# Bluetooth.h

This code will communicate with the countdown timer using the BLE Bluetooth module. This **BLE Bluetooth module sits on the right shoulder of the robot**.

A picture containing text

Description automatically generated

We define a ground PIN using BT\_GROUND\_PIN (this is because there were no ground pins left on the Arduino Mega), we will simply write LOW to this PIN so it can be used as ground. In this case we are using PIN 50.

We also define the BT\_CONNECTED\_PIN, this is the PIN we use to know if the BLE device is currently connected with another BLE device.

Text

Description automatically generated

The function “isBluetoothConnected()” returns a boolean (true, false) when the Bluetooth module is connected to another BLE device. When the “digitalRead()” function gets back a “1” (which means there is power coming from the BT\_CONNECTED\_PIN) we are connected.

Text

Description automatically generated

This part is used to blink the blue led light on the Arduino as long as we are not connected.

It is run using the “timer” library, which means we need to return true or false to tell the library if we want to run this function again in the next cycle. We also need to accept “void\*” as a parameter, since any type of parameter can potentially be passed using the timer library.

When the “ledLight” variable is set to “off” (This variable is defined in “LedControl.h” and holds the active color that is used for the LED light) we will change the LED to blue using the changeLED() function (also defined in “LEDControl.h”).

If ledLight is no set to “off” we will set it to “off” now, making it blink as long as this function is triggered multiple times in a row.

At the end we return true to tell the timer it can run this function again.

Graphical user interface, text, application, email

Description automatically generated

We create a new variable called “bluetoothTimer” which is a “Timer<>::Task”. This tells the Arduino that this variable will hold a value that can be used as a timer task. The timer task is returned by the “timer.every()” function and can then be used to cancel the task in a later stage using the “timer.cancel()” function.

The “checkBluetoothConnection()” function is used to trigger the “blinkLed()” function defined before as long as we are not connected to another BLE device. It will return true or false depending on the connection state.

If “isBluetoothConnected()” returns false we will want to start the timer.

At first instance the “bluetoothTimer” function is undefined, the moment we set it to the timer task it will hold a value. We only want to start the timer once, else multiple timer instances will start every single time the “checkBluetoothConnected()” get triggered.

To prevent this we simply do “if(!bluetoothTimer)”, if this returns true we know the bluetoothTimer is not set yet and we can start the timer using “timer.every(1000, blinkLed)” which will trigger the “blinkLed()” every second untill we cancel it.

We then return false, and stop the rest of the code from executing, to tell the initiater of this function that there is currently no active connection.

When there is a connection, we want to make sure the LED light is blue. However, we only need to do this when the LED currently is not blue yet. So “if(ledLight != “blue”)” we will trigger the “changeLED()” function and pass “blue” as a parameter to turn it blue.

Now we want to cancel the timer task that is running, but we only need to do this if the timer task is actually currently running. When “if(bluetoothTimer)” return true we know this is the case.

By triggering the “timer.cancel()” function and passing the variable holding the task (bluetoothTimer) we tell the timer library to stop this task.

At the end we will return true, to tell the initiator of the function that we are currently connected.

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

Text

Description automatically generated with medium confidence

The “getSerialString(): function is used to receive data from the other BLE bluetooth device. It will return an empty string (“”) when no message was received, and it will return the actual receive 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 te reserve up to 200 bytes for this string using “serialInput.reserve(200)” This is probally not a neccesity, but makes sure we can append longer messages.

For this device we are using Serial2 on the arduino, this is **TX pin 16** and **RX pin 17**. The arduino Mega automatically translate this to the “Serial2” instance, which allows us to read the data.

When if(!Serial2.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 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.

A picture containing text

Description automatically generated

If for whatever reason we lose connection with the other device while the loop is going, we also want to stop the loop. This is done using the “if(!isBluetoothConnected())” statement. When this returns true we will again set the serialInput to an empty string (“”) and set “endOfLine” to true to stop the while loop.

Graphical user interface, text, application

Description automatically generated

Since this while loop is triggered very quickly (10.000 of times or more per second) there is a chance that no characters are currently coming in. We can determine this by checking if “Serial2.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.

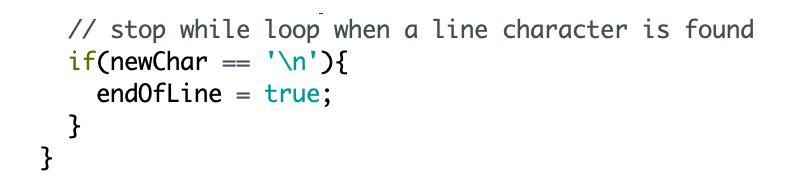
Graphical user interface, text, application

Description automatically generated

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

We receive this by using “(char)Serial2.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”.



