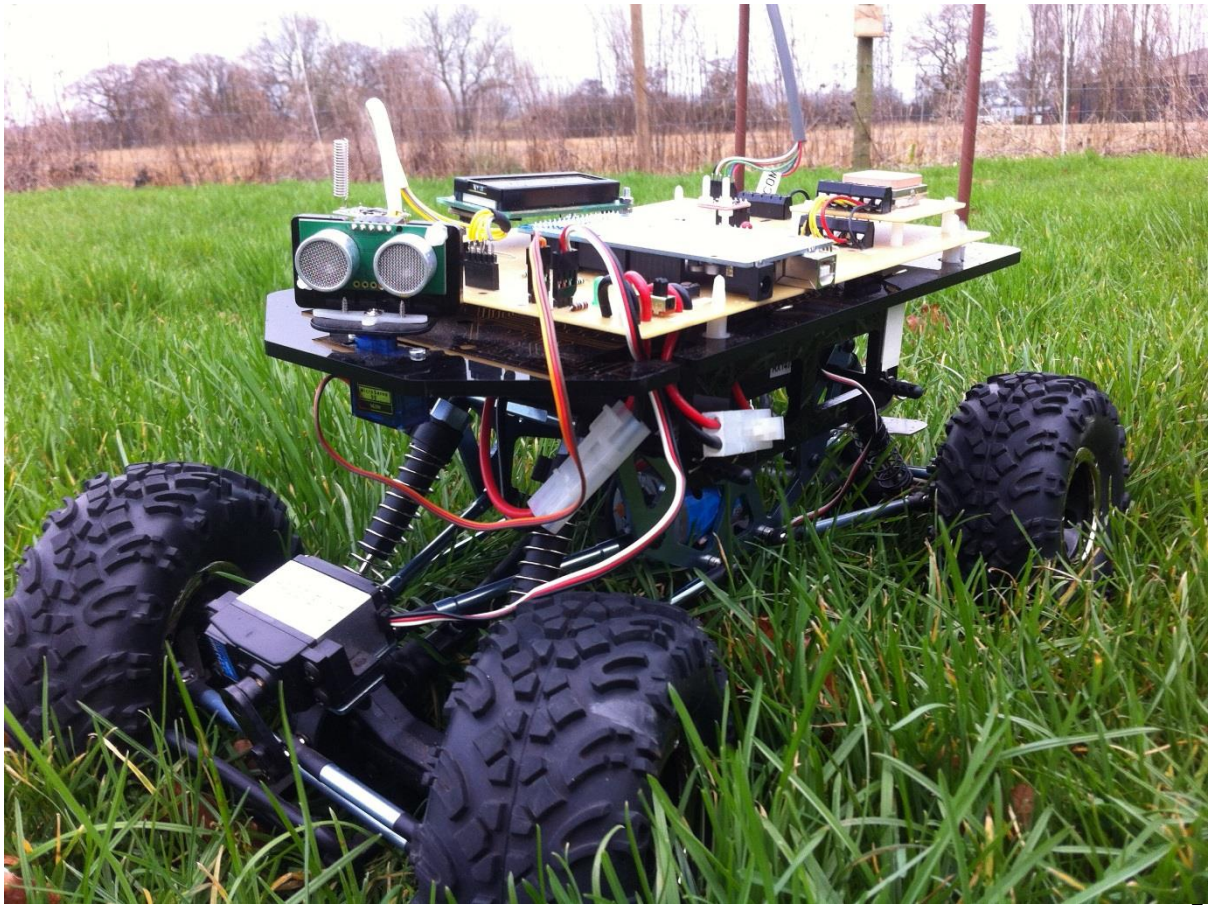
Welcome to Programming a GPS enabled robot using the Arduino.

