Implementation of Q learning and deep Q network for controlling a self balancing robot model

MD Muhaimin Rahman1* , S. M. Hasanur Rashid1 and M. M. Hossain2

ABSTRACT

본 논문에서는, 밸런싱 로봇의 Gazebo 모델에 관한 두 가지 강화 학습(Q-learning 과 Deep Q Network(DQN))의 구현에 대해 연구하였다. 이 실험의 목표는 로봇이 환경에서 균형을 유지하기 위한 best action 을 배우도록 하는 것이다. 정해진 한도 내에서 더 많은 시간을 유지할 수 있을수록 더 많은 보상을 축적하고, 따라서 더 균형을 잘 잡을 수 있다. 우리는 많은 hyperparameter 로 다양한 테스트를 했고, 이를 성능 곡선으로 보여주었다.

INTRODUCTION

제어 시스템은 로보틱스 연구 분야의 가장 중요한 요소 중 하나이다. Gazebo 는 현재 가장 강인한 multi-robot simulator 중 하나이다. Gazebo 와 함께 ROS 를 사용하면 더더욱 강력해진다. 그러나 컨트롤러 개발에 ROS 와 Gazebo 를 사용하는 방법에 대한 매뉴얼은 별로 없다. 이전 논문에서 우리는 ROS 와 Gazebo 를 활용한 PID, Fuzzy logic, LQR 컨트롤러의 사용을 밸런싱 로봇 모델에 구현하고 매뉴얼화를 시도했다. 추후에, 본 연구진은 강화 학습 연구에 임했다. 본 논문에서는 Q-learning 과 Deep Q Network(DQN)을 동일한 모델에 구현한다.

논문은 다음과 같이 구성된다. “Related Works” 섹션에서는 본 주제와 관련된 연구들을 서술한다. “Robot model” 섹션에서는 로봇 모델을 설명한다. Reinforcement learning methods as controllers” 섹션에서는 Q-learning 과 DQN 을 컨트롤러로서 구현하는 것을 설명한다. 마지막으로 “Conclusion and future work” 섹션에서 결론짓는다.

RELATED WORKS

Lei Tai and Ming Liu 는 CNN 기반 강화 학습을 이용한 모바일 로봇 탐사를 연구했다. 이들은 RGB-D 센서의 RAW 센서 값에 기반한 exploration strategy 을 개발하기 위해 Gazebo 에서 TurtleBot 을 학습하고 시뮬레이션 했다.

ErleRobotics 社는 OpenAI 환경을 Gazebo 로 확장했다. 이들은 다양한 exploration 환경에 Q-learning 과 Sarsa 알고리즘을 적용하였다.

Loc Tran et al. 은 static 한 장애물이 있는 Gazebo 와 실제 환경 모두에서 explore 하기 위한 무인 항공기의 훈련 모델을 개발했다. 그러나, 이들이 제안한 RL 은 논문에서 불명확하다.

Volodymyr Sereda 는 exploration strategy 에서 ROS 를 사용하는 맞춤형 Gazebo 모델에 Q-learning 을 사용했다.

Rowan Border 는 ROS 와 TurtleBot 을 이용한 로봇 탐색 구조를 위한 신경망 presentation 과 함께 Q-learning 을 사용했다.

ROBOT MODEL

로봇 모델은 [1]에 소개되어 있다. 하나의 샤시와 두 개의 바퀴로 구성된다. 모델의 task 는 로봇의 pitch angle 을 ± 5°로 유지하는 것이다. Angle 을 오래 유지할수록 더 많은 보상을 받는다. Fig.1 은 block diagram 을, Fig.2 는 밸런싱 로봇의 Gazebo 모델을 보여준다.

