HW1

Fuzzy-PID Altitude Hold

Trenton Ruf

December 11, 2022

1. ROSFLIGHT INTRO

ROSflight is open source autopilot software that runs on a small embedded flight controller. A companion computer can be used to receive sensor data from the flight controller and send back commands. Since ROS integrates well with the Gazebo physics simulator I am testing plane controls in simulations only this term. This is mainly so I can safely "crash" the plane a lot while learning how to program flight controls.

2. SETUP

I. Operating System

I HIGHLY recommend using Ubuntu for setting up ROSflight. The latest supported version of ROSflight requires ROS Melodic on Ubuntu 18.04. Which is a bit unfortunate since 18.04 is already out of LTS. Though there is work on getting it ready for ROS2. Some libraries I used such as Scikit-Learn's fuzzy controller requires manual installation of a later Python version than what is supported by 18.04.

II. Roslaunch

I followed the guides on the official ROSflight website cover the installation of ROS, ROSflight, and the Gazebo simulator. I modified the default Roslaunch file for simulating a fixed-wing aircraft to include rosflight_joy, a companion node that binds keystrokes to a simulated Radio Controller to manually maneuver the aircraft. The RC controller is necessary to "arm" the plane before the throttle can be activated. Since ROSflight is designed to run on an actual aircraft, there is a bit of configuration required begin simulations. You must calibrate the IMU and set the appropriate fixed wing parameters (i.e. tell ROSflight it is a plane and not a multi-copter). More details can be found in the official documentation. The modified roslaunch file called myfixedwing.launch includes links to two separate world files I created. Both of these worlds are airfields but unfortunately the computer I am running the simulation on currently cannot render them in real-time, so they are currently disabled.

III. Gazebo

The Gazebo installation should happen along side ROSflight, but you may encounter the following error when first running a simulation:
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[`http://gazebosim.org/models/`]
[Err] [ModelDatabase.cc:294] Unable to download model manifests

The location of the model database has changed but has not been reflected in the gazebo package yet. The fix is to redirect to the correct URL before running the any launch files:
`export GAZEBO_MODEL_DATABASE_URI=http://models.gazebosim.org/`

The gazebo website is undergoing a major reconstruction at the moment. You should open up the URL above in a web browser too to check if the fix is still working.

3. ALTITUDE HOLD PID

The ROSflight node outputs a "Barometer" message that contains temperature, pressure, and altitude data. It also listens on the "Command" message for setting the plane's throttle, aileron, elevator, and rudder control surface positions. I created a new node called altitudePID that subscribes to the altitude reading and publishes an elevator position command. The mapping of altitude inputs to elevator outputs was done with a PID controller from the "simple_pid" python library. The tuning of the PID gains was conducted with a trial and error approach. I followed the Ziegler–Nichols method for the most part. The altitude hold setpoint was hard-coded to be 10 Meters above the ground. All control surfaces have a deflection range of (-1,1). That said during normal flight they should never get close to the maximum ranges. When testing full elevator deflection in flight the plane immediately turns 90 degrees and stalls.

4. FUZZY PID!

The altitudePID node works well for holding altitude while the wings are level, but an abrupt bank will begin large altitude oscillations. I believe this is due to the amount vertical lift that the elevator can influence being decreased with larger bank angles. The PID controller was tuned for maximum lift influence, so it makes sense that it will have problems when it falls out its ideal environment. To fix this issue the gains of the PID controller need to be altered during operation.

I found a white paper that covers an implementation of fuzzy-inference for airplane pitch control [1]. Since pitch is controlled with the elevator position I believed it would also work as an altitude controller. I created the "fuzzy-altitudePID" node that takes the fuzzy logic rules presented in the paper and integrates them with the altitudePID node from earlier. While the rules where kept the same I changed the ranges of all other parameters. All of these values where found from experimentation. For instance, the error delta range was determined by sending large manual inputs to the plane in flight to see what maximum and minimum error values could be. The tuning method from the original altitudePID node did not work for the fuzzy rule set.

