# Geometric Modelling in Robot Motion Control

Research Question: How can mathematical models govern robot controls in the real world?

Allen Tao

Colonel By Secondary School

August 30, 2020

## Table of Contents

**Abstract**

Autonomous robot control in today's world is largely governed by mathematical models and algorithms in order to achieve consistent and repeatable movement. These models are critical as the robot should be able to adjust itself under the influence of random error accumulated throughout the procedure. An example of this could be a Roomba robotic vacuum vacuuming the house, but one of its powered wheels slip against the floor due to dust accumulated on its wheels, thus by extension, should the robot be programmed simply to run its drive motors for 2 seconds to move forwards, it will veer off course if its wheels slip, and not be able to adjust. Thus, an effective autonomous robot must be able to correct its course despite these random variances.

This paper aims to geometrically model an absolute position tracking system that models the robot's coordinates on a coordinate grid relative to its starting position calculated by the changes in the distances traversed by the drivebase wheels. This will then be used to describe the current and set the target robot position for the motion algorithm that governs how the robot should move in order to get from its current position to its final position. Since the absolute position tracking system reports the robot's coordinates relative to its starting position constantly, if the robot veers off course, this will be tracked, and accounted for when a different current position is fed into the motion algorithm, informing a different path from a different starting point.

**Introduction to Odometry**

Another name for absolute position tracking is *odometry*. In this case, the robot's coordinate position and where it is pointing is tracked relative to its starting position. To do this, 3 unpowered tracking wheels are added onto the drivebase (figure 1). Tracking wheels are free spinning wheels that are contacting the ground, attached to an optical shaft encoder that records how much the shaft where the wheel is mounted rotates. Given the change in rotation of the shaft recorded by the optical shaft encoder, and knowing the tracking wheel diameter, the distance the tracking wheel has traversed can be calculated using the arc length formula, where $\theta$ denotes the change in rotation of the shaft, and the radius being half the diameter of the wheel.

$$s = r\theta$$

My chassis setup for the robot I am currently building is detailed in figure 1 for clarity. In figure 2, the same drivebase is labelled with relevant variables used throughout this section.

