

**PROGETTAZIONE DI SISTEMA DI CONTROLLO PER UN PENDOLO INVERTITO**

Lavoro di Gruppo per il Corso di Progettazione di Sistema di Controllo

A.A. 2020/2021

Autori: Relatore:

Giovanni Latiano Prof. David Scaradozzi

Oleg Genova

Adolfo Damiano Cafaro

INDICE

[**INTRODUZIONE** 4](#_Toc76054861)

[**HARDWARE** 4](#_Toc76054862)

[**FASE 1** – **MODELLAZIONE** 5](#_Toc76054863)

[**FASE 2 – CONTROLLO E SIMULAZIONE SU MATLAB/SIMULINK** 8](#_Toc76054864)

[**FASE 3 – CONTROLLO DEL BALA-C TRAMITE PID** 16](#_Toc76054865)

# **INTRODUZIONE**

Il lavoro di gruppo svolto è stato quello della modellazione matematica e simulazione tramite gli strumenti Matlab/Simulink di un pendolo semplice ed un pendolo invertito. In seguito è stato progettato un sistema di controllo per la stabilizzazione del robot *M5stack Bala-C* per mezzo delle tecniche acquisite durante il corso*.*

Il lavoro si divide in tre fasi: la prima consiste nella **modellazione** del pendolo semplice di quello invertito con carrello, per poi procedere alla simulazione grafica dell’evoluzione dei pendoli tramite Matlab; la seconda si occupa della **sintesi del controllo** dei pendoli tramite tecniche classiche per poi passare alla simulazione grafica tramite Matlab/Simulink; la terza consiste nella **implementazione** del controllore effettivo per garantire la stabilità al robot *Bala-C.*

# **HARDWARE**

Il sistema sul quale è stato progettato e implementato il controllo è “BALA-C ESP32 Development Mini Self-Balancing Car”, prodotto dall’azienda M5Stack, consistente in un pendolo invertito dotato di:

* Microprocessore ESP32-Pico-D4, fornito di un clock di 240 MHz, 600 DMIPS, una memoria SRAM di 520 KB
* Due motori servo con angolo di rotazione di 360°, gestiti dal driver L9110S
* Batteria DC 3.7 V, 700mAh ricaricabile
* Due ruote collegantesi ai due motori servo
* Cavo USB per la programmazione via PC

La comunicazione tra ESP32 e il driver dei motori elettrici avviene tramite I2C.

Il prodotto ha un peso complessivo di 162 g e le dimensioni sono 30 x 100 x 105 mm.

# FASE 1 – MODELLAZIONE

In questa fase ci siamo concentrati sulla modellazione del **pendolo semplice** ed invertito con carrello usando la tecnica di **Newton-Eulero**. Il modello del pendolo semplice è stato trovato come segue:

Figure 1 - Diagramma delle forze sul pendolo semplice



Definiamo gli assi in funzione dell’angolo tra la posizione del pendolo e la posizione a riposo (verticale):

Derivando una volta la grandezza lungo l’asse x rispetto al tempo otteniamo la velocità tangenziale del pendolo e derivando ulteriormente descriviamo l’accelerazione tangenziale lungo x:

Dal bilanciamento delle forze lungo gli assi e dal bilanciamento dei momenti riferiti al cardine si ottiene:

Dall’ultima esplicitiamo , ottenendo

Introduciamo l’approssimazione per piccoli angoli, ottenendo:

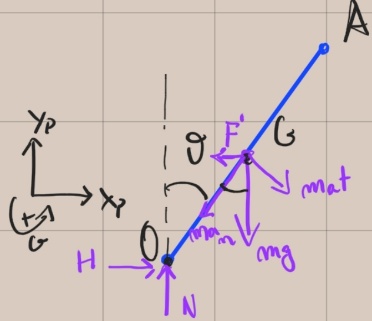
Definendo e la sua derivata , con le linearizzazioni, possiamo determinare la forma nello spazio di stato:

La modellazione del pendolo semplice è particolarmente utile per verificare l’efficacia del filtro di Kalman nelle simulazioni successive. Per facilità useremo un sistema stabile e che quindi non ha bisogno di alcun controllo.

Per poter applicare, invece, Newton-Eulero al **pendolo invertito con carrello** occorre suddividere il sistema in due sottosistemi, l’asta ed il carrello. Sono stati determinati i diagrammi delle forze.

Figure 2 - Diagramma delle forze sul pendolo

Per quanto riguarda l’asta abbiamo:



Invece per quanto riguarda il carrello abbiamo:

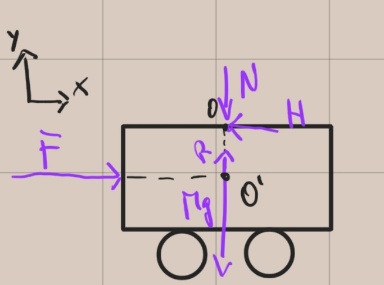


Figure 3 - Diagramma delle forze sul pendolo

Dove la forza è il forzamento di controllo, è la reazione vincolare orizzontale del pendolo, la reazione vincolare normale del pendolo, la reazione vincolare del pavimento, è il peso del carrello.

Definiamo gli assi, con l’angolo tra la posizione del pendolo e la verticale:

Analogamente a quanto operato per il pendolo semplice, ricaviamo l’accelerazione angolare

Considerando la dinamica complessiva, si ottiene:

Definendo il vettore come il vettore dello stato, tramite calcoli algebrici, otteniamo la forma nello spazio di stato:

# **FASE 2 – CONTROLLO E SIMULAZIONE SU MATLAB/SIMULINK**

**PENDOLO SEMPLICE CON FILTRO DI KALMAN**

Al fine di verificare l’efficacia dell’inserimento di un filtro di Kalman per il filtraggio del segnale uscente dal sensore giroscopico, è stato deciso di confrontare il segnale uscente dal sensore senza alcuna manipolazione e quello del sensore dopo il filtraggio con Kalman usando il modello del pendolo semplice.

