Factored Planning for Controlling a Robotic Arm: Theory

Jaesik Choi and Eyal Amir
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{choi31,eyal}@cs.uiuc.edu

Abstract

Controlling robotic arms is important for real, physical world applications. Such control is hard because movement of one joint affects the position of many of the rest. In this paper we present an algorithm that finds plans of motion from one arm configuration to a goal arm configuration. Our algorithm is unique in two ways: (a) It takes time that is only polynomial in the number of joints \(O(m^7 \cdot 2^n)\), with \(m\) joints and \(n\) island obstacles, thus allowing scaling up to complex arms; and (b) it decomposes the control problem to that of the separate joints, thus enabling future development of reactive modules. The algorithm leaves each joint somewhat independent of the rest by reformulating the domain description and re-partitioning it. Our algorithm is exact when the location of the end effector on the optimal path approaches to the goal location within finite steps (depth). Also it has bounded error with respect to the optimal path in the discretized environment. Our approach is promising because it leads naturally to a subsumption-architecture-like control of robotic arms.

1 Introduction

A robotic arm is a manipulator that has revolving joints connecting oval-like pegs, much like a human’s shoulder, elbow, wrist, palm, and fingers. The complexity of motion planning increases exponentially with the dimensionality of the space (the number of joints of the arm) (Canny 1987).

Nonetheless, robotic arms are crucial for general purpose mobile robots, so much work has been invested in studying their controls (Kuffner. & LaValle 2000), (Kavraki et al. 1996), (Brock & Khatib 2000). Unfortunately, current algorithms still depend exponentially on the dimensionality of the configuration space (which is proportional to the number of joints). In addition, the probabilistic approaches are not complete, and they provide a solution with probability that depends on the computation time spent.

In this paper we present a path planning algorithm that can scale to robotic arms with high degrees of freedom. To reduce the complexity of the planning problem, we focus on the fact that each angle of a robotic arm is independent in some sense. That is, the configurations of any joint is independent of other joints with the exception of the adjacent previous joint. Thus, we partition the planning problem into small subdomains of each joint.

Our approach is composed of two procedures: factoring and planning. We partition the problem manually into subdomains that correspond to each joint. Then, in each subdomain, we find a comprehensive set of plans of action and process a tree of subdomains in a dynamic programming fashion. This is done with a modified Factored Planning algorithm (Amir & Engelhardt 2003).

During manual factoring, the configuration space of our robotic arm is represented using distinctive groups of axioms in Propositional Logic (ground predicate calculus). We represent actions and complex actions on joints with preconditions and effects in PDDL. The actions are grouped based on the locations of an end effector. The actions, which are related to the location of an end effector, are merged into a group, if there is no obstacle. However, if there are some obstacles that prevent the actions from being merged, the actions are split into several groups. Based on the achieved PDDL actions, a planner finds a path which is sound and complete.

We prove the complexity of our algorithm in various environments; without any obstacles, with a convex island, with ‘\(n\)’ convex islands, and with an infinite number of obstacles. As a baseline, we prove that the complexity of the algorithm without any obstacles is \(O(m^7)\) (when \(m\) is the number of joints). We also prove both the complexity with a convex island and the complexity with ‘\(n\)’ convex islands. Although the complexity of our algorithm increases exponentially with the number of convex islands, \(O(m^7 \cdot 2^n)\), we also prove that the complexity of the algorithm is bounded by \(O(m^5 \cdot c^m)\) (when \(n\) is the number of obstacles, \(c\) is a constant and \(m\) is the number of joints). Thus, in many environments our algorithm is superior computationally to the previous approaches. Our algorithm is guaranteed to position the end effector in its target position. Furthermore, we give a condition that guarantees that the configuration error (distance from the requested overall configuration in some norm) in the planned path (with respect to the

---

1The returned paths of our algorithm are sound, because the paths always make the arm move to the goal location. The suggested algorithm is complete, because the algorithm returns a path if there is a path.
configuration in the optimal one) is bounded.