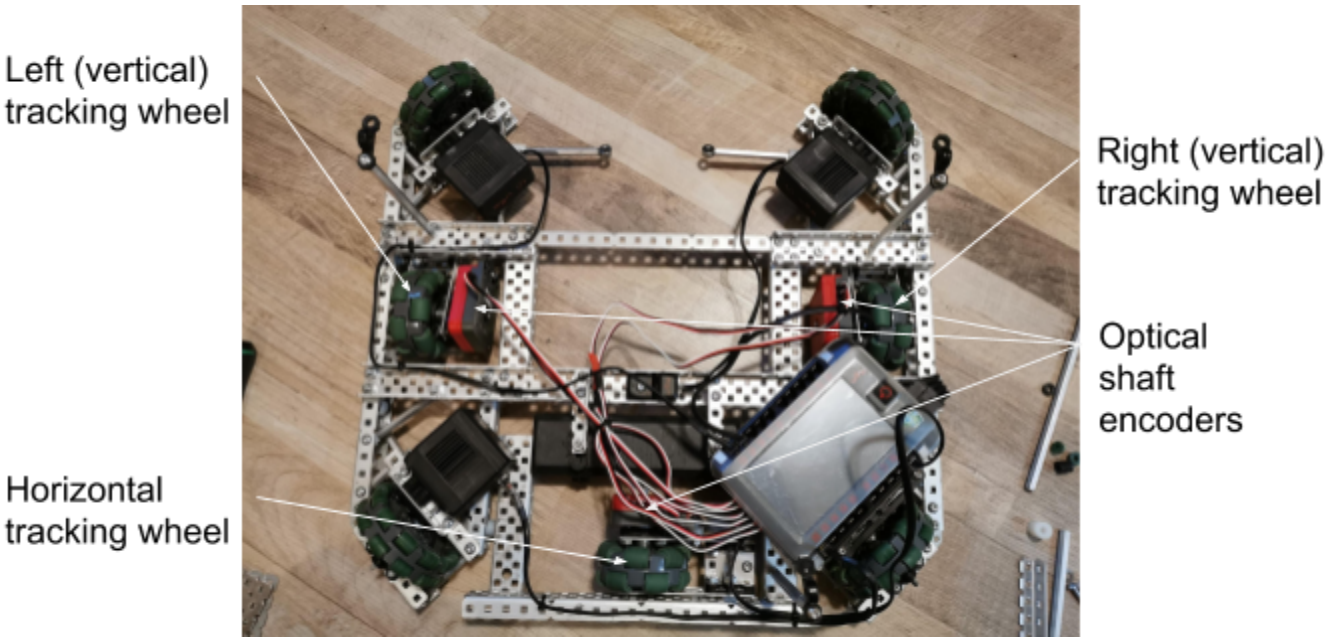


**Figure 1:** Tracking wheel setup on my robot chassis

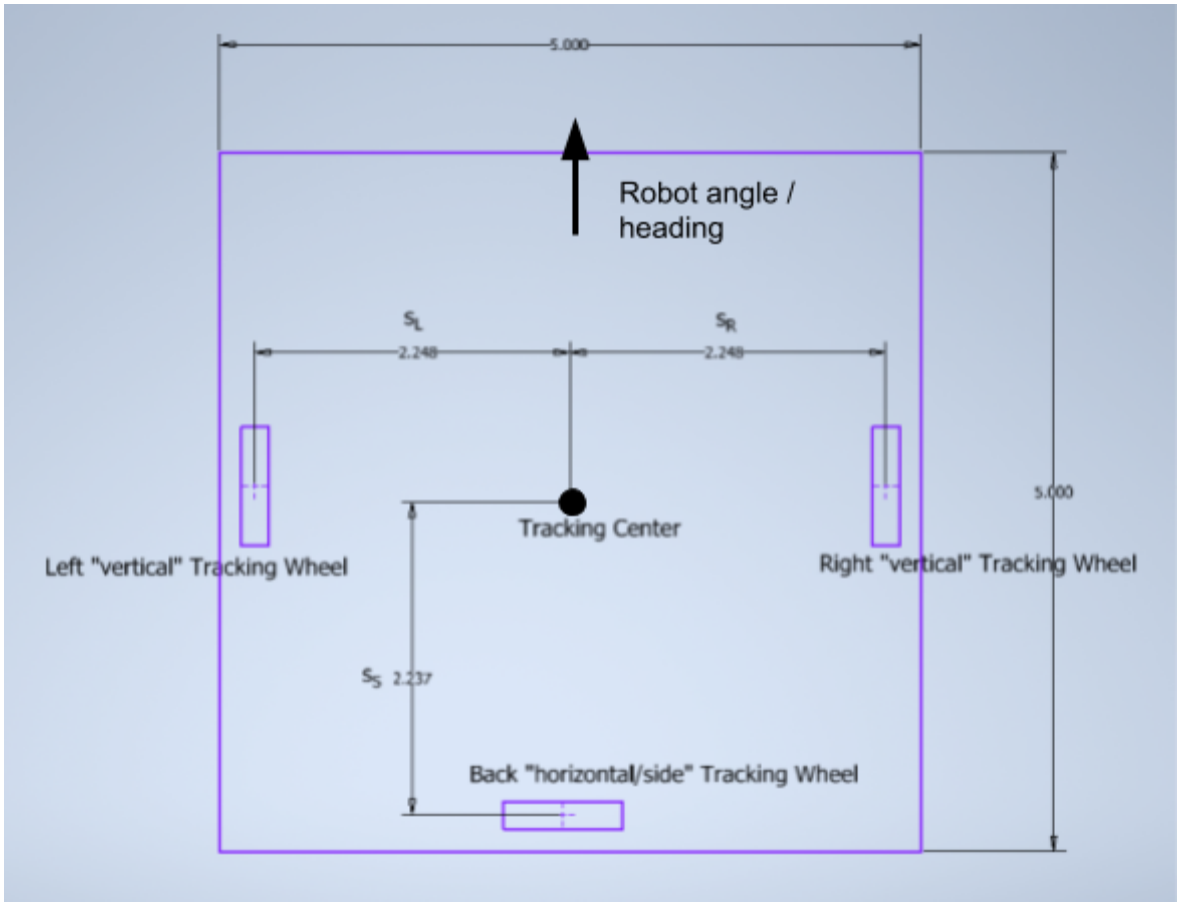


**Figure 2:** Geometric sketch of drive base setup with tracking wheels, and variable terms

The **tracking wheels** are the wheels whose rotations are being tracked by the optical shaft encoder sensors. The **tracking center** is the point we wish to know its position and heading relative to its starting position on the coordinate plane. To model the robot's position numerically on the cartesian plane, let $x$ be the initial x position of the robot on the coordinate plane, $y$ be the initial y position on the plane, and $\theta$ be the initial heading of the robot. The final x, y, and heading of the robot's position are similarly denoted as $x_2$, $y_2$, and $\theta_2$. Take note that $\theta$, the heading of the robot is relative to the y axis, meaning that when the robot is aligned with the y axis, the heading is 0 or a multiple of 360 degrees, tending positive clockwise. Finally, let $s_L$

denote the distance between the left tracking wheel to the tracking center, $s_R$ the distance between the right tracking wheel to the tracking center, and $s_S$ the distance between the back tracking wheel to the tracking center. The positive direction for the vertical tracking wheels is positive when the robot moves forwards, and the positive direction for the horizontal tracking wheel is positive when moving towards the right. Note that directional statements such as "forwards" or "right" are all relative to the robot angle/heading.

Assume for now that the robot cannot move from side to side, making it analogous to how a tank drives: forwards and backwards in arcs. Thus modelling movements with arcs from distances tracked by tracking wheels would be a good start. Let us prove that the change in robot heading will be congruent with the angle of the arc $AB$ traced by the tracking center from the arc movement in figure 3.

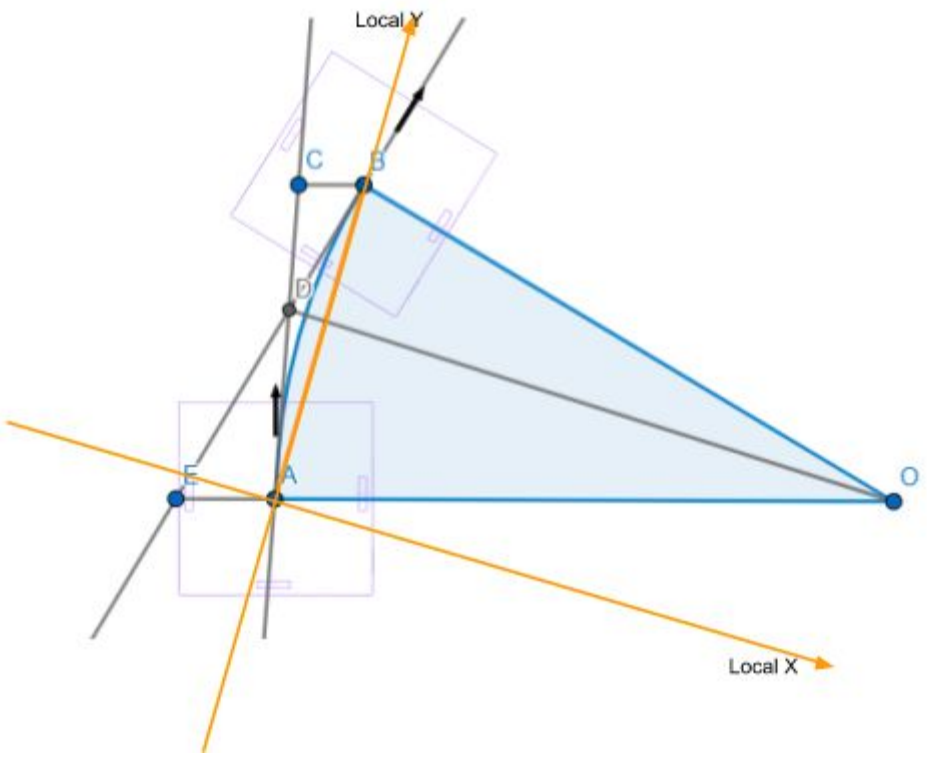


**Figure 3:** Geometric diagram for proofs 1 and 2

Point $A$ denotes the starting coordinates of the tracking center of the robot. Point $B$ denotes the end coordinates of the tracking center. Point $A$ lies on line $OE$, which is parallel to line $BC$. Line $AC$ is tangent to arc $AB$ at point $A$, and line $BE$ is tangent to arc $AB$ at point $B$. Point $D$ is the intersection of tangents $AC$ and $BE$. Since the robot cannot drift side to side, in order for the robot to reach point $B$ from $A$, its heading must have changed. This change in heading, $\Delta\theta$ is represented by $\angle CDB$, the angle of the intersection of tangents $AC$ and $BE$. Hence, prove that $\angle CDB = \angle AOB$, the latter being the angle of the arc $AB$.

**Proof 1:**

$\because$ Line $AC$ is tangent to arc $AB$ at point $A$

$\angle CAO = 90^\circ$

$\because$ Line $BE$ is tangent to arc $AB$ at point $B$

$\angle EBO = 90^\circ$

$\because$ $\angle CAO = 90^\circ$ and $\angle EAO = 180^\circ$

$\angle DAE = \angle EAO - \angle CAO = 90^\circ$

$\triangle DAE \sim \triangle OBE$ (angle angle similarity)

$A : \angle DAE = \angle EBO = 90^\circ$

$A : \angle DEA = \angle OEB$ (shared angle)

$\therefore$ $\angle ADE = \angle BOE$

By opposite angle theorem

$\angle CDB = \angle ADE$

$\therefore \angle CDB = \angle BOE$

$\therefore \angle CDB = \angle AOB$

With knowledge of this proof, let us try to calculate the change in angle from an example robot motion (figure 4).

