Geometric Modelling in Robot Control

How can euclidean and analytic geometry be used to model robot controls?

Mathematics Extended Essay

Words: 3380

Contents

1 Abstract

Autonomous robot control in today's world is largely governed by mathematical models and algorithms, considering a variety of inputs from sensors in pursuit of achieving consistent and repeatable movement. Such calculations from sensors are critical in robotics, as effective robots should be able to correct itself under influence of random error and tolerate variances. An example of this could be a self driving car driving to a particular destination, but one of its wheels slip against some gravel, causing it to veer off course. If the car's position was not tracked, the car will not know that it has veered off course, causing the error to compound, leading to life threatening consequences. Thus, an effective autonomous robot must be robust, and hence able to correct its course despite random external variances.

As such, this paper concerns the geometric modelling of an absolute position tracking system that continuously tracks the robot's position and orientation, and from this a geometric framework that models the robot's difference from its intended path, informing the robot to correct it. Given this tracking framework, should the prior scenario be reconsidered, the car would now know that it has veered off course, as its current tracked position would be different than what was intended. This important information can then inform a different series of movements to correct the robot's trajectory from modelling the difference between its current position and its intended position, helping it to reach its destination despite external influences.

2 Position Tracking (Odometry)

Another name for absolute position tracking is odometry, where the robot's coordinate position and where it is pointing is tracked relative to its starting position. To implement this, 3 unpowered wheels are added onto the drivebase (figure 1) called tracking wheels, which maintain contact with the ground, and are each fixed to an optical shaft encoder recording the angular rotation of the shaft each wheel its mounted on. Given the change in rotation of the shaft, and the tracking wheels' diameter, the displacement, s, that a tracking wheel has tracked can be calculated using the arc length formula, where $\Delta\theta_{shaft}$ denotes the change in rotation of the shaft, and the radius being half the diameter of the wheel.

$$s = r\Delta\theta_{shaft}$$

My chassis setup for the robot I was building is detailed in figure 1. In

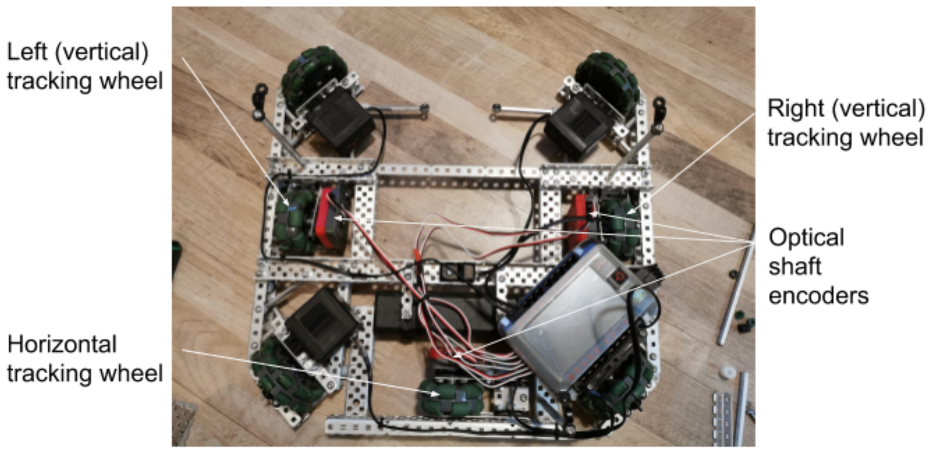
Left (vertical) tracking wheel
Right (vertical) tracking wheel
Optical shaft encoders
Horizontal tracking wheel

Figure 1: Tracking wheel setup on my robot chassis (Source: Allen Tao, Taken on May 20 2020

figure 2, the drivebase is described numerically, and marked with relevant variables used throughout this section regarding the physical drivebase.

In this section, we geometrically track the cartesian position of the tracking

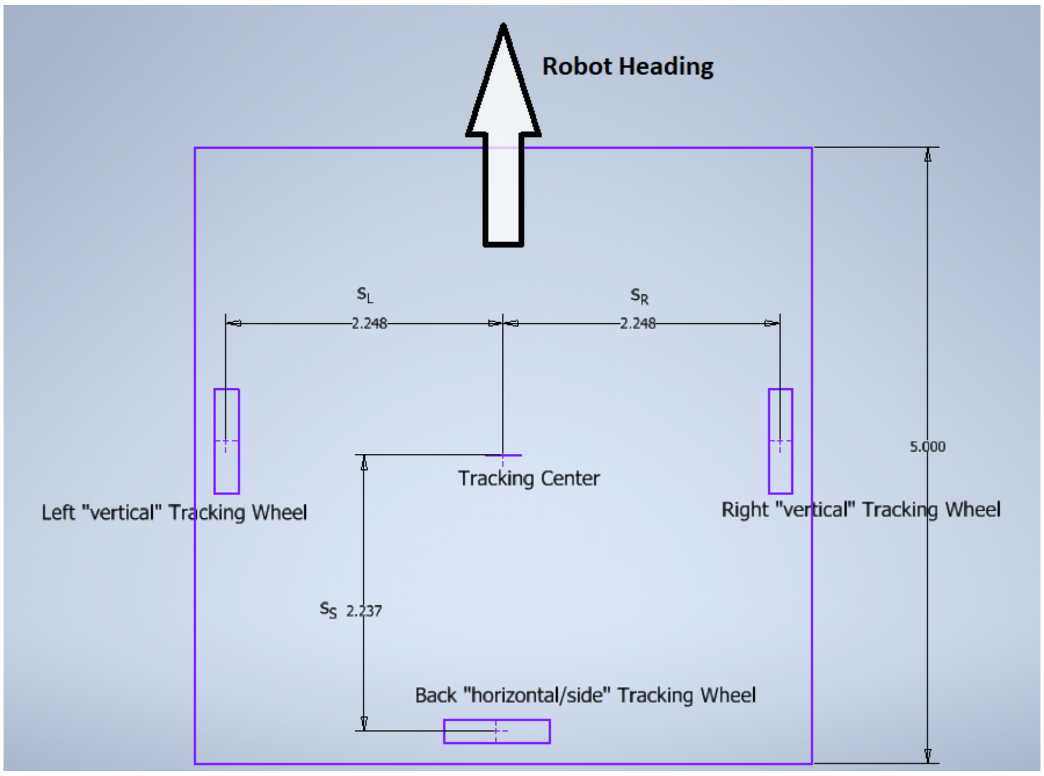
Robot Heading
s_L
2.248
s_R
2.248
5.000
Tracking Center
Left "vertical" Tracking Wheel
Right "vertical" Tracking Wheel
s_S 2.237
Back "horizontal/side" Tracking Wheel

Figure 2: Geometric model of drivebase, along with variable definitions

centre and angular heading relative to their starting values on a coordinate plane upon movement, based on the work of Team 5225 E-Bots Pilons (2018). Let s_L denote the minimum distance between the line on which the left tracking wheel lies and the tracking center, s_R the same thing but for the right tracking wheel, and s_S for the back tracking wheel similarily. Positive displacements are tracked by the vertical encoders when driving forwards, and positive displacements are tracked by the horizontal encoder when moving towards the right. Note that directional statements such as "forwards" or "right" are all relative to the robot heading (ie, where the front of the robot is pointing). Take note that the heading of the robot, θ, is relative to the y axis, meaning that when the arrowhead in figure 2 is aligned with y axis, the heading is equivalent to an integer multiple of 2π radians, tending positive clockwise. To express the robot's position, let x and y denote the initial x and y positions of the robot on the plane, and θ the initial heading of the robot. The final x, y, and θ of the robot's position is hence denoted as x_2, y_2, and θ_2.

