Determining Absolute Robot Position by Analysis of Various Sensors

15 Feb 2012

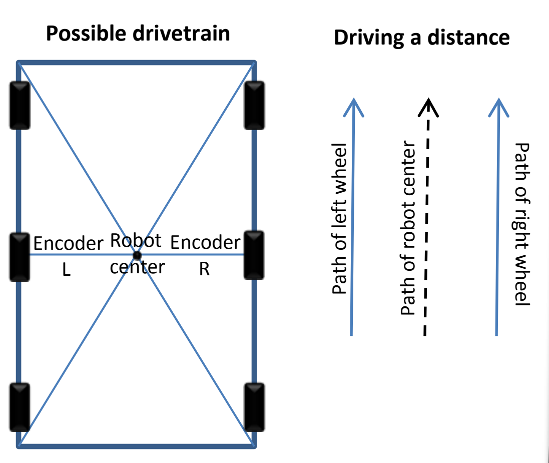
Peter Stoeckl

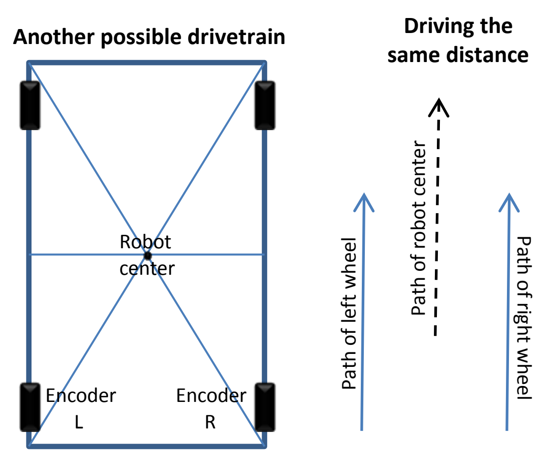
There are many ways of tracking the robot's position using sensors. One can use inputs from the wheel encoders, the gyro, the accelerometer, the camera, and likely more. The trick is combining all these various inputs, with their varying sensitivities and accuracies, into a more definitive, single, absolute calculated position. Before we can attack this problem, we must first figure out how to get from raw sensor data to potential position data.

To begin, we set the robot center at position (0,0), in whatever units are convenient, facing at an angle of 0 (i.e. forward along the x-axis). (Angles are in radians throughout, as this simplifies calculation. When starting the robot at a known position on the field, the relevant position data may be substituted for our generic zeros.) We then take the input from each sensor at regular time intervals (using a Timer to measure the actual time elapsed between readings), as often as possible, and calculate the change in each value from the previous reading.

The gyro reading is easy to translate into an angle; the code does this automatically, and the change is merely new angle minus old angle. (Calibration is another matter entirely, which we may not get to in this paper…)

The encoders require somewhat more work. Note: in this set of calculations it is assumed that (a) two encoders are used, one on each side; (b) these two encoders are attached to wheels one either side of the robot center; and (c) the drivetrain is a {straight drive}, like a standard six- or four-wheel (see left illustration). If assumption (a) and (c) hold, we can proceed as if (b) held as well, because we can shift the path of the center relative to the wheels without changing the value of the distance traveled (compare left and right illustrations). If there are more than one encoder to a side, a simple solution is to average the ticks of the encoders on each side. If the drivetrain is not a {straight drive}, the analysis becomes more complicated, and to save time, we will not cover it here.





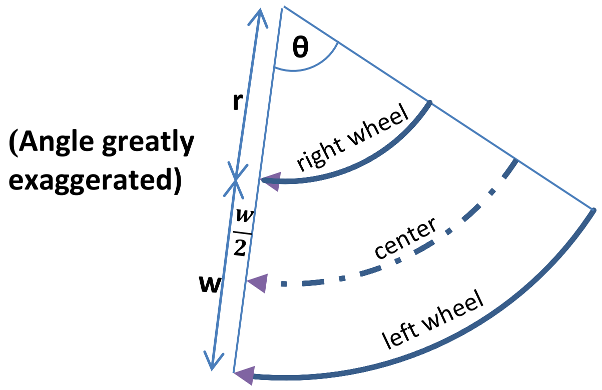
First, the change in ticks is calculated separately for each side (again via new value minus old value; quad encoders are a necessity here). Then this value is divided by the number of ticks per revolution and multiplied by the distance traveled per revolution to get the distance traveled: $\mathbf{\Delta d}_{left\ or\ right} = \mathbf{\Delta ticks}\left(\frac{\mathbf{revolution}}{\mathbf{ticks}}\right)\left(\frac{\mathbf{distance}}{\mathbf{revolution}}\right)$, where distance/revolution = π(wheel diameter). The above calculation is standard.

