Controlling Create® 2 with Arduino and Android

iRobot™ Education

In this project, we'll use an Arduino and an Android APP to move the robot around. This tutorial provides a simple platform for exploring how to connect the Arduino with the Create 2 robot via serial port and how to command its motors, LEDS and sound.

This project was originally developed by Marcelo Rovai with edits made by the iRobot Education Team.

Previous iterations can be sourced from Instructables and Arduino. This tutorial can also be accessed in Portuguese here.

Quick Overview

The Create® 2 platform is based upon the Roomba® robot vacuum cleaner and is therefore referred to in the code and this tutorial as "Roomba." The robot is a differential drive robot, with 2 wheels and a front caster. Its velocity has a maximum of 500 mm/s and can be commanded to be increased or decreased.

For signalization, with the Create offers four 7-segment displays and 5 LEDs (see figure):

- Clean
- Spot
- Dock
- Warning/Check
- Dirt/Debris

As internal sensors, we have, among others:

- Cliff detector (4 in front)
- Bump detectors (2 in front)
- Wheel encoders

STEM Skills:

- Electrical Engineering
- Mechanical Engineering
- Arduino language (C/C++)

Experience Level:

Advanced

Supplies:

- iRobot Create 2
- Arduino UNO
- Bluetooth module HC-06
- Push-button tactile switch
- Blue Tooth Remote Control App
- Code Repo on GitHub

Additional Resources:

- Getting Started with Create 2

For more details about programming the modes of Create 2, use the Specification Doc.
The robot can be programmed in 3 modes:

Passive Mode:

Upon sending the Start command or any one of the cleaning mode commands (e.g., Spot, Clean, Seek Dock), the OI enters into Passive mode. When the OI is in Passive mode, you can request and receive sensor data using any of the sensor commands, but you cannot change the current command parameters for the actuators (motors, speaker, lights, low side drivers, digital outputs) to something else.

Safe Mode:

Gives you full control of the robot, with the exception of the following safety-related conditions:

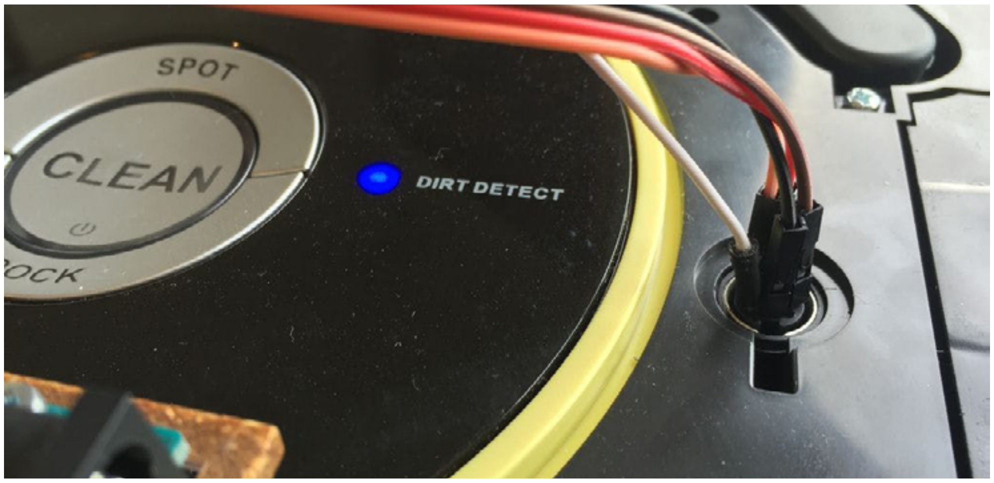
- Charger plugged in and powered.
- Detection of a wheel drop (on any wheel).
- Detection of a cliff while moving forward (or moving backward with a small turning radius, less than one robot radius).

Should one of the above safety-related conditions occur while the OI is in Safe mode, Roomba stops all motors and reverts to the Passive mode.

Full mode:

Gives you complete control over the robot, all of its actuators, and all of the safety-related conditions that are restricted when the OI is in Safe mode. Full mode shuts off the cliff, wheel-drop and internal charger safety features.

