



Hands-On Learning with the Low-Cost Bi-Copter: From Fundamentals to Advanced Controls

60698830

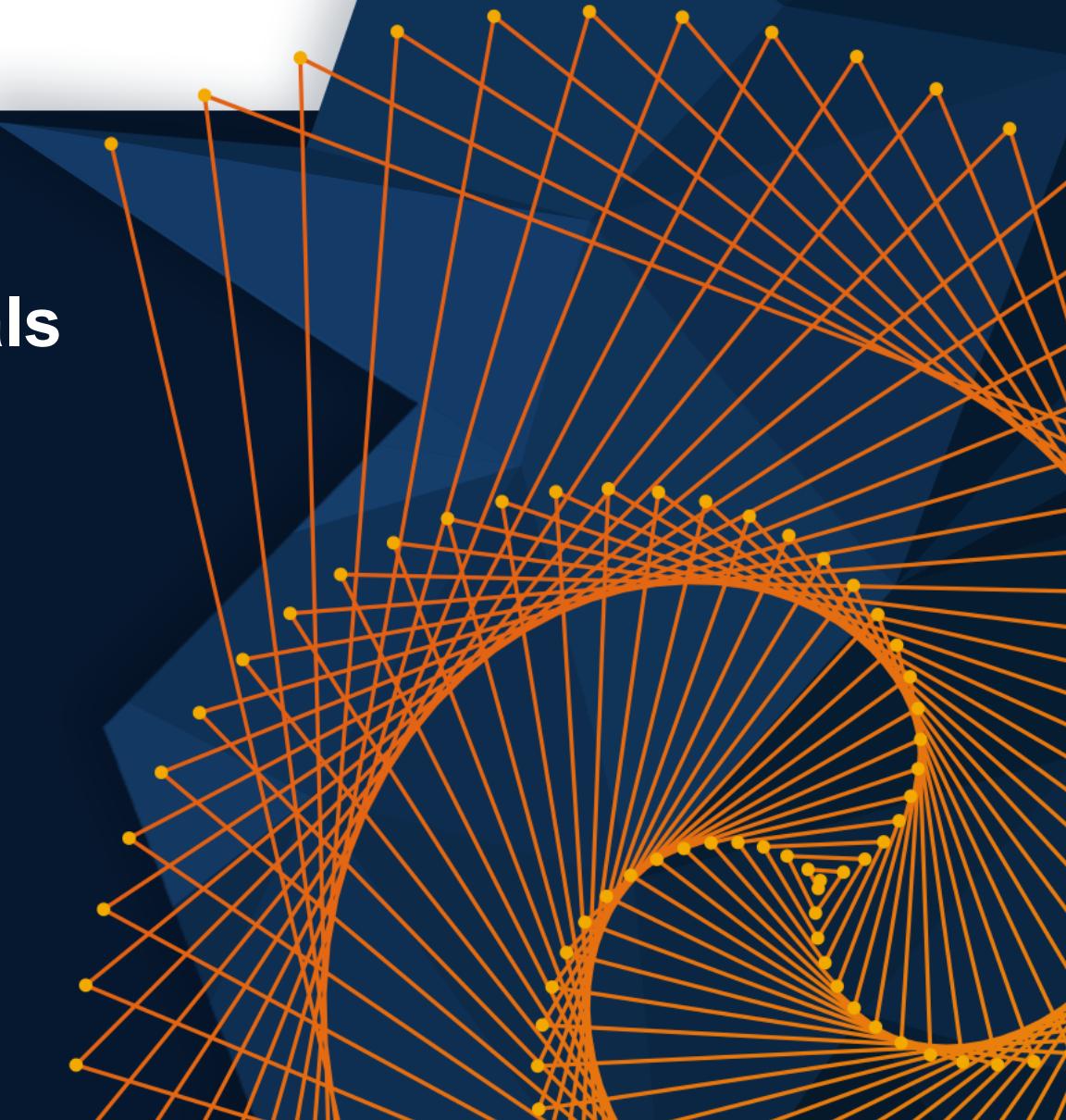
*Prof. Eniko T. Enikov,
University of Arizona*



Dr. Melda Ulusoy, MathWorks



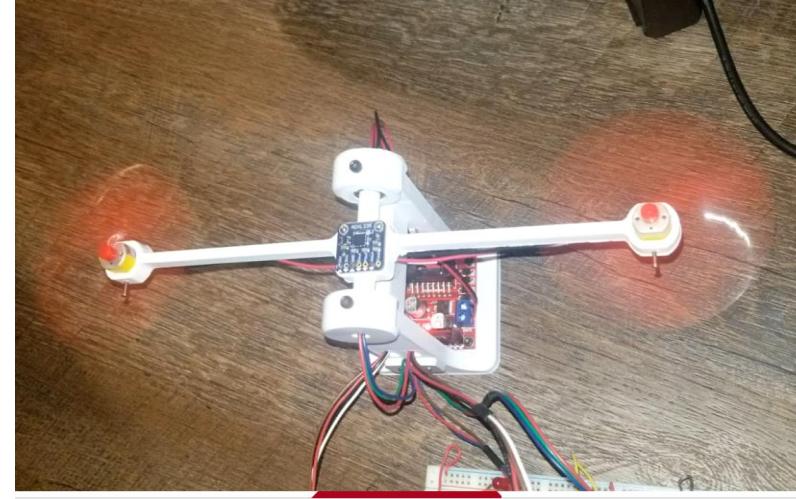
MATLAB EXPO



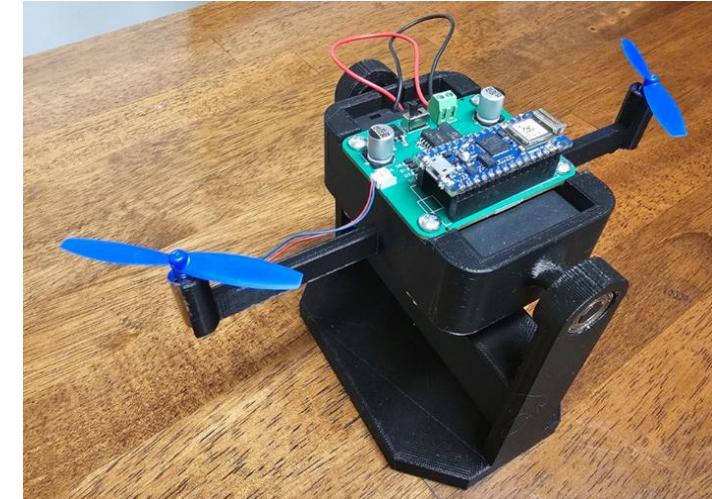
Motivation and Background: Low-cost hardware, allowing classical and advanced control experiments



**Aeropendulum
2011
(USB cable)**

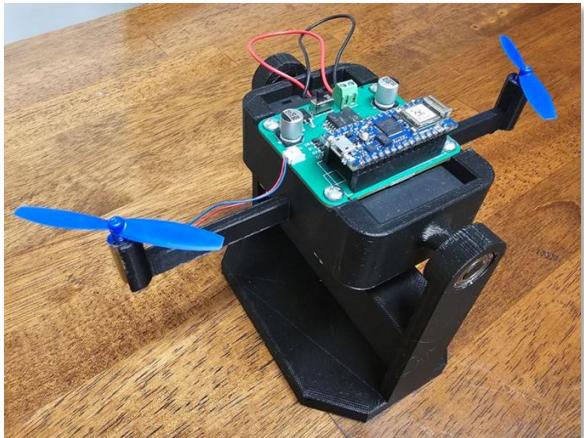


**Bi-Copter with
PIC16F690
2020
(USB cable)**

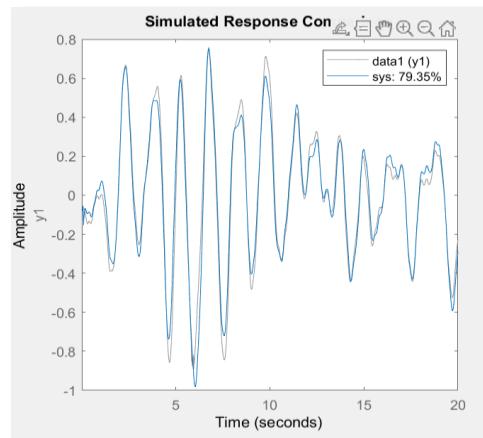


**Bi-Copter with
Arduino Nano
2024
(wireless)**

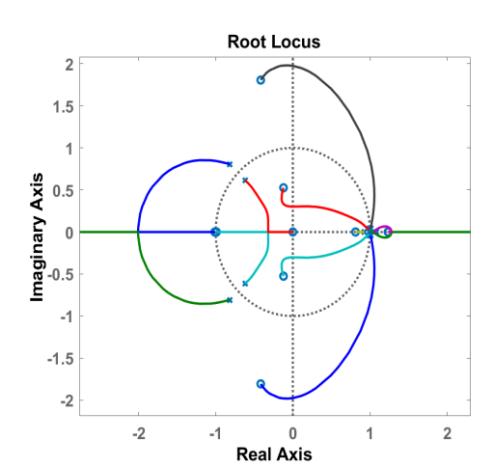
This presentation focuses on the following topics



