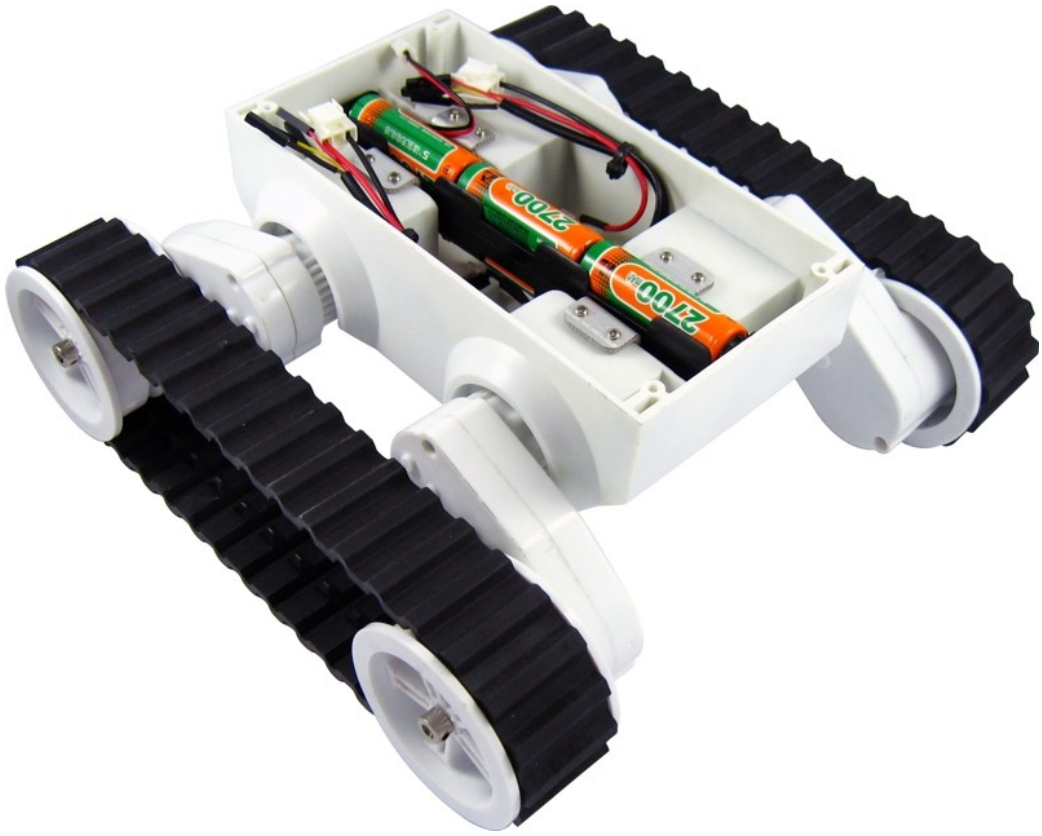
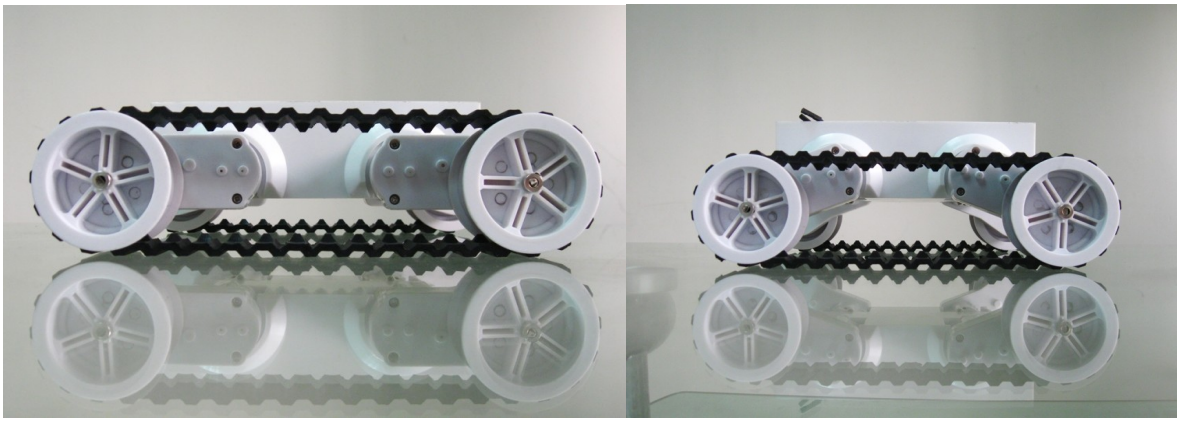
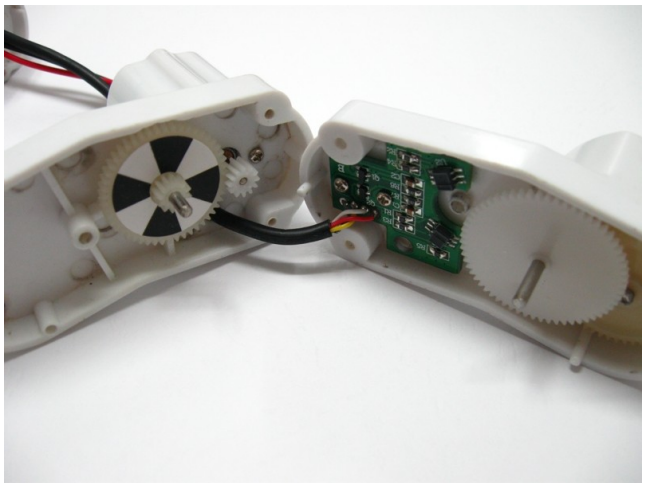
Rover 5

