Speech Controlled Motor Cart

Fengmao Zheng, *fengmao2,* ECE 110

Wenxian Zhang, *wzhan103,* ECE 120

Introduction

a. Statement of Purpose

Our purpose of project is to make a sound driven motor that can follow the audio instruction to move in expected direction. For this purpose, our design has two main components: a sound converter that receives audio instructions and transfers the sound to electric signals and instruction information and a motor that will receive the instruction from sound converter and follow this instruction to move (forward, back, left and right).

All the major tech companies — Apple, Google, Microsoft, and Amazon, etc.— are pouring incredible amount of resources into their new voice assistants, such as Siri or Alexa. The grant amount of resource resulted a huge wave of obsession with using oral voice command to electronic devices. According to Business Insiders, only 2% of iPhone owners have never used Siri. [1] Sound recognition now becomes a project that is necessary in an electronic device; and these projects inspired our ideas. However, Despite the popularity, lots of people do not use sound recognition system maybe because of embarrassment in speaking to their device, unfamiliarity of the command, or the belief that button or password control is good enough for their lives. In order to solve these difficulties that users faced, we do not only aim to use sound recognition in software or virtual system, but also aim to utilize it on a more regular basis, such as using oral command to lock a bike or to drive remote controlled cars.

We want to make a thing that will follow instructions by sound which is the most convenient. In order to accomplish this project, we do research about similar projects and how they have been made.

b. Features and Benefits

Password Recognition: This device is able to set a five digit password in combination of 0s and 1s. This feature strengthen the security of the speech controlled car.

Speaker Recognition: This device will recognize specific speaker and only listen to the instructions given by that speaker. This is designed by recording the acoustic feature from the speaker and verify these features in the audio command.

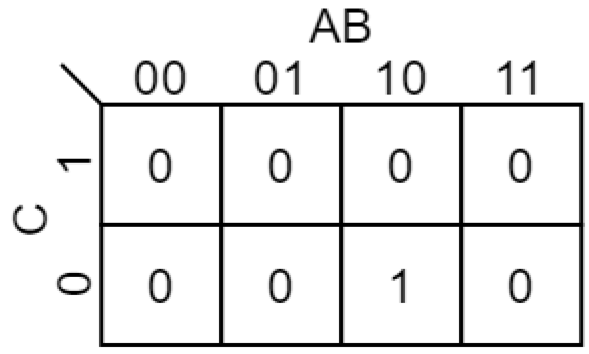
Design

a. System Overview

The system has three sections: Voice Recorders, Voice Analyzer and Motor. Voice Recorders will send the sound wave recorded from the environment to the Analyzer. The Analyzer will first amplify the sound wave and then compare to the sample sound wave and send signal to the motors with the angle, rate and direction of rotation. There are two key components in voice recognition: Speech Recognition and Speaker Recognition. Only if the sound wave produced match the phrase and the acoustic feature of the person, and the logical gates pass the key(which is 1 0 0) the circuit will send 1 to drive the motor; otherwise the circuit will send 0 to the motor which won't drive the motor. As for the past of logical gate, we have a swipe with a sequence of "1" and "0" so that there will be a password in combination of "1"s and "0"s. The password is 1 0 0. Only when the password is correct, we will output "1" that we can open a part of the key. After the system obtains and verify the voice input, the power supply will input “1” , open the second part of the key and give out a signal with exact "lock" or "unlock" command as programmed.

b. Design Details

Password Key Truth Table:



C \ AB	00	01	10	11
1	0	0	0	0
0	0	0	1	0

This is the truth table for password input. We set the key as 1 0 0 and if the switch is on 100, which means A = 1, B = 0, C = 0, the current will flow. If not, the current will not from the power supply because of the block of logical gates. In the part of the key design, we use three inputs ABC matched with S0,S1,S2 in the DIP switch. Only when F(output) = AB’C’, the output will be 1 and the current will flow. And we use De Morgan’s law to achieve this design with inverter, NAND and NOR gate.

For motor driving circuit, we used arduino redboard as a PMW power source and connected the power source to the motor via a BJT transistor.

