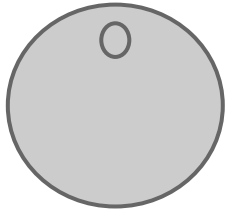
2517 Swerve Calculations

Key

A square robot with a wheel in each corner.



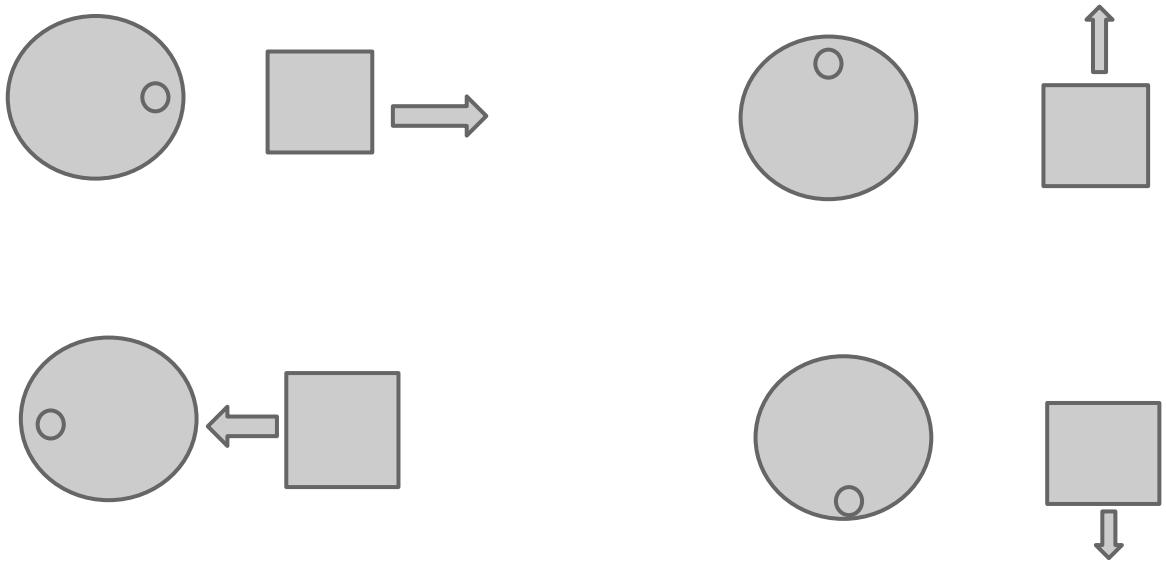
A joystick pushed forward.

The motion of the robot.

A wheel’s velocity vector.

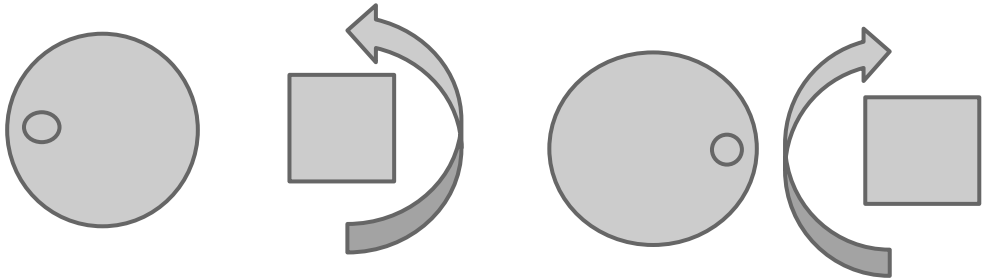
Control Scheme

The left stick controls translational movement.



Control Scheme

The right stick controls rotation.