Presented by Sam Wane, Senior Lecturer

swane@harper-adams.ac.uk

Harper Adams University



Contents

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.**

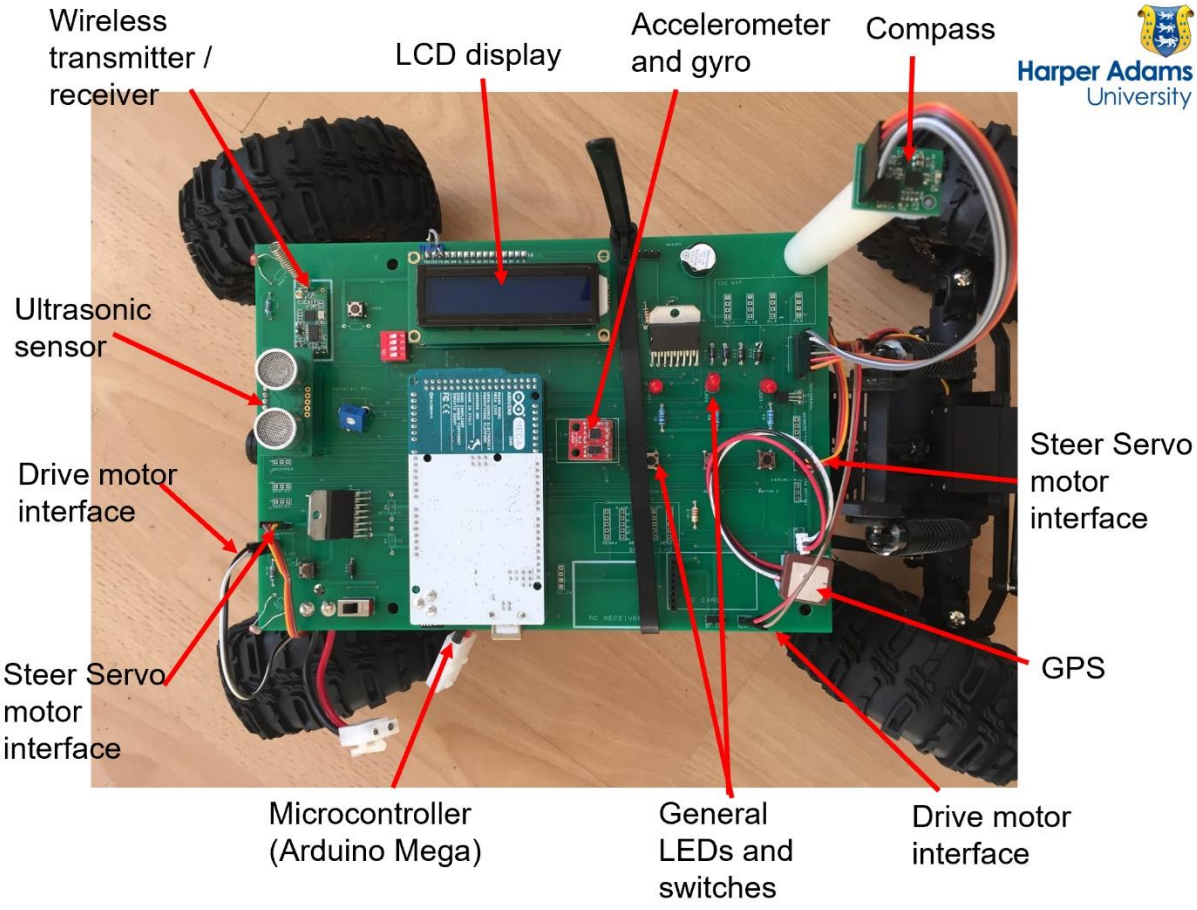


Figure 1 The Robot Hardware

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.

