Contents

0.1 Unicycle Robots Model

The kinematic model of each unicycle-type robot can be defined using the following state space 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 follows:

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

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 functions 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 red with a triangle, and the unique termination points of each robot are colored green. When the robot is close enough to its endpoint e.g. less than 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:

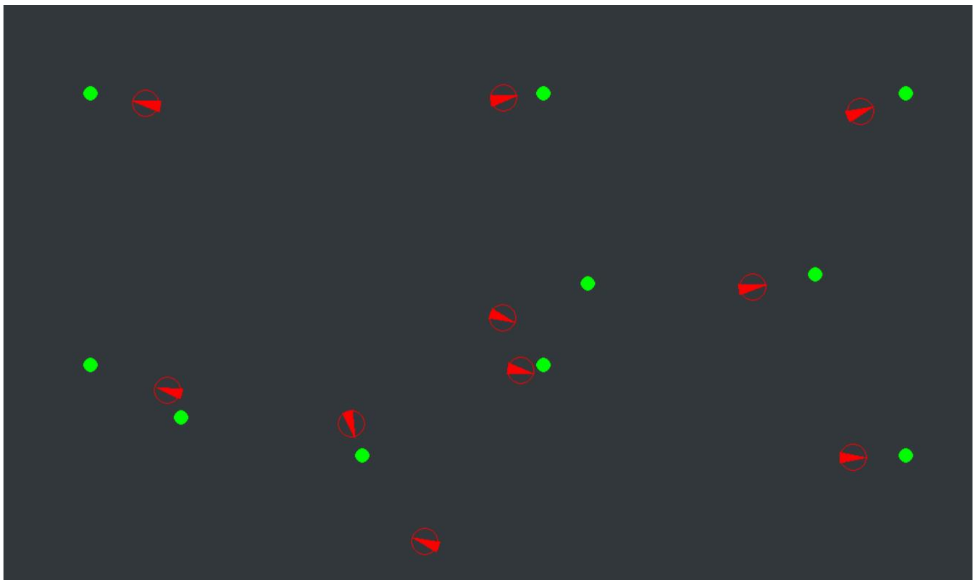
Figure 1: Simulation Overview with 10 Robots

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

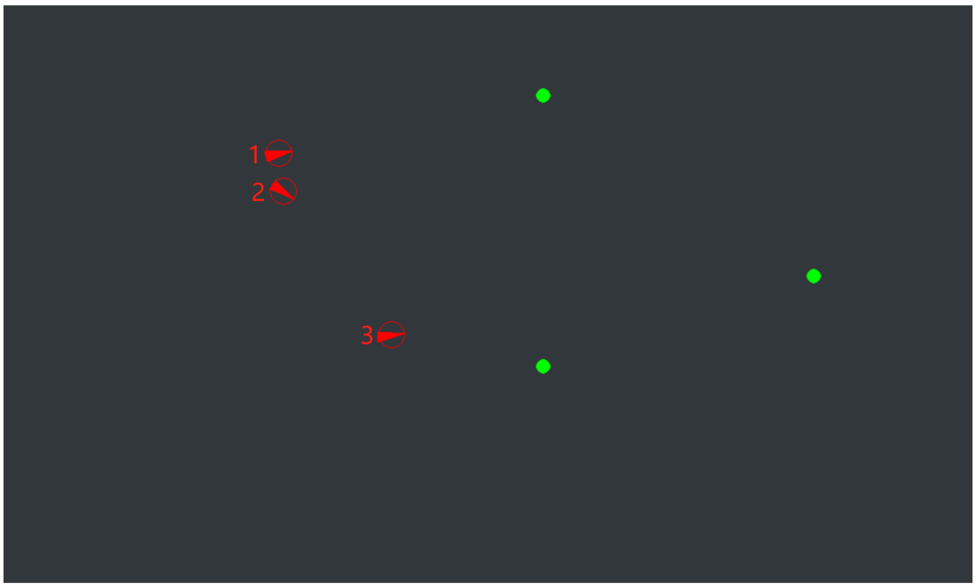

Figure 2: Simulation Overview with 3 Robots

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

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

Their headings were initialized to 0 degrees, pointing to the right edge.