Results

For the device we had a microphone as signal input. We performed multiple tests on the device. First time with a green LED light on pin 9 and a red LED light on pin 11. We designed a simple command to turn the green light on and off and then turn the red light on and off. We successfully demonstrated on the first demo day with 2 or 3 tries on each command. After we built the final prototype for the motor cart, we tested it many times with the original microphone. It had a success rate at about 40%. However, we thought that the original microphone is too closely attached to the cart, so we decided to extend the wire by ironing an extension cord onto the original microphone. The result is disastrous. The new microphone had an success rate of lower than 10% and it broke many times that finally yielded us to use new microphone which decreased the success rate even more.

Problems and Challenges

a, Problems:

When connecting the shield to PC, PC won't recognize the port COM 3 and keeps asking for bridge program update while the bridge program has updated to the latest version.

Need to learn which port on the shield needs to connect to the breadboard and the schematics of the shield.

Need a very sensitive microphone in order to better recognize the input.

b. Challenges

a.) People may feel embarrassed using sound system in public.

b.) Applying these motors on vehicles may need a lot of budgets supported.

c.) Have to use a wireless microphone instead of speaking to a wired microphone.

References

1. K. Leswing, "Here's why people don't use Siri regularly, even though 98% of iPhone users have tried it," in *Business Insider*, Business Insider, 2016. [Online]. Available:

http://www.businessinsider.com/98-of-iphone-users-have-tried-siri-but-most-dont-use-it-regularly-2016-6. Accessed: Sep. 19, 2016.

2. B. Clark, "Study: Most iPhone owners are too embarrassed to use Siri in public," The Next Web, 2016. [Online]. Available:http://thenextweb.com/insider/2016/06/06/study-most-iphone-owners-are-too-embarrassed-to-use-siri-in-public/#gref. Accessed: Sep. 19, 2016.

3.

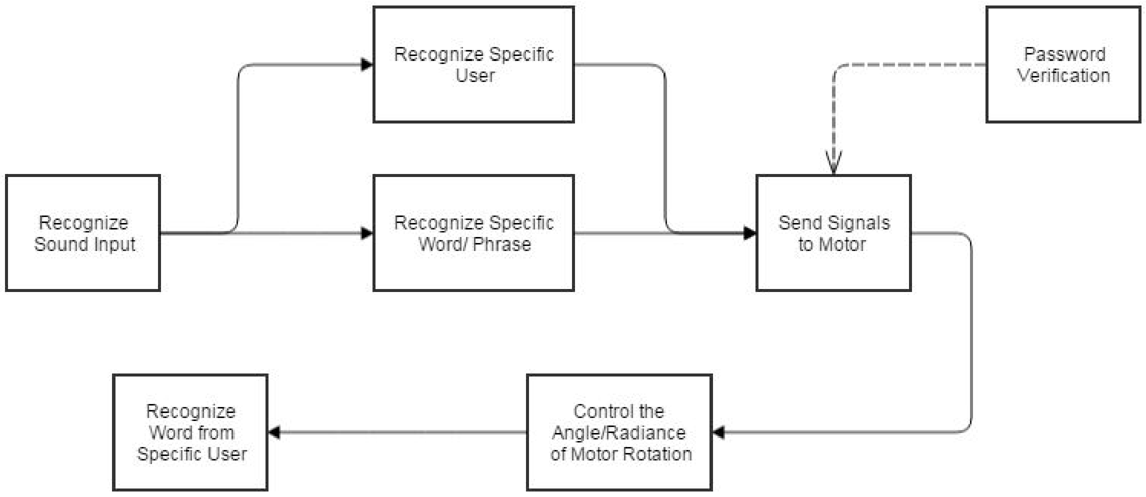
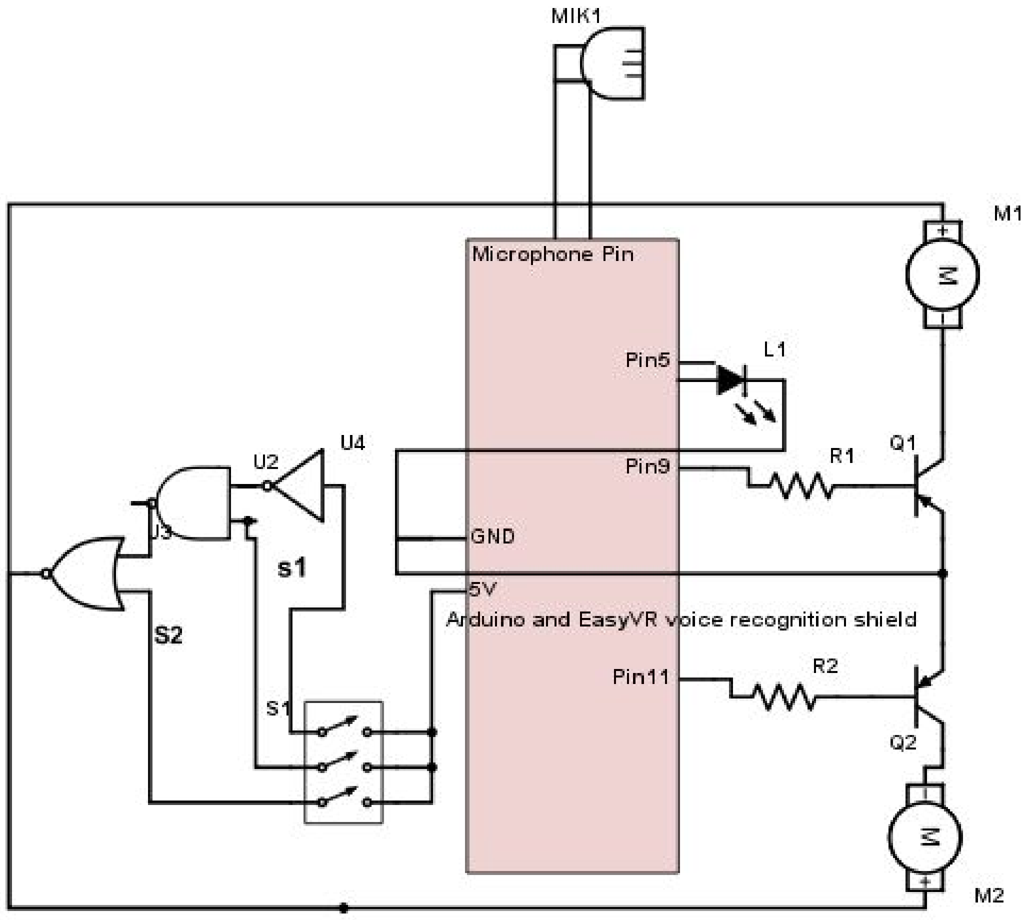

