BAB II
LANDASAN TEORI

Berdasarkan latar belakang masalah yang telah diuraikan pada bab 1, penulis akan meneliti implementasi algoritma *Q-learning* untuk perencanaan jalur *mobile robot*. *Q-learning* merupakan salah satu algoritma dalam domain *Reinforcement Learning*.

2.1 Perencanaan Jalur

Perencanaan jalur atau rute adalah masalah komputasi yang merupakan keterampilan penting dan esensial di bidang robotika. Tujuan utama dari perencanaan jalur adalah untuk menemukan rute terbaik antara awal dan titik akhir. Umumnya, rute terbaik adalah rute terpendek antara dua titik yang berbeda (awal dan tujuan) [6]. Perencanaan jalur *mobile robot* dapat dibagi menjadi tiga kategori berdasarkan pengetahuan robot yang dimilikinya tentang lingkungan, lingkungan, dan pendekatan yang digunakan untuk memecahkan masalah [1].

A. Lingkungan

Masalah perencanaan jalur dapat dilakukan di lingkungan statis dan dinamis. Lingkungan statis tidak berubah, posisi awal dan tujuan tetap, dan lokasi rintangan tidak berubah dari waktu ke waktu. Dalam lingkungan dinamis, lokasi rintangan dan posisi tujuan dapat bervariasi sselama proses pencarian.

B. Pengetahuan Peta

Menurut pengetahuan robot tentang lingkungan, perencanaan jalur dapat dibagi menjadi dua kelas: Pada kelas pertama, robot memiliki pengetahuan awal tentang lingkungan yang dimodelkan sebagai peta. Kategori perencanaan jalur ini dikenal sebagai perencanaan perencanaan jalur global (*global path planning*) atau perencanaan jalur deliberatif (*deliberative path planning*). Kelas perencanaan jalur kedua mengasumsikan bahwa robot tidak memiliki pengetahuan informasi sebelumnya tentang lingkunganya (yaitu, lingkungan tidak pasti). Akibatnya, ia harus merasakan/mendeteksi lokasi rintangan dan membangun perkiraan peta lingkungan secara *real time* selama proses pencarian untuk menghindari

rintangan dan memperoleh jalur yang sesuai menuju tujuan. Jenis perencanaan jalur ini dikenal sebagai perencanaan jalur lokal (*local path planning*) atau navigasi reaktif (*reactive navigation*).

C. Kelengkapan (*Completeness*)

Tergantung pada kelengkapannya, algoritma perencanaan jalur diklasifikasikan sebagai eksak atau heuristik. Algoritma eksak menemukan solusi optimal jika ada atau membuktikan bahwa tidak ada solusi yang layak. Algoritma heuristik mencari solusi berkualitas baik dalam waktu yang lebih singkat.

Perencanaan jalur robot juga dapat diklasifikasikan menurut jumlah robot yang ada di ruang kerja yang sama untuk melakukan misi yang sama. Dalam banyak aplikasi, beberapa *mobile robot* bekerjasama dalam lingkungan yang sama. Masalah ini disebut perencanaan jalur multi-robot.

Pada penelitian yang akan dilakukan, penulis akan meneliti masalah perencanaan jalur lokal dalam lingkungan statis dengan menggunakan algoritma *Q-learning*.

