A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators

by G.W. Lucas

Contents

Michael Gauland, one of The Rossum Project volunteers, recently contributed a neat little Java applet illustrating the concepts in this paper. To view the applet, click [Here].

Introduction

These notes present equations describing the path of a robot or vehicle equipped with a *differentially steered drive system*. This simple and reliable wheel-based propulsion system is commonly used in smaller robots. It is familiar from ordinary life because it is essentially the same system that is used in a wheelchair: two wheels mounted on a single axis are independently powered and controlled, thus providing both drive and steering functions. The equations in these notes provide a an elementary model for the differentially steered drive system (which is often called a differential steering system). This model can be used to predict how a robot equipped with such a system will respond to changes in its wheel speed and what path it will follow under various conditions. The model can also be used to calculate a robot's position in dead-reckoning or *localization by odometry* applications (techniques that estimate a robot's position based on distances measured with odometer devices mounted on each wheel).

It is worth emphasizing that the equations given here do represent an *elementary* model for the motion of a robot or vehicle. They describe the robot's position and orientation as a function of the movement of its

wheels, but ignore the underlying physics involved in making that motion happen. Issues such as torques and forces, friction, energy and inertia will be left for a more advanced discussion. In technical terms, this method of describing motion is referred to as a *kinematics approach*. It ignores the causes of motion (which would be studied in a *dynamics approach*) and focuses on the effects. These notes also ignore the many interesting details of motors, gearing, electromagnetics, power supplies, and other engineering considerations that make wheel-based actuators possible. As such they provide a first step to understanding the behavior of robots with differentially steered drive systems, not a complete description.

The intended audience for this web page includes high school and undergraduate students who may not be comfortable with the calculus techniques used to develop the differential steering model. Such students should be able to "parse out" the relevant information by scanning the text and paying attention to the results in equations [3] and [5]. To really take advantage of the model requires understanding something of the math and kinematics behind it. So the discussion on this page includes a lot of background information on the problem and a more-than-customary amount of explanation. By providing so much detail, these notes should help the reader apply the differential steering model to his or her own robotic applications.

The differential steering system is sometimes incorrectly called a "differential drive system." The term *differential drive* is used in automotive engineering to represent a kind of propulsion system that transfers power to its drive wheels through a differential gear or related device (as in the classic rear-wheel drive vehicle). Clearly, it is a different concept than the system described here. Robotics enthusiasts are careful to make the distinction.

Finally, I would like to gratefully acknowledge the contributions of Albert Gerheim, a gifted mathematician and engineer who suggested the approach used for the model.

The Differential Steering System

As mentioned above, the differentially steered drive system used in many robots is essentially the same arrangement as that used in a wheelchair. So, most of us have an intuitive grasp of the way it behaves. If both drive wheels turn in tandem, the robot moves in a straight line. If one wheel rotates faster than the other, the robot follows a curved path inward toward the slower wheel. If the wheels turn at equal speed, but in opposite directions, the robot pivots. Thus, steering the robot is just a matter of varying the speeds of the drive wheels. But how exactly do we choose the speeds so that the robot will move where we want it to go? In these notes, we will try to refine this intuitive understanding into mathematical expressions that can be used for implementing robot control software.

Rounding a Corner

Before tackling the differential steering model, let's consider the problem of getting a robot with a to make a 90-degree turn around a corner in a hallway or a road. A simple approach is to have the robot drive to the intersection, stop, and pivot. Clearly, though, such an approach is inefficient and a bit ungainly. A more elegant approach would be for the robot to *round the corner*, following a gradual circular arc through the intersection while maintaining a steady speed. To accomplish this end, we increase the speed of the outer wheel while slowing that of the inner.

Is the path that the robot follows truly circular, or is it some other kind of curve? Later on, we will show that if both wheels turn at constant, but different, speeds, the robot does indeed follow a circular path. For now, just assume that path is circular. With that in mind, we see that wheel speeds are selected based on how wide a turn we desire the robot to make.