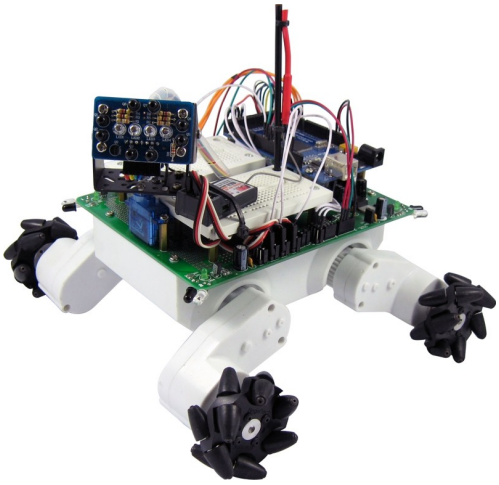


Rover 5 is a new breed of tracked robot chassis designed specifically for students and hobbyist. Unlike conventional tracked chassis's the clearance can be adjusted by rotating the gearboxes in 5-degree increments. "Stretchy" rubber treads maintain tension as the clearance is raised.

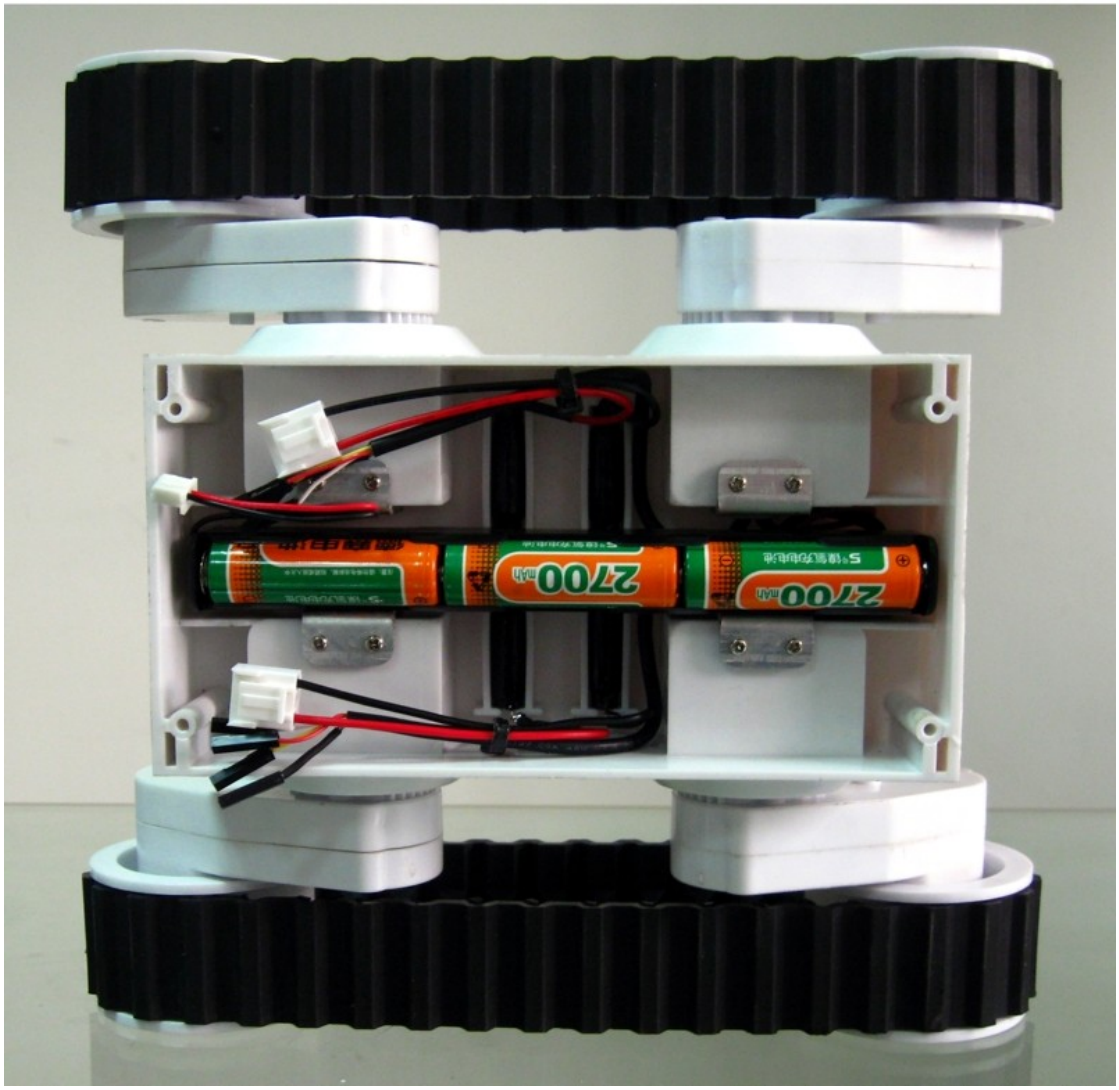


Each gearbox has an 87:1 ratio includes an optical quadrature encoder that gives 1000 pulses over 3 revolutions of the output shaft. The chassis can be upgraded to include four motors and encoders making it ideal for mecanum wheels.



