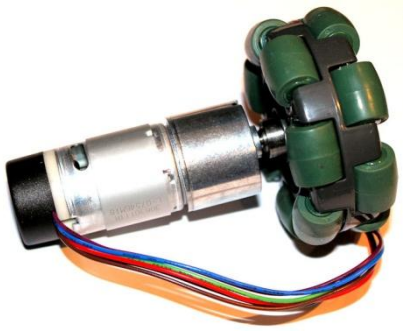
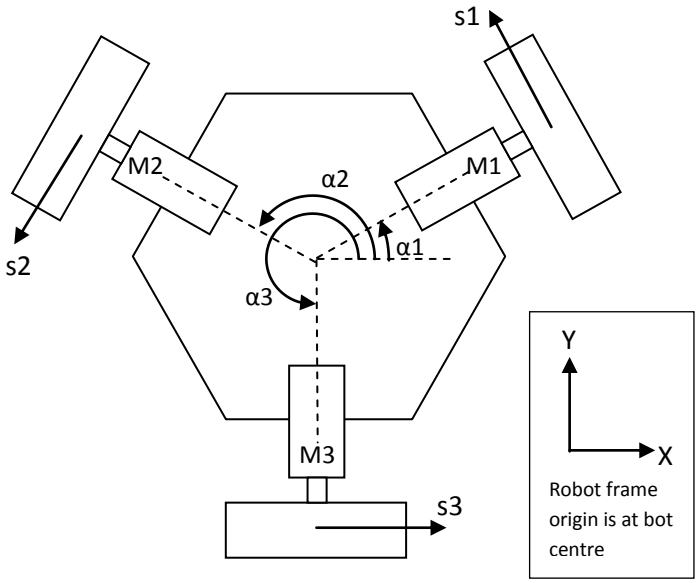
A Simple Introduction to Omni Roller Robots

(3rd April 2015)



Omni wheels have rollers all the way round the tread so they can slip laterally as well as drive in the direction of a regular wheel. The three-wheeled holonomic drive with omni roller wheels is usually called a 'Kiwi Drive'. But how to control this robot? We use a bit of trig and some matrices. We also look at forward kinematics and inverse kinematics.



α is the angle of the motor axis from the x axis of the robot coordinate frame

α1 = 30°; α2 = 150°; α3 = 270°

Say the motor is going to turn clockwise (as seen from the wheel end of the motor) then it will drive in the direction shown by the arrows 's'. Omni-wheels slip in the direction of the motor axis α. The wheel drive axis, shown by the 's' arrow, is 90° or π/2 from its respective α.

We will add π/2 to α to get the direction of each wheel. (But you could also subtract the π/2, it's not important actually). (We are using degrees here, but be aware any calculations with trig in Microsoft Excel will need radians. We don't use that here.)

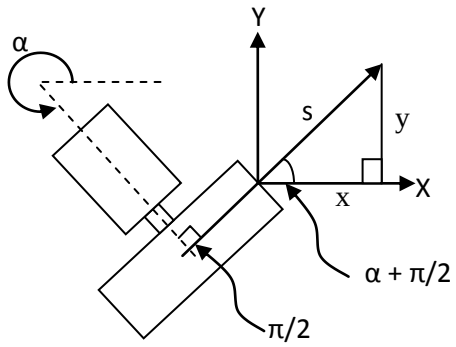
w1 = α1 + π/2 = 30° + 90° = <u>120°</u>

w2 = α3 + π/2 = 150° + 90° = <u>210°</u>

w3 = α3 + π/2 = 270° + 90° = <u>0°</u>

When each wheel turns +ve (clockwise) it has a speed 's' in the direction shown. When it turns -ve it has a speed -s. We need to resolve the vector associated with 's' (the wheel 'drive' orientation) into x and y components in the robot coordinate frame. In this design with each wheel arranged radially from the centre it's not necessary to factor in the distance of the wheel from the centre of the robot.

To resolve the vector 's' into x and y components we use some simple trigonometry.



