IB Higher Level Mathematical Exploration

Finding the Formulas That Allow a Vehicle to Track its Global Position Using Only the Rotation of Each Wheel

INTRODUCTION

In the realm of competitive robotics, the difference between a masterful victory and a crushing defeat can be just a millimetre. Precision and consistency reign supreme in controlling of the robot. This is especially true in programming challenges where a robot may need to travel long, winding distances to complete an objective. During these moments, programmers who use the complex odometry libraries find themselves with a significant advantage. This is because odometry is a system of using only wheel inputs from any vehicle to track that vehicle's position and heading. It is akin to a GPS system but based solely on the robot's dimensions and wheel-rotations.

Rationale

As a competitor myself, I have taken advantage of free code packages that automatically implement odometry tracking on my robots, however, they harbour many limitations, the main one being their lack of flexibility. The majority of existing odometry implementations use a specific, inefficient project structure, or may sometimes require the input of additional sensors for full tracking potential. To avoid these inconveniences, and to satisfy my curiosity behind the inner workings of these odometry libraries, I will derive the relevant equations necessary to build my own odometry implementation.

Aim

The objective of this investigation is to derive all the equations necessary to track a vehicle's heading, x-position, an y-position across any path it takes. These equations should apply to any vehicle with skid-steering or differential wheels, as long as it has a pair of wheels that are parallel to the forwards-backwards motion of the vehicle (like a car, pram, tank, truck, RC car, shopping trolley, robot, etc.).

Plan of Action

Finding the odometry equations will primarily delve into geometry. This will include trigonometry, arcs, angle-equivalence rules, coordinate graphing, and more. Many fundamental and advanced geometrical rules will play a crucial role in understanding the spatial configurations of the vehicle and its path to generalize the motion of the vehicle. In addition to this, other operators like summation ($\sum$) and piecewise functions will be used in conjunction with domain/range analysis to bridge the gap between theoretical mathematics and practical implementations.

Geometric outlining software, like Fusion360's sketching features will be used to represent each explored scenario and will be used to verify any resultant equations.

EXPLORATION

Pathing:

To begin this task, I started with basic geometrical reasoning. The path a vehicle takes to get from any point to the next is not a regular geometric shape. The path of the vehicle can be any curved line from an infinite spectrum of possibilities. I cannot use any of my mathematical knowledge on an irregular line, so I will need to break each path into small sections that can be approximated as a friendlier shape.

Figure 1.1: An example of a vehicle's path. The sketch is a bird's-eye view of the situation where the rectangle is the vehicle and the long, curving line is an example of the path that vehicle may travel. The dimensions assigned to the rectangle could be anything else.

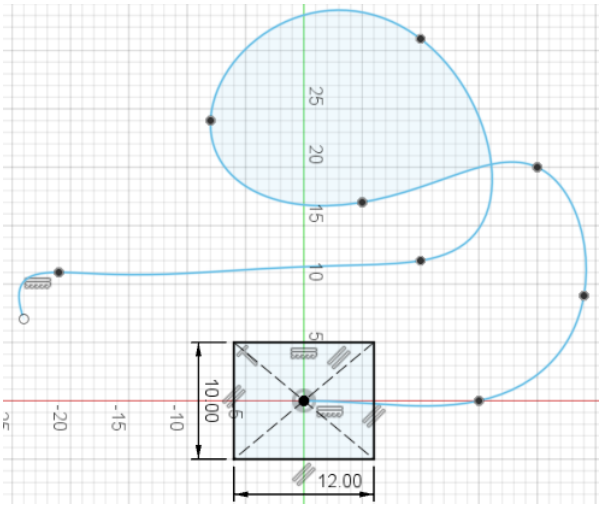
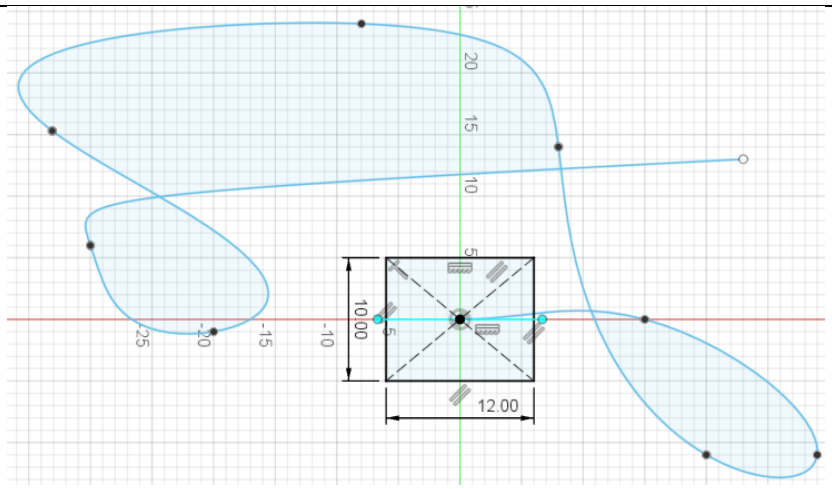


Figure 1.2: Another example of a vehicle's path. Once again, the path is random and irregular, making it difficult to use any data of the entire path.



Initially, I tried dividing any line into smaller, manageable straight lines, but this was unsuccessful because the lines would have angles between them that were difficult to solve for. Therefore, I divided the vehicle's path into arcs. Arcs allow me to take distance, curve, and angle change into account much more simply.