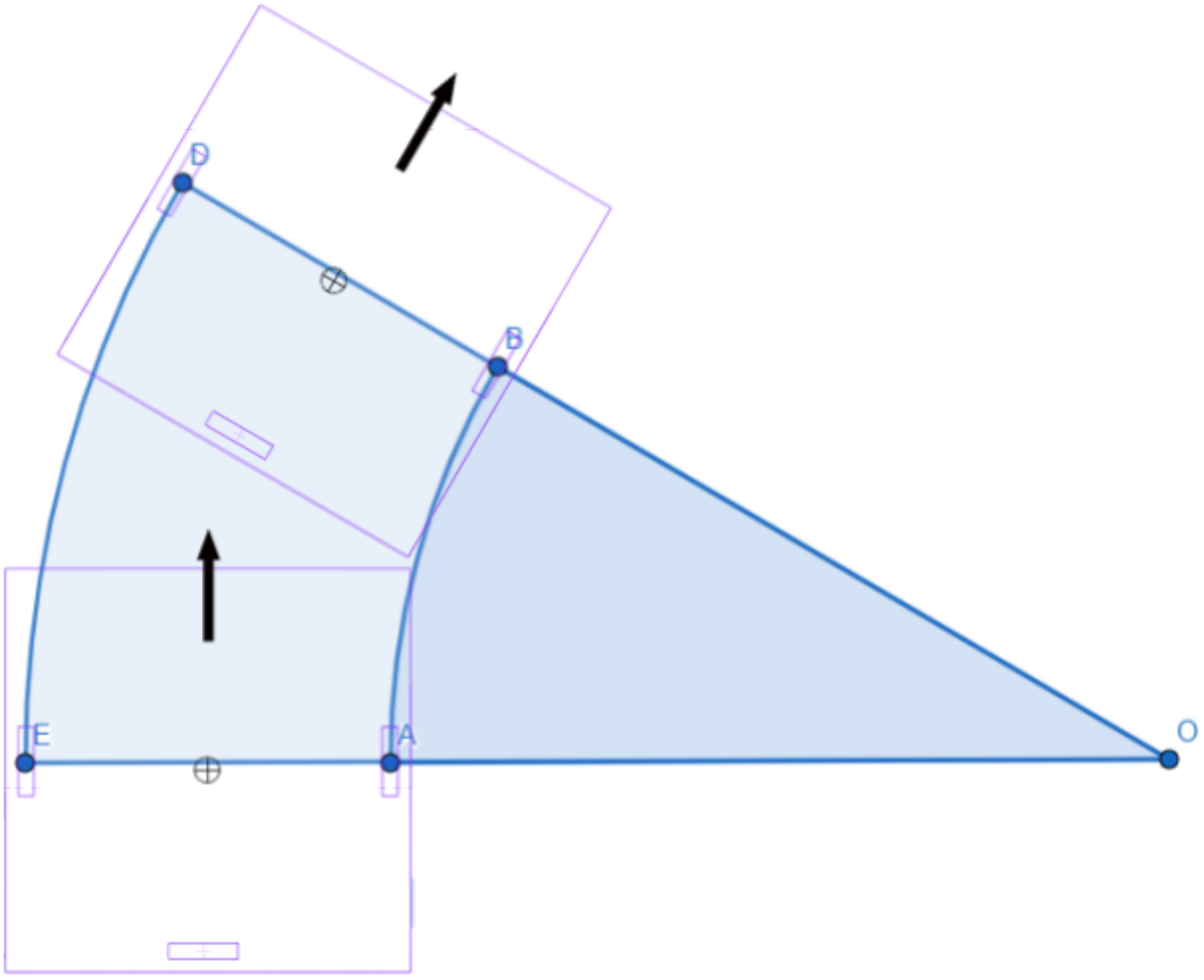


**Figure 4**: Simple arc motion from robot, similar to car making a turn

Let the distances tracked by the left and right tracking wheels (arcs $ED$ and $AB$ respectively) be denoted as $\Delta L$ and $\Delta R$, the change in the distances tracked on the left and right tracking wheels. Similarly, the distance tracked by the horizontal tracking wheel is denoted as $\Delta S$, which will be considered later. Let the radius of the arc modelling the motion of the robot be expressed as $r$. Also, let the change in the angle of the robot from point $A$ to $B$ be denoted as $\Delta\theta$. Knowing that the change in angle of the robot, $\Delta\theta$, is congruent to the angle of the arc, let us solve for it.

Using the arc length formula to solve for $r$ for both left and right tracking wheels:

$$s = r\theta$$

$$\Delta L = (r + s_L)\Delta\theta$$

$$r + s_L = \frac{\Delta L}{\Delta\theta}$$

$$r = \frac{\Delta L}{\Delta\theta} - s_L \qquad (1)$$

$$\Delta R = (r - s_R)\Delta\theta$$

$$r - s_R = \frac{\Delta R}{\Delta\theta}$$

$$r = \frac{\Delta R}{\Delta\theta} + s_R \qquad (2)$$

Substituting (1) in (2) for $r$

$$\frac{\Delta L}{\Delta\theta} - s_L = \frac{\Delta R}{\Delta\theta} + s_R$$

$$\Delta L - s_L\Delta\theta = \Delta R + s_R\Delta\theta$$

$$s_R\Delta\theta + s_L\Delta\theta = \Delta L - \Delta R$$

$$\Delta\theta = \frac{\Delta L - \Delta R}{s_L + s_R}$$

We now define a local coordinate axis, where the local y axis of the robot will be collinear and aligned with the straight line path from the robot's initial position to its final position. This is depicted in figure 3 in orange. Evidently, this local coordinate grid will be offset from $\theta$, the initial robot heading, by a certain angle. Let us prove this angle will always be equal to $\frac{\Delta\theta}{2}$, referring back to figure 3 as a geometric aid.

**Proof 2:**

As lines $BE$ and $AC$ are tangent to arc $AB$ at points $B$ and $A$ respectively:

$\angle OBE = \angle OAC = 90^\circ$

Invoking Pythagorean Theorem in $\triangle OBD$ with $\angle OBD = 90^\circ$ and $\triangle OAD$ with $\angle OAD = 90^\circ$

$$OD^2 = OB^2 + BD^2$$

$$OD^2 = r^2 + BD^2 \qquad (3)$$

$$OD^2 = OA^2 + AD^2$$

$$OD^2 = r^2 + AD^2 \qquad (4)$$

Substituting (3) in (4) for $OD^2$

$$r^2 + BD^2 = r^2 + AD^2$$

$BD^2 = AD^2$

$BD = AD$ (here, no side lengths are negative)

$\because$ In $\triangle ADB$, $BD = AD$

$\therefore$ $\triangle ADB$ is isosceles with $\angle DAB = \angle DBA$

Since $\angle CDB = \Delta\theta$, and $\angle CDA = 180^\circ$

$\angle BDA + \angle CDB = 180^\circ$

$\angle BDA = 180^\circ - \Delta\theta$

$\because$ Interior angles of a triangle sum to 180°

$\angle DAB + \angle DBA + \angle BDA = 180^\circ$

$2\angle DAB + (180^\circ - \Delta\theta) = 180^\circ$

$\angle DAB = \frac{\Delta\theta}{2}$

Therefore, the angle change between the current local coordinate system (with y axis aligned with the straight line path between the robot's starting and destination point, in orange depicted in figure 3), and the previous robot heading (relative to the global coordinate axis AC) is $\frac{\Delta\theta}{2}$.

