Akula-robot localization with wheeled odometry

Timur Bayburin
Skolkovo Institute of Science and Technology (Skoltech)
Moscow, Russia
Timur.Bayburin@skoltech.ru

Ivan Gerasimov
Skolkovo Institute of Science and Technology (Skoltech)
Moscow, Russia
Ivan.Gerasimov@skoltech.ru

Valerii Kornilov
Skolkovo Institute of Science and Technology (Skoltech)
Moscow, Russia
Valerii.Kornilov@skoltech.ru

Denis Kukushkin
Skolkovo Institute of Science and Technology (Skoltech)
Moscow, Russia
Denis.Kukushkin@skoltech.ru

Fakhriddin Tojiboev
Skolkovo Institute of Science and Technology (Skoltech)
Moscow, Russia
Fakhriddin.Tojiboev@skoltech.ru

***Abstract*—This document describes an approach used for planar pose estimation of skid-steering wheeled robot based on data from encoders.**

***Index Terms*—wheel odometry, skid kinematics, LiDAR**

I. Introduction

Odometry is dead reckoning estimation process that becomes unstable as the time growth. But in short term perspective it can provide reliable estimation of robot state. Odometry data can also be combined with other data sources to get more precise estimations.

We developed odometry for skid-steering four-wheeled robot using wheel encoders data and compared acquired estimations to lidar odometry data.

II. Kinematic and Sensor Models

A. Kinematic model

In the project we used arc kinematic model 1. Here the noise is added in the action space. v_t is a linear speed and w_t is an angle speed. We suppose that the noise has normal distribution, so: $\epsilon_{v_t} \sim \mathcal{N}(0, a_1 v_t^2 + a_2 w_t^2)$, $\epsilon_{w_t} \sim \mathcal{N}(0, a_3 v_t^2 + a_4 w_t^2)$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} \frac{-\hat{v}}{\hat{w}}(\sin(\theta) - \sin(\theta + \hat{w}\Delta t)) \\ \frac{\hat{v}}{\hat{w}}(\cos(\theta) - \cos(\theta + \hat{w}\Delta t)) \\ \hat{w}\Delta t \end{bmatrix} \quad (1)$$

$$\hat{v} = v + \epsilon_v,\ \hat{w} = w + \epsilon_w$$

B. Sensor model (wheeled odometry)

The sensor model is given in 2. Our sensor data is generated from the encoder. As input we have cumulative number of ticks for left and right wheels at each observation time. Firstly, we estimate distance passed by each wheel from the last timestamp (u_L, u_R). As we'd like to work with distances in meters rater than ticks, we introduce special coefficient k, which allows translation: $k = \frac{2\pi r}{CPR}$, where $CPR_l \approx 2594$ (for left wheel) and $CPR_r \approx 3248$ (for right wheel) is counts per rotation (estimated empirically) and $r = 0.13$ is wheel's radius in meters.

$$\begin{bmatrix} u_R \\ u_L \end{bmatrix}_t = \frac{\Delta\theta}{2} \begin{bmatrix} (\frac{c}{\sin(\frac{\Delta\theta}{2})} + l)/k_r \\ (\frac{c}{\sin(\frac{\Delta\theta}{2})} - l)/k_l \end{bmatrix} \quad (2)$$

$$c = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

$$\Delta x = x_t - x_{t-1},\ \Delta y = y_t - y_{t-1},\ \Delta\theta = \theta_t - \theta_{t-1}$$

$l = 0.49$ is inter-wheel distance.

III. Speed estimation

In order to find the velocity of the right and left wheels of robot we firstly empirically estimates the coefficient of linear relationship between PWM duty cycle and wheel speed. Let the left and right wheels of the robot passes Δl, Δr distances respectively for Δt and inp_t be the voltage at time t and we control the measure of voltage through devices such as phone or laptop. We find the speed of wheel as follows:

$$k_{il} = \frac{\Delta l}{\Delta t \cdot inp_t} \quad v_l = k_{il} \cdot inp_t \quad (3)$$

$$k_{ir} = \frac{\Delta r}{\Delta t \cdot inp_t} \quad v_r = k_{ir} \cdot inp_t \quad (4)$$

We divide by inp_t in order to normalize.
Now we can define v, w, which are used in 1:

$$v = \frac{v_r \cdot k_r \cdot k_{ir} + v_l \cdot k_l \cdot k_{il}}{2} \quad (5)$$

$$w = \frac{v_r \cdot k_r \cdot k_{ir} - v_l \cdot k_l \cdot k_{il}}{l} \quad (6)$$

