Makeblock Ranger Arduino Coding Reference
Murray Elliot, May 2023 (v1.4.3)

This document was inspired by Victor Leung's excellent document and encouraged by my own frustration at MakeBlock's complete lack of any decent Arduino documentation and helped hugely by Gosse Adema's excellent article.

Makeblock overview

Makeblock design and manufacture a number of robot kits and modules. Each robot kit is based on a number of control boards - each control board is different. At the time of writing there is the MegaPi Pro, the mCore, the Orion, the me Auriga and the MegaPi (you can find out more about each board here: https://www.makeblock.com/project_category/main-control-boards).

Mbots are designed for use with Mblock, and Arduino is just a side effect of that, so you'll discover that there's little of the kind of documentation you might expect to help an Arduino/C++ coder get started.

In addition to the kits, there are a large number of additional components - sensors, servos, displays, mechanicals which can be added to the robot, once you do that then you need even more knowledge on how to a) connect these devices correctly and b) how to program them in your code. We will hopefully cover additional modules in later editions of this guide.

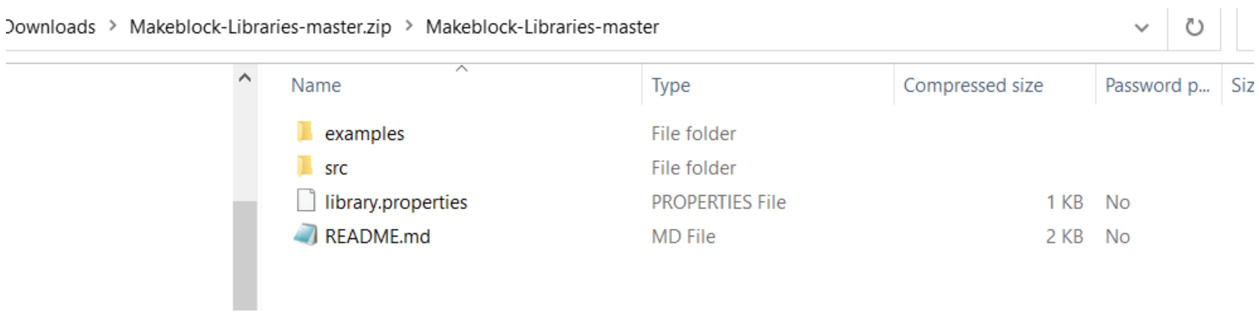
Installing and configuring the IDE

IDE = Integrated Development Environment. If you're unsure of how to write C++ code for an Arduino device, then you need to start with installing the IDE, and reading some of the online documentation. The IDE allows you to write, compile and upload C++ to Arduino boards. Arduino IDE runs on Windows, Mac OS X and Linux.

Full instructions on how to set up the Arduino IDE are here:
https://makeblocksupport.zendesk.com/hc/en-us/articles/1500004053721-Programming-mBot-Ranger-in-Arduino

Install the Makeblock Library
(Follow instructions on the GitHub page)
[GitHub - Makeblock-official/Makeblock-Libraries: Arduino Library for Makeblock Electronic Modules, learn more from Makeblock official website]

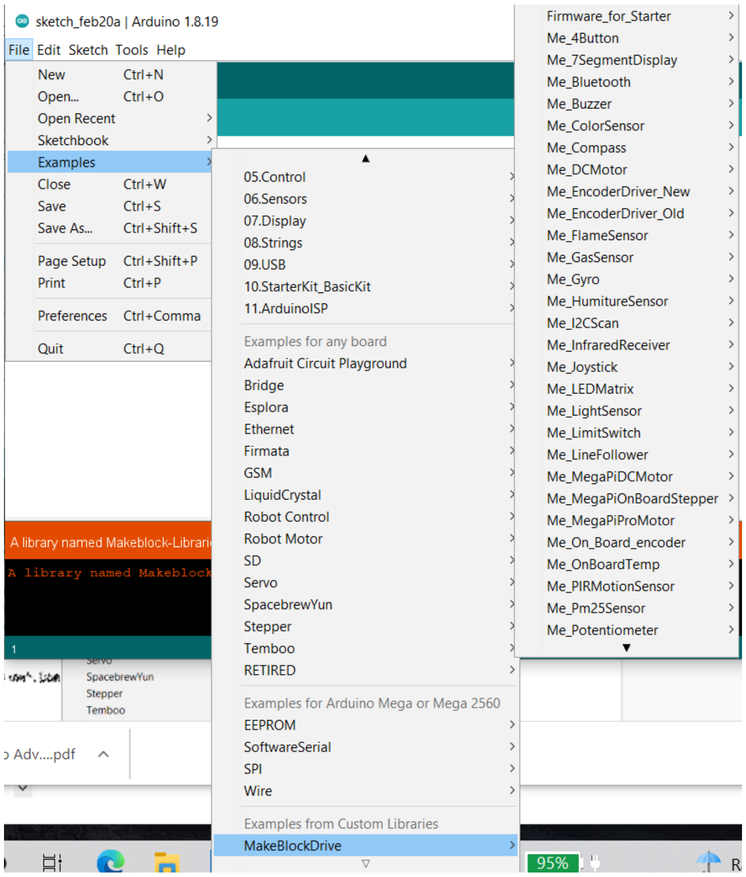
The folder structure of the downloaded library should look like this:


Downloads > Makeblock-Libraries-master.zip > Makeblock-Libraries-master

Name	Type	Compressed size	Password p...	Siz
examples	File folder			
src	File folder			
library.properties	PROPERTIES File	1 KB	No	
README.md	MD File	2 KB	No	

Once, you have installed the libraries into Arduino (see above link for instructions, and also in the README.md file), on the Arduino IDE you will notice a new folder that contains several examples, some of them may not be valid for your mBot Ranger board (as general rule if they contains `#include <MeAuriga.h>` they programmed for mBot Ranger board)

If the installation of the libraries and examples is correct on your Arduino IDE the following entry will appear



Install the mBlock IDE
The mBlock IDE is required, because it installs the proper serial port driver needed to flash (or upload) the compiled sketch with the Arduino IDE.

