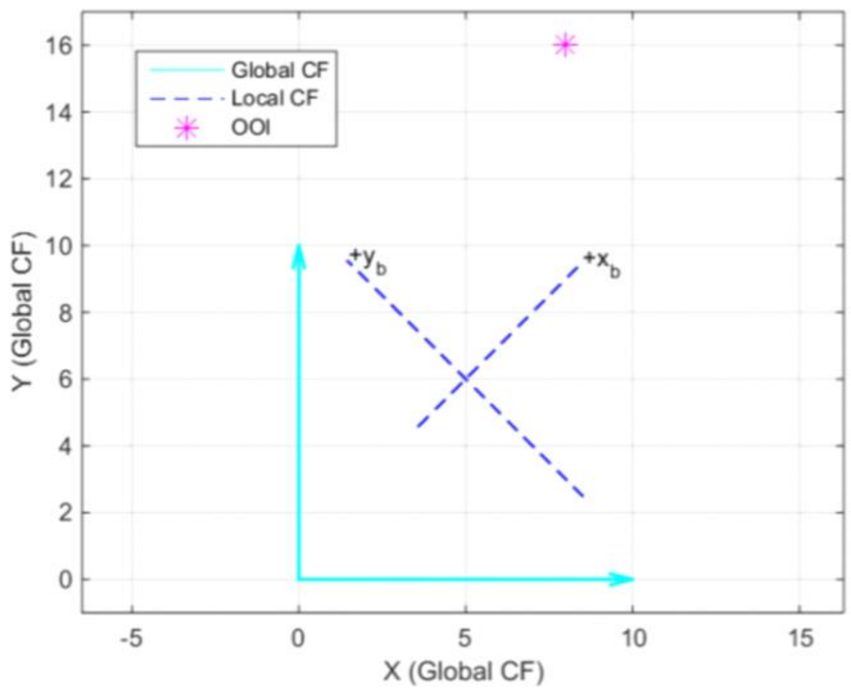
Coordinate Transforms

Sunday, 28 April 2024 2:00 PM



R = $\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

- theta = ~45 degrees in this example

T = $\begin{bmatrix} x_{trans} \\ y_{trans} \end{bmatrix}$

- [5;6] in this example

Local to global:

$$\boxed{p_{global} = R * p_{local} + T}$$

Global to local

$$\boxed{p_{local} = R^T\left(p_{global} - T\right)}$$

- since $R^{-1} = R^T$

Euler discrete time approximation

Thursday, 22 February 2024 5:01 PM

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$

$$\Downarrow$$

$$\mathbf{x}(t + \Delta t) \cong \mathbf{x}(t) + \Delta t \cdot \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$$

Matlab implementation:

```
function X1 = Example_Question(length, dt, X0, L1, velocity, steerAngle)
    % length = time in seconds
    % dt = sample time
    numSamples = length/dt;
    % Drive in a circle
    % Parameters:
%     L1 = 1.5;
%     L2 = 3;
%     steerAngle = 45*pi/180;
%     velocity = 1; % Constant velocity
%     X0 = [0; 0; 0];

    % Storage variables:
    X1 = X0;
%     X2 = X0;

    % Run sim:
    for i = 1:numSamples
        X1(:, i+1) = model(X1(:, i), dt, L1, velocity, steerAngle);
    end


%
end


function X = model(X0, dt, L, velocity, steerAngle)
    dX = [velocity*cos(X0(3)); velocity*sin(X0(3)); velocity/L*tan(steerAngle)];
    X = X0 + dt*dX;
end
```

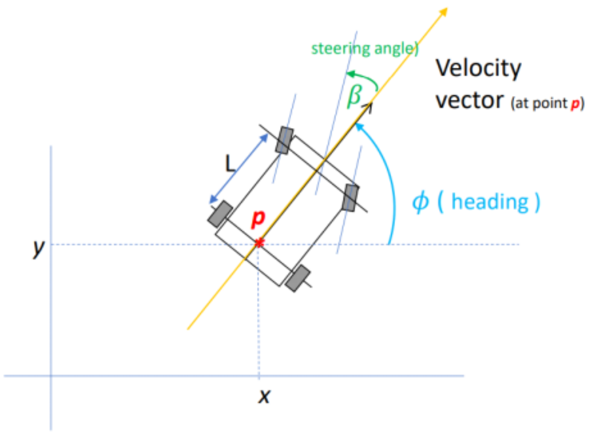
Kinematic Models in this course

Wednesday, 1 May 2024 2:59 PM

Inputs are the steering angle and the linear velocity (at point ***p***)

$$\mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \phi(t) \end{bmatrix}, \quad \mathbf{u}(t) = \begin{bmatrix} v(t) \\ \beta(t) \end{bmatrix}$$

$$\frac{d\mathbf{x}(t)}{dt} = \frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \phi(t) \end{bmatrix} = \begin{bmatrix} v(t)\cdot\cos(\phi(t)) \\ v(t)\cdot\sin(\phi(t)) \\ \frac{v(t)}{L}\cdot\tan(\beta(t)) \end{bmatrix}$$



$v(t)$: speed at the center-back of the vehicle, **p**, at time t

$\beta(t)$: steering angle at time t

$x(t), y(t), \varphi(t)$: vehicle's pose.

