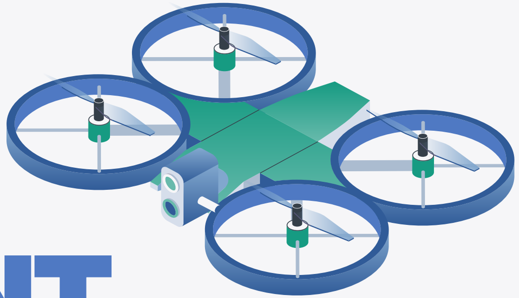
REINFORCEMENT LEARNING

TERM PROJECT

GROUP - 3



Learning Safe Flight Manoeuvres for a Mini-Drone

Members

Yash Sirvi (21CS10083)
Adyan Rizvi (21MA10006)
Sreejita Saha (21CS30052)
Soumojit Bhattacharya (21EC10071)
Allen Emmanuel Binny (21EC39035)
Anurag Sharma (24AI91R01)

INTRODUCTION

This project aims to develop a policy for a Crazyflie 2.1 mini-drone that enables it to perform stable and collision-free manoeuvres in a controlled environment. The primary objective is to teach the drone to hover at a target altitude while safely navigating towards a designated destination, avoiding obstacles along its path. The Crazyflie 2.1 mini-drone testbed will serve as the physical platform for testing and validating our policy. Through reinforcement learning, we aim to train the drone to autonomously adapt to various scenarios, maintaining stability and safety under different conditions.

CONTRIBUTIONS

Yash Sirvi (21CS10083) - Integrated SB3 PPO with gym environment, Implemented physical environment wrapper to allow for ‘wait’, ‘restart’ of device without interruption in training. Verified algorithm by testing hover task in simulation. Helped in physical testing.

Adyan Rizvi (21MA10006) - Setup Crazyflie lib, Performed Initial Testing of drone, Developed the initial gym environment, Performed offline training of the drone.

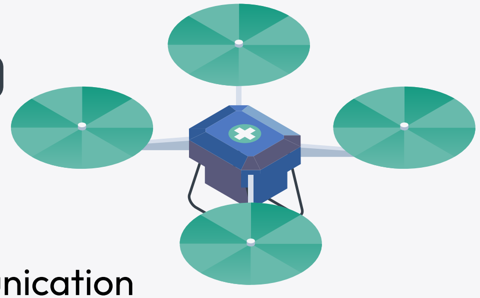
Sreejita Saha (21CS30052) - Developed the initial gym environment, trained the drone, helped in setting up the connection with the crazyflie, made the presentation.

Soumojit Bhattacharya (21EC10071) - Helped in offline training. Offline Testing using Gym Wrapper. Worked on DDPG training for Hover and movement to specific position.

Allen Emmanuel Binny (21EC39035) - PID Controller for Expert Policy. Setup of Simulators. Helped in Physical Testing.

Anurag Sharma (24AI91R01) - Helped in drone testing. Working on hover while moving to the destination region in the simulation.

Crazyflie Library and Client Setup

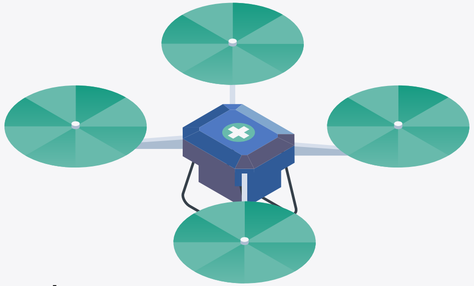


Installation of Crazyflie libraries and client:
The Crazyflie firmware and Python client library (`cflib`) are required for communication with the drone. To verify connection with the crazyflie, the PC client (`cfclient`) is installed.

Connect Crazyflie to System:
USB Radio Dongle: Plug in Crazyradio PA.
Drivers: With linux, the crazyradio is easily recognized, but we had to setup UDEVpermissions.

