Voice Controlled Robot using Arduino

Instructor: Dr. V. Këpuska

Speech Recognition

ECE 5526

Submitted by:

Karan Baxi

Miteshkumar Patel

Naincy Desai

1. Objective:

This project aims to control the robot using speech recognition, in this project we only work with DC motor to create movement. For more specified movements of robot we can use steeper motor and servo motor. BitVoicer Server makes speech recognition very convenient.

2. Introduction:

In today's world, speech recognition plays a key role for interaction between human being and robots. People can communicate through speech. Throughout our lives, we communication each other with speech which are our born skills and we develop this ability during our early childhood. Speech plays a vital role in interacting with different systems as there is development in communication technologies. Speech is the most convenient way to interact with computers rather than using complex interfaces. In this project, we are controlling a robot with speech commands. The spoken commands are recognized by robot to do specific functions. Using a microphone, the voice commands are send to the computer and the robot does its movement per it. The command is then recognized by the computer using speech recognition system and then voice command is converted to direction command that are predefined by robot. So the robot moves on spoken command whenever it gets the direction command.

3. Tool and Programs:

To implement this method and get best result, we should have these tools and programs

1. Download BitVoicer Server: first step we need to download BitVoicer Server 1.0, it is a speech recognition and synthesis server for speech automation. It was developed to enable simple devices, with low processing power, to become voice-operated. We will use this application for record our speech or sound.
2. Download Arduino IDE: download Arduino IDE 1.8.2 version, it is open source software. It supports the language C and C++ using special rules of code structuring. It is used for more projects from beginner to expert programmer.

3. Microphone: prepare a good type of microphone to help you record your voice with good quality.

4. Work implementation:

In general we will record our voice or speech by using microphone and BitVoicer Server. Here using Arduino Uno is heart of the system and we mounted Polou Duel MC 33926 motor driver shield, and on top of that we mounted xbee shield, and on top we used RN171VX wifi module, the next important is microphone to capture sound, the circuit diagram is as shown below.

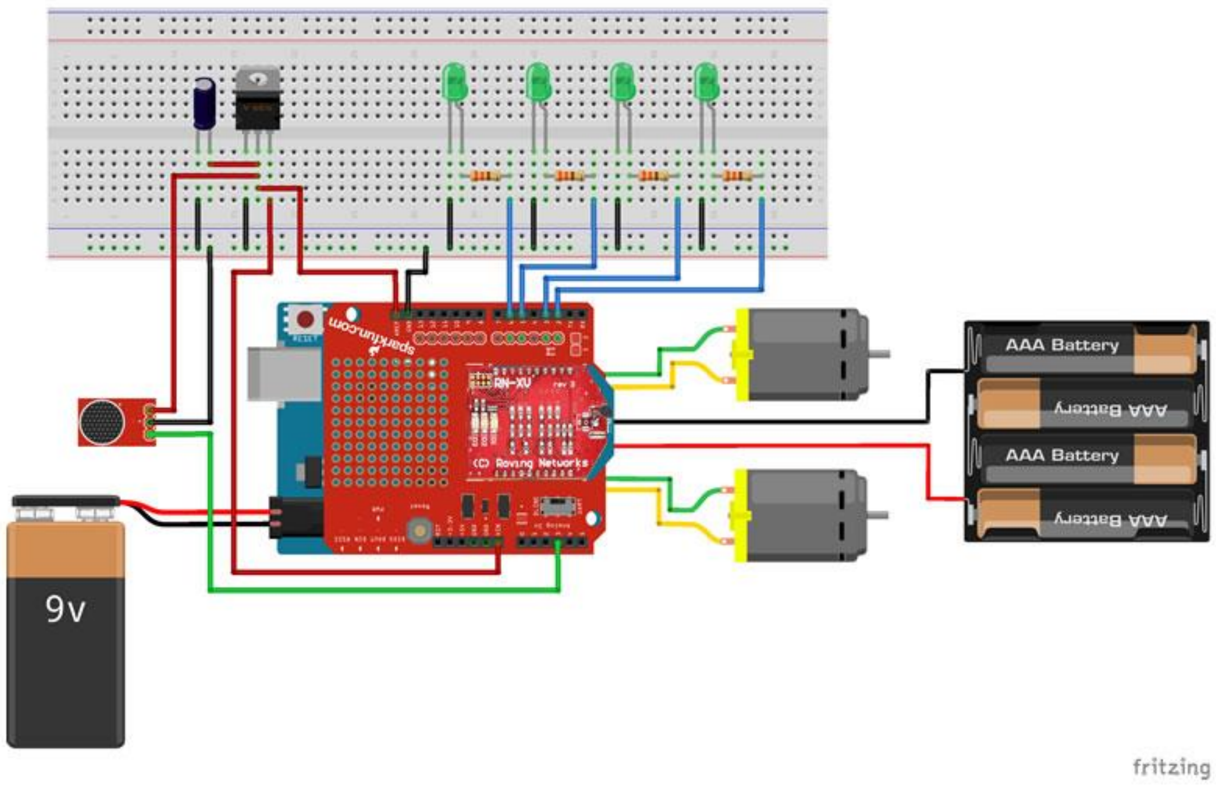


Fig. 1: Circuit Diagram of Robot

We have few LEDs that tell the status of some BitVoicer Server functionalities. As shown in previous circuit diagram, from the left to the right, the following information is exposed by the LEDs:

- Indicates whether BitVoicer Server is running and/or if the connection remains active;
- Indicates whether the data forwarding service is running;
- Indicates whether a speech recognition engine has been assigned to the Arduino;
- Indicates whether we are in the activation-word-activated-period. This LED lights up only when the activation word is identified.

5. Steps of work:

To implement this work and get the best results, follow these steps one by one:

Step 1: List of Component

In this project we need more component as shown below,

- Arduino UNO
- Pololu Dual MC33926 Motor Driver Shield
- SparkFun XBee Shield
- Microchip RN171VX Module w/ Antenna
- SparkFun Electret Microphone Breakout
- BitVoicer Server 1.0
- 2WD Robot Car Chassis
- Texas Instruments LM1117 (TO-220) Voltage Regulator
- 10μF Electrolytic Capacitor
- 4 x LEDs
- 4 x 330 Ohm Resistors
- 4 x 1.5V AA Batteries
- 9V Battery
- 9V to Barrel Jack Adapter
- Jumper Wires and Regular Wires
- Soldering Iron and Solder

Step 2: Mounting

In this step we mount all of the component mounted and crate the prototype of robot.

