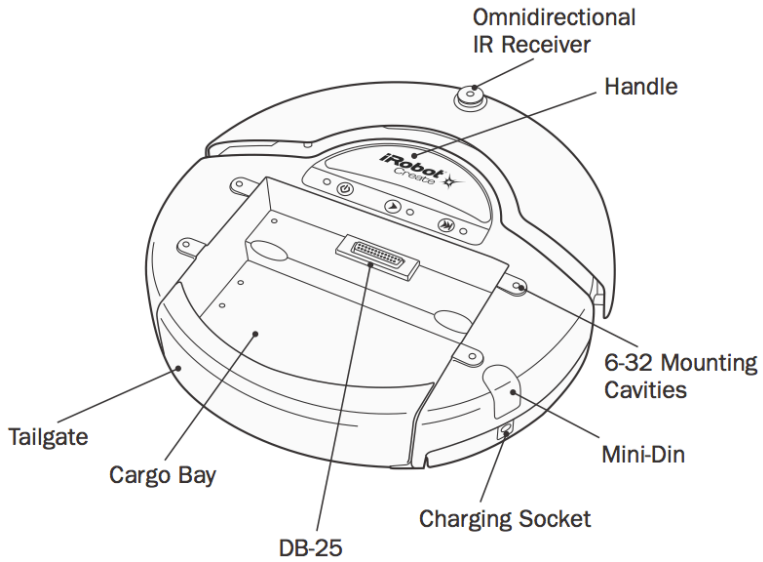
AD61600 Robots and Culture
Prof. Fabian Winkler
Spring 2014

## Arduino and iRobot Create

This workshop is about strategies and techniques controlling the iRobot Create with an Arduino board. This workshop is based on information in the Create Open Interface manual and adapted to Arduino code. You can find a copy of the Open Interface manual here:
http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf

**Introduction**
iRobot Create hardware features:




more information about the iRobot Create can be found in the robot's manual:
http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Manual_Final.pdf

**Sensors**
The iRobot Create comes with some basic sensors mounted on its platform, they are:

- 2 bump sensors (left, right)
- 4 cliff sensors (left, front left, right, front right)
- 3 wheel drop sensors (left, right, caster (center))
- 2 buttons (labeled advance and play)
- infrared sensor (receiving bytes sent from Roomba remote, home base or user created devices)
- 1 wall sensor (used with the virtual wall accessory: wall detection/signal strength)

Furthermore the iRobot Create allows access to sensor monitoring some of its internal functions:

- distance the robot has traveled since the distance was last requested (in millimeters)
- angle the robot has turned since the last time the angle was requested (in degrees)
- charging state
- battery voltage (in mV)
- current (flowing into the robot's battery, in mA)
- battery temperature (in degrees Celsius)
- battery charge (in mAh)
- battery capacity (in mAh)
- velocity information (left, right, in mm/s)

**Actuators**

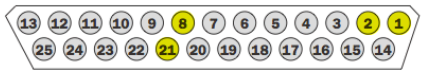
The iRobot Create Open interface allows access to the following on-board actuators

- wheel motors (speed left, speed right)
- LEDs (play, advance, bi-color power LED)
- speaker (define musical notes)

Using the Arduino board as a controller, other sensors and actuators can be added easily to the Arduino board's I/O pins (e.g. photocells, ultrasonic range finders, servo motors, etc.)

