AlgoC User Guide

In this document, you can find detailed information about all the AlgoC. We will describe supported elements of the AlgoC, how they can be used, and what parameters they can use.

Getting started with the AlgoC

Installing the Arduino IDE

To install the Arduino IDE go to the following web-site https://www.arduino.cc/en/software and download the version for your version of the operating system.

You should select the latest version 2.x.x as shown on the image below.

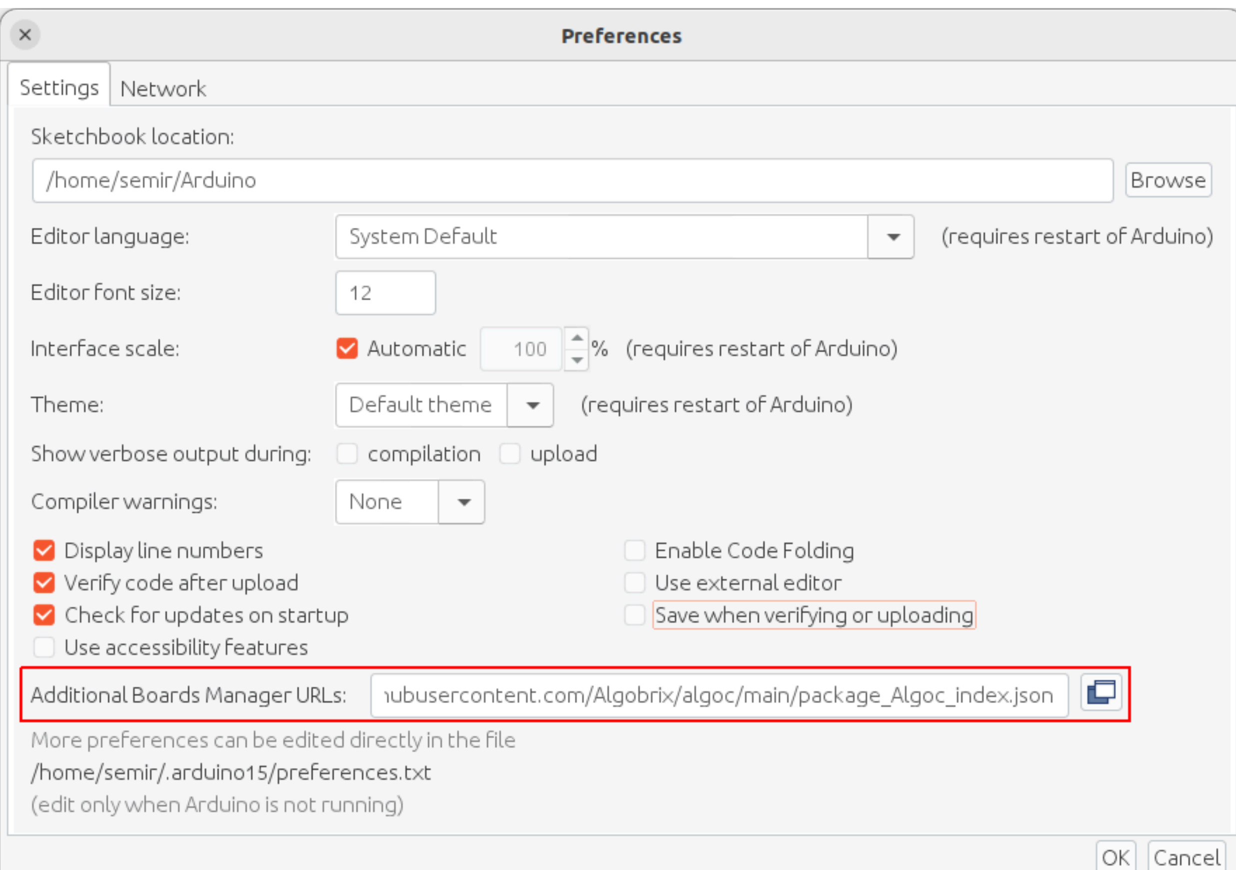
Installing the AlgoC

To start using the AlgoC we first need to install the AlgoC SDK inside the Arduino IDE environment. We can do that by following the next steps:

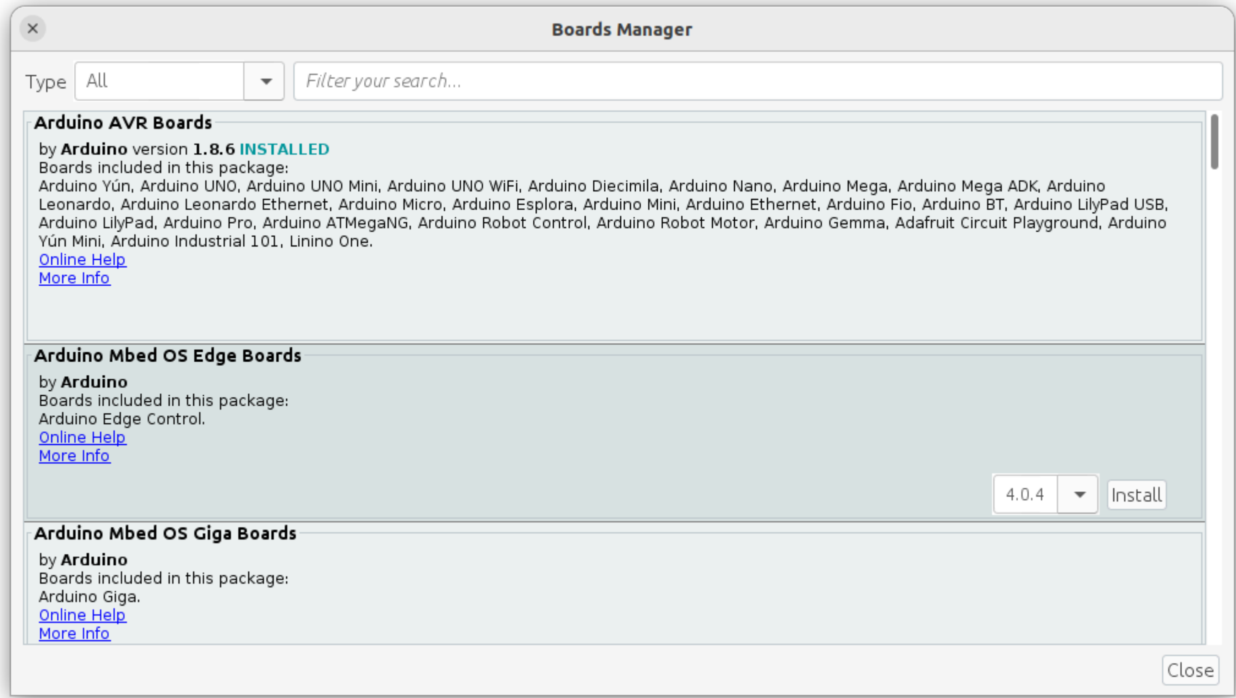
1. Open the Arduino IDE. You should see the following screen.

```
sketch_aug02a | Arduino 1.8.19
File  Edit  Sketch  Tools  Help
sketch_aug02a
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
1                                    AlgoBot on /dev/ttyUSB0
```

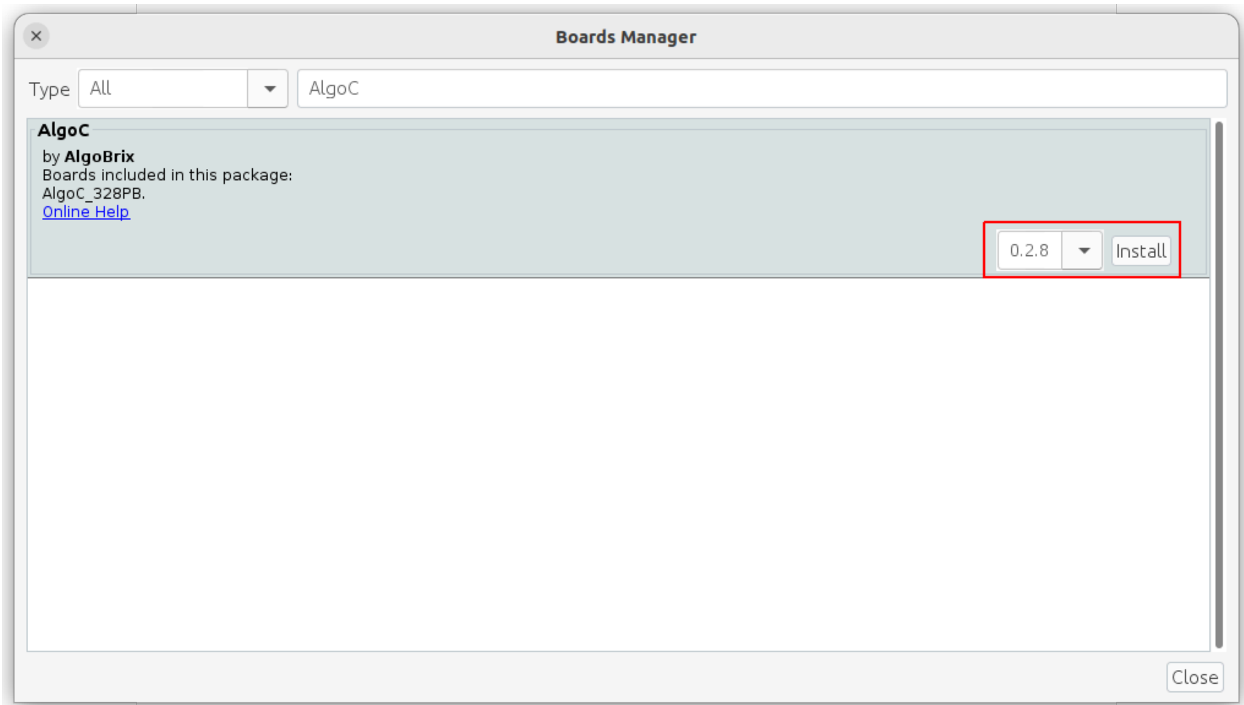
2. Go to the File->Preferences and inside the Additional Boards Manager URLs copy the following link and press OK:
   https://raw.githubusercontent.com/Algobrix/algoc/main/package_Algoc_index.json



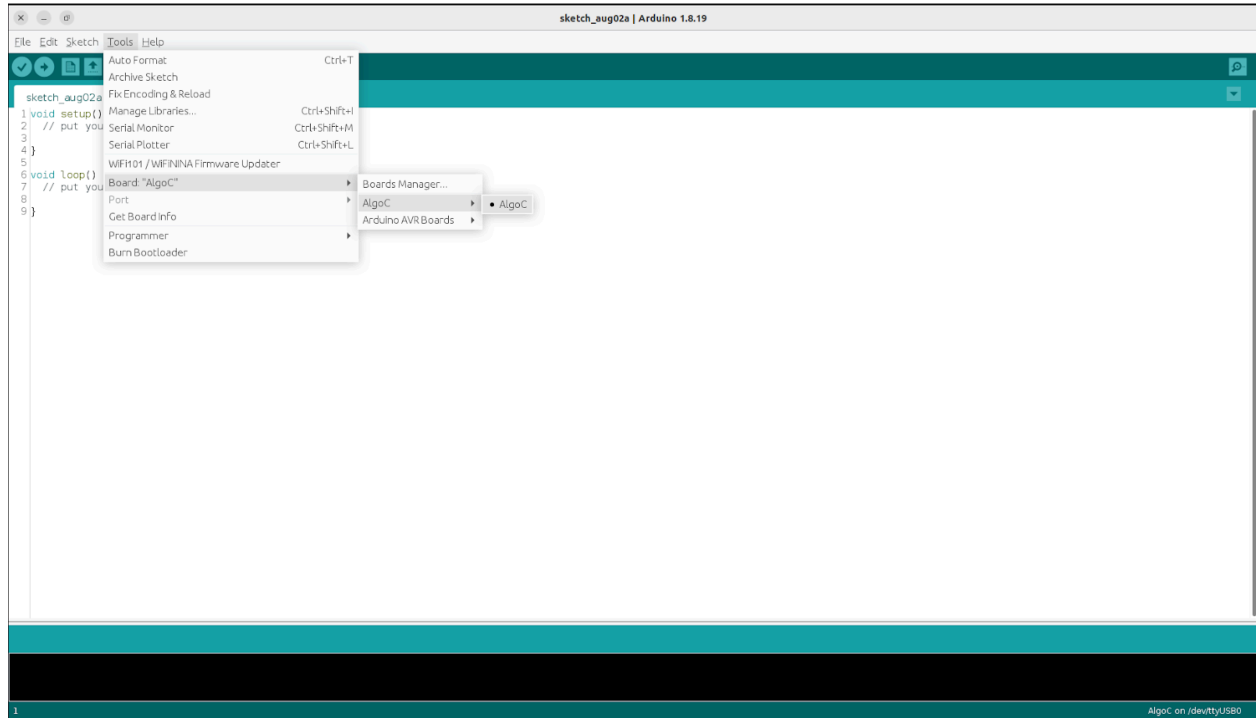
3. Now we need to install the SDK by going to the Tools->Board->Boards Manager. You should see the following screen.



