Smart Control of 2-Degree of Freedom Helicopters

by

Glenn Janiak and Ken Vonckx
Advisor: Dr. Suruz Miah

Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University

Abstract

This project proposes a modular and cost-effective smart real-time motion control framework for a group of two degrees of freedom (2-DOF) helicopters. The helicopter-sare controlled by a mobile device which sends position signals over a wireless network to a microcontroller. The microcontroller uses control algorithms to determine the amount of voltage to apply to DC motors. This project tests and compares three control algorithms, LQR, LQG, and ADP as well as theorizes possible methods to improve the control in future projects.

Acknowledgements

Special Thanks to Andrew Fandel, Anthony Birge, and Dr. Suruz Miah for their work with Machine Learning on a 2-DOF Helicopter.

Thanks to Mr. Christopher Mattus for his assistance in setting up our lab environment.

Thank you to everyone else who helped make this project possible and sucessful.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Abbreviations

2-DOF - 2 Degrees-Of-Freedom

ADP - Approximate Dynamic Programming

LQG - Linear Quadratic Gaussian

LQR - Linear Quadratic Regulator

RMSE - Root Mean Square Error

SPI - Serial Peripheral Interface

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

USB - Universal Serial Bus

Mathematical Symbols

D_p - viscous damping of the pitch axis

D_y - viscous damping of the yaw axis

J_p - moment of inertia about pitch axis

J_y - moment of inertia about yaw axis

K_{sp} - stiffness of the axes

K_{pp} - pitch motor thrust constant

K_{py} - thrust constant acting on the pitch angle from the yaw motor

K_{yp} - thrust constant acting on yaw angle from pitch motor

K_{yy} - yaw motor thrust constant

Chapter 1

Introduction

1.1 Problem Statement

Helicopters are of a paramount importance as they are used in many civilian and military applications due to their ability for vertical take-off and landing. To enable their use in such applications, intensive research has been conducted to date since helicopters involve complex nonlinear dynamics. Most of the work on helicopter based research requires dedicated computers for controlling their motion to specific configurations and resistant to turbulent conditions. Such methods are expensive and time-consuming to develop. Implementation of motion control techniques using cost-effective hardware is still a challenge.

In this project, we are proposing an algorithm for smart control of a team of two degree-of freedom (two-DOF) helicopters using conventional motion control in cooperation with machine learning techniques where a user will be able to configure helicopters from any initial position. Even though conventional techniques have been tested with simple platforms in the literature, the current project employs conventional motion control strategies in cooperation with machine learning technique (reinforcement learning, for instance) for a team of helicopters as well as introducing user control via mobile devices. This project is expected to encourage research in this area as well as serve as an educational tool in teaching environments.

1.2 Literature Review

Our project requires a great deal of research as some of our tasks have not been attempted before. As a result, we have examined research papers, work complete by other projects at Bradley University, and documentation/teaching materials from Quanser Inc.

Among the major challenges in developing unmanned systems is to implement a modular, cost-effective, and robust teleoperation system, where the motion of a group of helicopters is controlled by mobile devices. In some cases, computer simulations are conducted

to reveal the performance control structures of a 2-DOF helicopter [7]. A large body of research has been conducted in the literature to focus on developing different control structures, such as linear-quadratic regulators (LQR)/Gaussian (LQG), sliding-mode controls (SMC), and advance nonlinear controls, that are specifically applied to 2-DOF helicopters. See [8, 1], for example, and some references therein. Furthermore, soft-computing tools, such as fuzzy-logic, neural networks, and a few combinations of them are employed for controlling the motion of a 2-DOF helicopter [2, 6, 4, 5].

The documentation of Quanser AERO[1] employs LQR and LQG motion control techniques for teaching purposes. This involves creating a linearized system model to calculate the LQR state-feedback gain. A model reference adaptive control (MARC) scheme using Lyapunov functions has been used by [8] for adaptive motion control of a 2-DOF helicopter. SMC is a nonlinear control technique to drive the system states onto a surface in the state space. This method has been used by [1]. Fuzzy-logic controllers use an inference engine to produce an output as used in [2, 6]. The ADP technique does not rely on knowledge of the system model. Instead, it uses data to reconstruct the states as preformed by [4]. Authors in [5] used high order neural networks (HONN) to approximate non-linearities in the system model.

As can be noticed, most of the motion control techniques are either tested using computer simulations or use dedicated computational platforms, that may not be modular and/or cost-effective, for developing motion control algorithms. This is due to the fact that the aforementioned techniques are mainly to propose novel motion control techniques and not focused on hardware implementation platforms. Therefore, the current work considers implementing a conventional motion control technique using modular and cost-effective hardware platforms in the context of a teleoperation system, where a human operator has the ability to control the motion of a team of helicopters using a smart mobile device.

1.3 Report Organization

[1] See https://www.quanser.com/ for details

Chapter 2

Modeling 2-DOF Helicopters

The 2-DOF helicopter, Quanser Aero [manufactured by Quanser Inc. (https://www.quanser.com/)], used in the current work can be configured as a dual-rotor helicopter that has a fixed base. The front rotor (horizontal to the ground) is configured to rotate about pitch axis and the

Figure 2.1: The Quanser Aero 2-DOF used for experiments in the project.

tail rotor (parallel to vertical plane) is mounted to rotate about yaw axis. To measure the pitch (θ) and yaw (ψ) angles, two position sensors (encoders) are mounted as shown in Fig. 2. For sake of simplicity in modeling 2-DOF helicopter, the dynamic coupling of the Quanser Aero is omitted due to high-efficiency rotors mounted on it[1]. The main (tail) rotor is attached to the horizontal (vertical) propeller, which is rotated by applying input voltages to corresponding DC motors. Let $v_p(v_y)$ denote the input voltage to pitch (yaw) DC motor and the state of the 2-DOF helicopter at time $t \geq 0$ is denoted by $\mathbf{x}^T(t) \equiv [\theta(t), \psi(t), \dot{\theta}(t), \dot{\psi}(t)]$, where $\dot{\theta}(\dot{\psi})$ denote the rotational speed of the pitch (yaw). The free-body diagram of the 2-DOF helicopter is shown in Fig. 2.2,

where $F_p(F_y)$ is force from the pitch (yaw) motor. If $\mathbf{u}^T(t) \equiv [v_p(t), v_y(t)]$ denote the vector of the input voltages for DC motors at time $t \geq 0$, then the state-space model of

[1]See Quanser Aero user manual for details.