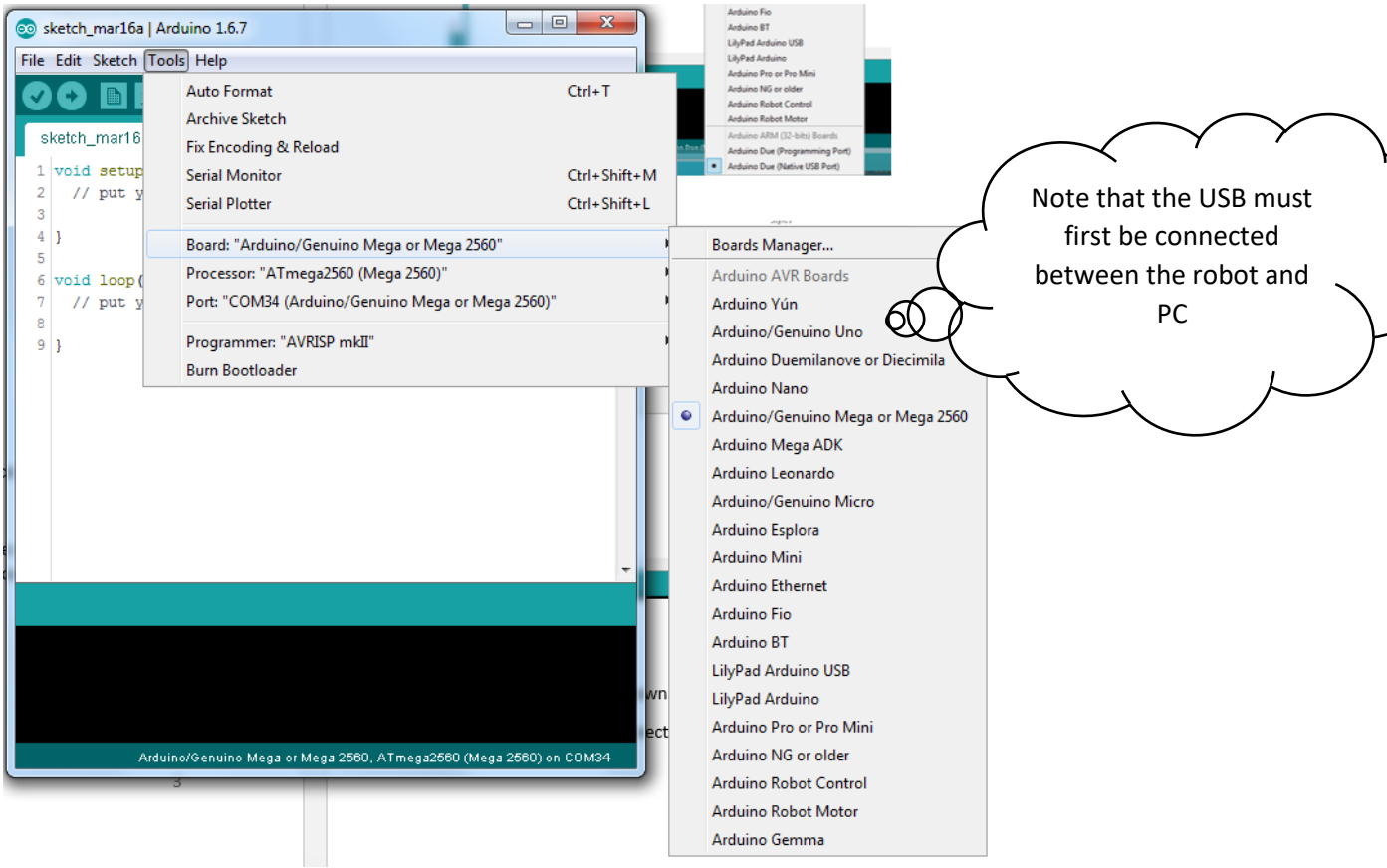


Figure 2 Selecting the Arduino type

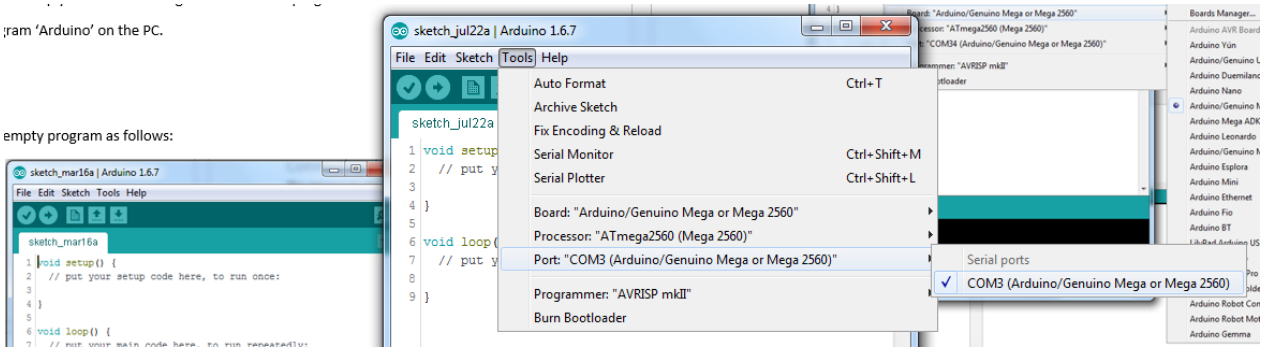


Figure 3 Selecting the Com port

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.