Step 3: Setting up Wi-Fi Module

The microchip RN171VX Wi-Fi module is totally operated through serial port only. This can be easy to configure the Wi-Fi module. All command are simply sent through Arduino Serial Port. We just have to send “$$$” to Arduino for entering in command mode and we need to write “exit” to return in data mode.

The most difficult task was to configure Wi-Fi module using Arduino, because whenever we want to upload code to Arduino we must set the switch of Xbee shild in DLINE position after, uploading the code we need to change the position of switch to UART.

Source Code:

```
void setup()

{

  Serial.begin(115200);

  pinMode(13, OUTPUT);

  delay(5000);

  Serial.print("$$$");

  delay(1000);

  Serial.println("set wlan auth 4");

  delay(1000);

  Serial.println("set wlan phrase dubaramatpuchna");

  delay(1000);

  Serial.println("set wlan ssid Don$chandu");

  delay(1000);

  Serial.println("set wlan channel 0");
```

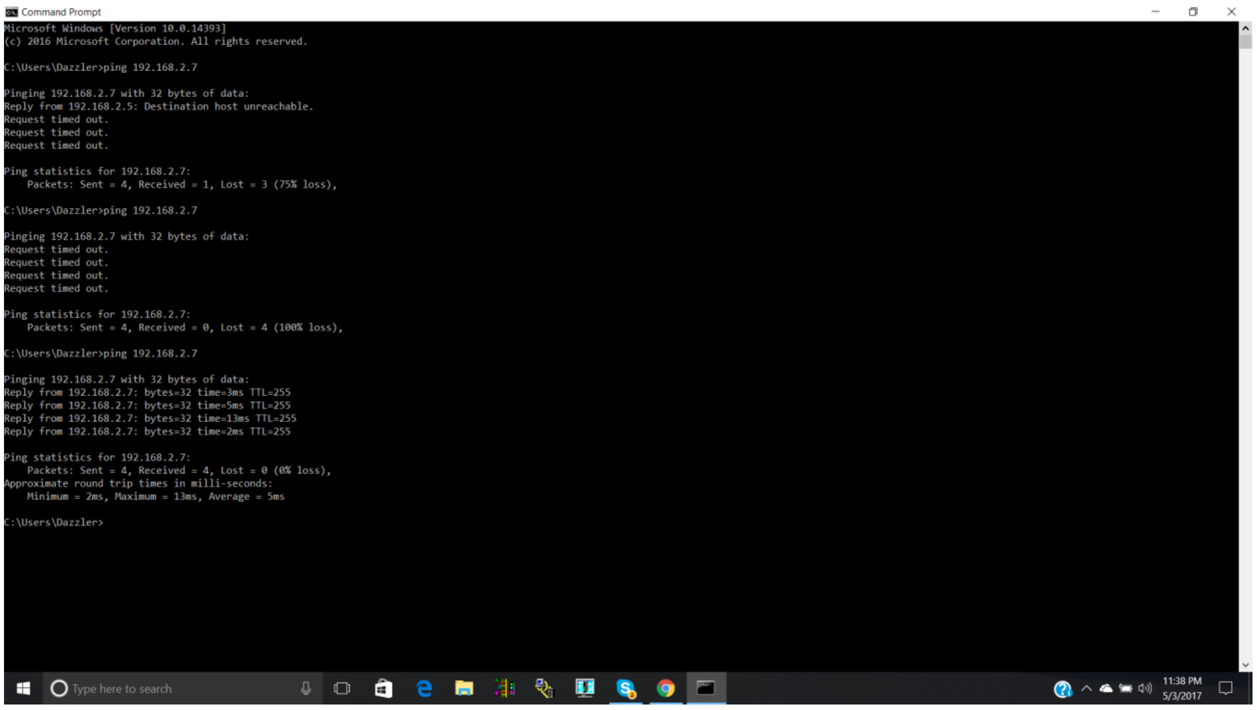
```
delay(1000);

Serial.println("set wlan join 1");

delay(1000);

Serial.println("set wlan tx 0");

delay(1000);

Serial.println("set ip dhcp 0");

delay(1000);

Serial.println("set ip address 192.168.2.7");

delay(1000);

Serial.println("set comm remote 0");

delay(1000);

Serial.println("set comm close 0");

delay(1000);

Serial.println("set comm open 0");

delay(1000);

Serial.println("set comm size 500");

delay(1000);

Serial.println("set comm time 50");

delay(1000);

Serial.println("set uart baud 115200");

delay(1000);
```

```
  Serial.println("set uart flow 0");
  delay(1000);
  Serial.println("save");
  delay(1000);
  Serial.println("exit");
  delay(1000);
  digitalWrite(13, LOW);
}
void loop()
{
}
```

During the module configuration, which takes about 25 seconds, the module LEDs will blink differently from its standard pattern. At this moment we know the Wi-Fi module is being configured.

After the module is configured, we try to ping (Command Prompt --> “ping [IP Address]” --> press Enter) the module using the IP address specified in the set ip address command.

The snap shot of CMD is as shown below,



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Dazzler>ping 192.168.2.7

Pinging 192.168.2.7 with 32 bytes of data:
Reply from 192.168.2.5: Destination host unreachable.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.7:
    Packets: Sent = 4, Received = 1, Lost = 3 (75% loss),

C:\Users\Dazzler>ping 192.168.2.7

Pinging 192.168.2.7 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.7:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Users\Dazzler>ping 192.168.2.7

Pinging 192.168.2.7 with 32 bytes of data:
Reply from 192.168.2.7: bytes=32 time=3ms TTL=255
Reply from 192.168.2.7: bytes=32 time=5ms TTL=255
Reply from 192.168.2.7: bytes=32 time=13ms TTL=255
Reply from 192.168.2.7: bytes=32 time=2ms TTL=255

Ping statistics for 192.168.2.7:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 13ms, Average = 5ms

C:\Users\Dazzler>
```

Step 4: Planning the Robot movement

The first thing you have to know to have the capacity to ascertain the fundamental execution time for every development is the robot normal speed. To do that, put a measuring tape parallel to the robot and actuate both engines at the same time for maybe a couple seconds, measure the voyaged remove and conclude the speed. In my arrangement, we got 13.7 centimeters for every second utilizing 62.5% of the most extreme engine speed (250/400, see Pololu Arduino library). At the end of the day, to move the robot 1 meter (100 cm) ahead, the engines must be actuated at the same time for 7.299270… seconds. We have kept the time tallying in the milliseconds determination, however in the event that we need to accomplish higher development accuracy, consider raising the determination to microseconds. Long story short, to move the robot 1 meter, we need to enact both engines at the same time for 7299 milliseconds. From this number, everything progresses toward becoming tenet of three for different separations. To perform bend or roundabout developments, one wheel needs to move speedier than the other. To turn the robot to the sides, just a single wheel must be enacted or both in inverse headings (to turn all alone pivot).

Step 5: Uploading the main code

In this step we uploaded speech robot source code to Arduino but before that we need to upload the library of BitVoicer Server and Polou motor driver libraries to Arduino IDE.

Code:

```
#include <BVSP.h>

#include <BVSMic.h>

#include <DualMC33926MotorShield.h>


// Defines the Arduino pins that will be used to control

// LEDs and capture audio

#define BVS_RUNNING       2

#define BVS_SRE           5

#define BVS_DATA_FWD      3

#define BVS_ACT_PERIOD    6

#define BVSM_AUDIO_INPUT  3


// Defines the constants that will be passed as parameters to

// the BVSP.begin function

const unsigned long STATUS_REQUEST_INTERVAL = 2000;

const unsigned long STATUS_REQUEST_TIMEOUT = 1000;


// Defines the size of the mic buffer
```

```
const int MIC_BUFFER_SIZE = 64;


// Initializes a new global instance of the BVSP class

BVSP bvsp = BVSP();


// Initializes a new global instance of the BVSMic class

BVSMic bvsm = BVSMic();


// Initializes a new global instance of the

// DualMC33926MotorShield class

DualMC33926MotorShield ms = DualMC33926MotorShield();


// Creates a buffer that will be used to read recorded samples

// from the BVSMic class

byte micBuffer[MIC_BUFFER_SIZE];


// Creates a global variable that indicates whether the

// Arduino is connected to BitVoicer Server

boolean connected = false;


// Defines some constants for the motor settings
```

```
const int SPEED_STOP = 0;

const int SPEED_SLOW = 100;

const int SPEED_NORMAL = 250;

const int SPEED_FAST = 400;

const int DIRECTION_FRONT = -1;

const int DIRECTION_BACK = 1;


// Declares a global variables to hold the current motor speed.

// The default is SPEED_NORMAL, but there are voice

// commands that change this setting.

int motorSpeed = SPEED_NORMAL;


// Stores the command duration in milliseconds

unsigned long cmdDuration = 0;


// Stores the time the command started running

unsigned long cmdStartTime = 0;


// Stores whether a command is running or not

bool cmdRunning = false;
```

```
// Stores the last MOVE_FORWARD command. This variable
// is used only for the COME_BACK command.
byte lastFwdCmd = 0;


// Defines some constants for command names/values
// Just to make the code more readable
const byte CMD_STOP = 0;
const byte CMD_MOVE_FORWARD = 1;
const byte CMD_MOVE_FORWARD_1_CM = 2;
const byte CMD_MOVE_FORWARD_2_CM = 3;
const byte CMD_MOVE_FORWARD_5_CM = 4;
const byte CMD_MOVE_FORWARD_10_CM = 5;
const byte CMD_MOVE_FORWARD_25_CM = 6;
const byte CMD_MOVE_FORWARD_50_CM = 7;
const byte CMD_MOVE_FORWARD_1_M = 8;
const byte CMD_MOVE_BACKWARD = 9;
const byte CMD_MOVE_BACKWARD_1_CM = 10;
const byte CMD_MOVE_BACKWARD_2_CM = 11;
const byte CMD_MOVE_BACKWARD_5_CM = 12;
const byte CMD_MOVE_BACKWARD_10_CM = 13;
const byte CMD_MOVE_BACKWARD_25_CM = 14;
const byte CMD_MOVE_BACKWARD_50_CM = 15;

const byte CMD_MOVE_BACKWARD_1_M = 16;

const byte CMD_TURN_AROUND = 17;

const byte CMD_TURN_AROUND_RIGHT = 18;

const byte CMD_TURN_AROUND_LEFT = 19;

const byte CMD_DO_360 = 20;

const byte CMD_TURN_RIGHT = 21;

const byte CMD_TURN_RIGHT_10 = 22;

const byte CMD_TURN_RIGHT_25 = 23;

const byte CMD_TURN_RIGHT_45 = 24;

const byte CMD_TURN_LEFT = 25;

const byte CMD_TURN_LEFT_10 = 26;

const byte CMD_TURN_LEFT_25 = 27;

const byte CMD_TURN_LEFT_45 = 28;

const byte CMD_DO_CIRCLE = 29;

const byte CMD_COME_BACK = 30;

const byte CMD_MOVE_FORWARD_2_M = 31;

const byte CMD_MOVE_FORWARD_3_M = 32;

const byte CMD_MOVE_BACKWARD_2_M = 33;

const byte CMD_MOVE_BACKWARD_3_M = 34;

const byte CMD_SET_SPEED_SLOW = 35;
const byte CMD_SET_SPEED_NORMAL = 36;
const byte CMD_SET_SPEED_FAST = 37;
const byte CMD_TURN_LEFT_45_BACKWARD = 38;
const byte CMD_TURN_RIGHT_45_BACKWARD = 39;

void setup()
{
  // Starts serial communication at 115200 bps
  Serial.begin(115200);

  // Sets the Arduino pin modes
  pinMode(BVS_RUNNING, OUTPUT);
  pinMode(BVS_SRE, OUTPUT);
  pinMode(BVS_DATA_FWD, OUTPUT);
  pinMode(BVS_ACT_PERIOD, OUTPUT);
  AllLEDsOff();

  // Sets the Arduino serial port that will be used for
  // communication, how long it will take before a status request
  // times out and how often status requests should be sent to
  // BitVoicer Server
  bvsp.begin(Serial, STATUS_REQUEST_TIMEOUT,
    STATUS_REQUEST_INTERVAL);

  // Sets the function that will handle the frameReceived
  // event
  bvsp.frameReceived = BVSP_frameReceived;

  // Prepares the BVSMic class timer
  bvsm.begin();

  // Prepares the motor shield class (pins and timer1)
  ms.init();
}

void loop()
{
  // If it is not connected to the server, opens a TCP/IP
  // connection, sets connected to true and resets the BVSP
  // class
  if (!connected)
  {
```

```
  Connect(Serial);

  connected = true;

  bvsp.reset();

}


// Checks if the status request interval has elapsed and if it

// has, sends a status request to BitVoicer Server

bvsp.keepAlive();


// Checks if there is data available at the serial port buffer

// and processes its content according to the specifications

// of the BitVoicer Server Protocol

bvsp.receive();


// Gets the respective status from the BVSP class and sets

// the LEDs on or off

digitalWrite(BVS_RUNNING, bvsp.isBVSRunning());

digitalWrite(BVS_DATA_FWD, bvsp.isDataFwdRunning());