The previous works can be categorized into three groups; Potential Field (Khatib 1986), Cell Decomposition (Schwartz & Sharir 1983), and Roadmap (Canny 1987) (Kavraki et al. 1996). They are based on the configuration space which is a set of all the possible configurations of all joints. Thus, the complexity of the methods for complete planning are exponentially proportional to the number of joints; the complexity of Potential Field is \( O(c^m) \), the complexity of Cell Decomposition is \( O(2^m) \), and the complexity of Roadmap Method is \( O(2^m) \) (when \( m \) is the number of joints). Although the original Potential Field approach (Khatib 1986) is not complete, the wavefront-expansion (Barraquand & Latombe 1991) is a complete algorithm. While Cell Decomposition approaches group the adjacent cells in the configuration space, we group the configurations of which the end effectors indicate the same location. Probabilistic Roadmap currently dominates the motion planning literature. Because it generates samples in the configuration space at random, the complexity is dramatically reduced. However, it is not complete without probabilistic assumption.

In the following sections, we specify our algorithm, analyze the complexity, and prove an error bound. In section 2, the state definitions in propositional logic, the PDDL actions and planning algorithm are suggested. At the section 3, we analyze the complexity of our algorithm. In section 4, we prove the error bound of the suggested algorithm comparison to the previous approaches. We respectively state the related works and conclusion in the last 2 sections.

2 Factored Planning for a Robotic Arm

In this section, we present the state definitions and a path planning algorithm for a robotic arm. Each subdomain of the robotic arm corresponds to a subset of the fluents and actions that are only related to a joint. A fluent represents the location of any joint of an arm. An action represents the movement of a joint. The fluents and actions are only shared by the adjacent joints.

An Illustrative Example

We want to start this section with an illustrative example. Figure 1 shows a simple movement of the joint. The left side of Figure 1 represents the Workspace of 2nd joint. Suppose that the location \( W_1 \) is \((x = 0, y = 0, \theta = \pi/2)\). That is, \( x, y \), and \( \theta \) are respectively 0, 0, and \( \pi/2 \). The location of \( W_2 \) is \((3, 3, \pi/4)\). When we move the 1st joint in \( W_1 \) toward the right, the location of 2nd joint changes from \( W_2(3, 3, \pi/4) \) to \( W_2'(4.1, 1.1, \pi/12) \). We call the movement an action of \( W_2 \) \((\text{act}(W_2, W_1, \text{moveright}))\). Although there are many actions which are related to the 2nd joint and the location \( W_2 \), we only consider the unit movement of any previous joints.

Our task one is relocating joint 3 from one location to another. Formally, our task is finding a plan for relocating the location of joint 3 from the start location to a goal one.

Suppose that all the possible positions of joint 3 are \( W_3, W_3', \) and \( W_3'' \). A possible task is relocating joint 3 from \( W_3 \) to \( W_3' \), from \( W_3' \) to \( W_3'' \), from \( W_3'' \) to \( W_3''' \), or the same in the opposite direction.

We find a plan of movement for the joints using the following dynamic programming algorithm. We process each joint separately as follows, starting from joint 1, and proceeding to joint 2 and finally 3. Processing each joint involves running a planner on all possible input starting states and all possible goal conditions. It records those input and goal conditions that have a valid plan, and transfers them all as new macro actions to the subsequent joint. For joint 1, we compute all the possible positions of joint 2 (\( W_2, W_2' \)) that joint 1 movements can entail. Then we record all those positions as new macro actions that joint 2 can request from joint 1. The planning problem for joint 2 becomes finding all the positions that it can entail for joint 3, given its original actions and its (new) macro actions.