4. Search for the AlgoC. Select the latest version and press install.



5. Now we need to select the AlgoC board by going to the Tools->Board->AlgoBrix and selecting the AlgoC

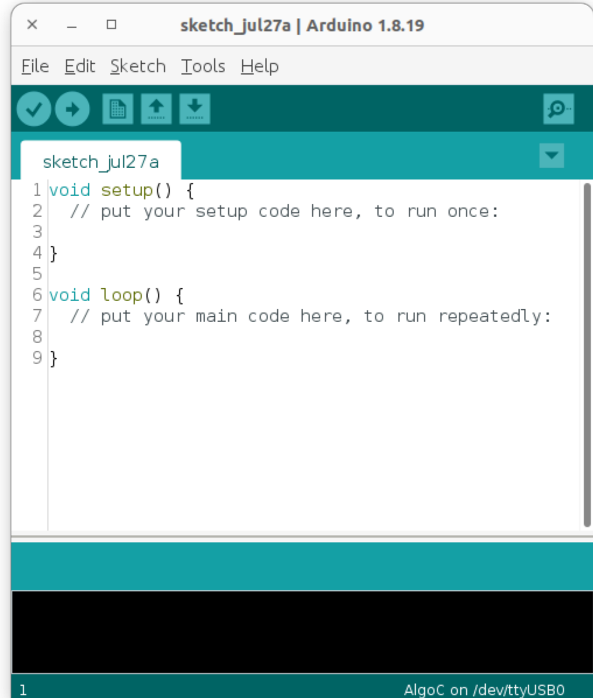


Congratulations, you are now ready to start writing your first application using the AlgoC SDK.

Compiling your first AlgoC application

Even though the AlgoC is based on the Arduino platform, SDK is relaying on different function when executing the code. This means that when starting the new sketch we need to do the following:

1. Open the Arduino IDE. You will see the default code structure that is provided by the Arduino platform.



2. Erase all text inside the Arduino IDE window.

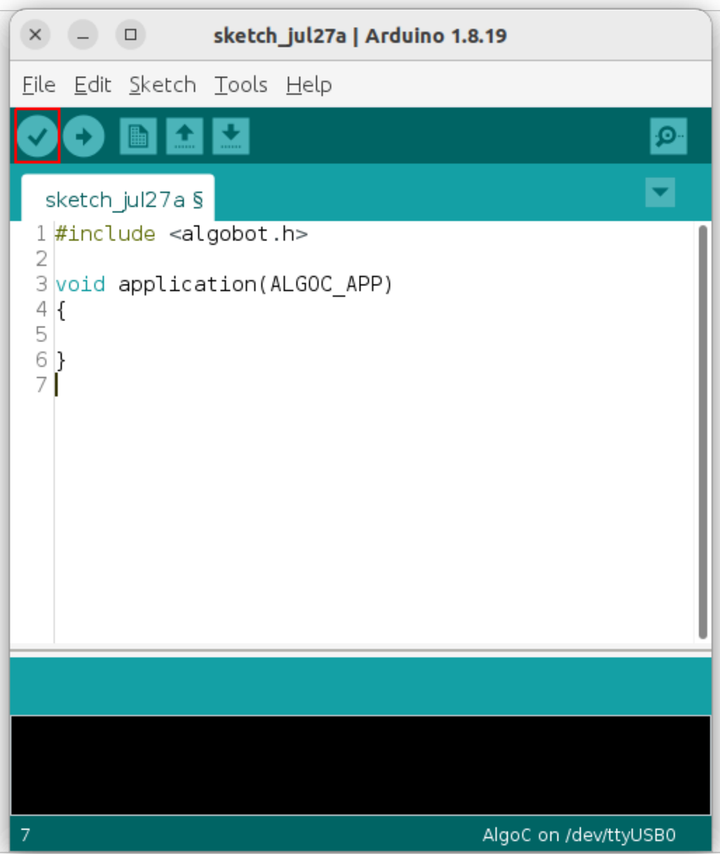
3. Add the following code to the Arduino IDE:

```C/C++
#include <algoC.h>


void application(ALGOC_APP)
{

}
```

4. Press the compile button.

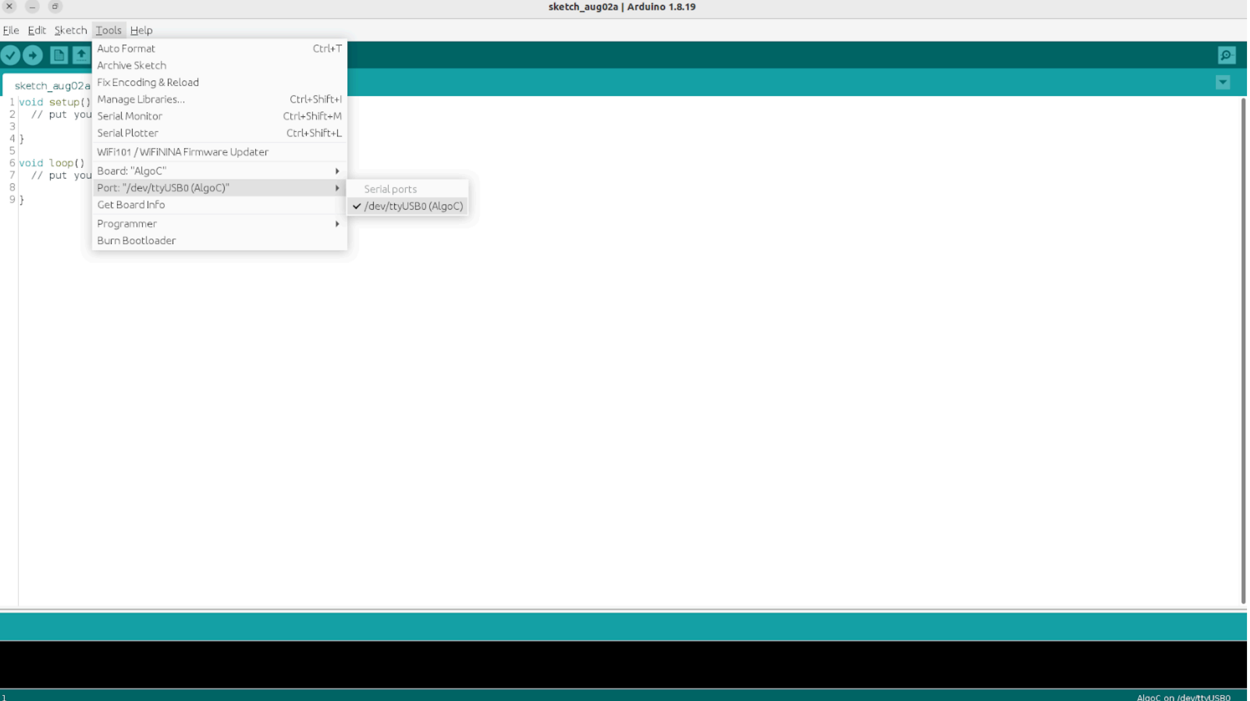


5. Congratulations, you have successfully written your first AlgoC application. Now continue exploring our SDK with the provided examples to learn how to control Motor and Light or how to read sensor state and much more.

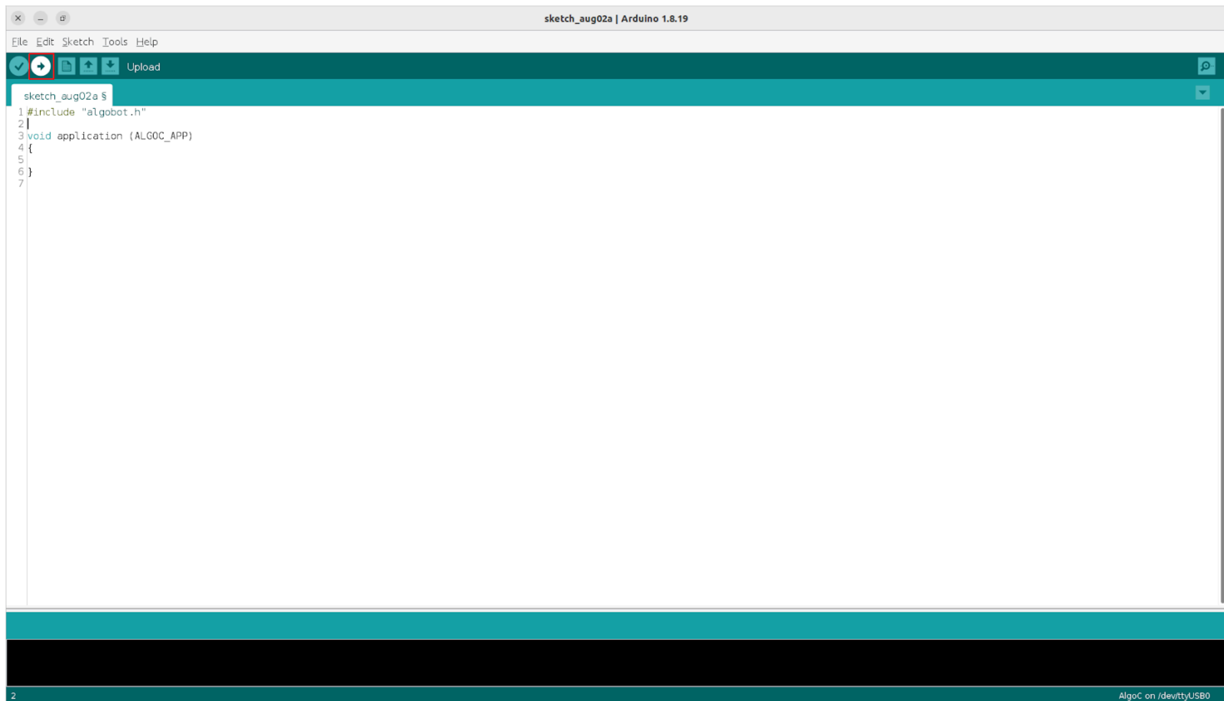
Uploading your first code to the AlgoC board.