In term of C++/Arduino, the examples are a bit of a mix - they assume some knowledge, i.e. that the main boards are all different in terms of how they're wired up, so an Orion isn't compatible with an Auriga etc etc, they are all written for a specific board, not all examples will work on the base kit (because there are examples for modules which are supplied as 'extras'). In order to figure it all out, some ability to read schematics and translate that information into code would be helpful. This guide is all about the base mbot Ranger model. There will be more examples for additional modules in the appendices (when I get around to writing them)

mBot Ranger's main board is the me-auriga.
(https://makeblocksupport.zendesk.com/hc/en-us/articles/4412149618327-About-Me-Auriga)
This board is compatible with Arduino Mega 2560, and that's the board you will use when programming in Arduino.

Mac Driver
If using Arduino/Ranger with a Mac, then you will need an additional driver :

http://blog.sengotta.net/signed-mac-os-driver-for-winchiphead-ch340-serial-bridge/ this driver enables you to connect to the Me Auriga chipset like it's a serial port. It took me various attempts with various drivers, but this is the one that worked for me on my 2012 Macbook Pro running macOS 10.13.6

For macOS Mojave (10.14) and later Apple provides a driver for these USB-serial bridges, so the driver should not be installed.
See https://github.com/adrianmihalko/ch340g-ch34g-ch34x-mac-os-x-driver

Hardware

Pin Mapping on the Auriga is defined in the meAuriga.h header file. There is a board layout diagram towards the rear of this document. It will no doubt be useful to familiarise yourself with the various schematics at some point you will need them! If you don't know how to read a schematic, don't worry, most of the work is done for you in this document.

Note that D means a digital pin, and A means an analogue pin. It's important to differentiate these, please check out the following tutorials on arduino pins if you're not familiar with the concept:
https://www.arduino.cc/en/Tutorial/DigitalPins
https://www.arduino.cc/en/Tutorial/AnalogInputPins

```
// On-board output devices

D44/PWM = 12 x ws12812 LED - the onboard ring LED
D45/PWM = Buzzer - the passive buzzer
D13 = Blue LED - a single LED on the board

// On-board sensors - some of these can be addressed via PORTS definition (see
below)

A0 = Temperature sensor (PORT_13)
A1 = Sound sensor (PORT_14)
A2 = Light sensor 2 (PORT_12)
A3 = Light sensor 1 (PORT_11)
A4 = Battery voltage sensor

// Motor driver pins - the motors supplied are encoder motors. These can be
accessed via the library rather than directly, but it's useful to know the pin
numbers

D19/RX1 = ENA A // Enable pins
D42 = ENA B  // Enable pins
D11/PMW = PWMA // Motor speed
D49 = DIR A2 // Motor direction
D48 = DIR A1 // Motor direction (opposite of A2)

D18/TX1 = ENB A  // Enable pins
D43 = ENB B // Enable pins
```

```
D10/PMW = PWMB // Motor speed
D47 = DIR B1 // Motor direction
D46/PWM = DIR B2  // Motor direction (opposite of B1)


There is a servo port on this board (Arduino pin D16/TX2 and D17/RX2). But this is
covered by the expansion board

D0 and D1 are connected to the Blue and Red LEds  respectively. These pins are also
connected to the BLE (Bluetooth Low Energy) and UART modules (Universal
Asynchronous Receiver/Transmitter), so programming these LEDs directly will disrupt
these modules.
```

PORT mapping of the Makeblock PORTs (telephone plugs).

```
// Each Makeblock port is connected to two Arduino pins. These pins are identified
as slot1 and slot2 (S1,S2) and are numbered in the arrays below

MePort_Sig mePort[15] - this array holds the pin numbers
(mePort[].s1 = pin number of slot 1, mePort[].s2 = pin number of slot 2)

PORT_0  {  NC,  NC }   Not connected
PORT_1  {   5,   4 }   red - motor driver
PORT_2  {   3,   2 }   red - motor driver
PORT_3  {   7,   6 }   red - motor driver
PORT_4  {   9,   8 }   red - motor driver
PORT_5  {  16,  17 }   grey - serial comms
PORT_6  { A10, A15 }   universal - Analogue ports
PORT_7  {  A9, A14 }   universal
PORT_8  {  A8, A13 }   universal
PORT_9  {  A7, A12 }   universal
PORT_10 {  A6, A11 }   universal
PORT_11 {  NC,  A2 }   light sensor 1
PORT_12 {  NC,  A3 }   light sensor 2
PORT_13 {  NC,  A0 }   temperature sensor
PORT_14 {  NC,  A1 }   sound sensor
```

Although this is a lot of information, it does not mean programming the Makeblock components is difficult. It is **especially important** to know which part is connected to which port. This is specified once when defining the object. After that, this information is no longer necessary

Most of the libraries for the various components have constructor code which can take either pin numbers or slot numbers or port numbers. This can be useful but also confusing for the first time coder.

Each Auriga port has 6 pins. SCL, SDA, GND, VCC, S1 and S2. Not all modules use all of these ports. Some only use the VCC, GND, and a single data port.

SCL is the I2C clock bus. SDA is the I2C data bus (see later Appendices for more information on I2C)

Red ports (1-4) have Output voltage of 6-12 Volt and one or two digital ports.
Blue/Yellow/Grey/White ports (6-10) have one or two analogue ports, one or two digital ports and an I2C port.

The light grey port (Port 5) is a serial port and has 4 pins : GND, 5V, TX2/D16, RX2/D17

Output - 12xRGB Led (On board)

The Auriga board has 12 multicolour LEDs, these are controlled via pin 44 and the device is a WS2812. The Auriga board can provide up to 3A to drive RGB LEDs. Each LED uses about 60mA, giving a maximum of about 50 LEDs.