Assume for now that the robot cannot slip 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 $\overset{\frown}{\mathrm{AB}}$ traced by the tracking center from the example arc movement in figure 3 below. Here, 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 $\overset{\frown}{\mathrm{AB}}$ at point A, and line BE is tangent to arc $\overset{\frown}{\mathrm{AB}}$ at point B. Point D is the intersection of tangents AC and BE. Thus, for the robot to move from point A to B, its heading must have changed. This change in heading, $\Delta\theta$ is denoted as $\angle CDB$, in angle of intersection of tangents AC and BE. I will now show that $\angle CDB = \angle AOB$, where $\angle AOB$ is angle of the $\overset{\frown}{\mathrm{AB}}$ traced by the tracking center.

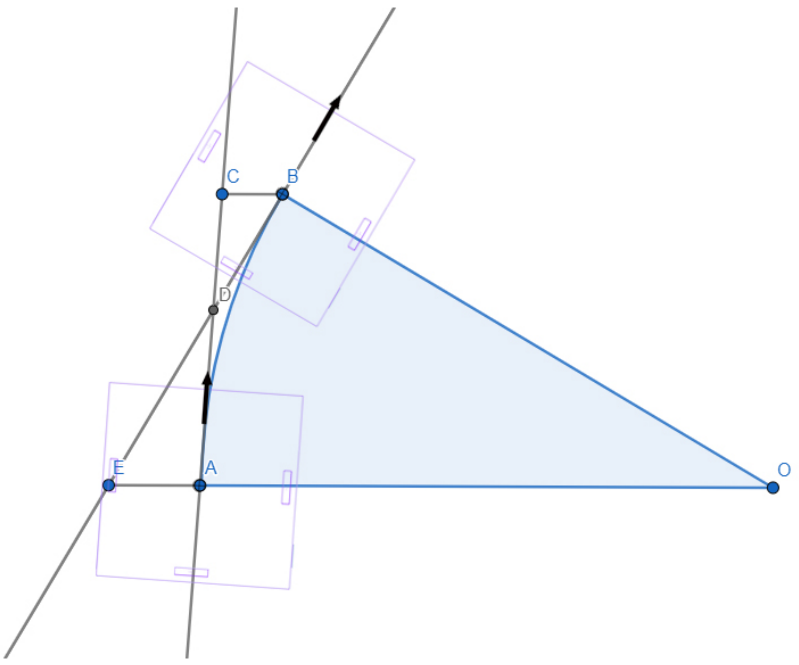

Figure 3: Geometric sketch for proving $\angle CDB = \angle AOB$

Proof.

$\because \overline{AC}$ is tangent to $\widehat{\text{AB}}$ at A

$\therefore \angle CAO = \frac{\pi}{2}$

$\because \overline{BE}$ is tangent to $\widehat{\text{AB}}$ at point B

$\therefore \angle EBO = \frac{\pi}{2}$

$\because \angle CAO = \frac{\pi}{2}$, and $\angle EAO = \pi$

$\therefore \angle DAE = \angle EAO - \angle CAO = \frac{\pi}{2}$

$\triangle DAE \sim \triangle OBE$ (Angle Angle Similarity)

$A : \angle DAE = \angle EBO = \frac{\pi}{2}$

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

$\therefore \angle ADE = \angle BOE$

By the opposite angle theorem

$\because \angle CDB = \angle ADE$

$\therefore \angle CDB = \angle BOE$

$\therefore \angle CDB = \angle AOB$ $\square$

With knowledge of this proof, let us calculate the change in heading from an example robot motion in figure 4.

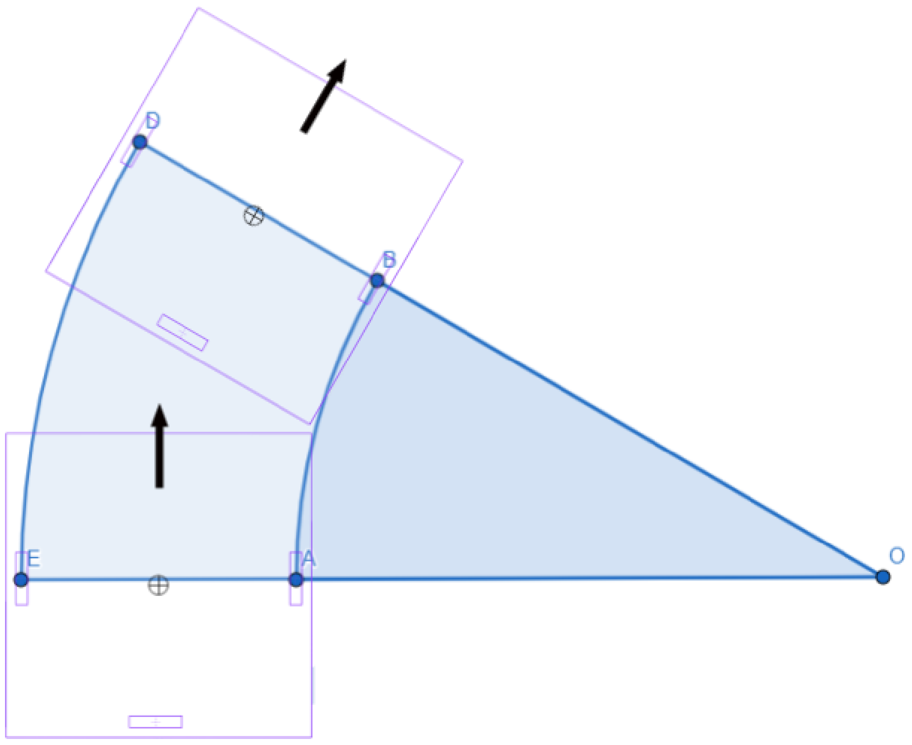

Figure 4: Simple arc motion from robot, similar to car making a turn

Let the displacements tracked by the left and right tracking wheels ($\overset{\frown}{\mathrm{ED}}$ $\overset{\frown}{\mathrm{AB}}$ respectively) be denoted as ΔL and ΔR. Similarly, the change in displacement tracked by the horizontal tracking wheel is denoted as ΔS, but I will show the calculation of heading does not need it. Let the radius of the arc modelling the motion of the robot be expressed as r. Finally, let the change in heading of the robot moving from the bottom towards the top of figure 4 be denoted as $\Delta\theta$. Knowing that $\Delta\theta$ is congruent to the angle of the arc, let us solve for $\Delta\theta$.

$$s = r\Delta\theta \qquad\qquad s = r\Delta\theta$$

$$\begin{aligned} \Delta L &= (r + s_L)\Delta\theta & \Delta R &= (r - s_R)\Delta\theta \\ r + s_L &= \frac{\Delta L}{\Delta\theta} & r - s_R &= \frac{\Delta R}{\Delta\theta} \\ r &= \frac{\Delta L}{\Delta\theta} - s_L & r &= \frac{\Delta R}{\Delta\theta} + s_R \end{aligned}$$

$$\begin{aligned} \frac{\Delta L}{\Delta\theta} - s_L &= \frac{\Delta R}{\Delta\theta} + s_R \\ \Delta L - s_L\Delta\theta &= \Delta R + s_R\Delta\theta \\ s_R\Delta\theta + s_L\Delta\theta &= \Delta L - \Delta R \\ \Delta\theta &= \frac{\Delta L - \Delta R}{s_L + s_R} \qquad (1) \end{aligned}$$

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 coordinates to its final coordinates. This is depicted in figure 5 in orange. Evidently, this local coordinate axis will be offset from θ, the current robot heading by a certain angle. Let us prove that this offset from the last robot heading, $\angle DAB$, will always be equal to $\frac{\Delta\theta}{2}$ using the same example motion from before.

