Contents

1 System Discription

1.1 Unicycle Robots Model

For the two-dimension localization task, according to [1], the kinematic model of each unicycle-type robot can be defined using the following state space iteration equation:

$$\begin{aligned} x(k+1) &= x(k) + v_x(k)cos(\theta(k)) + w_1(k) \\ y(k+1) &= y(k) + v_y(k)sin(\theta(k)) + w_2(k) \\ \theta(k+1) &= \theta(k) + \omega(k) + w_3(k) \end{aligned} \tag{1}$$

Where $X(k) = [x(k), y(k), \theta(k)]^T$ is defined as the states vector, it describes the global position $x(k), y(k)$ and orientation vector $\theta(k)$ of the robot at time k and $k \in \{1, 2, ..., N\}$. The control input at time k is defined as $u(k) = [v_x(k), v_y(k), \omega(k)]^T$, $v_x(k), v_y(k)$ and $\omega(k)$ stands for the linear and angular velocities, respectively. The noise $w_i(k)$ that happens in state transition is assumed as a zero-mean white Gaussian noise, whose variance is donated as $Q_i(k)$, the detail number of the variance can be tested through multiple experiments.

With the state vector and the control input vector, suppose there are N available sensors, the state space model of the robot can be rewritten as follows:

$$\begin{aligned} X(k+1) &= f(X(k), u(k), w(k)) \\ Z_i(k+1) &= h_i(X(k), v(k)), i = 1, ..., N \end{aligned} \tag{2}$$

where $f(X(k), u(k), w(k))$ is the state transition function, and states can be observed through different sensors, $h_i(X(k), v(k))$ is used as a measurement function for ith sensor. Both functions could be nonlinear and $w(k), v(k)$ are the corresponding perturbation on state transition and measurement respectively, they are both assumed to a zero-mean white Gaussian and are uncorrelated with each other.

1.2 Robot Controller

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

The controller is following the equation below:

$$\begin{aligned} e(k) &= [X - x(k), Y - y(k)] \\ V(k) &= \|K_{P1} e(k)\| \\ \phi(k) &= \arctan(X - x(k), Y - y(k)) \\ \omega(k) &= K_{P2} \arctan(sin(\phi(k) - \theta(k)), cos(\phi(k) - \theta(k))) \end{aligned} \tag{3}$$

where the parameter K_P was chosen as $K_P = [K_{P1}, K_{P2}] = [-0.001, -0.001, 0.001]$, 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:

$$u(k) = [v(k), \omega(k)] = f_1(\text{endpoint}, X(k)) \tag{4}$$

1.3 System Model Block Diagram

With the above expression, we can express the localization system behavior using a block diagram as shown in figure 1. The robot has a goal point, it will determine its velocity and heading and generate input $u(k)$ based on equation 3.

With the $u(k)$, the robot will move one step forward and reaches a new location. The current state of the robot can be measured in three ways, one way is using the system dynamics to estimate. The second way is the measurement provided by odometry inside the robot, the sampling rate of the odometry is donated as T_1. The other alternative is using the Optitrack camera system to capture the robot image and extract the location information, the sampling rate of the camera is T_2.

Since the odometry is embedded with the robot, the sampling rate T_1 can be as high as the rate of system dynamic, e.g. 60 Hz, while the camera information may come in a lower rate, e.g. 4 Hz.

