Bi-manual Learning for a Basketball Playing Robot

by

Nikhil Kalige

A Thesis Presented in Fulfillment
of the Requirements for the Degree
Master of Science

Approved September 2016 by the
Graduate Supervisory Committee:

Heni Ben Amor, Chair
Aviral Shrivastava
Yu Zhang

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

Sports activities have been a cornerstone in the evolution of humankind through the ages from the ancient Roman empire to the Olympics in the 21st century. These activities have been used as a benchmark to evaluate the how humans have progressed through the sands of time. In the 21st century, machines along with the help of powerful computing and relatively new computing paradigms have made a good case for taking up the mantle. Even though machines have been able to perform complex tasks and maneuvers, they have struggled to match the dexterity, coordination, manipulability and acuteness displayed by humans. Bi-manual tasks are more complex and bring in additional variables like coordination into the task making it harder to evaluate.

A task capable of demonstrating the above skillset would be a good measure of the progress in the field of robotic technology. Therefore a dual armed robot has been built and taught to handle the ball and make the basket successfully thus demonstrating the capability of using both arms. A combination of machine learning techniques, Reinforcement learning, and Imitation learning has been used along with advanced optimization algorithms to accomplish the task.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Robots are employed in different and diverse environments like manufacturing plants assisting the development of large machinery, helping doctors in critical and challenging surgical situations, working in the hazardous environment which the humans cannot reach, etc. They are catching up with the electronic gadgets in trying to occupy the human household. These examples give us a fundamental insight that robots are capable of performing tasks that are repetitive but are mostly incapable of adapting to changing and varied circumstances that come into play when interacting with the humans. Industrial robots are highly inflexible as they are usually programmed by the operator to perform the same set of tasks in a particular manner repeatedly. Hence, the intelligence quotient shown by these machines are pretty low. Humans are reactive and hence a robot that is coexisting with humans will need to be capable of rapidly adapting to these changing situations. They should be able to anticipate the actions so that they can intelligently predict the next action that needs to be performed.

Humans are considered one of the advanced organisms on planet earth, and this is partly due to their intelligence and also due to the dexterity shown by their limbs. Human limbs have the agility and the ability to perform complex actions that can otherwise be difficult to be carried out by a robot. Even the simplest of the tasks like grasping or lifting are difficult to reproduce using robots. These so-called simple tasks involve a huge number of variables like the shape of the objects, the environment in which it is present, the weight which in turn determines the amount of force that needs to be applied. Most of the robots used in the industry today are single armed, and there has been a tremendous amount of research that

has been done in this field. The common robots used for research like Universal Robots UR5 are single armed. The task-space control of such robots is a lot easier since the arm and the environment are the two most important factors.

Most tasks that require the manipulation of an object needs two arms. Bi-manual tasks that are performed using two arms will allow robots to perform tasks that are more complex in nature. The introduction of two arms into the task-space adds additional variables to the problem. These tasks require a high degree of coordination and feedback between the two arms, as even a minute deviation from the norm could lead from bad to disastrous results. Even the simple task of picking up a ball has an enormous amount of complexities involved. The arms should always maintain the distance between them as any variation would result in the ball falling. The delay between the time a command is sent to the time it is physically executed is significant since any time difference between the execution of the two arms results in a failed action. More and more bi-manual related research is being conducted over the past few years. Dual-armed robots like Baxter from Rethink Robotics has assisted this development as researchers are not inclined towards building their own robots.

The task of learning a robot skill is a very complex and the number of dimensions is very high. The task of making an arm move in a simple way is also complex because of the multiple joints involved. The operator needs to be able to define the point at which a joint starts moving precisely. As the number of joints increase, the task of specifying these parameters becomes more cumbersome. Hence recognizing the need for a simpler way of performing these tasks, researchers developed a technique called Imitation Learning. Imitation learning is a technique in which the robot operator moves the robot arm in a task-specific way, and the joint angles are recorded. The recorded joint angles from a trajectory which now represents the precise movement demonstrated by the operator so that it can be repeated in the

future. Imitation learning is also known as Kinesthetic Learning. Even though the motion can now be replayed, it is a very naive approach; the robots need to exhibit more intelligence in accordance with different situations. This need brings us to the topic of learning motor skills that are capable of taking into account the dynamics of the robot and its environment. Motor skills require more than just the recorded trajectory which is primitively the kinematic plan. Motor skills can be represented as a motor primitive that is capable of adapting a single movement to different situations. For example, a robot that has been demonstrated to throw the ball in X direction will now be able to generalize the motion using motor skills and then be able to throw in a different direction. Thus, motor skills provide a generic framework for learning actions for varied tasks from sports to industrial work in a generalized fashion.

Chapter 2

BACKGROUND INFORMATION

This chapter presents work that is described in the literature related to imitation learning, dynamic motor primitives, and Reinforcement learning. This chapter starts in section 2.1 and explains the principle behind imitation learning. Subsequently, in Section 2.2 the theory behind dynamic motor primitives is reviewed. Finally, in Section 2.3 Reinforcement learning for robotics is elaborated on.

2.1 Imitation Learning

Imitation learning has been described in the literature using several different terms like Programming by Example, Programming by Demonstration, Learning from Demonstration and Imitation Learning. Humans and other biological organisms have been a source of inspiration for roboticists over the years when developing complex robotic systems. Researchers across the world are trying their best to create machines that can replicate capabilities that are similar to their biological counterparts. Even though we can make an argument that certain elements that are needed to accomplish this objective like materials, power supply, sensors are currently unavailable. But even if we could build such a robotic system, we would still find it difficult to replicate its natural variant. Currently, the algorithms and programs that are capable of commanding a system to perform such maneuvers and tasks are not yet available.

Movement by imitation is an essential behavior demonstrated by humans. We exploit this behavior continuously through our entire life since our birth, when we are trying to

learn a new skill. You can consider the time you learned to flip a coin by looking at your friend doing it. By visually looking at the demonstration we pick up the necessary movements for the task and approximately repeat it again. Even the simplest task of picking up an object or flipping a coin involves the movement of a vast number of muscles, and there is also the task of appropriate co-ordination between these movements. This simplest task itself requires a great many variables that lead to an explosion of the number states that need to be accounted for if you are programmatically considering to achieve the same result. Therefore using a demonstration as the starting point to learning a new skill drastically reduces the state space that needs to be explored and also increases the speed of learning of the task by a huge degree. From the viewpoint of computational motor control, learning from demonstration is a highly complex problem that requires mapping a perceived action that is given in an external (world) coordinate frame into an entirely different internal frame of reference to activate motoneurons and subsequently the muscles. Recent work in behavioral neuroscience has shown that there are specialized neurons ("mirror neurons") in the frontal cortex of primates that seem to be the interface between perceived movement and generated movement, i.e., these neurons fire very selectively when a particular movement is shown to the primate, but also when the primate itself executes the movement. Imaging studies with humans confirmed the validity of these results Schaal 2016.

Recorded data plays a key role in imitation learning. The data here can in the form of positional data of the joints in radians, velocity, force, torque information. A simple example would be to for the teacher to physically hold an arm and record the data while the arm is moved in the required pattern. The recorded data can be replayed on the arm to reproduce the motion that was learned. The result is that the motion cannot be generalized, the above example can serve as the use case for an industrial robot performing a repeated task for its entire lifecycle. Even tiny variations in the environment and work will lead to a

failure. The data should be represented in a generic form so that it can be applied to similar but different conditions. Several research projects have tried to explore imitation learning as an approach to teaching robots new techniques and below are a few methods used to achieve this goal.

2.1.1 Directly Learning the Control Policy

Direct policy learning uses supervised learning to learn a control policy to control the robot Argall et al. 2009. During the demonstration process, the state x and the action u of the teacher is recorded. Consider the example of a human showing the task of throwing a ball, here we do not have knowledge about the commands sent to the different muscles and hence we represent the movement of the various joints in coordinate frames, e.g., the velocity of the wrist and angle of release of the ball. This approach is called Task Level Imitation Schaal 1999. This method needs knowledge of how the end effector(wrist) velocity can be converted to appropriate commands that need to be sent to the motors. In this approach the robot is not aware of the result that it needs to achieve in the task, it just tries to imitate the teacher.

2.1.2 Learning from Demonstrated Trajectories

A second approach is through recording the demonstrated trajectories (e.g. joint angles of an arm) and learning policies from them. This method has been successfully shown in the Ball in the Cup experiment which has a lot of variances and is challenging even for a skilled human demonstrator Kober and Peters 2009. Additional information about the goal of the task can be exploited by using the recorded data and optimizing the trajectory

to achieve the target. The recorded data can be can be stored in the raw form or can be used to extract vital information known as movement primitives. The primitives can be represented using Gaussian Mixture Models Calinon et al. 2010 or using Hidden Markov Models Billard et al. 2008.

2.1.3 Model Based Learning

A model based approach for learning the dynamics of the task is demonstrated in Atkeson and Schaal 1997. A predictive forward model is used to teach a robot arm the task of swinging up a pendulum Atkeson and Schaal 1997. P. Englert et al. Englert et al. 2013 proposed a similar model based technique to find the policies. A probabilistic forward model is learned and represented using Gaussian processes. The model is used to generate the robot trajectory, and the policy is discovered such that recorded path and the predicted trajectory match each other and Kullback-Leibler (KL) divergence is used as a measure to find the similarity between the two. A goal based imitation learning technique is proposed in Chung et al. n.d. that tries to solve the problem of humans and robots having different actuator architectures. It showcases a framework wherein the robot attempts to learn a probabilistic model to infer the goal from the demonstration and uses the self-exploratory probabilistic action model of itself to generate an action to achieve the target.

2.2 Dynamic Motor Primitives

Every action is made up of multiple primitives. Movements also can be considered to be built using primitives called movement primitives. These primitives are considered as one of the foundations for imitation learning. Robot learning systems strive to identify the

building blocks of complex actions which can be for example units of actions, basic behaviors, motor schemas, etc Schaal 2006. Schaal proposed a non-linear framework in Schaal 2006 to identify and manipulate these primitives for both robotic and biological motor control and was called Dynamic Motor Primitives (DMP). DMPs are capable of adapting the existing recorded primitives of complex movements to dynamically stochastic changing environments. Two sets of DMP techniques were proposed: Rhythmic primitives for movements like locomotion that utilize limit cycles and Discrete primitives for throwing, writing which is based on point attractor system.

DMPs are defined by two systems: a canonical system and a transformation system. Point attractor dynamic system represents the transformation system, this is nothing but a spring damper or a PD control signal that tries to attract the system towards the target.

$$\tau\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) \tag{2.1}$$

where y is state of the system, g is the goal, α and β are positive constants and τ is the time constant. A non-linear forcing function is introduced to the above simple dynamic system, and this transforms the above-unforced system to a non-linear system.

$$\tau\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f \tag{2.2}$$

Secondly, a Canonical system that models the generic behavior of both discrete and rhythmic system is also introduced. This system is a first order linear system.

$$\tau\dot{x} = -\alpha_x x \tag{2.3}$$

where a_x is a constant. The value of x is set to 1 which indicates that the system is in the initial state and it monotonically decreases over time to 0, when the system has reached its final target state.

$$f(x) = \frac{\Sigma_{i=1}^{N} \psi_i(x) w_i}{\Sigma_{i=1}^{N} \psi_i(x)} x(g - y_0) \tag{2.4}$$

$$\psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right) \tag{2.5}$$

where y_0 is the initial state of the system, w_i is the weights for the basis functions ψ_i, σ_i and c_i are constants that define the width and centers of the basis functions. Therefore the forcing function f is the summation of the Gaussians multiplied by the weights that are activated as the canonical system decays to 0. The weighted sum is normalized and multiplied by $x(g - y_0)$ that serves both as spatial scaling and the diminishing factor.

For Rhythmic movement primitives the canonical system is a phase oscillator and the forcing function has a slightly modified form.

$$\tau\dot{\phi} = 1 \tag{2.6}$$

$$f(\phi, r) = \frac{\Sigma_{i=1}^{N} \psi_i w_i}{\Sigma_{i=1}^{N} \psi_i} r \tag{2.7}$$

$$\psi_i = \exp(h_i(cos(\phi - c_i) - 1)) \tag{2.8}$$

Discrete DMPs forms one of the principal concepts used in this report, and they can easily be illustrated using an example. Figure 1 shows the original trajectory that is given as the input to the DMP mechanism. To learn the path given to the DMP, we need to find the set of weights w_i of the non-linear forcing function f. The parameters are learned using locally weighted regression Ijspeert et al. 2013. LWRs are chosen because of their capability to learn rapidly in a single shot, and the individual kernels can learn independently. In place of LWRs, other function approximators like mixture model and a Gaussian process

can be used too. To trace the trajectory y_d, we differentiate to obtain the velocity $\dot{y_d}$ and the acceleration $\ddot{y_d}$ and we get the equation for the forcing function f_target so that trajectory matches the desired one.

$$f_target = \tau^2 \ddot{y_d} - \alpha_z(\beta_z(g - y_d) - \tau \dot{y_d}) \tag{2.9}$$

Now we are presented with a weighted linear regression problem, and for each kernel ψ_i, its corresponding weight w_i is obtained by minimizing the locally weighted quadratic error criterion.

$$J_i = \Sigma_{t=1}^{P} \psi_i(t)(f_target(t) - w_i(x(t)(g - y_0)))^2 \tag{2.10}$$

And the solution is

$$w_i = \frac{s^T \mathbf{\Gamma}_i f_d}{s^T \mathbf{\Gamma}_i s} \tag{2.11}$$

where

$$s = \begin{pmatrix} x_{t_0}(g - y_0) \\ \vdots \\ x_{t_N}(g - y_0) \end{pmatrix}, \mathbf{\Gamma}_i = \begin{pmatrix} \psi_i(1) & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \psi_i(P) \end{pmatrix} \tag{2.12}$$

Figure 2 shows the trajectory in figure 1 reproduced using different number of basis functions. We can hence come to a simple conclusion that higher number of basis functions provide better reproduction but during the process of optimization of the weights in the learning process, this increased number of weights leads to an explosion in the number of dimensions. Therefore the number of basis functions is problem specific. Figure 3 shows a simpler trajectory and DMP is used with ten basis functions to reproduce the trajectory by changing the goal positions. It can be observed here that DMP has scaled the original trajectory to generate a path similar to the input. Figure 4 shows the location of the basis

functions. Figure 5 shows the weights of the basis functions for the trace from Figure 3 with goal value of 3. Figure 6 shows the forcing function and summation of the weights multiplied by the basis functions.

