Contents

0.1 Unicycle Robots Model

The kinematic model each unicycle-type robot can be defined by the following iteration equation:

$$
\begin{aligned}
x_{k+1} &= x_k + v_k cos(\theta_k) \\
y_{k+1} &= y_k + v_k sin(\theta_k) \\
\theta_{k+1} &= \theta_k + \omega_k
\end{aligned} \tag{1}
$$

where $X = [x_k, y_k, \theta_k]^T$ is defined as the states vector, it denotes the global position and orientation vector of the robot at time k and $k \in \{1, 2, ..., N\}$. $u_k = [v_k, \omega_k]^T$ is defined as the control input at time k, v_k and ω_k stands for the linear and angular velocities, respectively.

With the state vector and input vector, the system state space model is defined as follow:

$$
\begin{aligned}
X_{k+1} &= f(X_k, u_k, w_k^1) \\
Z_{k+1} &= h(X_k, w_k^2)
\end{aligned} \tag{2}
$$

where $f(X_k, u_k, w_k^1)$ is the state transition function, and states can be observed through a measurement function $h(X_k, w_k^2)$. Both function could be nonlinear and w_k^1, w_k^2 are the corresponding perturbation on states and measurement respectively.

0.2 Simulation Setup

The simulation is done with Python and Pygame.

The experiment playground size is set as 1080x640, the moving robots are colored as red with triangle, the unique termination points of each robot are colored as green. When the robot is close enough to its endpoint e.g. less then 30 pixels, the endpoint will move to another place, so the robot will keep moving.

Below figure 1 is a screenshot of 10 robots' simulation:

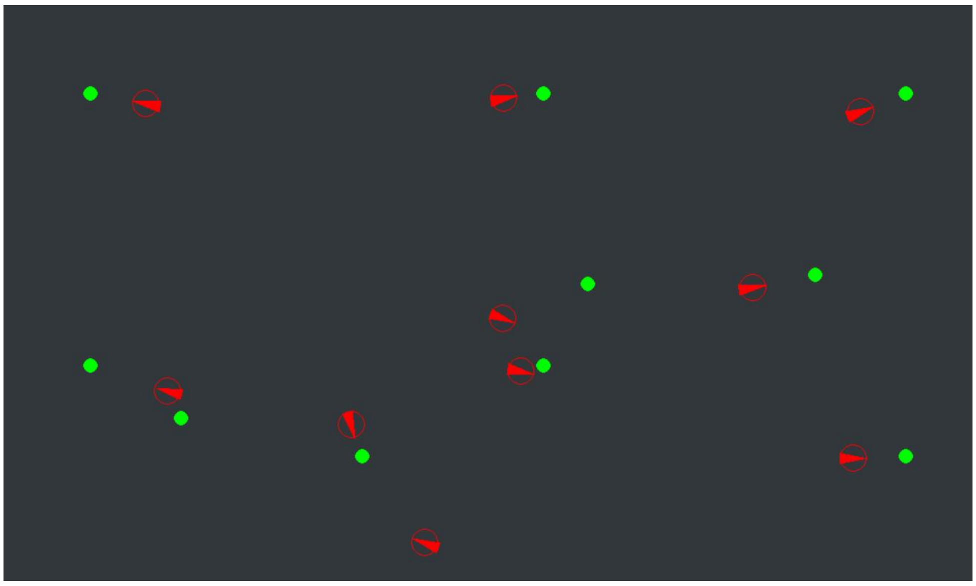

Figure 1: Simulation Overview with 10 Robots

For simplicity, only three moving robots(colored as red) are initialized in the simulation, as shown in the figure 2.