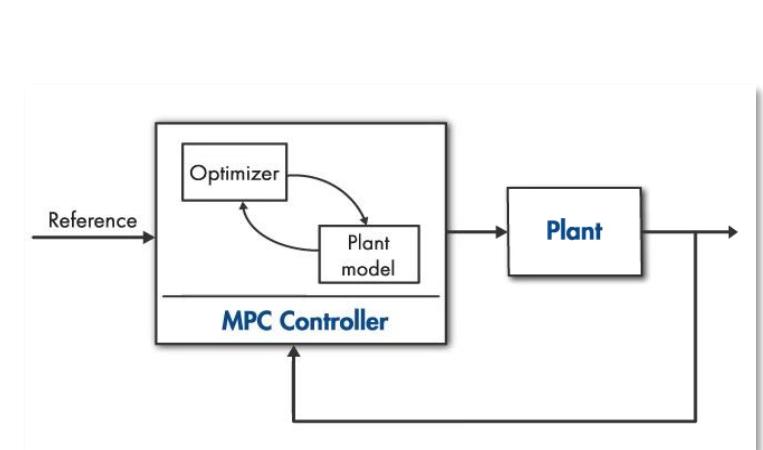
Bi-Copter



Modeling and System Identification

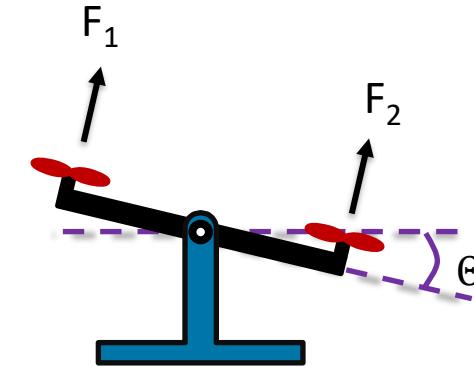
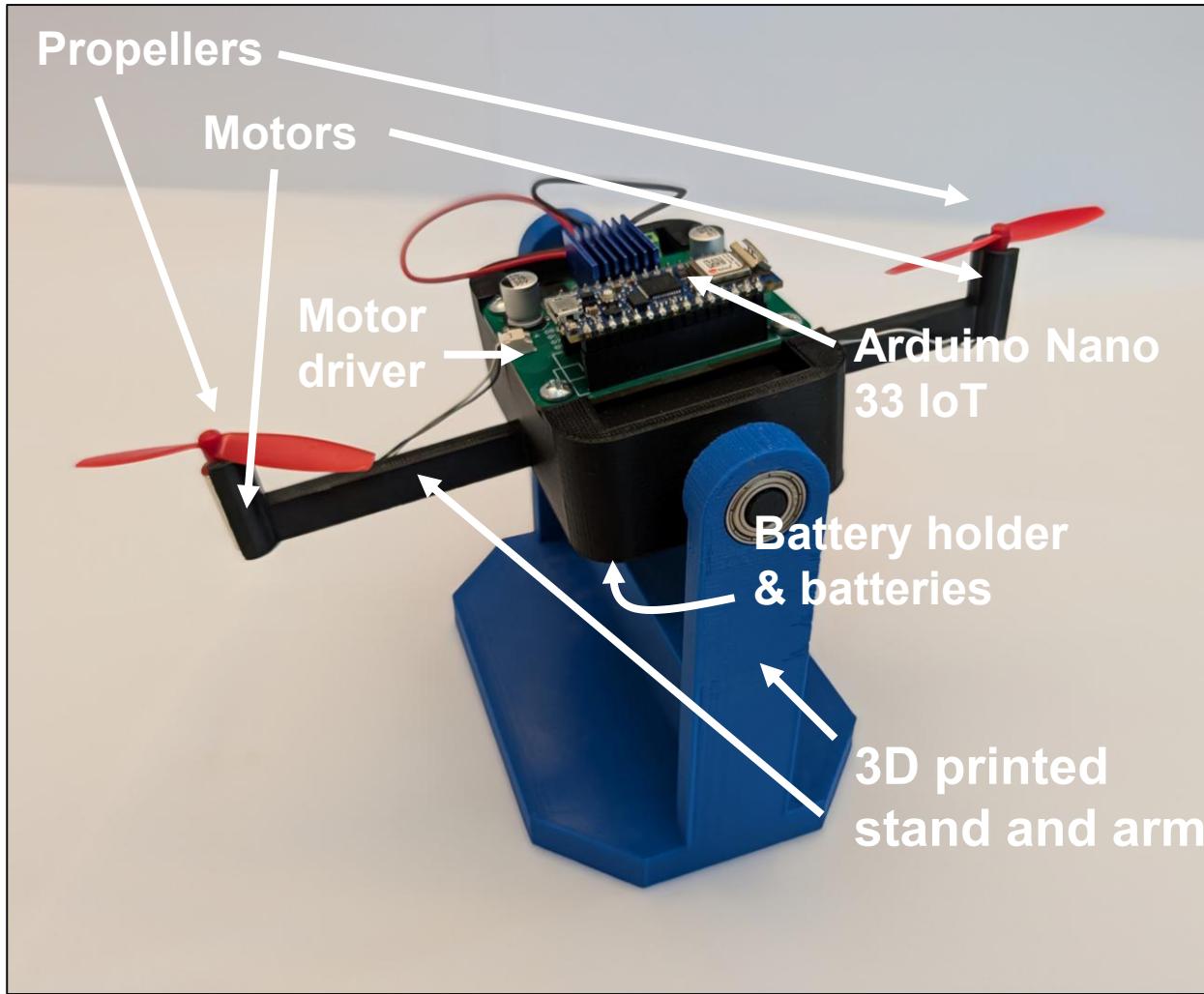


Classical Controller Design

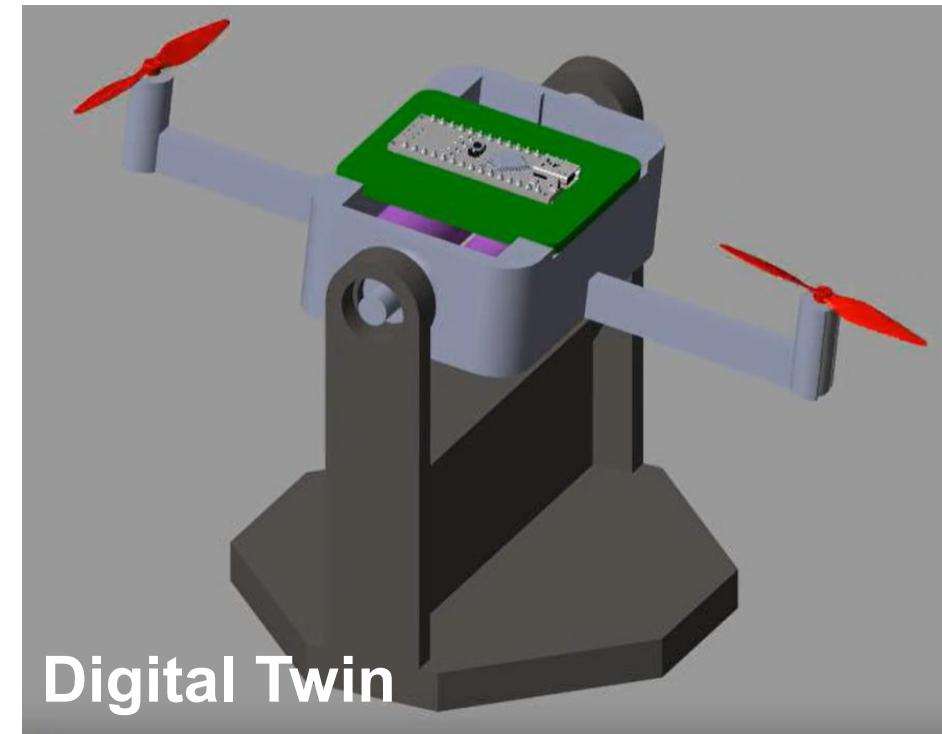
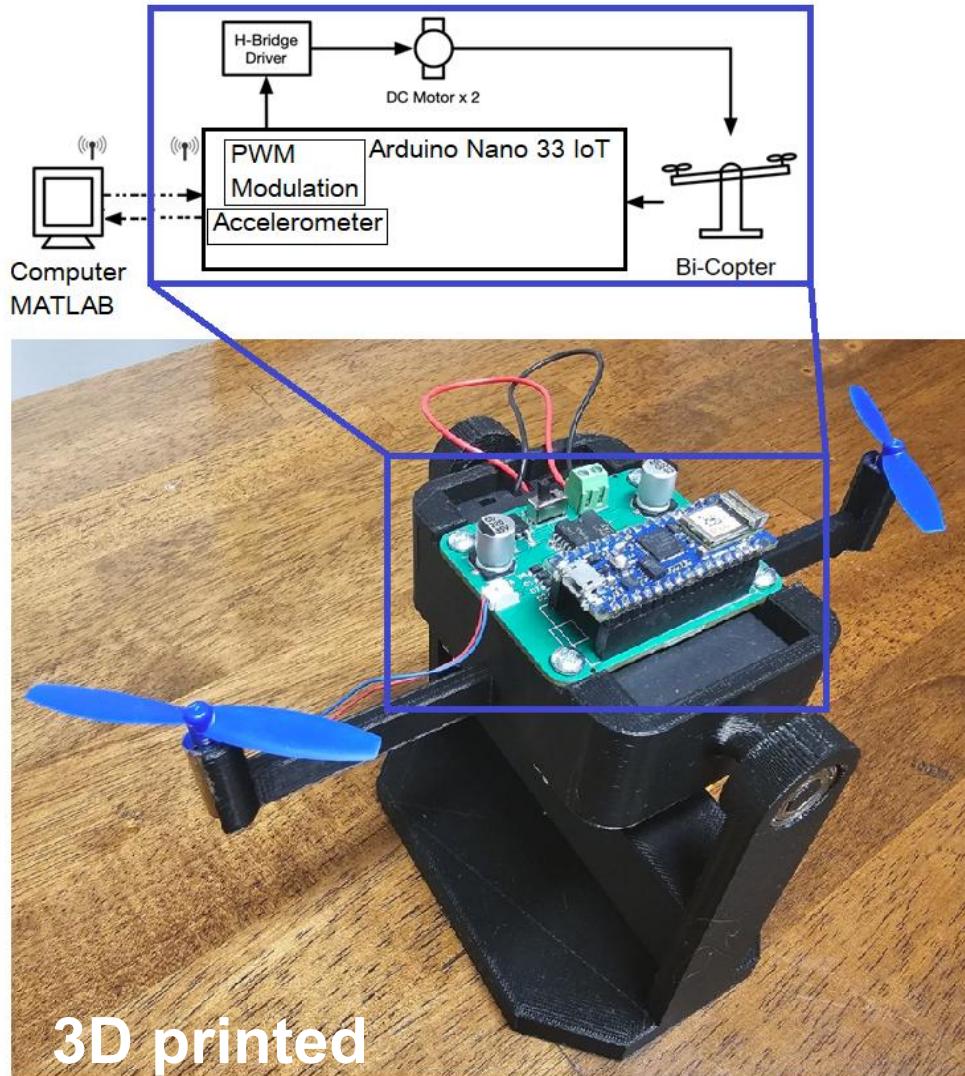


