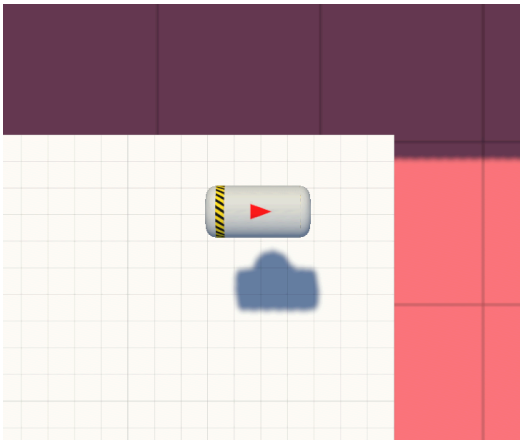
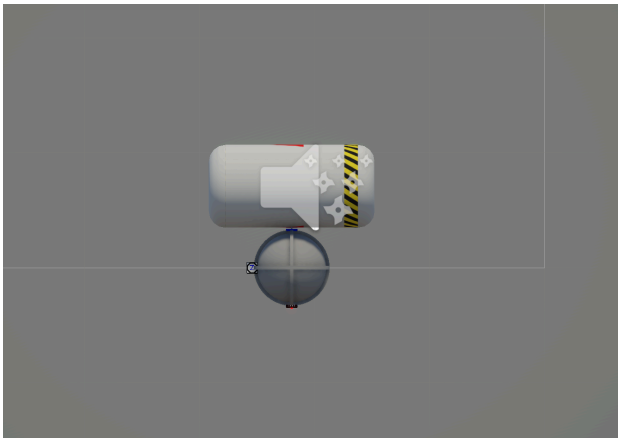
Introduction to Robotics: Project2 Report

Team 9
Name: 송우영, 양성지, 박정훈

1. Performance Record

Time [s]: 26.84s
Number of Collisions: 0
Final Time [s]: 26.84115s

2. Participation Details

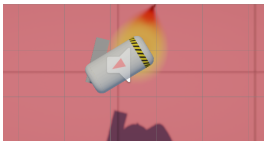
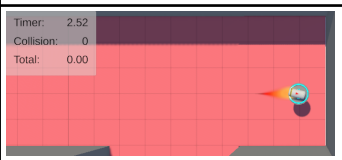
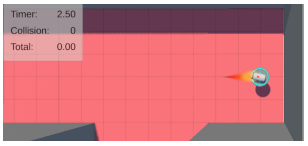
Rocket Design:

1. **Simple Structure (Seong-Ji Yang)**

   We kept the rocket body as simple and lightweight as possible. A minimal spherical rocket design reduces unnecessary inertia and avoids complex roll/pitch coupling, making the rocket's rotational behavior more predictable and easier to tune.

2. **No Airbrake (Seong-Ji Yang)**

   Airbrakes were removed because they consistently caused slight performance degradation. They introduce unnecessary deceleration, shorten full-throttle segments, and increase re-acceleration time. Ablation tests showed that the no-airbrake setup produced more consistent trajectories and faster overall times. Also, when combined with the simplest rocket structure, it changed center of mass, creating unwanted momentum on the body itself, shifting the body's degree, thus making the control of the servo motor even harder.

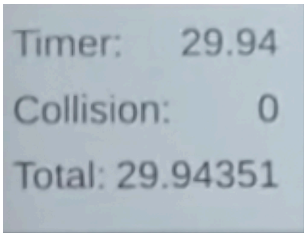
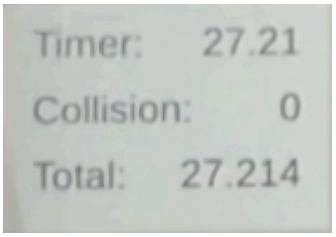
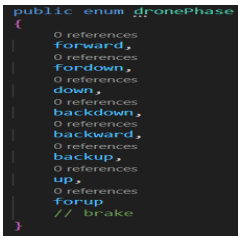
body turning due to brake	with brake	without brake
		

Control Implementation:

1. **Initial Implementation (Seong-Ji Yang)**

- Design objective:
  Like the implementation of the TA session, we tried to implement in a way to define the direction of the rocket, and use it as a switch for control. However, to maximize control, we needed to consider every degree, which would have made the state's number total 360. However, we thought that making that much switch would make implementation harder, but if we use the time that the servo motor's degree changes as an action of switch, we would be able to make a more simple, yet same accurate design.

- Implementation Detail & Verification Process:
  Therefore, to prove our theory, we made a simple testbed. We made two rockets, one with mostly 4 direction states for control, and the other for 8 directions for control. When we tried to improve this simple design as much as possible, the result proved that more accurate control meant better performance, proving the necessity of more states.