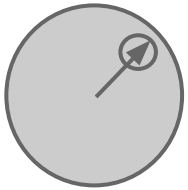


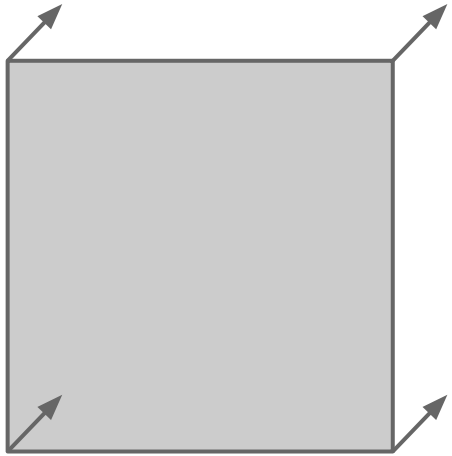
Translational Movement

If we only used the left stick, i.e. didn’t rotate, swerve would be easy. All the drive would need to do is point all the wheels in the direction that the left stick is pointing and run them all at the same speed.

Translational Movement

Example:



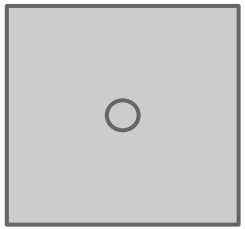


Translational Movement

The velocity vector for each wheel would simply be the vector of the joystick. So, if the left stick is at (1.00, 1.00), the velocity vector for each wheel would be <1.00, 1.00>.

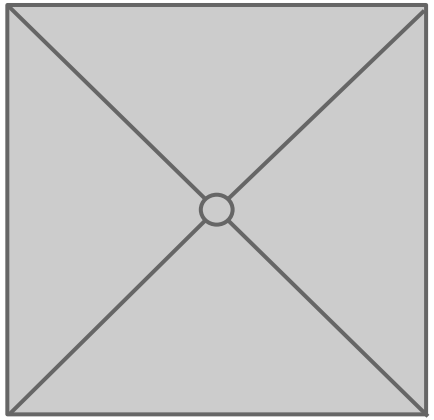
Rotation

This is a bit more complicated. First, a center of rotation has to be decided upon. This is the point that will stay stationary while the rest of the robot rotates around it. For simplicity, we will assume this is the center of the robot.



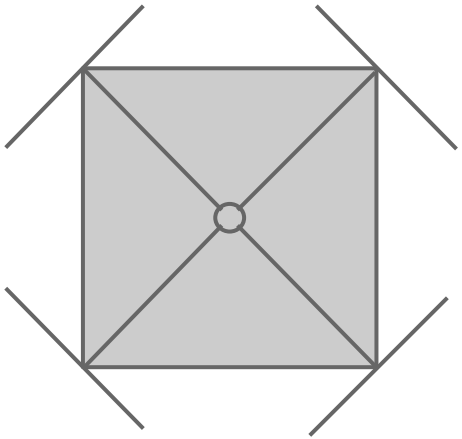
Clockwise Rotation

First, draw an imaginary line from the center of rotation to each wheel.



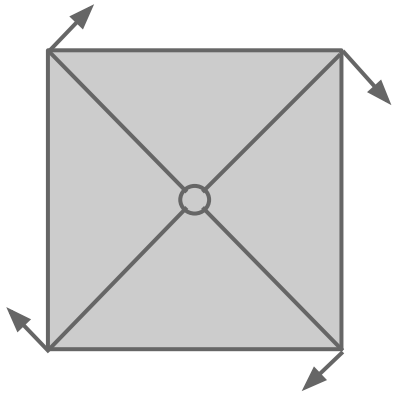
Clockwise Rotation

Then, draw lines at pi/2 radian angles from these lines from the center. These will be the orientation of the wheels to get the robot to rotate.