2.2 Mobile Robot

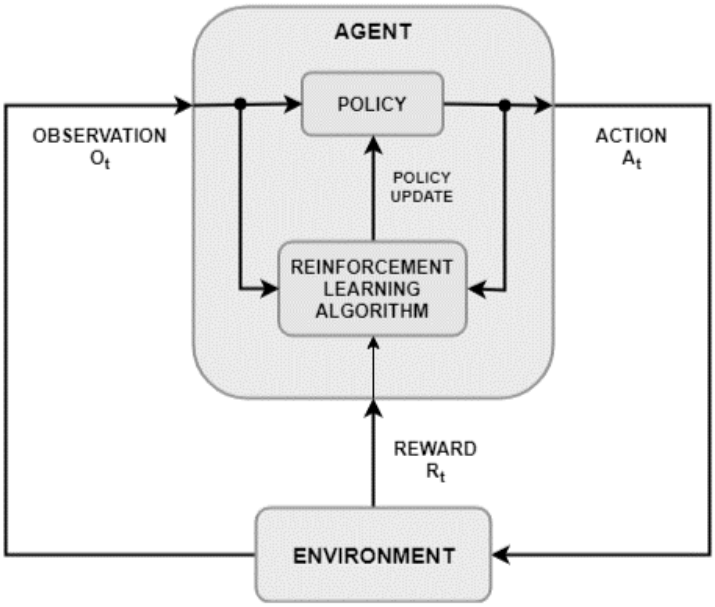
Mobile robot (robot mobil) adalah robot yang dapat berpindah dari satu tempat ke tempat lain secara mandiri, yaitu tanpa bantuan operator manusia dari luar. Tidak seperti kebanyakan robot industri yang hanya dapat bergerak di ruang kerja tertentu, *mobile robot* memiliki fitur khusus untuk bergerak bebas dalam ruang kerja yang telah ditentukan untuk mencapai tujuan yang diinginkan. Kemampuan mobilitas ini membuatnya cocok untuk perbendaharaan besar aplikasi di lingkungan terstruktur dan tidak terstruktur. *Mobile robot* darat dibedakan dalam *wheeled mobile robots* (WMRS / robot beroda) dan *legged mobile robots* (LMRS / robot berkaki), *mobile robot* juga termasuk *unmanned aerial vehicles* (UAV), dan *autonomous underwater vehicles* (AUV). Robot beroda sangat populer karena sesuai untuk aplikasi tipikal dengan kompleksitas mekanik dan konsumsi energi yang relatif rendah. Robot berkaki cocok untuk tugas-tugas di lingkungan yang tidak standar, tangga, tumpukan puing, dll. Biasanya, sistem dengan dua, tiga, empat, atau enam kaki menarik secara umum tetapi banyak kemungkinan lain juga

ada. Robot berkaki tunggal menemukan aplikasi langka karena mereka hanya dapat bergerak dengan melompat. *Mobile robot* juga mencakup *mobile manipulators* (robot beroda atau berkaki yang dilengkapi dengan satu atau lebih manipulator ringan untuk melakukan berbagai tugas) [7].

2.3 Reinforcement Learning

Reinforcement learning (RL) adalah pendekatan komputasi yang terarah pada tujuan (*goal-directed*) di mana komputer belajar untuk melakukan tugas dengan berinteraksi dengan lingkungan dinamis yang tidak diketahui. Pendekatan pembelajaran ini memungkinkan komputer untuk membuat serangkaian keputusan untuk memaksimalkan imbalan kumulatif (*cumulative reward*) untuk tugas tersebut tanpa campur tangan manusia dan tanpa diprogram secara eksplisit untuk mencapai tugas tersebut [8].

Reinforcement Learning adalah mempelajari apa yang harus dilakukan—bagaimana memetakan situasi (*state*) ke dalam aksi (*action*)—sehingga memaksimalkan sinyal imbalan (*reward signal*) numerik. Pembelajar tidak diberitahu aksi mana yang harus diambil tetapi sebaliknya harus menemukan aksi mana yang menghasilkan imbalan paling banyak dengan mencobanya. Dalam kasus yang paling menarik dan menantang, aksi dapat mempengaruhi tidak hanya imbalan langsung (*immediate reward*) tetapi juga *state* berikutnya dan, melalui itu, semua imbalan berikutnya. Kedua karakteristik ini—pencarian coba-coba (*trial-and-error*) dan imbalan tertunda (*delayed reward*)—adalah dua ciri pembeda terpenting dari RL [9]. Diagram berikut menunjukkan representasi umum dari skenario RL [8].



