Mark
lecture11
part1

Localization : Odometric localization

Robots need to know where is it, in order to move effectively in the environment. Localization is maintaining reasonably accurate estimate of the configuration of the robot as it moves. This is done using measurements and also using the model. By using these 2 sources of knowledge, we will build a system maintains estimate in time, over time of the robot configuration.

We use localization for control and planning. For both scenarios, we need to know where is the robot.

The first type of localization is Odometric Localization, which we will discuss during this lecture.

- Odometric localization.

Odometric localization is a type of localization when we localize the robot by using the internal sensors. For WMRs they can be encoders, inertia navigator sensors. All these sensors give information about the internal state of robot. They don't give information about external world.

As we said above, localization is needed for planning and feedback control. Because they require the knowledge of robot configuration

9.

We need to emphasize that, in fixed based manipulators if we have joint encoders we are always able to reconstruct precisely the position of every point of the robot. This is possible, because the robot has fixed base and by using simple kinematics computations and using joint encoder readings we can say always where each point of each joint is in the 3D World. However we cannot use this approach in WMRs, because they don't have fixed base. In WMRs, the wheel encoders, which count the rotation of the wheel, gives an information about how much space the robot has travelled, certainly. However it can't provide information about position and orientation of robot, because we don't know where the robot started from. So we can expect reasonably to compute some approximation of the travel space, but we can't transform it directly into an estimate of the position. Because we need to know the initial position of the robot, essentially.

To sum up, the localization is a procedure for maintaining such an estimate, over time and possibly in real-time, of the robot configuration q. So we want this to run at the same speed, the same frequency of the control cycles. Each control cycle

has possibility using the most updated estimate of the position

- We will discuss the localization with reference to the unicycle.

We consider that a unicycle moves under constant velocity inputs at each time interval $[t_k, t_{k+1}]$ (i.e. $v_k = const.$, $\omega_k = const$) as in a dipital control implementation. If v_k and ω_k are constant within each sampling interval, the robot moves along an arc of circle which has radius v_k/ω_k. If ω_k is zero, the robot moves a line segment because it will use only driving velocity. In this part, we need to remember the system equations of unicycle:

$$\dot{x} = v\cos\theta \quad (1)$$
$$\dot{y} = v\sin\theta \quad (2)$$
$$\dot{\theta} = \omega \quad (3)$$

We assume that q_k, v_k and ω_k are known. Thus, we can compute q_{k+1} by integration of the kinematic model over $[t_k, t_{k+1}]$. We can give generic example of change of velocities over time intervals in the figure 1.

Figure 1,a can be understood as piecewise constant driving velocity input and Figure 1,b can be understood as piecewise constant steering velocity input.

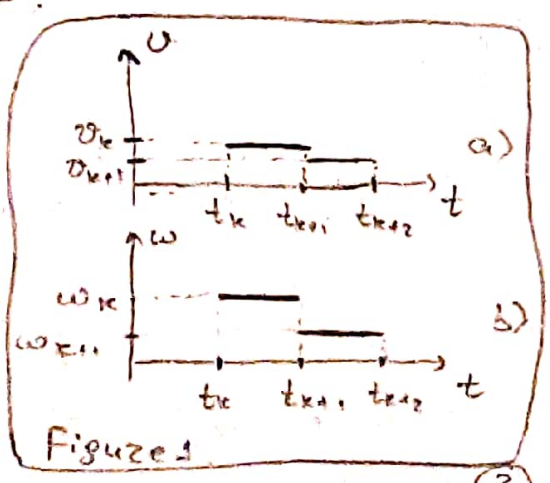

Figure 1

However v and ω are constant in the sampling interval, equations (1) and (2) are nonlinear, because θ is changing over time. Additionally because the change of θ is constant (ω_k) in time sample we can integrate easily. T_s is the time interval which is called as sampling time.

$$T_s = t_{k+1} - t_k \quad (4)$$

We can compute θ_{k+1} by integrating θ_k as below:

$$\theta_{k+1} = \theta_k + \omega_k T_s \quad (5)$$

Computation of (5) is exact, because equation(3) is linear. However change of x and y are not exact, because (1) and (2) are nonlinear. Thus we will compute their integration by changing θ with θ_k, which will give us approximate results. This method is called as Euler integration

$$x_{k+1} = x_k + T_s v_k \cos\theta_k \quad (6)$$

$$y_{k+1} = y_k + T_s v_k \sin\theta_k \quad (7)$$

Because of the term with θ_k, (6) and (7) are approximate integrations. We can show geometric interpretation as in Figure2. If we analyze the motion we will have following comments:

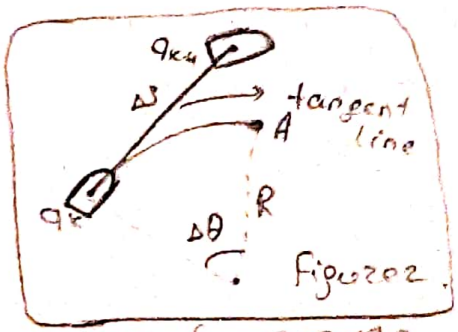

Figure2

→ We changed θ with θ_k at q_k configuration

→ Within $[t_k, t_{k+1}]$ time interval unicycle moves aligned the tangent line

→ Δs has the same length with arclength of the circle which has the radius R

→ The orientation of unicycle at the q_{k+1} configuration would be same with the orientation at A, if it move along the arc-length.