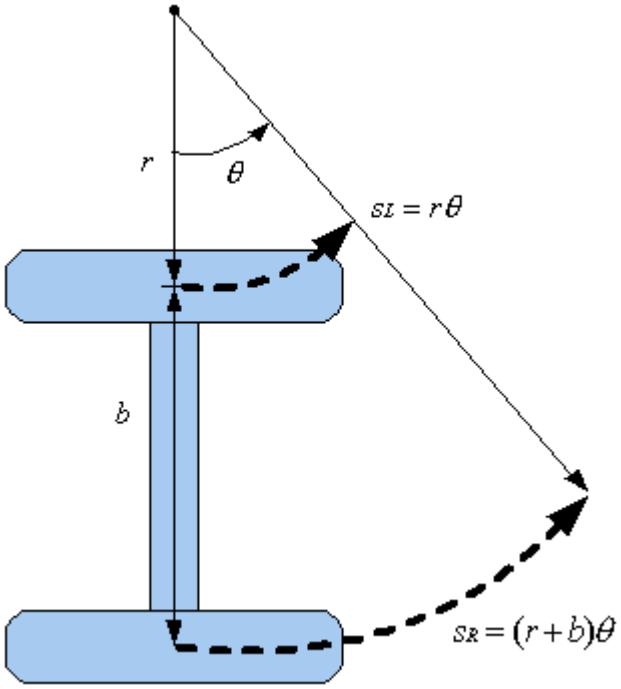

Figure 1. Path of wheels through a turn.

Referring to Figure 1, observe the relationships:

$$s_L = r\theta$$
$$s_R = (r + b)\theta$$
$$s_M = (r + b / 2)\theta$$

where s_L, s_R give the displacement (distance traveled) for the left and right wheels respectively, r is the turn radius for the inner (left) wheel, b is the distance between wheels (from center-to-center along the length of the axle), and θ is the angle of the turn in radians ($\theta_{radians} = \theta_{degrees}(\pi/180)$). s_M is the speed at the center point on the main axle. In this discussion, we will treat the axle's center point as the origin of the simulated robot's frame of reference.

Once we've established the simple geometry for the differential steering system, it is easy to develop algorithms for controlling the robot's path. Note, though, that we did make an important simplifying assumption: the wheels maintain a steady velocity. We neglected the effects of acceleration. If the wheels are allowed to accelerate, the curve which describes the robot's trajectory can become much more complicated. When working with very light robots, where the mass (and inertia) of the platform is small, we can often get away with treating changes in speed as nearly instantaneous. The path that the robot follows will not be truly circular, but it will be close enough for many applications. For larger, heavier robots, of course, mass is important and acceleration must be considered.

Some Preliminaries to Developing a Model

In the discussion above, we considered the problem of trying to *choose* wheel speeds based on a desired path. Now we will consider the opposite problem: how do we find the trajectory of the robot if we are *told* what speeds the wheels will be turning? The answer to this question leads to the development of our model and also turns out to be directly applicable to dead reckoning. Essentially, this is a problem in what is called *forward kinematics*, the technique of predicting a mechanical system's behavior based on the inputs to that system.

Developing a model for the differential steering system is not difficult, but does require the use of calculus and differential equations. If you are unfamiliar with calculus, you may prefer to just view the results given in equations [3] and [5] below.

To begin, consider that what we're really interested in is how x and y coordinates (and orientation) change with respect to time. At any instant, the x and y coordinates of the robot's center point are changing based on its speed and orientation. We treat orientation as an angle q measured in radians, counter-clockwise from the x-axis. Recall that the vector giving the direction of forward motion for the robot will be simply $(\cos\theta, \sin\theta)$. The x and y coordinates for the robot's center point will change depending on the speed of its motion along that vector.

These observations suggest that, taking $m(t)$ and $\theta(t)$ as time-dependent functions for the robot's speed and orientation, our solution is going to be in the form:

$$dx/dt = m(t)\cos(\theta(t)),$$
$$dy/dt = m(t)\sin(\theta(t)) \qquad [1]$$

The Derivation

Let's derive functions for the robot's trajectory. For now, it will be easier to defer thinking about acceleration and just assume that the speeds of the wheels are constant. If both wheels are moving at the same velocity, the robot travels in a straight line and the equation for its trajectory is trivial. So, instead, we will consider the case where the wheels travel at different velocities.

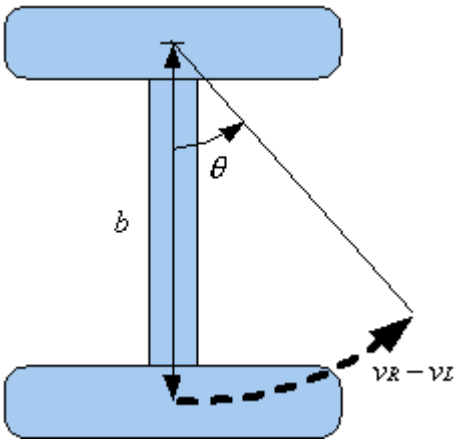

Figure 2. Wheels at different velocities.