Figure 2.1: A section of a vehicle's path

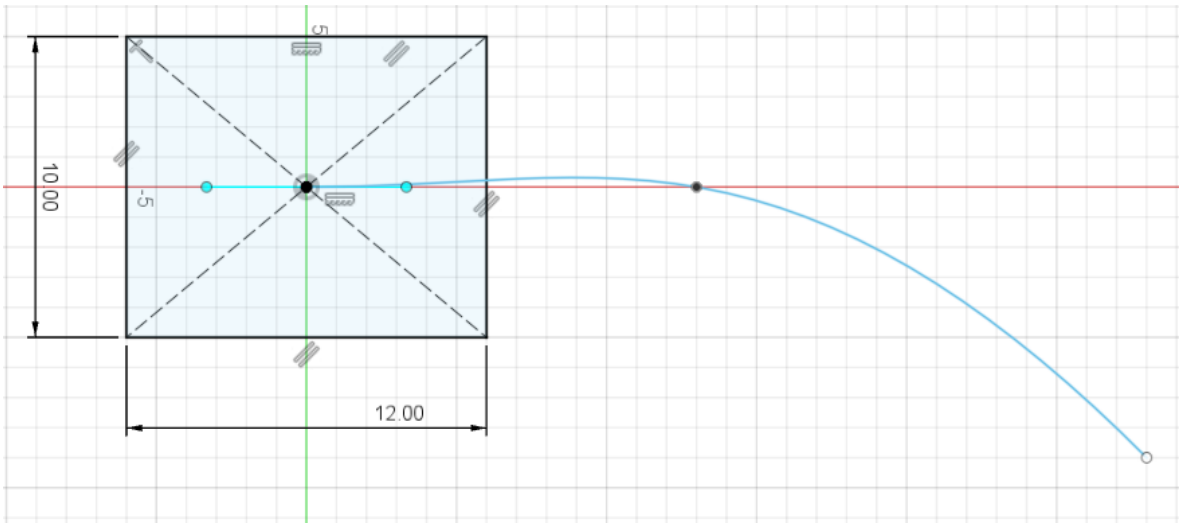
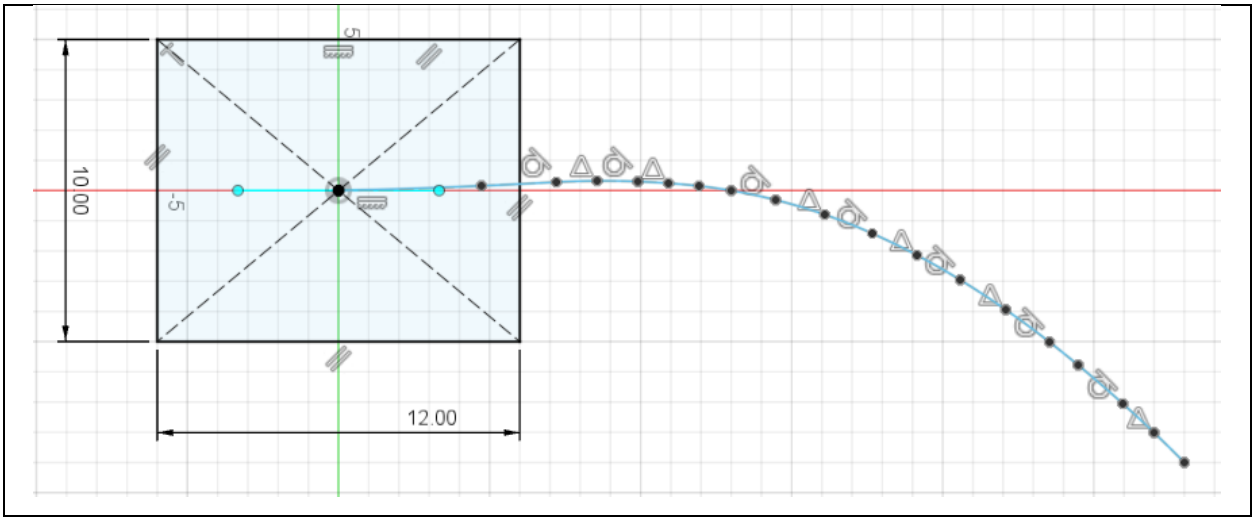
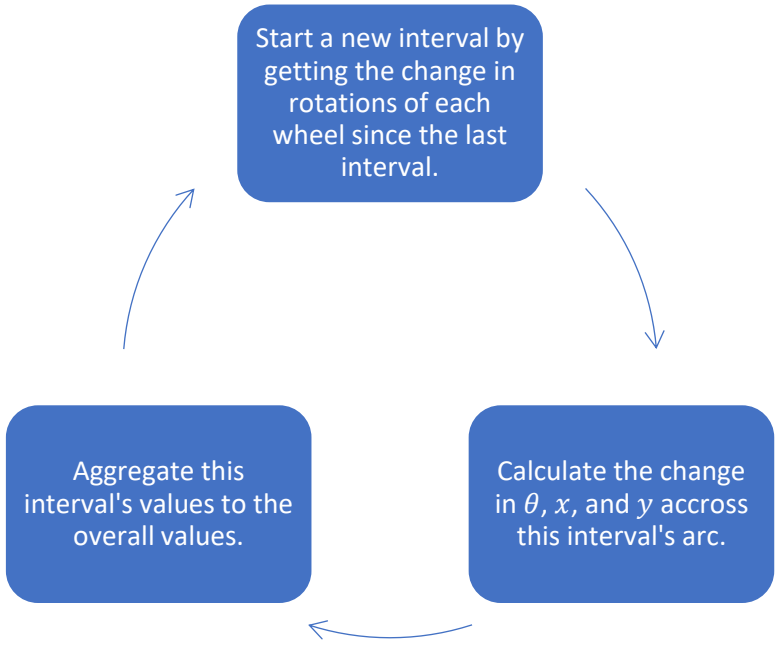


Figure 2.2: This same section as a series of arcs. Each black point on the path is the endpoint of an arc (both the end of one arc and the beginning of another). In this way, a series of arcs are excellent at approximating a complex line.



