HOLONOMIC DRIVE EQUATIONS AND DERIVATIONS

K. LOUX, FIRST ROBOTICS TEAM 3167

1. Introduction and Motivation

This document outlines the physics behind a mecanum wheeled (holonomic drive) robot. The motivation for generating this document arose from a need to have a clearly documented derivation of these equations at hand while writing the code that handles the holonomic drive calculations. At first a hand-written derivation was sufficient, but it quickly became clear that a more formal derivation with consistent notation would go a long way. This, along with the ease with which an electronic copy could be distributed and stored for future use, made the decision to create this document an easy one.

This derivation was kept generic on purpose to increase the likelihood that these equations can be used in the future. For example, the rolling elements on the mecanum wheels do not necessarily need to be at a 45 degree angle to the wheel's axis of rotation. If there is a need to design a holonomic drive system that is capable of going faster in one direction than another, custom wheels with a 60 degree roller angle can be fabricated, and these equations will still apply. Or, if omni wheels are used instead of mecanum wheels, just set the angle of the rolling element to 90 degrees, and these equations still apply. The math presented here treats each wheel independently, so combinations of different wheels can be used easily.

In addition, the assumption that there are four wheels was removed. It is necessary to use at least three wheels in order to constrain the robot's three degrees of freedom, but this document presents calculations that support a theoretically unlimited number of wheels.

1.1. Coordinate Systems.

This document assumes that the robot coordinate system (fixed to the body of the robot) has x positive to the right, y positive forward and z positive up (so that it obeys the right-hand rule) and has its origin at ground level. Although it is not a requirement in order for the math to work, one will likely find it convenient to choose a point under the center of the robot, approximately equidistant from all wheels.

2. Kinematics

How fast should each wheel spin in order to achieve the desired robot motion? This section will describe the process for calculating the required wheel speeds (from which the motor speeds can be calculated using the gear ratio). The derivation begins by identifying the parameters required to define the robot system.

Date: April 5, 2011.

Key words and phrases. FIRST Robotics, mecanum wheels.

2.1. **Assumptions.** Although this derivation is kept general as to apply to a wide variety of applications, there are some assumptions that must be obeyed in order for the model to remain valid. These are:

- Planar Motion. The robot must move only within one spatial plane. The robot is free to rotate and translate within the xy-plane, but if the robot rolls over a bump, the math is not guaranteed to provide the expected response.
- No Wheel Slip. No dragging or spinning wheels due to excessive torque or uncoordinated control (assumed perfect velocity control of each wheel). It is easy to design a robot that violates this assumption, so how is this requirement handled? The commanded velocity can be modulated such that it obeys an acceleration limit, and since acceleration is proportional to torque, the torque applied to each wheel is effectively limited. If a proper acceleration limit is used, this assumption will hold true.
- Wheels Rotate About Horizontal Axes. Wheels that are inclined relative to vertical will not be modeled correctly with the math presented here (but this should provide enough detail such that this feature could be added, if desired).
- Wheels Rotate About Fixed Axes. Non-steerable wheels are assumed. If a robot is designed with mecanum wheels, there is a good chance this will hold true. The matrix defining the robot configuration is assumed to be constant.
- Rolling Elements Are Infinitely Distributed About the Wheel Perimeter. This assumption is always violated in the real-world, but the resulting math is assumed to be close enough (not proven here, but experience shows that it is true).

2.2. **List of Symbols.** Below are the symbols used throughout this derivation. Although defined here, the same symbols will be used in future sections which expand on this derivation. Be sure to use consistent units when performing calculations.

