

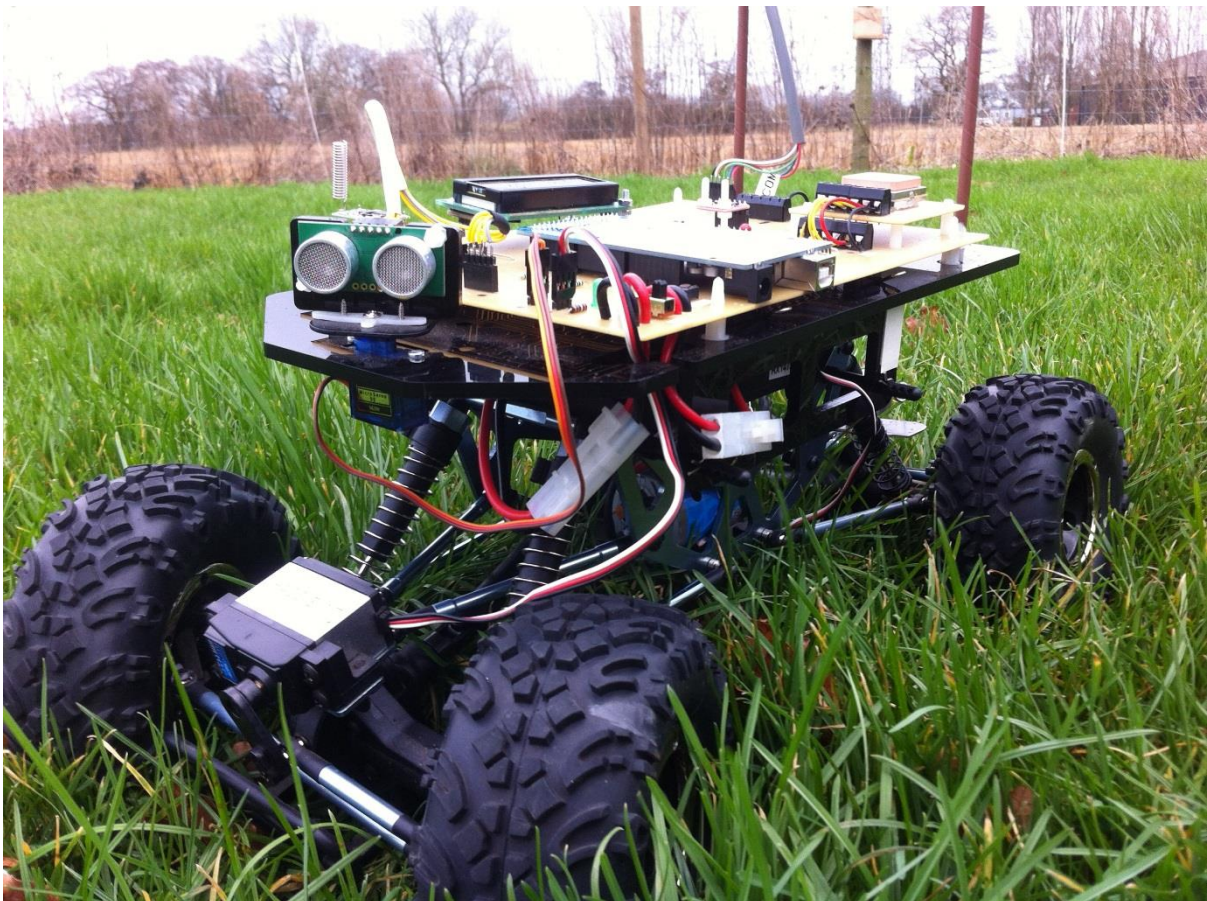


Welcome to Programming a GPS enabled robot using the Arduino.

Presented by Sam Wane, Senior Lecturer

swane@harper-adams.ac.uk

Harper Adams University



Contents

The robot hardware	1
Installing and setting up the IDE	2
The Atlas library	2
Connecting the robot to the computer	4
First program-Flashing LEDs.....	6
More basic programs	7
Steer	7
Drive	7
Reading sensors	8
Compass connection	9
Temperature	9
Wireless communication	10
HC-11.....	10
RC Control	10
General I/O.....	11
Calibration.....	12
Using Calibrate_steer.....	12
Using Calibrate_compass	12
Further examples	13
Lessons	13
Functions.....	14
General.....	14
initialise();.....	14
approx(a,b,c)	14
get_timer(int timer_no);.....	14
set_timer(int timer_no, unsigned long timerincrement);	14
timer_elapsed(int timer_no);	14
bearing_to_point_deg(current latitude,current longitude, goal latitude, goal longitude);	14
double distance_to_point_deg(current latitude,current longitude, goal latitude, goal longitude);	14
closest_bearing_difference(current_bearing,goal_bearing);.....	14
delay(millisecs);.....	14
if (statement) action ; else other action;	15
Filter	15
void set_veh_length(double v)	15

void set_veh_width(double v);	15
double Output_PID(double Kp, double Ki, double Kd, double error, double maxerror).....	15
read_offset_bearing()	15
write_offset_bearing(double ob).....	15
Inputs (sensors).....	15
battery()	15
trailer_angle();	15
set_trailer_centre();.....	15
RC_steer();	15
getE();.....	16
getN();	16
getLat();.....	16
getLon();.....	16
float getAlt();.....	16
get_Pitch();.....	16
get_Roll();.....	16
float get_Yaw();.....	16
double rightLDR()	16
double leftLDR()	16
read_distance();	17
Serial2.read	17
read_compass();.....	17
float temp();	17
int read_button(int button);	17
edge_rising(1..3)	17
float right_light();.....	17
float left_light();	17
unsigned long read_wheel();	17
void zero_wheel();	18
double RC_speed()	18
double RC_steer()	18
Outputs (actuators).....	18
steer_radius(double inv_R);.....	18
steer(double steer_angle);.....	18
rear_steer(double steer_angle);.....	18
steerslow(double steer_angle);	18

rear_steerslow(double steer_angle);	18
drive(float drive_speed);.....	19
drive(float drive_speed, byte accel);	19
turret(angle);.....	19
lcd.clear();	19
lcd.print("Text");	19
lcd.print(var, 6);	19
lcd.setCursor(column, row);	19
display(text) display(top line text, bottom line text)	20
Serial.print.....	20
Serial2.print.....	20
led_out(led,state);	20
void buzzer_on();	20
void buzzer_off();	20
drive(speed metres per second);	20
byte readDIP();	20
double gen_pot();	20

Make it move, then make it perfect – **David Tseng, CAVEDU Education**

The robot hardware

The robot car is built from a standard radio control all-terrain vehicle known as a Maverick. It is a four wheel drive, independent suspension vehicle allowing it to cope with rough terrain.