In this way, any vehicle only needs to assume its travelling in an arc for the last few milliseconds (interval length), then perform some calculations over that interval to determine the current arc's change in x-position, y-position, and angle, and sum up these values to all the previous values to update its total positioning. After this, the vehicle should calculate another arc for the next interval and repeat the process.

Figure 2.3: The intended use for the upcoming formulas



Wheel-sensor values into path-distances:

To find the 'calculations over that interval', much more geometry was needed. Firstly, I needed to determine the distance a wheel has travelled given only the rotation and dimensions of that wheel. This will come from the formula for the circumference of a circle.

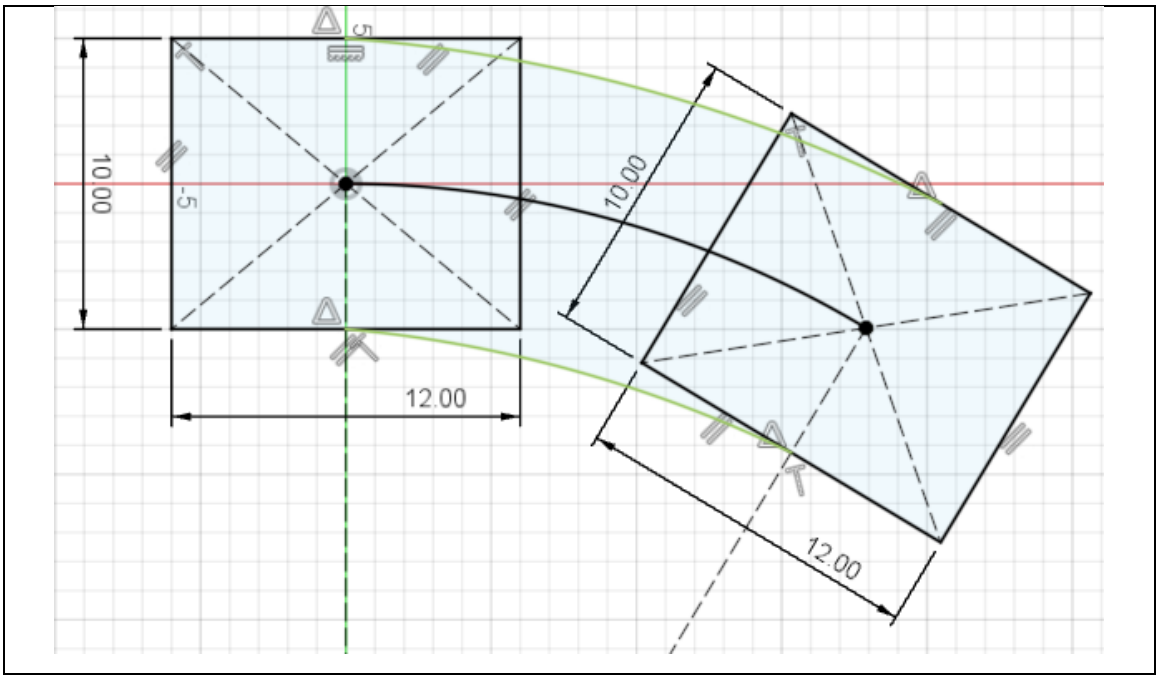
$$C = 2\pi r$$

I know the radius, r, of the wheel on any given vehicle for this purpose, so I can get the distance, D, the wheel travels by multiplying the circumference with the number of rotations, R (for a $540°$ rotation, $R = 1.5$). This R is also the value I get from a given wheel's sensor, coming from one of my 'givens'.

$$D = CR$$

$$D = 2\pi rR$$

Figure 3: A bird's-eye view of a vehicle (the rectangle) following the path of an arc. The black arc is the robot's path, the green arcs are different-length arcs that were followed by the right and left wheels of the vehicle. Each arc follows the formula $D = 2\pi rR$ where D is the length of the arc, and R is the number of rotations from the sensor (different between the right and left sides).



Finding vehicle heading:

Each arc will also follow the arc-length formula where D is still the length of the arc, r is the radius of that particular arc (unknown), and θ is the angle that arc covers (unknown):

$$D = r\theta$$

This formula involves θ, a convenient variable because the change in heading of the vehicle is the same as the internal angle of that arc (since the heading of the vehicle will be represented just like the unit circle). However, in the above formula, I have two unknowns, r and θ.

Fortunately, I will also have two equations because both the left and right arcs are different from each other while their angles remain the same.

$$D_L = r_L\theta \qquad D_R = r_R\theta$$

$$2\pi r R_L = r_L\theta \qquad 2\pi r R_R = r_R\theta$$

To further solve this into a system of equations, I will need another variable in common between the left and right arcs. This will happen by the observation that each arc's radius constitutes the central path-arc's radius (r_p) and either and addition or subtraction of half the vehicle's width (w) for the left and right sides respectively (can be seen in Figure 3).

$$r_L = r_p + w \qquad r_R = r_p - w$$

$$2\pi r R_L = (r_p + w)\theta \qquad 2\pi r R_R = (r_p - w)\theta$$

Solving for r_p in each equation:

$$2\pi r R_L = (r_p + w)\theta \qquad 2\pi r R_R = (r_p - w)\theta$$

$$\frac{2\pi r R_L}{\theta} - w = r_p \qquad \frac{2\pi r R_R}{\theta} + w = r_p$$

Now, setting both equations equal to cancel r_p and solve for θ:

$$\frac{2\pi r R_L}{\theta} - w = \frac{2\pi r R_R}{\theta} + w$$

$$\frac{2\pi r R_L}{\theta} - \frac{2\pi r R_R}{\theta} = 2w$$

$$\frac{2\pi r R_L - 2\pi r R_R}{\theta} = 2w$$

$$\frac{2\pi r R_L - 2\pi r R_R}{2w} = \theta$$

This formula allows a vehicle to calculate its change in heading when travelling in an arc using only the number of rotations in each wheel and some dimensions of the vehicle. The formula above is also generalizable to any arc the vehicle may follow. This is because, since the width of the vehicle will never be 0 and none of the other variables have bounds, the domain of the function is infinite. This formula will also effectively track arcs that turn in an anti-clockwise manner (unlike Figure 3) and will just represent those changes as negative.

