Programming Mbots using Arduino

Bas Vegt
Cognitive Robotics
Leiden University

2021-03-26

The Mbots' functionality is driven by their Arduino boards, which in our case are programmed using a language based on C++. You will be using the Arduino IDE to write and upload the code to your Mbot's board. The following contains a setup guide for programming your Mbot this way, as well as a standard list of the functions available using this setup.

Setup

To get started with writing code for your Mbot, follow the following steps:

1. If you have not, **download and install the Arduino IDE** from arduino.cc/en/software.
2. **Download the Makeblock libraries** .ZIP file from their github: github.com/Makeblock-official/Makeblock-Libraries *(via the button labelled '**Code**').*
3. Run the Arduino exe, then load the new libraries using **Sketch** ⇒ **Include Library** ⇒ **Add .ZIP Library...**.

You can now start writing your code. To upload the code unto your Mbot, do the following:

4. Connect your Mbot to a USB port and switch it on.
5. Under **Tools**, Ensure your **Board** is set to '***Arduino Uno***'.
6. Again under **Tools**, select the appropriate **Port**- whichever appears when you plug in your Mbot. (In case of windows it should be **COM1-4**; on Linux and Mac it'll be something else.)
7. Click the encircled arrow in the upper left corner and wait for the 'done uploading' message. The first time you do, you'll be prompted to save the code to file as well.

Before creating anything overly complex, it's recommended to get familiar with the capabilities of your bot. This is what today's exercises are for. Below are some tips for writing your code.

Writing the Code

Writing code in C is similar to what you're used to in Javascript for the virtual bots. An Arduino code file is called a 'sketch'- this is where you declare and call functions so that they can be performed during runtime.

Upon startup, the Arduino board will always run the setup() function first, the moment it is powered on. Afterwards, it will continuously run the loop() function until it is powered off. All other functions must be called from either of those two core functions, and only after it has been defined in an earlier line of code (or imported from a library). For a more exhaustive guide on understanding and building functions, see arduino.cc/en/Reference/FunctionDeclaration.

You can write comments in your code using '/*' and '*/' to bracket them, or simply succeeding '//' for short comments. This is all shown in a sample given below.

Example Code

Here's something basic to get you started: this code should make your bot drive forward for a bit after it is turned on, then stop. **(Be sure not to let the bot drive of your desk when you upload this!)**

```
// Access the required libraries:
#include <MeMCore.h>
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>

// Declaring the motor fields
MeDCMotor rightMotor(9); // Find one motor in slot 9 and call it rightMotor
MeDCMotor leftMotor(10); // Find another motor in slot 10 and call it leftMotor

void setup() {
   delay(1000);
   leftMotor.run(100);
   rightMotor.run(-100);
   delay(1000);
   leftMotor.run(0);
   rightMotor.run(0);
}


void loop() {
   //this function left empty for this example
}
```

Go ahead and test this, then expand on it or change anything you like. Experiment! You can always copy-paste this template again and start over.

Existing Arduino Functionality

You do not have to reinvent the wheel for every new task you want your bot to perform. There are many tools available Arduino's reference pages (at arduino.cc/refrence/en) provide a full account of the functions, variables, and structure used in Arduino. Consult these pages to learn about the exisitng core functionality of the Arduino board or any specific element.

Of course, you can use any other online sources for inspiration as well. This is particularly valuable in when you're trying to do something new to you. Anytime you try to do anything in code, odds are good someone else has done something similar in the past and posted an example or question about it somewhere. You can take advantage of this prior knowledge, as long as you don't carry whole pieces of someone else's code over and present it as your own. It's also good practise to try to understand the basic functionality of any piece of borrowed code you use, even if you couldn't replicate it perfectly yet.

The Mbots' sensors, motors, and output devices are all accessed using specific functions found in the MakeBlock libraries you installed. There is no exhaustive and up to date documentation of these libraries available at the moment, so we have listed the main functions you might need below.

Mbot Parts and how to Control them

All your mbots are equipped with several input and output devices, listed below. To access any of these components of your bot in your sketch, you'll have to declare an object of the appropriate type. For some components, you'll also need to indicate a number corresponding to the port that component is plugged in to, in order to assign it. We have listed all these components below, as well as their data type, slot number, and a brief description of the functions associated with each component. Try using the output functions in combination with the input values to make your bot respond to the environment!