- **Examine the chassis. Discuss why this vehicle is so well suited to rough terrain.**

The chassis has two motors, a drive motor and a steer motor. These are connected to the microcontroller board through the 'servo motor interface' plugs. The drive motor also has its own separate switch (for safety-we don't want it zooming off the table!) on the chassis close to the interface plug.

- **Locate the motor interfaces, ensure they are plugged in correctly. Locate the drive motor switch and ensure it is switched off!**

Note that the motors are driven from the onboard battery and will only work when the switch (bottom left) is on. Leave it off for now.

There is a variety of sensors on the microcontroller board, these are all connected to the microcontroller.

- **Identify the items below on your vehicle. Discuss how you think the information from the sensors is passed to the microcontroller.**

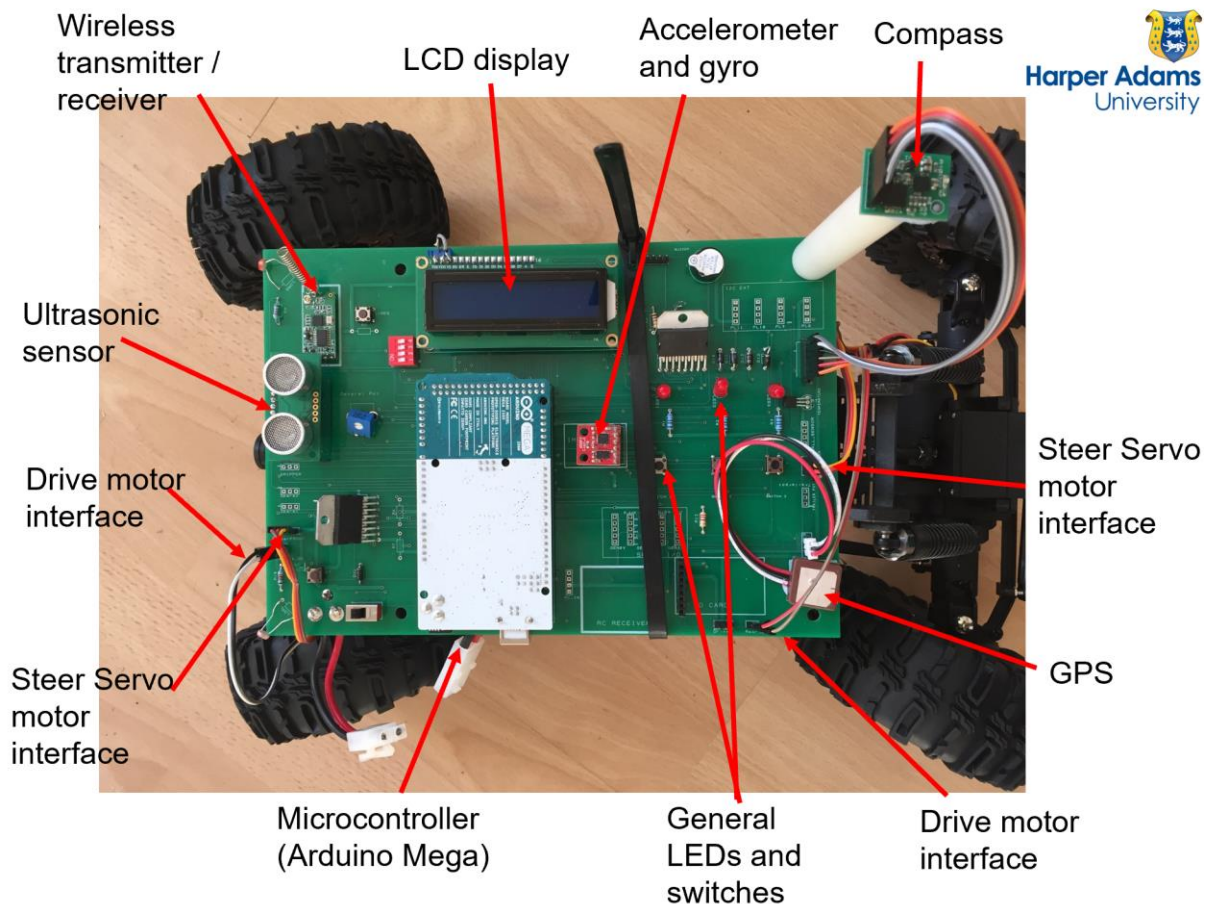
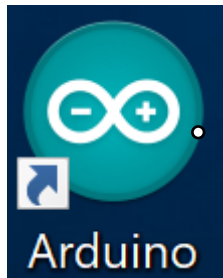


Figure 1 The Robot Hardware

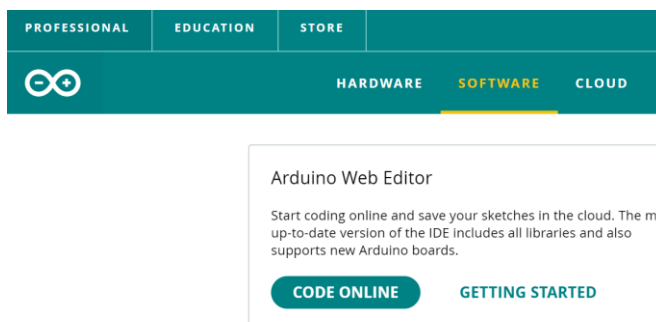
Installing and setting up the IDE

IDE stands for 'Integrated Development Environment', and this is the place where you write programs and download to the Arduino microcontroller from the PC.



This can be downloaded from www.arduino.cc if not installed already.

