Université de Toulon

IUT de Toulon

Département Génie Electrique et Informatique Industrielle (GEII)

Automatique

Compte Rendu Rapport

écrit le 13 février 2024

par

Bruno HANNA

Encadrant universitaire : Nicolas BOIZOT

Table des matières

Introduction

Les régulateurs PID (Proportionnel, Intégral, Dérivatif) sont au cœur de nombreux systèmes de contrôle automatique. Leur rôle est d'ajuster le signal de commande d'un système afin d'atteindre et de maintenir une valeur désirée, appelée consigne. Le régulateur PID combine trois stratégies de contrôle :

— La composante proportionnelle, qui réagit à l'écart actuel entre la consigne et la mesure.

— La composante intégrale, qui agit sur la somme des écarts passés, permettant de corriger les erreurs persistantes.

— La composante dérivative, qui anticipe les variations futures de l'écart, en se basant sur sa tendance actuelle.

Un exemple concret d'utilisation des régulateurs PID se trouve dans la gestion des métros. Dans ce contexte, le PID permet un contrôle fin de la vitesse et de la position du train, garantissant à la fois sécurité et précision dans le respect des horaires et des arrêts en station. En comparaison, un système de contrôle simpliste avec des boutons 'avancer' et 'stop' ne pourrait ni assurer la précision nécessaire pour l'arrêt en station, ni maintenir une vitesse stable et sûre, ni s'adapter aux variations de charge du train ou aux conditions du réseau. Le régulateur PID, en ajustant continuellement la commande du train, répond de manière optimale aux exigences de sécurité, de confort et d'efficacité du transport urbain.

Boucle Fermée

1.1 Prise de Données

L'organisation des données est essentielle pour les expériences menées en mode boucle fermée pour Position_P et Speed_PI. Les données collectées sont structurées dans des fichiers où chaque colonne correspond à une variable spécifique. La structure est détaillée ci-dessous :

Répertoire	Variable	Numéro	Commande MATLAB
Position_P	Signal de commande (entrée)	2	`Measure(2:end, 1:2)`
Position_P	Position angulaire	3	`Measure(2:end, 1:2:3)`
Position_P	Vitesse de rotation	4	`Measure(2:end, 1:3:4)`
Speed_PI	Signal de commande (entrée)	2	`Measure(2:end, 1:2)`
Speed_PI	Vitesse de rotation	3	`Measure(2:end, 1:2:3)`

Table 1.1 – Organisation des Données de Mesure dans les Fichiers

La sélection et l'analyse des données se basent sur le contenu du répertoire :

```
commandeData = Measure(2:end, 1:2); % Donnees pour l'entree
vitesseData = Measure(2:end, 1:3:4); % Pour la vitesse, colonnes 1 et 3
if contains(selectedFilePath, '\Position')
    positionData = Measure(2:end, 1:2:3); % Pour la position, colonnes 1,2,3
```

Listing 1.1 – Sélection des Données pour Analyse

1.1.1 Calcul des Paramètres K et Kc

La valeur Kc est utilisée pour convertir les données de position en tours par minute (tr/min). La valeur K, quant à elle, représente le ratio de la vitesse moyenne sur la valeur de la commande. Ce ratio est calculé une seconde après le début de l'acquisition pour refléter la réponse du système à la commande en cours.

```
vitesseMoyenne = mean(vitesseData(600:1600, 2)); % Calcul de la moyenne
valeurCommande = commandeData(1300, 2); % Recuperation valeur de commande

kValue = vitesseMoyenne / valeurCommande; % Calcul de K
assignin('base', 'kValue', kValue);

kcValue = 360/60000; % Definition de Kc pour la conversion en tr/min
```

```
assignin('base', 'kcValue', kcValue);
```