Entire installation guidelines are given in the below links :
https://github.com/bitcraze/crazyflie-lib-python/blob/master/docs/installation/install.md
https://github.com/bitcraze/crazyflie-clients-python/blob/master/docs/installation/install.md

PROBLEM FORMULATION



Environment Setup: Gym API was used to setup the environment where action space, observation space and reward function are defined.

Action Space: Thrust, roll, pitch and yaw (actions that control crazyflie). We have used a discrete action space with 10 discrete values for each action.

Observation Space: Nine-dimensional vector capturing the drone's position, velocities, and orientations i.e, x, y, z, vx, vy, vz, roll, pitch, yaw.

Reward System: Calculated from factors like position error, attitude stability, and penalties for drifting outside safe boundaries. Positive rewards are given to keep the drone within a 0.1m tolerance of the target position .

Logging and Safety Mechanisms: Real-time state updates through logging configurations and emergency stop features .

Safe Operation Constraints: Ensuring the drone remains within a predefined safe boundary to prevent collisions or accidents.

IMPLEMENTATION OF RL POLICY

The RL policy implemented in this project is **Proximal Policy Optimization(PPO)**

Objective : PPO aims to maximize the expected cumulative reward obtained from interacting with the environment. This is typically done by maximizing a surrogate objective function that approximates the policy improvement.

Features :

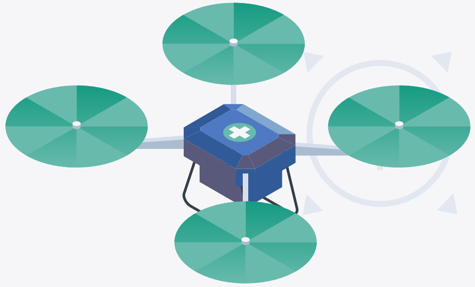
- PPO is an online actor-critic method
- It has a clipped surrogate objective. By clipping the ratio between the probability of actions under the new policy and the old policy, PPO ensures that the policy update stays within a safe range.

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s,a)\right)$$

Algorithm Steps:

- A neural network is initialized that represents the policy. This network takes the current state as input and outputs a probability distribution over possible actions.
- The agent interacts with the environment by following its current policy. During this interaction, it collects trajectories consisting of states, actions, and rewards.
- Using the collected trajectories, estimates of the advantages for each state-action pair are computed .
- The surrogate objective is computed. This objective approximates policy improvement while ensuring that the policy update remains within a safe range.
- Stochastic gradient descent (SGD) or a variant is used to update the parameters of the policy network, minimizing the surrogate objective.

Repeat : Steps 2–5 are repeated for multiple iterations or until convergence.



PID CONTROL FOR Bootstrapping

Traditional Control of the Drone is done through:

- Outer Controller Loop - Position Control
- Inner Controller Loop - Attitude Control

Methodology :

- We tried to develop a PID Controller for the Position Control for Hover

$$\text{Thrust} = 40000 + (K_p^t \cdot e^t) + \left(K_d^t \cdot \frac{\partial e}{\partial t}\right) + (K_i^t \cdot \sum e^t)$$

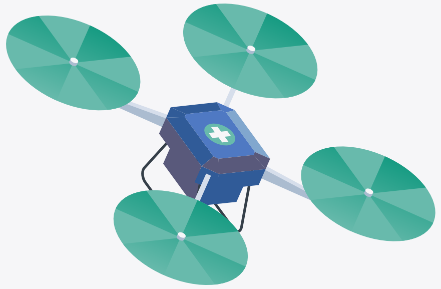
$$\text{Roll} = (K_p^r \cdot e^r) + \left(K_d^r \cdot \frac{\partial e}{\partial r}\right) + (K_i^r \cdot \sum e^r)$$

$$\text{Pitch} = -(K_p^p \cdot e^p) - \left(K_d^p \cdot \frac{\partial e}{\partial p}\right) - (K_i^p \cdot \sum e^p)$$

PID CONTROL FOR Bootstrapping

Challenges :

