RBE595 - Reinforcement Learning
End-to-End Motion Planning of Quadrotors Using Deep Reinforcement Learning

Author: Efe Camci and Erdal Kayacan
Project: Giorgio Mendoza

Introduction

- **Develop a Deep Reinforcement Learning (DRL) Model**: Implement a DRL system that utilizes raw depth images for navigating quadrotors in cluttered environments.
- **Utilize Depth Images for Navigation**:
  a. Generate and use depth images obtained from a front-facing camera.
  b. Transform depth images into local motion plans using deep neural networks.
- **Create Smooth Motion Primitives**:
  a. Employ Bézier curves to formulate smooth, dynamic motion paths (if possible).
  b. Train the model to select optimal motion primitives based on real-time environmental data.
- **Achieve Autonomous Navigation**:
  a. Enable the quadrotor to autonomously navigate without prior obstacle location information, relying solely on visual inputs.
- **Replicate System Functionality**:
  a. Validate the model's effectiveness in AirSim simulations, mirroring the original study's setup and results.

Algorithm Overview: Deep Q-Network (DQN) for Action-Value Estimation

- **Purpose:** Estimate action-value function Q(s,a)Q(s,a) for decision-making in environments with high-dimensional action spaces.
- **Deep Q-Network (DQN) Structure:** Combines convolutional neural networks (CNNs) and fully connected layers.
  a. Uses 2 main lanes: First lane & Second Lane
- **Processing Pipeline:**
  a. First lane, Second Lane
- **Combination and Output:** Combines outputs for both lanes
- **Training and Optimization:**
  a. Uses the Bellman equation to train the DQN, incorporating rewards and discount factors for future rewards.
  b. Employs the Huber loss for stability during training.
  c. Optimized using the Adam optimizer, a method for stochastic gradient descent.

State

- Current situation or status of the agent in environment
- **Components:**
    a. Depth Image - 32 x 32 pixel image
    b. Relative Position Information
- **Integration in State Definition:**
    a. Provides the agent with comprehensive situational awareness.
- **Functionality:**
    a. Allows the quadcopter to understand its environment

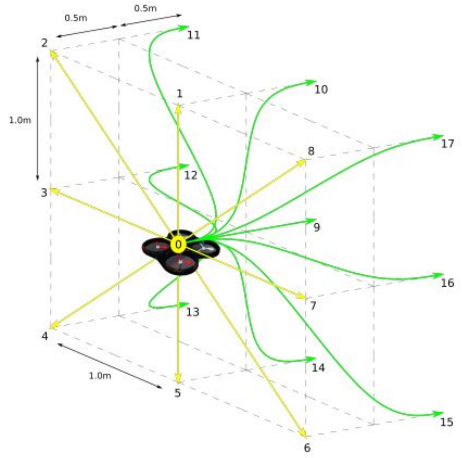
Rewards

- To assess the quality of the quadrotor's actions based on its movement relative to a dynamic setpoint.
- **Components:**

$$R = \begin{cases} \dfrac{R_l}{d_t}, & \text{for } \Delta d_u < \Delta d \\ \dfrac{R_l + (R_u - R_l)\frac{\Delta d_u - \Delta d}{\Delta d_u - \Delta d_l}}{d_t}, & \text{for } \Delta d_l \leq \Delta d \leq \Delta d_u \\ \dfrac{R_u}{d_t}, & \text{for } \Delta d < \Delta d_l \\ R_{dp}, & \text{for excessive deviation,} \\ R_{cp}, & \text{for collision.} \end{cases} \qquad (3)$$

