WPILib Mecanum Control Algorithm Comparison

Jon C. Anderson

25 January 2011

The 2011 WPILib Mecanum Control Algorithm differs from the 2010 version in that the speed output in the cardinal directions are maximized and the speed output is normalized over all four motors when it exceeds 100% for any given motor to ensure the relationship between translation and rotation instead of having the SpeedController.Set method clip a motor command in excess of 100%.

2010 Mecanum Control Algorithm

The 2010 Mecanum Control Algorithm is implemented in `RobotDrive::HolonomicDrive`. The algorithm is implemented as an omni-wheel based holonomic control defined as:

$$MotorOutput_n = Velocity \times \sin(Heading - MotorOffset_n) - Rotation$$
$$SpeedController_n.Set(MotorOutput_n)$$

where *Velocity* is the commanded robot velocity percentage (aka `magnitude`), *Heading* is the commanded robot heading (aka `direction`), *MotorOffset$_n$* is the angle of the particular wheel with respect to a *Heading* of 0 degrees, and *Rotation* is the yaw-rate scalar value.

Since Mecanum wheels are best placed at 90 degrees from each other around the robot for optimal performance and simplicity of control, the algorithm is implemented as follows:

```
/*----------------------------------------------------------------------------*/
/* Copyright (c) FIRST 2008. All Rights Reserved.                             */
/* Open Source Software - may be modified and shared by FRC teams. The code   */
/* must be accompanied by the FIRST BSD license file in $(WIND_BASE)/WPILib.  */
/*----------------------------------------------------------------------------*/

void RobotDrive::HolonomicDrive(float magnitude, float direction, float rotation)
{
      float frontLeftSpeed, rearLeftSpeed, frontRightSpeed, rearRightSpeed;
      magnitude = Limit(magnitude);
      float cosD = cos((float)(direction + 45.0) * 3.14159 / 180.0);
      float sinD = cos((float)(direction - 45.0) * 3.14159 / 180.0);
      frontLeftSpeed = Limit((float)(sinD * (float)magnitude + rotation));
      rearLeftSpeed = Limit((float)(cosD * (float)magnitude + rotation));
      frontRightSpeed = Limit((float)(cosD * (float)magnitude - rotation));
      rearRightSpeed = Limit((float)(sinD * (float)magnitude - rotation));

      m_frontLeftMotor->Set(frontLeftSpeed * m_invertedMotors[kFrontLeftMotor]);
      m_frontRightMotor->Set(frontRightSpeed * m_invertedMotors[kFrontRightMotor]);
      m_rearLeftMotor->Set(rearLeftSpeed * m_invertedMotors[kRearLeftMotor]);
      m_rearRightMotor->Set(rearRightSpeed * m_invertedMotors[kRearRightMotor]);
}
```

By applying the Forward Kinematic Analysis and solution presented in Ether's *Kinematic Analysis of Four-Wheeled Mecanum Vehicle* white paper a Mecanum robot's total *Velocity*, *Heading*, and *Rotation* can be calculated. To generate a full range of of *Velocity* and *Heading* values a set of simulated joystick inputs are used with no limiting or modification. The *Velocity* and *Heading* with no *Rotation* being applied can be visualized as:

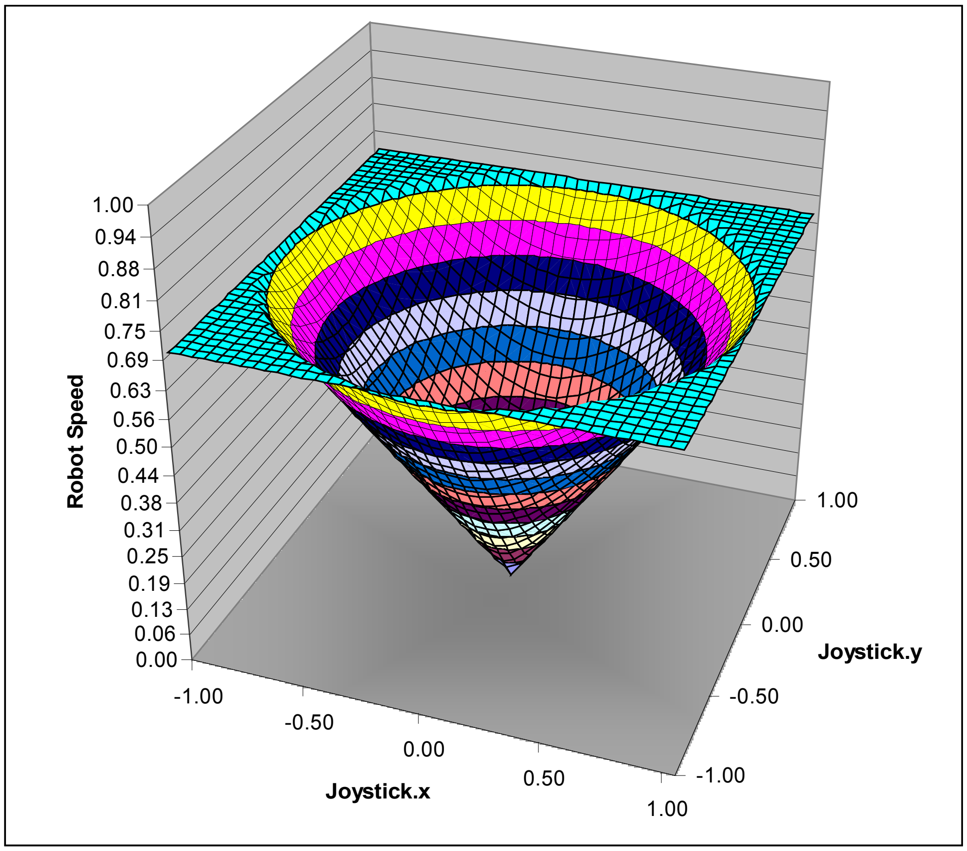

Illustration 1: 2010 Mecanum Control Algorithm Robot Speed vs. Velocity and Heading with No Rotation Applied

Note that limiting the *Velocity* to 1.00 limits the robot's total speed to $\sqrt{2}/2$ or approximately 0.71.

When applying a *Rotation* of 0.29, *Velocity* and *Heading* can be visualized as:

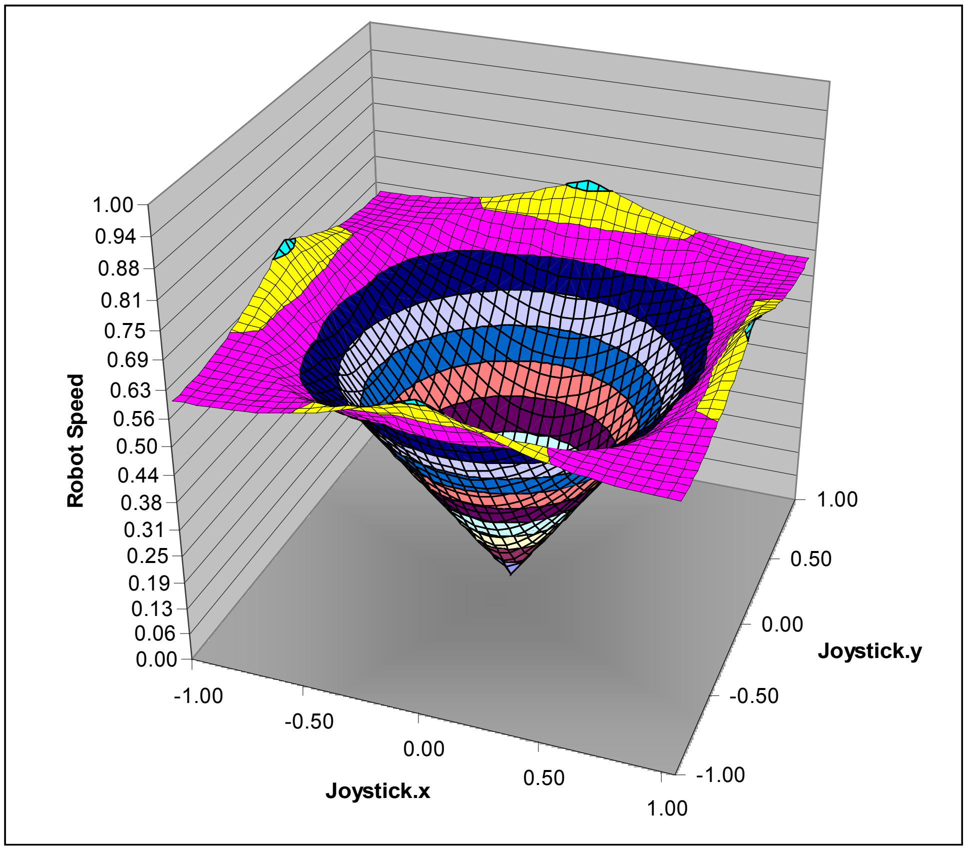

Illustration 2: 2010 Mecanum Control Algorithm Robot Speed vs. Velocity and Heading with a Rotation of 0.29

Note that the clipping applied by limiting the *MotorOutput* to the 100% of which it is capable begins to distort the robot's speed at the maximum everywhere except the cardinal directions as well as the shape of the cone which starts to become dented in along the cardinal directions as the relationship between the robot's x-axis component, y-axis component, and yaw-rate component can no longer be maintained. This results in a robot that is no longer strictly holonomic as the control of each of the robot's degrees of freedom are no longer independent of each other.

The point at which clipping occurs is at a *Velocity* where the maximum motor output exceeds $1-Rotation$ which occurs at a *Velocity* of 0.71 for a *Rotation* of 0.29 as seen in Illustration 3:

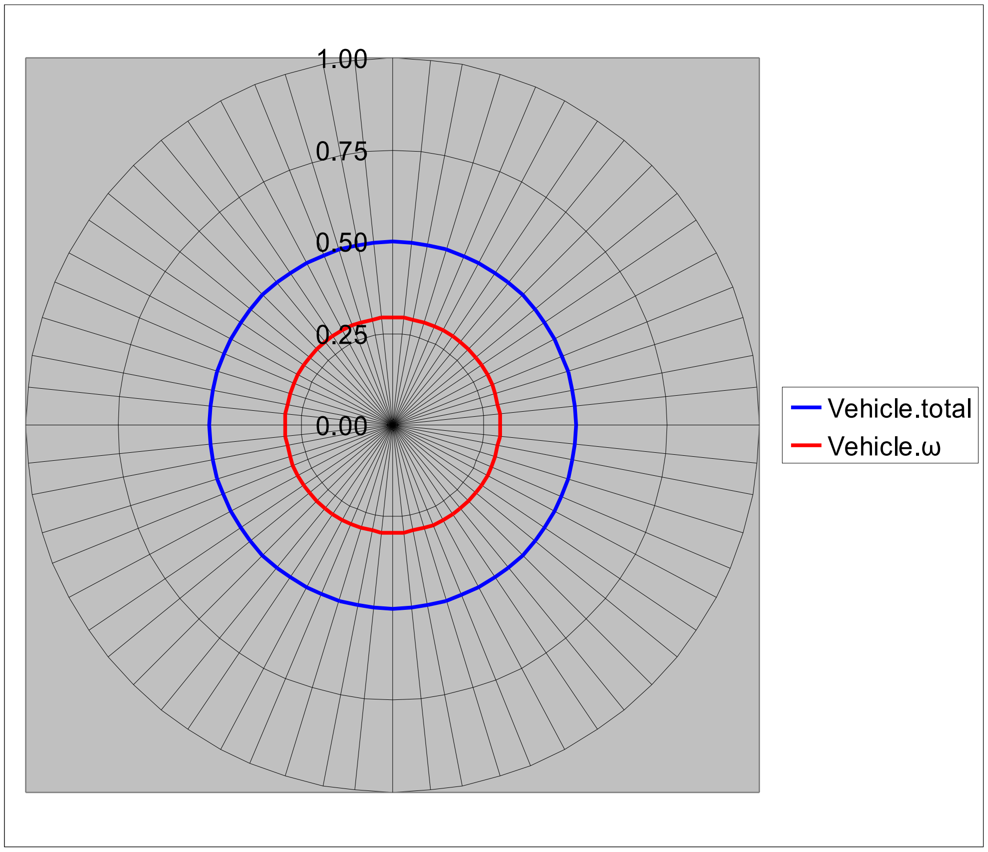

Illustration 3: 2010 Mecanum Algorithm Non-Clipping Limit

2011 Mecanum Control Algorithm

The 2011 Mecanum Control Algorithm is implemented in `RobotDrive::MecanumDrive_Polar`. The algorithm modifies the algorithm seen in 2010 so that the *Velocity* results in a maximum *MotorOutput* of 1.00 when the *Heading* is in the cardinal directions and normalizes the vector of *MotorOutput*s to maintain the relationship between each of the *MotorOutputs* and therefore *Velocity*, *Heading*, and *Rotation*. The algorithm is now defined as:

$$MotorOutput_n = Velocity \times \sqrt{2} \times \sin(Heading - MotorOffset_n) - Rotation$$
$$Normalize(MotorOutput_1, \cdots, MotorOutput_n)$$
$$SpeedController_n.Set(MotorOutput_n)$$

where the Normalize function produces a normal vector if any of the vector components exceed a value of 1.00.

The new algorithm is implemented as follows:

```
/*----------------------------------------------------------------------------*/
/* Copyright (c) FIRST 2008. All Rights Reserved.                             */
/* Open Source Software - may be modified and shared by FRC teams. The code   */
/* must be accompanied by the FIRST BSD license file in $(WIND_BASE)/WPILib.  */
/*----------------------------------------------------------------------------*/

void RobotDrive::MecanumDrive_Polar(float magnitude, float direction, float rotation)
{
      // Normalized for full power along the Cartesian axes.
      magnitude = Limit(magnitude) * sqrt(2.0);
      // The rollers are at 45 degree angles.
      double dirInRad = (direction + 45.0) * 3.14159 / 180.0;
      double cosD = cos(dirInRad);
      double sinD = sin(dirInRad);

      double wheelSpeeds[kMaxNumberOfMotors];
      wheelSpeeds[kFrontLeftMotor] = sinD * magnitude + rotation;
      wheelSpeeds[kFrontRightMotor] = cosD * magnitude - rotation;
      wheelSpeeds[kRearLeftMotor] = cosD * magnitude + rotation;
      wheelSpeeds[kRearRightMotor] = sinD * magnitude - rotation;

      Normalize(wheelSpeeds);

      UINT8 syncGroup = 0x80;

      m_frontLeftMotor->Set(wheelSpeeds[kFrontLeftMotor] *
            m_invertedMotors[kFrontLeftMotor] * m_maxOutput, syncGroup);
      m_frontRightMotor->Set(wheelSpeeds[kFrontRightMotor] *
            m_invertedMotors[kFrontRightMotor] * m_maxOutput, syncGroup);
      m_rearLeftMotor->Set(wheelSpeeds[kRearLeftMotor] *
            m_invertedMotors[kRearLeftMotor] * m_maxOutput, syncGroup);
      m_rearRightMotor->Set(wheelSpeeds[kRearRightMotor] *
            m_invertedMotors[kRearRightMotor] * m_maxOutput, syncGroup);

      CANJaguar::UpdateSyncGroup(syncGroup);

      m_safetyHelper->Feed();
}
```