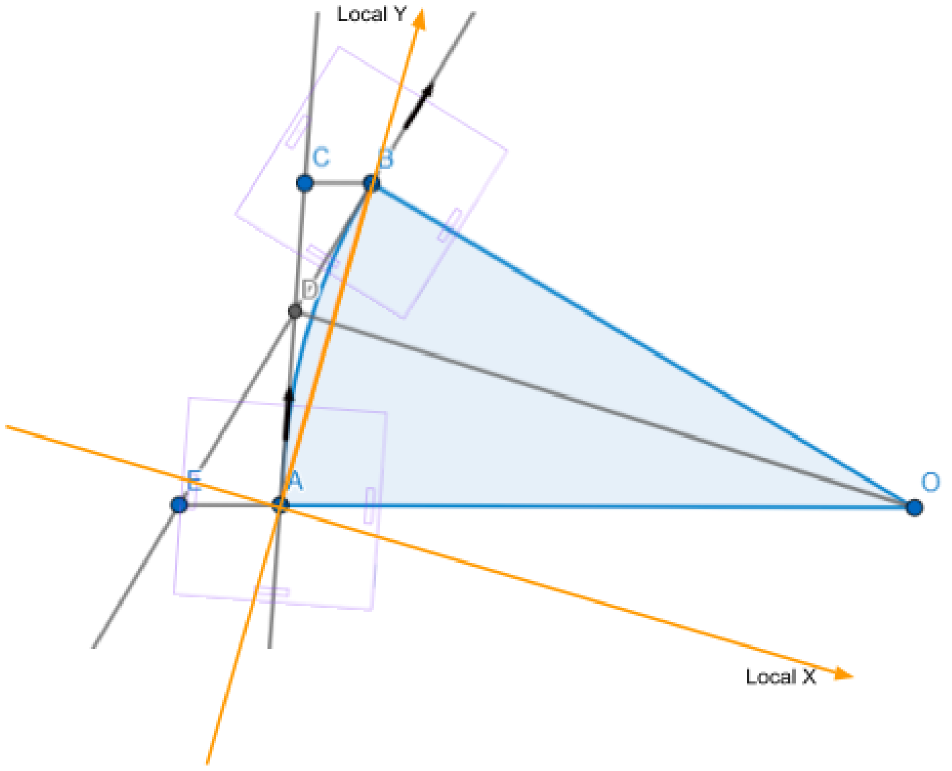

Figure 5: Geometric diagram for proof 2

Proof.

$\because \overline{BE}$ and $\overline{AC}$ are tangent to $\overset{\frown}{\mathrm{AB}}$ at points B and A respectively

$\angle OBE = \angle OAC = \frac{\pi}{2}$

Invoking Pythagorean Theorem in $\triangle OBD$ with $\angle OBD = \frac{\pi}{2}$

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

$OD^2 = r^2 + BD^2$

Invoking Pythagorean Theorem in $\triangle OAD$ with $\angle OAD = \frac{\pi}{2}$

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

$OD^2 = r^2 + AD^2$

Combing the two results obtained from Pythagoras for OD^2:

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

$BD^2 = AD^2$

$BD = AD$ (side lengths are non-negative)

$\because BD = AD$

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

$\because \angle CDB = \Delta\theta$, and $\angle CDA = \pi$

$\therefore \angle BDA + \angle CDB = \pi$

$\therefore \angle BDA = \pi - \Delta\theta$

$\because$ Interior angles of a triangle sum to π

$\angle DAB + \angle DBA + \angle BDA = \pi$

$2\angle DAB + (\pi - \Delta\theta) = \pi$

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

Therefore, the local coordinate axis is offset from the previous robot heading (robot heading prior to performing the motion aligned with $\overline{AC}$), is $\frac{\pi}{2}$.

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

First, notice how we haven't considered what happens when the robot slips or moves sideways, no longer conforming to a perfect motion arc.

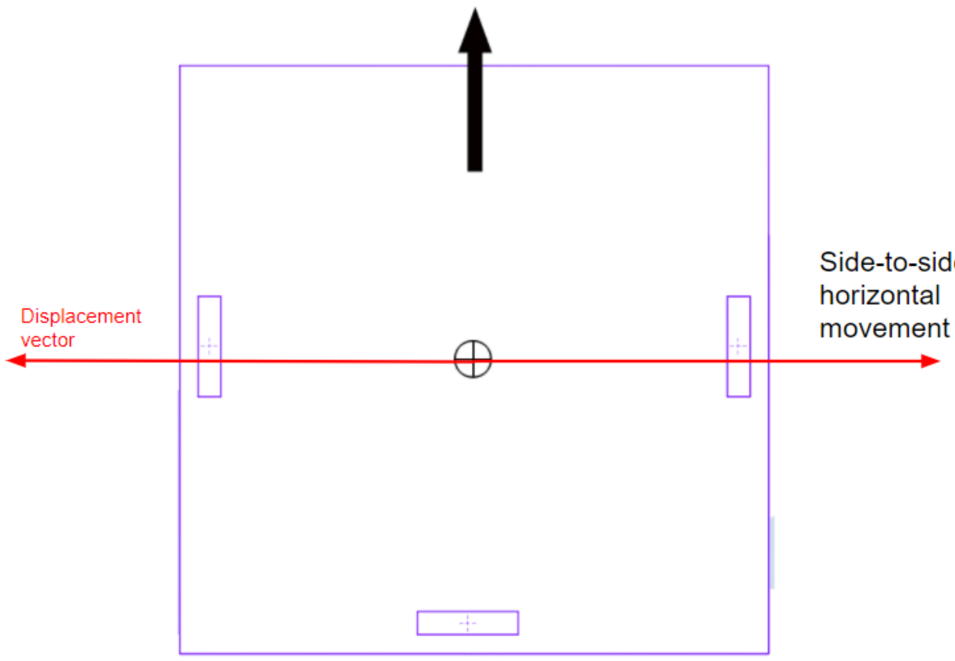

Figure 6: Side-to-side movement, also known as "strafing"

To take this into account in odometry, a similar process for calculating position using arc geometry can be done on the x component of the local coordinate system, using the back tracking wheel to track sideways movement.

In figure 7, suppose the robot is trying to drive in a perfect arc from N to O. However, it drifts towards the left side due to momentum. Since the local coordinate axis is defined with its y axis being aligned with $\overline{NO}$, the y displacement is simply the length of line NO, and thus the x displacement is expressed as $\overline{NM}$ on the local x axis, perpendicular to the y displacement.