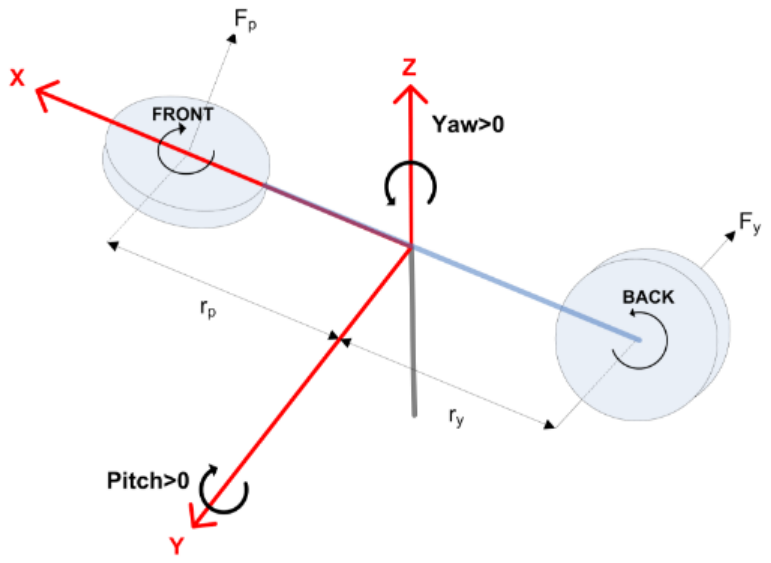


Figure 2.2: Freebody diagram of the forces exerted on a 2-DOF helicopter.

the 2-DOF helicopter (Quanser Aero) is described by [3]:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \text{ where} \tag{2.1}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_{\text{sp}}}{J_p} & 0 & -\frac{D_p}{J_p} & 0 \\ 0 & 0 & 0 & -\frac{D_y}{J_y} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{K_{\text{pp}}}{J_p} & \frac{K_{\text{py}}}{J_p} \\ \frac{K_{\text{yp}}}{J_y} & \frac{K_{\text{yy}}}{J_y} \end{bmatrix},$$

with K_{sp}, K_{pp}, K_{py}, K_{yp}, K_{yy}, J_p, J_y, D_p, and D_y being the stiffness of the axes, pitch motor thrust constant, thrust constant acting on the pitch angle from the yaw motor, thrust constant acting on yaw angle from pitch motor, yaw motor thrust constant, moment of inertia about pitch axis, moment of inertia about yaw axis, viscous damping of the pitch axis, and viscous damping of the yaw axis, respectively.

In the context of a teleoperation system, a standard problem in controlling motion of states of a 2-DOF helicopter is to find optimal actuator commands $\mathbf{u}^*(t) \equiv [v_p^*(t), v_y^*(t)]^T$, such that it follows the command signal, which is the desired (reference) state $\mathbf{x}^d = [\theta^d, \psi^d, 0, 0]^T$, sent by the human operator through a mobile device.

Chapter 3

Control Algorithms

3.1 LQR

After creating the system model

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

use the state feedback law

$$\mathbf{u} = -\mathbf{K}\mathbf{x}$$

to minimize the quadratic cost function:

$$J(\mathbf{u}) = \int_0^\infty (\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + 2\mathbf{x}^T\mathbf{N}\mathbf{u})\mathrm{dt}$$

Find the solution $\mathbf{S}$ to the Riccati equation

$$\mathbf{A}^T\mathbf{S} + \mathbf{S}\mathbf{A} - (\mathbf{S}\mathbf{B} + \mathbf{N})\mathbf{R}^{-1}(\mathbf{B}^T\mathbf{S} + \mathbf{N}^T) + \mathbf{Q} = 0$$

Calculate gain, $\mathbf{K}$

$$\mathbf{K} = \mathbf{R}^{-1}(\mathbf{B}^T\mathbf{S} + \mathbf{N}^T)$$

3.2 LQG

- Utilizes gain calculated in LQR
- Added Kalman filter to reduce external disturbances to the system

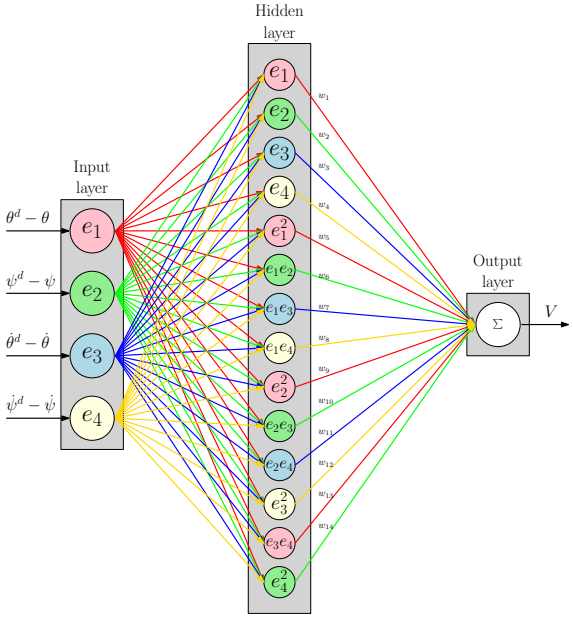


Figure 3.1: Input nodes are connected to the hidden layer and then to the output layer.