We describe this algorithm more precisely now. Each joint has an associated subdomain and messages that it computes and sends to its parent joint subdomain. A message from joint i includes actions that are constructed by subdomain i and its child subdomains. All messages are of the form “if \( X_0 \) holds, then actions of the 2nd joint can make \( X_1 \) hold”. In this example, the 2nd joint has a message “if the first joint is on \( W_1(0, 0, \pi/2) \) and the 2nd joint is on \( W_2(3, 3, \pi/4) \), then an action(\( \text{act}(W_2, W_1, \text{moveright}) \)) can make the 2nd joint be on \( W_2'(4.1, 1.1, \pi/12) \)”. This message is sent to the 3rd joint. The 3rd joint can convert the message into an action, because the 3rd joint has information for manipulating the shared fluents. From the message, the 3rd joint constructs a new action “if the first

Figure 1: The left side of figure represents the Workspace of the 2nd joint. \( W_2 \) and \( W_2' \) are the locations of the 2nd joint. For example, \( x, y, \) and \( \theta \) of \( W_2 \) are respectively 3, 4, and \( \pi/4 \). The \( \text{act}(W_2, W_1, \text{moveright}) \) is an action which moves the location of the 2nd joint from \( W_2(3, 4, \pi/4) \) to \( W_2'(4.1, 2.1, \pi/12) \). The right side of figure represents the Workspace of the 3rd joint. \( W_3, W_3', \) and \( W_3'' \) are the locations of 3rd joint. The 2nd joint sends the message to the 3rd joint. An action \((\text{act}(W_3, W_1, \text{moveright}))\) is constructed from the received action, \( \text{act}(W_2, W_1, \text{moveright}) \). Moreover, the 3rd joint makes a new action, \( \text{act}(W_3, W_2, \text{moveright}) \).
The set of actions are sent to (the set of actions which are related to the 3rd joint) to the arm. To simplify, we assume a 2-dimensional workspace. In the general case, we decompose the problem of controlling a robotic arm into the subdomains in Figure 2. Each joint (e.g. controlling a robotic arm) has actions that includes its movement (move_{j-1} left, move_{j-1} right, move_{j-1} left...move_{j-1} |A_{j-1}|) and sends the messages to the next joint (e.g. j + 1_{th}).

State Definition in Propositional Logic

Here, we formally define the Workspace (\(W\)), End-effector Space (\(ES_i\)) and the set of actions (\(A\)). In the simple case (without any obstacles), we can represent the state only with the Workspace and the set of actions without the End-effector Space. However, in the most general case (with many obstacles), the End-effector Space is also required for representing the complex environment, because a location of Workspace is divided into multiple points of End-effector Space by the obstacles.

The Workspace of the robot (\(W_{robot}\)) is the set of discretized locations that can be occupied by any joint of the arm. To simplify, we assume a 2-dimensional workspace. A location in the workspace (\(W\)) is represented by the \(w = (x, y, \theta)\) \((x \in X, y \in Y, \theta \in \Theta)\), when \(X\) is the discretized x axis, \(Y\) is the discretized y axis, and \(\Theta\) is the discretized angular orientations). That is, the set of all the locations is \(W_{robot} = \{(x, y, \theta)|(x \in X) \wedge (y \in Y) \wedge (\theta \in \Theta)\}\).

The End-effector Space of the \(i_{th}\) joint (\(ES_i\)) is the set of pairs of a location and its representative configuration; \(es = (w_i, \langle c_j \rangle)\) when \(c_j\) is an angular configuration of the \(j_{th}\) joint (the direction of \(j_{th}\) joint). Each element is indexed by \(loc\) and \(conf\), that is \(es(loc) = w\) and \(es(conf) = (c_j)\).

\(A_i\) is the set of actions of \(i_{th}\) subdomain. \(A_i(es)\) is the set of actions that are grouped by an element \(es \in ES_i\). An action \((act_{i(es,w,moveleft)})\) of \(A_i(es)\) is defined with PDDL form as following.

\[
\text{pre}: \text{end}_i(w_i) \wedge \text{end}_i(es(loc))
\]
\[
\text{del}: \text{end}_i(es(loc))
\]
\[
\text{add}: \text{end}_i(w_i')
\]

when \(\text{end}_i()\) is the predicate for the location of \(j_{th}\) joint and both \(w_i\) and \(w_i'\) are locations in \(W_{robot}\).

