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 two. 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.

NOTE: In difference to the previous coursework submissions, a short video explanation must submitted.

Timeline

Week 1 – coursework is set with accompanying lecture. Familiarisation with the simulator, sending basic control commands.
Week 2 – PID implementation, vertical position tuning attempt
Week 3 – Tuning of the horizontal position controller
Week 4 – Wrap up, data collection and testing
It is expected that this task should take no more than 25 hours to complete over the 4-week period.

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 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 2 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 (data, graphs etc.) is allowed within individual teams, video structure, method and explanation should be done individually.

Marking scheme

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

Python code (10 marks):

- Successful 3D stabilisation and tracking performance [4 marks]
- Inclusion of gusts [2 marks]
- Good coding practices [2 marks]
- Advanced methods demonstrated (i.e. state estimators, DOBC, cascade controllers, LQRs, reinforcement learning etc.) [2 marks]

Video (15 marks):

- Explanation of selected method [5 marks]
- Explanation of tuning method [5 marks]
- Overall video quality [3 marks]
- Explanation of advanced methods used (i.e. state estimators, DOBC, cascade controllers, LQRs, reinforcement learning etc.)) [2 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.

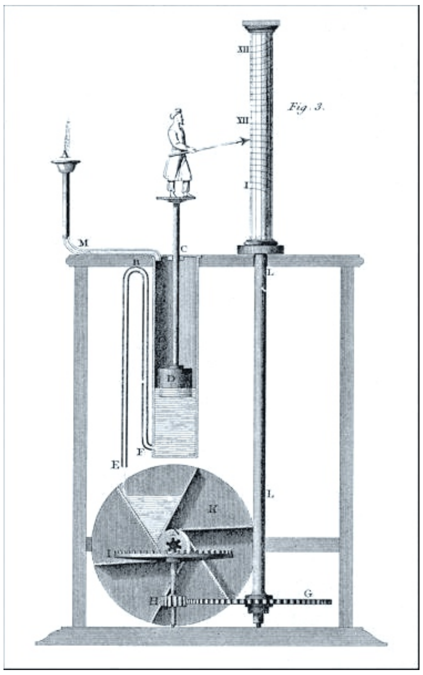

Figure 1. Ctesibius's water level regulator

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.

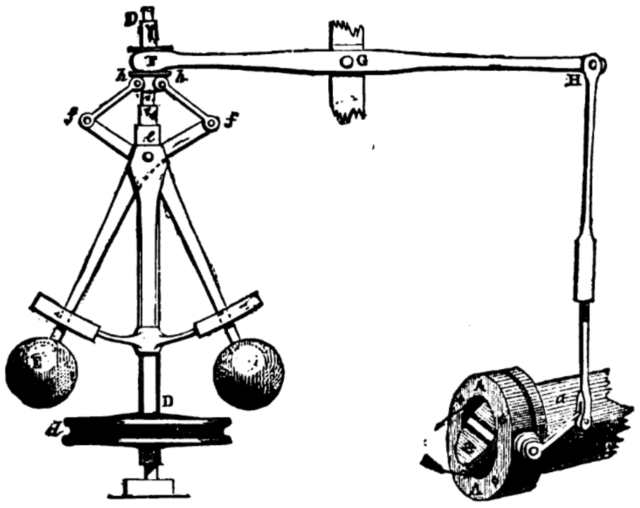

FIG. 4.—*Governor and Throttle-Valve.*

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 a 2D simulated UAV (see screenshot below). The objective of the controller is to achieve the desired position by controlling the thrust from two motors. The simulator will provide you with the current positions, velocities, and attitudes of the simulated quadcopter. Essentially, your task is to design an algorithm that takes the current state of the UAS and outputs thrust commands to guide and maintain the UAS 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 will be explained to you.

