

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://gazebo-sim.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.gazebo-sim.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 were kept the same I changed the ranges of all other parameters. All of these values were 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:

Listing 1: altitudePID

```
1  #!/usr/bin/env python
2
3  import rospy
4  from rosflight_msgs.msg import Command, Barometer
5  from simple_pid import PID
6
7  altitude = None
8  altitudeSetpoint = 10
9
10 # create PID controller
11 pid = PID(0.0015,0.0004,0.003, setpoint=altitudeSetpoint)
12 pid.output_limits = (-1, 1) # Aileron
13
14 # Create Message Structure
15 msg = Command()
16 # need to ignore Aileron, Rudder, and Throttle
17 msg.ignore = Command.IGNORE_X | Command.IGNORE_Z | Command.IGNORE_F
18 msg.mode = Command.MODE_PASS_THROUGH
19
20 # Create publisher
21 publisher = rospy.Publisher("/command",Command,queue_size=1)
22
23 # Recieves the altitude reading message and outputs elevator deflection commands
24 # Mapping of reading to command is done with a PID controller
25 def altitudeCallback(baro):
26     altitude = baro.altitude
27     pid.setpoint = altitudeSetpoint
28
29     msg.header.stamp = rospy.Time.now()
30     msg.y = pid(altitude)
31     publisher.publish(msg)
32     # Log info to console for Debugging
33     rospy.loginfo("Altitude:"+str(altitude) + " Elevator:"+str(msg.y) )
34
35 if __name__ == '__main__':
36     try:
37         # Init Node
38         rospy.init_node('altitudePID')
39
40         # Create listener
41         rospy.Subscriber("/baro",Barometer, altitudeCallback)
42
43         rospy.spin()
44
45     except rospy.ROSInterruptException:
46         pass
47
48 #msg.x = 0.0 #Aileron deflection (-1,1)
49 #msg.z = 0.0 #Rudder deflection (-1,1)
```

Listing 2: fuzzy-altitudePID

```
1  #!/usr/bin/env python
2
3  import rospy
4  from rosflight_msgs.msg import Command, Barometer
5  from simple_pid import PID
6  import numpy as np
7  import skfuzzy.control as ctrl
```

```

8 import time
9
10 altitude = None
11 altitudeSetpoint = 10
12
13 # create PID controller
14 pid = PID(0.0015,0.0004,0.003, setpoint=altitudeSetpoint) # PID tunings will be
    ↳ overwritten with fuzzy logic
15 pid.output_limits = (-1,1) # Maximum Elevator Deflections
16
17 # Time variables for calculating Error Delta
18 startTime=time.time()
19 endTime=0
20 lastPidError = 0
21
22 # Create Message Structure
23 msg = Command()
24 # need to ignore Aileron, Rudder, and Throttle command outputs.
25 msg.ignore = Command.IGNORE_X | Command.IGNORE_Z | Command.IGNORE_F
26 msg.mode = Command.MODE_PASS_THROUGH
27
28 # Create publisher
29 publisher = rospy.Publisher("/command",Command,queue_size=1)
30
31 #####
32 # Fuzzy Setup
33 #####
34
35 # Create five fuzzy variables – two inputs, three outputs
36 error = ctrl.Antecedent(np.linspace(-5,5,7), 'error')
37 delta = ctrl.Antecedent(np.linspace(-40,40,7), 'delta')
38
39 """
40 # Funtional
41 kp = ctrl.Consequent(np.linspace(0 ,0.00075,7), 'kp')
42 kd = ctrl.Consequent(np.linspace(0 ,0.0055,7), 'kd')
43 ki = ctrl.Consequent(np.linspace(0 ,0.00015,7), 'ki')
44 """
45
46 # Experimental
47 kp = ctrl.Consequent(np.linspace(-0.000000,0.01,7), 'kp')
48 kd = ctrl.Consequent(np.linspace(-0.000000,0.02,7), 'kd')
49 ki = ctrl.Consequent(np.linspace(-0.000000,0.01,7), 'ki')
50
51 # Fuzzy Terms
52 names = ['nb', 'nm', 'ns', 'zo', 'ps', 'pm', 'pb']
53 error.automf(names=names)
54 delta.automf(names=names)
55 kp.automf(names=names)
56 ki.automf(names=names)
57 kd.automf(names=names)
58
59 # So many rules... here we go
60
61 # kp rules #####
62 rule0 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
63                               (error['nm'] & delta['nb']) |
64                               (error['nb'] & delta['nm']) |
65                               (error['nm'] & delta['nm'])),
66                  consequent=kp['pb'], label='rule kp pb')
67
68 rule1 = ctrl.Rule(antecedent=((error['ns'] & delta['nb']) |