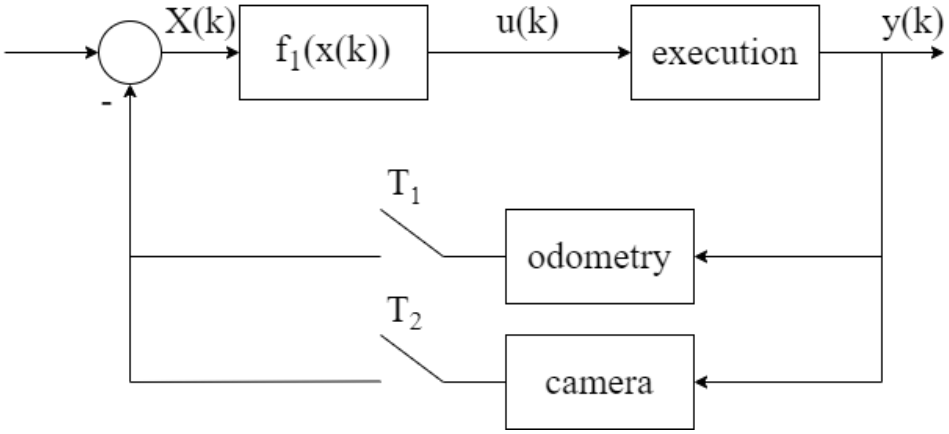

Figure 1: System Block Diagram with 2 Sensors

The alternative is that we assume the system dynamics is the same as the odometry, then we only have one measurement from the camera, and its sampling rate is T_1. Due to the fact that the camera measurement is accurate when the measurement is not "merged" or missed, only using camera information and some fusing technique to correct the odometry date and then generate a state estimation at a lower rate is less computationally complex comparing to above case, where a fusing technique is performed every step.

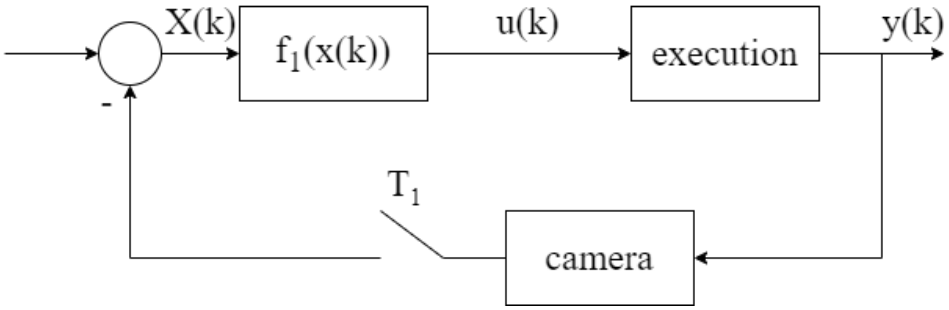

Figure 2: System Block Diagram with 1 Sensor

2 Sensor Data Fusing

We can express the Kalman filter estimation in the following equation:

$$X(k+1) := \hat{\boldsymbol{x}}(k+1 \mid k+1) = g(X(k), h(X(k))) \tag{5}$$

2.1 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 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, [3], [4], [5].

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 [2] is used here.

In the simulation, the EKF can be divided into two parts, the prediction part, and the update part, based on [6], the calculation of extended Kalman Filter is listed in the algorithm 1.

Algorithm 1 Single rate extended Kalman filter

Input: $\hat{\boldsymbol{x}}(k+1 \mid k)$, $\boldsymbol{z}(k+1)$
Output: $\hat{\boldsymbol{x}}(k+1 \mid k+1)$

predicted state value

$$\hat{\boldsymbol{x}}(k+1 \mid k) = f(X(k), u(k), w(k)) \tag{6}$$

predicted covariance of the state estimation

$$\boldsymbol{P}(k+1 \mid k) = \boldsymbol{F}(k+1)\boldsymbol{P}(k \mid k)\boldsymbol{F}(k+1)^{\top} + \boldsymbol{Q}(k+1) \tag{7}$$

calculate the measurement residual between estimation and measurement $\boldsymbol{z}(k+1)$

$$\tilde{\boldsymbol{y}}(k+1) = \boldsymbol{z}(k+1) - h\left(\hat{\boldsymbol{x}}(k+1 \mid k)\right) \tag{8}$$

computes the residual covariance

$$\boldsymbol{S}(k+1) = \boldsymbol{H}(k+1)\boldsymbol{P}(k+1 \mid k)\boldsymbol{H}(k+1)^{\top} + \boldsymbol{R}(k+1) \tag{9}$$

calculates the near-optimal Kalman gain

$$\boldsymbol{K}(k+1) = \boldsymbol{P}(k+1 \mid k)\boldsymbol{H}(k+1)^{\top}\boldsymbol{S}(k+1)^{-1} \tag{10}$$

updates the state estimation for the next step using the current estimation and measurements.

$$\hat{\boldsymbol{x}}(k+1 \mid k+1) = \hat{\boldsymbol{x}}(k+1 \mid k) + \boldsymbol{K}(k+1)\tilde{\boldsymbol{y}}(k+1) \tag{11}$$

In this case, the state transition function12 can be further extended as, where dk is the time step length and set as 1.

$$X(k+1) = \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ y(k) \\ \gamma(k) \end{bmatrix} + \begin{bmatrix} \cos\gamma(k) * dk & 0 \\ \sin\gamma(k) * dk & 0 \\ 0 & dk \end{bmatrix} \begin{bmatrix} v(k) \\ \omega(k) \end{bmatrix} + \begin{bmatrix} w_1(k) \\ w_2(k) \\ w_3(k) \end{bmatrix} \tag{12}$$