Step 1: The Serial Connection

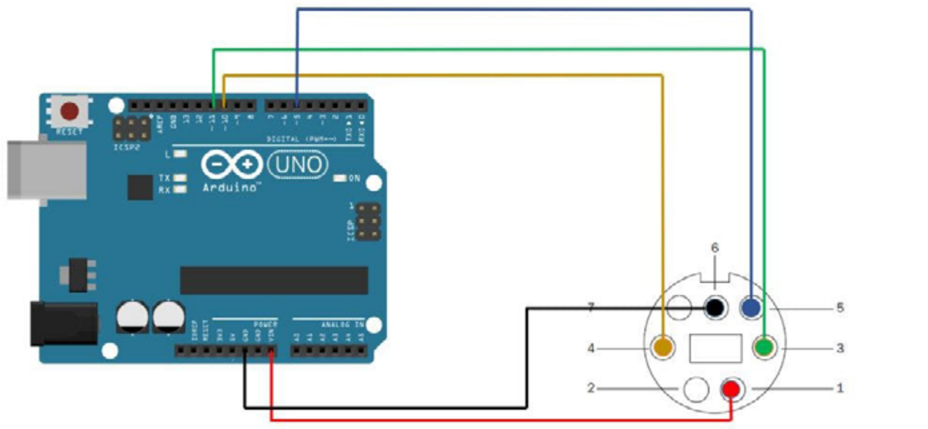


For communication between the Roomba and Arduino, the Serial Port will be used. By default, Roomba communicates at 115,200 baud, but in order to communicate with Arduino, we will switch it to 19,200.

There are 2 ways to set the robots baud rate:

1) While powering off the robot, continue to hold down the Clean/Power button after the light has turned off. After about 10 seconds, Roomba plays a tune of descending pitches. Roomba will communicate at 19,200 baud until the processor loses battery power or the baud rate is explicitly changed by way of the OI.

2) Use the Baud Rate Change pin (pin 5 on the Mini-DIN connector) to change Roomba's baud rate. After turning on Roomba, wait 2 seconds and then pulse the Baud Rate Change low three times. Each pulse should last between 50 and 500 milliseconds. Roomba will communicate at 19200 baud until the processor loses battery power or the baud rate is explicitly changed by way of the OI.



Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	DD	Device Detect input (active low) – used to wake up Roomba from sleep
6	GND	Roomba battery ground
7	GND	Roomba battery ground

This diagram shows how the Arduino should be connected to Create 2 Mini-DIN connector

Step 2: Starting with Create 2

To see examples of the code in the following steps, you can access the repository here.
The first thing that must be done when programming a Create 2 is to:

- "Wake up" the robot
- Define the mode (Safe or full)

We can "wake-up" it, sending a Low pulse to Mini-DIN pin 5 (detect device input) as shown on function below:

```
void wakeUp (void)
{
  setWarningLED(ON);
  digitalWrite(ddPin, HIGH);
  delay(100);
  digitalWrite(ddPin, LOW);
  delay(500);
  digitalWrite(ddPin, HIGH);
  delay(2000);
}
```

To start-up the robot, 2 codes must always be sent: "START" [128] and the mode, in our case "SAFE MODE" [131]. If you want a "FULL MODE", the code [132] should be sent instead.

```
void startSafe()
{
  Roomba.write(128);  //Start
  Roomba.write(131);  //Safe mode
  delay(1000);
}
```

Step 3: Controlling the Display and LEDs

As described in the Quick Overview section, the Create 2 has 5 LEDs:

- Power/Clean (bi-color red/green and intensity controlled)
- Spot (Green, fixed intensity)
- Dock (Green, fixed intensity)
- Warning/Check (Orange, fixed intensity)
- Dirt/Debris (Blue, fixed intensity)

All LEDs can be commanded using code [139]. To control the Power LED, you must send two data bytes to the robot: "**color**" and "**intensity**".

