Differential Steering 1 Impulsadventure code sample

Introduction by Jørn for the Joystick Piloting Project:

I have decided on the following terminology for the Joystick Piloting Project:

|  |  |  |
| --- | --- | --- |
| Skid steering: | Differential steering: | Vectored steering: |
|  |  |  |

Skid steering and differential steering is based on this video: <https://www.youtube.com/watch?v=F3G0sUz3_Jw>

Here is link to site and article:

<https://www.impulseadventure.com/elec/robot-differential-steering.html>

Below is copy of site just in case site is removed:

**Robot Control - Joystick for Differential Steering**

PS2 Controller for Robot

Most basic robot platforms use differential steering to control its motion. This is also known as tank drive or skid steering. With a pair of drive motors that can be independently driven in both direction, one can drive forward, backwards, turn and spin or pivot on the spot. The ability for a robot to pivot on the spot makes odometry (measurement of robot position by counting wheel rotations) much easier.

Differential Steering

Differential steering is typically implemented by two PWM (pulse-width modulated) bidirectional motor drivers, one for the left and the other for the right. If we assume the motors can be operated from -100% (full speed reverse) to +100% (full speed forward), then we get the following control:

Left Motor Right Motor Robot Motion

+100% +100% Full speed drive forward

+50% +50% Half speed drive forward

0% 0% Stop

-50% -50% Half speed drive reverse

-100% -100% Full speed drive reverse

+100% -100% Full speed pivot right

-100% +100% Full speed pivot left

+100% 0% Full speed drive forward with right turn

0% +100% Full speed drive forward with left turn

A differential steering robot often uses 2 or 4 drive wheels. When configured as a 4-wheel drive (4WD) robot, the left front and back wheels are often driven by the same motor controller channel, and the right pair is driven by another controller / channel.

4WD Turning on Carpet

In order for a 4WD robot to pivot (also known as a zero-radius turn), the wheels will need to slip sideways against the ground surface. This is much harder for a 4WD robot to do than a 2WD robot as there is much more lateral friction / resistance on the wheels. In fact, unless you have relatively powerful (high torque) motors, you may find that your 4WD robot can't turn on carpet or grass! The longer the wheelbase (distance between front and rear wheels), the more difficult turning becomes.

Controllers for Differential Drive Robots

There are many forms of robot remote controls possible -- the more common controller methods for a differential steering robot include:

Differential drive: Two single-axis joysticks, each with forward/backward movement

Throttle & turn: Two single-axis joysticks, one with forward/backward movement and the other with left/right movement

Throttle & turn: One dual-axis joystick, with both forward/backward and left/right movement

The following section describes the last option in more detail.

Joystick to Differential Drive - Throttle & Turn

After changing my robot's controller from a PS2 gamepad to a 9-channel RC transmitter, I then used an Arduino to convert the RC's multi-channel PWM signals a SPI slave for reading by a Raspberry Pi. The Raspberry Pi then implements the code to convert the joystick input to the motor control inputs (also a pair of PWM signals). Later, I replaced the RC transmitter with my own custom Robot Telemetry Remote Control using a ZigBee transceiver and my own 2-axis joystick / gimbals.

To improve the robot's tracking capabilities, I decided to change the controller mechanism from a two-joystick differential drive input to a single joystick throttle & turn methodology. Doing this requires a suitable algorithm to convert the X-Y input into left motor & right motor control.

My goals were the following:

Joystick center should stop the robot

Joystick full forward should drive robot forwards with max throttle

Joystick full backward should drive robot backwards with max throttle

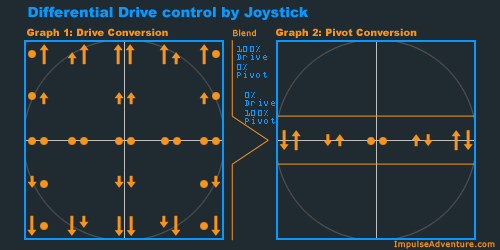
Joystick full right should spin robot clockwise at maximum rate

Joystick full left should spin robot counter-clockwise at maximum rate

Partial movement in any direction should blend the above controls proportionately

Mapping Algorithm

Robot differential drive joystick algorithm



To derive a control scheme or algorithm for the above goals requires mixing two concepts: drive and pivot. It is a simple matter to determine the mapping between a joystick X-Y input and the drive output (see Graph 1). Similarly, one can easily determine a mapping bewteen a joystick X input and the pivot output (Graph 2). However, combining the two is not quite as intuitive. The algorithm I use blends the two concepts on the basis of the joystick Y input.