```

```

69         (error['zo'] & delta['nb']) |
70         (error['ns'] & delta['nm']) |
71         (error['zo'] & delta['nm']) |
72         (error['nb'] & delta['ns']) |
73         (error['nm'] & delta['ns']) |
74         (error['ns'] & delta['ns']) |
75         (error['nb'] & delta['zo'])),
76     consequent=kp['pm'], label='rule kp pm')
77
78 rule2 = ctrl.Rule(antecedent=((error['ps'] & delta['nb']) |
79     (error['ps'] & delta['nm']) |
80     (error['zo'] & delta['ns']) |
81     (error['nm'] & delta['zo']) |
82     (error['ns'] & delta['zo']) |
83     (error['nb'] & delta['ps']) |
84     (error['nm'] & delta['ps']) |
85     (error['nb'] & delta['pm'])),
86     consequent=kp['ps'], label='rule kp ps')
87
88 rule3 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
89     (error['pb'] & delta['nb']) |
90     (error['pm'] & delta['nm']) |
91     (error['ps'] & delta['ns']) |
92     (error['zo'] & delta['zo']) |
93     (error['ns'] & delta['ps']) |
94     (error['nm'] & delta['pm']) |
95     (error['nb'] & delta['pb'])),
96     consequent=kp['zo'], label='rule kp zo')
97
98 rule4 = ctrl.Rule(antecedent=((error['pb'] & delta['nm']) |
99     (error['pm'] & delta['ns']) |
100     (error['ps'] & delta['zo']) |
101     (error['zo'] & delta['ps']) |
102     (error['ps'] & delta['ps']) |
103     (error['ns'] & delta['pm']) |
104     (error['nm'] & delta['pb'])),
105     consequent=kp['ns'], label='rule kp ns')
106
107 rule5 = ctrl.Rule(antecedent=((error['pb'] & delta['ns']) |
108     (error['pm'] & delta['zo']) |
109     (error['pb'] & delta['zo']) |
110     (error['pm'] & delta['ps']) |
111     (error['pb'] & delta['ps']) |
112     (error['pm'] & delta['pm']) |
113     (error['ps'] & delta['pm']) |
114     (error['zo'] & delta['pm']) |
115     (error['ns'] & delta['pb']) |
116     (error['zo'] & delta['pb']) |
117     (error['ps'] & delta['pb'])),
118     consequent=kp['nm'], label='rule kp nm')
119
120 rule6 = ctrl.Rule(antecedent=((error['pb'] & delta['pm']) |
121     (error['pm'] & delta['pb']) |
122     (error['pb'] & delta['pb'])),
123     consequent=kp['nb'], label='rule kp nb')
124
125 # ki rules #####
126 rule7 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
127     (error['nm'] & delta['nb']) |
128     (error['ns'] & delta['nb']) |
129     (error['nb'] & delta['nm']) |
130     (error['nm'] & delta['nm']) |