A complex action \((act_{i(es,w,moveleft)}) \in A_i(es)\) means that a unit left movement of the \(j_{th}\) joint at \(w_j\) changes the location of \((i + 1)_{th}\) joint from \(es(loc)\) to \(w_i'\). Based on the change of \(i_{th}\) joint, the movement of the next \((i + 1)_{th}\) joint can be described as following.

\[
\text{pre}: \text{end}_i(w_i) \wedge \text{end}_i(es(loc)) \wedge \text{end}_{i+1}(es'(loc))
\]
\[
\text{del}: \text{end}_i(es(loc)) \wedge \text{end}_{i+1}(es'(loc))
\]
\[
\text{add}: \text{end}_i(w_i') \wedge \text{end}_{i+1}(w_i')
\]

The location of \((i + 1)_{th}\) joint can be described with the previous location of the joint \((es'(loc))\) and the \(act_{i(es,w,moveleft)}\) of \(i_{th}\) joint. That is, the movement of the \(j_{th}\) joint at \(w_j\) not only changes the location of \(i_{th}\) joint from \(es(loc)\) to \(w_i'\) but also changes the location of \((i + 1)_{th}\) joint from \(es'(loc)\) to \(w_i''\). Now, the new action \(act_{i+1(es',w,moveleft)} \in A_{i+1}(es')\) can be defined without \(\text{end}_i()\).

\[
\text{pre}: \text{end}_i(w_i) \wedge \text{end}_{i+1}(es'(loc))
\]
\[
\text{del}: \text{end}_{i+1}(es'(loc))
\]
\[
\text{add}: \text{end}_i(w_i') \wedge \text{end}_{i+1}(w_i'')
\]

Figure 3 conceptually explains the reason why two joints are independent. Suppose that we make an action ‘moveleft’ of \(j_{th}\) joint located in \(w_j\). The moveleft of \(j_{th}\) joint at \(w_j\) with configuration ‘a’ moves the location of \(i_{th}\) joint from \(w_i\) to \(w_i'\). The movement of \(j_{th}\) joint at \(w_j\) with configuration ‘b’ also moves the location of \(i_{th}\) joint from \(w_i\) to \(w_i''\). Here, we want to note that the specific configuration of the arm is irrelevant given the location of \(j_{th}\) joint \((w_j)\) and the location of \(i_{th}\) joint \((w_i)\) of an action, \(act_{i(es,w,moveleft)}\) and \(es(loc) = w_i\).
and the length of joint given the location of joint and the number of shared fluents. Thus, we modify the algorithm to reduce the size of messages. For the same reason, we make a few assumptions, setting both the interaction (k) and the depth (d) equal to 1.

We reduce the number of messages with the domain knowledge. In the general-purpose factored planner, subdomains have to find the actions for all the possible combinations of preconditions and effects of shared fluents. However, in this special domain, it is enough to search the actions that are the subset of the possible locations of a joint. We can simply bound the size of Workspace $W$, if the number of shared symbols between the subdomains is $m$, the number of action symbols is $v$, and $l$ is the number of shared symbols between the subdomains. The depth (d) is the searching depth of planning, which sends messages between subdomains. The depth (d) is the searching depth of planning, and the depth (d) are equal to 1.

In the general case, all the possible pairs of preconditions and effects are $2^{|W|}$, because the any location ($w$) in $W$ can be TRUE of FALSE. However, the complexity is reduced, if we focus on the fact that the action requires one proposition (end$(w)$) in preconditions and another proposition (end$(w')$) in effect. When the $w$ and $w'$ are locations of $jth$ joint in the working space ($W$), all the possible combination of fluents is only $|W|^2$.

