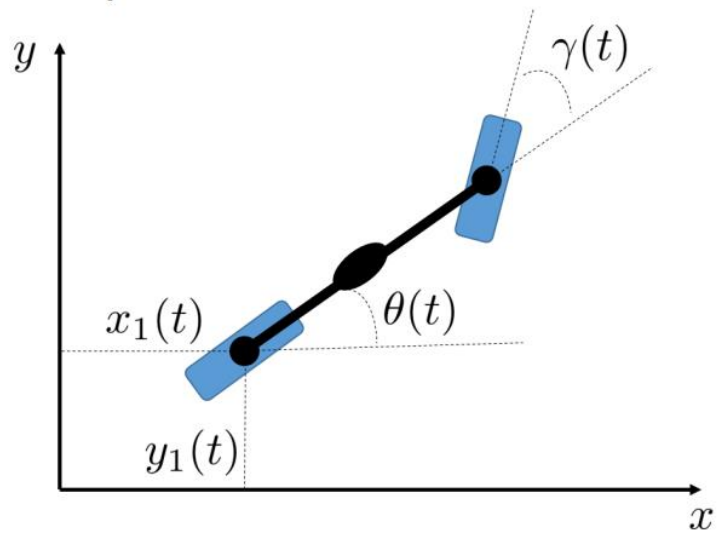
ME231B – Programming Project Report

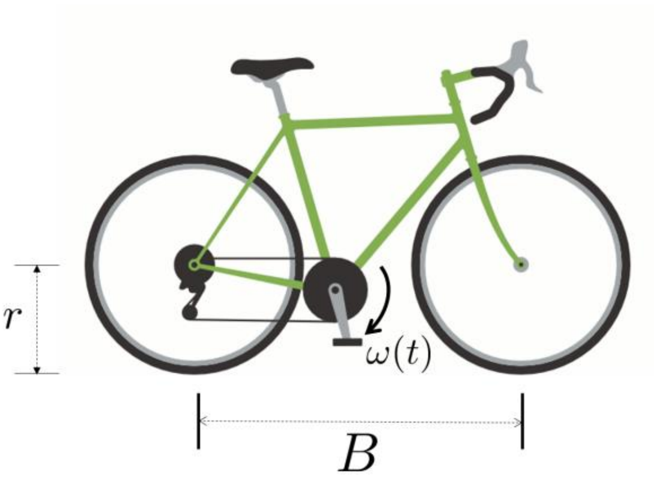
Ning Zhang 3037405708, Pu Wang 3036342321

***Abstract* - In this report, we explore the implementation of some state estimation methods, especially Particle Filter, on a real-world stochastic system. The physical model is abstracted from the system dynamics of a moving bicycle. Considering the trade-off of computational costs and errors, we chose to apply Particle Filter to obtain estimations of the bicycle's Cartesian positions and the heading angle. Using python to run the simulation, we can conclude that our well-designed Particle Filter is able to perform well on estimating the states and parameters that we are interested in.**

I. SYSTEM DYNAMICS

The abstract model of our bicycle is illustrated as below:





where $(x_1(t), y_1(t))$ is the Cartesian position of the rear wheel, $\theta(t)$ is the heading angle, $\gamma(t)$ is the steering angle, $\omega(t)$ is the pedaling speed, r is the wheel radius, B is the length of wheelbase. We assume that the bicycle moves in a flat horizontal plane, where the y-axis represents North, and x-axis represents East.

The system dynamics are described as below equations:

$$\dot{x}_1(t) = v(t)cos\big(\theta(t)\big)$$

$$\dot{y}_1(t) = v(t)sin\big(\theta(t)\big)$$

$$\dot{\theta}(t) = \frac{v(t)}{B}tan\big(\gamma(t)\big)$$

where $v(t)$ is the linear velocity of the bicycle. The bicycle is geared such that the angular velocity of the rear wheel is 5 times the pedaling speed.