Definiamo la matrice di convarianza che modella il rumore del sensore e la matrice di covarianza che modella il disturbo sul sistema. Aggiungendo al modello nello spazio di stato i nuovi ingressi (disturbo esterno) e (rumore sul sensore) riscriviamo il sistema come segue:

Possiamo ridefinire le matrici A, B, C compattandole. Definiamo come ingresso generico , ottenendo:

Lo schema grafico del nuovo sistema con disturbo e rumore è il seguente:

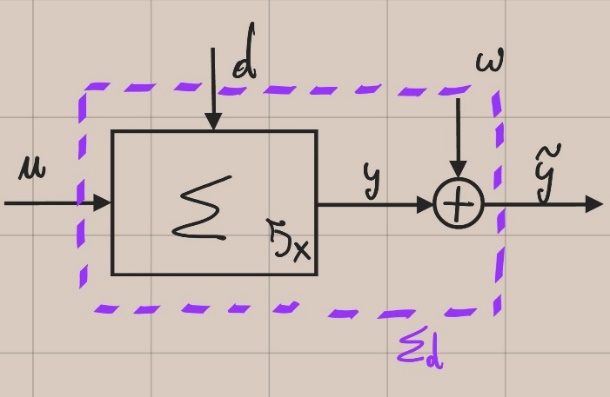


Figure 4 - Schema del sistema con rumore e disturbi

Di seguito il codice MATLAB di implementazione del sistema con i vari disturbi:

clc

clear all

g**=**10**;**

d**=**1**;**

m**=**0.5**;**

*%Matrici di sistema*

A**=[**0 1**;** **-**g**/**d 0**];**

B**=[**0**;** 1**/**d**^**2**\***m**];**

C**=[**1 0**];** *%misuriamo solo l'angolo theta del pendolo*

*%Raggiungibilità e Osservabilità*

R**=**ctrb**(**A**,**B**);**

r\_R**=**rank**(**R**);** *%RAGG*

O**=**obsv**(**A**,**C**);**

r\_O**=**rank**(**O**);** *%OSS*

Vd**=**0.1**\***eye**(**2**);** *%matrice cov disturbo processo*

Vw**=**0.1**;** *%matrice cov rumore misurazione*

Bd**=[**B Vd zeros**(**2**,**1**)];**

Dd**=[**0 0 0 1**];**

*%Costruzione sistema con disturbo e rumore*

sysD**=**ss**(**A**,**Bd**,**C**,**Dd**)**

*%Costruzione sistema con disturbo senza rumore e*

*%misura di tutte le variabili di stato(E per easy)*

*%Serve per confrontarlo con il KF*

sysE**=**ss**(**A**,**Bd**,**eye**(**size**(**A**,**1**)),**zeros**(**size**(**A**,**1**),**size**(**Bd**,**2**)))**

*%Costruzione del filtro di Kalman*

**[**Kf**,**P**,**E**]=**lqe**(**A**,**Vd**,**C**,**Vd**,**Vw**);** *%guadagno del filtro*

sysKF**=**ss**(**A**-**Kf**\***C**,[**B Kf**],**eye**(**2**),**0**\*[**B Kf**])**

*%Simulazioni*

dt**=**0.005**;** *%dt più piccolo maggiore precisione*

t**=**dt**:**dt**:**10**;**

u**=**zeros**(**1**,**size**(**t**,**2**));**

d**=**randn**(**2**,**size**(**t**,**2**));**

w**=**randn**(**1**,**size**(**t**,**2**));**

U**=[**u**;**d**;**w**];**

*%Simulazione sistema ed osservazione uscita sensore rumoroso*

**[**yD**,** t**]=**lsim**(**sysD**,**U**,**t**,[**0.1**;**0**]);**

figure**(**1**)**

plot**(**t**,**yD**)**

hold on

*%Simulazione sistema con osservazione degli stati*

**[**yE**,** t**]=**lsim**(**sysE**,**U**,**t**,[**0.1**;**0**]);**

plot**(**t**,**yE**(:,**1**))**

hold off

title**(**'Comparazione tra uscita sensore rumoroso e stato reale'**)**

legend**(**'\theta misurato'**,**'\theta reale'**)**

*%Simulazione sistema KF*

**[**yKF**,**t**]=**lsim**(**sysKF**,[**u**;** yD**'],**t**,[**0.1**;**0**]);**

figure**(**2**)**

plot**(**t**,**yKF**)**

hold on

plot**(**t**,**yE**)**

hold off

title**(**'Comparazione tra stima del KF e stato reale'**)**

legend**(**'\theta reale'**,**'\omega reale'**,**'\theta stimato'**,**'\omega stimato'**)**

Nel codice si vede come vengono costruiti 3 sistemi: il primo, sysD, ovvero il sistema con disturbo su processo e rumore su misura; il secondo sysE, ovvero il sistema con solo disturbo sul processo; infine sysKF che è il sistema Filtro di Kalman che prende in ingresso gli ingressi e le uscite “sporche” del sistema sysD e quindi stima lo stato del sistema sysD. Il sistema sysE viene usato per operare alcune comparazioni tra lo stato stimato e quello reale. Di seguito alcune comparazioni:

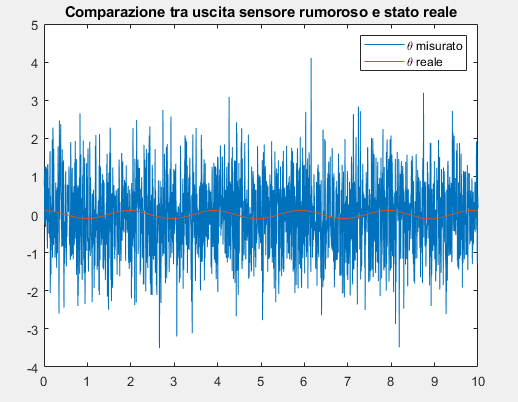


Figure 5

In *Figure 5* è stata fatta la comparazione tra l’uscita misurata del sistema con disturbo e rumore (sysD) e l’uscita misurata del sistema senza rumore bianco (sysE). Come si può notare a causa del grande valore della matrice di covarianza, la misura sul primo sistema viene completamente coperta dal rumore bianco.

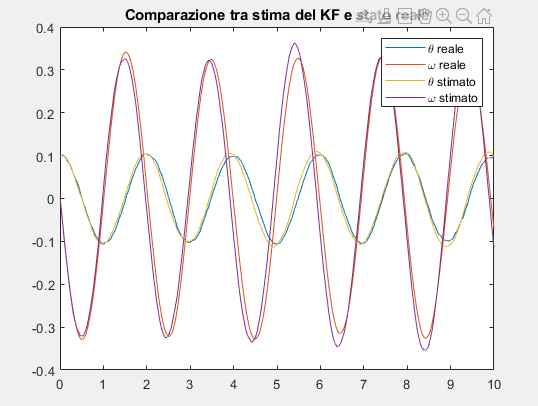


Figure 6

In *Figure 6* viene fatta la comparazione tra lo stato reale del sistema (sysE)e la stima dello stato effettuata dal filtro di Kalman (sysKf). Come si può notare, nonostante le misure del sistema siano affette da molto rumore, il filtro di Kalman riesce comunque ad effettuare una stima che si differenzia di poco dallo stato reale.

**CONTROLLO DEL PENDOLO INVERTITO TRAMITE ALLOCAZIONE DEGLI AUTOVALORI E OSSERVATORE**

In questa parte del lavoro è stata analizzata la stabilità del pendolo invertito con carrello, dimostrando che il sistema è instabile e che la risposta impulsiva in catena aperta è divergente.

Abbiamo quindi prima studiato la raggiungibilità e osservabilità del sistema per assicurarci di poter allocare gli autovalori a piacimento, per poi costruire il controllo in retroazione tramite osservatore dello stato.

Gli autovalori del controllore e dell’osservatore sono stati scelti per tentativi in modo da escludere la possibilità di saturazione degli attuatori e dei sensori, garantendo comunque un tempo di salita accettabile a regime permanente. Di seguito il codice MATLAB corrispondente:

clear all

close all

clc

*%definiamo le grandezze del sistema*

M **=** 0.5**;**

m **=** 0.2**;**

I **=** 0.006**;**

g **=** 9.8**;**

d **=** 1**;**

f**=(-**M**\***m**\***d**^**2**+**I**\*(**m**+**M**));**

A**=** **[**0 1 0 0**;**

0 0 **(**d**^**2**\***m**^**2**\***g**)/**f 0**;**

0 0 0 1**;**

0 0 **(-**d**\***M**\***g**)/**f 0**];**

B**=** **[**0**;**

**(**I**-**d**^**2 **\*** m**)/**f**;**

0**;**

**(**d**\***m**)/**f**];**

C**=** **[**1 0 0 0**;** *%misura posizione del carretto*

0 0 1 0**];** *%misura angolo del pendolo*

D**=** **[**0**;**

0**];**

SYS**=**ss**(**A**,**B**,**C**,**D**);**

R**=**ctrb**(**A**,**B**);**

O**=**obsv**(**A**,**C**);**

rR**=**rank**(**R**);** *%SYS Ragg*

rO**=**rank**(**O**);** *%SYS Oss*

*%risposta impulsiva catena aperta*

figure**(**1**)**

impulse**(**SYS**)**

title**(**'Risposta impulsiva in catena aperta'**)**

*%evoluzione libera catena aperta*

x0**=[**0**;**0**;**0.1**;**0**];**

figure**(**2**)**

**[**y**,**t**]=**initial**(**SYS**,**x0**,**0**:**0.01**:**10**);**

plot**(**t**,**y**(:,**1**),**t**,**y**(:,**2**))**

title**(**'Evoluzione libera in catena aperta'**)**

legend**(**'X carretto'**,** '\theta pendolo'**)**

*%Controllore e Osservatore*

K**=**place**(**A**,**B**,[-**2 **-**2.1 **-**2.3 **-**2.4**]);**

G**=**place**(**A**',**C**',[-**1 **-**2 **-**3 **-**4**]);**

G**=**G**';**

*%Sistema catena chiusa*

Ac**=[**A**-**B**\***K **-**B**\***K**;** zeros**(**4**)** A**-**G**\***C**];**

Bc**=[**B**;** zeros**(**4**,**1**)];**

Cc**=[**C zeros**(**2**,**4**)];**

Dc**=[**0**;**0**];**

SYS\_cc**=**ss**(**Ac**,**Bc**,**Cc**,**Dc**)**

*%Risposta impulsiva sistema in catena chiusa*

figure**(**3**)**

impulse**(**SYS\_cc**)**

title**(**'Risposta impulsiva in catena chiusa'**)**

*%Evoluzione libera sistema in catena chiusa*

x0**=[**0**;**0**;**0.1**;**0**;**0.2**;**0.3**;**0.01**;**0.01**];**

**[**y**,**t**]=**initial**(**SYS\_cc**,**x0**,**0**:**0.01**:**10**);**

figure**(**4**)**

plot**(**t**,**y**(:,**1**),**t**,**y**(:,**2**))**

legend**(**'X carretto'**,** '\theta pendolo'**)**

title**(**'Evoluzione libera in catena chiusa'**)**

*%Animazione sistema*

**for** i**=**1**:**2**:**1001

figure**(**5**)**

plot**(**y**(**i**,**1**),**0**,**'sb'**,**'MarkerSize'**,**40**)**

hold on

plot**([**y**(**i**,**1**)** y**(**i**,**1**)+**d**\***sin**(**y**(**i**,**2**))],[**0 d**\***cos**(**y**(**i**,**2**))],**'-r'**)**

hold off

xlim**([-**2 2**])**

ylim**([-**2 2**])**

drawnow

**end**

Figura 1 - Evoluzione libera in catena aperta

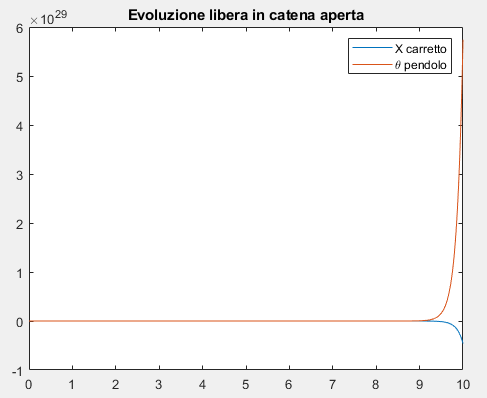
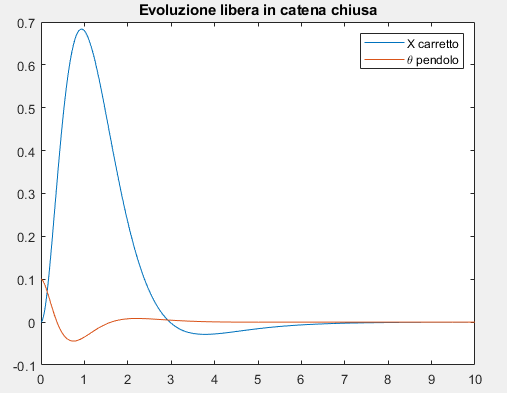
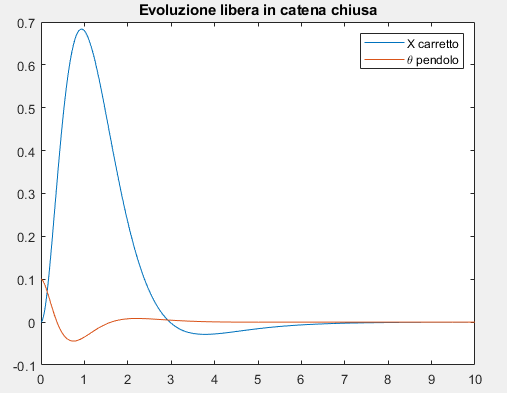
****

Figure 7 - Evoluzione libera in catena chiusa

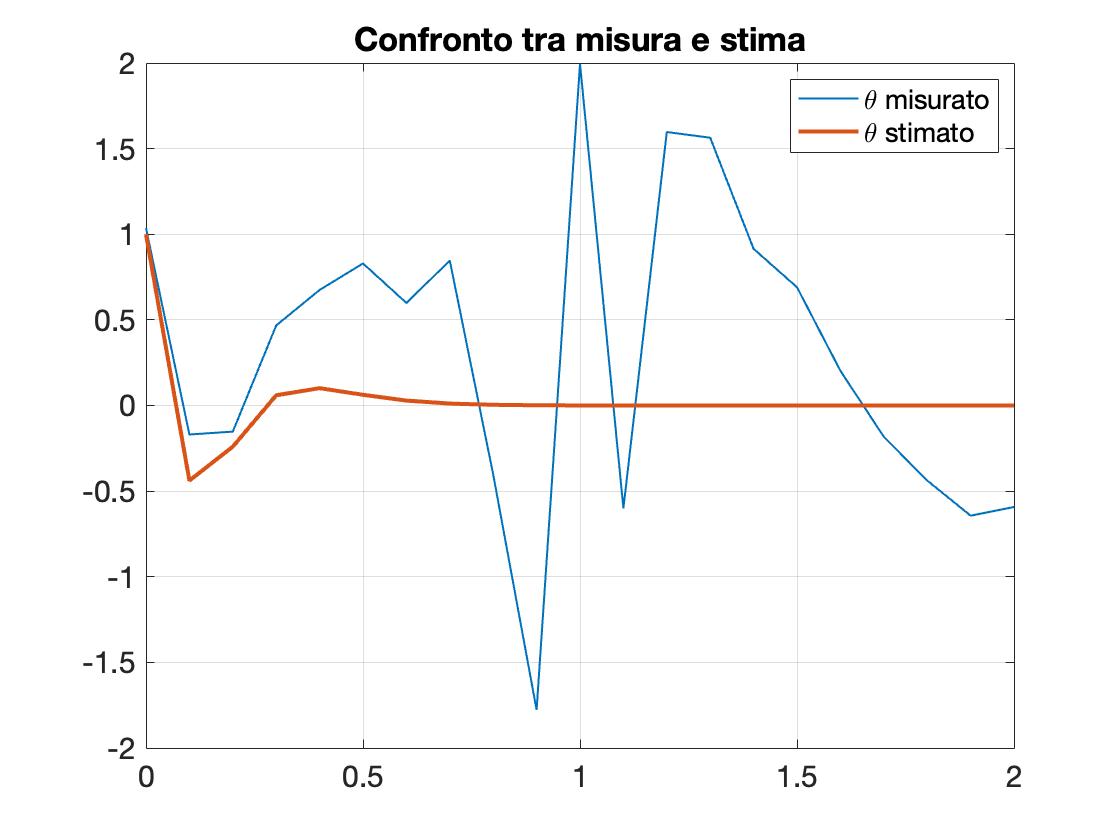
Figure 8 - Evoluzione libera in catena aperta

