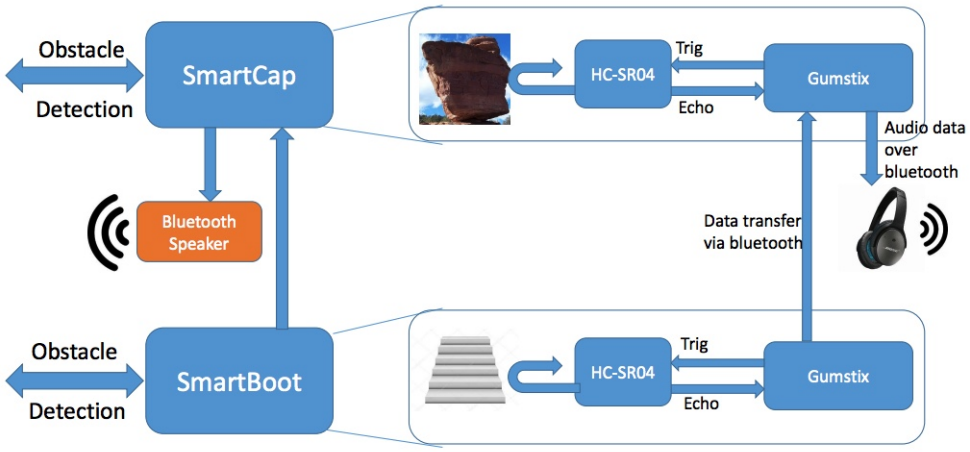

Fig. 1: High-Level Block Diagram

Overall Contributions:
While Aditya and Prachi focused more on C-coding, Furkan focused on setting up the circuit requirements correctly. We tried many options, before we finalized the final design. All the challenges have been mentioned in the last section of the report.
Aditya : 35%, Prachi: 35%, Furkan 30%

IV. Project Details

In this section, we describe details about the kernel modules and user-level applications running on the master and slave gumstix along with bluetooth communication protocol.

We used HC-SR04 as the ultrasonic sensor module in our project. This sensor provides a sensing range of 2cm to 4m. These sensors are inexpensive (1$) and provide a good ranging accuracy along with an ease of implementation. The pin diagram of HC-SR04 is shown in Fig 2. It consists of Vcc, GND, ECHO and TRIG pins. As these sensors need a constant 5V power supply, we used a separate battery pack to provide the power instead of supplying Vcc from the gumstix. The ranging accuracy is up to 3mm.

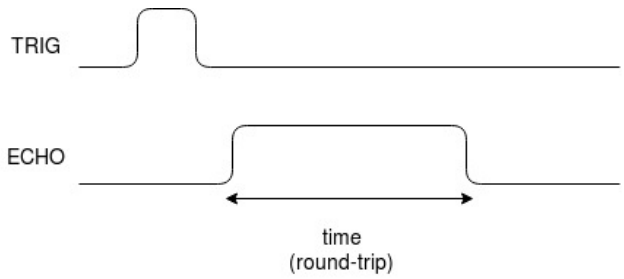

Fig. 2: TRIG and ECHO for HC-SR04

A. How ultrasonic sensor (HC-SR04) operates

The basic principle of working is as follows[2]:

- We program the gumstix to send a *HIGH* 10us pulse in the TRIG pin every 1 sec.
- The module automatically send eight 40kHz signal
- The ECHO pin is set *HIGH* immediately. The ECHO pin remains high until the pulse is reflected back from an obstacle and detected in the sensor.
- We calculate the total time for which ECHO pin remains *HIGH*, and multiply it by the speed of sound to calculate the obstacle distance.

We describe how the gumstix interacts with the sensor module on the kernel and the user side in the next section.

B. Kernel module in master gumstix

We implement 2 kernel modules in the master gumstix, one for the sensor to record the obstacle distance in front of the user, and one for the second sensor to record the object distance behind the user. Each master kernel module provides a *TRIG* to the ultrasonic sensor using a *GPIO* pin, receives an *ECHO* signal from the corresponding sensor and converts it to the distance of the obstacle in front the sensor. Obstacle within a distance of 0.425m is considered to be in 'danger' zone, 0.850m is considered to in 'warning' zone and beyond 0.850m in 'safe zone. Implementation details are briefly discussed below:

- Set *GPIO* pins for *TRIG* and *ECHO*.
- Register an interrupt handler for the *ECHO* signal from the sensor
- Use ktimers to send a periodic *TRIG* 10us pulse to the sensors every 2 seconds
- Record *ECHO* high signal by using the function do_gettimeofday(). NOTE: jiffies wasn't giving the finer time values that we wanted
- Convert the time recorded into distance using speed of sound.
- Decide if the distance calculated falls in 'safe', 'warning', or 'danger' zone.