$$\mathbf{v}(t) = \begin{bmatrix} v(t)\cdot\cos(\varphi(t)) \\ v(t)\cdot\sin(\varphi(t)) \end{bmatrix}$$: velocity vector at point **p**

$$\frac{dx}{dt} = v(t)\cdot\cos(\varphi(t))$$

$$\frac{dy}{dt} = v(t)\cdot\sin(\varphi(t))$$

$$\frac{d\varphi}{dt} = \omega(t)$$

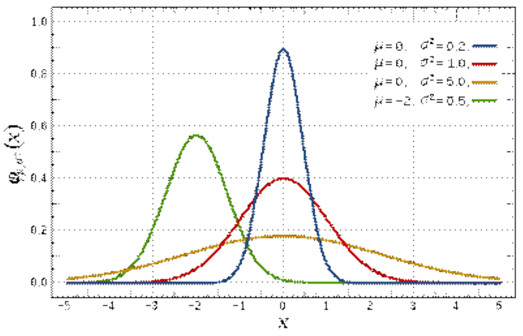
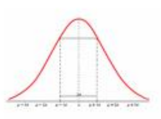
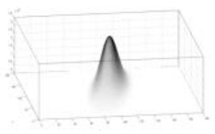
$v(t)$: speed the center-back of the vehicle at time t

$\omega(t)$: yaw rate at time t

$x(t), y(t), \varphi(t)$: vehicle's pose.

Probability / Statistics

Thursday, 22 February 2024 5:02 PM

Topic	Explanation	Image
$X \sim N(\mu, \sigma^2)$ "Normal" (aka "Gaussian") Distribution	This means that X is a random variable which follows a normal distribution with: - mean (expectation) μ - standard deviation σ - variance σ^2 https://en.wikipedia.org/wiki/Normal_distribution Note: "White gaussian noise" follows this distribution.	 Probability density function examples (normal distributions)
$\boldsymbol{x} \sim p(\boldsymbol{x})$ $= N(\hat{\boldsymbol{x}}, P_x)$	This is the matrix version of above.	**1D case:** Equation $p_x(x) = c \cdot e^{-\frac{(x-\hat{x})^2}{2 \cdot \sigma_x^2}}$ Parameters $\hat{x}$: expected value (scalar) σ_x^2 : variance (scalar) Visualization:  **2D case:** Equation $p_{\mathbf{X}}(\mathbf{X}) = c \cdot e^{-\frac{1}{2}(\mathbf{X}-\hat{\mathbf{X}})^T \cdot P^{-1} (\mathbf{X}-\hat{\mathbf{X}})}$ Parameters $\hat{\mathbf{x}}$: expected value (vector 2x1) $\mathbf{P}_x$: covariance matrix (matrix 2x2) Visualization:  **n-D case:** Equation $p_{\mathbf{X}}(\mathbf{X}) = c \cdot e^{-\frac{1}{2}(\mathbf{X}-\hat{\mathbf{X}})^T \cdot P_x^{-1} (\mathbf{X}-\hat{\mathbf{X}})}$ Parameters $\hat{\mathbf{x}}$: expected value (vector n × 1) $\mathbf{P}_x$: covariance matrix (matrix n × n) Visualization: (use imagination)

Obtaining a Marginal PDF	A marginal PDF is a subset of the original PDF, e.g. for just one of the states.
	
	E.g. if $\boldsymbol{x} \sim p(\boldsymbol{x}) = N(\hat{\boldsymbol{x}}, P_x)$, then the marginal PDF about the random variable x_2 is:
	- $p_{x_2}(x_2) = N\left(\hat{x}_2, P_{x_2}\right)$
	or more specifically:
	- $\hat{x}_2 = \hat{x}(2)$ (the second element in the matrix)
	- $P_{x_2} = P_x(2,2)$ (the element in the second row and second column)
	
	The marginal PDF about the random variables x_2 and x_4 is:
	- $\hat{w} = \hat{x}([2,4])$ (a two element vector, consisting of the second and fourth element of $\hat{x}$)
	- $P_w = P_x([2,4],[2,4])$ a 2x2 matrix consisting of the elements in the second and fourth rows and columns.
		given the RV $\mathbf{x} = [x_1 \quad x_2 \quad x_3 \quad x_4]^T \sim N(\hat{\mathbf{x}}, \mathbf{P}_x)$, $\hat{\mathbf{x}} = \begin{bmatrix} 7 \\ 5 \\ 6 \\ 12 \end{bmatrix}$, $\mathbf{P}_x = \begin{bmatrix} 20 & 1 & 3 & 2 \\ 1 & 15 & 1 & 3 \\ 3 & 1 & 9 & 4 \\ 2 & 3 & 4 & 10 \end{bmatrix}$
		-- 	
		example 1: $x_2 \sim p_{x_2}(x_2) = N(5,15)$
		-- 	
		example 2: $\mathbf{w} = \begin{bmatrix} x_2 \\ x_4 \end{bmatrix}$, $\mathbf{w} \sim p_{\mathbf{w}}(\mathbf{w}) = N(\hat{\mathbf{w}}, \mathbf{P}_w)$?
		$\Downarrow$
		$\hat{\mathbf{w}} = \hat{\mathbf{x}}([2,4])$; $\mathbf{P}_w = \mathbf{P}_x([2,4],[2,4])$; // using MATLAB notation and indexes
		$\hat{\mathbf{w}} = \hat{\mathbf{x}}([2,4]) = \begin{bmatrix} 5 \\ 12 \end{bmatrix}$; $\mathbf{P}_w = \mathbf{P}_x([2,4],[2,4]) = \begin{bmatrix} 15 & 3 \\ 3 & 10 \end{bmatrix}$