With this, we can now calculate the local x and y displacements of the robot.

Refer to figure 5. Note how in all the calculations that we have performed, it does not account for the robot slipping sideways, or moving sideways (sideways as defined as the robot moving on

the axis perpendicular to the forward direction of the robot). To take this into account in our odometry, a similar process for calculating the arc geometry has to be done on the x component of the local coordinate system, tracking sideways movement. This is where the horizontal tracking wheel comes into the calculations.

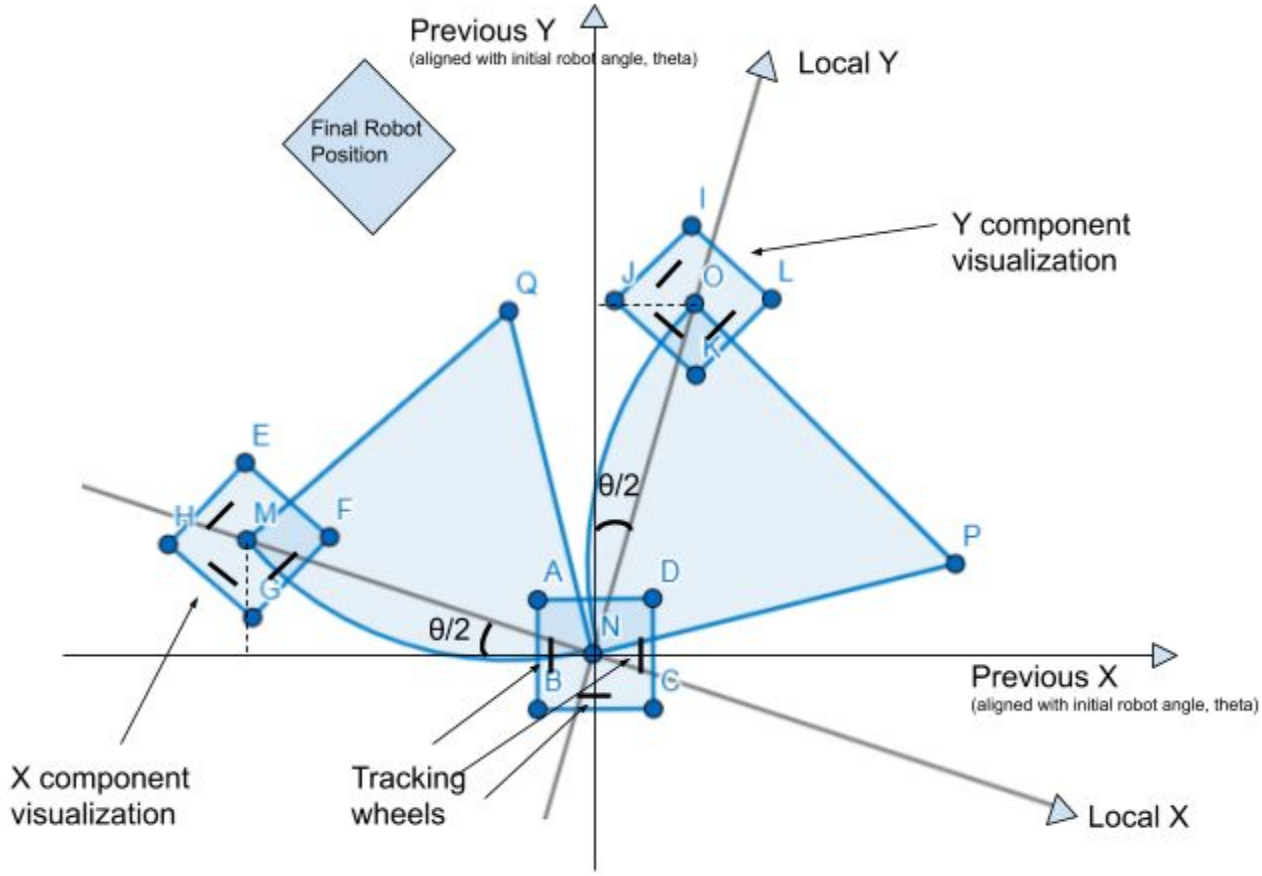


**Figure 5:** Local and global coordinate axis visualization

In figure 5, the robot is trying to drive in an arc from point N to O, however it drifts towards the left side due to momentum. This extra local horizontal displacement must be considered. Since we defined the y axis of the local coordinate system to be collinear with the straight line path from A and B, let us solve for the local y component first (line ON on figure 5).

$\because$ *ON is the chord of circular sector OPN*

Noting that the length of a chord is equal to $2rsin(\theta/2)$, where $r$ is the radius of the arc from before, and $\theta$ is the angle of the arc.

$\therefore ON = 2rsin(\theta/2)$

$$ON = 2(\frac{\Delta R}{\Delta\theta} + s_R)sin(\frac{\Delta\theta}{2}) \qquad (5)$$

Similar to expressing the arc travelled in the local y component, the horizontal offset must be added to make the horizontal arc representative of the path travelled by the tracking center, not the wheel.

$\because$ *MN is the chord of circular sector QMN*

Using the chord length formula we can solve for MN, the local horizontal displacement. Let $r_2$ denote the radius of this arc modelling the horizontal component relative to the robot.

$MN = 2r_2sin(\theta/2)$

$$MN = 2(\frac{\Delta S}{\Delta\theta} + s_S)sin(\frac{\Delta\theta}{2}) \qquad (6)$$

The final step is we now take our local y and x displacements (5 and 6 respectively), and add them to the global x and y displacements, as well as updating the global angle. This can be done with basic trigonometry.

Now, we can finally solve for $x_2,\ y_2,\ and\ \theta_2$ at every instant the optical shaft encoders update. Refer to figure 5 for geometric visualization.

$$x_2 = x + ONsin(\theta + \Delta\theta / 2) + MNcos(\theta + \Delta\theta / 2)$$

$$y_2 = y + ONcos(\theta + \Delta\theta / 2) - MNsin(\theta + \Delta\theta / 2)$$

$$\theta_2 = \theta + \Delta\theta$$

Take note that in the equation for the final y coordinate, $y_2$, the final adden of the equation is negative to cancel out negatives during the calculations from the conventions used. In the case of the example motion in figure 5, MN is negative as $\Delta S$ is negative, as the horizontal tracking wheel moves towards the left, defined to be negative.

**Motion Algorithm Model**

Armed with an odometry framework that tracks the coordinates and angle of the robot relative to the initial starting coordinates and y axis, the problem of deriving a motion algorithm boils down to "how to get the robot from point A (x, y, $\theta$) to point B ($x_2$, $y_2$, $\theta_2$)?" In designing a robot control algorithm, general principles are for the algorithm to be efficient (choosing a shorter path in place of a longer one), gentle/smooth in motions (sudden jerky movements, or wild movements are less reliable and produce more strain on the hardware), and is consistent / repeatable.