Color:

- Green = 0
- Orange = 128
- Red=255

Intensity:

- Low=0
- Max=255

The function ***setPowerLED (byte setColor, byte setIntensity)*** does it:

```
void setDigitLEDs(byte digit1, byte digit2, byte digit3, byte digit4)
{
    Roomba.write(163);
    Roomba.write(digit1);
    Roomba.write(digit2);
    Roomba.write(digit3);
    Roomba.write(digit4);
}
```

For example, to turn on the POWER LED with orange color at half its intensity, you can call the function as below:

```
setPowerLED (128, 128);
```

To turn ON the remaining 4 LEDs, the bellow functions must be used:

```
setDebrisLED(ON);
setDockLED(ON);
setSpotLED(ON);
setWarningLED(ON);
```

All the above functions have a code similar to this one:

```
void setDebrisLED(bool enable)
{
  debrisLED = enable;
  Roomba.write(139);
  Roomba.write((debrisLED ? 1 : 0) + (spotLED ? 2 : 0) + (dockLED ? 4
: 0) + (warningLED ? 8 : 0));
  Roomba.write((byte)color);
  Roomba.write((byte)intensity);
}
```

Basically, the difference is with the line:

```
debrisLED = enable;
```

That must be changed enabling each one of other LEDs (spotLED, dockLED, warningLED).

Sending Messages to Display

The robot has four 7-Segment Displays that you can use to send messages in two different ways:

- Code [163]: Digit LEDs Raw (Numeric)
- Code [164]: Digit LEDs ASCII (Approximation of letters and especial codes)

To display numbers is simple: you must send the code [163], following the 4 digits to be displayed. The function: ***setDigitLEDs(byte digit1, byte digit2, byte digit3, byte digit4)*** does this for you:

```
void setDigitLEDs(byte digit1, byte digit2, byte digit3, byte digit4)
{
    Roomba.write(163);
    Roomba.write(digit1);
    Roomba.write(digit2);
    Roomba.write(digit3);
    Roomba.write(digit4);
}
```

For example, to display “1, 2, 3, 4”, you must call the function:

```
setDigitLEDs(1, 2, 3, 4);
```

With code [164], it's possible to send approximation of ASCII. The function ***setDigitLEDFromASCII(byte digit, char letter)*** does this:

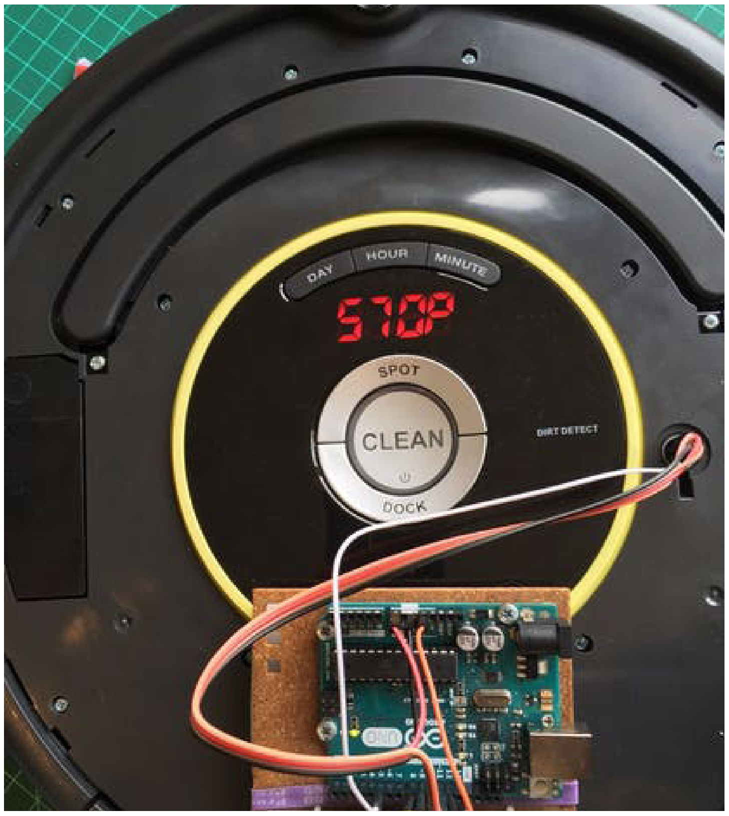
```
void setDigitLEDFromASCII(byte digit, char letter)
{
  switch (digit){
  case 1:
    digit1 = letter;
    break;
  case 2:
    digit2 = letter;
    break;
  case 3:
    digit3 = letter;
    break;
  case 4:
    digit4 = letter;
    break;
  }
  Roomba.write(164);
  Roomba.write(digit1);
  Roomba.write(digit2);
  Roomba.write(digit3);
  Roomba.write(digit4);
}
```