Gambar 2.1 Skenario *Reinforcement Learning*

Model *Reinforcement learning* belajar dari sebuah lingkungan (*environment*). Lingkungan memiliki serangkaian aturan dan biasanya diasumsikan deterministik. Agen (*agent*) memiliki *state* di dalam lingkungan, dan agen bisa melakukan aksi (*actions*) yang mengubah *state* tersebut di dalam lingkungan. Ketika agen melakukan aksi, lingkungan akan menerimanya sebagai input dan akan mengeluarkan *state* yang dihasilkan dan sebuah imbalan (*reward*).

Tujuan RL adalah untuk melatih agen untuk menyelesaikan tugas dalam lingkungan yang tidak diketahui. Agen menerima observasi dan imbalan dari lingkungan dan mengirimkan aksi ke lingkungan. Imbalan adalah ukuran seberapa sukses tindakan sehubungan dengan menyelesaikan tujuan tugas. Dengan kata lain, RL melibatkan agen yang mempelajari perilaku optimal melalui interaksi coba-coba (*trial-and-error*) berulang dengan lingkungan tanpa keterlibatan manusia [8].

Untuk menentukan aksi terbaik, RL bekerja dengan memperkirakan nilai (*value*) dari aksi yang dinotasikan dengan *Q(S, A)*. Nilai dari aksi menandakan seberapa bagus sebuah aksi, misal seberapa bagus sebuah gerakan robot. Semua RL berkisar pada gagasan ini, memperkirakan *value function* optimal.

Masalah RL dapat diformalkan menggunakan ide-ide dari teori sistem dinamis, khususnya, sebagai kontrol optimal dari proses keputusan Markov (*Markov decision* process / MDP) yang tidak diketahui secara lengkap (*incompleteley-known*). Ide dasarnya hanyalah untuk menangkap aspek terpenting dari masalah nyata yang dihadapi agen (*agent*) pembelajaran yang berinteraksi dari waktu ke waktu dengan lingkungannya (*environment*) untuk mencapai tujuan (*goal*). Sebuah agen pembelajaran harus mampu merasakan *state* lingkungannya sampai batas tertentu dan harus mampu mengambil aksi yang mempengaruhi *state* tersebut. Agen juga harus memiliki tujuan atau goal yang berkaitan dengan *state* lingkungannya. MDP dimaksudkan untuk memasukkan hanya tiga aspek ini—sensasi (*sensation*), aksi (*action*), dan tujuan (*goal*)—dalam bentuknya yang paling sederhana tanpa meremehkan salah satu dari mereka. Metode apa pun yang cocok untuk memecahkan masalah seperti itu bisa anggap sebagai metode RL [9].

Reinforcement learning adalah salah satu dari tiga kategori besar *Machine Learning*. Penelitian ini tidak berfokus pada *unsupervised learning* (pembelajaran tanpa pengawasan) atau *supervised learning* (pembelajaran yang diawasi), tetapi perlu dipahami bagaimana RL berbeda dari keduanya. *Unsupervised learning* digunakan untuk menemukan pola atau struktur tersembunyi dalam kumpulan data yang belum dikategorikan atau diberi label. Sebagai contoh, anda memiliki informasi tentang atribut fisik dan kecenderungan sosial 100.000 hewan. Anda dapat menggunakan *unsupervised learning* untuk mengelompokkan hewan atau mengelompokkannya ke dalam fitur yang serupa. Kelompok-kelompok ini dapat didasarkan pada jumlah kaki, atau berdasarkan pola yang mungkin tidak begitu jelas, seperti korelasi antara ciri-ciri fisik dan perilaku sosial yang tidak anda ketahui sebelumnya [10].