To construct the LED object: This instructs the constructor to use port 0 and 12 LEDs

```
MeRGBLed led( 0, 12 ); // the port is irrelevant because it's actually just connected to a pin which has no port.
```

To initialise the LEDs `// LED Ring controller is on Auriga D44/PWM`

```
led.setpin( 44 );
```

To set the value of a specific LED

```
led.setColorAt( t, red, green, blue );
// t = LED index 0-11
// 11 = 9 o'clock position (assuming power connector at bottom)
// 2 = 12 o'clock position
// 5 = 3 o'clock position
// 8 = 6 o'clock position
// Red, green, blue are byte values between 0 and 255
```

To set the value of a specific LED or all LEDs

```
led.setColor( t, red, green, blue );
// t = LED index 1-12 or 0 for all LEDs
// Red, green, blue are byte values between 0 and 255
// Note - this function simply calls setColorAt (sending LED index-1 into the function)
// Can also call this with two parameters where the second is the LED color defined as long type, for example (white) = 0xFFFFFF
```

To write the pattern to the LEDs (incorporates a 500us delay)

```
led.show();
```

Sample code below will send a spinning colour pattern to the LEDs...

```
#include <MeAuriga.h>

#define ALLLEDS        0
// Auriga on-board light ring has 12 LEDs
#define LEDNUM  12
// on-board LED ring, at PORT0 (onboard)
MeRGBLed led( 0, LEDNUM );

float j, f, k;

void setup()
{
led.setpin( 44 );
}

void loop()
{
  color_loop();
}

void color_loop()
{
  for (uint8_t t = 0; t < LEDNUM; t++ )
  {
    uint8_t red = 64 * (1 + sin(t / 2.0 + j / 4.0) );
    uint8_t green = 64 * (1 + sin(t / 1.0 + f / 9.0 + 2.1) );
    uint8_t blue = 64 * (1 + sin(t / 3.0 + k / 14.0 + 4.2) );
    led.setColorAt( t, red, green, blue );
  }
  led.show();

  j += random(1, 6) / 6.0;
  f += random(1, 6) / 6.0;
  k += random(1, 6) / 6.0;
}
```

Output - Encoder Motors (Left and right wheel)

The Auriga board contains an onboard encoder driver with two ports. And the motor speed is controlled by PWM. This signal is made by the microcontroller and allows the microcontroller to perform other things while driving. Such as checking the distance to an object. An encoder motor is different from a DC motor in that it can provide feedback as to distance travelled.

The functions provided by the MeEncoderOnBoard libraries are pretty complex, so the following example is quite basic, but will get you going. It moves the ranger forward, backward, turns left then turns right.

NOTE - If plugged into the PC, it's easy to forget the power's off. Make sure the GREEN led is lit on the board - this means battery power is being applied to the motors. If not, press the red button to turn the power on. It IS SAFE to connect both the battery supply and the usb cable, it is also safe to turn on the battery power while the USB cable is connected.

```
#include <MeAuriga.h>
MeEncoderOnBoard Encoder_1(SLOT1);
MeEncoderOnBoard Encoder_2(SLOT2);
int16_t moveSpeed = 200;

void Forward(void)
{
  Encoder_1.setMotorPwm(-moveSpeed); // setMotorPwm writes to the encoder controller
  Encoder_2.setMotorPwm(moveSpeed);  // so setting the speed change instantly
}
void Backward(void)
{
  Encoder_1.setMotorPwm(moveSpeed);
  Encoder_2.setMotorPwm(-moveSpeed);
}
void BackwardAndTurnLeft(void)
{
  Encoder_1.setMotorPwm(moveSpeed/4);
  Encoder_2.setMotorPwm(-moveSpeed);
}
void BackwardAndTurnRight(void)
{
  Encoder_1.setMotorPwm(moveSpeed);
  Encoder_2.setMotorPwm(-moveSpeed/4);
}
void TurnLeft(void)
{
  Encoder_1.setMotorPwm(-moveSpeed);
  Encoder_2.setMotorPwm(moveSpeed/2);
}
void TurnRight(void)
{
  Encoder_1.setMotorPwm(-moveSpeed/2);
  Encoder_2.setMotorPwm(moveSpeed);
}
void TurnLeft1(void)
{
  Encoder_1.setMotorPwm(-moveSpeed);
  Encoder_2.setMotorPwm(-moveSpeed);
}
void TurnRight1(void)
{
  Encoder_1.setMotorPwm(moveSpeed);
  Encoder_2.setMotorPwm(moveSpeed);
}
void Stop(void)
{
  Encoder_1.setMotorPwm(0);
  Encoder_2.setMotorPwm(0);
}
void ChangeSpeed(int16_t spd)
{
  moveSpeed = spd;
}
void setup()
{
  Serial.begin(115200);

  //Set PWM 8KHz
  TCCR1A = _BV(WGM10);
  TCCR1B = _BV(CS11) | _BV(WGM12);
  TCCR2A = _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS21);
}

void loop()
{
  ChangeSpeed(100);
  Forward();
  delay(500);
  Backward();
  delay(500);
  TurnLeft1();
  delay(500);
  TurnRight1();
  delay(500);
  Stop();
  delay(1000);
}
```

It’s possible to program the pins directly using PWM:

```
#include <MeAuriga.h>
#define PWMA  11 //Motor Left
#define DIRA1 48
```

```
#define DIRA2 49
#define PWMB  10 //Motor Right
#define DIRB1 47
#define DIRB2 46

void setup(){
int speed1=120; // full
int speed2= 60; // half
int speed3= 30; // turn
int speed4=  0; // stop
  pinMode(PWMA,  OUTPUT);
  pinMode(DIRA1, OUTPUT);
  pinMode(DIRA2, OUTPUT);
  pinMode(PWMB,  OUTPUT);
  pinMode(DIRB1, OUTPUT);
  pinMode(DIRB1, OUTPUT);

// Forward full speed
       analogWrite (PWMA,  speed1);
       digitalWrite(DIRA1, LOW); // forward
       digitalWrite(DIRA2, HIGH);
       analogWrite (PWMB,  speed1);
       digitalWrite(DIRB1, HIGH); // forward (motor faces opposite direction)
       digitalWrite(DIRB2, LOW);
       delay(500);
  // Backward half speed
       analogWrite (PWMA,  speed2);
       digitalWrite(DIRA1, HIGH); // back
       digitalWrite(DIRA2, LOW);
       analogWrite (PWMB,  speed2);
       digitalWrite(DIRB1, LOW); // back (motor faces opposite direction)
       digitalWrite(DIRB2, HIGH);
       delay(1000);
// Stop
       analogWrite (PWMA,  speed4);
       digitalWrite(DIRA1, LOW); // forward
       digitalWrite(DIRA2, HIGH);
       analogWrite (PWMB,  speed4);
       digitalWrite(DIRB1, HIGH); // forward (motor faces opposite direction)
       digitalWrite(DIRB2, LOW);
}
void loop ()
{
}
```