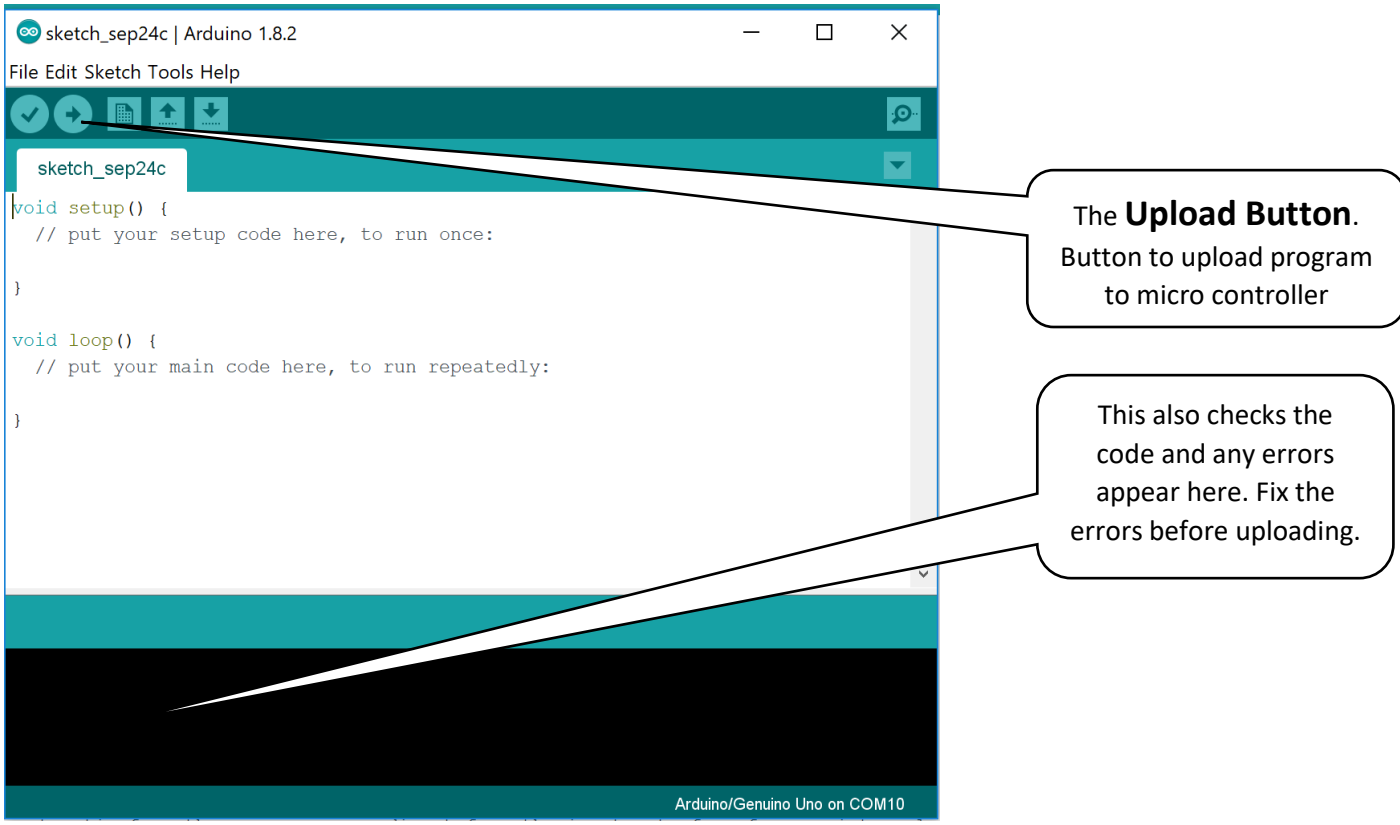


Figure 4 Empty program and how to upload

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

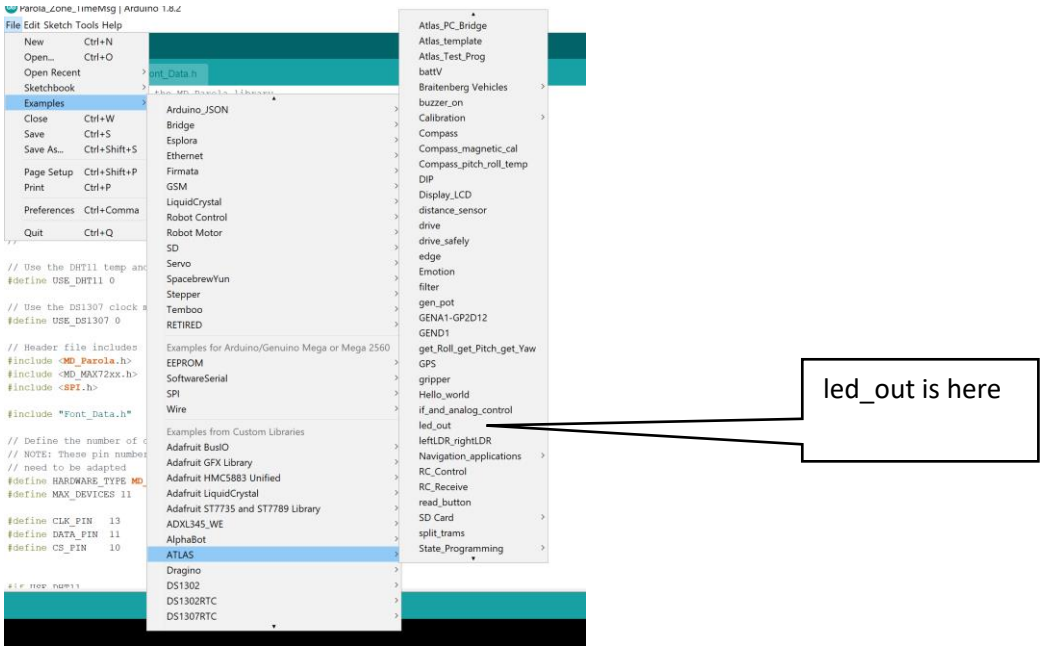
The ‘setup()’ section is used to initialise the hardware of the robot (it runs once and sets things up).