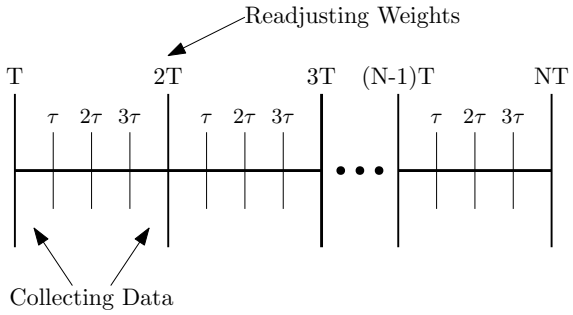


Figure 3.2: Data is collected for every τ seconds and then the weights are adjusted every T seconds.

3.3 ADP

3.4 Improvements

Most of these algorithms are typically implemented only using proportional gain which causes steady-state error to certain input signals in some systems. In our case, we experience steady-state error for a step input. To reduce this, the type of the controller needs to be increased by adding an integrator. We solve this problem by implementing a PI controller where LQR and ADP are used to find the optimal proportional and integral gain.
This is done by creating an augmented matrix with A and B with different dimensions

Chapter 4

Numerical Simulations

4.1 LQR (P controller)

Using MATLAB, we created a program, as seen in B.1, that calculates LQR for our helicopter and then uses differential equations to simulate the trajectory. 4.1(a) 4.1(b) 4.1(c)

4.2 LQR (PI controller)

Insert Block diagram for LQR PI simulation

Insert results for LQR PI simulation

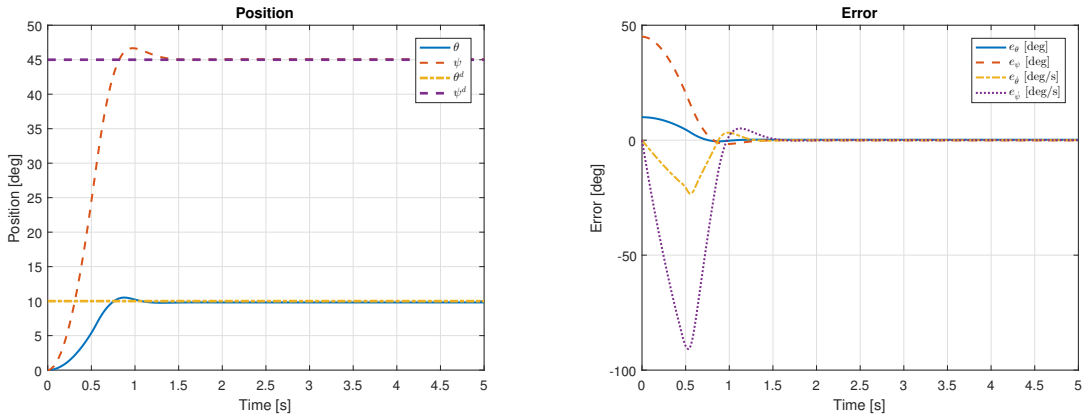
4.3 LQG (PI Controller)

Insert Block diagram for LQR PI simulation

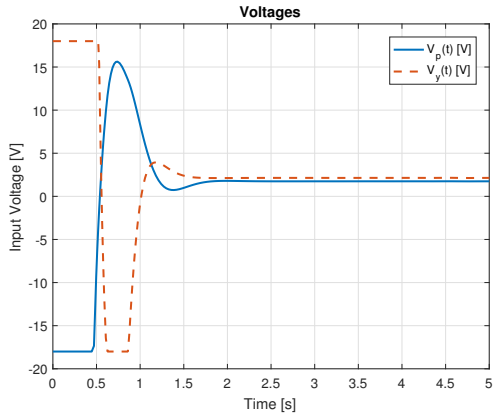
Insert results for LQR PI simulation

4.4 Conclusions

Note: constant used pitch XXXX degrees, yaw XXXX degrees
Note: square used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Note: sine used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Based on the results XXXX preformed better in the simulations.



(a) (b)



(c)

Figure 4.1: Simulations for proportional gain calculated by LQR.

	LQR(P)	LQR(PI)	LQG(PI)
RMSE Pitch Step	?	?	?
RMSE Yaw Step	?	?	?
RMSE Pitch Square	?	?	?
RMSE Yaw Square	?	?	?
RMSE Pitch Sine	?	?	?
RMSE Yaw Sine	?	?	?

Table 4.1: Error Comparison for Simulated Algorithms

Chapter 5

Implementation

5.1 Experimental Setup

5.1.1 USB

5.1.2 Raspberry Pi

The communication protocol utilized by the Q-Flex2 embedded panel is SPI.

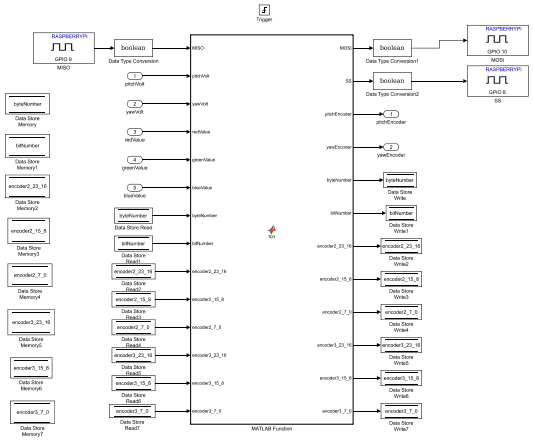


Figure 5.1: Block diagram of SPI communication protocol used for communication between the Raspberry Pi and Quanser Aero.