Now, we are going to upload our first code to the AlgoC board by following the below steps:

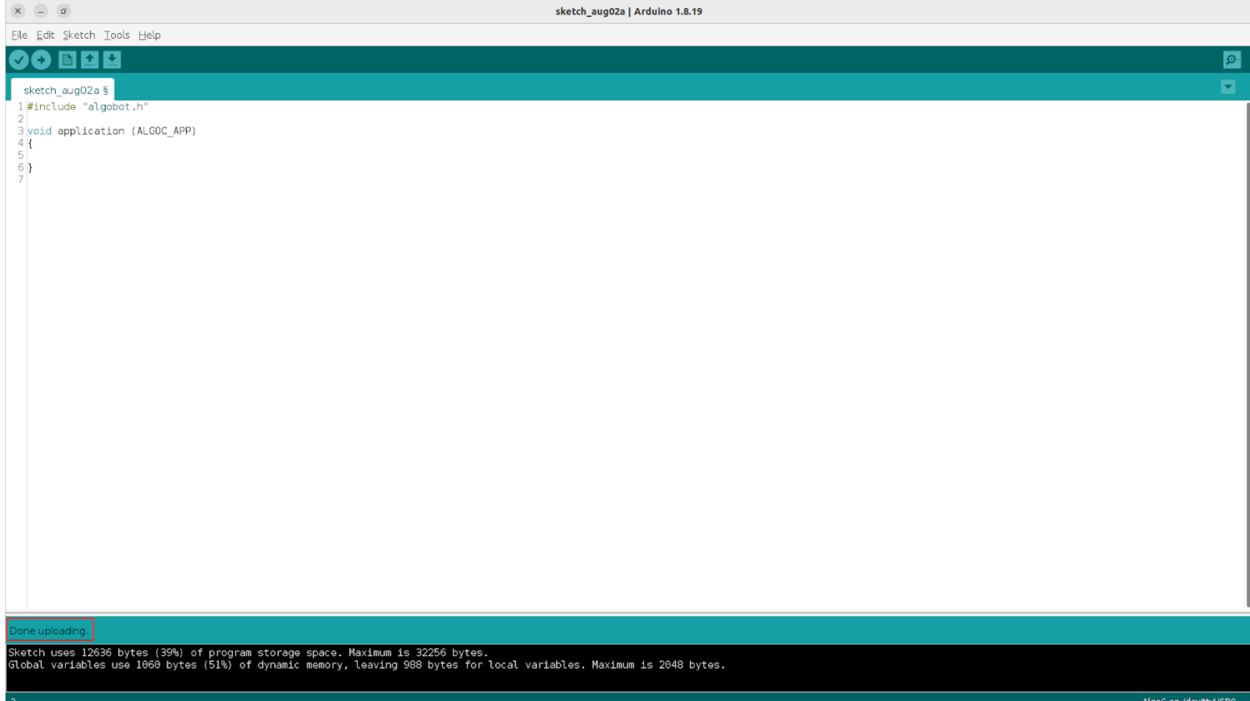
1. Make sure the brain block is connected to the battery block and it is turned on.
2. Connect the AlgoC board to your PC using the USB cable
3. Select the correct port by going to Tools->Port and selecting the board with the AlgoC name.



4. After you have selected the correct port, press the Upload button.



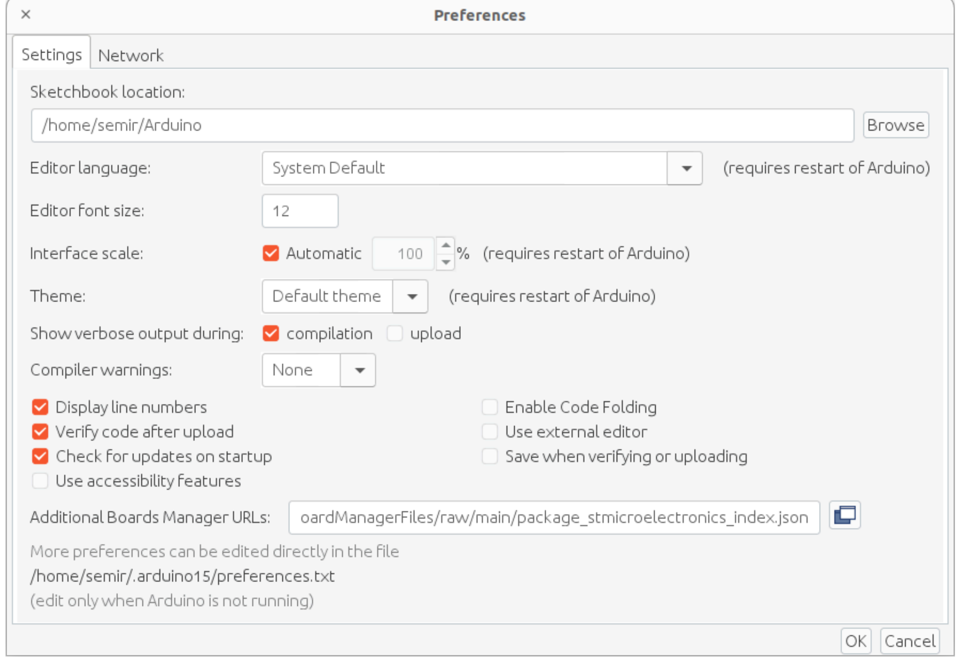
5. If the upload is successful you should see the following message



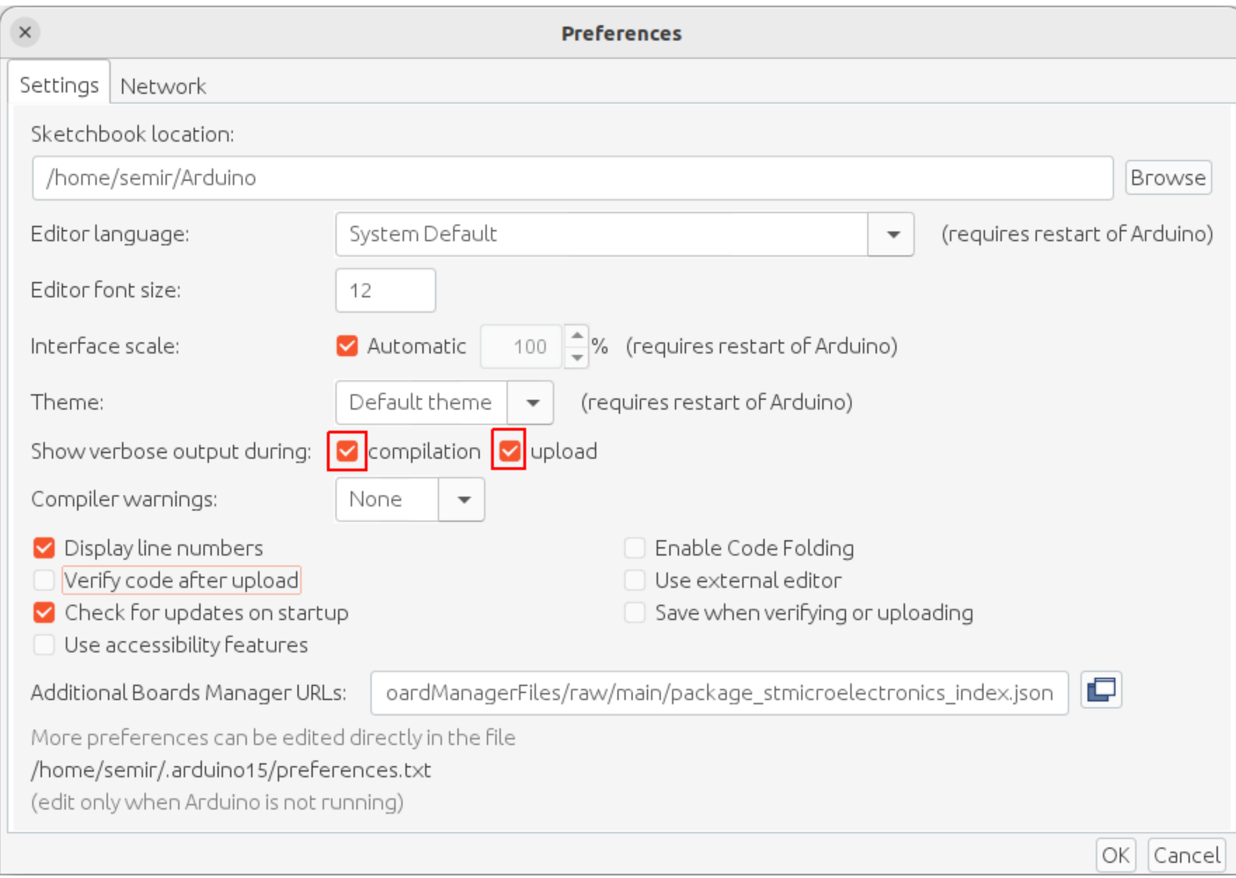
Improving the compile and upload experience

Sometimes the compilation and upload process might fail or it takes too much time to complete. To improve the user experience we recommend that you do the following.

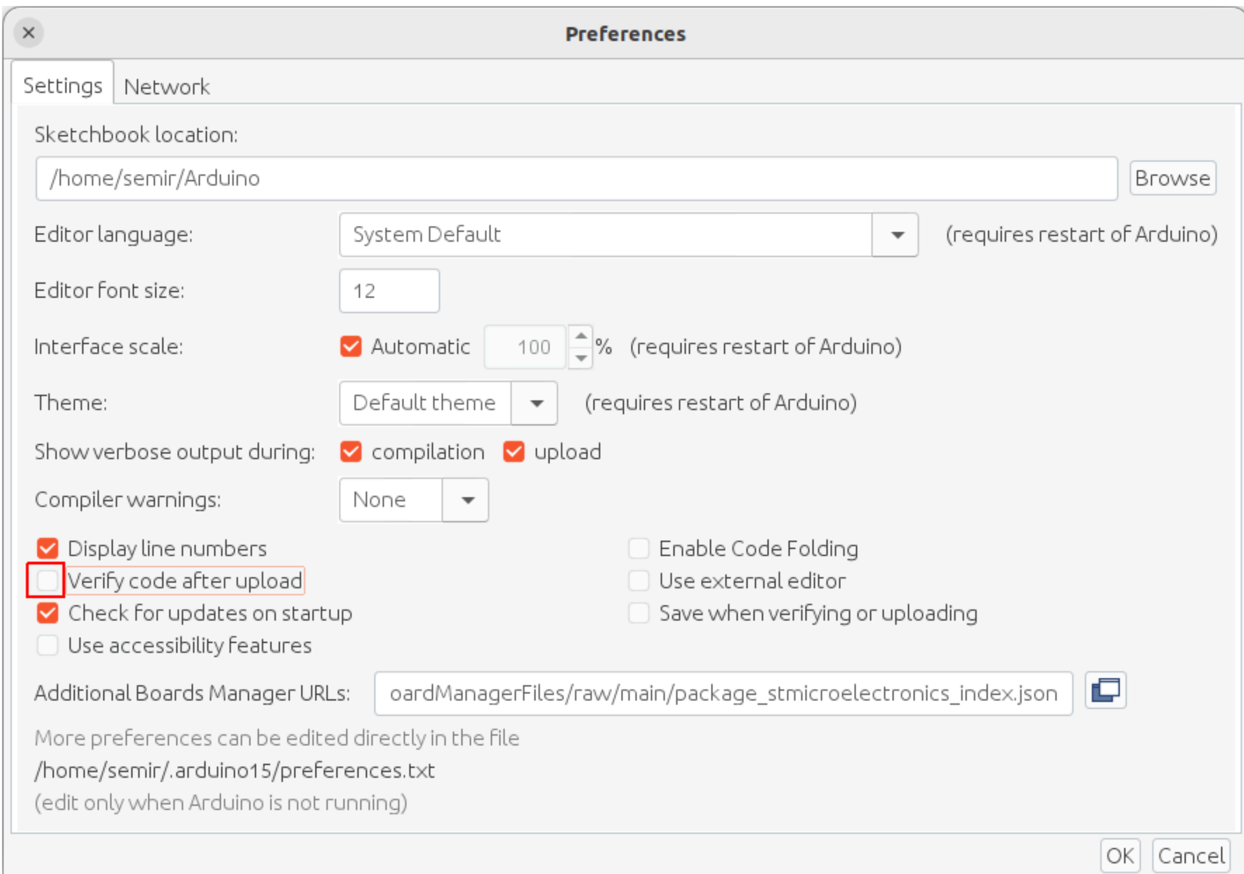
1. Go to the File->Preferences and you should see the following screen.



2. Check the *compilation* and *upload* box. This will give us more info during the compilation and upload process which can help us solve potential problems.



3. Uncheck the *Verify code after the upload* box which will improve the speed of the upload.



Troubleshooting Guide

During setup and use of the AlgoC board, you may encounter issues that prevent successful connection or operation. These problems can be caused by hardware, cables, drivers, or operating system compatibility. This section provides guidance on the most common issues and how to resolve them.

