Keyframe Sampling, Optimization, and Behavior Integration: A New Longest Kick in the RoboCup 3D Simulation League

Mike Depinet
Supervisor: Dr. Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
msdepinet@utexas.edu

May 2, 2014

Abstract

Even with improvements in machine learning enabling robots to quickly optimize and perfect their skills, developing a seed skill from which to begin an optimization remains a necessary challenge for large action spaces. This thesis proposes a method for creating and using such a seed by i) observing the effects of the actions of another robot, ii) further optimizing the skill starting from this seed, and iii) embedding the optimized skill in a full behavior. Called KSOBI, this method is fully implemented and tested in the complex RoboCup 3D simulation domain. The main result is a kick that, to the best of our knowledge, kicks the ball farther in this simulator than has been previously documented.

1 Introduction

Every optimization needs a starting point. If the starting point is not in a region of the search space with a meaningful gradient, optimization is unable to make progress. For example, if trying to maximize the speed of a robot's walk, the robot must be given a stable walk to begin with. We refer to the starting point of an optimization for skill learning as a *seed skill*. Even with improvements in optimization processes, developing seed skills remains a challenge.

Currently most seed skills are written by hand and then tuned by a human until they resemble the desired skill enough to begin an optimization. Some seeds can also be acquired by having a robot mimic a human in a motion capture suit [1]. We propose a third way of creating a seed, called *keyframe sampling*, which uses learning by observation. In this case, a robot observes the effects of actions of another object, and does its best to reproduce those effects. In our work robots observe another robot with the same model, although in principle this methodology could be applied to robots with different models or to humans using transfer learning ([2]) and different body mappings as described in [1].

Other forms of learning from observation have been explored, and are described in [3]. In the context of that review, keyframe sampling has several qualities. First, we may use either a human or a robot for teaching, whereas most approaches require a human teacher. Secondly, the data set from which we learn is limited to a single sequential series of observed actions, rather than a larger set covering more initial state spaces and possibly providing repetition. It should also be noted that our methodology is assuming a continuous state space and our policy derivation is completed by a mapping function (the identity map in this case since the observed robot has the same model as the learning robot). Finally, it is important to note that the policy we derive is intended only as the seed for further optimization. While initially the policy developed from observation may not perform as well as the observed policy from which it is derived, in general we expect the learned policy's performance after optimization to surpass that of the original observed policy.

This thesis considers a 3-step methodology of keyframe sampling, optimization, and behavior integration (*KSOBI*), which guides the development of a skill from watching a teacher to using the skill as part of an existing behavior. First, we describe KSOBI in Section 2, focusing on the keyframe sam-

pling (KS) step, and demonstrating this step on a physical robot in Section 3. We introduce the robot soccer domain in Section 4, and apply keyframe sampling (with a robot teacher) and optimization (O) to kicking in robot soccer in Section 5. The robot soccer domain has the added complication that a skill is only useful if it can be incorporated into the robot's existing behavior. We described work in progress toward this final step, behavior integration (BI), in Section 6 and conclude with a summary and future work in Section 7.

2 KSOBI Overview and Keyframe Sampling

The goal of KSOBI's KS step is to use observations of another robot to quickly create an imitation skill. This imitation will later be used as the seed for an optimization, the O step, to create an optimized skill, which hopefully matches or improves upon the observed skill. Finally, that skill will be incorporated into the robot's existing behavior during the BI step. KSOBI is outlined in Figure 1.