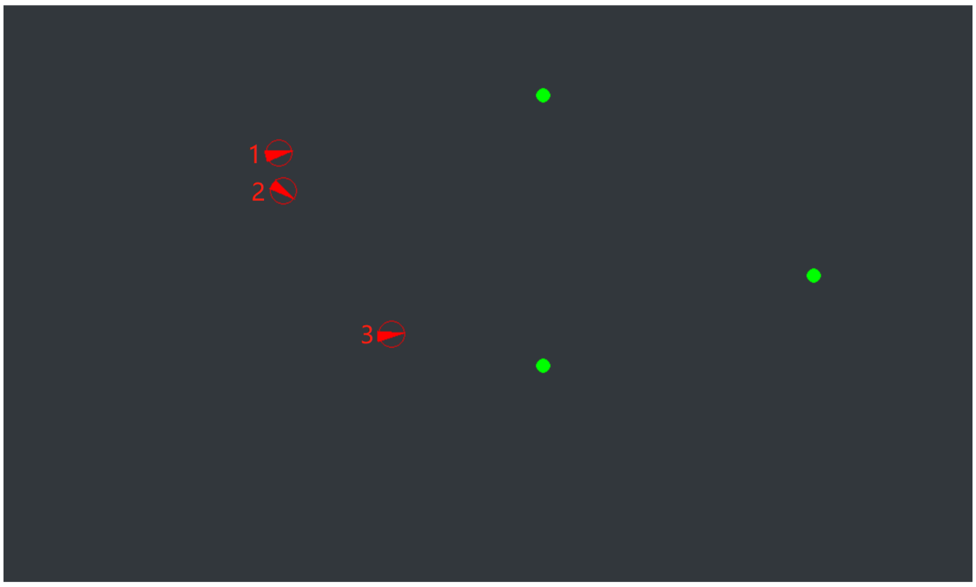

Figure 2: Simulation Overview with 3 Robots

These three robots are initialized in position [100, 100], [100, 200], [100, 400] respectively

The termination point of these three robots are set at [600, 400], [600, 100], [900, 300], respectively.

These robots are performing a simple go-to-goal task with a P controller, they following the equation below:

$$\begin{aligned} e_k &= [X - x_k, Y - y_k] \\ v_k &= \|K_1 e_k\| \\ \phi &= \arctan(X - x_k, Y - y_k) \\ \omega_k &= K_2 \arctan(sin(\phi - \theta_k), cos(\phi - \theta_k)) \end{aligned} \tag{3}$$

where $K = [K_1, K_2] = [-0.001, -0.001, 0.01]$, X and Y are the endpoint of one agent. Since this control task require robots' position information, if the measurements perturbation present, the trajectory will deviate from the case without perturbation.

The control input can be expressed as follows:

$$u_k = [v_k, \omega_k] = f_1(\text{endpoint}, X_k) \tag{4}$$

0.3 State Perturbation

The kinematic model of unicycle robot is ideal, while in reality the robots could encounter different situations leading the real position differs from the output of the model, such as wheel slipping or stuck.

A noise is add to control input to simulate above situations, the biased control input is defined as follows:

$$\begin{aligned} v_k &= v_k + a, a \in [-0.5v_k, 0.5v_k] \\ \omega_k &= \omega_k + b, b \in [-0.5\omega_k, 0.5\omega_k] \end{aligned} \tag{5}$$

0.4 Measurement Perturbation

Robots can record their historical estimated trajectory, they also receive position measurements from the top camera tracking system. We are assuming robots are receiving measurements from the Optitrack system with a certain sampling period T.

The measurement perturbation consists of two part, noise and merge condition.

While the robot are standing still, the position readings from the camera have turbulence, which is described as below:

$$\begin{aligned} x_k &= x_k + 0.2a, a \in [-0.1x_k, 0.1x_k] \\ y_k &= y_k + 0.2b, b \in [-0.1y_k, 0.1y_k] \end{aligned} \tag{6}$$

Since the camera can't tell the robot's direction using single picture, therefore no perturbation on heading angle is assumed here.

The other perturbation is merging, this could happen if the distance between two robots are relatively small. When robots are getting too close to each other, it would be difficult for the camera tracking system to identify each robot, then the tracking system would only give one position(belonging to one of the robot) as the output, since the other position are merged together.

In the simulation, if the distance between moving robot and other agents is small(less than 75 pixels), the tracking system will never give the right measurement(the worst case), it will give other agents' position as a wrong measurement, e.g. in the figure 2, robot No.1 and No.2 are too close for the camera system to identify both of them, as a result, the position measurement of the robot No.1 would become the position of robot No.2 and vice versa.

If there are multiple objects nearby, the simulation would randomly selected one robot from all possible options ans assign its center as the wrong measurement.

0.5 Extended Kalman Filter

In reality, the odometry in one robot could also have drift, therefore an additional information is needed for the correction, in this case, a global camera system is deployed. However, both of sensors are not perfect, their result can not be trusted easily, and adding more additional sensors would cause the system become over-complicated, as a result, the sensor fusion technique is necessary.

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to combine information, which contains noise and other inaccuracies and estimate the state during the transition, [2], [3], [4].

In this case the state transition is a nonlinear function, where the regular Kalman Filter are not suitable to solve. As a result, the extended Kalman Filter known as EKF [1] is used here.

In the simulation, the EKF can be divided into two part, the prediction part and update part, the calculation is listed below.