Cable Issues

Many USB cables are charge-only and do not support data transfer.

Symptom: The board powers on (LEDs light up), but no new serial port appears.

Solution: Use a proper USB data cable (often marked “sync cable”). If unsure, test with a cable that transfers files with a phone.

Driver Issues

Different operating systems handle USB-to-UART adapters differently:

Windows:

Most versions require a driver for the CH340 chip. Windows 10 and 11 often install this automatically, but if not, download from the WCH website.

macOS:

Big Sur (11.x) and newer: Built-in support for CH340. No extra driver required.

Catalina (10.15) and older: Requires installation of the official CH340 driver. After installation, a restart is mandatory.

If macOS blocks the driver, go to System Preferences → Security & Privacy → General and click Allow.

Linux:

The CH340 is supported natively on most modern distributions. If it does not appear, check permissions (dialout group) or try dmesg to confirm recognition.

Driver can be downloaded from: https://sparks.gogo.co.nz/ch340.html

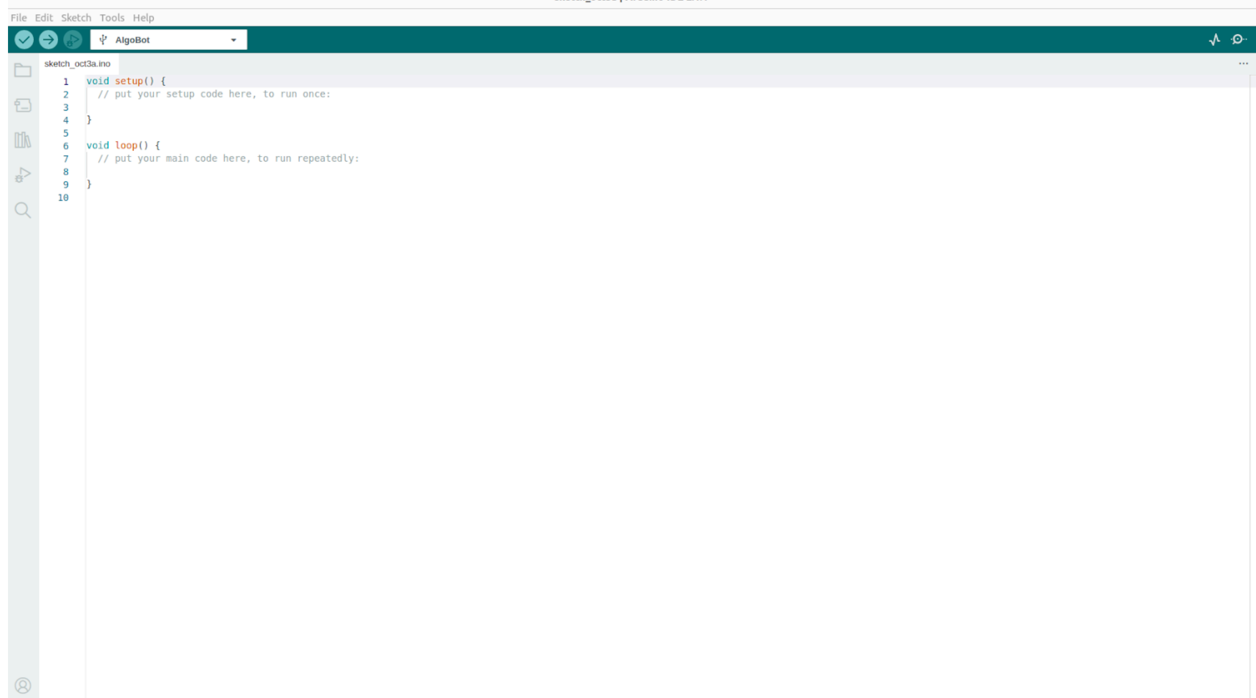
Uploading the Tangible code

If there is a need to revert to the original Tangible code and to use it with the GoAlgo application or Play device then we need to flash the original code. Information on how to do this will be presented below.

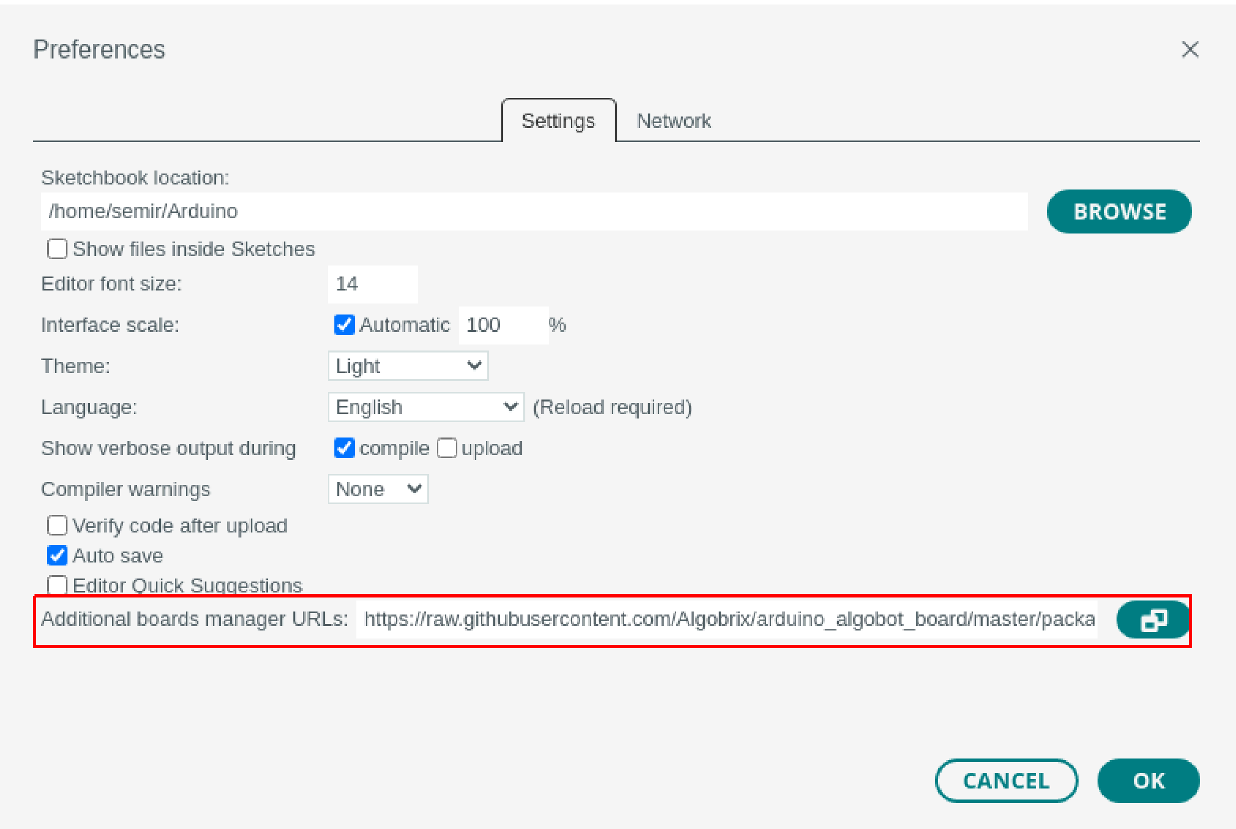
Installing the Algobrix

To flash the original tangible code we first need to install the Algobrix board. This step is similar to the step where we installed the AlgoC board.

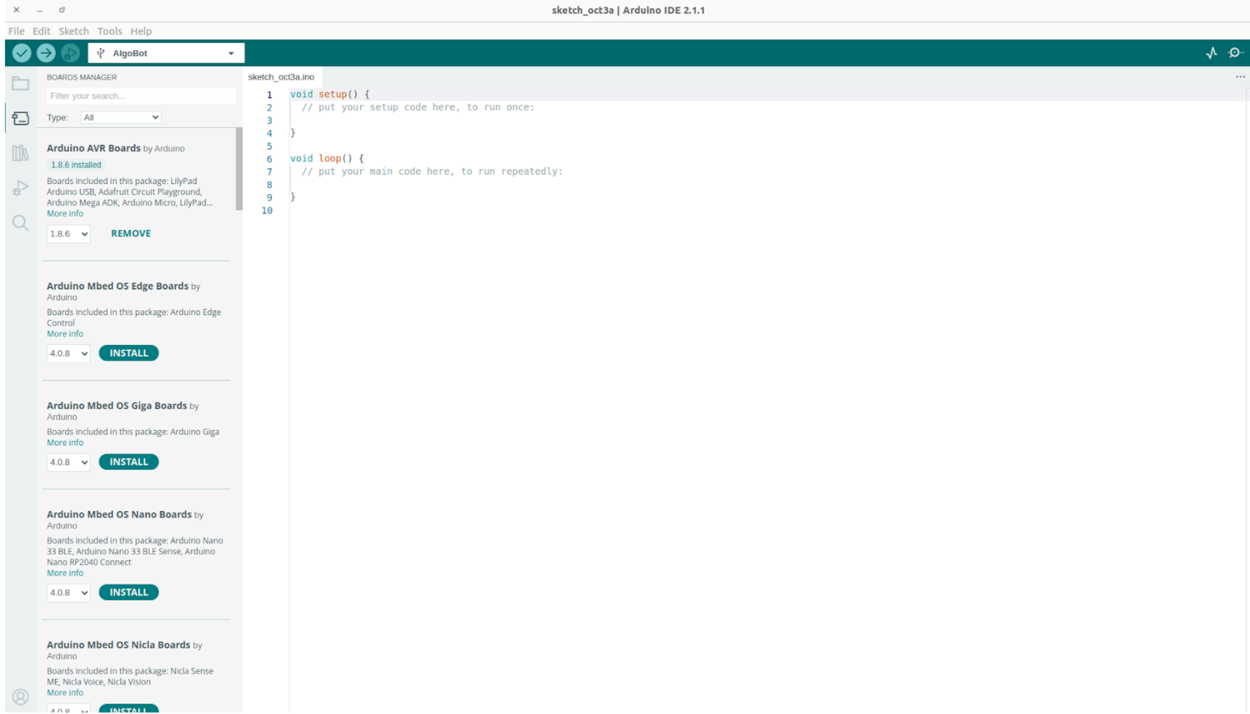
1. Open the Arduino IDE. You should see the following screen.