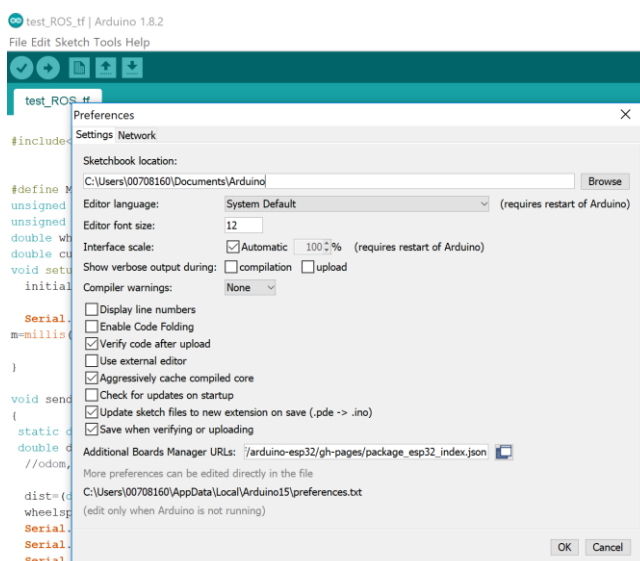
Install the Arduino IDE: <https://www.arduino.cc/en/software>



Downloads

The Atlas library

The library is to be installed in the Libraries folder of the Arduino IDE. To find this location, Open the Arduino IDE, select: File→Preferences. Copy the files into the Sketchbook location / libraries folder. E.G. in the example below this would be: C:\Users\00708160\Documents\Arduino\libraries



Install the library from the USB stick or from Git Hub location: <https://github.com/swane/atlas>

08160 > Documents > Arduino > libraries > ATLAS	
Name	Date
examples	15/11
ATLAS.cpp	24/11
ATLAS.h	24/11
deg2utm.cpp	19/03
deg2utm.h	19/03

The library is included in the program and allows easy access to the hardware of the robot.

Now, use the Library manager to search and install the following libraries. To do this, from the menu select: Sketch→Include Library→Manage Libraries. Then enter the following one by one into the search box and click the 'Install' button for each one:

- NeoGPS - <https://github.com/SlashDevin/NeoGPS>
 -
 - FreesixIMU - <https://github.com/simondlevy/FreeSixIMU2>
 - VarSpeedServo
 - SD (installed already with the IDE)
 - SPI (installed already with the IDE)
 - LiquidCrystal (installed already with the IDE)
 - SoftwareSerial (installed already with the IDE)
1. Close and restart the IDE
 2. You can check if the library has installed from the IDE by selecting: Files→Examples→ ATLAS.
You should see the example programs.

Connecting the robot to the computer

Connect the microcontroller in the robot to the PC using a suitable USB cable.

Now we will set the IDE to the correct COM port and board.

1. Under the menu **Tools**→**Board**, select 'Arduino / Genuino Mega or Mega 2560' (figure 2)
2. Under the menu **Tools**→**Port**, select the COM port with the 'Arduino / Genuino Mega', (this will be a port number higher than COM1), figure 3.

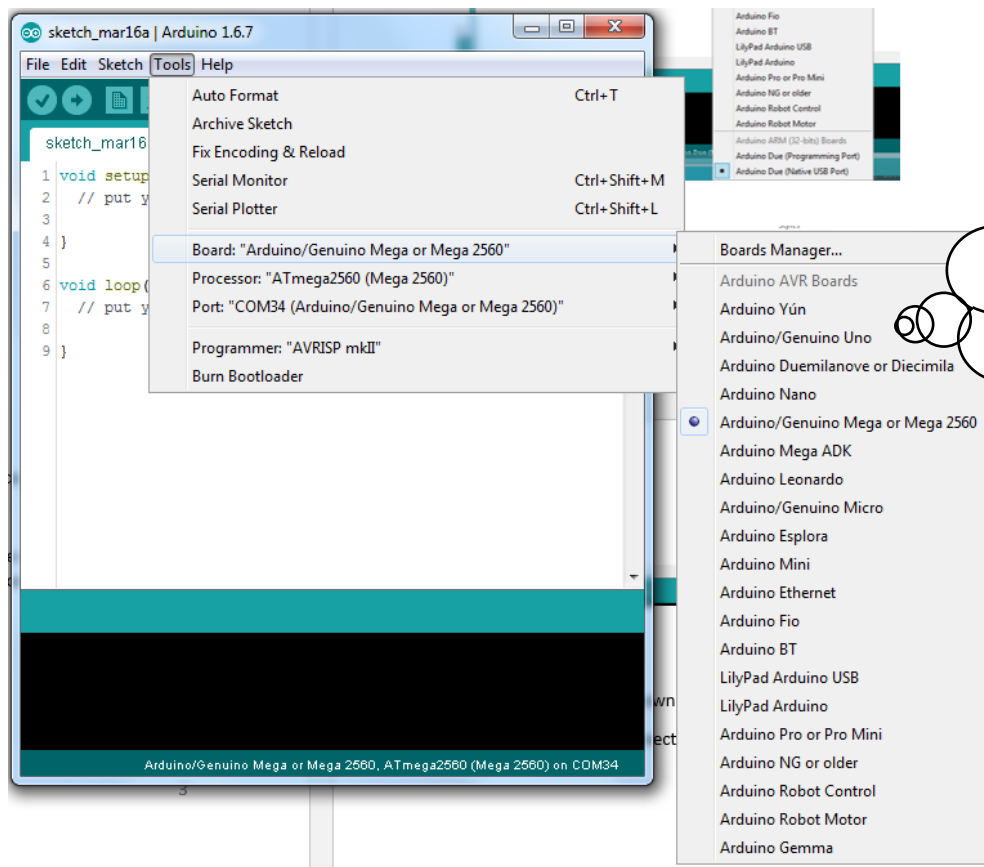


Figure 2 Selecting the Arduino type

Open the 'Arduino' on the PC.

empty program as follows:

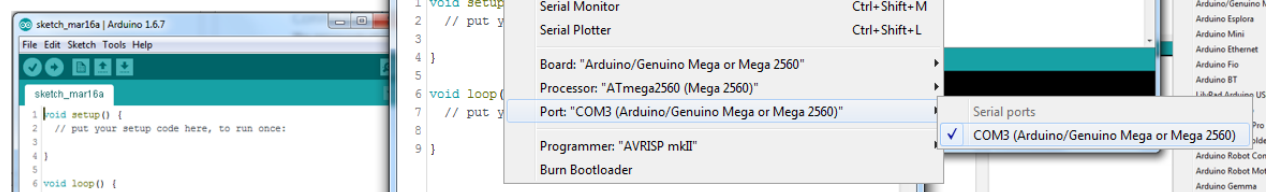
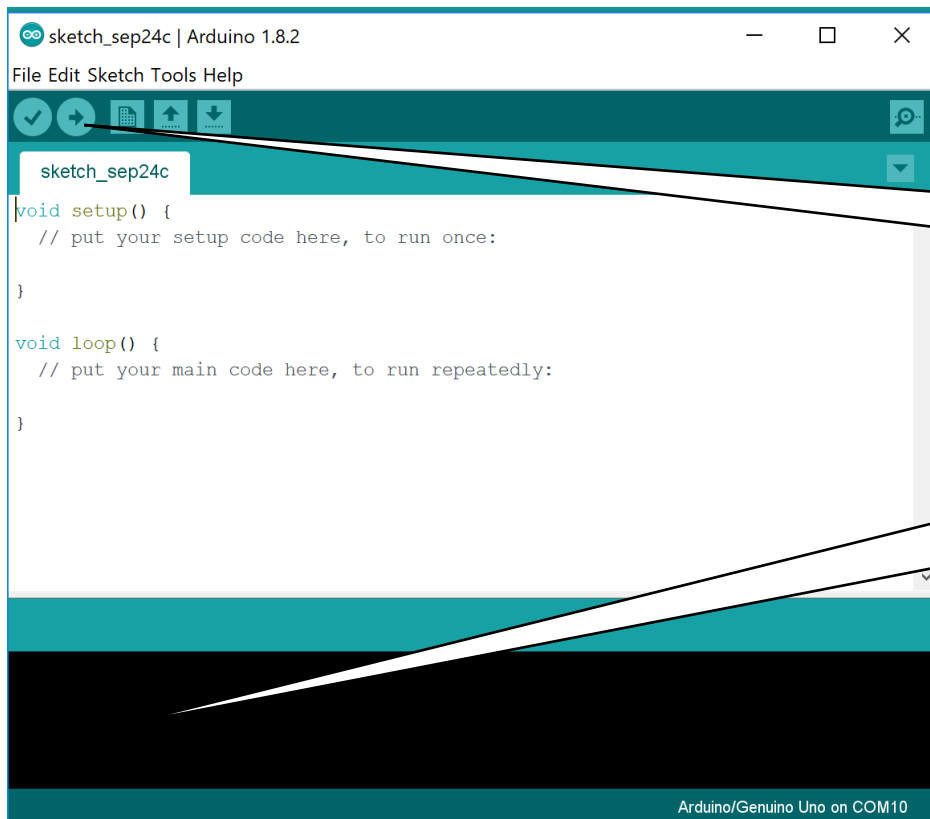


Figure 3 Selecting the Com port

3. Select **File**→**New**.

An empty program with 'void setup()' and 'void loop()' will be shown as in fig 3. This is where you will write your program.



The **Upload Button**.
Button to upload program
to micro controller

This also checks the
code and any errors
appear here. Fix the
errors before uploading.

Figure 4 Empty program and how to upload

You can now write your program in here. Press the right-arrow icon to upload and run once you have entered your program.

The 'setup()' section is used to initialise the hardware of the robot (it runs once and sets things up).

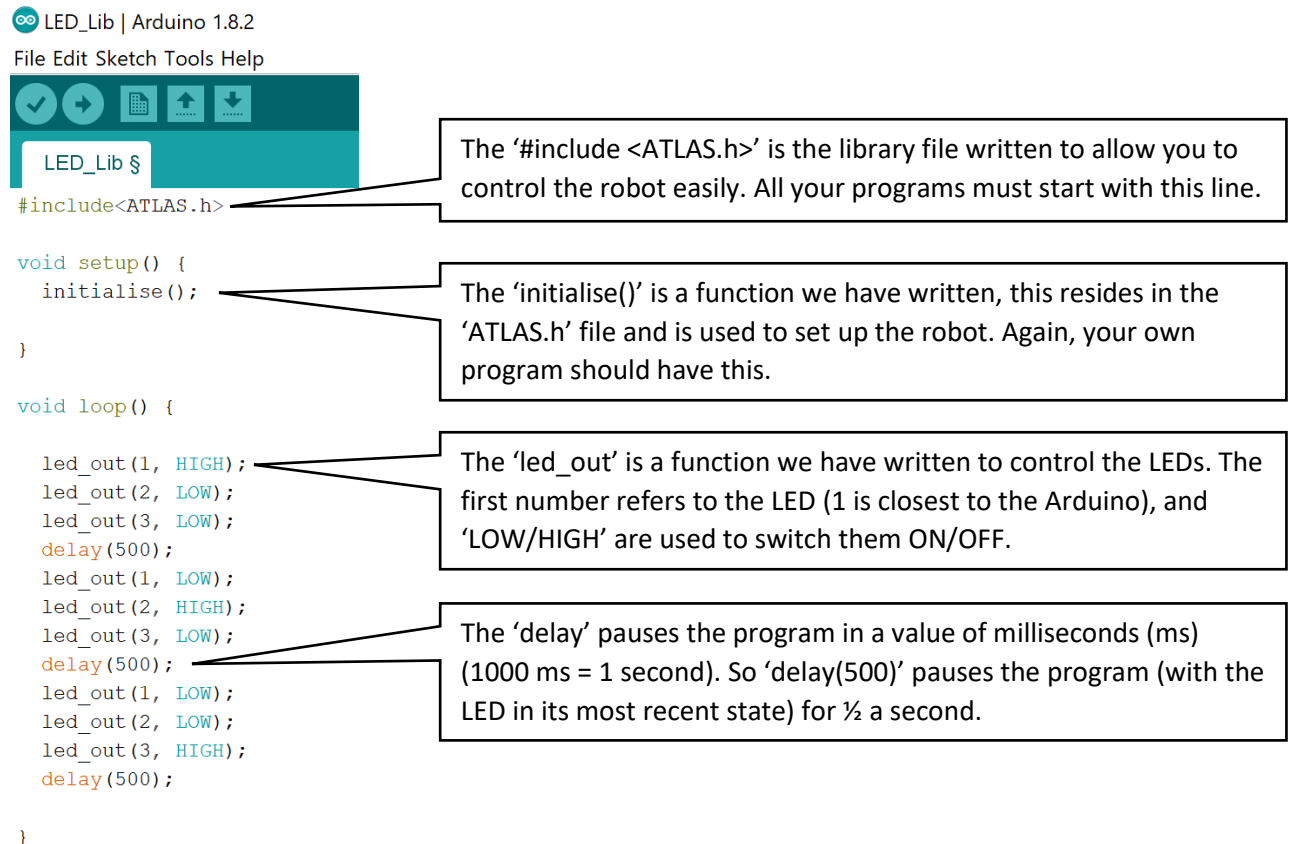
The 'loop()' runs continuously and is where your program runs in a loop.