Listing 1.2 – Calcul et Assignation de K et Kc

1.2 Modélisation du Système de Contrôle PID

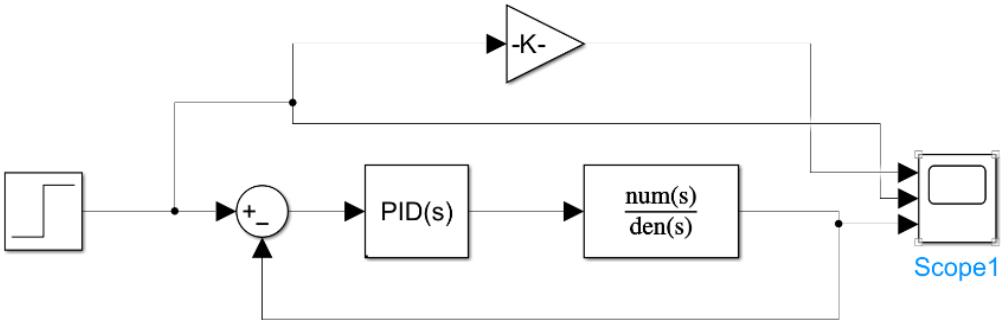


Figure 1.1 – Schéma du Modèle PID

Le schéma illustre un régulateur PID classique, utilisé pour maintenir une grandeur physique à une valeur cible prédéfinie. Le régulateur agit selon trois termes :

— Proportionnel (P) : réagit linéairement à l'écart actuel.
— Intégral (I) : corrige les erreurs cumulées dans le temps.
— Dérivatif (D) : répond à la vitesse de changement de l'écart.

La dynamique du système est décrite par la fonction de transfert :

$$G(s) = \frac{kValue \cdot kcValue}{36s^2 + s}$$

où $kValue$ et $kcValue$ sont des gains ajustables reflétant les propriétés proportionnelles du système, et les termes du dénominateur caractérisent la dynamique intrinsèque, comme l'inertie et la résistance.

Le réglage de ces paramètres détermine la précision et la stabilité de la régulation, avec un objectif de minimiser l'écart par rapport à la consigne sans induire de dépassement significatif.

1.3 Analyse de la Fonction de Transfert en Boucle Fermée

La fonction de transfert en boucle fermée d'un système contrôlé par un régulateur PID est donnée par :

$$\frac{Y(s)}{C(s)} = \frac{H(s)R(s)}{1 + H(s)R(s)} \tag{1.1}$$

où $Y(s)$ est la sortie, $C(s)$ est la commande d'entrée, $H(s)$ est la fonction de transfert du système et $R(s)$ est la fonction de transfert du régulateur.

1.3.1 Régulation de Position Angulaire avec Correcteur Proportionnel

Pour un correcteur proportionnel, la fonction de transfert du système $H(s)$ et la fonction de transfert du régulateur $R_1(s)$ sont définies comme :

$$H(s) = \frac{K_c}{s} \cdot \frac{K}{1+Ts} \tag{1.2}$$

$$R_1(s) = K_p = \frac{U(s)}{E(s)} \tag{1.3}$$

La fonction de transfert en boucle fermée $F_{BF}(s)$ devient alors :

$$F_{BF}(s) = \frac{K_p K_c K}{s + Ts^2 + K_p K_c K} \tag{1.4}$$

L'erreur $E(s)$ est calculée comme :

$$E(s) = \frac{s + Ts^2}{s + Ts^2 + K_c K K_p} \tag{1.5}$$

1.3.2 Régulation de Vitesse avec Correcteur Proportionnel

La fonction de transfert pour la régulation de vitesse avec un correcteur proportionnel devient :

$$H(s) = \frac{K}{1+Ts} \tag{1.6}$$

$$R_1(s) = K_p = \frac{U(s)}{E(s)} \tag{1.7}$$

La fonction de transfert en boucle fermée $F_{BF}(s)$ est :

$$F_{BF}(s) = \frac{K_p K}{1 + Ts + K_p K} \tag{1.8}$$

L'erreur $E(s)$ pour la régulation de vitesse est donnée par :

$$E(s) = \frac{1+Ts}{1 + Ts + K K_p} \tag{1.9}$$

1.3.3 Régulation de Vitesse avec Correcteur Proportionnel-Intégral

Pour un régulateur comprenant des composantes proportionnelle et intégrale :

$$H(s) = \frac{K}{1+Ts} \tag{1.10}$$

$$R_2(s) = K_p + \frac{K_i}{s} = \frac{U(s)}{E(s)} \tag{1.11}$$

La fonction de transfert en boucle fermée $F_{BF}(s)$ est :