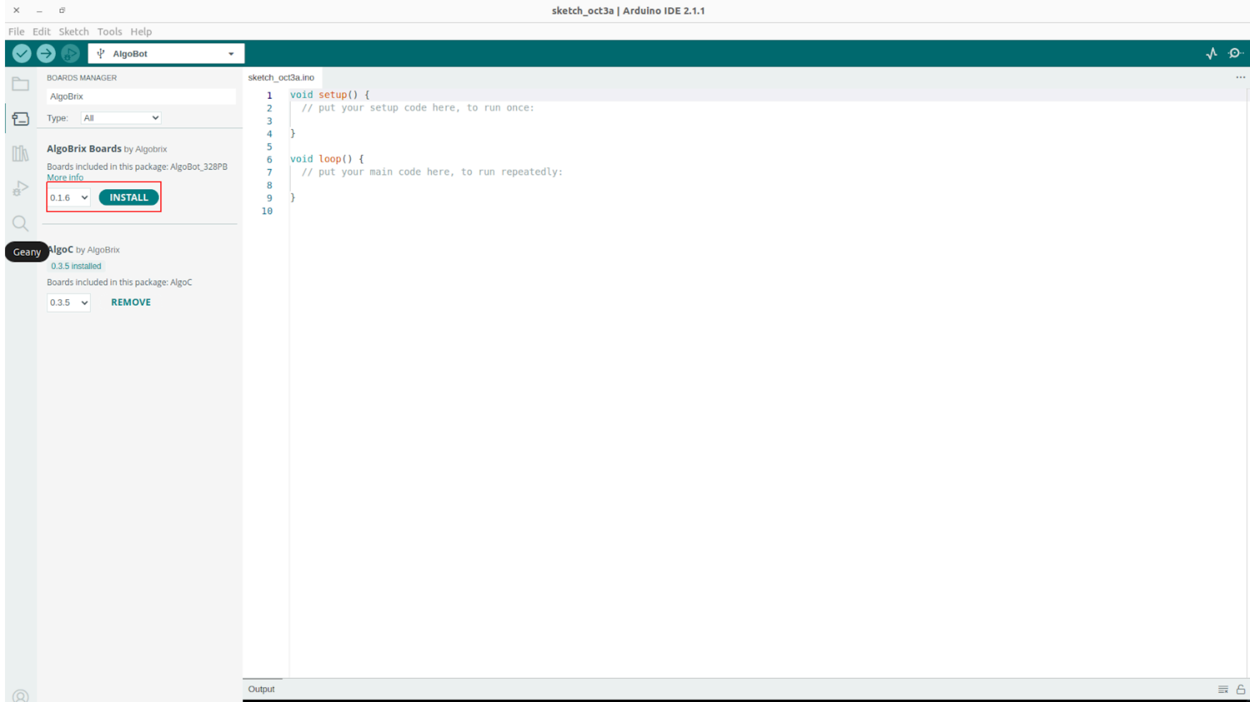
2. Go to the File->Preferences and inside the Additional Boards Manager URLs copy the following link and press OK: https://raw.githubusercontent.com/Algobrix/arduino_algobot_board/master/package_Algobot_index.json



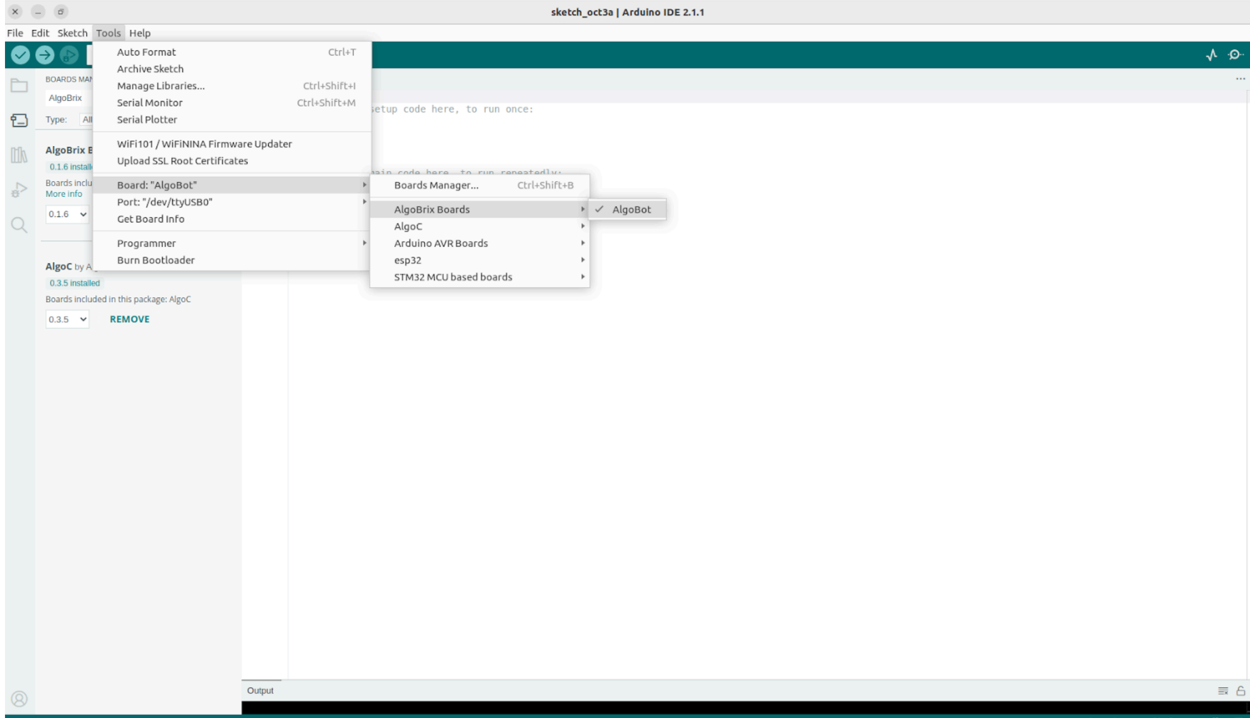
3. Now we need to install the SDK by going to the Tools->Board->Boards Manager. You should see the following screen.



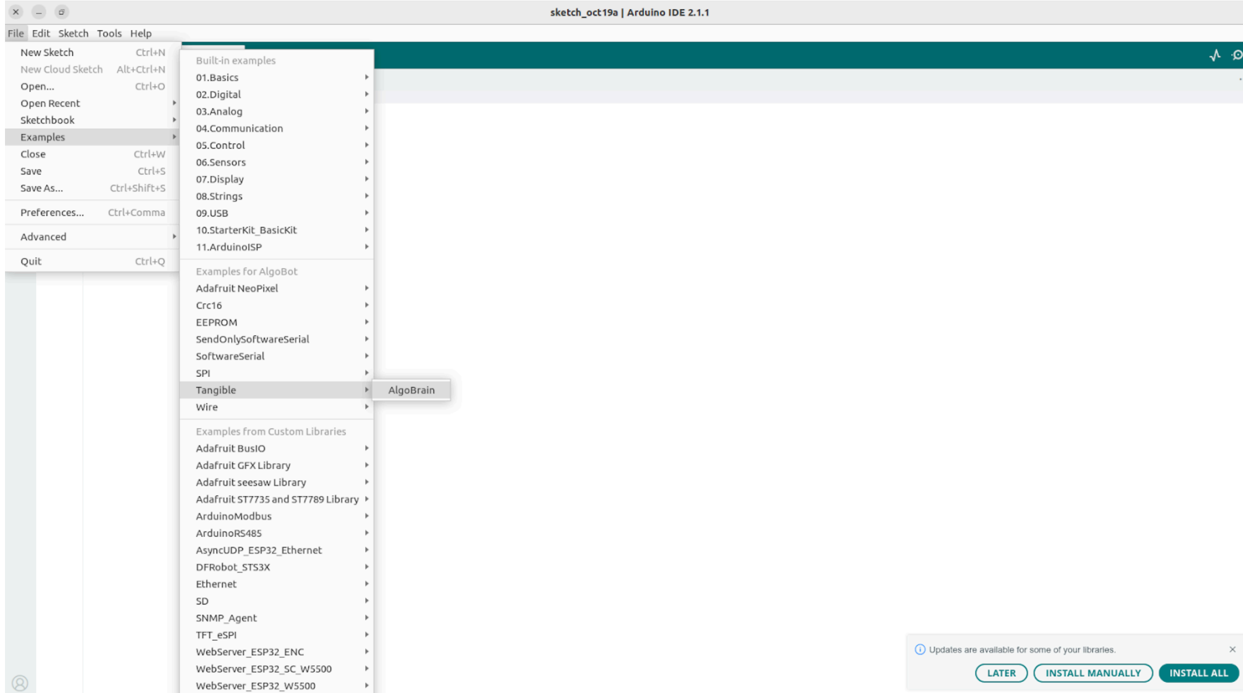
4. Search for the AlgoBrix. Select the latest version and press install.



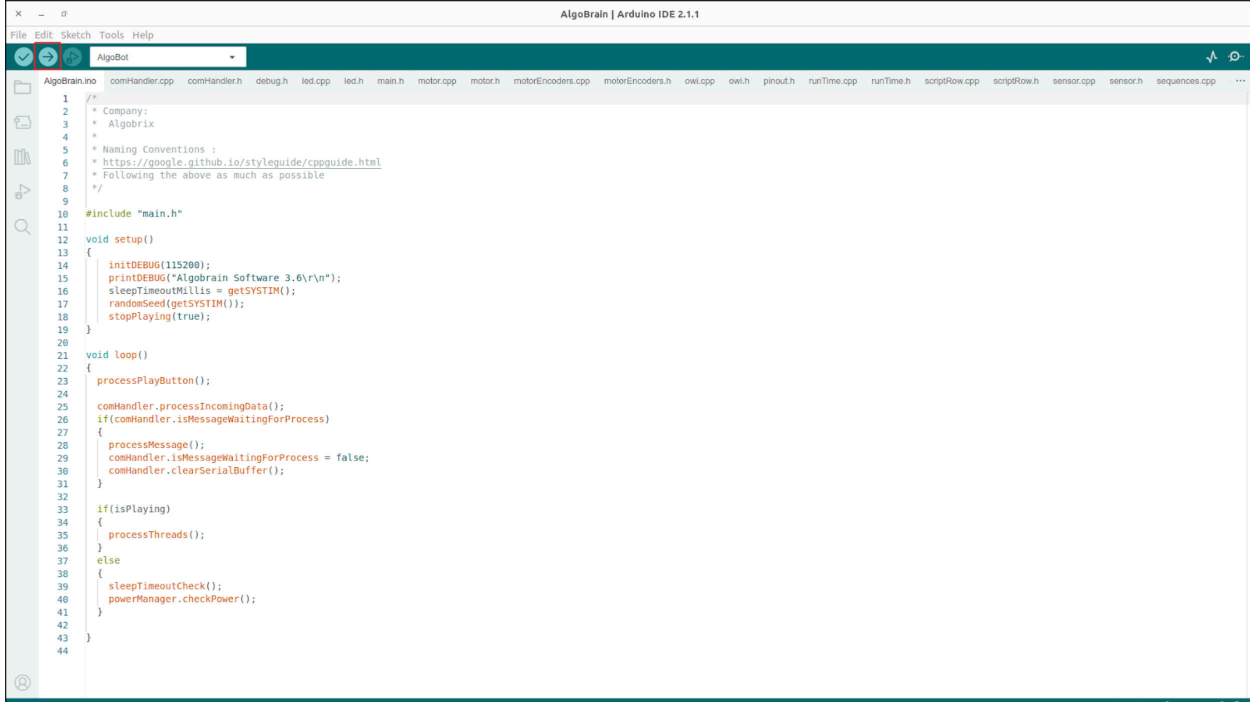
5. Now we need to select the AlgoBot board by going to the Tools->Board->AlgoBrix and selecting the AlgoBot



6. Next step is to open the Tangible example. Go to File->Example->Tangible->AlgoBrix



7. Upload the code by pressing the upload button.



Now you have Tangible code flashed to the Brain device and you can use it with the tangible Play device.

Software development kit (SDK)

User application

The user shall write the application inside the function *void application(ALGOC_APP)*:

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{

}
```

SDK allows writing two types of applications:

- Single Thread application
- Multi Threaded application

In addition, to achieve more control over the application each function can be declared blocking or non-blocking.

In the case of a Multithreaded application, declaring the function-blocking will only block the current thread and not other threads.

Users need to take special considerations when developing applications so as not to reach the end of the application too soon. For this purpose, the user needs to declare the functions blocking or non-blocking to achieve the desired behavior and not to reach the end of the application.

If we reach the end of the application function, all actuators and sensors will be stopped even though the actuators are still running.

This can be explained in the following example.

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
move(ALGOC,'A',2,10,CW,false);
```

```
light(ALGOC,1,1,10,"Red",true);
}
```

In this example, we will analyze the code line by line:

- We send the instruction to run motor A with power level 10 for 2 seconds in non-blocking mode. This means that we jump to the next line immediately.
- Immediately after the previous function, we send the instruction to turn light 1 to color “Red” with power 10 in blocking mode. This means that we don’t execute the next line until this one is finished.
- We wait until the Light command execution is completed.
- After 1 second the light command is done and we jump to the next command.
- Because we reached the end of the application, program execution was terminated and all of the actuators were stopped.

So based on this logic, we have the case that the Motor and Light start working at the same time and after 1 second when the light execution is completed motor is turned off as well as we have reached the end of the application function. So in this case, even though we have issued the command to run the motor for 2 seconds, the motor will run only for 1 second as we have reached the end of the application functions.

