System Identification and Simulation

Vladislav Ostankovich

Groupmate: Alexey Koshechkin

**Homework #5**

**Trajectory for groupmate guess**

The trajectory that the robot was given to follow and collect data.

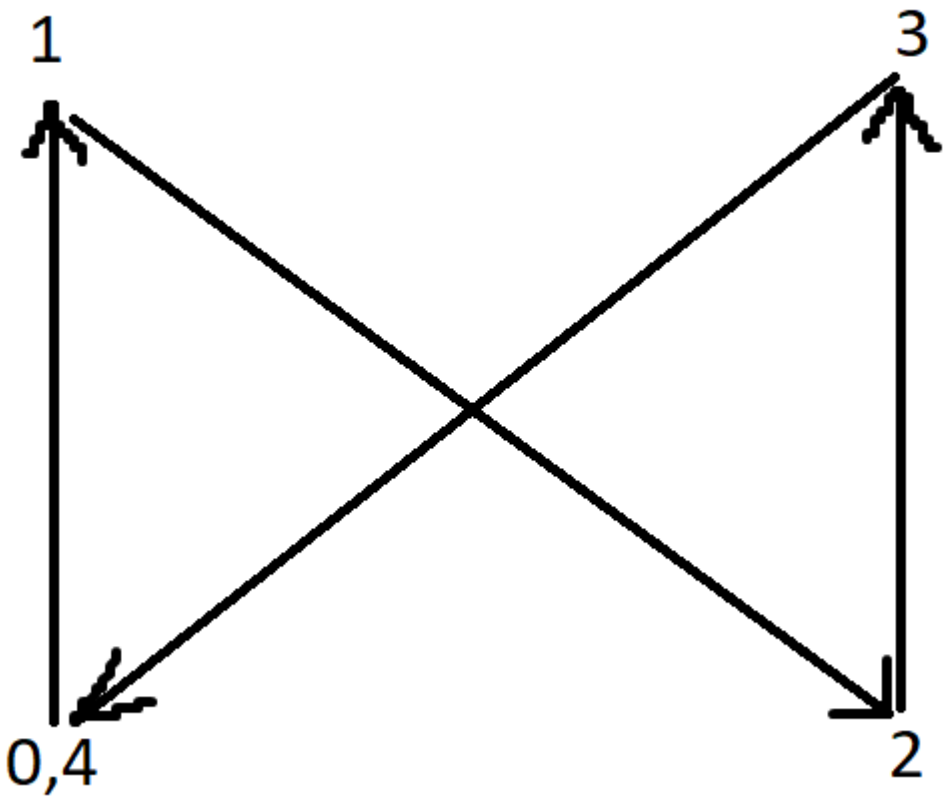


**Figure 1. Trajectory**

**Data**

- Motor encoders for odometry measurement (right and left wheels' rotation in degrees/unit time). The data was captured each 0.05 seconds. Total number of measurements is 360.
- iNEMO inertial module LSM6DS3 [1] of OnePlus3T smartphone was used for the experiment.

**Note!!!**

This data was given to my groupmate initially. For some unknown reasons my groupmate did not generate own trajectory and collect own data. For this reason, I had no choice but to use my own sensors' data and try to guess self-generated trajectory.