→ It seems there is error in the motion. However T_s is about 10^{-3}, 10^{-4} of the second, this error is neglectable.

Notice that, this result comes up when we implement Euler integration.

→ 2nd order Runge-Kutta integration

This method gives us better result than Euler integration method gave.

In this method, θ_{k+1} is the same as given by the equation (5):

$$(5) \Rightarrow \theta_{k+1} = \theta_k + \omega_k T_s$$

We see that, final value of θ at the end of the time interval is computing by the sum of the initial value of θ and change of it within sampling time. Thus, we see that change of angle is $\omega_k T_s$. In order to show how do x and y change within sampling time, we will use $\theta_k + \frac{\omega_k T_s}{2}$ term instead of θ value in (1) and (2). What does it mean? Instead of using exact value of θ_k, we add also half of change of orientation. Thus we get:

$$x_{k+1} = x_k + v_k T_s \cos\left(\theta_k + \frac{\omega_k T_s}{2}\right) \quad (2)$$

$$y_{k+1} = y_k + v_k T_s \sin\left(\theta_k + \frac{\omega_k T_s}{2}\right) \quad (3)$$

This method gives us the average orientation during $[t_k, t_{k+1}]$ which is demonstrated by figure 3. Notice that, as a consequence x_{k+1} and y_{k+1} are still approximate, but more accurate. The following comments can be made:

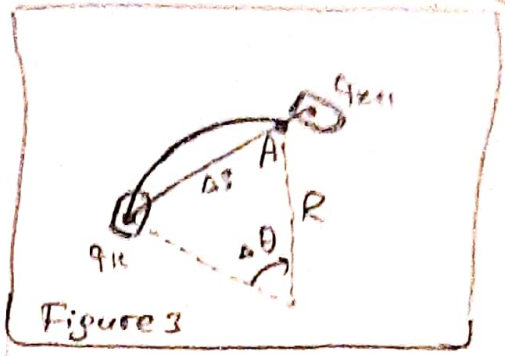

Figure 3

→ Length of Δs is same with arclength.

→ If unicycle would have moved along the arclength, it will be able to have the same orientation at the point A with the orientation it has at the configuration q_{k+1}.

→ The error is neglectable again, because of the value of T_s.

- **Exact integration**

In this case x_{k+1}, y_{k+1} will be computed as below:

$$x_{k+1} = x_k + \frac{v_k}{\omega_k}(\sin\theta_{k+1} - \sin\theta_k) \quad (10)$$

$$y_{k+1} = y_k + \frac{v_k}{\omega_k}(\cos\theta_{k+1} - \cos\theta_k) \quad (11)$$

On the other hand, θ_{k+1} is computed as given by the equation (5).

$$(5) \Rightarrow \theta_{k+1} = \theta_k + \omega_k T_s$$

Note 1: Generation of (10) and (11) are given as Appendix 11.1.

If $\omega_k = 0$, x_{k+1} and y_{k+1} are defined, regardless $\frac{v_k}{\omega_k}$ term in equations (10) and (11). Because, from (5), we get such result that angle does not

change. In this case, the result comes up that x_{k+1} and y_{k+1} are still defined and coincide with the solution by Euler and Runge-Kutta methods.

Notes: for $\omega \approx 0$, a conditional instruction may be used in the implementation.

Geometric expression is demonstrated in the Figure 4.

Figure 4

Every method that we use for integration, require the knowledge of v_k and ω_k. There might be an idea that, we can compute integrations by using velocities which are generated by controllers. In principle it is true. However in practice it's never done. If we remember Actual Control scheme as we mentioned before, we can analyze the idea precisely. The block diagram is given by Figure 5.