Figura Evoluzione libera in catena chiusa

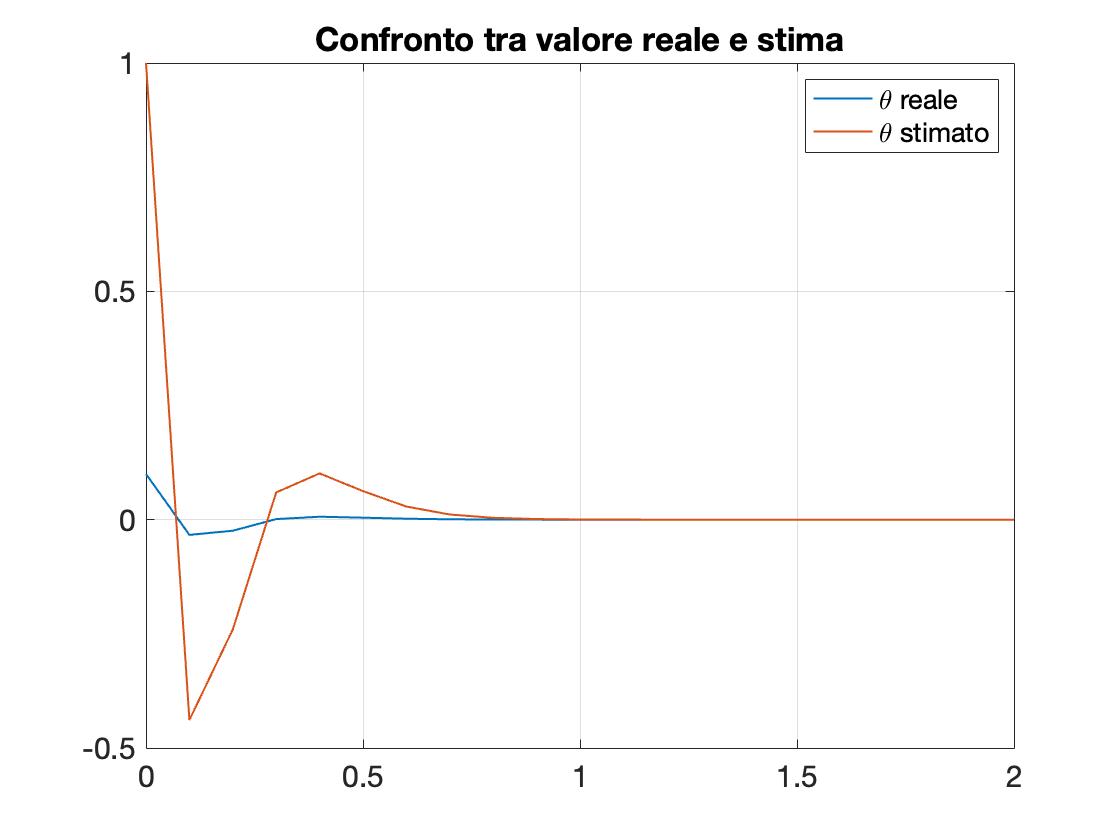
**CONTROLLO DEL PENDOLO CON CARRELLO CON ALLOCAZIONE DEGLI AUTOVALORI E FILTRO DI KALMAN**

Riprendiamo l’esperimento del paragrafo precedente, supponendo però che il sistema sia soggetto ad un disturbo sulla misurazione dell’angolo. In questo caso per utilizzare la tecnica di allocazione degli autovalori tramite retroazione dallo stato è necessario andare a stimare il vettore di stato tramite un filtro di Kalman. Essendo il sistema lineare posso utilizzare il principio di sovrapposizione degli effetti e risolvere separatamente il problema della stima con quello del controllo tramite retroazione. Andiamo a considerare come matrice K di retroazione quella definita nel paragrafo precedente. La stima del vettore di stato è fatta andando ad utilizzare il comando “kalman(sys,Q,R)” implementato nell’ambiente di Matlab.

Nella figura sottostante si osserva l’andamento dell’angolo misurato e della sua stima.



Nella figura sottostante si osserva l’andamento dell’angolo stimato e dell’angolo reale.



