Snapshot Raycasting Pose Correction

VEX Snapshot localization via distance-sensor raycasting.

Austin McGrath, 15800A - Atticus

January 2026

Abstract

A deterministic, one-shot pose correction system that uses V5 Distance Sensors and geometric raycasting against a tagged collision map. The goal is to provide a *snapshot* command that can be called inside an autonomous routine to correct odometry drift by inferring the robot's field position (x, y) at a known heading θ. The method is not a continuous probabilistic filter (it is not MCL / Bayesian tracking), but it does use a likelihood-style score: residuals are normalized by a sensor noise model and combined as a χ^2 error for gating and selection. The emphasis is reliability and minimal tuning via per-sensor field masks, robust sampling (median filtering), candidate surface selection via ray–segment intersection, and a conservative acceptance test.

1 Problem Statement and Design Goals

In VEX autonomous routines, wheel slip, contact events, and battery sag can cause odometry to drift. A full probabilistic localizer (e.g., MCL) can mitigate this continuously, but it adds computational cost and tuning overhead. This system provides a **snapshot** pose corrector: a one-shot command that takes a brief burst of distance measurements, solves for a consistent field position, and applies it as a one-time `set_position` update. It is intended as a quicker alternative when the robot can pause briefly and "see" stable landmarks (typically walls).

Non-goals. This method is not a continuous estimator. It does not attempt to track pose in real time. It also does not attempt to correct heading θ unless explicitly extended; heading is assumed to be supplied by IMU or the existing odometry heading state.

1.1 Design constraints (VEX Distance Sensor)

The V5 Distance Sensor measures distance in the range 20 mm to 2000 mm (about 0.79 in to 78.74 in). Accuracy is approximately ±15 mm below 200 mm, and approximately 5% above 200 mm. The sensor also provides a confidence value in $[0, 63]$ that is only meaningful at distances beyond 200 mm.

These constraints matter because the snapshot system must (i) prefer stable, large surfaces (walls) when possible, (ii) reject low-confidence returns, and (iii) avoid interpreting distant, low-signal readings as precise geometry.

1.2 Core idea

At a fixed heading θ, each distance sensor defines a geometric constraint:

- the sensor origin in the field frame is the robot pose plus a rotated mount offset;
- the sensor "looks" along a ray direction in the field frame;
- the observed distance corresponds to the first collision between that ray and the subset of map segments the sensor is allowed to detect.

With 2–3 sensors, the correct (x, y) is the translation that makes all measured distances agree with raycast distances to the same physical surfaces.

2 Geometry Model

2.1 Coordinate frames and pose

Let the robot pose in the field frame be

$$\mathbf{x} \;:=\; (x, y, \theta).$$

The field frame axes are fixed on the field. The robot frame is attached to the chassis center, with $+x_R$ to robot-right and $+y_R$ to robot-forward (this matches the binding convention used in the snapshot code).

Heading convention. This document uses the same convention as the JAR odometry binding: θ is measured in degrees with 0° pointing along $+Y$ (field "up"), and positive rotation is clockwise. To avoid ambiguity with the standard CCW-from-$+X$ math convention, we define field-frame basis vectors directly:

$$\hat{\mathbf{f}}(\theta) \;:=\; (\sin\theta,\ \cos\theta) \quad \text{(robot forward direction in field frame)},$$

$$\hat{\mathbf{r}}(\theta) \;:=\; (\cos\theta,\ -\sin\theta) \quad \text{(robot right direction in field frame)}.$$

Any robot-frame vector $\mathbf{v}^R = (v_{\text{right}}, v_{\text{fwd}})$ maps to the field frame as

$$\mathbf{v}^F = v_{\text{right}}\,\hat{\mathbf{r}}(\theta) + v_{\text{fwd}}\,\hat{\mathbf{f}}(\theta).$$

2.2 Sensor mount model

For sensor i, define:

- mount offset in robot frame: $\mathbf{o}_i^R = (x_{i,\text{right}}, y_{i,\text{fwd}})$ in inches;
- sensor look direction in robot frame: a unit vector $\hat{\mathbf{d}}_i^R(\phi_i)$ where ϕ_i is the sensor's relative angle (e.g., 0° forward, +90° right).

