Dynamic Localization through Deterministic Range Geometry and Sensor Fusion

Danish Tapia
MIT Tech Team
Email: danish.tapia017@gmail.com

Om Gunjal
MIT Tech Team
Email: om.gunjal@example.com

***Abstract*—This paper presents a lightweight deterministic localization algorithm for omnidirectional mobile robots operating in a known rectangular arena. The method fuses wheel-encoder dead-reckoning and IMU heading with up to four fixed range sensors to produce real-time pose estimates (x, y, θ). Localization is achieved by analytically predicting ray–wall intersections for each sensor, validating measured ranges with deterministic checks, and applying direct algebraic corrections to odometry offsets. The approach tolerates blocked sensors and transient occlusions, requires only a few arithmetic operations per cycle, and is suitable for microcontrollers. Representative C snippets are included. An experimental plan and baseline comparisons are proposed.**

***Index Terms*—Localization, Range sensors, Odometry, Embedded systems, Deterministic algorithm**

I. Introduction

Localization is a fundamental capability for mobile robots. In many practical systems (robotic soccer, arena robots, small warehouse robots) the environment map is known and compact (often a rectangular field). General-purpose localization solutions such as full SLAM, ICP-based scan matching, or particle filters are often computationally expensive for microcontroller-class hardware. We present an algorithm that operates with a minimal sensor suite (up to four single-beam distance sensors, wheel encoders and IMU heading) and uses analytic geometry and deterministic validation rules to correct odometry.

The main contributions of this work are:

- A deterministic, analytic localization pipeline that computes closed-form corrections from up to four range sensors and a known rectangular map.
- Robust validation heuristics to handle occlusions, sensor saturation, corner ambiguity and blocked sensors.
- A reference C-style implementation suitable for STM32-class microcontrollers, with an evaluation plan and baseline comparisons.

II. Related Work

Common localization approaches include Extended Kalman Filters (EKF), particle filters (Monte Carlo Localization), SLAM and scan-matching (ICP) techniques. EKF is lightweight but assumes near-linear Gaussian measurement models and can struggle with discontinuous measurement mappings induced by wall switching. Particle filters can represent multi-modal posteriors but require significant compute to maintain sufficient particle counts. ICP requires dense scans (lidar/depth) and nearest-neighbor matching, infeasible when only a few single-beam ranges exist.

Range-only localization has a dedicated literature: O'Kane and LaValle analyze minimal sensing limits for localization, showing that with heading information and a small set of contact/range sensors unique localization may be possible in polygonal maps. Recent optimization-based range-only solvers offer certificates but typically require heavier computation. Our method trades generality for speed by exploiting the rectangular map geometry to derive analytic constraints and deterministic updates — making it well suited for embedded platforms.

III. Problem Statement and Assumptions

We consider a planar holonomic robot whose pose is $p = (x, y, \theta)$ in world coordinates. The arena is a known rectangle defined by width W and length L. The robot carries:

- Up to four single-beam distance sensors at known rigid transforms $T_i = (x_{tf,i}, y_{tf,i}, \phi_i)$.
- Wheel encoders giving odometry (local pose increment).
- An IMU providing heading θ (with bounded drift).

Each sensor returns d_i, the distance along the ray at global angle $\theta + \phi_i$ to the nearest boundary of the rectangle. The measurement model is:

$$d_i = h_i(x, y, \theta) + \epsilon_i$$

where $h_i(\cdot)$ is the analytic ray–wall distance in the rectangular map.

Objective: compute corrected pose estimates suitable for control at moderate rates (e.g., 10–50 Hz) on microcontrollers with limited compute.

IV. Methodology

The algorithm pipeline (executed every localization cycle) is:

1) Form **combined pose** = odometry pose + stored offset.
2) Transform sensor mount poses to world frame using combined pose.
3) Compute analytic ray–wall intersections for each sensor beam to get predicted distances and wall IDs.
4) Validate each measured range with deterministic checks (saturation, occlusion, corner, wall-normal alignment).
5) For each valid sensor, algebraically compute a refined world-coordinate constraint (either x or y).

6) Resolve axis corrections by pairwise agreement (opposite sensors) or best-sensor selection; compute offsets and apply to odometry.

Below we provide selected, compact C-style snippets illustrating core computations (sensor transform, ray intersection and per-sensor correction). These are intentionally short; the full implementation follows the same logic and is available as supplementary material.

A. Sensor pose transform and ray intersection

Transform the sensor mount pose into world coordinates and compute analytic ray intersection with rectangle edges:

```
/* world pose of sensor i */
expected_sensor[i].x = combined_pose.x +
    sensor_tf[i].x * faster_cos(angle) -
    sensor_tf[i].y * faster_sin(angle);
expected_sensor[i].y = combined_pose.y +
    sensor_tf[i].x * faster_sin(angle) +
    sensor_tf[i].y * faster_cos(angle);
expected_sensor[i].theta =
    combined_pose.theta + sensor_tf[i].theta;

/* compute intersections with x=0, x=W, y=0,
    y=L */
t_min = LARGE;
if (fabs(dx) > EPS) {
    t = (0.0 - x_s) / dx; // left wall
    if (t>0 && y_s + t*dy >=0 && y_s + t*dy
    <= L) t_min = fmin(t_min,t);
    t = (W - x_s) / dx;   // right wall
    if (t>0 && y_s + t*dy >=0 && y_s + t*dy
    <= L) t_min = fmin(t_min,t);
}
if (fabs(dy) > EPS) {
    t = (0.0 - y_s) / dy; // bottom wall
    if (t>0 && x_s + t*dx >=0 && x_s + t*dx
    <= W) t_min = fmin(t_min,t);
    t = (L - y_s) / dy;   // top wall
    if (t>0 && x_s + t*dx >=0 && x_s + t*dx
    <= W) t_min = fmin(t_min,t);
}
expected[i] = t_min;
diff[i] = expected[i] - measured[i];
```