We implement Euler method to discretize the continuous system and use angular velocity to substitute linear velocity. Afterwards, the discretized system dynamics are:

$$x_1(k+1) = x_1(k) + 5r\omega(k)\cos(\theta(k)) \cdot T_s$$
$$y_1(k+1) = y_1(k) + 5r\omega(k)\sin(\theta(k)) \cdot T_s$$
$$\theta(k+1) = \theta(k) + \frac{5r}{B}\omega(k)\tan\big(\gamma(k)\big) \cdot T_s$$

The bicycle is furthermore instrumented so that the steering angle $\gamma(t)$ and the pedaling speed $\omega(t)$ are known at all times as piece-wise constants, which means that we can assume them to be unchanged during each time step. However, due to the slip of wheels and unperfect information, there are uncertainties on these two system states. Therefore, we need to add independent zero-mean process noises $v_\omega(k)$ and $v_\gamma(k)$ to them:

$$x_1(k+1) = x_1(k) + 5r(\omega(k) + v_\omega(k))\cos(\theta(k)) \cdot T_s$$
$$y_1(k+1) = y_1(k) + 5r(\omega(k) + v_\omega(k))\sin(\theta(k)) \cdot T_s$$
$$\theta(k+1) = \theta(k) + \frac{5r}{B}(\omega(k) + v_\omega(k))\tan\Big(\gamma(k) + v_\gamma(k)\Big) \cdot T_s$$

Based on engineering intuition and several tests, we tune the process noises as: $v_\omega(k) \sim \mathcal{N}(0, (\pi/90)^2)$, $v_\gamma(k) \sim \mathcal{N}(0, (\pi/36)^2)$.

II. MEASUREMENTS

Position measurements $p(k)$ of the bicycle center (with unit meters) are collected at discrete time t_k. An idealized model of these position measurements is given by

$$p_1(k) = x_1(k) + \frac{1}{2}B\cos(\theta(k))$$

$$p_2(k) = y_1(k) + \frac{1}{2}B\sin(\theta(k))$$

with the measurements available at a period of approximately 0.5 seconds (but not necessarily exact). The measurements are furthermore known to be unbiased. However, the measurements are corrupted by uncertainty, due to electrical noise in the sensor, timing imprecision, and atmospheric disturbances that warp the path of the GPS signals. Thus, we need to set independent zero-mean measurement noises:

$$w_1(k) \sim \mathcal{N}(0, \sigma_{w_1}^2), \quad w_2(k) \sim \mathcal{N}(0, \sigma_{w_2}^2)$$

Taking measurement noises into consideration, we obtain the true measurements as:

$$z_1(k) = x_1(k) + \frac{1}{2}B\cos(\theta(k)) + w_1(k)$$
$$z_2(k) = y_1(k) + \frac{1}{2}B\sin(\theta(k)) + w_2(k)$$

where we can easily see that $\sigma_\omega = \sigma_z$.

A calibration data file is provided, corresponding to position measurements for a stationary bicycle. From the calibration data we can make a rational estimate of variances of position measurements, which are identical to the variances of measurement noises. As a result, we obtain that $\sigma_{w_1}^2 = 1.18389655$, $\sigma_{w_2}^2 = 8.90707547$.

III. DESIGN OF PARTICLE FILTER

1. Initial Conditions:

We know that, for each provided data set, the cyclist starts near the origin (0,0) and is initially heading approximately North-East. We assume initial states of x_1, y_1 and θ are Gaussian random variables. Based on information of the cyclist, we can set the means as: $x_{10} = 0$, $y_{10} = 0$, $\theta_0 = \pi/4$. Based on engineering intuition and several tests, we tune the variances as: $\sigma_x = 0.25$, $\sigma_y = 0.25$, $\sigma_\theta = (\pi/36)^2$.

Also, the wheel radius r and wheelbase length B are not perfectly known due to manufacturing tolerances, flex due to the bicycle rider's weight, wear, varying levels of inflation pressure, and different models of tires. According to the knowledge that we have, we assume them Gaussian random variables as: $r \sim \mathcal{N}(0.425, 0.0005)$, $B \sim \mathcal{N}(0.8, 0.0007)$. These two parameters should also be initialized at $k = 0$, but remain unchanged during the process of simulation.