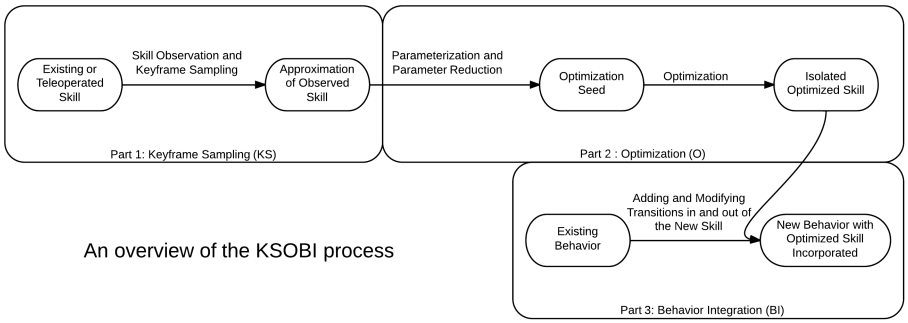

Figure 1: An outline of KSOBI

Keyframe sampling assumes that the actions of the observed robot have observable effects. In the case of the Mindstorm robot described in Section 3, the robot's actions are the torque applied to each of three motors. The effects are the change in rotation of each of the two wheels and the angle of the frontal claw. From the observed effects, a *keyframe skill* can be created directly.

A *keyframe* is defined to be a complete description of joint angles, either in absolute values or relative to the previous keyframe, with a *scale* for each joint indicating the percentage of the motor's maximum torque al-

lowed to be used to reach the target angle. (The torque applied at any point in time is determined by a controller - often a PID controller - but is multiplied by this value to affect how quickly a target angle is achieved.) A keyframe $k \in \mathcal{K} := \mathbb{R}^n \times \mathbb{R}^n \times \{0, 1\}$ where n is the number of joints, 0 indicates absolute angles, and 1 indicates relative angles.[1] The first n-vector gives target angles for each joint, while the second n-vector gives their scales. For example, the Mindstorm keyframe $k_1 = ((0, 0, 0), (0.5, 0.5, 0.5), 0)$ indicates all joints should be set to 0^o using half maximum torque, while $k_2 = ((180, 180, 0), (1, 1, 1), 1)$ indicates that the first and second motors should be rotated 180^o with maximum torque.

A *keyframe skill* (or *skill* unless otherwise noted) is defined as a list of keyframe-time pairs, where the time indicates how long to hold the paired keyframe. A skill $s \in (\mathcal{K} \times \mathbb{R})^m$ where m is the number of keyframes in the skill, and each of the m (k, t) pairs indicates that keyframe k should be the target for the next t seconds. For example, using k_1 and k_2 as defined above, the skill $s1 = ((k_1, 1.0), (k_2, 1.0))$ would indicate that the robot should take 1 second to get all its joints to 0^o (using at most half their torque) if possible, remaining there until 1 second has expired if time remains, then take another second to rotate joints 1 and 2 by 180^o as quickly as possible.

If the joint angles of an observed robot are directly observable, then a skill can be generated by recording each joint angle at specified time steps. This idea is the heart of keyframe sampling, which is made rigorous in the pseudocode below, where n is the number of joints in the robot model and T is the total time required by the skill divided by the time step:

```
define angle θ_{j,t} for j ∈ [1,n] ∩ ℤ and t ∈ [0,T) ∩ ℤ
define keyframe k_t for the same values of t
skill observeSkill(robot teacher, duration timeStep):
    int t = 0
   repeat:
      sampleKeyframe(teacher, t++)
     wait(timeStep)
   until teacher.skill.isDone()
   skill s = ((k_0,timeStep), (k_1,timeStep), ...(k_{T−timeStep},timeStep))
   return skill
```

[1]Note that in many robotic domains, including robot soccer, the distinction between relative and absolute joint positions is unnecessary since the joints have a specified non-overlapping range of possible values. This fact simplifies the above process since all keyframes may be unambiguously absolute.

```
void sampleKeyframe(robot teacher, int t):
    for joint j in teacher:
      θ_{j,t} = j.angle
   if t == 0:
      k_t = ((θ_{1,t}, θ_{2,t}, ..., θ_{n,t}), (1, 1, ..., 1), 0)
   else:
      k_t = ((θ_{1,t} − θ_{1,t−1}, θ_{2,t} − θ_{2,t−1}, ..., θ_{n,t} − θ_{n,t−1}), (1, 1, ..., 1), 1)
```

Using this method, the skill s will assume the observed starting position and, at each time step, attempt to assume the next set of observed joint angles as quickly as possible, imitating the observed object.