In order to make this formula better standardised for future calculations, I have multiplied the left side by -1. This is because the formula currently breaks the convention of the unit circle where anti-clockwise rotation is an increase in motion. Multiplying by -1 will effectively keep the magnitudes of the results the same, it will only flip the direction of increasing angle.

$$\frac{-2\pi r R_L + 2\pi r R_R}{2w} = \theta$$

$$\frac{2\pi r R_R - 2\pi r R_L}{2w} = \theta$$

Finding vehicle displacements:

Figure 4: an extension of the current path modelling which includes the lines for x (17.889) and y (4.965) displacement as well as a chord subtending the arc that represents the hypotenuse (18.575) of the triangle (in blue) formed with x and y.

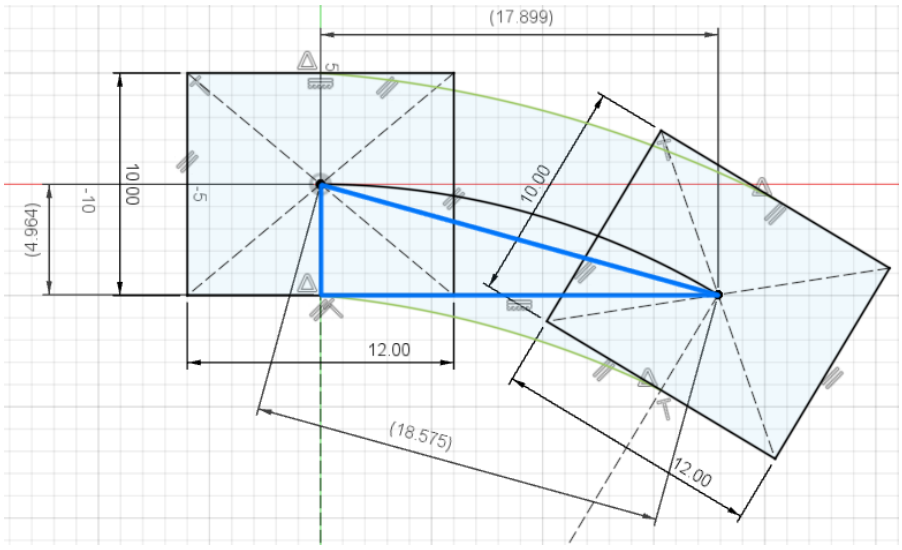


Figure 4 demonstrates another convenient geometry of this system: the x and y displacements of the arc-path form a right-triangle with the chord (h) that subtends the arc. This means I can use SOH-CAH-TOA trigonometry to solve for these values:

$$x = h\cos\theta \qquad\qquad y = h\sin\theta$$

Now, the cosine formula can be used to solve for h by solving on the triangle created with the radii of the arc and the chord.

$$h^2 = {r_p}^2 + {r_p}^2 - 2r_p r_p \ \cos\theta$$

$$h = \sqrt{2{r_p}^2 - 2{r_p}^2\cos\theta}$$

$$h = \sqrt{2{r_p}^2(1-\cos\theta)}$$

I have not solved for any of the arc's radii yet because I cancelled them in the initial system of equations. Choosing either the right or left arc formula would solve r_p. Here, I have selected the left side:

$$\frac{2\pi r R_L}{\theta} - w = r_p$$

This equation has a limitation in the fact that the sign of θ matters. In a path's arc, whether the vehicle turned $-20°$ or $20°$ should not matter because the arc and its chord will have the same dimensions. However, the dimensions of the chord's length will be incorrect if the angle is negative in this formula, therefore, the θ needs to be surrounded in an absolute value operator.

$$\frac{2\pi r R_L}{|\theta|} - w = r_p$$

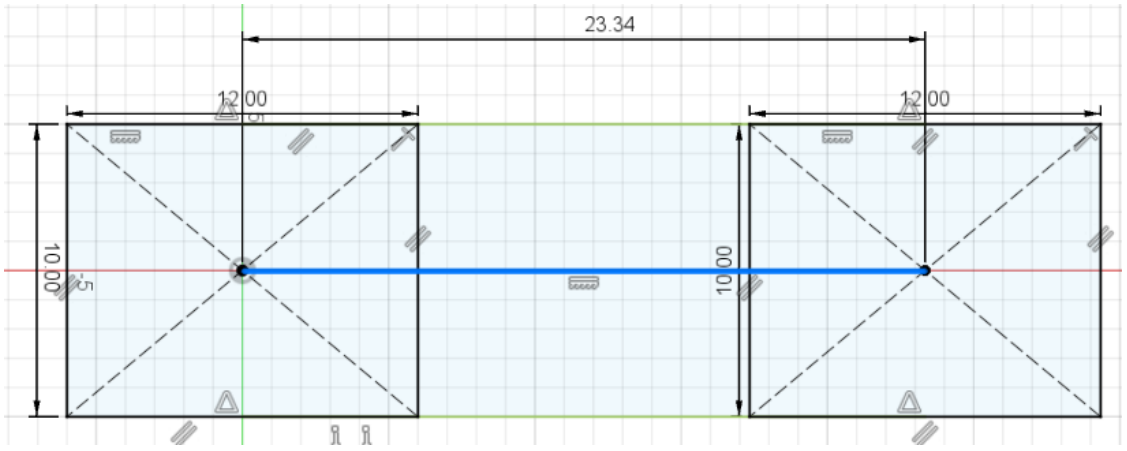
Substituting to solve for h, then also substituting into the equations for x and y:

$$h = \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2(1-\cos\theta)}$$

$$x = \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \cos\frac{\theta}{2} \qquad y = \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \sin\frac{\theta}{2}$$

These equation work well for almost all paths, but they have one small issue: their domains are restricted. This is because, since θ is in the denominator, a result of no angle change will cause h to be undefined, ruining the calculations of the interval that the vehicle moves on. To solve this, both x and y can use a special-case-handling using a piecewise definition.