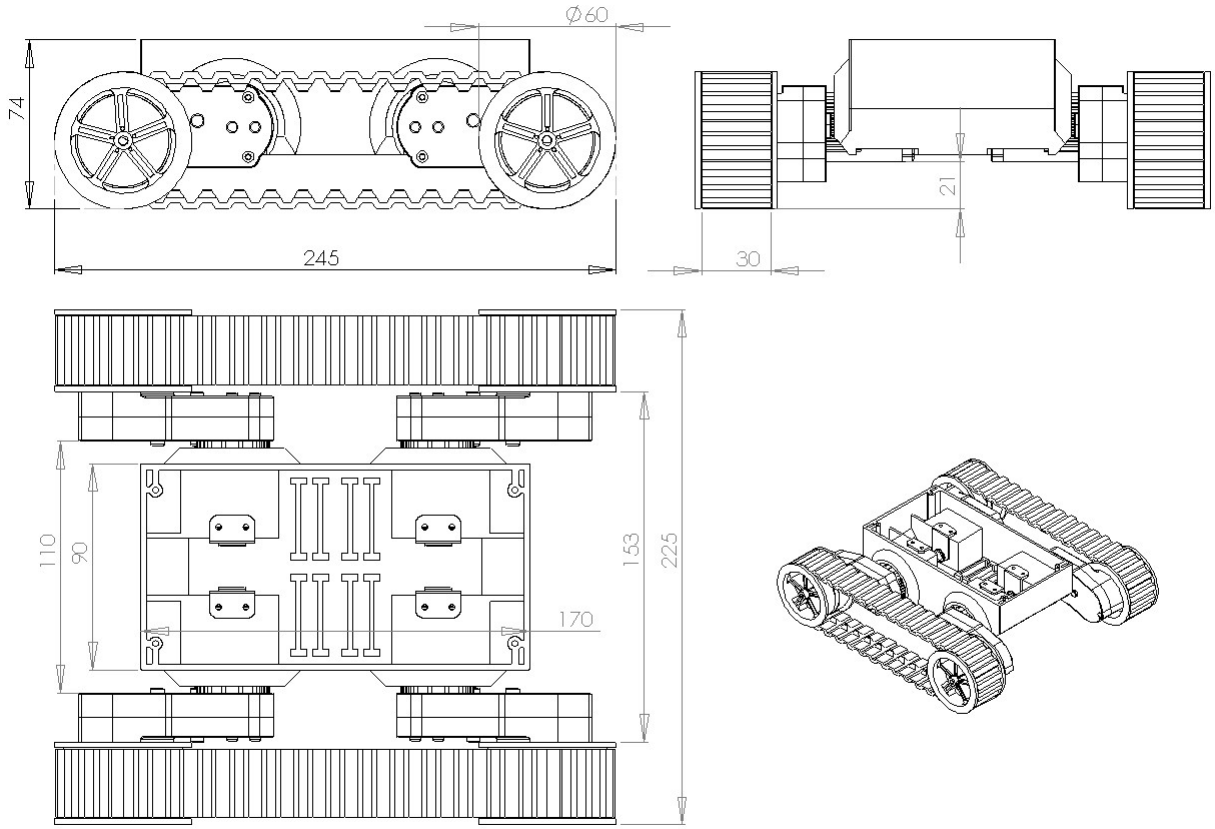


Inside of the chassis are 4 noise suppression coils at the bottom and a battery holder that accepts 6x AA batteries. It is recommended to use NiMh batteries as they last longer and have a higher current output than Alkaline batteries.



Video of the chassis in action can be seen here:
Video indoors autonomous: http://v.youku.com/v_show/id_XMjE5NzkwODA0.html
Video outdoors RC mode: http://v.youku.com/v_show/id_XMjIwMTkxODk2.html