$$F_{BF}(s) = \frac{K K_p s + K K_i}{s + Ts^2 + K K_p s + K K_i} \tag{1.12}$$

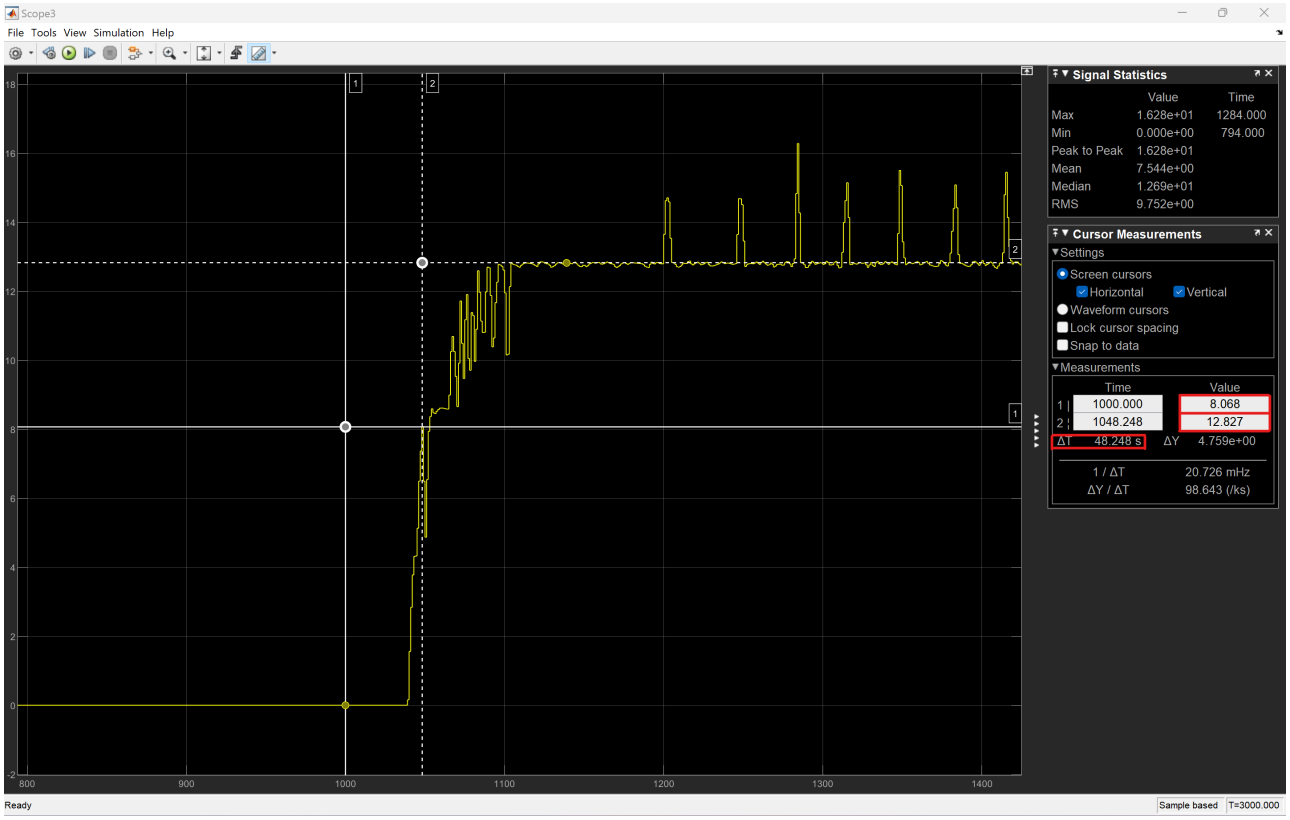


Figure 1.2 – Relevé de la vitesse du système

1.4 Calcul de Kp

La réponse du système atteint 63% de sa valeur moyenne avec un temps de réponse T de 48ms.

La fonction de transfert en boucle fermée est donnée par l'équation (1.4) où K est le gain du système, K_c est le gain du correcteur, et K_p est le gain proportionnel.