Understanding P:
- diagonal is the variance of each variable
- off-diagonal values are "how much x1 depends on x2" for example
    - cross-correlations

Transformations:
For two random variables "x" and "y", and constant matrix A and constant vector b

Transformation	Expected value	Covariance matrix P	Matlab Code

$z = x + y$	$\hat{z} = \hat{x} + \hat{y}$	$P_z = P_x + P_y$	P_x = [0 0; 0 0]
			P_y = [0 0; 0 0]
			P_z = P_x + P_y

$z = A * x + b$	$\hat{z} = A * \hat{x} + b$	$P_z = A * P_x * A^T$	
			A = [1, 2; 3, 4]
			b = [1, 2]

			x = 0;
			P_x = [0 0; 0 0];
			
			z = A*x + b;
			P_z = A*P_x*A';

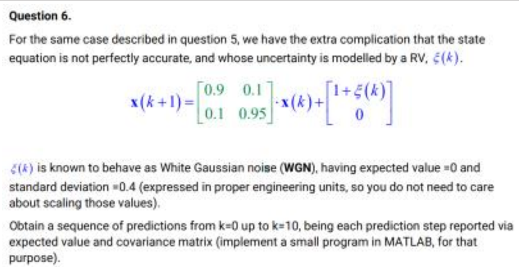
$z = a(x)$
(nonlinear transformation)
	$\hat{z} = a(\hat{x})$	$P_z = A * P_x * A^T$ where $A = \text{(jacobian matrix)}$
		$\begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_1}{\partial x_2} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \frac{\partial a_2}{\partial x_2} & \cdots & \cdots & \frac{\partial a_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial a_m}{\partial x_1} & \cdots & \cdots & \frac{\partial a_m}{\partial x_n} \end{bmatrix}$
		evaluated at $x = \hat{x}$
			A = jacobian(x)
			P_z = A*P_x*A'
			
			function J = MyJacobian(x)
			    J = [0 0; 0 0] % Depends on your transform
			end

$z = A * x + y + b$	$\hat{z} = A * \hat{x} + \hat{y} + b$	$P_z = A * P_x * A^T + P_y$	

Notes:
- the nonlinear transformation case is based on the linear approximation
  - $$\left(\begin{array}{l} \text{this is based on the linear approximation} \\ \mathbf{a}(\mathbf{x}) \cong \mathbf{a}(\hat{\mathbf{x}}) + \left[\frac{\partial \mathbf{a}}{\partial \mathbf{x}} \right]_{\mathbf{x}=\hat{\mathbf{x}}} \cdot (\mathbf{x} - \hat{\mathbf{x}}) \end{array} \right)$$
  - (approximate about the expected value)

Examples:

Question	Solution	Explanation

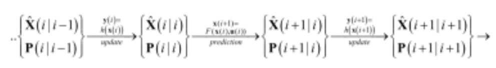
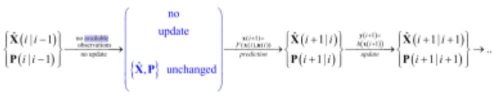
 **Question 6.**
For the same case described in question 5, we have the extra complication that the state equation is not perfectly accurate, and whose uncertainty is modelled by a RV, $\xi(k)$.
$\mathbf{x}(k+1) = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.95 \end{bmatrix} \cdot \mathbf{x}(k) + \begin{bmatrix} 1+\xi(k) \\ 0 \end{bmatrix}$
$\xi(k)$ is known to behave as White Gaussian noise (**WGN**), having expected value =0 and standard deviation =0.4 (expressed in proper engineering units, so you do not need to care about scaling those values).
Obtain a sequence of predictions from k=0 up to k=10, being each prediction step reported via expected value and covariance matrix (implement a small program in MATLAB, for that purpose).
	P = [[0.1, 0]; [0,0]]; % initial covariance matrix of X
	Xe = [0;1]; % initial expected value of X
	P_error=0.4^2; %
	B=[1;0]; % Error only effects x(1)
	Q = B*P_error*B'; % Covariance matrix of error signal
	A = [[0.9,0.1]; [0.1, 0.95]]; %
	b = [1; 0]; % x(k+1) = A*x + b + B*error
	% Get predictions:
	N=10;
	for i=1:N
	    P=A*P*A'+Q; % new covariance matrix
	    Xe = A*Xe+[1;0] ; % new expected value.
	end
		- This is both addition and multiplication transforms
		  • x and error (ξ) are random variables
		- standard deviation = 0.4 so variance = 0.4^2
		  • i.e. $P_\xi = P_{error} = 0.4^2$
		
		- "Q" refers to the covariance matrix for noise added at each step. In this case the noise only effects state 1, so we use B = [1;0] and Q=B*pe*B' to get Q = [pe, 0; 0, 0]

Kalman Filter (EKF + normal)

Thursday, 25 April 2024 5:40 PM

The only difference between the EKF and traditional KF is that the EKF does not have h_expected = H*x_expected.
- Hence, z = y_meas - h_expected and NOT z = y_meas - H*x

