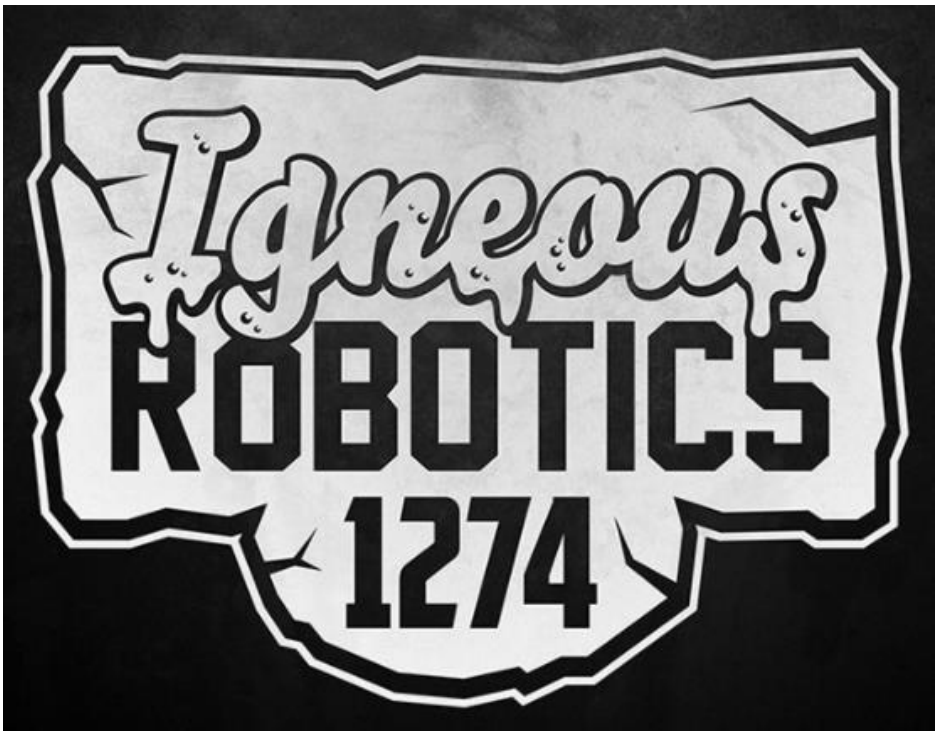

A Dive into Robot Odometry

By Benjamin Dai, 1274B Programmer

Other Team Members: Troy Madden, Justenn Wang, and Gavin Pasquale

Special Thanks to Ethan Harris, 1274A Programmer and E-Bots Pilons, Team 5225

Abstract

This is a documentation of what our two 1274 teams have achieved using an odometry system over the past two VEX Robotics Competition (VRC) seasons and the general processes of how a usable and reliable odometry system was accomplished. The purpose of this document is to contribute more to what the VRC community has available to teams that wish to pursue odometry tracking, hopefully proving useful in addition to the Introduction to Position Tracking document written by VRC Team 5225, as well as providing our team's own take on tackling odometry. This document is aimed at providing more information on the theory behind odometry tracking and how to use it successfully in VRC. As this document pertains to odometry used in VRC, it is targeted towards VRC team members. No code is provided in this document.

Table of Contents

Background Information

Odometry is the use of motion sensor data to estimate the change in position over time, and in our case, it is the use of 3-wire quad-encoders and an inertial sensor to combine distance traveled along the tracking wheel axes with the angle heading of the robot to keep track of change in position. Odometry is useful for the situations where your autonomous routine may drive off-course due to whatever outside interference and potentially ruin your autonomous run. Odometry also makes a variety of drive motions easier to implement, as well as making them less prone to small errors and thus be able to execute faster. The theory and algorithms behind our take on odometry involve the following concepts:

- Right triangle trigonometry
- Pythagorean theorem
- Basic circle math (like circumference)
- 2D kinematics
- Distance formula
- Cartesian plane angle versus inertial sensor heading (different directions)

Tracking Wheel Placements

Our tracking wheels were 2.75-inch omni-wheels with 3-wire quad-encoders mounted on the wheel axles. Theoretically, the tracking wheels can go anywhere on the robot base, so long as they are elasticized to the ground well and angled perpendicular to each other. The hypothetical tracking "center" can be put at any position, but to make things easier, let us continue with a simple tracking wheel layout:

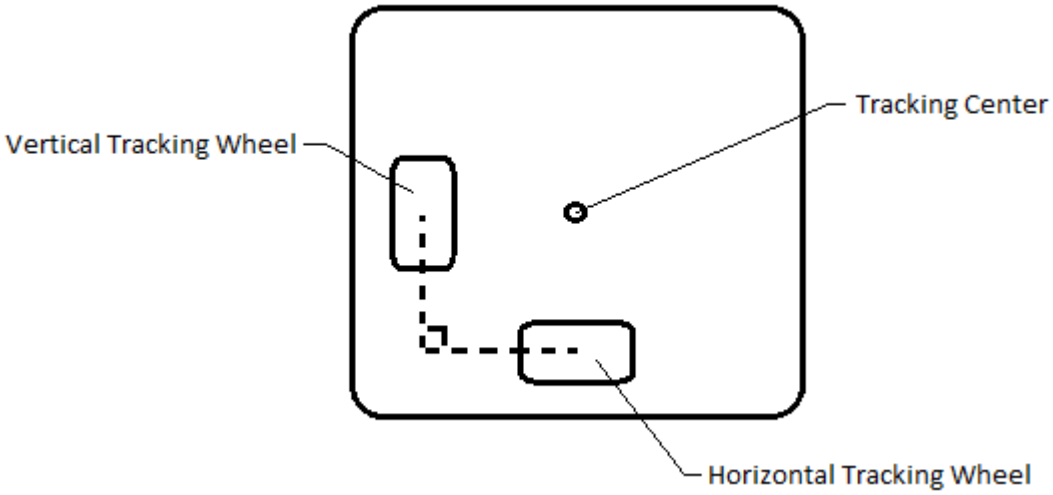

Figure 1 – Basic tracking wheel layout that we will be referring to.

Use of Inertial Sensor

The V5 inertial sensor continuously provides the angle heading of the robot during tracking. It should be mounted on a secure spot and preferably nested in a c-channel. We used rubber links to help absorb the shock on the inertial sensor when performing autonomous tasks (after all, the inertial sensor can only measure rotation up to 1000 degrees a second). To ensure the inertial sensor is as accurate as it can be during autonomous runs, make sure that you properly calibrate it with enough time before the run starts – any slight motion to the inertial sensor during calibration affects its accuracy over time.

Odometry

Our odometry theory takes the outputs of the quad-encoders and inertial sensor and translates them into XY coordinates (measured in inches) with the field acting as the first quadrant on a cartesian plane. Each calculation for this translation involves the filtering of readings from the quad-encoders to only include movement that would have occurred if driving straight, and then using right triangle trigonometry to project this filtered linear movement onto the field. Ideally, this calculation would occur as fast as possible, since movement usually cannot be perfectly straight in reality, thus requiring very small time intervals to simulate all movement (after being filtered) as straight within a small enough interval. The quicker the calculation interval, the longer the tracking will stay accurate – but the interval cannot be so small that sensor readings become meaningless.

Straight Movement

Let us look at the situation where the robot moves forward straight along the vertical tracking wheel axis shown in figure 1. In this situation, let us say that the robot started at position (0, 0) and was facing 0 degrees. If the robot moved 10 inches forward in this instance, then it would end up at position (0, 10) and still facing 0 degrees, indicating a distance traveled of 10 inches on only the y-axis of the field. Right angle trigonometry can be used to project this straight movement onto the field at any robot heading and for both tracking wheels (see next page):