Now, time to put in your first program, so let's start...

First program-Flashing LEDs

Select 'File→Examples→Atlas→LED'.

Your program should look like this:



LED_Lib | Arduino 1.8.2
File Edit Sketch Tools Help

LED_Lib \$

```
#include<ATLAS.h>

void setup() {
  initialise();
}

void loop() {
  led_out(1, HIGH);
  led_out(2, LOW);
  led_out(3, LOW);
  delay(500);
  led_out(1, LOW);
  led_out(2, HIGH);
  led_out(3, LOW);
  delay(500);
  led_out(1, LOW);
  led_out(2, LOW);
  led_out(3, HIGH);
  delay(500);
}
```

The '#include <ATLAS.h>' is the library file written to allow you to control the robot easily. All your programs must start with this line.

The 'initialise()' is a function we have written, this resides in the 'ATLAS.h' file and is used to set up the robot. Again, your own program should have this.

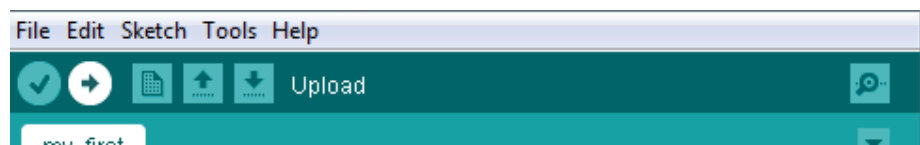
The 'led_out' is a function we have written to control the LEDs. The first number refers to the LED (1 is closest to the Arduino), and 'LOW/HIGH' are used to switch them ON/OFF.

The 'delay' pauses the program in a value of milliseconds (ms) (1000 ms = 1 second). So 'delay(500)' pauses the program (with the LED in its most recent state) for ½ a second.

Figure 5 Flash LEDs Program

You will find a list of the functions in the Appendix.

- Now, upload the program to the Arduino by using the 'Upload' button as shown below:



The robot should flash the LEDs as in the program.

In order to understand the program, let's make some changes, but first, save it under a new name:

- Now, use 'save as' to save the program under 'Documents' and give it the name 'my first'.
 - Try changing the program, for example, you can change the delay time, or delete which LEDs are accessed.
- Note that the end of each line has to have a semi-colon (;), this is a C programming standard.

More basic programs

There are many commands which are defined in the Atlas library, a full list is in the Appendix. This section is here to introduce and to whet your appetite for what the robot can do.

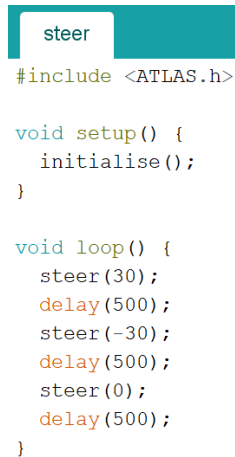
Steer

The robot can steer to an angle, this is called with:

`steer(angle);`

where *angle* is a value between -40° to 40° . Positive angles are turning to the right, 0° is straight ahead.

Try this program, either from the 'Examples→Atlas→Steer', or you can type it in:



```
steer

#include <ATLAS.h>

void setup() {
  initialise();
}

void loop() {
  steer(30);
  delay(500);
  steer(-30);
  delay(500);
  steer(0);
  delay(500);
}
```

Upload the program as before and ensure the main robot power is ON.

Drive

The robot will drive the motors:

`drive(speed);`

where *speed* is in m/s and is approximate. Positive values drive forwards, negative values are backwards, zero is stop. The maximum value is 1. There is a minimum value that is needed to get the robot moving due to friction, and this is approximately 0.5.

The example below is under 'Examples→Atlas→Drive':

```
#include <ATLAS.h>
```

```
void setup() {  
  initialise();  
}
```

```
void loop() {  
  drive(1);  
  delay(2000);  
  drive(0.7);  
  delay(1000);  
  drive(0);  
  delay(1000);  
  drive(-0.7);  
  delay(1000);  
  drive(-1);  
  delay(1000);  
  drive(0);  
  delay(1000);  
}
```

Reading sensors

The ultrasonic distance sensor is accessed through:

`read_distance();`

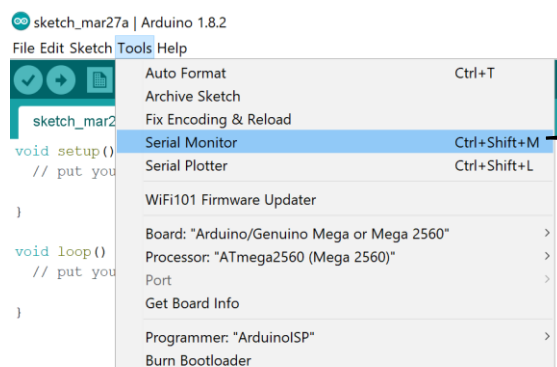
and returns the distance to the robot in cm.

This can be shown on the LCD screen and the example in 'Examples→Atlas→distance' shows this.

This can also be shown on the PC if it is still connected via the USB, and the following demonstrates how:

Upload the 'distance' program on the Arduino.

Open up the Serial Monitor:



This opens up a new window where you can see data coming from the Control Rig/Arduino.