Motors

Object Type: MeDCMotor **Slot:** 9 & 10
Your Mbot's motors are controlled using the .run method on a MeDCMotor object, passing an int argument between for the desired rotation power. Each of the two motors is a seperate component and their speed can thus be set independently, to achieve rotation, etc. Any integer can be passed, but values lower than -255 or higher than 255 will be set to -255 and 255, respectively. Each motor is also identifiable by being their M1 or M2, which can be evaluated using comparison operators.
Example Code:

```
MeDCMotor motor9(9);
MeDCMotor motor10(10);

void applyPower(int power){
   // Scale power so that 100% = 255, which is the max output
   power *= 2.55;
   int leftSpeed = power;
   int rightSpeed = -power; // reverse one of the rotations
   //evaluate which motor is M1
   if((9) == M1){
      motor10.run(leftSpeed);
      motor9.run(rightSpeed);
   }
   else if((10) == M1){
      motor9.run(leftSpeed);
      motor10.run(rightSpeed);
   }
}
```

LED Lights

Object Type: MeRGBLed **Slot:** 7
Our Mbots have two LED lights packed on their boards, one left and one right, which can each be lit up in any color using RGB values.
- **.fillPixelsBak(0, 2, 1)** is used for calibration purposes. Use this once, in your setup().
- **.setColor(int i, int r, int g, int b)** sets the color and brightness of the desired LEDs

(indicated bij i, ranged 0-2) to the RGB value indicated by r, g and b, each ranged 0-255.
- **.show()** applies changes made by .setColor calls.

Ultrasonic Sensor

Object Type: MeUltrasonicSensor **Slot:** 1-4 depending on configuration
On the front of your bot, there is a pair of ultrasonic sensors which measures the distance to the closest object detected, in cm. Using the .distanceCm() method returns this value.

Line Follower

Object Type: MeLineFollower **Slot:** 1-4 depending on configuration
On the bottom, in front of the middle wheel, is a pair of IR sensors which each classify the surface directly under them as either black, or white. The .readSensors method returns an integer indicating which sensor reports which value, as follows:

value	left sensor	right sensor
0	black	black
1	black	white
2	white	black
3	white	white

LED Matrix

Object Type: MeLEDMatrix **Slot:** 1-4 depending on configuration
Not all bots come equipped with this component.
This small screen can be used for simple display of numbers, text, etc. This is not necessary for the assignments at all, and so I myself have not taken the time to get acquinted with every method and their parameters, but for those who want to play around with this feature, here are all the methods I could find:

```
MeLEDMatrix ledMtx(1);
unsigned char drawBuffer[16];
unsigned char *drawTemp;

void showStuff(){
   ledMtx.setColorIndex(1);
   ledMtx.setBrightness(6);
   ledMtx.clearScreen();
   delay(1000);
   ledMtx.showNum(0, 3);
   delay(1000);
   ledMtx.showClock(12, 0, true);
   delay(1000);
   ledMtx.drawStr(0, 0 + 7, "hello");
   delay(1000);
   drawTemp = new unsigned char[16]{0,0,60,126,126,60,0,0,0,0,60,126,126,60,0,0};
   memcpy(drawBuffer, drawTemp, 16);
   free(drawTemp);
   ledMtx.drawBitmap(0, 0, 16, drawBuffer);
}
```

Buzzer

Object Type: MeBuzzer **Slot:** N/A
You do not need an argument (or brackets) in your object declaration for the buzzer.
Using .tone(x, y), the buzzer can play a tone of x Hz for y milliseconds.

Button

Object Type: N/A **Slot:** N/A
You do not need to declare or assign any object before using the button.
There is a small, black pin button on the top of your Mbot. Pushing this button sends a non-continuous signal to the board. Use **analogRead(A7)<10** to return TRUE when the button is pressed.

Acknowledgements

This document was written for the Cognitive Robotics course at Leiden University. Functions and datatypes listed are derived from the MakeBlocks IDE.