ECE500
Fundamentals of Autonomous Robots
Graduate Student Study

LEARNING APPLICATION ON REAL WORLD ROBOTS

Esin SAKA
University of Louisville
e0saka01@louisville.edu
1 REINFORCEMENT LEARNING

1.1 Reinforcement Learning
Reinforcement learning algorithms is a family of machine learning algorithms. They optimize sequential decision making process. Reinforcement approach uses reinforcement and scalar evaluations. The idea is maximizing the expected sum of future reinforcements for each state.

1.2 Reinforcement Learning Formulization
For maximizing the expected sum of future reinforcements the following formula is used.

\[ Q(s_t, a) = \sum_{i=0}^{\infty} \gamma^i R(s_{t+i+1}) \]  

(1)

where, \( R(s) \) is Reward at state \( s \), \( s_t \) is state the agents is in at time \( t \) and \( \gamma \) is discount factor \( 0 \leq \gamma < 1 \)

1.3 Reinforcement Learning Formulization Approximation
But it is hard to compute the above formula because of unknown values. Therefore an approximation formula is used, which is given as:

\[ Q_{n+1}(s, a) = R(s') + \gamma \max_{a' \in \text{actions in } s'} [Q_n(s', a')] \]  

(2)

where, \( R(s) \) is the reward at state \( s \), \( s' \) is state resulting from performing action \( a \) in state \( s \) and \( \gamma \) is the discount factor such that \( 0 \leq \gamma < 1 \)

1.4 Reinforcement Learning Examples
There are various methods for reinforcement learning. Two examples can be given as:

- Q-learning.
- Sarsa.

2 Learning Experiment
In the graduate study a learning algorithm on real world robots was implemented.
2.1 Job Definition

Job learned is drawing a circle on clockwise direction. Robot starts without knowing anything. By performing random moves, it learns to turn in clockwise direction.

2.2 Solution Approach

There are 4 different moves:

- forward
- backward
- left
- right

A simple q-learning approach is used. States and moves are combined, such that, performing a move means being in that move’s state.

A q-table of size 4 is kept. Initially all values are zero. As time passes, q-values are updated according to reward function.

Initially robot makes random moves. As the time passes, and the robot makes a correct move, it gets a reward. A reward is an increase in the q-value of correct move. In time, a right move becomes more effective than the other moves.

To select a move at a given time, robot has two options:

- Do a random move, which may be forward, backward, left and right.
- Do the most effective move.

At the beginning of learning procedure, the probability of random move is higher than the probability of performing the move with highest q-value. As the time passes, the probability of random move decreases, and after a while, robot stops learning and continues with performing only the best move. At this step, robot is called ”learned to move clockwise”.

3 Conclusion

Just in a short time, the robot learns to turn in clockwise direction. The learning application is successful.
4 Appendix

The code used is given below:

/* ECE500 Robotics Graduate Student Study
 * ****************************************************
 * Application of Q-Learning
 * ****************************************************
 * press start button to start learning process
 * ****************************************************
 * prepared by esin : e0saka01 at louisville dot edu dot
 * *****************************************************/

#define F 0
#define B 1
#define L 2
#define R 3
#define time 1000L
#define MAX_LEARN_STEP 25
#define MAX_REWARD 1000
#define POWER 50

/* motor and sensor ports */
int LEFT_MOTOR= 0;
int RIGHT_MOTOR= 3;
int GATE= 1;
int GATETOUCH= 13;
int IR= 2;

int RUN= 0;

int q[4];
int random_prob;

/******************************/
/* timer.c */
float _timer;

void reset_timer() {
    _timer= seconds();
}
```c
float timer() {
    return seconds() - _timer;
}

void initialize() {
    int i;

    for(i=0; i<4; i++) {
        q[i] = 0;
    }

    random_prob = MAX_LEARN_STEP;
    while(1) {
        if(start_button()) {
            RUN = 1;
            reset_timer();
            break;
        }
    }
}

void forward() {
    motor(LEFT_MOTOR, POWER);
    motor(RIGHT_MOTOR, POWER);
    msleep(time);
}

void backward() {
    motor(LEFT_MOTOR, -POWER);
    motor(RIGHT_MOTOR, -POWER);
    msleep(time);
}

void left() {
    motor(LEFT_MOTOR, 0);
    motor(RIGHT_MOTOR, POWER);
    msleep(time);
}
```
void right(){
    motor(LEFT_MOTOR, POWER);
    motor(RIGHT_MOTOR, 0);
    msleep(time);
}

int find_best_move(){
    int i, best;
    best=0;
    for(i=0; i<4; i++){
        if(q[i]>q[best]){
            best=i;
        }
    }
    return best;
}

int choose_move(){
    int m;
    int best_move;
    int random_move;
    int prob;
    random_move=random(4);
    best_move=find_best_move();
    prob=random(MAX_LEARN_STEP);
    if(prob>random_prob){
        m=best_move;
    }else{
        m=random_move;
    }
    return m;
}

void move(int m){
}
if (m == F) {
    printf("forward %d\n", random_prob);
    forward();
    msleep(time);
}
else if (m == B) {
    printf("backward %d\n", random_prob);
    backward();
    msleep(time);
} else if (m == L) {
    printf("left %d\n", random_prob);
    left();
    msleep(time);
} else if (m == R) {
    printf("right %d\n", random_prob);
    right();
    msleep(time);
} else {
    printf("INVALID MOVE %d\n", random_prob);
    msleep(time);
}

void reward_move(int m) {
    if (m == R) {
        if (q[R] < MAX_REWARD) {
            q[R]++;
        }
    }
}

void main() {
    int m;
    initialize();

    if (RUN) {
        while (1) {
            m = choose_move();
            move(m);
            if (random_prob) { // still learning
                reward_move(m);
            }
        }
    }
}
random_prob--; 
reward_move(m);
}
} 
} 

}