1. states estimation

   In the prediction part, the state can be estimated using the unicycle kinematic model, the model can be rewritten as follows:

$$X = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \gamma_k \end{bmatrix} + \begin{bmatrix} \cos\gamma_k * dk & 0 \\ \sin\gamma_k * dk & 0 \\ 0 & dk \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} \text{noise}_k \\ \text{noise}_k \\ \text{noise}_k \end{bmatrix} \tag{7}$$

where dk is time step length and set as 1.

2. Predicted covariance of the state estimate

   Predicted covariance of the state estimate is defined as follows:

$$\begin{gathered} \boldsymbol{P}_{k+1|k} = \boldsymbol{F}_{k+1}\boldsymbol{P}_{k|k}\boldsymbol{F}_{k+1}^{\top} + \boldsymbol{Q}_{k+1} \\ \boldsymbol{P}_{k|k} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}, \boldsymbol{F}_{k+1} = \boldsymbol{F}_{k+1}^{\top} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \end{gathered} \tag{8}$$

   In the predicted covariance, $\boldsymbol{P}_{k|k}$ is initialized as some guess value, and the Q matrix contains small value in the diagonal since in the simulation the camera is less trustful than the odometry.

3. Measurement Residual

$$\tilde{\boldsymbol{y}}_{k+1} = \boldsymbol{z}_{k+1} - h\left(\hat{\boldsymbol{x}}_{k+1|k}\right) \tag{9}$$

   The $\boldsymbol{z}_{k+1}$ is the actual measurements received from the camera system.

4. residual Covariance

$$\boldsymbol{S}_{k+1} = \boldsymbol{H}_{k+1}\boldsymbol{P}_{k+1|k}\boldsymbol{H}_{k+1}^{\top} + \boldsymbol{R}_{k+1} \tag{10}$$

   $\boldsymbol{R}_{k+1}$ is the sensor measurement noise covariance matrix, which is assumed as identity matrix.

5. Near-optimal Kalman Gain

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{H}_{k+1}^{\top}\boldsymbol{S}_{k+1}^{-1} \tag{11}$$

6. Updated State Estimate

$$\hat{\boldsymbol{x}}_{k+1|k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{K}_{k+1}\tilde{\boldsymbol{y}}_{k+1} \tag{12}$$

$\hat{\boldsymbol{x}}_{k+1|k+1}$ is the final output of the extended Kalman filter algorithm, this value can be used in the P controller to generate new control input u_{k+1}.

We can express the Kalman filter in following equation:

$$X_{k+1} := \hat{\boldsymbol{x}}_{k+1|k+1} = g(X_k, h(X_k)) \tag{13}$$

0.6 Workflow

Since the robot won't receive the camera measurement at every time instance, the robot will use its odometry information as its position estimation.

After considering equation 2, 4 and 13, the whole system would work iteratively following the equation below, the assumed noise is ignored here:

$$u_k = f_1(\text{endpoint}, X_k)$$
$$X_{k+1} = \begin{cases} f(X_k, u_k), \theta(k) = 0 \\ g(X_k, h(X_k)), \theta(k) = 1 \end{cases} \tag{14}$$

The $\theta(k)$ is a factor indicating whether the robot will receive the camera measurement at time instance k.

0.7 Result

Based on above simulation setup and system workflow14, several experiments was conducted, for simplicity, only the trajectory of No.1 robot (in the figure 2) was plotted.

Odometry only

I first tested the case only odometry were working, the result is shown in the figure 3.

