A NOVEL CONTINUUM MANIPULATOR

An Undergraduate Research Scholars Thesis

by

ERIC C. COCHRANE

Submitted to the Undergraduate Research Scholars program
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Dr. Dylan Shell

May 2016

Major: Computer Science

TABLE OF CONTENTS

Page

ABSTRACT 1

DEDICATION 2

ACKNOWLEDGMENTS 3

I INTRODUCTION 4

Chameleons to robots 4

II SO MANY STATES AND NOT ENOUGH TIME 5

The problem 5

Less is more 5

III THE DETAILS 7

The manipulator agent 7

The state space 10

Q-learning 10

Convergence 11

Exploration vs. Exploitation 11

Reward Structure 12

Simulation Mechanics 12

IV RESULTS 14

Results from Simulation 14

Physical Testing 14

Optimal Policy 14

REFERENCES 17

ABSTRACT

A Novel Continuum Manipulator

Eric C. Cochrane
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Dylan Shell
Department of Computer Science

We address the problem of controlling continuum manipulators and evaluate Reinforcement Learning to produce a control policy for a robotic platform. Our approach discretizes the state and action spaces to reduce the training needed to converge to an optimal policy. We integrate Q-Learning, computer vision, and a pneumatic system into a single robotic platform. The agent is tasked with tracking and striking a target with a continuum manipulator modeled by a party-blower. We describe Reinforcement Learning, the methods used to train the agent, and describe the performance of the optimal policy successfully striking the target.

DEDICATION

I dedicate this research to my parents, Maria Cruz and John Cochrane, who have tirelessly supported me throughout my academic journey, and taught me the true value of education.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Dylan Shell for his outstanding mentoring, patience and for fostering an atmosphere of creativity, and expression. His guidance has tremendously improved my capabilities as a researcher, a scholar, and student. I would also like to thank Dr. Robin Murphy for introducing me to research, and for her valuable guidance and expertise.

CHAPTER I

INTRODUCTION

Chameleons to robots

What do chameleon tongues, elephant trunks, and octopi have in common? They are all continuum manipulators due to the near infinite degrees of freedom of these appendages. Continuum manipulators are compliant, compact, and have the potential to solve some problems that rigid link manipulators can not. Traditional path planning methods are successful for rigid manipulators, however difficulties arise when applied to continuum manipulators due to their many degrees of freedom [4]. When applied to continuum manipulators, rigid manipulator path planning requires prohibitive computational overhead, so different methods of control are needed. Much of the existing work is concerned with exploiting the simplified path planning to easily grasp objects, while little work has been done trying to target and strike objects quickly. Continuum manipulators are often used in steerable catheter needles, and other medical applications where rigidity could cause injury to the patient [5]. Furthermore, the development of new methods and algorithms for computationally expensive problems could lead to an increased use of continuum manipulators in conventional robotics.

Reinforcement learning has promise in producing a control policy that scales well with the many degrees of freedom of continuum manipulators [6]. Model-free Reinforcement Learning algorithms, such as SARSA and Q-Learning, have the potential to provide an effective control system without explicitly modeling the complex kinematics of continuum manipulators. Our approach shows that Q-Learning can be used to control a robotic platform with a continuum manipulator in order to strike a target with some probability.

CHAPTER II

SO MANY STATES AND NOT ENOUGH TIME

The problem

Reinforcement learning algorithms such as Q-Learning are self-contained feedback loops between the learning agent and the environment. A learning agent performs an action in a given state, and the environment returns a reward and the new state. Depending on the exploration function used the agent will select and action based on whether it wishes to explore the environment for a greater reward or exploit its current knowledge. At first the solution may seem obvious: map all possible action-state pairs to their reward, and select the maximizing action for each pair. This seems plausible, however the manipulator agent is operating in continuous space and some simplification of the problem is needed to accelerate the learning process. The exploratory behavior varies based on the flavor the reinforcement learning algorithm, but an uninformed or random search through the state space would greatly increase the amount of training needed to form a policy since the probability for the agent to randomly strike the target is relatively low. To address this problem we discretized the state and action space coarsely in an attempt to minimize the amount of training needed to produce an optimal policy for the agent by reducing the total number of states-action pairs. Another potential problem is that even if the agent were to undergo thousands of training sessions it could fall victim to over-fitting the policy to the training scenario. For example, if the agent was only trained using targets that were outside of the range of the manipulator, the emergent behavior would learn never to strike at all. Although this behavior is undesirable it is the optimal policy for the given scenario under the current reward structure. To avoid convergence to similar unwanted optimal policies, target positions during training should occur with uniform probability within a fixed distance from the experiment space.