I. code

All files related to this project can be found at:

https://github.com/Trenton-Ruf/Intelligent_Robotics

Listing 1: altitudePID

```
#!/usr/bin/env python

import rospy
from rosflight_msgs.msg import Command, Barometer
from simple_pid import PID

altitude = None
altitudeSetpoint = 10

# create PID controller
pid = PID(0.0015,0.0004,0.003, setpoint=altitudeSetpoint)
pid.output_limits = (-1, 1) # Aileron

# Create Message Structure
msg = Command()
# need to ignore Aeileron, Rudder, and Throttle
msg.ignore = Command.IGNORE_X | Command.IGNORE_Z | Command.IGNORE_F
msg.mode = Command.MODE_PASS_THROUGH

# Create publisher
publisher = rospy.Publisher("/command",Command,queue_size=1)

# Recieves the altitude reading message and outputs elevator deflection commands
# Mapping of reading to command is done with a PID controller
def altitudeCallback(baro):
    altitude = baro.altitude
    pid.setpoint = altitudeSetpoint

    msg.header.stamp = rospy.Time.now()
    msg.y = pid(altitude)
    publisher.publish(msg)
    # Log info to console for Debugging
    rospy.loginfo("Altitude:"+str(altitude) + " Elevator:"+str(msg.y) )

if __name__ == '__main__':
    try:
        # Init Node
        rospy.init_node('altitudePID')

        # Create listener
        rospy.Subscriber("/baro",Barometer, altitudeCallback)

        rospy.spin()

    except rospy.ROSInterruptException:
        pass

#msg.x = 0.0 #Aeileron deflection (-1,1)
#msg.z = 0.0 #Rudder deflection (-1,1)
```