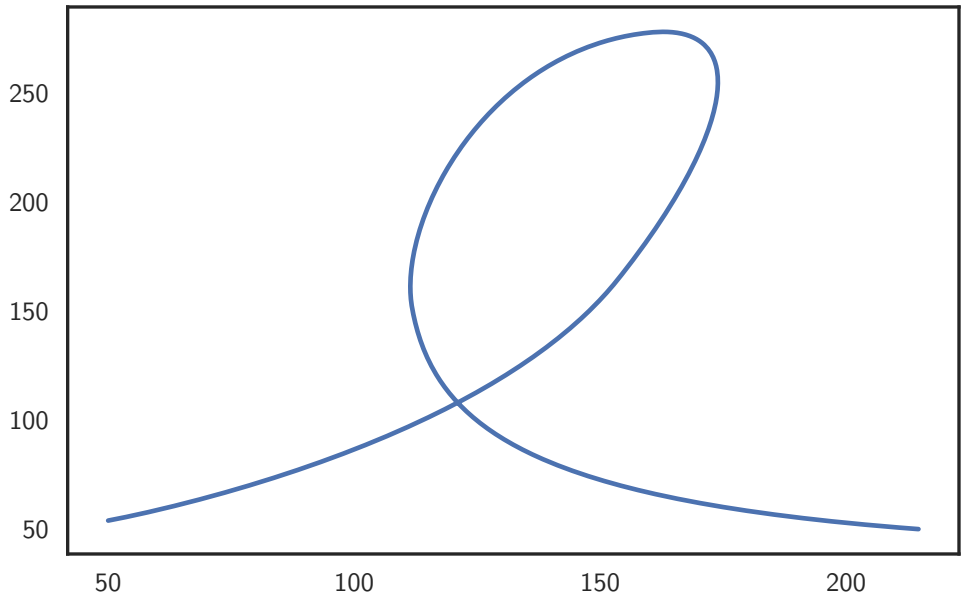


Figure 1: Recorded Trajectory

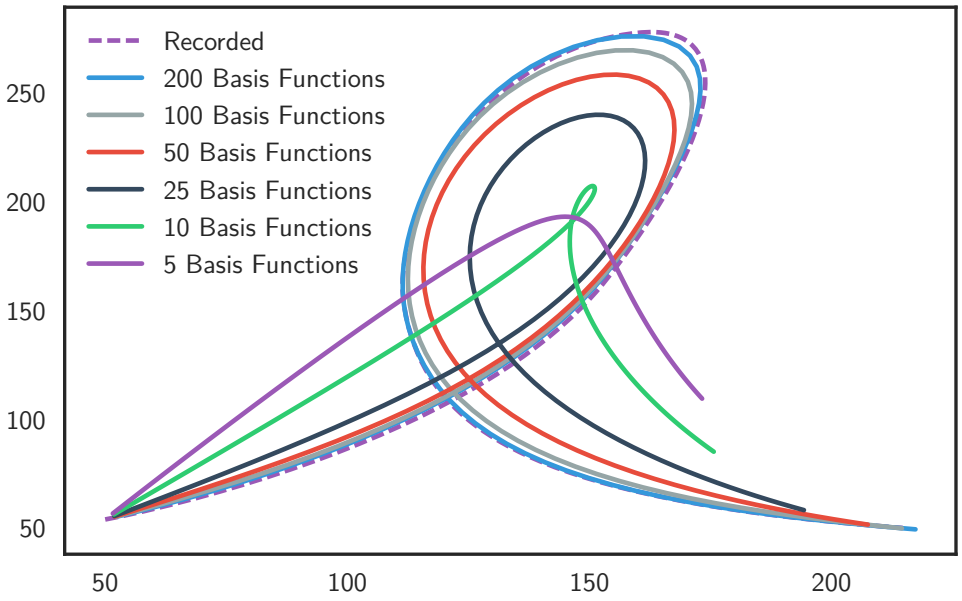


Figure 2: Trajectory reproduced using DMP with different number of basis functions

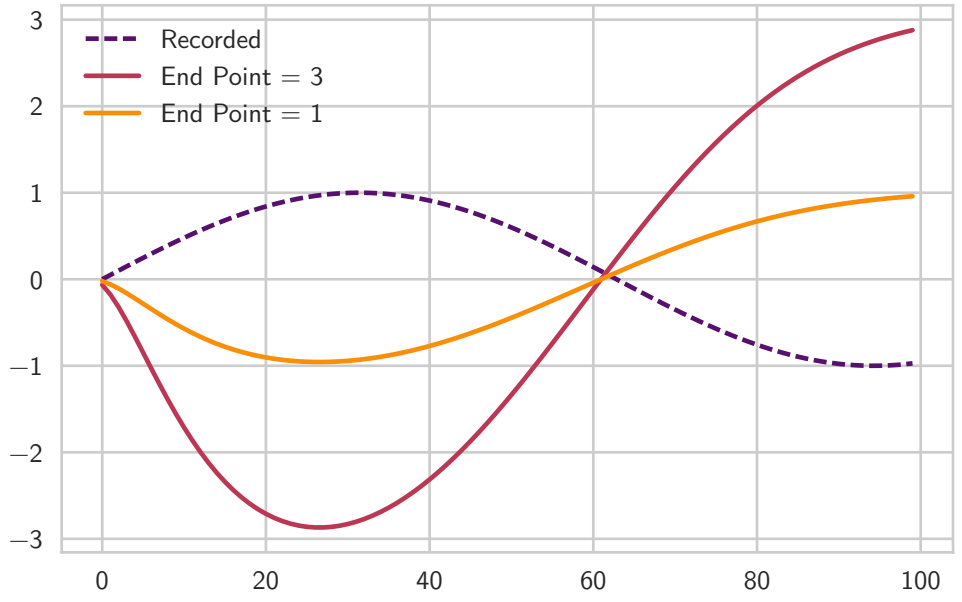


Figure 3: Trajectory reproduced using to DMP for different end positions

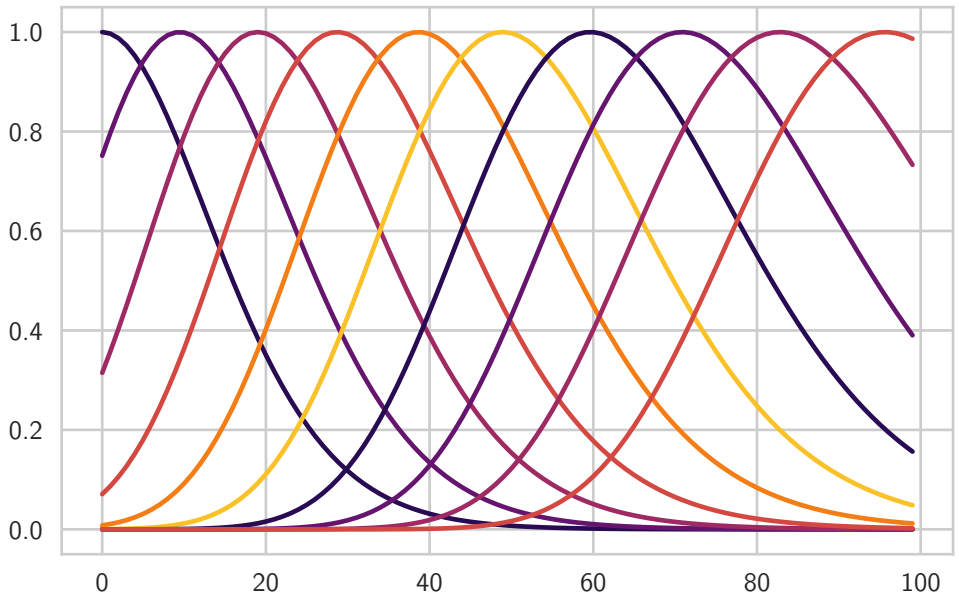


Figure 4: Basis functions used by DMP

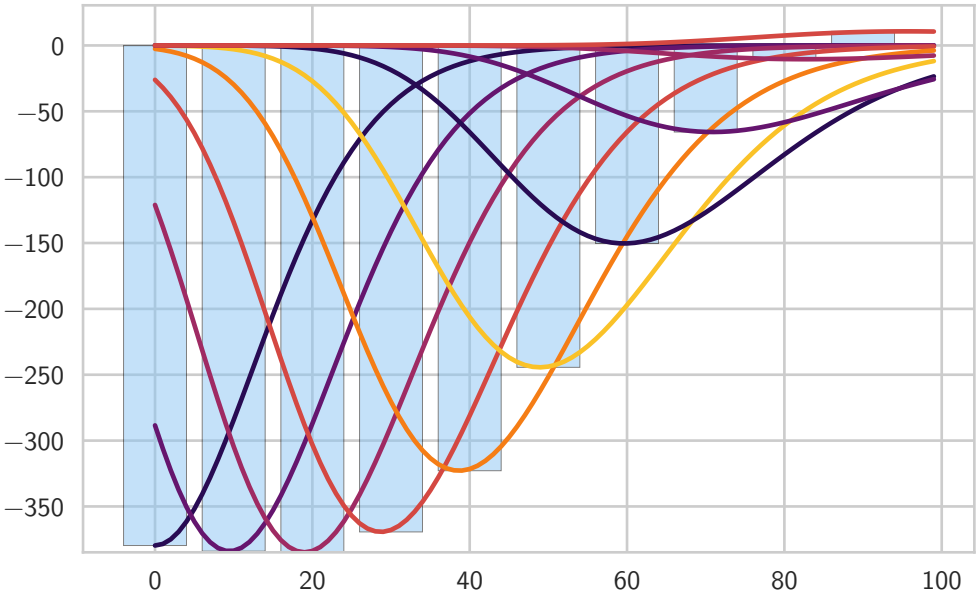


Figure 5: Weights of the basis functions

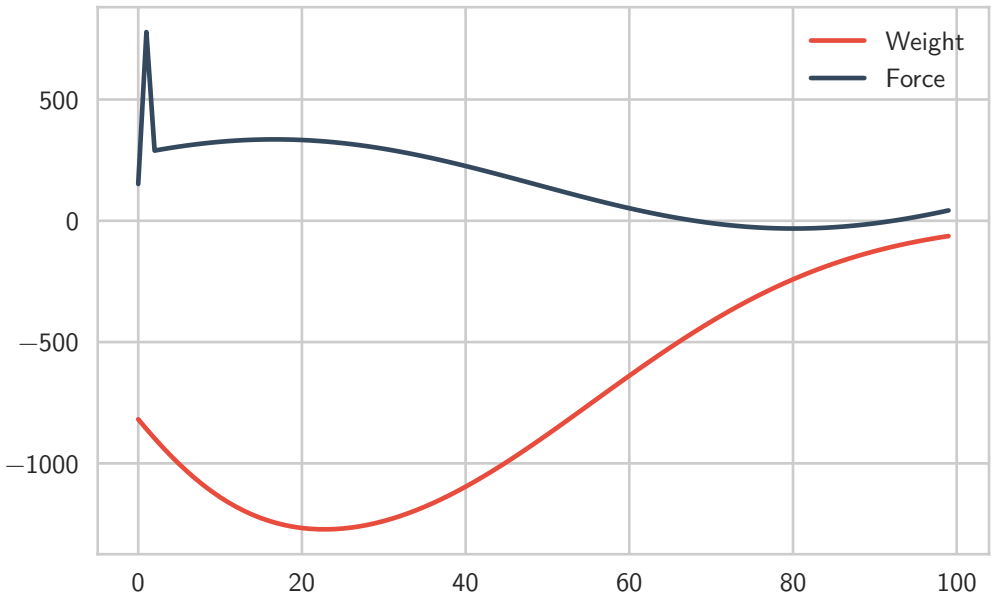


Figure 6: Weight and force parameter values

2.3 Reinforcement Learning

Reinforcement learning is a machine learning technique where the learner tries to map different situations to actions, the goal of the learner is to maximize or minimize a certain reward signal. It can be described as a trial and error learning scheme. The learning agent needs to self-discover the action that generates the best reward over time. A greedy approach may provide the best value now but over time may considerably degrade the reward. This condition brings one of the challenges of reinforcement learning that is the trade-off between exploration and exploitation. The learning agent must exploit previous actions that have generated good rewards, but it also has to explore new actions to discover them so it can exploit them in the future. Thus arises the problem of balancing exploration and exploitation of actions.

Reinforcement learning model has a few key elements.

- Agent: The agent is responsible for generating the actions a_t that is sent to the actor. The agent then receives the reward r_t from the actor and is also responsible for maintaining the mapping between the actions and rewards and holds the policy used to generate actions.
- Actor: The actor performs the received action and returns a reward value to the agent.
- Reward: The reward is a numerical value that defines the value associate with an action (e.g. achieved distance by the ball for ball throwing experiment).
- State: The state s_t of the actor before performing the action and state s_{t+1} after can also be given as inputs to the agent.