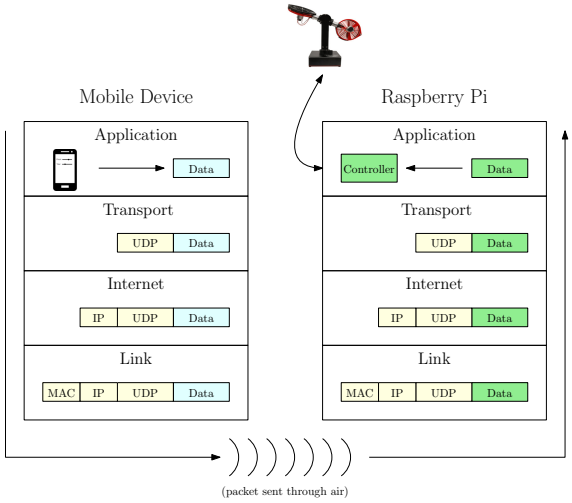


Figure 5.2: Illustration of the TCP model which describes how packets are sent and recieved from mobile devices to the Quanser Aero.

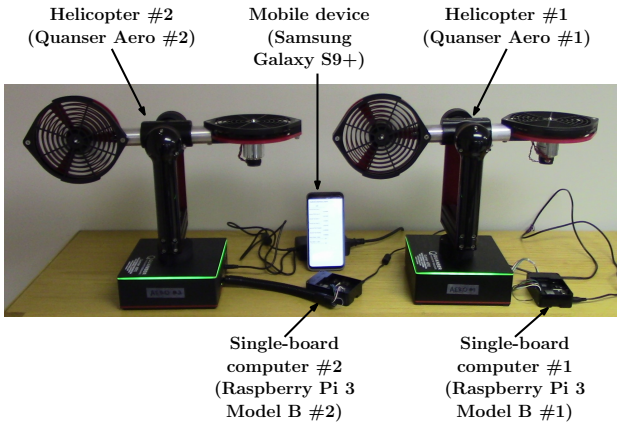


Figure 5.3: Experiment setup for controlling the Quanser Aeros via a mobile device.

5.1.3 Android

5.2 USB

5.2.1 LQR

5.2.2 LQG (PI Controller)

5.2.3 ADP

5.2.4 Conclusions

Note: constant used pitch 10 degrees, yaw 45 degrees
Note: square used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Note: sine used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Based on the results XXXX preformed better for USB.

	LQR(P)	**LQR(PI)**	**ADP(P)**
RMSE Pitch Step	2.6454	?	1.3067
RMSE Yaw Step	5.7991	?	6.1991
RMSE Pitch Square	?	?	?
RMSE Yaw Square	?	?	?
RMSE Pitch Sine	?	?	?
RMSE Yaw Sine	?	?	?

Table 5.1: Error Comparison for USB Algorithms

5.3 Raspberry Pi

5.3.1 LQR (P Controller)

5.3.2 ADP

5.3.3 Conclusions

Note: constant used pitch XXXX degrees, yaw XXXX degrees
Note: square used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX

Note: sine used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Based on the results XXXX preformed better for Raspberry Pi.

	LQR(P)	ADP(P)
RMSE Pitch Step	?	?
RMSE Yaw Step	?	?
RMSE Pitch Square	?	?
RMSE Yaw Square	?	?
RMSE Pitch Sine	?	?
RMSE Yaw Sine	?	?

Table 5.2: Error Comparison for USB Algorithms

5.4 Mobile Device

5.4.1 LQR (P Controller)

5.4.2 ADP

5.4.3 Conclusions

Note: constant used pitch XXXX degrees, yaw XXXX degrees
Note: square used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Note: sine used pitch XXXX degrees with period of XXXX, yaw XXXX degrees with period of XXXX
Based on the results XXXX preformed better for Android.

	LQR(P)	ADP(P)
RMSE Pitch Step	?	?
RMSE Yaw Step	?	?
RMSE Pitch Square	?	?
RMSE Yaw Square	?	?
RMSE Pitch Sine	?	?
RMSE Yaw Sine	?	?