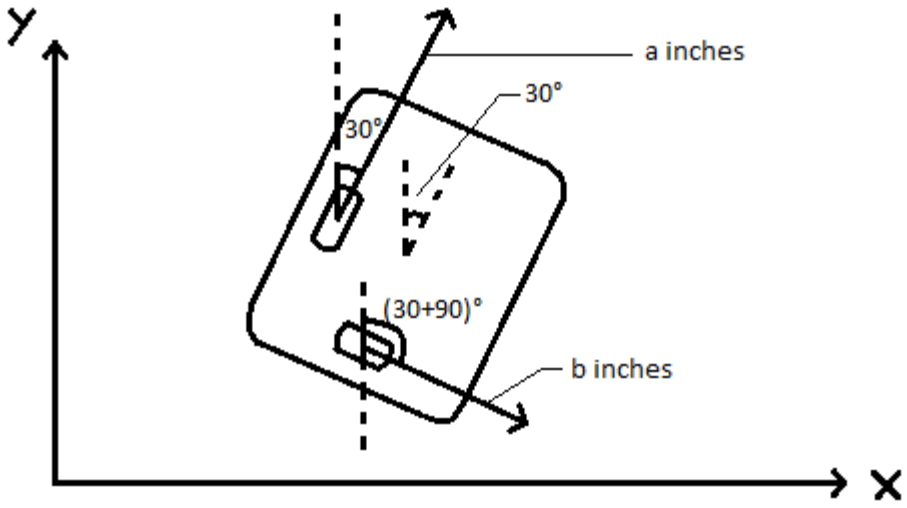

Figure 2 – Situation when the robot heading is 30 degrees and the vertical and horizontal tracking wheels move a and b inches, respectively. (Note: For future reference, let us assume that a and b are positive when detected as shown above but negative when opposite in direction for both wheels shown above.)

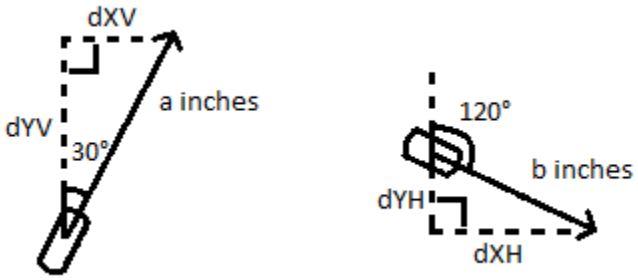

Figure 3 – The vertical (left) and horizontal (right) tracking wheels from the situation in figure 2 are shown. (Note: Because the tracking wheels are always perpendicular to each other, the direction of the horizontal tracking wheel is always 90 degrees from the vertical one.)

From the robot orientation in figure 2, we can see that the shift in the robot's position was detected by movement in both wheels by a and b inches on their respective axes. We can then separate a and b into their x and y component vectors (a.k.a. projecting a and b onto the field) using right angle trigonometry. The shift in position for the whole robot on the x-axis and y-axis would then be the sum of both wheels' x vectors and y vectors, respectively (see next page).

Let h be the robot heading (which was 30° in figure 2), and let dX and dY be the total changes in X and Y coordinates, respectively:

$$sin(h) = \frac{dXV}{a} \longrightarrow dXV = asin(h)$$

$$cos(h) = \frac{dYV}{a} \longrightarrow dYV = acos(h)$$

$$sin(h+90°) = \frac{dXH}{b} \longrightarrow dXH = bsin(h+90°)$$

$$cos(h+90°) = \frac{dYH}{b} \longrightarrow dYH = bcos(h+90°)$$

$$dX = dXV + dXH \qquad dY = dYV + dYH$$

In summary, dX and dY would be how much the robot center shifted along the x and y axes on the field, respectively. This shift would be a single instance of updating the robot position, and it would represent the movement that produces the sensor values in figure 2.

Turning Movement

A different situation: let us say that the robot still starts at position (0, 0) and is facing 0 degrees. If the robot turns clockwise to face 90 degrees, it will still be at position (0, 0), indicating the turning happened in place (assuming our tracking "center" is at the focal point of the drivetrain's turning arc). To ensure movement detected by the tracking wheels represents actual change in robot position, the turning movement must be filtered out of the movement readings:

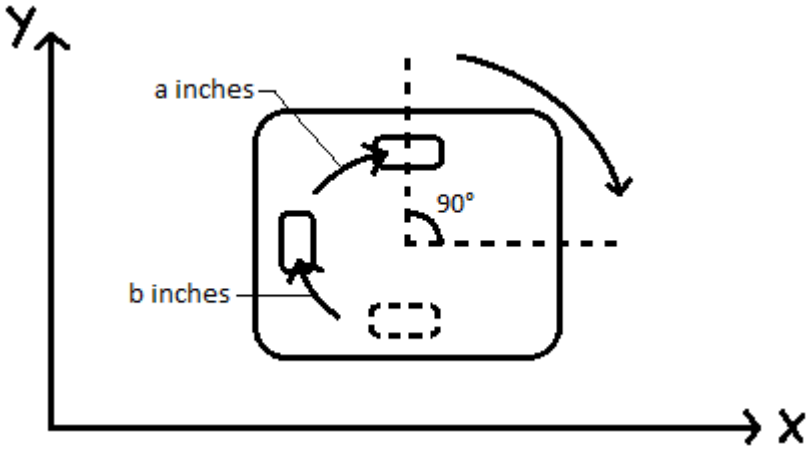

Figure 4 – Situation when the robot heading goes from 0 to 90 degrees after turning clockwise in place, the vertical and horizontal tracking wheels having moved a and b inches, respectively.