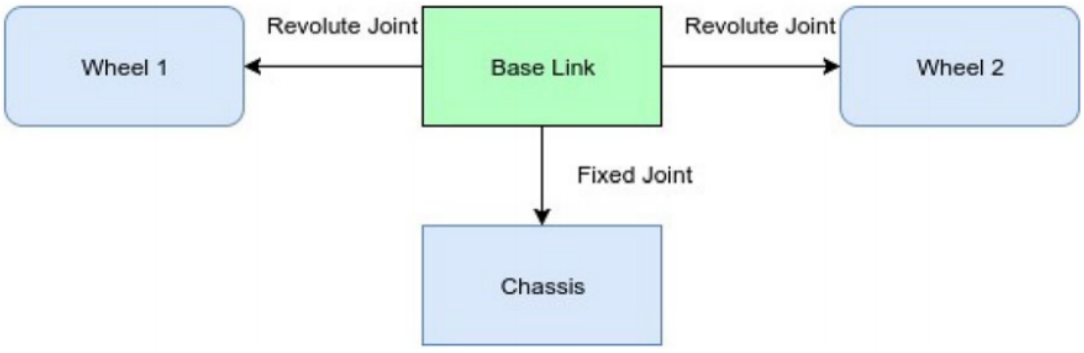

Fig. 1 Simple block diagram of the model

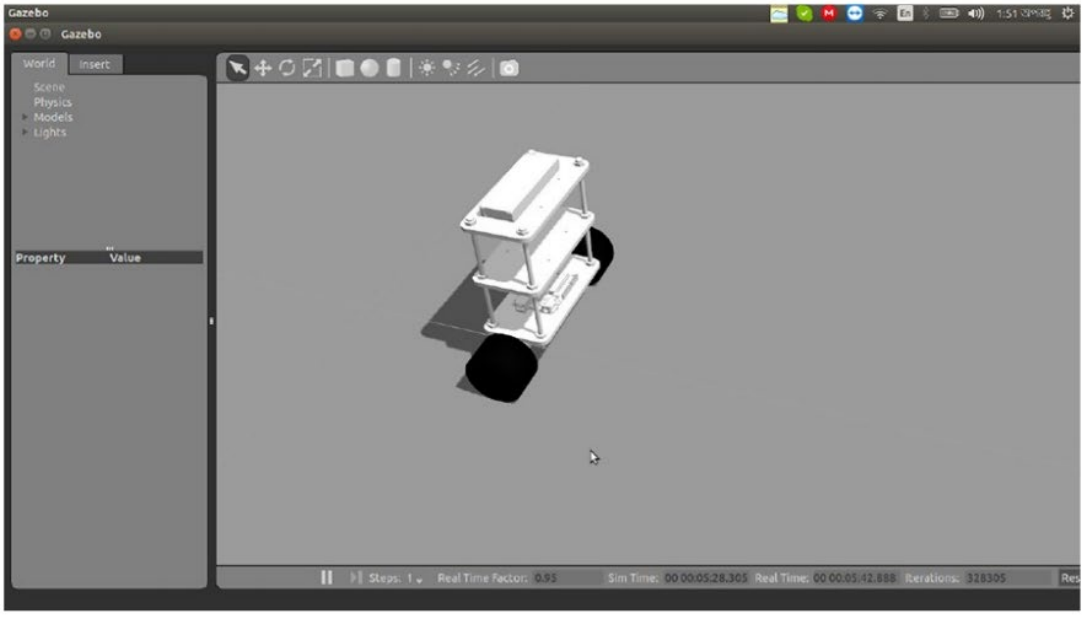

Fig. 2 Gazebo model

CONTROLLER

로봇의 IMU 센서는 매초마다 샤시의 roll, pitch, yaw angle 을 측정하여 컨트롤러로 전송한다. 그 다음, 컨트롤러는 설정된 값에 따라 샤시를 기울이도록 최적의 동작 값을 계산한다. Fig.3 은 로봇의 제어 시스템을 보여준다.