By again applying the Forward Kinematic Analysis and solution presented in Ether's *Kinematic Analysis of Four-Wheeled Mecanum Vehicle* white paper the *Velocity* and *Heading* with no *Rotation* being applied can be visualized as:

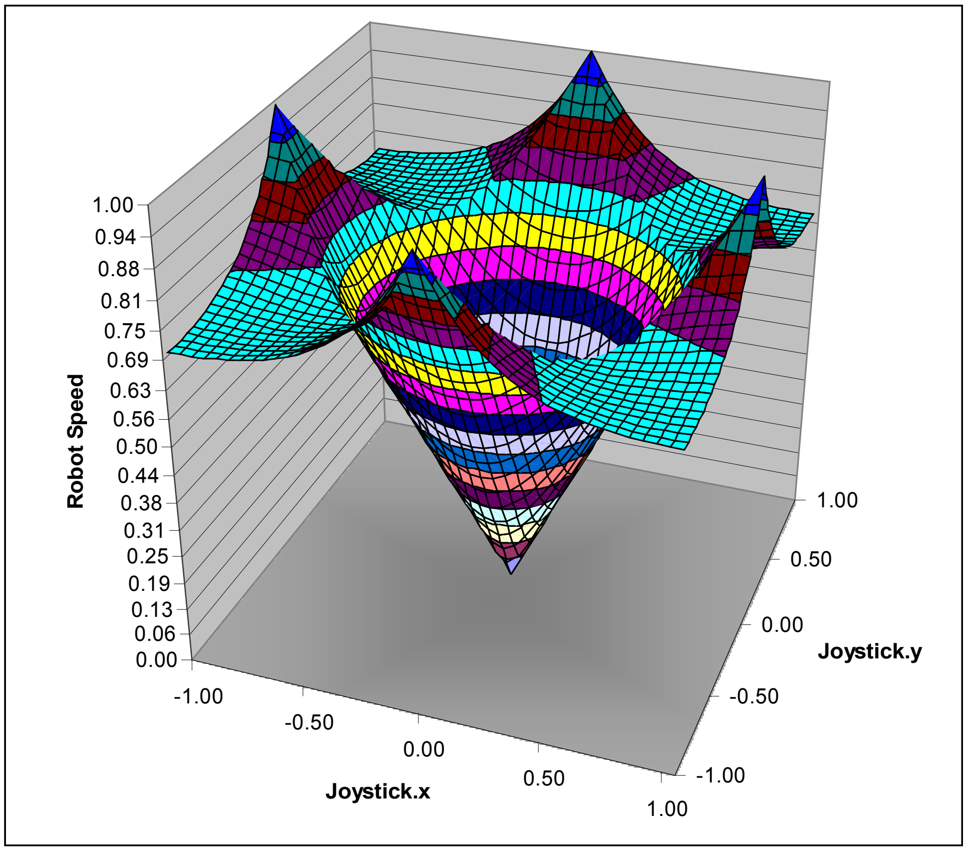

Illustration 4: 2011 Mecanum Control Algorithm Robot Speed vs. Velocity and Heading with no Rotation applied

Note that limiting the Velocity to 1.00 with this algorithm allows for a robot total speed of 1.00 in the cardinal directions and $\sqrt{2}/2$ or approximately 0.71 in the diagonals instead of limiting the maximum total speed to $\sqrt{2}/2$ in all directions. The cone is also wider when compared to the previous algorithm which is a result of the robot speed maintaining linearity from stopped to full speed.