In the covariance prediction7, matrix are initialized as follows:

$$
\begin{aligned}
\boldsymbol{P}(k \mid k) &= \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \qquad , \boldsymbol{F}(k+1) = \boldsymbol{F}(k+1)^{\top} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\boldsymbol{R}(k+1) &= \begin{bmatrix} 0.005 & 0 & 0 \\ 0 & 0.005 & 0 \\ 0 & 0 & 0.0 \end{bmatrix} \quad , Q(K+1) = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}
\end{aligned} \tag{13}
$$

In the predicted covariance, the $\boldsymbol{P}(k \mid k)$ represents the accuracy of the state estimation, this matrix has variances on the diagonal, and covariance on the off-diagonal position, is initialized as some guess value. $\boldsymbol{F}(k+1)$ and its transpose $\boldsymbol{F}(k+1)^{\top}$ are both equivalent to A.

The $\boldsymbol{R}(k+1)$ matrix is the sensor measurement noise covariance matrix, as we assumed above, the measurement noise is a zero mean white noise, and is uncorrelated to each other.

The Q matrix is the action uncertainty matrix, which is equivalence with the state transition noise in the state space model 2. Intuitively, a smaller Q value in the diagonal will cause the Kalman filter "trusts" state estimation, the odometry more than the camera measurement.

In the 11, the $\hat{\boldsymbol{x}}(k+1 \mid k+1)$ is the final output of the extended Kalman filter algorithm, this value can be used in the P controller to generate new control input $u(k+1)$.

2.2 Multi-rate Kalman Filter Method

Multi-sensor fusion is a powerful technique, which elicits significant information from multiple sensors to acquire an optimal or sub-optimal state estimation [7], [8], [9], [10] .

To express a model for sensor i with the lower sampling rate, the state space model of the unicycle robot is extended to the steps of M augmentation in time step k as follows, where the M is the sampling period of sensor i.

$$
\begin{aligned}
X(k+2) &= A(X(k+1)) + Bu(k+1) + w(k+1) \\
&= A(A(X(k)) + Bu(k) + w(k)) + Bu(k+1) + w(k+1) \\
&= A^2(X(k)) + \begin{bmatrix} AB & B \end{bmatrix} \begin{bmatrix} u(k) \\ u(k+1) \end{bmatrix} + \begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} w(k) \\ w(k+1) \end{bmatrix}
\end{aligned} \tag{14}
$$

Proceed above expression in an iterative way [8]:

$$
A^M(X(k)) + B_M B u_M(k) + B_M w_M(k) \tag{15}
$$

Where the B_M is the parameter matrix, $w_M(k)$ is called the noise matrix in one data block in the figure.

$$
\begin{aligned}
B_M &= \begin{bmatrix} A^{M-1} & \dots & I \end{bmatrix} \\
w_M(k) &= \begin{bmatrix} w(k) & \dots & w(k+M) \end{bmatrix}^T
\end{aligned} \tag{16}
$$

Then the noise in the state transition is assumed as a zero mean white noise, as a result, the covariance matrix of the block noise $w_M(k)$ is driven as follows, the $\delta(k+m)$ is a delta function which will be set as 1 at sampling time $k+M$:

$$
\begin{aligned}
E\left\{w_M(k)w_M^T(m)\right\} &= Q_M(k)\delta(k+m) \\
Q_M(k) &= \begin{bmatrix} Q(k+1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q(k+M) \end{bmatrix}
\end{aligned} \tag{17}
$$

Then with the above equation, the multi-rate Kalman filtering algorithm of a lower sampling rate sensor i can be driven in the 2:

Algorithm 2 Multi-rate extended Kalman filter

Input: $\hat{\boldsymbol{x}}(k+1 \mid k)$, $\boldsymbol{z}(k+1)$
Output: $\hat{\boldsymbol{x}}(k+1 \mid k+1)$

predicted state value

$$
\hat{\boldsymbol{x}}(k+M \mid k) = f(X(k), u(k), w(k)) \tag{18}
$$

predicted covariance of the state estimation

$$
\boldsymbol{P}(k+M \mid k) = \boldsymbol{F}(k+M)^M \boldsymbol{P}(k \mid k) \boldsymbol{F}(k+M)^{M^\top} + B_M \boldsymbol{Q}_M(k+1) B_M^T \tag{19}
$$

calculate the measurement residual between estimation and measurement $\boldsymbol{z}(k+1)$

$$
\tilde{\boldsymbol{y}}(k+1) = \boldsymbol{z}(k+1) - h\left(\hat{\boldsymbol{x}}(k+1 \mid k)\right) \tag{20}
$$

computes the residual covariance

$$
\boldsymbol{S}(k+1) = \boldsymbol{H}(k+1) \boldsymbol{P}(k+1 \mid k) \boldsymbol{H}(k+1)^\top + \boldsymbol{R}(k+1) \tag{21}
$$

calculates the near-optimal Kalman gain

$$
\boldsymbol{K}(k+1) = \boldsymbol{P}(k+1 \mid k) \boldsymbol{H}(k+1)^\top \boldsymbol{S}(k+1)^{-1} \tag{22}
$$

updates the state estimation for the next step using the current estimation and measurements.

$$
\hat{\boldsymbol{x}}(k+1 \mid k+1) = \hat{\boldsymbol{x}}(k+1 \mid k) + \boldsymbol{K}(k+1) \tilde{\boldsymbol{y}}(k+1) \tag{23}
$$

Further, the ith sensor may have a missing measurement with a certain possibility at time k, using a stochastic process $\gamma_i(k) \in \{0, 1\}$. When $\gamma_i(k) = 0$, it indicates that the sensor i has no measurement, in this case, only the estimation is trustful the output of the multi-rate Kalman filter will be shown as follows:

$$
\begin{aligned}
\hat{\boldsymbol{x}}(k+M \mid k+M) &= \hat{\boldsymbol{x}}(k+M \mid k) = f(X(k), u(k), w(k)) \\
\boldsymbol{P}(k+M \mid k+M) &= \boldsymbol{P}(k+M \mid k) = \boldsymbol{F}(k+M)^M \boldsymbol{P}(k \mid k) \boldsymbol{F}(k+M)^{M^\top} + \boldsymbol{Q}_M(k+1)
\end{aligned} \tag{24}
$$

2.3 State Estimation Fusing Method

Based on the above discussion, we have several methods to fuse the multiple sensors' data and construct the localization architecture.

Single-rate Extended Kalman Filter Method

In this case, only two EKF are applied. The odometry and system dynamic both have the same sampling rate, so the initial estimation is fused with odometry data and generates an intermediate estimation. If the camera information is available at this time step, then the final estimation is produced by fusing the camera information and intermediate estimation. The workflow is shown in figure 3.

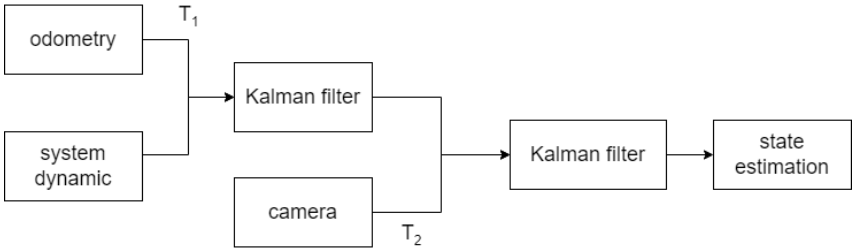

Figure 3: Single-rate EKF Localization Workflow

Multi-rate Extended Kalman Filter Method

Multi-rate Extended Kalman filter method is similar to the previous case, the only difference is the fusing between camera information and intermediate estimation: when the camera information is ready, an mr-EKF is applied. The workflow is shown in figure 4.

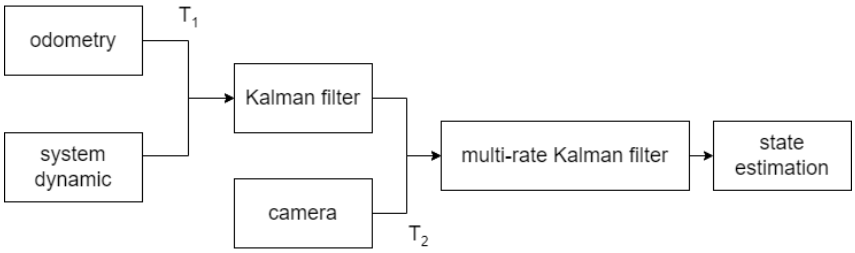

Figure 4: Multi-rate EKF Localization Workflow

Multi-rate Extended Kalman Filter Method with OWA

In the [8] [11], outputs from different sensors are integrated together in an OWA fashion, in paper [8], the author elaborates on the benefit of using an OWA integration rather than an mr-EKF. The workflow is shown in figure 5.