n	Number of driven wheels
$\vec{p}_i$	2D Vector describing the location of the i^{th} wheel
$\hat{a}_i$	2D Unit vector describing the i^{th} wheel's axis of rotation
$\hat{w}_i$	2D Unit vector along the direction in which the i^{th} wheel has control
β_i	Angle between the rolling elements of the i^{th} wheel and $\hat{a}_i$
ω_i	Rotational velocity of the i^{th} wheel
r_i	Radius of the i^{th} wheel
$\dot{x}$	Robot velocity in the x-direction
$\dot{y}$	Robot velocity in the y-direction
$\dot{\theta}$	Robot rotational velocity
$\vec{v}$	3-Component robot velocity vector
$\vec{v}_i$	2D Velocity of the point representing the location of the i^{th} wheel
v_{c_i}	Velocity of the i^{th} wheel projected along $\hat{w}_i$
$\mathbf{R}$	Matrix mapping the robot velocity to wheel velocity
c_{x_i}	Coefficient mapping $\dot{x}$ to ω_i
c_{y_i}	Coefficient mapping $\dot{y}$ to ω_i
c_{θ_i}	Coefficient mapping $\dot{\theta}$ to ω_i
ω_{max_i}	Maximum rotational velocity of the i^{th} wheel
f	Scaling factor for blending high-speed multi-axis motion

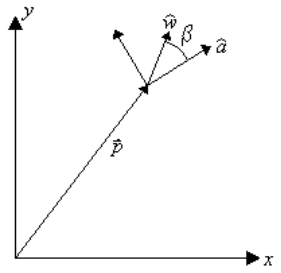

FIGURE 1. Schematic used to derive kinematic equations

2.3. **Transforming Robot Velocity to Wheel Velocity.** The robot velocity vector is defined in Eq. (1).

$$\vec{v} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{1}$$

A schematic showing the vectors describing the location and orientation of a wheel is provided in Figure 1.

From Eq. (1) and the location of each wheel $\vec{p}_i$, desired velocity of each wheel can be determined. The rotation can be resolved into velocity using the relationship $\vec{v} = \omega \times \vec{p}$. Note that the vector $\vec{p}_i$ provides the location of the point where the i^{th} wheel contacts the ground.

$$\vec{v}_i = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \vec{\dot{\theta}} \times \vec{p}_i = \begin{bmatrix} \dot{x} - \dot{\theta} p_{y_i} \\ \dot{y} + \dot{\theta} p_{x_i} \end{bmatrix} \tag{2}$$

Due to the wheel's rolling elements, the wheel can be though of as controlling velocity in one direction, while being unable to control velocity at all perpendicular to that direction. The direction in which the wheel can control velocity is the direction along the axis of the wheel's rolling element that is touching the ground. This direction depends only on the angle of the roller element relative to the wheel's axis of rotation and the direction of the wheel's axis of rotation. Note that when defining the angle of the roller elements, the angle must be for the element in contact with the ground, not for the element seen when viewing the wheel from the top. This is important because mecanum wheels are handed, meaning that wheels with a $+45$ degree angle are different from wheels with a -45 degree angle. Sometimes wheels are said to be right or left handed. This is a misnomer, as typically a four-wheeled robot will require one wheel of each hand on a particular side.

To calculate $\hat{w}$ for each wheel, apply Eq. (3). This is a 2D counter-clockwise rotation matrix applied to the direction of the wheel's rotation axis. Note that $\hat{a}$ depends on the direction defined as positive for the rotation of the wheel and the right-hand rule.

$$\hat{w}_i = \begin{bmatrix} \cos\beta_i & -\sin\beta_i \\ \sin\beta_i & \cos\beta_i \end{bmatrix} \hat{a}_i \tag{3}$$

Because the wheel can only control velocity in one direction, it is necessary to take the projection of the desired wheel velocity along the direction in which the

wheel can control its own velocity. By doing this, the component of the velocity that is required to be generated by the wheel is extracted. The wheel can't control velocity in one direction, so it is assumed that the velocity in the uncontrolled direction will be correct (it is assumed that other wheels generate this component), and when added to the controlled velocity, the wheel's velocity will also be correct. The process of ensuring the validity of this assumption is explored later on in this section.

A projection is closely related to a dot product. The derivation of the math behind it is beyond the scope of this document, but for the interested reader, an Internet search for "Vector Projection" is suggested.