**CONTROLLO DEL PENDOLO CON CARRELLO CON PID E FILTRO DI KALMAN**

Questa sezione si occupa del controllo del pendolo con carrello risolvendo numericamente le equazioni differenziali con il metodo di **Eulero forward**. E’ stato sintetizzato un controllore **PID** simulando del rumore bianco di natura gaussiana sulla misura di posizione del centro di massa del pendolo rispetto al carrello. Quindi per ripulire la misura della posizione abbiamo usato un filtro di Kalman.

clc

clear all

close all

*%Pendolo linearizzato range di lavoro 0°%13°, 0rad%0.23rad*

g**=**9.81**;** *%accelerazione gravitazionale [m/s^2]*

theta**=**0.15**;** *%angolo iniziale [rad]*

d**=**0.3**;** *%lunghezza asta [m], regolare il proporzionale in base all'asta*

xc**=**0**;** *%posizione iniziale carretto [m]*

v\_xc**=**0**;** *%velocità iniziale carretto [m/s]*

a\_xc**=**0**;** *%accelerazione iniziale carretto [m/s^2]*

xp**=**theta**\***d**;** *%posizione iniziale pendolo rispetto al carretto [m]*

v\_xp**=**0**;** *%velocità iniziale pendolo rispetto al carretto [m/s]*

a\_xp**=**g**\***xp**/**abs**(**d**);** *%accelerazione iniziale pendolo rispetto al carretto[m/s^2]*

pos\_p**=[**xp d**\***cos**(**xp**/**d**)];** *%coordinate x y del pendolo pendolo rispetto al carretto*

h**=**0.025**;** *%passo*

kp**=**20**;** *%proporzionale*

ki**=**2**;** *%integrativa*

kd**=**1**;** *%derivativa*

e**=**0**;** *%errore*

p\_e**=**0**;** *%errore precedente*

intgr**=**0**;** *%memoria*

rif**=**0**;** *%posizione desiderata del pendolo rispetto al carretto [m]*

M**=**0.5**;** *%massa carrello [kg]*

m**=**0.2**;** *%massa pendolo [kg]*

I**=**0.006**;** *%momento d'inerzia pendolo [Kgm^2]*

F**=**0**;** *%forzamento impresso al carrello per controllare il pendolo [N]*

l**=**true**;**

A **=** 1**;**

B **=** 1**;**

C **=** 1**;**

D **=** 1**;**

Q **=** 0.1**;**

R **=** 0.0288**;**

K **=** 0**;**

P\_1 **=** 0**;**

P **=** 0**;**

P\_pre **=** 0**;**

xs = 0**;** *%posizione del pendolo stimata dal FdK*

xs\_pre = 0;

**for** t**=**0**:**h**:**50

*%VISUALIZZAZIONE GRAFICA DEL PENDOLO CON CARRETTO*

figure**(**2**)** *%plot che vede il carretto dal punto di vista di un osservatore esterno*

plot**(**xc**,**0**,**'sb'**,**'MarkerSize'**,**40**)** *%plot carretto*

hold on

plot**([**xc xc**+**xp**],[**0 d**\***cos**(**xp**/**d**)],**'r-'**);** *%plot pendolo*

hold off

title**(**"Carretto con pendolo visto dall'esterno"**);**

xlim**([-**2**\***d 2**\***d**]);**

ylim**([-**2**\***d 2**\***d**]);**

drawnow

*%SIMULAZIONE DI SENSORE RUMOROSO*

y = xp;

w **=** **(-**1 **+** 2**\***rand**())/**100**;**

y **=** y **+** w**;**

*%IMPLEMENTAZIONE DELLE EQUAZIONI DEL FILTRO DI KALMAN*

P\_1**=**A**\***P\_pre**\***A **+** B**\***Q**\***B**;**

K **=** P\_1**\***C**/(**C**\***P\_1**\***C **+** D**\***R**\***D**);**

xs **=** A**\***xs\_pre **+** K**\*(**y**-**C**\***A**\***xs\_pre**)** **;**

P **=** **(**1 **-** K**)\***P\_1**;**

P\_pre **=** P**;**

xs\_pre **=** xs**;**

e**=**rif**-**xs**;** *%calcolo errore posizione del pendolo rispetto al CM del carretto*

intgr**=**intgr**+**e**\***h**;** *%memorizzazione*

dervt**=(**e**-**p\_e**)/**h**;** *%predizione*

p\_e**=**e**;** *%aggiorno errore precedente*

F**=(**kp**\***e**+**ki**\***intgr**+**kd**\***dervt**)\*(-**1**);** *%forzamento*

*%VEDI MODELLO MATEMATICO PER CHIARIMENTI*

theta**=**xp**/**d**;**

a\_xc**=** **(**g**\***theta**\***d**^**2**\***m**^**2 **+** F**\***d**^**2**\***m **+** F**\***I**)/(**M**\***m**\***d**^**2 **+** M**\***I **+** m**\***I**);** *%accelerazione del carretto*

a\_xp**=** d**\*-(**d**\***m**\*(**M**\***g**\***theta **-** F **+** g**\***m**\***theta**))/(-** M**\***m**\***d**^**2 **+** M**\***I **+** m**\***I**)** **-** a\_xc**;** *%accelerazione del pendolo*

v\_xc**=**v\_xc**+**a\_xc**\***h**;** *%velocità del carretto*

v\_xp**=**v\_xp**+**a\_xp**\***h**;** *%velocità del pendolo*

xc**=**xc**+**v\_xc**\***h**;** *%posizione del carretto*

xp**=**xp**+**v\_xp**\***h**;** *%posizione del pendolo*

**if** abs**(**v\_xp**)<=** 0.5 **&&** abs**(**e**)<=**0.5 *%resetto memoria NB:diminuire i parametri se si spegne il derivatore*

intgr**=**0**;** *%serve per impedire che il pendolo cada*

**end**

**if** mod**(**t**,**4**)==**0 *%introduco in disturbo randomico*

**if** l**==**true

v\_xp**=**v\_xp**+**rand**()/**15**;**

l**=**false**;**

**else**

v\_xp**=**v\_xp**-**rand**()/**15**;**

l**=**true**;**

**end**

print**=**"Introduco un disturbo..."

**end**

**end**

Il sistema risulta stabile ai disturbi esterni e la stima della posizione del pendolo rispetto al carretto fornita dal FdK permette un controllo preciso.