Figure 5: the vehicle moving in a straight line, an exception case for the main formulas



The original equation is still valid for any other angle, but if the robot travelled in a straight line without turning, the expression for determining h would have to be just the distance travelled by any one of the wheels.

$$x = \begin{cases} \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \cos\frac{\theta}{2}, & \theta \neq 0 \\ 2\pi r R_L \cdot \cos\frac{\theta}{2}, & \theta = 0 \end{cases}$$

$$y = \begin{cases} \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \sin\frac{\theta}{2}, & \theta \neq 0 \\ 2\pi r R_L \cdot \sin\frac{\theta}{2}, & \theta = 0 \end{cases}$$

I have kept the sine and cosine functions for each of the special straight-line cases for future reference since their inclusion still maintains proper results.

In this way, the value for y when the robot moves straight will always be 0 because $\sin\frac{0}{2} = 0$ and the value for x will always be the distance that the wheel travelled ($2\pi r R_L$) because $\cos\frac{0}{2} = 1$. This maintains the consistency with unit-circle conventions where 0° is a horizontal heading.

Arc Formulas:

From the above calculations, using only the radius of the vehicle's wheel, r, the number of rotations of each wheel's sensor, R_L and R_R, and the width of robot, $2w$, one can determine the vehicle's change in heading, θ and cartesian position, x and y, when moving in an arc.

$$\frac{2\pi r R_R - 2\pi r R_L}{2w} = \theta$$

$$x = \begin{cases} \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \cos\frac{\theta}{2}, & \theta \neq 0 \\ 2\pi r R_L \cdot \cos\frac{\theta}{2}, & \theta = 0 \end{cases}$$

$$y = \begin{cases} \sqrt{2\left(\frac{2\pi r R_L}{|\theta|} - w\right)^2 (1 - \cos\theta)} \cdot \sin\frac{\theta}{2}, & \theta \neq 0 \\ 2\pi r R_L \cdot \sin\frac{\theta}{2}, & \theta = 0 \end{cases}$$

These formulas can also be verified for special cases of non-arc pathing.

If the vehicle travels in a straight line, the geometry no longer follows an arc. Fortunately, the math and formulas above still work. The change in angle will be correctly represented as 0 (because R_L and R_R are equal, so the numerator turns to 0). The value of x will also correctly be 0, as handled by the piecewise operator, and the value for y would also be correct, since it would use the raw distance that the wheel travelled.

If the vehicle turns in place (as some differential-drive robots can), the formulas still work because the wheels would still rotate, giving an accurate angle. The geometric arc representation would just have zero-length radii while still maintaining the proper central angle. The $\frac{2\pi r R_L}{\theta}$ that both the x and y equations share would always equal w in this special case of turning-in-place, meaning that $\left(\frac{2\pi r R_L}{|\theta|} - w\right)$ is 0 and that both entire expressions are 0.

Arcs into paths:

As previously assumed, and shown in Figures 1.1 and 1.2, the current formulas only apply to any small segment of a vehicle's path modelled as an arc. For these formulas to work on a full path, one must aggregate all the small arcs' changes that the path is modelled at. For example, the total, global value of the vehicle's heading θ_G will need to be the sum of every arc's θ:

$$\theta_G = \theta_1 + \theta_2 + \theta_3 + \cdots + \theta_n \text{ for } n \text{ intervals in the path}$$

$$\theta_G = \sum_{n=0}^{intervals} \theta_n$$

$$\theta_G = \sum_{n=0}^{intervals} \frac{2\pi r R_{R_n} - 2\pi r R_{L_n}}{2w}$$

The total, global value of the vehicle's x-position x_G will need to be the sum of every arc's x:

$$x_G = x_1 + x_2 + x_3 + \cdots + x_n \text{ for } n \text{ intervals in the path}$$

$$x_G = \sum_{n=0}^{intervals} x_n$$

$$x_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \cos\frac{\theta_n}{2}, & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \cos\frac{\theta_n}{2}, & \theta_n = 0 \end{cases}$$

And the total, global value of the vehicle's y-position y_G will need to be the sum of every arc's y:

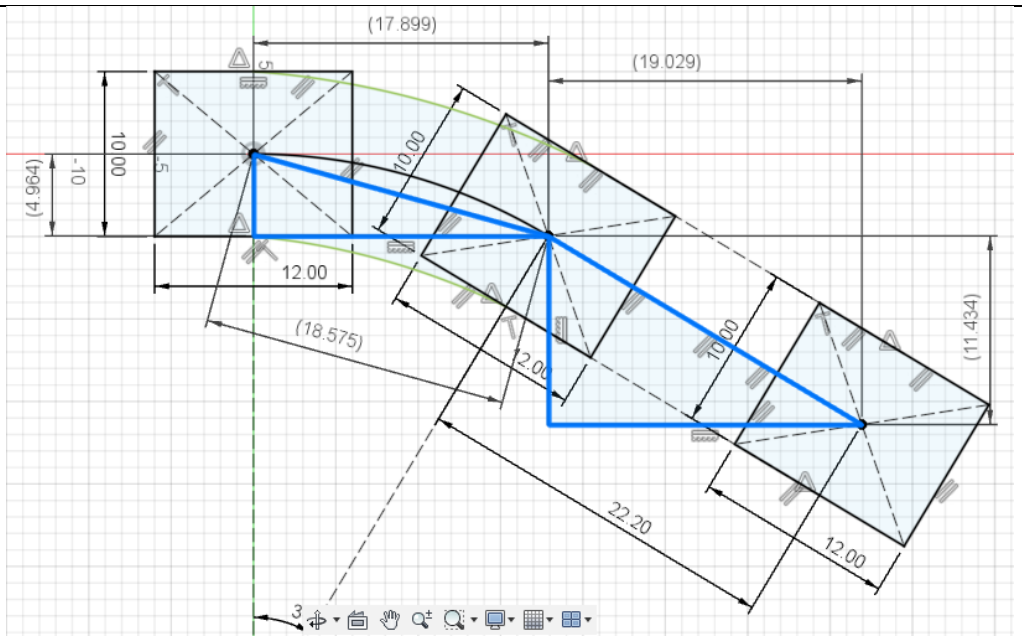
$$y_G = y_1 + y_2 + y_3 + \cdots + y_n \text{ for } n \text{ intervals in the path}$$