Let us begin by solving the y displacement first, denoted by NO, the chord of circular sector OPN. Recall the angle of $\overset{\frown}{\mathrm{NO}}$ and $\overset{\frown}{\mathrm{NM}}$ are both $\Delta\theta$. The chord length formula states that the length of a chord of a circular

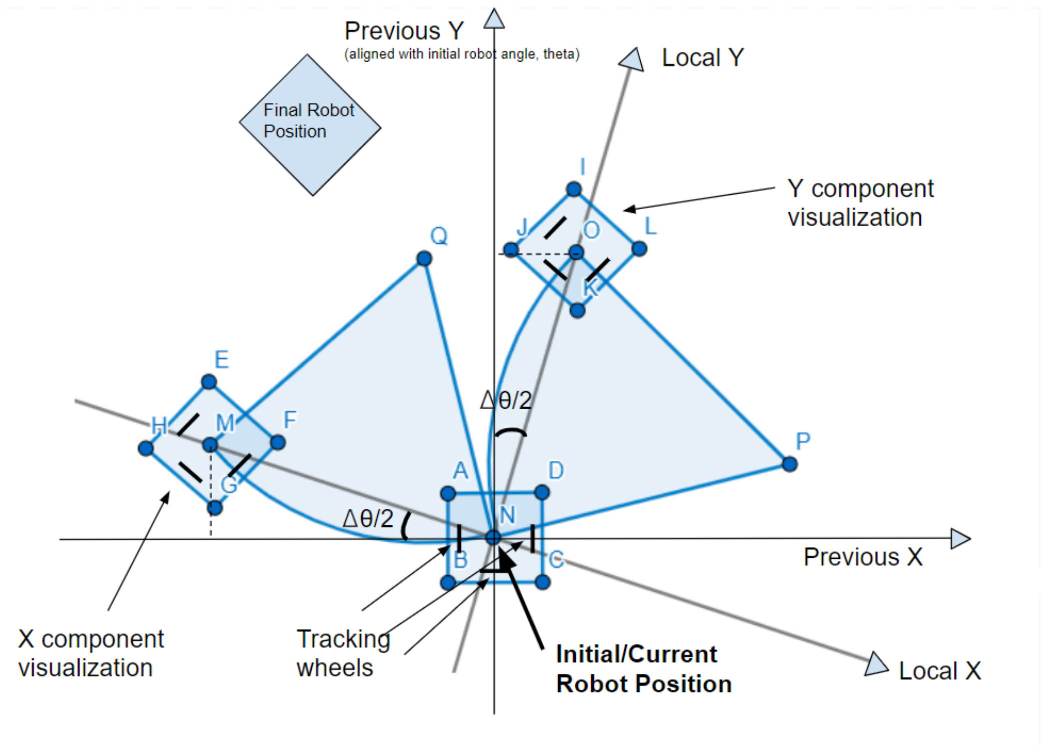
Previous Y
(aligned with initial robot angle, theta)
Local Y
Final Robot Position
I
Y component visualization
J
O
L
K
Q
E
H
M
F
G
Δθ/2
Δθ/2
A
D
N
P
B
C
Previous X
X component visualization
Tracking wheels
Initial/Current Robot Position
Local X

Figure 7: Geometric aid for calculating x and y displacements

sector is: $2rsin(\frac{\phi}{2})$, where r is the radius of the arc, and ϕ the angle of the arc. We use this to calculate NO in sector OPN.

$$
\begin{aligned}
ON &= 2rsin(\frac{\phi}{2}) \\
ON &= 2(\frac{\Delta R}{\Delta\theta} + s_R)sin(\frac{\Delta\theta}{2}) \qquad (2)
\end{aligned}
$$

Next, we repeat the identical process for the x displacement, NM, the chord of circular sector MQN, using the back tracking wheel's tracked distance. It is noteworthy that since the robot is moving horizontally towards its left, ΔS would be negative, resulting in a shorter arc radius than the radius swept by the back tracking wheel itself, yielding a negative horizontal displacement value (which means, towards robot's left). Let r_2 denote the radius of $\widehat{\mathrm{MN}}$.

$$NM = 2r_2 sin(\frac{\phi}{2})$$
$$NM = 2(\frac{\Delta S}{\Delta \theta} + s_S) sin(\frac{\Delta \theta}{2}) \tag{3}$$

Finally, we take the local x and y displacements on the local coordinate axis, NO and MN respectively, and translate them to the global x and y displacements on the previous axis prior to the motion, using trigonometry (figure 8).

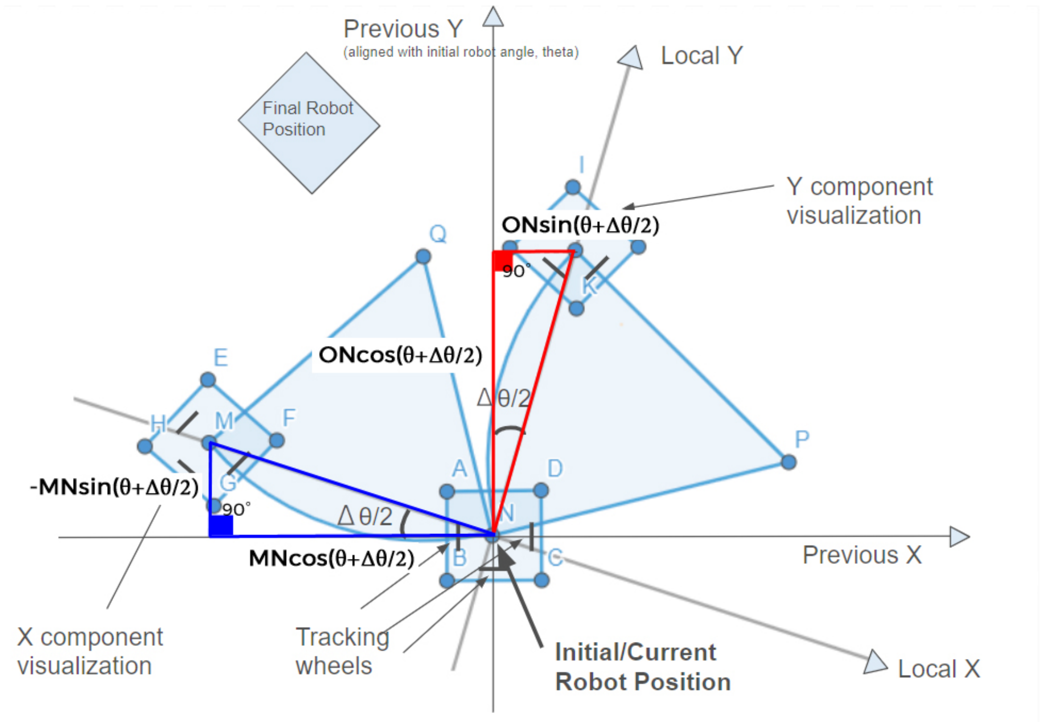

Figure 8: Geometric aid for converting local displacements to global

Now, we have a mathematical expression that tracks the robot's present position moving from its previous position, solely off of the distances tracked by the 3 tracking wheels, and the tracking center's last known coordinates and heading. We remember that $\Delta\theta$, $\overline{ON}$, and $\overline{MN}$, can be expressed in the known previous robot position (x, y, θ), and/or the known distances tracked by the 3 tracking wheels, ΔL, ΔR, and ΔS, derived from (1), (2), and (3) respectively.

$$\boxed{\begin{aligned} &x_2 = x + ONsin(\theta + \frac{\Delta\theta}{2}) + MNcos(\theta + \frac{\Delta\theta}{2}) \\ &y_2 = y + ONcos(\theta + \frac{\Delta\theta}{2}) - MNsin(\theta + \frac{\Delta\theta}{2}) \\ &\theta_2 = \theta + \Delta\theta \end{aligned}}$$

3 Motion Algorithm Model

Armed with odometry, the coordinates and heading of the robot is known relative to its starting position at any time using the results from the previous section, namely the equations for solving for the current x, y, and θ given their last known values. This information is crucial in creating a motion algorithm that is responsible for solving the problem: how can we get the robot from point A (x, y, θ) to point B (x_2, y_2, θ_2)?