Ce qui nous permet d'exprimer la pulsation propre ω_0 et le coefficient d'amortissement ζ :

$$\frac{1}{\omega_0^2} = \frac{T}{K \cdot K_c \cdot K_p} \tag{1.13}$$

$$2\zeta \left(\frac{1}{\omega_0}\right) = \frac{1}{K \cdot K_c \cdot K_p} \tag{1.14}$$

En combinant les équations (1.13) et (1.14), on obtient :

$$4\zeta^2 \cdot \frac{T}{K \cdot K_c \cdot K_p} = \frac{1}{(K \cdot K_c \cdot K_p)^2} \tag{1.15}$$

L'équation (1.15) peut être réécrite pour isoler K_p, avec un coefficient d'amortissement $\zeta = 0.7$ pour un dépassement maximal de 5% comme indiqué dans la consigne :

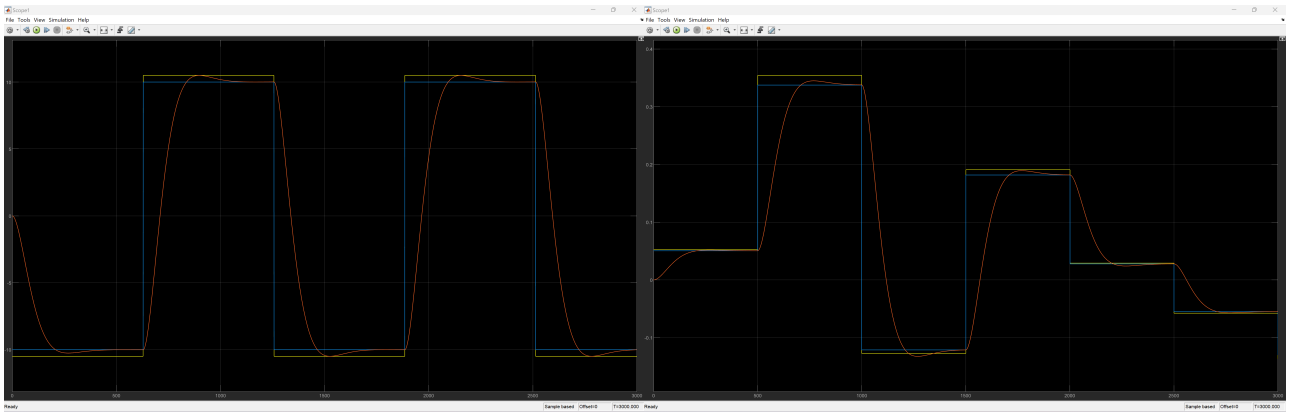
$$\zeta^2 = \frac{1}{4T \cdot K \cdot K_c \cdot K_p} \tag{1.16}$$

$$K_p = \frac{1}{4T \cdot K \cdot K_c \cdot \zeta^2} = 52.75 \tag{1.17}$$

Dans l'équation (1.17), K représente le rapport de la vitesse moyenne sur la valeur de la commande, ce qui reflète l'efficacité du système à suivre la consigne de vitesse et donc pour nous à le simuler sur simulink.

1.5 Résultats de Simulation

Les simulations montrent que notre modèle s'adapte bien à différentes formes de signaux d'entrée avec Kp $= 52.75$ et Ki $= 0.003$.



(a) Réponse du modèle à un signal carré (b) Réponse du modèle à un signal aléatoire

Figure 1.3 – Exemples de réponse du modèle à différentes entrées

Bonus

6 Explication du Script MATLAB

6.1 Recherche des fichiers de données

La procédure de recherche des fichiers est automatisée pour identifier tous les fichiers ‘Measure.txt‘ disponibles dans le répertoire de travail et ses sous-répertoires.

```
% Recherche des fichiers 'Measure.txt'
files = dir('**/Measure.txt');

% Extraction et tri des chemins de dossiers contenant les fichiers
filepaths = {files.folder};
[~, uniqueIdx] = unique(filepaths);
uniqueFolders = filepaths(sort(uniqueIdx));

% Preparation des noms de dossiers pour l'affichage
displayNames = cellfun(@(x) x(find(x=='\', 1, 'last')+1:end), uniqueFolders,
    'UniformOutput', false);
```

Listing 1.3 – Script pour la recherche des fichiers ‘Measure.txt‘

Cette portion de code recherche les fichiers ‘Measure.txt‘ et prépare une liste des dossiers uniques où ces fichiers sont localisés pour les afficher dans l'interface graphique de l'application. La fonction ‘unique‘ est utilisée pour s'assurer que chaque dossier est listé une seule fois, même s'il contient plusieurs fichiers ‘Measure.txt‘.

