ROB501 Assignment 4
Image Based Visual Servoing
Ethan Rajah | 1006722370

With my IBVS algorithm complete, it was necessary to tune the controller to optimize for convergence speed. To do so, a tuning code block was created to facilitate testing and plotting simulation times and results over various gains. I limited the set of tested gains to a range where the IBVS algorithm converges and was done for both the known depth simulation setting, as well as the unknown depth setting. The estimated depth case used a gain range from 0.001 to 1, and the known depth case used a range of 0.001 to 1.4. Since this code was not required for submission, it is presented in **Figure 1** below for the estimated depth case (in part_03_learner_example.py):

```
# Run simulation - estimate depths.
gain_tuning = [0.001, 0.01, 0.05, 0.1, 0.5, 0.7, 0.9, 1]
delta_t = np.zeros(len(gain_tuning))
for i, gain in enumerate(gain_tuning):
   sim_start = time.time()
   ibvs_simulation(Twc_init, Twc_last, pts, K, gain, True)
   sim_end = time.time()
   delta = sim_end - sim_start
   delta_t[i] = delta
   print("Simulation time for gain = ", gain, " is: ", delta, " seconds.")
   plt.savefig("A4/results/final_ibvs_gain_{}_estimate.png".format(gain))
plt.clf()
plt.plot(gain_tuning, delta_t)
plt.xlabel("Gain")
plt.ylabel("Simulation time (s)")
plt.title("Simulation Convergence Time vs Gain - Estimated Depths")
plt.savefig("A4/results/final_ibvs_time_estimate.png")
plt.show()
```

Figure 1: gain tuning code block used for visualizing the convergence times of various controller gains

To optimize the gain for fastest convergence while also maintaining algorithm robustness, a challenging initial camera pose was chosen so that convergence could be guaranteed for simpler cases:

$$x_{init} = \begin{bmatrix} -0.01 \\ 10 \\ -2 \\ \pi/3 \\ -\pi/16 \\ -\pi/6 \end{bmatrix}$$

Using this setup, the gains were simulated and timed for both the known depth case and the estimated depth case. An M3 Macbook Pro was used for running the simulations, so the times are relative to this hardware. The simulation time results are presented in **Table 1**, as well as visually in **Figure 2** and **Figure 3**.

Gain	Known Depth Time (s)	Estimated Depth Time (s)
0.001	324.95	208.87
0.01	267.87	148.75
0.05	413.82	36.00
0.1	120.59	20.69
0.5	3.14	5.26
0.7	N/A	4.42
0.9	N/A	3.79
1	1.47	209.02
1.1	1.89	N/A
1.2	2.33	N/A
1.3	3.14	N/A
1.4	324.67	N/A

Table 1: Simulation time results for various gains tested for the known depth and estimated depth cases. N/A values indicate that the specific gain was not tested for the case.

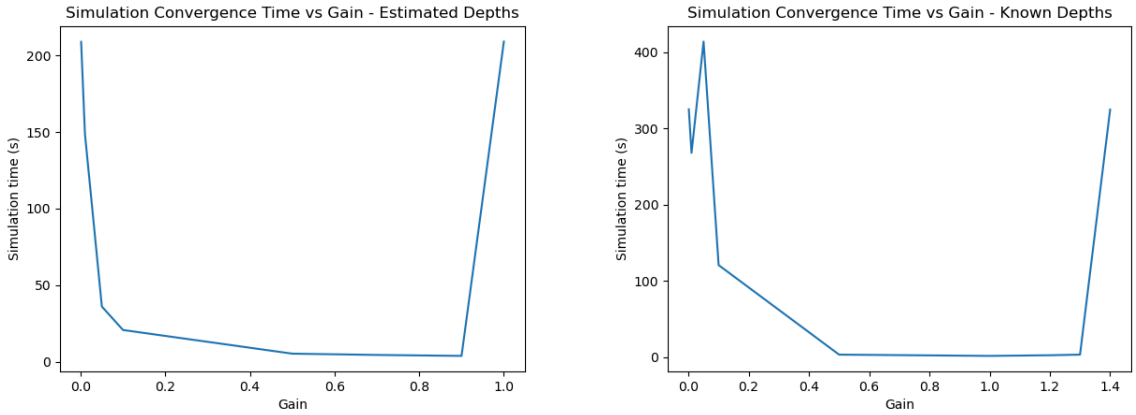

Figure 2 & 3: Visual depiction of simulation time vs gain results presented in Table 1 for both the known depth case and the estimated depth.