If we want to have the motor running for 2 seconds and the light for 1 second we need to rewrite the code as follows:

C/C++
```
#include <algoC.h>

void application(ALGOC_APP)
{
light(ALGOC,1,1,10,"Red",false);
move(ALGOC,'A',2,10,CW,true);
}
```

In this example, we will analyze the code line by line:

- We send the instruction to turn on Light 1 to color “Red” with power 10 for 1 second in non-blocking mode. Because of this, we jump to the execution of the next command immediately.
- We now control motor A for 2 seconds with power level 10 in a blocking mode.
- After running the motor for 1 second the light will turn off. The motor will continue for 1 second more.

- After 1 second the motor is turned off and we have reached the end of the application.

Lets now look at the following code:

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
move(ALGOC,'A',2,10,CW,false);
}
```

Based on what we have learned above, what should be the behavior of such code? Because we only have one function which is non-blocking, as soon as we run this function we have reached the end of the application() code and because of that, as we know, program execution is finished and all motors and sensors will be turned off.

This is happening so fast that we don't even see the motor running and we get the perception that the code doesn't work.

Hardware Abstraction Layer (HAL)

Currently, AlgoC SDK provides the following elements:

- Motor
- Light
- Wait
- Sound
- Stop

Motor

We can control any of the 3 available motor ports by using following functions:

move (System name, char motorPort, float seconds, float power, int direction, bool isBlocking)

This function enables us to control any port we want. Arguments for the function are:

- name – Name is always ALGOC
- motorPort – We can specify the following value 'A', 'B', or 'C'
- seconds – We can specify the time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the motor indefinitely.

- power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- direction – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

moveAB(System name,float seconds, float power, int direction, bool isBlocking)

This function enables us to control ports A and B at the same time. Arguments for the function are:

- name – Name is always ALGOC
- seconds – We can specify the time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the motor indefinitely.
- power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- direction – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

moveABC(System name,float seconds, float power, int direction, bool isBlocking)

This function enables us to control ports A, B, and C at the same time. Arguments for the function are:

- name – Name is always ALGOC
- seconds – We can specify the time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the motor indefinitely.
- power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- direction – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

rotations (System name, char motorPort, float rotations, float power, int direction, bool isBlocking)

This function enables us to control any port we want. Arguments for the function are:

- Name – Name is always ALGOC
- motorPort – We can specify the following values ‘A’, ‘B’, or ‘C’
- Rotation – We can specify the number of rotations for the desired motor. Value can be in the range of 0.1-1000.
- Power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- Dir – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false. If we don’t specify this value, the function will be by default BLOCKING.

rotationsAB(System name, float rotations, float power, int direction, bool isBlocking)

This function enables us to control ports A and B at the same time. Arguments for the function are:

- name – Name is always ALGOC
- rotation – We can specify the number of rotations for the desired motor. Value can be in the range of 0.1-1000.
- power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- direction – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.Alternatively, you can use 1 for clockwise (CW) and -1 for counterclockwise (CCW).
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

rotationsABC(System name, float rotations, float power, int direction, bool isBlocking)

This function enables us to control ports A, B, and C at the same time. Arguments for the function are:

- name – Name is always ALGOC
- rotation – We can specify the number of rotations for the desired motor. Value can be in the range of 0.1-1000.
- power – We can specify the power of the motor. Value for this argument can be in the range 0-10
- direction – We can specify the direction in which the motor will turn. We have two values at our disposal and they are: CW and CCW. The first one (CW) will run the motor clockwise and the second one will run it counterclockwise.

- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

Example 1: Run the motor A for exact number of seconds

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
    move(ALGOC,'A',5,10,CW,true);
}
```

Example 2: Start motor A in non-blocking mode with the period set to FOREVER. Run motor B in blocking mode for 4 seconds. When motor B completes, stop motor A. Both motors are running with power set to 5.

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
move(ALGOC,'A',FOREVER,5,CW,false);
move(ALGOC,'B',4,5,CW,true;
stopMotor(ALGOC,'A');
}
```

startCounting(System name, char motorPort, float & rotationCounter)

This function enables us to count the number of rotations of the desired motor. When we call this function we initiate the counting for the desired motor. Arguments for the function are:

- name – Name is always ALGOC
- motorPort – We can specify the following values ‘A’, ‘B’, or ‘C’
- rotationCounter – We pass the variable which will be updated with the current rotation count.

After we have initiated the counting, we can then at any moment in time get the current number of rotations by simply reading the content of the passed variable rotationCounter

Example 1: Read the number of rotations after 1 second of running the motor.

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
float rotationCounter = 0;
startCounting(ALGOC,'A',rotationCounter);
move(ALGOC,'A',3,5,CW,true);
wait(ALGOC,1);
Serial.print("Number of rotations made by motorA after 1 second: ");
Serial.println(rotationCounter);
wait(ALGOC,2);
stopCounting(ALGOC,'A');
}
```

NOTE: In a multithreading application, if we call the StartCounting() function from multiple threads then the call will be discarded and we will start counting from the latest call of the function.

NOTE: The number of rotations is counter regardless of the direction. So if we start counting and we go 1 rotation clockwise and then 1 rotation counterclockwise, the value we are going to read from the dedicated variable is 2.

stopCounting(System name, char motorPort)

We call this function to stop counting rotations for the desired motor. Arguments for the function are:

- name – Name is always ALGOC
- motorPort – We can specify the following values ‘A’, ‘B’, or ‘C’

isMotorBusy (System name, char motorPort)

This function fetches details about the motor, specifically indicating whether the motor is running. The parameters for this function are::

- name – Name is always ALGOC
- motorPort – We can specify the following values ‘A’, ‘B’, or ‘C’

resistanceToStop (System name, char motorPort, float threshold)

In a situation when the Motor stalls for whatever reason, AlgoC provides the functionality that automatically stops the motor in this situation. By default, if the motor speed stops by 50% from the initial speed, the system will mark this situation as a stall

of the motor. Using this function we can change this threshold. The parameters for this function are::

- name – Name is always ALGOC
- motorPort – We can specify the following values 'A', 'B', or 'C'
- threshold – Threshold in range 0-1.

Light

We can control any of the 2 available light ports by using the following functions:

light (System name, int lightPort, float seconds, float power, char * Color, bool isBlocking)

This function enables us to control any port we want. Using this function we can only use predefined colors. Arguments for the function are:

- name – Name is always ALGOC
- port – We can specify the following values: 1 or 2
- time – We can specify time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the lights indefinitely.
- power – We can specify the power of the light. Value for this argument can be in the range 0-10
- color – We can specify the color of the light. We have the following values at our disposal:
    - "White",
    - "Red",
    - "Green",
    - "Blue",
    - "Purple",
    - "Yellow",
    - "Orange",
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

light12(System name, float seconds, float power, char * color, bool isBlocking)

This function enables us to control ports 1 and 2 at the same time. Using this function we can only use predefined colors. Arguments for the function are:

- name – Name is always ALGOC
- time – We can specify time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the lights indefinitely.
- power – We can specify the power of the light. Value for this argument can be in the range 0-10

- color – We can specify the color of the light. We have the following values at our disposal:
    - “White”,
    - “Red”,
    - “Green”,
    - “Blue”,
    - “Purple”,
    - “Yellow”,
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

RGB(System name,int lightPort, float seconds, float power,int R, int G, int B, bool isBlocking)

This function enables us to control any port we want. Using this function we can get any color by specifying the value for the RGB channels. Arguments for the function are:

- name – Name is always ALGOC
- port – We can specify the following values: 1 or 2
- time – We can specify time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the lights indefinitely.
- power – We can specify the power of the light. Value for this argument can be in the range 0-10
- R – We can specify the channel R value for the desired color. This value has to be in the range 0-255
- G – We can specify channel G value for the desired color. This value has to be in the range 0-255
- B – We can specify the channel B value for the desired color. This value has to be in the range 0-255
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

RGB12(System name, float seconds, float power,int R, int G, int B, bool isBlocking)

This function enables us to control ports 1 and 2 at the same time. Using this function we can get any color by specifying the value for the RGB channels. Arguments for the function are:

- name – Name is always ALGOC
- time – We can specify time in seconds. Value can be in the range of 0.1-10. Additionally, we can use FOREVER value to run the lights indefinitely.
- power – We can specify the power of the light. Value for this argument can be in the range 0-10

- R – We can specify the channel R value for the desired color. This value has to be in the range of 0-255
- G – We can specify channel G value for the desired color. This value has to be in the range 0-255
- B – We can specify the channel B value for the desired color. This value has to be in the range 0-255
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

Example 1: Create a breath-like effect by changing the power of the Light from 0 to 10.

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
light(ALGOC,1,0.1,0,"Red",true);
light(ALGOC,1,0.1,1,"Red",true);
light(ALGOC,1,0.1,2,"Red",true);
light(ALGOC,1,0.1,3,"Red",true);
light(ALGOC,1,0.1,4,"Red",true);
light(ALGOC,1,0.1,5,"Red",true);
light(ALGOC,1,0.1,6,"Red",true);
light(ALGOC,1,0.1,7,"Red",true);
light(ALGOC,1,0.1,8,"Red",true);
light(ALGOC,1,0.1,9,"Red",true);
light(ALGOC,1,0.1,10,"Red",true);
light(ALGOC,1,0.1,9,"Red",true);
light(ALGOC,1,0.1,8,"Red",true);
light(ALGOC,1,0.1,7,"Red",true);
light(ALGOC,1,0.1,6,"Red",true);
light(ALGOC,1,0.1,5,"Red",true);
light(ALGOC,1,0.1,4,"Red",true);
light(ALGOC,1,0.1,3,"Red",true);
light(ALGOC,1,0.1,2,"Red",true);
light(ALGOC,1,0.1,1,"Red",true);
light(ALGOC,1,0.1,0,"Red",true);
}
```

Example 2: Create a police-like effect by changing the power of the Light from 0 to 10.

```
C/C++
#include <algoC.h>
```

```
void application(ALGOC_APP)
{
uint8_t k = 0;
for( k = 0; k < 10; k++)
{
light(ALGOC,1,0.2,10,"Red",true);
light(ALGOC,2,0.2,10,"Blue",true);
}
}
```

isLightBusy (System name, int lightPort)

This function fetches details about the light specifically indicating whether the light is running. The parameters for this function are::

- name – Name is always ALGOC
- lightPort – We can specify the following values: 1 or 2

Sound

AlgoC only has 1 port dedicated to the sound player. Using the following function we can control this port:

playSound(System name,int sound,float power,bool isBlocking);

Using this function we can play the desired sound with the desired power. Arguments for this function are:

- name – Name is always ALGOC
- sound – We can specify what sound we want to play. The value of this argument has to be in the range 1-15 or you can specify the name of the sound. Available sounds can be found by calling the function *listAvailableSounds(ALGOC)*.
- volume – We can specify the volume of the sound. Value for this argument can be in the range 0-10
- isBlocking – We can specify if the call to u function will be blocking or not. This parameter accepts two values and they are: true or false.

Example 1: Play bird sound on startup. This will only once at the start of the code execution.