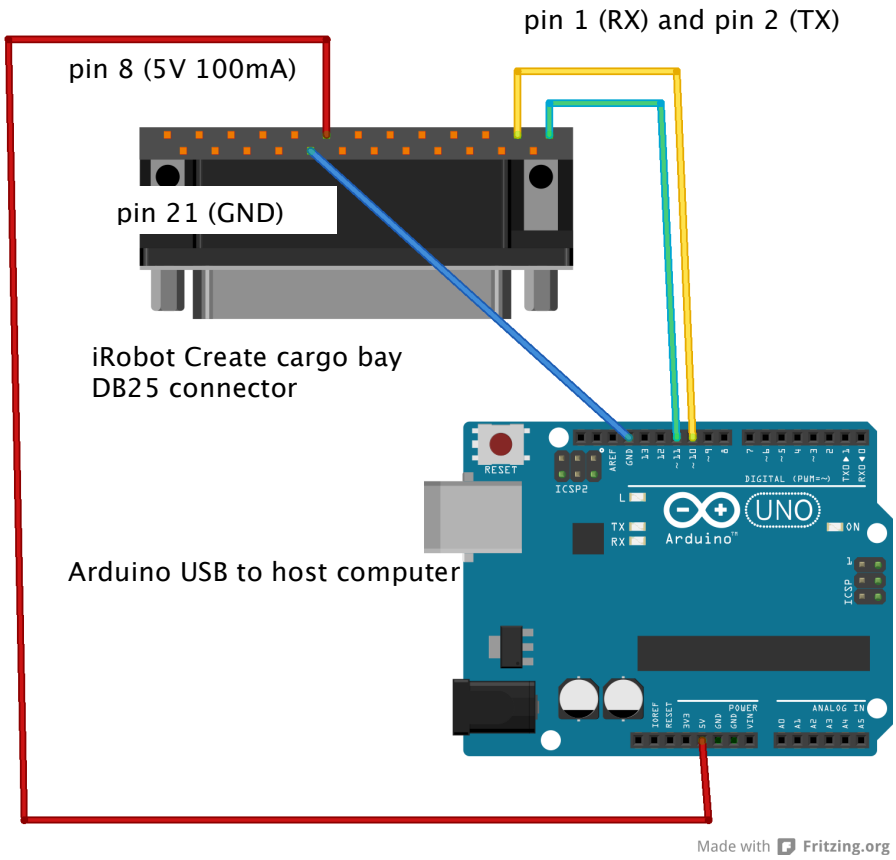
**Connecting the Arduino board**

Connecting the Arduino board is easy through the cargo bay DB25 connector, in the diagram below we see the necessary pins:




| Pin | Name | Description |
|---|---|---|
| 1 | RXD | 0 – 5V Serial input to Create |
| 2 | TXD | 0 – 5V Serial output from Create |
| 3 | Power control toggle | Turns Create on or off on a low-to-high transition |
| 4 | Analog input | 0 - 5V analog input to Create |
| 5 | Digital input 1 | 0 - 5V digital input to Create |
| 6 | Digital input 3 | 0 - 5V digital input to Create |
| 7 | Digital output 1 | 0 - 5V, 20 mA digital output from Create |
| 8 | Switched 5V | Provides a regulated 5V 100 mA supply and analog reference voltage when Create is switched on |
| 9 | Vpwr | Create battery voltage (unregulated), 0.5A |
| 10 | Switched Vpwr | Provides battery power @ 1.5 A when Create is powered on. |
| 11 | Switched Vpwr | Provides battery power @ 1.5 A when Create is powered on. |
| 12 | Switched Vpwr | Provides battery power @ 1.5 A when Create is powered on. |
| 13 | Robot charging | When Create is charging, this pin is high (5V) |
| 14 | GND | Create battery ground |
| 15 | Device Detect/Baud Rate Change Pin | 0-5V digital input to Create which can also be used to change the baud rate to 19200 (see below) |
| 16 | GND | Create battery ground |
| 17 | Digital input 0 | 0 - 5V digital input to Create |
| 18 | Digital input 2 | 0 - 5V digital input to Create |
| 19 | Digital output 0 | 0 - 5V, 20 mA digital output from Create |
| 20 | Digital output 2 | 0 - 5V, 20 mA digital output from Create |
| 21 | GND | Create battery ground |
| 22 | Low side driver 0 | 0.5A low side driver from Create |
| 23 | Low side driver 1 | 0.5A low side driver from Create |
| 24 | Low side driver 2 | 1.5A low side driver from Create |
| 25 | GND | Create battery ground |

Connecting an Arduino UNO board to the iRobot Create:




**Reading basic sensor values**

We are making use of Arduino's softSerial library, so we can work with two independent serial ports on one Arduino board. The reason for this is simple: in order to see if the Arduino can read OI sensor values we need to be able to send requests to OI that return values (serial call and response) and at the same time communicate these readings back to the Arduino serial monitor so we can see what values we get. In a later step we can then trigger driving commands based on the sensor readings. For now however we are just interested in reading sensor values.