# **FASE 3 – CONTROLLO DEL BALA-C TRAMITE PID**

In quest’ultima fase ci siamo concentrati sulla progettazione di un sistema di controllo per garantire la stabilità del Bala-c e quindi non farlo cadere. Abbiamo usato due approcci, il primo è stato lo sviluppo di un controllore PID senza filtro di Kalman, il secondo è stato lo sviluppo di un controllore PID con filtro di Kalman. Entrambi i controlli sono scritti su Arduino e caricati sul microcontrollore M5StackCPlus. Lo schema di controllo del primo metodo è il seguente:

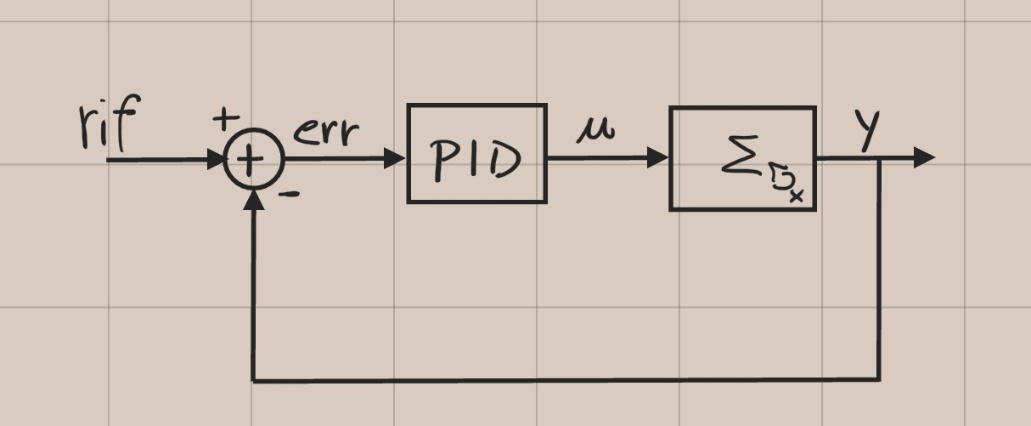


Figure 9 - Schema di controllo PID

dove *rif* è l’angolo desiderato, nel nostro caso il robot è in equilibrio instabile quando l’angolo misurato dalla IMU è di -101.0°. Si vede poi come l’errore trovato tra la misura *y* ed il rifermento *rif* vada a finire nel PID il quale genera l’opportuno segnale di controllo. Il codice del controllo in C++ è il seguente:

#include <M5StickCPlus.h>

#include "bugC.h"

**float** roll = 0.0F; *//rick*

**float** v1 = 0.0F;

**float** v2 = 0.0F;

**float** rif = -101.0F;

**float** err = 0.0F;

**float** pre\_err = 0.0F;

**float** sum\_err = 0.0F;

**float** proporzionale = 0.0F;

**float** integrale = 0.0F;

**float** derivativo = 0.0F;

**float** P = 3.5F;

**float** I = 0.025F;

**float** D = 1.5F;

**float** PID = 0.0F;

**float** pre\_time = 0.0F;

**float** act\_time = 0.0F;

**float** dt = 0.0F;

BUGC BugC;

**void** setup() {

M5.begin();

M5.IMU.Init();

BugC.Init();

M5.Lcd.setRotation(2);

M5.Lcd.fillScreen(BLACK);

M5.Lcd.setTextSize(1);

M5.Lcd.setCursor(15, 20);

M5.Lcd.println("Roll");

}

void loop() {

act\_time = millis();

dt=act\_time - pre\_time;

M5.IMU.getAhrsData(&v1,&roll,&v2);

M5.Lcd.setCursor(15, 40);

M5.Lcd.printf("%5.2f", roll);

err= rif - roll;

if(err > 0.1 or err < -0.1){

sum\_err = sum\_err + err \* dt;

proporzionale = P \* err;

integrale = I \* sum\_err ;

derivativo = D \* ((err - pre\_err)/dt);

PID = proporzionale + integrale + derivativo;

BugC.BugCSetAllSpeed(-PID, PID, 0, 0);

pre\_err = err;

}

else

{

BugC.BugCSetAllSpeed(0, 0, 0, 0);

}

pre\_time = act\_time;

delay(15);

}

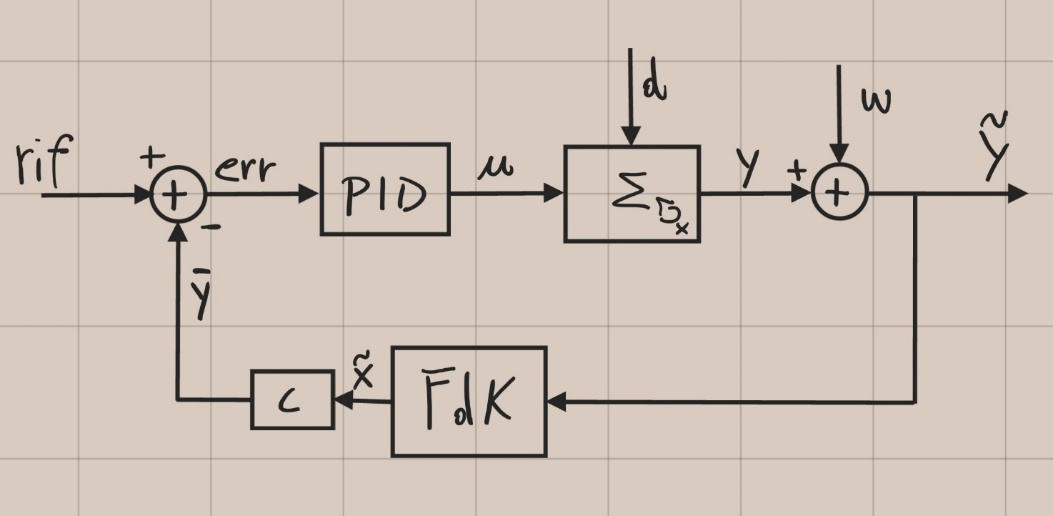
Per quanto riguarda il controllo tramite PID e filtro di Kalman, lo schema di controllo è il seguente:

Figure 10 - Schema di controllo PID con FdK in retroazione

Il filtro di Kalman in questo caso è molto utile in quanto l’uscita del sensore IMU è affetta da rumore bianco di tipo Gaussiano. Per determinare che l’uscita fosse affetta da rumore bianco Gaussiano abbiamo prelevato un numero di campioni dell’IMU in una posizione stabile (ad es fermo sul tavolo) ed abbiamo analizzato la distribuzione del rumore tramite un istogramma. Per quanto riguarda il controllo, in questo caso si vede come l’uscita rumorosa del sensore passa nel filtro di Kalman per essere pulita, per poi venire confrontata con il riferimento e generare l’errore di ingresso per il PID.



Figure - Distribuzione del rumore attorno al valore medio (5700 campioni ca)

#include <M5StickCPlus.h>

#include "bugC.h"

*//*

**float** K = 0; *//guadagno FdK*

**float** A = 1; *//matrice A*

**float** B = 1; *//matrice B*

**float** C = 1; *//matrice C*

**float** D = 1; *//matrice D*

**float** P = 0; *//matrice cov errore*

**float** P\_pre = 0;

**float** P\_1 = 0;

**float** y = 0; *//uscita IMU*

**float** x = 0; *//stima angolo*

**float** x\_pre = 0;

**float** v1 = 0;

**float** v2 = 0;

**float** Q = 0.1; *//cov disturbo e incertezza sul processo/sensore IMU (parecchia incertezza sulla modellazione del sensore/sistema)*

**float** R = 0.0228; *//cov rumore sensore ROll IMU (trovata sperimentalmente)*

*//*

**float** rif = -101.0; *//angolo di riferimento*

**float** err = 0.0;

**float** pre\_err = 0.0;

**float** sum\_err = 0.0;

**float** proporzionale = 0.0; *//controllo proporzionale*

**float** integrale = 0.0; *//controllo integrale*

**float** derivativo = 0.0; *//controllo derivativo*

**float** P\_gain = 2.9; *//guadagno proporzionale*

**float** I\_gain = 0.018; *//guadagno integrale*

**float** D\_gain = 0.5; *//guadagno derivativo*

**float** PID = 0.0;

**float** pre\_time = 0.0;

**float** act\_time = 0.0;

**float** dt = 0.0;

*//*

BUGC BugC;

**void** setup() {

M5.begin();

M5.IMU.Init();

BugC.Init();

M5.Lcd.setRotation(2);

M5.Lcd.fillScreen(BLACK);

M5.Lcd.setTextSize(1);

M5.Lcd.setCursor(15, 20);

M5.Lcd.println("Roll");

M5.Lcd.setCursor(15, 70);

M5.Lcd.println("Real Roll");

}

**void** loop() {

*//Acquisizione misura angolo*

M5.IMU.getAhrsData(&v1, &y,&v2);

*//Implementazione in linea del filtro di Kalman*

P\_1=A\*P\_pre\*A + B\*Q\*B;

K = P\_1\*C/(C\*P\_1\*C + D\*R\*D);

x = A\*x\_pre + K\*(y-C\*A\*x\_pre);

P = (1 - K)\*P\_1;

x\_pre = x;

P\_pre = P;

*//Print su LCD*

M5.Lcd.setCursor(15, 90);

M5.Lcd.printf("%5.2f", x);

M5.Lcd.setCursor(15, 40);

M5.Lcd.printf("%5.2f", y);

*//Controllo PID*

err= rif - x; *//scostamento posizione desiderata*

act\_time = millis();

dt = act\_time - pre\_time;

**if**(err > 0.1 **or** err < -0.1){ *//se /err/>0.1 il PID interviene*

sum\_err = sum\_err + err \* dt;

proporzionale = P\_gain \* err;

integrale = I\_gain \* sum\_err ;

derivativo = D\_gain \* ((err - pre\_err) / dt);

PID = proporzionale + integrale + derivativo;

BugC.BugCSetAllSpeed(-PID, PID, 0, 0);

pre\_err = err;

}

**else**

{

BugC.BugCSetAllSpeed(0, 0, 0, 0);

}

pre\_time = act\_time;

delay(10);

}

In questo caso abbiamo implementato le equazioni del filtro di Kalman in linea in quanto dato che il sistema è instabile non è possibile determinare un guadagno K di regime. La matrice di covarianza R = 0.0228 del sensore è stata trovata salvando un campione di 1000 misure in una posizione di equilibrio (ad es IMU ferma sul tavolo) per poi calcolare la covarianza del vettore delle misurazioni tramite la funzione cov() di Matlab. Per quanto riguarda la matrice di covarianza del processo Q = 0.1 è stato assegnato tale valore perché è stata considerata una consistente incertezza nella modellazione della IMU.

Facendo partire il robot con i due controlli è risultato evidente come il controllo tramite l’uscita pulita dal FdK risulti più preciso ed in alcuni casi riesca a portate il sistema della posizione di equilibrio instabile, di fatto arrestando il robot.