Other examples using different modes of operation can be found in Appendix C

Output - Buzzer

The buzzer is connected to pin 45 on the Auriga

Buzzer constructor

```
MeBuzzer buzzer;
// optional parameters:
//     none: Sets buzzer pin to 8
//     int pin: sets buzzer pin to the variable
//     unit8_t port: Sets RJ25 port
//     unit8_t port, uint8_t slot
```

To set the buzzer pin

```
buzzer.setpin(int buzzerpin); // buzzerpin must be 45 to use default buzzer
```

To play a tone

```
buzzer.tone([int pin], uint16_t frequency, uint32_t duration);
// pin - optional output pin (45 if you're going to use it)
// Frequency (Hz)
// Duration (ms)
```

To cancel playing a tone

```
buzzer.noTone([int pin]); // optional output pin parameter
```

Sample code below will sound the buzzer two times when the program is loaded.
The buzzer.tone() function blocks program execution for the duration of the sound

```
#include <MeAuriga.h>

MeBuzzer buzzer;
#define BUZZER_PORT  45

void setup() {
 buzzer.setpin(BUZZER_PORT);
 buzzer.noTone();


  buzzer.tone(600, 1000);   //Buzzer sounds 600Hz for 1000ms
  delay(2000);              //Pause for 2000ms, Buzzer no sound
  buzzer.tone(1200, 1000);  //Buzzer sounds 1200Hz for 1000ms
  delay(2000);              //Pause for 2000ms, Buzzer no sound
}

void loop(){}
```

Here's some frequencies to play with:

	0	1	2	3	4	5	6	7	8	9
C	16	33	65	131	262	523	1046	2093	4186	8372
C#	17	35	69	139	277	554	1108	2217	4434	8869
D	18	37	73	147	294	587	1174	2349	4698	9397
D#	19	39	78	156	311	622	1244	2489	4978	9956
E	21	41	82	165	330	659	1318	2637	5274	10548
F	22	44	87	175	349	698	1396	2793	5587	11175
F#	23	46	92	185	370	740	1479	2959	5919	11839
G	24	49	98	196	392	784	1567	3135	6271	12543
G#	26	52	104	208	415	831	1661	3322	6644	13289
A	28	55	110	220	440	880	1760	3520	7040	14080
A#	29	58	117	233	466	932	1864	3729	7458	14917
B	31	62	123	247	494	988	1975	3951	7902	15804

The Auriga uses pin 45 for the buzzer, this is a PWM port which also means that the analogwrite() function can be used to write a PWM wave e.g.:

```
#define buzzerOn()  pinMode(45,OUTPUT),analogWrite(45, 127)
#define buzzerOff() pinMode(45,OUTPUT),analogWrite(45, 0)
```

The arduino analogWrite() function generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin

---

Input/Output - Serial ports

The arduino Mega has 4 serial ports on pins 0/1, 19/18, 17/16 and 15/14 (RX/TX respectively). You can connect arduino's together by connecting RX of one device to TX of the other and vice versa

These ports are referred to in the serial library as Serial, Serial2, Serial3 and Serial4. On the Ranger, these have the following function:
Serial - the connection to the PC, this is normally used to send debug information back to the arduino app for display on the PC screen.
Serial1 - the ranger uses these for the motor driver
Serial2 - this is connected to PORT_5 (and also to the SERVO white 4 pin connector under the LED panel)
Serial3 - this is connected to the ""extension header 4"

The following example takes input from PORT_5 and outputs it to the connected PC

```
#include <MeAuriga.h>

void setup() {
// Begin the Serial at 9600 Baud
Serial.begin(9600);
Serial2.begin(9600);
}

void loop() {
Serial2.readBytes(mystr,5); //Read the serial data and store in var
Serial.println(mystr); //Print data on Serial Monitor
delay(1000);
}
```

Input - Black Line Finder Sensor

Sample code below will read the line finder sensor, write the sensed value back to Serial Port. There are only 4 possible resulting values from the sensor. Line is assumed to be black on white paper.

```
#include <MeAuriga.h>

MeLineFollower lineFinder(PORT_9);

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

void loop()
{
  int sensorState = lineFinder.readSensors();
  switch(sensorState)
  {
    case S1_IN_S2_IN:   Serial.println("S1_IN_S2_IN"); break;
    case S1_IN_S2_OUT:  Serial.println("S1_IN_S2_OUT"); break;
    case S1_OUT_S2_IN:  Serial.println("S1_OUT_S2_IN"); break;
    case S1_OUT_S2_OUT: Serial.println("S1_OUT_S2_OUT"); break;
    default: break;
  }
  delay(200);
}
```

Input- Light Intensity Sensor

Sample code below will read one of the light sensors, write the sensed value out to Serial Port. The measured value range from 0 (dimmest) to 1023 (brightest)

```
#include <MeAuriga.h>

MeLightSensor lightSensor(PORT_12);

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

void loop()
{
  Serial.print("value = ");            // Print the results to the serial monitor
  Serial.println(lightSensor.read()); // Brightness value from 0-1023
  delay(50);                           // Wait 50 milliseconds before next measurement
}
```

Auriga has two sensors - one connected to PORT_11 and the other to PORT_12

Input- Temperature Sensor

The Ranger has an on-board temperature sensor.