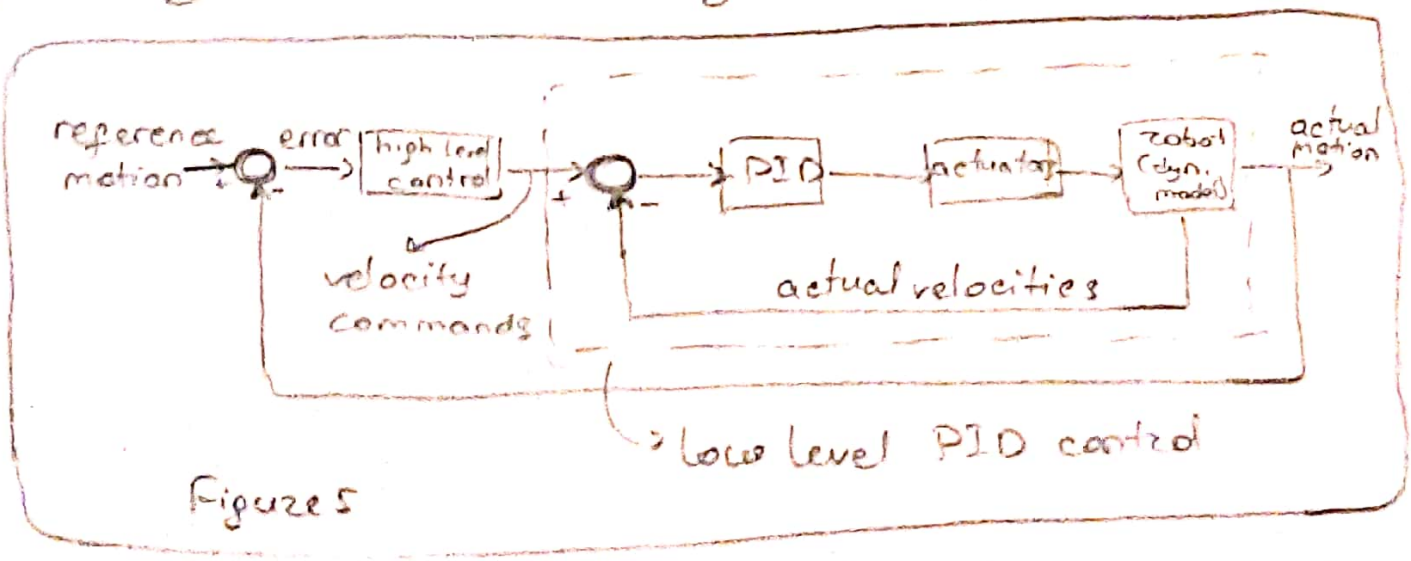

Figure 5

The main idea that they are not used is; v_k and ω_k are reference (nominal) velocity inputs, and they are not actual velocities that we give to the robot, directly. Actual velocities that robot experiences are not same as velocity commands (which are injecting to the model)

There becomes an error that low level PID control take care of it, in order to minimize the error. Thus, we don't use v_k and ω_k are coming from our controller. What do we do?

* Using sensors

Instead of this approach we use sensors, in order to reconstruct the values for v_k and ω_k. The way of doing it is to measure the effect of the velocity inputs.

How will we do that?

We know that the space that robot travels within time interval (T_s) with constant driving velocity can be defined as below:

$$\Delta s = v_k T_s \quad (12)$$

On the other hand, total change of orientation during T_s with constant steering velocity is defined as below:

$$\Delta\theta = \omega_k T_s \quad (13)$$

Then we can find the ratio between velocities

$$\frac{\Delta s}{\Delta\theta} = \frac{v_k}{\omega_k} \quad (14)$$

The idea is, let's compute Δs and $\Delta\theta$ and reconstruct actual values of inputs from those. By doing it we will use actual velocities instead of nominal velocities, which change travelling space and orientation.

If we substitute those values in integration methods we will have actual values according to Δs and $\Delta\theta$ values.

How do we compute Δs and $\Delta\theta$ values?

We compute these values by using proprioceptive sensors (encoders). We use in WMRs wheel encoders typically.

For instance, for a differential-drive robot Δs and $\Delta\theta$ are computed as below:

$$\Delta s = \frac{r}{2}(\Delta\phi_R + \Delta\phi_L) \quad (15)$$

$$\Delta\theta = \frac{r}{d}(\Delta\phi_R - \Delta\phi_L) \quad (16)$$

In (15) and (16), r is the radius of the wheel, ϕ_R and ϕ_L are change of orientation of right and left wheels, respectively, and d is the distance between wheels. $\Delta\phi_R$ and $\Delta\phi_L$ are measured by wheel encoders.

To sum up, we will compute Δs and $\Delta\theta$ by using the wheel encoders. Then, we will use these values to compute actual values by using integrations. This is called as odometric localization or dead-reckoning. Subject to an error (odometric drift) that grows over time, becoming significant over sufficiently long paths. The errors can happen because of wheel slippage (model perturbation), inaccurate calibration of r, d, wheel radius (model uncertainty) or numerical integration error.

Effective localization methods use proprioceptive sensors as well as exteroceptive sensors. In other words, we use proprioceptive sensors to get actual values according to the internal mechanism and we will add exteroceptive results, in order to correct the mistakes might be made by internal computations.

- Kalman Filter

The motivation of Kalman Filter is, estimating the robot configuration by iterative integration of the kinematic model (dead reckoning) is subject to an error that diverges over time.

Effective localization methods use proprioceptive as well as exteroceptive sensors: If an environment map is known, compare the actual sensor readings with those predicted using the current estimate. This will lead us to perform probabilistic localization.

Probabilistic localization: Instead of maintaining a single hypothesis on the configuration, maintain a probability distribution over the space of all possible hypotheses. Kalman filter usage is one of the possible approaches.

Before going ahead, we will remember some basic concepts we are going to use for Kalman filter.

- Expected (or mean) value: Given a vector random variable X with probability density function $f_X(x)$, its expected value is

computed as below:

$$E(X) = \bar{X} \int_{x \in \mathbb{R}^n} x f_X(x)\, dx \quad (17)$$

Covariance matrix of it is:

$$P_X = E((X-\bar{X})(X-\bar{X})^T) \quad (18)$$

On the other hand, X has multivariate Gaussian distribution if:

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^n |P_X|}} e^{-\frac{1}{2}(x-\bar{X})^T P_X^{-1} (x-\bar{X})} \quad (19)$$

- Geometric interpretation of Gaussian distribution is given by the figure 6.

If we find constant values for the equation (19), they will be ellipsoids as in figure 6.

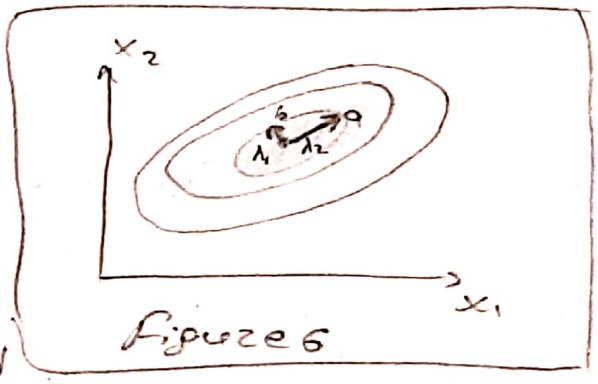

figure 6

The principal axes of ellipsoid are directed as the eigenvectors of P_X. Their squared relative lengths are given by the corresponding eigenvalues.