### The Complexity of this Algorithm

The Factored Planning algorithm (Amir & Engelhardt 2003) is sound and complete, given the subdomains and certain parameters, k (interactions) and d (depth). The algorithm terminates at time $O(m \cdot 2^{m+k} \cdot \min((a+k)^d, k \cdot 2^d))$ with parameters $m$, $a$, and $v$ (when $m$ represents the number of subdomains; $a$ is the largest number of action symbols; $v$ is the largest number of fluent symbols in any subdomains; and $l$ is the largest number of shared symbols between the two subdomains). The complexity of our modified Factored Planning algorithm is reduced as following, because we need only the part of the truth assignments of shared fluents. Moreover, we assign both $k$ and $d$ a value of 1.

$$O(m \cdot l \cdot \min((a+1)^d, 1 \cdot 2^d)) = O(m \cdot l \cdot a)$$

Here, m is the number joints, l is the number of discretized Workspace, and $a$ is the number actions at the last (mL) joint.

We can simply bound the size of Workspace $W \equiv (X,Y,\theta)$. Suppose a robotic arm which has $m$ joints and each joint has $c(=\frac{2\pi}{m})$ discrete angles. The space of the end effector can be bounded by $[-N..N] \times [-N..N] \times [0..2\pi]$, if the $N$ is defined by the average length of the joints ($N = mL \equiv \sum_{i=1}^{m} length_i$). Thus, the complexity of Workspace is following.
The complexity of the modified Factored Planning is following (when $c (= \frac{2\pi}{\Delta \theta})$ is the number of discrete orientations, and $\Delta \theta$ is a unit angular displacement.)

$$O(m^2 (\Delta \theta)^2 \cdot |ES_m|)$$

Algorithm 2 SingleJointPlan

- $es$: a point in the end effector space
- $ang$: the direction to the next link
- $length_i$: the length of $j_{th}$ link
- $A_{j-1}(es)$: the actions of $es$

SUBROUTINE SingleJointPlan($es$, $ang$, $length_j$, $A_{j-1}(es)$)
1. Let $es'$ new position and configuration of $j_{th}$ joint, and act ← ∅
   $es'(loc) ← es(loc) + \text{trans}(es(loc), ang, length_j)$ \(^3\)
   $es'(conf) = es(conf) \cup ang$
2. For each act$_{j-1} \in A_{j-1}(es)$
   (a) Make act$_{new}$ from act$_{j-1}$ (The $end_i(w_i)$ is in the pre-condition of act$_{j-1}$)
      pre: $end_i(w_i) \land end_j(es'(loc))$
      eff: $-end_i(es'(loc)) \land end_j(w_i)$
   (b) If $j_{th}$ joint of act$_{new}$ do not collide with any obstacle
      Insert act$_{new}$ into act
3. For each leftmove and rightmove of $j_{th}$ joint by $\Delta \theta$
   (a) Make act$_{new}$ as following
      pre: $end_{j-1}(es(loc)) \land end_j(es'(loc))$
      eff: $-end_j(es'(loc)) \land end_{j-1}(w_i)$
   (b) If $j_{th}$ joint of act$_{new}$ do not collide with any obstacle
      Insert act$_{new}$ into act
4. Return ($es'$, act)

Grouping the Set of Actions

We reduce the complexity of the planning problem by grouping the actions based on the location of joint. That is, we don’t consider a specific configurations of joints of an arm, if the location of end effector are same. If we separately store the set of actions for the specific configuration, the complexity is same with the configuration space which is exponentially proportional to the number of joints.

We group the set of actions, because storing the sets of actions for every configurations is not only expensive but also redundant. Thus, we merge some sets of actions into a large group, when the locations of end-effector are on the same location. Our grouping method may lose some data, because a specific configuration only contributes to the part of actions. This prevent our algorithm finding an optimal path in some case. However, this method is beneficial for reducing complexity.

For grouping the sets of actions, we define the Homotopic Configurations. Two configurations with the same endpoints are homotopic if one can be continuously deformed into the other. The concept of the Homotopic Configurations is based on the Homotopic Paths (Brock & Khatib 2000)\(^3\). Although it is similar to the concept with Homotopic Paths, we adapt the notion of Homotopic to the configuration. For example, in Figure 4, the two configurations ‘a’ and ‘c’ are Homotopic and ‘a’ and ‘b’ are not Homotopic.