Modern (MPC) Controller Design

Hardware overview



The low-cost bi-copter and its features

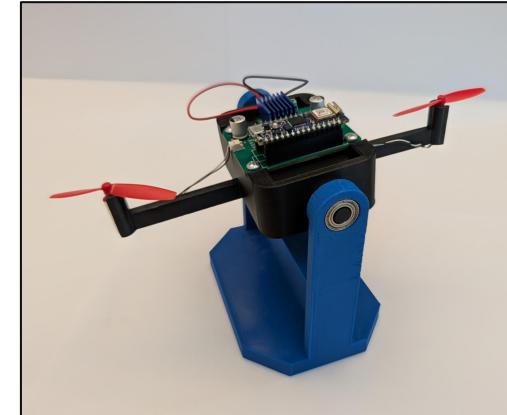
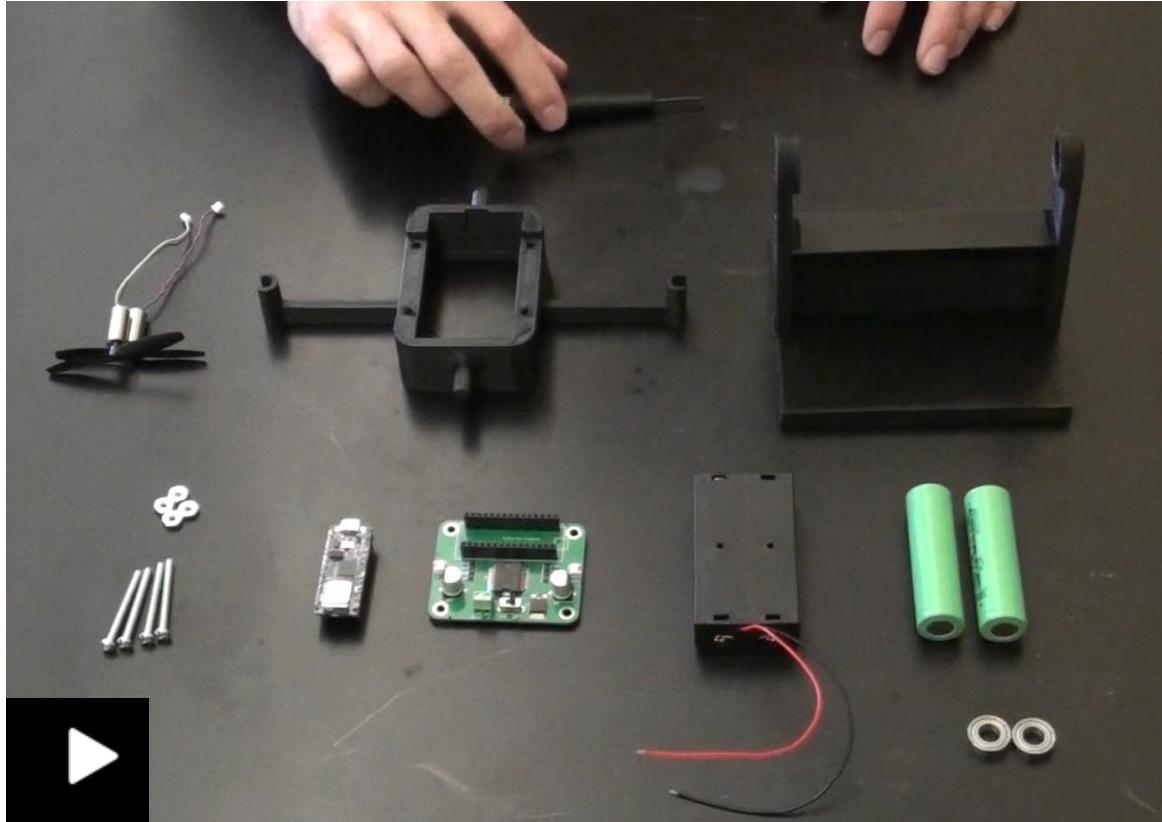


3D printing and assembly instructions are available for the bi-copter hardware

Assembly instructions [[link](#)]

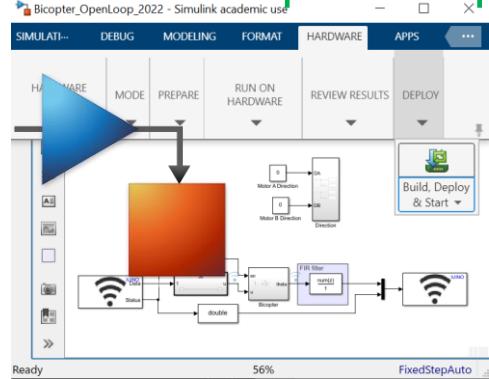
3D printing instructions [[link](#)]

Source Files [[link](#)]



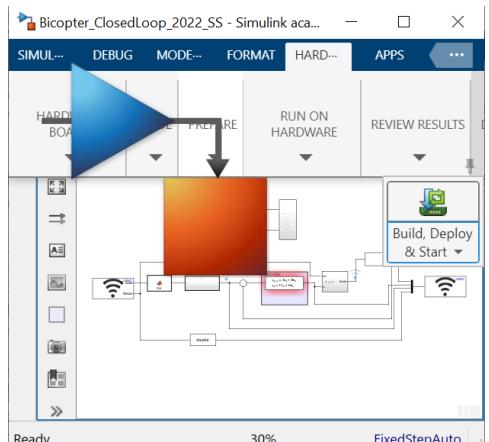
Bi-copter runs a pre-loaded Simulink controller (open- or closed-loop) and sends/receives data to MATLAB Live Script

Simulink Open Loop



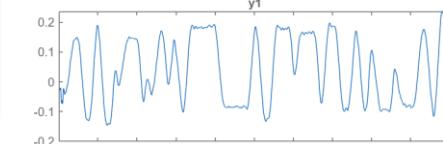
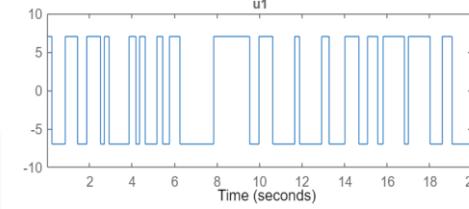
Step 1: Deploy Controller

