# Voice-activated Robotic Kart

**Team BlueGum**

**Juan Jhong Chung (jfjhong@bu.edu)**

**Laura Kamfonik(kamfonik@bu.edu)**

**Ankit Sharma (ankitsh@bu.edu)**

***Abstract***

*The goal of this project was to implement a robot kart that can be controlled using voice commands and leveraging the embedded systems knowledge learned in class. We accomplished a final solution that made used of the Raspberry Pi microcomputer and Jasper, an open source Natural Language UI, as well as the Gumstix Verdex, and a kart chassis with DC motors. We successfully implemented a set of voice commands to control the movement of a kart via Bluetooth communication.*

**1. Introduction**

This project revolves around integrating Natural Language User Interface (NLUI) software to automate the movement of hardware. Human Machine Interface is a field in Computer Science that studies the interaction of computers with humans. From the traditional keyboard and mouse, to the now popular touchscreens, the field of human machine interface is always exploring the human senses as means to interact with computers[1]. The popularity of artificial personal assistants like Apple's Siri has increased in recent years[2]. Two major high technology companies recently released solutions that make use of speech as a way to control a machine. Microsoft has developed "Cortana"[3], an assistant for its line of Windows Phone smartphones. Amazon has developed a hardware solution called "Echo"[4], a living room media center and personal assistant.

Our project makes use of Jasper[5], an open source personal assistant, to control the movement of a robotic kart. The main motivation of this project is helping people with physical disabilities interact more easily with the world leveraging the popularity of AI and speech control software. Our solution is divided in two main parts: the Speech Processing module, and the Robotic Kart module. These modules are integrated using Bluetooth technology. The Speech Processing component includes a Raspberry Pi connected to a USB microphone, and self powered speakers, running a modified version of Jasper that makes use of a USB Bluetooth dongle to send commands to our Robotic Kart. Our Robotic Kart component consists of a plastic kart chassis with two wheels carrying the Gumstix Verdex Pro running a user application that receives messages sent via Bluetooth. These messages are copied to a kernel module that controls two motors using an H-bridge chip.

We accomplished to operate the Robotic Kart using Jasper via Bluetooth with the following commands:

Move “Forward,” “Reverse,” “Stop,” Turn “Left,” Turn “Right.” Additionally we managed to implement other commands “Fast” Mode and “Slow” Mode using PWM, Set Movement Durations: “Five” and “Ten” seconds, and “Dance.”