Why do we need two serial ports for this?
It's simple – we do not want to get our serial values/commands mixed up – OI commands that we send to the iRobot Create should not show up in the serial monitor and values we send back to the serial monitor should not be interpreted as OI commands.

Why don't we connect the Arduino UNO's hardware serial port (RX/TX)?

> "The serial port output TXD from the Roomba/Create is too weak to drive the RX serial port (hardware serial port) input of an Arduino properly. This is because of the USB-Serial converter on the Arduino: it also tries to drive the RX serial port input via a pullup resistor, but the Roomba does not have enough drive to pull the RX down below about 2.5 volts, which is insufficient to be reliably detected as a TTL serial input of 0."

(from: http://projectsfromtech.blogspot.com/2013/11/irobot-create-arduino-control.html).

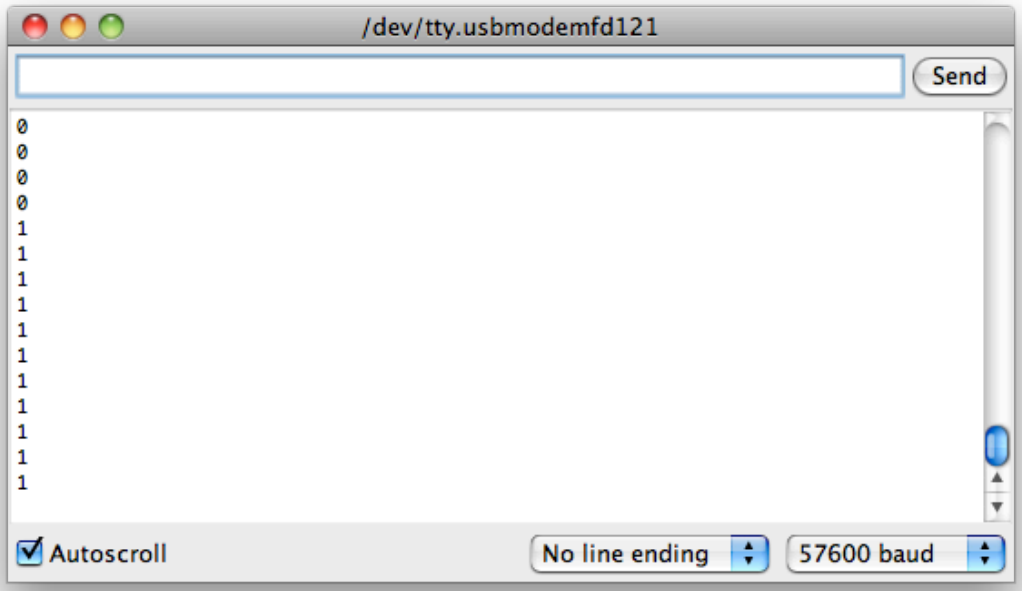
Notes on the Arduino Mega: the Arduino Mega has 4 serial ports, use **Serial1**(TX1/RX1), **Serial2**(TX2/RX2) or **Serial13**(TX3/RX3) instead of **Serial0**. You can still use Serial0 to communicate back to a serial monitor via USB or XBee for debugging if needed.

Here is the Arduino code to read the status of the left cliff sensor:

```
#include <SoftwareSerial.h>

 //*   RX is digital pin 10 (connect to TX of other device - iRobot DB25
 //    pin 2)
 //*   TX is digital pin 11 (connect to RX of other device - iRobot DB25
 //    pin 1)

#define rxPin 10
#define txPin 11
// set up a new software serial port:
SoftwareSerial softSerial =  SoftwareSerial(rxPin, txPin);

int inByte = 0;         // incoming serial byte

/*************************************************************
SETUP
*************************************************************/
void setup()
{
delay(2000); // Needed to let the robot initialize

// define pin modes for software tx, rx pins:
   pinMode(rxPin, INPUT);
   pinMode(txPin, OUTPUT);
// start the the SoftwareSerial port 57600 bps (robot's default)
   softSerial.begin(57600);

// start hardware serial port
Serial.begin(57600);

softSerial.write(128); // This command starts the OI.
softSerial.write(131); // set mode to safe (see p.7 of OI manual)

}
/*************************************************************
LOOP
*************************************************************/
void loop()
{
  softSerial.write(142);  // requests the OI to send a packet of
                          // sensor data bytes
    softSerial.write(9);  // request cliff sensor value specifically
  delay(250); // poll sensor 4 times a second
  if (softSerial.available() > 0) {
    inByte = softSerial.read();
  }
  Serial.println(inByte);
}
```