// Checks if there is a SRE assigned to the Arduino

if (bvsp.isSREAvailable())
```

```
{
  // Turns on the SRE available LED
  digitalWrite(BVS_SRE, HIGH);

  // If the BVSMic class is not recording, sets up the audio
  // input and starts recording
  if (!bvsm.isRecording)
  {
    bvsm.setAudioInput(BVSM_AUDIO_INPUT, EXTERNAL);
    bvsm.startRecording();
  }

  // Checks if the BVSMic class has available samples
  if (bvsm.available)
  {
    // Makes sure the inbound mode is STREAM_MODE before
    // transmitting the stream
    if (bvsp.inboundMode == FRAMED_MODE)
      bvsp.setInboundMode(STREAM_MODE);

    // Reads the audio samples from the BVSMic class
    int bytesRead = bvsm.read(micBuffer, MIC_BUFFER_SIZE);


    // Sends the audio stream to BitVoicer Server
    bvsp.sendStream(micBuffer, bytesRead);
  }
}
else
{
  // There is no SRE available
  // Turns off the SRE and ACT_PERIOD LEDs
  digitalWrite(BVS_SRE, LOW);
  digitalWrite(BVS_ACT_PERIOD, LOW);


  // If the BVSMic class is recording, stops it
  if (bvsm.isRecording)
    bvsm.stopRecording();
}


// If the status has timed out, the connection is considered
// lost
if (bvsp.hasStatusTimedOut())
```

```
  {
    // If the BVSMic is recording, stops it
    if (bvsm.isRecording)
      bvsm.stopRecording();

    // Closes the TCP/IP connection
    Disconnect(Serial);

    AllLEDsOff();
    connected = false;
  }

  // If a command is running, checks if its duration has
  // expired. If it has, stop the motors.
  if (cmdRunning)
    if (millis() - cmdStartTime >= cmdDuration)
      RunCommand(CMD_STOP);
}

// Handles the frameReceived event
void BVSP_frameReceived(byte dataType, int payloadSize)
```

```
{
  // Performs the appropriate actions based on the frame
  // data type. If the data type is byte, it is a command.
  // If the data type is int, changes the activated
  // period LED.
  switch (dataType)
  {
    case DATA_TYPE_BYTE:
      RunCommand(bvsp.getReceivedByte());
      break;
    case DATA_TYPE_INT16:
      digitalWrite(BVS_ACT_PERIOD, bvsp.getReceivedInt16());
      break;
  }
}


// Runs the command received from the server
void RunCommand(byte cmd)
{
  switch (cmd)
  {
```

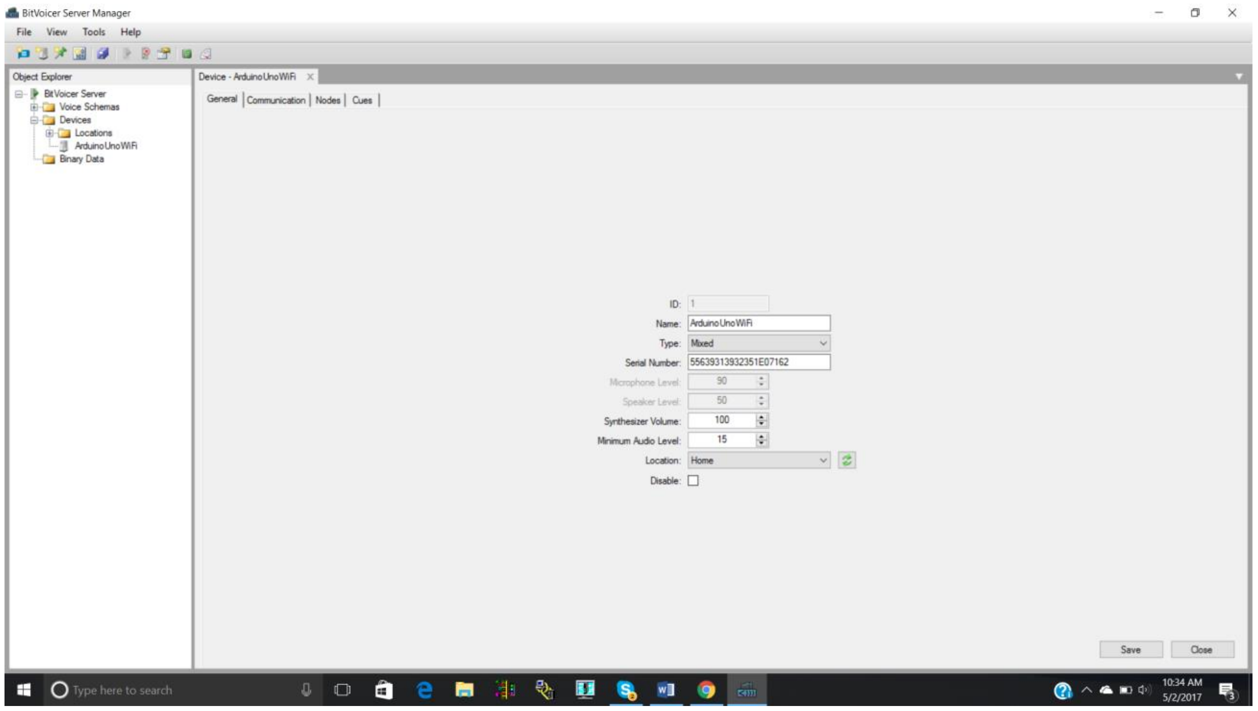
```
case CMD_STOP:
  ms.setSpeeds(SPEED_STOP, SPEED_STOP);
  cmdRunning = false;
  return;
case CMD_MOVE_FORWARD:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 60000;
  break;
case CMD_MOVE_FORWARD_1_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 23;
  break;
case CMD_MOVE_FORWARD_2_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 47;
  break;
case CMD_MOVE_FORWARD_5_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 117;
  break;
case CMD_MOVE_FORWARD_10_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 234;
  break;
case CMD_MOVE_FORWARD_25_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 468;
  break;
case CMD_MOVE_FORWARD_50_CM:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 1170;
  break;
case CMD_MOVE_FORWARD_1_M:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 2339;
  break;
case CMD_MOVE_FORWARD_2_M:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 4678;
  break;
case CMD_MOVE_FORWARD_3_M:
  lastFwdCmd = cmd;
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 7018;
  break;
case CMD_MOVE_BACKWARD:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 60000;
  break;
case CMD_MOVE_BACKWARD_1_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 23;
  break;
case CMD_MOVE_BACKWARD_2_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 47;
  break;
case CMD_MOVE_BACKWARD_5_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 117;
  break;
case CMD_MOVE_BACKWARD_10_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 234;
  break;
case CMD_MOVE_BACKWARD_25_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 468;
  break;
case CMD_MOVE_BACKWARD_50_CM:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 1170;
  break;
case CMD_MOVE_BACKWARD_1_M:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 2339;
  break;
case CMD_MOVE_BACKWARD_2_M:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 4678;