**2. Design Flow**

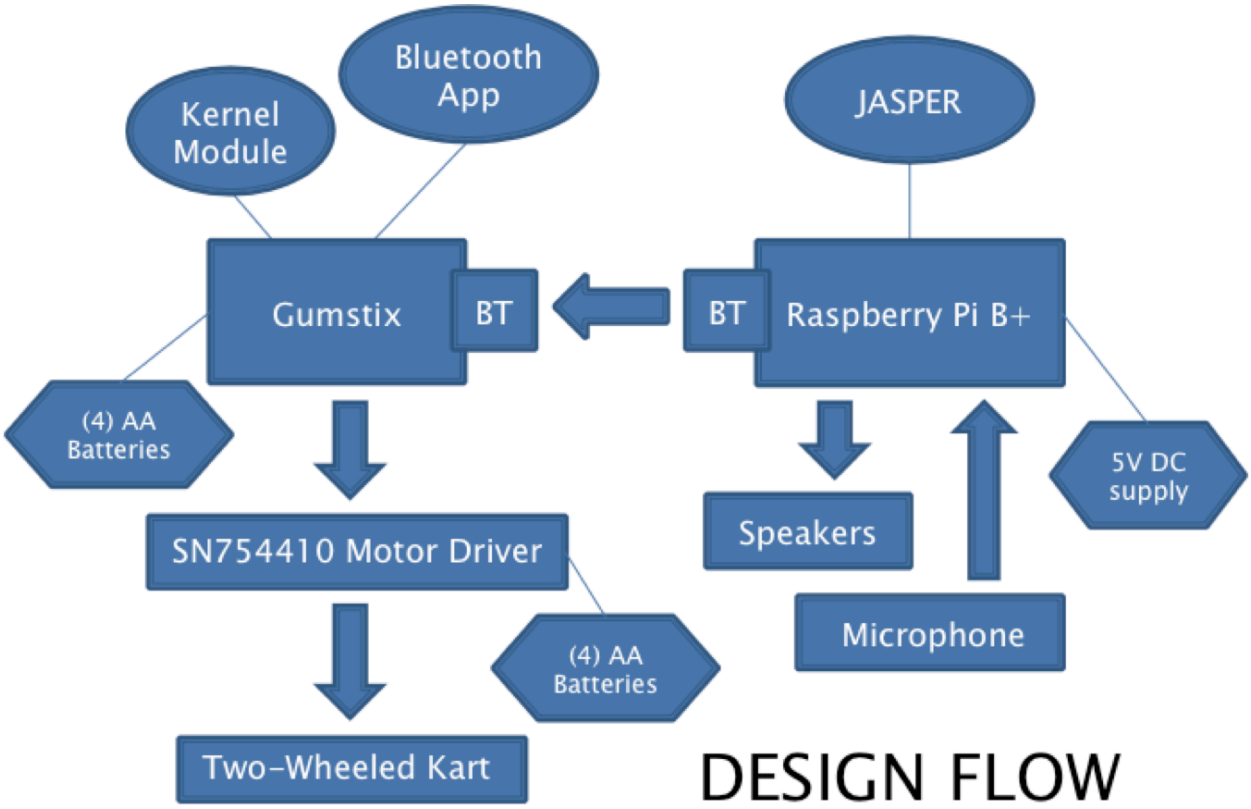



**Figure 1. Design Flow**

The Speech Processing component was implemented in the Raspberry Pi using Jasper’s APIs written in Python to extend the functionality to control our kart. Commands spoken by the user are parsed by Jasper and passed over a Bluetooth connection to the Gumstix board. Voice commands are input via a USB microphone, and a small speaker allows the user to hear Jasper's responses. A USB dongle enables the Bluetooth connection to the Gumstix, and an additional Bluetooth to serial chip allows us to control the Pi using any Bluetooth enabled computer with a serial client. A Bluetooth enabled laptop is helpful for debugging purposes, but the whole system can run independent from a laptop or desktop computer.

The kart is a small two-wheeled cart propelled by a pair of DC motors and controlled using an SN754410 H-Bridge driver IC[6], which is in turn controlled by four GPIO pins and one PWM pin on the Gumstix

board. A kernel module on the Gumstix controls these pins. In turn, a user-level application handles the Bluetooth code and passes commands to the kernel via the proc file system.

The system is extremely portable. The kart power is self-contained. Four AA batteries power the kart motors and driver IC. A second set of 4 AA batteries power the Gumstix. We were able to setup a connection to the Gumstix over Bluetooth from a Linux laptop to debug and to send/receive files, completely eliminating the need for the serial cable.

Among the team members, Juan was primarily responsible for the Bluetooth Setup and networking code as well as installing and configuring Jasper on the Raspberry Pi. Laura wrote most of the Gumstix kernel module as well as some of the customized Jasper code. Ankit worked mainly on hardware, including the choice of kart and motor driver, assembling the kart, and performing electrical tests and calculations in order to operate the system safely under self-contained power sources. He also took on the filming and editing of our project video. We met many times as a group to help each other with debugging and talk through the difficulties, so each person's primary task also contains contributions from the other team members.