C. User application in master gumstix

User application in the master gumstix performs the following functions periodically every 2 seconds:

- Reads proc filesystem of the master gumstix to read the object distances relative to the two sensors on the smart cap.
- Reads from device /dev/rfcomm0 to read the object distance from the slave gumstix relative to the sensor on the smart boot. This is a read over bluetooth channel, which is explained in later sections.
- Distances read from the 3 sensors is collected and prioritized

- The application communicates with the speaker/headset over a bluetooth channel. This way, the user is informed about the obstacle distance by the smart cap. The audio messages sent to the bluetooth headset/speaker are of the type:
  - Danger Messages: 'danger step', 'danger behind', 'danger ahead'
  - Warning Messages: 'warning step', 'warning behind', 'warning ahead'
  - Safe Massage : 'safe'

Audio bluetooth communication is explained in later sections

D. Kernel module in slave gumstix

The kernel module in the slave gumstix provides a TRIG to the ultrasonic sensor using a GPIO pin. It receives an ECHO signal from the sensor and calculates the distance of the object in front the sensor. Similar to the thresholds defined above in the master gumstix, obstacle within a distance of 0.425m is considered to be in 'danger' zone, 0.850m is considered to in 'warning' zone and beyond 0.850m in 'safe zone. Implementation details are the same as that of master gumstix:

- Set *GPIO* pins for *TRIG* and *ECHO*.
- Register an interrupt handler for the *ECHO* signal from the sensor
- Use ktimers to send a periodic *TRIG* 10us pulse to the sensors every 2 seconds
- Record *ECHO* high signal by using the function do_gettimeofday(). NOTE: jiffies wasn't giving the finer time values that we wanted
- Convert the time recorded into distance using speed of sound.
- Decide if the distance calculated falls in 'safe', 'warning', or 'danger' zone.

*E. User application in slave gumstix (*slave_ul.c*)*

User application in slave gumstix reads the proc filesystem of the slave kernel module to get the obstacle distance. It then writes to /dev/rfcomm0. /dev/rfcomm0 is used for communication over bluetooth, which is explained below. We use *rfcomm* protocol to establish connection between the master and the slave gumstix boards. Once a connection is established, a device file name "/dev/rfcomm0" is created in both the gumstix. The user application running on the slave gumstix has to read the sensor updates from the slave kernel module's proc file and write to the device file.

A small snippet of code from our slave_ul.c is as follows:

```
pFile=open("/proc/sensor", O_RDONLY) // opens the proc file for read-only
rfcomm=open("/dev/rfcomm0", O_WRONLY) //opens the rfcomm file for write-only
read(pFile, word, 1) //reads 1 character from proc file
write(rfcomm, word, 1) //writes the character to rfcomm file
```

F. Bluetooth communication between 2 gumstix

We use *rfcomm* protocol to establish connection between the two gumstix boards. The rfcomm protocol consists of a simple set of transport protocols which provide the serial RS-232 emulation. We ensure that the bluetooth in both the gumstix boards are turned on and lists all nearby bluetooth devices in the master gumstix. We record the MAC ID of the slave gumstix and use it to establish a rfcomm connection between the 2 devices. We use the following set of commands[3]:

```
>> hcitool scan //Lists all bluetooth devices nearby
>> rfcomm -r connect 0 <Slave_MAC_ID> 1 &
```

This will create a /dev/rfcomm0 in both the gumstix and can be treated as a regular driver node which the user-level C code can perform file I/O operations on.