0.3 Robot Controller

These robots need to decide the velocity and heading angle at the next step to reach their termination. Therefore, they are performing a simple go-to-goal task with a P controller.

The controller is following the equation below:

$$\begin{aligned} e_k &= [X - x_k, Y - y_k] \\ v_k &= \| K_P 1 e_k \| \\ \phi &= \arctan(X - x_k, Y - y_k) \\ \omega_k &= K_P 2 \arctan(sin(\phi - \theta_k), cos(\phi - \theta_k)) \end{aligned} \tag{3}$$

where the parameter K_P was chosen as $K_P = [K_P 1, K_P 2] = [-0.001, -0.001, 0.001]$, with that the parameter, the robot will move in a moderate velocity and a smooth turning during the simulation. And X and Y are the endpoints of one agent.

The control input can be expressed as follows:

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

0.4 State Perturbation

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

A uniformly distributed random noise is added to the control input to simulate the above uncertain situations, the biased control input is defined as follows:

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

0.5 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 can happen in two places, noise and merge conditions.

1. Camera Noise

   While the robot is standing still, the position readings from the camera have turbulence due to the noise on the Optitrack system, which is described 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 a single picture, therefore no perturbation on the heading angle is assumed here.

2. Merging

   The other perturbation is merging, this could happen if the distance between two robots is 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 robots) as the output since the other nearby robots' positions are merged together.

   In the simulation, if the distance between the moving robot and other agents is small(less than 75 pixels), the tracking system won't always give the right measurement(the worst case), it will either give a merged position as a wrong measurement.

   For example, 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 and No.2 would be the midpoint of these two robots. Based on the merged position, the camera noise was added.

   If there are multiple objects nearby, the above simulation was also applied, the wrong measurement would be the midpoint of all possible robots.

0.6 Identify the Position from the Camera Outputs

The Optitrack System can track objects and sense their positions, despite the merging condition, however, this system can't link each robot to its own position.

In that case, the camera will give a list of possible positions as the output (note that, the number of records could be less than the number of robots, because of the merging condition). Each robot needs to find out which record is the correct position record, then take the specific records as its measurement.

We are assuming that the robot knows its initial position, then in this case, despite the possible large drift, they will have a relatively accurate and smooth estimation of its global coordination using the unicycle kinematic model, described in the equation 1.

We donate $\hat{X}_k = [\hat{x}_k, \hat{y}_k]$ as the estimation coordination on current time step, $H_k = \{(x_0, y_0)_k, ..., (x_n, y_n)_k\}$ as the reformed camera measurement frame, where n is the total number of robots, and $X_k = [x_k, y_k]$ as the camera measurement at current time step.

Then the measurement X_k can be determined in the following way:

Algorithm 1 Identify position knowing initial position

```
Input: $\hat{X}_k$, $H_k$
Output: $X_k$
  for $(x_i, y_i)_K$ in $H_k$ do
      compute distance between $\hat{X}_k$ and $(x_i, y_i)_k$
      find the minimal distance and corresponding camera measurement $(x_j, y_j)_k$
  end for
  $X_k \leftarrow (x_j, y_j)_k$
```