The goal of this section is to derive a function, when supplied the target x, y, and θ values, moves the robot from its current position to the target efficiently. A note must be made about the nature of the robot drivebase

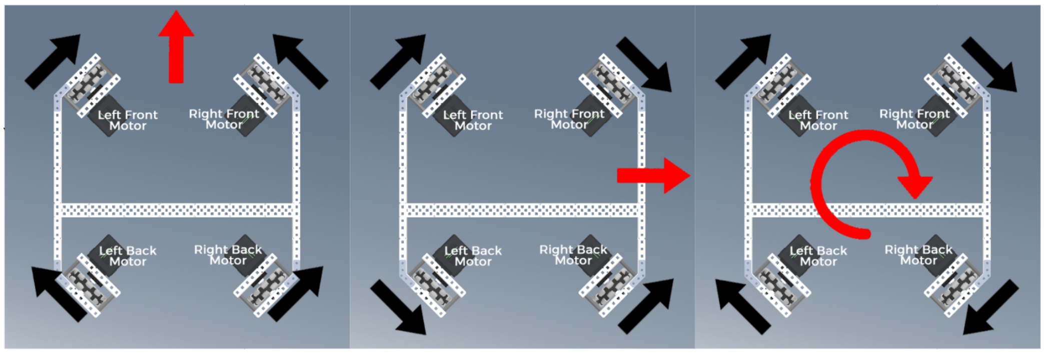

Figure 9: How an X drive can move omnidirectionally (in all directions)

I am using for this extended essay, which is an X drive, a holonomic drivebase (figure 9). Notice how the powered wheels are mounted at 45 degree angles, rather than being inline. As the wheels have freespinning rollers on its exterior (figure 1), this drivebase is able to move in all directions.

Consistent with this paper, forwards, right, and clockwise are defined to be positive directions (as shown in figure 9), with their opposites negative. The angle between the robot heading and the global y axis is θ.

3.1 Modelling the errors

The first step I took to deriving such a motion function/algorithm was to model the direction and magnitudes of *errors*.

Definition 1 (Error)**.** *Error is defined as the difference between the target and initial position/value.*

Thanks to odometry, we know the robot's position (x_N, y_N, θ_N). The destination (x_M, y_M, θ_M) is given by the programmer. We begin by calculating the error between the destination and current robot position and orientation. In figure 10 below, N and M are the tracking centers of the initial and destination positions respectively. $\overline{NT}$ is aligned with the initial/current robot heading and represents the y component error. $\overline{MT}$ represents the x component error, thus it is perpendicular to $\overline{NT}$ at point T.

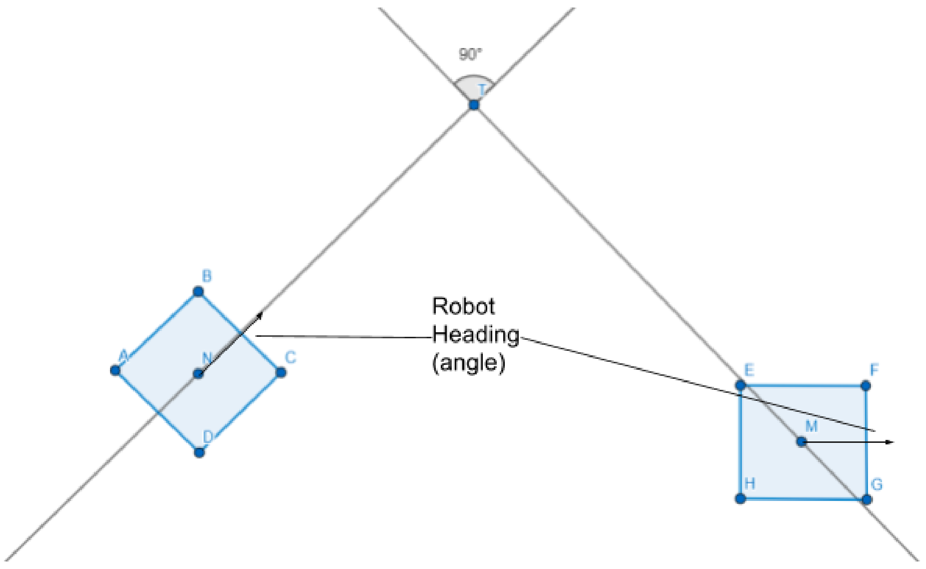

Figure 10: Example motion where robot is currently at N, and needs to get to M with a small clockwise heading change

Here, we need to find magnitude and direction (signage) for: $error_x$, $error_y$, $error_\theta$, the error in the x component, y component, and heading. Finding the magnitude and direction of $error_\theta$ is trivial. By definition of error, it is simply the destination robot heading minus the initial/current robot heading.

$$error_\theta = final_\theta - \theta \tag{4}$$

However, $error_x$ and $error_y$ will require analytic geometry and vectors. I will model lines in standard form to simplify the mathematics involved. To express $\overline{NT}$ in standard form, a second point is needed to define a line. Since the angle between $\overline{NT}$ and the y axis is known (θ, the robot heading), a second point on $\overline{NT}$ can be expressed using trigonometry.

Let x_N, y_N and x_M, y_M denote the coordinates of N and M. Let l be an arbitrary integer that denotes the euclidean distance between N, and an arbitrary second point lying on $\overline{NT}$, (x_{N2}, y_{N2}), which I will use to express $\overline{NT}$ in standard form.

$$x_{N2} = x_N + lcos(\frac{\pi}{2} - \theta) = x_N + lsin(\theta)$$
$$y_{N2} = y_N + lsin(\frac{\pi}{2} - \theta) = y_N + lcos(\theta)$$

Definition 2 (Standard Form of a Line)**.** *$Ax + By + C = 0$, where A, B, and $C \in \mathbb{R}$.*

We are currently interested in finding A_N, B_N, and C_N, the standard form equation for $\overline{NT}$.

$$A_N x + B_N y + C_N = 0 \tag{5}$$

Rearranging this into slope-intercept form gives:

$$\begin{aligned} A_N x + B_N y + C_N &= 0 \\ B_N y &= -A_N x - C_N \\ y &= \frac{-A_N}{B_N} x - \frac{C_N}{B_N} \end{aligned} \tag{6}$$