Listing 2: fuzzy-altitudePID

```
#!/usr/bin/env python

import rospy
from rosflight_msgs.msg import Command, Barometer
from simple_pid import PID
import numpy as np
import skfuzzy.control as ctrl
```

```
import time

altitude = None
altitudeSetpoint = 10

# create PID controller
pid = PID(0.0015,0.0004,0.003, setpoint=altitudeSetpoint) # PID tunings will be
    ↪ overwritten with fuzzy logic
pid.output_limits =(-1,1) # Maximum Elevator Deflections

# Time variables for calculating Error Delta
startTime=time.time()
endTime=0
lastPidError = 0

# Create Message Structure
msg = Command()
# need to ignore Aeileron, Rudder, and Throttle command outputs.
msg.ignore = Command.IGNORE_X | Command.IGNORE_Z | Command.IGNORE_F
msg.mode = Command.MODE_PASS_THROUGH

# Create publisher
publisher = rospy.Publisher("/command",Command,queue_size=1)

##########################
# Fuzzy Setup
##########################

# Create five fuzzy variables - two inputs, three outputs
error = ctrl.Antecedent(np.linspace(-5,5,7), 'error')
delta = ctrl.Antecedent(np.linspace(-40,40,7), 'delta')

"""
# Funtional
kp = ctrl.Consequent(np.linspace(0 ,0.00075,7), 'kp')
kd = ctrl.Consequent(np.linspace(0 ,0.0055,7), 'kd')
ki = ctrl.Consequent(np.linspace(0 ,0.00015,7), 'ki')
"""

# Experimental
kp = ctrl.Consequent(np.linspace(-0.000000,0.01,7), 'kp')
kd = ctrl.Consequent(np.linspace(-0.000000,0.02,7), 'kd')
ki = ctrl.Consequent(np.linspace(-0.000000,0.01,7), 'ki')

# Fuzzy Terms
names = ['nb', 'nm', 'ns', 'zo', 'ps', 'pm', 'pb']
error.automf(names=names)
delta.automf(names=names)
kp.automf(names=names)
ki.automf(names=names)
kd.automf(names=names)

# So many rules... here we go

# kp rules ################################################
rule0 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
                              (error['nm'] & delta['nb']) |
                              (error['nb'] & delta['nm']) |
                              (error['nm'] & delta['nm'])),
                  consequent=kp['pb'], label='rule kp pb')

rule1 = ctrl.Rule(antecedent=((error['ns'] & delta['nb']) |
```

```
                                (error['zo'] & delta['nb']) |
                                (error['ns'] & delta['nm']) |
                                (error['zo'] & delta['nm']) |
                                (error['nb'] & delta['ns']) |
                                (error['nm'] & delta['ns']) |
                                (error['ns'] & delta['ns']) |
                                (error['nb'] & delta['zo'])),
                    consequent=kp['pm'], label='rule kp pm')

rule2 = ctrl.Rule(antecedent=((error['ps'] & delta['nb']) |
                                (error['ps'] & delta['nm']) |
                                (error['zo'] & delta['ns']) |
                                (error['nm'] & delta['zo']) |
                                (error['ns'] & delta['zo']) |
                                (error['nb'] & delta['ps']) |
                                (error['nm'] & delta['ps']) |
                                (error['nb'] & delta['pm'])),
                    consequent=kp['ps'], label='rule kp ps')

rule3 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
                                (error['pb'] & delta['nb']) |
                                (error['pm'] & delta['nm']) |
                                (error['ps'] & delta['ns']) |
                                (error['zo'] & delta['zo']) |
                                (error['ns'] & delta['ps']) |
                                (error['nm'] & delta['pm']) |
                                (error['nb'] & delta['pb'])),
                    consequent=kp['zo'], label='rule kp zo')

rule4 = ctrl.Rule(antecedent=((error['pb'] & delta['nm']) |
                                (error['pm'] & delta['ns']) |
                                (error['ps'] & delta['zo']) |
                                (error['zo'] & delta['ps']) |
                                (error['ps'] & delta['ps']) |
                                (error['ns'] & delta['pm']) |
                                (error['nm'] & delta['pb'])),
                    consequent=kp['ns'], label='rule kp ns')

rule5 = ctrl.Rule(antecedent=((error['pb'] & delta['ns']) |
                                (error['pm'] & delta['zo']) |
                                (error['pb'] & delta['zo']) |
                                (error['pm'] & delta['ps']) |
                                (error['pb'] & delta['ps']) |
                                (error['pm'] & delta['pm']) |
                                (error['ps'] & delta['pm']) |
                                (error['zo'] & delta['pm']) |
                                (error['ns'] & delta['pb']) |
                                (error['zo'] & delta['pb']) |
                                (error['ps'] & delta['pb'])),
                    consequent=kp['nm'], label='rule kp nm')

rule6 = ctrl.Rule(antecedent=((error['pb'] & delta['pm']) |
                                (error['pm'] & delta['pb']) |
                                (error['pb'] & delta['pb'])),