**System Setup**

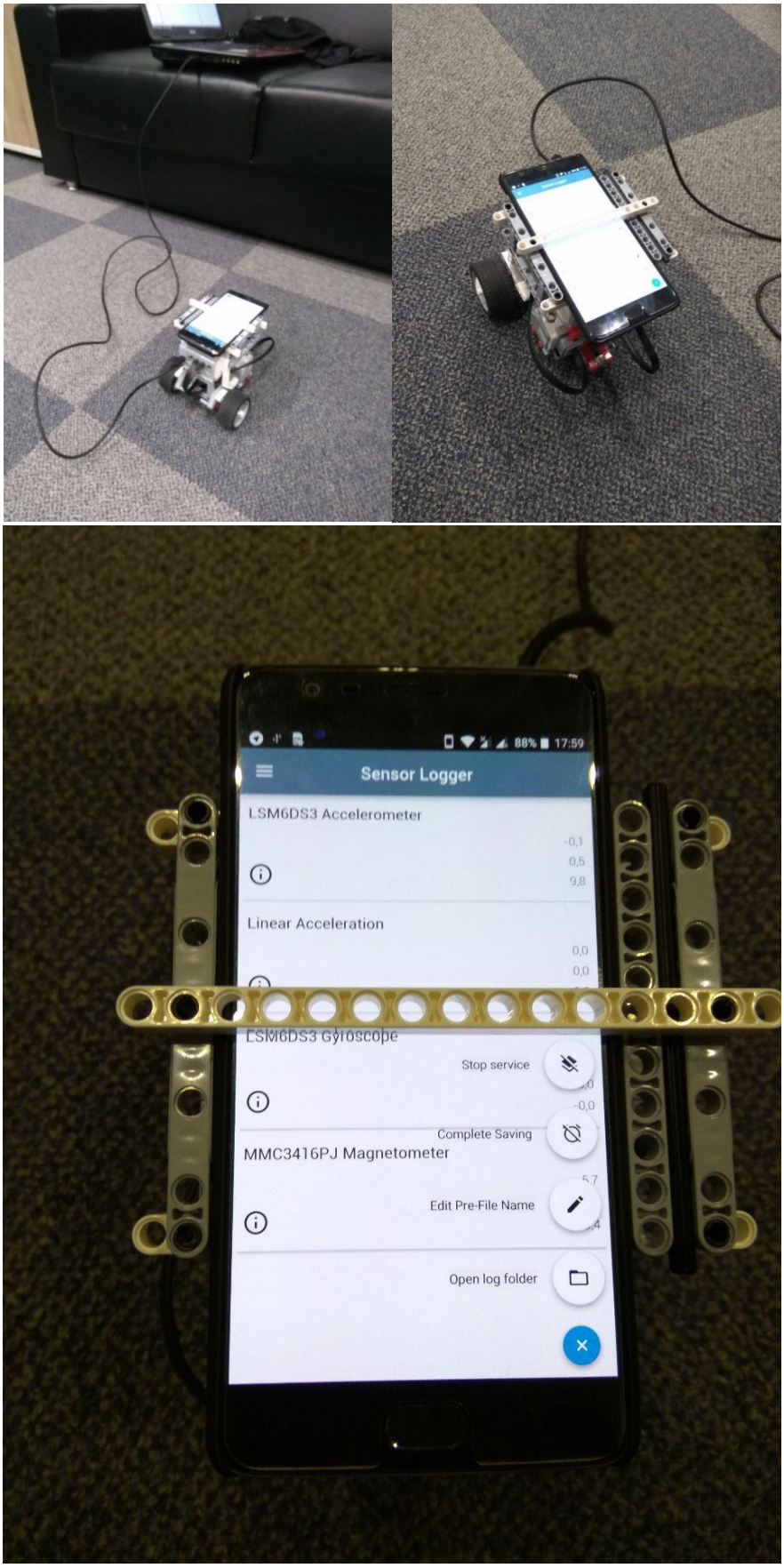


**Figure 2. System setup**

**Trajectory from odometry**

The reconstruction of the current robot configuration is based on the incremental encoder data (odometry). Let $\Delta\varphi R$ and $\Delta\varphi L$ be the no. of wheel rotations measured during the sampling time Ts by the encoders. Linear and angular displacements of the robot is given as

$$\Delta s = \frac{r}{2}(\Delta\varphi_R + \Delta\varphi_L) \ , \ \Delta\theta = \frac{r}{d}(\Delta\varphi_R - \Delta\varphi_L)$$

Where, r = radius of wheels and d = axial distance between wheels.