Now, since we are using a minimal time period, it does not matter if the robot is traveling straight or turning, {as in either case its path is a straight line}. We can therefore assume its path straight or curved without significant {error}. Assuming the path curved, the inner wheel travels along an arc of radius **r**, the outer wheel follows an arc of radius **r+w** (where **w** is the width of the drivetrain, wheel center to wheel center), and the robot center traces an arc of radius **r+w/2**. Without loss of generality, take the outer wheel to be the right wheel (as in a right turn). Assuming (reasonably given such a short time frame) that the arcs subtend the equal angle **θ** (see drawing below), the arc lengths (distances traveled) are $\mathbf{\Delta d_r = \theta r}$, $\mathbf{\Delta d_l = \theta(r+w)}$, and $\mathbf{\Delta d = \theta(r+w/2)}$ for left, right, and center respectively. Then $\mathbf{\Delta d} = \boldsymbol{\theta}\left(\mathbf{r} + \frac{w}{2}\right) = \frac{\mathbf{2\theta r + \theta w}}{\mathbf{2}} = \frac{\mathbf{\theta(r+w) + \theta r}}{\mathbf{2}}$, therefore:

$$\mathbf{\Delta d} = \frac{\mathbf{\Delta d_l + \Delta d_r}}{\mathbf{2}} \qquad \mathbf{(1)}$$

according to the encoders!

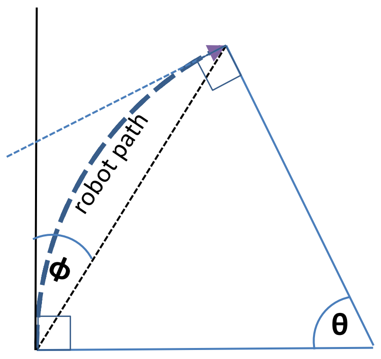
Note that this is accurate whether the robot was turning left or right, and that if it was in fact going straight, $\mathbf{\Delta d} = \mathbf{\Delta d_l} = \mathbf{\Delta d_r}$!



Now to determine the change in the robot's angle relative to the field coordinate system. Rearranging a few earlier equations, $\boldsymbol{\theta} = \frac{\mathbf{\Delta d_r}}{\mathbf{r}} = \frac{\mathbf{\Delta d_l}}{\mathbf{r+w}}$. Rearranging this proportion, $\mathbf{r\Delta d_l = r\Delta d_r + w\Delta d_r}$, so $\mathbf{r(\Delta d_l - \Delta d_r) = w\Delta d_r}$, that is, $\mathbf{r} = \frac{\mathbf{w\Delta d_r}}{\mathbf{\Delta d_l - \Delta d_r}}$. Substituting this into $\boldsymbol{\theta} = \frac{\mathbf{\Delta d_r}}{\mathbf{r}}$, we get $\boldsymbol{\theta} = \mathbf{\Delta d_r} \div \left(\frac{\mathbf{w\Delta d_r}}{\mathbf{\Delta d_l - \Delta d_r}}\right) = \frac{\mathbf{\Delta d_r(\Delta d_l - \Delta d_r)}}{\mathbf{w\Delta d_r}}$; that is:

$$\boldsymbol{\theta} = \frac{\mathbf{\Delta d_l - \Delta d_r}}{\mathbf{w}} \qquad \mathbf{(2)}$$

This is also the change in the robot's angle in terms of the encoder data, since the sides of the two angles are mutually perpendicular (in the limit). See the diagram below:



The change in robot angle **φ** is the angle between the two black lines; the solid black line is perpendicular to one of the radii that form the angle **θ**. The tangent line to the other radius (perpendicular by definition) is the blue dotted line. Since the time period used is so short, both dotted lines are practically the same as the circular arc followed by the robot, and thus are effectively identical. Therefore the sides of the angles **φ** and **θ** are perpendicular to each other, and so **φ = θ —** that is, **θ** is also the change in the robot's angle relative to the field! Note that **θ** is positive when the robot turns to the right, and negative when it turns to the left (fitting the gyro's angle convention but against the usual trigonometric convention).