Figure 5 – The vertical (left) and horizontal (right) tracking wheels from the situation in figure 4 are shown. (Note: We are assuming that both tracking wheels are equidistant to the robot center.)

From the robot orientation in figure 4, we can see that there was actually no shift in the robot's position after the turn, despite there still being movement detected in both wheels (a and b inches). So, for this situation in particular, a and b both need to be completely filtered down to 0 inches by calculating that a and b are both the full arcs made by the change in robot heading on their respective wheels.

Let dH, which we will always assume is an absolute value, be the change in robot heading during this turn (which was 90° in figure 4), and let sA and sB be the distance moved after filtering out turning movement for the vertical and horizontal tracking wheels, respectively:

$$sA = a - 2\pi r \cdot \frac{dH}{360^\circ}$$

$$sB = b + 2\pi r \cdot \frac{dH}{360^\circ}$$

For counterclockwise turning:

$$sA = a + 2\pi r \cdot \frac{dH}{360^\circ}$$

$$sB = b - 2\pi r \cdot \frac{dH}{360^\circ}$$

In summary, because we adjusted a and b using the turning arcs to make sA and sB, we now know the change that a and b actually caused to the robot position. In this case, since a and b were the complete turning arcs on their respective wheels, sA and sB are both calculated to be 0 inches, thus each projecting onto the field axes as 0 inches total in dX and 0 inches total in dY during this instance of movement.

Ratios due to Tracking Wheel Placements

Depending on where your tracking wheels are placed relative to the chosen robot "center" on the base, they might experience a difference in movement detected while the robot is turning. This difference makes it so that a tracking wheel may experience an incorrect tracked distance per actual turned distance of the robot. This would happen when a tracking wheel is placed a certain distance (radius) away from the center, but the wheel is not perfectly perpendicular to the radius of the turning arc (circle) created by the robot center and the tracking wheel:

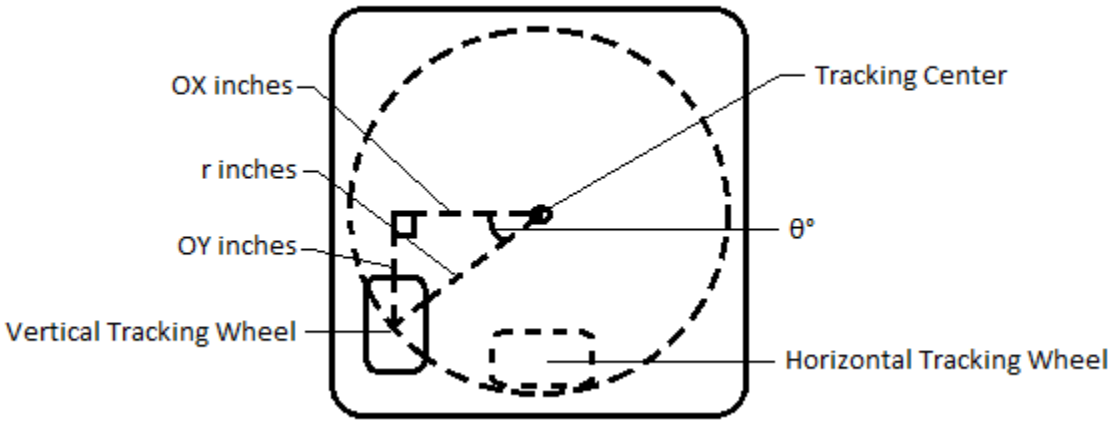

Figure 6 – The situation described above with a (vertical) tracking wheel not tangent to the turning arc is shown. The Pythagorean theorem can be used to calculate r as $\sqrt{(OX^2 + OY^2)}$ after measuring OX and OY by hand (this r is used for the purpose in figures 4 and 5).

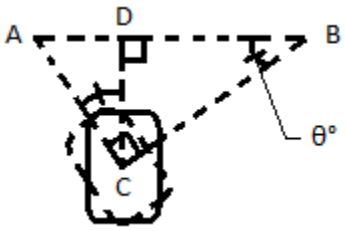

Figure 7 – Diagram to help prove that the θ from figure 6 is the same angle value as the difference in heading direction between the vertical tracking wheel in figure 6 and if the same wheel was perfectly perpendicular to the radius of the turning arc.