Description	Explanation	Image	Pseudocode

High level sequence of steps	1. Start with an initial state estimate X and covariance matrix P
	2. For each iteration:
	  1. update X and P using the kinematic model ("Prediction step")
	  2. if measurement available, update X and P using y_meas, H and R. ("Update step")
		
		We never stop, even if sometimes there are no observations (no update)
		
		
		e.g.
		Full sequence of prediction and update steps.
		initialize, at k=0: $\hat{x}(0\|0)=3,\ \mathbf{P}(0\|0)=0.05^2$
		k=1 : do prediction (use proces model, inputs values) $\{\hat{x}(1\|0),\ \mathbf{P}(1\|0)\}$
		k=1 : if observations are available $\Rightarrow$ perform update : $\{\hat{x}(1\|1),\ \mathbf{P}(1\|1)\}$
		else keep current values of parameters
		"go to sleep for τ"
		wake up!
		k=2 : do prediction $\{\hat{x}(2\|1),\ \mathbf{P}(2\|1)\}$
		k=2 : if observations are available $\Rightarrow$ perform update : $\{\hat{x}(2\|2),\ \mathbf{P}(2\|2)\}$
		else keep current values of parameters
		go to sleep for τ
		keep repeating k>2 ("ad infinitum")
		so that always, at present time k, we have $\{\hat{x}(k\|k),\ \mathbf{P}(k\|k)\}$
			for k = 1:k_max
			  % Run EKF estimator:
			  [X_est(:,k+1), Px(:,:,k+1)] = EKF_Prediction(X_est(:,k), Px(:,:,k), U_meas(k), Pu, dt, params_real);
			
			  if is_data_available
			    H = [0, 1, 0]; % h(X) = X(2) in this case
			    z = y_meas - h_expected;
			    [X_est(:,k+1), Px(:,:,k+1)] = EKF_Update(X_est(:,k+1), Px(:,:,k+1), H, R, z);
			  end
			end

Prediction step	X is updated normally, as per the kinematic model.
	
	P is updated using the jacobians of the model (df/dx and df/du) and the current P
		prediction step: $\hat{\mathbf{x}}(k+1\|k)=\mathbf{f}\left(\hat{\mathbf{x}}(k\|k),\breve{\mathbf{u}}(k)\right)$
		$\mathbf{P}(k+1\|k)=\mathbf{J}\cdot\mathbf{P}(k\|k)\cdot\mathbf{J}^T+\mathbf{Q}_\mathbf{u}$
		$\mathbf{Q}_\mathbf{u}=\mathbf{J}_\mathbf{u}\cdot\mathbf{P}_\mathbf{u}\cdot\mathbf{J}_\mathbf{u}^T$
		in which
		$\breve{\mathbf{u}}(k)$: measured/known input value
		$\mathbf{P}_\mathbf{u}$: covariance matrix of noise which affects inputs
		$\mathbf{J}=\left[\frac{\partial\mathbf{f}(\mathbf{x},\mathbf{u})}{\partial\mathbf{x}}\right]_{\substack{\mathbf{x}=\hat{\mathbf{x}}(k\|k)\\ \mathbf{u}=\breve{\mathbf{u}}(k)}}$
		$\mathbf{J}_\mathbf{u}=\left[\frac{\partial\mathbf{f}(\mathbf{x},\mathbf{u})}{\partial\mathbf{u}}\right]_{\substack{\mathbf{x}=\hat{\mathbf{x}}(k\|k)\\ \mathbf{u}=\breve{\mathbf{u}}(k)}}$
			function [X2, P2] = EKF_Prediction(X1, P1, U1, P_u, dt, params)
			  J = MyJacobian_dfdx(X1, U1, dt, params);
			  J_u = MyJacobian_dfdu(X1, U1, dt, params);
			  Qu = J_u*P_u*J_u';
			  P2 = J*P1*J' + Qu; % Here Q = Qu only since the model doesnt add any other uncertainty.
			Otherwise Q = Qu + Qmodel
			  X2 = model_f(X1, U1, dt, params);
			end

Update step	- H is the jacobian of h(x)
	- h(x) = the expected value of the output variable
	- y_meas = the measured value of the output variable
	  • can be a transformed version of directly measured output data
	- R = covariance matrix of y_meas
		Relevant "actors" in the Update step
		$\{\hat{\mathbf{x}}(k\|k-1),\mathbf{P}(k\|k-1)\}$ {expected value and , covariance matrix of PRIOR}
		$\hat{\mathbf{x}}(k\|k-1)$: expected value of $\mathbf{x}(k)$ before update
		$\mathbf{P}(k\|k-1)$: covariance matrix before update
		$\mathbf{H}\cdot\hat{\mathbf{x}}(k\|k-1)$ expected measurement of output variable
		(the value of the output variable, $\mathbf{y}(k)$, if $\mathbf{x}(k)$ was $=\hat{\mathbf{x}}(k\|k-1)$, based on our assumed output model $\mathbf{y}(k)=\mathbf{H}\cdot\mathbf{x}(k)$)
		$\mathbf{y}_{measurement}(k)$: actual measurement of output variable
		(affected by sensor noise and other uncertainties)
		$\mathbf{R}(k)$: covariance matrix
		of the uncertainty that pollutes the measurement of the output variable
		$\{\hat{\mathbf{x}}(k\|k),\mathbf{P}(k\|k)\}$ {expected value and , covariance matrix of POSTERIOR}
		$\hat{\mathbf{x}}(k\|k)$: expected value of $\mathbf{x}(k)$ as result of the update
		$\mathbf{P}(k\|k)$: covariance matrix as result of the update
		