The sample code below will read the temperature sensor then write the sensed value in degrees Celcius back to Serial Port. Range is -40 C to +125 C

```
#include <MeAuriga.h>


MeOnBoardTemp temp(PORT_13);

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

void loop()
{
  Serial.print("value = ");            // Print the results to the serial monitor
  Serial.println(temp.readValue());   // temp value in Celcius
  delay(50);                           // Wait 50 milliseconds before next measurement
}
```

Input- Sound Sensor

The Ranger has an on-board audio sensor.
The sample code below will read the audio sensor then write the sensed value back to Serial Port. This is a 16-bit integer value. Unknown units. Primary component is an LM2904 low-power amplifier.

```
#include <MeAuriga.h>


MeSoundSensor dbsense(14);

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

void loop()
{
  Serial.print("value = ");           // Print the results to the serial monitor
  Serial.println(dbsense.strength()); // the sound intensity revealed by the sensor
  delay(50);                          // Wait 50 milliseconds before next measurement
}
```

Input/Output- Bluetooth

Not yet documented - sorry!

Input- Ultrasonic Distance Sensor

The sensor emits an ultrasound which travels through the air, and if there is an object on the way it will bounce back to the module. The transmission speed of the sonic wave in the air is 340 m/s (equals 34 cm/ms or 0.034 cm/μs). The time recorded by the sensor, can be used to calculate the distance to the object (this calculation is taken care of by the library thankfully!)

Sample code below will read the ultrasonic distance sensor, write the sensed value back to Serial Port.

The measured value range from 3cm to 400cm.
Closer than 3cm or farther than 400cm measurement will appear as 400cm, it is not possible to distinguish between the two.

```
#include <MeAuriga.h>

MeUltrasonicSensor ultrasonic(PORT_10);

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

void loop()
{
  Serial.print("distance(cm) = ");           // Print the results to the serial monitor
  Serial.println(ultrasonic.distanceCm());  // Distance value from 3cm - 400cm
  delay(50);                                // Wait 50 milliseconds before next measurement
}
```

Input - Gyro

The Gyro will measure position and angle of the device. It uses an MPU-6050, and combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) capable of processing complex 9-axis Motion Fusion algorithms.

https://media.digikey.com/pdf/Data%20Sheets/Makeblock%20PDFs/11012_Web.pdf

This sounds ideal for turn calculation, but beware of gyroscope drift which means the angles will rarely be accurate, but the best you can do with the Ranger kit.

This device is extremely capable, but the libraries provided thankfully simplifies it all for us to a few basic functions:

The X axis angle relates to the front/back angle.
    Nose down 45 degrees = +45. Tail down 45 degrees = -45
    Once the reading goes beyond 90, readings head back towards zero
    (i.e. does not recognise upside down)
The Y axis angle related to side to side angle.
    Right side down 45 degrees = +45. Left side down 45 degrees = -45
    Once the reading goes beyond 90, readings head back towards zero
The Z axis angle related to the turn angle.
    Turn right 45 degrees = +45. Range 0.1 to 179.9
    Turn left 45 degrees = -45. Range -0.1 to -179.9

gyro.begin will reset the gyro axis readings back to zero.
gyro.update will read the gyro position.
gyro.fast_update will read the gyro position, unlike previous function, this function doesn't use filter coefficients based on dynamic read times (in reality, it produces very similar sets of

readings)

gro.deviceCalibration will calibrate the gyro.

getAngle(1) == getAngleX() == returns angle in degrees
getAngle(2) == getAngleY()
getAngle(3) == getAngleZ()

getGyroX() - returns rate of change in X axis - Degrees per second I *think*
getGyroY() - as above for Y axis

Sample code below will read the angle of the rover

```
#include "MeAuriga.h"
#include <Wire.h>

MeGyro gyro_ext(0,0x68);  //external gryo sensor
MeGyro gyro(1,0x69);

void setup()
{
  Serial.begin(115200);
  gyro.begin();
}

void loop()
{
  gyro.update();
  Serial.read();
  Serial.print("X:");
  Serial.print(gyro.getAngleX() );
  Serial.print(" Y:");
  Serial.print(gyro.getAngleY() );
  Serial.print(" Z:");
  Serial.println(gyro.getAngleZ() );
  delay(10);
}
```

Example - Basic Line Following

Sample code below is a basic line following algorithm. Four possible states of the line sensor provides five different motor response.

Left Sensor	Right Sensor	Sensor Reading	Motor Response	Left Motor Power	Right Motor Power
In	In	S1_IN_S2_IN	Go Straight	255	255
In	Out	S1_IN_S2_OUT	Left turn	0	255
Out	In	S1_OUT_S2_IN	Right turn	255	0
Out	Out	S1_OUT_S2_OUT	(If previously left turn) Left Turn	0	255
			(If previously right turn) Right Turn	255	0

```
// This code being redeveloped
```

Example - Ultrasonic Lap Timer (for timing MBot Race)

Example code below will use the ultrasonic sensor as an object detector for timing mBot lap race.