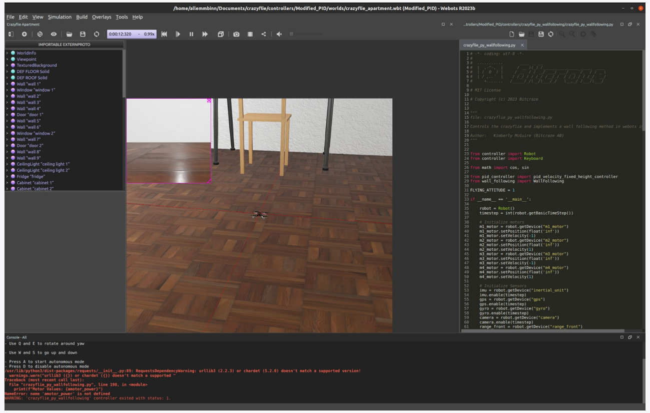
- Initially configured the Kp values to match attitude stability however there was a constant drift in the drone towards the x and y direction
- While we tried to configure the Ki values to compensate for the drift, constant flipping of the drone occurred making the tuning of PID extremely difficult



SIMULATOR SETUP

Worked on 3 different simulator setup

- Webots
  - Challenge - Controlled directly the individual motor thrusts, which in real bot, only rpy-thrust setpoints
- CrazySim
  - On GazeboSim - Able to spawn the bot
- Gym-pybullet-drones
  - Pybullet based simulator
  - Almost similar values in URDF, matched with SDF file as published by Crazyflie
  - Mismatch in action space: provides rpm, but not roll, pitch, yaw and thrust input


/home/allemmbinn/Documents/crazyflie/controllers/Modified_PID/worlds/crazyflie_apartment.wbt (Modified_PID) - Webots R2023b
crazyflie_py_wallfollowing.py
Console - All

```python
# -*- coding: utf-8 -*-
#
# MIT License

# Copyright (c) 2023 Bitcraze


"""
file: crazyflie_py_wallfollowing.py

Controls the crazyflie and implements a wall following method in webots i

Author:   Kimberly McGuire (Bitcraze AB)
"""


from controller import Robot
from controller import Keyboard

from math import cos, sin

from pid_controller import pid_velocity_fixed_height_controller
from wall_following import WallFollowing

FLYING_ATTITUDE = 1

if __name__ == '__main__':

    robot = Robot()
    timestep = int(robot.getBasicTimeStep())

    # Initialize motors
    m1_motor = robot.getDevice("m1_motor")
    m1_motor.setPosition(float('inf'))
    m1_motor.setVelocity(-1)
    m2_motor = robot.getDevice("m2_motor")
    m2_motor.setPosition(float('inf'))
    m2_motor.setVelocity(1)
    m3_motor = robot.getDevice("m3_motor")
    m3_motor.setPosition(float('inf'))
    m3_motor.setVelocity(-1)
    m4_motor = robot.getDevice("m4_motor")
    m4_motor.setPosition(float('inf'))
    m4_motor.setVelocity(1)

    # Initialize Sensors
    imu = robot.getDevice("inertial_unit")
    imu.enable(timestep)
    gps = robot.getDevice("gps")
    gps.enable(timestep)
    gyro = robot.getDevice("gyro")
    gyro.enable(timestep)
    camera = robot.getDevice("camera")
    camera.enable(timestep)
    range_front = robot.getDevice("range_front")
```

```
- Use Q and E to rotate around yaw

- Use W and S to go up and down

- Press A to start autonomous mode
- Press D to disable autonomous mode
/usr/lib/python3/dist-packages/requests/__init__.py:89: RequestsDependencyWarning: urllib3 (2.2.3) or chardet (5.2.0) doesn't match a supported version!
  warnings.warn("urllib3 ({}) or chardet ({}) doesn't match a supported "
Traceback (most recent call last):
  File "crazyflie_py_wallfollowing.py", line 198, in <module>
    print(f"Motor Values: {amotor_power}")
NameError: name 'amotor_power' is not defined
WARNING: 'crazyflie_py_wallfollowing' controller exited with status: 1.
```