		$\mathbf{z}(k)=\mathbf{y}_{measurement}(k)-\mathbf{H}\cdot\hat{\mathbf{x}}(k\|k-1)$
		$\mathbf{S}=\mathbf{H}\cdot\mathbf{P}(k\|k-1)\cdot\mathbf{H}^T+\mathbf{R}(k)$
		$\mathbf{K}(k)=\mathbf{P}(k\|k-1)\cdot\mathbf{H}^T\cdot\mathbf{S}^{-1}$
		$\hat{\mathbf{x}}(k\|k)=\hat{\mathbf{x}}(k\|k-1)+\mathbf{K}(k)\cdot\mathbf{z}(k)$
		$\mathbf{P}(k\|k)=$
		$=\mathbf{P}(k\|k-1)-\mathbf{P}(k\|k-1)\cdot\mathbf{H}^T\cdot\mathbf{S}^{-1}\cdot\mathbf{H}\cdot\mathbf{P}(k\|k-1)$
		NOTE: z = y_meas - h(x)
		  - h(x) is the same as H*x only when h(x) is linear!
			function [X2, P2] = do_EKF_update(X1, P1, landmark_measured_LiDAR_CF, landmark_known_GCF, R, lidar_offset, handleLidarPlots)
			  h_expected = landmarkGCFtoLidarCF_h(landmark_known_GCF, X1, lidar_offset, handleLidarPlots);
			  y_meas = landmark_measured_LiDAR_CF;
			  H = MyJacobian_H_dhdx(X1, landmark_known_GCF);
			
			  set(handleLidarPlots(7), 'xdata', h_expected(1,:), 'ydata', h_expected(2,:))
			  set(handleLidarPlots(5), 'xdata', y_meas(1,:), 'ydata', y_meas(2,:))
			  z = y_meas - h_expected;
			  [X2, P2] = EKF_Update(X1, P1, H, R, z);
			end
			
			function [X2, P2] = EKF_Update(X1, P1, H, R, z)
			  % X1: current expected value (PRIOR)
			  % P1 : current covariance matrix (PRIOR)
			  % H : output model is y=H*X (linear)
			  % R : covariance matrix of noise polluting output measurements/
			  % z : ym - h(x), the difference between measurement of output y and
			  % estimate of output y (y_est = h(X1))
			
			    S=H*P1*H'+R;
			    Si=inv(S);
			    K = P1*H'*Si;
			
			    X2 = X1+K*z; %updated expected value
			    P2=P1 - P1*H'*Si*H*P1; %updated covariance matrix
			end

Symbols explanation:

Symbol	Meaning
x $\hat{x}$	- State vector - Estimate of the state vector
P	Covariance matrix associated to the expected state values (propagation of initial covariance)
P(k|k-1)	Covariance matrix prior to performing EKF update step (for the estimated state values)
P(k|k)	Covariance matrix after performing EKF update step (for the estimated state values)
Q	Covariance matrix of noise which is added at every time step we refer to $\mathbf{P}_{\xi(k)}$ using the name $\mathbf{Q}(k)$ $(\ \xi(k)\sim N(\mathbf{0},\mathbf{Q}(k))\ )$ Q_u = due to sensor inaccuracies (inputs) Q_other = due to inaccuracies in the model (per time step)
$y_{meas}(k)$	measurement of the **output variable**, at time step (k). This does not have to be the directly measured data, but can be a transformation of the measured data (e.g. a coordinate frame transformation).
R(k)	Covariance matrix for the measured output (y_meas). If y_meas is derived using a transformation, the covariance matrix R needs to be derived using the same transformations (I think)
h(x)	a function which transforms the current estimated state ($\hat{x}$) to the **output variable** (expected output variable value "h_expected").
H	Jacobian matrix of the model's output equation h(x). i.e. $\Big[\ \partial a\quad \partial a_1\quad \cdots\quad \partial a$

	$\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_1}{\partial x_2} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \frac{\partial a_2}{\partial x_2} & \cdots & \cdots & \frac{\partial a_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial a_m}{\partial x_1} & \cdots & \cdots & \frac{\partial a_m}{\partial x_n} \end{bmatrix}$ where a(x) = h(x). (note the da2/dx2 should be da2/dx1) I.e. $\mathbf{H} = \left. \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} \right\|_{\mathbf{x}=\hat{\mathbf{x}}}$ - evaluated at the current estimate of X (after the prediction step)
ξ	error
J	Jacobian matrix $\mathbf{J} = \left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} \right]_{\mathbf{x}=\hat{\mathbf{x}}(k\|k),\mathbf{u}=\mathbf{u}(k)}$ (jacobian matrix)
$\eta(k)$	error due to sensor noise and inaccuracies in output model $\mathbf{y}_{measurement}(k) = \mathbf{h}(\mathbf{x}(k)) + \boldsymbol{\eta}(k)$ - assume WGN, with known variance R(k)
z	difference between measured output and expected output. Also known as the "innovation"
z(k)	difference between measured output and expected (prior to EKF correction) output. Also known as the "innovation" - $z(k) = y_{meas}(k) - h\left(\hat{x}(k\|k-1)\right)$

Optimisers

Thursday, 25 April 2024 7:52 PM

General background:

- The goal of optimisation is to minimise a cost function (a 1D function which depends on the values of an X vector input)

fminsearch:

- minimise the difference between the expected and measured output, to estimate the current state vector.
- i.e. the cost function is norm(expected output - measured output) (order doesn't matter due to the norm())

example:

```
% This file estimates the pose via optimisation, given a set of detected
% landmarks and their associated known locations. Valid if at least 2
% landmark locations are given.
% by Alex Hunter z5312469 16/03/2024
function [epose, valid] = EstimatePoseD(Useful_OOIs, AssociatedLandmarks, currPose)
    [~, numUsefulOOIs] = size(Useful_OOIs);
    if (numUsefulOOIs < 2) % At least two landmarks required
        valid = 0;
        epose = [0;0;0];
        return
    else
        valid = 1;
    end
    poseTransform = fminsearch(@(poseTransform)costFunction(Useful_OOIs, AssociatedLandmarks, poseTransform), 
[0;0;0]);
    R = rot(poseTransform(3));
    T = poseTransform(1:2);
    epose = [R*currPose(1:2) + T; currPose(3) + poseTransform(3)];

end
function cost = costFunction(Useful_OOIs, AssociatedLandmarks, transformPose)
    T = transformPose(1:2);
    alpha = transformPose(3);
    R = rot(alpha);
    OOIs_in_GCF = R*Useful_OOIs + T;
    cost = norm(OOIs_in_GCF - AssociatedLandmarks)^2;
end
```

- fminsearch only optimises a single input variable, so we need to use this lambda function format in order to use constant variable inputs as well.
  - i.e. @(poseTransform)... intends to optimise the poseTransform state.
  - anonymous functions have the form "handle = @(argument) argument.^2" where the output is argument.^2 here

Particle swarm optimisation:

State update		$\mathbf{X}_{i,n+1} = \mathbf{X}_{i,n} + \mathbf{V}_{i,n+1},$ $(1 \le j \le N)$

Velocity update		$\mathbf{V}_{i,n+1} = \mathbf{V}_{i,n} + c_1 \cdot r_{i,n} \cdot (\mathbf{P}_{i,n} - \mathbf{X}_{i,n}) + c_2 \cdot R_{i,n} \cdot (\mathbf{G}_n - \mathbf{X}_{i,n}),$ (velocity of particle #i at iteration $n+1$, $\mathbf{V}_{i,n+1}$) $r_{i,n} \sim U(0,1)$, $R_{i,n} \sim U(0,1)$ (random coefficients) c_1, c_2 : constants ("Acceleration constants") $\mathbf{V}_{i,n}$: previous velocity , known as the "inertia" part $c_1 \cdot r_{i,n} \cdot (\mathbf{P}_{i,n} - \mathbf{X}_{i,n})$: The "cognition" part $c_2 \cdot R^j_{i,n} \cdot (\mathbf{G}_n - \mathbf{X}_{i,n})$: The "social" part
personal best update		$\mathbf{P}_{i,n} = \begin{cases} \mathbf{X}_{i,n}, & f(\mathbf{X}_{i,n}) < f(\mathbf{P}_{i,n-1}) \\ \mathbf{P}_{i,n-1}, & f(\mathbf{X}_{i,n}) \geq f(\mathbf{P}_{i,n-1}), \end{cases}$
global best update		$\mathbf{G}_n = \mathbf{P}_{g,n}$ $g = \text{argmin}_{1 \leq j \leq M} \{ f(\mathbf{P}_{j,n}) \}.$

Swarm size (number of particles)	M = 10 + 2*sqrt(D)
	or
	M = 40	

Weighted inertia variant		$\mathbf{V}_{i,n+1} = w_n \cdot \mathbf{V}_{i,n} + c_1 \cdot r_{i,n} \cdot (\mathbf{P}_{i,n} - \mathbf{X}_{i,n}) + c_2 \cdot R^j_{i,n} \cdot (\mathbf{G}_n - \mathbf{X}_{i,n}),$ $w_n = \frac{(w_{initial} - w_{final}) \cdot (n_{max} - n)}{n_{max}} + w_{final},$

algorithm	- for L iterations
	  ○ for each particle
	    ▪ evaluate cost
	    ▪ if position is a new best, save new best
	  ○ update global best
	  ○ exit if global best is good enough
	  ○ for each particle
	    ▪ update velocity and position
	- solution = global best
		P=particle initialization();
		for i =1 to L //max num iterations
		{ for each particle p_k in P //P: population of M particles)
		    { do cost= F(p_k.X) //evaluate function at that point.
		        If cost < p_k.MyBestCost
		        { p_k.bestX =p_k.X; p_k.MyBestCost=cost ; }
		        }
		    gbest.X : choose based on best p_k.MyBestCost fpr all p_k in P.
		