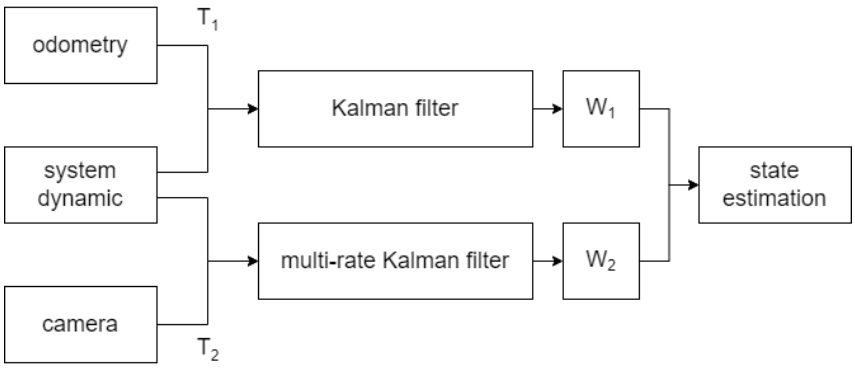

Figure 5: Multi-rate EKF with OWA Localization Workflow

The fused estimation can be expressed in the following way:

$$\hat{\boldsymbol{x}}(k \mid k) = \sum_{i=1}^{N} W_{i,k} \hat{\boldsymbol{x}}(k \mid k) \tag{25}$$

The weight for ith sensors at time k $W_{i,k}$ is determined in 26.

$$\begin{aligned} W_{i,k} &= \hat{C}_i^{-1}(k \mid k)(\sum_{j=1}^{N} \hat{C}_j^{-1}(k \mid k))^{-1} \\ \hat{C}_i^{-1}(k) &= \frac{1}{l_w} \sum_{i=1}^{l_w} r_i(k+1-i) r_i(k+1-i)^T \\ r_i(k) &= y_i(k) - H_i(\hat{(}x(k \mid k)) \end{aligned} \tag{26}$$

The $r_i(k)$ is the innovation matrix for sensor i, and the $\hat{C}_i(k)$ implies the real state estimation covariance during the mr-EKF fusing, the l_w is the length of sliding windows, which could be the sampling period of sensor i.

3 Simulation

3.1 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 6 is a screenshot of 10 robots' simulation:

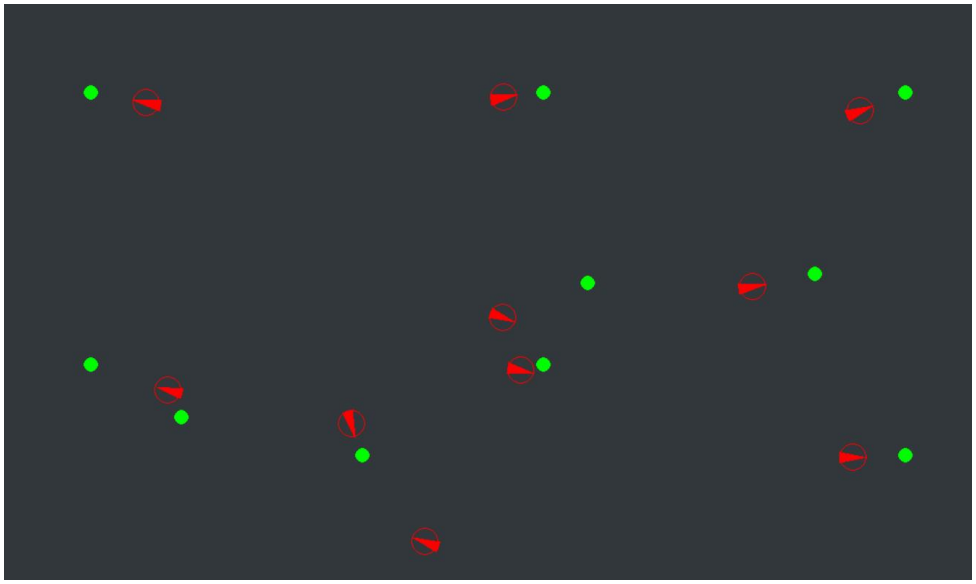

Figure 6: Simulation Overview with 10 Robots

For simplicity, only three moving robots(colored red) are initialized in the simulation, as shown in the figure 7.