2.3.1 Reinforcement Learning in Robotics

Reinforcement learning in robotics is similar to its counterpart, but it introduces additional complexities. RL can help robots learn new skills without human demonstration and learn to adapt an ability to new situations Kober and Peters 2012. Modern robots are high-dimensional and represented by continuous states and actions which result in a huge number of variables to optimize. Physical robots are noisy, and thus they introduce uncertainty into the different states which need to be eliminated by using different filters. The rewards obtained by executing a particular action are also noisy and can be difficult to achieve the same reward when it is repeated. Besides these factors, real-world executions are time-consuming and costly. Also using learned simulation models is hard as small errors in the model amplify the error Kober and Peters 2012. The algorithms should also converge faster to reduce the effort and wear and tear of the hardware.

The goal of reinforcement learning is to find a policy π that picks an action a that maximizes the cumulative reward. Hence, the agent has to find a mapping between the actions, states, and the reward. The reward functions can be dependent on the current state $R = R(s)$, current state and the action $R = R(s, a)$ or current state, action and the transition $R = R(s', a, s)$. RL strives to find a policy π^* which maximizes the average return $J(\pi)$

$$J(\pi) = \Sigma_{s,a}\mu^{\pi}(s)\pi(s, a)R(s, a) \tag{2.13}$$

where μ^{π} is the stationary state distribution generated by policy π acting in the environment. Optimizing in the primal formulation is called policy search in RL while optimizing in the dual formulation is known as a value function-based approach Kober and Peters 2012.

Cost function methods approximate the Lagrangian multipliers $V^*(s)$, also called the value function, and use it to reconstruct the optimal policy. Monte-Carlo methods, Q-learning and SARSA are a few examples of value function methods. Problems in Robotics are high dimensional, and cost function based approaches fail under these conditions as they need function approximation for the value function. Cost functions require total coverage of the state space and a small change in the value function causes a substantial shift in the policy and can lead to dangerous decisions in robotic systems.

Policy search is reinforcement technique commonly used in robotics Kober and Peters 2012. These methods perform local optimization around existing policies π parametrized by a set of policy parameters θ_i and the policy is updated by changing the parameter by $\Delta\theta_i$ that results in an increase of the reward.

$$\theta_{i+1} = \theta_i + \Delta\theta_i \tag{2.14}$$

The policy search can be classified as black-box and white-box methods. Black box methods are stochastic optimization algorithms and they only consider the returned reward value and are transparent to the problem in hand. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is one such algorithm that is used in the experiments.

2.4 Covariance Matrix Adaptation Evolution Strategy

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is an evolutionary algorithm that was proposed by Hansen Hansen 2006. CMA-ES is stochastic evolutionary, black box algorithm that is used to optimizing non-linear or non-convex continuous functions. As is the case in an evolutionary algorithm, it works similarly to biological evolution, for every generation samples(children) are generated in a stochastic way by using the cur-

rent samples(parents). The generated samples (children) becomes the parents for the next generation based on their fitness values which are evaluated by using the objective function $f(x)$.

CMA-ES can be computed using the following steps.

2.4.1 Sampling

At the start of each generation, a new set of points that need to be evaluated using the objective function is generated by sampling a multivariate normal distribution $\mathbb{R}^n$. The parameters that define the distribution are updated at the end of every generation based on the fitness values obtained.

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)}\mathcal{N}(0, C^{(g)}) \quad \text{for } k = 1 \dots \lambda \tag{2.15}$$

where

$\mathcal{N}(0, C^{(g)})$ is a multivariate normal distribution with zero mean and covariance matrix $C^{(g)}$

λ is the population size.

$m(g) \in \mathbb{R}^n$ is the mean of the distribution at generation g.

$\sigma^{(g)} \in \mathbb{R} > 0$ is the standard deviation at generation g.

2.4.2 Selection

The samples x_k obtained from the previous step are evaluated, and the fitness values are obtained. The new mean of the distribution is calculated using the weighted average of μ selected points from the sample.

$$m^{(g+1)} = \Sigma_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)} \tag{2.16}$$

where

$\mu < \lambda$ is the number of selected points.

w_i are the positive weight coefficients for recombination.

2.4.3 Covariance Matrix

A covariance matrix along with the mean can be uniquely used to define a multivariate normal distribution. The covariance matrix is updated using the μ best samples from the current generation to get the new distribution. CMA-ES uses rank-one and rank-μ estimators to update the covariance matrix to overcome the problems associated with small and large population sizes.

$$C_{\mu}^{(g+1)} = \Sigma_{i}^{\mu} w_i (x_{i:\lambda}^{(g+1)} - m^{(g)})(x_{i:\lambda}^{(g+1)} - m^{(g)})^T \tag{2.17}$$

2.4.4 Example

Rosenbrock function also known as Rosenbrock's banana function Rosenbrock 1960 is a non-convex function that is used to test optimization algorithms. The global minimum for the function is in a narrow parabolic valley. The function is defined by

$$f(x, y) = (a - x)^2 + b(y - x^2)^2 \quad \text{Usually } a = 1, b = 100 \tag{2.18}$$

The global minimum of the function $f(x, y) = 0$ is at $(x, y) = (a, a^2)$.

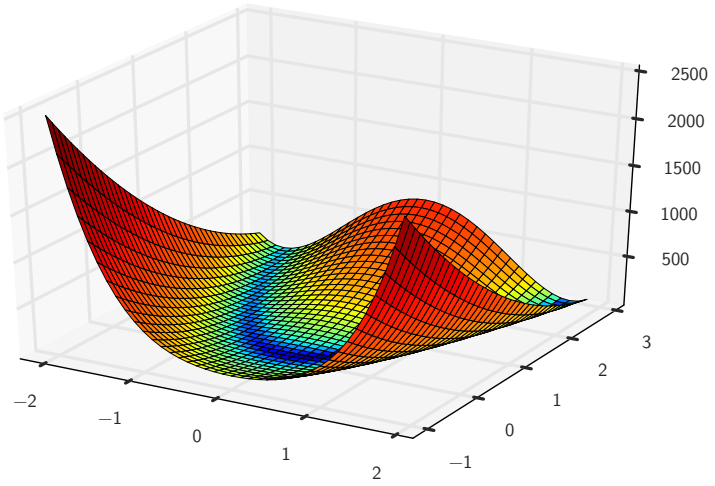


Figure 7: Plot of the Rosenbrock function of two variables

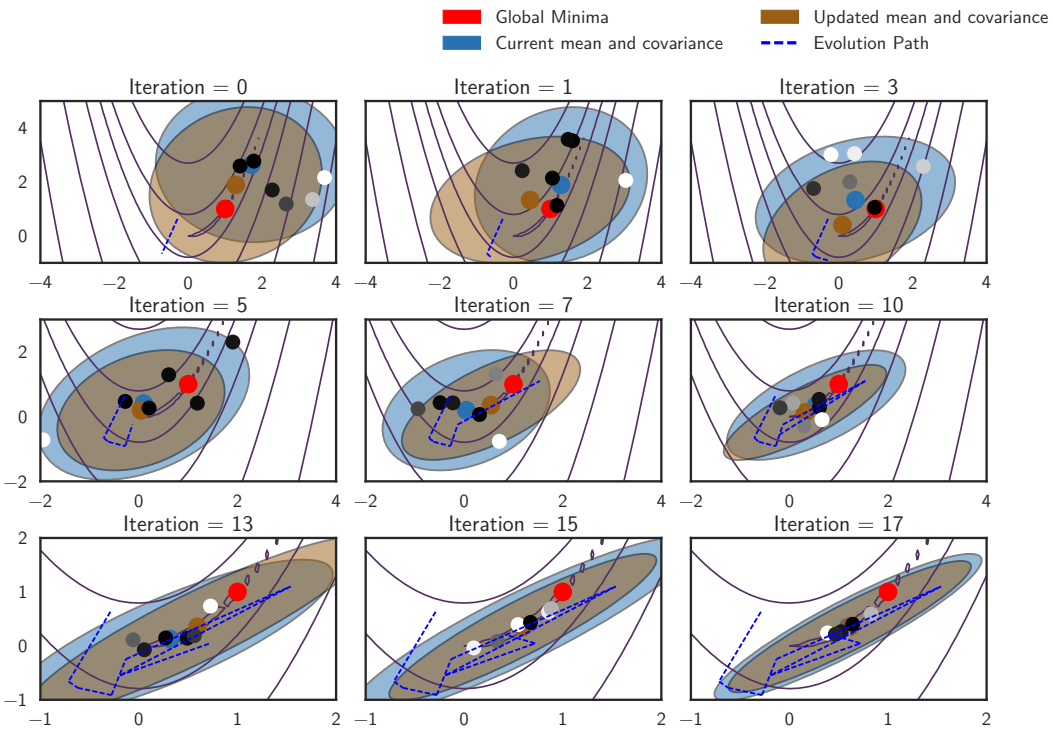


Figure 8: Optimization run of CMA-ES for two dimensional Rosenbrock function

Chapter 3

ROBOT CONSTRUCTION

A general purpose dual armed robot capable of handling the objects bi-manually was needed for the task that needed to be performed. Most of the popular robots currently used in the industry like UR5, Motoman, are single armed. The most popular dual armed robot that is extensively used in research is Baxter from Rethink Robotics. Baxter can be used to perform different types of tasks like manipulating objects, but it does not have the acceleration (jerk) which is the most important behavior needed for the task that was designated to be performed by the robot. The above requirements resulted in a need to build a new dual armed robot.

3.1 Arms

Manipulator-H is a multi-purpose low-cost manipulator that is offered by Robotis. Manipulator-H is a 6 DOF manipulator arm. The arms are built using a combination of 6 Dynamixel Pro servos. Dynamixel servos from Robotis have been one of the most popular servos constantly used in different research projects. Dynamixel Pro servos are the next generation of servos developed in Dynamixel servo lines that provide better precision and higher power compared to their older counterpart and they house new gear reduction systems. The arms are manufactured in different configurations, and the H variant was chosen because it has the highest torque and acceleration among the variants. The position and the orientation of the various joints of the arm can be seen in Figure 9. The arm has six

servos, and three different servo variants which have different power and precision values are used in the construction.

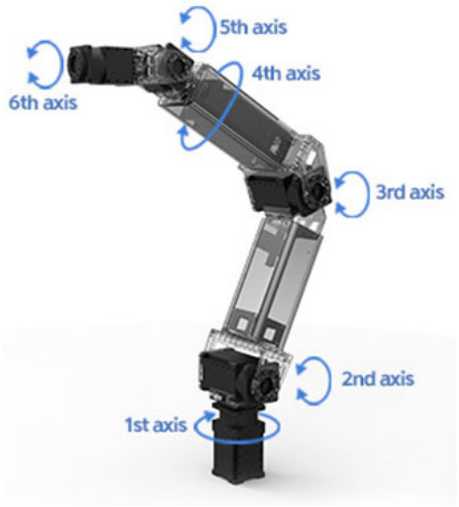


Figure 9: Manipulator-H Arm

Source: Robotis

The specifications of the arm are listed in Table 1

Table 1: Specifications of the arm

DOF	6
Payload	3 kg
Speed	180 deg/sec
Weight	5.5 kg
Reach	645 mm
Voltage	24 VDC
Communication	RS-485

The Denavit-Hartenberg parameters for the arm are listed in Table 2

The six joints are named shoulder-pan, shoulder-lift, elbow, wrist-1, wrist-2 and wrist-3. The shoulder-pan and shoulder-lift joints are built using H54-200-S500-R servos. They are rated at 200W and have a precision of 0.0007 degrees. The elbow and wrist-1 joints are

Table 2: D-H Parameters of the arm

Link	Link Length(mm)	Link Twist(rad)	Joint Offset(mm)	Joint Angle(rad)
1	0	$(-\pi/2)$	0	0
2	265.69	0	0	0
3	30	$(-\pi/2)$	0	0
4	0	$(\pi/2)$	258	0
5	0	$(-\pi/2)$	0	0
6	0	$(-\pi/2)$	0	0

constructed using H54-100-S500-R servos which also have an accuracy of 0.0007 degrees but are rated at 100W. H42-20-S300-R servo is used by the last two joints wrist-2 and wrist-3, and they are rated at 20W and have a lower precision of 0.0011 degrees compared to their counterpart. The operating range of the different joints is listed in Table 3.