		    If (gBest.X)<tolerance) // gbest is good enough, done!
		    { solution = gbest.X;
		        END NOW
		    }
		    for each particle p_k in P do
		    { update p_k.V //following PSO equation
		        update p_k.X
		    }
		}
		//end of iteration loop (L iterations) //max num iterations reached,
		solution = gbest.X ;

(old) Basic kalman filter

Thursday, 4 April 2024 10:24 AM

Prediction step: (regular kalman filter, linear outputs only)

$$z(k) = y_{meas}(k) - h(x)\Big|_{x=\hat{x}(k|k-1)}$$

$$H = \left[\frac{\partial h(x)}{\partial x}\right]\Bigg|_{x=\hat{x}(k|k-1)}$$

$$S = H * P(k|k-1) * H^T + R(k)$$
$$K(k) = P(k|k-1) * H^T * S^{-1}$$
$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k) * z(k)$$
$$P(k|k) = P(k|k-1) - P(k|k-1) * H^T * S^{-1} * H * P(k|k-1)$$
$$P(k|k) = P(k|k-1) - K(k) * H * P(k|k-1)$$

or

$$\boldsymbol{\hat{x}(k|k) = \hat{x}(k|k-1) + \left(P(k|k-1) * H^T * \left(H * P(k|k-1) * H^T + R(k)\right)^{-1}\right) * z(k)}$$

$$\boldsymbol{P(k|k)} = \boldsymbol{P(k|k-1) - P(k|k-1) * H^T} * (H * P(k|k-1) * H^T + R(k))^{\boldsymbol{-1}} * \boldsymbol{H} * \boldsymbol{P(k|k-1)}$$

In matlab:

```matlab
% KF update, for a LINEAR Output equation
% there are diverse ways to perform it, THIS is ONE of them.
function [X_updated,P_updated]=PerformUpdate(ym,H,R,Xe,P)
% ym : measurement of output y
% H  : output model is y=H*X  (linear)
% Xe: current expected value (PRIOR)
% P : current covariance matrix (PRIOR)
% R : covariance matrix of noise polluting output measurements/
% Returned variables.
% [Xu,Pu]: updated expected value and covariance matrix.
%    implement KF update equations, here
   ye=H*Xe ; %expected output
   z=ym-ye;
   S=H*P*H'+R;
   Si=inv(S);
   K = P*H'*Si;

   X_updated = Xe+K*z;        %updated expected value
   P_updated=P- P*H'*Si*H*P;  %updated covariance matrix
end
```

(old) Extended Kalman Filter

Wednesday, 3 April 2024 7:06 PM

Use EKF when the model is nonlinear. I.e.
$\boldsymbol{x}(k+1\mid k) = f(\boldsymbol{x}(k), \boldsymbol{u}(k)) + \xi(k)$

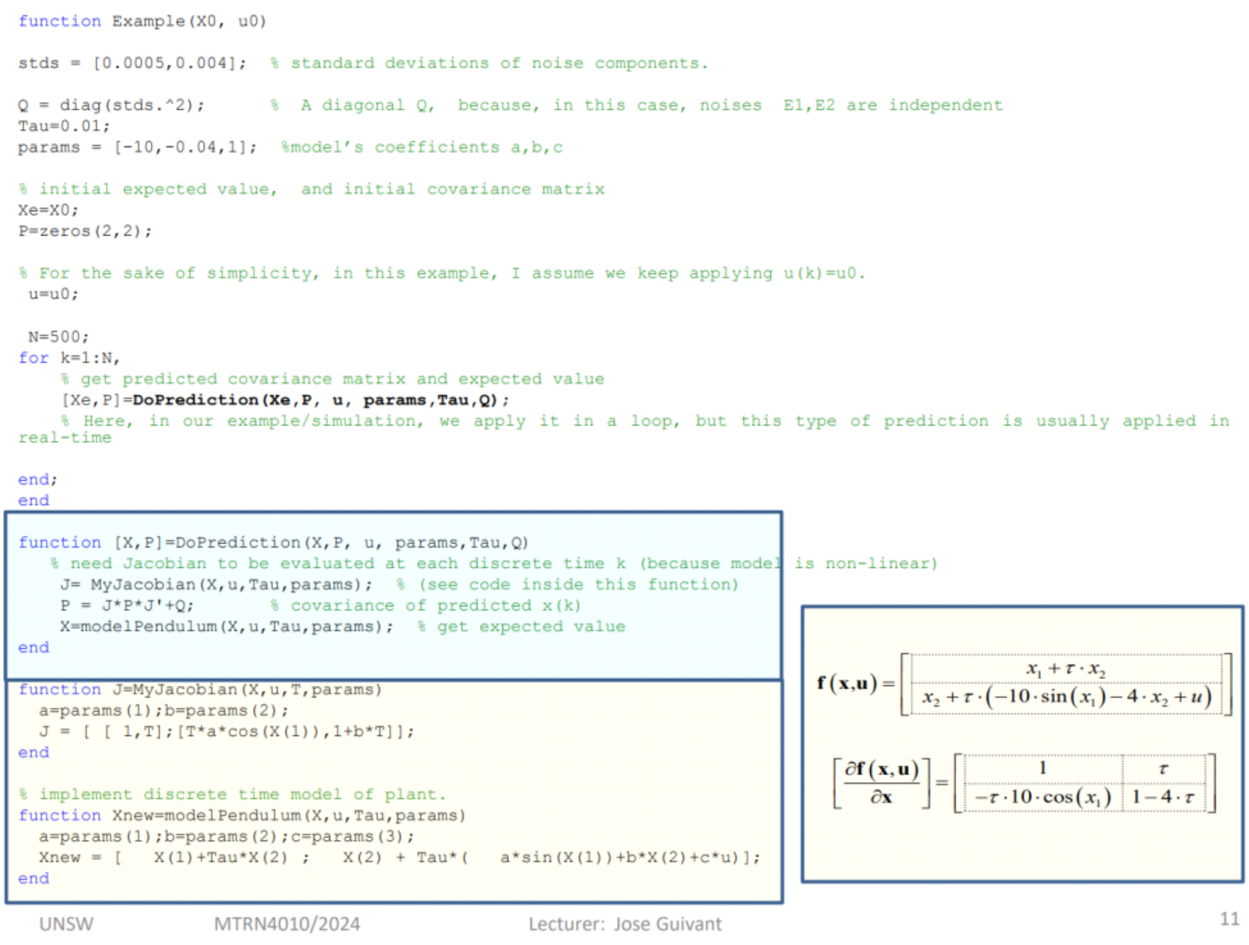
Prediction Correction Step	$\hat{\boldsymbol{x}}(k+1\mid k) = f\left(\hat{\boldsymbol{x}}(k\|k), \breve{\boldsymbol{u}}(k)\right) + \xi(k)$ $P(k+1\mid k) = J^*P(k\|k)^*J^T + Q(k)$ where: - $J = jacobian\ of\ f(x,u)$, evaluated at: ○ $x = \hat{x}(k\|k)$ ▪ $\hat{x}(k\|k)$ is given by $x(k+1) = f(x(k), u(k)) + \xi(k)$ ▪ *i.e.* $\hat{x}(k\|k) = f(x(k-1), u(k-1)) + \xi(k)$ ▪ i.e. after prediction step ○ $u = u(k)$ - $\breve{u}(k) = measured/known\ input\ value$ - $Q = covariance\ matrix\ of\ error\ added\ at\ each\ step$ - $Q_u = covariance\ matrix\ of$	**prediction step:** $\hat{\mathbf{x}}(k+1\|k) = \mathbf{f}(\hat{\mathbf{x}}(k\|k), \breve{\mathbf{u}}(k))$ $\mathbf{P}(k+1\|k) = \mathbf{J}\cdot\mathbf{P}(k\|k)\cdot\mathbf{J}^T + \mathbf{Q}_\mathbf{u}$ $\mathbf{Q}_\mathbf{u} = \mathbf{J}_\mathbf{u}\cdot\mathbf{P}_\mathbf{u}\cdot\mathbf{J}_\mathbf{u}^T$ in which $\breve{\mathbf{u}}(k)$: measured/known input value $\mathbf{P}_\mathbf{u}$: covariance matrix of noise which affects inputs $\mathbf{J} = \left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right]_{\substack{\mathbf{x}=\hat{\mathbf{x}}(k\|k)\\ \mathbf{u}=\breve{\mathbf{u}}(k)}}$ $\mathbf{J}_\mathbf{u} = \left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\right]_{\substack{\mathbf{x}=\hat{\mathbf{x}}(k\|k)\\ \mathbf{u}=\breve{\mathbf{u}}(k)}}$ -J and J_u are jacobian matrixes