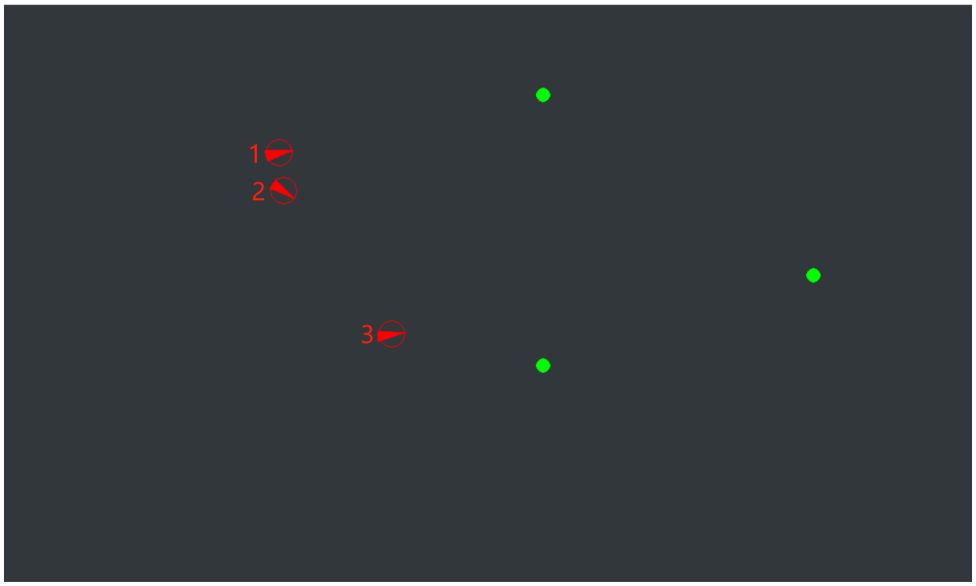

Figure 7: Simulation Overview with 3 Robots

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.

3.2 Perturbation

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.

This perturbation is represented by a set of zero mean Gaussian white noise $w(k)$ in the 2, their covariance was assumed as 0.01, 0.01 and 0.0001.

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, as stated in the state space model, the measurement noise is $v(k)$, their covariance was assumed as 0.005, 0.005 and 0.

   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 7, 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.

3.3 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 $H(k) = \{(x_0, y_0)_k, ..., (x_i, y_i)_k\}, i \in [0, n]$ as the camera output, where n is the total number of robots, when merging happens, the number of output is less than the number of robots. $\hat{X}(k) = [\hat{x}(k), \hat{y}(k)]$ as the estimation coordination on current time step, and $X(k) = [x(k), y(k)]$ as the assigned camera measurement at current time step.

Then the measurement $X(k)$ can be determined in the following way:

Algorithm 3 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
  $X(k) \leftarrow (x_j, y_j)_k$

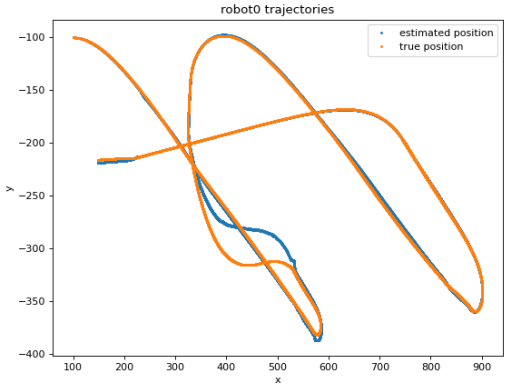

Figure 8: Trajectories of Robot 0

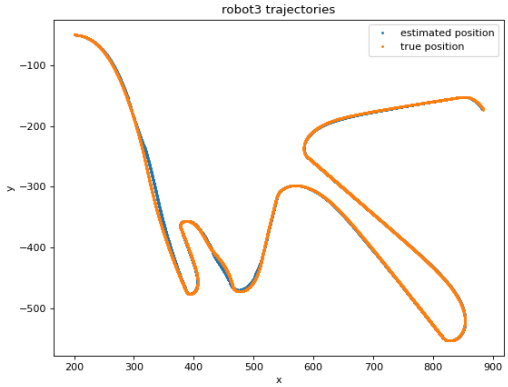

Figure 9: Trajectories of Robot 3

The simulation was done with 10 robots to verify the method above, for simplicity, only two pictures are shown here. From the figure 8 and 9, 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 the equation 1.