```

```

131         (error['nb'] & delta['ns'])),
132     consequent=ki['nb'], label='rule ki nb')
133
134 rule8 = ctrl.Rule(antecedent=((error['zo'] & delta['nb']) |
135     (error['ns'] & delta['nm']) |
136     (error['nm'] & delta['ns']) |
137     (error['nb'] & delta['zo'])),
138     consequent=ki['nm'], label='rule ki nm')
139
140 rule9 = ctrl.Rule(antecedent=((error['ps'] & delta['nb']) |
141     (error['zo'] & delta['nm']) |
142     (error['ps'] & delta['nm']) |
143     (error['ns'] & delta['ns']) |
144     (error['zo'] & delta['ns']) |
145     (error['nm'] & delta['zo']) |
146     (error['ns'] & delta['zo']) |
147     (error['nm'] & delta['ps']) |
148     (error['nb'] & delta['ps'])),
149     consequent=ki['ns'], label='rule ki ns')
150
151 rule10 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
152     (error['pb'] & delta['nb']) |
153     (error['pm'] & delta['nm']) |
154     (error['pb'] & delta['nm']) |
155     (error['ps'] & delta['ns']) |
156     (error['zo'] & delta['zo']) |
157     (error['ns'] & delta['ps']) |
158     (error['nm'] & delta['pm']) |
159     (error['nb'] & delta['pm']) |
160     (error['nm'] & delta['pb']) |
161     (error['nb'] & delta['pb'])),
162     consequent=ki['zo'], label='rule ki zo')
163
164 rule11 = ctrl.Rule(antecedent=((error['pm'] & delta['ns']) |
165     (error['pb'] & delta['ns']) |
166     (error['ps'] & delta['zo']) |
167     (error['pm'] & delta['zo']) |
168     (error['zo'] & delta['ps']) |
169     (error['ps'] & delta['ps']) |
170     (error['ns'] & delta['pm']) |
171     (error['ns'] & delta['pb'])),
172     consequent=ki['ps'], label='rule ki ps')
173
174 rule12 = ctrl.Rule(antecedent=((error['pb'] & delta['zo']) |
175     (error['pm'] & delta['ps']) |
176     (error['ps'] & delta['pm']) |
177     (error['zo'] & delta['pm']) |
178     (error['zo'] & delta['pb'])),
179     consequent=ki['pm'], label='rule ki pm')
180
181 rule13 = ctrl.Rule(antecedent=((error['pb'] & delta['ps']) |
182     (error['pm'] & delta['pm']) |
183     (error['pb'] & delta['pm']) |
184     (error['pb'] & delta['pb']) |
185     (error['pm'] & delta['pb']) |
186     (error['ps'] & delta['pb'])),
187     consequent=ki['pb'], label='rule ki pb')
188
189 # kd rules #####
190 rule14 = ctrl.Rule(antecedent=((error['nb'] & delta['ns']) |
191     (error['nm'] & delta['ns']) |
192     (error['nb'] & delta['zo']) |

```

```

193         (error['nb'] & delta['ps'])),
194     consequent=kd['nb'], label='rule kd nb')
195
196 rule15 = ctrl.Rule(antecedent=((error['nb'] & delta['nm']) |
197     (error['ns'] & delta['ns']) |
198     (error['nm'] & delta['zo']) |
199     (error['ns'] & delta['zo']) |
200     (error['nm'] & delta['ps']) |
201     (error['nb'] & delta['pm'])),
202     consequent=kd['nm'], label='rule kd nm')
203
204 rule16 = ctrl.Rule(antecedent=((error['nm'] & delta['nm']) |
205     (error['ns'] & delta['nm']) |
206     (error['zo'] & delta['nm']) |
207     (error['ps'] & delta['nm']) |
208     (error['pm'] & delta['nm']) |
209     (error['zo'] & delta['ns']) |
210     (error['zo'] & delta['zo']) |
211     (error['zo'] & delta['ps']) |
212     (error['ns'] & delta['ps']) |
213     (error['zo'] & delta['pm']) |
214     (error['ns'] & delta['pm']) |
215     (error['nm'] & delta['pm'])),
216     # (error['nb'] & delta['pb']) | Test this change , and
217     # ↪ remove from 'rule kd ps'
218     # (error['nm'] & delta['pb']) |
219     consequent=kd['ns'], label='rule kd ns')
220
221 rule17 = ctrl.Rule(antecedent=((error['ns'] & delta['nb']) |
222     (error['zo'] & delta['nb']) |
223     (error['ps'] & delta['nb']) |
224     (error['ps'] & delta['ns']) |
225     (error['ps'] & delta['zo']) |
226     (error['ps'] & delta['ps']) |
227     (error['ps'] & delta['pm']) |
228     (error['zo'] & delta['pb']) |
229     (error['ns'] & delta['pb'])),
230     consequent=kd['zo'], label='rule kd zo')
231
232 rule18 = ctrl.Rule(antecedent=((error['nb'] & delta['nb']) |
233     (error['nm'] & delta['nb']) |
234     (error['pm'] & delta['ns']) |
235     (error['pm'] & delta['zo']) |
236     (error['pm'] & delta['ps']) |
237     (error['pb'] & delta['ps']) |
238     (error['pm'] & delta['pm']) |
239     (error['pb'] & delta['pm']) |
240     (error['nb'] & delta['pb']) | # Really? I'm thinkin delta
241     # ↪ is 'ns' instead
242     (error['nm'] & delta['pb']) | # Really? I'm thinkin delta
243     # ↪ is 'ns' instead
244     (error['ps'] & delta['pb'])),
245     consequent=kd['ps'], label='rule kd ps')
246
247 rule19 = ctrl.Rule(antecedent=((error['pb'] & delta['nm']) |
248     (error['pb'] & delta['ns']) |
249     (error['pb'] & delta['zo'])),
250     consequent=kd['pm'], label='rule kd pm')
251
252 rule20 = ctrl.Rule(antecedent=((error['pm'] & delta['nb']) |
253     (error['pb'] & delta['nb']) |
254     (error['pm'] & delta['pb']) |