$$y_G = \sum_{n=0}^{intervals} y_n$$

$$y_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \sin\frac{\theta_n}{2}, & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \sin\frac{\theta_n}{2}, & \theta_n = 0 \end{cases}$$

Through this, another problem arises. While the value of θ_G will aggregate correctly in accordance with the vehicle's path, x_G and y_G will be calculated incorrectly. Figure 6 demonstrates a possible case of failure:

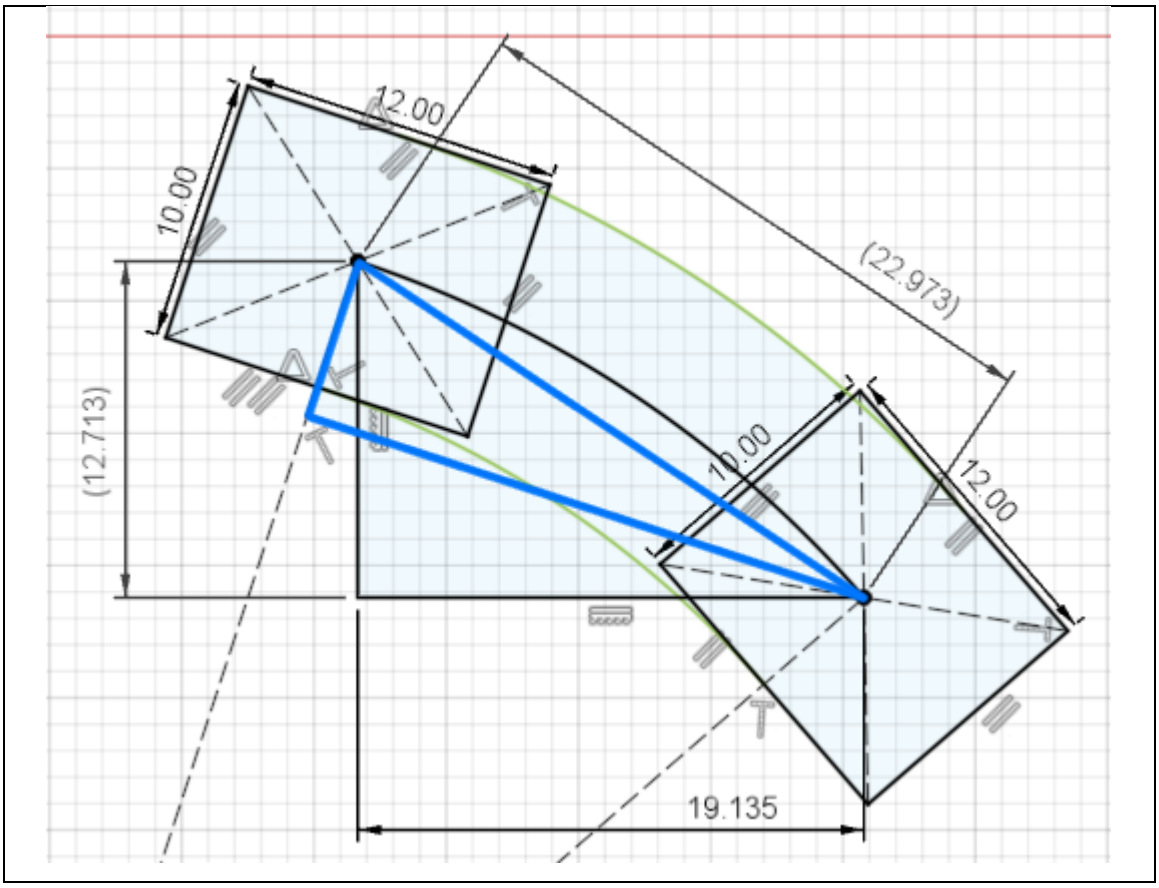
Figure 6: a path with a combination of intervals can mess up the formulas.



In this image, the initial values of x and y will be correct after the first interval's arc, but in the next interval, as the vehicle moves in the straight line, the y value that the formula would return is 0, meaning y_G remains the same. This wrong because even as the robot is moving in the straight line, the y_G should still change since the robot is moving in a diagonal trajectory upwards.

To solve this issue, the initial heading of the vehicle also needs to be taken into account. Expanding the geometry of this situation and applying some transversal equivalency rules, it is found that every arc's x and y result needs to include the vehicle's initial heading at the start of the arc as part of the calculation.

Figure 7: another arcing path, but the robot starts at an angle of 18.1 degrees. In this case, the formulas above would not yield the correct change in the global x and y positions. This is because the blue triangle does not match the actual triangle of x (19.135) and y (12.713) displacement.