- Buzzer will provide audio feedback when an object is detected.
- Elapsed Time and Lap Time in milliseconds will be printed to serial port.
- Detection distance (10cm) can be adjusted.
- Minimal lap time (1000ms) can be adjusted to avoid mistrigger.

```
#include <MeAuriga.h>

#define BUZZER_PORT  45
MeBuzzer buzzer;
const int buzzerDuration = 20;

MeUltrasonicSensor ultrasonic(PORT_10);
const float distanceThreshold = 10.0;
float distance = 10.0;
boolean detected = false;
unsigned int detectCount = 0;

unsigned long currentTime = 0;
unsigned long firstDetectMills = 0;
unsigned long lastDetectMills = 0;
const float minimumLapTime = 1000; //ms

void setup()
{
  buzzer.setpin(BUZZER_PORT);
  Serial.begin(115200);
  Serial.println("Lap Timer.");
  Serial.println("Trigger the sensor to start timing");
  Serial.print("Sensor detection distance: ");
  Serial.print(distance);
  Serial.println("cm");
  buzzer.tone(600, buzzerDuration);   //Buzzer sounds 600Hz for 1000ms
  delay(100);
  buzzer.tone(600, buzzerDuration);   //Buzzer sounds 600Hz for 1000ms
  delay(100);
  buzzer.tone(600, buzzerDuration);   //Buzzer sounds 600Hz for 1000ms
  delay(100);
  buzzer.tone(900, buzzerDuration * 2); //Buzzer sounds 600Hz for 1000ms
}

void loop()
{
  currentTime = millis();
  if ((currentTime - lastDetectMills) > minimumLapTime) {
    distance = ultrasonic.distanceCm();
    if ((distance < distanceThreshold)) {
      if (!detected) {
        detected = true;
        if (detectCount == 0) {
          firstDetectMills = currentTime;
          buzzer.tone(300, buzzerDuration);   //Buzzer sounds 600Hz for 1000ms
        }
        Serial.print("Lap:");
        Serial.print(detectCount);
        Serial.print(",  Time:");
        Serial.print(currentTime - firstDetectMills);
        Serial.print("ms,  LapTime:");
        Serial.print(currentTime - lastDetectMills);
        Serial.print(" ms, UltrasoundDistance:");
        Serial.print(distance);
        Serial.println(" cm");
        buzzer.tone(600, buzzerDuration);   //Buzzer sounds 600Hz for 1000ms
        detectCount++;
        lastDetectMills = currentTime;
      }
    } else {
      detected = false;
    }//distance < distanceThreshold
  }//minimumLapTime
}//loop
```

Appendix A - Terminology

Explaining some terminology

TX/RX are transmit and receive pins for serial communications. These operate at logic levels (5V or 3.3V) https://www.arduino.cc/reference/en/language/functions/communication/serial/. Do NOT connect these to an RS232 port directly as that uses +/- 12V and will damage the board.

PWM means Pulse width Modulation (The pin can generate a square wave of specified duty cycle (on to off ratio)) A ration of 0 is off, 255 is on, 64 is on 25% of the time. 191 is on 75% of the time. This can be used to control the frequency of a buzzer, speed of a motor or the brightness of an LED (for example) https://www.arduino.cc/en/tutorial/PWM

I2C - this is a communications mechanism which allows a master device to send commands to slaves (and optionally get a response).
Two wires are connected to all devices - the SCL wire (clock) and the SDA wire (data). Everything on the I2C is either a slave or a master. There is normally only one master, and many slaves. On a robot, the controller is the master and the slaves are the various interfaces. Each interface has an address from 0 to 127
https://www.robot-electronics.co.uk/i2c-tutorial
https://dronebotworkshop.com/i2c-arduino-arduino/

The Arduino has a built-in library for working with I2C called the Wire Library. It makes it very easy to communicate on the I2C bus, and it can configure the Arduino to become either a master or a slave.

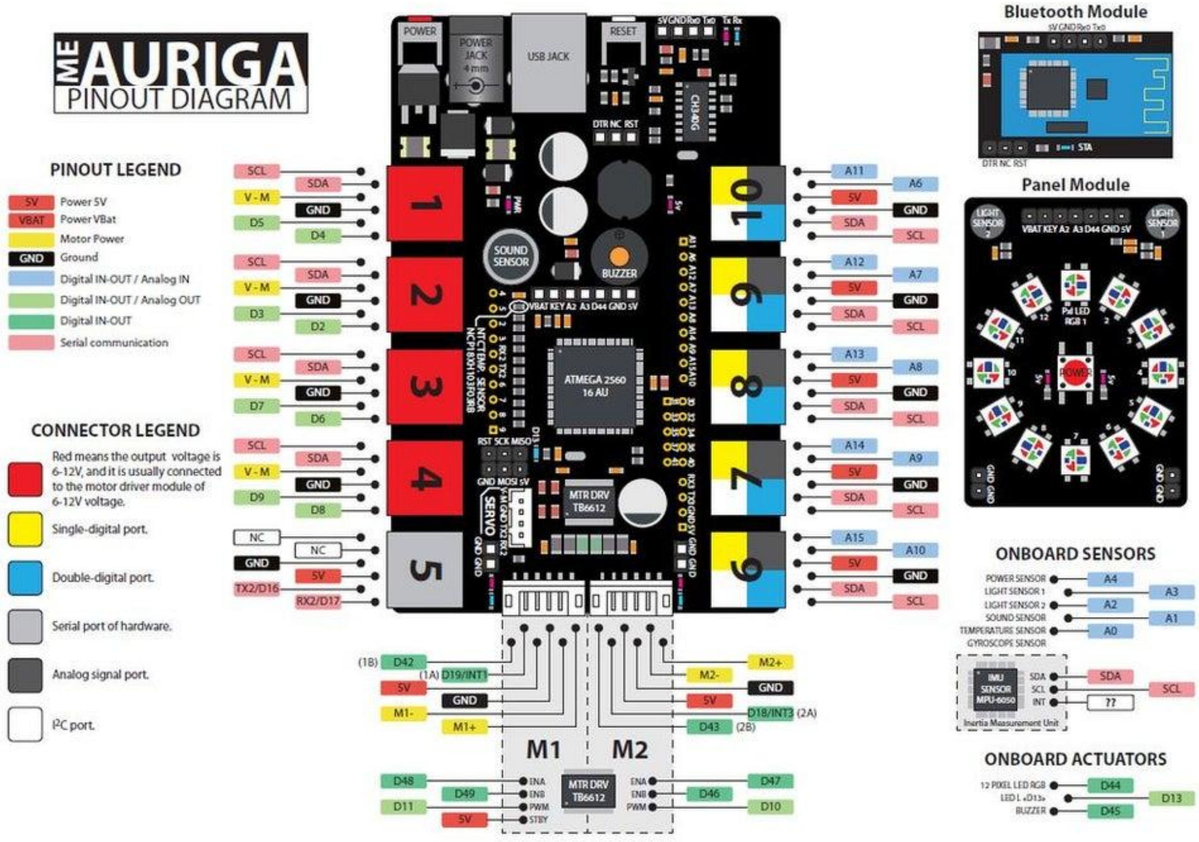
Appendix B - Auriga Board layout

The Me Auriga board
Link to image online:
https://content.instructables.com/FGO/JLQU/J9EHPRJ1/FGOJLQUJ9EHPRJ1.png?auto=webp&frame=1&width=1024&fit=bounds&md=95eecd3a412356cac10502925fa672f4
from this excellent instructable:
https://www.instructables.com/id/Advanced-Makeblock-Sensors-DIY/