For each of the three wheels, the x and y components of its orientation on the robot are:

Wheel 1: $x1 = \cos(\alpha1 + \pi/2).s1$ and $y1 = \sin(\alpha1 + \pi/2).s1$

Wheel 2: $x2 = \cos(\alpha2 + \pi/2).s2$ and $y2 = \sin(\alpha2 + \pi/2).s2$

Wheel 3: $x3 = \cos(\alpha3 + \pi/2).s3$ and $y3 = \sin(\alpha3 + \pi/2).s3$

It would be possible to drive each motor at a certain speed and then use the above to work out the robot speed and direction as a whole, relative to the robot coordinate frame. To calculate this we add up all the individual wheel x and y contributions:

$x = x1 + x2 + x3; \quad y = y1 + y2 + y3$

To do this we substitute for x_i and y_i using the trig expressions we worked out for each wheel:

$x = \cos(\alpha1 + \pi/2).s1 + \cos(\alpha2 + \pi/2).s2 + \cos(\alpha3 + \pi/2).s3$

$y = \sin(\alpha1 + \pi/2).s1 + \sin(\alpha2 + \pi/2).s2 + \sin(\alpha3 + \pi/2).s3$

There's also a rotational component ω, where the robot can turn about its z axis. (We can't move up and down the 'z' axis unless we have a helicopter.) The robot rotation is just the simple summation of each motor speed. Even if the motors are turning opposite ways we still generally get some amount of overall rotation of the robot. Just add up the motor speeds to find the robot rotation ω.

$\omega = s1 + s2 + s3$

We can write out the components for x, y, and ω as a matrix, and create a matrix equation that relates all the motor speeds to the resulting motion in x, y and ω:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} \cos(\alpha1 + \pi/2) & \cos(\alpha2 + \pi/2) & \cos(\alpha3 + \pi/2) \\ \sin(\alpha1 + \pi/2) & \sin(\alpha2 + \pi/2) & \sin(\alpha3 + \pi/2) \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \end{pmatrix}$$

Let's write this matrix equation in a simplified form that we can refer back to:

$$\underset{v}{\rightarrow} = (M) \underset{s}{\rightarrow}$$

This equation shows that we can take a vector $\underset{s}{\rightarrow}$ of speed information for all the motors, then pre-multiply it by the matrix M, to determine a velocity vector $\underset{v}{\rightarrow}$ that the robot will follow.

Kinematics and Inverse Kinematics

The problem is, we don't want to set the motor speeds 's' and then do a calculation to find where the robot will go to in x, y, and ω. This is called the forward kinematics. We want to decide where to send the robot and then calculate the motor speeds we must set in order to send it there. This is the inverse kinematics, and the first step to do this is to invert the matrix 'M' above. We could do this long-hand, but instead let's paste it into Wolfram Alpha and click '='.

Here's the matrix 'M' written in Wolfram Alpha plaintext form, together with the function 'invert':

invert {{cos(30+90), cos(150+90), cos(270+90)}, {sin(30+90), sin(150+90), sin(270+90)}, {1, 1, 1}}

Paste it into http://www.wolframalpha.com to get the inverse matrix M^{-1} which Wolfram Alpha gives as:

$$M^{-1} = \frac{1}{3}\begin{pmatrix} -1 & \sqrt{3} & 1 \\ -1 & -\sqrt{3} & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -0.33 & 0.58 & 0.33 \\ -0.33 & -0.58 & 0.33 \\ 0.67 & 0 & 0.33 \end{pmatrix}$$

As an aside, let's write this inverse matrix in Wolfram Alpha plaintext notation because we will use it in some examples for driving the robot: M^{-1} = {{-0.33, 0.58, 0.33}, {-0.33, -0.58, 0.33}, {0.67, 0, 0.33}}

Back to the math: We then multiply the inverse matrix M^{-1} into both sides of the matrix equation we had before:

$$(M)^{-1} \underset{v}{\rightarrow} = (M)^{-1}(M) \underset{s}{\rightarrow}$$

On the right hand side, the product of the matrix and its inverse cancels out leaving:

$$(M)^{-1} \underset{v}{\rightarrow} = \underset{s}{\rightarrow}$$

This can be used to navigate the robot. We can form a command to drive the robot in x, y and ω. We format the command as $\underset{v}{\rightarrow}$ = (x, y, ω) then pre-multiply that by the inverse kinematic matrix $(M)^{-1}$ to find the set of individual speeds $\underset{s}{\rightarrow}$ = (s1, s2, s3) that we need to apply for each motor.

Driving the Robot

Everything so far has been to explain inverse kinematics and the derivation of the inverse matrix we need. Now we come onto how this is used in robot control. For example to move forward, in the +y direction, (x, y, ω) = (0,1,0). We just multiply that vector by the inverse kinematic matrix. We can do the simple multiplication $(M)^{-1} \underset{v}{\rightarrow}$ in micro-controller code. But for the example we can do it using using Wolfram Alpha, as:

{{-0.33, 0.58, 0.33}, {-0.33, -0.58, 0.33}, {0.67, 0, 0.33}}*{0,1,0}

This will give us speeds (s1, s2, s3) = (0.58, -0.58, 0)

These will be the relative motor speeds for M1, M2 and M3. Note -ve is CCW and +ve is CW. All that remains is to program these matrix multiplications into the robot micro-controller and feed in some

direction commands. The direction commands can come from a joystick or from machine vision in openCV. It doesn't matter. Once we have the (x, y, ω) command, it's a simple calculation to work out the motor speeds to drive the robot accordingly.