recall the Jacobian matrix:

$$\begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_1}{\partial x_2} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \frac{\partial a_2}{\partial x_2} & \cdots & \cdots & \frac{\partial a_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial a_m}{\partial x_1} & \cdots & \cdots & \frac{\partial a_m}{\partial x_n} \end{bmatrix}$$

- but replace "a" with "f"

Matlab Example:



```
function Example(X0, u0)

stds = [0.0005,0.004];  % standard deviations of noise components.

Q = diag(stds.^2);       %  A diagonal Q,  because, in this case, noises  E1,E2 are independent
Tau=0.01;
params = [-10,-0.04,1];  %model's coefficients a,b,c

% initial expected value,  and initial covariance matrix
Xe=X0;
P=zeros(2,2);

% For the sake of simplicity, in this example, I assume we keep applying u(k)=u0.
 u=u0;

 N=500;
for k=1:N,
    % get predicted covariance matrix and expected value
    [Xe,P]=DoPrediction(Xe,P, u, params,Tau,Q);
    % Here, in our example/simulation, we apply it in a loop, but this type of prediction is usually applied in
real-time

end;
end
```

```
function [X,P]=DoPrediction(X,P, u, params,Tau,Q)
   % need Jacobian to be evaluated at each discrete time k (because model is non-linear)
    J= MyJacobian(X,u,Tau,params);  % (see code inside this function)
    P = J*P*J'+Q;        % covariance of predicted x(k)
    X=modelPendulum(X,u,Tau,params);  % get expected value
end
```

```
function J=MyJacobian(X,u,T,params)
  a=params(1);b=params(2);
  J = [ [ 1,T];[T*a*cos(X(1)),1+b*T]];
end

% implement discrete time model of plant.
function Xnew=modelPendulum(X,u,Tau,params)
  a=params(1);b=params(2);c=params(3);
  Xnew = [   X(1)+Tau*X(2)  ;    X(2)  + Tau*(    a*sin(X(1))+b*X(2)+c*u)];
end
```

$$\mathbf{f}(\mathbf{x},\mathbf{u}) = \begin{bmatrix} x_1 + \tau \cdot x_2 \\ x_2 + \tau \cdot \left(-10 \cdot \sin(x_1) - 4 \cdot x_2 + u\right) \end{bmatrix}$$

$$\left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right] = \begin{bmatrix} 1 & \tau \\ -\tau \cdot 10 \cdot \cos(x_1) & 1 - 4 \cdot \tau \end{bmatrix}$$

UNSW MTRN4010/2024 Lecturer: Jose Guivant 11