You may have noticed that in figure 1, the powered wheels were mounted at 45 degree angles (not inline on the left and right side as wheels are typically mounted, say on a car). Since the wheels have rollers on them, this allows the robot to move omnidirectionally (i.e. move in all directions, such as side to side). This side to side movement is called strafing. It is able to do this in addition to all other motions available to "tank drive" (wheels on each side inline).

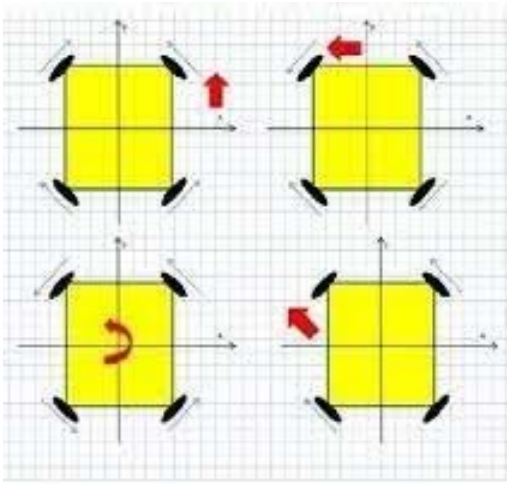

**Figure 6:** X-Drive strafing motions

By adjusting the power we send to each wheel, the robot can effectively move in all directions, like a sphere rolling around, in addition to being able to rotate. This allows maximum flexibility in robot motion in avoiding obstacles, and moving from point A to B efficiently.

The goal of this section is to have a function that takes in the x, y, and angle from the robot's initial position, provided with the destination x, y, and angle, to calculate how we should power the motors on the drivebase to actually get the robot there.

*Modelling the errors*

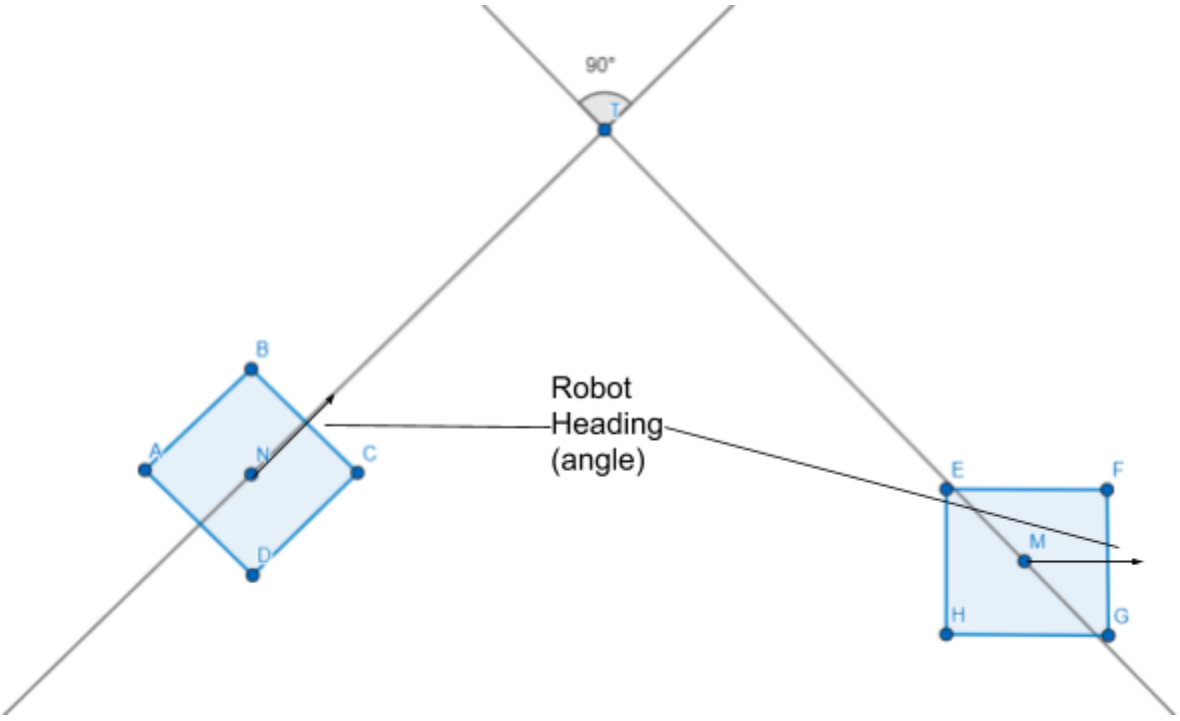


**Figure 7:** Example motion (robot to move from N to M with above change in angle)

Here, we know the robot's current x, y, and angle (thanks to odometry), and the destination x, y, and angle as given by the programmer. The goal of this section is to compute how much "error" is there in the local x, y direction and angle. Error is formally defined as the difference between the target value and the initial value.

$$Error = Destination - Current$$

In figure 7, N and M are the tracking centers of the initial and destination robot positions respectively. Line NT is aligned with the initial robot angle/heading. Line MT is perpendicular to line NT (as the local x component is perpendicular to the local y component), intersection point M, the destination tracking center of the robot.

Thus, we only have 3 parameters to solve for: the $error_x$, $error_y$, $error_\theta$.

The error in the angle is trivial: it is simply the destination robot angle minus the current robot angle.

$$error_\theta = final_\theta - \theta \quad \text{(a1)}$$

The x and y errors require analytic geometry. Lines are modelled in standard form in this paper to make the mathematics pertaining to this more elegant.

To define lines NT and MT, a second point is needed that lies on each line in addition to the known point N and M for the line to be defined. Since the angle of the robot is known, a second point can be derived on each line using trigonometry.

Let us define a few terms. $x_N$ , $y_N$ and $x_M$ , $y_M$ will be the coordinates of N and M respectively. Let $l$ be an arbitrary positive number that denotes the euclidean distance from N second point on NT.

On line NT:

$$x_{N_2} = x_N + lcos(\pi/2 - \theta) = x_N + lsin(\theta)$$

$$y_{N_2} = y_N + lsin(\pi/2 - \theta) = y_N + lcos(\theta)$$

We will define line MT after NT is defined in standard form, as it is perpendicular to NT, intersecting M

The standard form equation of a line is defined as such:

$$Ax + By + C = 0$$

Rearranging this into slope intercept for gives:

$$By = -Ax - C$$

$$y = \frac{-A}{B}x - \frac{C}{B} \qquad (7)$$

Traditionally, slope intercept form is expressed with $m$ being the slope and $b$ being the y intercept. From (7), it is true that:

$$m = -A / B \qquad (8)$$

$$b = -C / B \qquad (9)$$

From (8), A and B can be calculated for both NT and MT. Let the constants A, B, and C representing line NT be subscripted with N, and be subscripted with M if representing line MT.