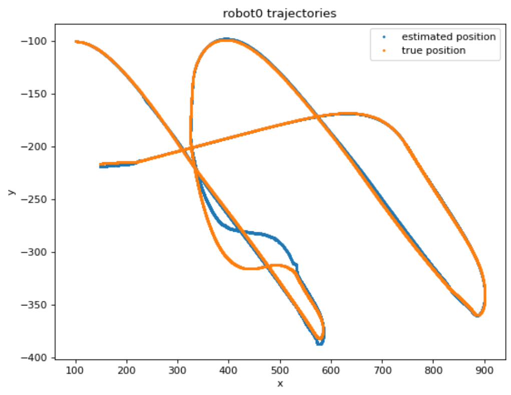

Figure 3: Trajectories of Robot 0

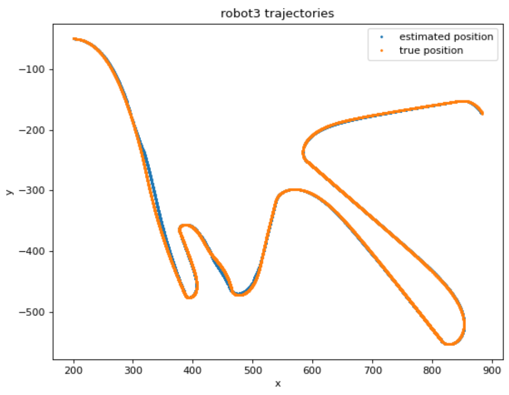

Figure 4: Trajectories of Robot 3

The simulation was done with 10 robots to verify the method above, for simplicity, only two pictures are shown here. From the figure 3 and 4, the results show that the estimation is close to the true position most of the time, especially for robot No.3, there is a deviation happened on robot No.0 due to the merging condition, but this deviation didn't last for a long period.

This indicates that when the robot knows its initial position, the method can help the robot to find the correct measurement among all the possible options given by the camera during the run time.

Moreover, the camera will lose some of the robots' position even if there are no other robots near them, this might happen when the connection quality is low. In this case, all the possible positions given by the Optitrack system are far from the true position. Assigning a wrong position will cause the robot loses its position because the robot will make a wrong estimation based on a wrong measurement, then use the wrong estimation to identify the next measurement.

A good way to avoid that is the following when all the possible positions are far enough, the robot will drop the camera information and localize itself only using the open loop estimation based on the kinetic model described in 1.

0.7 Extended Kalman Filter

In reality, the odometry in one robot could also drift, therefore 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 to 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 is 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 parts, the prediction part, and the 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 the time step length and set as 1.

2. Predicted covariance of the state estimate

   The predicted covariance of the state estimate is defined as follows:

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

   In the predicted covariance, $\boldsymbol{P}_{k|k}$ is initialized as some guess value, and the Q matrix contains a 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 the following equation:

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

0.8 Simulation 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 when the camera information is absent.

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

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

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

Then based on the above iterative equation, the single robot simulation process can be described by the following pseudo-code.

Algorithm 2 Single robots simulation workflow

```
Input: $X_0$
Output: $\hat{X}_k$, $H_k$
  $k \leftarrow 1$
  initialize robots with $X_0$
  while running do
      update $v_k, \omega_k$ based on 4, and perturb the control input
      if $\theta(k) = 1$ then
          receive $H_k$, and identify position $h(X_k)$ from camera
          update state using extend Kalman filter
          $X_{k+1} = g(X_k, h(X_k))$
      else if $\theta(k) = 0$ then
          update state using open loop estimation
          $X_{k+1} = f(X_k, u_k)$
      end if
      if reaches termination then
          set a new termination
      end if
      $k \leftarrow k + 1$
  end while
  log the data
```

0.9 Result

Based on the above simulation setup and system workflow14 and above pseudo code, several experiments with 10 robots were conducted, for simplicity, only the trajectory of some robots (in the figure 1) was shown.

Odometry only

The case only odometry was working was tested first, the result is shown in figure 5.