Table 3: Operating range of the joints

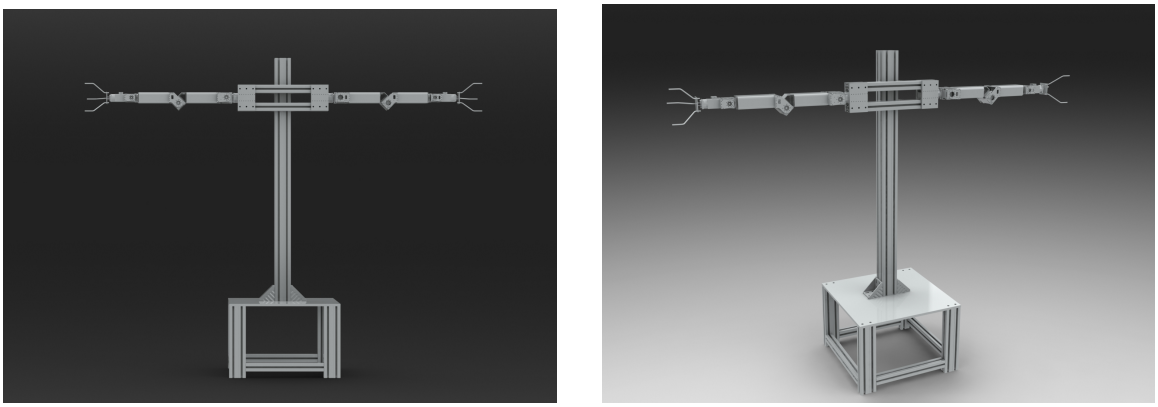
Joint	Range
Shoulder-Pan	$-\pi$ to π
Shoulder-Lift	$-\pi/2$ to $\pi/2$
Elbow	$-\pi/2$ to $3\pi/4$
Wrist-1	$-\pi$ to π
Wrist-2	$-\pi/2$ to $\pi/2$
Wrist-3	$-\pi$ to π

The RS-485 communication protocol is used to talk to the individual servos. The servos are chained together serially such that a single bus can be used to talk to all the servos rather than having multiple buses to communicate to the servos. An FTDI based USB to Serial converter is used to connect the PC to the arm. The operating system exposes this interface as a TTY character device. The arms support multiple baud rates, and 3000000 bps is used as the default baud rate. Each servo internally has a control table that is used to maintain the current status and the operation being performed. The control table can be

divided into two sections: RAM holds volatile data that is reset when the power is turned off and EEPROM that holds non-volatile data that is saved through a power cycle event. The EEPROM area of the table is responsible for data like model number, device address, baud rate, upper and lower limits of position, etc. The RAM region holds PI gain values, current position, velocity and acceleration, goal position, velocity commands. The servo provides feedback about the current position, velocity, acceleration and the torque of the motor. Each servo is assigned a different address value which ensures that host can talk to each servo in the serial chain.

3.2 Dual Armed Robot

Two Manipulator-H arms were used to build a dual armed robot. A mechanical structure was built using Bosch aluminum profiles to support the two arms. The rendered images of the 3d model can be seen in Figure 10. Unactuated grippers were custom designed for the task using multiple metal rods. Two USB to Serial converters were used to connect the host PC to the arms. A 30V bench power supply capable of delivering 30A is used to power the arms. An emergency stop button was also added to cut-off power to the arms to ensure the safety of the operator under dangerous circumstances.



(a) Front View (b) Perspective View

Figure 10: Basketball Robot: 3d Renders

3.3 Software

Robot Operating System (ROS) is a software framework that is extensively used for robot software development. ROS provides lots of tools and libraries that assist in creating robot applications. It provides features like hardware abstraction, device drivers, libraries, visualizers, messaging system, package management. ROS can be described as a network of nodes wherein each node performs a certain functionality. The nodes then communicate with each other using topics. Nodes publish messages to a topic, and any nodes that need this information subscribes to the topic by notifying the master node.

The entire software stack for the robot was built on ROS because of the flexibility, ease of use, libraries and the support provided by the vast open-source community. The low-level driver for the arms is written using a control architecture called ROS-Control. ROS-Control provides a set of tools that can be leveraged to write real-time controllers for the robot hardware. It provides a robot hardware abstraction layer. It exposes a common interface for the robot hardware control software so that ROS control libraries like Joint Trajectory Controller can be reused between robots. An ROS control loop runs continuously at

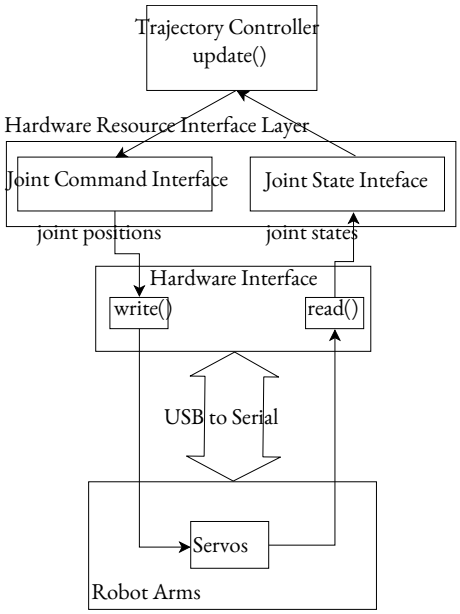
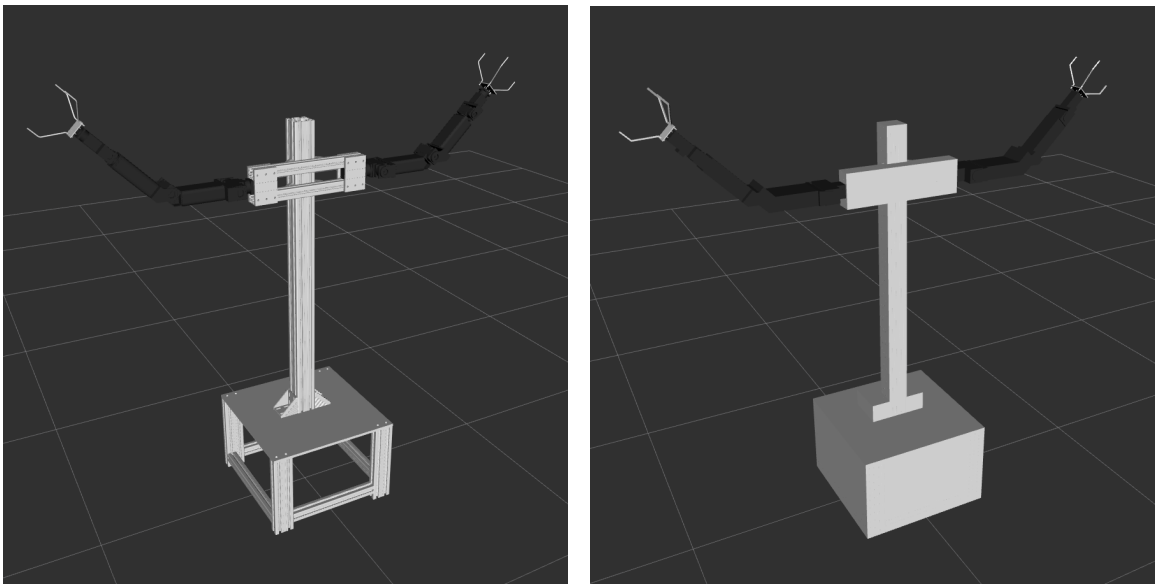


Figure 11: ROS driver control architecture

a specific rate and runs three functions repeatedly. The low-level communication interface to the arms is written using a Linux serial device driver.

- Read(): The necessary information from the arms is read using the usb-to-serial interface exposed using the serial character device. The position, velocity and acceleration data is read, and the controller manager is updated.
- Update(): This function calls the controller manager internal state updater. Controllers like joint trajectory controller, gazebo controller that need the data from the robot register themselves with the controller manager. The controller manager provides the data to the different controller so that controllers can manipulate the data and return the data that needs to be written to the robot hardware.
- Write(): Updated data from the controller manager is retrieved and written to the robot using the same serial interface.

Each arm has a separate control loop that runs at approximately 250 Hz. The arms



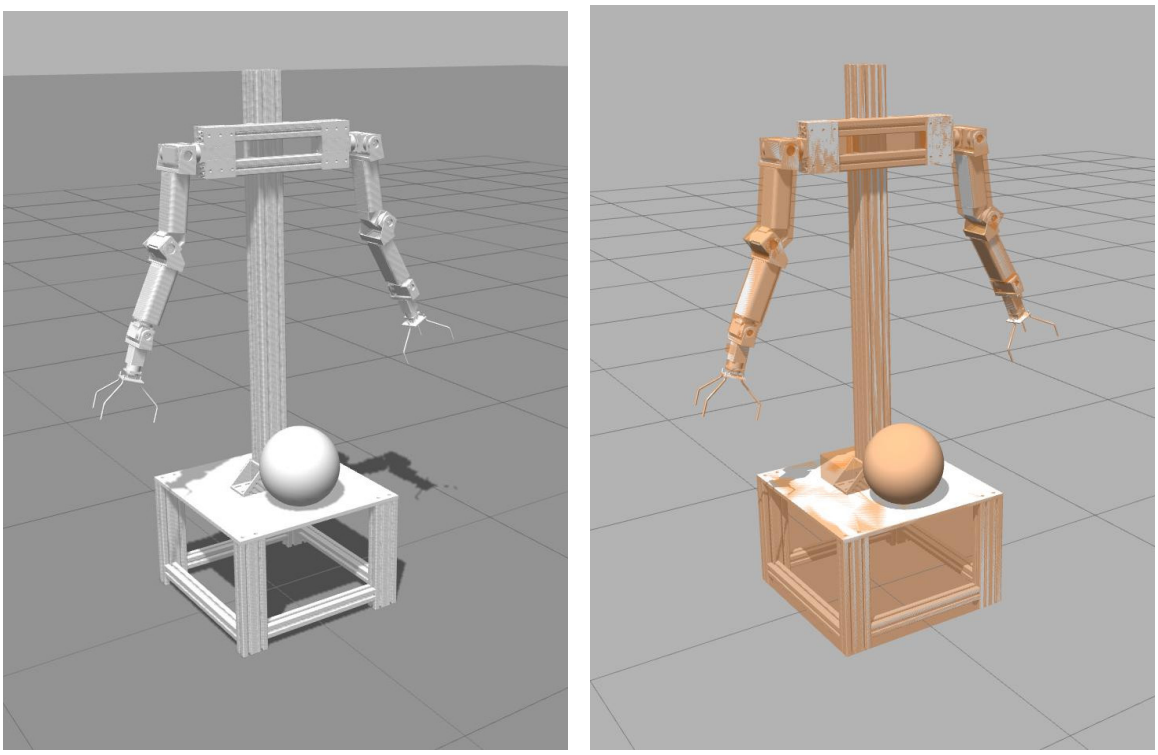
(a) Rviz Robot Visual Model (b) Rviz Robot Collision Model

Figure 12: Basketball Robot: Rviz

can be controlled using different control modes namely position and velocity. Both joint position and velocity based control interfaces are exposed, but position based control is used for this task.

Moveit is the state of the art software that is used for mobile manipulation, kinematics, 3D perception, and navigation. It is widely used for planning and collision detection for robot arms. The reinforcement algorithm performs the planning for the robot arms, but Moveit is utilized in each and every stage to validate the plan that is generated by our algorithm so that there are no collisions between that arms and the environment. A low polygon 3D mesh was generated to increase the speed of the collision detection system. Rviz, the 3D visualization tool for ROS is used to visualize the generated trajectory and validate it before running it on the real hardware.

Robot simulations play a crucial role in reinforcement learning. During the process of teaching a new behavior to a robot, the parameters that are used in the experiment is the key



(a) Gazebo Robot Model (b) Gazebo Collision Model

Figure 13: Basketball Robot: Gazebo

and getting these values right usually involves running the experiment, observing the results and varying the parameters repeatedly. Running the experiments on the real hardware is a costly process both in terms of time as well as robot wear and tear. Thus simulation systems help is drastically cutting down this cost. Simulation models of the robot are usually not accurate, and it is tough to replicate the real robot model exactly, but the real world models can be approximated, and they can help in obtaining a rough estimate of the parameters, and these parameters can then be fine tuned by running on the real robot.

Gazebo is open source robot simulation software that provides high-quality graphics and programmatic and graphical interfaces for various physics engines like ODE, Bullet, Simbody, and DART. The models of the robot in Gazebo can be seen in Figure 13. Gazebo-

ros-control library was used to emulate similar interface as the real hardware, and a PID controller was used to obtain similar behavior as the actual hardware. ODE was used as the physics engine as it is better integrated into gazebo compared to the other physics engines.

Chapter 4

LEARNING TO THROW BY REINFORCEMENT LEARNING

Learning to throw the ball farthest and throwing the ball into a hoop were chosen as the two tasks to be learned using a reinforcement learning technique. The task would primarily involve learning a policy to generate the trajectories(joint angles) for the two arms. The task of throwing a ball using a single arm is a simpler task as the learning agent only needs to maximize the velocity and select the optimum release angle for the ball. However, in dual arm case, a lot number of parameters are introduced. The agent now, along with optimizing the release and the increasing the velocity, it also has to ensure that there is no collision between the arms, the distance between the two end effectors is taken into consideration as any mismatch in the distance will result in the ball being dropped. Besides the learning process, the control software responsible for actuating the two arms should minimize the time delay between the commands being sent to the arms because any loss of synchronization in the movement will not only result in a failed trial but will also corrode the results obtained from the learning algorithm.

4.1 Robot Controller

The commands to the two arms continuously written at a rate of 250 Hz. The trajectory that needs to run on the arms is written using joint trajectory controller. The trajectory is transferred from the learning agent to the controller in the form of multiple waypoints. The plan obtained from the agent is linearly interpolated, and waypoints are generated at intervals of 0.06radians and are inserted with a time interval of 0.019s to ensure the arms

move at the maximum possible velocity. The values of delta and the time interval are chosen according to the Equations 4.1. The controller then performs cubic interpolation between a pair of waypoints and then writes the position data to the arms during the control loop.

$$speed = 3.14 \text{ rad/s} \quad \text{(Max Speed)} \tag{4.1}$$

$$\Delta d = 0.06 \text{ rad} \quad \text{(Distance between waypoints)} \tag{4.2}$$

$$\Delta t = \frac{\Delta d}{speed} = \frac{0.06}{3.14} = 0.019 \text{ s} \tag{4.3}$$

Besides increasing the speed of the movement, the movement between the two arms is also synchronized. The synchronization is performed on a waypoint basis since the distance between two waypoints is a constant value, the controller ensures that the two arms are finished executing the current waypoint before the move to the next waypoint is performed.