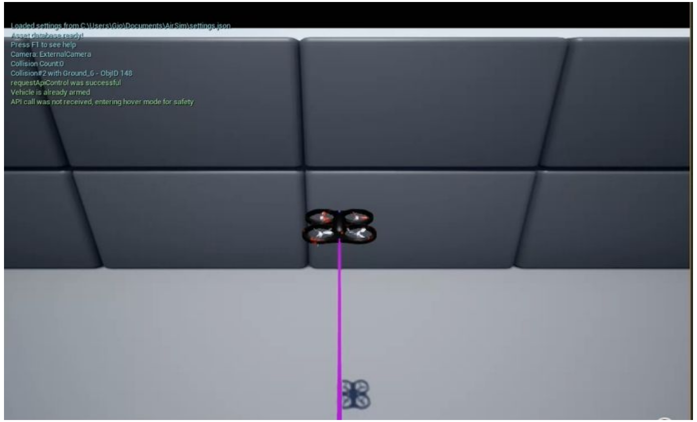
Actions

- **Action Definition:** The agent's move at each timestep, designed to increase rewards through smooth motion primitives.
- **Bézier Curves for Motion**
  a. **Parametric Definition**
  b. **Control Points**
- **Advantages:**
  a. **Smooth Trajectories**
  b. **Reactivity**



RL Framework

- The RL framework and tools used for this project consists of:
    a. Airsim
    b. Open AI Gym env
    c. stable-baselines3



CI passing | docs passing | coverage 96.00% | code style black

Stable Baselines3

OpenAI Gym

Architecture/Implementation

- The implementation of this project consists of three Python files and a JSON file.
    a. Json FIle for initial config
    b. Movements
    c. DroneEnvironment
    d. Main file

```json
{
  "SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings.md",
  "SettingsVersion": 1.2,
  "SimMode": "Multirotor",
  "ViewMode": "NoDisplay",
  "ClockSpeed": 100,
  "CameraDefaults": {
    "CaptureSettings": [
      {
        "ImageType": 4,
        "Width": 32,
        "Height": 32,
        "FOV_Degrees": 90,
        "AutoExposureSpeed": 100,
        "MotionBlurAmount": 0
      }
    ]
  }
}
```

```python
    current_yaw = get_current_yaw(client)
    yaw_radians = math.radians(current_yaw)
    vx = math.cos(yaw_radians) * forward_speed
    vy = math.sin(yaw_radians) * forward_speed

    # Calculate the current altitude from which to start the descent
    current_altitude = client.getMultirotorState().kinematics_estimated.position.z_val
    desired_altitude = current_altitude + 5  # Assuming you want to move down by 5 meters

    # Move forward and downward simultaneously
    client.moveByVelocityZAsync(vx=vx, vy=vy, z=desired_altitude, duration=duration).join()


def move_diagonally_down_yaw(client, yaw_change, duration, forward_speed, downward_speed):
    # Rotate to the new yaw
    current_yaw = get_current_yaw(client)
    target_yaw = current_yaw + yaw_change
    client.rotateToYawAsync(target_yaw, timeout_sec=duration).join()
    # Calculate diagonal movement velocities
    vx = forward_speed * math.cos(math.radians(target_yaw))
    vy = forward_speed * math.sin(math.radians(target_yaw))
    z = client.getMultirotorState().kinematics_estimated.position.z_val + downward_speed
    # Move diagonally down with the new yaw
    client.moveByVelocityZAsync(vx=vx, vy=vy, z=z, duration=duration).join()
    #print(f"Moved {yaw_change} degrees to {'right' if yaw_change > 0 else 'left'} and down")

def move_45_degrees_right_down(client, duration, speed):
    move_diagonally_down_yaw(client, 45, duration, speed, 5)

def move_45_degrees_left_down(client, duration, speed):
    move_diagonally_down_yaw(client, -45, duration, speed, 5)

def move_diagonally_up_yaw(client, yaw_change, duration, forward_speed, upward_speed):
    # Rotate to the new yaw
    current_yaw = get_current_yaw(client)
    target_yaw = current_yaw + yaw_change
    client.rotateToYawAsync(target_yaw, timeout_sec=duration).join()
    # Calculate diagonal movement velocities
    vx = forward_speed * math.cos(math.radians(target_yaw))
    vy = forward_speed * math.sin(math.radians(target_yaw))
    z = client.getMultirotorState().kinematics_estimated.position.z_val - upward_speed
    # Move diagonally up with the new yaw
    client.moveByVelocityZAsync(vx=vx, vy=vy, z=z, duration=duration).join()
    #print(f"Moved {yaw_change} degrees to {'right' if yaw_change > 0 else 'left'} and up")

def move_45_degrees_right_up(client, duration, speed):
    move_diagonally_up_yaw(client, 45, duration, speed, 5)
```

```python
def step(self, action):
    if self.start_time is not None and (time.time() - self.start_time > self.time_limit or self.num_actions >= self.action_limit):
        return self.reset(), -100, True, {'timeout': True}  # Resetting with a timeout or action limit reached
    self.num_actions += 1

    # Store previous position for movement calculation
    previous_position = self.client.simGetVehiclePose().position

    # Action handling
    if action == 0:
        move_forward(self.client, 1, 2.5)  # Move forward
    elif action == 1:
        move_45_degrees_right_up(self.client, 2, 5)  # Move diagonally up to the right
    elif action == 2:
        move_45_degrees_left_up(self.client, 2, 5)  # Move diagonally up to the left
    elif action == 3:
        move_45_degrees_right_down(self.client, 2, 2)  # Move diagonally down to the right
    elif action == 4:
        move_45_degrees_left_down(self.client, 2, 2)  # Move diagonally down to the left
    elif action == 5:
        rotate_45_degrees_right(self.client, 1)  # Rotate 45 degrees to the right
    elif action == 6:
        rotate_45_degrees_left(self.client, 1)  # Rotate 45 degrees to the left
```