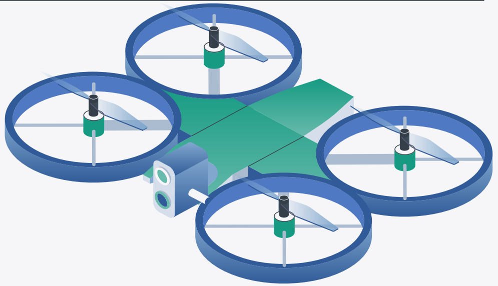
REAL-TIME TRAINING

Hyperparameter and training setup:
Learning Rate : 0.005 , learning rate for stable convergence.
Batch Size : 16 , for balancing update frequency and memory efficiency for effective training.
GAE Parameter(λ) : 0.95 , for stable advantage estimation.
Discount Factor(γ) : 0.98 , ensuring balance between long-term and short-term rewards.
Clipping Range : 0.1 , to control the update magnitude.

Training Process:

- Initial training encourages exploration to learn diverse strategies.Exploitation dominates in later stages for fine-tuning stability.
- Regular checkpoints are saved to store stable policy versions and prevent data loss during crashes ,unstable episodes or battery loss of drone.
- Adjustments based on Crazyflie's feedback loop for position corrections and emergency stops if limits are breached.

CHALLENGES

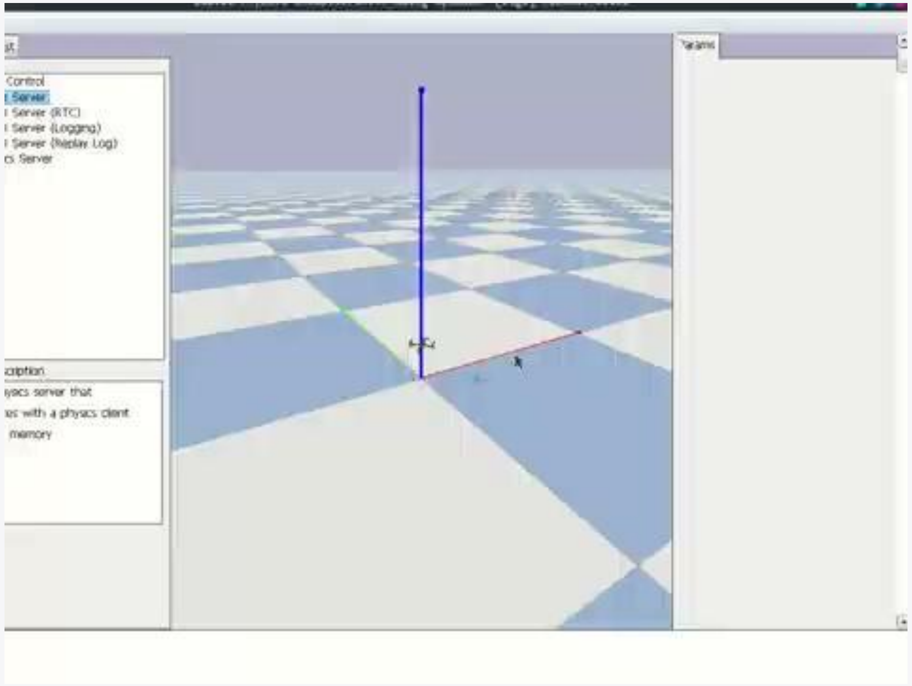


- Sensors in the drone especially in the x,y axes were not very accurate.
- Frequent battery loss of the drone resulting in training disruption.
- Whenever the drone hits the net, it assumes it is the new base instead of the foam base.
- Mismatch in the action space of the motion commander and cflib . Motion Commander takes waypoints, hoverpoints(mainly high level) as action space while cflib takes position,orientation,velocities as action space (mainly low level control commands). Additionally, the hover in motion commander publishes their velocity setpoints directly into thread.
- Mismatch in the action space of crazyflie simulator and cflib. In simulator, we need rpm or torque and in cflib, we require thrust as the action space.
- Collecting data in the real time is time consuming.
- Multiple challenges when the drone was close to the ground including very inaccurate readings and ground effect where the drone seems to glide across the ground.