Referring to figure 2, note that as the robot changes position, all points on the robot may be in motion. To develop a forward kinematic equation for the motion of a differential steering system, we start by specifying a *frame of reference* in which an arbitarily chosen point is treated as stationary. All other points in the system are treated as moving relative to the reference point. The robot is considered a rigid body. By approaching the problem in this manner, we will gain insights which can be applied to the more general problem of modeling the robot's motion in an absolute frame of reference.

The point that we select as our reference is the center point of the left wheel. This is the point where an idealized wheel makes contact with the floor. Again, all motion in this frame of reference is treated in relative to the left-wheel point. Because the right wheel is mounted perpendicular to the axle, its motion *within the frame of reference* follows a circular arc with a radius corresponding to the length of the axle (from hub center to hub center).

Now the central point itself may be in motion, so the *actual* path of the right wheel will not necessarily correspond to that particular circular arc. But the change in orientation θ is *not* restricted to the robot's frame of reference. Because we treat the robot as a rigid body, all points in the system undergo the same change in orientation. If we pivot the robot 10 degrees about the left wheel, all points undergo a 10 degree change in orientation. And any change in orientation in the special frame of reference is, in fact, equivalent to that of the more general case.

Based on these observations, we can derive a differential equation describing the change in orientation as with respect to time of time. The definition of an angle given in radians is the length of a circular arc divided by the radius of that circle. The relative velocity of the right wheel gives us that length of arc per unit time. The length from the wheel to the center point gives us the radius. Combining these facts, we have:

$$d\theta / dt = (v_R - v_L) / b \qquad [2]$$

Integrating [2] and taking the initial orientation of the robot as $\theta(0) = \theta_0$, we find a function for calculating the robot's orientation as a function of wheel velocity and time:

$$\theta(t) = (v_R - v_L)t / b + \theta_0 \qquad [3]$$

As noted above, this change in orientation also applies to the absolute frame of reference. Shifting our attention back to the absolute frame, we recall from [1] above that the robot's overall motion depends on the the velocity its center point (the midpoint of the axle). That velocity is simply the average of that for the two wheels, or $(v_R + v_L) / 2$. We combine this fact with what we know the about orientation as a function of time, and get the following differential equations:

$$\begin{aligned} dx/dt &= [(v_R + v_L) / 2]\cos(\theta(t)) \\ dy/dt &= [(v_R + v_L) / 2]\sin(\theta(t)) \end{aligned} \qquad [4]$$

Note that the equations in [4] are in the same form as that shown in [1]. Integrating and applying the initial position of the robot $x(0) = x_0, y(0) = y_0$, we have

$$\begin{aligned} x(t) &= x_0 + \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\sin((v_R - v_L)t / b + \theta_0) - \sin(\theta_0)\right] \\ y(t) &= y_0 - \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\cos((v_R - v_L)t / b + \theta_0) - \cos(\theta_0)\right] \end{aligned} \qquad [5]$$

Note that the equations given in [5] confirm the earlier assertion that, when the wheels turn at fixed velocities, the robot follows a circular path. The term $(b / 2)(v_R + v_L) / (v_R - v_L)$ is actually the turn radius for circular trajectory of the robot's center.

When using [5] in a computer application, it is necessary to implement special handling for cases where the wheel speeds are nearly equal and $v_R - v_L \approx 0$. In such cases, of course, the robot travels in a nearly straight line. Using *L'Hospital's rule* from elementary calculus, we can show that the equations do have limits approaching a straight line in the direction of the initial orientation θ_0.

Dead Reckoning and Odometry

Once we have [5], it's a small step to be able to perform dead reckoning. In traditional aviation or nautical navigation, the term "dead reckon" means to estimate position without external references. Position is dead reckoned using knowledge about a vehicle's course and speed over a period of time. In robotics, the necessary information is often obtained by measuring wheel revolutions. Devices called encoders are coupled to a robot's drive wheels and act like digital odometers. Although things can go wrong (as when the robot "spins out" on a slippery floor), encoders generally provide a good estimate of displacements s_R, s_L for the right and left wheels, respectively. These displacement values can, in turn, be used to determine position based on odometry calculations.

The equations given in [5] can be used for dead reckoning by substituting s_R, s_L for the terms v_R, v_L and dropping the time value t, then solving for the values x, y, and θ. When using the equations, you need to be careful about cases where the robot travels in a nearly straight-line path and the displacements s_R, s_L become

nearly equal resulting in values near zero in the denominators.