Simulink Closed-Loop



Parameter Updates
Via Workspace

Step 2: Run Live Script



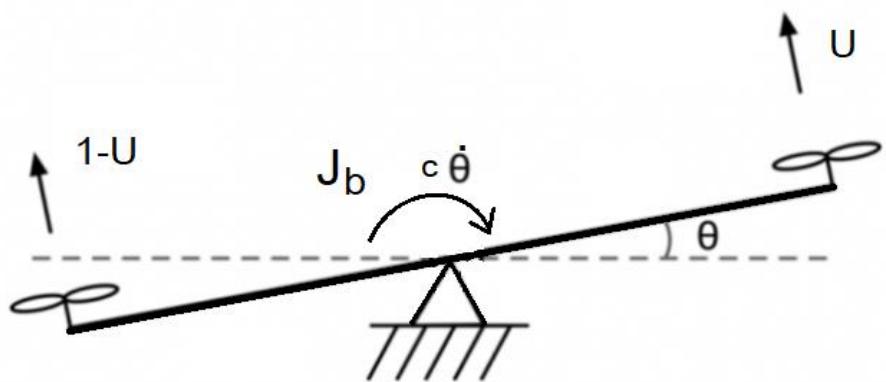
Live Script

Workflow for classical controller design

Workflow



Modeling based on physics (linear approximation)



$$\text{Propeller: } J_m \frac{d\omega_m}{dt} + b\omega_m = K_m i \quad (1)$$

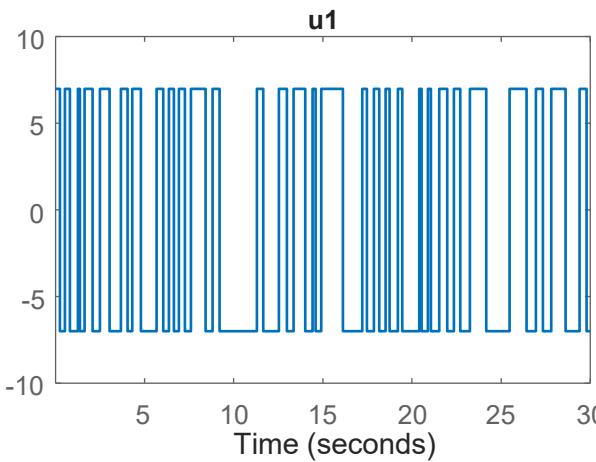
$$\text{DC Motors: } L \frac{di}{dt} + Ri + K_b \omega_m = v, \quad (2)$$

$$\frac{\Omega_m(s)}{V(s)} = \frac{K_m}{(J_m s + b)(L s + R) + K_b K_m}, \quad (3)$$

Overall Model:

$$G(s) = \frac{\Theta(s)}{U(s)} = \frac{K_1}{(J_b s^2 + c s + k)[(J_m s + b)(L s + R) + K_2]}, \quad (4)$$

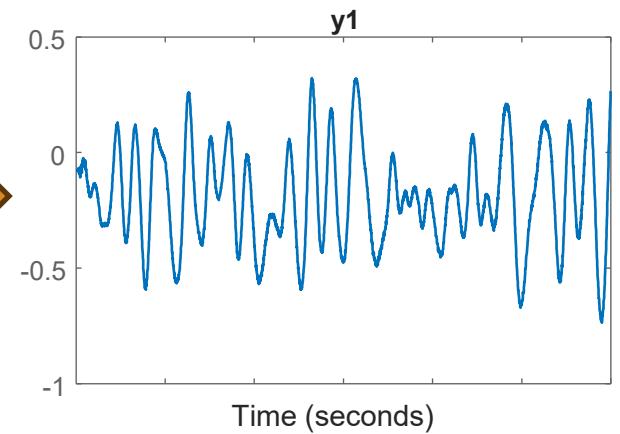
Design of experiments are performed to collect input-output data for system identification



Excite
→



Collect
response
→



All experiments are carried out using MATLAB Live Script interfacing with the bi-copter



```
2 close all
3 IPADDR = '192.168.11.46'
IPADDR = '192.168.11.46'

Task 3: Open Loop Test
To carry out the open-loop test, disconnect the USB cable from the device, and flip the power switch to that the device green LED is on (battery have been loaded in the Arduino).

The input test data can be generated using the MATLAB function idinput() as follows:

4 dur=2000; % number of data points
5 Input = idinput(dur, 'rbs', [0 1/30], [-7 7]);

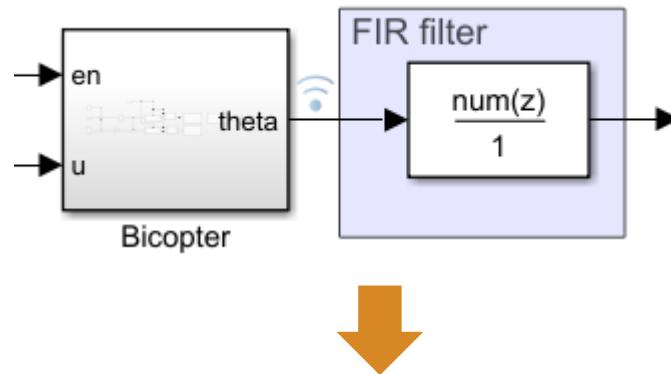
This generates a binary input with a frequency bandwidth from 0 to 0.1 of the Nyquist frequency and a range from -10 to 10. Once data has been generated, run the following script to send it to the Arduino and collect response. Make sure to keep clear from the propellers before running the following sequence!!!

6 tcp = tcpclient(IPADDR,25000);
7 %write(tcp,0,"int8");
8 %pause(1);
9 flush(tcp);
10 write(tcp,Input,'int8');
11 tic
12 en = 0;
13 while en == 0
14     dat = read(tcp,2,'double');
15     en = dat(2);
16 end
17 D = read(tcp,2*dur,'double');
18 Output = D(1:2:end);
```

|| emailarizona-my.sharepoint.com is sharing your screen and audio. [Stop sharing](#) [Hide](#)

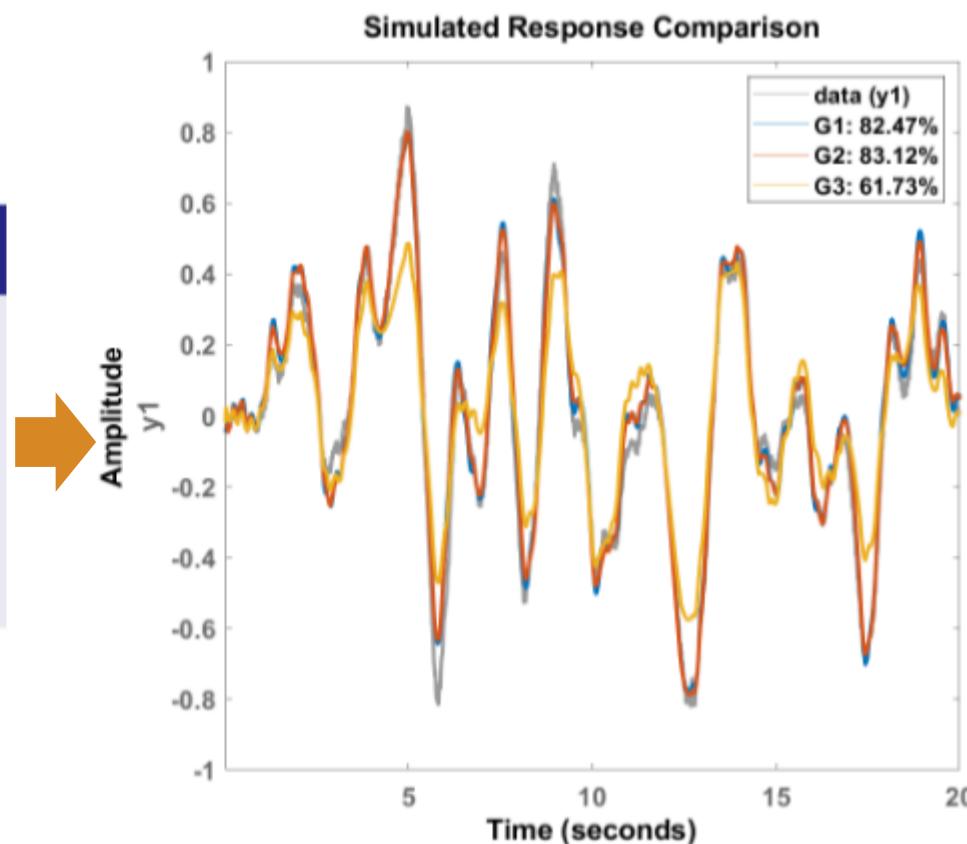
Zoom: 100% | UTF-8 | LF | script | Ln 7 | Col 22

Select most appropriate model



System ID Models

```
G1 = tfest(data,4,3,'Ts',ts)  
G2=armax(data,[4 4 10 0])  
G3 = n4sid(data,4)  
compare(data,G1,G2,G3)
```