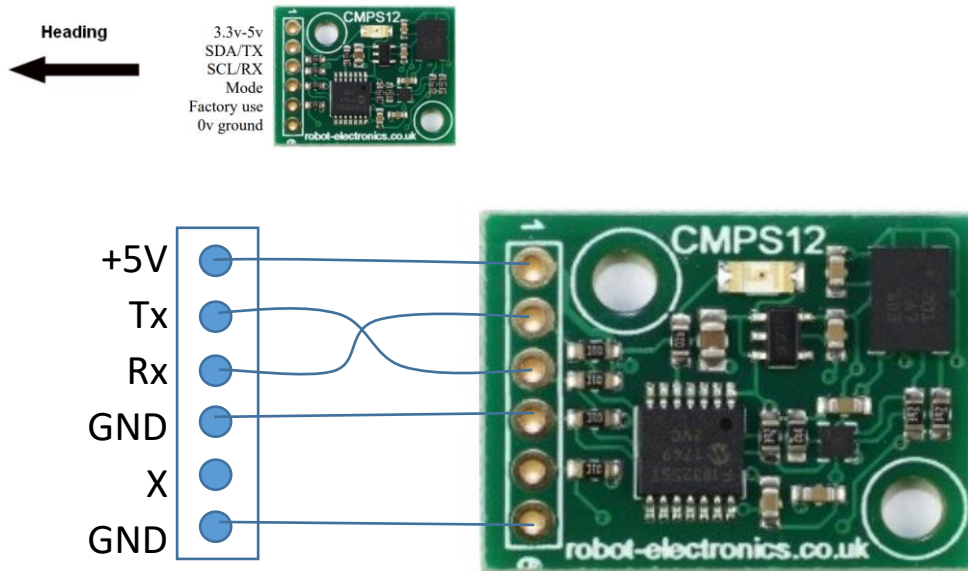
You should see the values on the PC.

You can close the Serial Monitor and open Serial Plotter if you'd like a graphical display.

Compass connection

The compass is read in 'Serial' mode. The device is a CMP12 available at <https://www.robot-electronics.co.uk>

The Tx and Rx are swapped on the PCB. The compass is to be mounted away from motors and any other magnetic interference.



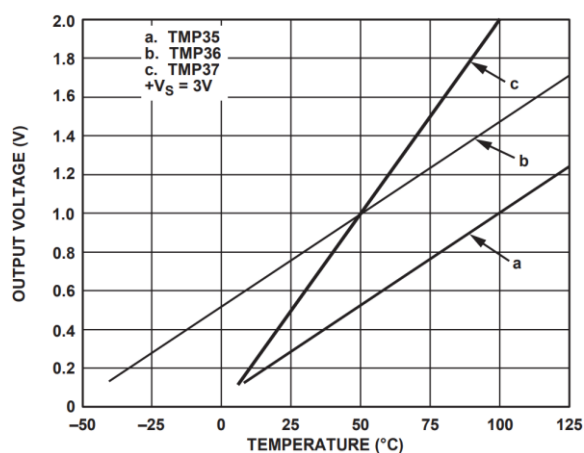
Connections as shown

The command 'read_compass()' returns the heading in 0-360.

Temperature

The temperature sensor is type TMP36.

The command 'temp()' returns a floating point value of the temperature. This is converted from an analogue voltage as in the datasheet diagram below:



Wireless communication

HC-11

The wireless device uses a HC-11 wireless RS232 transceiver connected to Serial 2 at 9600 baud.

It is set up using AT commands and there is an example program to do this. It can also be set in the Atlas test program. When setting it up, the CON button to the right must be held during this procedure.

Data is send by using `Serial.write(data)` and read using `Serial.read()`.

e.g.

```
Serial2.write("1");  
and:  
if (Serial2.available()) {  
    c=Serial2.read();  
    Serial2.flush();  
    lcd.setCursor(0, 1);  
    if (c=='1') {led_out(2, HIGH);lcd.print("ON ");delay(200);}  
    {lcd.setCursor(0, 1);led_out(2, LOW);lcd.print("OFF");}
```

More examples are in:

HC-11_Setup

HC-11_Send_AB

Note, the HC-11 can be prone to hang. To reset, remove all power, then connect power whilst holding the CON button. Then the AT commands can be sent to set it up.

RC Control

The slot for the RC receiver can allow two signals from a remote control receiver to send data to Atlas. RC receiver pulses are received from a paired transmitter. The pulses vary in length from 1-2ms depending on the control stick position.

These are read using the commands: `RC_steer -data` on channel 1, and `RC_speed-data` on channel 2. These functions return a double value in the range of -1 to +1, with 0 being the resting position of the control sticks.

The receiver is connected to the +7.2V, GND, channel1 and channel2. It operates even when only USB power is applied to the Arduino.

For speed on channel 2, +1=fastest, 0=stop, -1=reverse

For steer on channel 1, +1=full right, 0=straight, -1=full left

It has a 50 ms timeout, so if communication is lost, it will not halt the program. A 2.0 is returned if communication is lost.

General I/O

Analogue ports are referred to as *GENAportnumber* and range from GENA1 to GENA5

e.g. `a=analogRead(GENA1);`

Digital ports are referred to as *GENDportnumber* and range from GEND1 to GEND5.

e.g. `a=digitalRead(GEND1);`

`digitalWrite(GEND1,HIGH);`

They are set as inputs with internal pull up resistors, but this can be changed with:

`pinMode(GEND1,OUTPUT);`

`pinMode(GEND1,INPUT);`

`pinMode(GEND1,INPUT_PULLUP);`

Pin name	Actual pin
GENA1	A11
GENA2	A12
GENA3	A13
GENA4	A14
GENA5	A15
GEND1	30
GEND2	33
GEND3	32
GEND4	23
GEND5	3

Calibration

The steering angle zero offset along with other values can be fine-tuned for each vehicle once built. These offset values are stored in the EEPROM of the Arduino.

The EEPROM is addressed from 0 onwards and can store bytes only. The first value (at address 0) is set to 170 (10101010 in binary) if calibration has occurred, otherwise, standard pre-programmed values (in the ATLAS.cpp) file are used. The following table states the EEPROM values:

0	Notification if calibration has occurred
1	Front zero angle offset
2	Rear zero angle offset
3	Front max left
4	Front max right
5	Rear max left
6	Rear max right
7	Compass offset

Programs to accomplish calibration:

Calibrate_steer	Allows the front and rear steer servos to have a zero and min / max angles
Calibrate_compass	Allows the compass offset to be calibrated

Using Calibrate_steer

The program is found under Examples/Atlas/Calibrate_steer

This allows the min / max steering angles and the zero (straight ahead) offsets to be defined.

The DIP switches are used to define the function of the program. These are all raised when OFF.