The generated skill s likely will not replicate the observed skill exactly, as will be seen in Section 3, since it is just a sampling of several points in a presumably continuous motion. However, as will be seen in Section 5, it may be close enough to use as the seed for an optimization. The hope is that the optimization will overcome the discontinuities in the seed skill s and create a skill which replicates or improves upon the observed motion.

Prior to optimization, it is necessary to parametrize the generated skill, allowing each value set by keyframe sampling to be varied by the optimization. Often it will then be necessary to freeze a subset of the parameters, preventing them from changing during the optimization and reducing the dimension of the parameter space. Parameter reduction is addressed in Section 5.2. Having chosen which values may vary, a fitness function should be chosen and the optimization may begin. The optimization process is described in Section 5.3. Finally, once the new skill has been optimized in isolation, it must be incorporated into existing behavior. The behavior integration (BI) process is discussed in Section 6.

3 Keyframe Sampling Example Application: Mindstorm

For an initial example of KSOBI's KS step, we work with the Lego^{TM}Mindstorm NXT robot, pictured in Figure 2. Our sample task is to drive forward, pick up a ball, and bring it back to the starting location. The robot has only 3 motors.

To create a seed for this task, we directly control the robot using its bluetooth connectivity. Another thread on the robot records the rotation

Figure 2: Lego^{TM}Mindstorm NXT Robot

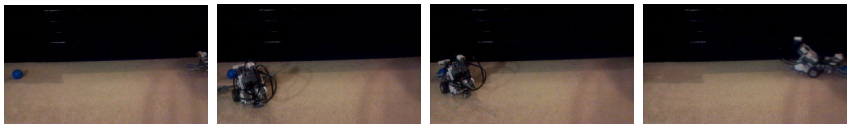

Figure 3: The controlled Mindstorm. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/MindstormControlled.mp4`

of each of its motors at 5Hz and saves them to a file. In this case, we are creating a seed by allowing the robot to observe itself while being controlled by a human, avoiding the computer vision problems associated with one robot observing another, as those problems are beyond the scope of this research. The robot being driven by bluetooth is shown in Figure 3.

Figure 4 shows the robot executing the recorded skill. While the teleoperated robot is able to adequately complete the task, the robot executing the observed skill is not. This failure is expected, as the skill generated from observation is only a discrete sampling of a continuous motion. Since the robot is always trying to get joints to particular locations as quickly as possible instead of continually applying a lesser torque, the incomplete sampling causes a jerky motion, which in this case results in the robot not quite reaching the ball before trying to pick it up. While we limit ourselves to the keyframe sampling portion of our approach for this illustration, it would be possible to optimize from here using a method similar to the one described in [4]. Sections 5 and 6 detail the full KSOBI process applied to the RoboCup 3D Simulation domain introduced in Section 4.

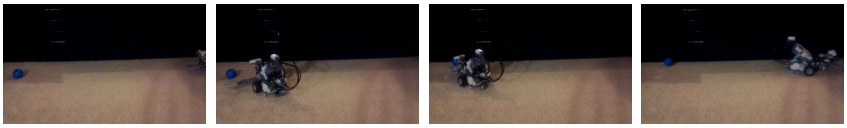

Figure 4: The controlled Mindstorm. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/MindstormObserved.mp4`

4 Application Domain: RoboCup 3D Simulation League

RoboCup is an annual competition in the AI community that uses robotic soccer as a challenging domain to push forward research in subfields like computer vision, machine learning, multiagent systems, and robotics. The competition has multiple leagues including a 2D simulation league, a 3D simulation league, and a standard platform league in which all teams run their code on a standard set of humanoid robots (NAOs). [5] Although the Mindstorm is a physical robot, with just 3 motors it is a relatively simple system. The true target application for this work is the more complex RoboCup 3D Simulation domain and its simulated robotic agents, as pictured in Figure 5.