```
C/C++
#include <algoC.h>
void application(ALGOC_APP)
{
```

```
playSound(ALGOC,3,BIRD,true);
}
```

listAvailableSounds(System name);

Using this function we can play the desired sound with the desired power. Arguments for this function are:

- name – Name is always ALGOC

Example 1: List available sounds.

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
    listAvailableSounds(ALGOC);
}
```

When running the previous code, you should see the following inside the Serial monitor.

```
C/C++
[1] – SIREN
[2] – BELL
[3] – BIRD
[4] – BEAT
[5] – DOG
[6] – MONKEY
[7] – ELEPHANT
[8] – APPLAUSE
[9] – VIOLIN
[10] – GUITAR
[11] – ROBOT_LIFT
[12] – TRUCK
[13] – SMASH
[14] – CLOWN
[15] – CHEERING
```

isSoundBusy (System name)

This function fetches details about the sound specifically indicating whether the sound is playing. The parameters for this function are:

- name – Name is always ALGOC

Stop

We can stop the execution of the Light, Motor, or Sound by using the following functions.

stopLight(System name,port)

This function enables us to stop current Light execution on the desired port. Arguments for the function are:

- name – Name is always ALGOC
- port – We can specify the following values: ‘1’ or ‘2’

stopMotor(System name,port)

This function enables us to stop the current Motor execution on the desired port. Arguments for the function are:

- name – Name is always ALGOC
- port – We can specify the following values: ‘A’, ‘B’, or ‘C’

stopSound(System name)

This function enables us to stop current Sound execution on the desired port. Arguments for the function are:

- Name – Name is always ALGOC

Example 1: Stop the sound player after 1 second.

```C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
playSound(ALGOC,1,3,false);
wait(ALGOC,1);
stopSound(ALGOC);
}
```

Wait

We can use the wait function to create the pause in the code execution. For this purpose we have the following function at our disposal.

wait(System name,float seconds)
This function enables us to create a pause in the program execution for the desired number of seconds. Arguments for the function are:

- name – Name is always ALGOC
- seconds – We can specify the time in seconds. Value can be in the range 0.1-10

Wait Sensor

We can use the waitSensor function to wait for the specific event that can be detected by the sensors. The following functions are at our disposal:

Currently ALGOC supports following sensors:

Sensor	Min value	Max value	Levels	Note
Sound sensor	0	2	3	Sound sensor detects 3 levels of sound. Lowest level has value 0 and max level has value 2
Distance sensor	1	9	9	Distance sensor detects 10 levels of distance. Distance sensor outputs value 1 when the object is really close to the sensor. The sensor will return -1 if the

				object is out of the sensor's range.
Touch sensor	0	1	2	Touch sensor has only 2 levels (ON-OFF states)

waitSensor(System name,int sensorPort, int minSignalValue, int maxSignalValue)

This function enables us to create a pause in the program execution until we have detected the value that belongs to the range specified in the call of the function. When the value detected belongs to the specified range, program execution will continue, otherwise, it will continue to block program execution. Arguments for the function are:

- name – Name is always ALGOC
- sensorPort – We can specify the following values: '1' or '2'
- minSignalValue – Lower limit of the range we want to detect
- maxSignalValue – Higher limit of the range we want to detect

This function returns the value detected by the sensor.

waitForPressSensor(System name,int sensorPort, bool logicState)

This function enables us to create a pause in the program execution until we have detected the exact value. Arguments for the function are:

- name – Name is always ALGOC
- sensorPort – We can specify the following values: '1playButtonReset' or '2'
- logicState – The sensor will wait to read this value. If the sensor reads this value function will continue program execution, otherwise it will continue to block code execution

This function returns the value detected by the sensor.

Example 1: If the sensor value is in range 1-4 light will become Red. If the sensor value is in the range 5-10 turn off the light.

C/C++

include

```C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
int value;
value = waitSensor(ALGOC,'1',1,4);
light(ALGOC,'1',FOREVER,10,"Red",false);
value = waitSensor(ALGOC,'1',5,10);
stopLight(ALGOC,'1');
}
```

Get Sensor

In the above section, we saw how we can stop the program execution while the desired value from the sensor is received. We can also just get the value of the sensor without stopping the code execution.

getSensor(System name,int sensorPort)

This function enables us to retrieve the current value of the sensor.

- name – Name is always ALGOC
- sensorPort – We can specify the following values: ‘1’ or ‘2’

This function returns the value detected by the sensor at the moment of a call to a function. Function returns negative value if measured value is out of the range of the sensor used or if there is some other error inside the sensor.

Play button input

We can use the play button input to give the user capability to interact with the brain through the play button. The following functions are at our disposal:

playButtonReset(System name)

The Play button has a default functionality to Play, Stop and Pause code execution. Once we execute a command through which we would like to utilize the play button in different way the default functionality is disabled. To enable the default functionality we call this function. Arguments for the function are:

- name – Name is always ALGOC

There is no return value.

waitPlayButtonPress(System name,int numberOfPresses)

This function enables us to create a pause in the program execution until we have detected the exact number of presses of the Play button. Arguments for the function are:

- name – Name is always ALGOC
- numberOfPresses – Specify for how many presses function waits

This function returns number of play button presses

getPlayButtonState(System name)

This function enables us to get the current state of the Play Button. Arguments for the function are:

- name – Name is always ALGOC

This function returns current Play button state

Example 1: Wait for 3 presses and turn light 1 to Blue. Then, wait for one press after which application is completed.

```C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
waitPlayButtonPress(ALGOC,3);
light(ALGOC,'1',FOREVER,10,"Blue",false);
repeat(ALGOC,1)
{
if(getPlayButtonState(ALGOC) == true)
{
break;
}
}
playButtonReset(ALGOC);
}
```

Break Point

We can use the Break Points functions to stop the code execution at a specific moment. For this purpose we have the following function at our disposal.

breakPoint(System name)

This function enables us to stop the code execution when the MCU reaches this line. This function doesn't have any arguments and to continue code execution we need to enter any char through the Serial Monitor.

- name – Name is always ALGOC

breakPoint(System name,char breakChar)

This function enables us to stop the code execution when the MCU reaches this line. This function had one argument and to continue code execution we need to enter a character which is specified in the call of this function through the Serial Monitor.

- name – Name is always ALGOC
- breakChar – We can specify which character will continue code execution

Example 1: Stop the code execution in the middle of the breath effect. Continue code execution when we receive character 'c' through the Serial Monitor.

```C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
light(ALGOC,1,0.1,0,"Red",true);
light(ALGOC,1,0.1,1,"Red",true);
light(ALGOC,1,0.1,2,"Red",true);
light(ALGOC,1,0.1,3,"Red",true);
light(ALGOC,1,0.1,4,"Red",true);
light(ALGOC,1,0.1,5,"Red",true);
light(ALGOC,1,0.1,6,"Red",true);
light(ALGOC,1,0.1,7,"Red",true);
light(ALGOC,1,0.1,8,"Red",true);
light(ALGOC,1,0.1,9,"Red",true);
light(ALGOC,1,0.1,10,"Red",true);
breakPoint(ALGOC,'c');
light(ALGOC,1,0.1,9,"Red",true);
light(ALGOC,1,0.1,8,"Red",true);
light(ALGOC,1,0.1,7,"Red",true);
light(ALGOC,1,0.1,6,"Red",true);
light(ALGOC,1,0.1,5,"Red",true);
light(ALGOC,1,0.1,4,"Red",true);
light(ALGOC,1,0.1,3,"Red",true);
light(ALGOC,1,0.1,2,"Red",true);
light(ALGOC,1,0.1,1,"Red",true);
```

```
light(ALGOC,1,0.1,0,"Red",true);
}
```

Random number generator

In addition to functions used for controlling the actuators and reading the values from the sensor, we have created a function which can generate random numbers for you in a range from 0-10.

random(System name)

This function enables us to get random number in a range from 0-10.

- name – Name is always ALGOC

Repeat

Sometimes we have a need to repeat the desired part of the code with some conditions. For example, we could want to repeat some code 10 times, or to run the specified sets of commands while some condition is met. For this purpose, we have the *Repeat command.*

Notice that by default the Repeat function is Blocking. In order to use it in a non-blocking state use the multithreading special syntax.

repeat(System name,bool condition)

This command enables us to repeat the desired set of commands while conditions are met.

- name – Name is always ALGOC
- condition – Code will repeat while this condition is true, otherwise when false it will stop repeating the code.

Example 1: Execute the code 5 times

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
int counter = 5;
repeat(ALGOC,counter != 0)
{
move(ALGOC,'A',5,10,CW,true);
light(ALGOC,'1',0.1,10,"Red", true);
```

```
counter--;
}
}
```

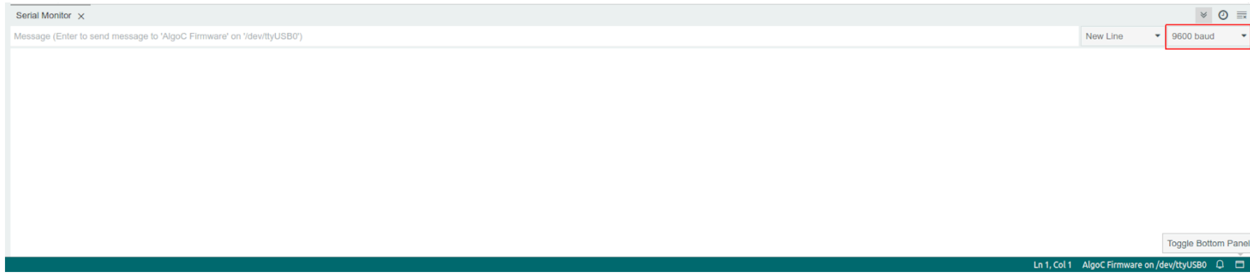
Write & Enter

As a developer, we have a powerful tool at our disposal, and that is the Serial monitor. This Serial monitor enables us to print messages about the current status, sensor values, etc, and also to get some input from the user.

We can open Serial Monitor by clicking on Serial Monitor under Tools which is located in the menu bar or by clicking on the button marked with a red rectangle in the following image:

After clicking on the button, at the bottom of the Arduino IDE you will see the following:

Before we start writing and receiving messages through the Serial monitor we need to configure the baud rate to 115200 baud. You can do this by clicking on the dropdown menu marked with the red rectangle on the previous image and selecting the specified baud rate.



For this, we have the following functions at our disposal:

write(System name, line)

This function enables us to write any basic data types available in C/C++. For this function, it is important to notice that after the data we print on the Serial monitor, we are staying in the same row.

- name – Name is always ALGOC
- line – Data we want to print out.

writeLine(System name, line)

This function enables us to write any basic data types available in C/C++. For this function, it is important to notice that after the data we print on the Serial monitor, we move to the next row.

- name – Name is always ALGOC
- line – Data we want to print out.

enterNumber(System name)

This function enables us to get the number from the user through a Serial monitor. This function returns a float type of data.

- name – Name is always ALGOC

enterChar(System name)

This function enables us to get the character from the user through a Serial monitor. This function returns a char type of data.

- name – Name is always ALGOC

enterString(System name)

This function enables us to get the string from the user through a Serial monitor. This function returns a char * type of data.

- name – Name is always ALGOC

Example 1: Write & Enter example

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
//enterNumber() enterString() enterChar()
writeLine(ALGOC,"enter number");
int number = enterNumber(ALGOC);
write(ALGOC,"the number you entered is ");
writeLine(ALGOC,number);

writeLine(ALGOC,"enter char");
char oneChar = enterChar(ALGOC);
write(ALGOC,"the char you entered is ");
writeLine(ALGOC,oneChar);

writeLine(ALGOC,"enter string");
char *multipleChars = enterString(ALGOC);
write(ALGOC,"the string you entered is ");
writeLine(ALGOC,multipleChars);
}
```

Multithreading

The AlgoC SDK supports both single-threaded and multi-threaded applications, providing developers with the flexibility to execute multiple tasks concurrently. This is particularly useful when you want different actions (like controlling motors, lights, or playing sounds) to run at the same time without waiting for each action to complete sequentially. We have following types of applications AlgoC applications:

1. Single-Threaded Application: In this type of application, all code runs in a single thread.
2. Multi-Threaded Application: In a multi-threaded application, multiple tasks (threads) run concurrently. Each thread can execute independently, allowing for more responsive and efficient code. Threads can be managed using special macros like START_THREAD, END_THREAD, START_SUBTHREAD, END_SUBTHREAD, etc. Every multithreading function must add those Macros to perform as a multithreading.