Let the slope of $\overline{NT}$ be m_{NT}. From (6), we can express m_{NT} in terms of A_N and B_N.

$$m_{NT} = \frac{-A_N}{B_N} \tag{7}$$

We remember that slope can be alternatively expressed as:

$$m_{NT} = \frac{y_{N2} - y_N}{x_{N2} - x_N} \tag{8}$$

By equating m_{NT} in (7) and (8), by inspection, we can solve for A_N and B_N.

$$A_N = -(y_{N2} - y_N) \tag{9}$$

$$B_N = x_{N2} - x_N \tag{10}$$

We substitute (9) and (10) into (5) for A_N and B_N respectively to express

C_N in terms of known values.

$$\begin{aligned} -(y_{N2} - y_N)x + (x_{N2} - x_N)y + C_N = 0 \\ C_N = (y_{N2} - y_N)x - (x_{N2} - x_N)y \\ C_N = (y_{N2} - y_N)x_N - (x_{N2} - x_N)y_N \end{aligned} \tag{11}$$

With $\overline{NT}$ fully expressed in known terms, we are now interested in finding A_M, B_M, and C_M, the standard form equation for $\overline{MT}$ in known terms.

$$A_M x + B_M y + C_M = 0 \tag{12}$$

Because $\overline{MT}$ is perpendicular to $\overline{NT}$, its slope is the negative reciprocal of m_{NT} (8).

$$m_{MT} = \frac{-(x_{N2} - x_N)}{y_{N2} - y_N} \tag{13}$$

By rearranging (12) to slope intercept form, $y = \frac{-A_M}{B_M}x - \frac{C_M}{B_M}$, we see that:

$$m_{MT} = \frac{-A_M}{B_M} \tag{14}$$

By similar argument for solving $\overline{NT}$, by equating equation (13) and (14) for m_{MT}, we can solve for A_M and B_M. Then, substituting A_M and B_M into (12) gives C_M.

$$\begin{aligned} A_M &= x_{N2} - x_N \\ B_M &= y_{N2} - y_N \\ C_M &= -(x_{N2} - x_N)x_M - (y_{N2} - y_N)y_M \end{aligned}$$

With both $\overline{NT}$ and $\overline{MT}$ expressed in known values above, let us solve for the point of intersection T using elimination.We first solve for x_T:

$$\begin{cases} A_N x + B_N y + C_N = 0 \\ A_M x + B_M y + C_M = 0 \end{cases}$$

$$\begin{cases} B_M A_N x + B_M B_N y + B_M C_N = 0 \\ B_N A_M x + B_N B_M y + B_N C_M = 0 \end{cases}$$

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

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

$$x = \frac{B_N C_M - B_M C_N}{B_M A_N - B_N A_M}$$

$$x_T = \frac{B_N C_M - B_M C_N}{B_M A_N - B_N A_M}$$

Similarly, we can solve for y_T.

$$\begin{cases} A_N x + B_N y + C_N = 0 \\ A_M x + B_M y + C_M = 0 \end{cases}$$

$$\begin{cases} A_M A_N x + A_M B_N y + A_M C_N = 0 \\ A_N A_M x + A_N B_M y + A_N C_M = 0 \end{cases}$$

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

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

$$y = \frac{A_N C_M - A_M C_N}{A_M B_N - A_N B_M}$$

$$y_T = \frac{A_N C_M - A_M C_N}{A_M B_N - A_N B_M}$$

With points N, M, and T all expressed in known terms, the magnitudes of $error_x$ and $error_y$ can be calculated from euclidean distances between points M and T, and N and T respectively.

$$error_x = \sqrt{(x_M - x_T)^2 + (y_M - y_T)^2} \tag{15}$$

$$error_y = \sqrt{(x_N - x_T)^2 + (y_N - y_T)^2} \tag{16}$$

However, since euclidean distances are non negative, whether the error is positive or negative is unknown (suppose the robot needs to move a certain magnitude *backwards* instead of *forwards*?) To fix this, I remodeled $\overline{NT}$ and $\overline{TM}$ as vectors, and defined $\overrightarrow{NS}$, which is $\overrightarrow{NT}$ rotated counterclockwise by $\frac{\pi}{2}$.

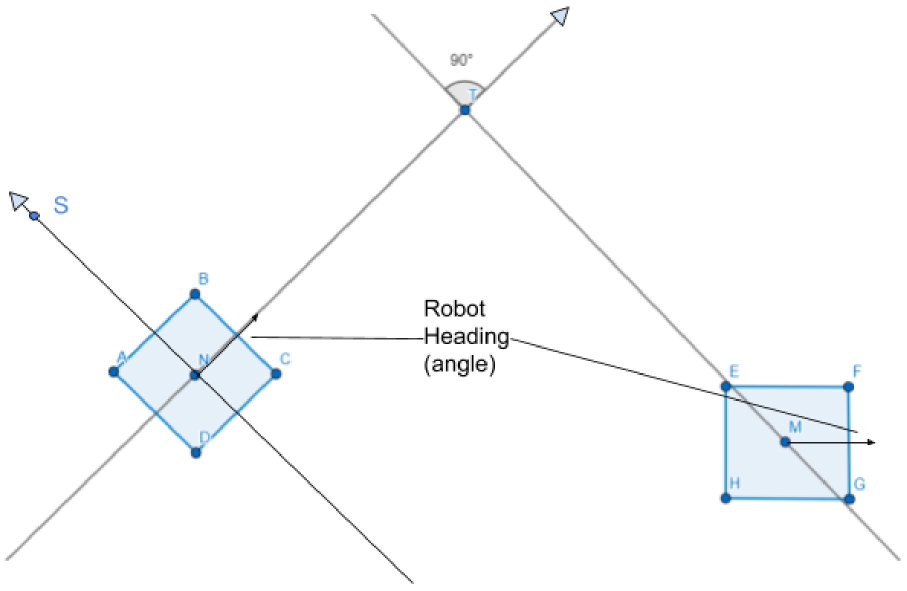

Figure 11: Remodelling figure 10 using vectors

Since the vertical error $error_y$ is governed by NT, we care if T is in front or behind of $\overrightarrow{NS}$. Similarly for the horizontal error $error_x$, being governed by TM, we care about whether point M is to the right or to the left of the $\overline{NT}$. This can be modeled by constructing $\overrightarrow{NS}$ and considering $\overrightarrow{NT}$. If T is on the right side of $\overrightarrow{NS}$, then the sign of the magnitude NT ($error_y$), would be positive, and vice versa. If M is on the right side of $\overrightarrow{NT}$, then the sign of the magnitude MT ($error_x$), would be positive, and vice versa. Vectors are needed to determine such direction as right and left is subjective to which way a line is pointing, and lines do not point in one definite direction like vectors do.