Menggunakan *supervised learning*, anda melatih komputer untuk menerapkan label ke input yang diberikan. Misalnya, jika salah satu kolom kumpulan data fitur hewan adalah spesies, anda dapat memperlakukan spesies sebagai label dan data lainnya sebagai input ke dalam model matematika. Anda dapat menggunakan *supervised learning* untuk melatih model agar memberi label

yang benar pada setiap kumpulan fitur hewan di kumpulan data. Model menebak spesies, dan kemudian algoritma *machine learning* secara sistematis mengubah model tersebut. Dengan data pelatihan yang cukup untuk mendapatkan model yang andal, anda kemudian dapat memasukkan fitur untuk hewan baru yang tidak berlabel, dan model terlatih akan menerapkan label spesies yang paling mungkin untuknya [10].

Reinforcement learning sangatlah berbeda, tidak seperti dua *framework* pembelajaran lainnya, yang beroperasi menggunakan kumpulan data statis, RL bekerja dengan data dari lingkungan dinamis. Dan tujuannya bukan untuk mengelompokkan data atau melabeli data, tetapi untuk menemukan urutan aksi terbaik yang akan menghasilkan hasil yang optimal. Cara RL memecahkan masalah ini adalah dengan membiarkan perangkat lunak yang disebut agen (*agent*) untuk mengeksplorasi, berinteraksi dengan lingkungan (*environment*), dan belajar dari lingkungan [10].

2.3.1 Elemen-elemen dari *Reinforcement Learning*

Terdapat beberapa elemen utama dari sistem *reinforcement learning*, diantaranya:

A. *Environment* (Lingkungan)

Environment adalah segala sesuatu yang ada di luar agen, termasuk dinamika sistem, di situlah agen mengirimkan aksi, dan itulah yang menghasilkan imbalan dan observasi. Oleh karena itu, sebagian besar sistem sebenarnya adalah bagian dari lingkungan. Agen hanyalah bagian kecil dari perangkat lunak yang menghasilkan aksi dan memperbarui *policy* (aturan) melalui pembelajaran.

B. *Policy* (Aturan)

Policy dari algoritma adalah bagaimana algoritma tersebut memilih aksi mana yang harus dilakukan berdasarkan nilai (*value*) dari *state* saat ini. *Policy* adalah pemetaan yang memilih aksi berdasarkan observasi dari lingkungan. *Policy* mendefinisikan cara agen berperilaku pada waktu tertentu. Secara kasar, *policy* adalah pemetaan dari *state* lingkungan yang dirasakan ke

aksi yang harus diambil ketika berada di situasi tersebut. Dalam beberapa kasus, *policy* mungkin merupakan fungsi sederhana atau tabel pencarian, sedangkan di kasus lain mungkin melibatkan perhitungan ekstensif seperti proses pencarian. *Policy* merupakan inti dari agen RL dalam arti bahwa *policy* itu sendiri cukup untuk menentukan perilaku [9]. Biasanya, *policy* adalah *function approximator* dengan parameter yang dapat disesuaikan, seperti jaringan saraf dalam (*deep neural network*).

C. *Learning Algorithm* (Algoritma Pembelajaran)

Algoritma pembelajaran terus memperbarui parameter *policy* berdasarkan aksi, observasi, dan *reward*. Tujuan dari algoritma pembelajaran adalah untuk menemukan *policy* optimal yang memaksimalkan *reward* kumulatif yang diterima selama tugas.

D. *Reward signal* (Sinyal imbalan)

Reward signal atau *reward* adalah fungsi yang menghasilkan angka skalar yang mewakili "kebaikan" sebuah agen berada dalam *state* tertentu dan mengambil aksi tertentu [10]. *Reward* mendefinisikan tujuan dari masalah RL. Pada setiap langkah waktu, lingkungan mengirimkan ke agen RL sebuah angka yang disebut *reward*. Satu-satunya tujuan agen adalah untuk memaksimalkan total *reward* yang diterimanya dalam jangka panjang. *Reward signal* dengan demikian menentukan peristiwa baik dan buruk apa yang terjadi pada agen. *Reward signal* adalah dasar utama untuk mengubah *policy*; jika suatu aksi yang dipilih oleh *policy* diikuti dengan *reward* yang rendah, maka *policy* tersebut dapat diubah untuk memilih beberapa aksi lain dalam *state* itu di masa depan. Secara umum, *reward* mungkin merupakan fungsi stokastik dari *state* lingkungan dan aksi yang diambil [9].