Less is more

The approach discussed in this paper discretizes the state space by limiting the servo positions, and hashing the target location into one of several partitions. The target location is taken from the image produced by the camera, and its state is represented by pixel coordinate of the centroid

of the target. Even with the reduction of the state-action space, it is still large enough to warrant an automated testing system or computer simulation for training. A simulator was created in Matlab and is used to train the agent using a simple 3-dimensional model of the experiment. The training data gathered in simulation will not take into account the possibility that the party-blower deviates from a straight line trajectory but it is less time consuming than running thousands of trials while the algorithm has a sparse number of successful actions in its exploration phase. Real-world training should correct for the non-deterministic behavior of the party blower not taken into consideration by the simulation producing a more accurate policy.

CHAPTER III

THE DETAILS

The manipulator agent

The base

The base is a pan and tilt gimbal in the horizontal and vertical plane. The base is connected to a servo controller, and a computer. The SCC-32 servo controller has a serial connection and has supporting software and firmware that reduce the execution time of passing commands to the base. A python script sends commands to the SSC-32 from the workstation computer in the following format: `#<Channel Number>P<Servo Position>`. The servo position for both the lateral and horizontal servos are fixed to 3 uniformly space positions that span each servos range. The horizontal servo ranges from 48° to 132° and the vertical servo ranges from 57° to 123°.