To build $\overrightarrow{NS}$, we need to define point S, an arbitrary point to the *left* of $\overline{NT}$ perpendicular to $\overline{NT}$. Note that $\overrightarrow{NS}$'s direction is a $-\frac{\pi}{2}$ rotation upon $\overrightarrow{NT}$. Hence S can be expressed as follows, where l is a positive real number:

$$x_S = x_N + lsin(\theta - \frac{\pi}{2})$$
$$y_S = y_N + lcos(\theta - \frac{\pi}{2})$$

Hence:

$$\overrightarrow{NS} = \begin{bmatrix} x_S - x_N \\ y_S - y_N \end{bmatrix} \tag{17}$$

$$\overrightarrow{NT} = \begin{bmatrix} x_T - x_N \\ y_T - y_N \end{bmatrix} \tag{18}$$

With figure 12, let us algebraically express which side a point lies relative to a vector in general terms based on the work of Shard (2013). Let $\overrightarrow{AB}$ be our vector, with $A(x_A, y_A)$, $B(x_B, y_B)$. We wish to know which side point

P lies relative to $\overrightarrow{AB}$. Let $\overrightarrow{BP}$ be the vector pointing from points B to P, with $\overrightarrow{BP}^{\parallel}$ being the component vector of $\overrightarrow{BP}$ parallel with $\overrightarrow{AB}$, and $\overrightarrow{BP}^{\perp}$ being the component vector perpendicular to $\overrightarrow{AB}$. Lastly, we designate $\overrightarrow{n}$ to be a vector orthogonal to $\overrightarrow{AB}$. With this, we begin deriving an algebraic

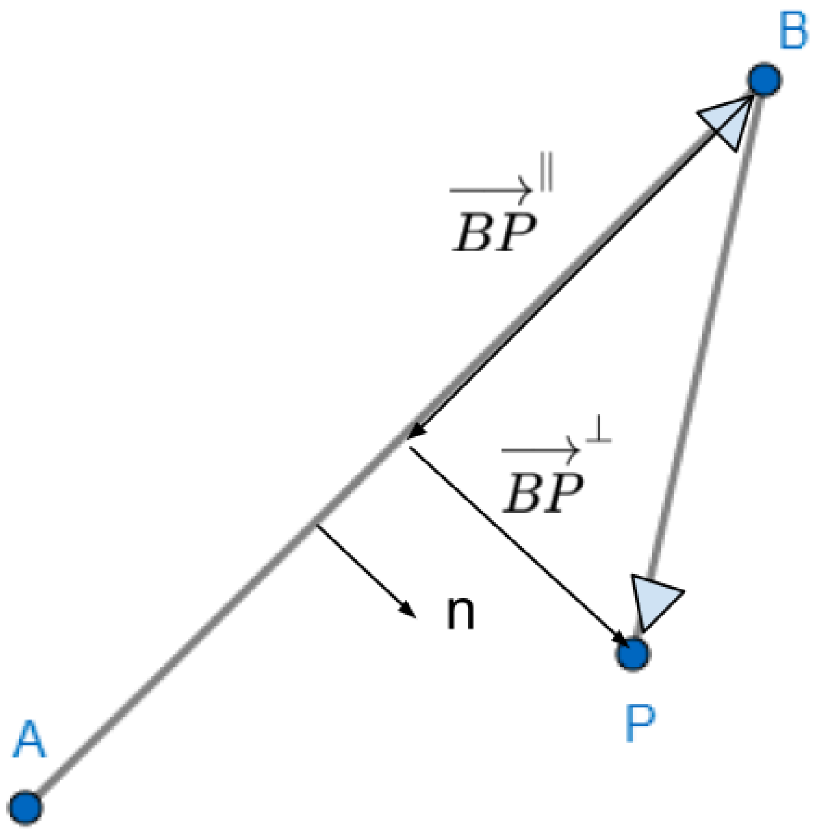
B
$\overrightarrow{BP}^{\parallel}$
$\overrightarrow{BP}^{\perp}$
n
P
A

Figure 12: Geometric visualization for which side an arbitrary point P lies relative to an arbitrary vector AB

expression that tells us which side point P lies relative to $\overrightarrow{AB}$.

$$\overrightarrow{AB} = \begin{bmatrix} x_B - x_A \\ y_B - y_A \end{bmatrix} \tag{19}$$

A general observation of dot products of 2 vectors can be considered in 3 cases:

Theorem 3.1. *Consider two arbitrary 2D vectors*

- *Vectors are pointing in the same (general) directions (angle between them is acute). Dot product will be positive.*
- *Vectors are orthogonal (perpendicular to each other). Dot product will be 0.*
- *Vectors are pointing in opposite (general) directions (angle between them is obtuse). Dot product will be negative.*

Since $\overrightarrow{n}$ is an orthogonal vector to $\overrightarrow{AB}$:

$$\overrightarrow{n} \cdot \overrightarrow{AB} = 0$$

Substituting (19) for $\overrightarrow{AB}$:

$$\overrightarrow{n} \cdot \begin{bmatrix} x_B - x_A \\ y_B - y_A \end{bmatrix} = 0$$

$$(x_B - x_A)n_x + (y_B - y_A)n_y = 0 \tag{20}$$

By inspection in (20), notice that the above equality can hold if $n_x = y_B - y_A$ and $n_y = -(x_B - x_A)$. Note that the equality also holds if $n_x = -(y_B - y_A)$ and $n_y = x_B - x_A$, but we will stick to the prior for consistency with the right side being positive, and left side negative by having $\overrightarrow{n}$ pointing to the right side of $\overrightarrow{AB}$. Hence:

$$\overrightarrow{n} = \begin{bmatrix} y_B - y_A \\ -(x_B - x_A) \end{bmatrix} \tag{21}$$

Letting the coordinates of P be (x_P, y_P), let us express $\overrightarrow{BP}$.

$$\overrightarrow{BP} = \begin{bmatrix} x_P - x_B \\ y_P - y_B \end{bmatrix} \tag{22}$$

Invoking theorem 3.1 on figure 12 from before, notice when $\overrightarrow{n} \cdot \overrightarrow{BP}^{\perp}$ is *positive*, it means $\overrightarrow{BP}^{\perp}$ points to the right of $\overrightarrow{AB}$ with $\overrightarrow{n}$, meaning P is on the right of $\overrightarrow{AB}$. Furthermore, notice when the $\overrightarrow{n} \cdot \overrightarrow{BP}^{\perp}$ is *negative*, $\overrightarrow{BP}^{\perp}$ would point to the left of $\overrightarrow{AB}$, opposite to the side $\overrightarrow{n}$ is pointing. Finally, note when $\overrightarrow{BP}^{\perp}$ is a zero vector, $\overrightarrow{n} \cdot \overrightarrow{BP}^{\perp}$ is also zero, implying that P lies on $\overrightarrow{AB}$. This reveals a beautiful algebraic relationship between which side P lies relative to $\overrightarrow{AB}$, and signage. Let us take advantage of this relationship.

We first need to isolate $\overrightarrow{BP}^{\perp}$ from $\overrightarrow{BP}$ somehow. Let us prove $\overrightarrow{BP} \cdot \overrightarrow{n} = \overrightarrow{BP}^{\perp} \cdot \overrightarrow{n}$.