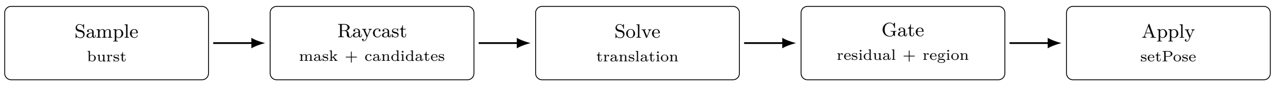

Figure 1: Snapshot pose pipeline (one-shot correction).

Define the sensor unit look direction in the robot frame using the same (right, fwd) axes:

$$\hat{\mathbf{d}}_i^R(\phi_i) \;:=\; (\sin\phi_i,\; \cos\phi_i).$$

Given a pose (x, y, θ), the sensor origin and direction in the field frame are

$$\mathbf{o}_i^F = (x, y) + x_{i,\text{right}}\,\hat{\mathbf{r}}(\theta) + y_{i,\text{fwd}}\,\hat{\mathbf{f}}(\theta),$$

$$\hat{\mathbf{d}}_i^F = (\sin\phi_i)\,\hat{\mathbf{r}}(\theta) + (\cos\phi_i)\,\hat{\mathbf{f}}(\theta).$$

A distance measurement z_i is interpreted as a point along the ray

$$\mathbf{r}_i(t) = \mathbf{o}_i^F + t\,\hat{\mathbf{d}}_i^F, \qquad t \geq 0,$$

where t is measured in inches (the implementation converts sensor mm to inches).

2.3 Collision map as tagged line segments

The environment is modeled as a finite set of **line segments**

$$\mathcal{S} = \{(\mathbf{a}_j, \mathbf{b}_j, m_j)\}_{j=1}^{N},$$

where $\mathbf{a}_j, \mathbf{b}_j \in \mathbb{R}^2$ are endpoints and m_j is a bitmask tag.

Masks represent which physical objects a segment belongs to, such as:

- `MAP_PERIMETER`: outer field walls (recommended default);
- `MAP_LONG_GOALS`: long goals / wall-aligned structures;
- `MAP_MATCHLOADERS`: matchloaders (optional; may require a higher-mounted sensor to see reliably);
- `MAP_CENTER_GOAL_POS45`, `MAP_CENTER_GOAL_NEG45`: individual center goals (optional; can be ambiguous);
- `MAP_CENTER_GOALS`: alias for both center goals;
- `MAP_ALL`: union of all categories.

Per-sensor mask overrides allow different sensors (mounted at different heights) to see different subsets of the map. A low sensor can be restricted to `MAP_PERIMETER`while a higher sensor may include `MAP_LONG_GOALS`or `MAP_CENTER_GOALS`.

3 Raycasting and Pose Solve

3.1 Ray–segment intersection

Each map segment j is represented as

$$\mathbf{s}_j(u) = \mathbf{a}_j + u(\mathbf{b}_j - \mathbf{a}_j), \qquad u \in [0, 1].$$

Ray–segment intersection solves

$$\mathbf{o}_i^F + t\,\hat{\mathbf{d}}_i^F = \mathbf{a}_j + u(\mathbf{b}_j - \mathbf{a}_j),$$

for unknowns $t \geq 0$ and $u \in [0, 1]$.