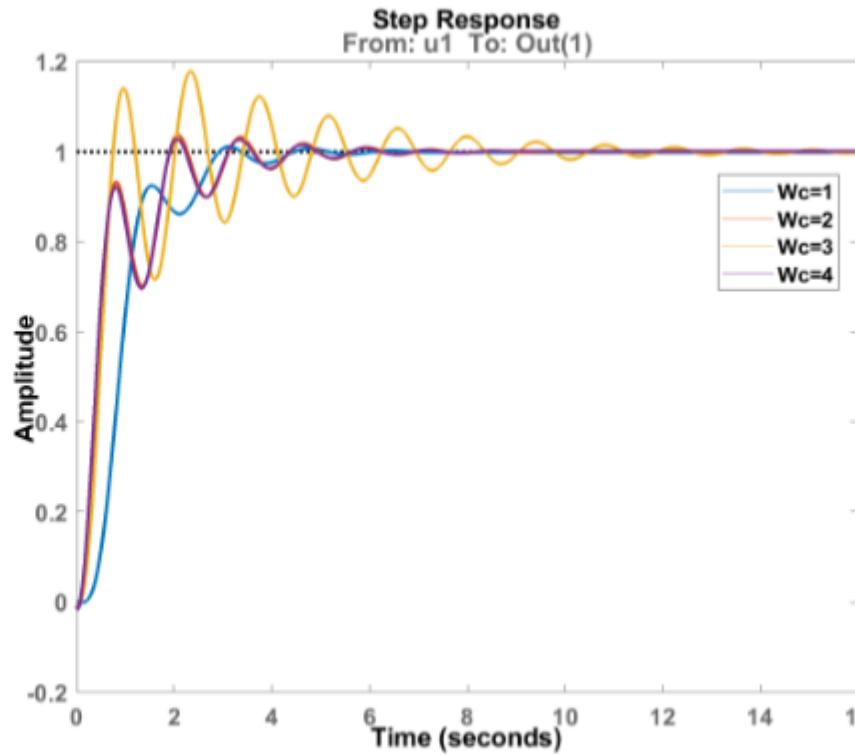


PID controller is tuned using different gain cross-over frequencies



PIDtune

```
C1=pidtune(G2,C0,1)
C2=pidtune(G2,C0,2)
C3=pidtune(G2,C0,3)
C4=pidtune(G2,C0,4)
step(feedback(C1*G2,1),...
feedback(C2*G2,1),...
feedback(C3*G2,1),...
feedback(C4*G2,1))
```



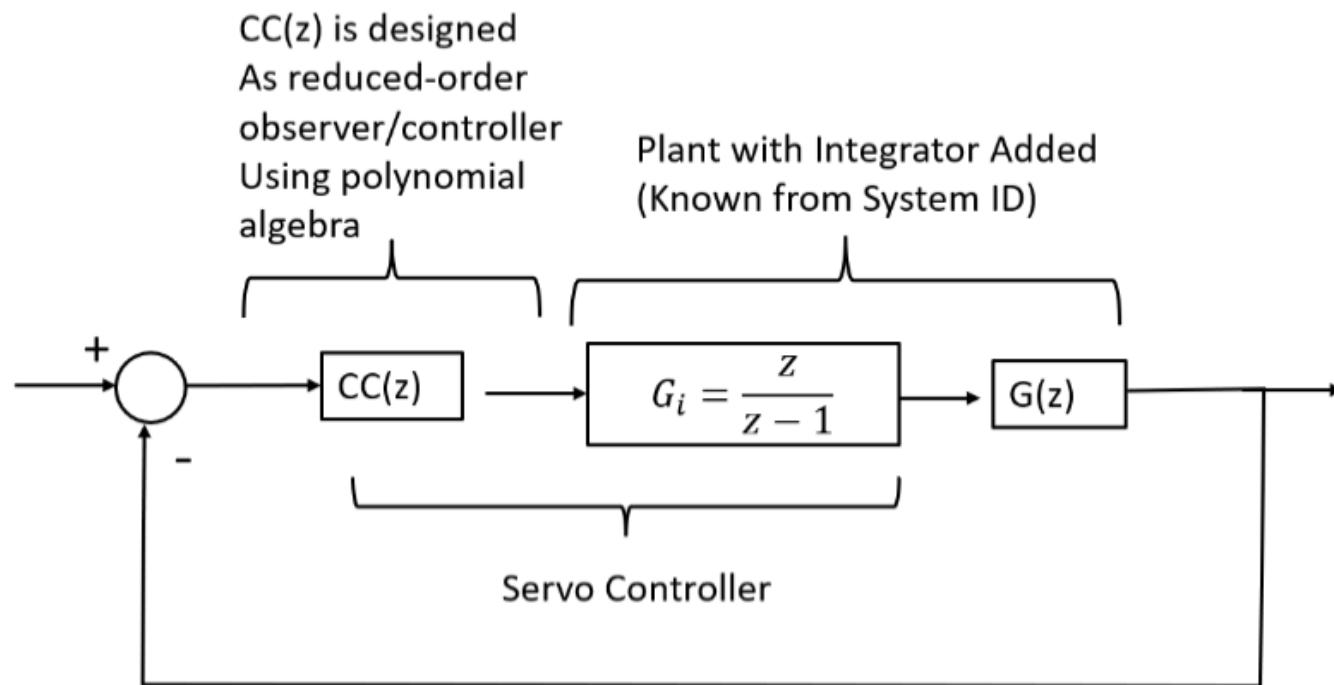
$$C2 = \frac{K_p + K_i * \frac{T_s * z}{z-1} + K_d * \frac{z-1}{T_s * z}}{z-1}$$

with $K_p = -6.79$, $K_i = -23.1$, $K_d = -0.499$, $T_s = 0.01$

Servo controller is designed using discrete linear quadratic regulator method (dLQR function)



$$\int_0^{\infty} y^2 + Ru^2 dt = \int_0^{\infty} x^T C^T C x + Ru^2 dt$$



Optimal pole locations are obtained for various values of the parameter R. Let's see it in practice!



Live Editor - C:\Data\Agencies\SmallGrant\Mathworks\Bicopter\Bicopter_Live.mlx

LIVE EDITOR FIGURE VIEW

FILE NAVIGATE TEXT CODE SECTION

New Open Save Print Go To Find Normal Text Code Control Task Refactor Run

ASMEPAPERFig9.m Bicopter_Live.mlx Report.m

Task 5: Controller Design and Implementation

After fitting a model to the bicopter i/o data, we proceed with the design of a controller. Use Tune PID Controller Live Task (or iin command mode below)

```
31 C0 = pid(1,1,1,'Ts',ts,'IFormula','BackwardEuler','DFormula','BackwardEuler');
32 C=pidtune(G,C0,1.5)

C =
  Ts*z
  Kp + Ki = ----- + Kd = -----
  z-1           Ts*z

with Kp = -5.82, Ki = -71.5, Kd = -0.118, Ts = 0.01

Sample time: 0.01 seconds
Discrete-time PID controller in parallel form.

33 figure;step(feedback(C*G,.5))
```

Step Response
From: u1 To: Out(1)

emailarizona-my.sharepoint.com is sharing your screen and audio. Stop sharing Hide

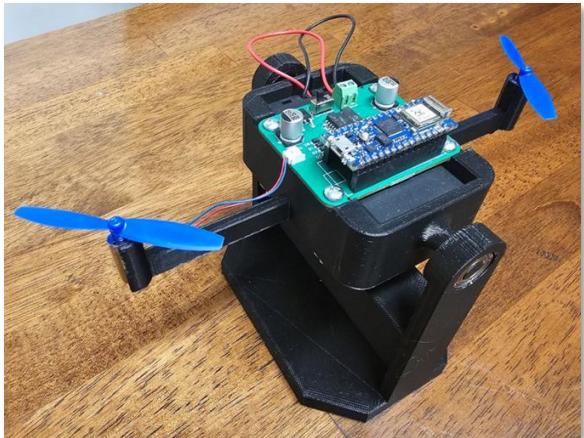
Zoom: 100% UTF-8 LF script Ln 28 Col 20

Closed-loop testing of the servo controller

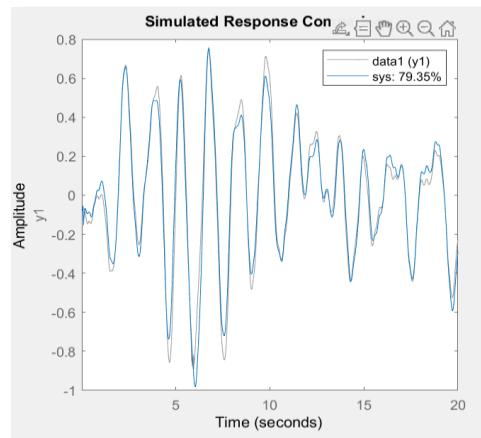


The screenshot shows the MATLAB/Simulink interface for a Bicopter control system. The top navigation bar includes SIMULATION, DEBUG, MODELING, FORMAT, HARDWARE, and APPS. The HARDWARE tab is selected, showing an Arduino Nano 33 IoT board. The Model Browser on the left lists the model 'Bicopter_ClosedLoop_2022_SS'. The main workspace displays a Simulink block diagram for the Bicopter. The diagram includes a 'Read Setpoint' block (Wi-Fi icon) connected to a 'tun' block, which then feeds into a '1' block. The output of the '1' block is connected to a 'Controller' block (represented by a purple rectangle with the equations $x_{t+1} = Ax_t + Bu_t$ and $y_t = Cx_t + Du_t$). The controller's output is connected to a 'Bicopter' block, which is shown as a physical Bicopter model on the right. The Bicopter has two motors, A and B, with direction signals (DA and DB) connected to a 'Direction' block. The Bicopter also has a 'TCP Output' block (Wi-Fi icon). The bottom of the screen shows the Diagnostic Viewer with the message 'Build process completed successfully' and a 'Build Summary' table. The table lists the top model target as 'Bicopter_ClosedLoop_2022_SS' and provides a rebuild reason. The bottom right corner shows a sharing status message: 'emailarizona-my.sharepoint.com is sharing your screen and audio.'