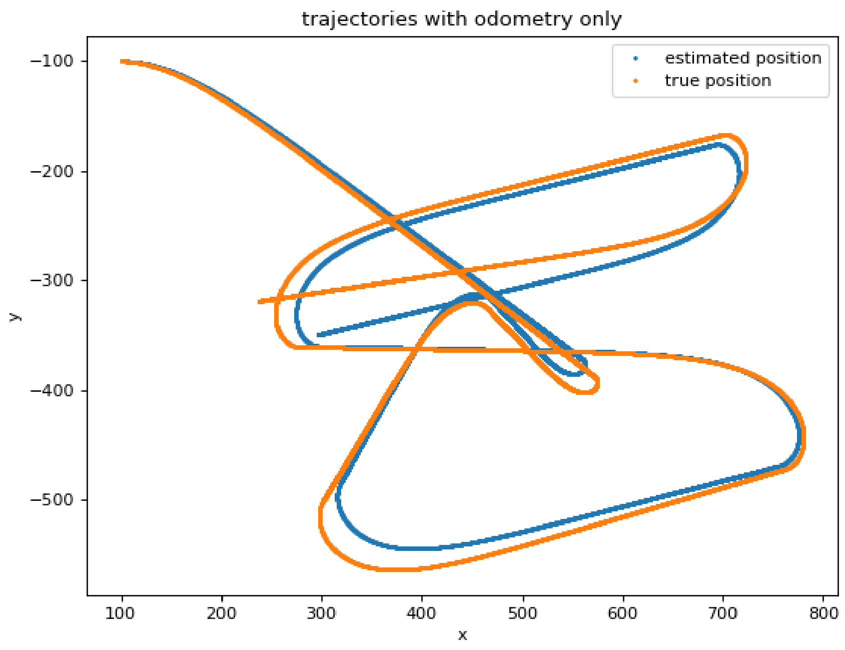

Figure 3: Trajectories with Odometry only

The robot was initialized at the top left of the picture, the result shows that only with the biased odometry, the position estimation has a large different with the true trajectory, therefore, additional sensors are needed.

Using Camera Measurement without Kalman Filter

In all simulation, the sampling period of camera measurement is set as 10 frames. If the robot receive and fully trust the camera measurement, the result is shown in the figure 4.

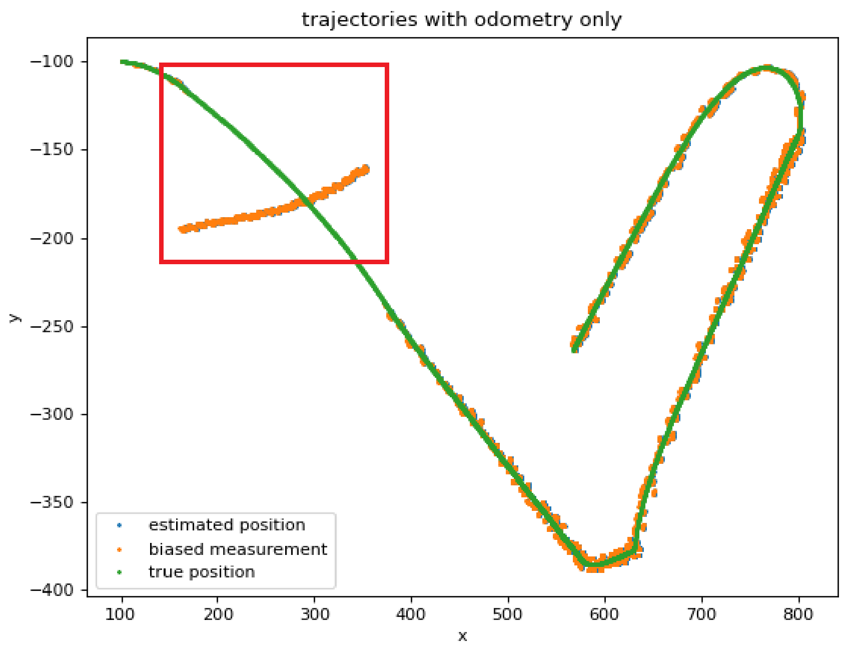

Figure 4: Trajectories Using Camera without Kalman Filter

The orange dots represent the measurement from the camera, and blue dots represent the estimation(almost invincible since robot took measurement as its estimation), and the green dots stand for the true position.

In the red rectangle, the robot No.1 begins at the left top, at the same time, robot No.2 is close to it, as in the figure 2. As we assumed, the camera will give No.2 robot's position as a wrong measurement. In the red rectangle, the estimated position is the same as the measurement(orange dots), it has a large deviation compared to the true position. That indicating with the wrong camera measurement, the robot will lost its position if they fully trust the information from the camera, as a result, a sensor confusion technique is necessary

Using Camera Measurement with Kalman Filter

In this simulation, robots using EKF to correct the biased measurement from both odometry and camera. The parameter Q (equation 8) has a large impact on the result, and the comparison are posted in the following figure.