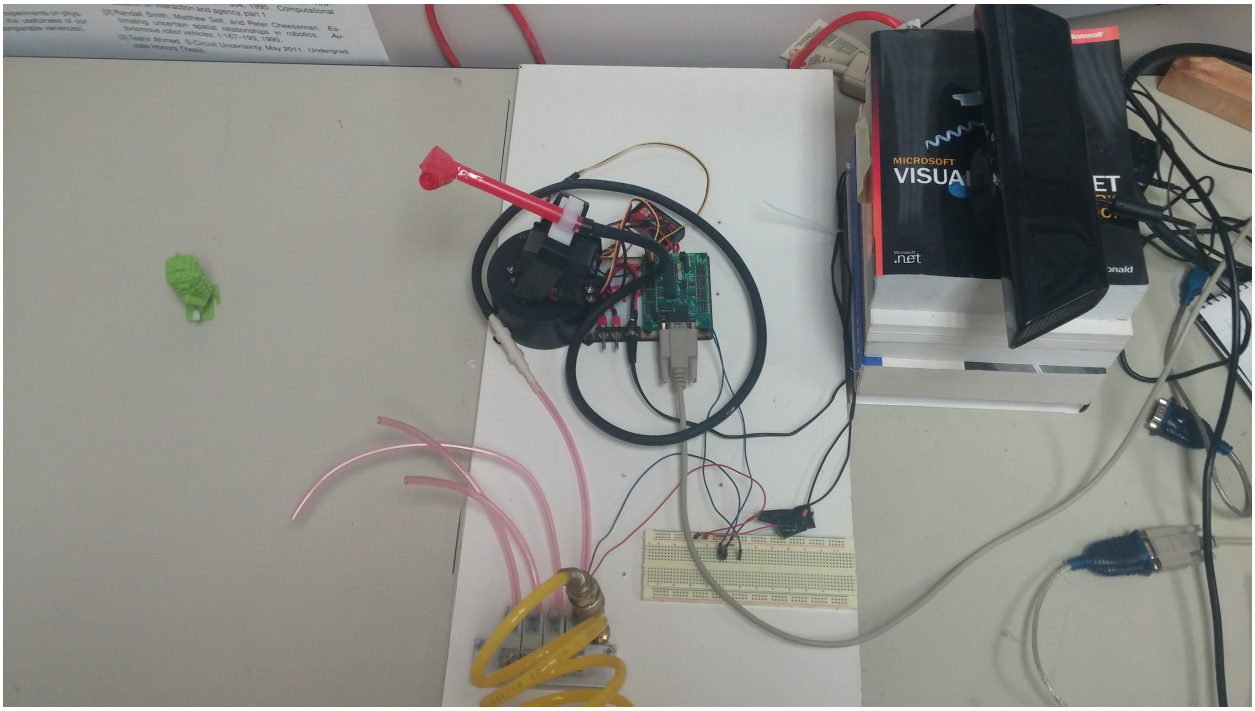

Fig. III.1. A view from above of the the robotic agent.

The pneumatic system

The pneumatic system consists of a small capacity air compressor, a manifold, an electronically controlled pneumatic valve, a regulator, and a party blower. The air compressor feeds a $\frac{3}{4}$ inch hose into the manifold that houses the pneumatic valve. The valve will then connects to the fitting on the party blower. The party blower is mounted on top of a C bracket perched on top of the base. The valve is controlled by the servo controller, and the regulator flow will be static. For the time being, the learning algorithm would be over-complicated by the addition of another dimension to the action space if the regulator was dynamically adjusted. The regulator is set to around 10 PSI.

Fig. III.2. The pneumatic system of experimental setup.

The sensors

The only true sensor is the Kinect camera which is perched behind the manipulator facing the operating area of the manipulator agent. Its location is fixed slightly above the party blower, and does not move with the base. OpenCv [1] is used to track the target's position in the plane of the image, and only tracks objects that are a particular shade of green. This is accomplished by passing the acquired image through a threshold that only accepts a range of the color green. Then the moments of the image are calculated:

$$m_{i,j} = \sum_x \sum_y x^i y^j I(x,y). \tag{III.1}$$

where $I(x,y)$ is the grey-scale pixel intensity at pixel (x,y), and m_{ij} is the raw image moment. The pixel coordinate of the centroid of the largest moment is computed. Using color tracking simplifies the problem of tracking so that we may focus our efforts on the learning component of the agent rather than its sensors and actuators. The calculation of the centroid returns a singe pixel coordinate which is placed into one of nine grid cells. Each grid is indexed and reduces the targets location to membership of the partitions.

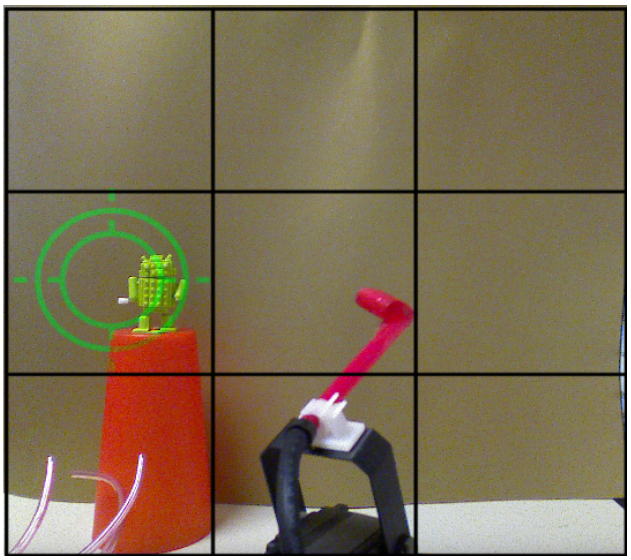

Fig. III.3. The camera image partitioned into the 9 target positions; the green target is located in the middle left cell.