The drive mapping takes priority except when we are close to the midpoint of the joystick Y position -- at which point we prioritize the pivot operations. The resulting blend is shown by the graph in the middle.. In this manner, we have the ability to spin the robot when the Y input is near zero (the midpoint), but retain full drive control at all other times. I used a linear slope for the blend line, and made it reach its maximum blend at a Y position that was a quarter of maximum position.

Joystick to Differential Steering - C Code

The following code assumes that the joystick input has been normalized to an 8-bit signed range: -128...+127. The resulting motor outputs may also use the same range, in which case they will require a final conversion to the PWM range (eg. 1000us...2000us pulse width). The conversion algorithm can be implemented in a few component steps:

Calculate drive turn output due to joystick X input

Scale drive output due to joystick Y input

Calculate pivot output due to joystick X input

Calculate drive vs pivot scale due to joystick Y input

Calculate final mix of drive and pivot

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

// Differential Steering Joystick Algorithm

// ========================================

// by Calvin Hass

// https://www.impulseadventure.com/elec/

//

// Converts a single dual-axis joystick into a differential

// drive motor control, with support for both drive, turn

// and pivot operations.

//

// INPUTS

int nJoyX; // Joystick X input (-128..+127)

int nJoyY; // Joystick Y input (-128..+127)

// OUTPUTS

int nMotMixL; // Motor (left) mixed output (-128..+127)

int nMotMixR; // Motor (right) mixed output (-128..+127)

// CONFIG

// - fPivYLimt : The threshold at which the pivot action starts

// This threshold is measured in units on the Y-axis

// away from the X-axis (Y=0). A greater value will assign

// more of the joystick's range to pivot actions.

// Allowable range: (0..+127)

float fPivYLimit = 32.0;

// TEMP VARIABLES

float nMotPremixL; // Motor (left) premixed output (-128..+127)

float nMotPremixR; // Motor (right) premixed output (-128..+127)

int nPivSpeed; // Pivot Speed (-128..+127)

float fPivScale; // Balance scale b/w drive and pivot ( 0..1 )

// Calculate Drive Turn output due to Joystick X input

if (nJoyY >= 0) {

// Forward

nMotPremixL = (nJoyX>=0)? 127.0 : (127.0 + nJoyX);

nMotPremixR = (nJoyX>=0)? (127.0 - nJoyX) : 127.0;

} else {

// Reverse

nMotPremixL = (nJoyX>=0)? (127.0 - nJoyX) : 127.0;

nMotPremixR = (nJoyX>=0)? 127.0 : (127.0 + nJoyX);

}

// Scale Drive output due to Joystick Y input (throttle)

nMotPremixL = nMotPremixL \* nJoyY/128.0;

nMotPremixR = nMotPremixR \* nJoyY/128.0;

// Now calculate pivot amount

// - Strength of pivot (nPivSpeed) based on Joystick X input

// - Blending of pivot vs drive (fPivScale) based on Joystick Y input

nPivSpeed = nJoyX;

fPivScale = (abs(nJoyY)>fPivYLimit)? 0.0 : (1.0 - abs(nJoyY)/fPivYLimit);

// Calculate final mix of Drive and Pivot

nMotMixL = (1.0-fPivScale)\*nMotPremixL + fPivScale\*( nPivSpeed);

nMotMixR = (1.0-fPivScale)\*nMotPremixR + fPivScale\*(-nPivSpeed);

// Convert to Motor PWM range

// ...