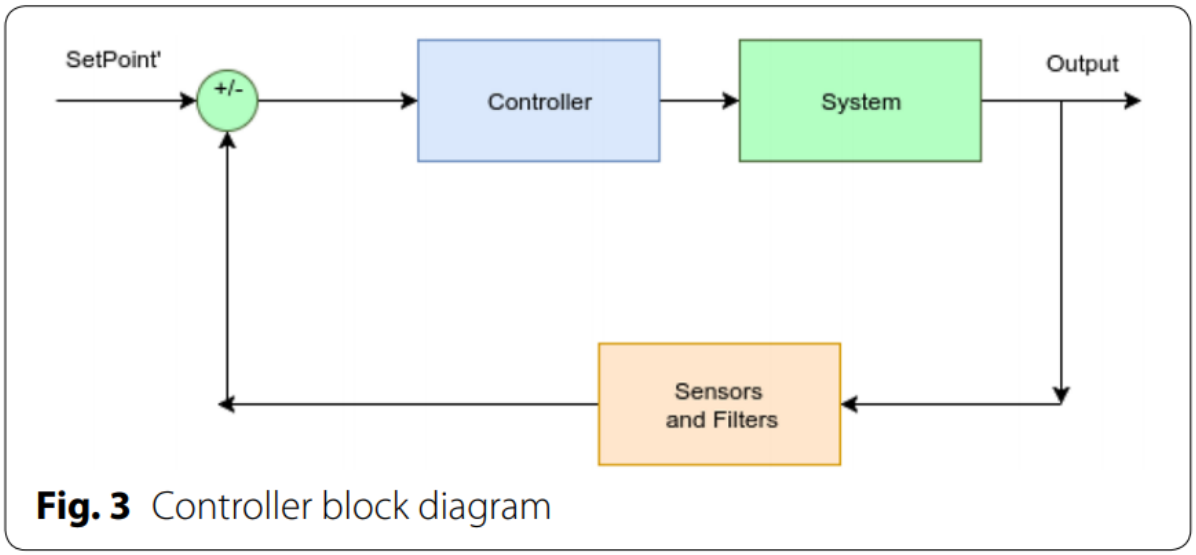

Fig. 3 Controller block diagram

REINFORCEMENT LEARNING METHODS AS CONTROLLERS

이전에는, PID, Fuzzy PID, PD+I & LQR 과 같은 기존 컨트롤러에 대해서 연구했다. 이 방법들의 가장 큰 문제는 수동으로 튜닝해야 한다는 것이다. 따라서 컨트롤러의 최적화된 값을 찾는 것은 많은 시도와 에러에 의존할 수밖에 없다. 대부분의 경우, 최적화된 값은 전혀 도달할 수 없다. RL 이 컨트롤러로서 갖는 가장 큰 장점은 모델이 스스로 튜닝하여 최적의 값에 도달한다는 것이다.

다음 두 섹션에서는 Q-learning 과 DQN 에 대해 논한다.

Q LEARNING

Q-learning 은 Christopher John Cornish Hellaby Watkins 에 의해 고안되었다. Watkins 에 따르면, "이것은 agent 에 domain map 을 만들 필요 없이, action 의 결과를 경험함으로써 Markovian domain 에서 최적으로 행동하는 법을 배울 수 있는 능력을 제공한다."라고 하였다. Markovian domain 에서, Q 함수 -알고리즘을 사용하여 생성된 모델- 는 주어진 finite state s 와 가능한 모든 finite action a 에 대한 기대 유틸리티를 계산한다. 이 경우 agent-여기서는 로봇-는 Q(s,a)의 값이 가장 큰 최적의 action 을 선택하며, 이 action 선택 규칙을 Policy 라고 부른다. 처음 Q(s,a) 값은 0 으로 가정한다. 모든 훈련 단계가 끝나면, 값은 아래의 식에 따라 업데이트된다.

$$Q(s, a_t) \leftarrow Q(s, a_t) + \alpha(r + \gamma \, max \, Q(s_{t+1}, a)) \qquad (1)$$

ALGORITHM

본 논문에서 모델의 목적은 제한 각도(± 5°)를 유지하는 것이다. 우선 로봇 모델, Q matrix, policy π를 초기화한다. 몇 가지 흥미로운 점들이 있는데, state 는 유한하지 않다. 제한 범위 내에서, 수백, 수천 개의 pitch angle 이 가능하다. 수천 개의 column 을 갖는 건 가능하지 않다. 따라서 본 연구에서는 state 값을 -10°에서 10°까지 20 개의 state angle 로 분류했다. action 값은 [-200, 200]ms^{-1} 에서 10 개의 다른 속도 값을 선택했다. Q matrix 에는 20 개의 column 이 있으며, 각 column 은 state 를 나타내고 10 개의 row 는 각각 모든 action 을 나타낸다. 처음 Q 값은 0 으로 가정되었고, policy π의 모든 state 에 대해 일부 무작위 action 이 지정되었다. 2000 회 반복되는 1500 가지의 에피소드를 훈련했다. 각 에피소드의 시작마다 시뮬레이션을 refresh 했다. 로봇의 상태가 제한을 벗어날 때마다 리워드로 -100 의 패널티를 주었다. Q Table 은 각 단계에서 Eq.(1)에 따라 업데이트된다. Algorithm 1 은 전체 알고리즘을 보여준다.

Algorithm 1 시스템에 적용된 Q Learning 알고리즘

로봇 초기화;
Q Matrix Q 초기화;
Policy π 초기화;
패널티 Reward pen 초기화;
for episodes 수 **do**
    시뮬레이션 리셋;
    1초 대기;
    시뮬레이션 일시정지;
    로봇의 pitch angle Π 읽기;
    $state \leftarrow \Pi$;
    시뮬레이션 재생;
    for iterations 수 **do**
        랜덤 수 $rand$ 생성;
        if $rand \leq \epsilon$ **then**
            랜덤 액션 시행;
        else
            π에 기반한 액션 시행;
        end if
        $state_{new} \leftarrow \phi$;
        시뮬레이션 일시중지;
        if $|state_{new}| \geq limit$ **then**
            if $reward_{total} \leq Target$ **then**
                $reward \leftarrow pen$
                Q 업데이트;
                π 업데이트;
                Break;
            else
                Passed 출력;
                Break;
            end if
        else
            $reward \leftarrow 1$;
            Q 업데이트;
            π $state \leftarrow state_{new}$ 업데이트;
        end if
    end for
end for

