# VoiceCommands.h

This file is responsible for handling incoming voice commands that are processed by the ESP32-WROVER. The ESP32-WROVER module has code running on its own and is explained in the documentation written for the “ESP32-WROVER”.

A picture containing text

Description automatically generated

This function runs when the Arduino starts, and it will start the Serial communication with the ESP32-WROVER using Serial3 (TX PIN 18 + RX PIN 19) using BAUD 9600 (port used to communicate with).

The next function will be split in segments since there is quite a lot going on.

Text

Description automatically generated

The “getVoiceCommandSerialString()” function is used to receive data from the ESP32-WROVER module.

It will return an empty string (“”) when no message was received, and it will return the actual message if a message is received.

We define a new string variable called “serialInput” which is set as an empty string (“”).

We will add all the incoming data to this string over time. Serial communication doesn’t send the data in 1 go, but it sends it character by character. So we will need a loop that can catch all these characters and append them to the “serialInput” variable.

We also tell the Arduino to reserve up to 200 bytes for this string using “serialInput.reserve(200)” This is probally not a necessity, but it makes sure we can append longer messages.

For this device we are using Serial3 on the Arduino, this is **TX pin 18** and **RX pin 19**. The arduino Mega automatically translate this to the “Serial3” instance, which allows us to read the data.

When if(!Serial3.available()) is true, it means that no data is currently coming in.

If this is the case there is no reason to continue with the rest of the function, so we return the “serialInput” variable which currently is an empty string.

Text

Description automatically generated

Here we are creating a variable called “startMillis” which will be set to the value that “millis()” returns (an integer). “millis()” is an Arduino function that tells us how many milliseconds has passed since the arduino started. We will use this value to determine how much time has passed in the “while” loop to stop the loop if no data was passed back within a certain time.

We also create a boolean “endOfLine” which is set to false. This variable is used to know whether the end of the data was received. This will always be an enter whitespace character (which can be read as the “\n” character).

Next we will start a while loop, which will keep on running as long as “endOfLine” is false: “while(!endOfLine)”. Inside of the loop we will again get the time passed since the Arduino was turned on using the “millis()” function and set this to the “currentMillis” variable.

We can now calculate to total seconds that passed since the while loop was started.

By substracting the startMillis from the currentMillis we know how many milliseconds passed. If then we divide this by 1000 we get the amount of seconds.

This is exactly what we do when we set the “seconds” variable.

Now that we know how many seconds have been that went by in the loop, we can do “if(seconds >= 1)” to check if more then 1 seconds was passed.

Something to note here is that the arduino executes code really fast, and in 1second it will probally have cycled the while loop more then 10 thousands of times!

If more then 1 seconds passed, we will print a message to the Serial monitor from the Arduino IDE to inform that the message took to long.

After that we set the serialInput back to an empty string, and set the endOfLine to true so the while loop stops.

Graphical user interface, application

Description automatically generated

Since this while loop is triggered very quickly (10.000 of times or more per second) there is a change that no characters are currently coming in. We can determine this by checking if “Serial3.available()” is false. If this is the case, we will tell the loop to continue. This will stop the rest of the while loop from executing, and make it start the while from the start again.

Graphical user interface, text, application

Description automatically generated

Here we are receiving the character that came in through the Serial comminucation.

We receive this by using “(char)Serial3.read()”. Char here defines the type that the function will return. We will then set this to the “newChar” variable.

We then append this character to the “serialInput” string created before by doing “serialInput += newChar”.

A picture containing text

Description automatically generated

This is where we listen if we received the last character to end the serial communcation. We will always receive an enter character (“\n”). So if(newChar == “\n”) we know the communcation is done, and we can set the “endOfLine” to true to stop the while loop.

This is also where the code for the while loop ends. (the final “}” that you can see)

Text

Description automatically generated with medium confidence

This is the last part of the “getVoiceCommandSerialString()” function. Here we trim the “serialInput” string. What this does is remove all the whitespace at the start and at the end of the string. In our case it will remove the last enter character (“\n”) from the serialInput string.

We then return the “serialInput” back to the initator of this function so it can be used to do something depending on the data we received.

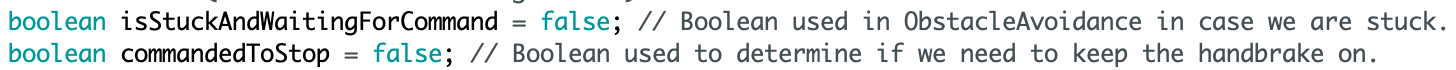
Graphical user interface, text

Description automatically generated with medium confidence

The next section will handle the incoming voice command.

