mBot codes and Instructions

The mBot by Makeblock is an economical robot kit that allows basic programming of electronics and robotics. It uses a microcontroller board to run its code.

Each robot requires a corresponding Makeblock library to be downloaded in order to execute the code. The libraries can be found at https://github.com/Makeblock-official/Makeblock-Libraries.

After downloading the appropriate library, the robot can be operated with code on the microcontroller. For this document, a mCore has been used together with the Arduino UNO for running the code.

1. The Basic Sketch

A header file is required to use the standard functions of the Makeblock robot. Each robot has its own unique library, and so a different header file is needed.

Orion	<MeOrion.h>
BaseBoard	<MeBaseBoard.h>
mCore	<MeMCore.h>
Shield	<MeShield.h>
Auriga	<MeAuriga.h>
MegaPi	<MeMegaPi.h>

There are five components to a basic sketch:

a. The Makeblock Library

```
#include <MeMCore.h>
```

- The library imports standard files, classes, and functions for the robot to use.

b. Declare Constants and Variables

```
#define BUZZER_PORT 45
int distance;
```

- Constants and Variables allow the permanent and temporary storage of data.

c. Instantiate Objects for each Sensor

```
MeUltrasonicSensor ultra (PORT_6);
MeBuzzer buzzer;
```

- Classes are used to create Objects so that the functions in the class may be used.

d. The Setup Code

```
void setup()
{
  //setup code here
}
```

- Code that runs once. Can be used to assign ports and configure settings.

e. The Loop Code

```
void loop()
{
  //loop code here
  delay(100);
}
```

- Code that keeps running. Responsible for most continuous actions of the robot.
- delay() is a common function that causes the robot to perform no other actions for the parameter provided, given in milliseconds. This allows some settings, such as motors, to execute their functions for a while before changing.

2. Components of the mCore

The mCore comes with a variety of hardware, configured to ports on the mCore.

Buzzer	x1
Light sensor	x1
Button	x1
WS2812 LEDs	x2
PWM motor controllers	x2
IR receiver	x1
IR emitter	x1
Blue LED	x1
Bluetooth module	x1
RJ25 connectors	x4

3. Port Information

These mCore sensors have been defined to specific numbers in the Makeblock library.

Port_1	{11,12}	RJ25 port 1
Port_2	{9,10}	RJ25 port 2
Port_3	{A2,A3}	RJ25 port 3
Port_4	{A0,A1}	RJ25 port 4
Port_5	{NC,NC}	(Not Connected)
Port_6	{8,A6}	Buzzer, Light sensor
Port_7	{A7,13}	Button, WS2812 LED x2
Port_8	{8,A6}	Buzzer, Light sensor
Port_9	{6,7}	PWM motor 2, DIR Motor 2
Port_10	{5,4}	PWM motor 1, DIR Motor 1

Some sensors do not have a port number to define them. They can be accessed by specific Arduino port numbers.

2	IR Receiver
3	IR emitter LED
5	light sensor
13	Blue LED

4. Hardware Features

The hardware components of the mCore can be accessed by instantiating specific classes and calling their functions.

4.1 Ultrasonic Sensor

The ultrasonic sensor emits ultrasound directly in front of the robot. If an object is present in front of the robot, the ultrasound it will bounce back to the module. This transmission time of the ultrasonic wave is used to calculate the object's distance.
The ultrasonic sensor uses the MeUltrasonicSensor class.

```
#include "MeMCore.h"

// create an object named ultrasensor
MeUltrasonicSensor ultraSensor(PORT_1);

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("Distance : ");
  Serial.print(ultraSensor.distanceInch() );    //or distanceCm()
  Serial.printInch(" inch");
  delay(100);
}
```

4.2 Buzzer

The buzzer emits a sound at a declared frequency in hertz, for a declared time in milliseconds.
The buzzer uses the MeBuzzer class.

```
#include <MeMCore.h>
MeBuzzer buzzer;

void setup(){
  buzzer.tone(600, 1000);  //Buzzer sounds at 600hz for 1000ms
  delay(2000);
  buzzer.tone(1200, 1000);  //Buzzer sounds at 1200hz for 1000ms
  delay(2000);
}
```

4.3 LED

The LEDs can be set to specific colors by standard RGB values. The left LED is set with the number 0, while the right LED is set with the number 1.