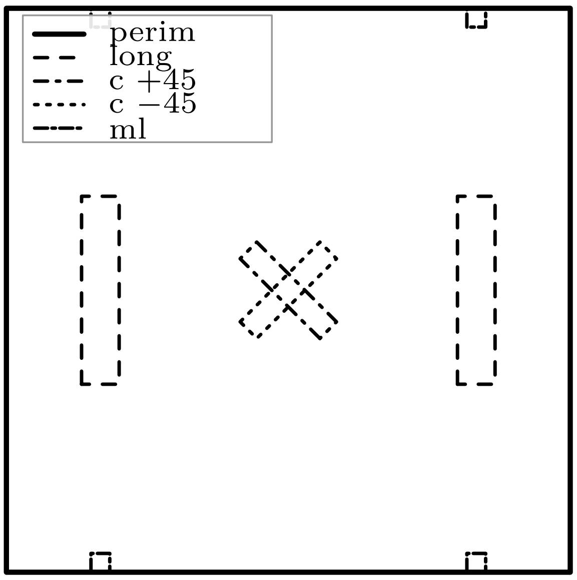

Figure 2: Tagged collision-map categories (schematic). Per-sensor masks select which categories participate in raycasts.

In implementation, this is solved as a 2D linear system in (t, u). A valid hit requires:

$$t \geq 0, \quad 0 \leq u \leq 1, \quad t \leq t_{\max},$$

where $t_{\max}$ is the sensor's max range (typically 2000 mm $\approx$ 78.7 in).

3.2 Mask filtering and candidate hits

For sensor i, define an effective mask:

$$m_i = \begin{cases} m_i^{\text{override}}, & m_i^{\text{override}} \neq 0 \\ m^{\text{global}}, & \text{otherwise.} \end{cases}$$

A segment j is eligible iff $(m_i \;\&\; m_j) \neq 0$.

Raycasting returns all valid hits, sorted by increasing t. The first hit is the predicted distance for a perfect measurement, but the solver retains up to K closest hits per sensor as **candidates** to handle ambiguity (e.g., a ray that could hit a goal edge or a wall depending on pose).

3.3 Snapshot measurements and robust sampling

Each sensor reading is sampled n times with a short delay and converted to inches. The median is used:

$$z_i = \text{median}\{z_{i,1}, z_{i,2}, \ldots, z_{i,n}\}.$$

This rejects occasional spikes without requiring a tuned filter gain.

Optional gates may reject a sensor for a snapshot:

- **range gate:** discard if z_i is out of the sensor's valid range;

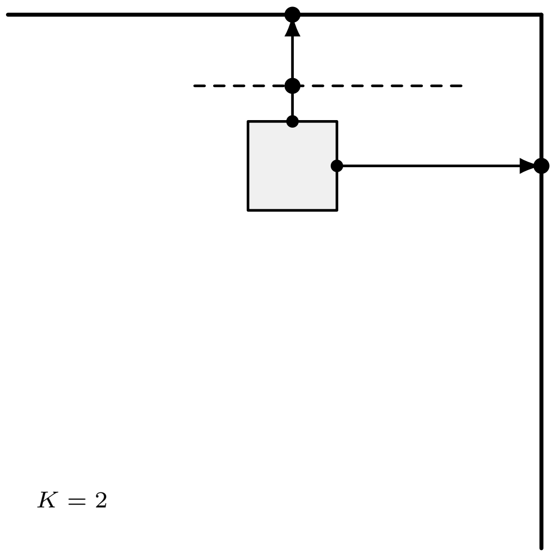

Figure 3: Raycasting on a local map patch. Under permissive masks, one ray may admit multiple plausible hit candidates (kept up to K); restrictive masks collapse ambiguity.

- **confidence gate:** discard if confidence $< c_{\min}$;
- **velocity / object-size gates:** optional, used only if your robot has consistent returns for those features.

3.4 Fixed-heading translation solve

This system assumes θ is known (IMU or odom heading). The unknown is translation (x, y). Given a candidate set of segments (one candidate per sensor), define the residual for sensor i as

$$r_i(x, y) = z_i - \hat{z}_i(x, y),$$

where $\hat{z}_i(x, y)$ is the raycast distance from the sensor origin at pose (x, y, θ) to the selected candidate segment.