DIP Switch operation

OFF -The Button 2 drives the robot for two seconds

TOP DIP ON-Steering centre: Button 1 increases, Button 3 decreases, Button 2 stores centre offset

TOP TWO DIP ON:Sets the max steer angle (steer to the right), , Button 3 decreases, Button 2 stores

TOP THREE DIP ON:Sets the max steer angle (steer to the right), , Button 3 decreases, Button 2 stores

Using Calibrate_compass

The program is found under Examples/Atlas/Calibrate_compass

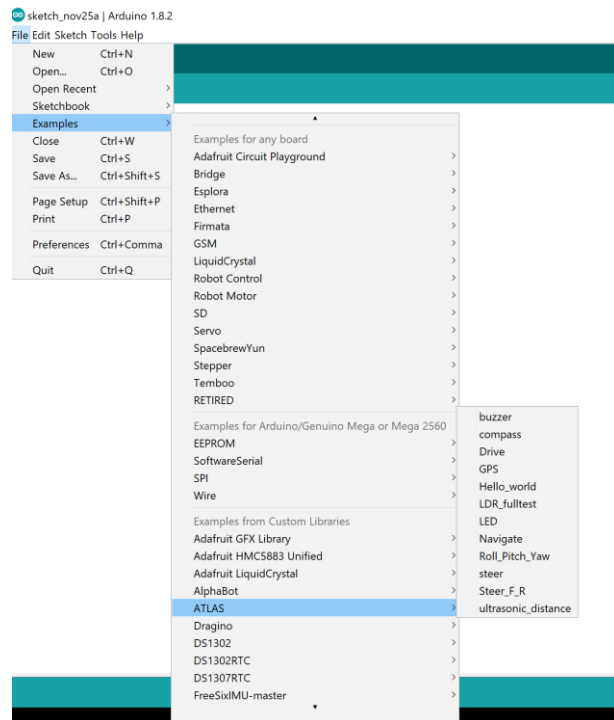
The LCD shows the current bearing. Button 1 increases the bearing, Button 3 decreases the bearing, Button 2 resets the offset to zero.

After each button press, the value is stored into EEPROM for access next time it is switched on. The calibration will be fixed and work for any future program.

Further examples

More example programs are available under: Files→Examples→ATLAS→

See below:



Lessons

Task 1. Robots in Agriculture

Task 2. Robot Chassis

Task 3. Buttons and Logic

Task 4. Variables and Display

Task 5. Distance and Tilt Sensors

Task 5b. Filtering

Task 6. Compass and Steering

Task 7. Drive

Task 8. GPS

Task 9. Bearing and Navigation

Task 10.

Functions

General

`initialise();`

Input: Nothing

Returns: Nothing

Function: Sets up the robot, called in the setup function. This must be called prior to using any other functions.

`approx(a,b,c)`

returns true if a is within the range of b with a threshold of c

`get_timer(int timer_no);`

Input: timer number 0-9

Returns: long milliseconds since the timer was reset

Function:

`set_timer(int timer_no, unsigned long timerincrement);`

Input: timer number 0-9, value to count down in ms

Returns: nothing

Function: Sets up a countdown timer. Starts a timer counting in ms. There are ten timers which can be accessed. The timer elapsed is checked with `timer_elapsed(timer number 0-9)`

e.g.

```
set_timer(0,500);
```

```
if (timer_elapsed(0)) Serial3.print("Timer 0 triggered after 0.5 seconds");
```

`timer_elapsed(int timer_no);`

Input: timer number 0-9

Returns: true if the timer has passed

Function: Returns a logical TRUE when the timer as set in 'set_timer' is triggered. This does not automatically reset the timer however. A 'set_timer' command must be called to set this again.

`bearing_to_point_deg(current latitude,current longitude, goal latitude, goal longitude);`

Returns the bearing to a point, in degrees.

`double distance_to_point_deg(current latitude,current longitude, goal latitude, goal longitude);`

Returns the distance to a point in degrees.

`closest_bearing_difference(current_bearing,goal_bearing);`

Returns the difference between the current and goal bearing. Solves the problem of the 360 to 0 rollover.

e.g.

```
steer_angle=closest_bearing_difference(bearing,0);
```

`delay(millisecs);`

Pauses the program for the time stated in 'millisecs'

e.g. delay (500);

`if (statement) action ; else other action;`

Is used to perform an action is the statement is true.

Use a double equals sign (==) to test equality, or a greater than (>) or less than (<) to test if a value is bigger or smaller.

Filter

`double filter(double input, double f)`

Returns the low pass filtered signal, f is the cut-off frequency in Hz, input is the signal. The sample rate is automatically calculated.

`void set_veh_length(double v)`

`void set_veh_width(double v);`

Sets the vehicle length and width (in metres), the distance between the axles.

`double Output_PID(double Kp, double Ki, double Kd, double error, double maxerror)`

outputs the PID corrected value within the limits of maxerror. It stops integral windup if outside these boundaries.

`read_offset_bearing()`

Returns the offset bearing as stored in EEPROM

`write_offset_bearing(double ob)`

Sets the offset bearing, stored in EEPROM

Inputs (sensors)

`battery()`

Returns a value representing the battery voltage

`trailer_angle();`

Input: Nothing

Returns: double

Function: Returns the angle of the trailer attached to the potentiometer.

`set_trailer_centre();`

Input: Nothing

Returns: double

Function: Calibrates the trailer potentiometer to centre.

`RC_steer();`

Input: nothing

Returns: The steer angle command from the Radio transmitter in the range of +-1. + is right, - is left

`getE();`

Returns a double

Returns Easting as a type double.

`getN();`

Returns a double

Returns Nothing as a type double.

e.g.

```
E=getE();
```

```
N=getN();
```

`getLat();`

Returns a double

Returns latitude as a type double.

`getLon();`

Returns a double

Returns longitude as a type double.

e.g.

```
lt=getLat();
```

```
ln=getLon();
```

`float getAlt();`

Returns the altitude in m.

`get_Pitch();`

Returns int

Reads the pitch of the robot (tilt which occurs when climbing a hill).

e.g.

```
int pitch; //declare variables to hold the pitch value
pitch = get_Pitch(); //assign values by using the provided functions
```

`get_Roll();`

Returns int

Reads the roll of the robot (side to side movement)

e.g.

```
int roll;
roll = get_Roll();
```

`float get_Yaw();`

Returns the yaw of the vehicle.

`double rightLDR()`

Returns the brightness at the right LDR in percentage (0-dark, 100-brightest)

`double leftLDR()`

Returns the brightness at the left LDR in percentage (0-dark, 100-brightest)

`read_distance();`

Input: nothing

Returns: double: the distance sensed by the ultrasonic ranger in cm

Function: Reads distance.

