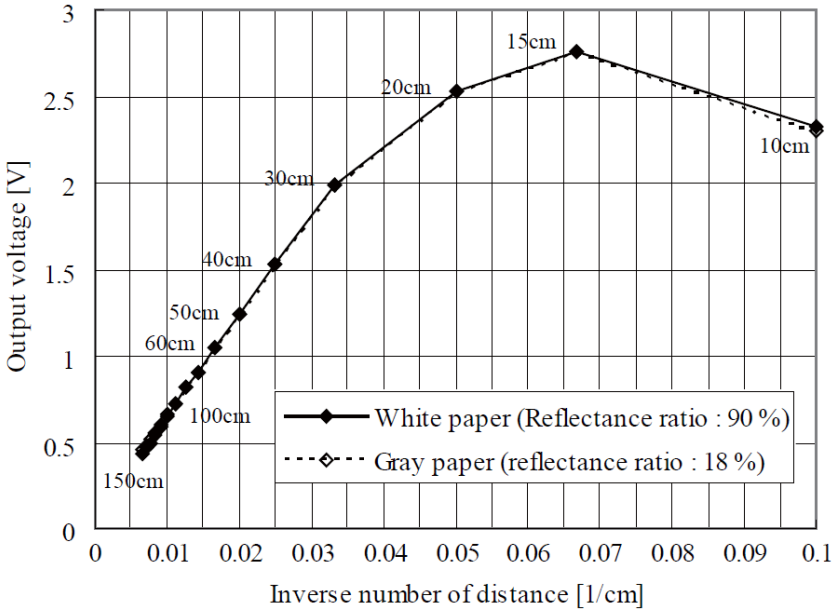
Laboratoire N^{o} 7 : Filtre de Kalman étendu (EKF)

1) Calibration du capteur longue portée Sharp

Prenez une série de mesures séparées de 5 cm, entre 30 cm et 120 cm. Soyez attentifs, car ces mesures serviront à calibrer votre capteur infrarouge et aussi à tester votre filtre de Kalman. Sauvegarder vos données dans un fichier texte (notepad) nommé `calibration.csv`, avec un format comme suit :

```
30,1.89
35,1.66
```

Assurez-vous d'utiliser un point décimal et une virgule comme séparateur.



Comme illustré à l'image ci-haut, le capteur infrarouge fonctionne essentiellement avec l'inverse de la distance. Il sera donc approximé par la fonction suivante :

$$f_{\text{infrarouge}}(x) = K_{1\text{Sharp}} + \frac{K_{2\text{Sharp}}}{x}, \text{ pour } x > 0.30\ m \quad (1)$$

Utilisez le script matlab `CalibrationInfrarouge.m` pour trouver automatiquement les paramètres K_{1Sharp} et K_{2Sharp}, en fonction des mesures enregistrées dans le fichier `.csv`. Vous devriez obtenir des valeurs proches de K_{1Sharp} = 0.06 et K_{2Sharp} = 0.55, et visuellement les cercles et les +rouges devraient être proches. Ce script sauvegardera ces paramètres dans le fichier `CalibSharp.mat`, qui sera utilisé par le script du filtre de Kalman étendu pour les expériences suivantes. Comme cette fonction est bijective, on peut trouver un estimé de distance x directement à partir de la mesure infrarouge z, avec l'inverse de la fonction :

$$f^{-1}_{\text{infrarouge}}(z) = \frac{K_{2\text{Sharp}}}{z - K_{1\text{Sharp}}} \quad (2)$$

Vous aurez besoin de cette fonction pour initialiser l'estimé `X` d'un filtre quand vous ne connaissez pas la position de départ du système. Elle sera aussi utilisée par le script pour comparer directement les mesures du capteur infrarouge avec vos estimés. Notez que ces équations (1) et (2) ne sont valides que pour x > 0.30 *m* : votre robot ne pourra donc jamais être à moins de 0.30 *m* du mur.

2) Modèle du système

Comme estimé de bruit sur le capteur infrarouge, nous allons prendre la valeur de $\sigma^2_{\text{infrarouge}} = (0.02V)^2$, ce qui correspond approximativement à un palier du convertisseur analogique-numérique[1]. Le bruit de déplacement sera $\sigma^2_{\text{pas}} = (0.002\ m)^2$, ce qui donne essentiellement un écart-type de 2 *mm* sur un pas du robot. Fait à noter, le choix de ces valeurs influencera grandement le comportement du filtre. Ainsi, si vous choisissez un estimé de bruit de capteur $\sigma^2_{\text{infrarouge}}$ plus petit, vous indiquez au filtre qu'il doit augmenter sa confiance envers les mesures du capteur. De la même manière, si vous diminuez la valeur estimée du bruit sur les commandes σ^2_{pas}, il augmentera sa confiance envers l'estimé $x(k+1|k)$ obtenu lors de la phase de prédiction. Nous y reviendrons lors des manipulations.