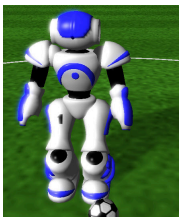

Figure 5: A simulated NAO robot

In the simulation league, each agent runs in its own process (inter-process communication is illegal) and connects to a soccer server via TCP/IP. At each step (0.02 seconds), the server first sends information to each connected robot about what it sees, feels, and hears through each of its preceptors. Next the robot sends back the torque to apply to each of its 22 motors (effectors). If the agent fails to respond before the next step, the server assumes it does nothing. The server is responsible for enforcing physics and determining the current state of the game. More information can be found on the Simspark Wiki at `http://www.simspark.sourceforge.net/wiki/index.php` ([6]).

Each agent controls a NAO robot with 22 total effectors: 2 for the head, 4 for each arm, and 6 for each leg. The field is a scale model of a full sized

human field, totaling 30 meters long, as the robots are about 0.5 meters tall. The exact specifications for the field and robot models can be found on the Simspark Wiki ([6]).

In recent years, the RoboCup 3D Simulation League has been won primarily by creating fast and robust walks ([7], [8]). However, teams are now developing their own kicks ([9], [10]). Kicking is difficult for three main reasons. First, robust kicking requires a smooth transition from walking and most walks involve some noise in reaching a target point. Second, kicking requires high precision in that a difference of a couple degrees on any joint in any keyframe will likely result in a failed kick. Third, there are many joints involved in a long distance kick and there are many keyframes between planting the foot and kicking the ball. This complexity results in a large search space for optimal kicks. Existing machine learning techniques help alleviate some of these problems, but there remains a need for finding reasonable starting seeds to guide the search through such a large parameter space, as well as a methodology for incorporating the resulting optimized skill into a full behavior, as is provided by KSOBI.

5 Learning for Kickoffs

We begin learning a kick under the assumption of a chosen starting location. In the RoboCup domain, an agent can expect this situation for its own kickoff. To learn a kick skill for a fixed starting location, we observe the previously furthest documented kick, belonging to FC Portugal [11] (see Figure 6). Using keyframe sampling, we create an approximation of this kick, which we use as a seed for optimization. The result of this optimization is the new longest known kick in the RoboCup 3D simulation environment.

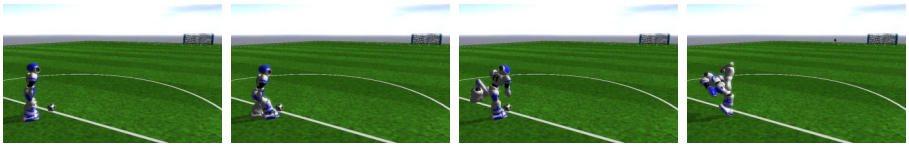

Figure 6: The observed kick. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/FCPKick.ogv`

5.1 Observing a Seed: Keyframe Sampling

The RoboCup server currently only provides the location of the head, torso, each leg, and each arm of robots to observers. This is not enough information to mimic another robot since there are multiple sets of joint angles that give the same locations for each body part. To solve this problem, we modify the server such that observers receive all of the joint angles of the observed robot, as required for keyframe sampling. The joint angles could reasonably be estimated by a real robot watching another real robot, so it seems like a reasonable level of detail to request. With this added information, we apply keyframe sampling at 16.67Hz (every 3 server cycles). As expected, the result is not an exact match of the observed skill. The imitation skill results in the robot kicking the ground behind the ball and falling over (Figure 7). However, the imitation is close enough to use as a seed for the optimization.

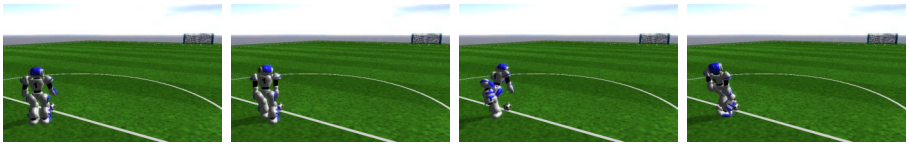

Figure 7: The seed after observation. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/InitialKick.ogv`

5.2 Single Agent Training

So far we have focused primarily on the keyframe sampling step of KSOBI. Now we will shift focus to step 2, optimization.