In our algorithm, the grouping is decided by a local planner. The local planner could find a path from one configuration to another, if the two configurations are Homotopic. This can be achieved by any inverse-kinematics algorithm which avoids obstacles. There are many local

\(^3\)Two paths with the same endpoints are homotopic if one can be continuously deformed into the other
path planning algorithms for such a calculation (Zlajpah & Nemec 2002). The local planner is used for finding the Homotopic Configurations which are deformable each other. However, in the real environment, two Homotopic configurations are not always continuously deformable each other, due to the rigid body of link6.

### Soundness and Completeness

**Algorithm 4 PathPlan**

- **esstart**: the initial location and its configuration
- **esgoal**: the goal location and its configuration
- **Am**: the set of actions of mth joint
- **ESm**: the end effector space of mth joint
- **depth**: the depth for searching path

```plaintext
SUBROUTINE PathPlan(esstart, esgoal, Am, ES_m, depth)
1. esstart ← FindES(esstart, Am, ES_m)
2. esgoal ← FindES(esgoal, Am, ES_m)
3. j ← 1, Rj ← {esstart}, Rtotal = ∅, A’s = ∅
4. Do until j=depth
   (a) For each es ∈ Rj−1
      i. For each act(m,es,ui,...) ∈ Am(es)
         If the act(m,es,ui,...) is valid for es
            es0 ← moved es by the act(m,es,ui,...)
            es ← FindES(es0, Am, ES_m)
            If es /∈ Rtotal then
               Make new Action(move(es,es')) as following
               pre: es ∧ donei−1 ∧ ¬donei
               eff: es' ∧ donei
               Add move(es,es') to Act_global
               Add es' to Rtotal
      (c) j ← j + 1
   5. Init(esstart, done0, ¬done1, ..., ¬donej−1)
   6. Goal(esgoal, done0, done1, ..., donej−1, donej)
   7. Search for plans (Φ) in Init, Goal, Act_global
   8. return Φ
```

We prove this path planning algorithm is sound and complete. Our planning algorithm is sound, because all the returned paths are valid. That is, we can control the robotic arm along the returned path. Moreover, the planning algorithm is complete. If there is a path from the start to the goal, the algorithm finds a path that can reach to the goal position.

**Lemma 1.1**: With the robotic arm, we can find movements for every move(es,es') ∈ Act_global in PathPlan

**Proof**. Suppose that there is a move(es,es') ∈ Act_global and an act(m,es,ui,...) contributes to the movement. We can assign a location es0 which is moved from es by the act(m,es,ui,...) in the PathPlan. There are movements from es0 to es', because the FindES(es0, ...) returns an es' only if the local planner finds a path from es0 to es'.

*We check the Homotopic relationship with local planner. To reduce the complexity of this algorithm we allow the end point can be moved, if the movement can be managed by the local planner. This makes the algorithm require a verification function in the PathPlan, because the transition between the configurations may cause collision at the outer link, if the location of end point is not fixed.*

**Theorem 1.1**: (Soundness of RobotArmPlan) When a path ⟨es0, es1, es2, ..., esn⟩ is returned by the algorithm, there is a path that passes through the configurations ⟨es0(conf), es1(conf), ..., esn(conf)⟩ with the robotic arm (when es0 is esstart and esn is esgoal)

**Proof**. By the Lemma 1.1, we can find movements for any consecutive es points. For any es−1 and es pair, we can find movements that control the robotic arm from an esj−1(conf) to an esj(conf). Thus, we can use mathematical induction for the whole path.

**Theorem 1.2**: (Completeness of RobotArmPlan) If there exist a unique path whose number of movements is less than depth, our algorithm finds a path.