  break;

case CMD_MOVE_BACKWARD_3_M:

  ms.setSpeeds(

    motorSpeed * DIRECTION_BACK,

    motorSpeed * DIRECTION_BACK);

  cmdDuration = 7017;

  break;

case CMD_TURN_AROUND:

  ms.setSpeeds(

    motorSpeed * DIRECTION_FRONT,

    motorSpeed * DIRECTION_BACK);

  cmdDuration = 540;

  break;

case CMD_TURN_AROUND_RIGHT:

  ms.setSpeeds(

    motorSpeed * DIRECTION_FRONT,

    motorSpeed * DIRECTION_BACK);

  cmdDuration = 540;

  break;

case CMD_TURN_AROUND_LEFT:
  ms.setSpeeds(
    motorSpeed * DIRECTION_BACK,
    motorSpeed * DIRECTION_FRONT);
  cmdDuration = 540;
  break;
case CMD_DO_360:
  ms.setSpeeds(
    motorSpeed * DIRECTION_FRONT,
    motorSpeed * DIRECTION_BACK);
  cmdDuration = 1065;
  break;
case CMD_TURN_RIGHT:
  ms.setSpeeds(motorSpeed * DIRECTION_FRONT, 0);
  cmdDuration = 503;
  break;
case CMD_TURN_RIGHT_10:
  ms.setSpeeds(motorSpeed * DIRECTION_FRONT, 0);
  cmdDuration = 56;
  break;
case CMD_TURN_RIGHT_25:
  ms.setSpeeds(motorSpeed * DIRECTION_FRONT, 0);
  cmdDuration = 140;
  break;
case CMD_TURN_RIGHT_45:
  ms.setSpeeds(motorSpeed * DIRECTION_FRONT, 0);
  cmdDuration = 252;
  break;
case CMD_TURN_LEFT:
  ms.setSpeeds(0, motorSpeed * DIRECTION_FRONT);
  cmdDuration = 503;
  break;
case CMD_TURN_LEFT_10:
  ms.setSpeeds(0, motorSpeed * DIRECTION_FRONT);
  cmdDuration = 56;
  break;
case CMD_TURN_LEFT_25:
  ms.setSpeeds(0, motorSpeed * DIRECTION_FRONT);
  cmdDuration = 140;
  break;
case CMD_TURN_LEFT_45:
  ms.setSpeeds(0, motorSpeed * DIRECTION_FRONT);
  cmdDuration = 252;
  break;
case CMD_DO_CIRCLE:
  ms.setSpeeds(
    SPEED_NORMAL * DIRECTION_FRONT,
    SPEED_NORMAL * DIRECTION_FRONT * 0.60);
    cmdDuration = 4587;
  break;
case CMD_COME_BACK:
  RunCommand(lastFwdCmd);
  return;
case CMD_SET_SPEED_SLOW:
  motorSpeed = SPEED_SLOW;
  return;
case CMD_SET_SPEED_NORMAL:
  motorSpeed = SPEED_NORMAL;
  return;
case CMD_SET_SPEED_FAST:
  motorSpeed = SPEED_FAST;
  return;
case CMD_TURN_LEFT_45_BACKWARD:
  ms.setSpeeds(motorSpeed * DIRECTION_BACK, 0);
```

```
      cmdDuration = 252;
      break;
    case CMD_TURN_RIGHT_45_BACKWARD:
      ms.setSpeeds(0, motorSpeed * DIRECTION_BACK);
      cmdDuration = 252;
      break;
  }

  // Sets the command start time
  cmdStartTime = millis();

  // Sets cmdRunning to true
  cmdRunning = true;
}

// Opens a TCP/IP connection with the BitVoicer Server
void Connect(HardwareSerial &serialPort)
{
  serialPort.print("$$$");
  delay(500);
  // Use the IP address of the server and the TCP port set
  // in the server properties
  serialPort.println("open 192.168.0.11 4194");

  delay(1000);
  serialPort.println("exit");
  delay(500);
}

// Closes the TCP/IP connection with the BitVoicer Server
void Disconnect(HardwareSerial &serialPort)
{
  serialPort.print("$$$");
  delay(500);
  serialPort.println("close");
  delay(1000);
  serialPort.println("exit");
  delay(500);
}

// Turns all LEDs off
```

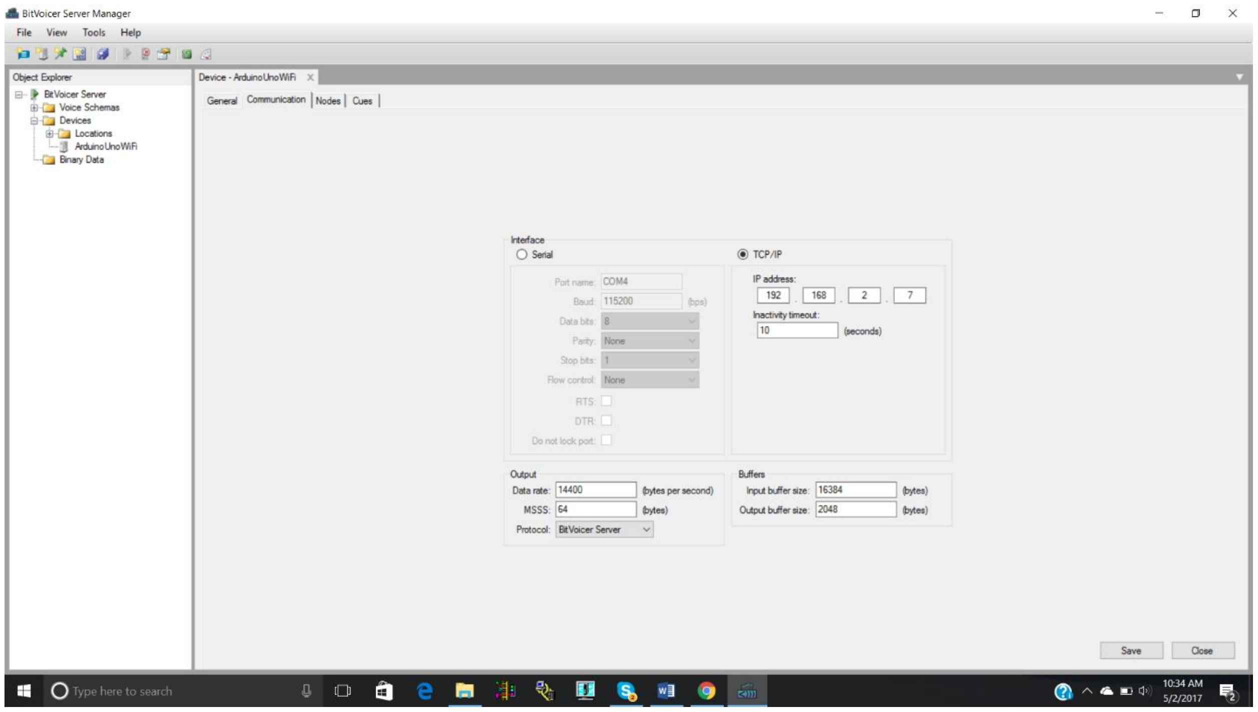
```
void AllLEDsOff()