If the robot goes forward instead of reverse you can swap all the signs for the motor speeds (or redo the matrix, subtracting the π/2 instead of adding it). You can scale the relative speeds for each motor. Obviously the speed for a motor can't exceed the maximum possible motor speed. It's good to write program code that checks for this and if necessary, normalises each speed so the fastest required speed of any motor is the maximum possible motor speed, reducing the other speeds proportionately. Let's look at some more examples:

To move right, in +x direction, (x, y, ω) = {1,0,0}

{{-0.33, 0.58, 0.33}, {-0.33, -0.58, 0.33}, {0.67, 0, 0.33}}*{1,0,0} = (-0.33, -0.33, 0.67)

To rotate on the spot, (x, y, ω) = {0,0,1}

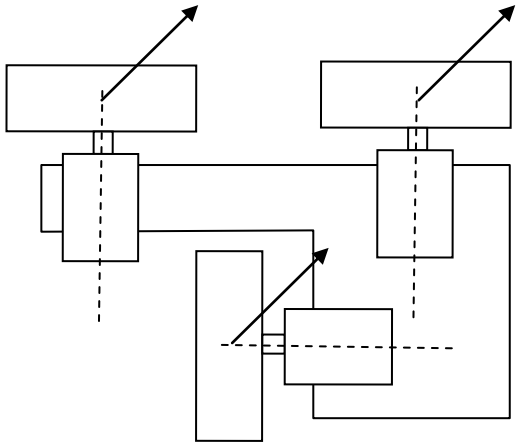
{{-0.33, 0.58, 0.33}, {-0.33, -0.58, 0.33}, {0.67, 0, 0.33}}*{0,0,1} = (0.33, 0.33, 0.33)

To move at 45° forward and right, (x, y, ω) = {1,1,0}

{{-0.33, 0.58, 0.33}, {-0.33, -0.58, 0.33}, {0.67, 0, 0.33}}*{1,1,0} = (0.25, -0.91, 0.67)

Irregular Wheel Spacing?

In fact to move the robot in straight lines, its wheels don't have to be spaced equally or placed on a circular layout. They can be anywhere, unevenly placed, angled and spaced. All that matters is to know the drive angle of each omni-wheel in the robot coordinate frame and use that in the exact same calculations above. This 'T' shaped robot would be kind of OK:



The steps would be: for each omni-wheel, decompose the wheel's drive axis into x and y components in the robot coordinate frame; create the forward kinematic matrix and then invert it; pick any direction to move in and decompose that into x and y components in the robot coordinate frame (set ω to 0 for no rotation); then pre-multiply the command by the robot inverse kinematic matrix to get the speed for each omni-wheel. The omni-wheels will just slip laterally on their rollers as needed to achieve the desired direction. Easy. But for rotations though, with uneven layouts, it would be necessary to allow for the distance of each omni-wheel from the robot centre or datum point. The more asymmetric the layout, the more likely it is that there will be directions that the robot doesn't easily move along, and it can become mechanically unstable. Symmetric designs are somewhat easier as regards the math (regarding rotation) and can be more stable.