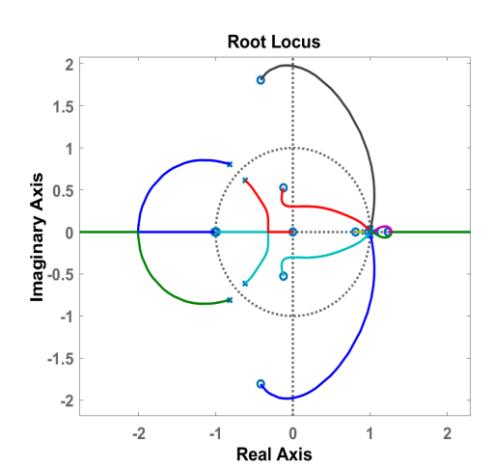
Next, we will focus on MPC design and deployment



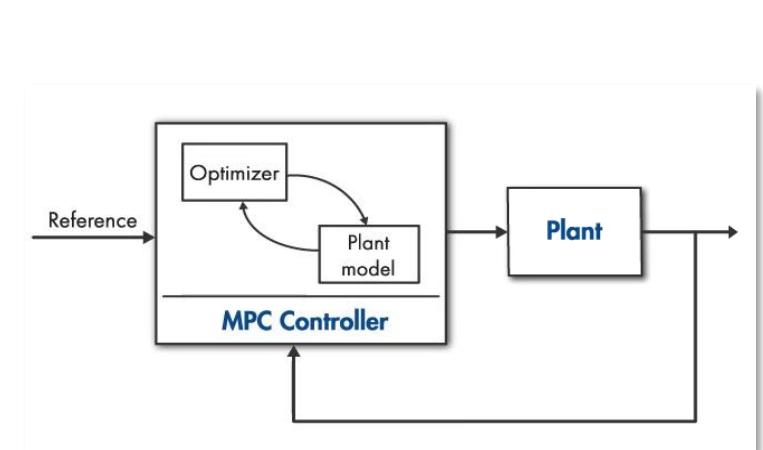
Bi-Copter



Modeling and System Identification

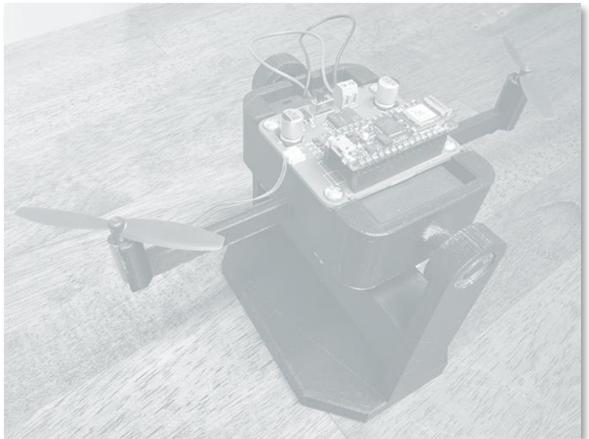


Classical Controller Design

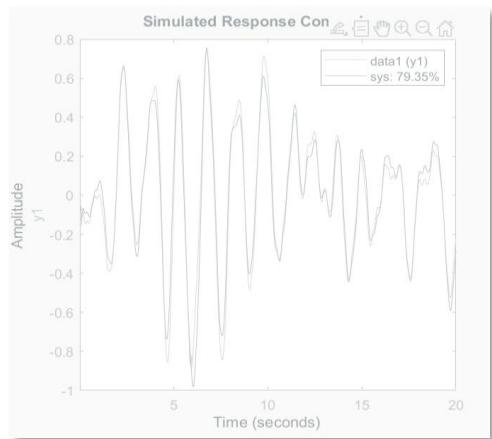


Modern (MPC) Controller Design

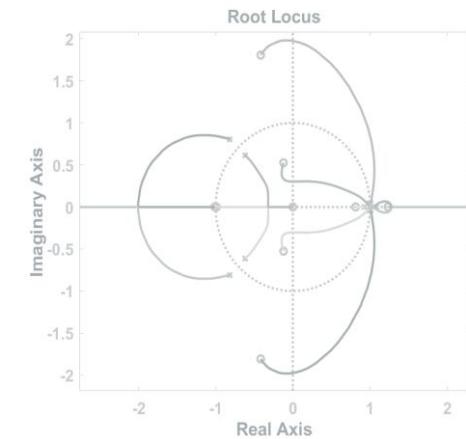
Next, we will focus on MPC design and deployment



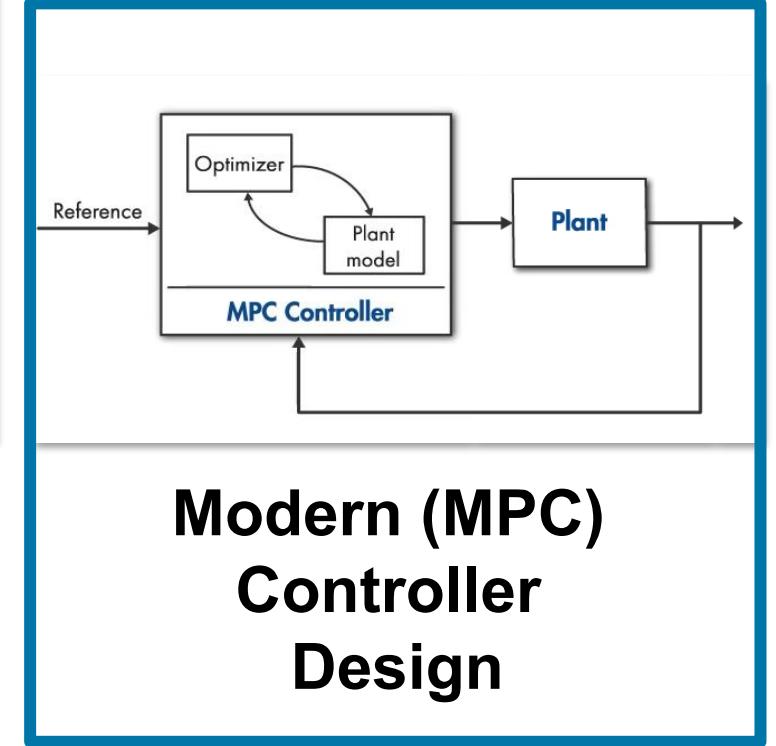
Bi-Copter



Modeling and
System
Identification

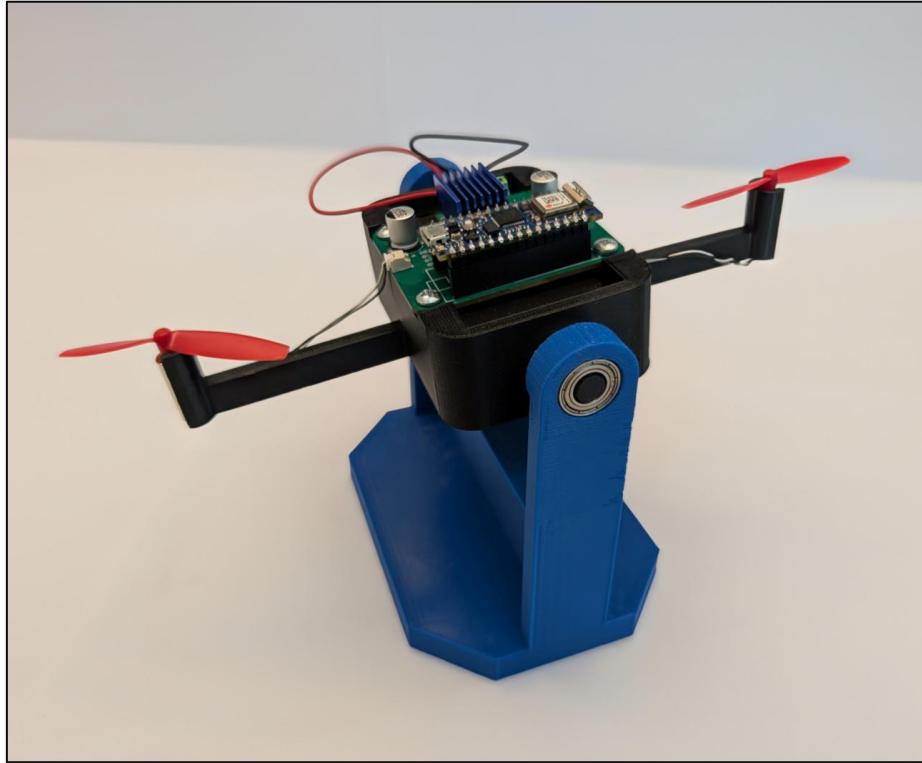


Classical
Controller
Design



**Modern (MPC)
Controller
Design**

In addition to fundamentals, bi-copter hardware also lets educators introduce advanced control concepts such as MPC.