## 3. Project Details

### a. Raspberry Pi Model B+

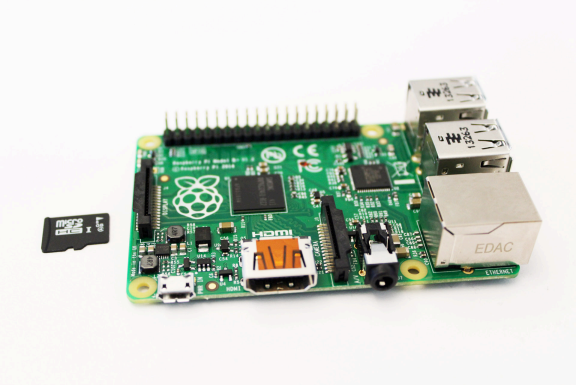



**Figure 2. Raspberry Pi Model B+**

To setup the Raspberry Pi, we chose to flash a 16GB Class 10 MicroSD card with the latest Raspbian[7] image (09/14/14) that uses the Linux Kernel 3.12. This image contains all the basic drivers and

configuration to use the Raspberry Pi out of the box. Additionally, we installed Alsa mixer drivers for the USB microphone (Kinobo), and the BlueZ v4 libraries and tools for our USB Bluetooth Dongle (AsusTek B404). To install Jasper we followed the manual installation instructions available in []. The two most common output interfaces of the Raspberry Pi are HDMI and a Network Interface Card. However, we decided to connect the serial pins of the Raspberry Pi to the VirtuaBotix BT2S[8] Bluetooth-to-serial adaptor to help with portability and debugging.

**b. Customized Jasper Module**

To configure Jasper to recognize our kart commands and send them to the Gumstix, we wrote a small module called *Forward.py* using Jasper APIs[9] and placed it into Jasper's modules folder. Jasper determines whether input is a valid command using the function:

```
def isValid(text):

  return bool(re.search(r'\bForward,Reverse,Stop,Left,Right\b', text, re.IGNORECASE))
```

which appears in all modules. This tells the main Jasper processes which words to recognize as valid input. If the user says one of these words, it will activate the module containing it. These words must also be added to a WORDS list in the file, as follows:

```
WORDS = ["FORWARD", "REVERSE", "STOP", “LEFT”, “RIGHT”]
```

These words are added to Jasper's active vocabulary when the program is started, and the screen output shows that the words are recognized from speech at all times. However, Jasper will only activate a module if the corresponding words are spoken while it is actively listening for a new command. This happens when the user says "Jasper", and Jasper responds with a high beep. The user then speaks a command, and Jasper responds with a low beep when it finishes listening. We found that using multiple similar words as commands could confuse the speech-to-text engine; "backward" and "forward" were frequently mixed up, so we changed the command for going backwards to "BACK". We then discovered that certain sounds (hard C or T sounds in particular) are extremely common in background noise. Since "back" is a short word with a very hard C sound, Jasper would frequently detect a similar sound somewhere in the background before we could speak the intended command. To adapt, we changed the backwards command to "REVERSE", which works much more consistently.

When a module is activated, it has available to it the string *text* which contains the output of Jasper's speech-to-text engine. If the module is activated, *text* will always contain one of the command words, but it can also contain other words that Jasper picked up from background noise. Our code searches through the text string and looks for the first valid command word. It then converts that command word to a number corresponding to the word's index in the WORDS list. It is important that the numbers also be mapped correctly to commands in the Gumstix code, or the kart will not function correctly.

### a. Bluetooth Modules


**Figure 4 Asus BT404**

In the Raspberry Pi, we created a file called *BlueGlobals.py* located in */home/pi/jasper/client/modules*. It contains all global variables and functions needed to initialize a Bluetooth link between the Raspberry Pi and the Gumstix. The code starts by creating a Bluetooth socket using the L2CAP and port 0x1001 to transmit our data. This socket also contains the address it will attempt to connect to. In our case we have hardcoded the Gumstix Bluetooth address. Forward.py makes use of the functions in BlueGlobals.py. It uses a flag to check if a Bluetooth socket has been created or not. Then it uses the global socket BlueGlobals.py to send a number value (as a string) to the Gumstix using the “send” function.