A Simpler Formula for Dead Reckoning

Many popular authors on robotics recommend the formulas shown in [6] below as a way to avoid the complications that we saw in [5]. For example, Johann Borenstein, et al., propose something very similar in their widely-cited web book Where Am I? - Systems and Methods for Mobile Robot Positioning. The methods in [6] are especially well suited to small robot applications where on-board computing power is limited (and floating-point operations might not be available).

$$\bar{s} = (s_R + s_L) / 2$$
$$\theta = (s_R - s_L) / b + \theta_0$$
$$x = \bar{s} \cos(\theta) + x_0$$
$$y = \bar{s} \sin(\theta) + y_0$$

[6]

The forumulas in [6] are simple and convenient. They work well for algorithms as long as you remember that they are approximations... a fact that is sometimes overlooked. Essentially, [6] computes the robot's total rotation and distance traveled, then treats it as if it had completed the full rotation as a pivot the very beginning of the maneuver and then traveled in a straight line. Of course, this kind of "point-and-shoot" description is an incomplete representation of the reality. But so long as the robot's actual path doesn't involve too much of a turn, everything is fine.

How much of a turn is *too much*? That depends on the geometry of the particular robot. You can probably use [6] as long as the angle of the turn does not get larger than 10 degrees. If it does, you should apply the approximation in stages. That is, if the robot's transit results in a turn of 20 degrees, treat it as two segments of 10 degrees each.

Updated June 2004: Tom Brown of the University of Illinois at Urbana-Champaign proposes a nice little improvement to [6]. Imagine that the robot is following a circular path curving to the left. It is easy to see that the point-and-shoot maneuver described above would create a path to the inside of the curve. Tom suggests that the accuracy of the approximation can be improved by simply dividing the robot's change in orientation by two. So the theta term in the equations becomes $\theta = (s_R - s_L) / 2b + \theta_0$. When adjusted this way, the point-and-shoot path will not be so far inside the curved path and, eventually, the two may even cross. Tom also generously provided an upgrade to the MotionApplet to demonstrate the effect of this improvement. *The Rossum Project is grateful to Tom Brown for his contribution and extends to him our thanks.*

Whether you apply [5] or [6] for your dead reckoning, remember that the realities of mechanical systems limit the accuracy of your calculations. If the robot's wheels hit a slippery spot on the floor and spin wildly, it will degrade the accuracy of the distances measured by the encoders. Even when your wheels maintain traction, your position estimates will be vulnerable to backlash (slop) in the robot's gearing, inaccuracies in the encoders, and small variations in the wheel speed.

Acceleration

Adding acceleration to the model is simply a matter of substituting appropriate time-dependent equations for wheel velocities into [2] and [4]. Unfortunately, the resulting differential equations are often not solvable, but must be evaluated using numerical methods. Let's consider a simple example where the wheels undergo a fixed rate of acceleration. Let

[7]

$$v_R(t) = a_R t + w_R$$
$$v_L(t) = a_L t + w_L$$

give the velocities of the right and left wheels where a_R, a_L give the values for a constant acceleration (or deceleration) and w_R, w_L give the initial velocities. Substituting into [2] and [4] we have

$$A = (a_R + a_L) / 2$$
$$B = (w_R + w_L) / 2$$
$$C = (a_R - a_L) / 2b$$
$$D = (w_R - w_L) / b$$

$$\theta(t) = Ct^2 + Dt + \theta_0 \qquad [8]$$

$$dx/dt = (At + B)\cos(Ct^2 + Dt + \theta_0)$$

$$dy/dt = (At + B)\sin(Ct^2 + Dt + \theta_0)$$

We were able to solve the differential equation for orientation directly. The equations for position are intractable and must be evaluated numerically. Jing YE, a student at the University of Melbourne, recently used Simpson's Rule, a technique from elementary calculus, to add a treatment of acceleration to Michael Gauland's MotionApplet.

In the real world, when a robot applies constant power to a wheel, it does not achieve a constant rate of acceleration. As the parts of the drive system turns faster, it experiences greater internal friction and requires more and more power to realize the same increase in speed. Based on your own system characteristics, you may wish to implement an acceleration model based on a time-dependent polynomial rather than the linear function used in [7]. An even more sophisticated analysis might yield better, more complicated functions for velocity and acceleration. Due to the inaccuracies inherent in robot mechanical systems, however, you will soon reach a point where improving the model is worth neither the human effort or extra CPU processing power needed to do so.

Direct comments to: gwlucas@users.sourceforge.net.

About The Rossum Project

The Rossum Project is an attempt to collect open-source software and other resources for robotics application. To find out more, visit our web page by clicking the icon to the right.

The Rossum Project gratefully acknowledges the support of SourceForge, an web service provider dedicated to the promotion and propagation of free, open-source software.