In terms of uncertainty, we can say that along the λ_2 vector in figure 6, uncertainty is more than along the λ_1. However, at points a and b measurement errors are same.

To sum up, covariance matrix will give us information about uncertainty of Random Variable X.

Now we can start the explaining Kalman filter. At the first step we will explain Kalman filter without noise.

⇨ Kalman Filter without noise.

Kalman Filter without noise can be considered as simple observer. In control theory or signal processing filter is used when there is noise. However, observer is more deterministic than filters.

We consider a linear discrete time system without noise which is depicted by following expressions:

$$x_{k+1} = A x_k + B u_k \quad (20)$$

$$y_k = C_k x_k \quad (21)$$

$x_k \rightarrow$ current state

$x_{k+1} \rightarrow$ next state

$u_k \rightarrow$ input to the model

$y_k \rightarrow$ output of the current state.

Notice that, in basic Kalman Filter is applied to linear systems. On the other hand, in general A_k, B_k and C_k matrices are depend on time. However, they can be constant as well.

We will build recursive observer (each state depends on previous state) that computes an estimate $\hat{x}_{k+1}$ of x_{k+1} from u_k, y_{k+1} and previous estimate $\hat{x}_k$.

The general scheme of this Kalman filter that we described (recursive observer) is demonstrated in Figure 7.

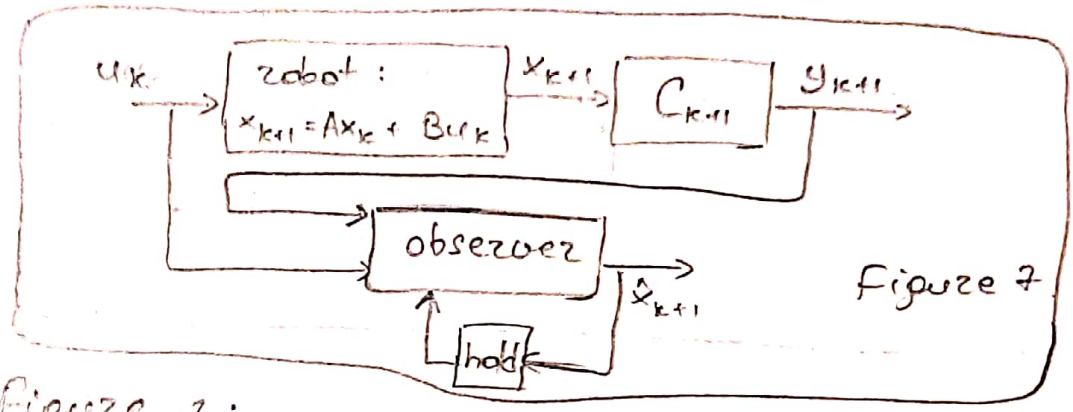

Figure 7

In Figure 7:

→ We assume that we know u_k;

→ The known u_k is not nominal inputs. They are reconstructed from the proprioceptive sensors.

→ We measure y_{k+1}, thus we also know it.

→ We want to compute $\hat{x}_{k+1}$, which is the **estimate**.

→ Because we have recursive observer, we hold $\hat{x}_{k+1}$ for a sampling interval use it in the observer equation.

The Kalman filter we are going to build on this observer is made by 2 steps: **Prediction and Correction**.

→ **Prediction** stage is essentially is **odometric localization**. We are going to use odometric localization for prediction. In other words, we are going to use system equations (or process dynamics), in order to generate the first estimate, which is called as **intermediate estimate** : $\hat{x}_{k+1/k}$

Intermediate estimate propagates the previous estimate on the basis of system equations and available inputs.

→ In the correction stage, we are going to correct the intermediate estimate that we got from the first stage. This is done by using external measurements. This step does the correction by the difference of predicted output and actual output.

To sum up Kalman filter can be shown as in Figure 8.

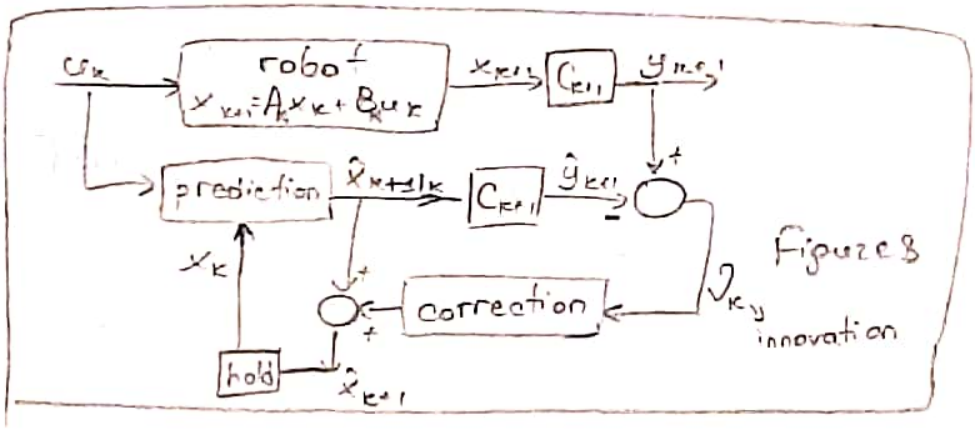

Figure 8

In Figure 8, y_{k+1} and $\hat{y}_{k+1}$ are measured and predicted outputs, respectively.