Controller design and deployment

System identification

PID control

Algorithm development

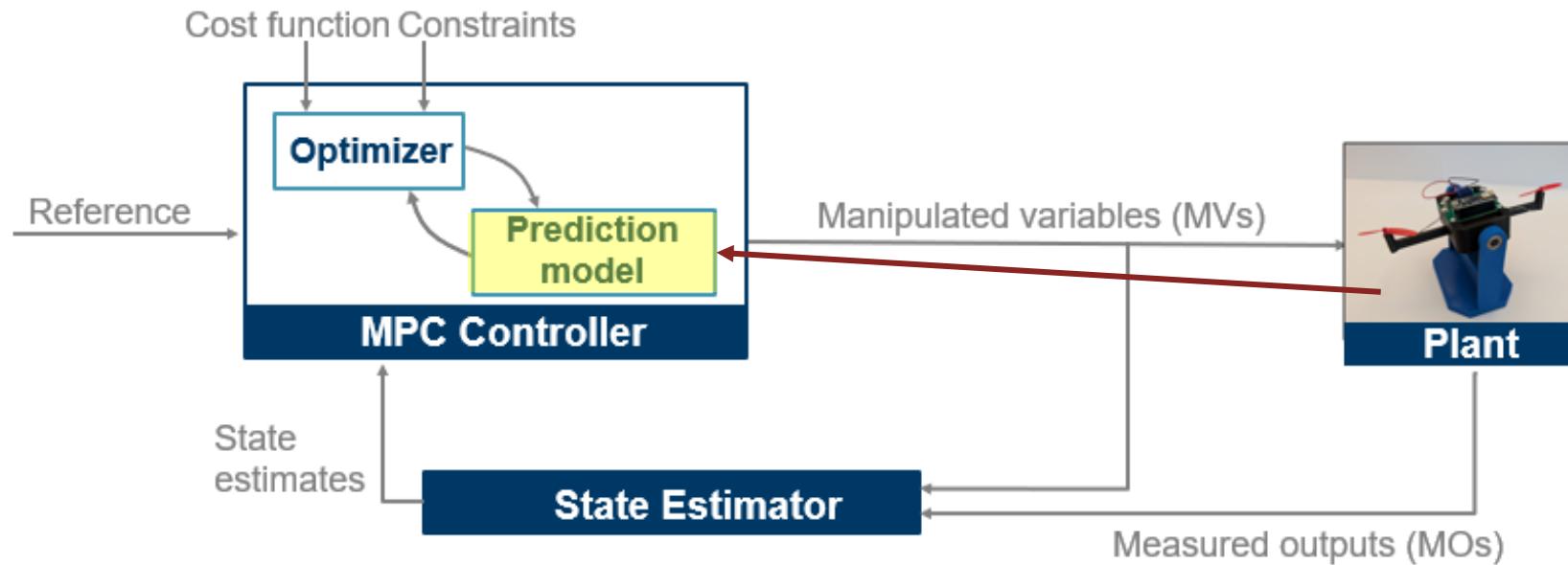
Model predictive control (MPC)

Mechatronics

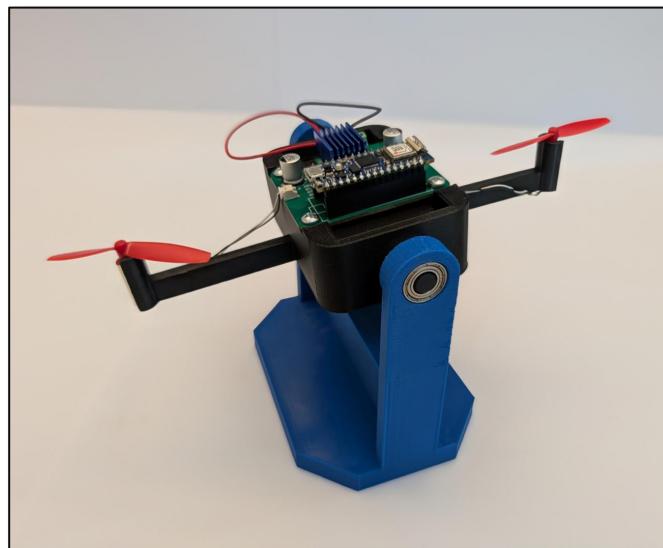
Hands-on learning

Sensors and actuators

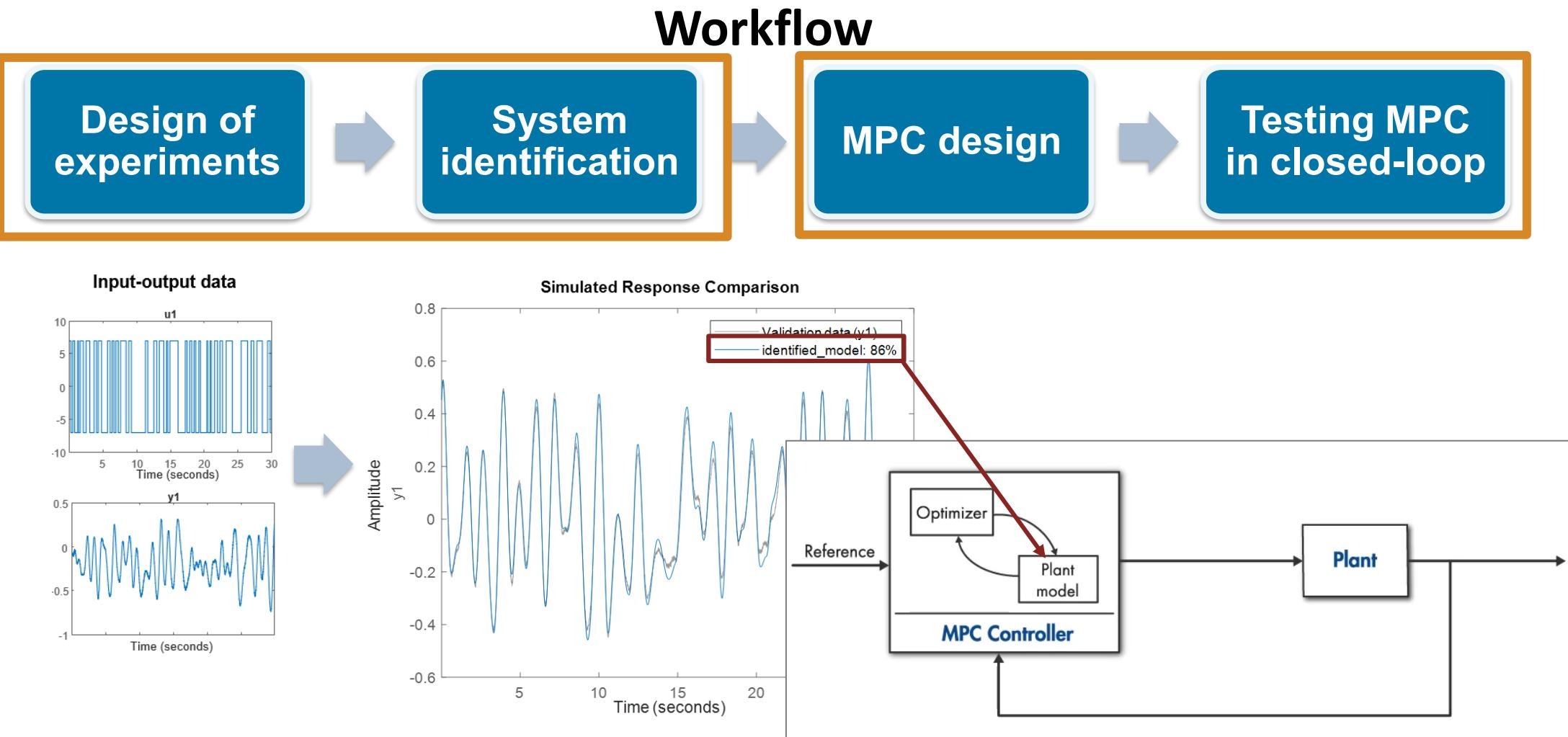
MPC uses a dynamic model of a system to predict future behavior and optimize control actions in real time, subject to constraints.



