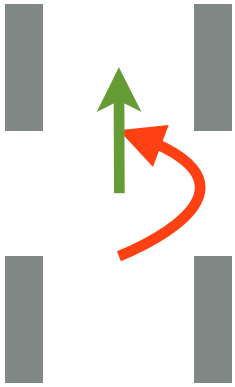
Karsten Becker

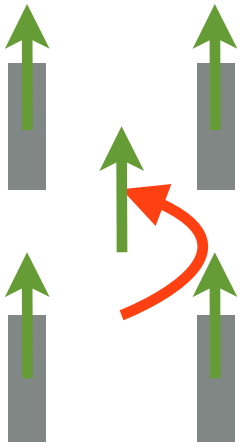
Montag, 16. Januar 2012

"KINEMATIC MODEL"

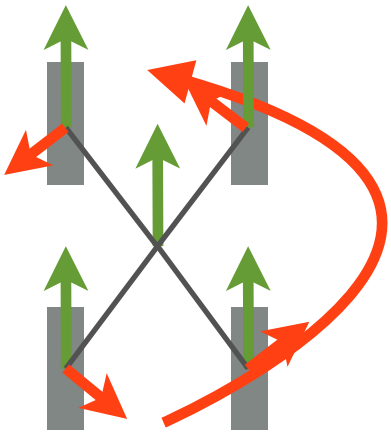
The kinematic model of our rovers is based on a very simple vector model. The idea is that one vector is indicating the speed and direction of the rover, while a circular one is indicating the rotation speed. This can be illustrated as following:



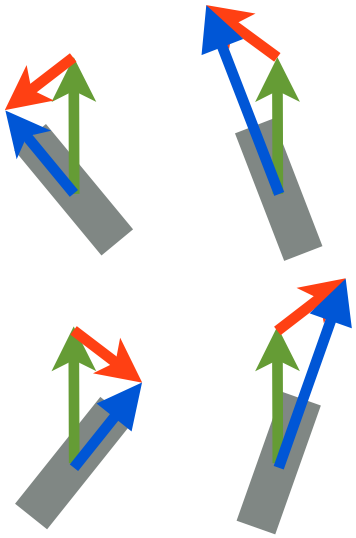
While this would be easy to implement on a rover with just one wheel, it has to be adopted for 4 wheels. This is done by translating the vectors to the wheel. This is very simple for the direction/ speed vector (green).



The translation of the rotational vector is done by creating vectors that are perpendicular to the shortest line to the center of the rover.



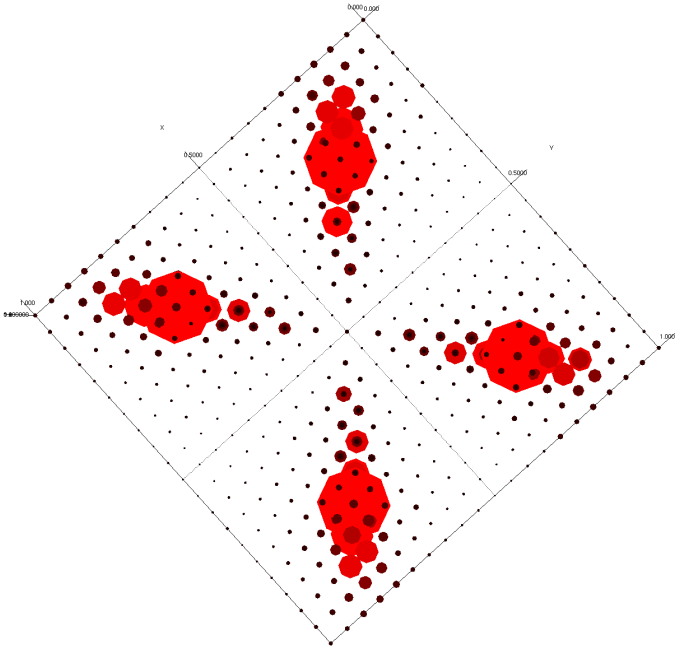
Now all that is left to do is to add those vectors. The resulting vector indicates the direction as well as the speed of each wheel.