Table 5.3: Error Comparison for USB Algorithms

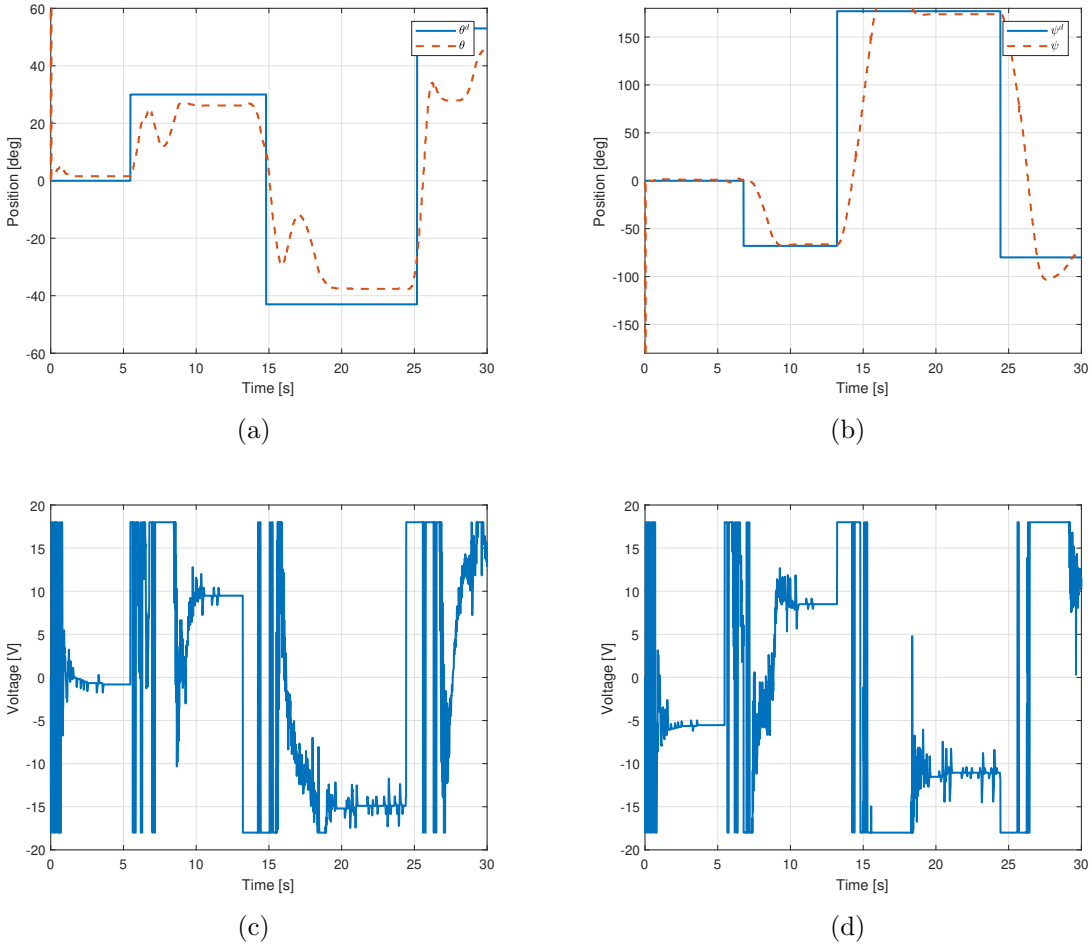


Figure 5.4: LQR Android Results

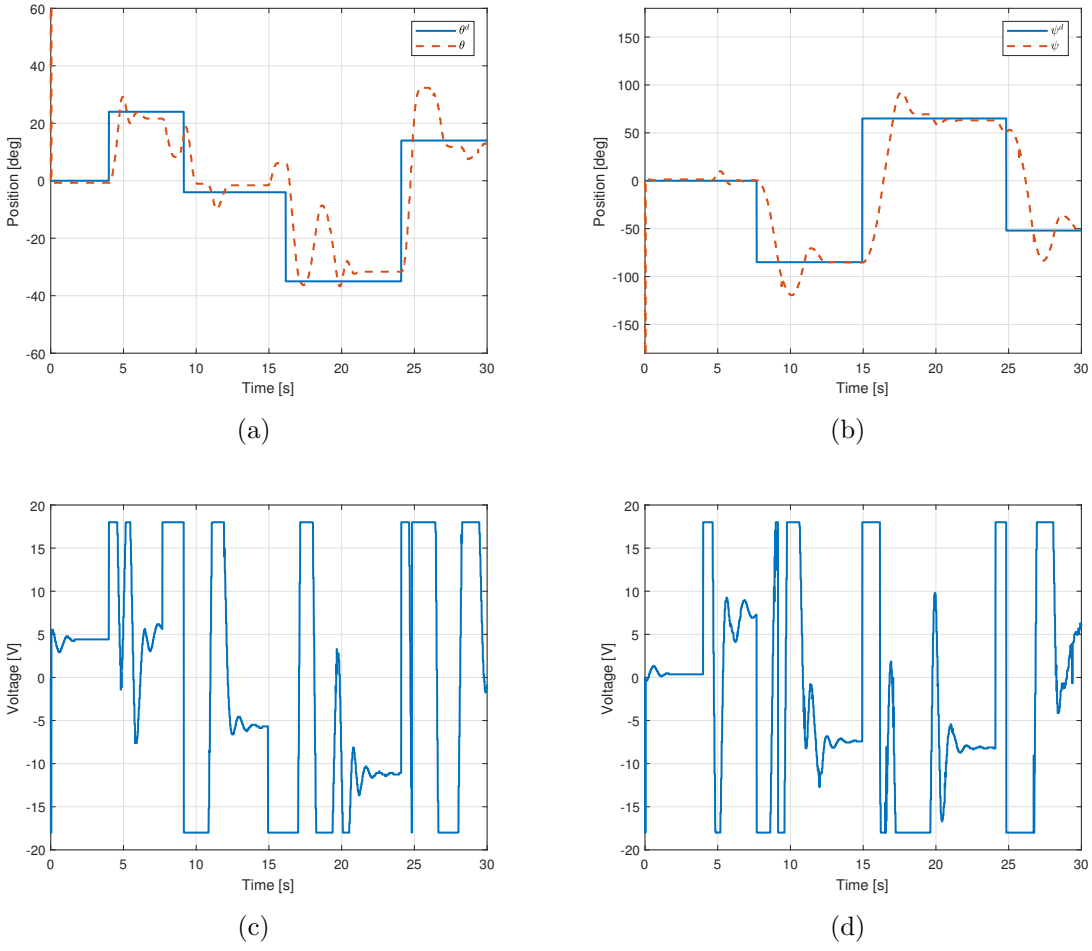


Figure 5.5: ADP Android Results

Chapter 6

Conclusion and Future Work

6.1 Algorithm Comparison

6.2 Conclusion

6.3 Future Work

6.3.1 Enhanced Smart Framework

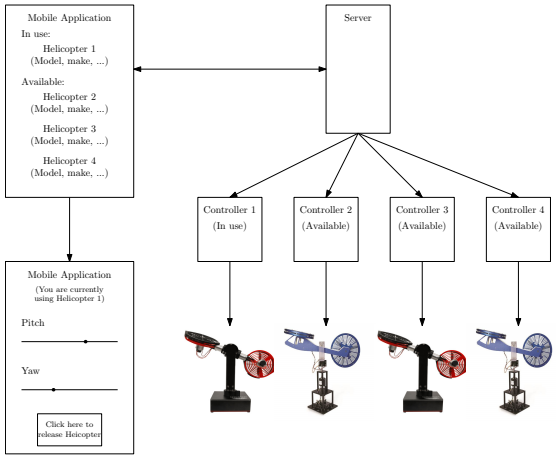


Figure 6.1: Smart Algorithm Server Connectivity

6.3.2 Digital Compass

6.3.3 Expanded PI Control

We had trouble implementing the PI control architecture as well as LQG on the Raspberry Pi. This may be because the integrator and the kalman filter require a fixed sampling time. Simulink has the option to use a variable sampling time when set to continuous time. Since the model is being loaded onto a embedded system which relies upon a fixed sampling time, this may be affected the values being outputted by these blocks.
To correct this issue, we recommend that future work done on this project convert the model to discrete time. Consider 6.1 and 6.2 to convert the continuous model in 2.1:

$$x(k+1) = \Phi x(k) + \Gamma u(k) \tag{6.1}$$

$$y(k) = Hx(k) + Ju(k) \tag{6.2}$$

where T is the sampling time, $\Phi = e^{AT}$, and $\Gamma = \int_0^T e^{A\eta} d\eta B$.

Appendix A

Parameters and State-Space

A.1 Parameters

Symbol	Description	Value
DC Motor		
V_{nom}	Nominal input voltage	18.0 V
τ_{nom}	Nominal torque	22.0 mN-m
ω_{nom}	Nominal speed	3050 RPM
I_{nom}	Nominal current	0.540 A
R_m	Terminal resistance	8.4 Ω
k_t	Torque constant	0.042 N-m/A
k_m	Motor back-emf constant	0.042 V/(rad/s)
J_m	Rotor inertia	4.0×10^{-6} kg-m^2
L_m	Rotor inductance	1.16 mH
Aero Body		
M_b	Mass of body	1.075 kg
D_m	Center of mass	-7.59 mm
J_p	Pitch inertia	2.15×10^{-2} kg-m^2
J_y	Yaw inertia	2.37×10^{-2} kg-m^2
D_t	Thrust displacement	15.8 cm
Motor and Pitch Encoders		
	Encoder line count	512 lines/rev
	Encoder line count in quadrature	2048 lines/rev
	Encoder resolution (in quadrature)	0.176 deg/count
Yaw Encoder		
	Encoder line count	1024 lines/rev
	Encoder line count in quadrature	4096 lines/rev