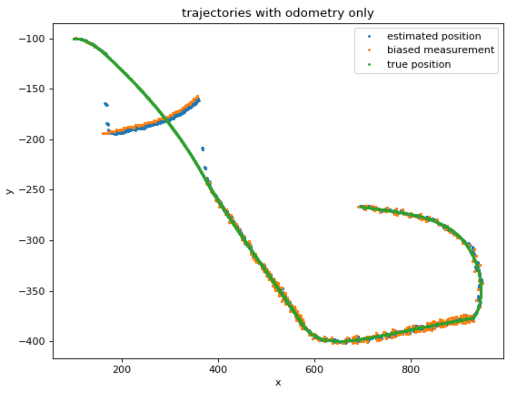

Figure 5: Trajectories Using Camera without Kalman Filter, Q = 1

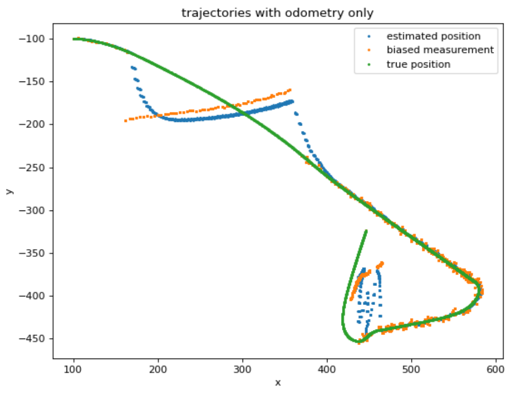

Figure 6: Trajectories Using Camera without Kalman Filter, Q = 0.05

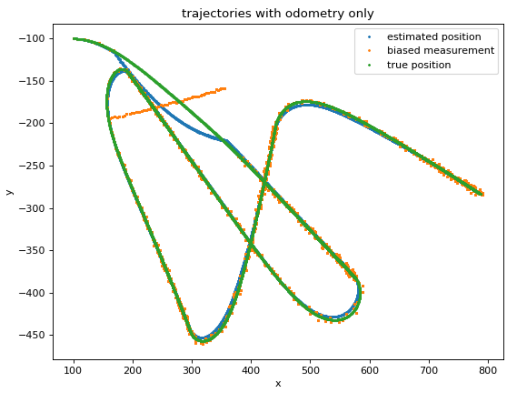

Figure 7: Trajectories Using Camera without Kalman Filter, Q = 0.0005

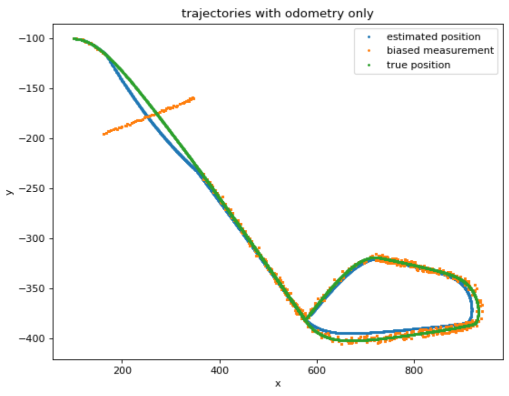

Figure 8: Robot Trajectory with EKF

From the figure 5 and 6, smaller Q will cause the robot trust the camera more than its odometry, so the estimation will have a large difference compared to the true position when the camera information is wrong.

From the figure 7 and 8, the worng measurement has little impact on the estimation, indicating with the help of EKF, the robot can overcome the bias of the camera and have better localization ability.

While extremely small Q will eliminate the effect of the camera, the estimation will behave as the case where only the odometry is working, so there is a drift in some part of the estimation, as shown in the figure 8.

0.8 Summary

Last time:

From the simulation result, the localization ability is improved if the robot takes both odometry and camera into account(using EKF to fuse these two sensors' readings).

I think this architecture could work, but I don't know how much we can trust the odometry in the robot, so some of the parameters should be tuned.

This time:

I put more accurate plots with multiple robots running. I think those pictures can tell the measurement merge situation and how the EKF can eliminate that effect, I think this architecture could work not bad.

I also tested different parameter, choosing the right parameter is kind of searching points in Pareto front, big Q the wrong camera dominates, small Q the biased odometry dominates.

I think with a adjustable Q, this situation can be avoided.

A Reference

[1] Q. Li, R. Li, K. Ji, and W. Dai, “Kalman filter and its application,” in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE, 2015, pp. 74–77.

[2] H. A. Patel and D. G. Thakore, “Moving object tracking using kalman filter,” *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 4, pp. 326–332, 2013.

[3] T. H. Dinh, M. D. Phung, T. H. Tran, and Q. V. Tran, “Localization of a unicycle-like mobile robot using LRF and omni-directional camera,” *CoRR*, vol. abs/1611.09431, 2016. [Online]. Available: http://arxiv.org/abs/1611.09431

[4] J. N. Greenberg and X. Tan, “Dynamic optical localization of a mobile robot using kalman filtering-based position prediction,” *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 5, pp. 2483–2492, 2020.