Coordinate Frames

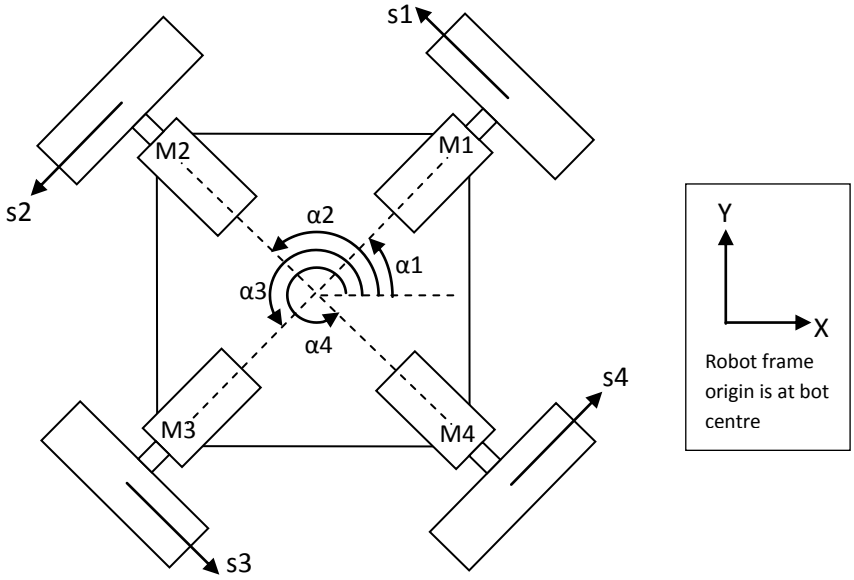
The robot coordinate frame is a datum point (or origin) with orthogonal X and Y axes, referenced to the robot chassis. It may be useful to mark this on the robot. It's OK to plan motion by reference to the robot frame. This allows the robot to plan to move in particular directions relative to its own orientation and then move accordingly. But to navigate in the world it can be useful to consider the world coordinate frame as well. This is like having an X and Y on the floor, or a map with a grid reference system. The robot frame and the world frame can be in arbitrary orientations with respect to each other. People that use RC models get used to the idea that commands have to be referenced to the coordinate system (or orientation) of the model. Some models (like Arducopter drones) have a 'super simple mode' where they use sensors to factor out the model orientation and allow control inputs to be given the same way regardless of the model orientation: In 'super simple mode' forwards on the stick will fly the model away from the pilot even if its orientation is facing in towards the pilot. This is moving in the world coordinate frame.

By knowing the relationship between the robot frame and the world frame, it's possible to plan robot moves relative to the real world. Measuring the relationship between the robot frame and the world frame needs sensors like GPS, an electronic compass, a laser-ranging system or like gyros, accelerometers or odometer wheels. It's possible to easily map between the robot frame and world frame using matrices. For a robot moving in 2D on the floor, a 3x3 matrix works to fully determine position and orientation parameters between the robot and world coordinate frames. Values in the matrix would be constantly updated by sensor readings.

Writing code so that you can easily switch between movements in the robot frame and the world frame is a good idea. For example this would make it easy to navigation to a target position while avoiding unexpected or moving obstacles. Details are not covered in this article, which only looks at motion in the robot coordinate frame.

More than Three Wheels? The Killough Drive

A symmetrical holonomic drive with four omni wheels is usually called a 'Killough Drive'. This is shown in an 'X' configuration below but it works in a similar way if it's rotated by 45° into a '+' configuration, where the motor angles would be 0°, 90°, 180° and 270°. One thing to bear in mind with more than three wheels is that some form of chassis flexibility or suspension will help to keep all the wheels touching the ground in cases where the floor isn't mirror-flat. Omni-wheel robots though, are probably somewhat better-behaved than Mecanum wheel robots when a wheel does lose contact with the ground.



In the 'X' configuration, the angle of each motor axis from the robot coordinate frame 'x' axis is:

α1 = 45°; α2 = 135°; α3 = 225°; α4 = 315°

We add π/2 to α to get the drive direction of each wheel:

w1 = α1 + π/2 = 135°

w2 = α3 + π/2 = 225°

w3 = α3 + π/2 = 315°

w4 = α3 + π/2 = 45°

We use trig to express the direction of each wheel as components in x and y relative to the robot coordinate frame:

For wheel 1: $x1 = \cos(\alpha1 + \pi/2).s1$ and $y1 = \sin(\alpha1 + \pi/2).s1$

and the same for the other three wheels. Then gather up all the individual motor contributions to the robot motion in x, in y and in ω:

$x = x1 + x2 + x3 + x4$; $y = y1 + y2 + y3 + y4$; $\omega = s1 + s2 + s3 + s4$

To use the summed x and y contributions we need to substitute in the trig expressions we worked out that express the x and y contributions of each motor in terms of 'α' and 's' to get:

$x = \cos(\alpha1 + \pi/2).s1 + \cos(\alpha2 + \pi/2).s2 + \cos(\alpha3 + \pi/2).s3 + \cos(\alpha4 + \pi/2).s4$

$y = \sin(\alpha1 + \pi/2).s1 + \sin(\alpha2 + \pi/2).s2 + \sin(\alpha3 + \pi/2).s3 + \sin(\alpha4 + \pi/2).s4$