$A_N = -(y_{N_2} - y_N)$

$B_N = x_{N_2} - x_N$

$C_N = -A_N x_N - B_N y_N$

Since the slope of line NT is expressed as:

$m_{NT} = (y_{N_2} - y_N) / (x_{N_2} - x_N)$

The slope of line MT, defined to be perpendicular to NT, would then be the negative reciprocal of the slope of NT.

$m_{MT} = -(x_{N_2} - x_N) / (y_{N_2} - y_N)$

From (8), we can, by observation, solve for $A_M$ and $B_M$.

$A_M = (x_{N_2} - x_N) = B_N$

$B_M = (y_{N_2} - y_N) = -A_N$

$C_M = -A_M x_M - B_M y_M$

Writing out NT and MT respectively:

$A_N x + B_N y + C_N = 0$ (10)

$A_M x + B_M y + C_M = 0$ (11)

Since the standard form equations of NT and MT are known, let us solve for the point of intersection T. Let us use elimination..

Multiply (10) by $B_M$, and (11) by $B_N$.

$$B_M A_N x + B_M B_N y + B_M C_N = 0 \qquad (10)'$$

$$B_N A_M x + B_N B_M y + B_N C_M = 0 \qquad (11)'$$

(10)' - (11)' gives:

$$B_M A_N x - B_N A_M x + B_M C_N - B_N C_M = 0$$

$$(B_M A_N - B_N A_M)x = B_N C_M - B_M C_N$$

$$\therefore x_T = (B_N C_M - B_M C_N) / (B_M A_N - B_N A_M)$$

Multiply (10) by $A_M$, and (11) by $A_N$

$$A_M A_N x + A_M B_N y + A_M C_N = 0 \qquad (10)''$$

$$A_N A_M x + A_N B_M y + A_N C_M = 0 \qquad (11)''$$

(10)'' - (11)'' gives:

$$A_M B_N y - A_N B_M y + A_M C_N - A_N C_M = 0$$

$$(A_M B_N - A_N B_M)y = A_N C_M - A_M C_N$$

$$\therefore y_T = (A_N C_M - A_M C_N) / (A_M B_N - A_N B_M)$$

Since points N, T, M are all known, the $error_x$ and $error_y$ can be calculated from the euclidean distances between points MT and NT respectively.

$$error_x = \sqrt{(x_M - x_T)^2 + (y_M - y_T)^2} \qquad (e1)$$

$$error_y = \sqrt{(x_N - x_T)^2 + (y_N - y_T)^2} \qquad \text{(e2)}$$

However, since euclidean distances are non negative, whether the error is negative or positive is unknown (suppose the error of the robot needs it to move backwards instead of forwards). To address this problem, let us construct another line on point N, S, such that NS is perpendicular to NT (figure 8).

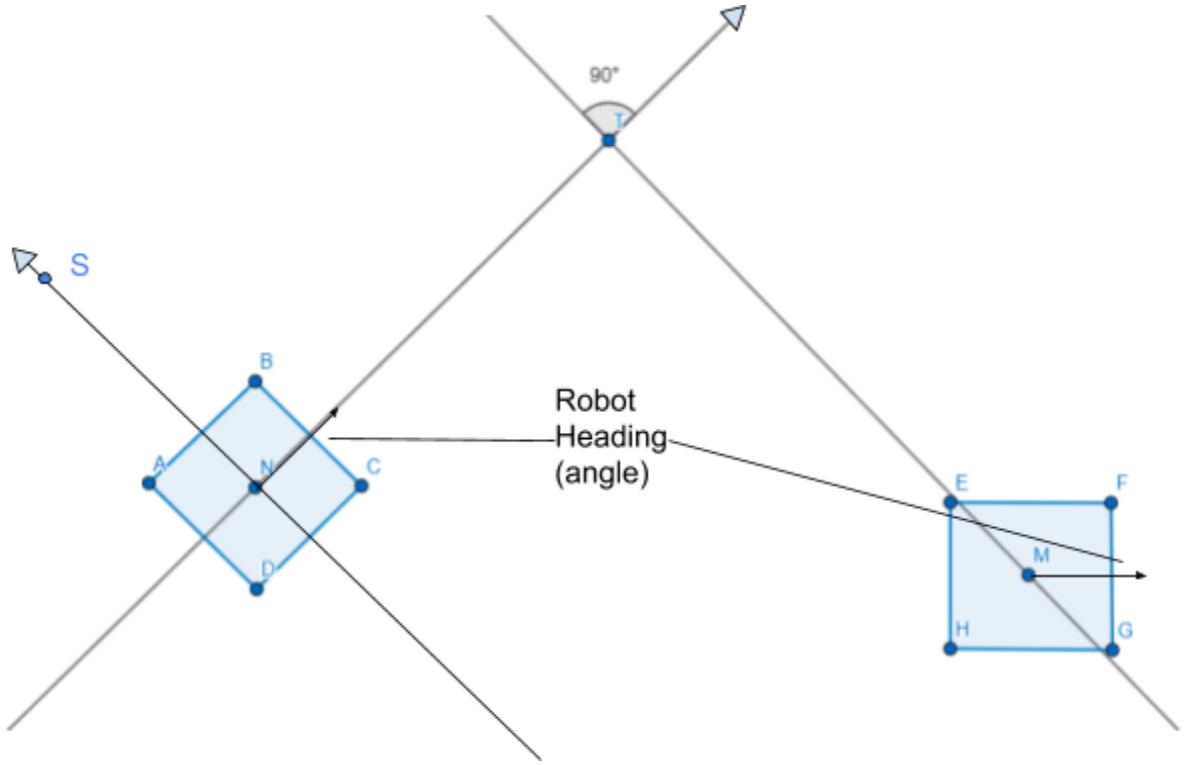


**Figure 8:** Remodelling figure 7 with NT and NS as vectors

Since the local vertical error is governed by the distance NT, we care if T is in front of the robot, or behind the robot. Similarly, the horizontal error, being governed by the distance TM, we care about whether point M is to the right of the robot, or to the left. This can be modelled by

constructing line NS and considering line NT. For the vertical error, if it is known which side point T lies relative to NS (to the right, or left), the sign of $error_y$ would be known. By similar argument, if it is known which side point M lies relative to NT, the sign of $error_x$ is known. However, notice that saying point X lies on the right of line Y is ambiguous, it depends on where you define the "front" of the line. Thus, the direction of the line matters, which is why they are now modelled as vectors (thus having a direction). From this, we can see that vector NS is rotated 90 degrees counterclockwise ($-\pi/2$). Let us use this to define a second point on NS.

$$x_S = x_N + lcos(\theta - \pi/2)$$

$$y_S = y_N + lsin(\theta - \pi/2)$$

Let us then model NS and NT as vectors.

$$\overrightarrow{NS} = \langle x_S - x_N,\, y_S - y_N \rangle \qquad (12)$$

$$\overrightarrow{NT} = \langle x_T - x_N,\, y_T - y_N \rangle \qquad (13)$$

For the sake of brevity, let us define this setup in general terms (so the above vectors could be substituted in to yield the results). Let us define $\overrightarrow{AB}$ as our vector, with point A defined with coordinates $x_1$ and $y_1$, and B defined with coordinates $x_2$ and $y_2$. Note that $x_1$ and $y_1$ are analogous to $x_N$ and $y_N$ above, and similarly $x_2$ and $y_2$ are analogous to $x_S$, $x_T$ and $y_S$, $y_T$. We want to know which side point P lies relative to AB (on the right? On the left?). Let $\overrightarrow{BP}$ be a vector pointing from B to P, with $\overrightarrow{BP}^{\parallel}$ being a component vector of it parallel with $\overrightarrow{AB}$, and

$\overrightarrow{BP}^{\perp}$ being the other component vector perpendicular to $\overrightarrow{AB}$. Lastly, let $\vec{n}$ be a vector orthogonal to $\overrightarrow{AB}$.