3.4 Simulation Workflow

Based on the description of above section, there are three types of localization architecture, the following simulation is done based on the single-rate EKF method, as shown in the figure 3.

After considering equation 2, 4 and 5, the whole system would work iteratively following the equation below, the assumed noise is ignored here:

$$\begin{aligned} u(k) &= f_1(\text{endpoint}, X(k)) \\ X(k+1) &= \begin{cases} f(X(k), u(k)), \theta(k) = 0 \\ g(X(k), h(X(k))), \theta(k) = 1 \end{cases} \end{aligned} \tag{27}$$

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 4 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

3.5 Result

Based on the above simulation setup and system workflow27 and above pseudo code, several experiments with 10 robots were conducted, for simplicity, only the trajectory of some robots (in the figure 6) was shown.

Odometry only

The case only odometry was working was tested first, the result is shown in figure 10.

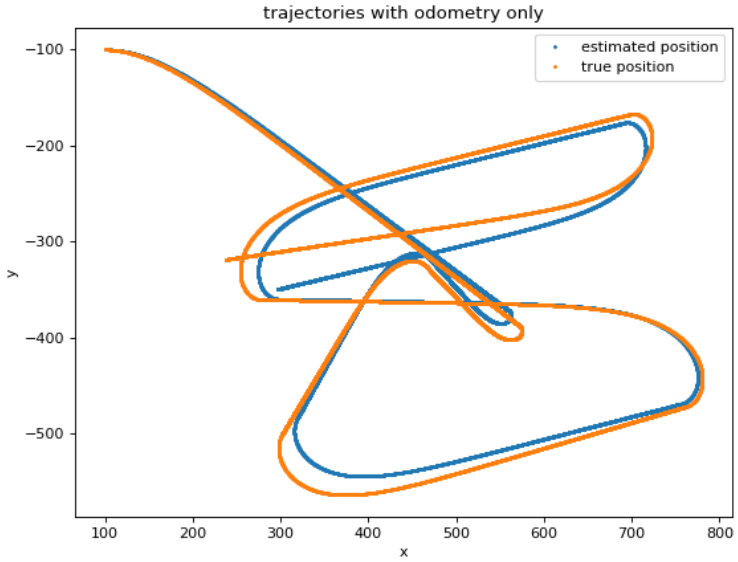

Figure 10: Trajectories with Odometry only

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 11.

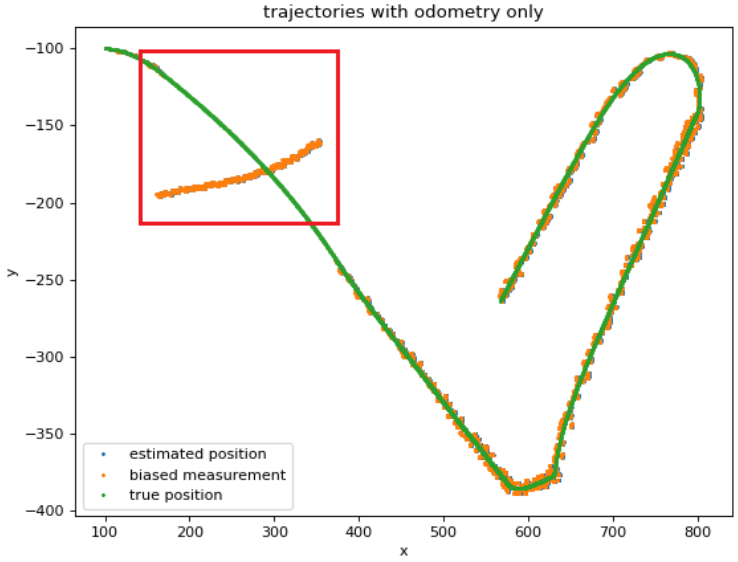

Figure 11: Trajectories Using Camera without Kalman Filter

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 7. 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 7) has a large impact on the result, and the comparison is posted in the following figure.