Or better yet, write the x, y and ω contributions as a matrix equation:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} \cos(\alpha 1 + \pi/2) & \cos(\alpha 2 + \pi/2) & \cos(\alpha 3 + \pi/2) & \cos(\alpha 4 + \pi/2) \\ \sin(\alpha 1 + \pi/2) & \sin(\alpha 2 + \pi/2) & \sin(\alpha 3 + \pi/2) & \sin(\alpha 4 + \pi/2) \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \\ s4 \end{pmatrix}$$

We can write this in a more simplified way as $\underset{v}{\rightarrow} = (M) \underset{s}{\rightarrow}$

In order to navigate the robot we need to rearrange the matrix equation for $\underset{s}{\rightarrow}$

To do this we need to calculate the inverse of matrix M and multiply it into both the right and left of the equation. We can write the forward matrix in Wolfram Alpha plaintext notation in order to invert it with a mouse-click, via http://www.wolframalpha.com

invert {{cos(45+90), cos(135+90), cos(225+90), cos(315+90)}, {sin(45+90), sin(135+90), sin(225+90), sin(315+90)}, {1, 1, 1, 1}}

This gives the inverse kinematic matrix M^{-1}

$$M^{-1} = \frac{1}{4} \begin{pmatrix} -\sqrt{2} & \sqrt{2} & 1 \\ -\sqrt{2} & -\sqrt{2} & 1 \\ \sqrt{2} & -\sqrt{2} & 1 \\ \sqrt{2} & \sqrt{2} & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -0.35 & 0.35 & 0.25 \\ -0.35 & -0.35 & 0.25 \\ 0.35 & -0.35 & 0.25 \\ 0.35 & 0.35 & 0.25 \end{pmatrix}$$

In Wolfram Alpha plaintext this is:

M^{-1} = {{-0.35, 0.35, 0.25}, {-0.35, -0.35, 0.25}, {0.35, -0.35, 0.25}, {0.35, 0.35, 0.25}}

Now we have M^{-1} we can decide what robot motion we want in x, y and ω and calculate the motor speeds to carry out the motion. For example to move forward in the y direction, (x, y, w) = (0,1,0) we multiply the inverse matrix into that to determine the necessary motor speeds. This would be done in the robot micro-controller, but in Wolfram Alpha it would be:

{{-0.35, 0.35, 0.25}, {-0.35, -0.35, 0.25}, {0.35, -0.35, 0.25}, {0.35, 0.35, 0.25}}*{0,1,0}

= (0.35, -0.35, -0.35, 0.35) (note -ve is CCW rotation and +ve is CW rotation)

or to move at 45 degrees forward and right (x, y, w) = (1,1,0)

{{-0.35, 0.35, 0.25}, {-0.35, -0.35, 0.25}, {0.35, -0.35, 0.25}, {0.35, 0.35, 0.25}}*{1,1,0}

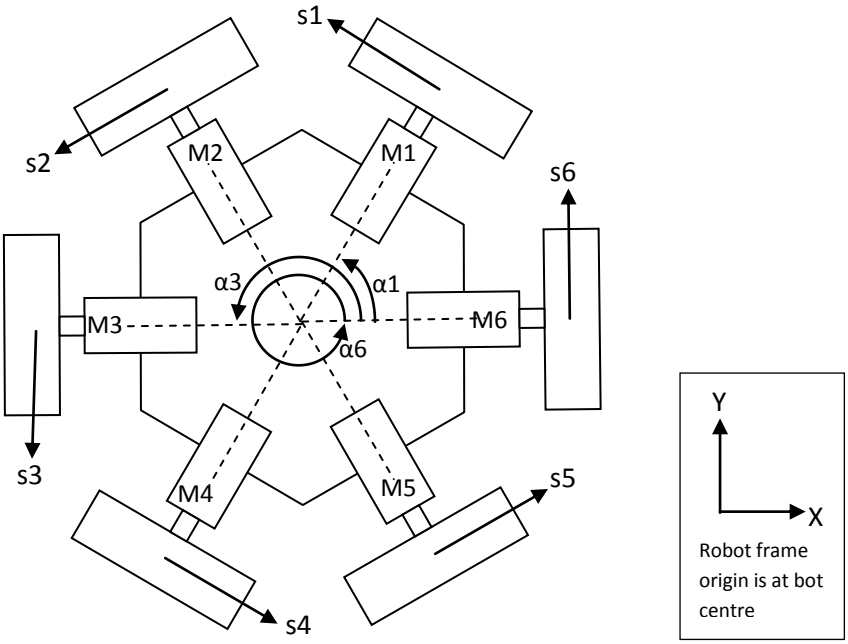
= {0, -0.7, 0, 0.7}

Just by looking at the robot diagram above, it's obvious that these determined speed settings would move the robot in the desired direction. This is one of the good things about running some examples in Wolfram Alpha before getting into the micro-controller coding. By choosing some careful test cases that would make some robot wheels turn in particular directions and other wheels have zero rotation, you can see if the math is likely correct before getting into the process of debugging microcontroller code.