Eq. (4) shows the projection of the i^{th} wheel's translational velocity, $\vec{v}_i$ along it's controllable direction, $\hat{w}_i$. Note that there is no division by the magnitude of the direction vector because it is a unit vector, and by definition has a magnitude of 1. The result of this operation is a scalar, but it is still necessary to associate it with a direction. The direction in which this velocity acts is $\hat{w}_i$.

$$v_{c_i} = \vec{v}_i \cdot \hat{w}_i \tag{4}$$

With this velocity, the required rotation rate of the wheel can be determined. The relationship between rotation rate and translational velocity at the wheel perimeter is $\omega = v/r$. The next step is finding a unit vector describing the direction of controllable velocity for the wheel. Given a unit vector for the direction of the wheel's controllable velocity, another dot product could be applied, but instead, given an angle, it is more convenient to use trigonometry. Also note that a positive rotation of the wheel (according to the right-hand rule) will result in a negative robot velocity. A negative sign is included here to account for this effect. Doing this yields Eq. (5).

$$\omega_i = -\frac{v_{c_i}}{r_i \sin\beta_i} \tag{5}$$

By starting with Eq. (5) and making substitutions according to Eq. (2), Eq. (3) and Eq. (4), this relationship can be expanded so that it maps robot velocity, $\vec{v}$, into wheel rotational velocity.

$$\omega_i = \frac{(\hat{a}_{y_i}\sin\beta_i - \hat{a}_{x_i}\cos\beta_i)\left(\dot{x} - \dot{\theta}p_{y_i}\right) - (\hat{a}_{x_i}\sin\beta_i + \hat{a}_{y_i}\cos\beta_i)\left(\dot{y} + \dot{\theta}p_{x_i}\right)}{r_i \sin\beta_i} \tag{6}$$

If the equations are expanded and terms collected, this equation can be manipulated into a form that is convenient for use in a matrix. The use of matrices is what allows the math (and programming) to remain generalized and to support an arbitrary number of wheels.

Three intermediate variables, c_{x_i}, c_{y_i} and c_{θ_i} are introduced for the purpose of saving space on this page. They are created by expanding Eq. (6) and collecting terms.

$$c_{x_i} = \frac{\hat{a}_{y_i}\sin\beta_i - \hat{a}_{x_i}\cos\beta_i}{r_i \sin\beta_i} \tag{7}$$

$$c_{y_i} = -\frac{\hat{a}_{x_i}\sin\beta_i + \hat{a}_{y_i}\cos\beta_i}{r_i \sin\beta_i} \tag{8}$$

$$c_{\theta_i} = p_{x_i} c_{y_i} - p_{y_i} c_{x_i} \tag{9}$$

These coefficients are substituted into Eq. (6) to arrive at Eq. (10).

$$\omega_i = \dot{x} c_{x_i} + \dot{y} c_{y_i} + \dot{\theta} c_{\theta_i} \tag{10}$$

Since there exists one equation for each wheel ($i = 1$ to n), these terms can be placed row-by-row into a matrix. The resulting matrix will be n-by-3, where n is the number of wheels. Since the robot has three degrees-of-freedom, at least three wheels are required (i.e. there is no way to orient the controllable velocity directions for fewer than three wheels in such a way that x and y translation and planar rotation are controlled). Depending on how the wheels are configured, however, three wheels may not be enough to control all of the DOF (imagine that three identical wheels were placed along the same axis, for example). So how can a specific set of wheels and robot geometry be analyzed to determine it's ability to control all three robot degrees-of-freedom? Check the rank of the matrix. The rank is a measure of uniqueness of the rows and columns of a matrix. Because there are three DOF, the rank of the $\mathbf{R}$ matrix must be three. If the rank is less than three, it is not possible to control the robot in all three directions. Determining the rank of a matrix is outside the scope of this document, but knowing that it can be used to evaluate wheel placement is important.

$$\mathbf{R} = \begin{bmatrix} c_{x_1} & c_{y_1} & c_{\theta_1} \\ c_{x_2} & c_{y_2} & c_{\theta_2} \\ \vdots & \vdots & \vdots \\ c_{x_n} & c_{y_n} & c_{\theta_n} \end{bmatrix} \tag{11}$$

Now that the robot's $\mathbf{R}$ matrix has been created, the required wheel speeds for any robot velocity can be calculated! A clever implementation of Eq. (12) and a PID velocity loop for each wheel is that is required to start driving a holonomic robot!