E. *Value function*

Jika *reward* menunjukkan apa yang baik dalam arti langsung, *value function* menentukan apa yang baik dalam jangka panjang. Secara kasar, *value* dari sebuah *state* adalah jumlah total dari *reward* yang dapat diharapkan agen untuk dikumpulkan di masa depan, mulai dari *state* tersebut. Sementara

reward menentukan keinginan intrinsik langsung dari *state* lingkungan, *value* menunjukkan keinginan jangka panjang dari *state* setelah memperhitungkan state yang mungkin mengikuti dan *reward* yang tersedia di *state* tersebut. Misalnya, suatu *state* mungkin selalu menghasilkan *reward* langsung yang rendah tetapi masih memiliki *value* tinggi karena secara teratur diikuti oleh *state-state* lain yang menghasilkan *reward* tinggi. Atau sebaliknya bisa jadi benar.

Value dari sebuah *action* didefinisikan sebagai jumlah dari *immediate reward* yang diterima dengan melakukan sebuah *action* ditambah *expected value* dari *state* yang dihasilkan dikalikan dengan istilah penskalaan (*scaling term*). Dengan kata lain, *value* dari sebuah *action* adalah seberapa bagus *state* selanjutnya setelah melakukan *action* tersebut, ditambah *expected reward* dari *state* baru tersebut.

Model RL memperbarui *value function*-nya dengan berinteraksi dengan *environment*, memilih sebuah *action*, melihat *state* baru, melihat *reward* lalu memperbarui.

F. *Environment model*

Environment model adalah sesuatu yang meniru perilaku lingkungan, atau lebih umum, yang memungkinkan kesimpulan dibuat tentang bagaimana lingkungan akan berperilaku. Misalnya, dengan *state* dan aksi tertentu, model dapat memprediksi *state* berikutnya yang dihasilkan dan *reward* berikutnya. Model digunakan untuk perencanaan (*planning*), yang dimaksudkan dengan cara apa pun untuk memutuskan aksi dengan mempertimbangkan kemungkinan *state* masa depan sebelum benar-benar dialami. Metode untuk memecahkan masalah RL yang menggunakan *model* dan *planning* disebut metode berbasis model (*model-based*), sebagai lawan dari metode bebas model (*model-free*) yang lebih sederhana yang secara eksplisit merupakan pembelajar coba-coba—dipandang sebagai kebalikan dari *planning*.

2.3.2 Explorasi dan Exploitasi

Satu aspek penting dari *reinforcement learning* adalah pertukaran (*trade-off*) antara eksplorasi (*exploration*) dan eksploitasi (*exploitation*) saat agen berinteraksi dengan lingkungan. Alasan keputusan ini muncul dalam RL adalah karena pembelajaran dilakukan secara *online*. Alih-alih bekerja dari kumpulan data statis, aksi agen menentukan data mana yang dikembalikan dari lingkungan. Pilihan yang dibuat agen menentukan informasi yang diterimanya dan, oleh karena itu, informasi yang dapat dipelajarinya [10].

Masalahnya adalah: haruskah agen mengeksploitasi lingkungan dengan memilih aksi yang mengumpulkan *reward* paling banyak yang sudah diketahuinya, atau haruskah ia memilih aksi yang mengeksplorasi bagian lingkungan yang masih belum diketahui?