                    consequent=kp['nb'], label='rule kp nb')

# ki rules ################################################
rule7 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
                                (error['nm'] & delta['nb']) |
                                (error['ns'] & delta['nb']) |
                                (error['nb'] & delta['nm']) |
                                (error['nm'] & delta['nm']) |
```

```
                              (error['nb'] & delta['ns'])),
                  consequent=ki['nb'], label='rule ki nb')

rule8 = ctrl.Rule(antecedent=((error['zo'] & delta['nb']) |
                              (error['ns'] & delta['nm']) |
                              (error['nm'] & delta['ns']) |
                              (error['nb'] & delta['zo'])),
                  consequent=ki['nm'], label='rule ki nm')

rule9 = ctrl.Rule(antecedent=((error['ps'] & delta['nb']) |
                              (error['zo'] & delta['nm']) |
                              (error['ps'] & delta['nm']) |
                              (error['ns'] & delta['ns']) |
                              (error['zo'] & delta['ns']) |
                              (error['nm'] & delta['zo']) |
                              (error['ns'] & delta['zo']) |
                              (error['nm'] & delta['ps']) |
                              (error['nb'] & delta['ps'])),
                  consequent=ki['ns'], label='rule ki ns')

rule10 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
                              (error['pb'] & delta['nb']) |
                              (error['pm'] & delta['nm']) |
                              (error['pb'] & delta['nm']) |
                              (error['ps'] & delta['ns']) |
                              (error['zo'] & delta['zo']) |
                              (error['ns'] & delta['ps']) |
                              (error['nm'] & delta['pm']) |
                              (error['nb'] & delta['pm']) |
                              (error['nm'] & delta['pb']) |
                              (error['nb'] & delta['pb'])),
                  consequent=ki['zo'], label='rule ki zo')

rule11 = ctrl.Rule(antecedent=((error['pm'] & delta['ns']) |
                              (error['pb'] & delta['ns']) |
                              (error['ps'] & delta['zo']) |
                              (error['pm'] & delta['zo']) |
                              (error['zo'] & delta['ps']) |
                              (error['ps'] & delta['ps']) |
                              (error['ns'] & delta['pm']) |
                              (error['ns'] & delta['pb'])),
                  consequent=ki['ps'], label='rule ki ps')

rule12 = ctrl.Rule(antecedent=((error['pb'] & delta['zo']) |
                              (error['pm'] & delta['ps']) |
                              (error['ps'] & delta['pm']) |
                              (error['zo'] & delta['pm']) |
                              (error['zo'] & delta['pb'])),
                  consequent=ki['pm'], label='rule ki pm')

rule13 = ctrl.Rule(antecedent=((error['pb'] & delta['ps']) |
                              (error['pm'] & delta['pm']) |
                              (error['pb'] & delta['pm']) |
                              (error['pb'] & delta['pb']) |
                              (error['pm'] & delta['pb']) |
                              (error['ps'] & delta['pb'])),
                  consequent=ki['pb'], label='rule ki pb')

# kd rules ################################################
rule14 = ctrl.Rule(antecedent=((error['nb'] & delta['ns']) |
                              (error['nm'] & delta['ns']) |
                              (error['nb'] & delta['zo']) |
```

```
                                (error['nb'] & delta['ps'])),
                    consequent=kd['nb'], label='rule kd nb')

rule15 = ctrl.Rule(antecedent=((error['nb'] & delta['nm']) |
                                (error['ns'] & delta['ns']) |
                                (error['nm'] & delta['zo']) |
                                (error['ns'] & delta['zo']) |
                                (error['nm'] & delta['ps']) |
                                (error['nb'] & delta['pm'])),
                    consequent=kd['nm'], label='rule kd nm')

rule16 = ctrl.Rule(antecedent=((error['nm'] & delta['nm']) |
                                (error['ns'] & delta['nm']) |
                                (error['zo'] & delta['nm']) |
                                (error['ps'] & delta['nm']) |
                                (error['pm'] & delta['nm']) |
                                (error['zo'] & delta['ns']) |
                                (error['zo'] & delta['zo']) |
                                (error['zo'] & delta['ps']) |
                                (error['ns'] & delta['ps']) |
                                (error['zo'] & delta['pm']) |
                                (error['ns'] & delta['pm']) |
                                (error['nm'] & delta['pm'])),
                                #(error['nb'] & delta['pb']) | Test this change , and
