Coaxial Swerve Drivetrain kinematics

How a coaxial drivetrain works in an in depth explanation

Pranav Y

Robopocalypse#8872

A coaxial swerve drive is a type of drivetrain commonly used in FRC. It consists of 2-4 swerve modules that allow the robot to move in all directions. A swerve allows for greater stability of the robot and flexibility in motion control.

---

2 Wheel Swerve Drive (with a third passive wheel)

Lets start of with a 2 wheel swerve. Consider the following image and its coordinate axes:

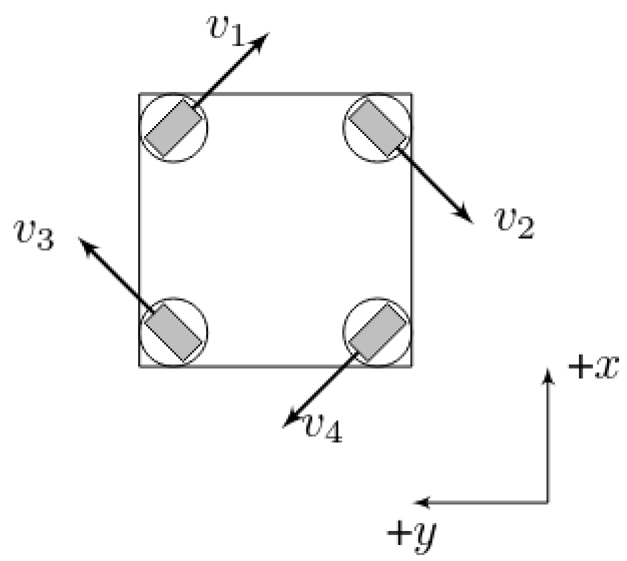

Figure 12.4: Swerve drive free body diagram

Each module has two actuators and thus two states: velocity of the module along the x axis

$$v_{1x}$$

for module 1, and velocity of the module along the y axis

$$v_{1y}$$

module 1. Since your third wheel is just there to keep the robot from tipping over, your inverse kinematics (skipping over the steps shown in the book) would be:

$$\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_{1y} \\ 0 & 1 & r_{1x} \\ 1 & 0 & -r_{2y} \\ 0 & 1 & r_{2x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

where r1x is the displacement from the center of rotation † to the wheel module along the x axis (same logic applies for other module and axis combos), vx is the chassis x velocity, vy is the chassis y velocity, and ω is the chassis angular velocity. You could multiply that out to get four separate equations for the velocity components

$$\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \end{bmatrix} = v_x \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + v_y \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} + \omega \begin{bmatrix} -r_{1y} \\ r_{1x} \\ -r_{2y} \\ r_{2x} \end{bmatrix}$$

$$\begin{aligned} v_{1x} &= v_x - \omega r_{1y} \\ v_{1y} &= v_y + \omega r_{1x} \\ v_{2x} &= v_x - \omega r_{2y} \\ v_{2y} &= v_y + \omega r_{2x} \end{aligned}$$

but the matrix notation makes it easier to do the forward kinematics (i.e., going from wheel speeds to chassis speed for use with odometry). You invert the 4x3 matrix (linear algebra libraries have functions that make this easy) then multiply your wheel speed vector by it to get your chassis speed vector.

To convert the swerve module x and y velocity components to a velocity and heading, use the Pythagorean theorem and arctangent respectively. Here's an example for module 1.

$$\begin{aligned} v_1 &= \sqrt{v_{1x}^2 + v_{1y}^2} \\ \theta_1 &= tan^{-1}(\frac{v_{1y}}{v_{1x}}) \end{aligned}$$

In java that would be

```java
double v1 = Math.hypot(v1x, v1y);
double heading1 = Math.atat2(v1y,v1x);
```

---

4 Wheel Swerve Drive

THE IMPLEMENTATION

Let FWD, STR, and RCW be the Forward, Strafe Right, and Rotate Clockwise driver commands, respectively.

FWD = -Y (vehicle goes forward when joystick is pushed forward)

STR = X (vehicle strafes right when joystick is pushed to the right)

RCW = Z (vehicle rotates clockwise when joystick is twisted clockwise)

in Java

```
double FWD = -gamepad1.left_stick_y;
double STR = gamepad1.left_stick_x;
double RCW = gamepad1.right_stick_x;
```

… this can easily be replaced by another interface which provides driver inputs for the 3 degrees of freedom

RCW will be scaled in the formulas below to create a +1 (or –1) wheel speed command at each wheel for pure rotation when RCW equals +1 (or –1) and FWD & STR both equal zero. If desired, multiply RCW by some fraction to reduce the gain of the rotate command.