The observed seed in Section 5.1 results in a skill with 89 key frames, each with every joint included, giving a total of 1958 parameters to train. Although this search space is prohibitively large, many of the parameters can be safely ignored. In fact, there is a need for parameter reduction before optimization in general when the seed is created by keyframe sampling. The goal in parameter reduction is to freeze parameters whose varied values will not strongly affect the skill, removing them from the optimization and reducing the dimension of the search space. In addition to domain heuristics, seeds from keyframe sampling offer some general heuristics. First, any joint that does not change significantly between two keyframes can be fused between frames. Second, beginning and ending keyframes can sometimes be removed entirely. In this case, it is important to be careful not to disrupt

the transition in and out of the skill (e.g. you would not want to remove keyframes responsible for setting the plant foot from a kicking skill).

In the case of our kicking seed, removing the head joints (a domain heuristic), any joint that does not change by more than 0.5 degrees between two frames, and the keyframes before the plant foot is set limits the skill to only 59 parameters. Adding 3 parameters for the starting location (x, y, and angle), results in 62 parameters to optimize. This is still a large state space, but it is manageable.

With the optimization parameters chosen, the next step is to define a fitness function. We use the distance traveled by the ball

$$fitness_{initial} = \begin{cases} -1 & : \text{Failure} \\ finalBallLocation.x & : \text{Otherwise} \end{cases}$$

where a "Failure" is any run in which the robot falls over, kicks backward, or runs into the ball before kicking it.

5.3 Optimizing with CMA-ES

Optimizing a set of parameters is the same as finding the global maximum of a fitness function $g(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ where n is the number of parameters being tuned. Several methods for finding this maximum exist and each has its own advantages and disadvantages. If a gradient of g can be calculated (or approximated) at a point, hill climbing is a mathematically proven way of finding local maxima of an objective function. Starting from multiple points results in finding multiple local maxima, one of which is hopefully the global maximum. Evolutionary strategies are also popular. Although such strategies have less theoretical basis, experimental results have shown that they often perform quite well.

Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is one such evolutionary strategy. After each generation is evaluated in CMA-ES, the weighted average is calculated and the covariance matrix (the predicted relationship between each pair of parameters) is updated. When creating the next generation, CMA-ES uses the mean value of its population to pick new species that are similar to species that have done well previously. It also increases the probability of species moving in the direction of previously successful steps (while increasing that step size) using the updated covariance matrix. Combining these methods is similar to approximating a gradient, in a way combining hill climbing with an evolutionary strategy,

giving CMA-ES generally faster observed convergence times than a simple genetic algorithm [12]. Although algorithms with even faster convergence exist [13], we have had previous success using CMA-ES for optimizing a walk, as documented in [14].

5.4 Single Agent Results

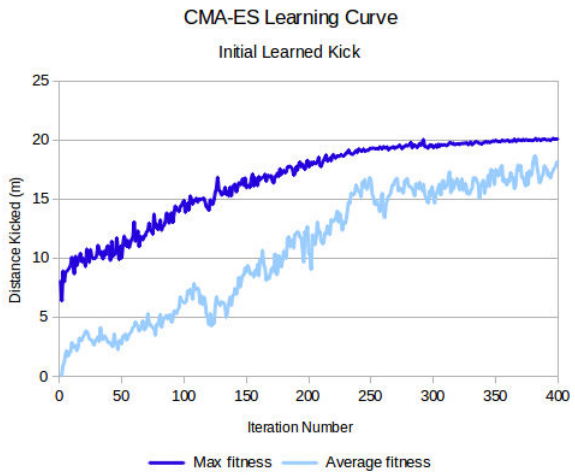

Figure 8: CMA-ES Learning Curve for Initial Kick

After 400 iterations of CMA-ES with a population size of 200, the resulting skill is able to kick the ball 20 meters on average (see Figure 8). In addition to solving the problem of the robot kicking the ground behind the ball and helplessly falling over, the optimization produced a kick which exceeded the length of the original observed kick by more than 5 meters (Figure 9)!