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Transitions between Turn and Pivot

When writing a differential drive algorithm, one has to be careful with handling the transitions between forward & reverse as well as between turn & pivot. If you don't address these transitions properly, your robot may behave erratically or be difficult to control.

Transition between Forward and Reverse

Assuming that the differential drive algorithm favors a pivot (zero radius turn) near the Y-axis (ie. transition between forward and reverse drive), then it is essential that the pivot direction be consistent between a small positive Y input and a small negative Y input. In other words, if (X=60,Y=5) gives a half-speed clockwise zero-radius turn, then (X=60,Y=-5) must also give a clockwise zero-radius turn.

Originally, I had attempted to map the controls differently in the reverse direction (as I felt they were more intuitive), but quickly discovered that an incorrect transition here could throw the robot into an extremely abrupt change of direction!

Transitions between Reverse Pivot and Turn

When designing the algorithm for the reverse direction, I thought it would make more sense to have the robot turn (backwards) in the direction of the stick input. In other words:

Joystick input of (X=60,Y=-120) - Half-speed backwards turn to the rear right (robot rotating counter-clockwise)

Joystick input of (X=-120,Y=-120) - Full-speed backwards turn to the rear left (robot rotating clockwise)

My first implementation of the algorithm did exactly this. And at moderate speed (throttle input), the steering algorithm seemed like it was working well. However, if you were to do this, what you would soon discover is that there will be a rotational transition between pivot and turn when you increase throttle input in the reverse direction. What this means is that, for the same X-input (turn/pivot speed), the direction of rotation (CW vs CCW) will change as you increase the reverse throttle! The net result of this is that your controls will be flipped at the point in which your blend ratio flips from pivot to turn.

The most intuitive solution I have found is to ensure that, in the reverse direction, we match the direction of rotation in both the reverse pivot and reverse turn. For example:

Joystick input of (X=60,Y=-120) - Half-speed backwards turn to the rear left (robot rotating clockwise)

Joystick input of (X=-120,Y=-120) - Full-speed backwards turn to the rear right (robot rotating counter-clockwise)

Advanced Steering Controls

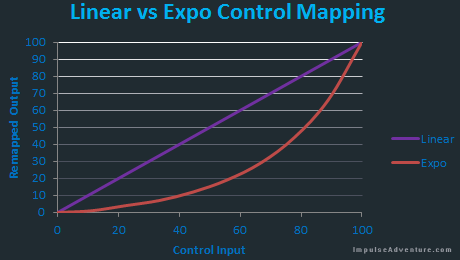
In addition to the above algorithm, there are a few other input and output controls that may be worth considering in a real robot:

Controller deadband

Most controllers use a pair of analog potentiometers (variable resistors) to determine the stick position. Ideally, these potentiometers would be calibrated to provide a zero/midpoint-output when the stick is perfectly centered. In reality, a centered stick will often tend to show a small offset in the measured X and/or Y position. Of course for a robot it is usually desirable to have the robot be motionless when the control stick is released! Adding a small amount of deadband will ensure that slight movement in the stick position near the center doesn't affect the robot.

Exponential controller input (Expo)

In some cases, it may be beneficial to support fine-grained control near the center position (outside the deadband) and then large/fast motions as one approaches the limits of the joystick deflection. This "ease-in" allows much more precise control at low speed navigation, but sacrifices some joystick resolution as you increase velocity. As the name suggests, this is accomplished by providing a controller remapping curve that translates a linear input range to an exponential range. For an example of this, please see the graph below (which assumes a controller input range of 0..100). In the case of C2Bot, I plan to add some controller exponential to the pivot action since it seems like I get more rotational velocity than I would like (when the control stick is approaching the Y axis).



Self-driving / Object Avoidance

Although this is not truly a controller input, simple robot object avoidance schemes can be designed to mimic joystick deflection when nearing an obstacle. By doing this, you can arm your robot with the ability to avoid objects in case the operator didn't react in time. In C2Bot, I have implemented Infrared and Ultrasonic sensors that will adjust the current heading (or even stop) if it approaches an object on the sides or front. Paired with an autopilot throttle, the robot can perform limited self-driving in this way.