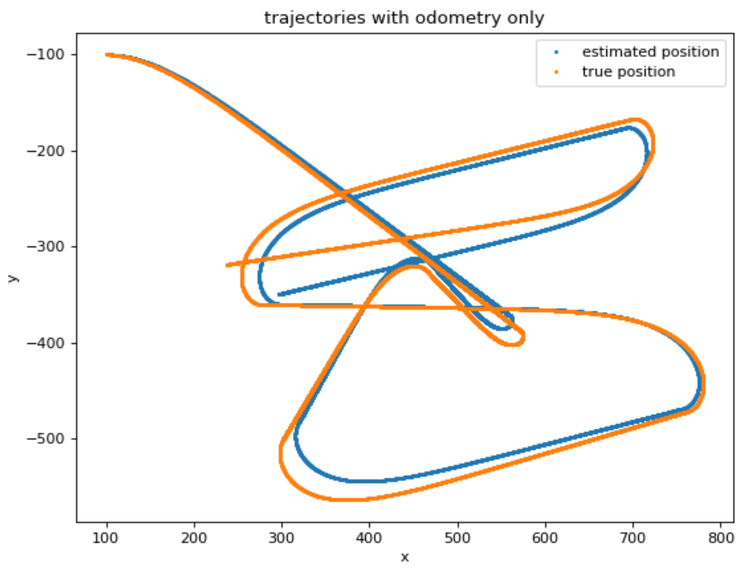

Figure 5: Trajectories with Odometry only

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 difference from the true trajectory. Since the open loop estimation uses an iterative way, the error accumulated over time goes, therefore, additional sensors are needed to correct the biased estimation.

Using Camera Measurement without Kalman Filter

In all simulations, the sampling period of camera measurement is set as 10 frames.

If the robot receives and fully trusts the camera measurement and takes it as the estimation, the experiment result is shown in the figure 6.

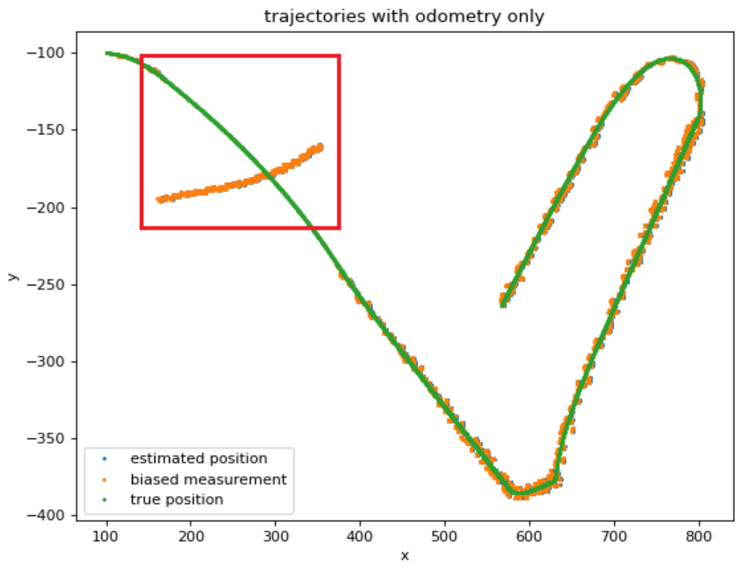

Figure 6: Trajectories Using Camera without Kalman Filter

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

In the red rectangle, 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 the midpoint of these two robots as the 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 lose 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 use EKF to correct the biased measurement from both the odometry and the camera. The parameter Q (equation 8) has a large impact on the result, and the comparison is posted in the following figure.