Considering the trade-off of computational cost and estimation accuracy, we set the number of particles as $N = 1000$. So, we directly draw N initial particles $\{[x_m^n(0) \;\; y_m^n(0) \;\; \theta_m^n(0)]^T\}$ from the related Gaussian distributions.

2. Prior Update:

In the prediction step, we firstly draw N process noise samples $\{[v_\omega^n(k) \;\; v_\gamma^n(k)]^T\}$ from the Gaussian distributions of process noises that we assumed before. Based on the system dynamics, the particles evolve as:

$$x_p^n(k+1) = x_m^n(k) + 5r(\omega(k) + v_\omega^n(k))\cos(\theta_m^n(k)) \cdot T_s$$
$$y_p^n(k+1) = y_m^n(k) + 5r(\omega(k) + v_\omega^n(k))\sin(\theta_m^n(k)) \cdot T_s$$
$$\theta_p^n(k+1) = \theta_m^n(k) + \frac{5r}{B}(\omega(k) + v_\omega^n(k))\tan\left(\gamma(k) + v_\gamma^n(k)\right) \cdot T_s$$

3. Measurement Update:

For a particle filter, we need to scale each particle by the measurement likelihood, which requires that we compute the posterior PDF $f_{z(k+1)|x(k+1)}(\bar{z}(k+1)|x_p^n(k+$

1)), where x is the state vector. In this problem, by changing of variables, we obtain:

$$f_{z(k+1)|x(k+1)}\left(\bar{z}(k+1)\middle|x_p^n(k+1)\right)$$

$$= f_{w_1(k)}(\bar{z_1}(k+1) - x_p^n(k+1) - \frac{1}{2}B\cos(\theta_p^n(k+1)))$$

$$\cdot f_{w_2(k)}(\bar{z_2}(k+1) - y(k+1) - \frac{1}{2}B\sin(\theta_p^n(k+1)))$$

By this equation we evaluate the value of PDF at each $z(k)$, with respect to N prior particles. After this, we compute the weight of each particle as β_n:

$$\beta_n = \alpha f_{z(k+1)|x(k+1)}\left(\bar{z}(k+1)\middle|x_p^n(k+1)\right)$$

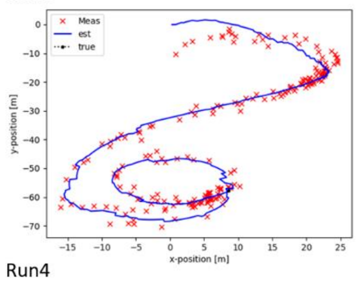
where $\alpha = \sum_{n=1}^{N} f_{z(k+1)|x(k+1)}\left(\bar{z}(k+1)\middle|x_p^n(k+1)\right)$ is a normalization constant.

The next step is resampling the particles so that they will have equal weights. We do this as: 1) Draw a random number $\rho \sim \mathcal{U}(0,1)$; 2) Pick the particle $\bar{n}$ such that $\sum_{n=1}^{\bar{n}} \beta_n \geq \rho$ and $\sum_{n=1}^{\bar{n}} \beta_n < \rho$. Repeating this process for N times gives us a new set of N particles $\{[x_m^n(k+1) \quad y_m^n(k+1) \quad \theta_m^n(k+1)]^T\}$, which will be fed to the next iteration.

