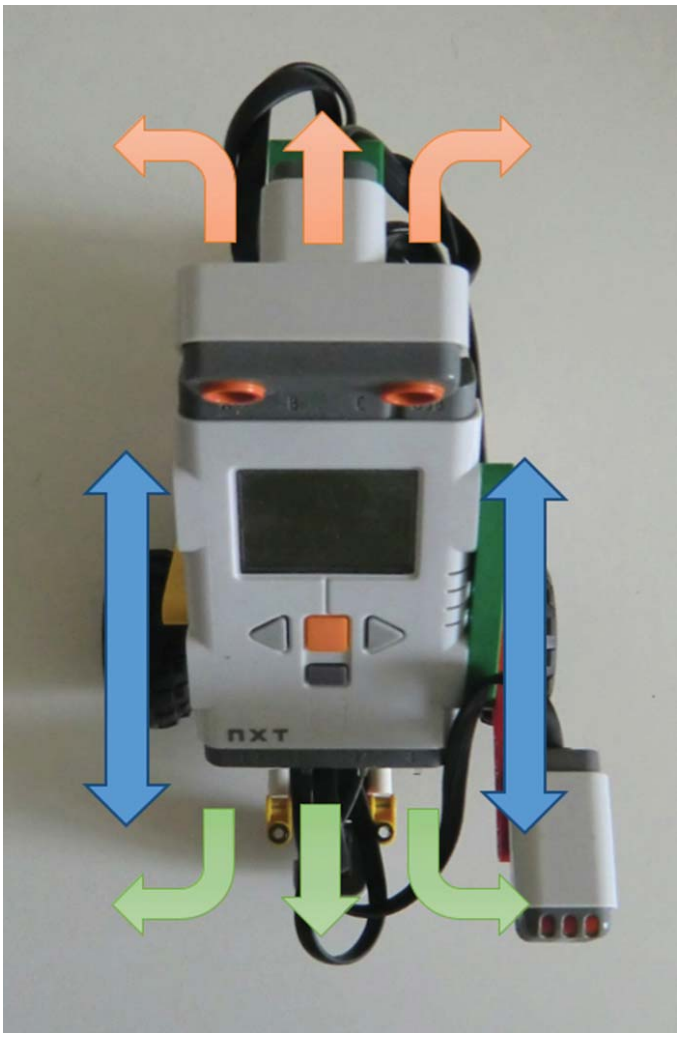
Any which way!

The wheels on the bot go round and round

Xander Soldaat, Infrastructure Architect and robotics enthusiast

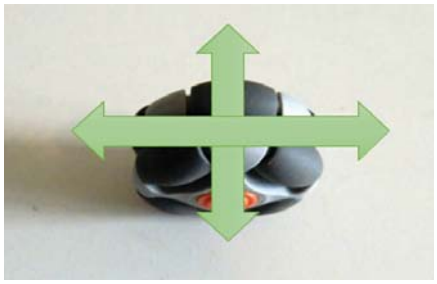
Robots come in all shapes and sizes, none more so than ones made with LEGO MINDSTORMS, where the only limit is your imagination, and possibly the number of pieces at your disposal.

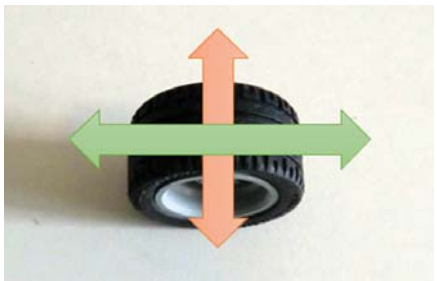
Traditionally, most robots used for education use the standard differential drive configuration, two powered wheels, one on each side. When the motors move in the same direction, the robot goes forward, when they move in opposite directions, the robot pivots.



