

On SLAM: ARAEL optimization framework

Rust crate for building robust and efficient SLAM systems

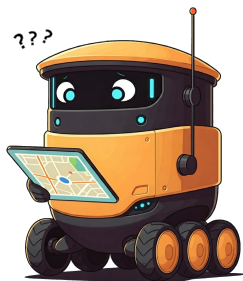
Indrek Mandre <indrek@mare.ee>

Computer Vision & Perception Meetup
02.07.2026 Tallinn

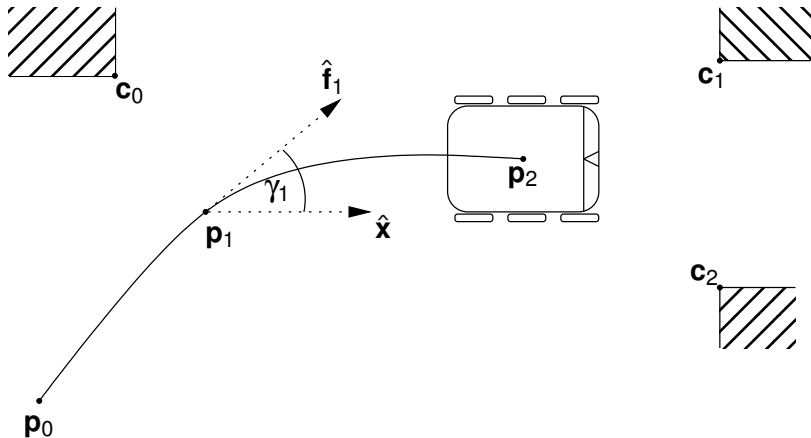


Mapping algorithms: SLAM algorithms

- To know where to go, we need to know where we are
- To know where we are, we need a map
- To create a map, we need to know where we are
- Chicken and egg problem
- Solution: Simultaneous Localization and Mapping (SLAM)
- The path of the mapping agent is part of the map



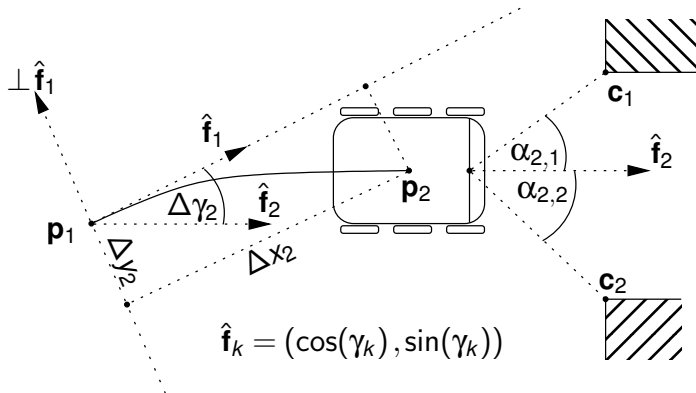
Model of the map



$$\mathbf{p}_k = (x_k, y_k, \gamma_k), \quad \mathbf{c}_k = (c_{k,x}, c_{k,y}),$$

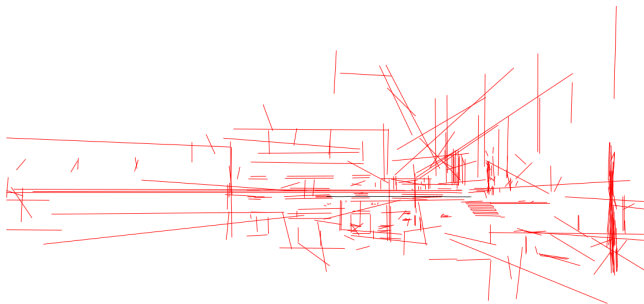
$$M = [x_0, y_0, \gamma_0, \dots, x_N, y_N, \gamma_N, c_{0,x}, c_{0,y}, \dots, c_{|C|,x}, c_{|C|,y}]$$

Sensor readings



$$\Delta \mathbf{p}_2 = (\Delta x_2, \Delta y_2, \Delta \gamma_2), \quad s_{2,1} = (\alpha_{2,1}), \quad s_{2,2} = (\alpha_{2,2}),$$

$$L = [\Delta x_1, \Delta y_1, \Delta \gamma_1, \dots, \Delta x_N, \Delta y_N, \Delta \gamma_N, \alpha_{i,j}, \dots]$$



