AERO60492 – Autonomous Mobile Robots
Coursework 3 – Feedback Control

Aims

The aim of this task is to create a simple feedback control algorithm for position stabilisation of an Unmanned Aerial System (UAS). Processing measurements from sensors, such as GPS location, to generate actuation commands that achieve a specific vehicle state (position), is a fundamental building block of robotics systems. Many advanced guidance and navigation methods depend on the fundamental ability of a robotic system to move to a designated 'point'.

The task will involve creating an algorithm that can read data from a position/attitude/velocity feedback sensor, perform the necessary coordinate transformations, calculate an actuation command, and send it to a UAS platform to enable the achievement of a specified position. The exercise will be completed in teams of four. It is expected that you already have a good understanding of the Python programming language, feedback controllers and video making. For a refresher on these topics please visit:

- Screen recording in Windows: https://www.microsoft.com/en-us/windows/learning-center/how-to-record-screen-windows-11
- PowerPoint screen recording: https://support.microsoft.com/en-gb/office/record-a-presentation-2570dff5-f81c-40bc-b404-e04e95ffab33
- Free video editing: https://www.blackmagicdesign.com/uk/products/davinciresolve
- Understanding Python: https://wiki.python.org/moin/BeginnersGuide/Programmers
- Understanding feedback controllers: https://www.cim.mcgill.ca/~ialab/ev/Intro_control1.pdf

By the time you complete this coursework you should:

- Understand how to design and tune controllers in practical setting.
- Appreciate the dynamics and kinematics of the unmanned aerial vehicle platform.
- Understand limitation of the simple controller and the need for more advanced methods.
- Apply axis transformations in a real-world setting.

Timeline

Week 8 – coursework is set. Familiarisation and installation with the simulator, sending basic control commands.
Week 9 – Controller implementation, in simulator
Week 10 – Familiarization with experimental setup
(easter break is here)
Week 11 – Flight test of the controller, submission portal opens
Week 12 – Final chance to collect necessary data
It is expected that this task should take no more than 25 hours to complete over the 4-week period.

Deadline: 3 pm Friday 9th of May 2025 (Last day of the semester)

Submission

The submission consists of two elements:

- Python3 script (controller.py) which runs a feedback control algorithm to stabilise the position of a UAV with external reference position and yaw inputs provided by a user. Commenting should be used to explain the steps of your method and good coding standards should be followed. This is submitted as one for the group of four and the mark is shared between two members.
- Video which shows your understanding of the controller and the tuning methodology. This is an individual video, while material sharing (such as data, experimental videos) is allowed within individual teams, video structure, method and explanation must be done individually. Video should be no longer than 3 minutes. Any content after 3-minute mark will not be marked. You do not need to provide any background information beyond what is required to explain workings on your controller and/or tuning.

Marking scheme

This coursework has a 25% contribution to the unit overall mark. Marks out of 25 will be awarded according to:

Python code (8 marks):

- Successful 3D stabilisation and tracking performance [4 marks]
- Good coding practices [2 marks]
- Advanced methods explained in code using comments (i.e. state estimators, DOBC, cascade controllers, LQRs, reinforcement learning etc.) [2 marks]

Video (17 marks):

- Explanation of selected method [4 marks]
- Explanation of tuning method [4 marks]
- Overall video quality [2 marks]
- Explanation of advanced methods used (i.e. state estimators, DOBC, cascade controllers, LQRs, reinforcement learning etc.)) [2 marks]
- Explanation of experiment [5 marks]

Background

Feedback control can trace its history back to the Water Clocks of the Greeks and Arabs. The primary motivation for feedback control in the times of antiquity was the need for accurate time determination. Thus, around 270 BCE, the Greek Ctesibius invented a float regulator for a water clock. This regulator's function was to maintain a constant water level in a tank. Ctesibius's regulator used a float to control the inflow of water through a valve; as the water level fell, the valve opened to replenish the reservoir. This float regulator performed a similar function to the modern flush toilet's ballcock. In this system, a sensor measurement (the height of water) informed an actuator (the inlet valve) what to do to maintain a setpoint (the required depth). This loop was an early instance of a feedback control loop. The actuation was very basic, offering a somewhat 'on/off' option regarding whether the valve was open or closed.