The command you see in in the screenshot above are the possible commands we listen to.

We are using numbers because they are smaller in size and prevent losing data during communication.



The “isStuckAndWaitingForCommand” boolean is used in the “ObstacleAvoidance.h” to determine if we are currently in need of voice command to tell the robot where to go.

The “commandedToStop” will be a boolean used to determine if we need to put the handbrake (defined in “Driver.h”) on or not, so we can prevent the robot from driving off.

Text

Description automatically generated

The “handleVoiceCommand” accepts the command as a parameter and will handle the command accordingly.

Most of these if statemens use the “drive()” function defined in “Motor.h”. It will accept the direction to drive in as a first parameter, and the (optional) amount of time to go in this direction in ms. The directions are used in the following way:

Graphical user interface, text, application

Description automatically generated

“if(command == “11”)”:

We will drive forward for 500ms and set the “commandedToStop” to false, since we just told it to drive forward.

“else if(command == “12”)”:

We will drive backward for 500ms, and then use the “drive(0)” function to tell it to stop all together, else the robot will directly drive forwards again when it triggers the “selfDriving()” function (defined in “Driver.h”) in the main “loop()”.

“else if(command == “13”)”:

Stop driving by passing 0 to the “drive()” function. We also set the “commandToStop” to true, since we just told it to stop.

“else if(command == “14”)”:

Drive to the left for 500ms using “drive(2, 500)”.

“else if(command == “15”):

Drive to the right for 500ms using “drive(3, 500”.

“else if(command == “16”):

We use the “robotDance()” defined in “Celebrate.h” to make the robot dance.

“else if(command == “17”):

We set the handbrake to false, doing this will allow the “selfDriving()” function (defined in “Driver.h”) to be executed from the main loop() function.

Text

Description automatically generated

If the received command is not “11” (drive forward) and the commandedToStop variable is currently true, we turn on the handbrake. If we don’t do this, the robot will directly drive forwards again when it triggers the “selfDriving()” function (defined in “Driver.h”) in the main “loop()”.

Text

Description automatically generated

Since we received a command, we set “isStuckAndWaitingForCommand” to false. This can then be handled accordinly in the “ObstacleAvoidance.h” code. Here we came to end of this function (“}”).

Graphical user interface, text

Description automatically generated

This function is run in the main “loop()” function on the Arduino, it will listen to the incomming data and handle what to do with it. It returns true or false depending if it returned a voice command was received. This is used in the “ObstacleAvoidance.h” to check if we received a command before the robot decided where to go next on his own.

We define a new variable called “incomingCommand” and set it to the value returned from the “getVoiceCommandSerialString()” that was explained before. When this returns an empty string (“”) it means no command was received. We then return false and stop the rest of the code from executing.

Text

Description automatically generated with low confidence

We output the received incomming command to the Serial Monitor in the Arduino IDE for debugging.

If we receive the command “01” from the ESP32-WROVER, it means we lost bluetooth connection with the phone. If this happens when we are currently playing the game, we will play the "I can't here you” MP3 file (track 14).

If we receive the command “02” from the ESP32-WROVER, it means we reconnected and have established a bluetooth connection again. If this happens when we are currently playing the game, we will will play the “Lets go” MP3 file (track 2).

Graphical user interface, text, application, email

Description automatically generated

Voice commands will always start with a “1”. We can use the “startWith()” function on the string to determine if this is true: “if(incomingCommand.startsWith(“1”))”.

When this is true, we will trigger the “handleVoiceCommand” function explained before, and pass the voice command we received as the first parameter.

We will now check if we just started the game. The game will automatically start with the first voice command that comes in, with the exception of the “dance” and “drive” commands (“16” and “17”).

So when we are currently not voiceControlled (playing the game) and neither of the incoming commands are “16” or “17” we will start the game.

We first set the “hasWon” to false, this allows us to trigger the “winner()” function defined in “Celebrate.h” in the main “loop()” function as soon as the color red is detected.

We set the “voiceControlled” to true, so we can determine in other parts of the code if we are currently playing the came.

We then send a “1” to the countdown timer using “Serial2.println(“1”)”. Serial2 (pin 16 + 17) are used to communicate with the countdown timer using the BLE bluetooth module.

When this is done we will play the “Let’s go!” MP3 file (track 2).

We then return true because we received a voice command, and can stop the rest of the code from executing.

At the end of the function we will return false, since we either received a command we don’t understand, or we received a command that is not a voice command.