Sensor readings: normal distribution

We have

$$L = [\Delta x_1, \Delta y_1, \Delta \gamma_1, \dots, \Delta x_N, \Delta y_N, \Delta \gamma_N, \alpha_{0,0}, \dots].$$

We say that L follows a normal distribution:

$$L \sim \mathcal{N}(\mu_L, \mathbf{K}_L).$$

$\mathbf{K}_L$ is a block-diagonal matrix

$$\mathbf{K}_L = \begin{bmatrix} K_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & K_m \end{bmatrix}.$$

If we had a perfectly correct map (ground truth) M_{GT} then

$$\mu_L = (F_0(M_{\text{GT}}), \dots, F_{|L|}(M_{\text{GT}})).$$

Finding the map is an optimization problem

The best map is the most likely one given the sensor readings:

$$M_{\text{BEST}} = \arg \max_{M \in \mathbb{M}_{\text{search}}} P(M|L).$$

Through Bayes' theorem, assuming flat prior and $L \sim \mathcal{N}(\mu_L, \mathbf{K}_L)$:

$$P(M|L) \simeq C e^{-\frac{1}{2}(L - \mu(M))^T K_L^{-1} (L - \mu(M))},$$

$$M_{\text{BEST}} = \arg \min_{M \in \mathbb{M}_{\text{search}}} (L - \mu(M))^T K_L^{-1} (L - \mu(M)).$$

Diagonalize by substituting $K_L = RDR^T$, $L^D = R^T L$, $G = R^T \mu(M)$:

$$M_{\text{BEST}} = \arg \min_{M \in \mathbb{M}_{\text{search}}} \sum_i \frac{(L_i^D - G_i(M))^2}{\sigma_i^2}$$

Levenberg-Marquardt algorithm – Gauss-Newton based

Let

$$S = \sum_i \frac{(L_i^D - G_i(M))^2}{\sigma_i^2} = \sum_i r_i^2.$$

We calculate the approximate Hessian matrix $\tilde{\mathcal{H}}$ of S :

$$\left[\tilde{\mathcal{H}} \right]_{jl} = \sum_i 2 \frac{\partial r_i}{\partial m_j} \cdot \frac{\partial r_i}{\partial m_l} \bigg|_{M_{k-1}}.$$

We find step Δ_k and iterate until reaching a minimum

$$\begin{aligned} \left(\tilde{\mathcal{H}} + \lambda \cdot \text{diag}(\tilde{\mathcal{H}}) \right) \Delta_k &= -\nabla S(M_{k-1}), \\ M_k &= M_{k-1} + \Delta_k. \end{aligned}$$

Rust: data structure

```
1  struct Path {
2      poses: refs::Deque<Pose>,
3      landmarks: refs::Vec<Landmark>,
4  }
5  struct Pose {
6      pos: Param<vect2f>,
7      gamma: Param<f32>,
8      delta_pos: vect2f,
9      delta_gamma: f32,
10 }
11 struct Landmark {
12     pos: Param<vect2f>,
13     frines: std::vec::Vec<Frine>,
14 }
15 struct Frine {
16     pose: Ref<Pose>,
17     bearing: f32,
18 }
```

ARAE: defining the constraint

ARAE: Rust macro-based compile-time solver generation –

<https://github.com/harakas/arael> – <https://crates.io/crates/arael>

```
1  #[arael::model]
2  #[arael(constraint(hb, parent = lm, {
3      let world_angle = pose.gamma + frine.bearing;
4      let aligned = matrix2sym::rotation(world_angle)
5          .transpose() * (lm.pos - pose.pos);
6      [atan2(aligned.y, aligned.x) * frine.isigma]
7  })])
8  struct Frine {
9      #[arael(ref = root.poses)]
10     pose: Ref<Pose>,
11     bearing: f32,
12     isigma: f32,
13     hb: CrossBlock<Landmark, Pose, f32>,
14 }
```