{

  digitalWrite(BVS_RUNNING, LOW);

  digitalWrite(BVS_SRE, LOW);

  digitalWrite(BVS_DATA_FWD, LOW);

  digitalWrite(BVS_ACT_PERIOD, LOW);

}
```

Step 6: Setting up BitVoicer Server

In BitVoicer Server first of all we need to create device as shown in below diagram,



ID:	1
Name:	ArduinoUnoWiFi
Type:	Mixed
Serial Number:	55639313932351E07162
Microphone Level:	90
Speaker Level:	50
Synthesizer Volume:	100
Minimum Audio Level:	15
Location:	Home
Disable:	☐



Interface

( ) Serial

Port name: COM4
Baud: 115200 (bps)
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
RTS: ☐
DTR: ☐
Do not lock port: ☐

(●) TCP/IP

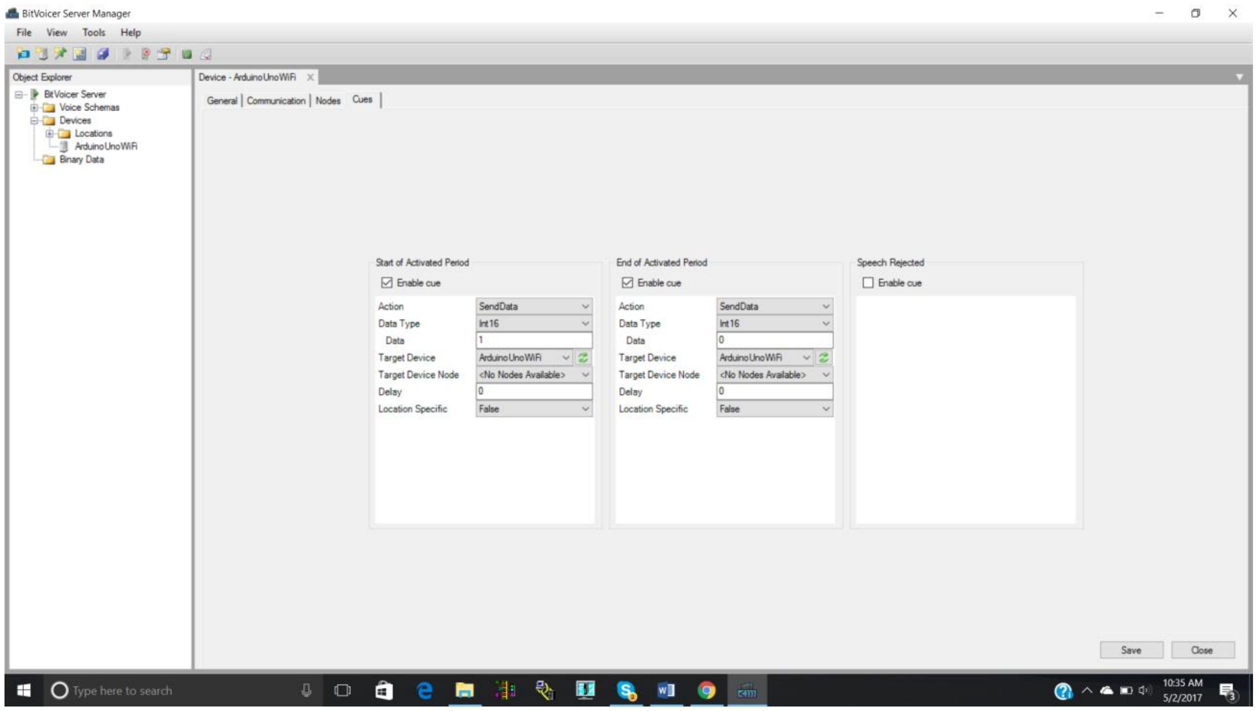
IP address: 192 . 168 . 2 . 7
Inactivity timeout: 10 (seconds)

Output

Data rate: 14400 (bytes per second)
MSSS: 64 (bytes)
Protocol: BitVoicer Server

Buffers

Input buffer size: 16384 (bytes)
Output buffer size: 2048 (bytes)



Start of Activated Period

☑ Enable cue

Action	SendData
Data Type	Int16
Data	1
Target Device	ArduinoUnoWiFi
Target Device Node	<No Nodes Available>
Delay	0
Location Specific	False

End of Activated Period

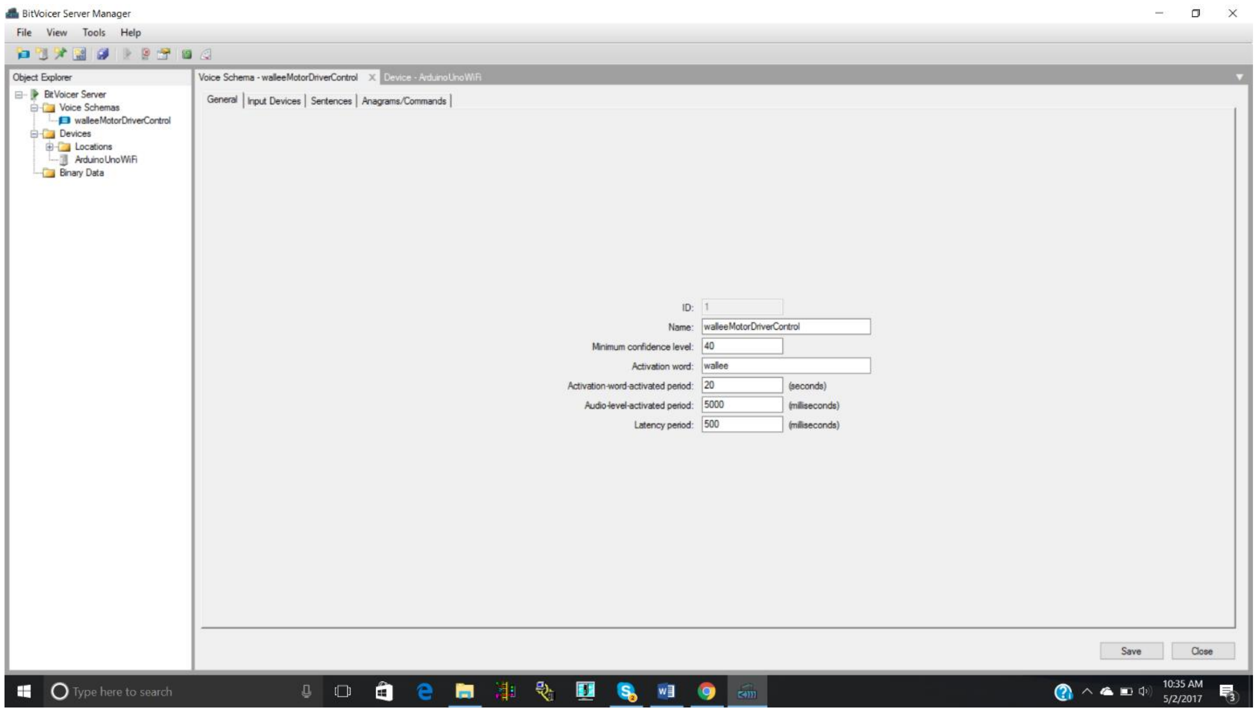
☑ Enable cue

Action	SendData
Data Type	Int16
Data	0
Target Device	ArduinoUnoWiFi
Target Device Node	<No Nodes Available>
Delay	0
Location Specific	False

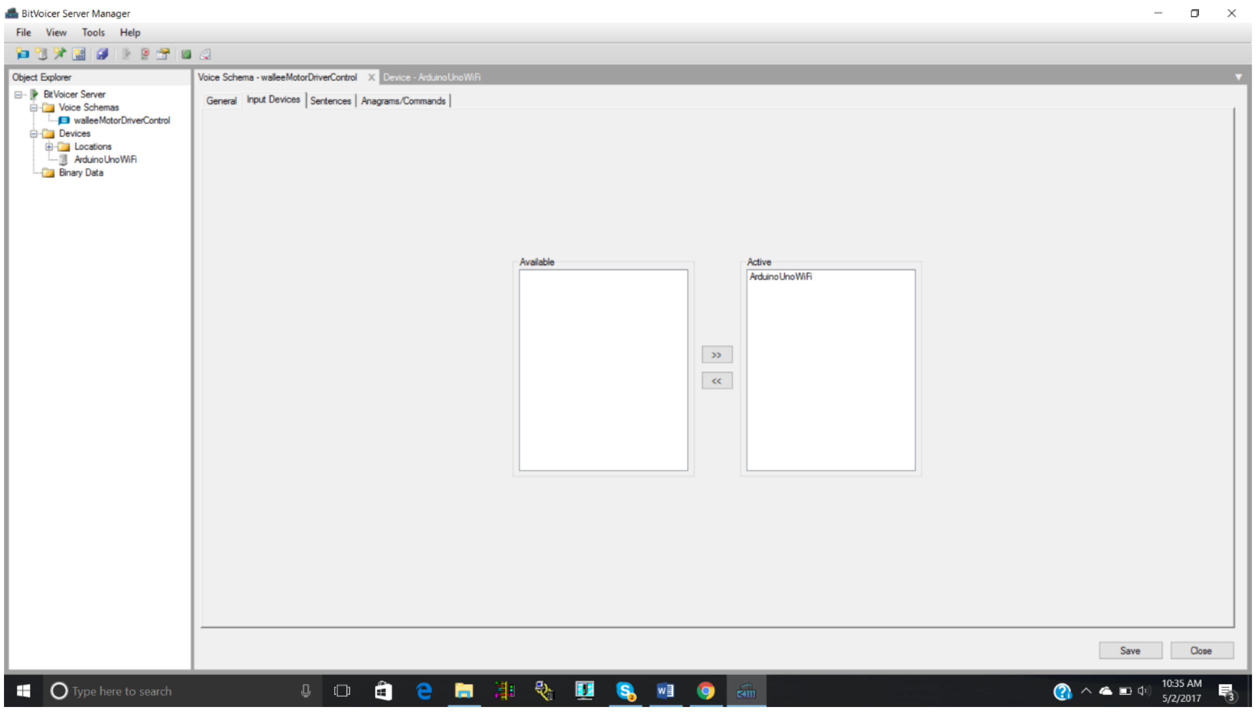
Speech Rejected

☐ Enable cue

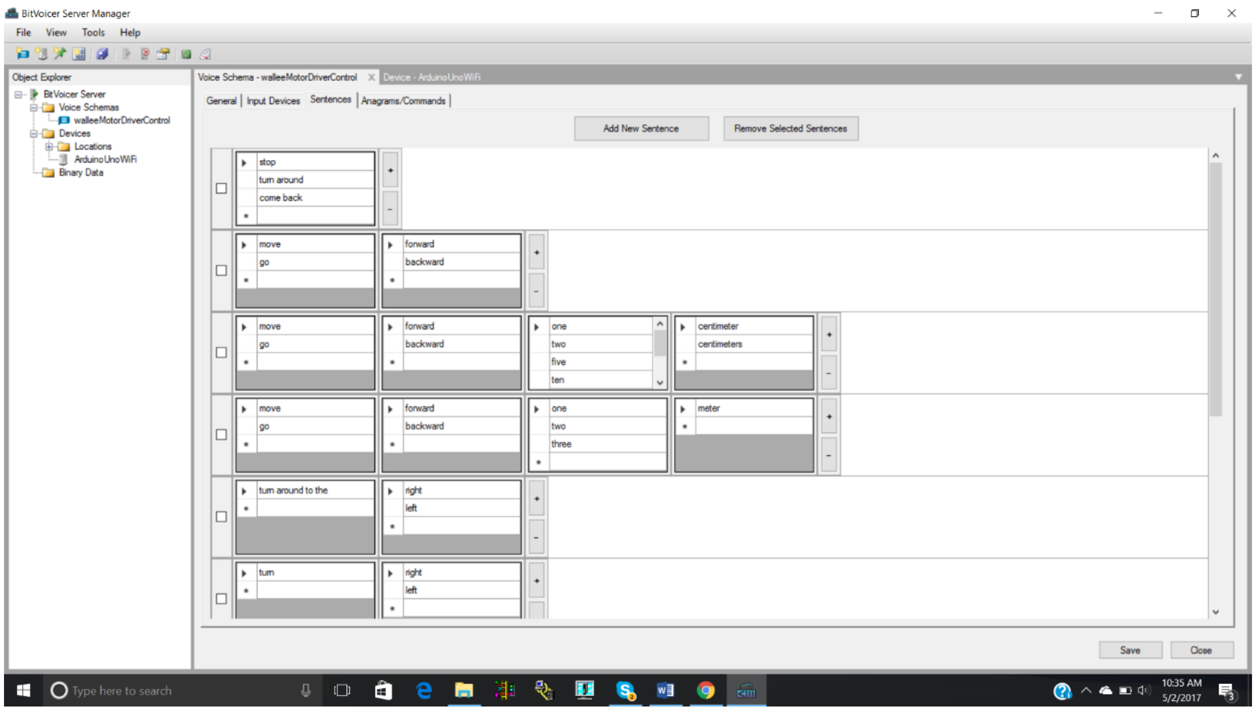