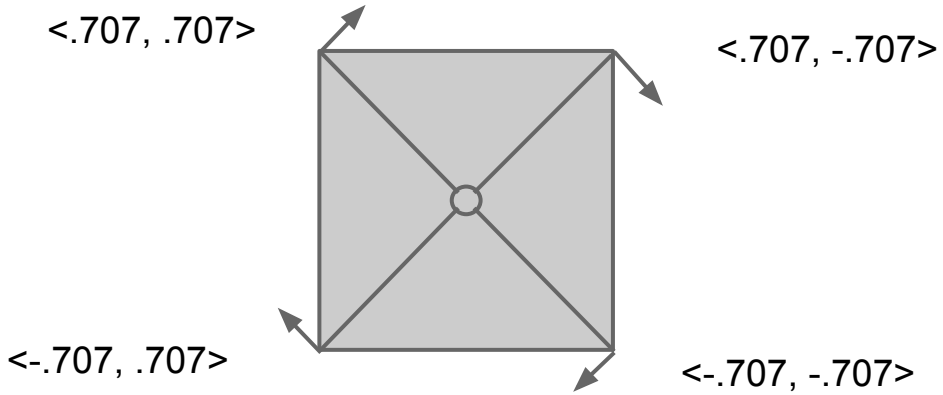
Clockwise Rotation

For clockwise rotation, we would run the wheels this way. For counterclockwise, we would simply reverse the direction.



Clockwise Rotation

Here would be the vectors for each wheel:



Clockwise Rotation

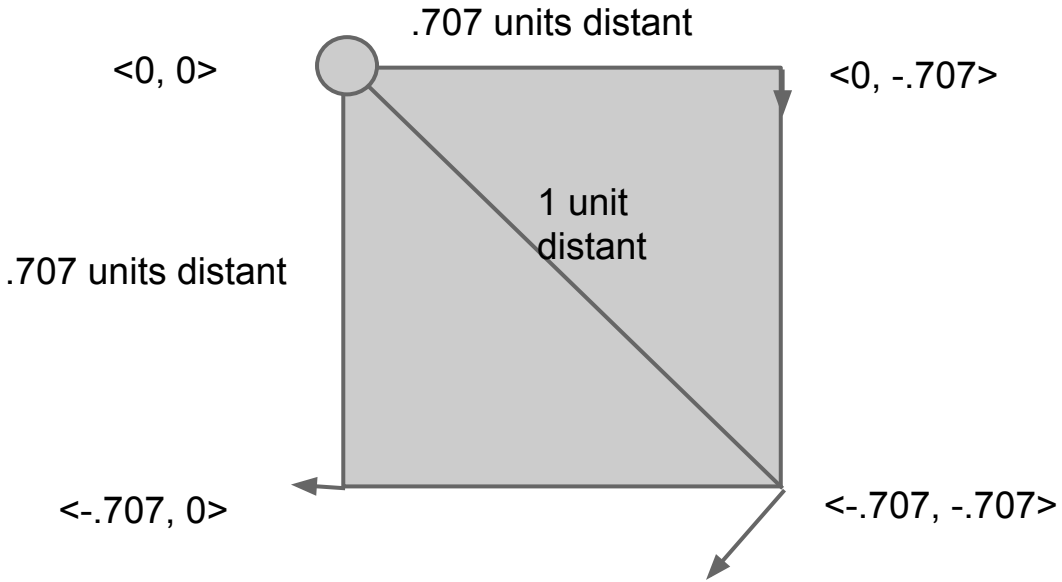
To relate this to input from the controller, we take these four vectors and multiply them by the x coordinate of the right joystick. This changes the speed of the wheels, without changing their orientation. Since the joystick returns values between -1 and 1, we can pass the resulting magnitudes of the vectors directly to the jaguars.

Rotation Around Corner

Rotation is a bit more complicated if the wheels aren’t an equal distance from the center of rotation. In this case, the furthest wheel will have a magnitude of 1, while a wheel ½ the distance of the furthest wheel would have a velocity vector magnitude of .5.

Rotation Around Corner

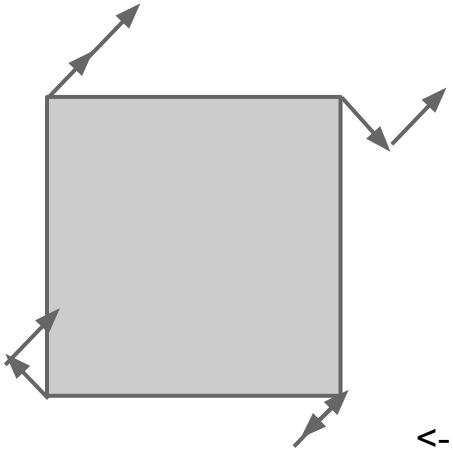
Following the above method:



Full Motion

To combine the two, we simply add the two vectors for each wheel. For instance, here is diagonal motion with rotation.