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 “getSerialInput()” 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.

Text

Description automatically generated

The “handleTimerCommand” will handle what to do with the incomming command from the timer. We first define a variable that will tell us if the countdown timer is currently running and counting down. This will be the “timerRunning” boolean that is intially set to false. This variable is used later on in an if statement to send the command to the countdown timer to start the countdown in case we are already playing the game and the countdown didn’t start yet.

The “handleTimerCommand” accepts a parameter called “command” which is a string holding the command that was sent from the other BLE bluetooth device. We use numbers since those are smaller in bytes and can prevent misreadings compared to longer strings (else it could, for example, lose a character during transmission). We use double digits, so we can gp above 10 if we need to. As a bonus it will also make the communication faster.

It can be any of the following commands:

* 01 means the halfway mark of the countdown timer has passed
* 02 means that we have 1/5 of the total time left
* 03 means that there are 10 seconds left
* 04 means that there are 6 seconds left (playtime of the losing sound)
* 05 means we lost the game (0 seconds left)
* 06 means the countdown timer has started
* 07 means the countdown timer has stopped

We simply do an “if / else” logic to determine what to do on each command.

For most commands we will trigger an MP3 file to play, we do this using the “playMP3()” function defined in “MP3Player.h” by passing the track we want to play.

“if(command == “01”)”:

We will play the “don’t forget, time is ticking” MP3 file (track 3).

“else if(command == “02”)”:

We will play the “Running out of time!” MP3 file (track 4).

“else if(command == “03”)”:

We will play the “10 seconds left” MP3 file (track5).

“else if(command == “04”)”:

We will play the losing sound MP3 file (track6).

“else if(command == “05”)”:

We lost, in this case we stop driving using “drive(0)” defined in “Driver.h” after that we will play the “We didn't lose the game, we just ran out of time.” MP3 file (track 7). We then play the “Time for a new round. Give me directions when you are ready.” MP3 file (track 13). We however want this track to play with a delay, else it won’t be able to finish the previous MP3 file that is currently playing. We do this by passing a second parameter to the “playMP3()” function, which holds the delay in ms. So when we trigger “playMP3(13, 5000)” it will play track 13 5000ms from now. When everything is done, we will set the “voiceControlled” variable to false, this will tell the robot that a new game can be started and that we are no longer controlled by voice commands until a new voice command is received.

“else if(command == “06”)”:  
The countdown timer started, so we set the “timerRunning” variable to true.

“else if(command == “07”)”:

The countdown timer ended, so we set the “timerRunning” variable to false.

Text

Description automatically generated

This is the function that will run when the Arduino starts.

It sets the BT\_CONNECTED\_PIN to the INPUT mode using the “pinMode” function from the Arduino. This allows us to read a power value coming from this PIN (used for the the BLE bluetooth state).

It sets the BT\_GROUND\_PIN to an OUTPUT, so we can send a power value to this pin.

We then use “digitalWrite()” to set the BT\_GROUND\_PIN to LOW, this allows us to use this PIN as ground.

We also begin Serial2 on BAUD 9600, this is essentially the port that the Arduino uses to communicate with the BLE module.

Graphical user interface, text, application

Description automatically generated

The “runBLECommunication()” function is run in the main “loop()” function defined in “Arduino\_MEGA.ino”. It will try to read an incoming command and it will trigger the functions that are needed to handle them.

We will first check if “checkBluetoothConnection()” is true or false, this function will make the LED light blink blue (as explained before). When this returns false there is no active connection and nothing else to do, so we do a return to stop the rest of the function from executing.

When “checkBluetoothConnection()” we will continue the function, we will then check if we are currently voiceControlled (so playing the game) and if the countdown timer is already running.

If we are voiceControlled, but the countdown timer is not running yet (“if(voiceControlled && !timerRunning)”) we will send a “1” to the countdown timer using “Serial.println(“1”)”. When the countdown timer receives this command, it will start the countdown immediately.

We will then create a new variable (incommingCommand) and set it to the string returned from the “getSerialString()” function explained before.

If an empty string is returned: “if(incommingCommand == “”)”, we didn’t receive an actual command from the countdown timer and will do a return so the rest of the function is not executed.

If we do get a command back, we will print it to the Serial Monitor in the Arduino IDE for debugging purpose. We will then trigger the “handleTimerCommand” function (explained before) and pass the received command along.