```

```

252         (error['pb'] & delta['pb'])),
253         consequent=kd['pb'], label='rule kd pb')
254
255 system = ctrl.ControlSystem(rules=[
256     rule0, rule1, rule2, rule3, rule4, rule5, rule6
257     ↪ ,
258     rule7, rule8, rule9, rule10, rule11, rule12,
259     ↪ rule13,
260     rule14, rule15, rule16, rule17, rule18, rule19,
261     ↪ rule20
262 ])
263 sim = ctrl.ControlSystemSimulation(system, flush_after_run=1000) # lower flush if memory
264 ↪ is scarce
265
266 altList=[]
267 # Averages the last three altimeter readings before sending the reading to altitudePID
268 # Currently not used!
269 def altimeterFilter(barometer):
270     altList.insert(0,barometer.altitude)
271     if len(altList) > 3:
272         altList.pop()
273         avgAltitude = sum(altList) / float(len(altList))
274         altitudePID(avgAltitude)
275
276 # Recieves altitude readings and outputs Elevator deflection commands
277 def altitudePID(barometer):
278     altitude = barometer.altitude
279
280     global startTime
281     global endTime
282     global lastPidError
283     pid.setpoint = altitudeSetpoint
284
285     # Get error and delta
286     pidError= altitudeSetpoint - altitude
287     pidDelta = float(pidError - lastPidError)/float(endTime - startTime)
288     lastPidError = pidError
289
290     # Reset timer for calculating error delta
291     endTime = startTime
292     startTime = time.time()
293
294     # Compute Fuzzy Inference
295     sim.input['error']= pidError
296     sim.input['delta']= pidDelta
297     sim.compute()
298
299     #Set pid tunings from fuzzy logic
300     pid.tunings = (sim.output['kp'],
301                   sim.output['ki'],
302                   sim.output['kd']
303                   )
304
305     msg.header.stamp = rospy.Time.now()
306     msg.y = pid(altitude)
307     publisher.publish(msg)
308     # Send info to the console for debugging
309     rospy.loginfo("Altitude:"+str(round(altitude, 4)) +
310                  " Elevator:"+str(round(msg.y, 4)) +
311                  " Kp:"+str(round(sim.output['kp'], 4)) +
312                  " Ki:"+str(round(sim.output['ki'], 4)) +
313                  " Kd:"+str(round(sim.output['kd'], 4)) +

```



```

310         " Error:" +str(round(pidError)) +
311         " Delta:" +str(round(pidDelta))
312     )
313
314 if __name__ == '__main__':
315     try:
316         # Init Node
317         rospy.init_node('fuzzy-altitudePID')
318
319         # Create listener
320         #rospy.Subscriber("/baro",Barometer, altimeterFilter) # with filter
321         rospy.Subscriber("/baro",Barometer, altitudePID ) # bypass filter
322
323         rospy.spin()
324
325     except rospy.ROSInterruptException:
326         pass
327
328 #msg.x = 0.0 #Aileron deflection (-1,1)
329 #msg.z = 0.0 #Rudder deflection (-1,1)

```

REFERENCES

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