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} = \mathbf{R}\vec{v} \tag{12}$$

2.4. **Determination of Maximum Speeds and Command Scaling.** When scaling input from a joystick or other input device to send as a command to the robot drive system, it is necessary to know the maximum speeds achievable in each direction. An easy way to tackle this problem is to compute the required wheel speed for a unit input in one direction. The maximum allowable wheel speed can be divided by the maximum resulting wheel speed for this input to create a scaling factor that when applied to the input (here a unit input) will yield the maximum allowable input velocity for that direction. Repeating the process for the other directions will complete the set of velocity limits.

If Eq. (12) is implemented, it is possible that the robot tracks well when moving with velocity in only one direction or low speeds, but it does not behave as expected when moving at moderate to high speeds with a combination of x, y and rotation. Why is this? Looking back to the motors, it makes sense that all motors will be

running at their maximum speed when moving at top speed in one direction. So if the robot is asked to move at full speed in the x direction and the y direction at the same time, it will ask the motors to run at twice their top speed!

How can this problem be fixed? There are many solutions to this problem, but only one options is presented here. This method is a check-and-scale approach, where first the required velocities of each wheel are calculated, then checked to see if they are above the capabilities of the motor. If they are, all of the velocities are scaled back so that the motor driving the wheel with the maximum velocity is spinning at the maximum motor speed, but the relative velocities of all of the wheels are maintained. Note that this maximum motor speed may not be (and probably is not) the no-load speed of the motor - there is no torque available at this speed, so it is wise to choose something less than this as the maximum commanded speed for each wheel.

The approach presented here requires only that the maximum wheel speed, ω_{max_i} is known prior to the robot velocity command being issued. First calculate the required wheel speeds to achieve the desired motion, then apply the process as illustrated by the psuedocode below:

```
function ScaleVelocityCommand(omega[], maxOmega[], velCmd)
{
    // Initialize the scaling factor
    f = 1

    // Determine the proper value for the scaling factor
    for i = 1 to size(omega)
    {
        if maxOmega[i] / abs(omega[i]) < f
            f = maxOmega[i] / abs(omega[i])
    }

    // Return the scaled velocity command (which still needs
    // to be transformed into wheel speeds by multiplying
    // with the R matrix
    return f * velCmd

}
```

Note that the calculation of the scaling factor is very simple if ω_{max} is the same for all wheels:

$$f = \frac{\omega_{max}}{\max\{|\omega_1|, |\omega_2|, ..., |\omega_n|\}} \tag{13}$$

2.5. **Robot Velocity Estimation From Wheel Velocities.** Another very useful calculation is the estimation of the robot's velocity based on the feedback from wheel speed sensors. This is the relationship opposite to Eq. (12).

Note that in most cases there will be more equations than unknowns (i.e. four wheels but only three robot velocities to estimate). Rather than choose three of them to solve, it is possible to use the feedback from all of the wheels to get a better estimate for the three robot velocities, much like finding the best-fit line through a series of points. This is another problem simplified by the use of matrices.

According to the rules of matrix algebra, it is not possible to simply rearrange Eq. (12) to solve for $\vec{v}$ instead of $\vec{\omega}$. It is possible, however, to perform just one additional operation to find a new matrix that does allow the mapping to occur in the reverse direction — the matrix inverse.

The inverse of a matrix is only defined for square matrices, however, and in some cases there may be more than three wheels, which makes the $\mathbf{R}$ matrix rectangular. Luckily, this can be handled by calculating the psuedoinverse of the matrix. The calculation of this matrix is outside the scope of this document, but there are a number of references on the Internet that have much more information. Numerical Recpies in C or C++ has algorithms for computing the psuedoinverse that can be implemented in the robot code, or software like MATLAB (and its `pinv` function) can be used to calculate the pseudoinverse off-line and hard-code the values afterward. The result can be used as shown in Eq. (14).

$$\vec{v} = \mathbf{R}^{-1} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} \tag{14}$$

E-mail address: kerryloux@gmail.com