4 direction	8 direction	test bed states
		

2. **Smooth Rotation Implementation (Jeonghoon Park)**

- Design Objective:
  In sections like C, we wanted the engine direction to change gradually rather than rotate abruptly. Manually implementing this behavior for every turning point would be complex and impractical, so a smooth rotation method was added to automatically generate gradual heading changes.

- Implementation Detail:
  Smooth rotation is applied only when specifically enabled for certain segments. When activated, the system checks the next target angle and the time until the next update, then computes the incremental change.

$$\Delta\theta = \frac{\theta_{\text{target}} - \theta_{\text{current}}}{\Delta t}$$

- Verification Process: (Song WooYoung)

To evaluate the effect of rotation interpolation, we compared two versions of the servo timeline:

1. **servo_timeline**
   – The original timeline used during most trials (baseline).

2. **servo_timeline_smooth**
   – A modified version where the rotation segments in the C section were smoothed using rotate = 1.

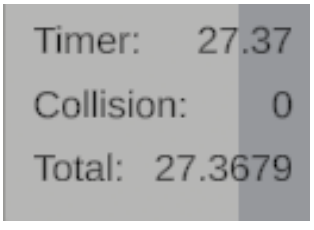
Across the full course, both versions demonstrated **highly similar overall trajectories**, confirming that large-scale movement is dictated primarily by corridor geometry and thrust rather than small variations in yaw commands. However, the behavior inside the C-shaped corridor revealed subtle but important differences.

In the **smooth version**, continuous yaw blending sometimes introduced intermediate angles that were not aligned with the intended movement direction. As a result, the drone occasionally drifted outward along the curve, requiring additional micro-corrections to return to center. This overshoot behavior increased the effective path length and introduced minor lateral instability.

In contrast, the **baseline version** executed each yaw step directly toward the intended heading. Although the drone's rigidbody still enforces physically smooth rotation, the absence of unnecessary blended angles helped it stay closer to the desired line through the C-section.

- Results: (SongWooYoung)

The comparison shows that rotation smoothing does **not** always yield a performance advantage. While smooth interpolation is conceptually appealing, applying it uniformly across multiple direction changes can introduce unintended drift.

servo_timeline	servo_timeline_smooth.
	

Key observations:

- **Smooth (`rotate = 1`) should be used only in segments where turns are sharp and abrupt.**

In these cases, smoothing softens the transition and avoids angular spikes.

- **Applying smoothing across moderate or gentle turns is counterproductive.**
  Interpolation generates intermediate headings that are unnecessary, causing the drone to deviate outward before correcting back.

- **The baseline (discrete) version generally maintains a cleaner trajectory**, because each yaw command points directly along the reference path without additional blending.

- **Instant mode (rotate = 0 for the entire timeline) is not recommended.**
  Even though physical rotation is inherently smooth due to inertia and damping, forcing large instantaneous yaw commands:
  - creates abrupt changes in desired direction,
  - increases temporary thrust misalignment,
  - and ultimately requires extensive servo-timeline redesign to avoid instability.

These findings suggest that the optimal strategy is a **hybrid** one: Use smooth interpolation only at major turning points.

This prevents unnecessary interpolation drift and keeps the drone centered in the C corridor.

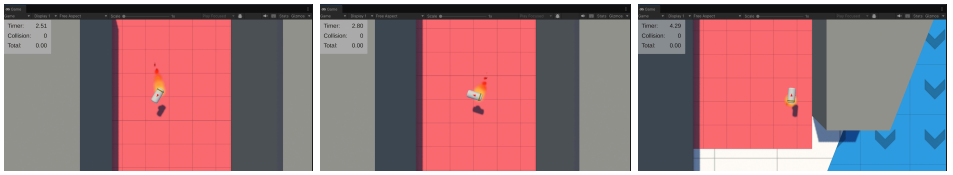
Although the overall time differences between the baseline and smooth versions were small, the trajectory analysis shows that **full smooth mode does not improve stability**. Instead, **selective smoothing** produces the most reliable and efficient path.

3. Rotation Direction Control (Jeonghoon Park)

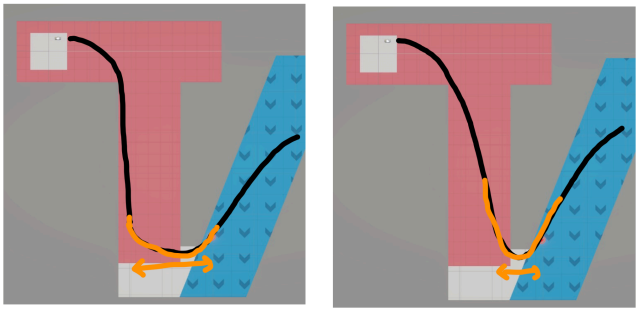
- Design Objective:
  The default controller always rotates toward the smaller angle difference. Inspired by [Youtube] Runner U-turn Drill, we implemented an option to rotate in the opposite direction in narrow corridors, where rotating “normally” increases the chance of collision due to limited space and the lack of braking during rotation.