G. Audio communication over gumstix

For establishing an audio communication channel over gumstix, we downloaded and installed several packages. It is easier to copy and install all these packages in the SD card, and create a soft link to these libraries in the /usr/lib folder. The following steps provides a guide on establishing audio communication over gumstix [4],[5]:

1) Install all the packages in the SD card using the following command:
   >> ipkg -d external install libaudiofile0_0.2.6-r6_armv5te.ipk
   In some cases, it gives an error about not finding certain dependencies. Use *–force-depends* flag to ignore this as a warning.
2) Create a soft link for all the installed libraries in the folder /usr/lib using the following command
   >> cd /usr/lib
   >> ln -s /media/card/installed_packages/usr/lib/libaudiofile.so.0.0.2 libaudiofile.so.0
3) Symbolically link madplay and aplay using the following commands
   >> ln -s /media/card/installed_packages/usr/bin/aplay /usr/bin/aplay
   >> ln -s /media/card/installed_packages/usr/bin/madplay /usr/bin/madplay
4) Edit */etc/bluetooth/audio.service* to enable autostart to connect audio device on running *aplay*.
5) Turn on the bluetooth speaker. Run *hcitool scan* on gumstix to search for bluetooth devices nearby. Record the MAC address of the speaker.
6) Create a structure in the /etc/asound.conf with the bluetooth device name and it's MAC address as follows:
   pcm.jbl{
   type bluetooth
   device "AA:BB:CC:DD:EE:FF" }
7) Use passkey-agent to set up a default passkey, which is used to pair with the bluetooth device. Since most devices uses 0000 as the default passkey, we use this as our passkey.
   >> passkey-agent 0000 AA:BB:CC:DD:EE:FF &" using your MAC address
8) Finally use aplay to pair to the bluetooth device, and use madplay to send audio files via bluetooth. We can pipe the 2 commands as follows:
   >> madplay song.mp3 -r 44100 –output=wave:- | aplay -D jbl &

V. Challenges and Mistakes

This section gives description on some of the failed attempts we had while reaching the working final product. We will give the challenges as a list with brief descriptions while trying to keep the flow mainly in the direction we went as we completed the project.

1) We initially began with a more advanced sensor [6] this sensor gave multiple outputs one of which was a serial bit stream output. We decided to use this output with the gumstix using STUART protocol. This proved harder than we thought even when connected correctly we could not find how to read the serial registers and if we were getting an output from the sensor. The advanced sensor is more compact with more pins as can be seen: Figure 3
2) Moving into the second output type [6] this was an analog voltage output based on how far an object was. This output was easier to read with a multimeter and understand but there was a significant level of difficulty on how to actually read analog signals with the gumstix. We tried to use I2C for analog to digital but there was again difficulty in doing so.
3) We decided on implementing the analog to digital conversion manually. We used multiple Opamp and diodes to create a voltage window circuit that gave output high if the voltage was in a certain range. Even though the circuit worked there was a high overhead of wires, circuit elements and power added on with a fine resolution required for the number of windows. Deciding against this we went with a simpler sensor that was our final sensor and only had two pins TRIG and ECHO.

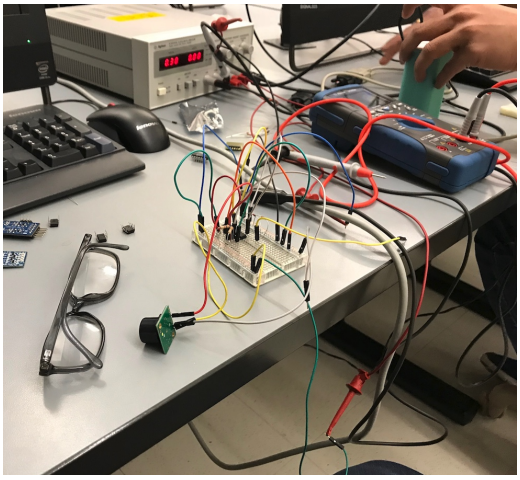

Fig. 3: Advanced Sensor

4) With this new sensor we initially thought voltage scale up was required. We could not get the kernel module to work using jiffies. We explored multiple options where the error could lie such as the sensor outputting a lower voltage than we expected. We scaled this up with an amplifier circuit using analog elements but found later on that there was no circuit problem and the resolution with jiffies was unreliable. Some of our testing was done using multimeters for electronic errors Figure: 4.