A normalized error uses a distance-dependent noise model $\sigma(z)$. A practical model that matches published sensor behavior is:

$$\sigma(z) = \begin{cases} 15 \text{ mm}, & z < 200 \text{ mm} \\ 0.05\, z, & z \geq 200 \text{ mm}. \end{cases}$$

(Implementation converts mm to inches before use.)

The snapshot objective is a chi-square score:

$$\chi^2(x, y) = \sum_{i \in \mathcal{I}} \left(\frac{r_i(x, y)}{\sigma(z_i)} \right)^2,$$

where $\mathcal{I}$ is the set of sensors that passed gating.

Solve strategy used in the implementation. The code avoids iterative optimizers by converting each candidate hit segment into a *locus of feasible robot centers* and then finding a consistent intersection point.

For a fixed heading θ, sensor i with measurement z_i and field ray direction $\hat{\mathbf{d}}_i^F$ implies that the collision point is $\mathbf{p} = \mathbf{o}_i^F + z_i \hat{\mathbf{d}}_i^F$. If the ray hits map segment j with endpoints $\mathbf{a}_j, \mathbf{b}_j$, then the robot center (x, y) must satisfy $\mathbf{p} \in [\mathbf{a}_j, \mathbf{b}_j]$, i.e. $\mathbf{o}_i^F = \mathbf{p} - z_i \hat{\mathbf{d}}_i^F$. Rearranging yields a locus segment in the (x, y) plane:

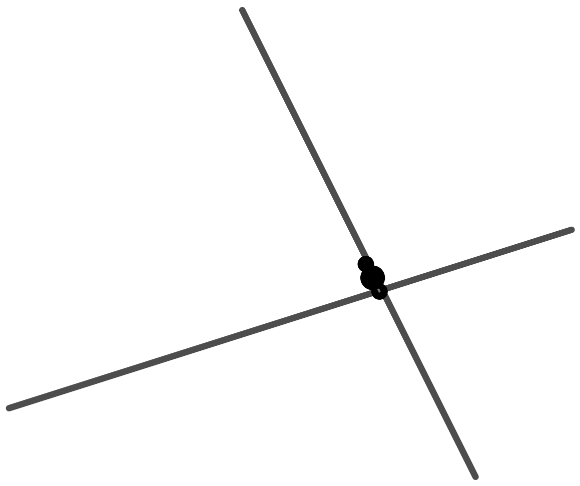

Figure 4: Deterministic translation estimate from locus segments. The implementation computes pairwise closest points between locus segments, then aggregates midpoints (median) and scores the result by χ^2.

$$\mathcal{L}_{i,j} := \left\{ \mathbf{a}_j - (\mathbf{o}_{i,\text{mount}}^F + z_i \hat{\mathbf{d}}_i^F),\ \mathbf{b}_j - (\mathbf{o}_{i,\text{mount}}^F + z_i \hat{\mathbf{d}}_i^F) \right\},$$

where $\mathbf{o}_{i,\text{mount}}^F$ is the sensor's mount offset rotated into the field frame (so $\mathbf{o}_i^F = (x, y) + \mathbf{o}_{i,\text{mount}}^F$). Every point on $\mathcal{L}_{i,j}$ is a robot-center location consistent with "sensor i hit segment j at distance z_i".

Deterministic search with likelihood-style scoring. The enumeration and locus construction are deterministic; the χ^2 score is the standard Gaussian normalized-residual sum (equivalently, a negative log-likelihood up to a constant) and is used only for gating and choosing between candidate combinations.

The solve proceeds as:

1. For each sensor i, raycast from a *guess pose* and keep up to K closest eligible hit segments as candidates.
2. Enumerate combinations of candidates (one segment per sensor).
3. Convert each chosen segment into a locus segment $\mathcal{L}_{i,j}$.
4. Estimate (x, y) by computing pairwise closest points between locus segments and taking the median of the resulting midpoints.
5. Score the estimate by χ^2 and keep the best passing solution.