Misalnya, katakanlah agen berada dalam *state* tertentu dan dapat mengambil salah satu dari dua aksi: ke kiri atau ke kanan. Ia tahu bahwa ke kiri akan menghasilkan *reward* +1 dan ke kanan menghasilkan *reward* -1. Agen tidak tahu apa-apa lagi tentang lingkungan di sebelah kanan dari *state* awal dengan *reward* rendah itu. Jika agen mengambil pendekatan serakah (*greedy*) dengan selalu mengeksploitasi lingkungan, ia akan pergi ke kiri untuk mengumpulkan *reward* tertinggi yang diketahuinya dan mengabaikan *state* lain sepenuhnya [10].

Dapat dilihat bahwa jika agen selalu mengeksploitasi apa yang dianggapnya sebagai aksi terbaik pada waktu tertentu, agen tersebut mungkin tidak akan pernah menerima informasi tambahan tentang *state* yang ada di balik *aksi* dengan *reward* rendah. Eksploitasi murni ini dapat meningkatkan jumlah waktu yang diperlukan untuk menemukan *policy* yang optimal atau dapat menyebabkan algoritma pembelajaran konvergen pada *policy* suboptimal karena seluruh bagian dari ruang *state* (*state space*) mungkin tidak pernah dieksplorasi [10].

Sebaliknya, jika sesekali membiarkan agen mengeksplorasi, bahkan dengan risiko mengumpulkan lebih sedikit *reward*, agen tersebut dapat memperluas *policy*-nya untuk *state* baru. Ini membuka kemungkinan menemukan *reward* yang lebih tinggi yang tidak diketahuinya dan meningkatkan peluang untuk mencapai solusi

global. Tetapi anda tidak ingin agen terlalu banyak mengeksplorasi karena ada kerugian dengan pendekatan ini juga. Pertama, eksplorasi murni bukanlah pendekatan yang baik saat melatih perangkat keras fisik karena agen berisiko mengeksplorasi tindakan yang menyebabkan kerusakan. Pikirkan tentang kerusakan yang dapat disebabkan oleh mobil otonom yang mengeksplorasi input roda kemudi acak saat berada di jalan raya [10].

Namun, bahkan dengan lingkungan simulasi di mana kerusakan tidak menjadi masalah, eksplorasi murni bukanlah cara yang efisien untuk belajar karena agen kemungkinan akan menghabiskan waktu untuk mengeksplorasi sebagian besar ruang *state*. Meskipun ini bermanfaat untuk menemukan solusi global, eksplorasi yang berlebihan dapat memperlambat laju pembelajaran sedemikian rupa sehingga tidak ada solusi yang cukup ditemukan dalam jumlah waktu pembelajaran yang wajar. Oleh karena itu, algoritma pembelajaran terbaik menyeimbangkan antara mengeksplorasi dan mengeksploitasi lingkungan [10].

2.3.3 Markov Decision Process (MDP)

MDP (*Markov Desicion Process*) adalah sebuah proses pengambilan-keputusan yang memungkinkan kita untuk merepresentasikan sebuah *environment*, sebagian besar masalah *reinforcement learning* dapat diformalkan atau dirumuskan sebagai MDP.

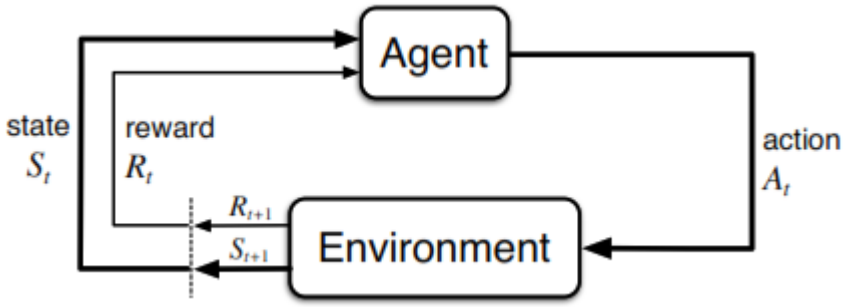
MDP adalah formalisasi klasik dari pengambilan keputusan sekuensial, di mana aksi memengaruhi tidak hanya reward langsung, tetapi juga *state* atau *state* berikutnya, dan melalui *reward* di masa depan itu. Jadi MDP melibatkan *reward* yang tertunda dan kebutuhan untuk menukar *reward* yang segera dan yang tertunda.