To simplify, I create a new function to send the 4 digits at same time:

```
void writeLEDs (char a, char b, char c, char d)
{
  setDigitLEDFromASCII(1, a);
  setDigitLEDFromASCII(2, b);
  setDigitLEDFromASCII(3, c);
  setDigitLEDFromASCII(4, d);
}
```

For example, to display “STOP”, you must call the function:

```
writeLEDs (‘s’, ‘t’, ‘o’, ‘p’);
```



Step 4: Moving the Robot Around

For mobility, the Create 2 has two independent DC motors that can be programed to run up to 500mm/s. There are several commands that can be used to drive the robot. The main ones are:

- Code [137]: Drive ==> Must send +/- Speed in mm/s and +/- Radius in mm
- Code [145]: Drive Direct ==> Must send Left/Right Speed in mm/s (+ for Forward and - for Backward)
- Code [146]: Drive PWM ==> Must send +/- PWM data for Left and Right wheels

Below is the code for those 3 options:

```
void drive(int velocity, int radius)
{
  clamp(velocity, -500, 500); //def max and min velocity in mm/s
  clamp(radius, -2000, 2000); //def max and min radius in mm

  Roomba.write(137);
  Roomba.write(velocity >> 8);
  Roomba.write(velocity);
  Roomba.write(radius >> 8);
  Roomba.write(radius);
}
//---------------------------------------------------------------
```

(Continued on the next page.)

```
void driveWheels(int right, int left)
{
  clamp(right, -500, 500);
  clamp(left, -500, 500);

  Roomba.write(145);
  Roomba.write(right >> 8);
  Roomba.write(right);
  Roomba.write(left >> 8);
  Roomba.write(left);
  }
//----------------------------------------------------------------
void driveWheelsPWM(int rightPWM, int leftPWM)
{
  clamp(rightPWM, -255, 255);
  clamp(leftPWM, -255, 255);

  Roomba.write(146);
  Roomba.write(rightPWM >> 8);
  Roomba.write(rightPWM);
  Roomba.write(leftPWM >> 8);
  Roomba.write(leftPWM);
```

Note that the "clamp" function defines the maximum and minimum values that are allowed to input. This function is defined in the file roombaDefines.h:

```
#define clamp(value, min, max) (value < min ? min : value > max ? max :
value)
```

Using the above code, simpler functions can be created to drive the robot around:

```
/------------------------------------------------------------------
void turnCW(unsigned short velocity, unsigned short degrees)
{
  drive(velocity, -1);
  clamp(velocity, 0, 500);
  delay(6600);
  drive(0,0);
}
//-----------------------------------------------------------------
void turnCCW(unsigned short velocity, unsigned short degrees)
{
  drive(velocity, 1);
  clamp(velocity, 0, 500);
  delay(6600);
  drive(0,0);
}
//-----------------------------------------------------------------
void driveStop(void)
{
  drive(0,0);
}
//-----------------------------------------------------------------
void driveLeft(int left)
{
  driveWheels(left, 0);
}
//-----------------------------------------------------------------
void driveRight(int right)
{
  driveWheels(0, right);
}
```

Note that to turn in angle, the "delay" argument must be calculated specifically for a given speed. Below are some examples that can be used for testing the motors:

```
turnCW (40, 180);  // spin clockwise 180 degrees and stop
driveWheels(20, -20); // spin
driveLeft(20); // turning left
```

For testing the motors, it's good to add an external push button (in this case connected to Arduino pin 12), so you can download the code to Arduino, starting the Create 2, but stopping the execution until the push button is pressed. Usually, for motors testing you can do it on the set-up part of the code.