To do this, the section of the formula that converts the h value into the x and y components, $\cos\frac{\theta_n}{2}$ and $\sin\frac{\theta_n}{2}$ respectively, will also add the total heading of the robot before the arc (θ_G). This new consideration will also need to be applied to the straight-line driving situation (when $\theta = 0$). This is because, currently, the formula always results in 0 for the y value during $\theta = 0$ which is wrong for the cases that the robot moves in a straight line either vertically or diagonally. Straight lines will need to multiply the h value with cosine and sine of θ_G.

$$x_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \cos\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \cos\theta_G, & \theta_n = 0 \end{cases}$$

$$y_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \sin\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \sin\theta_G, & \theta_n = 0 \end{cases}$$

Implementation:

Now that all the special cases have been covered and that the vehicle's path can be tracked as a series of small arcs and lines, these are the equations that successfully use only the rotations of each wheel and the dimensions of the vehicle's wheel and width to derive its heading, x position, and y position:

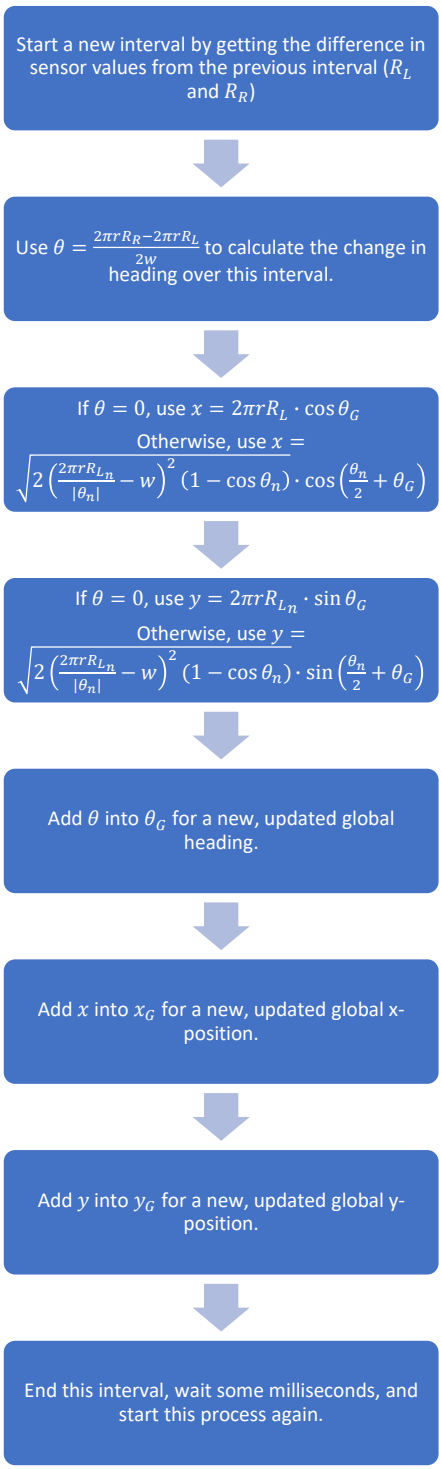
$$\theta_G = \sum_{n=0}^{intervals} \frac{2\pi r R_{R_n} - 2\pi r R_{L_n}}{2w}$$

$$x_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \cos\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \cos\theta_G, & \theta_n = 0 \end{cases}$$

$$y_G = \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \sin\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \sin\theta_G, & \theta_n = 0 \end{cases}$$

To implement these equations in real-life, the vehicle needs to have a pair of wheels that are parallel to the forwards-backwards motion of the vehicle, include rotation sensors (encoders) on those wheels, and have known width and wheel-radius dimensions. After this, the main way to use the equations would be to run them through code. Each summation (∑) would be represented by a continuous loop, and the piecewise operator would be a conditional statement. While tracking position, each interval should be as small as reasonably possible, (perhaps less than 20 milliseconds long). In this way, a long, intricate path would be treated as thousands of tiny arcs and straight lines.

Figure 8: a flowchart representing the general process of tracking a vehicle's position using the equations above.



CONCLUSION

Overall, this exploration was a success. I was able to use only information about a vehicle's left and right wheel's rotations to determine that vehicle's live heading, x-position, and y-position no matter what path the vehicle takes as it moves.

I have also been able to test the validity of these formulas with several varying trials in Fusion360's sketching tools. Here, I can draw out any path, create the arcs/lines on that path for each interval, and measure the resulting $2\pi rR$ values to plug-into the formulas. I can then check the equation's result by measuring the difference in x and y positions between the starting point and the final point. There has not been any failure in all the regular functionality, special scenarios, or edge-cases that I could make, demonstrating a practical support for these odometry formulas.

Extensions

Although these equations are great at accomplishing the initial outlined task, they can be developed further to enhance functionality. One example of this is the consideration of an offset in the vehicle's starting position. In many competition scenarios, it is convenient to assign your robot as starting at a different heading or coordinate position than $0°$ and $(0,0)$. This is helpful when setting a reference landmark as the origin and beginning the robot's path somewhere else. The implementation of this quite simple: just add the assigned initial heading (θ_i), x (x_i), and y (y_i) to their respective equations as constants outside the summation.

$$\theta_G = \theta_i + \sum_{n=0}^{intervals} \frac{2\pi r R_{R_n} - 2\pi r R_{L_n}}{2w}$$