Dalam MDP diperkirakan nilai $q_*(s, a)$ dari setiap aksi a di setiap keadaan s, atau diperkirakan nilai $v_*(s)$ dari setiap *state* yang diberikan pilihan aksi yang optimal. Kuantitas yang bergantung pada *state* ini penting untuk secara akurat menetapkan kredit untuk konsekuensi jangka panjang pada pilihan aksi individu.

MDP adalah bentuk ideal matematis dari masalah *reinforcement learning* dimana pernyataan teoritis yang tepat dapat dibuat. Disini diperkenalkan elemen

kunci dari struktur matematika masalah MDP, seperti *return*, *value function*, dan persamaan Bellman [9].

MDP dimaksudkan untuk menjadi pembingkaian langsung dari masalah belajar dari interaksi untuk mencapai tujuan. Pembelajar dan pembuat keputusan disebut *agent* (agen). Hal yang berinteraksi dengannya, terdiri dari segala sesuatu di luar *agent*, disebut *environment* (lingkungan). Ini berinteraksi terus-menerus, *agent* memilih *action* (aksi) dan *environment* merespons *action* ini dan menghadirkan *state* (situasi) baru kepada *agent*. *Environment* juga menimbulkan *reward* (imbalan), nilai numerik khusus yang ingin dimaksimalkan oleh *agent* dari waktu ke waktu melalui pilihan *action*-nya [9].



Gambar 2.2 Interaksi *Agent-Environment* dalam MDP

2.3.4 Monte Carlo Method

Metode Monte Carlo menunggu akhir episode untuk memperbarui, metode ini memperbarui secara murni dari pengalaman, membuatnya tidak efisien. Di bawah ini adalah persamaan Monte Carlo untuk memperbarui *value function*:

$$V(S) \leftarrow V(S) + \alpha[G - V(S)] \tag{2.1}$$

Dimana *V* adalah *value function, S* adalah *state* pada waktu tertentu, alpha adalah *step size*, dan *G* adalah *reward* di akhir *episode*. *Value function* optimal V(S) akan cocok dengan *real reward* di setiap askhir *episode* untuk semua *state.* Untuk menentukan *value function* optimal, fungsi pembaruan diatas ditampilan.

Metode pembaruan dari *value function* Monte Carlo harus menunggu akhir episode (dimana *G* diketahui). Bayangkan jika mencoba melatih sebuah model

untuk memainkan catur tetapi hanya dapat memperbarui model di setiap akhir permainan, ini akan menjadi proses yang sangat lambat.

2.3.5 Temporal-Difference Learning

Untuk mempelajari *value function,* TD *learning* menggabungkan pendekatan Monte Carlo dan *Dynamic Programming* [9]. TD *learning* belajar dari pengalaman mentah (memperbarui *value function* berdasarkan *reward* masa lalu) dan juga memperbarui secara dinamis estimasi dari *value function* tanpa menunggu akhir *episode*. Kombinasi dari dua strategi ini yang membuat TD *learning* ssangat kuat.