Recognize Specific User
Password Verification
Recognize Sound Input
Recognize Specific Word/ Phrase
Send Signals to Motor
Recognize Word from Specific User
Control the Angle/Radiance of Motor Rotation

Diagram 1: Block Diagram of Design



4.

Diagram 2: Schematics of design

5. Arduino Code:

```
#include "Arduino.h"
#if !defined(SERIAL_PORT_MONITOR)
  #error "Arduino version not supported. Please update your IDE to the latest version."
#endif

#if defined(SERIAL_PORT_USBVIRTUAL)
  // Shield Jumper on HW (for Leonardo and Due)
  #define port SERIAL_PORT_HARDWARE
  #define pcSerial SERIAL_PORT_USBVIRTUAL
#else
  // Shield Jumper on SW (using pins 12/13 or 8/9 as RX/TX)
  #include "SoftwareSerial.h"
  SoftwareSerial port(12, 13);
  #define pcSerial SERIAL_PORT_MONITOR
#endif

#include "EasyVR.h"

EasyVR easyvr(port);

//Groups and Commands
enum Groups
{
  GROUP_0  = 0,
  GROUP_1  = 1,
};

enum Group0
{
  G0_READY = 0,
};

enum Group1
{
  G1_GO = 0,
  G1_STOP = 1,
  G1_SPEED_UP = 2,
  G1_SLOW_DOWN = 3,
  G1_TURN_LEFT = 4,
  G1_TURN_RIGHT = 5,
  G1_TURN_OFF = 6,
};


int8_t group, idx;

void setup()
{
  // setup PC serial port
  pinMode(9,OUTPUT);
  pinMode(11,OUTPUT);
  pcSerial.begin(9600);

  // bridge mode?
  int mode = easyvr.bridgeRequested(pcSerial);
  switch (mode)
  {
  case EasyVR::BRIDGE_NONE:
    // setup EasyVR serial port
    port.begin(9600);
    // run normally
    pcSerial.println(F("---"));
    pcSerial.println(F("Bridge not started!"));
    break;

  case EasyVR::BRIDGE_NORMAL:
    // setup EasyVR serial port (low speed)
    port.begin(9600);
    // soft-connect the two serial ports (PC and EasyVR)
    easyvr.bridgeLoop(pcSerial);
    // resume normally if aborted
    pcSerial.println(F("---"));
    pcSerial.println(F("Bridge connection aborted!"));
    break;

  case EasyVR::BRIDGE_BOOT:
    // setup EasyVR serial port (high speed)
    port.begin(115200);
    // soft-connect the two serial ports (PC and EasyVR)
    easyvr.bridgeLoop(pcSerial);
    // resume normally if aborted
    pcSerial.println(F("---"));
    pcSerial.println(F("Bridge connection aborted!"));
    break;
  }

  while (!easyvr.detect())
  {
    Serial.println("EasyVR not detected!");
```

```
    delay(1000);
  }

  easyvr.setPinOutput(EasyVR::IO1, LOW);
  Serial.println("EasyVR detected!");
  easyvr.setTimeout(5);
  easyvr.setLanguage(0);

  group = EasyVR::TRIGGER; //<-- start group (customize)
}

void action();

void loop()
{
  if (easyvr.getID() < EasyVR::EASYVR3)
    easyvr.setPinOutput(EasyVR::IO1, HIGH); // LED on (listening)

  Serial.print("Say a command in Group ");
  Serial.println(group);
  easyvr.recognizeCommand(group);

  do
  {
    // can do some processing while waiting for a spoken command
  }
  while (!easyvr.hasFinished());

  if (easyvr.getID() < EasyVR::EASYVR3)
    easyvr.setPinOutput(EasyVR::IO1, LOW); // LED off

  idx = easyvr.getWord();
  if (idx >= 0)
  {
    // built-in trigger (ROBOT)
    group = GROUP_1; //<-- jump to another group X
    return;
  }
  idx = easyvr.getCommand();
  if (idx >= 0)
  {
    // print debug message
    uint8_t train = 0;
    char name[32];
    Serial.print("Command: ");
    Serial.print(idx);
    if (easyvr.dumpCommand(group, idx, name, train))
    {
      Serial.print(" = ");
      Serial.println(name);
    }
    else
      Serial.println();
          // beep
    easyvr.playSound(0, EasyVR::VOL_FULL);
    // perform some action
    action();
  }
  else // errors or timeout
  {
    if (easyvr.isTimeout())
      Serial.println("Timed out, try again...");
    int16_t err = easyvr.getError();
    if (err >= 0)
    {
      Serial.print("Error ");
      Serial.println(err, HEX);
    }
  }
}

void action()
{
    switch (group)
    {
    case GROUP_0:
      switch (idx)
      {
      case G0_READY:
        // write your action code here
        group = GROUP_1; //<-- or jump to another group X for composite commands
        break;
      }
      break;
    case GROUP_1:
```

```
switch (idx)
{
case G1_GO:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
   analogWrite(9,150);
   analogWrite(11,150);
  break;
case G1_STOP:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
  analogWrite(9,0);
  analogWrite(11,0);
  break;
case G1_SPEED_UP:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
  analogWrite(9,225);
  analogWrite(11,225);
  break;
case G1_SLOW_DOWN:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
  analogWrite(9,110);
  analogWrite(11,110);
  break;
case G1_TURN_LEFT:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
  unsigned long timeStart;
  timeStart = millis();
  unsigned long timeEnd;
  timeEnd = timeStart;
  while((timeEnd - timeStart) <= 3000){
    analogWrite(9,0);
    analogWrite(11,225);
    timeEnd = millis();
  }
  break;
case G1_TURN_RIGHT:
  // write your action code here
  // group = GROUP_X; <-- or jump to another group X for composite commands
  unsigned long timeStart1;
```

```
        timeStart1 = millis();
        unsigned long timeEnd1;
        timeEnd1 = timeStart1;
        while((timeEnd1 - timeStart1) <= 3000){
          analogWrite(9,225);
          analogWrite(11,0);
          timeEnd1 = millis();
        }
        break;
      case G1_TURN_OFF:
        // write your action code here
        analogWrite(9,0);
        analogWrite(11,0);
        group = GROUP_0; //<-- or jump to another group X for composite commands
        break;
      }
      break;
    }
}
```