The state space

In reality, the full description of the state space takes into account the servo positions, denoted by α and θ, in the horizontal and vertical plane, whether the party blower is inflated or deflated, and the location of the target in 3 dimensions:

$$State := \{\theta, \alpha, x_{target}, y_{target}, z_{target}\}. \tag{III.2}$$

Our approach only uses one camera, and depth is not explicitly sensed by the agent:

$$State := \{\theta, \alpha, x_{target}, y_{target}\}. \tag{III.3}$$

In this case the target's x and y position are the pixel coordinates where x and y axis are the width and height of the image. Furthermore, the both servos have a range of 0°-180°, but due to the limited field of view of the camera the pan servo in the horizontal plane has a range of 30°-150° , and the tilt servo has a range of 50-120°. The are 9 possible combined servo positions. Although this makes the motion of the manipulator more coarse and less precise when aiming the party blower, it greatly reduces the size of the Q-Table, and decreases the number of trials needed to converge on an optimal policy.

Q-learning

Q-learning is an alternate method of temporal difference reinforcement learning which learns action-utility pairs and their associated reward; if the agent takes action a in state s then reward r is given. The most important property of Q-learning for our purposes is that Q-learning, unlike other machine learning methods, does not require a model of the task environment, i.e. is a model-free method [3]. The model-free aspect of Q-learning allows us to tackle problems which have unknown or complicated models like our control and targeting system for the continuum manipulator.

There are several other properties of Q-learning like the reward structure, the exploration function and delayed gratification which affect the way the agent learns and navigates through the paired state-action space, and the rate at which the agent will converge on an optimal policy.

Convergence

While Q-learning may seem to be the Excalibur of machine learning it has several trade-offs and limitations imposed by the construction of the action and state spaces, and the complexity of the problem faced by the agent. If the state and action spaces are both very large, the rate of convergence to an optimal policy will be slow, and may only converge after a hundred-thousand trials or more. In our case, each learning episode is a physical experiment that requires confirmation that the agent struck the target with the continuum manipulator. If the strike was a success, a reward signal is then passed to the learning agent, but without an automated training system to move the target and confirm a successful strike this tedious operation must be performed by a human. Providing the validation for each learning episode is prohibitively time consuming, especially over several thousand trials.

Exploration vs. Exploitation

One of the fundamental, and unsolved problems of Reinforcement Learning is selecting the optimal scheme for balancing exploration and exploitation in the general case. Reinforcement learning agents must explore their environment to gain any useful information that may later be exploited to maximize their reward and yield the optimal policy for a given state-action pair. The *k*-armed bandit problem is the simplest illustration of the tradeoff between exploration and exploitation. The *k*-armed bandit problem proposes that an agent is an a casino with a infinitely long row of slot machines (called one-armed bandits) hence the name `"k-armed bandit"`. Imagine that you are given a fixed number of pulls, and each slot machine has a independent fixed probability of a payout yielding 1 or 0. The problem arises when we seek to maximize our total payout after *n* pulls. Should the agent pull a new lever each time possibly finding the slot machine with the greatest chance of a payout or stick to a slot machine where the chance of a payout is `"good enough"` and repeatedly pull the lever. It should be noted that the larger *n* is, the larger the penalty of converging to a sub-optimal policy by favoring exploitation over exploration [2].

Reward Structure

The reward structure of any RL agent is central to posing a problem and directing the agent to the problem it needs to solve effectively. We wish to encourage the agent to strike the target so the state-action pair corresponding to a successful strike receives the greatest maximal reward. Lesser rewards are used to shape the agent's behavior away from undesirable policies. In our case, the agent is penalized for moving from one servo position to another, and for firing at a target but missing. If the agent was not penalized for striking a target, it would humorously remain in the same place forever firing expecting the target to wander into its trajectory. Without this penalty the agent would converge to maximal value after thousands of trials since the target could coincidentally be in that position with a high enough probability to skew the expected reward.