Even More Wheels? Hexa-bot

Motors and wheels can only carry so much weight. One solution is to build a bigger robot. This would mean buying bigger motors, bigger wheels, more powerful motor controllers, bigger batteries

etc. Another way is just to add more wheels. It's easy to add more of the same wheels, and it's just more of the same regarding the math.



The motor angles are α1 = 60°; α2 = 120°; α3 = 180°; α4 = 240°; α5 = 300°; α6 = 0°

The wheel drive angles are found by adding 90° or π/2 added onto the motor angle.

The matrix equation that expresses the contribution of each wheel to the robot motion is:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} \cos(150) & \cos(210) & \cos(270) & \cos(330) & \cos(30) & \cos(90) \\ \sin(150) & \sin(210) & \sin(270) & \sin(330) & \sin(30) & \sin(90) \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \\ s4 \\ s5 \\ s6 \end{pmatrix}$$

We can use Wolfram Alpha to invert the forward kinematic matrix:

invert {{cos(150), cos(210), cos(270), cos(330), cos(30), cos(90)}, {sin(150), sin(210), sin(270), sin(330), sin(30), sin(90)}, {1, 1, 1, 1, 1, 1}}

This gives the inverse kinematic matrix M^{-1}

$$M^{-1} = \frac{1}{6} \begin{pmatrix} -\sqrt{3} & 1 & 1 \\ -\sqrt{3} & -1 & 1 \\ 0 & -2 & 1 \\ \sqrt{3} & -1 & 1 \\ \sqrt{3} & 1 & 1 \\ 0 & 2 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -0.29 & 0.17 & 0.17 \\ -0.29 & -0.17 & 0.17 \\ 0 & -0.33 & 0.17 \\ 0.29 & -0.17 & 0.17 \\ 0.29 & 0.17 & 0.17 \\ 0 & 0.33 & 0.17 \end{pmatrix}$$

With this inverse matrix we can command the robot to move in given directions, using a matrix multiplication to determine the motor speed settings that must be applied in order to accomplish the required motion.

For example, to move rightward in the +x direction, (x, y, w) = (1,0,0)

This would be multiplied by M^{-1} in the controller code, but we can test it in Wolfram Alpha:

{{-1/(2 Sqrt[3]), 1/6, 1/6}, {-1/(2 Sqrt[3]), -1/6, 1/6}, {0, -1/3, 1/6}, {1/(2 Sqrt[3]), -1/6, 1/6}, {1/(2 Sqrt[3]), 1/6, 1/6}, {0, 1/3, 1/6}}*{1,0,0}

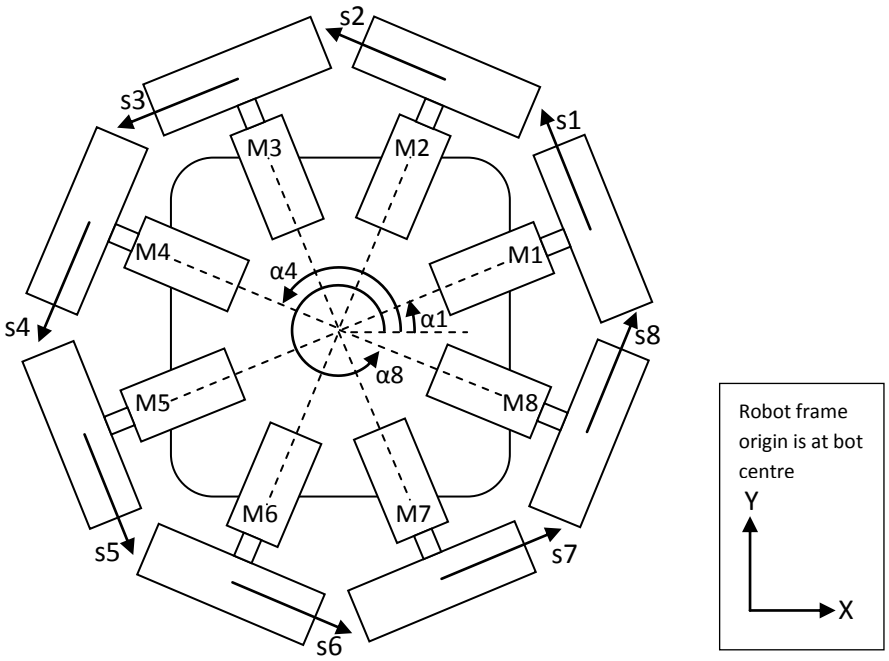
$$= \left(-\frac{1}{2\sqrt{3}}, \quad -\frac{1}{2\sqrt{3}}, \quad 0, \quad \frac{1}{2\sqrt{3}}, \quad \frac{1}{2\sqrt{3}}, \quad 0\right)$$