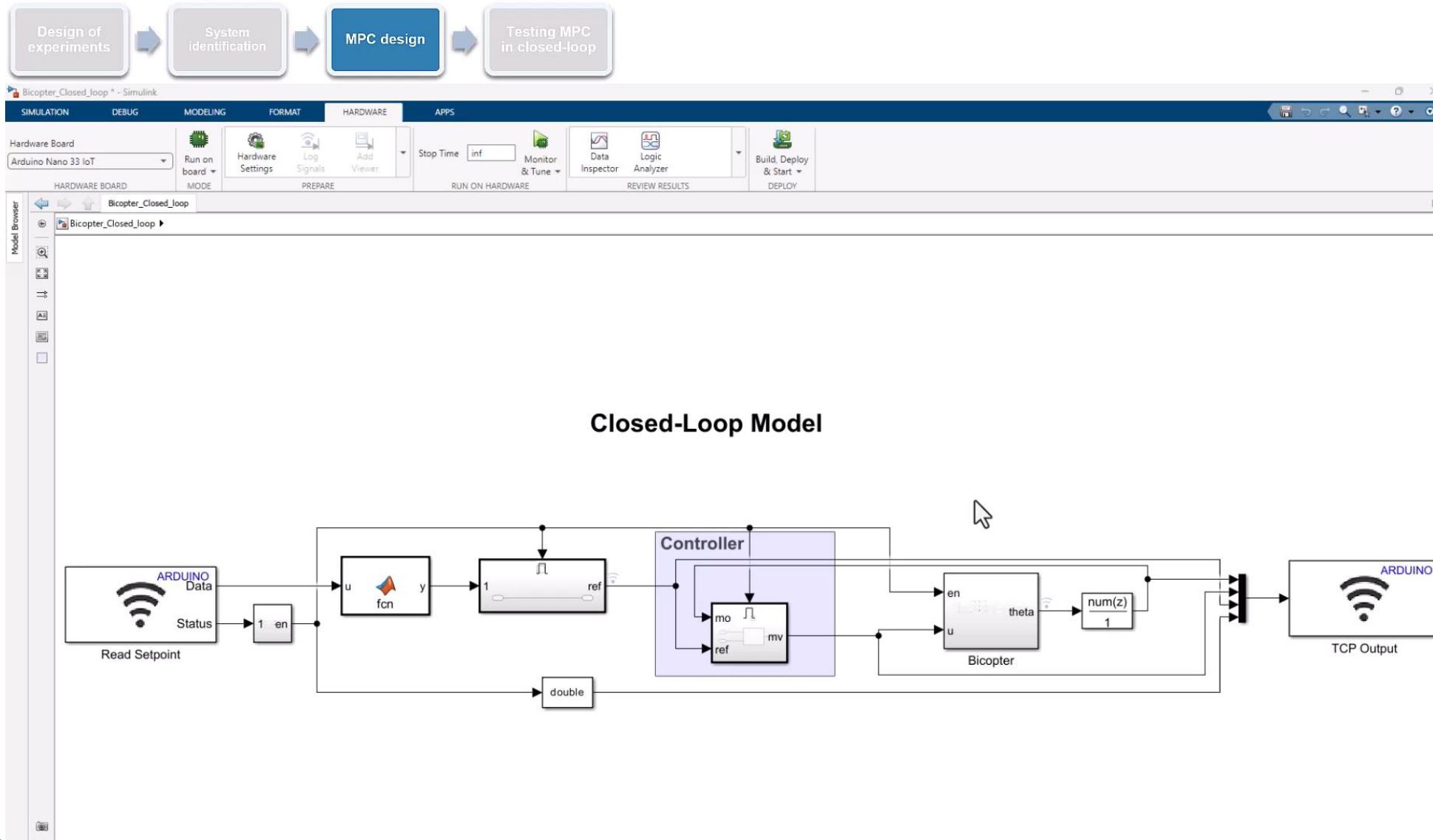
Here's the end-to-end workflow for MPC design and deployment.



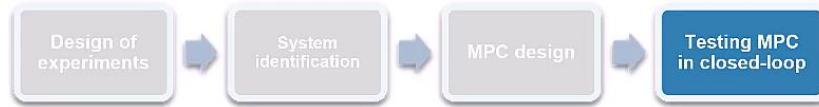
Here's the end-to-end workflow for MPC design and deployment.



An MPC is designed using the MPC Designer app to maintain the bi-copter at the specified reference angles.



Testing the deployed MPC in closed-loop



CAMPCBi_copter_MPC.mlx *

LIVE EDITOR INSERT FIGURE VIEW

FILE + New - Open Save Compare Print Go To Text B I U M Normal Refactor

NAVIGATE Export Bookmarks Section Break Run Run and Advance Run Section Run to End Run Step Stop

TEXT CODE SECTION RUN

Testing the designed MPC in closed-loop

For testing the designed MPC, deploy the closed-loop model to Arduino first.

```
% Generate three-level square wave reference signal
length = 2500;
Ref = zeros(length,1);
Ref(501:1000) = -0.30;
Ref(1501:2000) = 0.30;
plot(Ref);
xlabel("Time [samples]"); ylabel("Reference angle");
```

66
67
68
69
70
71
72

0.3
0.2
0.1
0
-0.1
-0.2
-0.3

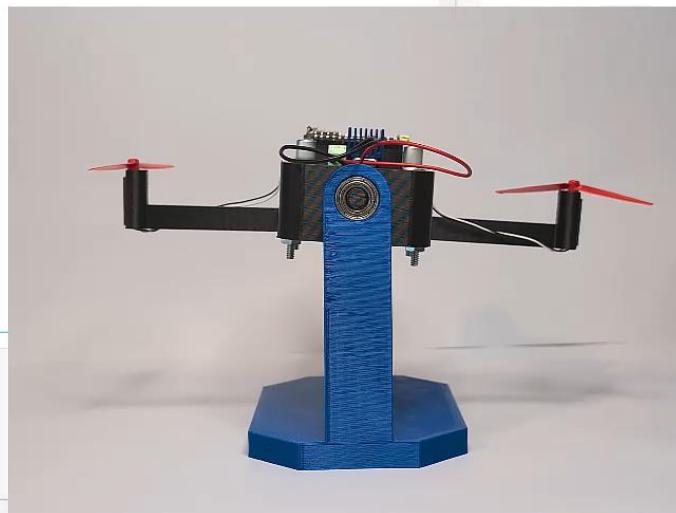
Time [samples]

Reference angle

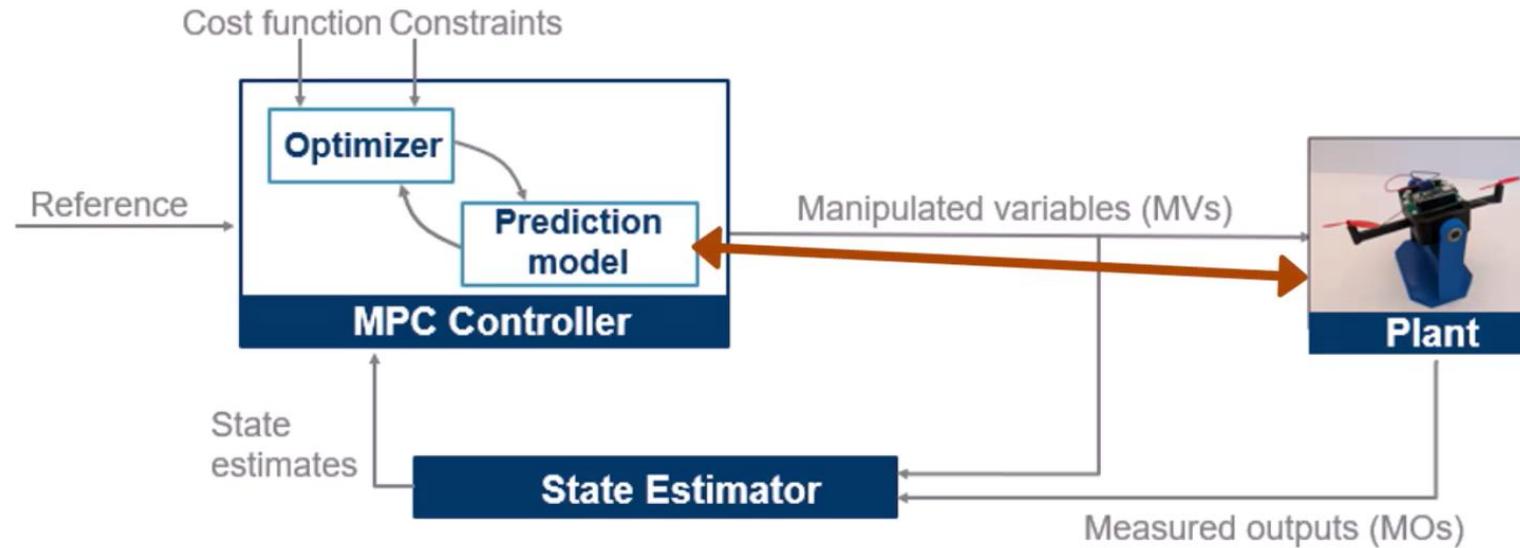
```
% Run experiment to test the designed MPC in closed-loop
IPADDR = '192.168.1.27'; %Typically IP address remains unchanged after deployment of closed-loop model
clear tcp;
tcp = tcpclient(IPADDR,25000);
write(tcp,0, 'single');
pause(5);
flush(tcp);
uinit(tcp);
Ref = 'cinelo'.
```

73
74
75
76
77
78
79
80

Zoom: 110% UTF-8



MPC successfully compensates for the unmodeled disturbance.



In summary, educators can use the bi-copter hardware to provide students with hands-on experience across a wide range of control topics, from fundamentals to advanced controls.

Check out these resources for more information:

- [Prof. Enikov's repository](#) contains:
 - Bi-copter 3D printing and assembly instructions
 - MATLAB code and Simulink models for system identification & PID / state-space control
- MPC with bi-copter
 - [MPC code and models](#)
 - [Tech talk video](#)



System Identification and MPC Design using a Low-Cost Bi-Copter Hardware



MathWorks

@MathWorks

Share the EXPO experience
#MATLABEXPO



in/eniko-enikov-
618b8b106



in/melda-ulusoy

MATLAB EXPO

Thank you



© 2025 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