	Encoder resolution (in quadrature)	0.088 deg/count
Amplifier		
	Amplifier type	PWM
	Peak Current	2 A
	Continuous Current	0.5 A
	Output voltage range (recommended)	$\pm$18 V
	Output voltage range (maximum)	$\pm$24 V

Figure A.1: Quanser Aero System Parameters

```
%
%% Quanser Aero Parameters
% Moment of Inertia of helicopter body (kg-m^2)
L_body = 6.5*0.0254; % length of horizontal body (metal tube)
m_body = 0.094; % mass of horizontal body (metal tube)
J_body = m_body * L_body^2 / 12; % horizontal cylinder rotating about CM
%
% Moment of Inertia of yoke fork that rotates about yaw axis (kg-m^2)
m_yoke = 0.526; % mass of entire yoke assembly (kg)
% h_yoke = 9*0.0254; % height of yoke assembly (m)
r_fork = 0.04/2; % radius of each fork (approximated as cylinder)
J_yoke = 0.5*m_yoke*r_fork^2;
%
% Moment of Inertia from motor + guard assembly about pivot (kg-m^2)
m_prop = 0.43; % mass of dc motor + shield + propeller shield
% m_motor = 0.203; % mass of dc motor
r_prop = 6.25*0.0254; % distance from CM to center of pitch axis
J_prop = m_prop * r_prop^2; % using parallel axis theorem
%
% Equivalent Moment of Inertia about Pitch and Yaw Axis (kg-m^2)
Jp = J_body + 2*J_prop; % pitch: body and 2 props
Jy = J_body + 2*J_prop + J_yoke; % yaw: body, 2 props, and yoke
%
% Thrust-torque constant (N-m/V) [found experimentally]
Kpp = 0.0011; % (pre-production unit: 0.0015)
Kyy = 0.0022; % (pre-production unit: 0.0040)
Kpy = 0.0021; % thrust acting on pitch from yaw (pre-production unit: ...
    0.0020)
Kyp = -0.0027; % thrust acting on yaw from pitch (pre-production ...
    unit: -0.0017)
%
% Stiffness (N-m/rad)[found experimentally]
Ksp = 0.037463;
%
% Viscous damping (N-m-s/rad) [found experimentally]
Dp = 0.0071116; % pitch axis (pre-production unit: Dp = 0.0226)
Dy = 0.0220; % yaw axis (pre-production unit: Dy = 0.0211)
%
```

A.2 State-Space Model

```
%% State-Space Representation
A = [0 0 1 0;
     0 0 0 1;
     -Ksp/Jp 0 -Dp/Jp 0;
     0 0 0 -Dy/Jy];

B = [0 0;
     0 0;
```

```
9       Kpp/Jp Kpy/Jp;
10      Kyp/Jy Kyy/Jy];
11
12  C = eye(2,4);
13  D = zeros(2,2);
14  %
```