When applying the same *Rotation* of 0.29 for this algorithm, *Velocity* and *Heading* can be visualized as:

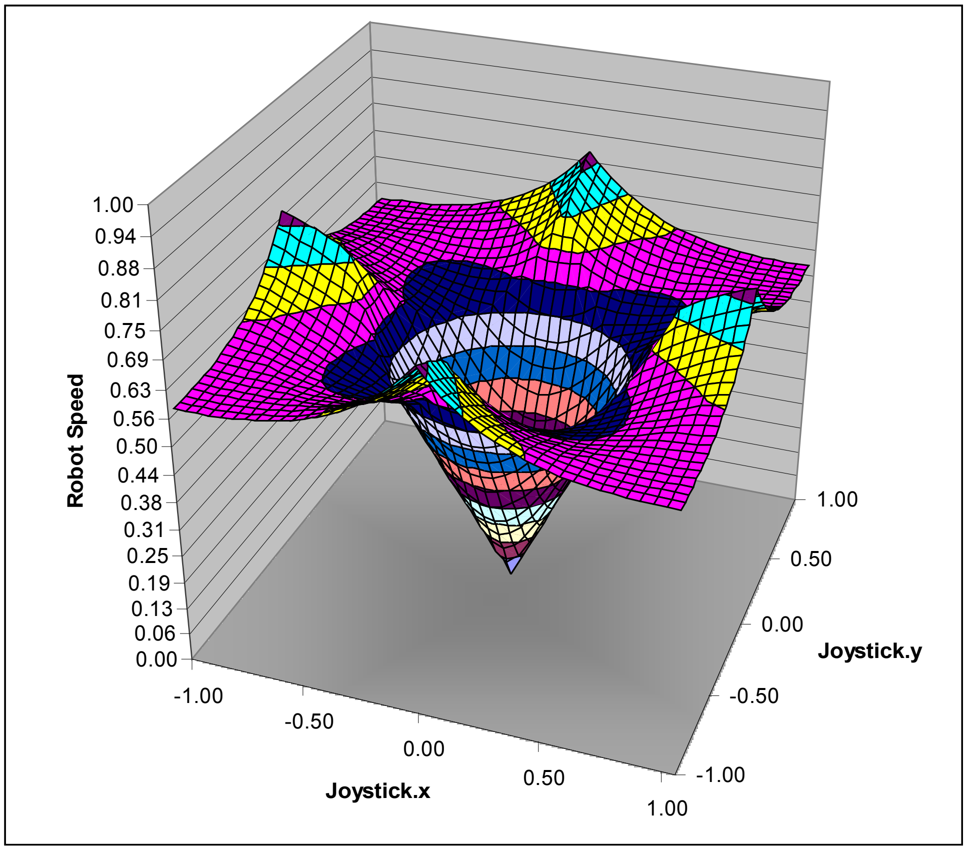

Illustration 5: 2011 Mecanum Control Algorithm Robot Speed vs. Velocity and Heading with a Rotation of 0.29

Note that the shape of the curve has not been changed with the application of *Rotation* but is scaled down equally. Along the cardinal directions the relatively higher robot speed of the algorithm is maintained in comparison to the previous algorithm. Along the diagonals the maximum robot speed is reduced to the same levels as the clipping produced in the previous algorithm.

It is also interesting to note that with a Velocity of 0.50 the 2011 Mecanum Control Algorithm has the same robot speed output as the 2010 Mecanum Control Algorithm has at a Velocity of $\sqrt{2}/2$ as seen below because the only difference under this point in the curve for each is the scalar of $\sqrt{2}/2$:

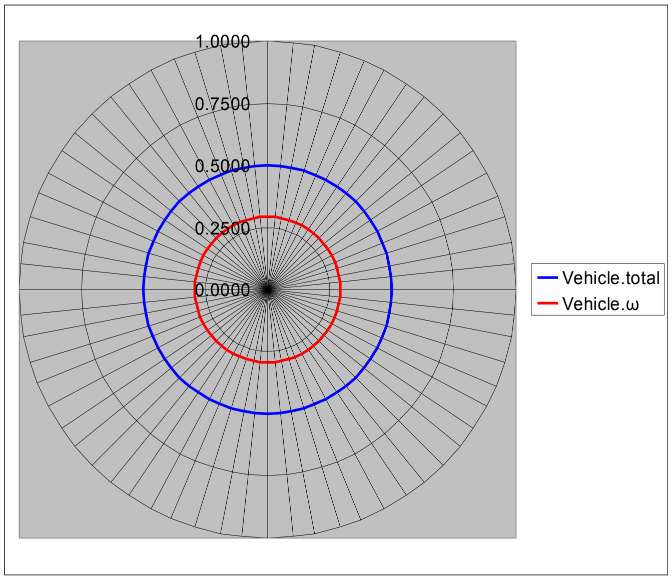

Illustration 6: 2011 Mecanum Control Algorithm Robot Speed vs. Heading with a Velocity of 0.50 and a Rotation of 0.29

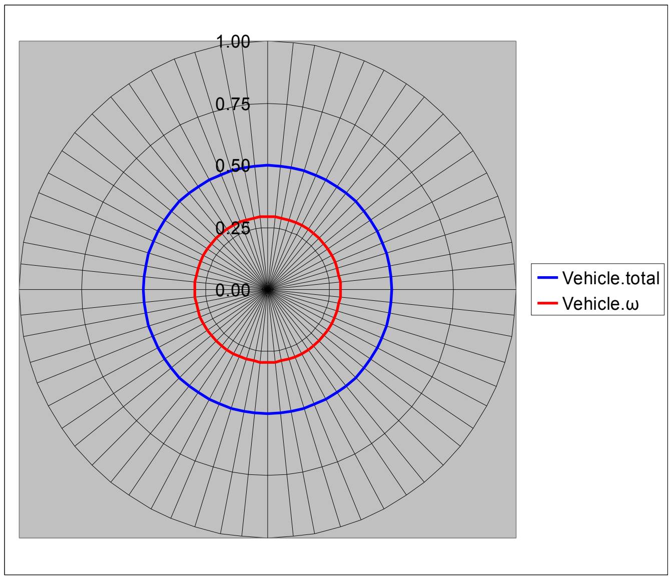

Illustration 7: 2010 Mecanum Control Algorithm Robot Speed vs. Heading with a Velocity of 0.71 with a Rotation of 0.29

Conclusion

There are obviously many benefits of the 2011 Mecanum Control Algorithm over the 2010 version including: maximizing the speed of the robot where possible; maintaining as much independence between the x-axis velocity, y-axis velocity, and yaw-rate; and a quicker reaction of robot to joystick command due to the slope of speed curve. There are also a few benefits of the 2010 Mecanum Control Algorithm over the 2011 version including: higher precision speed control as the slope of the speed curve is softer and more larger area of linear response to joystick commands. The benefits of each are a result of the scaling and normalization applied in the 2011 version.

As the goal of any robot control system is to make the robot controllable by the driver the final call for how the robot reacts to joystick commands lies with the drivers. If the driver's style is to always peg the sticks at the physical stops when driving regardless of the necessity of speed, then a linear response to joystick commands is more desirable so the robot always reacts as expected. If the driver's style is small precise movements and being conscious using only the necessary speed, a higher precision response to joystick commands is more desirable to all for the finest control available. While these would typically be cases for using the 2010 version of the algorithm they are not.

The tradeoff that gives precision and a larger area of linear response is against maximum speed. This is achieved by the scalar multiplier in the 2011 version of the algorithm which is hard coded to $\sqrt{2}/2$ and not the vector normalization. If the multiplier is tweaked to give the driver the response they need (or preferably the joystick inputs are scaled or filtered before being applied) the 2011 version can give the same response to translation commands as the 2010 version. The normalization in the 2011 version is an upgrade from the clipping that occurs in the 2010 version as it can better maintain the relationship between each of the wheel speeds and therefore maintain holonomic control.