For a differential-drive robot the position can be estimated starting from a known position by integrating the movement (summing the increment travel distances). The estimate of robot configuration at time tk is computed as:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta s \\ \Delta\theta \end{bmatrix}$$

Robot localization using the above odometric prediction (commonly referred to as dead reckoning) is accurate enough in the absence of wheel slippage and backlash [2]

**Trajectory from gyroscope and accelerometer [3]**

Accelerometer gives us measurements in xyz axes in m/s^2, while gyroscope gives rad/s rotation along the 3 axes. Having acceleration in 3D space, we can double integrate values to get displacement, adding the rotations obtained from gyroscope we can additionally estimate orientation. In practice to detect position over time, using these two sensors is very hard and hardly achievable. It is also needed to remove gravity from the acceleration seen by the IMU. If this is not done perfectly, the errors add up fast.

In reality, measurements taken during time become more and more noisy and incorrect, and the error increases with time, since when we integrate data (acceleration to position) each next measurement becomes more erroneous than previous. The errors could reach very big numbers. In general, I would not trust a MEMS IMU output for too long, since there would always be bias drift. However, for the experiment and for Kalman Filter sensors fusion testing it was decided to continue working with obtained data.

**Kalman Filter**

Formulas taken from [4]

**Extended Kalman Filter**

This approach is hard to apply to collected sensors data, because the EKF is generally designed for dynamic systems. In our context, it is sufficient to say that dynamic systems are driven by some known process in between measurements.

For instance, if we were estimating position and velocity, there is a known relationship (derivative) that describes how the velocity affects position in between measurements from, e.g., a GPS receiver. I have not mentioned any other measurements, like GPS, that we want to use in the filter. If we do not have any other measurements available, then an EKF is not going to give me anything in practice, and is probably much more challenging to set up.

Therefore, EKF is hard to apply for IMU and odometry data, at least we need additional data like GPS (but it is very erroneous to measure GPS for such a small distances like couple of meters, uncertainties are too many). If we have good estimation of velocity/position from other sensors like GPS or any additional ones, we can estimate state transition and measurement function F and H needed for EKF.

Referring to [5] the good choice of sensors to estimate position are:

- GPS, NE position, NED velocity and height
- Altimeter
- Magnetometer
- Range finder, downwards looking aligned with the Z body axis
- Optical flow, downwards looking sensor aligned with the Z body axis
- External vision system, quaternion (currently used for yaw only) and NED position

IMU data is used by the state and covariance prediction steps, it is not used as an observation. IMU data must be converted to delta angles and delta velocities before it can be used by the EKF. It is assumed that any coning and skulling corrections required have already been applied to the data before it is used by the EKF.

**Uncented Kalman Filter**

Meaningless to use it for obtained sensors data for the same reasons as EKF.

**Particle Filter**

Idea taken from [6]

**Results**

Unfortunately, estimation of position from IMU sensor is hardly achievable. Meanwhile odometry from differential drive wheels is enough for guessing trajectory; sensor fusion with odometry, accelerometer and gyroscope using KF does not show much improvement for trajectory prediction. Particle filter also has problems with data from IMU, since it's much erroneous, the particles spread bad. All results are shown in the code as plots. Trajectory guess is mainly from odometry data, and it is clearly visible that it is similar to figure 1.

**References**


[1] http://www.st.com/content/ccc/resource/technical/document/datasheet/a3/f5/4f/ae/8e/44/41/d7/DM00133076.pdf/files/DM00133076.pdf/jcr:content/translations/en.DM00133076.pdf

[2] https://globaljournals.org/GJRE_Volume14/1-Kinematics-Localization-and-Control.pdf

[3] http://www.starlino.com/imu_guide.html

[4] http://home.wlu.edu/~levys/kalman_tutorial/

[5] https://github.com/PX4/ecl/wiki/Extended-Kalman-Filter---Attitude,-Velocity-and-Position-Estimator

[6] https://www.mathworks.com/help/robotics/ug/particle-filter-workflow.html