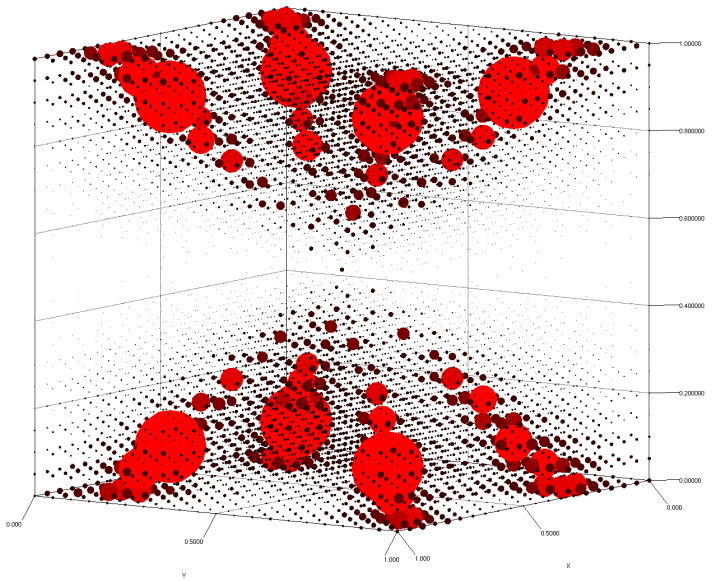


The resulting source code can be found at the end of this report.

Influences of deflection

The first obvious result that one can easily see from the source code is that a deflection does not affect the results whatsoever if the rotational compound is 0. A search through the whole steering space gives the following results for a maximum widening of 1 cm per wheel. In this diagram the rotational vector is at the z axis. The biggest spheres represent a 15° orientation mismatch under the worst circumstances of wheel deflection. It can easily be seen that the orientation mismatch can easily avoided if one of the direction vectors remains zero.





```
@Override
public byte[] compute(double y, double x, double ry) {

	// contains vectorized speeds and servo values for each wheel
	// log(String.format("vectorControl: x:%5.3f z:%5.3f ry:%5.3f", x, z,
	// ry));

	int[] pwm_dir = new int[4];
	int[] pwm_spd = new int[4];

	// calculate rotation component of the wheel's speeds
	double rfac = ry / HYPOT_ROVER;
	wheelValues[FRONTLEFT].x = -robot_width * rfac;
	wheelValues[FRONTLEFT].y = robot_length * rfac;
	wheelValues[REARLEFT].x = -robot_width * rfac;
	wheelValues[REARLEFT].y = -robot_length * rfac;
	wheelValues[REARRIGHT].x = robot_width * rfac;
	wheelValues[REARRIGHT].y = -robot_length * rfac;
	wheelValues[FRONTRIGHT].x = robot_width * rfac;
	wheelValues[FRONTRIGHT].y = robot_length * rfac;

	double maxspeed = 0.0;
	// add the linear movement vector
	for (int i = 0; i < 4; ++i) {
		wheelValues[i].x += x;
		wheelValues[i].y -= y;
		maxspeed = Math.max(maxspeed,
			Math.hypot(wheelValues[i].x, wheelValues[i].y));
	}
	for (int i = 0; i < 4; ++i) {
		// normalize all speeds to [-1..1]: Divide by the largest speed if
		// it is >1.0
		if (maxspeed > 1.0) {
			wheelValues[i].y /= maxspeed;
			wheelValues[i].x /= maxspeed;
		}

		// now the wheel speed vectors are finished.
		// calculate angles and absolute speeds.
		wheelValues[i].orientation =
			Math.atan2(wheelValues[i].y, wheelValues[i].x);
		wheelValues[i].speed = Math.hypot(wheelValues[i].x, wheelValues[i].y);
		// angles are limited to [-pi/2..pi/2] => reverse speed if necessary
		if (wheelValues[i].orientation < -M_PI_2) {
			wheelValues[i].orientation += Math.PI;
			wheelValues[i].speed *= -1.0;
		}
		if (wheelValues[i].orientation > M_PI_2) {
			wheelValues[i].orientation -= Math.PI;
			wheelValues[i].speed *= -1.0;
		}
```

```
                // calculate steering servo value
                if (wheelValues[i].orientation < 0.0) {
                        pwm_dir[i] = (int) (0x80 + (
                                (wheelValues[i].orientation * (0x80 - 0xFF)) / M_PI_2));
                } else {
                        pwm_dir[i] = (int) (0x80 + (
                        (wheelValues[i].orientation * (0 - 0x80)) / M_PI_2));
                }

                // calculate driving servo value
                pwm_spd[i] = (int) (0x80 + (60 * wheelValues[i].speed));

        }

        return sendSpeed(pwm_dir, pwm_spd);
}
```