Simulation Mechanics

In order to reduce the number of physical training sessions needed a simulator was created to provide virtual training sessions. The simulator takes into account the base and arm servo positions, and the target's position. Although the agent does not perceive any depth information about the target, the simulator must have the distance from the target to the physical agent. In simulation, the target position is randomly generated within the defined parameters of the physical experiment, and placed into one of the target partitions of the agent's state space;the agent behaves exactly as it would if the input of the target's coordinates were coming from the vision system. The core feature of the simulation is computing the expected distance between the extended party-blower and the target. Geometrically, this problem can be phrased as that of computing the distance between a line segment and a point in three dimensions.

To compute this distance we must recognize that the solution will fall into three cases: the point is perpendicular to the line segment, or closest to one of the end points of the line segment. To

determine which case is at hand we first find the dot products of the line segment vector and the vector of the endpoints and the point:

$$\begin{aligned}
\vec{V} &= P_1 - P_0, \\
\vec{W_0} &= T - P_0, \\
\vec{W_1} &= T - P_1, \\
&\quad \vec{W_0} \cdot \vec{V} \leq 0, \\
&\qquad \Rightarrow dist(T, L) = dist(T, P_0). \\
&\quad \vec{W_1} \cdot \vec{V} \geq 0, \\
&\qquad \Rightarrow dist(T, L) = dist(T, P_1).
\end{aligned}$$

where P_0 is the center of the base, P_1 is the end point of the extended party-blower, and T is the target's location. The dot product is negative when the angle between the base of the party-blower and the target is greater than 90°, since $\vec{A} \cdot \vec{B} = |A||B|cos(\theta)$.

CHAPTER IV

RESULTS

Results from Simulation

The training epochs in simulation worked well in shaping the behaviors observed in physical testing. The agent developed a belief that it could hit the target regardless of its distance to the agent if the agent aimed at the perceived target location. This is expected as the agent does not perceive any depth from itself to the target. This is the behavior we should expect to be optimal for the given reward structure and phrasing of the problem. An optimal policy was found for each target location in which the agent moves from its current position to the position corresponding to the target location, and fires when in the target location. Since the servo positions are fixed to the centers of the target partition there is a high probability that the agent will miss the target if the target is not positioned near the partition center.

Physical Testing

Physical testing confirms the results from simulation in that the target is successfully struck if it near the center of the partition and within the party-blower's trajectory. Fifty training epochs of a maximum of 25 actions were conducted in order to adjust the deterministic predictions of the simulator with the probabilistic nature of the party-blower: occasionally the party-blower will fold and strike in a random of perpendicular direction. Execution of the optimal policy also allows the agent to strike a slowly moving target without modification of the policy found.

Optimal Policy

The optimal policy produced exhibits a high probability of success when the target is in range of the manipulator and either near the center of the target partition or along the trajectory between the center and manipulator. The penalty accrued by moving to a new servo position ensures that the agent takes the shortest possible path to the target's partition, and the randomness from previous optimal policies has been replaced by a policy that moves the agent closer to the target.

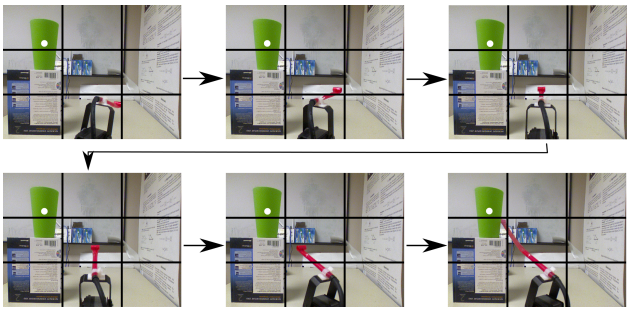

Fig. IV.1. The agent executing the optimal policy for target partition 6. Agent begins in the bottom right grid cell.