Next step is Create Voice Schemas as shown in below,



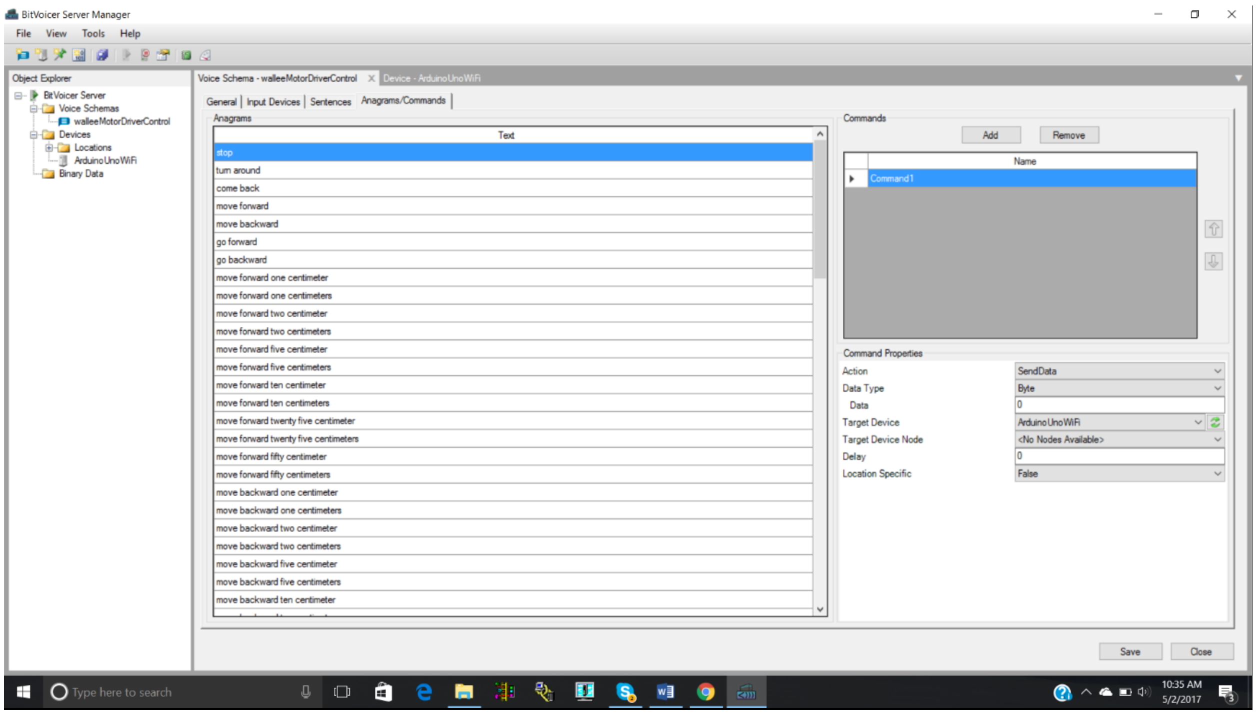
ID: 1
Name: walleeMotorDriverControl
Minimum confidence level: 40
Activation word: wallee
Activation-word-activated period: 20 (seconds)
Audio-level-activated period: 5000 (milliseconds)
Latency period: 500 (milliseconds)



Available	Active
	ArduinoUnoWiFi



stop / turn around / come back			
move / go	forward / backward		
move / go	forward / backward	one / two / five / ten	centimeter / centimeters
move / go	forward / backward	one / two / three	meter
turn around to the	right / left		
turn	right / left		



Anagrams

Text
stop
turn around
come back
move forward
move backward
go forward
go backward
move forward one centimeter
move forward one centimeters
move forward two centimeter
move forward two centimeters
move forward five centimeter
move forward five centimeters
move forward ten centimeter
move forward ten centimeters
move forward twenty five centimeter
move forward twenty five centimeters
move forward fifty centimeter
move forward fifty centimeters
move backward one centimeter
move backward one centimeters
move backward two centimeter
move backward two centimeters
move backward five centimeter
move backward five centimeters
move backward ten centimeter

Commands

Name
Command1

Command Properties

Action	SendData
Data Type	Byte
Data	0
Target Device	ArduinoUnoWiFi
Target Device Node	<No Nodes Available>