`Serial2.read`

Reads data from the wireless transceiver

e.g.

`int i;`

```
while (Serial2.available()){  
    i=Serial2.read();  
    Serial.print(char(i));  
}
```

`read_compass();`

Input: nothing

Returns: Double of degrees

Function: Returns the bearing as measured by the magnetometer (compass) in the range 0-360.

`float temp();`

Input: nothing

Returns: temperature in degrees C

Function: Reads the temperature from the temperature sensor.

`int read_button(int button);`

Returns a '1' if the button 1,2,3 is pressed.

e.g. `if (read_button(1)==1) Serial.print("button on"); else Serial.print("Button off");`

`edge_rising(1..3)`

returns HIGH if a button has just been pressed,

e.g.

```
void loop() {  
    if (edge_rising(2))  
    {  
        n++;  
        clear_LCD();  
        Serial3.print(n);  
    }  
}
```

`float right_light();`

reads the right light sensor

`float left_light();`

reads the left light sensor

`unsigned long read_wheel();`

Reads the Hall sensor

`void zero_wheel();`

Resets the Hall wheel count

`double RC_speed()`

For speed on channel 2, +1=fastest, 0=stop, -1=reverse

It has a 50 ms timeout, so if communication is lost, it will not halt the program. A 2.0 is returned if communication is lost.

`double RC_steer()`

For steer on channel 1, +1=full right, 0=straight, -1=full left

It has a 50 ms timeout, so if communication is lost, it will not halt the program. A 2.0 is returned if communication is lost.

These routines are coded by hand to use the `micros()` timer, as the `'pulseIn'` function is unstable due to interrupts used elsewhere in the code.

Outputs (actuators)

`steer_radius(double inv_R);`

Input: double, inverse radius

Returns: nothing

Function: sets the front steer angle to steer to a radius where a positive value will steer LEFT.

All servos were found to suffer interference when not being addressed, this was cured by only attaching them when needed and then detached after a short delay as in:

```
frservo.attach(frontservo);  
frservo.write(steer_angle);  
delay(50);  
frservo.detach();
```

`steer(double steer_angle);`

Input: double steer angle

Returns: nothing

Function: sets the front steer angle in degrees with 0 – centre, and positive is steer right

`rear_steer(double steer_angle);`

Input: double steer angle

Returns: nothing

Function: sets the steer angle in degrees with 0 – centre, and positive is steer right

`steerslow(double steer_angle);`

Input: double steer angle

Returns: nothing

Function: sets the front steer angle moving slowly, in degrees with 0 – centre, and positive is steer right

`rear_steerslow(double steer_angle);`

Input: double steer angle

Returns: nothing

Function: sets the rear steer angle moving slowly, in degrees with 0 – centre, and positive is steer right

`drive(float drive_speed);`

Drives the motors. If speed = 0 then it stops, a positive value makes it go forwards, and a negative value backwards. The value is calibrated in metres per second.

The minimum and maximum values for 'speed' are -1 → +1 m/s.

e.g. `drive(0.5);` //Drive the vehicle at 0.5 m/s

`drive(float drive_speed, byte accel);`

If used, acceleration can be in the range +1 to +255. In practice the useful range is around 5 to 10.

Usage example

`drive(0.7,5);` will gently accelerate to 0.7 Metres/Sec. A subsequent instruction `drive(0,5);` will decelerate to a halt. The program continues to run during the acceleration/deceleration phase. `drive(0.7);` and `drive(0);` will start and stop abruptly as before.

`turret(angle);`

int angle

Moves the ultrasonic servo turret by 'angle'. Angle is from -80 to +80, with 0 pointing forwards and negative values are to its right.

`turret()` now has two extra optional parameters: speed, and whether to block program execution until the move is complete. This is very useful when sweeping the ultrasonic sensor in a series of steps, in a for loop for example.

Usage example

`turret(30,25,1);` will move to angle 30 at constant speed 25. The program will be blocked until it arrives. The third parameter is a Boolean. It can be entered as 1/0 or true/false (or omitted).

`turret(30,25);` will move to angle 30 at speed 25. The program continues to execute during the move. Speed can be set in the range +1 to +255, but the useful range is around 30 – 60.

`turret(30);` moves at maximum speed as before.

`lcd.clear();`

Input: nothing

Returns: nothing

Function: clears the LCD

`lcd.print("Text");`

Input: Text

Returns: nothing

Function: Sends text to the LCD display.

`lcd.print(var, 6);`

Input: variable value

Returns: nothing

Function: Sends a number held in 'var' to the LCD display. The number after the comma is how many decimal places to show.

`lcd.setCursor(column, row);`

Sets the cursor position to the values. These start at 0. So, 0,0 is the top left of the screen.

E.G to send the next text to the bottom row, first column, use `lcd.setCursor(0, 1);`

`display(text)` `display(top line text, bottom line text)`

Clears and displays text on the LCD. If an item is included after the comma, it is displayed on the bottom line. E.g.

```
display("Harper Adams Uni", "Atlas tests ");
```

Variables must first be converted into text using 'String(variable)' as in:

```
display("Bearing", String(bearing));
```

`Serial.print`

Sends data to the PC monitor to display or graph.

e.g.

```
Serial.print("Hello");
```

sends 'Hello' to the serial monitor.

```
Serial.println("Hello");
```

sends 'Hello' to the serial monitor and goes on to the next line down.

```
int a=20;
```

```
Serial.println(a);
```

Sends '20' to the serial monitor, or it can form part of a graph on the serial plotter.

`Serial2.print`

Sends data to the wireless transmitter.

e.g.

```
Serial2.print("Harper Adams Uni");
```

`led_out(led,state);`

int led

int state

Switches the LEDs on or off.

led can be 1, 2, or 3 (1 is closest to the Arduino).

state can be HIGH or LOW.

e.g. `led(1,HIGH);`

`void buzzer_on();`

`void buzzer_off();`

`drive(speed metres per second);`

double speed metres per second

`byte readDIP();`

Returns the DIP switch state as a 4 bit byte.

Switches are ON when raised, OFF when pressed in.

Lowest bit is at the top of the PCB.

`double gen_pot();`

Returns the double value from the PCB mounted potentiometer.

1023.00=fully left, 0.00=fully clockwise