6.2 Calcul de K

Après avoir sélectionné le fichier de données, le script calcule le gain K du système à partir des données de vitesse et de commande.

```
% Calcul de la moyenne des valeurs de vitesse
vitesseMoyenne = mean(vitesseData(600:1600, 2));

% Recuperation de la valeur de commande
valeurCommande = commandeData(1300, 2);

% Calcul du gain K
kValue = vitesseMoyenne / valeurCommande;
assignin('base', 'kValue', kValue);
```

Listing 1.4 – Calcul du gain K du système

Dans ce bloc de code, ‘vitesseMoyenne‘ est la moyenne des mesures de vitesse dans un intervalle spécifique, et ‘valeurCommande‘ est la valeur de la commande à un moment donné, qui est ensuite assigné au workspace de MATLAB.

6.3 Sélection des paramètres Kp, Ki et Kc

Le script permet également à l'utilisateur d'entrer les valeurs de K_p, K_i, et K_c via l'interface graphique.

```
% Creation des champs d'entree pour Kp, Ki et Kc
hKpEdit = uicontrol('Parent', hFig, 'Style', 'edit', ...);
...

% Recuperation et stockage des valeurs entrees par l'utilisateur
kpValue = str2double(get(hKpEdit, 'String'));
...

% Assignation des valeurs au workspace de MATLAB
assignin('base', 'kpValue', kpValue);
...
```

Listing 1.5 – Saisie des valeurs de Kp Ki et Kc

Ces champs d'entrée graphique recueillent les données saisies qui sont ensuite assignées au workspace de MATLAB. Ces valeurs seront utilisées pour configurer le régulateur PID lors de la simulation du système.

6.4 Visualisation des Données avec MATLAB

Si l'utilisateur choisit de visualiser les données avec MATLAB, le script génère des graphiques pour afficher les données de manière appropriée.

```
function plotWithMATLAB(commandeData, positionData, vitesseData, dossier)
    figure; % Creer une nouvelle figure pour le tracage

    % Determiner le nombre de graphiques a afficher
    if contains(dossier, 'Ouvert')
        % Extraire les donnees specifiques pour le dossier 'Ouvert'
        subplot(3, 1, 1); % Position
        plot(positionData(:,1), positionData(:,2));
        ...
        subplot(3, 1, 2); % Vitesse
        plot(vitesseData(:,1), vitesseData(:,2));
        ...
        subplot(3, 1, 3); % Commande
        plot(commandeData(:,1), commandeData(:,2));
        ...
```

```
    else
        % Utiliser les donnees d'entree et de sortie pour d'autres dossiers
        subplot(2, 1, 1); % Entree
        plot(commandeData(:,1), commandeData(:,2));
        ...
        subplot(2, 1, 2); % Sortie
        plot(vitesseData(:,1), vitesseData(:,2));
        ...
    end
end
```

Listing 1.6 – Visualisation des données avec MATLAB

Ce bloc de code définit une fonction 'plotWithMATLAB' qui crée des graphiques. Les graphiques sont organisés en sous-graphiques ('subplot') au sein d'une fenêtre de figure MATLAB.