= (-0.29, -0.29, 0, 0.29, 0.29, 0)

Looking at these six motor speeds and at the robot diagram, it's clear that the robot would move in the correct direction given these speed settings.

Octo-Omni-Bot?

Eight omni-wheels is a lot. This would be a fairly unique configuration. With this many wheels the math needs to be right or the wheels could end up fighting each other.



The math is no more complex than for bots with fewer wheels. The motor angles in this configuration are:

α1 = 22.5°; α2 = 67.5°; α3 = 112.5°; α4 = 157.5°; α5 = 202.5°; α6 = 247.5°; α7 = 292.5°; α8 = 337.5°

The wheel drive angles are derived by adding 90° or π/2 onto the motor angle.

The matrix equation that expresses the contribution of each wheel to the robot motion is:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} \cos(112.5) & \cos(157.5) & \cos(202.5) & \cos(247.5) & \cos(292.5) & \cos(337.5) & \cos(22.5) & \cos(67.5) \\ \sin(112.5) & \sin(157.5) & \sin(202.5) & \sin(247.5) & \sin(292.5) & \sin(337.5) & \sin(22.5) & \sin(67.5) \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \\ s4 \\ s5 \\ s6 \\ s7 \\ s8 \end{pmatrix}$$

We can use Wolfram Alpha to invert the forward kinematic matrix. But in this case we manually reduce the sin and cos terms to make the expression a bit shorter. So this:

invert {{cos (112.5), cos(157.5), cos(202.5), cos(247.5), cos(292.5), cos(337.5), cos(22.5), cos(67.5)}, {sin (112.5), sin(157.5), sin(202.5), sin(247.5), sin(292.5), sin(337.5), sin(22.5), sin(67.5)}, {1, 1, 1, 1, 1, 1, 1, 1}}

gets reduced to this:

invert {{.924, .383, -.383, -.924, -.924, -.383, .383, .924}, {-.383, -.924, -.924, -.383, .383, .924, .924, .383}, {1, 1, 1, 1, 1, 1, 1, 1}}

Which in turn gives the inverse kinematic matrix M^{-1}

$$M^{-1} = \begin{pmatrix} 0.231 & -0.096 & 0.125 \\ 0.096 & -0.231 & 0.125 \\ -0.096 & -0.231 & 0.125 \\ -0.231 & -0.096 & 0.125 \\ -0.231 & 0.096 & 0.125 \\ -0.096 & 0.231 & 0.125 \\ 0.096 & 0.231 & 0.125 \\ 0.231 & 0.096 & 0.125 \end{pmatrix}$$

With this inverse matrix we can command the robot to move in any given direction, using a matrix multiplication to determine the motor speed settings that must be applied in order to accomplish the required motion. All the wheels will help, so long as we got the math right.

For example, let's drive the robot on a bearing of 22.5°. If we do this we would expect motors M1 and M5 to have zero rotation and motors M3 and M7 to turn fastest. Motors M2, M3, M4 should have a negative rotation and motors M6, M7, M8 should have a positive rotation (look at the robot diagram!)

We need to convert the heading of 22.5° into x and y components. The x component is cos(22.5) and the y component is sin(22.5), so this is 0.383 and 0.924 respectively. We will set the rotation ω to zero here.

To drive forwards and right on a heading 22.5°, (x, y, w) = (0.383, 0.924, 0)