If your goal is to make the robot operate in field-centric mode, apply the gyro angle.

$$\theta$$

(in java)

```
double degrees = imu.getHeading();
```

is the gyro angle, measured clockwise from the zero position (zero normall being set to straight downfield). Take the following math into account. F and S stand for Forward and Strafe

$$F = F \times \cos(\theta) + S \times \sin(\theta)$$
$$S = -F \times \sin(\theta) + S \times \cos(\theta)$$

In Java that would be

```
double temp = FWD*Math.cos(imu.getHeading()) + STR*Math.sin(imu.getHeading());
double STR = -FWD*Math.sin(imu.getHeading()) + STR*Math.cos(imu.getHeading());
FWD = temp;
```

Now, convert the vehicle motion commands into wheel speed and angle commands (inverse kinematics)

Define the following constants:

L is the vehicle's wheelbase

W is the vehicle's trackwidth

R the hypotenuse of them

$$R = \sqrt{L^2 + W^2}$$

```
double L = 8; //the vehicle's wheelbase
double W = 8; //the vehicle's trackwidth
double R = Math.hypot(L,W);
```

It doesn't matter what measurement units are used for L and W, since only ratios will be used in the calculations

To simply the math lets declare 4 variables

A, B, C, and **D**

in Java

```
double A;
double B;
double C;
double D;
```

Perform the following calculations for each new set of FWD, STR, and RCW commands:

$$\begin{gathered} A = STR - RCW \times (L/R) \\ B = STR + RCW \times (L/R) \\ C = FWD - RCW \times (W/R) \\ D = FWD + RCW \times (W/R) \end{gathered}$$

```
A = STR - RCW * (L/R);
B = STR + RCW * (L/R);
C = FWD - RCW * (W/R);
D = FWD + RCW * (W/R);
```

Now lets calculate the wheel speeds

$$\begin{gathered} ws1 = \sqrt{B^2 + C^2} \\ ws2 = \sqrt{B^2 + D^2} \\ ws3 = \sqrt{A^2 + D^2} \\ ws4 = \sqrt{A^2 + C^2} \end{gathered}$$

In Java that would be

```
double ws1 = Math.hypot(B,C);
double ws2 = Math.hypot(B,D);
double ws3 = Math.hypot(A,D);
double ws4 = Math.hypot(A,C);
```

Now, lets calculate angles

$$wa1 = \arctan 2(B, C) \times \frac{180}{\pi}$$

$$wa2 = \arctan 2(B, D) \times \frac{180}{\pi}$$

$$wa3 = \arctan 2(A, D) \times \frac{180}{\pi}$$

$$wa4 = \arctan 2(A, C) \times \frac{180}{\pi}$$

In Java:

```
double wa1 = Math.atan2(B,C)* 180/Math.pi;
double wa2 = Math.atan2(B,D)* 180/Math.pi;
double wa3 = Math.atan2(A,D)* 180/Math.pi;
double wa4 = Math.atan2(A,C)* 180/Math.pi;
```

The angles are in the range -180 to +180, measured clockwise, with zero being the straight ahead position.

The wheel speeds need to be normalized before being used, as follows:

```
double max=ws1;
if(ws2>max)
    max=ws2;
if(ws3>max)
    max=ws3;
if(ws4>max)
    max=ws4;
if(max>1){
    ws1/=max; ws2/=max; ws3/=max; ws4/=max;
}
```

The 4 wheel speeds are now in the range 0 to +1. Notice 0 to +1 and not -1 to +1, because each speed is always in the direction of the corresponding wheel angle. The 4 wheel speed and wheel angle pairs are now suitable for sending to a control algorithm which will decide how to actuate the steering and drive motors to achieve those speeds and angles for each wheel. This is a separate and interesting problem, which may involve, for example, logic to reverse the wheel speed instead of steering the wheel 180 degrees (and similarly for other steering/speed commands). This control logic is highly dependent on the limitations of the vehicle design, such as the response of the steering motor, and the fact that some vehicles may have limitless steering (coaxial or slip rings) whereas others may be steering-limited. More complex logic may also take into account the current vehicle speed and recent past history in innovative ways. Computing the necessary speed and angle errors to send to a PID control (for example) may therefore become a non-trivial task in an effort to achieve smooth and seamless control.