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_out'. You may need to hover your cursor over the down arrow to access the menu items. It should look like this:



Your program should look like this:

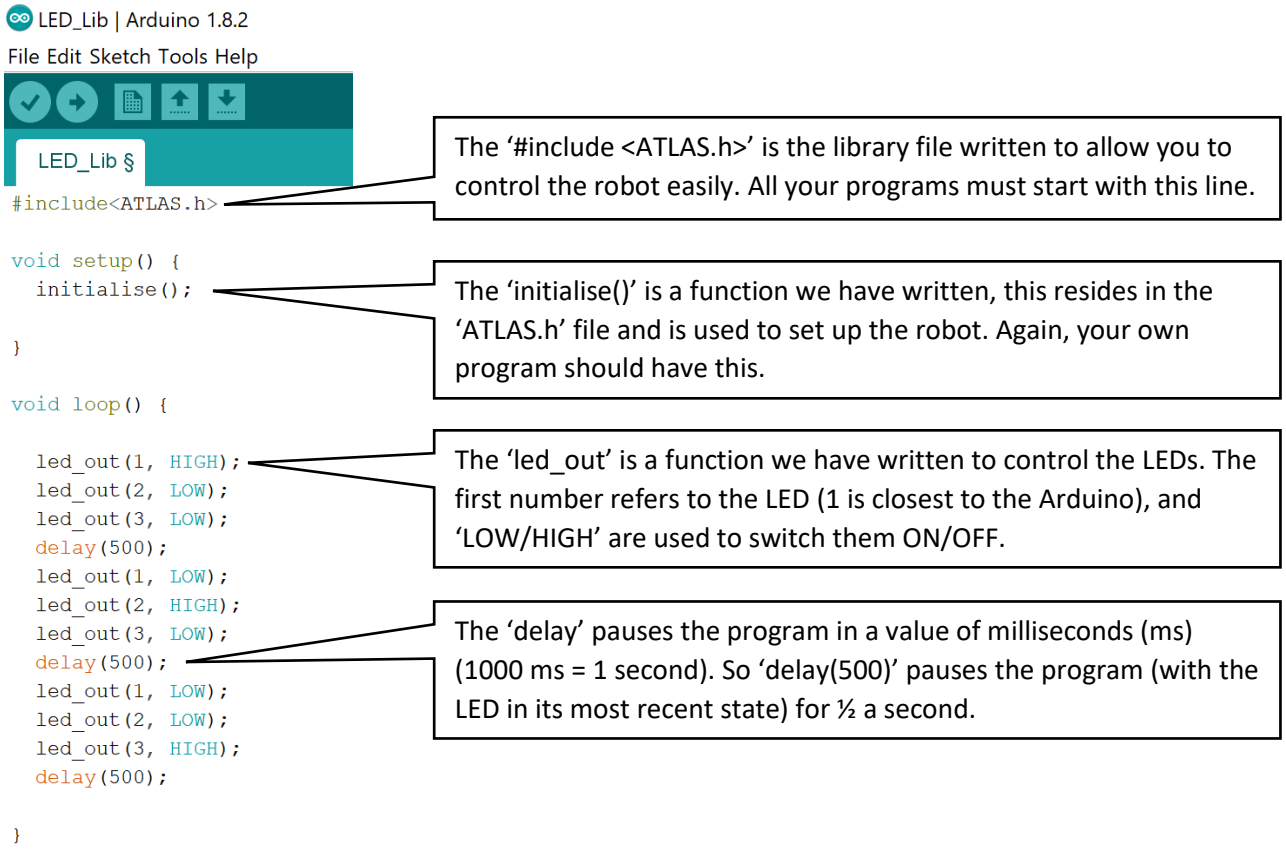
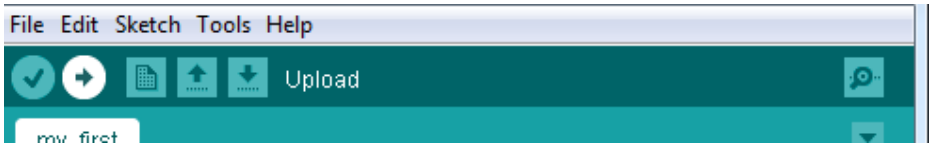


Figure 5 Flash LEDs Program

<u>You will find a list of the functions in the Appendix.</u>

- **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

Try 'File→Examples→Atlas→Display_LCD'. Again, upload the program and watch the robot.