Acceleration limiter

With larger robots, it can be useful to reduce the maximum acceleration / deacceleration that can be applied from the control inputs. This may be necessary to prevent damage to the robot (eg. full speed forward to reverse). Limiting acceleration may also be useful if you are implementing wheel odometry calculations to track the robot position. Errors are introduced into odometry calculations any time that the wheels slip. Therefore, reducing the motor acceleration can help prevent wheel slip when starting the robot's movement. In the case of C2Bot, the RoboClaw Motor Controller is responsible for the acceleration limiting function.

Quadrature encoder / PID control

PID control over wheel speeds is not strictly a function of the controller input, but it is worth mentioning here. The output of the above differential drive algorithm is a pair of motor drive commands. These could be converted to PWM and interface with a motor driver circuit directly, with the PWM pulse width directing a voltage to each motor which translates into some degree of wheel speed. Unfortunately, differences in friction, slip and other factors make the actual wheel speed less not directly proportional to the voltage input. With such an implementation, there is no feedback from the motor back to the controller to indicate how fast the wheels are actually rotating. Adding a PID (proportional integrating differential) controller allows wheel speed encoders (quadrature encoders) to measure the actual wheel speed and compare these against the desired motor velocity (from the control stick deflection). The closed-loop PID control enables precise velocity control, resulting in excellent vehicle tracking and rotation control. In the case of C2Bot, all four gear motors have integrated quadrature encoders, but the motor controller only uses the rear left and rear right sensors in modulating the motor input.

Alternate Steering Algorithm using Bearing

In the comments, reader John asked about another approach that involved using the joystick's bearing / heading as the input factor for blending between drive / turn and pivot. It is an interesting option, as the user is more likely to want to initiate a pivot when the control stick is rotated closer to the X-axis. In many ways, it is very similar to the original algorithm described above. The transition point or threshold is then based on joystick bearing / heading instead of joystick Y-axis value.

Changes to the C code to use bearing-based algorithm:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

Replace fPivYLimit with:

// CONFIG

float fPivBearLimit = 75.0; // Bearing threshold for pivot action (degrees)

Replace fPivScale code with:

// Calculate radial bearing (away from Y-axis)

// - This is an absolute value in degrees (0..90)

float fBearMag;

if (nJoyY == 0) {

// Handle special case of Y-axis=0

if (nJoyX == 0) {

fBearMag = 0;

} else {

fBearMag = 90;

}

} else {

// Bearing (magnitude) away from the Y-axis is calculated based on the

// Joystick X & Y input. The arc-tangent angle is then converted

// from radians to degrees.

fBearMag = atan( (float)abs(nJoyX) / (float)abs(nJoyY) ) \*90.0/(3.14159/2.0);

}

// Blending of pivot vs drive (fPivScale) based on Joystick bearing

fPivScale = (fBearMag<fPivBearLimit)? 0.0 : (fBearMag-fPivBearLimit)/(90.0-fPivBearLimit);

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In fact, if we choose a bearing threshold (fPivBearLimit) that matches the same angle that the Y-axis limit makes (at full joystick X input), the behavior is very similar in most of the "outer" areas of the control space. This is because the bearing's tangent is roughly linear at small angles to displacement from the X-axis. For example, a Y-axis threshold (fPivYLimit) of 32 (out of 128) for the original algorithm is approximately equal to a bearing threshold (fPivBearLimit) of 75 degrees (measured away from X=0, Y=127).

Upon testing this algorithm, I discovered that it worked well for much of the control input, but became very touchy as we attempt pivots with small joystick X-input values. The reason for this is that the bearing becomes higly sensitive as the X-axis becomes small. A minor change in the Y-input can lead to major changes in the bearing, which result in a large change to the blend ratio. An abrupt change in the blend ratio (ie. from pivot to drive/turn) can quickly cause one of the drive wheels to reverse its direction. Further tweaking to this algorithm may make it easier to control, but for the moment I find I prefer the algorithm that blends based on the Y-input.