Proof.
$\overrightarrow{BP} \cdot \overrightarrow{n}$
$= (\overrightarrow{BP}^{\parallel} + \overrightarrow{BP}^{\perp}) \cdot \overrightarrow{n}$
$= \overrightarrow{BP}^{\parallel} \cdot \overrightarrow{n} + \overrightarrow{BP}^{\perp} \cdot \overrightarrow{n}$
$\because \overrightarrow{BP}^{\parallel} \parallel \overrightarrow{AB}$

$\therefore \overrightarrow{BP}^{\parallel} \perp \overrightarrow{n}$

$\therefore \overrightarrow{BP}^{\parallel} \cdot \overrightarrow{n} = 0$

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

With this result, we can determine algebraically which side P lies relative to $\overrightarrow{AB}$ by simply taking the dot product between $\overrightarrow{BP}$ and $\overrightarrow{n}$. We substitute $\overrightarrow{BP}$ (22), and $\overrightarrow{n}$ (21), and let the value it evaluates to be denoted as d.

$$d = \overrightarrow{BP} \cdot \overrightarrow{n}$$

$$d = \begin{bmatrix} x_P - x_B \\ y_P - y_B \end{bmatrix} \cdot \begin{bmatrix} y_B - y_A \\ -(x_B - x_A) \end{bmatrix}$$

$$d = (x_P - x_B)(y_B - y_A) - (y_P - y_B)(x_B - x_A) \tag{23}$$

Therefore, when $d > 0$, P lies to the right of $\overrightarrow{AB}$. When $d < 0$, P lies to the left of $\overrightarrow{AB}$. Finally, when $d = 0$, P lies on $\overrightarrow{AB}$.

When considering vector (17), we want to know which side point T lies on (y component). When considering vector (18), we want to know which side point M lies on (x component). Let us use this information to substitute into (23). Let d_x denote the d for the x component, and d_y for the y component.

$$d_x = (x_M - x_T)(y_T - y_N) - (y_M - y_T)(x_T - x_N) \tag{24}$$

$$d_y = (x_T - x_S)(y_S - y_N) - (y_T - y_S)(x_S - x_N) \tag{25}$$

Combining our results for $error_x$ and $error_y$, with its magnitudes being (15) and (16), and its directions being governed by (24) and (25), we have succeeded in modelling the direction and magnitude of heading (4), and x and y displacement errors in known terms throughout this section. Knowing the errors in each of these 3 parameters will be crucial in informing what the robot should do to correct those.

$$\boxed{\begin{aligned} error_x &= \sqrt{(x_M - x_T)^2 + (y_M - y_T)^2} \\ d_x &= (x_M - x_T)(y_T - y_N) - (y_M - y_T)(x_T - x_N) \\ error_y &= \sqrt{(x_N - x_T)^2 + (y_N - y_T)^2} \\ d_y &= (x_T - x_S)(y_S - y_N) - (y_T - y_S)(x_S - x_N) \\ error_\theta &= final_\theta - \theta \end{aligned}}$$

3.2 Dealing with the errors

Knowing the magnitude and the direction of $error_x$, $error_y$, and $error_\theta$, the problem now is how the robot should move to correct the known errors. Sure, something simple like a bang-bang controller, where motors are set to full speed as long as the error is existent would work in a perfect world, but we wish to develop a smooth, efficient motion rather than a sporadic, jerky one.

Pondering about this problem, we can make a few conclusions. When the error is large, the robot should prioritize correcting that more, hence moving faster to correct that error. Conversely, when the error gets smaller, the robot should start slowing down, similar to how we would gradually depress the brake pedal in a car when approaching a red light (we wouldn't just slam down the brakes, right?). If we look at this carefully, it can be seen that error should be proportional to the motor output used to correct it. Furthermore, we assign a proportional constant, K_p, as a scalar to the motor output to allow for tweaking of this value to tailor this proportional relationship for different robots and applications.

$$motoroutput = K_p error$$

We can apply this on our X-drive, but we need to consider in which combination we must move the motors in order to correct each error (figure 9/13).

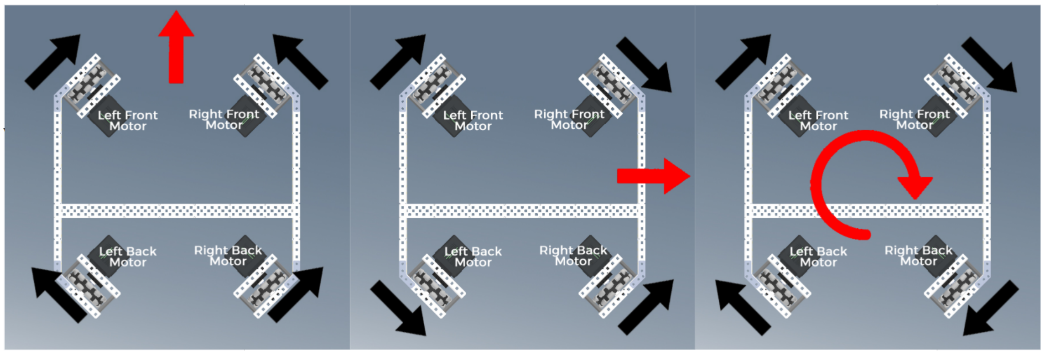

Figure 13: Motor combinations to allow for error correction for all 3 errors

Hence, we may power each of the four drivebase motors using its proportional relationship with $error_x$, $error_y$, and $error_\theta$, and put this in a control loop in a robot program in real life. Forwards, right, and clockwise are positive, and vice versa.

$$\boxed{\begin{aligned}
LeftFrontMotorOut &= (K_p)(error_x) + (K_p)(error_y) + (K_p)(error_\theta) \\
LeftBackMotorOut &= (K_p)(error_x) - (K_p)(error_y) + (K_p)(error_\theta) \\
RightFrontMotorOut &= -(K_p)(error_x) + (K_p)(error_y) + (K_p)(error_\theta) \\
RightBackMotorOut &= -(K_p)(error_x) - (K_p)(error_y) + (K_p)(error_\theta)
\end{aligned}}$$

A note must be made that in industry robot control, simply using a proportional loop (also known as the *P-loop*) to govern motor output is not enough. Consider what happens if a forklift is controlled using only a P-loop when lifting a heavy crate to max height. As the lift moves higher and higher (and thus error decreases), the power sent to the motor/hydraulics powering the forklift decreases too, which will likely cause the lift to stall midway though lifting the crate. It is clear that for different applications, different algorithms must be used. A simple and popular control algorithm is the PID loop, where the P-loop we've investigated forms the first part.

Definition 3 (PID Loop)**.** *A software loop that calculates three components: Proportional, Integral, and Derivative, to determine motor output levels. The proportional component dictates that the motor output is directly proportional to the error its correcting. The integral component integrates the error over time (hence, as long as error exists, motor output will keep growing). Finally, the Derivative component takes the slope against time between the current and last error.*

The issue regarding the forklift lifting a heavy crate to desired height can be solved with the *integral* component, integrating the error over time, thus increasing motor output as long as there exists error. An application that requires the *Derivative* component could be if the mechanism has to move gently. *Derivative* could then calculate the change between the current and previous error over time, and if its being corrected too quickly, reduces the motor output, smoothing the motion. Thus combining *proportional*, *integral*, and *derivative* gives the highly versatile PID loop.

References

1, S. (2013, Jan). Calculate on which side of a straight line is a given point located?. 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, Oct). Team 5225 E-Bots Pilons. Retrieved from http://thepilons.ca/wp-content/uploads/2018/10/Tracking.pdf

Veness, T. (n.d.). Controls engineering in the first robotics competition.. Retrieved from https://file.tavsys.net/control/controls-engineering-in-frc.pdf doi: https://file.tavsys.net/control/controls-engineering-in-frc.pdf