4.2 Simuation

A software PID was used to model the arms to the real world approximately and also the same controller is used to control the arms both in simulation as well as in real world. The basketball used in the experiment is a soft ball. The softness of the ball was also modeled, as grasping rigid body results in arms vibrating. The softness of the ball is mainly modeled using the stiffness coefficient(kp) and damping coefficient (kd) parameters provided by the ODE physics engine. ODE uses soft constraint force mixing ($soft\_cfm$) and soft error reduction parameter ($soft\_erp$) values to introduce to softness to a rigid body. However, gazebo uses kp and kd internally to generate the values for the softness parameters according to the equations below.

$$soft\_cfm = \frac{1}{h * kp + kd}$$
$$soft\_erp = \frac{h * kp}{h * kp + kd}$$

The values used for modeling the ball are listed in Table 4.

Table 4: Gazebo ball model parameters

Name	Value
Stiffness coefficient (kp)	$1e9$
Damping coefficient (kd)	1
Max contact correction velocity truncation (max_vel)	0.1
Minimum allowable depth (min_depth)	0.0001
Coefficient of friction (mu)	$1e15$
Second coefficient of friction ($mu2$)	$1e15$
Restitution coefficient	0.01
Bounce velocity threshold	0.01

4.3 Learning and Optimization

CMA-ES is the used as the reinforcement learning algorithm. The agent generates a trajectory from the start to end of the throw. In the case of the real world experiments, the ball is placed at a fixed position on a stand, and once the agent provides the start position of the throw, forward kinematics is used to ensure that the distance between the end effectors is such that the ball can be held firmly by the arms. After this condition has been satisfied, a new trajectory is generated from the fixed ball position to the start position obtained from the agent by linear interpolation. The throwing action is checked for collisions and validity using inverse kinematics before the ball is thrown. However, in the case of simulation, instead of picking the ball from a fixed position, the arms are moved to generated throwing

start position and then the ball is spawned at the center of the two end effectors. The point of collision between the lines from two end effectors is chosen as the ball spawn point. The rest of the process is similar to the real robot case.

The choice of parameters for the CMA-ES play a very key role in the learning process. The parameters need to be selected with care as an excess of parameters will lead to problems like the curse of dimensionality. 54 parameters were used to learn the basketball throwing motion and the parameters are listed below

- Time interval between the start of shoulder lift joint and elbow joint. (1)
- Time interval between the start of shoulder lift joint and wrist-1 joint. $(1 + 1 = 2)$
- Constant shoulder pan joint angle. $(2 + 1 = 3)$
- Start and end angle of shoulder lift joint. $(3 + 2 = 5)$
- Start and end angle of elbow joint. $(5 + 2 = 7)$
- Start and end angle of wrist-1 joint. $(7 + 2 = 9)$
- Learnt DMP parameters for shoulder lift, elbow and wrist-1 joint. $(9 + 15 * 3 = 54)$

The second key factor in reinforcement learning is the reward function. The value returned by the reward function is used for optimizing the parameters over time. Two sets of experiments were conducted, throwing the ball farthest and throwing the ball into a hoop. The algorithm behind the objective function is described below

$$position = [(x_{t=0}, z_{t=1}), (x_{t=0}, z_{t=1}), \dots (x_{t=0}, z_{t=1})$$

- Maximizing throw distance.
- Throwing ball into a hoop.

4.4 Vision

CMA-ES is a black box optimization algorithm and hence the objective function is the key part of the system. Hence tracking the ball for the experiment is done using Microsoft Kinect for Xbox One. Kinect has a high-resolution color camera with a resolution of 1920 x 1080 pixels. It also has a depth image sensor of resolution of 512 x 424 pixels. The depth sensor has a range of 0.5 to 4.5 meters. It has a depth resolution of 1.5 mm @ 50 cm and increases to 5cm @ 5m.

Interaction with kinect sensor is done using a python library called pylibfreenect2. OpenCV is then used to manipulate the images obtained from the sensor. The ball is tracked using its color, and hence a bright yellow colored ball is used in the experiments. The algorithm used for ball tracking is described in

Algorithm 1 Ball Tracking algorithm

```
1: procedure BallPosition
2:     color_image ← get_kinect_color_image
3:     depth_image ← get_kinect_depth_image
4:     hsv ← hsv_filter_image(color_image, ball_hsv_color)
5:     contours ← find_contours(hsv)
6:     ball_contour ← max(contours)
7:     return average(get_depth_for_contour(depth_image, ball_contour))
```

4.5 DMP

DMP can be used to reproduce a path using basis functions as described in the previous chapters. DMP is used to regenerate trajectories related to three different joints and increasing the number of basis functions will increase the complexity of finding a solution using CMA-ES. Thus the number of basis functions needs to be carefully chosen. 15 basis func-

tions are used for reproducing the trajectory of each joint. It was observed that the number chosen gives a good approximation of the trajectory without much loss of information.

Chapter 5

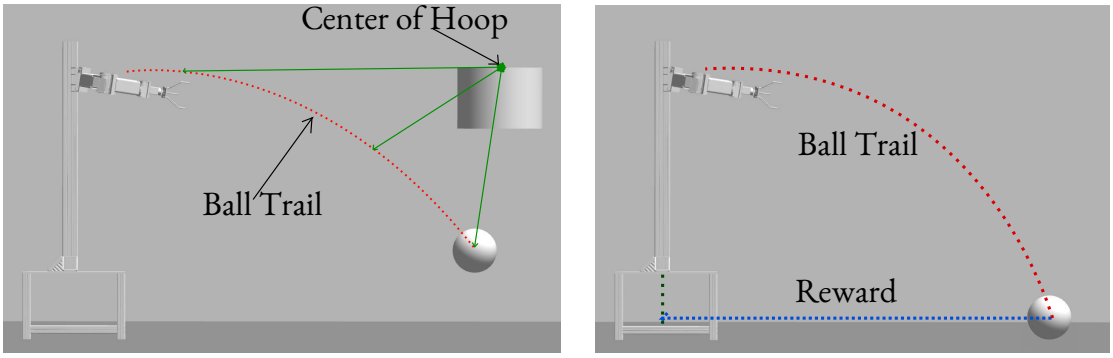
EXPERIMENT AND RESULTS

Two sets of experiments (maximizing the distance of ball thrown and throwing the ball into a hoop) were conducted in simulation using Gazebo. On the real robot, throwing the ball into a hoop experiment was run.

5.1 Experimental Setup

The experimental setup for simulation for the simulation can be seen in Figure 14. A yellow colored fluffy ball of radius 0.13 m and a weight of 200 grams was used for the robot experiments. According to the NBA regulations, the radius of the hoop is twice the radius of the ball. To increase the difficulty of the task, a hoop of radius 0.16 m was placed at a distance of 1.5m from the robot at the height of 1.7 m from the floor whereas the arms are at the height of 1.6 m. For the simulation experiments, a ball of radius 0.13 m and weight 250 grams was used, and the hoop had a radius of 0.25 m.

The trail of the ball during the throw was stored and used as the reference for calculating the rewards in both sets of experiments. In distance maximization experiment, the distance from the arms to the point where the ball hit the ground was used. The distance from the center of the hoop to the position of the ball was calculated for every sample in the ball trail, and minimum of the distance was used as the reward for the throwing into the hoop experiment. A Xbox one Kinect camera was used to track the ball in real-time and generate the rewards for the real robot experiments.

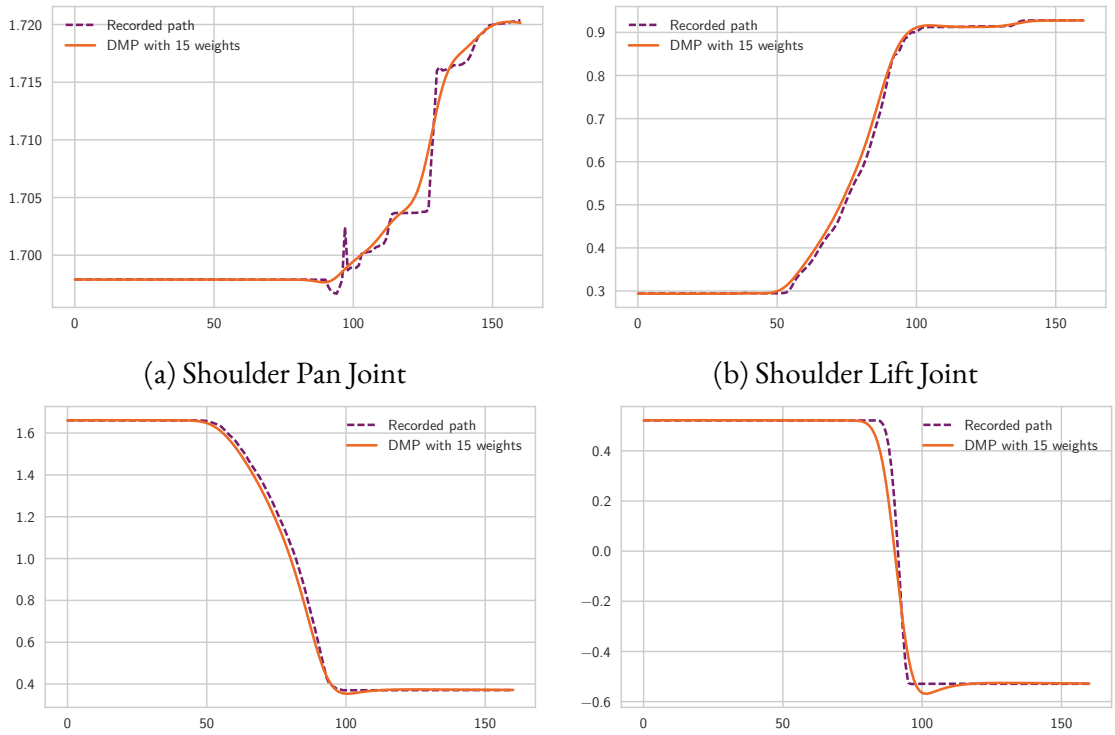


(a) Hoop experiment (b) Distance maximization experiment

Figure 14: Simulation Experiment Setup

5.2 Training Data

The initial set of data that is needed to initialize CMA-ES was recorded by two individuals holding the two arms and attempting to throw the ball. The joint angles during the throw were recorded and used to train the initial set of DMP values. The recorded trajectory can be seen in Figure 15. The same recorded data is used for the all the experiments performed both in simulation and also on the real robot. Fifteen basis functions were used to reproduce the recorded training data.



(a) Shoulder Pan Joint

(b) Shoulder Lift Joint

(c) Elbow Joint

(d) Wrist-1 Joint

Figure 15: Recorded Joint Trajectories reproduced with DMPs

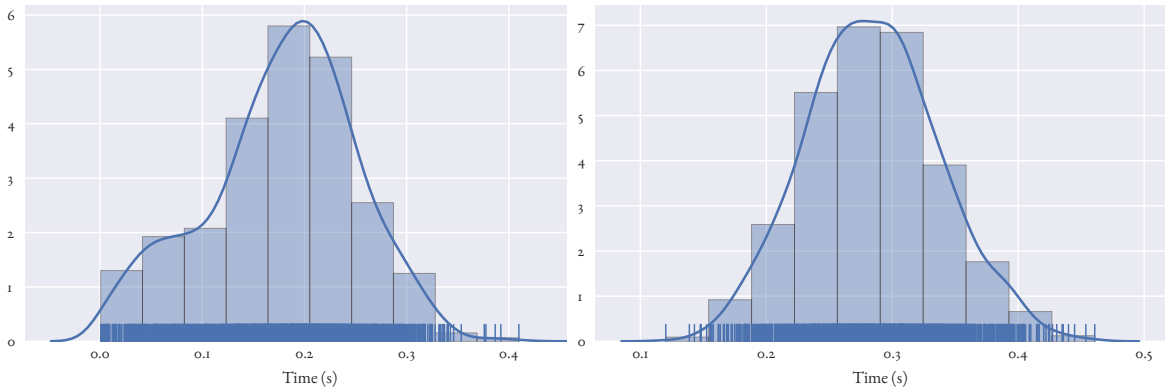
5.3 Simulation

The results obtained from the two experiments conducted on the simulation platform are show below.

5.3.1 Simulation: Distance Maximization

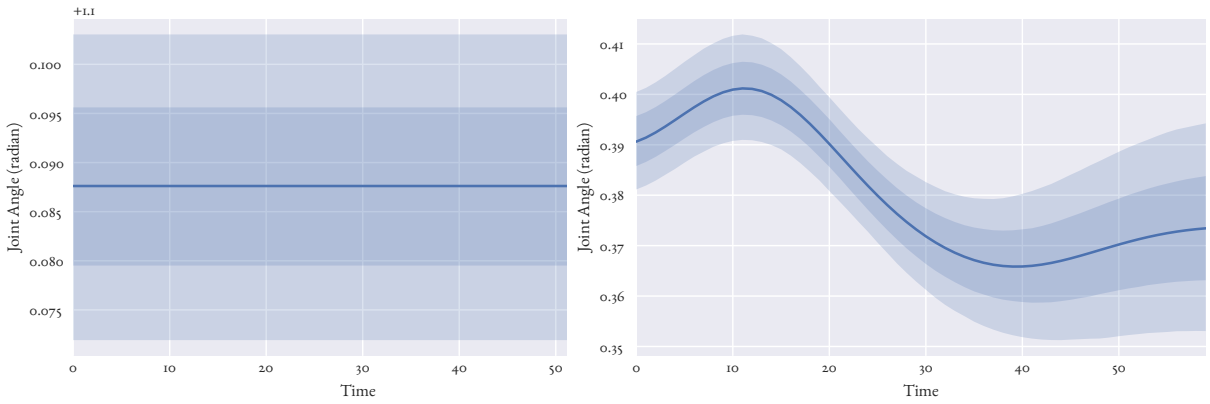
In this experiment, the setup as described in the previous section was used in Gazebo simulation framework to learn a new set of arm movements to maximize the distance to which the robot can throw the ball. CMA-ES was initialized with the previously recorded training data and a sigma value of 0.85. The experiment was executed for 130 iterations. The

fitness plots in figure 19 show the distance traveled by the ball. At the end of the experiment, the robot had learned a new policy to throw the ball to a distance of 720cm. Figure 16 shows the time delay values and the trajectories generated by CMA-ES for the entire 130 iterations that were executed. Kernel density plots as a function of the different joint angle combinations are showcased in Figure 18 .