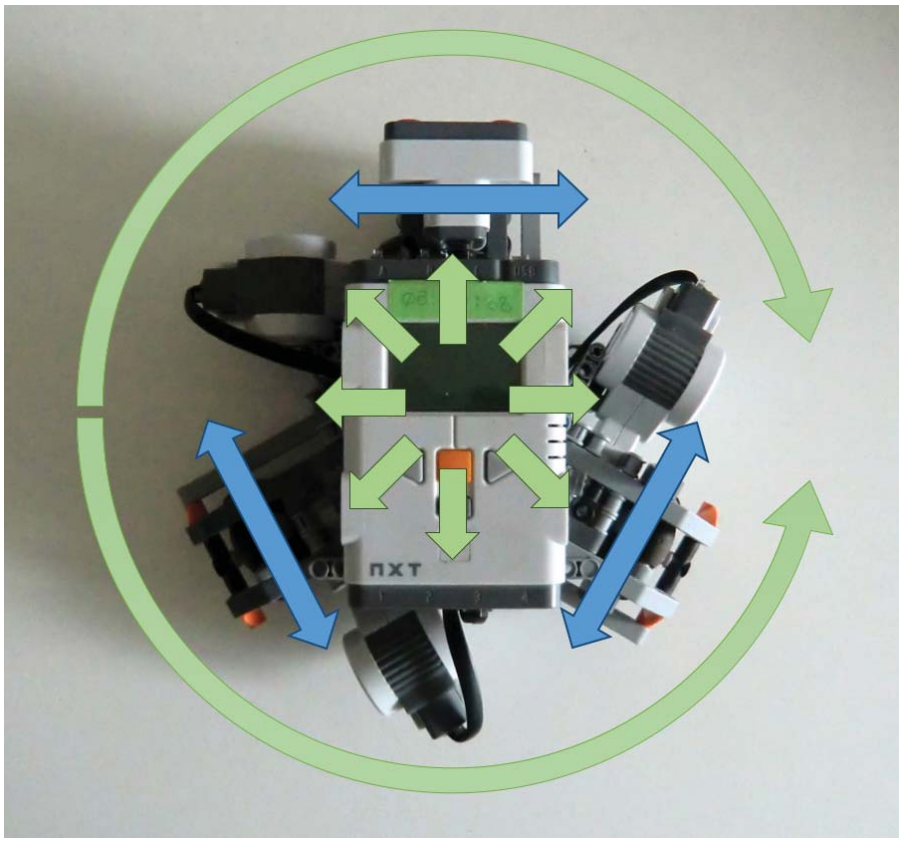
Such a configuration limits the number of directions a robot can move. Sometimes, being able to move in 'any' direction can be a massive advantage. Think of a robot in a large warehouse with very tight corners or little space between racks. Another example where extra manoeuvrability is a strategic advantage would be a football-playing robot.

Normal wheels are made to turn in one direction and tend to stick to the surface quite well if pushed sideways, up to a point, of course. Omnidirectional wheels or omniwheels for short, are different. Unlike their ordinary cousins, these ones are able to roll in any direction due to small rollers that are able to turn in the length of the axle.





Omniwheels, being able to move freely in one direction and retain grip in the other can be mounted in a triangular fashion. When these wheels are mounted at 120° angles, they make a very stable platform. This type of setup is also known as a Killough platform.



If we power all the wheels in the same direction, the platform would simply rotate about its centre. However, it is possible to make the platform move in useful directions too.

The maths behind the movement

Due to the nature of the configuration of the wheels, a little maths is required to convert robot motion to motor speed. We'll ignore the robots' ability to rotate for now and come back to that later.

The robot's linear speed has velocity and direction. Being polar co-ordinates, these can be transformed into Cartesian co-ordinates, or a forward speed (X-axis) and a sideways speed (Y-axis).

1. $V_{rx} = V_{linear} \cdot \cos(direction)$
2. $V_{ry} = V_{linear} \cdot \sin(direction)$

To calculate the wheel speed we can now project the two components of the robot's speed on the wheel driving axis using the wheel angle. This gives two speeds, one on account of the robot's forward motion, the other on account of its sideways motion. These two components added up give the wheel speed on behalf of the robot's linear motion.

3. $V_{wx} = V_{rx} \cdot \cos(wheelAngle)$
4. $V_{wy} = V_{ry} \cdot -\sin(wheelAngle)$

Now, to get back to the angular motion of the robot. Due to its body shape, where the driving wheels are aligned with the center, we do not have to perform any additional transformation of the robot's angular speed, so it can be stated that:

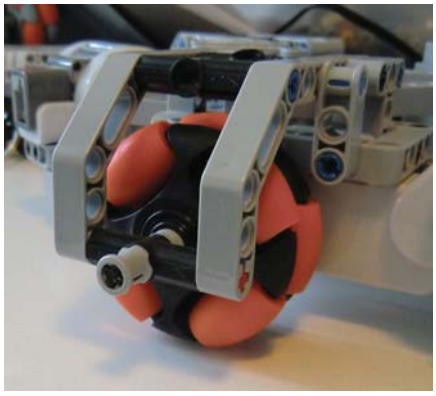
5. $V_{wa} = V_{angular}$

It is now possible to add these three components (3, 4 and 5) to get the wheel speed:

6. $V_w = V_{linear} \cdot \cos(direction) \cdot \cos(wheelAngle) - V_{linear} \cdot \sin(direction) \cdot \sin(wheelAngle) + V_{angular}$

Which can be simplified as:

7. $V_w = V_{linear} \cdot (\cos(direction) \cdot \cos(wheelAngle) - \sin(direction) \cdot \sin(wheelAngle)) + V_{angular}$



Using 7, we can now calculate the motor speeds for our robot, which has its wheels A, B and C at 0°, 120° and 240°. Substituting the wheel angles in the equation, we get the following:

8. $V_{wA} = V_{linear} \cdot (\cos(direction) \cdot \cos(0) - \sin(direction) \cdot \sin(0)) + V_{angular}$
9. $V_{wB} = V_{linear} \cdot (\cos(direction) \cdot \cos(120) - \sin(direction) \cdot \sin(120)) + V_{angular}$
10. $V_{wC} = V_{linear} \cdot (\cos(direction) \cdot \cos(240) - \sin(direction) \cdot \sin(240)) + V_{angular}$

These can be simplified to these:

11. $V_{wA} = V_{linear} \cdot \cos(direction) + V_{angular}$
12. $V_{wB} = V_{linear} \cdot \left(-0.5 \cdot \cos(direction) - \frac{\sqrt{3}}{2} \cdot \sin(direction)\right) + V_{angular}$
13. $V_{wC} = V_{linear} \cdot \left(-0.5 \cdot \cos(direction) + \frac{\sqrt{3}}{2} \cdot \sin(direction)\right) + V_{angular}$

Putting it into practice

Now that we have our three equations for each motor, we can start putting them into ROBOTC code. We'll create a function that takes the travel angle, our linear speed and the speed at which we wish to rotate as its arguments and translates these into individual motor speeds:

```
void MoveRobot(int angle, int Vlinear, int
Vangular) {

  // Calculate the motor speeds
  float VwA = Vangular + Vlinear *
cosDegrees(angle);
  float VwB = Vangular + Vlinear * (-0.5 *
cosDegrees(angle) - 0.866 * sinDegrees(angle));
  float VwC = Vangular + Vlinear * (-0.5 *
cosDegrees(angle) + 0.866 * sinDegrees(angle));

  // Set the motor speeds
  motor[motorA] = round(VwA);
  motor[motorB] = round(VwB);
  motor[motorC] = round(VwC);
}
```

If we want to move the robot along the X-axis (0°) for 1 second at speed 50, we'd call the MoveRobot() function as follows:

```
task main ()
{
  MoveRobot(0, 50, 0);
  wait1Msec(1000);
}
```

The robot will move like this:

To move the robot along the Y-axis (90°), use the following code:

```
task main ()
{
  MoveRobot(90, 50, 0);
  wait1Msec(1000);
}
```

The robot can also move along both axes by specifying a 45° angle:

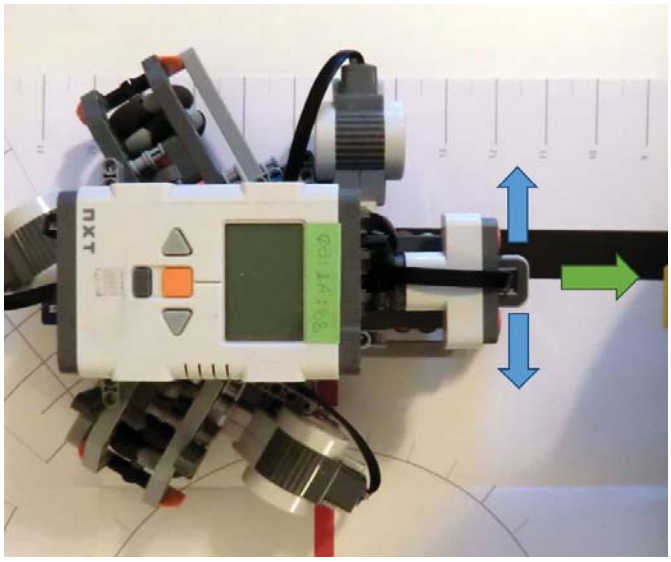
```
task main ()
{
  MoveRobot(45, 50, 0);
  wait1Msec(1000);
}
```

The movements can be combined to create shapes, this makes a square:

```
task main ()
{
  MoveRobot(90, 50, 0);
  wait1Msec(500);
  MoveRobot(180, 50, 0);
  wait1Msec(500);
  MoveRobot(270, 50, 0);
  wait1Msec(500);
  MoveRobot(0, 50, 0);
  wait1Msec(500);
}
```

Toeing the line

Now that we've seen the effects of `Vlinear` and `direction`, how do we apply the `Vangular` part? If we mount a light sensor on the front of the robot, we could use it to follow a line. We can use `Vangular` to maintain the position of the robot above the edge of the line.