Try a few more, e.g. 'File→Examples→Atlas→distance_sensor'

Distance sensor and graphing

The ultrasonic distance sensor is accessed through:

read_distance();

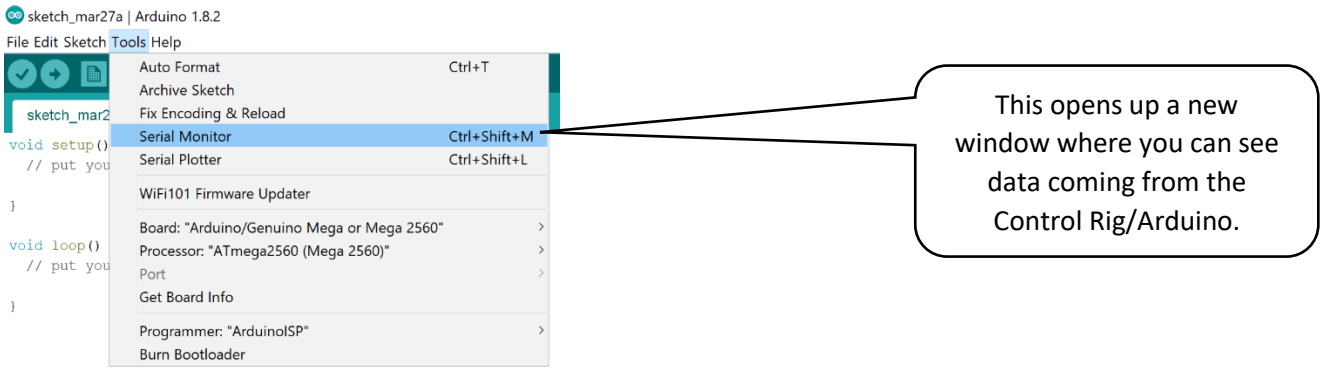
and returns the distance to the robot in cm.

This can be shown on the LCD screen and the example 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:



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.

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^{o} to 40^{o}. Positive angles are turning to the right, 0^{o} is straight ahead.

Try this program, either from the 'Examples→Atlas→Steering→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. The motors (drive and steer) will only operate if the external battery is connected and the main robot power is ON. Check this. You can also now unplug the robot from the computer and it will still continue to run the program.

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.

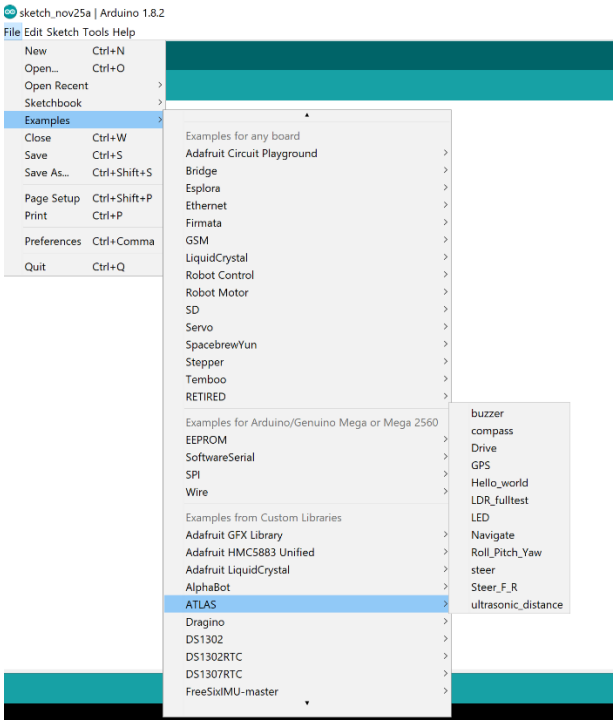
The example below is under 'Examples→Atlas→Drive':

```
#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);
}
```

Example programs

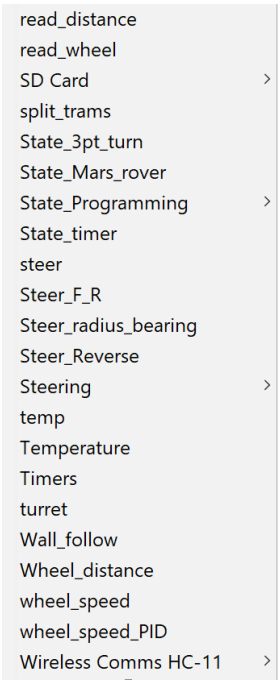
Example programs are available under: Files→Examples→ATLAS→

See below:



- ATLAS keywords-A list of keywords used in the libraries
- Atlas_PC_bridge-Interfacing with a controlling PC
- Atlas_template-A basic Atlas program
- Atlas_Test_Prog-Tests all the I/O of the Atlas
- battV-reads the battery level
- Braitenberg Vehicles→Dark_Power-drives faster in no light
- Braitenberg Vehicles→Light_Power-drives faster in bright light
- Braitenberg Vehicles→Fear_light-steers away from bright light
- Braitenberg Vehicles→Love_light-steers towards bright light
- buzzer_on-tests the buzzer
- Calibration→Calibration functions
- Compass-shows compass values
- DIP-shows DIP switches working
- Display_LCD-gives LCD display examples
- distance_sensor-The ultrasonic distance sensor
- drive-Make the drive motors operate
- drive_safely-drive forwards but poll the distance sensor to stop the drive at obstavcle
- edge-demonstrates edge detection on the button
- Emotion-demonstrates how anger can build up and dissipate with inputs and time
- Filter-the digital filter operating on a sensor
- gen_pot-reading the potentiometer
- GENA1-GP2D12-using the GP2D12 distance sensor via the general A1 input

- GEND1-using the general digital input
- Get_Roll_get_Pitch_get_Yaw-reading the accelerometer
- GPS-reading the latitude and longitude
- gripper-using the external servo port for a gripper
- Hello_world-basic program test
- If_and_analog_control-demonstrates discrete and analogue control
- led_out-controlling the LEDs
- left_LDR_right_LDR-read values of the light sensors
- Navigation_applications→Motorways
- Navigation_applications→NavigateArrayAB
- Navigation_applications→NavigateArrayGPS
- Navigation_applications→NavigateGPS
- Navigation_applications→Rcv_lat_lon_rescue_HC11
- Navigation_applications→ Rcv_lat_lon_rescue_PC
- Navigation_applications→Send_lat_lon_chars
- Navigation_applications→split_trams
- Navigation_applications→X_track_AB
- Navigation_applications→X_track_AB_application
- RC_Ccontrol
- RC_Receive
- Read_button
- Read_distance
- Read_wheel
- SD_Card



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 time.r 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 howerver. 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);
```

calc_x_track(double* AimN, double* AimE, double S_N,double S_E,double E_N, double E_E,double CN,double CE,double Aimdist);

double calc_x_track_bear(double S_lat,double S_lon,double E_lat, double E_lon,double C_lat,double C_lon,double Aimdist);

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.

```
void set_hall_ppr(double v);
void set_wheel_circ(double v);
```

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.

error=Output_PID(0.9, 0, 0.2, error, 45);
```

read_offset_bearing()
Returns the offset bearing as stored in EEPROM

write_offset_bearing(double ob)
Sets the offset bearing, stored in EEPROM

```
double velocity_input(double a);
```

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

GPS

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.

Orientation

get_Pitch(); or read_compass_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(); or read_compass_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.

Light sensors

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)

Ultrasonic distance sensor

read_distance();

Input: nothing
Returns: double: the distance sensed by the ultrasonic ranger in cm
Function:Reads distance.

Wireless

Serial2.read

Reads data from the wireless transceiver
e.g.

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

Compass

read_compass();

Input: nothing
Returns: Double of degrees
Function: Returns the bearing as measured by the magnetometer (compass) in the range 0-360.

For calibration:

```
store_compass_cal();
del_compass_cal();
read_compass_cal();
```

Temperature

float temp(); or read_compass_temp();

Input: nothing
Returns: temperature in degrees C
Function: Reads the temperature from the temperature sensor.

Input buttons

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);
 }
```

Wheel distance/speed sensors

unsigned long read_wheel();

Reads the Hall sensor

void zero_wheel();

Resets the Hall wheel count

unsigned long read_wheel_time();

unsigned long read_wheel();

unsigned long read_wheel_time();

double veh_speed();

Radio control input

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

All commands in the library

General

```
void initialise();

unsigned long get_timer(int timer_no);
void set_timer(int timer_no, unsigned long timerincrement);
bool timer_elapsed(int timer_no);

double Output_PID(double Kp, double Ki, double Kd, double error,
double maxerror);
double Output_PID(double Kp, double Ki, double Kd, double error);
double filter(double input, double f);
bool approx(double a,double b,double t);
```

Inputs

```
double trailer_angle();
void set_trailer_centre();
float rightLDR();
float leftLDR();
int read_button(int button);
double gen_pot();
double rev_pot();
double battery();

void imu_init();
float get_Yaw();
float get_Pitch();
float get_Roll();
float temp();
```