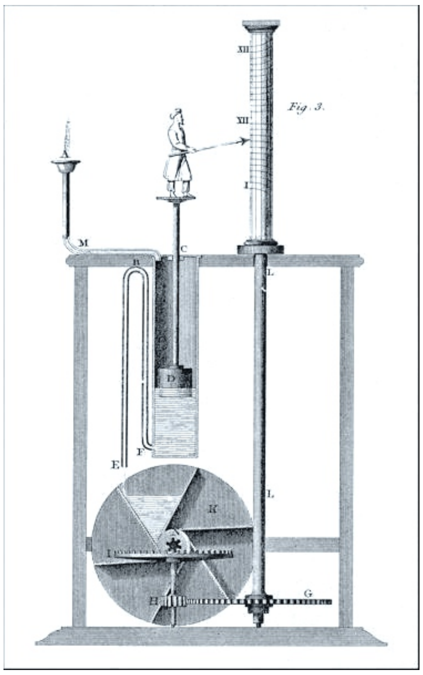
Figure 1. Ctesibius's water level regulator

The field of control remained rudimentary until machinery began to gain prominence in the 16th and 17th centuries. Some of the earliest examples from this period include temperature control for furnaces as well as the famous flyball governor by Christiaan Huygens, which was adapted with great success to steam engines by James Watt. In the flyball governor, a number of masses (usually two) are suspended from a rotating shaft that is connected to the machinery. At a certain speed of the shaft, the centrifugal force exerted by the masses causes the throttle valve to close, preventing the shaft's speed from exceeding a certain threshold. This threshold could represent a desired speed or a safety limit.

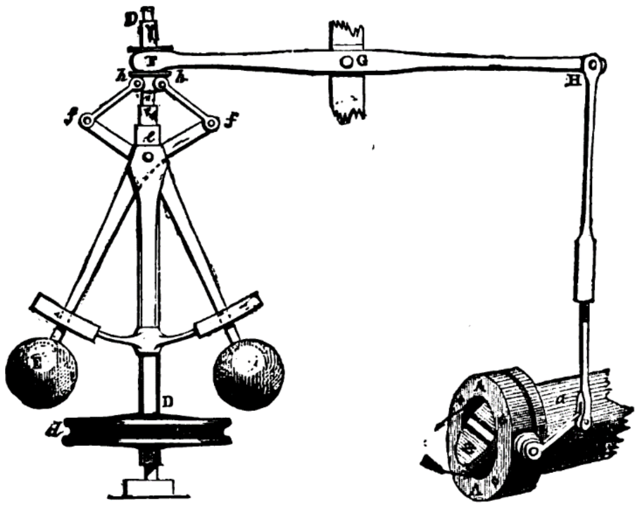
FIG. 4.—*Governor and Throttle-Valve.*

Figure 2. A flyball regulator to smoothly control rates and speeds.

The origins of the formal field of control are often attributed to the paper titled 'On Governors' by J.C. Maxwell (the same Maxwell known for his electromagnetic field equations). Since then, significant progress has been made in the field, in terms of understanding, modelling, and controlling systems. The most commonly used controller, known as PID (Proportional, Integral, and Derivative), was invented in 1911 by Elmer Sperry (the same Sperry's who also invented the first aircraft control) for use by the US Navy. The key advantage of the PID controller is that it can be utilised and tuned without detailed knowledge of the system's model. Modern controllers are still based on the principles of PID but are more advanced. For example, in UAVs, a cascade controller is commonly used, which consists of several PID controllers in series, to accommodate different update rates of sensors.

Task: implement and tune feedback controller

Your task is to implement and tune a feedback controller on an UAV. The work is conducted in two stages:

1. Develop and test your approach in the simulator
2. Test your approach on a real UAV

The objective of the controller is to achieve the desired position and yaw by controlling x,y,z velocities and yaw rate. We will provide you with the current positions and attitudes of the quadcopter. The interface for your controller will be the same between simulator and the real experiment. Essentially, your task is to design an algorithm that takes the current state of the UAV and outputs velocity commands to guide and maintain the UAV at the desired position. As part of the coursework, you are expected to conduct independent research into controllers and tuning methods, although basic controllers and terminology was explained to you.