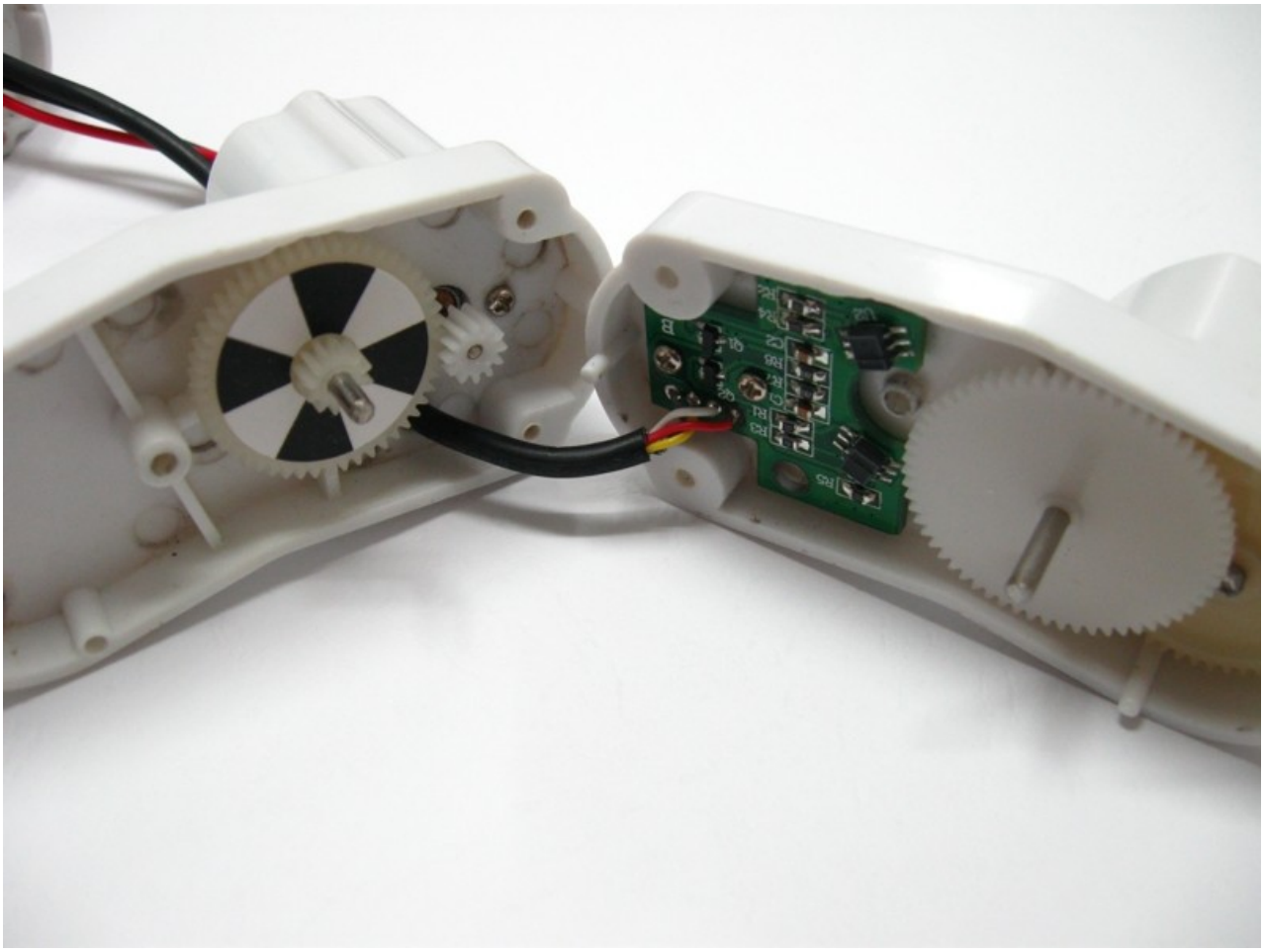
Dimensions:



Specifications:

Motor rated voltage: 7.2V
Motor stall current: 2.5A
Output shaft stall torque: 10Kg/cm
Gearbox ratio: 86.8:1
Encoder type: Quadrature
Encoder resolution: 1000 state changes per 3 wheel rotations
Speed: 1Km/hr

Using Quadrature Encoders



Each motor housing has two thick power wires that go directly to the motor. There are also 4 small wires with female headers.

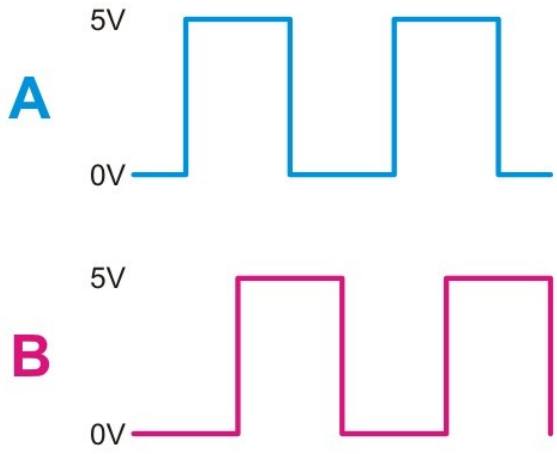
RED is +5V for the encoder
BLACK is 0V (ground)
WHITE is signal A
YELLOW is signal B

Swapping signals A & B will reverse the sense of direction for the encoder. Do not swap the power wires for the encoder as this will damage the circuit.

A quadrature encoder, also known as an incremental rotary encoder measures the speed and direction of a rotating shaft.