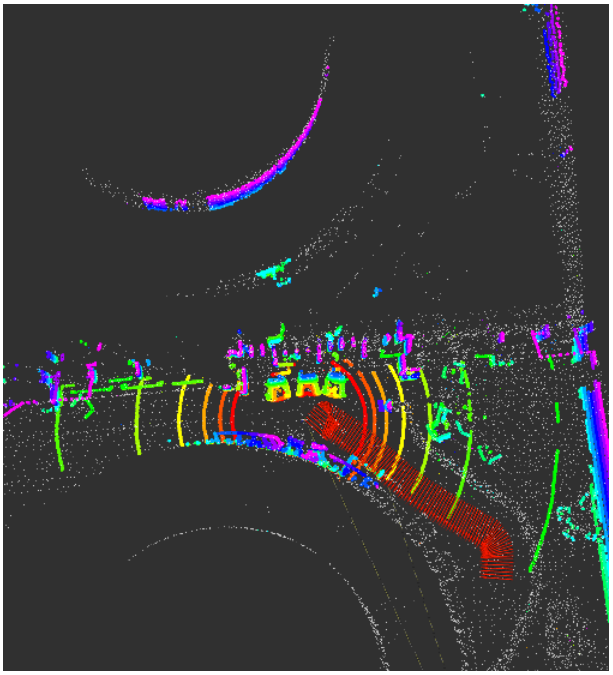

Fig. 1. Visualization of point cloud that obtained from lidar in RViz

IV. Covariance estimation

As a matter of fact, we don't know covariances for motion and observation models. Still, we have their possible parametrizations and could optimize hyperparameters by simple search, minimizing some similarity metric (e.g ATE RMSE) between ground truth and ekf trajectories. However, in our case we could simplify the task and estimate close to true covariances, which we further could use to test other covariances' estimation approaches. With assumption that kinematic motion model and sensor model are both correct we could estimate the motion model error at step t, ϵ_t from: $x_t = g(l_{t-1}, u_t + \epsilon_t)$, where $g()$ - is linearized motion model, x_t is the robot pose estimated from controls and l_{t-1} is ground truth robot pose at the previous step (LiDAR data). So, covariance for motion model can be estimated as:

$$M = \frac{1}{T-1}\sum_{t=1}^{T} \epsilon_t \epsilon_t^T$$

The same goes for observation model. We may estimate observation model noise η_t from $z_t = h(\Delta l + \eta_t)$, where h is linearized observation model, $\Delta l = l_t - l_{t-1}$. Covariance for observation is then defined as:

$$Q = \frac{1}{T-1}\sum_{t=1}^{T} \eta_t \eta_t^T$$

With the same logic we could estimate parameters of covariances when some parametrization is given. This approach has very limited application but still can be useful for our work. In the future more advanced techniques could be tested (like [1], [2])

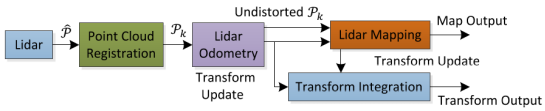

Fig. 2. Block diagram of the lidar odometry and mapping software system.

V. Pose Estimation

A. Ground Truth

We gathered the poses of robot using LOAM algorithm [3]. It's a realtime method for state estimation and mapping using a 3D lidar. We extracted the 2D pose and orientation (heading) of the robot using this method. In Fig. 1 you see the visualization of point cloud that obtained from lidar.

You can find the implementation of the method in this github repository: loam_velodyne . The workflow of the software system is shown in Fig. 2.

Method LOAM considers the case of creating maps with low-drift odometry using a 2-axis lidar moving in 6-DOF. Without the need of high-accuracy range or inertial measurements, the approach achieves both low-drift and low-computational complexity. The key to achieving this level of performance is splitting the generally complex problem of simultaneous localization and mapping (SLAM) [4], which aims to optimize a large number of variables at the same time, into two algorithms. One algorithm uses odometry at a high frequency but low fidelity to estimate the lidar's velocity. Another algorithm runs at a frequency that is an order of magnitude lower for fine matching and registration of the point cloud.

Let $\mathcal{P}_k$ be point cloud perceived during sweep k, $k \in \mathbb{Z}^+$. Consider the following two coordinate systems:

- The lidar coordinate system $\{L\}$ is a three-dimensional coordinate system with its origin at the lidar's geometric center. The x - axis is to the left, the y - axis is to the right, and the z - axis is to the ahead.
- The world coordinate system $\{W\}$ is a three-dimensional coordinate system that coincides with $\{L\}$ at the start.