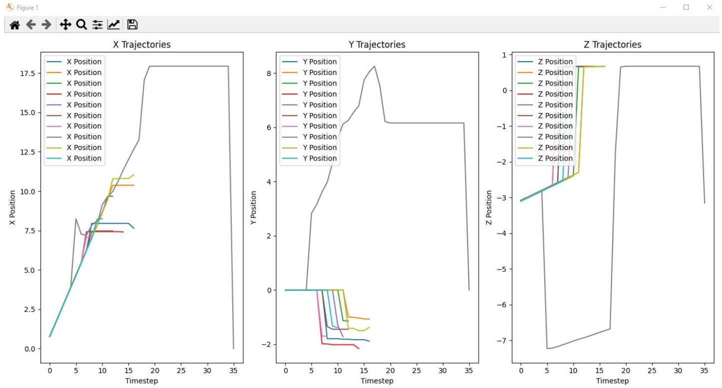
```python
def main():
    env = DroneEnv()
    model = DQN("CnnPolicy", env, verbose=1, buffer_size=10000, learning_starts=1000)

    # Start training
    model.learn(total_timesteps=250)

    # Log rewards and positions for evaluation
    reward_log = []
    all_positions = []
    for i in range(10):  # Example: 10 episodes
        obs = env.reset()
        done = False
        total_reward = 0
        positions = []  # Store positions for the current episode
        while not done:
            action, _states = model.predict(obs, deterministic=True)
            obs, reward, done, info = env.step(action)
            total_reward += reward
            pose = env.client.simGetVehiclePose().position
            positions.append((pose.x_val, pose.y_val, pose.z_val))  # Store the tuple of x, y, z positions
        reward_log.append(total_reward)
        all_positions.append(positions)
        print(f'Episode {i+1}: Total Reward: {total_reward}')

    # Save the model
    model.save("dqn_drone")

    # Evaluate and display results
    mean_reward, std_reward = evaluate_policy(model, model.get_env(), n_eval_episodes=10)
    print(f'Mean reward: {mean_reward}, Std reward: {std_reward}')

    # Plot the rewards and average trajectories
    plot_rewards(reward_log)
    plot_trajectories(all_positions)  # Plot trajectories for each episode
```