La dynamique du système et la commande sont linéaires pour ce système :

$$x(k+1|k) = x(k) + u(k)$$

Ce qui nous donne les matrices `F` (ou Φ sur les acétates) = [1], et `G` (ou Γ sur les acétates) = [1]. La fonction de mesure (1) est, quant à elle, non-linéaire :

$$f_{\text{infrarouge}}(x) = K_{\text{1Sharp}} + \frac{K_{\text{2Sharp}}}{x}$$

Pour l'utiliser dans un filtre de Kalman, nous allons donc devoir linéariser cette fonction autour du point d'opération *x* (la position estimée du robot). La jacobienne `H` (ou Λ sur les acétates) est utilisé par le filtre de Kalman étendu (EKF) pour effectuer cette linéarisation :

$$H = \left[\frac{\partial}{\partial x} f_{\text{infrarouge}}(x) \right] = \left[\frac{-K_{\text{2Sharp}}}{x^2} \right]$$

Cette matrice `H` (ou Λ sur les acétates) est de taille *n*x*m*, où *n* est le nombre de capteurs et *m* la longueur du vecteur d'état. Pour ce système, sa taille est de 1x1, car nous avons une variable d'état (la position), et un seul capteur (infrarouge Sharp). **Important!** La valeur numérique de `H` est recalculée à chaque itération par le filtre EKF, car la pente de la fonction du capteur dépend de la position *x*. Aussi, le filtre utilise l'estimé `X` pour la calculer, car nous ne connaissons pas la vraie valeur de *x*. C'est d'ailleurs une des sources d'échec du filtre : si `X` et *x* sont très différents, alors la valeur de `H` sera incorrecte, et le filtre risque de diverger. Plus la fonction sera non-linéaire, et plus la distance |`X`-*x*| posera problème.

[1] Le bruit associé à la conversion analogique-numérique est généralement modélisé comme une distribution uniforme entre ±un demi-palier de conversion. Pour les curieux, une discussion complète de ce sujet est disponible sur wikipedia : http://en.wikipedia.org/wiki/Quantization_error .

3) Expérimentation avec les données de calibration

Il est possible d'utiliser un filtre de manière *offline*, c'est-à-dire après que les données soient toutes capturées par le robot. Ceci a l'avantage de faciliter votre familiarisation avec ce filtre. Dans le script `EkfCreateSharp.m`, assurez-vous que la variable `CasSimule` (ligne 44) est à `false`. Assurez-vous aussi que le fichier de données est celui correspondant à la calibration :

```
Data = csvread('calibration.csv');
```

Les valeurs de bruits doivent être les suivantes :

```
SVSharp  = 0.02;   % (V) ecart-type sur voltage du capteur infrarouge
SCreate  = 0.002;  % (m) ecart-type sur un pas du create
```

Et la valeur de la commande (longueur d'un pas) doit être la suivante :

```
u = 0.05; % (m) Longeur d'un pas du robot
```

Faites tourner le script, et observez les différents graphiques générés en sortie. En particulier, vous devriez voir que l'histogramme des erreurs pour les mesures de distances avec le capteur infrarouge est beaucoup plus large que l'histogramme des erreurs du filtre Kalman. Aussi, vous devriez voir que les mesures du capteur infrarouge * sont beaucoup plus dispersé que les estimés du filtre.

4) Saisis de données automatique par déplacement du robot

Dans l'exemple précédent, les données ont été prises en déplaçant manuellement le robot par intervalle de 0.05 *m*. Or, ceci crée une situation particulière : bien qu'il y ait des petites erreurs de positionnement çà et là de 1 ou 2 *mm*, vous n'accumulez pas ces erreurs comme le ferait un robot qui se déplace. Ainsi donc, la distance entre chaque mesure est, en moyenne, très près de u=0.05 *m*, soit la valeur de la commande spécifiée dans le filtre.

Pour les expériences suivantes, nous allons maintenant utiliser les déplacements du robot pour saisir les données. Ceci donnera une situation plus proche de la réalité, où l'on cherche à estimer la position du robot en temps réel pour un robot se déplaçant de façon autonome.

Saisi des données : placez votre robot à la position de départ 0.30 m. Utilisez le script `Robot.m` qui effectue la boucle suivante, pour faire reculer le robot jusqu'à 1.40 m du mur :

```
for i=1:22
     recule le robot de 0.05 m
     capture mesure distance infrarouge
end
sauvegarde toutes les données
```

Assurez-vous que la cible soit assez large et que le robot ne tourne pas trop lorsqu'il se déplace. Saisissez quelques-uns de ces fichiers en vous assurant de :

- noter précisément la position de départ et la position finale du robot;
- changer le nom du fichier à la fin du script entre les saisis de données, pour ne pas écraser les données précédentes.

Nous les utiliserons pour le restant du laboratoire.

5) Post-traitement des données prises par déplacement du robot

Cas 1 : Pour ce cas, nous allons assumons que votre robot connaisse sa position parfaitement au début. Pour refléter cette situation, vous devez donc vous assurer d'initialiser votre filtre de la façon suivante, à la ligne 72 pour la condition `if (iStep == 1)` :

```
X=0.30-u;
P = 0;
```

Il faut soustraire la valeur de `u` dans l'implémentation du filtre dans le script donné, car le code additionnera automatiquement `u` lors de la prédiction dans les lignes suivantes (ligne 99). Une valeur de P=0 indique que vous voulez informer le filtre que vous avez une information parfaite sur la position.