It is important to note that the mCore LEDs, and so the MeRGB functions, must be accessed by the Arduino port number 13.
The LED uses the MeRGBLed class.

```
#include "MeMCore.h"

#define red        255,000,000
#define blue       000,000,255

MeRGBLed led(PORT_7, 2);

void setup()
{
}

void loop()
{
  led.setColorAt (0, blue);
  led.setColorAt (1, red);
  led.show();
  delay (100);
  led.setColorAt (0, red);
  led.setColorAt (1, blue);
  led.show();
  delay (100);
}
```

4.4 DC Motor Activation and Movement

The motors of the mCore are used to rotate the wheels of the mBot, allowing movement and the ability to rotate in place.
Note that the motors turn clockwise.
For the left motor, negative values move the mBot forward.
For the right motor, positive values move the mBot forward.
The motors use the MeDCMotor class.

```
#include <MeCore.h>

MeDCMotor motor1(M1);
MeDCMotor motor2(M2);

void setup(){}

void loop()
{
  //motor.run() maximum speed is 255 to -255, 0 is stop
  motor1.run(-100);  //Motor1 (Left) forward is -negative
  motor2.run(100);  //Motor2 (Right) forward is +positive
  delay(500);
  //motor.run() maximum speed is 255 to -255, 0 is stop
  motor1.run(100);  //Motor1 (Left) forward is -negative
  motor2.run(-100);  //Motor2 (Right) forward is +positive
  delay(500);

  motor1.stop();
  motor2.stop();
  delay(500);
}
```

Note that low values of speed may not produce enough torque to move the mbot.

Turning is dependent on both speed and turning time. The movement and turning could be made more precise by a gyro module for precise turns, or a compass moddule for geographical orientation.

The code below approximates a left-turning algorithm.

```
int speed = 100;
int turnTime = 2750;
int maxDistance = 18;
int state = 0;
while (true) {
  if (ultrasonic.distanceCm() > maxDistance){
  motor1.run(-100); //Motor1 (Left)  forward is -negative
  motor2.run(100);  //Motor2 (Right) forward is +positive
  }
  if (state==0 && ultrasonic.distanceCm() < maxDistance){
  motor1.run(100); //Motor1 (Left)  forward is -negative
  motor2.run(100);  //Motor2 (Right) forward is +positive
  delay(turnTime/4);
  if (ultrasonic.distanceCm() < maxDistance){
  state++;
  }
  else state=0;
  }
  if (state==1 && ultrasonic.distanceCm() < maxDistance){
  motor1.run(-100); //Motor1 (Left)  forward is -negative
  motor2.run(-100);  //Motor2 (Right) forward is +positive
  delay(turnTime/2);
  if (ultrasonic.distanceCm() < maxDistance){
  state++;
  }
  else state=0;
  }
  if (state==2 && ultrasonic.distanceCm() < maxDistance){
  motor1.run(-100); //Motor1 (Left)  forward is -negative
  motor2.run(-100);  //Motor2 (Right) forward is +positive
  delay(turnTime/4);
  if (ultrasonic.distanceCm() < maxDistance){
  state++;
  }
  else state=0;
  }
  if (state==3 && ultrasonic.distanceCm() < maxDistance){
  motor1.run(-100); //Motor1 (Left)  forward is -negative
  motor2.run(-100);  //Motor2 (Right) forward is +positive
  delay(turnTime/8);
  if (ultrasonic.distanceCm() < maxDistance){
  state=0;
  }
}

void stopmovement(){
  motor1.run(0); //Motor1 (Left)  forward is -negative
  motor2.run(0);  //Motor2 (Right) forward is +positive
}
```

4.5 Infrared Floor Line Detection Sensor

The mCore comes with two infrared sensors. They are used to detect if the color on the floor is white(light) or black(dark).
S1 is the left sensor.
S2 is the right sensor.
The two infrared sensors use a single MeLineFollower class.

```
#include <MeMCore.h>
MeLineFollower lineFinder(PORT_2);

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int sensorState = lineFinder.readSensors();
  switch(sensorState)
  {
    case S1_IN_S2_IN:  Serial.println("Sensor 1 and 2 are inside of black
line"); break;
    case S1_IN_S2_OUT:  Serial.println("Sensor 2 is outside of black
line"); break;
    case S1_OUT_S2_IN:  Serial.println("Sensor 1 is outside of black
line"); break;
    case S1_OUT_S2_IN:  Serial.println("Sensor 1 and 2 are outside of
black line "); break;
    default:  break;
  }
  delay(200);
}
```

These can be used to keep the mBot on a pre-drawn black line path on on the floor.