A diagram of the software system is shown in Fig. 2. Let $\mathcal{P}$ stand for the number of points received during a laser scan. $\mathcal{P}$ is recorded in $\{L\}$ during each sweep. During sweep k, the combined point Laser cloud forms $\mathcal{P}_k$. $\mathcal{P}_k$ is then processed using two algorithms. The point cloud is used to compute the mobility of the lidar between two consecutive sweeps in lidar odometry. $\mathcal{P}_k$ is distorted, and the estimated motion is utilized to fix it. Lidar mapping, which matches and registers the undistorted cloud into a map, is applied to the outputs. Finally, the two algorithms' pose transforms are combined to provide a transform output for the lidar pose.

B. EKF

Based on the kinematic model and sensor model, we can build a Kalman filter [5], [6], [7]. Prediction and update steps for the covariance matrix and state describing the position of the robot are given in the algorithm 1.

Algorithm 1 EKFArc

Input: $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, l, k_l, k_r, \bar{\alpha}, Q$

function PREDICT($u_t, \Delta t$) ▷ $u_t = [v_t, w_t]$

$\theta = \mu_{t-1,\theta}$

if $w_t \neq 0$ **then**:

$$\hat{\mu}_t = \mu_{t-1} + \begin{bmatrix} \frac{-v_t}{w_t}(\sin(\theta) - \sin(\theta + w_t\Delta t)) \\ \frac{v_t}{w_t}(\cos(\theta) - \cos(\theta + w_t\Delta t)) \\ w_t\Delta t \end{bmatrix}$$

else:

$$\hat{\mu}_t = \mu_{t-1} + \begin{bmatrix} v_t\Delta t\cos(\theta) \\ v_t\Delta t\sin(\theta) \\ 0 \end{bmatrix}$$

end if

$$G_t = \begin{bmatrix} 1 & 0 & -\frac{v_t}{w_t}(\cos(\theta) - \cos(\theta + w_t\Delta t)) \\ 0 & 1 & -\frac{v_t}{w_t}(\sin(\theta) - \sin(\theta + w_t\Delta t)) \\ 0 & 0 & 1 \end{bmatrix}$$

$$V_t = \begin{bmatrix} \frac{-\sin(\theta)+\sin(\theta+w_t\Delta t)}{w_t} & \frac{v_t(\sin(\theta)-\sin(\theta+w_t\Delta t))}{w_t^2} + \frac{v_t\cos(\theta+w_t\Delta t)\Delta t}{w_t} \\ \frac{\cos(\theta)-\cos(\theta+w_t\Delta t)}{w_t} & -\frac{v_t(\cos(\theta)-\cos(\theta+w_t\Delta t))}{w_t^2} + \frac{v_t\sin(\theta+w_t\Delta t)\Delta t}{w_t} \\ 0 & \Delta t \end{bmatrix}$$

$$M_t = \begin{bmatrix} \alpha_1 v_t^2 + \alpha_2 w_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 w_t^2 \end{bmatrix}$$

$\hat{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + V_t M_t V_t^T$

return $\hat{\mu}_t, \hat{\Sigma}_t$

end function

function UPDATE($z_t, \Delta t$)

$\Delta x = \hat{x}_t - x_{t-1}$

$\Delta y = \hat{y}_t - y_{t-1}$

$\Delta\theta = \text{wrap\_angle}(\hat{\theta}_t - \theta_{t-1})$

$c = \sqrt{(\Delta x)^2 + (\Delta y)^2}$

$s = \Delta x\cos(\Delta\theta) + \Delta y\sin(\Delta\theta)$ ▷ If scalar product has negative sign, then direction of movement and robot orientation are not aligned. That means that the robot is moving backwards.

if $\Delta\theta \neq 0$ **then**:

$$\hat{z} = \frac{\Delta\theta}{2}\begin{bmatrix} (\frac{c}{\sin(\frac{\Delta\theta}{2})} + l\cdot sgn(s))/k_l \\ (\frac{c}{\sin(\frac{\Delta\theta}{2})} - l\cdot sgn(s))/k_r \end{bmatrix}$$