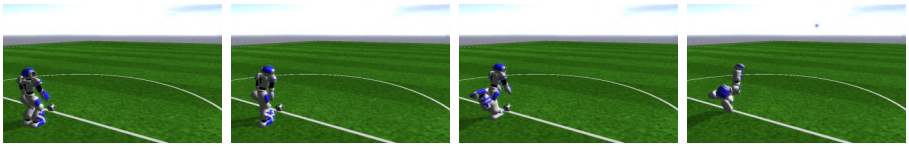

Figure 9: The first learned kick. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/LearnedKick.ogv`

We also consider two other fitness functions, producing slightly different resulting kicks. The first is a fitness function centered around accuracy. This function uses the same ball distance fitness as before except with a Gaussian penalty for the difference between the desired and actual angles. Optimization with this function gives the powerful and predictable kick seen

in Figure 10.

$$f_{accuracy} = \begin{cases} -1 & : \text{Failure} \\ finalBallLoc.x * e^{-angleOffset^2/180} & : \text{Otherwise} \end{cases}$$

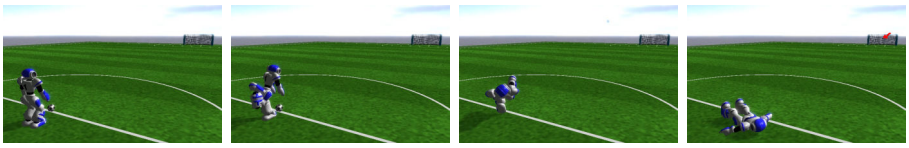

Figure 10: Improved accuracy. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/AccuracyKick.ogv`

The second was a fitness function centered around distance in the air. As the idea is to kick the ball long distances above opponents' heads, the fitness function heavily rewarded the distance traveled by the ball before descending to 0.5m above the ground. It also moderately rewards total distance and heavily penalizes missing the goal (ignoring any other tests of accuracy). This resulted in a noisy kick, but one that travels over 11m in the air (see figure 11).

$$f_{air} = \begin{cases} -1 & : \text{Failure} \\ 0 & : \text{Missed goal} \\ 100 + finalBallLoc.x + 2 * airDist & : \text{Otherwise} \end{cases}$$

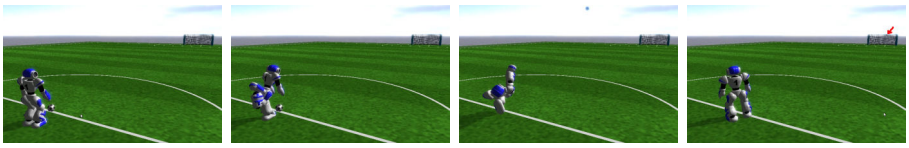

Figure 11: Increased air distance. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/AirDistKick.ogv`

These results are summarized in Table 1. In addition to being the longest documented kicks to our knowledge, these kicks are also the first ones able to score from any point in the offensive half of the field.

Table 1: Kick distances

Kick	Avg Distance (m)	Notes
Observed seed	About 15	
FCPortugal	About 17	Based on empirical data and verbal confirmation
Learned Kick	20.0($\pm$0.12)	
Accuracy Kick	18.8($\pm$0.29)	With placement $1.3^o(\pm 1.78^o)$ from target angle
Air Distance Kick	19.2($\pm$0.38)	With $11.4m(\pm 0.25m)$ higher than 0.5m

5.5 Multi-Agent Training

With the kick optimized in isolation, we continue now to the behavior integration (BI) step. As the kick was optimized from a fixed point, integration into a legal kickoff is a natural first integration.