To correct the error when the tracking wheel is not placed perpendicular to the center radius (and thus enable us to place the tracking wheels anywhere on the robot base and still track properly), there is a constant ratio of actual distance traveled by the tracking wheel to distance traveled if the wheel was tangent to the turning arc that can be calculated. This ratio can then be applied to the actual tracking wheel readings to adjust the scale of the calculated error due to turning movement.

Figure 7 shows what the vertical tracking wheel from figure 6 would look like if tangent to its turning arc (shown as the dashed wheel at C). Of course, the vertical tracking wheel cannot actually have this orientation, since it must be perpendicular to the horizontal tracking wheel (as shown in figure 7 as the solid wheel at C). So, we must calculate the angle difference, $\angle ACD$, between the two orientations of the vertical tracking wheel to find the ratio of DC to AC, which is the cosine of $\angle ACD$. Conveniently, $\angle ACD$ is actually the same angle value as θ (shown in figure 7), which can be calculated using the tangent of θ. The tangent of θ is DC divided by BD, both of which can be easily measured using a ruler.

Proof that θ is equal to $\angle ACD$:

If,

$$\theta + \angle A + 90° = 180° \longrightarrow \theta + \angle A = 90°$$

$$\angle ACD + \angle A + 90° = 180° \longrightarrow \angle ACD + \angle A = 90°$$

then,

$$\theta + \angle A = \angle ACD + \angle A$$

$$\theta = \angle ACD$$

Calculating θ:

$$tan(\theta) = \frac{DC}{BD} \longrightarrow \theta = \arctan\frac{DC}{BD}$$

Calculating the ratio of DC to AC:

$$cos(\angle ACD) = \frac{DC}{AC} \longrightarrow cos(\theta) = \frac{DC}{AC}$$

Now that we know the ratio of DC to AC, we must apply it to the turning movement errors calculated in figure 5. This ratio is actually the triangular approximation of the difference between how much less the tracking wheel turns about its axis compared to if it was tangent to its turning arc (which is when distance detected by the wheel is 1:1 with the actual distance moved along the turning arc):

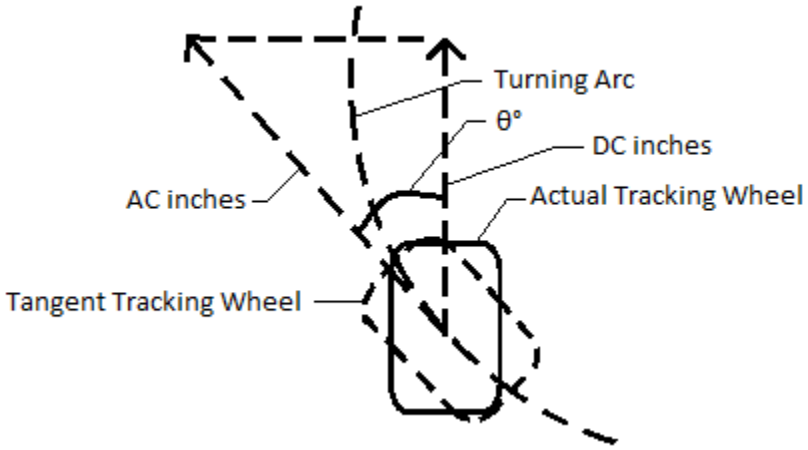

Figure 8 – Assuming that the distance traveled along the turning arc is linear in a small enough time interval, the actual tracking wheel would be getting a distance reading (DC) that is a constant fraction of the actual distance (approximated because linear) that the wheel travels along the turning arc (AC).

This ratio of DC to AC allows us to go back and forth between the actual tracking wheel reading and the hypothetical reading when tangent. Since we use the change in heading, dH, to calculate the turning arcs as the errors, we must go from the hypothetical readings (distances along these arcs) to the actual wheel readings, which are a and b in figure 5.

Let r_h and r_v be the radii of the horizontal and vertical tracking wheels' arcs, respectively, and let θ_h and θ_v be the angle differences between the tangent and actual orientations of the horizontal and vertical tracking wheels, respectively (θ_v and r_v are equivalent to θ and r in figure 6):

$$sA = a - 2\pi r_v \cdot \frac{dH}{360^\circ} \cdot cos(\theta_v)$$

$$sB = b + 2\pi r_h \cdot \frac{dH}{360^\circ} \cdot cos(\theta_h)$$

For counterclockwise turning:

$$sA = a + 2\pi r_v \cdot \frac{dH}{360^\circ} \cdot cos(\theta_v)$$

$$sB = b - 2\pi r_h \cdot \frac{dH}{360^\circ} \cdot cos(\theta_h)$$

In summary, because we had the measurements to calculate θ, we could then use the cosine of θ to convert between sides DC and AC on the approximated triangle formed by the tangent and actual orientations of the tracking wheel. In multiplying the turning arc error by $cos(\theta)$, we translated AC to DC for a single movement. This modification to the turning error corrected the raw tracking wheel reading to account for the wheel's position and orientation, as both the turning error and raw reading would be in terms of the actual wheel orientation. Thus, sA and sB would then be the filtered straight movement for the vertical and horizontal tracking wheels, respectively. These straight movements would then be accurate in terms of actual change in robot position when used to project onto the field axes as inches in dX and dY during this instance.

Implementation of Tracking System

The described odometry theory applies to a single instance of movement, meaning such an instance needs to keep being tracked over time repeatedly. Since our odometry only approximates distances by assuming they are always straight or perfectly in line with our arcs and triangles, only super short instances can be used to a degree that is accurate enough for robot autonomous programming to be effective. Short instances entail short time intervals, as we would use a while loop to continuously track position every iteration, waiting some milliseconds in between (we have found success with around 5-100 milliseconds).

We have found it helpful to treat the field as the first quadrant in a cartesian plane, making the robot's X and Y coordinates both positive values along axes that originate from a chosen corner of the field. The robot's coordinates would then always be relative to the chosen origin (0, 0).

It is important to not make any movements to the robot that may cause vibrations to the inertial sensor while it is calibrating, during which it should be given at least 2 seconds to fully and safely calibrate (this can be done before a match by starting the program and using the pre-auton capability).

Right before the start of the robot's autonomous routine, it is important to initialize the X and Y coordinates as the robot's actual starting position relative to (0, 0), and the inertial sensor and quad-encoders should be checked for any possible incorrect starting readings.

Extra Uses of Tracking System

In addition to pure position tracking, the odometry system can also be used for other purposes as well. Such purposes can include using the robot position on the field to determine distance from other points or determining which section of the field the robot is in. The robot position can also be used to create an algorithm to determine when a robot gets stuck somewhere along its autonomous routine.

Writing Motion Algorithms

The tracking system is only as useful as the motion algorithms that use it. One of the motion algorithms we created receives a target X and Y coordinate and other adjustable values, making a tank drive move along an arc on the field based on the initial robot heading and position. The algorithm ensures that, no matter what happens along the path taken to the target point, the robot will end up within an acceptable threshold relative to the target point at the end of the algorithm. We also enhanced our straight distance drive algorithm by allowing it to receive variable distances to drive based on the robot's position and desired target point. The tracking system was also used to enhance the accuracy of our point turns, allowing us to ensure that the robot ends up facing the actual point it will drive to after the turn algorithm completes. This has proved to be more consistent than just telling the robot to do point turns to end up at a specified angle heading.

Dealing with Accumulated Error

Since the tracking system's odometry theory is not perfect (it makes many assumptions), error will be accumulated in the robot's X and Y coordinates as time passes. Of course, there are ways you can slow the rate at which error is accumulated, such as adjusting the tracking time interval or making the tracking wheels more stable, but error will still eventually accumulate to the point where the position tracking is too inaccurate for autonomous tasks. Realistically speaking, the longest period of time where position tracking must be accurate in VRC is only 1 minute during the skills autonomous period. If there is still too much error within this minute, there are ways to "correct" the position tracking during the autonomous routine. For instance, there may be certain game objects that your robot can align itself with consistently, providing an opportunity for the tracking system to adjust its X and Y coordinates to the position aligned with the game object (obviously, you must be able to trust that this alignment comes reliably every autonomous run). Such game objects could be the field walls or game elements, such as poles, towers, platforms, barriers, or anything else that you might trust consistently.

Conclusion

We hope you find this document useful in your odometry endeavors. Please keep in mind that, since most of the theory in this document was developed with our own perspective on how to achieve odometry, you should take this information with a grain of salt (we are not necessarily professional mathematicians here at Igneous Robotics, but we can at least say that our odometry has been crucial to our success). VRC team 5225 (E-Bots Pilons) deserves a special thanks for their insightful Introduction to Position Tracking document, which was a big help and inspiration in developing our own odometry system. If you have any further questions or concerns, you can reach out to us on the VEX Forum or the unofficial public VEX Robotics Competition Discord server.

You can access the Pilons' document here:

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