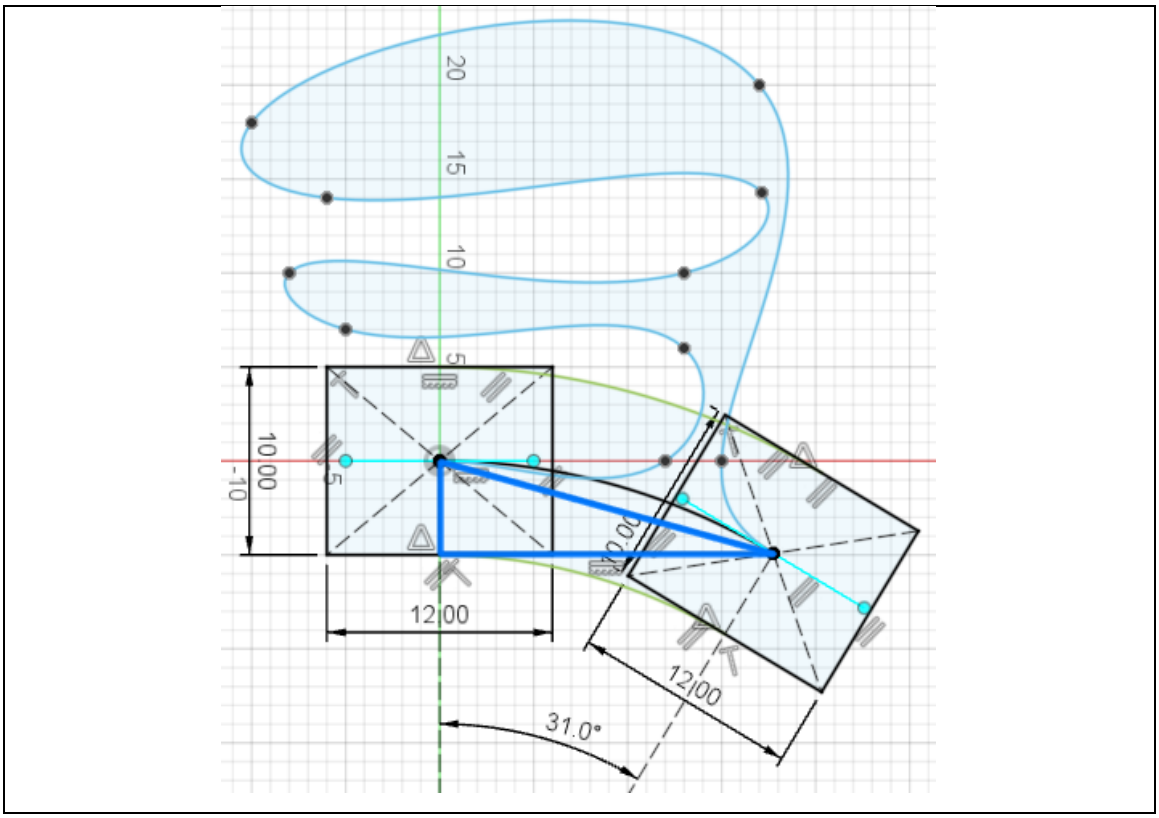
$$x_G = x_i + \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \cos\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \cos\theta_G, & \theta_n = 0 \end{cases}$$

$$y_G = y_i + \sum_{n=0}^{intervals} \begin{cases} \sqrt{2\left(\frac{2\pi r R_{L_n}}{|\theta_n|} - w\right)^2 (1 - \cos\theta_n)} \cdot \sin\left(\frac{\theta_n}{2} + \theta_G\right), & \theta_n \neq 0 \\ 2\pi r R_{L_n} \cdot \sin\theta_G, & \theta_n = 0 \end{cases}$$

Another extension to this exploration would have been an evaluation of the possibility of asynchronous odometery. As mentioned in the *Pathing* and *Implementation* sections, this current system relies on the computer to valuate a new interval every few milliseconds in order to simplify a complex, irregular line into more manageable and friendly sections. The main disadvantage of this is that there is no guarantee that the vehicle will fully follow an arcing or straight path in any interval. Rather, the vehicle is likely not following any arcing motion, but our intervals are small enough that the approximation of whatever motion into arcs still yields a satisfactory result. However, after long-use of the same odometry session, a buildup of error may occur where the little deviances from ideal behaviour add up to a large inaccuracy.

In this case, a mathematical system that uses only the final R_R and R_L values after any complex path may prove to be more accurate and computationally efficient. I tried deriving equations that could use the overall wheel-rotation values after the completion of any path but was not able to. Accomplishing asynchronous odometry will likely require the discovery of some relationship between R_R and R_L regardless of the path length, shape, or behaviour.

Figure 9: Can the ending position of a path like this be determined only with the overall rotations that it took the right and left wheels to get there?



Significance

Although the concept of odometry is not my idea, and neither are some concepts around its derivation (like breaking a path into arc-intervals), I have not been able to find this set of equations published anywhere. As mentioned in the *Rationale* section, almost all odometry implementations or explanations I have come across use additional sensor input or have inefficient (and inconsistent) systems. On the other hand, the equations found here are portable to any calculating system and to any vehicle just by changing the r and w constants. These formulas could even be implemented in a spreadsheet! By importing the vehicle's sensor data into a spreadsheet and creating some calculation columns with these formulas, the spreadsheet will be able to evaluate intervals and sum them up!

Additionally, the fact that these formulas are not just for robots or cars but could be used in any mobile base is another important feature. Just installing some rotation

sensors on the wheels of a shopping trolley, a tricycle, or even a battle tank will still yield valid results (as long as the wheels don't slip, and the ground is flat). The odometry formulas here can be especially helpful to autonomous robots in stable conditions. Currently, many industrial robots will use ultrasonic sensors, object-detection, apriltags, and more to track their position because those methods are more reliable than wheels that may slip, get stuck, or traverse bumps. However, as the quality of hardware in those robots improve, so will the capabilities of wheel-based odometry systems. Robotic applications with no need for significant redundancy in controlled environments may benefit from the lower complexity and cheaper expense of this odometry system.

Overall, whether or not this odometry system is implemented in a child's tricycle, a hobbyist's car, or a businesses' machines, it will definitely boast a competitive advantage after being implemented in my next tournament robot.

Candidate declaration: I confirm that this work is my own work and is the final version. I have acknowledged each use of the words or ideas of another person, whether written, oral or visual.

Candidate's signature: *SG* Date: *March 21, 2024*