Demo 1

Demo 2

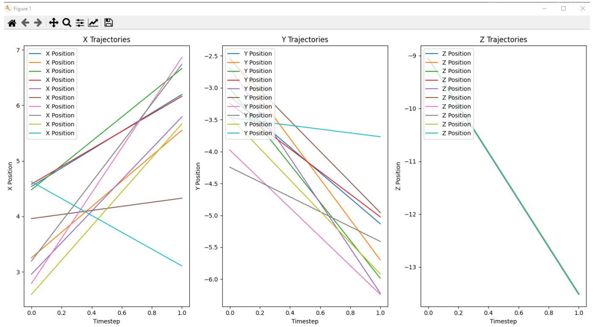
Demo 3

Results - 1000 timesteps



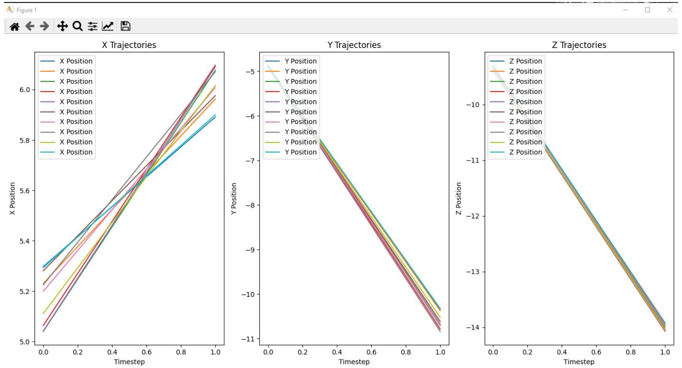
Results - 500 timesteps

```
---------------------------------
| rollout/            |         |
|    ep_len_mean      | 7.3     |
|    ep_rew_mean      | 31.6    |
|    exploration_rate | 0.05    |
| time/               |         |
|    episodes         | 56      |
|    fps              | 6       |
|    time_elapsed     | 65      |
|    total_timesteps  | 409     |
---------------------------------
---------------------------------
| rollout/            |         |
|    ep_len_mean      | 7.38    |
|    ep_rew_mean      | 34.1    |
|    exploration_rate | 0.05    |
| time/               |         |
|    episodes         | 60      |
|    fps              | 6       |
|    time_elapsed     | 70      |
|    total_timesteps  | 443     |
---------------------------------
---------------------------------
| rollout/            |         |
|    ep_len_mean      | 7.39    |
|    ep_rew_mean      | 31.3    |
|    exploration_rate | 0.05    |
| time/               |         |
|    episodes         | 64      |
|    fps              | 6       |
|    time_elapsed     | 75      |
|    total_timesteps  | 473     |
---------------------------------
Episode 1: Total Reward: -21.35452437400818
Episode 2: Total Reward: -19.624688863754272
Episode 3: Total Reward: -18.63892102241516
Episode 4: Total Reward: -7.090811252593994
Episode 5: Total Reward: -17.023419618606567
Episode 6: Total Reward: -19.189175128936768
Episode 7: Total Reward: -20.585936546325684
Episode 8: Total Reward: -7.0480241775512695
Episode 9: Total Reward: -17.162187814712524
Episode 10: Total Reward: -20.57037878036499
Mean reward: -12.989644000000002, Std reward: 3.688986061996413
```



Results - 100 timesteps

```
Wrapping the env with a `Monitor` wrapper
Wrapping the env in a DummyVecEnv.
Wrapping the env in a VecTransposeImage.
----------------------------------
| rollout/            |          |
|    ep_len_mean      | 11.2     |
|    ep_rew_mean      | 20.7     |
|    exploration_rate | 0.05     |
| time/               |          |
|    episodes         | 4        |
|    fps              | 2        |
|    time_elapsed     | 21       |
|    total_timesteps  | 45       |
----------------------------------
----------------------------------
| rollout/            |          |
|    ep_len_mean      | 12.5     |
|    ep_rew_mean      | 6.95     |
|    exploration_rate | 0.05     |
| time/               |          |
|    episodes         | 8        |
|    fps              | 2        |
|    time_elapsed     | 44       |
|    total_timesteps  | 100      |
----------------------------------
Episode 1: Total Reward: -3.0030083656311035
Episode 2: Total Reward: -3.578701972961426
Episode 3: Total Reward: -5.084336757659912
Episode 4: Total Reward: -4.90485954284668
Episode 5: Total Reward: -5.061425685882568
Episode 6: Total Reward: -3.144712448120117
Episode 7: Total Reward: -3.838348388671875
Episode 8: Total Reward: -3.633866786956787
Episode 9: Total Reward: -4.534327983856201
Episode 10: Total Reward: -3.0369057655334473
Mean reward: -3.4133483, Std reward: 0.896197729407975
```



Room for improvement

- Solve issue with drone going upwards initially
- Implement and use Bézier curves from research paper
- Implement PID algorithm for improved actions precision
- Determine proper use of NoDisplay setting and Overclock setting

Thank you