Appendix B

MATLAB Simulation Code

B.1 LQR (P controller)

```
function KV_LQRContinuousTime
%% Setup Differential Equation
close all
clear
clc

quanser_aero_parameters;  % contains parameters Quanser Aero
quanser_aero_state_space;  % puts parameters into state-space model

Q = diag([3500 500 0 0 ]); % Quadratic term
R = 0.005*eye(2,2);  % Regulator term
K = lqr(A,B,Q,R);

xInit = zeros(1,4);  % set the initial conditions to 0 degrees pitch
                     % and 0 degrees yaw
xD = [deg2rad(10) deg2rad(45) 0 0]';  % desided position is 45 degrees
                                      % pitch and 90 degrees yaw
eInit = xD'-xInit;  % error matrix is desired transposed minus actual

maxInputVoltage = 18; % cannot exceed this output in [V]

options = odeset('RelTol', 1e-4, 'AbsTol', 1e-4*ones(1,4));

t_sim = 10; % [sec]
[Te,E] = ode45(@(t,e) stateEqn(t,e,A,B,K,xD,maxInputVoltage),[0 ...
    t_sim/2],...
     eInit, options);
xD=-xD;
[Te,E] = ode45(@(t,e) stateEqn(t,e,A,B,K,xD,maxInputVoltage),[t_sim/2 ...
    t_sim],...
     eInit, options);
%% Position Error Graph
figure;
```

```
e_1 = plot(Te,rad2deg(E(:,1)),'-','LineWidth', 1.5);
hold on
e_2 = plot(Te,rad2deg(E(:,2)),'--','LineWidth', 1.5);
hold on
e_3 = plot(Te,rad2deg(E(:,3)),'-.','LineWidth', 1.5);
hold on
e_4 = plot(Te,rad2deg(E(:,4)),':','LineWidth', 1.5);

xlabel('Time [s]');
ylabel('Error [deg]');
title('Error');
legend([e_1 e_2 e_3 e_4],{'$e_\theta$¬[deg]', '$e_\psi$¬[deg]',...
    '$e_{\dot\theta}$¬[deg/s]','$e_{\dot\psi}$¬[deg/s]'},...
    'Interpreter', 'latex');

grid on

%% Position Graph
for i=1:size(E,1)
    u(i,:) = (K*(E(i,:))')';
    uLimit(i,:) = sign(u(i,:)).*min(abs(u(i,:)),[maxInputVoltage,...
        maxInputVoltage]);
end

xDt = repmat(xD',size(E,1),1); % repeat xDt for all time instants
QuanserAeroStates = xDt - E;

figure;
theta = plot(Te,rad2deg(QuanserAeroStates(:,1)),'-','LineWidth', 1.5);
hold on
psi = plot(Te,rad2deg(QuanserAeroStates(:,2)),'--','LineWidth', 1.5);
hold on
thetaD = plot(Te,rad2deg(xDt(:,1)),'-.','LineWidth', 2);
hold on
psiD = plot(Te,rad2deg(xDt(:,2)),'--','LineWidth', 2);

xlabel('Time [s]');
ylabel('Position [deg]');
title('Position');
legend([theta psi thetaD psiD],{'$\theta$', '$\psi$','${\theta^d}$',...
    '${\psi^d}$'}, 'Interpreter', 'latex');
grid on

%% Voltage Graph
figure;
vP = plot(Te,uLimit(:,1),'-','LineWidth', 1.5);
hold on
vY = plot(Te,uLimit(:,2),'--','LineWidth', 1.5);

xlabel('Time [s]');
ylabel('Input Voltage [V]');
title('Voltages');
legend([vP vY],'V_p(t) [V]', 'V_y(t) [V]');
grid on
```

```
86
87 function eDot = stateEqn(t,e,A,B,K,xD,uMax)
88     u = K*e;
89     uLimit = sign(u).*min(abs(u),[uMax;uMax]);
90     eDot = A*e-B*uLimit - A*xD;
```

B.2 LQR (PI controller)

B.3 LQG (PI Controller)

Appendix C

USB MATLAB Code

Appendix D

Raspberry Pi MATLAB Code

D.1

D.2 Initialization Code

```
1
2  %rpi = raspi('192.168.1.79','pi','raspberry');   %Raspberry Pi 1
3  rpi = raspi('192.168.1.20','pi','raspberry');   %Raspberry Pi 2
4
5  enableSPI(rpi);
6  disableSPI(rpi);
7
8  % Configure pins on Raspberry Pi for the SPI comunication
9  % MOSI Output
10 configurePin(rpi, 10,'DigitalOutput');
11 % SPI Clock Output
12 configurePin(rpi, 11,'DigitalOutput');
13 % SS Output
14 configurePin(rpi, 8,'DigitalOutput');
15 % MISO Input
16 configurePin(rpi, 9,'DigitalInput');
17
18 % SPI output clock period 2us (500kHz)
19 spiPeriod = 0.0001;
20
21 enableSPI(rpi);
22
23 clearvars -except rpi spiPeriod K
```

Appendix E

Tutorials

E.1 USB Connection

1. Verify QuaRC is installed on computer
2. Install QFLEX 2 USB into Quanser Aero as seen in Figure E.1
3. Open Simulink model
4. Connect Quanser Aero to computer via the included USB adapter cable
5. Set simulation mode to external
6. Run MATLAB initialization code for motion controller
7. Click "Build Model" button
8. Click "Connect To Target" button
9. Click "Run" button