(download: http://web.ics.purdue.edu/~fwinkler/AD61600_S14/helloCliffSensor.zip)

When you lift the iRobot Create this is what it should look like in your Arduino serial window, the values of the cliff sensor are changing from 0 (on the ground) to 1 (in the air):




**Making the Robot go Forward**
Here is a simple function in Arduino code that makes the robot move forward, it is based on the Drive serial sequence documented in the Open Interface manual on page 9: `[137] [Velocity high byte] [Velocity low byte] [Radius high byte] [Radius low byte]`

Making the robot go forward at a speed of 200 mm/s translates into the following Arduino function:

```
void goForward() {

//Serial sequence: [137] [Velocity high byte] [Velocity low byte] [Radius
//high byte] [Radius low byte]
  softSerial.write(137);       // Opcode number for DRIVE

// Velocity (-500 – 500 mm/s)
  softSerial.write((byte)0);   // 0x00c8 == 200
  softSerial.write((byte)200);
// Radius (-2000 – 2000 mm)
// Special case: straight = 32768 or 32767 = hex 8000 or 7FFF
  softSerial.write((byte)128);  // hex 80
  softSerial.write((byte)0);    // hex 00
}
```

Integrated into a full Arduino sketch it looks like this:

```
#include <SoftwareSerial.h>

 // RX is digital pin 10
 // (connect to TX of other device - iRobot DB25 pin 2)
 // TX is digital pin 11
 // (connect to RX of other device - iRobot DB25 pin 1)
#define rxPin 10
#define txPin 11

// set up a new software serial port:
SoftwareSerial softSerial =  SoftwareSerial(rxPin, txPin);

void setup() {
  delay(2000); // NEEDED!!!! To let the robot initialize

  // define pin modes for software tx, rx pins:
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);

  // set the data rate for the SoftwareSerial port, this is the
  // iRobot's default rate of 57600 baud:
  softSerial.begin(57600);

  softSerial.write(128); // This command starts the OI. You must
                         // always send the Start command before
                         // sending any other commands to the OI
  softSerial.write(131); // safe mode
}

void loop() {
  goForward();
}


 void goForward() {
  softSerial.write(137);       // Opcode number for DRIVE
  // Velocity (-500 – 500 mm/s)
  softSerial.write((byte)0);   // 0x00c8 == 200
  softSerial.write((byte)200);
  // Radius (-2000 – 2000 mm)
  // Special case: straight = 32768 or 32767 = hex 8000 or 7FFF
  softSerial.write((byte)128);  // hex 80
  softSerial.write((byte)0);    // hex 00
}
```

(download: http://web.ics.purdue.edu/~fwinkler/AD61600_S14/goForward.zip)

**Simple Robot Behavior: Bump and Run**

This is one of the simplest behaviors for a mobile robot that still keeps it relatively autonomous. The robot moves forward until it bumps into an obstacle, depending on which bump sensor was triggered, the robot will drive backward a bit, then turn in the opposite direction of the bump sensor that was triggered and continue to move forward.