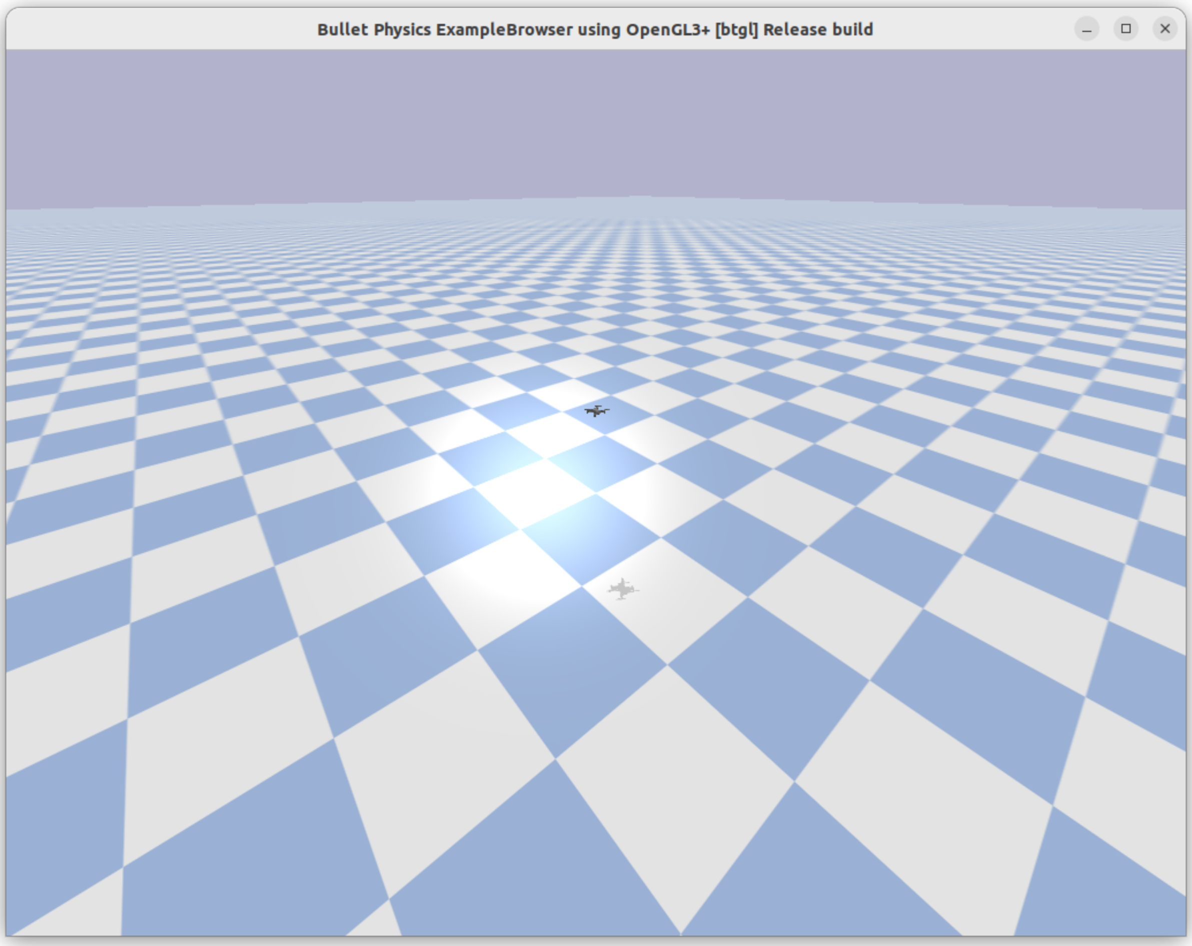
Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

Figure 3. The 3D drone simulation environment.

The simulation part of the coursework will be conducted using a custom simulator, which will feature an empty land area and a simulated 3D quadcopter. The simulator allows you to input the UAS position, attitude, and respective linear and rotational velocities into the controller function.

The experiment part of the coursework will be using a DJI tello quadcopter (https://store.dji.com/uk/product/tello?vid=38421). The interface will be the same as for the simulated quadcopter. However, physics and behaviour of the tello will not be the same. You should not expect your controller to work out of the box. The experimental Tello will be limited to flying +/- 1 meter in each direction. You will be provided with more information about experiment setup closer to the experiment.

The activity will be undertaken in teams of four. Your assigned groups are available on Blackboard. It is expected that teams will submit one Python script and four individual videos. The division of workload within the teams is at the discretion of the team members.