$$H_t = \begin{bmatrix} \frac{\Delta x\Delta\theta}{2k_l c\sin(\frac{\Delta\theta}{2})} & \frac{\Delta y\Delta\theta}{2k_l c\sin(\frac{\Delta\theta}{2})} & \frac{2c\sin(\frac{\Delta\theta}{2}) - c\Delta\theta\cos(\frac{\Delta\theta}{2})}{4k_l\sin^2(\frac{\Delta\theta}{2})} + \frac{l}{2k_l} \\ \frac{\Delta x\Delta\theta}{2k_r c\sin(\frac{\Delta\theta}{2})} & \frac{\Delta y\Delta\theta}{2k_r c\sin(\frac{\Delta\theta}{2})} & \frac{2c\sin(\frac{\Delta\theta}{2}) - c\Delta\theta\cos(\frac{\Delta\theta}{2})}{4k_r\sin^2(\frac{\Delta\theta}{2})} - \frac{l}{2k_r} \end{bmatrix}$$

else($\Delta\theta \to 0$):

$$\hat{z} = \begin{bmatrix} \frac{c}{k_l} \\ \frac{c}{k_r} \end{bmatrix}$$

$$H_t = \frac{1}{lc}\begin{bmatrix} \Delta x & \Delta y & 0 \\ \Delta x & \Delta y & 0 \end{bmatrix}$$

end if

$\hat{z}* = sgn(s)$

$H_t* = sgn(s)$

$S_t = H_t\bar{\Sigma}_t H_t^T + Q_t$

$K_t = \bar{\Sigma}_t H_t^T S_t^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$

$\Sigma_t = (I_{3\times 3} - K_t H_t)\bar{\Sigma}_t$

return μ_t, Σ_t

end function

VI. RESULTS

We conducted a simple experiment: send robot for a straight line for a while and then stop it. We extracted input control

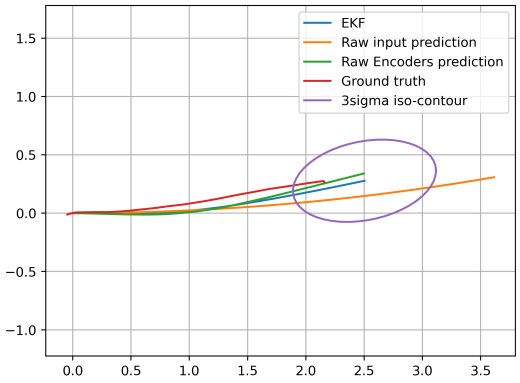

Fig. 3. Extented Kalman Filter

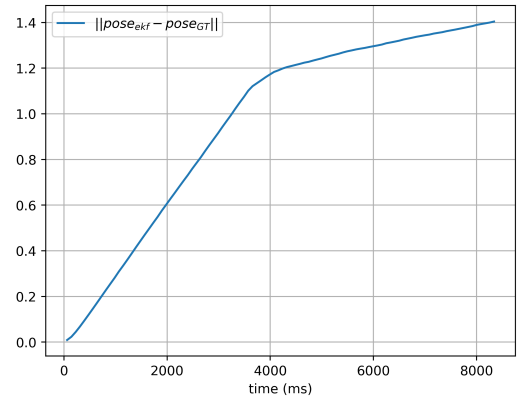

Fig. 4. Error of filter estimation

and encoder data with lidar data for creating ground truth. All data has their timestamps to synchronize them in Algorithm 1.

Recorded data is processed by timestamp order relatively to its start. If next data point is coming from sensor the algorithm performs prediction and update for timestamp difference from last prediction. Otherwise, is next data point is coming from input action the filter make prediction only.

As robot has different wheels movement for two sides when equal input given the robot motion becomes distorted. Fig. 3 shows obtained ground truth trajectory using LOAM, trajectory estimated by EKF and estimations based on raw data. It can be seen that EKF estimation lies between raw data estimation channels.

The L2-mean error is shown on Fig. 4. As wheel-odometry is dead reckoning localization method the error always increasing.

Results on more complicated trajectories can be found on our GitHub repository (https://github.com/GiveMeTheReason/perception22-akula).

References

[1] Vega-Brown, W., Bachrach, A., Bry, A., Kelly, J., Roy, N. (2013, May). CELLO: A fast algorithm for covariance estimation. In 2013 IEEE International Conference on Robotics and Automation (pp. 3160-3167). IEEE.

[2] Mehra, R. (1970). On the identification of variances and adaptive Kalman filtering. IEEE Transactions on automatic control, 15(2), 175-184.

[3] Zhang, Ji Singh, Sanjiv. (2014). LOAM : Lidar Odometry and Mapping in real-time. Robotics: Science and Systems Conference (RSS). 109-111.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.

[5] Thrun, S. (2002). Probabilistic robotics. Communications of the ACM, 45(3), 52-57.

[6] Ribeiro, M. I. (2004). Kalman and extended kalman filters: Concept, derivation and properties. Institute for Systems and Robotics, 43, 46.

[7] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.