Bebeda dengan Monte Carlo, daripada menunggu akhir episode untuk mendapat nilai dari *G,* TD memperkirakan *G* di setiap *timestep* berdasarkan informasi baru yang dikumpulkan dari *action* terpilih. Di bawah ini adalah pembaruan dari *value function* TD:

$$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)] \tag{2.2}$$

Dimana R adalah *immediate reward* setelah melakukan *action* berdasarkan *policy, S'* adalah *state* setelah *action* dilakukan berdasarkan policy, dan *gamma* adalah istilah penskalaan (*scaling term*). Perbedaan dengan metode Monte Carlo adalah *G* diganti dengan $R + \gamma V(S')$. Ini adalah persamaan Bellman dan berperan penting dalam RL. TD memperkirakan *G* sebagai *immediate reward* ditambah *value* perkiraan dari *state* selanjutnya dikali sebuah *scaling term (gamma).*

2.4 Q-Learning

Q-Learning merupakan sebuah algoritma kontrol *off-policy* TD. Sebuah algoritma *off-policy* mempelajari *policy* optimal tanpa memedulikan *action* dari *agent,* sedangkan *on-policy* mempelajari *value function* optimal untuk policy yang dijalankan oleh *agent. Q-learning* mempelajari Q optimal berdasarkan *state* selanjutnya dan *greedy action* (*action* dengan *value* tertinggi).

Ide dari *Q-Learning* adalah *agent* memilih *action (*aksi*)* berdasarkan *behavior policy*, tetapi juga mempertimbangkan sebuah aksi alternatif yang mungkin telah diambil, jika mengikuti *target policy*. Hal ini memungkinkan

behavior dan *target policy* untuk meningkat/bertambah baik, memanfaatkan *action-values Q(s, a)*. Prosesnya mirip dengan algoritma *off-policy* Monte Carlo. Pembaruanya dirumuskan sebagai berikut [9]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \qquad (2.3)$$

Dalam hal ini, *action-value function* yang dipelajari, *Q*, secara langsung mendekati q_*, *action-value function* optimal, tidak tergantung pada *policy* yang diikuti. Ini secara dramatis menyederhanakan analisis algoritma dan memungkinkan bukti konvergensi awal. *Policy* tersebut masih berpengaruh dalam menentukan pasangan *state-action* mana yang dikunjungi dan diperbarui. Namun, semua yang diperlukan untuk konvergensi yang benar adalah bahwa semua pasangan terus diperbarui. Ini adalah persyaratan minimal dalam arti bahwa metode apa pun yang dijamin untuk menemukan perilaku optimal dalam kasus umum harus memerlukannya. Di bawah asumsi ini dan varian dari kondisi aproksimasi stokastik biasa pada urutan parameter *step-size*, *Q* telah terbukti konvergen dengan probabilitas 1 hingga q_*. Algoritma *Q-learning* ditunjukkan di bawah ini dalam bentuk prosedural [9].

Algoritma Q-learning

Parameter algoritma: *step size* $\alpha \in (0, 1]$, *small* $\varepsilon > 0$

Inisialisasi $Q(s, a)$, untuk semua $s \in S^+, a \in A(s)$, secara bebas kecuali *Q(terminal, ·)* = 0

Ulangi untuk setiap epsiode:

Inisialisasi *S*

Ulangi untuk setiap step dari episode:

Pilih *A* dari *S* menggunakan policy yang diturunkan dari *Q* (misal, *ε-greedy*)

Ambil *action A*, observasi *R, S'*

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

sampai *S* adalah terminal

Selama pelatihan, agen mengeksplorasi ruang aksi menggunakan eksplorasi *epsilon-greedy*. Selama setiap interval kontrol, agen memilih aksi acak dengan probabilitas ϵ, jika tidak, agen memilih aksi secara *greedy* sehubungan dengan *value function* dengan probabilitas 1-ϵ. Aksi *greedy* ini adalah aksi yang *value function*-nya (*Q-value*) paling besar.

Untuk memperkirakan *value function*, agen Q-learning mempertahankan fungsi nilai (*value function*) Q(S,A), yang merupakan tabel atau aproksimator fungsi. Fungsi nilai Q(S, A) mengambil observasi S dan aksi A sebagai masukan dan mengembalikan harapan yang sesuai dari imbalan jangka panjang. Ketika pelatihan selesai, pendekatan *value function* yang dilatih disimpan dalam fungsi nilai Q(S,A).