How do we do computations on those stages?

→ Prediction stage:

By assumption of knowing A_k, B_k and u_k we compute intermediate estimate as below:

$$\hat{x}_{k+1|k} = A\hat{x}_k + Bu_k \quad (22)$$

What is the problem with this prediction that we need to correct it?

In general, the result of (22) will be inconsistent with the measurement.

Since we don't know the true value of the state in order to compute output, we predict output by using the intermediate estimate. However, the measured output and predicted output are not same,

→ **Correction stage**:

To be consistent with the measured value of the output, x_{k+1} must belong to the hyperplane Ω which is given by the equation (23) and demonstrated in Figure 9.

$$\Omega = \{x : C_{k+1} x = y_{k+1}\} \quad (23)$$

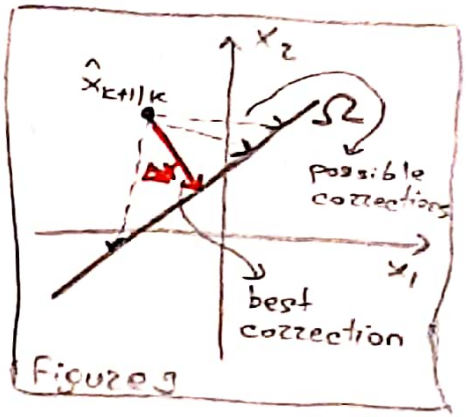

Figure 9

If we denote the correction as Δx, it must satisfy:

$$C_{k+1}(\hat{x}_{k+1|k} + \Delta x) = y_{k+1} \quad (24)$$

If $\hat{x}_{k+1|k}$ was on the Ω hyperplane, it is obvious that, we would not have needed any correction. Because of the inconcistency between measured and predicted outputs, we need to use the correction stage.

In Figure 9, Δx is intuitively "best" correction, because it is the closest correction that will bring $\hat{x}_{k+1|k}$ to Ω hyperplane and we believe it is accurate.

At this point finding Δx becomes an optimization problem which implies that, the estimate value of state after correction must give us such predicted output that gives us the same value as measured output. Then we will use minimum norm optimization method in order to satisfy the constraint is generated as below:

$$C_{k+1} \cdot \hat{x}_{k+1} = y_{k+1} \quad (25)$$

If we use the value of $\hat{x}_{k+1}$ after correction as in (26) then we will get the equation (24):

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + \Delta x \quad (26)$$

(26) =>(25): $C_{k+1}(\hat{x}_{k+1|k} + \Delta x) = y_{k+1}$ <=(24)

By using (24) we can generate our constraint for finding Δx:

$$C_{k+1} \Delta x = y_{k+1} - C_{k+1} \hat{x}_{k+1|k} \quad (27)$$

(27) is the constraint that minimizing Δx will minimize this constraint, as well. We know that, the solution of this optimization problem is following:

$$\Delta x = C_{k+1}^{\dagger}(y_{k+1} - C_{k+1} \hat{x}_{k+1|k}) = C_{k+1}^{\dagger} v_{k+1} \quad (28)$$

where $C_{k+1}^{\dagger}$ and v_{k+1} are pseudoinverse of C_{k+1} and the difference between measured and predicted outputs (innovation), respectively.

Pseudoinverse of C_{k+1} is computed as below:

$$C_{k+1}^{\dagger} = C_{k+1}^{T}\left(C_{k+1}C_{k+1}^{T}\right)^{-1} \quad (29)$$

Innovation is computed as below:

$$\nu_{k+1} = y_{k+1} - C_{k+1}\hat{x}_{k+1|k} \quad (30)$$

To sum up, we can write the equation of the correction process as below:

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + C_{k+1}^{\dagger}\nu_{k+1} \quad (31)$$

In general, the estimate $\hat{x}_{k+1}$ will not converge to the true value x_{k+1}, because the correction is naive: estimation errors directed as $\mathcal{R}$ are not corrected. We are performing the smallest corrections by assuming the best correction is the one that keeps us closer to the previous estimate. Notice that, it is just the assumption and we can't expect it happens realistically.

➡ Kalman filter with noise:

There will be noise in the process, because of modelling error, perturbations, noise in the inputs which are collected in one term and are added to the process part of the system equations. Additionally, noise in the output presents the sensor error. For now, we will assume that our sensors do not have error and because of that computation of the output will remain same as given by the equation (21). Then we can show our system equations as following:

$$x_{k+1} = A_k x_k + B_k u_k + v_k \quad (32)$$

(31) $\Rightarrow$ $y_k = C_k x_k$

In the equation (32), v_k represents process noise that all sources of nonlinear errors are collected in that single term. On the other hand we assume that, v_k process error is white Gaussian noise. What is white noise? Each value of the error is independent of the previous value (i.e. v_k doesn't dependent on v_{k-1}). We also assume that, expected value of this error is zero (i.e., it is zero mean error). It can get positive and negative values and, it will get zero as mean value. Covariance matrix of this error will be V_k. V_k will give us information about how much uncertainty we have and in which direction. Notice that, because of the randomness of the system, v_k is also random variable.

By using this Kalman filter, what are we going to do now?

We don't simply estimate the state. We're going to estimate the state and the associated covariance, because now the state and its prediction are going to be random variables. So, we are going to add description of the uncertainty to the variable estimate. However, the method changed, we are going to keep

prediction/correction structure as previous Kalman filter. Notice that, we will denote associated covariance as P_{k+1}.