You can also find more board details here:
https://support.makeblock.com/hc/en-us/articles/4412149618327-About-Me-Auriga#Schematic%20Diagram%20of%20Electronic%20Components%20of%20Me%20Auriga

And if these links stop working, pinout and v1.1 schematic are here:
https://drive.google.com/drive/folders/1WVejv69KuLaoQ3zhe5ld14_TkpRiet5u?usp=sharing

Finally, pinout included here for quick access:



Appendix C - Encoder motor examples

Encoder motor examples:

Example 1 - takes keyboard input and programs the PWM mode of each motor to move it forward or back. Unlike the setMotorPwm() function used in earlier examples, this uses serTarPWM() which sets a target PWM value, and each call to .loop() increases the speed towards the target. New speed = (80% current speed + 20% target speed) offering smoother acceleration

```
#include <MeAuriga.h>
MeEncoderOnBoard Encoder_1(SLOT1);
MeEncoderOnBoard Encoder_2(SLOT2);
void setup()
{
 Serial.begin(115200);

  //Set PWM 8KHz - black magic, don't question it!
  TCCR1A = _BV(WGM10);
  TCCR1B = _BV(CS11) | _BV(WGM12);
  TCCR2A = _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS21);
}
void loop()
{
  if(Serial.available())
  {
       char a = Serial.read();
       switch(a)
       {
       case '0':
       Encoder_1.setTarPWM(0);
       Encoder_2.setTarPWM(0);
       break;
       case '1':
       Encoder_1.setTarPWM(100); // Back slow (40% speed)
       Encoder_2.setTarPWM(-100);
       break;
       case '2':
       Encoder_1.setTarPWM(200); // Back medium (80% speed)
       Encoder_2.setTarPWM(-200);
       break;
       case '3':
       Encoder_1.setTarPWM(255); // Back fast (full speed)
       Encoder_2.setTarPWM(-255);
       break;
       case '4':
       Encoder_1.setTarPWM(-100); // Forward slow
       Encoder_2.setTarPWM(100);
       break;
       case '5':
       Encoder_1.setTarPWM(-200); // Forward medium
       Encoder_2.setTarPWM(200);
       break;
       case '6':
       Encoder_1.setTarPWM(-255); // Forward fast
       Encoder_2.setTarPWM(255);
       break;
       case '7':
       Encoder_1.setTarPWM(100); // Left slow
       Encoder_2.setTarPWM(100);
       break;
       case '8':
       Encoder_1.setTarPWM(-100); // Right slow
       Encoder_2.setTarPWM(-100);
       break;
       default:
       break;
       }
  }
  Encoder_1.loop();
  Encoder_2.loop();
  Serial.print("Speed 1:");
  Serial.print(Encoder_1.getCurrentSpeed());
  Serial.print(" ,Speed 2:");
  Serial.println(Encoder_2.getCurrentSpeed());
}
```

Example 2 - takes keyboard input and programs each motor to move it forward or back. Unlike the setTarPwm() function used above, this code sets the speed using a PID algorithm to get to intended speed using feedback loops. Appendix D has some useful information on PID control.

```
#include <MeAuriga.h>
MeEncoderOnBoard Encoder_1(SLOT1);
MeEncoderOnBoard Encoder_2(SLOT2);
void setup()
{
  Serial.begin(115200);

  //Set PWM 8KHz
  TCCR1A = _BV(WGM10);
  TCCR1B = _BV(CS11) | _BV(WGM12);
  TCCR2A = _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS21);
  Encoder_1.setPulse(9);
  Encoder_2.setPulse(9);
  Encoder_1.setRatio(39.267);
  Encoder_2.setRatio(39.267);
  Encoder_1.setPosPid(0.18,0,0);
  Encoder_2.setPosPid(0.18,0,0);
  Encoder_1.setSpeedPid(0.18,0,0);
  Encoder_2.setSpeedPid(0.18,0,0);
}
void loop()
{
  if(Serial.available())
  {
        char a = Serial.read();
        switch(a)
        {
        case '0':
        Encoder_1.runSpeed(0); // Stop
        Encoder_2.runSpeed(0);
        break;
        case '1':
        Encoder_1.runSpeed(100); // Back slow (100rpm)
        Encoder_2.runSpeed(-100);
        break;
        case '2':
        Encoder_1.runSpeed(200); // Back medium (200rpm)
        Encoder_2.runSpeed(-200);
        break;
        case '3':
        Encoder_1.runSpeed(255); // Back fast (255rpm)
        Encoder_2.runSpeed(-255);
        break;
        case '4':
        Encoder_1.runSpeed(-100); // Forward slow
        Encoder_2.runSpeed(100);
        break;
        case '5':
        Encoder_1.runSpeed(-200); // Forward medium
        Encoder_2.runSpeed(200);
        break;
        case '6':
        Encoder_1.runSpeed(-255); // Forward fast
        Encoder_2.runSpeed(255);
        break;
        default:
        break;
        }
  }
  Encoder_1.loop();
  Encoder_2.loop();
  Serial.print("Speed 1:");
  Serial.print(Encoder_1.getCurrentSpeed());
  Serial.print(" ,Speed 2:");
  Serial.println(Encoder_2.getCurrentSpeed());
}
```

Example 4 takes keyboard input and programs the PID mode to move to a specific position