- Implementation Detail:
  Since the raw rotate value in the CSV could not guarantee the desired turning direction, we redefined:
  - rotate = 1 : force clockwise rotation
  - rotate = -1 : force counterclockwise rotation
  - rotate = 0 : no forced rotation (default shortest-angle rotation)

- Verification Process:
  We tested the behavior in the narrow T => A section, where a left U-turn is required.
  1. Default counterclockwise method: To avoid hitting the right wall in the narrow corridor (due to right-side thrust during rotation), we shifted the drone as far left as possible before entering the turn and repeated trials to set the path to the innermost line.
  2. New clockwise-turn method: Similarly, repeated trials were performed to set the path to the innermost line.



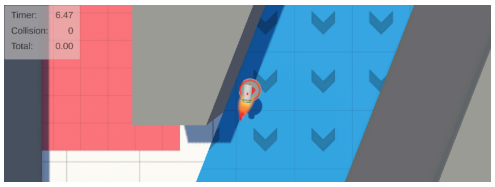
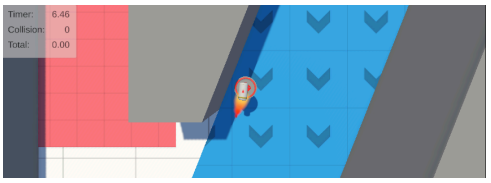
[U-turn executed with clockwise rotation]



[Left : 1. counterclockwise] [Right : 2. clockwise]

- Results - ablation Study (Song WooYoung):
  Normally the drone rotates toward the direction with the smaller angular difference, but the T => A section is extremely narrow, so reverse rotation becomes advantageous.

  At the same location with LU it has much higher speed than w.o LU.

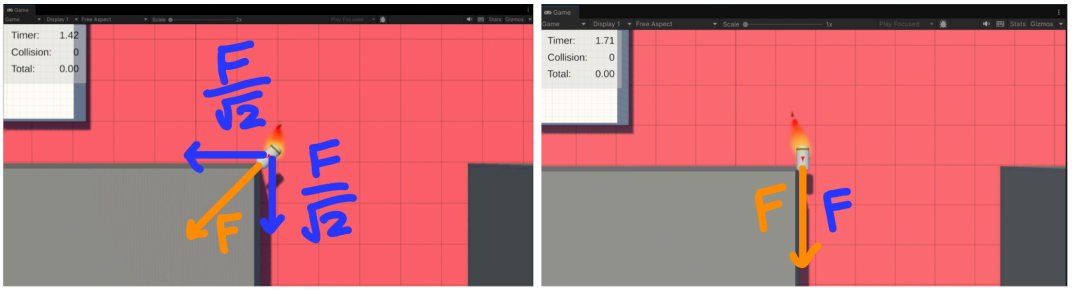
instantaneous velocity w.o. LU (6.47s)	instantaneous velocity w. LU (6.46s)
	
7.8144	9.6818

4. Starting Position Adjustment (Jeonghoon Park)

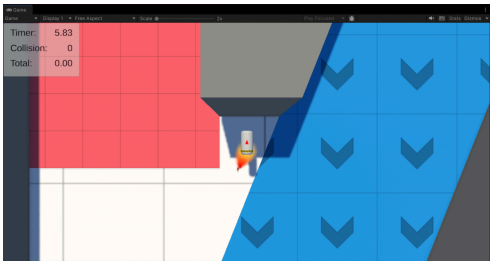
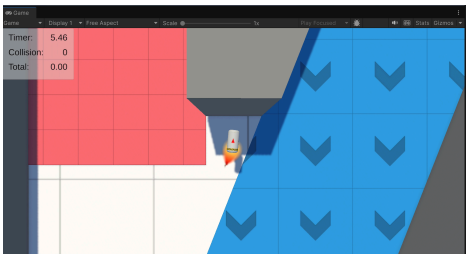
- Design Objective:
  The downward slope in section T is long and narrow, so the drone must generate as much downward thrust as possible while minimizing lateral thrust. If the drone rotates more than 90° before entering this slope, some engine power is wasted counteracting inertia. Therefore, it is optimal to begin the timed run already facing 90° downward, so the initial thrust is used almost entirely in the downward direction. Since the starting position is adjustable, we positioned the drone to start the run with this 90° orientation inside the allowed starting area.