These results show that for both the estimated and known depth cases, there is a range of gains which result in fast convergence times, however, outside of this range causes instability and therefore no convergence. For the case where feature depths are known exactly, the optimal gain value was found to be a value of 1. As shown in **Table 1**, gains over 1 resulted in a small incremental increase in simulation time, until reaching instability at gain 1.4 and therefore not converging. Similarly, gains that were less than 1 appear to result in larger simulation times, growing exponentially for gains less than 0.5. The known depth case had an interesting result for gain 0.05 as the simulation does not converge and its

recorded duration is greater than that of gains 0.01 and 0.001. This does not follow the quadratic convergence trend observed during the gain tuning, so my hypothesis is that it is a random outlier resulting from hardware loading during that specific gain simulation (further testing would need to be conducted using gain 0.05 to conclude on this).

As shown in **Figure 2** and **Figure 3**, both types of simulations were observed to have the same quadratic convergence shape. The optimal gain value for the case where feature depths are estimated was found to be 0.9. This gain is less than the optimal known depth gain of 1 and **Table 1** shows that a gain of 1 for the estimated depth case gives instability and therefore no convergence. Furthermore, the gain tuning results between the two cases demonstrates that the known depth case is able to achieve lower simulation convergence times than the estimated depth case. For instance, the optimal gains for each case gives a convergence time difference of about 2.3 seconds. Since the convergence trials were not done repeatedly over the exact same ranges of numbers and the time difference is small, the variation could be a result of hardware compute loading. For this reason, the difference between the two cases is not significant in this situation.

Overall, this analysis showed that while it is often desired to increase the proportional gain of a controller to speed up the convergence tracking rate of the system to the desired points, too large of an increase will result in instability of the servoing algorithm. The simulation runs would clearly demonstrate this, as gains larger than the optimal value would cause the four points to converge into one point and translate randomly throughout the image plane until the loop maximum number of iterations is reached. Contrasting this to very small gains, the four points were observed to rotate and translate extremely slowly towards the image plane desired point, resulting in the simulation reaching the maximum number of iterations before the algorithm could converge to the desired pose. An illustration of this behavior is shown in **Figure 4**, where one can observe that the final pose of the four points at the end of the simulation has a large error from the desired pose when using a gain of 0.001. In general, a proportional controller is adequate for this particular IBVS problem, but more sophisticated controllers may be required for more complex problems, where computation speed needs to be sacrificed for better control over response transients.

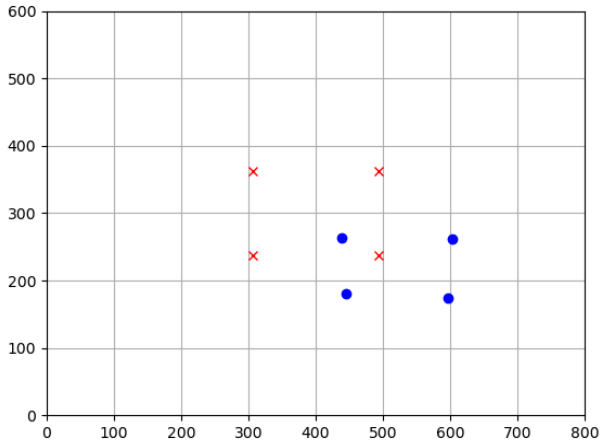

Figure 4: Estimated depth end-of-simulation image, for gain of 0.001. The red points indicate the desired pose, whereas the blue points indicate the current pose.