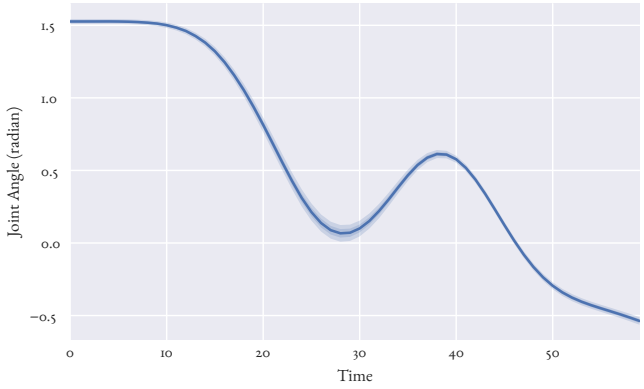
(a) Elbow joint start delay (b) Wrist joint start delay

Figure 16: Simulation distance maximization: Joint start delay



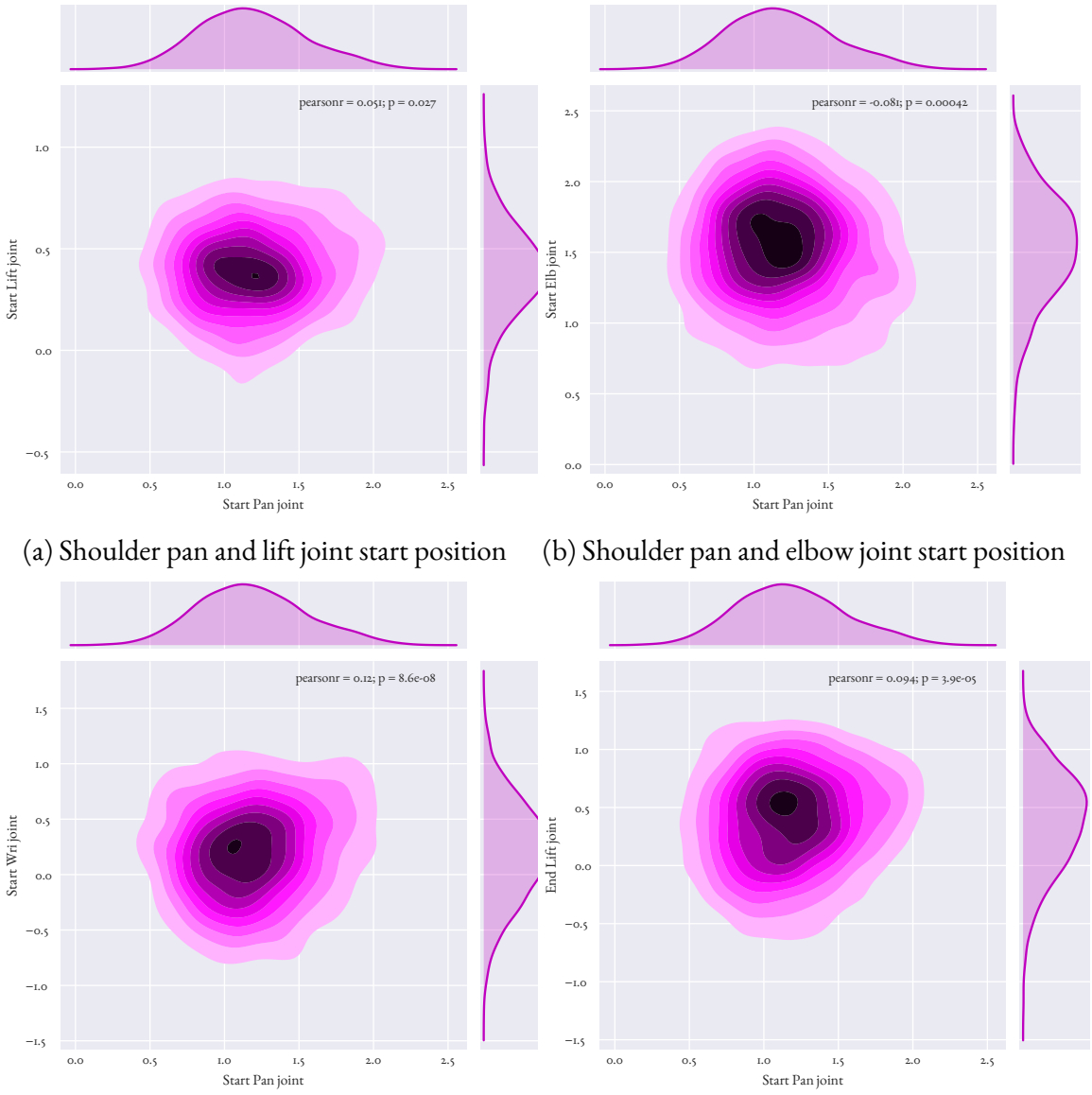
(a) Shoulder lift joint DMP

(b) Elbow joint DMP



(c) Wrist joint DMP

Figure 17: Simulation distance maximization: DMP



(a) Shoulder pan and lift joint start position (b) Shoulder pan and elbow joint start position

(c) Shoulder pan and wrist joint start position (d) Shoulder pan start and lift joint end position

Figure 18: Simulation distance maximization: Joint angle density plots

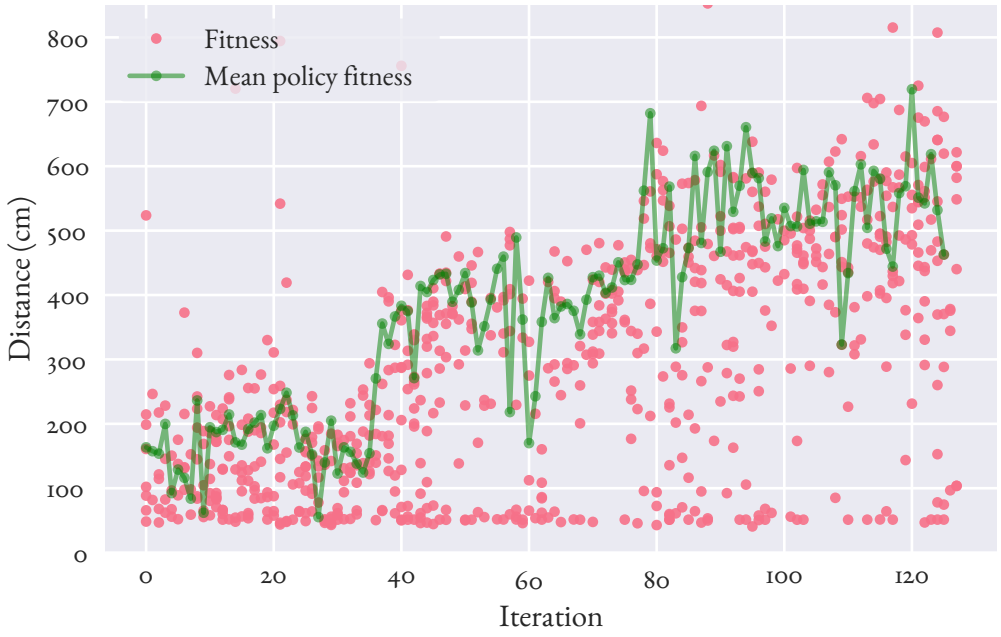
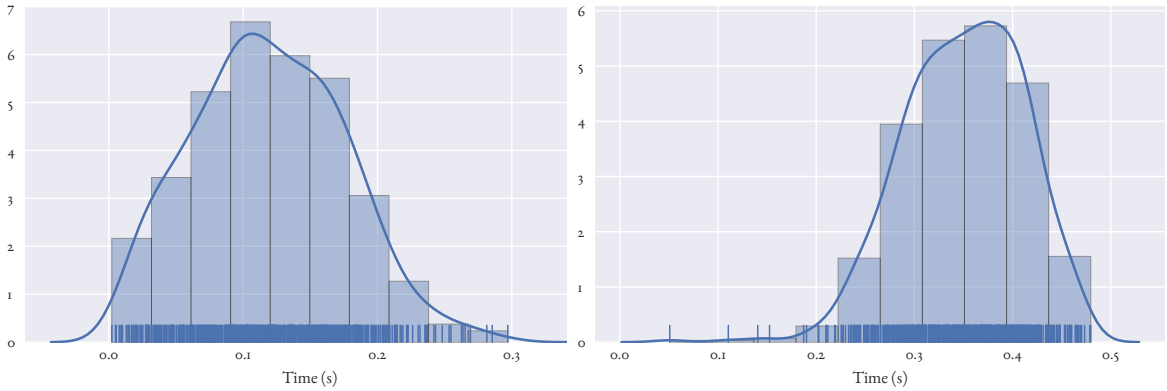


Figure 19: Simulation distance maximization: Fitness plot

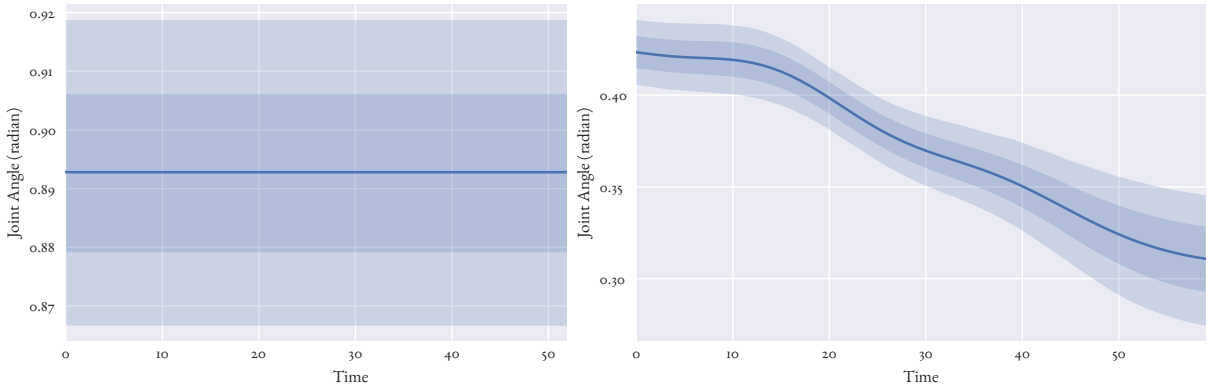
5.3.2 Simulation: Ball into Hoop

Ball into hoop experiment was performed to learn a policy to throw the ball into a hoop as in the game of basketball. The reinforcement algorithm was initialized with a sigma value of 1. The experiment was run for a total of 48 iterations. The robot was successfully able to learn a policy to throw the ball into the hoop by the end of 22 iteration. The fitness plot in figure 23 shows the distance of the ball from the hoop for every iteration.



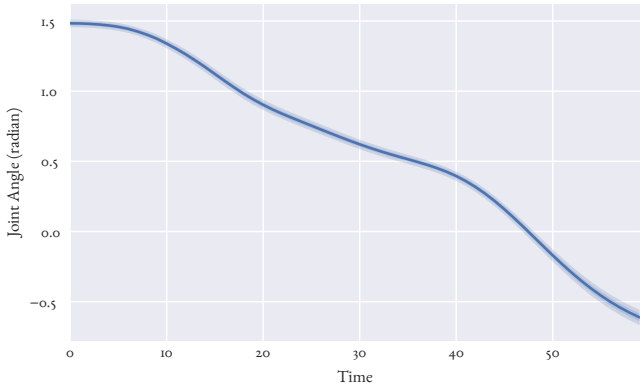
(a) Elbow joint start delay (b) Wrist joint start delay