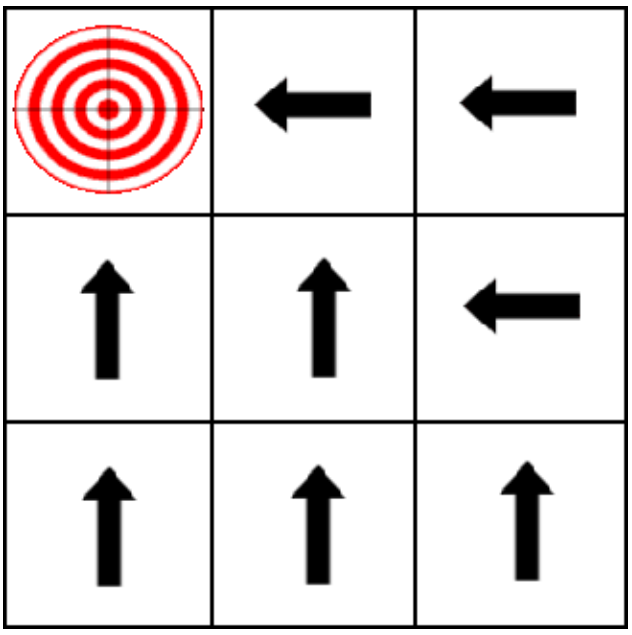

Fig. IV.2. The optimal policy for all states with the target in the upper left grid cell.

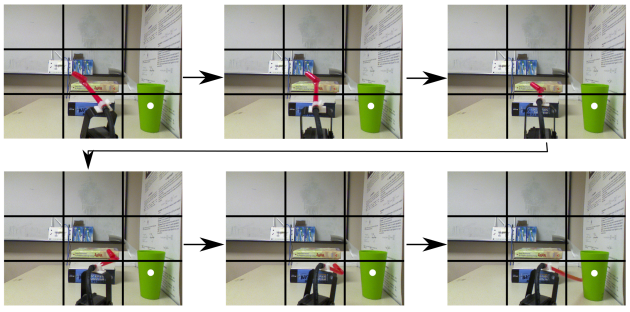

Fig. IV.3. The agent executing the optimal policy for target partition 2. Agent begins in the top left grid cell.

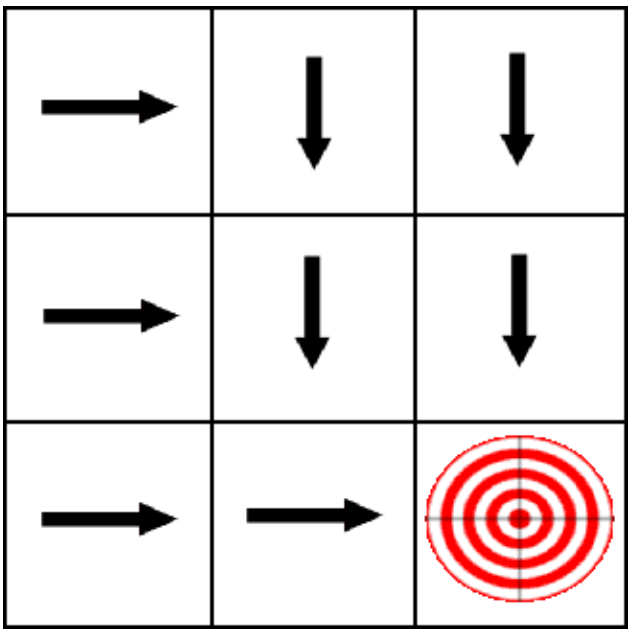

Fig. IV.4. The optimal policy for all states with the target in the bottom right grid cell.

REFERENCES

[1] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2016.

[2] Leslie P. Kaelbling, Michael Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, Volume 4:237–285, 1996.

[3] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[4] J. Kenneth Salisbury. Whole arm manipulation. *Proceedings of the 4th Symposium on Robotics Research*, pages 183–189, 1987.

[5] Ian D. Walker. Continuous backbone continuum robot manipulators. *International Scholarly Research Notices Robotics*, Volume 2013, 2013.

[6] J. Xiao and R. Vatcha. Perceived ct-space for motion planning in unknown and unpredictable environments. *8th International Workshop Algorithmic' Foundations of Robotics*, 2008.

[7] J. Xiao and R. Vatcha. Real-time adaptive motion planning for a continuum manipulator. *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.