Here is a function that returns the state of the bump sensors:

```
void checkBumpSensors() {
  char sensorbytes[10]; // variable to hold the returned 10 bytes
                        // from iRobot Create

  softSerial.write((byte)142); // get sensor packets
  softSerial.write((byte)1);   // sensor group packet ID 1, size 10
                               // bytes, contains packets: 7-16
  delay(64);

  // wipe old sensor data
  char i = 0;
  while (i < 10) {
    sensorbytes[i++] = 0;
  }
  i = 0;

  while(softSerial.available()) {
    int c = softSerial.read();
    sensorbytes[i++] = c;
  }

  bumpRight = sensorbytes[0] & 0x01;
  // if right bumper is triggered sensorbytes[0] is: 00000001
  // bitwise AND with 0x01, i.e. 00000001 equals 1
  // see: http://arduino.cc/en/Reference/BitwiseAnd
  bumpLeft = sensorbytes[0] & 0x02;
  // if left bumper is triggered sensorbytes[0] is: 00000010
  // bitwise AND with 0x02, i.e. 00000010 equals 2

  // So if the right bumper is triggered bumpRight is 1
  // (if not triggered then 0)
  // if the left bumper is triggered bumpLeft is 2
  // (if not triggered then 0)

  Serial.print(bumpRight);
  Serial.print("   ");
  Serial.println(bumpLeft);
}
```

Now, can you write some code that implements the Bump and Run behavior? Check the special cases in the Drive serial codes for stopping the motors and look up driving in a radius (iRobot Create Open Interface manual, p.9).

**Hexadecimal, Decimal and Binary Number Representations**
In order to better understand how to interpret numbers sent to and received from the iRobot Create we need to look at the representation of numbers in different number systems. The following diagram tries to compare three important number systems:

| decimal | hexadecimal | | binary | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $2^7$ (=128) | $2^6$ (=64) | $2^5$ (=32) | $2^4$ (=16) | $2^3$ (=8) | $2^2$ (=4) | $2^1$ (=2) | $2^0$ (=1) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 10 | 0 | A | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 11 | 0 | B | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 12 | 0 | C | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 13 | 0 | D | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 14 | 0 | E | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 15 | 0 | F | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 16 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 17 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 18 | 1 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Hexadecimal numbers are convenient (but difficult to understand initially!) because they can hold a 4 bit value (rather than sending or receiving 4 individual bits). A byte (8bits) can be expressed using only 2 hexadecimal numbers.

This little online calculator can be helpful: http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html (this one does not do negative numbers correctly for the iRobot Create though!)

**How to express negative decimal numbers in the hexadecimal system?**
Suppose we want to find the hexadecimal representation of -53 (base 10). First, we convert 53 to binary:

53 (base 10) = 110101 (base 2)

Next, we take the two's complement. We'll use 8-bit values, so we have to fill the value to 8 bits by adding 2 leading zeroes on the left. Then invert the bits and add 1. We then have:

```
  00110101   (positive value)
      |
      v
  11001010   (invert)
+        1   (add 1)
 ----------
  11001011   (two's complement negative result)
```

This is how -53 is represented in two's complement signed binary. The final step is to convert the binary to hexadecimal. We simply group the bits into groups of four, then convert each group to its hexadecimal equivalent, like so:

```
1100 1011
\__/ \__/
 12   11
  C    B
```

So -53 (base 10) = CB (signed hexadecimal using complements).

velocity -200:

200 in binary is 00000000 11001000 (we need to add 8 zeros to the left because we need to send 2 bytes)

two's complement:

```
  11001000

  00110111
+        1
---------
  00111000
```

because:

```
    1
   +1
   ----
    10  (carry result to the next bit to the left)
```

So, 00111000 (binary) = 56 (decimal)

**Next steps:**

- Monitoring iRobot Create's behaviors (battery, distance travelled, turning angles, etc.)
- Wireless serial communication with the Arduino on the iRobot Create using XBee.
- More fun with bits and bytes on the Arduino, merging 2 bytes into one decimal number
- Working with timers on the Arduino (rather than the delay() function)

**Further resources:**

Helpful tutorials, project notes on controlling the iRobot Create via Arduino
http://projectsfromtech.blogspot.com/2013/11/irobot-create-arduino-control.html
http://www.netfluvia.org/layer8/?p=127

Roomba library for Arduino (not tested!)
http://www.airspayce.com/mikem/arduino/Roomba/

Emitter Follower circuit to connect the iRobot Create to Serial0 or RX/TX on the Arduino board: http://www.airspayce.com/mikem/arduino/Roomba/Create-Arduino.pdf

Kurt, Tod E. *Hacking Roomba*. Indianapolis, IN: Wiley,2007.