                                    ↪ remove from 'rule kd ps'
                                #(error['nm'] & delta['pb']) |
                    consequent=kd['ns'], label='rule kd ns')

rule17 = ctrl.Rule(antecedent=((error['ns'] & delta['nb']) |
                                (error['zo'] & delta['nb']) |
                                (error['ps'] & delta['nb']) |
                                (error['ps'] & delta['ns']) |
                                (error['ps'] & delta['zo']) |
                                (error['ps'] & delta['ps']) |
                                (error['ps'] & delta['pm']) |
                                (error['zo'] & delta['pb']) |
                                (error['ns'] & delta['pb'])),
                    consequent=kd['zo'], label='rule kd zo')

rule18 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
                                (error['nm'] & delta['nb']) |
                                (error['pm'] & delta['ns']) |
                                (error['pm'] & delta['zo']) |
                                (error['pm'] & delta['ps']) |
                                (error['pb'] & delta['ps']) |
                                (error['pm'] & delta['pm']) |
                                (error['pb'] & delta['pm']) |
                                (error['nb'] & delta['pb']) | # Really? I'm thinkin delta
                                    ↪ is 'ns' instead
                                (error['nm'] & delta['pb']) | # Really? I'm thinkin delta
                                    ↪ is 'ns' instead
                                (error['ps'] & delta['pb'])),
                    consequent=kd['ps'], label='rule kd ps')

rule19 = ctrl.Rule(antecedent=((error['pb'] & delta['nm']) |
                                (error['pb'] & delta['ns']) |
                                (error['pb'] & delta['zo'])),
                    consequent=kd['pm'], label='rule kd pm')

rule20 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
                                (error['pb'] & delta['nb']) |
                                (error['pm'] & delta['pb']) |
```

```
                                (error['pb'] & delta['pb'])),
                    consequent=kd['pb'], label='rule kd pb')

system = ctrl.ControlSystem(rules=[
                                    rule0, rule1, rule2, rule3, rule4, rule5, rule6
                                        ↪ ,
                                    rule7, rule8, rule9, rule10, rule11, rule12,
                                        ↪ rule13,
                                    rule14, rule15, rule16, rule17, rule18, rule19,
                                        ↪ rule20
                                    ])
sim = ctrl.ControlSystemSimulation(system, flush_after_run=1000) # lower flush if memory
    ↪ is scarce

altList=[]
# Averages the last three altimeter readings before sending the reading to altitudePID
# Currently not used!
def altimeterFilter(baro):
    altList.insert(0,baro.altitude)
    if len(altList) > 3:
        altList.pop()
        avgAltitude = sum(altList) / float(len(altList))
        altitudePID(avgAltitude)

# Recieves altitude readings and outputs Elevator deflection commands
def altitudePID(baro):
    altitude = baro.altitude

    global startTime
    global endTime
    global lastPidError
    pid.setpoint = altitudeSetpoint

    # Get error and delta
    pidError= altitudeSetpoint - altitude
    pidDelta = float(pidError - lastPidError)/float(endTime - startTime)
    lastPidError = pidError

    # Reset timer for calculating error delta
    endTime = startTime
    startTime = time.time()

    # Compute Fuzzy Inference
    sim.input['error']= pidError
    sim.input['delta']= pidDelta
    sim.compute()

    #Set pid tunings from fuzzy logic
    pid.tunings = (sim.output['kp'],
                   sim.output['ki'],
                   sim.output['kd']
                   )

    msg.header.stamp = rospy.Time.now()
    msg.y = pid(altitude)
    publisher.publish(msg)
    # Send info to the console for debugging
    rospy.loginfo("Altitude:"+str(round(altitude, 4)) +
                  " Elevator:"+str(round(msg.y, 4)) +
                  " Kp:"+str(round(sim.output['kp'], 4)) +
                  " Ki:"+str(round(sim.output['ki'], 4)) +
                  " Kd:"+str(round(sim.output['kd'], 4)) +
```

```
                    " Error:"+str(round(pidError)) +
                    " Delta:"+str(round(pidDelta))
                    )

if __name__ == '__main__':
    try:
        # Init Node
        rospy.init_node('fuzzy-altitudePID')

        # Create listener
        #rospy.Subscriber("/baro",Barometer, altimeterFilter) # with filter
        rospy.Subscriber("/baro",Barometer, altitudePID ) # bypass filter

        rospy.spin()

    except rospy.ROSInterruptException:
        pass

#msg.x = 0.0 #Aeileron deflection (-1,1)
#msg.z = 0.0 #Rudder deflection (-1,1)
```

REFERENCES

[1] N. Beygi, M. Beigy, and M. Siahi, "Design of fuzzy self-tuning pid controller for pitch control system of aircraft autopilot," 2015. [Online]. Available: https://arxiv.org/abs/1510.02588