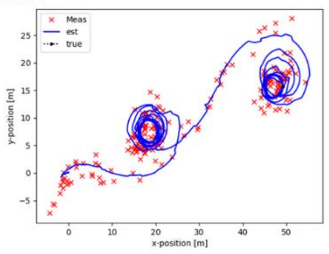
IV. IMPLEMENTATION RESULTS

The estimator described above has been built with Python. We run the simulation with the few data sets (run #1 to run #5), evaluating our estimator by the final state error and program running time. The results show that the estimator's performance is relatively good. Plots of the five runs are shown below:

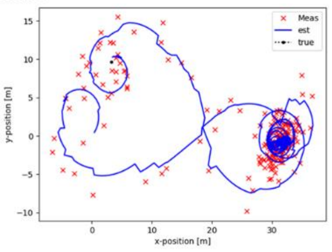
Run1



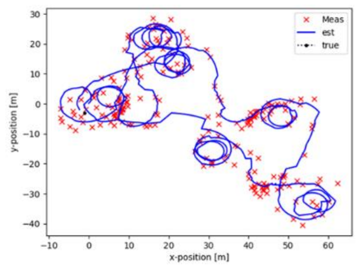
Run2



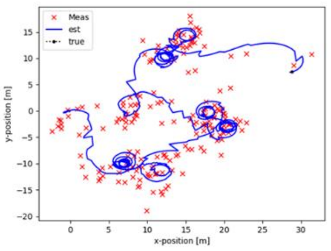
Run3

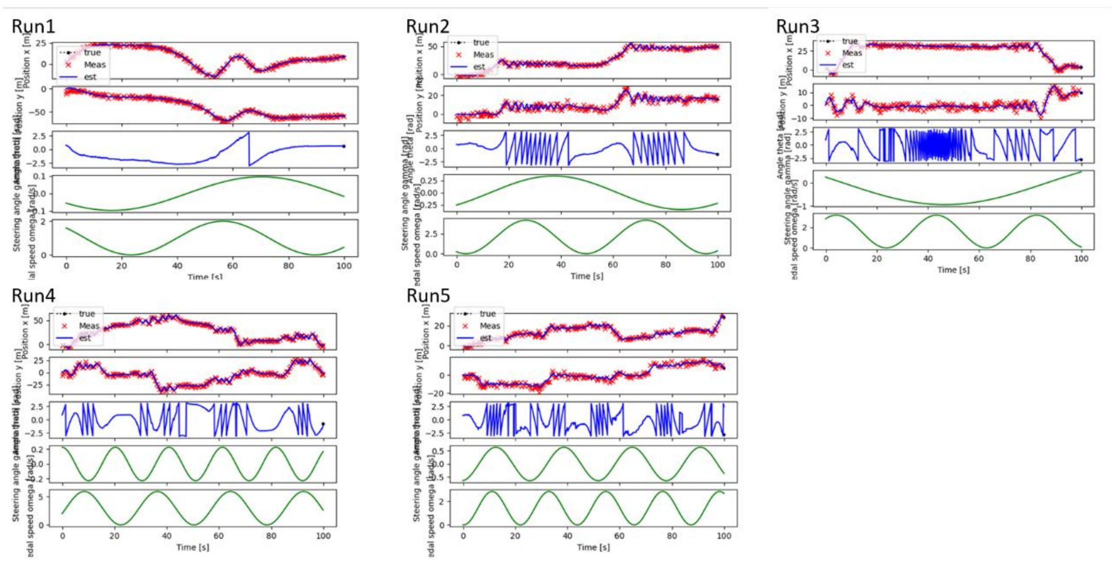


Run4



Run5





The first figure shows the estimated trajectory of bicycle, while the second one shows the evolution of system states during simulations. We can see that measurements do help the Particle Filter correct its estimation of positions. Since we do not have the access to measurements of θ, the estimator did not perform very well on this state.

Listing below are final errors of the instructor's estimator and ours:

Instructor's Estimator				Our Estimator			
Run#	x [m]	y [m]	Θ [rad]	Run#	x [m]	y [m]	Θ [rad]
1	0.246	0.304	0.066	1	-0.125	-0.250	0.038
2	0.110	0.198	0.091	2	0.124	0.226	-0.029
3	-0.139	0.594	0.210	3	0.162	0.640	0.105
4	0.235	-1.580	0.059	4	-1.287	1.176	-0.381
5	-0.147	-1.419	-0.001	5	-0.252	-0.094	-0.993

From the comparison of final errors, we can also conclude that our designed estimator did a great job, even though some errors of some simulations are slightly larger than that of the instructor's.