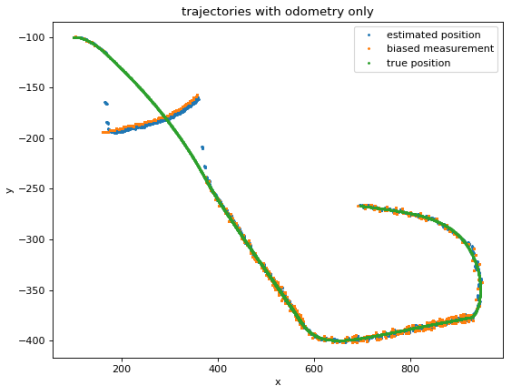

Figure 12: Trajectories Using Camera without Kalman Filter, Q = 1

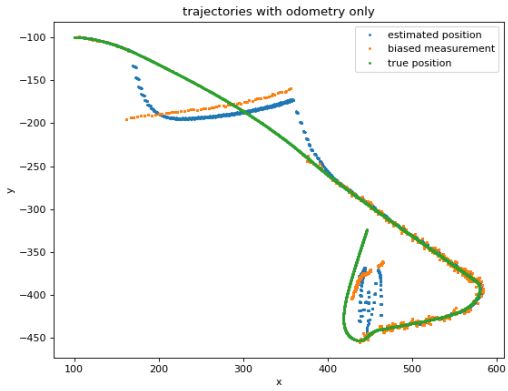

Figure 13: Trajectories Using Camera without Kalman Filter, Q = 0.05

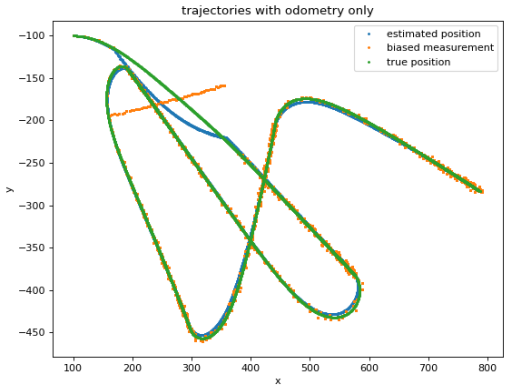

Figure 14: Trajectories Using Camera without Kalman Filter, Q = 0.0005

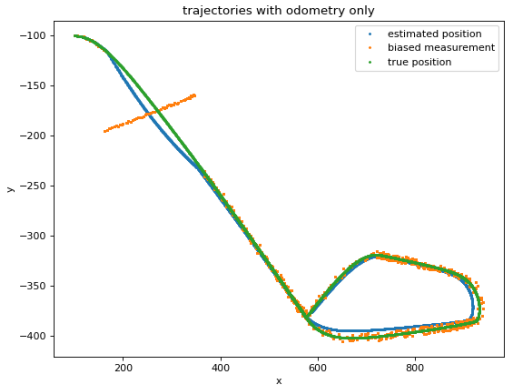

Figure 15: Robot Trajectory with EKF

From the figure 12 and 13, 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 14 and 15, 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 15.

Summary

I use an indicator(MSE mean square error) 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.

$$\text{MSE} = \frac{1}{l_w}\sum_{i}^{l_w} = 1(\hat{x_i} - x_i)^2 \tag{28}$$

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: MSE Result Comparison

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.

3.6 Conclusion

2022/12/12

4 Elisa-3

4.1 command

roscore

A Reference

[1] J. Ghommam, H. Mehrjerdi, M. Saad, and F. Mnif, "Formation path following control of unicycle-type mobile robots," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 727–736, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889009001894

[2] 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.

[3] 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.

[4] 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

[5] 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.

[6] C. K. Chui, G. Chen *et al.*, *Kalman filtering*. Springer, 2017.

[7] L. Armesto, S. Chroust, M. Vincze, and J. Tornero, "Multi-rate fusion with vision and inertial sensors," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, 2004, pp. 193–199 Vol.1.

[8] M. Kordestani, M. Dehghani, B. Moshiri, and M. Saif, "A new fusion estimation method for multi-rate multi-sensor systems with missing measurements," *IEEE Access*, vol. 8, pp. 47 522–47 532, 2020.

[9] A. Smyth and M. Wu, "Multi-rate kalman filtering for the data fusion of displacement and acceleration response measurements in dynamic system monitoring," *Mechanical Systems and Signal Processing*, vol. 21, no. 2, pp. 706–723, 2007.

[10] L. Yan, D. Zhou, M. Fu, and Y. Xia, "State estimation for asynchronous multirate multisensor dynamic systems with missing measurements," *IET Signal Processing*, vol. 4, no. 6, pp. 728–739, 2010.

[11] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 183–190, 1988.