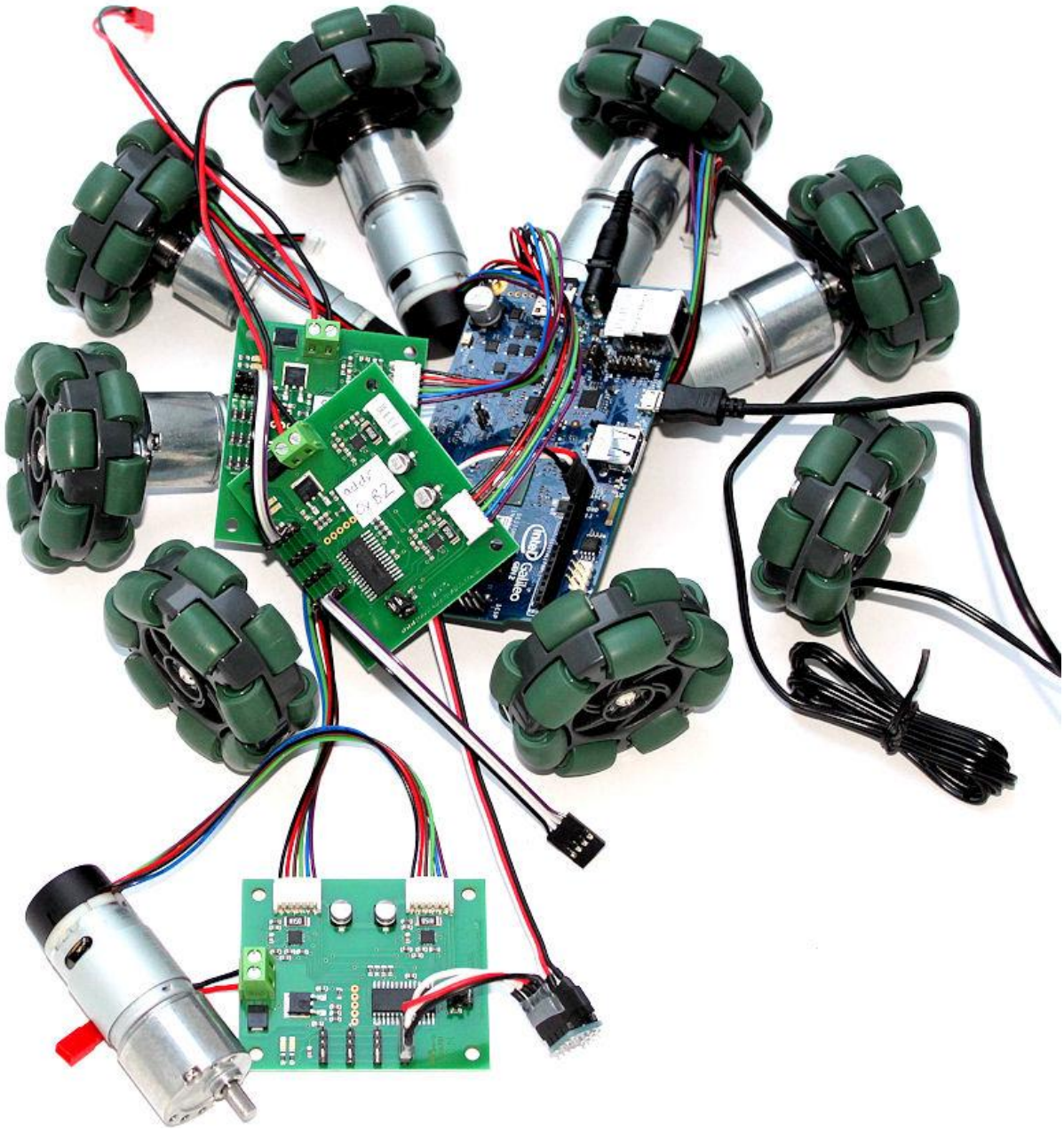
This would be multiplied by M^{-1} in the controller code, but we can test it in Wolfram Alpha:

{{.231, -.096, .125}, {.096, -.231, .125}, {-.096, -.231, .125}, {-.231, -.096, .125}, {-.231, .096, .125}, {-.096, .231, .125}, {.096, .231, .125}, {.231, .096, .125}}*{.383,.924,0}

$$= (0, \quad -0.177, \quad -0.250, \quad -0.177, \quad 0, \quad 0.177 \quad 0.250 \quad 0.177)$$

It is clear that the speeds came out as we expected. Looking at the eight motor speeds and at the robot diagram, it's clear that the robot would move in the correct direction given these speed settings. The octo version is under construction with an Intel Galileo Mk2 controller; Devantech EM25 motor drives; Devantech EMG30 motors with quadrature shaft encoders and; using modified Vex omni-wheels. The Vex omni-wheels have custom stainless hubs grafted into them so they fit nicely on the 5mm shafts of the EMG30 motors.

Raspberry Pi (especially the new Model 2 B version) would also be a great controller for this type of robot. It probably depends whether you prefer coding in Arduino 'sketches' (for Galileo or any Arduino type controller) or Python (for Raspberry Pi).



Prototype Octo-Omni-Bot - under construction