Now we can start to design Kalman filter.

→ State prediction.

We are going to compute state prediction as before, which is given by the equation (22):

$$(22): \quad \hat{x}_{k+1/k} = A_k \hat{x}_k + B_k u_k$$

Why we did not use the noise? Because we don't know what is noise and it has zero mean. Thus, while predicting the state, we take the noise as zero.

→ Covariance prediction.

As we said, by designing this Kalman filter we will consider associated covariance matrix P_{k+1}, too. At this step we will predict the associated covariance matrix.

$$P_{k+1/k} = E((x_{k+1} - \hat{x}_{k+1/k})(x_{k+1} - \hat{x}_{k+1/k})^T) =$$

$$= E((A_k(x_k - \hat{x}_k) + v_k)(A_k(x_k - \hat{x}_k) + v_k)^T) =$$

$$= E(A_k(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T A_k^T) +$$

$$+ E(A_k(x_k - \hat{x}_k) v_k^T + v_k(x_k - \hat{x}_k^T) A_k^T) + E(v_k, v_k^T) \quad (33)$$

$P_{k+1/k}$ is the associated covariance to the intermediate estimate. (33) is computed by using the definition of covariance matrix. To sum up, $P_{k+1/k}$ can be written as below:

$$P_{k+1/k} = A_k P_k A_k^T + V_k \quad (36).$$

Generation of (34) is given by Appendix 11.2. We can visualize the covariance prediction by Figure 10. If we want to explain what is demonstrated in Figure 10, we'll be following the steps below:

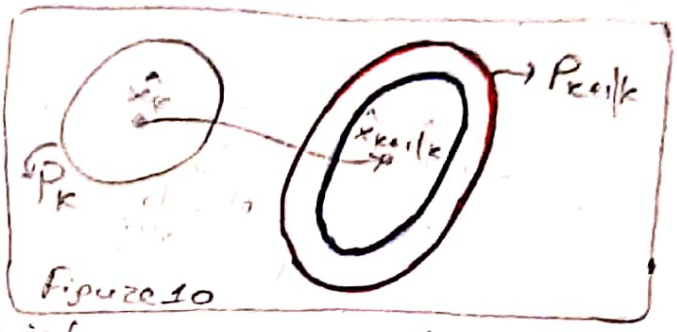

Figure 10

- We predict the state which comes with associated covariance (ellipse in Figure 10.).
- We compute intermediate estimate by using the equation (32) and because of v_k and drift it becomes $\hat{x}_{k+1|k}$ (by using system equations of KF).
- The ellipse ($P_{k+1|k}$) around $\hat{x}_{k+1|k}$ is the associated covariance matrix. According to the equation (34):
  - It rotates and squished → blue ellipse
  - Because of v_k it grows → red ellipse

Notice that the P_k becomes blue ellipse because of the motion

Why ellipse grows because of the v_k? We know that ellipse represents uncertainty, Because we have noisy input when we determine $\hat{x}_{k+1|k}$. It results by growing of uncertainty (i.e., ellipse). Our task is shrink the red ellipse. Correction step has the objective to contain the growth of uncertainty (shrink the red ellipse).

EKF Lecture 1 part 6

•Correction stage.

In previous version, because we had deterministic approach, we could use naive correction. In this case our estimate has associated covariance matrix and because of it we can compute correction more precisely. Why? Because this time we have a description of uncertainty which is associated to our estimate. This description are the ellipsoids which have certain axes, length and because of those our correction will be more effective. Ellipsoids tell us that what are the most likely case around mean value. Figure 11 demonstrates this situation.

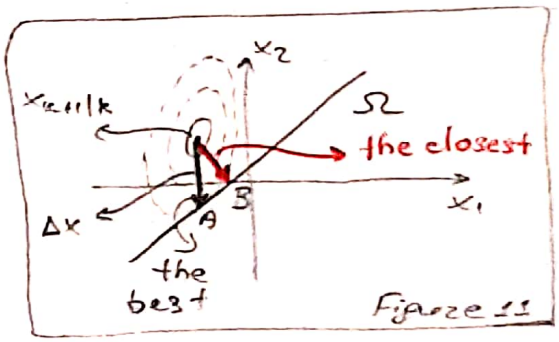

Figure 11

The ellipsoids set is the representation of the uncertainty which is associated to our intermediate estimate $\hat{x}_{k+1|k}$.

The smallest correction (red) is actually less-likely than the best correction (Black). Because, the value of the ellipsoid associated to the point A is larger than the value of ellipsoid (in terms of probability) associated to the point B. Because B would be in an external ellipsoid cort the point A. So the state at the point A, is the most likely given the current estimate of the covariance.

If we look for such correction that brings our estimate to the Ω hyperplane, it should be the "best" correction in Figure 11.

How do compute this "best" correction?

Now we will compute the smallest correction by using Gaussian.

We should choose Δx so as to get the most likely x in S_2, i.e., the x that maximizes the gaussian distribution defined by $\hat{x}_{k+1|k}$ and $P_{k+1|k}$.

$$p(x) = \frac{1}{\sqrt{(2\pi)^n |P_{k+1|k}|}} e^{-\frac{1}{2}(x-\hat{x}_{k+1|k})^T P_{k+1|k}^{-1} (x-\hat{x}_{k+1|k})} \quad (35)$$

We need to maximize $p(x)$ and it happens when the exponent minimizes.