DEMONSTRATION

Training in real-time proved to be quite challenging. Despite extensive efforts, the final outcome showed that the system attempted to hover at the specified height but eventually drifted sideways.

SIMULATION RESULTS



Results on pybullet-gym environment for crazyflie drone with PPO and 10^6 Steps (~1000 episodes)

Reward = 2-norm(pos-target) - norm(pos_xy - target_xy)

GOAL STATE NAVIGATION [sim]

- Goal position within the environment:
  Distance to goal: dist_to_goal = math.sqrt((x-x_goal)^2 + (y-y_goal)^2+ (z-z_goal)^2)

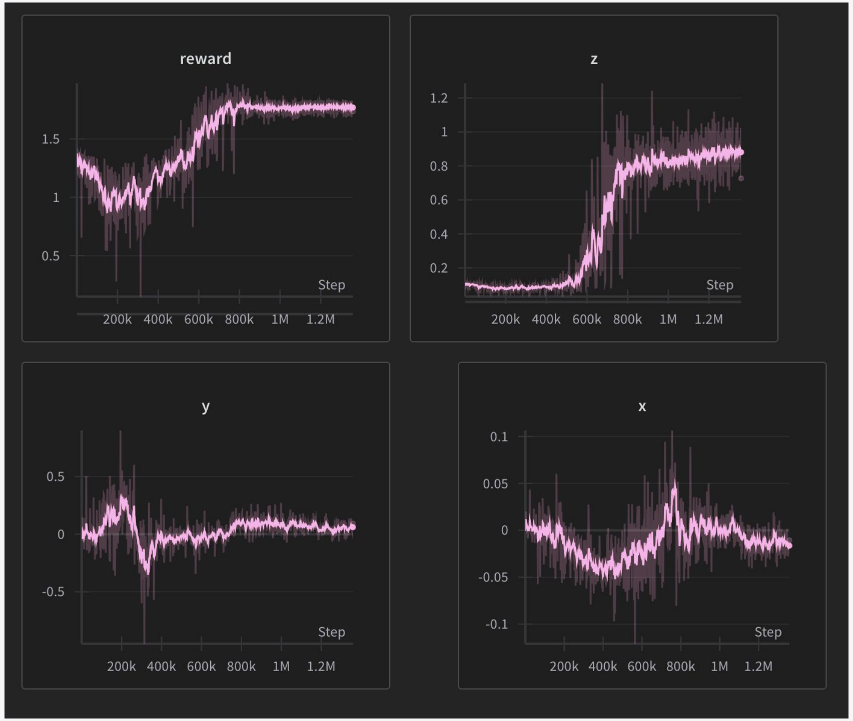
- Distance Reward:
  Reward: $-\alpha1$ * dist_to_goal

- Height Reward: h` = |z - h_target|
  Reward: $-\alpha2$ * h`

  $\alpha1$, $\alpha2$= constant penalizing distance

- Goal Reward: When the drone reaches within a specified tolerance of the goal position.
  if distance_to_goal < d_tolerance and h` < h_tolerance:
  reward += β tolerance = 0.5 or 1.0
- Termination Condition: the episode terminates once the drone is within a defined tolerance.
  terminated = distance_to_goal < tolerance

SIMULATION RESULTS



Results on pybullet-gym environment for crazyflie drone with DDPG and 1.2*10^6 Steps (~1200 episodes)

Reward = max(0, 2-norm(target_pos-pos)^4)

After a 10^6 steps the drone seems to be able to reach the target height consistently The change in x and y position also decreases after further training

THANK YOU