Delay	0
Location Specific	False

Here as shown in figure we configure the BitVoicer Server. Here I have put 40 different command which define 80 different sentence in our project.

Conclusion:

After some testing with the robot, we were happy with the discourse acknowledgment, even though it didn't perceive 100% of the orders every one of the circumstances. In this viewpoint BitVoicer Server truly amazed us. In any case, we were not that abundantly happy with the accuracy of the robot's developments. To settle this issue, we would need to add rotation sensors to the wheels. The case pack we utilized as a part of the robot as of now accompanies decoder plates that can be attached to the gear. Applying sensors to these disks would enable me to move the robot in view of the genuine traveled distance so that its development would be more exact. We could add ultrasonic sensors to avoid bumping around furniture.

1. **References :**

[1] http://rossum.sourceforge.net/papers/CalculationsForRobotics/CirclePath.htm

[2] http://math.stackexchange.com/questions/60176/move-two-wheeled-robot-from-one-point-to another.

[3] http://www.bitsophia.com/en-US/BitVoicerServer/Overview.aspx

[4] http://cdn.sparkfun.com/datasheets/Wireless/WiFi/rn-wiflycr-ug-v1.2r.pdf

[5] http://www.bitsophia.com/en-US/BitVoicerServer/v1/Documentation/Manual.aspx

[6] https://www.pololu.com/docs/pdf/0J55/dual_mc33926_motor_driver_shield.pdf