The marking between simulator and the experiment will be split as follows:

- **Simulator (controller performance):** During marking, we will send a random desired position (within +/- 4 meters of your vehicle) and random yaw demand to your quadcopter. Then, your vehicle will have 15 seconds to reach the desired setpoint. Afterwards, we will collect position data over 10 seconds, averaged to determine your positional error. We will also measure your standard deviation to assess the consistency of the achieved position. For fairness, this test will be repeated fifty times (each with a newly generated random position), and the results will be averaged. This will determine your final positional error, which will be used for marking.
- **Simulator (understanding):** Tested during video. Marks for this will be based on:
  - **Understanding of the controller (either PID or your advanced method you don’t need to describe both)** Provide brief background to your chosen method. Also justify why you have selected this method. If your team made several controllers, please choose and describe one. Please provide evidence of the tests in form of graphs/videos.
  - **Understanding and application of the tuning method.** Show your understanding of tuning you performed. Provide intermediary results to justify your final solution. Explain the biggest difficulty for tuning of your selected method. Please provide evidence of the tests in form of graphs/videos.
- **Experiment (understanding):** Experiment will be marked during video. The focus on the experiment is showing your understanding why your controller worked/didn’t work. Whether your controller performed well or poorly, full marks/partial marks can still be awarded, focus here is on understanding the reasons for the performance. Please provide evidence of the test in form of graphs/videos. If no evidence of performed experiment is provided, this section will receive no marks.

Please refer to the marking scheme above for exact mark allocation.

Simulator Code

Extensive installation instruction is provided to you in the coursework folder. The code is available as .zip in blackboard. Two most relevant files are *controller.py* and *targets.csv.*

controller.py

This is the file you need to submit as part of the coursework. Your controller code must go inside here as a function. Please do not modify inputs/outputs of the controller.py file.

If you decide to implement more advanced method, please indicate in the code comments which method you used.

targets.csv

Targets.csv file provides a list of target positions. You can change the target in the simulator by pressing the left and right arrow keys on your keyboard. You can reset simulator with 'r' key and quit it altogether with 'q' key. You are encouraged to define custom targets so that you are sure of your performance. When we test your code, this file will contain randomly generated goal points.

Advanced controllers list:
Here is non exhaustive list of what is considered advanced:

- Cascade PID of more than 2 layers (2 layers or less is not considered advanced)
- Disturbance observer-based controller
- Automatic parameter tuning
- Reinforcement learning
- Model predictive control
- Linear Quadratic Regulator
- Decision transformer

If it is not on the list, ask either Dr. Pawel Ladosz or Dr. Kieran Wood.

NOTES:

- No additional libraries are allowed beyond what is provided to you as dependencies of the python script. The only exceptions will be given on individual group basis, should your advanced method reasonably require it (for example machine learning framework for reinforcement learning approaches). Please talk to Dr. Pawel Ladosz over email/teams to get the package approved before submission. Packages which were not agreed on beforehand will be ignored during marking.
- It is strongly recommended that you use virtual environments (Anaconda). They allow your environment to be deleted in case of any problems.
- DO NOT MODIFY INPUT/OUTPUT variables names or order! Otherwise, automatic marking will fail, and you will not receive marks for your code. Code that does not run will score zero for performance, but other marks will still be given where possible. The testing environment will be the same as the one provided to you, thus you will have ample opportunity to test your code for correctness.
- The output from your function controls x,y,z velocity as well as yaw of the vehicle.
- If your approach is to use advanced approach in the simulator and simple PID in the experiment, you do not need to describe PID in the video. Just explain your advanced method and explain why in the experiment you used simple PID.
- In the unlikely event that you did not get flight time, and the fault was on our side, the marks for the explanation of the controller in the video will be multiplied to give you marks out of 25.

- Hint: You need to use axis transformation to get desired yaw as well as position. Think about in which coordinate frame you are given states, and in which frame you are sending commands to the tello.
- If your advanced method utilises PID controller, in the video, you do not need to describe underlying PID if you don’t want to. The focus should be on the advanced method
- The control frame of the Tello (i.e. the frame which accepts control commands) in the simulator is not rotated in pitch and roll. If you give a forward velocity even if it's tipped over it will follow global coordinate frame convention in roll and pitch.
- If you have implemented multiple controllers, please choose one and submit only one.