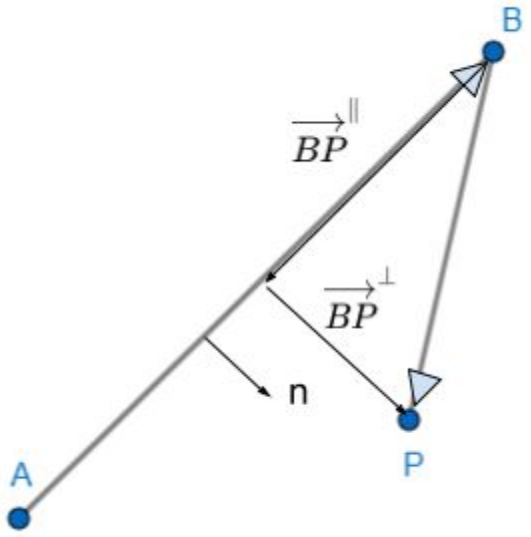


**Figure 9:** General form for determining which side point P lies relative to AB

$\overrightarrow{AB}$, as coordinates of A and B are both defined, can be expressed by those terms.

$\overrightarrow{AB} = \langle x_2 - x_1,\, y_2 - y_1 \rangle$

A fundamental property of dot products between 2 vectors is considered in 3 cases:

1) Vectors are pointing in the same (general) directions (angle between them is acute). Dot product will be positive.
2) Vectors are orthogonal (perpendicular to each other). Dot product will be 0.

3) Vectors are pointing in opposite (general) directions (angle between them is obtuse). Dot product will be negative.

$\vec{n}$ is defined to be a vector orthogonal to $\overrightarrow{AB}$, thus $(x_2 - x_1)n_x + (y_2 - y_1)n_y = 0$ (property 2). By inspection, notice that the above equality holds if $n_x = y_2 - y_1$ and $n_y = -(x_2 - x_1)$.

Thus, $\vec{n} = \langle y_2 - y_1,\ -(x_2 - x_1) \rangle$ (14)

Note that if $n_x = -(y_2 - y_1)$ and $n_y = (x_2 - x_1)$, the above equation still holds true for $\vec{n}$, however the vector would be orthogonal counterclockwise to $\overrightarrow{AB}$, unlike in figure 9. Thus, in this paper we will stick to the prior definition for consistency. Letting the coordinates of point P be $x_p$ and $y_p$, let us express $\overrightarrow{BP}$.

$\overrightarrow{BP} = \langle x_P - x_2, y_P - y_2 \rangle$ (15)

By properties 1 and 3, as long as two vectors are known, should they be pointing on the same side, or otherwise, the answer would be positive or negative respectively. One of the vectors, $\vec{n}$, is defined above. Thus referring back to figure 9, we need the dot product between $\vec{n}$ and $\overrightarrow{BP}^{\perp}$, as if $\overrightarrow{BP}^{\perp}$ points to the same side as $\vec{n}$ (property 1), the dot product will be positive, telling us point P lies to the right of $\overrightarrow{AB}$ (figure 9), and consequently should the product be negative, point P lies to the left of $\overrightarrow{AB}$ (figure 9).

Taking the dot product between $\overrightarrow{BP}$ and $\vec{n}$:

$$\overrightarrow{BP} \cdot \vec{n}$$

$$= \left( \overrightarrow{BP}^{\parallel} + \overrightarrow{BP}^{\perp} \right) \cdot \vec{n}$$

$$= \overrightarrow{BP}^{\parallel} \cdot \vec{n} + \overrightarrow{BP}^{\perp} \cdot \vec{n}$$

$\because$ $\overrightarrow{BP}^{\parallel}$ is parallel with $\overrightarrow{AB}$, and $\vec{n}$ is orthogonal with $\overrightarrow{AB}$

$\therefore$ By rule 2, $\overrightarrow{BP}^{\parallel} \cdot \vec{n} = 0$. Substituting that into the prior expression

$$= \overrightarrow{BP}^{\perp} \cdot \vec{n}$$

$$\therefore \overrightarrow{BP} \cdot \vec{n} = \overrightarrow{BP}^{\perp} \cdot \vec{n}$$

Therefore, simply evaluating the dot product of $\overrightarrow{BP} \cdot \vec{n}$ is equivalent to evaluating the dot product of $\overrightarrow{BP}^{\perp} \cdot \vec{n}$. Knowing this result, let us substitute (14) for $\vec{n}$, and (15) for $\overrightarrow{BP}$. Let the real number it evaluates to be $d$.

$$d = \overrightarrow{BP} \cdot \vec{n}$$
$$d = \langle x_P - x_2,\, y_P - y_2 \rangle \cdot \langle y_2 - y_1,\ -(x_2 - x_1) \rangle$$
$$d = (x_P - x_2)(y_2 - y_1) - (y_P - y_2)(x_2 - x_1) \qquad (16)$$

$\therefore$ If $d > 0$, P is on the right of $\overrightarrow{AB}$ (as shown in figure 9), otherwise if $d < 0$, P is on the left, else, P lies on $\overrightarrow{AB}$ ($\overrightarrow{BP}^{\perp} = 0$).

When considering vector (12), we want to know which side point T lies on (y component). When considering vector (13), we want to know which side point M lies on (x component). Let us use this information to substitute into (16). Let $d_x$ denote the $d$ for the x component, and $d_y$ for the y component.

$$d_x = (x_M - x_T)(y_T - y_N) - (y_M - y_T)(x_T - x_N)$$

$$d_y = (x_T - x_S)(y_S - y_N) - (y_T - y_S)(x_S - x_N)$$

Combining this with our results for (e1) and (e2), the **magnitude** and **direction** of the errors in the local x and y direction are now known. Restating our angular error here for completeness (a1) as well gives us the all the robot errors from its destination position:

$$error_x = \sqrt{(x_M - x_T)^2 + (y_M - y_T)^2} \quad d_x = (x_M - x_T)(y_T - y_N) - (y_M - y_T)(x_T - x_N)$$

$$error_y = \sqrt{(x_N - x_T)^2 + (y_N - y_T)^2} \quad d_y = (x_T - x_S)(y_S - y_N) - (y_T - y_S)(x_S - x_N)$$

$$error_\theta = final_\theta - \theta$$

*Dealing with the errors*

Knowing the magnitude and the direction of all of our errors (in the local x, and y direction, in addition to angle), the problem now is implementing an effective function that governs how

much the motor should move to correct the known error. Sure, something simple like simply setting the motors to full speed as long as error is existent would work, but again this would be not ideal as described in the smooth efficient motion we want described in the opening of this section of the paper.