Faites tourner le code du filtre, en vous assurant de changer le fichier de données qui est lu :

```
Data = csvread('votreficher.csv');
```

Quel comportement voyez-vous? En particulier, vous devriez observer que :

- la précision sur la position estimée diminue au fil du temps, ce qui reflète l'accumulation progressive des erreurs;
- la matrice de covariance *P* reflète bien cette situation, en accroissant avec le temps;
- la valeur absolue[2] du gain *K* du filtre augmente au début, car le filtre doit tenir compte de plus en plus de la mesure pour corriger l'estimé de la position à mesure que notre estimé perd de la précision.

Normalement, la valeur de la covariance P et du gain K se stabilise, ce qui n'est pas le cas en ce moment. Pourquoi? Eh bien à mesure que vous vous éloignez, votre capteur perd énormément de précision, et la matrice P et le gain K doivent refléter cette situation. Pour un scénario plus réaliste, le robot resterait toujours dans les mêmes parages, et P et K convergeraient vers une valeur fixe. Rappelez-vous que pour ce système :

$$K = \frac{PH^T}{HPH^T + \sigma^2_{\text{infrarouge}}}$$

Renversez maintenant l'ordre des données, pour faire comme si votre robot s'était rapproché du mur au lieu de s'en éloigner. Vous pouvez renverser un vecteur dans matlab avec `fliplr`. Nous allons activer les lignes suivantes

```
u = -u;
Xcal = fliplr(Xcal')';
Zcal = fliplr(Zcal')';
```

en mettant un `(1)` à la condition `if` à la ligne 38.

Observez les changements sur les matrices P et K, en fonction du temps.

[2] Attention pour ce système, le gain K est négatif. La courbe montera donc une valeur descendante pour une augmentation de la valeur absolue du gain.

Cas 2 : Pour ce cas, nous allons assumer que votre robot ne connaisse pas sa position au début. Pour refléter cette situation, vous devez donc à la première itération sauter l'exécution du filtre de Kalman, et plutôt utiliser la première mesure comme estimé :

$\texttt{X} = f^{-1}_{\text{infrarouge}}(z)$

$\texttt{P} = H^{-1}\sigma^2_{\text{infrarouge}}(H^T)^{-1}$

La valeur de P indique au filtre que la précision de l'estimé est égale à celle du capteur. Pourquoi pré- et post-multiplier par l'inverse de *H*? Parce que $\sigma^2_{\text{infrarouge}}$ est défini dans l'espace des mesures du capteur (en Volt), et qu'il faut donc estimer quelle est la précision de l'estimé de la position (en *m*). Cette opération va faire cette conversion pour nous. La fonction d'inverse dans matlab est `inv`.

Enlever les modifications précédentes (celles qui faisaient tourner les données en sens inverse), et exécuter le filtre à nouveau. Quel comportement voyez-vous? En particulier, vous devriez observer que :

- la matrice de covariance *P* diminue après la première itération, car on a acquis de l'information.

Cas 3 : si la valeur moyenne des pas de votre robot ne correspond pas à `u`, ceci correspond à la situation où le bruit de déplacement est biaisé, ce qui viole les hypothèses d'utilisation de ce type de filtre. Cette situation est très néfaste pour le filtre, qui aura tendance à avoir une erreur beaucoup plus élevée que normalement. Pour ce test, utilisez à nouveau les données du fichier de calibration, car celle-ci est sans biais

```
Data = csvread('calibration.csv');
```

En modifiant légèrement la valeur de `u` dans le début du code du filtre (par exemple le faire passer de 0.05 à 0.0475), regardez comment le filtre accumule une erreur importante : vous verrez les cercles verts qui s'éloignent progressivement de la valeur réelle. À la lumière de cette expérience, croyez-vous qu'il est important de ne pas avoir un système contenant une erreur de commande biaisé?

Cas 3 : faites 8 déplacements de 10 pas de 0.05 m, en utilisant le script `Robot.m`. Partez toujours du même endroit. Mesurez la position finale du robot avec la règle, et notez-là. Calculez l'erreur sur la position finale pour les 8 séries. Quelle est la variance sur cette erreur? Est-elle comparable à la variance estimée par le filtre?

Cas 4: Comparer la simulation avec la réalité de vos données de calibration. Notez d'abord quelle est l'erreur sur la localisation avec le capteur infrarouge (graphe supérieur droit). Puis, activez la simulation avec le drapeau suivant à la ligne 43 :

```
CasSimule = true;
```

Modifier le bruit du capteur `SVSharp` dans le script afin d'avoir un écart-type des erreurs du capteur infrarouge similaire avec vos données réelles. Quelle est la valeur $\sigma^2_{\text{infrarouge}}$ trouvée?