The mBot comes with a black line track on white paper.
Below is a basic line-following algorithm.

```
#include <MeMCore.h>
MeBuzzer buzzer;
MeLineFollower lineFinder(PORT_2);
MeDCMotor motor1(M1); //Motor1 is Left Motor
MeDCMotor motor2(M2); //Motor2 is Left Motor
MeRGBLed led(0, 30);

void setup() {
  led.setpin(13);
  pinMode(7, INPUT); //Define button pin as input
  while (analogRead(7) > 100) {
    delay(50); //Wait till button pressed to start.
  }
  buzzer.tone(200, 200); //Buzzer beep to indicate start
}

float MOTOR1_TUNE= -1.0; //Left motor scale factor
float MOTOR2_TUNE= 1.0; //Right motor scale factor
float turning_left= true;

void loop() {
  int sensorState = lineFinder.readSensors();
  switch (sensorState)
  {
    case S1_IN_S2_IN:
      motor1.run(MOTOR1_TUNE* 255.0); //Left
      motor Run motor2.run(MOTOR2_TUNE* 255.0); //Right motor Run
      led.setColorAt(1, 0, 255, 0); //Set LED1 (RGBLED2) (LeftSide)
      led.setColorAt(0, 0, 255, 0); //Set LED0 (RGBLED1) (RightSide)
      led.show();
      break;
    case S1_IN_S2_OUT:
      //turn left
      motor1.run(MOTOR1_TUNE* 0); //Left motor Stop
      motor2.run(MOTOR2_TUNE* 255.0); //Right motor Run
      led.setColorAt(1, 0, 0, 0); //Set LED1 (RGBLED2) (LeftSide)
      led.setColorAt(0, 0, 255, 0); //Set LED0 (RGBLED1) (RightSide)
      led.show();
      turning_left= true;
      break;
    case S1_OUT_S2_IN:
      //turn right
      motor1.run(MOTOR1_TUNE* 255.0); //Left motor Run
      motor2.run(MOTOR2_TUNE* 0); //Right motor Stop
      led.setColorAt(1, 0, 255, 0); //Set LED1 (RGBLED2) (LeftSide)
      led.setColorAt(0, 0, 0, 0); //Set LED0 (RGBLED1) (RightSide)
      led.show();
      turning_left= false;
      break;
    case S1_OUT_S2_OUT:
      //keep turning what it was turning
      if (turning_left) {
      motor1.run(MOTOR1_TUNE* 0); //Left motor Stop
      motor2.run(MOTOR2_TUNE* 255.0); //Right motor Run
      led.setColorAt(1, 0, 0, 0); //Set LED1 (RGBLED2) (LeftSide)
      led.setColorAt(0, 255, 0, 0); //Set LED0 (RGBLED1) (RightSide)
      led.show();
    } else {
      motor1.run(MOTOR1_TUNE* 255.0); //Left motor Run
      motor2.run(MOTOR2_TUNE* 0); //Right motor Stop
      led.setColorAt(1, 255, 0, 0); //Set LED1 (RGBLED2) (LeftSide)
      led.setColorAt(0, 0, 0, 0); //Set LED0 (RGBLED1) (RightSide)
      led.show();
    }
    break;
    default: break;
    }
}
```

4.5 IR Remote Infrared Sensor

The mCore comes with an IR Sensor, and an IR remote with 21 buttons. Each of these buttons corresponds to a specific unsigned int32 value, which can be used as an input to call a switch case. For test purposes, each case has simply been set to print the name of the button.
The infrared sensor uses the MeIR class.

```
#include <MeMCore.h>
MeIR ir;
void setup()
{
  ir.begin();
  Serial.begin(9600);
  Serial.println("Infrared Receiver Decoder"  );
}

void loop()
{
  if(ir.decode())
  {
    uint32_t value = ir.value;
    Serial.print("Raw Value:  ");
    Serial.println(value);
    value = value >> 16 & 0xff;
    Serial.print("Button Code: ");
    Serial.println(value);
    Serial.print("Button: ");
    switch(value)
      {
            case IR_BUTTON_A: Serial.println("A");break;
            case IR_BUTTON_B: Serial.println("B");break;
            case IR_BUTTON_C: Serial.println("C");break;
            case IR_BUTTON_D: Serial.println("D");break;
            case IR_BUTTON_E: Serial.println("E");break;
            case IR_BUTTON_F: Serial.println("F");break;
            case IR_BUTTON_SETTING: Serial.println("SETTING");break;
            case IR_BUTTON_LEFT: Serial.println("LEFT");break;
            case IR_BUTTON_RIGHT: Serial.println("RIGHT");break;
            case IR_BUTTON_UP: Serial.println("UP");break;
            case IR_BUTTON_DOWN: Serial.println("DOWN");break;
            case IR_BUTTON_0: Serial.println("0");break;
            case IR_BUTTON_1: Serial.println("1");break;
            case IR_BUTTON_2: Serial.println("2");break;
            case IR_BUTTON_3: Serial.println("3");break;
            case IR_BUTTON_4: Serial.println("4");break;
            case IR_BUTTON_5: Serial.println("5");break;
            case IR_BUTTON_6: Serial.println("6");break;
            case IR_BUTTON_7: Serial.println("7");break;
            case IR_BUTTON_8: Serial.println("8");break;
            case IR_BUTTON_9: Serial.println("9");break;
            default:break;
      }
  }
}
```

Functions can be assigned to each IR remote button in this manner.