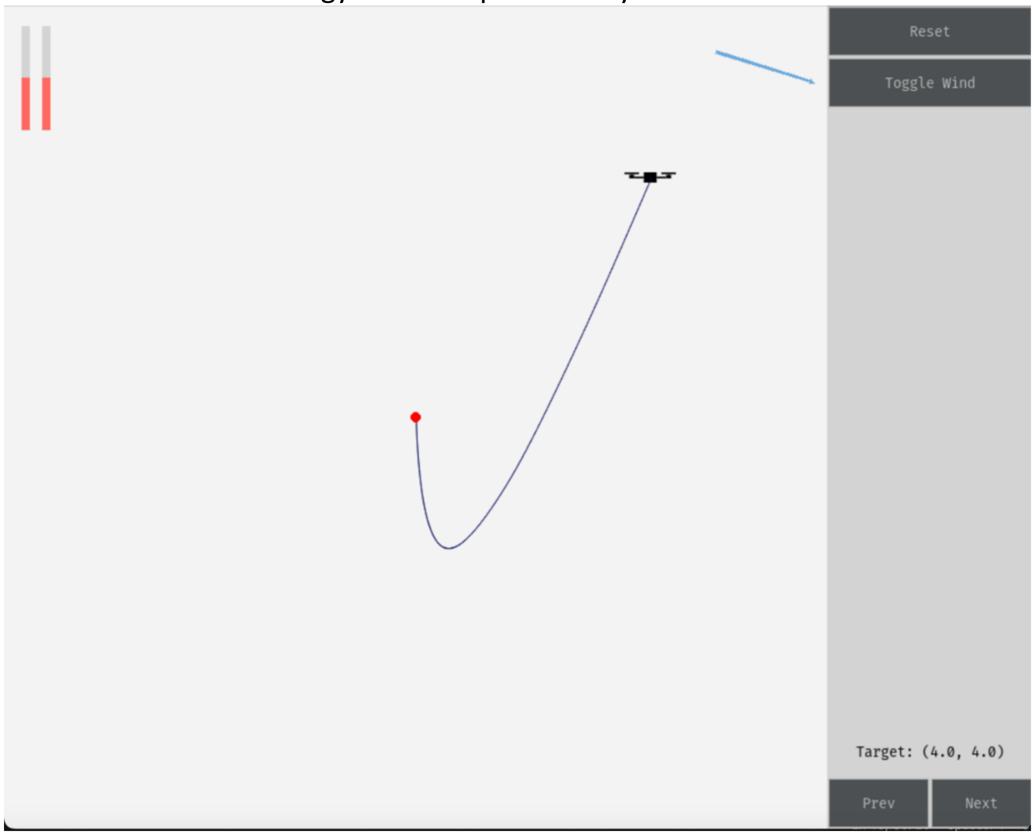

Figure 3. The 2D drone simulation environment.

Completing the coursework will require you to understand and implement the concept of throttle mixing. An extensive explanation and an example of how to perform this in 2D can be found here: https://cookierobotics.com/066/.

The coursework will be conducted using a custom simulator, which will feature an empty land area and a simulated 2D quadcopter. The simulator allows you to input the UAS position, attitude, and respective linear and rotational velocities into the controller function. The output of the function should be normalised thrust (0 to 1). Instructions for installing and running the simulator will be provided. The environment includes an option to introduce gusts. You will receive extra marks if your controller performs well in the gusty environment and for advanced method implementation.

The activity will be undertaken in teams of two. Your assigned groups are available on Blackboard. It is expected that teams will submit one Python script and two separate videos. The division of workload within the teams is at the discretion of the team members. During marking, we will send a random desired position to your quadcopter (within +/- 4 meters of your vehicle). Then, your vehicle will have 10 seconds to reach the desired position. 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 five times (each with a different random position), and the results will be averaged. This will determine your final positional error, which will be used for marking.

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. You must indicate if your controller works with wind and your group number. You must do so by modifying top two lines of the controller.py:

```
wind_active = False  # Select whether you want to activate wind or not
group_number = None  # Enter your group number here
```

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 prev/next button. 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.

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). The packages approval can ONLY happen during lab sessions/lectures. Packages which were not agreed on beforehand will be ignored during marking.
- It is strongly recommended that you use virtual environments (Anaconda) just like the last coursework. 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. 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 trust output from your function should be normalized between 0 and 1.
- Your group number modifies some properties of the quadrotor. Group number you entered will be checked against our record. If it is incorrect we will change to the correct one and mark your performance based on the correct group number.