Figure 20: Simulation ball into hoop: Joint start delay



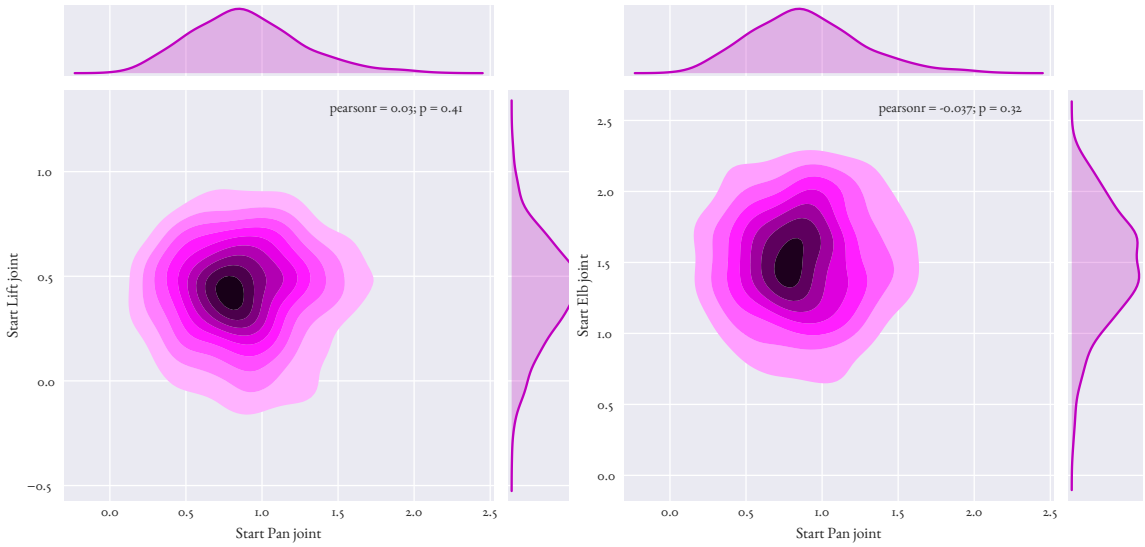
(a) Shoulder lift joint DMP

(b) Elbow joint DMP

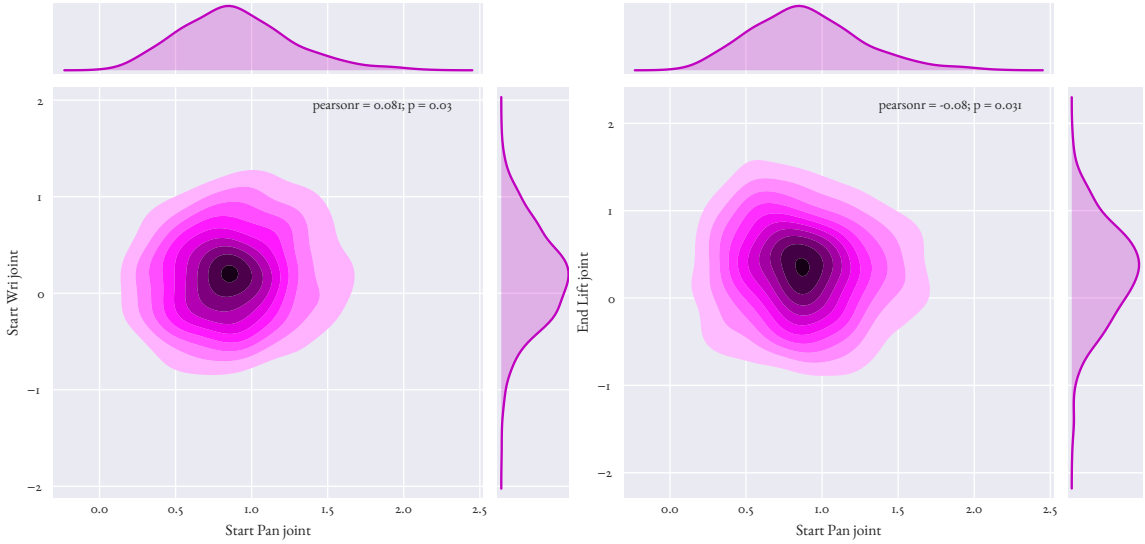


(c) Wrist joint DMP

Figure 21: Simulation ball into hoop: DMP



(a) Shoulder pan and lift joint start position (b) Shoulder pan and elbow joint start position



(c) Shoulder pan and wrist joint start position (d) Shoulder pan start and lift joint end position

Figure 22: Simulation ball into hoop: Joint angle density plots

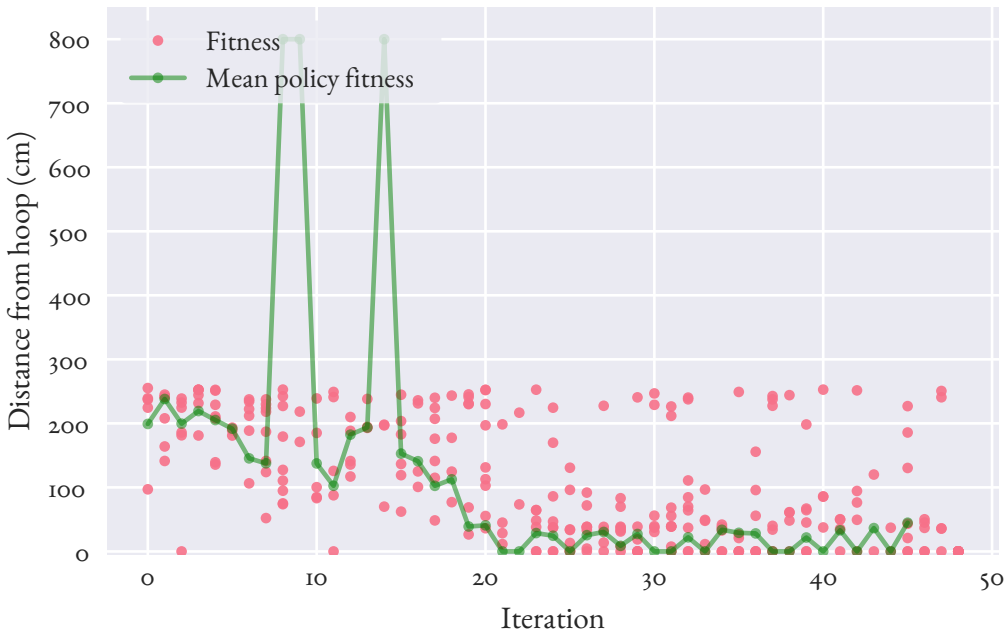
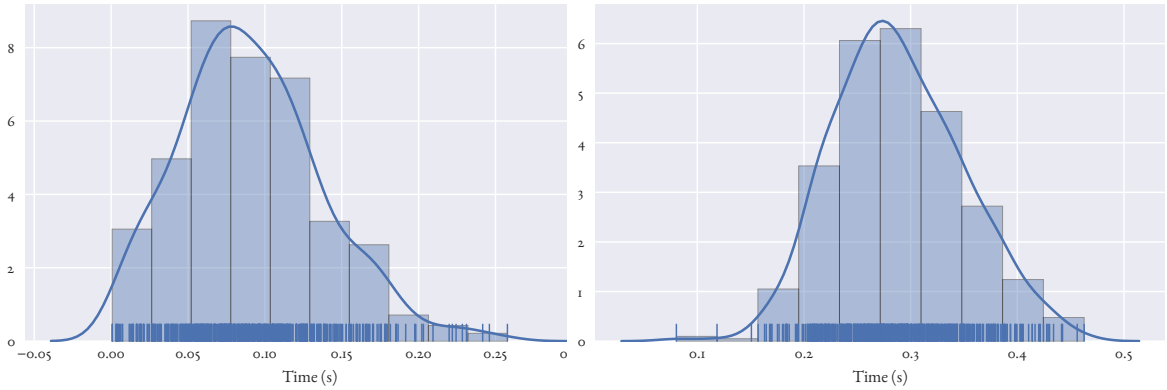


Figure 23: Simulation ball into hoop: Fitness plot

5.4 Robot Experiments

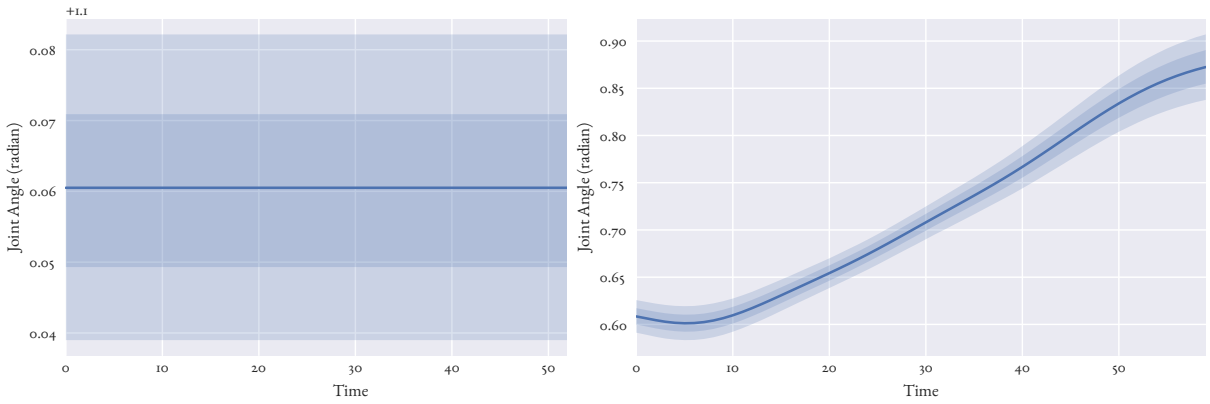
The ball into hoop experiment that was performed in the simulation, was also executed on the real robot hardware. The same set of parameters that were used for simulation was also used to conduct the experiment on the hardware. The experiment was executed for 36 iterations. The robot was able to successfully throw the ball into the hoop after 35 iterations. The fitness plots for the experiment can be seen in figure 27. Figure 28 shows a sequence of images of the robot executing the optimal policy that was learned during the experiment to throw the ball into the hoop.



(a) Elbow joint start delay

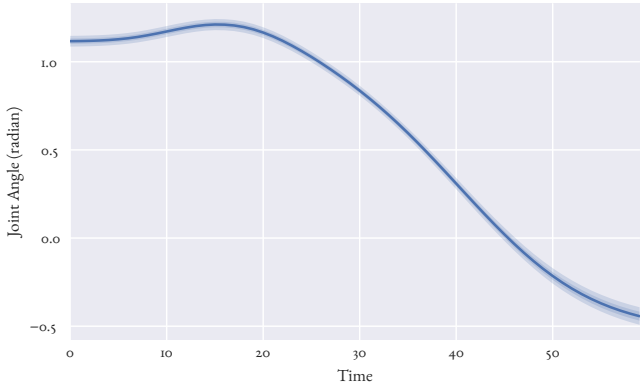
(b) Wrist joint start delay

Figure 24: Robot ball into hoop: Joint start delay



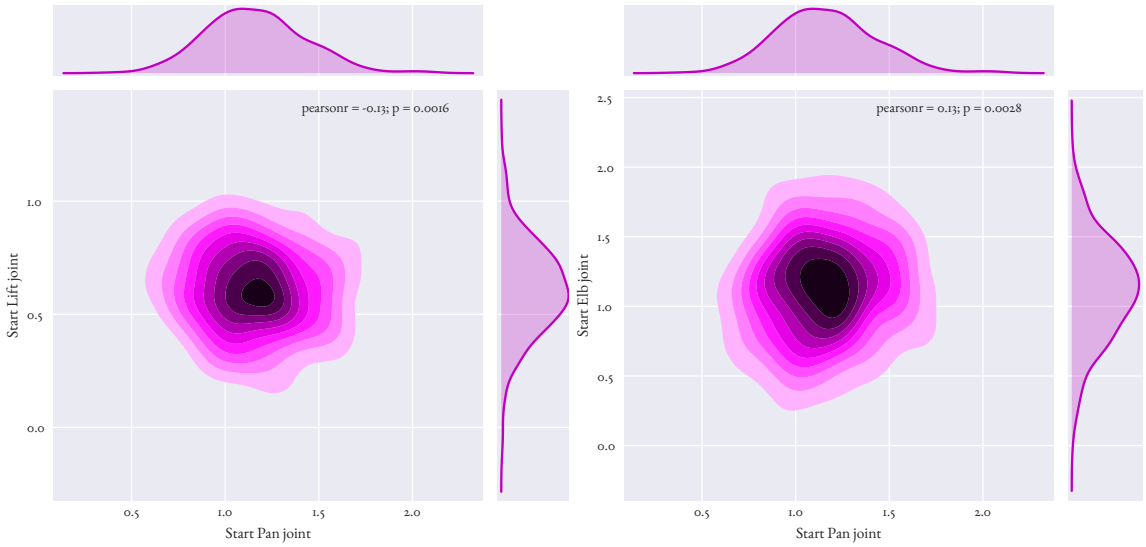
(a) Shoulder lift joint DMP

(b) Elbow joint DMP

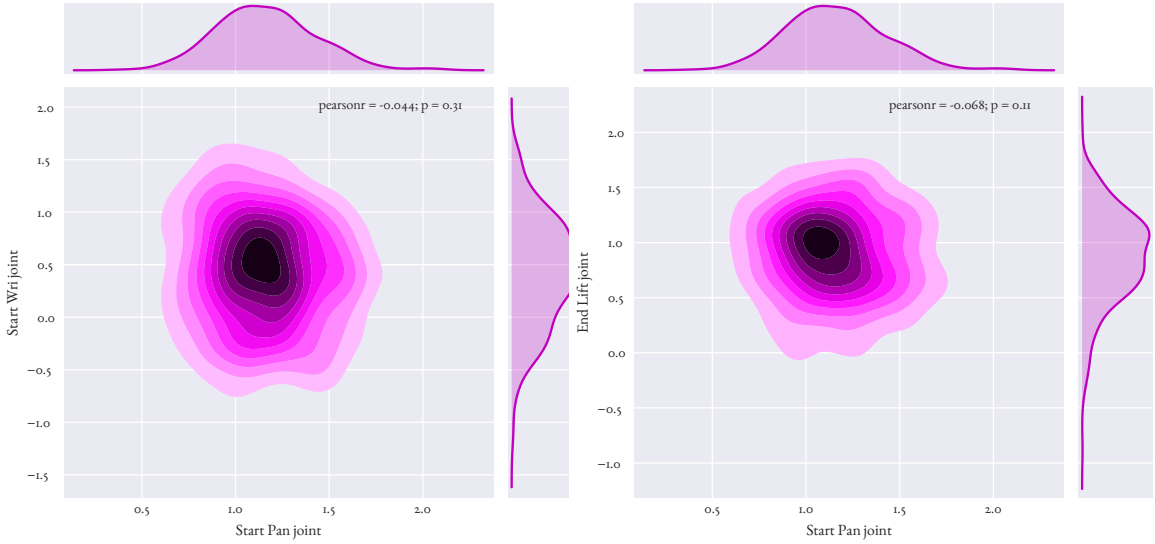


(c) Wrist joint DMP

Figure 25: Robot ball into hoop: DMP



(a) Shoulder pan and lift joint start position (b) Shoulder pan and elbow joint start position