To create a multi-threaded application in AlgoC:

1. Declare a Function: Define separate functions for each threaded/sub-thread function.
    - Single thread function declaration: function(ALGOC_APP)
    - Threaded/sub-threaded function declaration: function()
2. Use Thread Management Macros:
    - Single Threads Flow:
      START_THREAD(thread_name) and END_THREAD(thread_name) to define the start and end of a thread, while "thread_name" must be any of the following:
      thread0, thread1, thread2… thread15
    - Nested/Sub Threads:
      In the same way, use START_SUBTHREAD(parent_thread, this_thread) and END_SUBTHREAD(parent_thread, this_thread) to manage sub-threads.
3. Invoke Threads in the Main Application: Call the thread functions from within the application(ALGOC_APP) function, to run them concurrently. To call a thread function, use the thread() command.
4. Control Loops: Calling the repeat() will block the other threads from running. Thus, use START_LOOP(num_of_iterations) and END_LOOP() to repeat certain actions within a thread. Forever loop is not optional at this stage.

Thread Management in AlgoC

AlgoC supports a maximum of 16 threads or sub-threads, each tied to a predefined thread variable (thread0 to thread15).

- Use START_THREAD(thread_name), END_THREAD(thread_name) for regular threads.
- START_SUBTHREAD(parent_thread_name, this_thread_name), an END_SUBTHREAD(parent_thread_name, this_thread_name), each thread must reference one of these variables, i.e., START_THREAD(thread3), END_THREAD(thread3).

Each thread or sub-thread must have its own unique thread variable, however, thread0 doesn't necessarily need to come before thread15. For instance, if you are using two threads, the first thread might use thread2 and the second might use thread1. This structure ensures controlled execution, allowing concurrent operations within the application while respecting the thread limit.

Example 1: Threads example

include

```c
C/C++
#include <algoC.h>

void thread1_run(void);
void thread2_run(void);

void application(ALGOC_APP)
{
thread1_run();
thread2_run();
}

void thread1_run(void)
{
START_THREAD(thread1);
move(ALGOC, 'A', 3, 5, CW, true);
END_THREAD(thread1);
}

void thread2_run(void)
{
START_THREAD(thread2);
move(ALGOC, 'B', 3, 5, CCW, true);
END_THREAD(thread2);
}
```

This example creates two threads that control two motors (A and B).

1. Thread 1 (thread1_run): Starts and moves motor A clockwise (CW) for 3 seconds with power level 5.
2. Thread 2 (thread2_run): Starts and moves motor B counterclockwise (CCW) for 3 seconds with power level 5.

Both threads run independently, and the motors perform their actions at the same time.

Nested threads/subthreads

In AlgoC, subthreads are created within a parent thread using two specific macros:

- START_SUBTHREAD(parent_thread, this_thread): This marks the beginning of a subthread under the control of the parent thread.
- END_SUBTHREAD(parent_thread, this_thread): This marks the end of the subthread.

```C/C++
#include <algoC.h>

void parent_thread_run(void);
void sub_thread_run(void);

void application(ALGOC_APP)
{
parent_thread_run();
}

void parent_thread_run(void)
{
START_THREAD(thread0);
move(ALGOC, 'A', 4, 6, CW, false);
sub_thread_run();
wait(ALGOC, 4);
END_THREAD(thread0);
}

void sub_thread_run(void)
{
START_SUBTHREAD(thread0, thread1);
light(ALGOC, 1, 2, 10, "Green", true);
END_SUBTHREAD(thread0, thread1);
}
```

Explanation:

1. Creating the Subthread:
    - START_SUBTHREAD(thread0,thread1) is used to begin the subthread. It takes the name of the parent thread (thread0) and this subthread (thread1).
    - Inside the subthread, we have the action of turning on the light.
2. Running the Subthread:
    - While the parent thread is running (moving motor A), the subthread runs concurrently. In this case, the subthread turns on a green light for 2 seconds while the motor is moving.
3. Ending the Subthread:
    - END_SUBTHREAD(thread0,thread1) marks the end of the subthread, completing the subtask.
4. Concurrency:

- Since the motor in the parent thread runs in non-blocking mode, the parent thread doesn't wait for it to finish before starting the subthread. Therefore, the motor and the light operate simultaneously.

Loops in threads

In AlgoC, you can use START_LOOP(n) and END_LOOP() macros to repeat actions within a thread a specific number of times. This is especially useful when you want certain tasks to execute repeatedly, such as blinking lights, moving motors, or playing sounds. These macros define the boundaries of the loop, where n is the number of iterations. At the moment the maximum number of repeats inside threads is limited to 255.

Key Points:

- START_LOOP(n): This starts a loop that will repeat the enclosed code n times.
- END_LOOP(): This ends the loop, after which the thread continues with any remaining code.

```C/C++
#include <algoC.h>

void thread_one_run(void);
void thread_two_run(void);

void application(ALGOC_APP)
{
thread_one_run();
thread_two_run();
}

void thread_one_run(void)
{
START_THREAD(thread0);
START_LOOP(3);
move(ALGOC, 'A', 2, 5, CW, true);
wait(ALGOC, 1);
END_LOOP();
END_THREAD(thread0);
}

void thread_two_run(void)
{
START_THREAD(thread1);
START_LOOP(3);
```

```
light(ALGOC, 1, 1, 10, "Red", true);
wait(ALGOC, 1);
END_LOOP();
END_THREAD(thread1);
}
```

What This Example Does:

- Thread One (thread_one_run):
    - Runs a loop three times.
    - In each iteration, it moves motor A for 2 seconds with power level 5 and then waits for 1 second.
- Thread Two (thread_two_run):
    - Runs a loop three times.
    - In each iteration, it turns on light 1 with red color for 1 second and then waits for 1 second.

Loops

Loops in programming allows you to repeat a block of code multiple times. This is particularly useful when you want to perform a task repeatedly, like counting numbers, processing items in a list, or performing an action until a certain condition is met.

They help reduce code duplication and make programs more efficient. Instead of writing the same code over and over, you write it once inside a loop, and the loop takes care of repeating it as needed. Loops also make code easier to maintain and update since changes need to be made only in one place.

We will cover 2 types of loops:

- for
- while

For Loop

A for loop is used when you know in advance how many times you want to repeat a block of code. It's a great way to count through numbers or repeat actions a specific number of times. The loop has three main parts: initialization, condition, and update. Each part helps control the loop's behavior:

```
C/C++
for (int i = 0; i < 5; i++) {
isStopDetected(ALGOC);  // Check for stop signal
```

```
    // Code to repeat
}
```

1. Initialization (int i = 0): Starts a counter variable, here set to 0.
2. Condition (i < 5): The loop runs as long as this condition is true.
3. Update (i++): Adjusts the counter after each loop cycle, moving it closer to ending the loop.

In this example, the code inside the loop will repeat five times.

While Loop

A while loop is best when you want to repeat code based on a condition that might change during the loop. It continues until the condition becomes false. If the condition starts as false, the code inside won't run at all.

```
C/C++
int i = 0;
while (i < 5) {
isStopDetected(ALGOC);  // Check for stop signal
      // Code to repeat
      i++;
}
```

1. Condition (i < 5): Checked before each loop cycle. If true, the loop runs.
2. Update (i++): This should be included inside the loop body to prevent an endless loop.

This loop also runs five times, like the for loop example, but uses a while structure instead.

Stopping the application execution from within the loop

In AlgoC, it is essential to include the isStopDetected(ALGOC); function inside loops to provide a way to stop the application from running, either through the serial monitor or user button. If this function is missing, the application will continue running indefinitely, without an option to stop it.

The isStopDetected(ALGOC); function is essential in AlgoC applications because it checks if a stop signal has been received, allowing the loop to exit when a stop condition is detected. For this reason, isStopDetected(ALGOC); should be placed inside

every loop, particularly in cases of infinite loops or any repeated actions, ensuring the program can be interrupted if needed.

If this function is omitted, the loop will not be able to respond to external input to stop, which means the application will continue running indefinitely. Including isStopDetected(ALGOC); ensures smooth and controllable execution, making it a critical addition to any loop that may need to be interrupted by user input or a serial monitor command. This approach provides a safe way to manage the application's flow and prevents it from becoming unresponsive.

Here's an example where the isStopDetected(ALGOC); function is correctly placed to allow stopping the application at any point:

```
C/C++
#include <algoC.h>

void application(ALGOC_APP)
{
while(1) // Infinite loop
{
isStopDetected(ALGOC); // Check for stop signal in the main loop
for(int k = 0; k = 0; k--)
{
isStopDetected(ALGOC); // Check for stop signal within the inner loop
light(ALGOC, 1, 0.1, k, "Red", true); // Gradually decrease light
intensity
}
for(int k = 9; k >= 0; k--)
{
isStopDetected(ALGOC); // Check for stop signal within the inner loop
light(ALGOC, 1, 0.1, k, "Red", true); // Gradually decrease light
intensity
}
}
}
```

Creating your own function

C and C++ have functions to enable the programmer to group the code so that one repetitive task can be used over and over again without a need to rewrite the code. To use the capabilities of these functions we just need to call the appropriate custom function.

The syntax for our custom functions is the same as for any other C and C++ functions, with the only difference being that the first argument is always an internally defined macro.

But, let's learn about this from an example. Imagine the situation in which you want to create the function to calculate the sum of 3 numbers. This function has to get these values as arguments, do the appropriate calculation with received arguments, and return the calculated value.

Apart from the 3 arguments specified above, the first parameter for every custom function must be an internally defined macro ALGOC_APP. Think of a C macro as a shortcut in your code. When you define a macro, you're telling the compiler, "Whenever you see this name, replace it with this block of code before you start compiling." This happens automatically, saving you from having to type out the same code multiple times.

You can see the example below on how the function is declared, defined, and called.

```
C/C++
#include <algoC.h>

int customSum(ALGOC_APP,int data1, int data2, int data3);
void application(ALGOC_APP)
{
int sumOfNumbers = customSum(ALGOC_FUNCTION,1,2,3);
move(ALGOC,'A',sumOfNumbers,10,CW,true);
}
int customSum(ALGOC_APP,int data1, int data2, int data3)
{
return data1 + data2 + data3;
}
```

As you can see, when we are calling the function we specify 4 arguments (*Note: The number of arguments must be the same as the number of parameters in the function declaration).*

These are the arguments we are using in the above example:

- *ALGOC_FUNCTION – internally defined macro. This argument will always be the first argument of the function call. If this is not the case in some cases your program won't compile or if it compiles it won't run as you have envisioned*

- *data1 – First value we send to function*
- *data2 – Second value we send to function.*
- *data3 – Third value we send to function.*

As you can see, for our custom function we can have as many arguments as needed by the custom application. What is important is that the first argument is always internal macro as explained above.

Note: even if your custom functions don't need any arguments you need to specify the internal macro. So, the custom function will always have at least one argument.

Control over Serial (using Serial Monitor)

Sometimes there is a need to manually control some parts of the system. In this case, we can use a serial monitor to issue different commands to the device. To do this you need to open the Serial Monitor by going to *Tools->Serial Monitor.*

Before issuing commands you need to configure the Serial monitor as shown on the previous image. Following should be configured:

- New Line
- 115200 baud

We will present capabilities of control over serial below.

How to start and stop code execution from Serial Monitor

We can start and stop the code execution through Serial Monitor. For this purpose we use

- 'S' – to stop code execution
- 'P' – to start code execution

How to control the motor from the Serial Monitor

We have in total 3 motors that we can control. Each motor port is marked with letters 'A', 'B', and 'C'. This letter is the first part of the command telling the system which motor we

want to control. After the letter we specify the number of rotations we want to make with a positive number rotating the motor in a clockwise direction and a negative number rotating the motor counter clockwise.

Example: Move motor A for half rotation clockwise

```None
A0.5
```

Example: Move motor C for 2 rotations counterclockwise

```None
C-2.0
```

It is important to note that the motor will by default drive with power 10. If there is a need to change the default value of the motor power we can use the ‘M’ command followed by the power level.

Example: Change the default power level for all motors to 5

```None
M5
```

How to read sensor value through Serial Monitor

We have in total 2 sensor input ports from which we can read and print the sensor value. Each input port is marked with numbers 1 and 2. To read the sensor value we provide the char ‘S’ followed by the port number.

Example: Read and print the sensor connected on the port 2

```None
S2
```