<.707, .707> + <.707, .707>



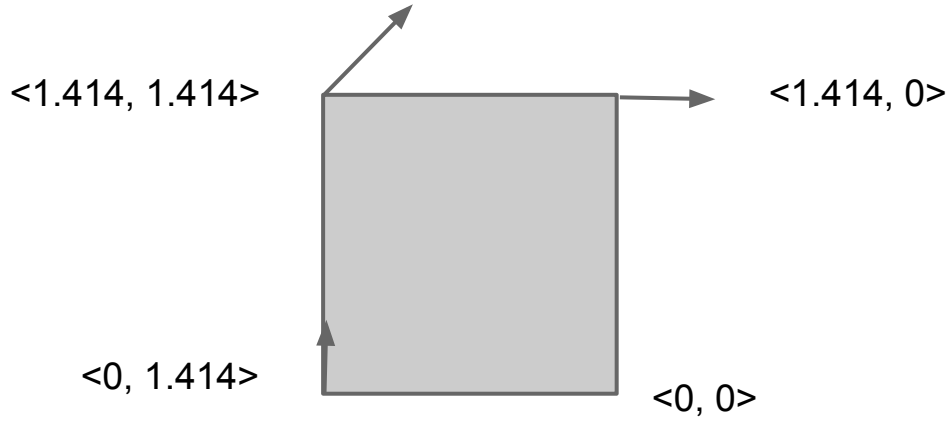
<.707, -.707> + <.707, .707>

<-.707, .707> + <.707, .707>

<-.707, -.707> + + <.707, .707>

Full Motion

Here are the sums for each set of vectors. This controller input would result in the robot rotating around the back right corner.



Full Motion

From this point, we would use one set of motors to face the wheels in the correct, and another set to move the wheels proportional to the magnitude of the vector for that wheel.

Full Motion

However, we run into an issue where we can possibly have a velocity vector with a magnitude greater than 1. To deal with this, we divide the velocities of all the wheels by the same factor. This maintains the same direction of movement, while decreasing the speed.

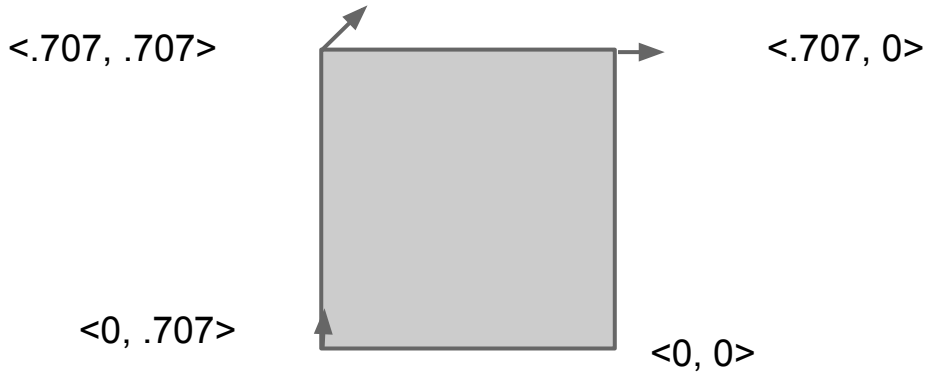
Reducing Speed

(1.414^2 + 1.414^2)^(½) = 2

To make the fastest wheel have a velocity vector magnitude of 1 while maintaining the same kind of motion, we need to divide every velocity vector by the magnitude of that largest vector.

Reducing Speed

After reining in the motors that would be moving too fast, we are left with these vectors that the robot can actually achieve.



Implementation

To see these methods in use, refer to the code at:

https://github.com/team2517/2014FRC/blob/master/MyRobot.cpp