Quadrature encoders can use different types of sensors, optical and hall effect are both commonly used. The photo shows inside of a Rover 5 gearbox. There are two IR sensors on the PCB that look at the black and white pattern on one of the gears.

No matter what type of sensors are used the output is typically two square waveforms 90° out of phase as shown below.



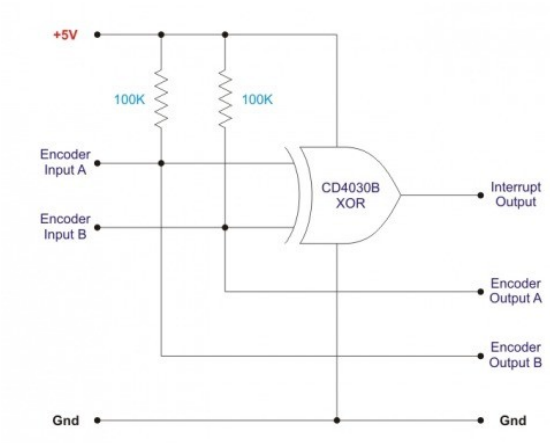
If you only wish to monitor the speed of rotation then you can use either output and simply measure the frequency. The reason for having two outputs is that you can also determine the direction of shaft rotation by looking at the pattern of binary numbers generated by the two outputs. Depending on the direction of rotation you will get either:

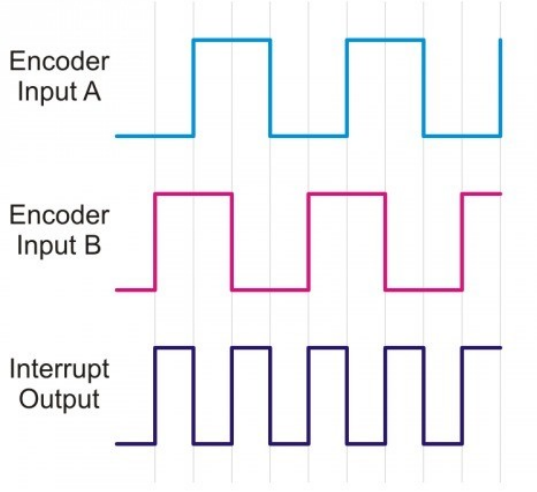
00 = 0
01 = 1
11 = 3
10 = 2

or

00 = 0
10 = 2
11 = 3
01 = 1

By feeding both outputs into an XOR gate (exclusive OR) you will get a square wave with twice the frequency regardless of direction. This can be useful as it allows one interrupt pin to monitor both encoder inputs.





I was looking at how to write efficient code to convert these binary inputs into a simple "forward or backward" output. I ended up with a 2 dimensional array (matrix) that made the code quick and easy.
The binary values above convert to 0,1,3,2 or 0,2,3,1 depending on the direction. This pattern repeats continuously. By using the current value from the encoder to index one dimension of the array and the previous value to index the other dimension you can quickly get a -1, 0, or +1 output. My array looks like this.

Old Value \ Current Value	0	1	2	3
0	0	-1	+1	X
1	+1	0	X	-1
2	-1	X	0	+1
3	X	+1	-1	0

As you can see, if the value has not changed then the output is 0.
The sequence of 0, 1, 3, 2 gives an output of -1.
The sequence of 0, 2, 3, 1 gives an output of +1.
X represents a disallowed state and would most likely occur if the encoder outputs are changing too quickly for your code to keep up.

Normally this should not happen. In my code I put a 2 here. When I get an output of 2 I know that I got an error, perhaps due to electrical noise or my code being too slow. If you replace X with 0 then the disallowed state will be ignored.
In my Arduino code I make this a 1 dimensional array. that looks like this:

```
int QEM [16] = {0,-1,1,2,1,0,2,-1,-1,2,0,1,2,1,-1,0};          // Quadrature Encoder Matrix
```

To read the array my index is: Old * 4 + New
So my code reads like this:

```
Old = New;
New = digitalRead (inputA) * 2 + digitalRead (inputB);          // Convert binary input to decimal value
Out = QEM [Old * 4 + New];
```

Good luck and enjoy.