This approach is robust, fast, and requires no gradient-based tuning parameters.

3.5 Quadrant constraint

To reduce incorrect snaps, a user may specify a quadrant:

`BL`, `BR`, `TL`, `TR`, or `ANY`.

Quadrant bounds (in inches) are:

$$\begin{aligned}\texttt{BL} &: 0 \le x \le 72,\quad 0 \le y \le 72\\ \texttt{BR} &: 72 \le x \le 144,\quad 0 \le y \le 72\\ \texttt{TL} &: 0 \le x \le 72,\quad 72 \le y \le 144\\ \texttt{TR} &: 72 \le x \le 144,\quad 72 \le y \le 144\end{aligned}$$

Optionally expand bounds by a small margin (e.g., 2 in) to tolerate boundary approaches.

3.6 Acceptance and validation

After a candidate solution $(x^\star, y^\star)$ is found, it is **validated** by re-raycasting from $(x^\star, y^\star, \theta)$ using each sensor's *effective mask* (global mask or per-sensor override) and verifying:

- each sensor residual satisfies $|r_i| \le r_{\max,i}$ (or $\chi_i^2 \le \chi_{\max}^2$);
- the final $(x^\star, y^\star)$ lies within the allowed quadrant (if specified);
- at least two sensors contributed (recommended minimum for 2D pose).

Implementation note: validation should use the same *effective* mask per sensor (global mask or per-sensor override) and compare against the nearest raycast hit under that mask. This ensures the final pose is consistent with what the sensor is actually allowed to "see".

If validation fails, the snapshot returns `ok=false` and does not modify odometry.

4 Implementation Notes (PROS), modeled around JAR

Portability note. The reference implementation and examples target PROS + JAR-Template odometry conventions (notably the heading convention used by `set_position`). The raycasting, masking, and fixed-heading translation solve are framework-agnostic: to port, replace only the pose/heading accessors and the final "apply pose" call.

4.1 Where the snapshot applies the result

The snapshot call is designed to be non-intrusive in auton code: it can live behind a single function call such as

```
snapshot_setpose_quadrant(q);
```

On success, it applies:

```
odom.set_position(x_in, y_in, heading_deg,
                  fwdTrackerIn, sideTrackerIn);
```

Tracker distances are passed through so odom remains internally consistent.

4.2 Configuration lives in `snapshot_bindings.cpp`

To keep user-facing code minimal, all setup is centralized:

- Distance sensor port definitions (or extern references to existing devices);
- sensor mount offsets and relative angles;
- per-sensor mask overrides (height differences);
- heading source (odom heading or IMU);
- optional tracker distance accessors.

Auton scripts should only call the snapshot function.

4.3 Recommended defaults

Table 1 lists safe defaults that work for most VEX robots without tuning.

4.4 Why the robot should be stopped

The snapshot assumes that all samples come from the same pose. If the robot moves during the sample window, different samples correspond to different (x, y, θ), which can cause:

- inconsistent constraints (solver failure or incorrect compromise solution);
- rotation-induced changes to sensor origin due to mount offsets;
- surface switching (the ray intersects a different segment while turning).

If a moving snapshot is required, an extension is to time-align each sensor sample with the odom pose at that timestamp and solve for a constant translation offset across the sample window. That extension remains deterministic but is outside the strict "stop-and-snap" operating mode.

4.5 Common failure modes and fixes

- **Ambiguous interior objects:** restrict masks to `MAP_PERIMETER`, or require a quadrant.
- **Low-confidence returns:** raise $c_{\min}$ and shorten range by lowering $t_{\max}$.
- **Wrong offsets:** re-measure mount offsets from the robot center; errors directly bias (x, y).
- **Heading mismatch:** ensure θ convention matches odom's convention used by `set_position`.