Listing 1: Sensor transform and ray–wall intersection (compact)

B. Validation logic

Each reading is classified deterministically:

- `MAXVAL` if saturated (no valid reading).
- `CORNER` if predicted hit is close to a corner (ambiguous).
- `BLOCKED` if measured is significantly smaller than predicted (occlusion).
- `OUTSIDE` if measured much larger than prediction.
- `ANGLE_INVALID` if beam angle differs from wall normal beyond tolerance (e.g., 10°).
- `VALID` otherwise.

Only `VALID` sensors are used for corrections.

C. Per-sensor pose inference and offset application

If a sensor hits a vertical wall (x=0 or x=W), we algebraically derive x:

$$x = X_{\text{wall}} - d\cos(\theta_s)$$

and similarly for horizontal walls (derive y). After transforming these refined coordinates back into odometry frame (by subtracting sensor transform rotated by heading), the algorithm uses either:

- Pairwise agreement if front/back or left/right both see the same axis — average them and mark axis corrected.
- Best-sensor selection: choose the valid sensor whose refined coordinate is closest to odometry (if within tolerance).

When an axis correction is accepted:

```
offset.x = sensor_derived_pose.x -
    xy_module_pose.x;
combined_pose.x = xy_module_pose.x + offset.x;
```

Listing 2: Apply offset when axis corrected

V. Implementation Notes

A. Timing and rates

The algorithm is designed for periodic invocation (e.g., 20–50 Hz) using a hardware timer. Heavy computation is avoided: each cycle performs $O(\text{NUM\_SENSORS})$ trig and arithmetic operations and cheap branching logic, making it suitable for 32-bit microcontrollers (STM32 family).

B. Calibration

Accurate `sensor_tf` (x,y,theta) and ADC-to-distance scaling are essential. The code includes per-sensor scale/offset constants; automated calibration is recommended in the deployment phase.

VI. Evaluation Plan

We propose the following experiments to quantify performance and compare baselines.

A. Metrics

- Pose RMSE for x, y, and θ against ground truth (motion capture).
- Correction latency (ms per cycle) and CPU utilization on target MCU.
- Axis-correction success rate (fraction of cycles producing $\text{dead}_x/dead_y$).
- Robustness to blocked sensors: evaluate for 0–4 sensors blocked intermittently.

B. Baselines

- EKF fusing odometry, heading, and range measures (linearized measurement model).
- Lightweight particle filter (small particle budget tuned to MCU).

C. Scenarios

- Static pose sweeps: rotate-in-place at several known coordinates.
- Trajectory runs: straight, curves, and corner transits with intermittent occlusions.
- Adversarial occlusion tests: temporary occluders placed in front of sensors.

VII. Discussion

A. Strengths

- Deterministic, closed-form corrections when sufficient sensor data exist.
- Very low computational overhead — suitable for microcontrollers.
- Graceful degradation: falls back to odometry if sensors blocked.

B. Limitations and failure modes

- Requires a known map (here, a rectangle). Extension to arbitrary polygons requires generalizing ray–wall intersection logic.
- Sensitive to heading error: if IMU heading is severely off, wall identification fails. Alignment checks mitigate but do not fully remove this risk.
- Ambiguity near corners — the implementation classifies and rejects near-corner readings.
- Calibration of sensor mounts and ADC scaling is crucial; small biases translate to pose bias.

C. Mitigations and extensions

Possible extensions include:

- Lightweight probabilistic smoothing (small EKF) to handle sensor noise without heavy compute.
- Joint heading refinement when multiple sensors disagree — solve small nonlinear system for θ when computational budget allows.
- Extend analytic geometry to general polygonal maps while keeping operations efficient.

VIII. Conclusion

We have described a deterministic, minimal-compute localization approach for robots in known rectangular arenas exploiting analytic ray–wall geometry and deterministic sensor validation. The algorithm is implementable in C on microcontrollers, tolerates partial sensor failure/occlusion, and provides reliable axis corrections to bound odometry drift. Future work will evaluate empirical performance on hardware and extend the approach to more complex maps.

Acknowledgment

The author thanks the MIT Tech Team for support and testers who provided early feedback on embedded implementation. (Fill formal acknowledgments as needed.)

References

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in *AAAI*, 1999.
[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press.
[3] J. M. O'Kane and S. M. LaValle, "Localization with limited sensing," 2007.
[4] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. PAMI*, 1992.
[5] F. Dümbgen et al., "Certified Continuous-Time Range-Only Localization," arXiv, 2022.