**Proof**. We use the mathematical induction to prove the completeness.

For the 1st step, if the path is a movement of robotic arm, we can simply find it. It is because we have all the actions for each point (es ∈ ES_m). That is, if there exists a unique movement from a location es to a location es1, we have a complex action for the movement in A(es).

For the n − 1th step, we assume that if there exist n − 1 movements from a location es to a location esn−1, we can find a path.

For the nth step, suppose that a path ⟨es0, es1, ..., esn−1, esn⟩ is unique from es0 to esn. Based on the assumption of mathematical induction, we already have a path from es0 to esn−1. Moreover, we have an action (act(m,esn−1,...)) from esn−1 to esn. Thus, the algorithm finds a path ⟨es0, es1, ..., esn−1, act(m,esn−1,...), esn⟩.

**Algorithm 5 FindES**

- **es**: a configuration
- **Act**: the set of actions
- **ES**: the end effector space

```plaintext
SUBROUTINE FindES(es, Act, ES)
1. For each es' ∈ ES for es'(loc) = es(loc)
   (a) If es'(conf) and es(conf) are Homotopic Configurations
      i. If local planner find a path from es' to es
         return es'
   2. Make a new es''(≡ es)
   3. Add es'' to ES
   4. Act(es'') ← Act(es)
   5. return es''
```

### 3 Complexity Analysis

Here we analyze the complexity of the suggested algorithm. We prove the complexity of this algorithm in various environments; without obstacles; with a convex island obstacle; with “n” convex island obstacles; and with infinite number

*Here, the action does not always guarantee the optimal path, because we don’t store the actions of a specific configuration.*
of obstacles.

If we find the path with an exact path planning algorithm (Khatib 1986) (Brock & Khatib 2000) in the dimensional space, the complexity is

$$O \left( \min \left( e^m, m^d \right) \right)$$

Without Obstacles

We have \(|W| = cN^2 = \frac{2\pi c}{\Delta \theta} mL^2\) axioms representing the positions of the end effector of the robotic arm. Moreover, we have all the possible “move” actions between the axioms (positions). At the \(t_{th}\) step, we can find the reachable position with, at most, \(t\) actions through dynamic programming in Algorithm 4. If we execute further \(d\) steps, we can find all the possible paths whose lengths are shorter than \(d\). Even if there are certain non-valid paths, which are caused by the incomplete previous step, we can locate a path from the returned paths.

Lemma 2.1: Without any obstacles, the complexity of RobotArmPlan is following given m links.

$$O(m \cdot |W|^2 \cdot |ES_m|) = O(m \cdot |W|^3) = O(m^7)$$

Proof. For the one positions in \(ES_m\), we have maximum (\(|W| = cN^2\) neighbor positions which are reachable with a move action. Because we have a total of \(cN^2\) positions and visit each position once, the total step is not more than \(cN^2\). Moreover, there are no two \(es_0\) and \(es_1\), which are in \(ES_m\) and \(es_0(\text{loa}) = es_1(\text{loc})\). In the StorePartPlan, \(es_1\) merges into the \(es_0\), if the \(es_0\) is already added to the \(ES_m\). Without obstacles, the local planner ⁸ finds a path between two configurations if they are Homotopic Configurations.

With a Convex Island

Lemma 2.2: If there exists a convex island, the size of end effector, \(|ES_m|\), is bounded by

$$O(|ES_m|) = O(2 \cdot |W|).$$

That is, for each position, \(w \in W\) (when \(es_0(\text{loc}) = es_1(\text{loc}) = w\) and \(es_0 \neq es_1\)) there are at most 2 distinct elements (\(es_0\) and \(es_1\)) for the location \(w\).

Proof. Suppose that there are 3 (or more) distinctive groups (a, b, and c) whose end effector locate on the same location, and there is an convex island obstacle are blocked by the same object. That is, ‘a’ and ‘b’ are blocked by the island; ‘b’ and ‘c’ are blocked by the island; and ‘c’ and ‘a’ are blocked by the island.