5 User Workflow (Setup and Proper Use)

5.1 Step-by-step setup

1. **Place the snapshot module in your PROS project.**
   Add the snapshot source files and headers (raycast, collision map, snapshot solver, bindings).
2. **Wire sensors in `snapshot_bindings.cpp`.**
   Set sensor ports or extern references for your project.
3. **Measure sensor offsets.**
   Measure $(x_{\text{right}}, y_{\text{fwd}})$ from robot center to each sensor's front face. Record in inches.
4. **Set sensor directions.**
   Set $\phi \in \{0, 90, 180, -90\}$ based on which way each sensor faces.
5. **Select field masks per sensor.**
   Low sensors: `MAP_PERIMETER`only. Higher sensors: `MAP_PERIMETER`plus goal segments if reliable.

6. **Select a heading source.**
   Use odom heading or IMU heading. Ensure convention matches JAR ($0°$ = +Y, CW positive).

7. **Build and perform a debug snapshot.**
   Park the robot at a known field location, stop, then trigger a snapshot and compare reported (x, y).

5.2 Proper use inside auton (minimal intrusion)

A recommended call site:

1. brake drive and allow motion to settle (150–250 ms);
2. call `snapshot_setpose_quadrant(QTR)` (or your known quadrant);
3. if `ok=true`, continue; if `ok=false`, proceed without modification.

5.3 When to call snapshot

Snapshot pose is most valuable when:

- just before an alignment-critical scoring action;
- after a contact-heavy interaction (goal clamp, wall intake sweep);
- at the start of auton when initial pose might be wrong.

5.4 Minimal tuning philosophy

The snapshot system is designed so the user only has to adjust:

- sensor offsets and directions (physical facts);
- per-sensor masks (height facts);
- optionally a quadrant (context the routine already knows).

Core thresholds (samples, confidence, max range) are stable across robots and can remain at defaults.

Table 1: Recommended default parameters for snapshot pose.

Parameter (default)	Rationale
Global fallback geometry mask (`MAP_PERIMETER`)	Walls are the most stable and least ambiguous landmarks. Start with perimeter-only until offsets, heading convention, and sensor wiring are verified.
Per-sensor mask override (`0` = inherit)	Each sensor may “see” different objects based on mounting height and view angle. Leave at 0 to inherit the global mask; override per sensor to exclude objects the sensor cannot reliably detect (e.g., low sensor: walls only; high sensor: walls + long goals).
Samples per sensor $n = 5$	Median-of-5 suppresses occasional spikes (weak returns, grazing angles) without creating a long time window. If you need faster snaps, reduce to $n = 3$ and keep the residual gate.
Inter-sample delay (35 ms)	Reduces correlation between successive reads (avoids repeatedly sampling the same internal frame). Increase slightly if readings appear “sticky” on your robot.
Valid range window (20 mm to 2000 mm)	Enforces the published operating range and rejects no-object / clipped values. Lowering $z_{\max}$ can improve reliability if interior objects are enabled and you want to ignore far geometry.
Confidence gating (enabled; $c_{\min} = 35$ for $z > 200$ mm)	`get_confidence()` is only meaningful at longer range; a moderate threshold rejects weak reflections without needing per-robot tuning.
Candidates per sensor ($K = 1$ walls; $K = 2$ if interior enabled)	K is how many ray-hit hypotheses are retained per sensor from the initial guess. Use $K = 1$ for perimeter-only; use $K = 2$ if any sensor mask includes interior objects so the solver can test alternate hit explanations instead of locking onto the wrong surface.
Quadrant margin (2 in)	Allows poses near $x = 72$ or $y = 72$ while still using a quadrant to reduce global ambiguity.
Per-sensor residual cap ($\chi^2_{\max} = 9$)	Rejects solutions where any single sensor is inconsistent by more than roughly 3σ. This prevents one bad hit from forcing a wrong snap.
Minimum sensors used (2)	Two independent constraints are required to solve a 2D translation (x, y). A third sensor improves robustness and makes bad-surface rejection more reliable.