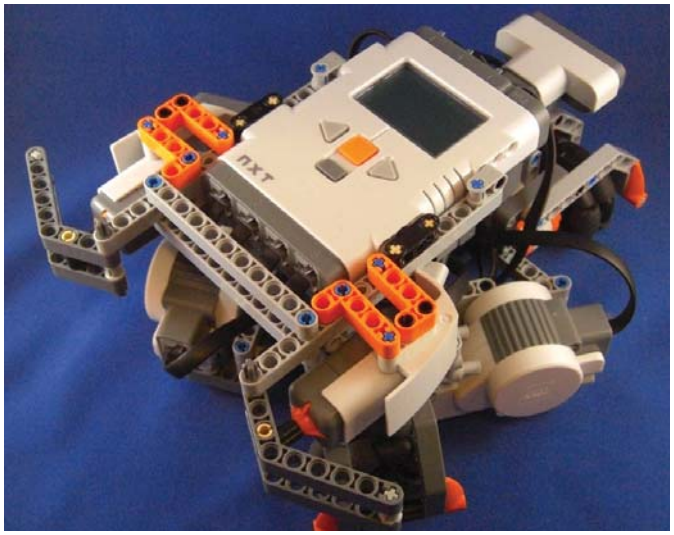
The line follower will use a simple proportional response to the deviation from the line. By sitting the sensor right at the edge of the line, a value that is somewhere between the white and black value will be considered the centre.
Use the following piece of code to determine what that value is:

```
#pragma config(Sensor, S1,     LIGHT,
sensorLightActive)
//*!!Code automatically generated by 'ROBOTC'
configuration wizard               !!*//

task main()
{
  while (true)
  {
    nxtDisplayCenteredBigTextLine(2, "%3d",
SensorValue[LIGHT]);
    wait1Msec(50);
  }
}
```

This will display the current value of the sensor on the screen. In the case of this experiment, the values were found to be 55 for white and 30 for black, so 43 will be considered the value for the sensor being centred over the edge.

To make the robot follow the line, consider the following code. Note that "LIGHT" is an alias for a LEGO Light Sensor attached to port S1. A link to the complete listing can be found in the links section.



```
task main ()
{
  // Change these to suit your environment
  int white_value = 55;
  int black_value = 30;
  int Vlinear = 30;
  // Make this negative if you find the robot
moving in the wrong
  // direction with regards to the line.
  float multiplier = 1.0;

  //
  float off_centre = 0.0;
  float Vangular = 0.0;

  while (true)
  {
    // Read the sensor value and calculate how
off-centre the robot is
    off_centre = SensorValue[LIGHT] -
((white_value + black_value) / 2);
    Vangular = round(off_centre * multiplier);

    // Display the amount of correction on the
screen (useful for debugging
    nxtDisplayBigTextLine(1, "%d", off_centre);

    // Keep in mind that the sensor is mounted
at 90 degrees with
    // respect to motor A's axle
    MoveRobot(90, Vlinear, Vangular);

    // Do this 20 times per second
    wait1Msec(50);
  }
}
```

Conclusion

Omniwheels offer very exciting new types of robotic locomotion to explore. With a little experimentation, it is possible to make the robot move straight and rotate simultaneously. To use omniwheels you will need at least three wheels per robot. The wheels on this specific robot are made by Rotacaster. Combining this platform with an IR football-seeking sensor, can make it a fun game to help students in learning advanced programming skills. A gyroscope or compass sensor could be used as tools for concepts such as accurate positioning instead of simply using dead reckoning. With a little imagination and skill, omniwheels can take your robotics curriculum in a new direction.

Links

- Xander Soldaat's Botbench website: **www.botbench.com**
- Software for this article: **www.botbench.com/blog/dt-article**
- ROBOTC for NXT: **www.robotc.net/download/nxt/**
- Building Instructions for the HiTechnic RotaBot: **www.hitechnic.com/models**
- Laurens Valk's website: **http://robotsquare.com/**
- Aswin Bouwmeester's site: **http://nxttime.wordpress.com/**
- Rotacaster: **http://rotacaster.com.au/**

Credits

Thank you to Laurens Valk and Aswin Bouwmeester for helping out with the maths.