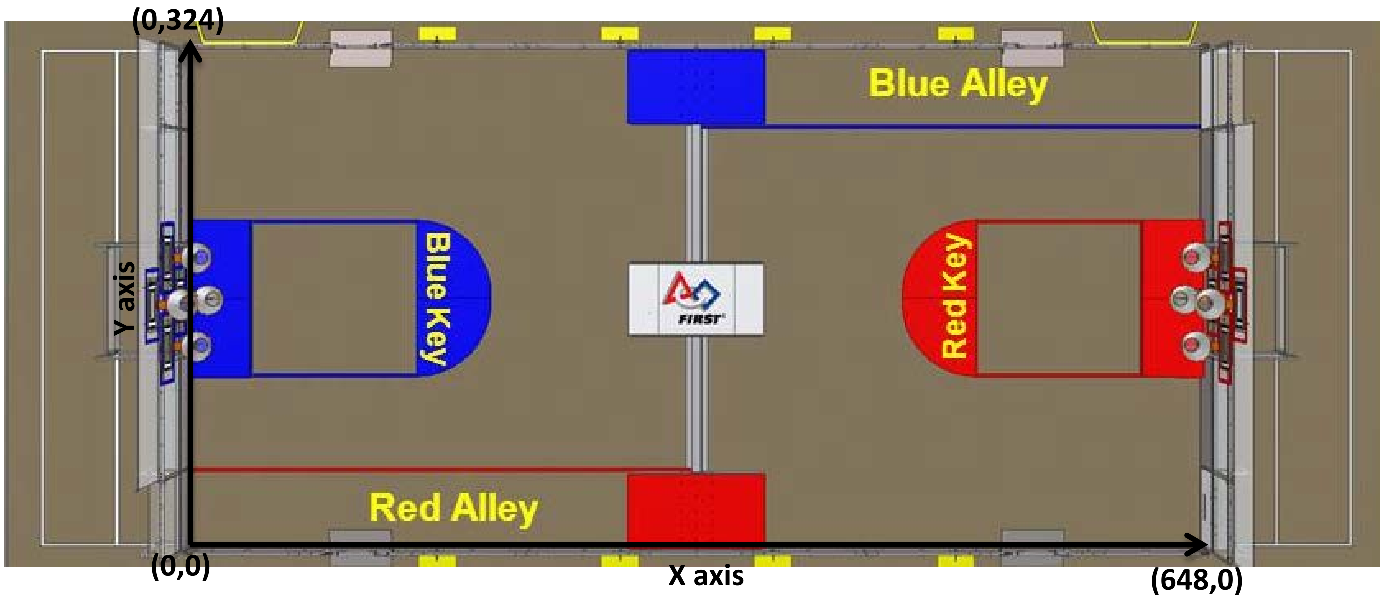
On to the accelerometer. First of all, getting from acceleration to distance: over any time interval the change in velocity $\Delta\mathbf{v} = \int_{t_0}^{t_1} \mathbf{a}(t)\mathbf{dt}$**,** where **a(t)** is acceleration as a function of time; over a small enough time frame this is effectively a linear function. By the {secant approximation}, $\Delta\mathbf{v} \approx \overline{\mathbf{a(t)}}\int_{t_0}^{t_1}\mathbf{dt} = \mathbf{a}\left(\mathbf{t_0} + \frac{\Delta t}{2}\right)\cdot(\mathbf{t_1} - \mathbf{t_0}) = \left(\mathbf{a_0} + \frac{\Delta a}{2}\right)\Delta\mathbf{t}$ (since a(t) is linear over our short time frame {further explanation?}), where $\mathbf{a_0}$ is the acceleration recorded last time, **Δa** is the calculated change in acceleration (again new value minus old value), and **Δt** is the calculated change in time (from the Timer). Then an equivalent process yields $\Delta\mathbf{d} \approx \left(\mathbf{v_0} + \frac{\Delta v}{2}\right)\Delta\mathbf{t}$**,** where $\mathbf{v_0}$ is the velocity calculated from the acceleration recorded last time.

Now, to use the accelerometer effectively for determining position, it is necessary that it be able to measure acceleration independently on separate axes; the analog accelerometer is thus disqualified, and an ADXL345 (via I2C or SPI) or similar required. Then (with the accelerometer arranged so its x-axis is the robot's forward/backward axis, and forward acceleration causes positive x-acceleration) we can calculate the change in x-axis acceleration and y-axis acceleration separately, and from these the change in x position $\Delta\mathbf{d}_x$ and the change in y position $\Delta\mathbf{d}_y$ via the approximations described above. We can do this by approximating the path as a straight line (rather than an arc as with the encoders); over the minimal time frame used, we can assume that the two accelerations are directly translated into speed and therefore distance changes. We cannot use these directly, since the accelerometer xy frame is not necessarily the same as the field xy coordinate system. Instead we must use the change in (robot-centered) x / y to calculate in robot angle $\Delta\theta = \tan^{-1}\frac{\Delta d_y}{\Delta d_x} = \frac{\Delta d_y}{\Delta d_x}$ (since our time frame is minimal, $\tan^{-1} x \approx x$) and the distance traveled $\Delta\mathbf{d} = \sqrt{\Delta\mathbf{d}_x^2 + \Delta\mathbf{d}_y^2}$.

For methods of interpretation of camera images, see PositionDetermination. For efficiency and avoiding the tie-up of too much processing time in the intensive image-analysis computation, the camera should be consulted less often than the other sensors.

Once all these values are calculated, we can calculate the robot's change in position in several combinations: encoder and accelerometer distances each combined with encoder, gyro, and accelerometer angle changes. (Camera gives only absolute positions, not changes, and must be treated separately—see below.) Now, we use these 12 different values (6 for x and 6 for y) and weighted-average them (independently for x and y, of course). The weights should be based on relative accuracies of the different combinations and must be tuned by experiment.
Then we add the average calculated changes in x and y, add them to our previously calculated position (or the original known position in the first run-through) to get the current position. In iterations that consult the camera (with a successfully analyzed image), the coordinates calculated from the image data can then be weighted-averaged in with the other data, again with a weight that reflects the camera's relative accuracy.

Further Note: my coordinate system of choice (with the z axis pointing up; units are inches):


(0,324)
Blue Alley
Y axis
Blue Key
FIRST
Red Key
Red Alley
(0,0)
X axis
(648,0)