6.5 Visualisation des Données avec Simulink

Si l'utilisateur choisit de visualiser les données avec Simulink, le script lance une simulation Simulink avec les données de commande et de vitesse.

```
function displayWithSimulink(commandeData, vitesseData, positionData)
    modelPath = 'affichage'; \% Nom du modele Simulink
    load_system(modelPath);

    % Assigner les variables au workspace du modele Simulink
    assignin('base', 'commandeData', commandeData);
    ...

    % Lancer la simulation Simulink
    set_param(modelPath, 'SimulationCommand', 'start');
    ...

    % Ouvrir l'oscilloscope Scope1
    open_system([modelPath '/Scope1']);
end
```

Listing 1.7 – Visualisation des données avec Simulink

Ce bloc de code définit une fonction 'displayWithSimulink' qui charge un modèle Simulink spécifié, assigne les données nécessaires au workspace du modèle, et lance une simulation. Après la simulation, les oscilloscopes Simulink intégrés sont utilisés pour visualiser les résultats.

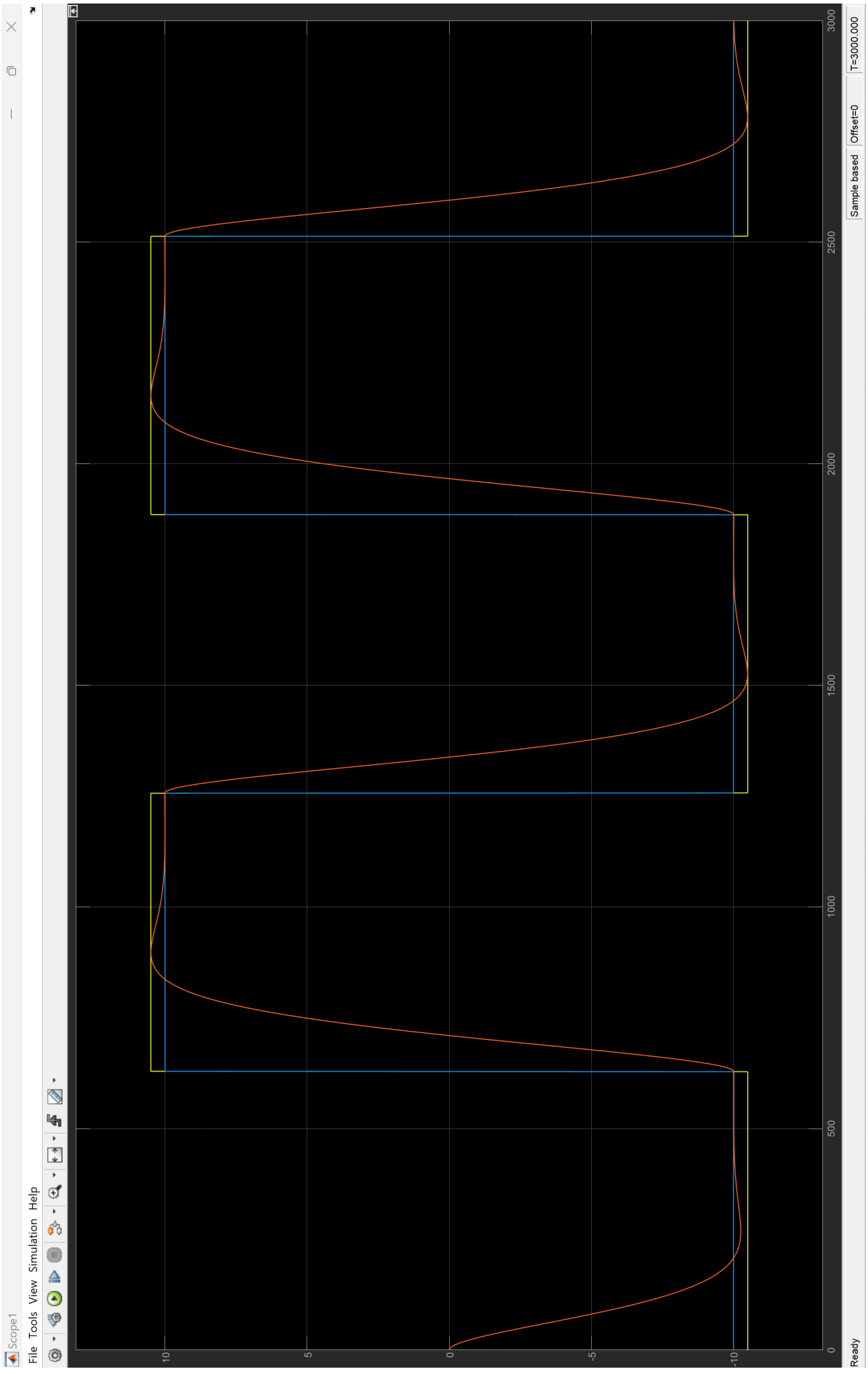


Figure 4 – Simulation avec un signal d'entrée carré

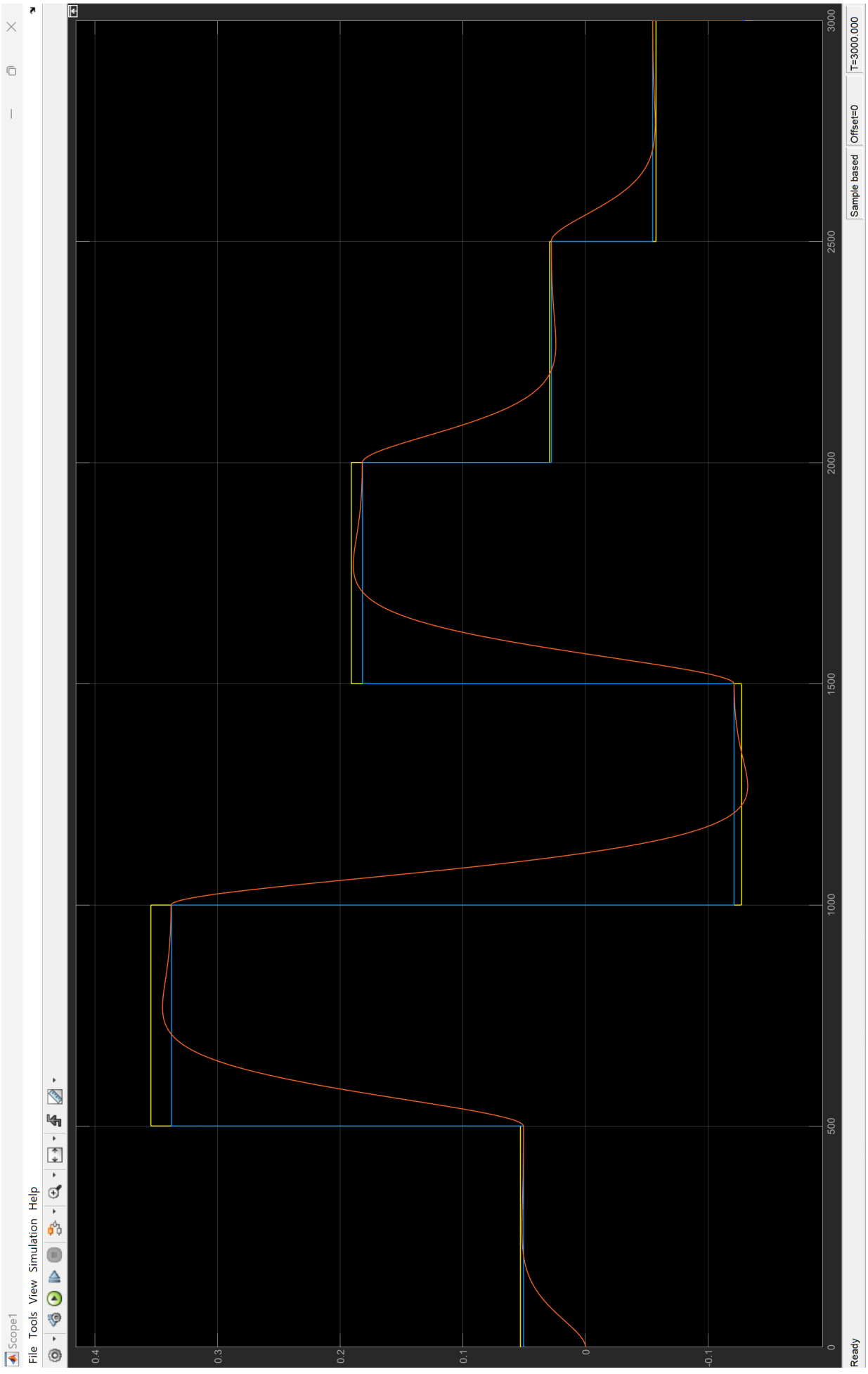


Figure 5 – Simulation avec un signal d'entrée aléatoire

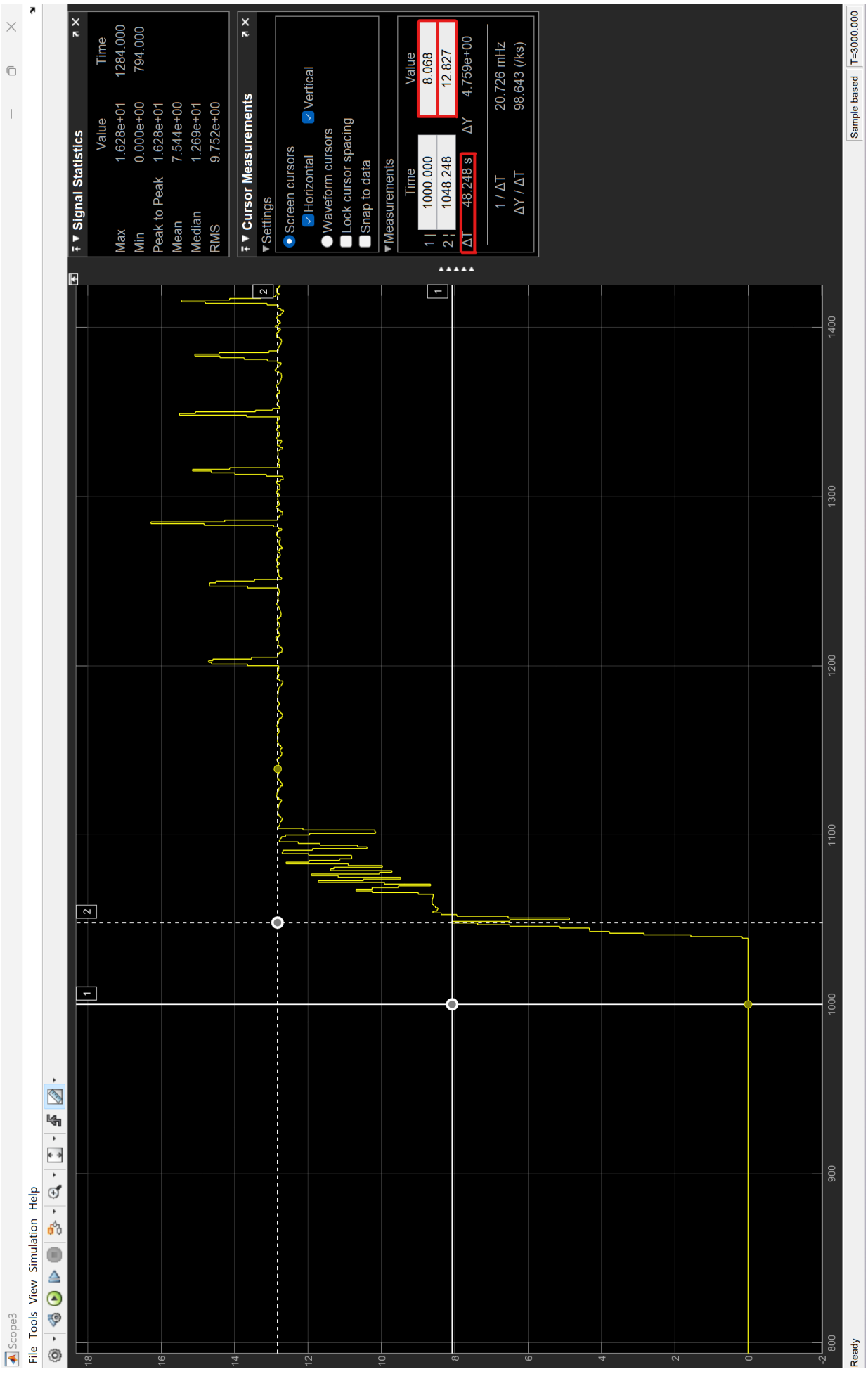


Figure 6 – Relevé de la vitesse du système