Figure E.1: Quanser Aero with QFLEX 2 USB panel installed

E.2 Raspberry Pi Implementation

1. Make sure software add ons are installed on MATLAB
   (a) MATLAB Support Package for Raspberry Pi Hardware
   (b) Simulink Support Package for Raspberry Pi Hardware
2. Install QFLEX 2 Embedded into Quanser Aero as seen in Figure E.2
3. Assuming Raspberry PI is not connected to network, establish a serial connection to the Raspberry PI
   (a) Plug SD card into Linux computer and enable serial connectivity
   (b) Insert SD card back into Raspberry Pi
   (c) Connect serial cable to the Raspberry Pi, Pin 3 (black wire), Pin 4 (white wire), Pin 5 (green wire)
   (d) Plug Raspberry Pi into commputer
   (e) Locate COM port for the connection
   (f) Open Putty, select Serial connection, input COM number and 115200 as the speed
   (g) Log into Raspberry Pi
   (h) Configure network, type "sudo nano /etc/wpa_supplicant/wpa_supplicant.conf" and add network settings
4. Connect SPI wires to Raspberry PI. Pin 1 (white wire) on the embedded panel connects to +5V on the Pi. Pin 2 (yellow wire) connects to GPIO 10 on the Pi. Pin 3 (blue wire) connects to GPIO 9 on the Pi. Pin 4 (green wire) connects to GPIO 11 on the Pi. Pin 6 (purple wire) connects to GPIO 8 on the Pi. Pin 7 (Red wire) connects to ground on the Pi.
5. Connect Raspberry PI to QFLEX 2 Embedded
6. Right click on Simulink model
7. Open "Model Configuration Parameters"
8. Select "Hardware Implementation"
9. Set "Hardware Board" to "Raspberry Pi"
10. Under "Board Parameters" under "Groups" type in the IP address of the Raspberry PI, the username, and password
11. Under "Build options", set build action to "Build and run" and type in the file path that you want the model to be stored on the Raspberry PI

12. Set simulation mode to external
13. Run MATLAB initialization code for motion controller
14. Run MATLAB Raspberry PI/SPI initialization code D.2
15. Click "Deploy to Hardware" button
16. To stop program you must "kill" it
17. Open Putty, connect to your Raspberry PI, and log in
18. Type "ps -A" to view running programs and find the process number of your Simulink model
19. Type "sudo kill -9 ####" where #### is your process number. This will kill the program
20. To run a file already on the Raspberry Pi use the command "sudo FILEPATH/FILENAME.elf", if file is in current directory use "sudo ./FILENAME.elf"
21. If the model runs to time infinity when started from the Raspberry Pi you can use Control-C to stop the model

Figure E.2: Quanser Aero with QFLEX 2 Embedded panel installed

E.3 Android Application

1. Make sure software add ons are installed on MATLAB, "Simulink Support Package for Android Devices"
2. Install Android Studio
3. Repeat steps from Appendix E.2 to set up Raspberry Pi and push model to Raspberry Pi

4. From MATLAB open add on manager and click on the "Setup" gear next to the Android package
5. Verify Android Studio installed, Next
6. Verify Android SDK Tools installed
7. NOTE: The recent version of Android Studio removed one of their files that MATLAB needs to build android applications. Fixed by downloading and installing an older verison of Android Studio NDK bundle (December 2017) and copying over the missing folder into the file path that MATLAB was calling. This issue might be fixed in a newer version of Android Studio.
8. Click next
9. Follow intrustions on prompts to turn android on in developer mode
10. Turn on USB debugging
11. Install driver
12. Connect android to computer with USB cable
13. Allow USB debugging
14. Make sure both computer and android are on same network with internet access
15. Select device
16. Build and run Test App, App should close and delete itself after test is complete
17. Open simulink model's configuration parameters
18. Under "Hardware Implementation" set "Hardware Board" to "Android Device"
19. Build and deploy simulink model to android
20. Once APP is running, swtich network to same network as the Raspberry Pi
21. Execute code on Raspberry Pi
22. Control helicopter

Bibliography

[1] Q. Ahmed, A. I. Bhatti, S. Iqbal, and I. H. Kazmi. 2-sliding mode based robust control for 2-dof helicopter. In *2010 11th International Workshop on Variable Structure Systems (VSS)*, pages 481–486, June 2010.

[2] W. Chang, J.H. Moon, and H.J. Lee. Fuzzy model-based output-tracking control for 2 degree-of-freedom helicopter. *Journal of Electrical Engineering Technology*, 12.00(1):1921–1928, 2017. Quanser product(s): 2 DOF Helicopter.

[3] Andrew Fandel, Anthony Birge, and Suruz Miah. Development of reinforcement learning algorithm for 2-dof helicopter model. In *IEEE International Symposium on Industrial Electronics*, Cairns, Australia, June 2018.

[4] W. Gao and Z. P. Jiang. Data-driven adaptive optimal output-feedback control of a 2-dof helicopter. In *2016 American Control Conference (ACC)*, pages 2512–2517, July 2016.

[5] M. Hernandez-Gonzalez, A.Y. Alanis, and E.A. Hernandez-Vargas. Decentralized discrete-time neural control for a quanser 2-dof helicopter. In *Applied Soft Computing*, pages 2462–2469, February 2012.

[6] E. Kayacan and M.A. Khanesar. Recurrent interval type-2 fuzzy control of 2-dof helicopter with finite time training algorithm. In *IFAC-PapersOnLine*, pages 293–299, July 2016.

[7] C. Prez-D'Arpino, W. Medina-Melndez, L. Fermn-Len, J. M. Bogado, R. R. Torrealba, and G. Fernndez-Lpez. Generalized bilateral mimo control by states convergence with time delay and application for the teleoperation of a 2-dof helicopter. In *2010 IEEE International Conference on Robotics and Automation*, pages 5328–5333, May 2010.

[8] R.G. Subramanian and V.K. Elumalai. Robust mrac augmented baseline lqr for tracking control of 2-dof helicopter. In *Robotics and Autonomous Systems*, pages 70–77, August 2016.