As an example, please see the simple Arduino code below (note that the code uses functions and definitions developed before):

```
#include "roombaDefines.h"
#include <SoftwareSerial.h>
// Roomba Create2 connection
int rxPin=10;
int txPin=11;
SoftwareSerial Roomba(rxPin,txPin);
//---------------------------------------------
void setup()
{
  Roomba.begin(19200);

  pinMode(ddPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP); // connected to Arduino pin 12 and
used for "starting"
  delay(2000);
```

(Continued on the next page.)

```
  wakeUp ();   // Wake-up Roomba
  startSafe(); // Start Roomba in Safe Mode
  while (digitalRead(buttonPin)) {  } // wait button to be pressed to 
continous run code

  turnCW (40, 180);  //test Roomba spin clock-wise 180 degrees and 
stop
}
//---------------------------------------------
void loop()
{

}
```

Step 5: Controlling Create 2 via Bluetooth

To complete the project, let's install a Bluetooth module (HC-06) to our Arduino. The above diagram shows how to do it. Usually the HC-06 is set-up from factory with a baud rate of 9,600. It is important that you change it to 19,200 to be compatible with Arduino-Roomba communication speed. You can do that sending an AT command to the module (AT+BAUD5 where "5" is the code for 19,200).

If you have any doubt about how the HC-06 works, please have a look at my tutorial: Connecting "stuff" via Bluetooth / Android / Arduino.

To control the Roomba, we will use a generic App that was developed to control mobile robots, using the MIT AppInventor 2: "MJRoBot BT Remote Control". The app can be downloaded for free from MIT's App Inventor via the link: Blue Tooth Remote Control App.

The app has a simple interface, allowing you to send commands to the BT module in both TEXT mode or directly via pre-programmed buttons (each time a button is pressed, a character is sent):

- w: Forward
- s: Backward
- d: Right
- a: Left
- f: Stop
- p: ON / OFF (not used on this first part)
- m: Manual / Automatic (Used to restart Roomba if an obstacle as a cliff is found in Safe Mode)
- +: Speed +
- -: Speed -

You can also send other commands as text if necessary. There is also a text window for messages received from the BT module. This feature is very important during the testing phase, it can be used in the same way as the “Serial Monitor”.

The ***loop()*** part of the code will be “listening” to the Bluetooth device and depending on the command received, take an action:

```
void loop()
{
   checkBTcmd();  // verify if a comand is received from BT remote
control
   manualCmd ();
}
```

The function ***checkBTcmd()*** is shown below:

```
void checkBTcmd()  // verify if a command is received from BT remote
control
 {
    if (BT1.available())
    {
      command = BT1.read();
      BT1.flush();
    }
 }
```

Once a command is received, the function ***manualCmd()*** will take the appropriated action:

```
void manualCmd()
{
  switch (command)
  {

    case ‘m’:
      startSafe();
      Serial.print(“Roomba in Safe mode”);
      BT1.print(“Roomba BT Ctrl OK – Safe mode”);
      BT1.println(‘\n’);
      command = ‘f’;
      playSound (3);
      break;
```

(Continued on the next page.)

```
case 'f':
      driveStop(); //turn off both motors
      writeLEDs ('s', 't', 'o', 'p');
      state = command;
      break;
case 'w':
      drive (motorSpeed, 0);
      writeLEDs (' ', 'g', 'o', ' ');
      state = command;
```

Step 6: Conclusion

The complete Arduino code used in this tutorial and other related documents can be found at Marcelo's GITHUB: Roomba_BT_Ctrl.

Please note that not all Roomba actuators were commented on this tutorial. There are other motors, used for cleaning, other LEDs used for schedule, buttons, sensors, etc.

Several of the functions were created for this program were based on the Create 2 library developed by Dom Amato. You can download the complete library at: https://github.com/brinnLabs/Create2.