Modern Code Optimization Snippets

A compact, aesthetic reference for MATLAB implementations used in the SLAM project

Quick overview

Each algorithm below is presented as a small MATLAB code snippet (suitable for copy-paste into a ‘.m‘ file) followed by practical optimization notes to improve numerical robustness, runtime, and real-world performance.

1. Dead-Reckoning

Dead-Reckoning (core update)

```
% Dead-Reckoning pose update (discrete)
dt = 0.1;                    % timestep
x = x_prev(1); y = x_prev(2); th = x_prev(3);
v = u(1); w = u(2);

x_next = x + v * dt * cos(th);
y_next = y + v * dt * sin(th);
th_next = wrapToPi(th + w * dt);

x_pred = [x_next; y_next; th_next];
```

Optimization Tips

- Apply periodic landmark-based corrections (every k steps) to reset accumulated drift.
- Estimate and subtract sensor biases (gyro/encoder) online using slow adaptive filters.
- If available, fuse a low-rate absolute sensor (GPS/visual) with a complementary filter to bound drift.

2. Extended Kalman Filter (EKF)

EKF (predict + update - compact)

```
% Predict
x_pred = motionModel(x, u, dt);          % nonlinear propagate
F = jacobian_motion(x, u, dt);          % 3x3 jacobian
P_pred = F * P * F' + Q;

% Update (for one landmark measurement z = [r; bearing])
[z_pred, H] = measurementModel(x_pred, lm); % z_pred (2x1), H (2x3)
S = H * P_pred * H' + R;
K = P_pred * H' / S;
y = [r_meas; b_meas] - z_pred;
y(2) = wrapToPi(y(2));
```

```
x_upd = x_pred + K * y;
P_upd = (eye(size(P)) - K * H) * P_pred;
```

Optimization Tips

- Maintain numerical symmetry and positive-definiteness of P (enforce $P = (P + P')/2$).
- Use innovation gating (Mahalanobis test) to reject outliers: accept if $y^\top S^{-1} y < \chi^2_{df,\alpha}$.
- Adapt Q, R conservatively when repeated large innovations occur (scale by small factors).

3. Unscented Kalman Filter (UKF)

UKF (sigma-point prediction)

```
% Generate sigma points (n=3)
[n, ~] = size(x);
lambda = alpha^2 * (n + kappa) - n;
chi = sigma_points(x, P, lambda); % returns 2n+1 columns

% Propagate sigma points through motion
for i = 1:size(chi,2)
    chi_pred(:,i) = motionModel(chi(:,i), u, dt);
end

% Weighted mean & covariance
x_pred = sum(repmat(Wm, n, 1) .* chi_pred, 2);
P_pred = Q;
for i = 1:size(chi_pred,2)
    d = chi_pred(:,i) - x_pred;
    d(3) = wrapToPi(d(3));
    P_pred = P_pred + Wc(i) * (d * d');
end
```

Optimization Tips

- Use numerically stable Cholesky for square-root of P when forming sigma points.
- Tune (α, β, κ) to match prior statistics (typical: $\alpha = 1e-3, \beta = 2, \kappa = 0$).
- Wrap angles whenever computing differences to avoid filter inconsistency.

4. Quantum Particle Swarm Optimization (QPSO)

QPSO (core loop)

```
Np = 50; max_iter = 30;
particles = repmat(x_init,1,Np) + randn(3,Np).*diag([0.5,0.5,
deg2rad(10)]);
pbest = particles; pbestJ = inf(1,Np);

for it = 1:max_iter
    mbest = mean(pbest, 2);
    beta = 0.9 - 0.8 * (it/max_iter); % contraction schedule
    for i = 1:Np
        u = rand(3,1);
        phi = rand(3,1);
        attractor = phi .* pbest(:,i) + (1-phi) .* gbest;
        particles(:,i) = attractor + beta .*
        abs(mbest - particles(:,i)) .* log(1./u);
        particles(3,i) = wrapToPi(particles(3,i));
        J = evaluateFitness(particles(:,i), z, lm);
        if J < pbestJ(i), pbest(:,i) = particles(:,i);
        pbestJ(i) = J; end
    end
    [gbestJ, idx] = min(pbestJ); gbest = pbest(:,idx);
end
```

Optimization Tips

- Start with broader spread, decay contraction parameter to refine search.
- Use parallel fitness evaluation (parfor) to speed up evaluation across particles.
- Use multi-start restarts to reduce risk of converging to poor local minima.

5. LSTM Residual Learning

LSTM residual prediction (training + infer)

```
% Define model (MATLAB Deep Learning Toolbox style)
layers = [ sequenceInputLayer(inputDim)
           lstmLayer(64,'OutputMode','last')
           fullyConnectedLayer(32)
           reluLayer
           fullyConnectedLayer(3)
           regressionLayer ];

options = trainingOptions('adam', 'MaxEpochs',100,
'MiniBatchSize',64);

% Train: net = trainNetwork(XTrain, YTrain, layers, options);
```

```
% Inference (predict residual)
res = predict(net, feat);        % feat = [x; u] sequence window
x_nn_next = motionModel(x, u, dt) + res;
x_nn_next(3) = wrapToPi(x_nn_next(3));
```

Optimization Tips

- Train with varied noise, speeds, and maneuvers to generalize residual corrections.
- Regularize (dropout/L2) to avoid overfitting to particular trajectories.
- Combine NN residuals with filter uncertainty (e.g., blend by confidence) rather than naive replacement.

6. GraphSLAM (pose graph optimization)

GraphSLAM (Gauss–Newton solve sketch)

```
% X_init: stacked poses (3 x T)
% Build residuals & Jacobians for odom and measurement edges
% Form sparse H and b
H = sparse(3*T, 3*T); b = zeros(3*T,1);

% (pseudo-loop)
for each edge
    [e, J_i, J_j] = edgeErrorAndJacobian(X, edge);
    idx_i = indexOfPose(edge.i);
    idx_j = indexOfPose(edge.j);
    Omega = inv(edge.cov);
    H(idx_i,idx_i) = H(idx_i,idx_i) + J_i' * Omega * J_i;
    H(idx_i,idx_j) = H(idx_i,idx_j) + J_i' * Omega * J_j;
    ...
    b(idx_i) = b(idx_i) + J_i' * Omega * e;
end

% Anchor first pose to fix gauge
H(1:3,1:3) = H(1:3,1:3) + eye(3)*1e6;

dx = - (H \ b); % use sparse Cholesky for large graphs
X_opt = X_init + reshape(dx,3,[]);
```

Optimization Tips

- Exploit block-sparsity and use sparse Cholesky (CHOLMOD) or incremental solvers (iSAM2) for scalability.
- Add robust kernels (Huber) on measurement residuals before assembling H to reduce outlier influence.

- Initialize with odometry/EKF to ensure faster convergence.

Notes:

- The snippets are intentionally compact — expand modular helper functions (e.g., ‘motion-Model‘, ‘measurementModel‘, ‘sigma$_{p}oints$‘, ‘$evaluateFitness$‘)$forclarityandreuse.$
- Always wrap angular differences with ‘wrapToPi‘ before using them in cost or covariance calculations.
- For production, prefer matrix-safe operations and preallocate arrays for performance.