Notice that we solve this correction problem by using weighted norm. If weighted norm matrix is identity we would get a circle instead of ellipsoid. That circle would have taken us to the previous solution (naive correction). However, if we choose this weighted norm matrix, appropriately, then we get the approach we explained by Figure 11.

As we said we need to minimize the exponent in order to maximize $p(x)$. If we analyze the exponent we can see that it is the distance (between true value of state, and intermediate estimate) which is weighted by inverse covariance matrix which is associated with intermediate estimate ($P_{k+1|k}^{-1}$). This exponent which is weighted distance, is called as (squared) Mahalanobis distance and computed as below:

$$\Delta x^T P_{k+1|k}^{-1} \Delta x = \|\Delta x\|_m^2 \quad (36)$$

Now we have an optimization problem that we need to solve.

$$\min \|\Delta x\|_M \quad (37)$$

The constraint to this optimization is same as previous KF approach. We can rewrite the explanation as:

- We need to such Δx that minimize the error (or difference) between measured and predicted output. The best solution would be that they become equal which is given by (25):

$$(25) \Rightarrow C_{k+1} \hat{x}_{k+1} = y_{k+1}$$

Then by using (26), (24) again we get (27) which is constraint for our optimization problem (37):

$$C_{k+1} \Delta x = y_{k+1} - C_{k+1} \hat{x}_{k+1|k} \quad (27)$$

Then we can find Δx according to the optimization problem under the given constraint as following:

$$\Delta x = C^{\dagger}_{k+1,M} (y_{k+1} - C_{k+1} \hat{x}_{k+1|k}) = C^{\dagger}_{k+1,M} \nu_{k+1} \quad (38)$$

In (38), we know that ν_{k+1} is the innovation and it is computed by the equation (30). On the other hand $C^{\dagger}_{k+1,M}$ is weighted pseudoinverse of C_{k+1} is defined as below:

$$C^{\dagger}_{k+1,M} = P_{k+1|k} C^{T}_{k+1} (C_{k+1} P_{k+1|k} C^{T}_{k+1})^{-1} \quad (39)$$

If covariance matrix $P_{k+1|k}$ is identify (39) becomes to simple pseudoinverse that leads us to previous solution (naive correction)

Now we can apply covariance correction.

$$P_{k+1} = P_{k+1|k} - C^{\dagger}_{k+1,M} C_{k+1} P_{k+1|k} \quad (40)$$

As a result of this correction we will shrink the ellipse $P_{k+1|k}$ and we'll get P_{k+1} ellipse, as we expected.

To sum up we can wrap up a step filter by following steps:

→ Intermediate estimate:

(42) => $\hat{x}_{k+1|k} = A\hat{x}_k + Bu_k$

→ Associated Covariance Matrix to $\hat{x}_{k+1|k}$:

(34) => $P_{k+1|k} = A_k P_k A_k^T + V_k$

→ Correction to the state:

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + C^{\dagger}_{k+1,M} \nu_k \quad (41)$$

→ Correction to the associated Covariance Matrix to $\hat{x}_{k+1}$:

(40) => $P_{k+1} = P_{k+1|k} - C^{\dagger}_{k+1,M} C_{k+1} P_{k+1|k}$

In this case there will be one problem:

If there is no measurement noise, the covariance estimate will become singular (no uncertainty in the normal direction to the measurement hyperplane).

Complete Kalman Filter.

In this case we will also include measurement noise. Measurement noise appears in KF system equations as a single term and it includes all errors because of sensors. We will denote this noise with w_k which is white gaussian noise. As we told before, white gaussian noise means it has zero mean and, its current value is independent of its precious value. In this case only output representation of KF equations will change and process dynamics will remain as in Kalman filter with measurement noise case.

$$(32) \Rightarrow \quad x_{k+1} = A_k x_k + B_k u_k + v_k$$

$$y_{k+1} = C_k \cdot x_k + w_k \qquad (42)$$

Associated covariance matrices to v_k and w_k will be V_k and W_k, respectively.

Because process dynamics is same as before, the equations of prediction will remain as well.

$$(22) \Rightarrow \quad \hat{x}_{k+1|k} = A_k \hat{x}_k + B_k u_k$$

$$(34) \Rightarrow \quad P_{k+1|k} = A_k P_k A_k^T + V_k$$

The situation in correction will change, because now we involved the measurement noise, too. The measurement hyperplane in Figure 11 was associated the actual measurement because we assumed there is no measurement noise. However, now in our case there is randomness in measurement as well. Because the measurement hyperplane will change, we can certainly say that correction will also change. The $\hat{x}$ must change because it is not most likely measurement that we have.

Instead of this, it represents actual measurements that we read. The output that Ω represents is one sample from Gaussian distribution of measurements because it is affected by random noise. We only know that y_{k+1} is drawn from Gaussian distribution with mean value $C_{k+1} \hat{x}_{k+1|k}$ (w_k doesn't effect mean value because it is gaussian white noise) and covariance matrix W_{k+1}. This can be visualized by Figure 12.

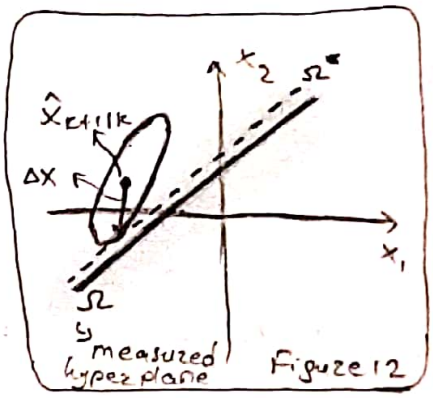

Figure 12

It is seen from figure 12 that the "best" correction Δx is still the closest to $\hat{x}_{k+1|k}$ according to $P_{k+1|k}$, but now it lies on Ω^*, which is "most likely" hyperplane.