(c) Shoulder pan and wrist joint start position (d) Shoulder pan start and lift joint end position

Figure 26: Robot ball into hoop: Joint angle density plots

Figure 27: Robot ball into hoop: Fitness plot

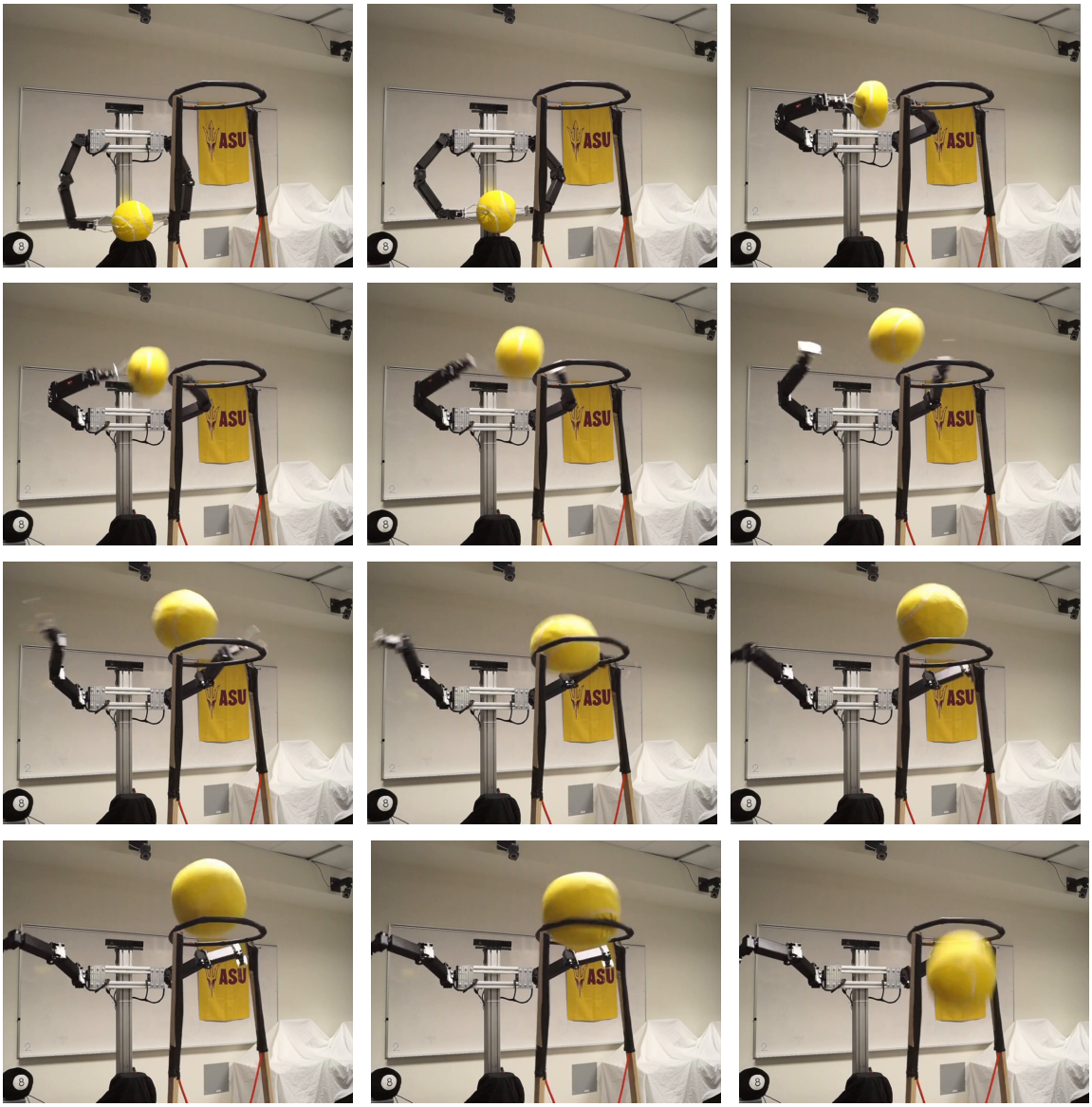
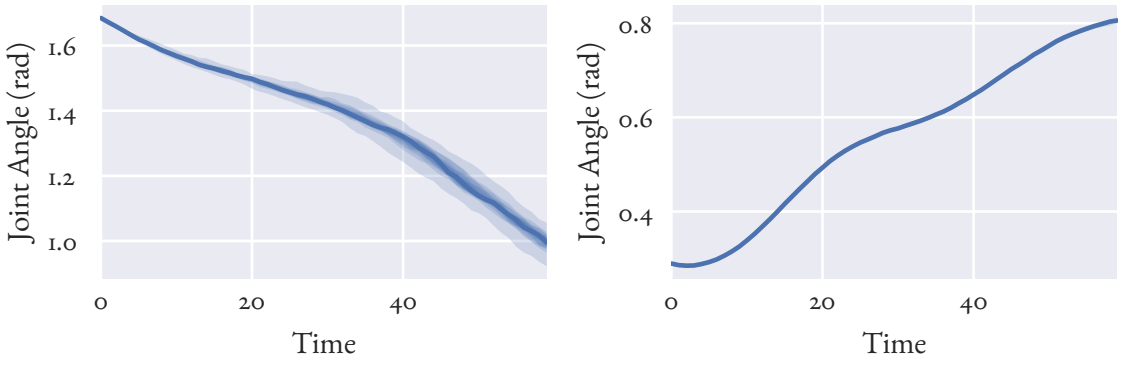


Figure 28: Robot optimal policy execution sequence

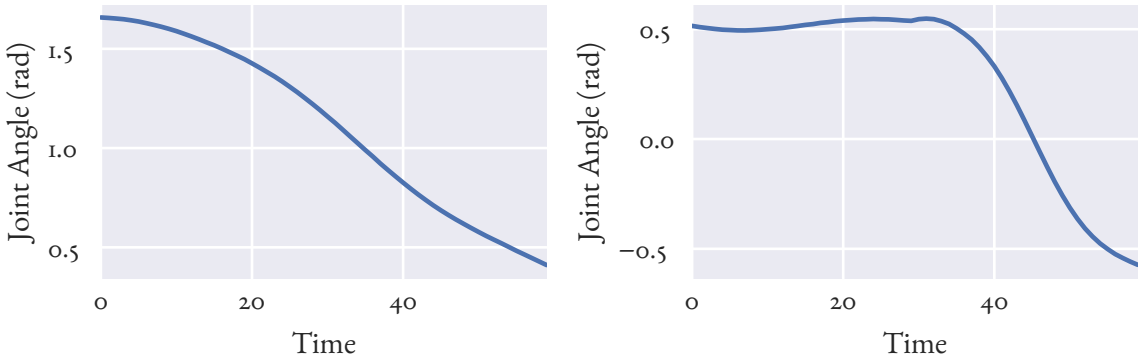
5.5 GrouPS

Group Factor Policy Search (GrouPS) is a policy search method that uncovers latent space on-the-fly based on prior structural information Luck et al. 2016. The ball into hoop experiment was also conducted using GrouPS and the additional structural information given to the algorithm resulted in faster convergence. The algorithm learned a policy as early as the 18 iteration. The fitness plot for the experiment can be seen in figure 31.



(a) Shoulder pan joint

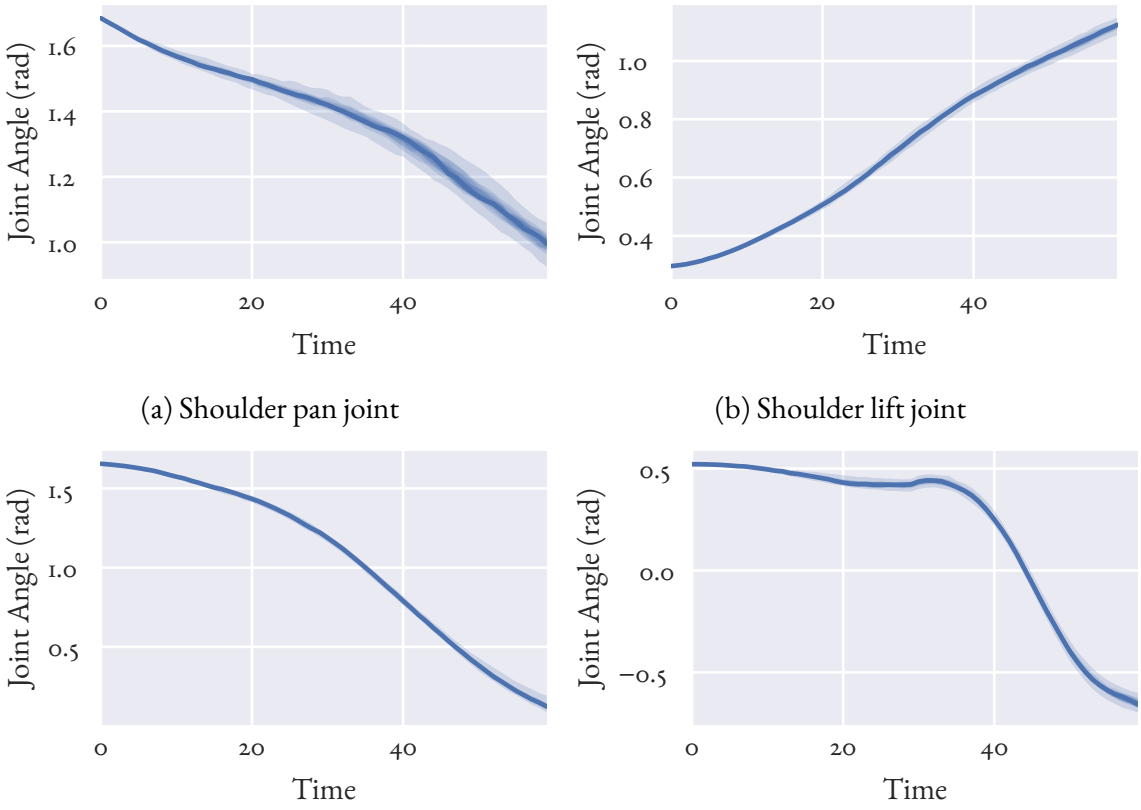
(b) Shoulder lift joint



(c) Elbow joint

(d) Wrist-1 joint

Figure 29: GrouPS: Left arm joint angle variations



(a) Shoulder pan joint

(b) Shoulder lift joint

(c) Elbow joint

(d) Wrist-1 joint

Figure 30: GrouPS: Right arm joint angle variations

Figure 31: GrouPS: Fitness plot

Chapter 6

CONCLUSION AND FUTURE WORK

A dual armed robot was built for this work, and a robust software framework was written during this work such that the robot can be used for future research work in the lab. A simulation of the robot was also built to help ease testing reinforcement learning and optimization related experiments. We have also shown how CMA-ES and DMP's can be used to learn a new skill like successfully throwing a ball into a hoop using a bi-manual robot. In the future, we would like to do more exhaustive comparisons of the results obtained with other reinforcement learning algorithms.

REFERENCES

Argall, Brenna D, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. "A survey of robot learning from demonstration." Robotics and autonomous systems 57 (5): 469–483.

Atkeson, Christopher G, and Stefan Schaal. 1997. "Robot learning from demonstration." In ICML, 97:12–20.

Billard, Aude, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. "Robot programming by demonstration." In Springer handbook of robotics, 1371–1394. Springer.

Calinon, Sylvain, Florent D'halluin, Eric L Sauser, Darwin G Caldwell, and Aude G Billard. 2010. "Learning and reproduction of gestures by imitation." IEEE Robotics & Automation Magazine 17 (2): 44–54.

Chung, Michael Jae-Yoon, Jinna Lei, Ankit Gupta, Dieter Fox, Andrew N Meltzoff, and Rajesh PN Rao. n.d. "A Developmental Approach to Goal-Based Imitation Learning in Robots."

Englert, Peter, Alexandros Paraschos, Jan Peters, and Marc Peter Deisenroth. 2013. "Model-based imitation learning by probabilistic trajectory matching." In Robotics and Automation (ICRA), 2013 IEEE International Conference on, 1922–1927. IEEE.

Hansen, Nikolaus. 2006. "The CMA evolution strategy: a comparing review." In Towards a new evolutionary computation, 75–102. Springer Berlin Heidelberg.

Ijspeert, Auke Jan, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. 2013. "Dynamical movement primitives: learning attractor models for motor behaviors." Neural computation 25 (2): 328–373.

Kober, Jens, and Jan Peters. 2009. "Learning motor primitives for robotics." In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, 2112–2118. IEEE.

———. 2012. "Reinforcement learning in robotics: A survey." In Reinforcement Learning, 579–610. Springer.

Luck, Kevin Sebastian, Joni Pajarinen, Erik Berger, Ville Kyrki, and Heni Ben Amor. 2016. "Sparse Latent Space Policy Search." In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI Press.

Rosenbrock, HoHo. 1960. "An automatic method for finding the greatest or least value of a function." The Computer Journal 3 (3): 175–184.

Schaal, Stefan. 1999. "Is imitation learning the route to humanoid robots?" Trends in cognitive sciences 3 (6): 233–242.

———. 2006. "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics." In Adaptive Motion of Animals and Machines, 261–280. Springer.

———. 2016. Imitation Learning. [Online; accessed 5-September-2016]. http://www-clmc.usc.edu/Research/ImitationLearning.