Given the two distinct groups, we can label that one as ‘left-side’ of the island the other as ‘right-side’ of the island. Without loss of generality, suppose that ‘a’ is the left-side and ‘b’ is the right-side. In that case, ‘c’ can be assigned to neither left-side nor right-side. There is no ‘c’ in the given space.

With ‘n’ Convex Islands

Lemma 2.3: The search space for \(n\) convex islands is bounded by \(O(2^n \cdot |W|)\).

Proof. Suppose the search space with \(n - 1\) convex islands is bounded by \(O(2^{n-1} \cdot |W|)\) for mathematical induction. Assume that each location \((w)\) in the \(n - 1\) convex islands environment can have up to \(2^{n-1}\) distinct groups of configurations. That is, \(|ES_{m,n-1}(w)| \leq 2^{n-1}\), when \(ES_{m,n-1}(w) = \{ es|es(\text{loc}) = w \land es \in ES_m \text{ with } n-1 \text{ convex islands} \}\). With an additional island, \(O_n\), each distinct group can be divided into two (and no more than two) groups, which would be either left-side or right-side with respect to \(O_n\). Thus, the distinctive groups for each position, \(w \in W\), can be bounded by \(O(2^n)\).  

With an Infinite Number of Obstacles

Lemma 2.4: If there is an infinite number of obstacles, the search space of the end effector, \(|ES_m|\), is bounded by

$$O \left( \left( \frac{2\pi}{\Delta \theta} \right)^m \right) = O(e^m).$$

Proof. The size of \(|ES_m|\) cannot be greater than all possible configurations of \(m\) joints \((O(\left( \frac{2\pi}{\Delta \theta} \right)^m))\) when \(\Delta \theta\) is the unit angles and \(c = \frac{2\pi}{\Delta \theta}\).

The Size of \(|ES_m|\)

Theorem 2: If there are \(n\) convex islands (obstacles), the search space of the end effector, \(|ES_m|\), is bounded by

$$O \left( \min \left( 2^n \cdot m^2 , e^m \right) \right).$$

When \(n\) is the number of obstacles, \(c = \frac{\pi}{\Delta \theta}\), and \(m\) is the number of joints.

Proof. by Lemma 1.1, 1.2 and 1.3

4 Error Bound

Here, we examine the error of the suggested algorithm with respect to the discretized configuration space algorithm.⁹

Our algorithm approximately discretized the position of each joint, at \((x, y, \theta)\). That is, if the position of the \(n_{th}\) joint of an arm is close enough to a position \((x', y', \theta')\), we assumed that the \(n_{th}\) joint of arm is located on the \((x', y', \theta')\). The position error of inner joints cumulatively influences the position of the end effector, due to the dynamic programming fashion of our algorithm.

The error can be divided into two categories: (1) the displacement of location; and (2) the difference of angles. When we assume that the size of cell in the discretized Workspace is small, relative to the length of joints, the error

⁸Here, we use the discretized configuration space in contrast to the continuous configuration space. That is, all the algorithms, which uniformly split the configuration space into the cells of a configuration, are included in the definition.
Proof. The maximum error of $\Delta x$ is sum of $\Delta x_{pos}$ and $\Delta x_{ang}$ because $\Delta x_{pos}$ and $\Delta x_{ang}$ are respectively bounded by $\frac{1}{2} \text{size(cell)}$ and $L \sin (m \cdot \Delta \theta)$. To bound $\Delta x$ to $r$, we simply bound both the $\Delta x_{pos}$ and $\Delta x_{ang}$ to $\frac{1}{2} r$.

6 Conclusion

Two contributions of our work are (1) an algorithm whose complexity is polynomial to the number of joints, and (2) the decomposition of the control problem into sub problems. When the optimal path approaches the goal location within finite steps (depth), the planning algorithm is sound and complete. Although the complexity is exponential in the number of obstacles, it is only polynomial to the number of joints. Our theoretical results show that the planning algorithm scales well in planning the path of a robotic arm.

References