Pondering about this problem, we can agree that if the error is large, the robot should move faster, and when the error gets smaller, the robot should start slowing down, similar to how we would brake our car approaching a red light (we wouldn't just slam the brakes, right?) This can be modelled mathematically by setting up a proportional constant, $K_p$, that is multiplied to this error (always destination minus current position for x, y, or angle).

This is an effective first step to attaining smoother motion. However, consider when the error is small (about an inch), the proportion in turn would be small as well. Due to the chassis having mass, the motors might not be able to move the chassis as they are not receiving enough power. A method to counter this is to integrate the error over time, and add that to the motor output. This way, the integral component will keep accumulating as long the error is existent. Similarly, a constant, $K_I$ can be multiplied to this integral to help tweak it.

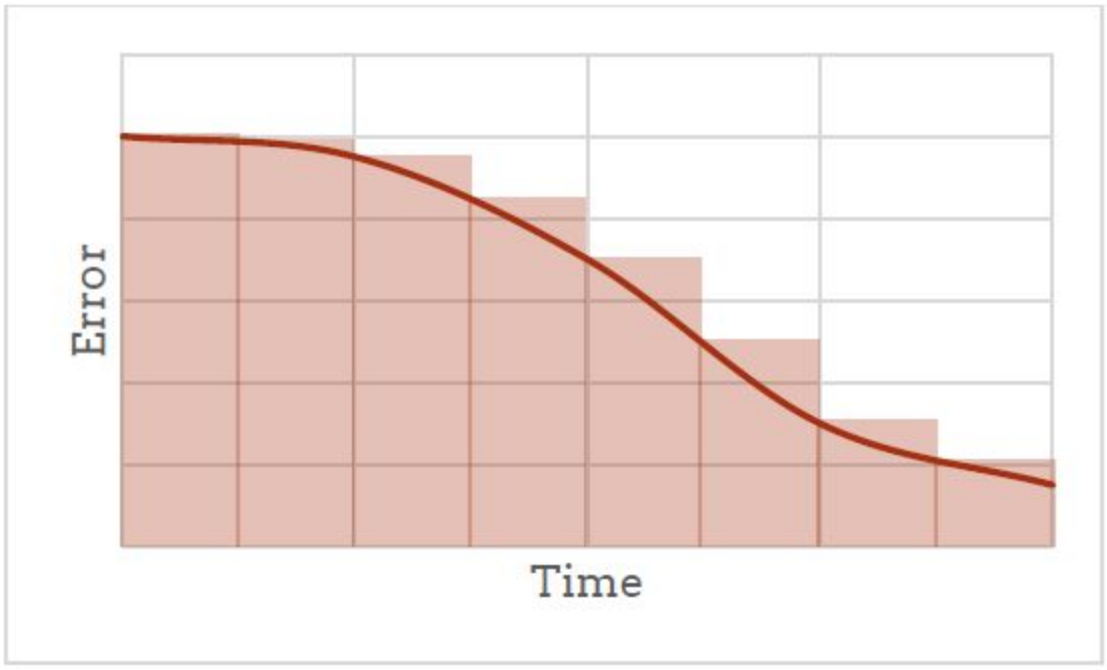


**Figure 10:** Integral

Even with $K_I$, the integral may end up accumulating too much and send too much power to the motors, causing the robot to overshoot horribly. This can be regulated by adding a third component, the derivative, that records the change in error from right before to right now. If this change is too big (thanks to integral), we can use this information to slow down the motors. Again, a constant $K_D$ can be multiplied to this to tweak the power outputs, as robots are typically different.

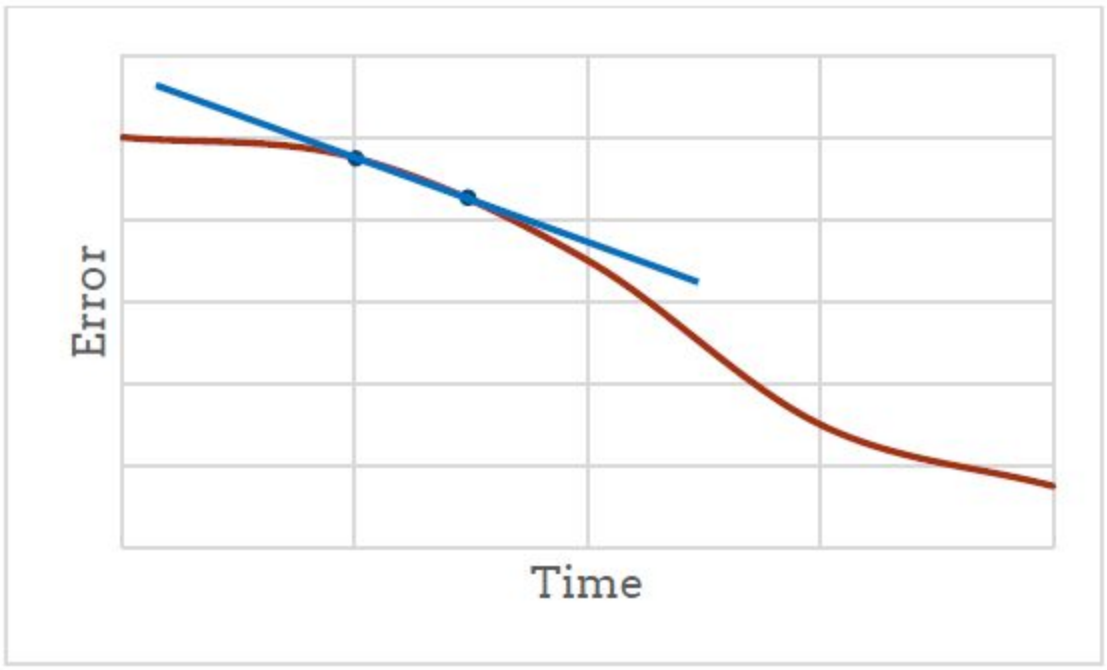


**Figure 11:** Derivative

This proportional, integral, and derivative components make up the classic PID loop, widely used in the robotics industry. Applying this to our $error_x$, $error_y$, and $error_\theta$, after tuning the 3 constants, will allow the robot to smoothly adjust to these errors, until it reaches the desired destination point.

**References**

Calculate on which side of a straight line is a given point located? (1962, June 01). Retrieved from https://math.stackexchange.com/questions/274712/calculate-on-which-side-of-a-straight-line-is-a-given-point-located

Introduction to Position Tracking. (2018, October 7). Retrieved from

http://thepilons.ca/wp-content/uploads/2018/10/Tracking.pdf

Veness, T. (n.d.). Controls Engineering in the FIRST Robotics Competition. doi:https://file.tavsys.net/control/controls-engineering-in-frc.pdf