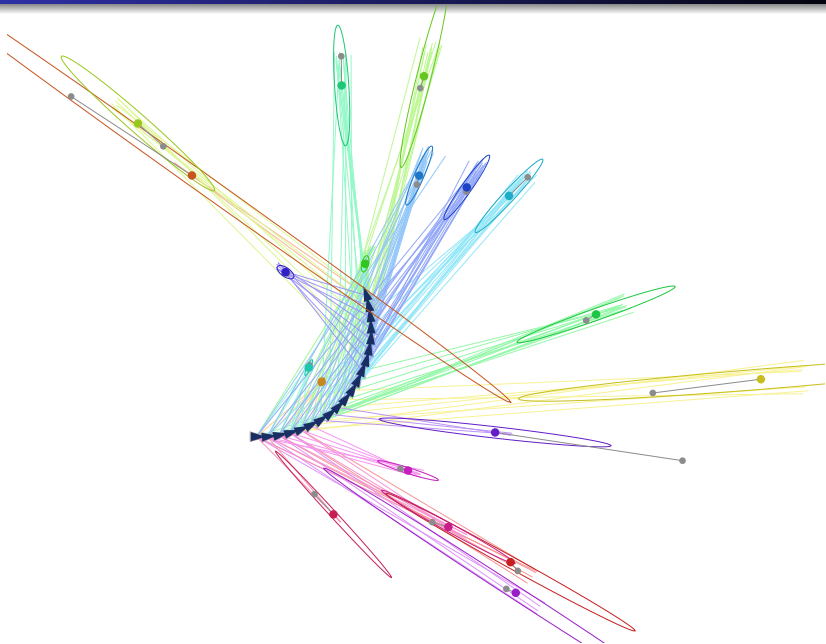
ARAE: running the solver

```
1  let cfg = Cfg::default();
2  let mut path = build_path(&cfg);
3  let mut params: std::vec::Vec<f32> = std::vec::Vec::new();
4  path.serialize32(&mut params);
5  let lm_cfg = arael::simple_lm::LmConfig::<f32>
6      { verbose: true, ..Default::default() };
7  let result = arael::simple_lm::solve_sparse_faer_f32(&params,
8      &mut path, &lm_cfg);
9  path.deserialize32(&result.x);
10
11  // To manually calculate the gradient and Hessian:
12  let n = params.len();
13  let mut grad = vec![0.0_f32; n];
14  let mut hessian = vec![0.0_f32; n * n];    // row-major
15  path.calc_grad_hessian_dense(&params, &mut grad, &mut hessian);
```

ARAE: compile-time macro generated code

```
1  $ cargo expand --example slam2d_simple_demo |grep -A 256 "fn __compute_blocks"
2      fn __compute_blocks(&mut self, params: &[f32], grad: &mut [f32]) {
3          ..
4          for __item in self.landmarks.iter_mut() {
5              let landmark = unsafe { &*(__item as *const Landmark) };
6              let path = &*__self_ref;
7              {
8                  /// arael: Frine[Frine] @ examples/slam2d_simple_demo.rs:85
9                  let lm = unsafe { &*(__item as *const Landmark) };
10                 for __frine in __item.frines.iter_mut() {
11                     let pose = &self.poses[__frine.pose];
12                     let path = &*__self_ref;
13                     {
14                         let __x13 = lm.pos.work().y - pose.pos.work().y;
15                         let __x11 = lm.pos.work().x - pose.pos.work().x;
16                         let __x2 = __frine.bearing + pose.gamma.work();
17                         let __x5 = __x2.sin();
18                         ..
19                         let __r_0 = __frine.isigma * __x0.atan2(__x1);
20                         let __dr_0_0 = __frine.isigma * (-__x14 - __x9) * __x15;
21                         let __dr_0_1 = __frine.isigma * (__x12 - __x8) * __x15;
22                         unsafe {
23                             (*__a_self_block_ptr)
24                                 .add_residual(
25                                     __r_0 as f32,
26                                     &[__dr_0_0 as f32, __dr_0_1 as f32],
27                                     grad,
28                                 );
29                     }
30                     ..

```



ARAE: robust estimation to suppress outliers