In the Gumstix, there is a user level application called l2cap-server that must be initialized after boot. It is written in C and contain all the basic setup to create a Bluetooth “server” waiting on an incoming connection using the L2CAP protocol and port 0x1001. Similar to the Bluetooth code in the Raspberry Pi, we have to create a socket, specifying the protocol and port number, and start a listening service on this port using the function “listen.” This application is always waiting for a connection, and when it receives a message, it decodes it into a string a uses that same string to write to */proc/robokart*. From then on, the Kart device driver takes care of executing the appropriate commands based on the string received.

**b. Kart Device Driver**

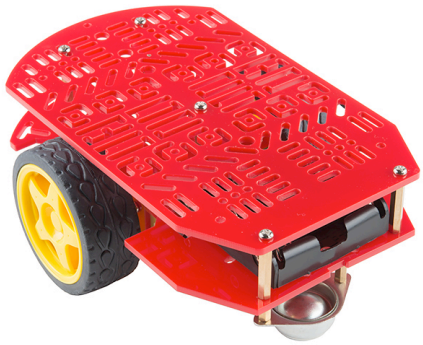


**Figure 3 Sparkfun Robotic Kart Chassis**

The Gumstix device driver receives and sends messages from a user-space application via file operations to the file */proc/robokart*. When a command number is received by the *proc_write()* function in the module, the corresponding movement function is called via a switch statement. Jasper's movements are primarily controlled by two functions, *moveKart()* and *turnKart()*. Each function turns on the appropriate GPIO pins, then sets a timer. When the timer expires, it calls the function *stopKart()*, which turns off all four GPIO pins. Forward and backward movements are handled by *moveKart(),* and turning motions are handled by *turnKart()*. By default *moveKart()* moves the kart for 5 seconds, but the user can change the move time to tens seconds and back, which is implemented by reassigning the variable *move_time* in the switch statement itself.

The SN754410 motor driver includes two enable inputs, each controlling a pair of the driver outputs. We were able to implement two different speed settings by using PWM on the enable signal. The PWM settings are established when the kernel module is initialized, but PWM is disabled, allowing the kart to default to the "Fast" speed. Invoking the "Slow" command calls *changeSpeed()* and enables PWM, while invoking "Fast" disables it again.

The user can also stop the kart manually with the command "Stop", which calls a function that is a duplicate of *stopKart()*. A separate function is required because we found that calling *stopKart()* manually in the module code interferes with timer operation and sometimes causes kernel panic. The "stop" implementation is difficult to demonstrate with shorter move times; because of the signal delay between speaking the command and the kart implementing the command, the kart generally has

already stopped itself by the time the command is received. The command works as expected when a longer move time is used.

The final command, "Dance", is implemented in the function *danceKart(),* which demonstrates the possiblity of hard coding a relatively complex series of actions and tying them to a single command. The function turns the GPIO pins on and off in a specific pattern, using *msleep()* to determine the move time for individual segments.

The switch statement in *proc_write()* performs one additional function; it copies a feedback message corresponding to the movement invoked into a message buffer, which is then sent to user-space by *proc_read()* when the user-level application reads */proc/robokart*. The feedback provided by the current set of commands is very simple, but this functionality could allow a more complex kart to send new information to user-space. For example, if the kart were equipped with a sensor, it could send sensor information back to user-space.

### c. Hardware & Circuitry

*Robot Kart:* We bought the robot kart from [10]. The kart comes with 2 levels, one to hold the battery compartment with the DC motors screwed under it. The upper level holds the breadboard and the rest of the circuitry. The DC motors can run on 4.5V DC, with a no load current of 190mA. For driving the DC motors, we use an H-bridge motor driver SN754410. It can take 4.5V – 36V. It is a 16 pin package and can implement both Full and Half bridge Motor driver. It can drive a max. of 1 A current.

*Gumstix Verdex Pro:* We were given this board for the class, and this board houses the Kart software. It uses Xscale processor running at 600MHz, multiple I/O pins, and comes with a microSD card slot, a Bluetooth module which can be used with an antenna.