- Verification Process:
  We compared two starting positions: the lower side (-3.2, 0, -1.5) and the upper side (2.2, 0, 3.2), and measured how long it took for the drone to exit section T. To ensure fairness, each starting position was tuned through repeated trials so that both entered the T => A corner along the inside line and produced their best possible trajectory. As an intermediate checkpoint, we also recorded the time and instantaneous speed when the drone passed the first corner inside section T and used these values for comparison.

- Results:



Although the downward-start configuration takes slightly longer to settle at the very beginning, it allows 100% of the thrust to be used in the downward direction (right image), greatly increasing vertical descent speed while minimizing unwanted lateral velocity.
The instantaneous velocity shows that at the same place with different start point, the z axis velocity of (2.2, 0, 3.2) closer to 0, which means it can soon change the direction to where it should go.

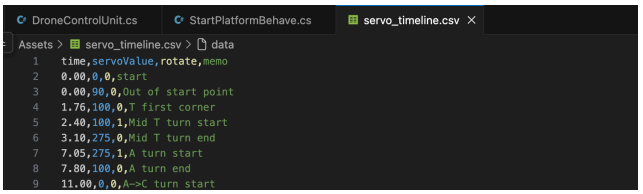
Starting position at lower side (-3.2, 0, -1.5)	Starting position at upper side (2.2, 0, 3.2)
	
5.83(s)	5.46(s)

5.7435 (5.4779, 0, -1.7266)	4.5882 (4.5866, 0, -0.1197)

Other Implementations

1. **Servo turning timeline CSV File Making & Loading (Jeonghoon Park)**

   Previously, any change in rotation timing required modifying DroneControlUnit.cs, recompiling, and manually tracking values, which reduced readability and slowed debugging. To simplify testing, the system was modified to load control values directly from a CSV file.



2. **Using time as a state (Seong-Ji Yang)**

   To use time as a state, we needed to know how many frames were used to calculate 1 second. Thus, we checked Edit → Project Settings → Time → Fixed Timestep, which told us 0.02 second per frame.

3. **Synchronizing Timer With Test View (Jeonghoon Park)**

   The internal time used in ServoKeyframe (sec/frame) starts immediately at launch, which did not match the visible on-screen timer used during testing.
   This mismatch made debugging inconvenient, so we adjusted the internal timing to match the test timer.

4. **Instantaneous Speed Logging (Song WooYoung)**

   The *Instantaneous Speed Logging* module records the drone's velocity at fixed time intervals (e.g., every 0.1 seconds) throughout the entire flight. At each interval, the logger captures the X-, Y-, and Z-axis velocity components as well as the overall speed magnitude, and stores them in a CSV file for later analysis. This enables precise comparison of different control strategies—such as smooth versus instant rotation or clockwise versus counterclockwise turning—by providing a quantitative measure of how the drone accelerates, decelerates, and stabilizes under each configuration. Since all values are measured directly from the physics engine, the logged data reflects the true physical behavior of the system without requiring any manual input during testing.

5. **Version synchronizing (Seong-Ji Yang)**

   We found that the GetKeyDown() function created an error which seemed to be a problem due to the function used in the lower unity version, making rocket control

impossible. Thus we changed Edit → Project Settings → Player → Active Input Handling from New to Both, which solved the problem.

3. Results

- From our rocket design test experiments, we found that a rocket with no brake was easier to control and also able to gain more inertia.
- We also made the rocket to be as close to the starting point as possible. Since if we start at the back, it means more momentum, meaning the rocket colliding to the wall of the first downward T lane, also meaning backward thrust, slowing down speed.
- We also made the rocket turn clockwise on the end of the first T lane to avoid collision.

4. Discussion (optional)

1. What did your team learn from this project?

- We learned that as the drone's speed increases, the efficient use of inertia becomes critical. Small differences in servo angle or timing accumulate rapidly along the trajectory, making precise control scheduling far more important than raw engine thrust. This emphasized the importance of optimizing both the turning angle and the exact timing at which each rotation begins.

2. What would you improve if you had more time?

- Because our smooth-rotation implementation required knowledge of upcoming turns, we could not directly rely on checkpoint-based detection during the run. With more time, we would incorporate checkpoint-driven control logic to perform more detailed and fair ablation studies, allowing each turning strategy to be evaluated under identical conditions.
- Additionally, applying mathematical curve-generation methods—such as smooth spline interpolation between two key positions—could help produce more natural trajectories and reduce abrupt changes in orientation. This would likely improve both stability and overall time performance.
- adding many brakes so that the rocket's center of mass remains the same. Also, which might be able to help speed up even if the rocket's mass increases, we could reduce the exact direction of inertia, while using the full force of the engine.