```
void initSD();
void openSD();
double read_offset_bearing();
void write_offset_bearing(double ob);
byte readDIP();
unsigned long read_wheel();
void zero_wheel();
double read_distance();
```

Navigation

```
double read_compass(void);
float getLat();
float getLon();
float getAlt();
float getN();
float getE();

double closest_bearing_difference(double current_bearing, double
goal_bearing);
double bearing_to_point_UTM(double current_northing, double
current_easting, double goal_northing, double goal_easting);
double distance_to_point_UTM(double current_northing, double
current_easting, double goal_northing, double goal_easting);

double bearing_to_point_deg(double current_lat, double current_lon,
double goal_lat, double goal_lon);
double distance_to_point_deg(double current_lat, double current_lon,
double goal_lat, double goal_lon);
double drive_target(double goal_northing, double goal_easting,
double offset);
double follow_wall(double des_dist, double gain);
bool edge_rising(int port);
bool scan(int stand_out, int* distr, int* bearr); //Returns 999,999
if not found


void calc_x_track(double* AimN, double* AimE, double S_N,double
S_E,double E_N, double E_E,double CN,double CE,double Aimdist)


double calc_x_track_bear(double S_lat,double S_lon,double E_lat,
double E_lon,double C_lat,double C_lon,double Aimdist)




  deg2utm(C_lat, C_lon, &CN, &CE);
```

Outputs

Display

```
void display(String topLine);
void display(String topLine, String bottomLine );
```

```
void clear_LCD();
void nextline_LCD();

void buzzer_on();
void buzzer_off();
void led_out(int led,bool value);
```

Steering

```
void steer(double steer_angle);
void steerc(double steer_angle);
double read_steer(double steer_angle);
void set_front_calibrate(int c);
void set_rear_calibrate(int c);
int read_front_calibrate();
int read_rear_calibrate();
void rear_steer(double steer_angle);
void steerslow(double steer_angle);
void rear_steerslow(double steer_angle);
double steer_radius(double inv_R);
void set_veh_length(double v);
void set_veh_width(double v);
void set_max_steerf(double s);
void set_min_steerf(double s);
```

Drive

```
void front_RC(float drive_speed);
void rear_RC(float drive_speed);
void drive(float drive_speed);
void drive(float drive_speed, byte accel);
void front_drive(float drive_speed);
void rear_drive(float drive_speed);
double RC_speed();
double RC_steer();


void turret(int angle);
void turret(int steer_angle, byte speed);
void turret(int steer_angle, byte speed, bool block);
void sweep(int *scanarray); //returns array of scans 10 deg apart
```

All pins

```
//Gen IO
#define GENA1 A11
#define GENA2 A12
#define GENA3 A13
#define GENA4 A14
#define GENA5 A15

#define GEND1 30
#define GEND2 33
#define GEND3 32
#define GEND4 23
#define GEND5 3
```

J19.CONNECTION	FUNCTION	PIN
Battery monitor	Analogue	A0
Trailer pot		A1
Right LDR		A2
Left LDR		A3
		A4
		A5
Temp sensor		A6
		A7
General POT		A8
		A9
		A10
Gen Analogue		A11
Gen Analogue		A12
Gen Analogue		A13
Gen Analogue		A14
Gen Analogue		A15
	S/Comm's	
Bluetooth Rx	Serial3	14(TX)
Bluetooth Tx		15(RX)
Wireless HC-11 RX	Serial2	16(TX)
Wireless HC-11 TX		17(RX)
spare TX,9600	Serial1	18(TX) N/C
GPS		19(RX)9600
IMU	I2C	20 SDA
IMU	I2C	21 SCL
	PWM	
Hall sensor		2
Spare interrupt on Dig5		3
Drive servo pin		4
Gripper Servo		5
Steer servo pin		6
General RC Servo2		7
L298 Rear IN1		8
L298 Rear IN2		9
L298 Front IN1		10

J19.CONNECTION	FUNCTION	PIN
L298 Front IN2		11
Compass RX		12
LCD RS PIN		13
Gen Dig4	DIGITAL	22
LCD E PIN		23
LCD DB4		24
LCD DB5		25
Ultrasonic echo		26
LCD DB6		27
Ultrasonic trigger		28
Sounder		29
LCD DB7		30
General Dig 1		31
General Dig 2		32
General Dig 3		33
REAR STEER SERVO		34
Compass TX		35
RC Rec Ch2		36
RC Rec Ch1		37
Led1		38
DIP1		39
Led2		40
DIP2		41
Led3		42
DIP3		43
Button1		45
DIP4		45
Button2		47
Turret servo pin		46
Button3		49
Gen servo 1		49
Serial SD card D0		50
Serial SD card D1		51
Serial SD card CLK		52
Serial SD card CS		53