The calculations assume one rotation = 360 ‘ticks’

```
#include <MeAuriga.h>
MeEncoderOnBoard Encoder_1(SLOT1);
MeEncoderOnBoard Encoder_2(SLOT2);
void isr_process_encoder1(void) // count the ticks - i.e. how far the motor has moved
{
  if(digitalRead(Encoder_1.getPortB()) == 0)
  {
        Encoder_1.pulsePosMinus();
  }
  else
  {
        Encoder_1.pulsePosPlus();;
  }
}
void isr_process_encoder2(void) // count the ticks - i.e. how far the motor has moved
{
  if(digitalRead(Encoder_2.getPortB()) == 0)
  {
        Encoder_2.pulsePosMinus();
  }
  else
  {
        Encoder_2.pulsePosPlus();
  }
}
void setup()
{
// these interrupts are necesssary to
// enable the motor to know where it is
  attachInterrupt(Encoder_1.getIntNum(), isr_process_encoder1, RISING);
  attachInterrupt(Encoder_2.getIntNum(), isr_process_encoder2, RISING);
  Serial.begin(115200);

  //Set PWM 8KHz
  TCCR1A = _BV(WGM10);
  TCCR1B = _BV(CS11) | _BV(WGM12);
  TCCR2A = _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS21);
// No clue how these work!? Any help would be gratefully received
  Encoder_1.setPulse(9);
  Encoder_2.setPulse(9);
  Encoder_1.setRatio(39.267);
  Encoder_2.setRatio(39.267);
  Encoder_1.setPosPid(1.8,0,1.2);
  Encoder_2.setPosPid(1.8,0,1.2);
  Encoder_1.setSpeedPid(0.18,0,0);
  Encoder_2.setSpeedPid(0.18,0,0);
}
void loop()
{
  if(Serial.available())
  {
       char a = Serial.read();
       switch(a)
       {
       // It appears that each rotation is 360 ticks, and so 360 is 3
       // full rotations of the wheel.
       // In these examples I'm using the 4cm dia wheel (12.6cm circumference)
       case '0':
       Encoder_1.moveTo(0,50); // Move to 0cm
       Encoder_2.moveTo(0,50);
       break;
       case '1':
       Encoder_1.moveTo(360); // Move to -12.6cm (1 rotation)
       Encoder_2.moveTo(-360);
       break;
       case '2':
       Encoder_1.moveTo(1800); // move to -62.8cm (5 rotations)
       Encoder_2.moveTo(-1800);
       break;
       case '3':
       Encoder_1.moveTo(3600); // move to -1.26m (10 rotations)
       Encoder_2.moveTo(-3600);
       break;
       case '4':
       Encoder_1.moveTo(-360); // move to +12.6cm
       Encoder_2.moveTo(360);
       break;
       case '5':
       Encoder_1.moveTo(-1800); // move to +62.8cm
       Encoder_2.moveTo(1800);
       break;
       case '6':
       Encoder_1.moveTo(-3600); // Move to -1.26m
       Encoder_2.moveTo(3600);
       break;
       case '7':
       Encoder_1.moveTo(360);  // Turn right 45 degrees?
       Encoder_2.moveTo(360);  // My maths says this should be closer to 99 degrees
       break;
       case '8':
       Encoder_1.moveTo(720);  // Turn right 135 degrees
       Encoder_2.moveTo(720);  // My maths says this should be closer to 199 degrees
       break;
       default:
       break;
       }
  }
  Encoder_1.loop();
  Encoder_2.loop();
  Serial.print("Speed 1:");
  Serial.print(Encoder_1.getCurrentSpeed());
  Serial.print(" ,Speed 2:");
  Serial.print(" ,CurPos 1:");
  Serial.print(Encoder_1.getCurPos()); // Prints the number of ticks turned
  Serial.print(" ,CurPos 2:");
  Serial.println(Encoder_2.getCurPos());
}
```

Anyone wondering about my calculations for rotation angle above, there is 14.5cm between the two tracks on the ranger tank-track build, giving a potential turning circle of 45.6cm.
Assuming the wheels are travelling around a circle of that diameter when turning in opposite directions, the angle turned should be 360 * (distance travelled by wheels / 45.6). (arc angle = 360 * arc length / circumference)

Some will be lost through friction perhaps and the fact the drive wheels are at the front of the tracks, not the middle, but I didn't expect quite so much! Not much accuracy then when using tracked vehicle, so best to use the Gyro functions (see earlier in the document) - these are more accurate, but not perfect due to gyroscope drift.

Appendix D - More information on PID control

The Encoder motor libraries use PID control. Whilst I'm still not entirely sure HOW the functions work, the principles of PID control are fairly straightforward.

In order to programme a system to reach a specific value (e.g. position of a motor, temperature of an oven, height of a quadcopter etc, you can imagine that you can easily measure the current status using sensors, you know what the desired result is, and you can apply an input to make the system change.

What PID control does is measure the current position and the error (i.e. difference between the current position and goal), then applies a calculation using three parameters (proportional, integral and derivative) to adjust the input to move the reading towards the desired goal. Each of the three terms can be tuned to make the system react appropriately.

The working principle behind a PID controller is that the proportional, integral and derivative terms must be individually adjusted or "tuned." Based on the difference between these values a correction factor is calculated and applied to the input. For example, if an oven is cooler than required, the heat will be increased. Here are the three steps:

1. Proportional tuning involves correcting a target proportional to the difference. Thus, the target value is never achieved because as the difference approaches zero, so too does the applied correction.
2. Integral tuning attempts to remedy this by effectively cumulating the error result from the "P" action to increase the correction factor. For example, if the oven remained below temperature, “I” would act to increase the head delivered. However, rather than stop heating when the target is reached, "I" attempts to drive the cumulative error to zero, resulting in an overshoot.
3. Derivative tuning attempts to minimize this overshoot by slowing the correction factor applied as the target is approached.

These factors are added together to produce a correction factor which is then applied to the input.

More detailed information here:
https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y
And an excellent example of PID tuning here: https://www.youtube.com/watch?v=fusr9eTceEo
.