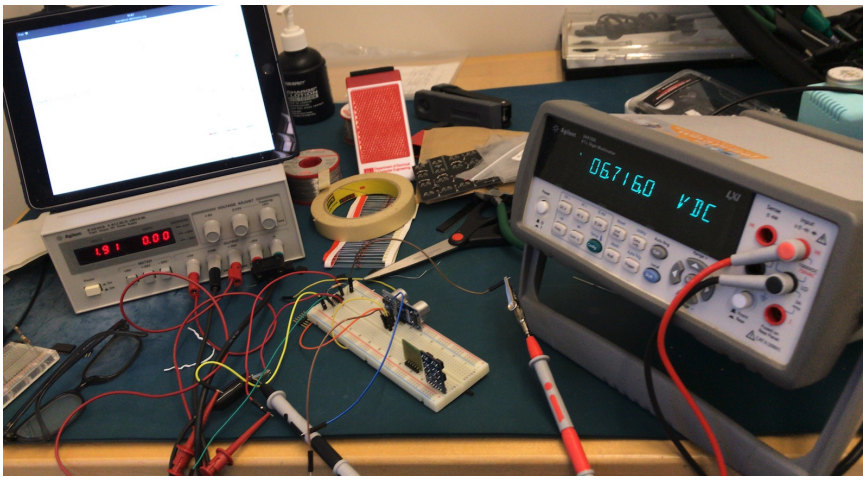

Fig. 4: Opamp Struggles

5) Moving into the next version of the kernel module and the sensor we finally got some output that was correct using timeofday() instead of jiffies. The module kept crashing using extensive while loops would give soft lock up. To get rid of most of the while loops we implemented timers and interrupts which would keep the kernel module on standby.
6) In the master gumstix we needed to implement two sensors together the initial idea was to use a single kernel module that would send one TRIG and receive two ECHO from the separate sensors this proved difficult because the sensors would be hungry for resources and would enter deadlock. Our final setup can be seen in Figure 5.
7) Separating the two sensors to two TRIG output still did not give the desired output this case would work sometimes but was highly dependent on the signal values. Sometimes the sensors would enter deadlock with no apparent reason. Thus our final implementation was using two of the same kernel

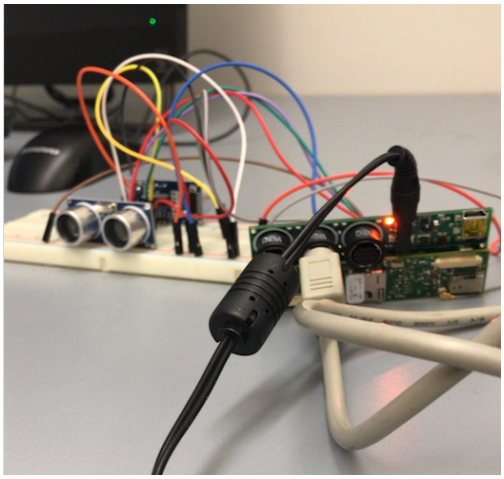

Fig. 5: Master Module

modules one for each sensor on the master gumstix. Along with an additional module for the foot slave module.

8) Packaging of the product could have been better. We did not have the time to instal the gumstix on the cap and the boot for the final demo.
9) The delay in bluetooth communication was investigated. However, we wonder if there was a fix that we were unable to implement.

VI. Conclusion

Using embedded systems we can help improve the lives of those less fortunate. We designed Smart Guide to help out visually impaired people using echolocation technology enabling them to have better lives.

Appendix A
Youtube Video Link

To see a demonstration of our project go to this YouTube Link.

Acknowledgment

Fig. 6: The Team

The authors would like to thank Prof. Coskun for her valuable class and her great effort to get us excited about the wide world of embedded systems and programming. We would also like to thank the wonderful TA Rushi Patel for being one of the most attentive TAs.

REFERENCES

[1] Hcsr04 specifications. http://www.electroschematics.com/8902/hc-sr04-datasheet/. Accessed: 2017.
[2] Hc-sr04 ultrasonic range sensor on the raspberry pi. https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi. Accessed: 2017.
[3] Gumstix bluetooth guide. https://www.ece.cmu.edu/~ee349/bluetooth.pdf. Accessed: 2017.
[4] High fidelity audio in gumstix using bluetooth. https://wiki.gumstix.com/index.php/Category:How_to_-_bluetooth#High-fidelity_audio. Accessed: 2017.
[5] Basic-bluetooth-audio-with-lcd-controls. https://github.com/saribe0/Basic-Bluetooth-Audio-with-LCD-Controls. Accessed: 2017.
[6] Mb1040 lv sensor user manual. https://www.maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf. Accessed: 2017.