3/28/2011 Ether

SWERVE DRIVE

Calculate wheel speeds and wheel steering angles
for holonomic (3 degrees of freedom) control

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

So for example, with a 3-axis joystick, we might have:

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)

- OR -

With two 2-axis joysticks, we might have:

FWD = -Y1 (vehicle goes forward when joystick1 is pushed forward)

STR = X1 (vehicle strafes right when joystick1 is pushed to the right)

RCW = X2 (vehicle rotates clockwise when joystick2 is pushed to the right)

... *or any other interface that 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.

Next, if so desired, apply the gyro angle so that these become field-centric commands. θ is the gyro angle, measured clockwise from the zero position (zero normally being set to straight downfield):

```
temp  =  FWD·cos(θ) + STR·sin(θ);
STR   = -FWD·sin(θ) + STR·cos(θ);
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 = sqrt(L^2+W^2);

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

To simplify the math, define variables **A**, **B**, **C**, and **D**.

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

```
A = STR - RCW·(L/R);

B = STR + RCW·(L/R);

C = FWD - RCW·(W/R);

D = FWD + RCW·(W/R);
```

```
ws1 = sqrt(B²+C²);           wa1 = atan2(B,C)·180/pi;

ws2 = sqrt(B²+D²);           wa2 = atan2(B,D)·180/pi;

ws3 = sqrt(A²+D²);           wa3 = atan2(A,D)·180/pi;

ws4 = sqrt(A²+C²);           wa4 = atan2(A,C)·180/pi;
```

ws1..ws4 and wa1..wa4 are the wheel speeds and wheel angles for wheels 1 through 4, which are front_right, front_left, rear_left, and rear_right, respectively.

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

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

```
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.