4.6 Testing the GSM Library

The Global System for Mobile Communications (GSM) is a standard of protocols for digital cellular networks. A GSM shield peripheral can provide an interface for the mBot with which to perform serial communications and access networks from. The code below will print out the name of the network carrier, and the signal strength.
Note that a SIM card is required for use of the GSM shield.
The GSM shield requires its own library.

```
#include <GSM.h>

// PIN Number
```

```
#define PINNUMBER ""

// initialize the library instance
GSM gsmAccess(true);     // include a 'true' parameter for debug enabled
GSMScanner scannerNetworks;
GSMModem modemTest;

// Save data variables
String IMEI = "";

// serial monitor result messages
String errortext = "ERROR";

void setup()
{
  // initialize serial communications
  Serial.begin(9600);
  Serial.println("GSM networks scanner");
  scannerNetworks.begin();

  // connection state
  boolean notConnected = true;

  // Start GSM shield
  // If your SIM has PIN, pass it as a parameter of begin() in quotes
  while(notConnected)
  {
    if(gsmAccess.begin(PINNUMBER)==GSM_READY)
      notConnected = false;
    else
    {
      Serial.println("Not connected");
      delay(1000);
    }
  }

  // get modem parameters
  // IMEI, modem unique identifier
  Serial.print("Modem IMEI: ");
  IMEI = modemTest.getIMEI();
  IMEI.replace("\n","");
  if(IMEI != NULL)
    Serial.println(IMEI);

  // currently connected carrier
  Serial.print("Current carrier: ");
  Serial.println(scannerNetworks.getCurrentCarrier());

  // returns strength and ber
  // signal strength in 0-31 scale. 31 means power > 51dBm
  // BER is the Bit Error Rate. 0-7 scale. 99=not detectable
  Serial.print("Signal Strength: ");
  Serial.print(scannerNetworks.getSignalStrength());
  Serial.println(" [0-31]");
}

void loop()
{
  // scan for existing networks, displays a list of networks
  Serial.println("Scanning available networks. May take some seconds.");

  Serial.println(scannerNetworks.readNetworks());

    // currently connected carrier
  Serial.print("Current carrier: ");
  Serial.println(scannerNetworks.getCurrentCarrier());

  // returns strength and ber
  // signal strength in 0-31 scale. 31 means power > 51dBm
  // BER is the Bit Error Rate. 0-7 scale. 99=not detectable
  Serial.print("Signal Strength: ");
  Serial.print(scannerNetworks.getSignalStrength());
  Serial.println(" [0-31]");

}
```

4.7 Sending an SMS Message

With the help of a registered SIM card, the GSM shield can be used to sent text messages.

```
#include <GSM.h>

#define PINNUMBER ""

// initialize the library instance
GSM gsmAccess; // include a 'true' parameter for debug enabled
GSM_SMS sms;

// char array of the telephone number to send SMS
char remoteNumber[20]= "0000000000";      //dummy number

// char array of the message
char txtMsg[200]="Test";

void setup()
{
  // initialize serial communications
  Serial.begin(9600);
  Serial.println("SMS Messages Sender");

  // connection state
  boolean notConnected = true;

  // Start GSM shield
  // If your SIM has PIN, pass it as a parameter of begin() in quotes
  while(notConnected)
  {
    if(gsmAccess.begin(PINNUMBER)==GSM_READY)
      notConnected = false;
    else
    {
      Serial.println("Not connected");
      delay(1000);
    }
  }
  Serial.println("GSM initialized");
  sendSMS();
}

void loop()
{
// nothing to see here
}

void sendSMS(){

  Serial.print("Message to mobile number: ");
  Serial.println(remoteNumber);

  // sms text
  Serial.println("SENDING");
  Serial.println();
  Serial.println("Message:");
  Serial.println(txtMsg);

  // send the message
  sms.beginSMS(remoteNumber);
  sms.print(txtMsg);
  sms.endSMS();
  Serial.println("\nCOMPLETE!\n");
}
```