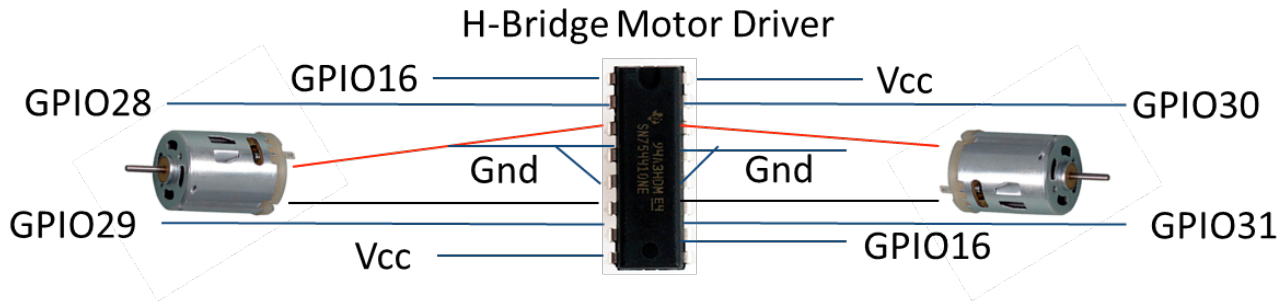
*Raspberry Pi Model B+ :* The Pi houses the open source voice processing. It has Broadcom BCM2835 CPU with 512 MB RAM.

*Kinobo Microphone:* The microphone is the input for the voice processing and is used to give the commands.

*XBOOM speaker:* The speaker plays feedback from the voice software.

*AA Batteries:* A total of 8 batteries power the Gumstix Board and the H-bridge motor driver.

Wiring Diagram:




In this diagram, GPIO16 is a PWM pin, and GPIO28,29,30,31 are I/O pins in the Gumstix[11].

## 4. Summary

**We successfully accomplished a functional kart that can be controlled using voice commands via Bluetooth. We believe that Voice control will continue to be a popular choice for Human-Machine interaction.**

**In the future we would like to explore:**

- **How to increase Jaspers speech recognition accuracy under a noisy environment.**
- **Add additional sensors to the kart: a proximity detector to avoid collisions.**
- **Implement an algorithm to provide feedback about the location of the kart.**
- **Implement 2-way Bluetooth communication to send feedback to the Raspberry Pi.**
- **Reusing our code to retrofit a wheelchair to assist physically disable people control it with voice commands.**

**References**

[1] A.P Breen “Issue in the Development of an Intelligent Human-Machine Interface”, in AAAI Technical Report, February 1998.

[2] Over one-half of Apple iPhone 4S users satisfied with Siri and want Siri-style voice command for TV. http://www.parksassociates.com/blog/article/parks-pr2012-siri

[3] Meet Cortana for Windows Phone. http://www.windowsphone.com/en-us/how-to/wp8/cortana/meet-cortana

[4] Introducing Amazon Echo. http://www.amazon.com/oc/echo/ref_=ods_dp_ae

[5] Jasper Project. https://jasperproject.github.io/

[6] SN774410 H-Bridge Driver. https://www.sparkfun.com/products/315

[7] Raspberry Pi Downloads. http://www.raspberrypi.org/downloads/

[8] VirtuaBotix BT2S https://www.virtuabotix.com/product/bt2s-bluetooth-serial-slave-arduino-versalino-microcontrollers/

[9] How to create a Standard Jasper Module. http://jasperproject.github.io/documentation/api/standard/

[10] SparkFun. https://www.sparkfun.com

[11] Gumstix Header. https://pubs.gumstix.com/boards/CONSOLE/VX/PCB10003-R1753/PCB10003_MULTI-HEADERS.png

[12] Gumstix Pins. https://pubs.gumstix.com/boards/CONSOLE/VX/PCB10003-R1753/PCB10003_HIROSE60.png