RESULT AND DISCUSSION

시뮬레이션은 $\alpha(0.7, 0.65, 0.8), \gamma(0.999)$ 값으로 시행되었다. Fig.4 는 reward vs α에 대한 episode 를 보여준다. 로봇이 이 learning rate 에 대해 학습 기간 내에 목표한 만큼의 reward 를 얻을 수 없었다는 것은, 자명하다. α 0.7 과 0.8 값에 대해, 로봇은 400 회 episode 이내에 최대로 가능한 누적 reward 값인 2000 에 도달했다는 것을 볼 수 있다. $\alpha = 0.7$인 곡선은 $\alpha = 0.8$에 비해 덜 안정적이다. 그러나, $\alpha = 0.65$인 곡선은 최대 누적 reward 를 달성하지 못하였다.

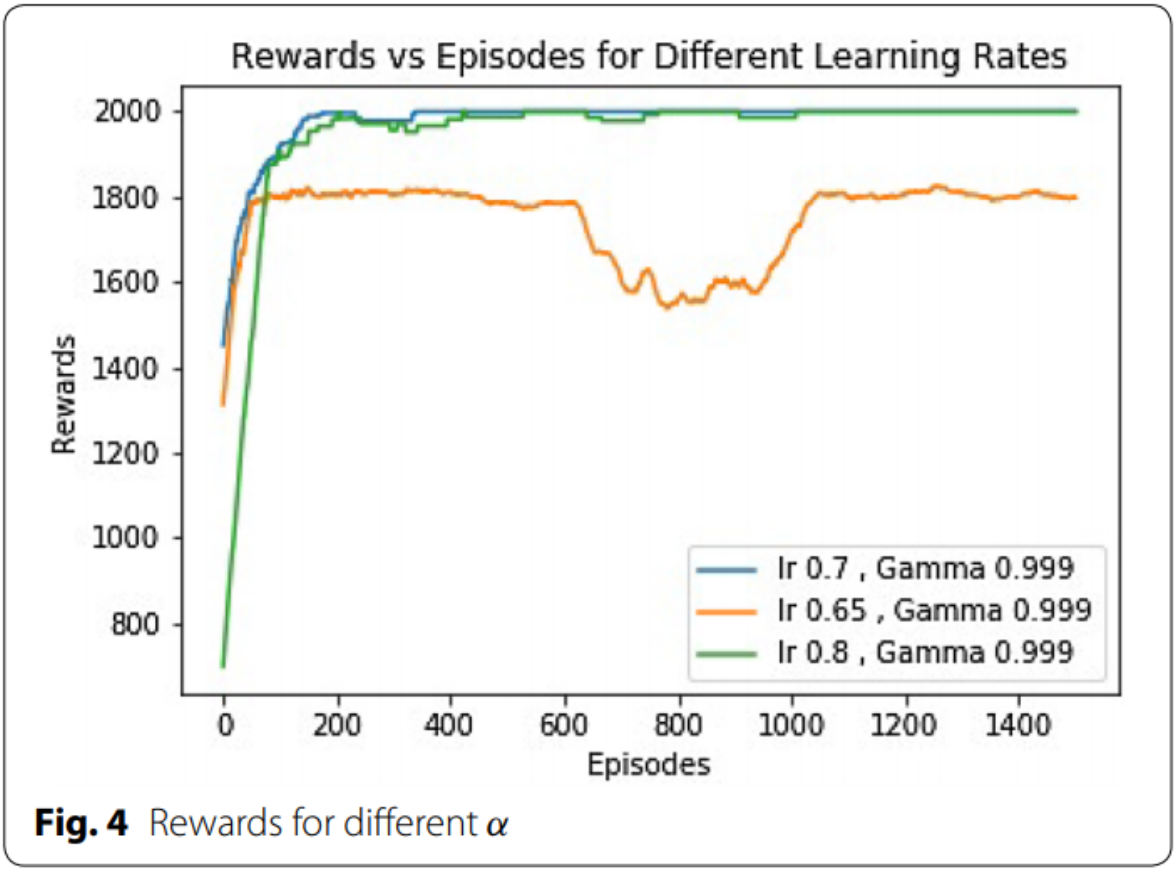

Fig. 4 Rewards for different α