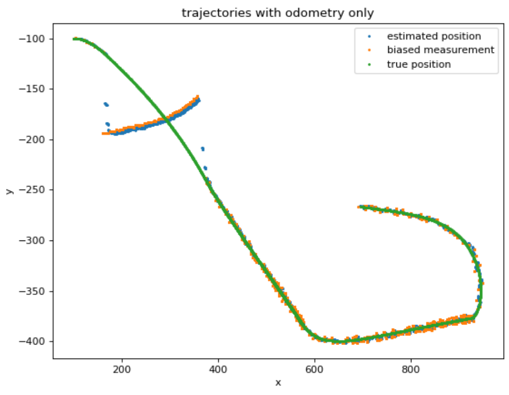

Figure 7: Trajectories Using Camera without Kalman Filter, Q = 1

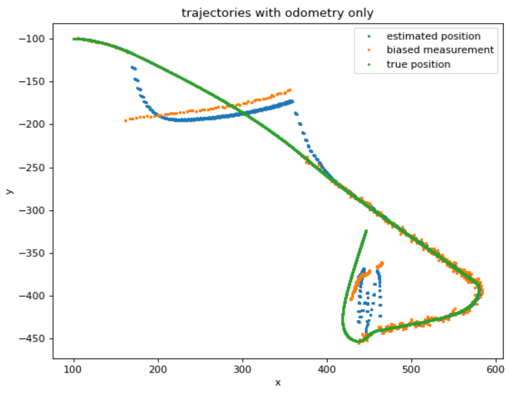

Figure 8: Trajectories Using Camera without Kalman Filter, Q = 0.05

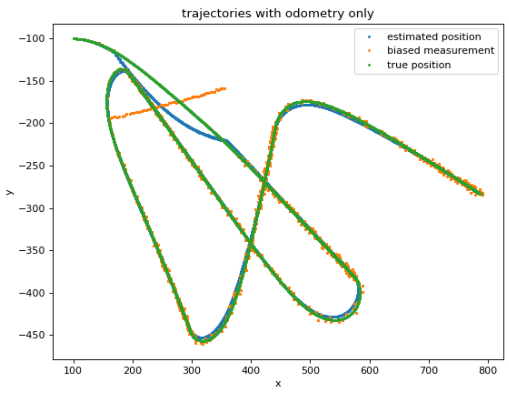

Figure 9: Trajectories Using Camera without Kalman Filter, Q = 0.0005

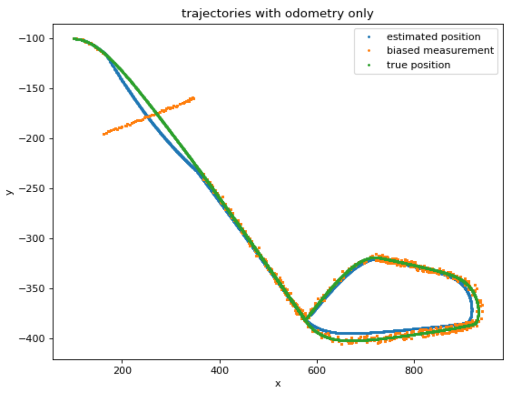

Figure 10: Robot Trajectory with EKF

From the figure 7 and 8, 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 9 and 10, the wrong 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 10.

Summary

I use an indicator to measure how well the above localization architecture can improve localization performance. For each robot, I do the summation of the error between its estimation and true position, the result is shown in the following table.

method	robot 1	robot 2	robot 3	robot 4	robot 5
Odometry	33407	36449	151605	113707	55725
EKF	97710	102536	84901	89062	100149
method	robot 6	robot 7	robot 8	robot 9	robot 10
Odometry	100500	138329	96480	143470	213174
EKF	82998	55320	105632	68949	92236

Table 1: summation

From the above table, using the EKF can sometimes cause the estimation to deviate more compare to the case using odometry only, due to the merging condition, but the average error using EKF is 879497, while the average error using odometry is 1082849, thus indicating that using the above architecture can improve the average localization ability.

0.10 Conclusion

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.

Last 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 badly.

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

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

2022/11/25:

I try to identify the position from the camera with minimal distance, but this method won't work well, and it's quite unstable. This could happen after robots run for a while, at that time, the odometry has a large bias, and if the estimation goes wrong, it will lose position for long.

Then I test the case that the robot knows exactly where they are, it won't lose. Is there a better way to avoid that?

2022/12/1:

I simulate the camera in the wrong way so which caused the "closest assignment" not to function well, then I modify it, and the "closest" works well (not perfectly).

I also add the "smart" way we discussed last time. I used a drop rate to simulate that, one measurement have 5% of losing, and when one finds all measurement are far (> 200), one chooses to use the open loop localization rather than EKF.

I also use an "indicator" to measure how the EKF helps localization, and accumulate the error between each estimation and true position(I don't know if there is a better name for this kind of indicator?), the results show the EKF helps. But due to the random termination, I found some experiments show the opposite result.

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.