In mathematical representation we will follow the steps below:

→ First we compute the most likely output value y^*_{k+1} given the predictions and the measured output y_{k+1}.

→ Then compute the associated "most likely" hyperplane

$$\Omega^* = \{ x : C_{k+1} x = y^*_{k+1} \} \quad (43)$$

→ Finally we compute the correction Δx as before but using Ω^* instead of Ω.

Then we get following equations for correction stage:

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + R_{k+1} v_{k+1} \quad (44)$$

$$P_{k+1} = P_{k+1|k} - R_{k+1} C_{k+1} P_{k+1|k} \quad (45)$$

In (45), R_{k+1} is Kalman gain matrix which is defined as below:

$$R_{k+1} = P_{k+1|k} C_{k+1}^{T} (C_{k+1} P_{k+1|k} C_{k+1}^{T} + W_{k+1})^{-1} \quad (45)$$

Notice that if W_{k+1} will be zero matrix (i.e. sensors are ideal), R_{k+1} becomes weighted pseudoinverse matrix. In this case, we also consider measurement noises that leads us to have R_{k+1} (Kalman gain matrix) instead of weighted pseudoinverse matrix.

Matrix R weighs the accuracy of the prediction versus that of the measurements. If R is large then measurements are more reliable and if R is small then prediction is more reliable. We can see them from equation (44). If R is large we perform more correction because R is used as "coefficient" of innovation. If we perform more correction, thus we believe our measurement than prediction. If R is small we perform less correction that means we believe our prediction.

The block diagram can be shown as below:

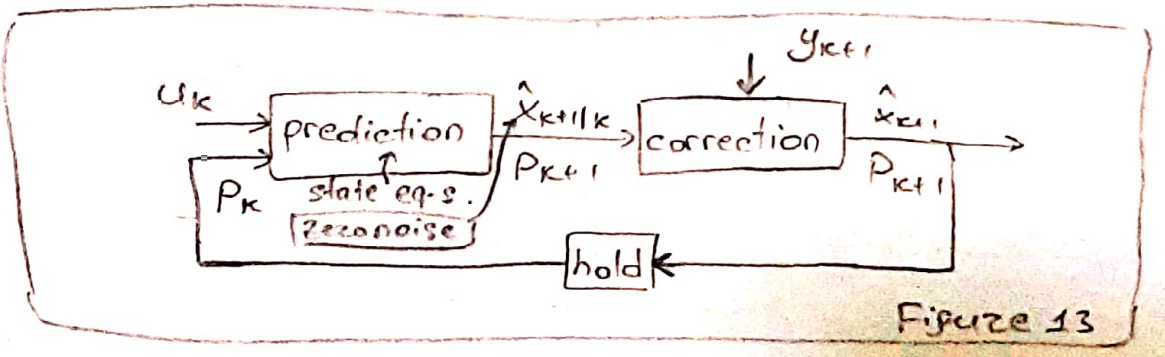

Figure 13

Few comments on Kalman Filter:

- The Kalman filter provides an optimal estimate in the sense that $E(x_{k+1} - \hat{x}_{k+1})$, which is expected error, is minimized for each k.
- The KF is also correct, i.e., it provides mean value and covariance of the posterior gaussian distribution
- If the noises have non-gaussian distributions, the KF is still the best linear estimator but there might exist more accurate nonlinear filters
- If the process is observable, the estimate produced by the KF converges, in the sense that $E(x_{k+1} - \hat{x}_{k+1})$ is bounded for all k.

- Extended Kalman Filter.

Let's consider a nonlinear discrete-time system with noise which is defined by following expression

$$x_{k+1} = f_k(x_k, u_k) + \theta_k \quad (47)$$

$$y_k = h_k(x_k) + \omega_k \quad (48)$$

In these equations, we assume that f_k and h_k are differentiable wrt to x_k for each k.

One simple way to build a filter is to linearize the system dynamic equations around the current estimate and then apply KF equations on the resulting linear approximation.

- The resulting Extended Kalman filter (EKF) is:

$$\hat{x}_{k+1|k} = f(\hat{x}_k, u_k) \quad (49)$$

$$P_{k+1|k} = F_k P_k F_k^T + V_k \quad (50)$$

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + R_{k+1} v_{k+1} \quad (51)$$

$$P_{k+1} = P_{k+1|k} - R_{k+1} H_{k+1} P_{k+1|k} \quad (52)$$

Equations (49), (50) are prediction stage's representation and, (51) and (52) stand for correction stage.

In equation (50), F_k is defined as below:

$$F_k = \left.\frac{\partial f(x_k)}{\partial x_k}\right|_{x_k=\hat{x}_k} \quad (53)$$

In equation (52), H_{k+1} is defined as below:

$$H_{k+1} = \left.\frac{\partial h(x_k)}{\partial x_k}\right|_{x_k=\hat{x}_k} \quad (54)$$

Additionally Kalman Gain matrix is defined as below:

$$K_{k+1} = P_{k+1|k} H_{k+1}^T \left(H_{k+1} P_{k+1|k} H_{k+1}^T + W_{k+1}\right)^{-1} \quad (55)$$