Unfortunately, scoring from the kickoff is illegal unless someone else touches the ball first in soccer. To rectify this, we introduce another agent with another skill which moves the ball as little as possible then gets out of the way. After optimizing this skill alone, using the server's play mode and the distance of the ball's movement to determine fitness, we optimize the touch and kick together, using the same fitness function as used for the kicker with an added penalty for either agent missing the ball or the kicker hitting the ball before the toucher.

$$f_{touch} = \begin{cases} -1 & : \text{Failure} \\ 10 - finalBallLoc.magnitude & : \text{Otherwise} \end{cases}$$

where for this single case, a "Failure" is when the robot falls over, fails to touch the ball, or touches the ball more than once.

$$f_{kickoff} = \begin{cases} -1 & : \text{Failure} \\ -1 & : \text{Wrong touch order} \\ -1 & : \text{Either agent missed} \\ 100 + finalBallLoc.x + 2{*}airDist & : \text{Otherwise} \end{cases}$$

5.6 Multi-Agent Results

After a successful 400 iteration optimization with population size of 150 (see figure 12), two agents are able to legally and reliably score within 8 seconds of the game starting and within 3 seconds of the ball first being

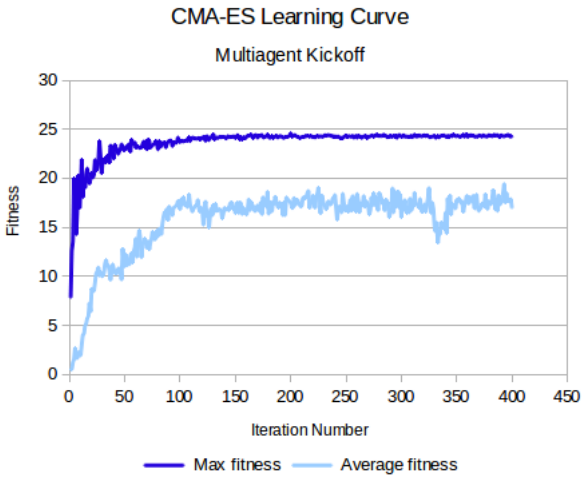

Figure 12: CMA-ES Learning Curve for Multiagent Kickoff

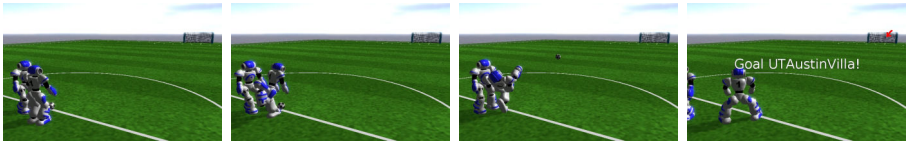

Figure 13: Multiagent Kickoff. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/Kickoff.ogv`

touched (see figure 13). Adding this kickoff behavior and changing nothing else dramatically improves the team's overall performance versus the team from whom the new kick was initially observed (see Table 2).

The percentage of kickoffs which score varies with the opponent team (see table 3). Most kickoffs which fail to score are a result of opponent player formation. We have not found a kick that makes it all the way to the goal in the air, so a player located where the ball bounces on its path to the goal effectively stops kickoff goals. That said, the location at which the ball bounces is not always the same. In the future, the kicking agent could have multiple kickoff kicks and information on where the ball bounces for each and could use that information at run-time to choose a kick that misses opponents.

6 Adding an Approach

The second necessary behavior integration step is adding the new kick to existing walking behavior to make it available during game play.

Table 2: Game statistics

Using Kickoff	Opponent	Average Goal Differential	W-L-T
No	FCPortugal2013	0.335 (+/-0.023)	368-85-547
Yes	FCPortugal2013	1.017 (+/-0.029)	775-6-219
Yes	UTAustinVilla2013	0.385 (+/-0.022)	416-44-540

While the results described in Section 5 apply to the kickoff when the robot begins at a fixed and known distance from the ball, to be able to use such a kick during game play, for example to enable robust passing, the robot must be able to kick the ball after approaching it from any position. Additionally, both the approach and the kick must be quick for the kick to be useful during a game. This section describes work in progress toward such an approach.

The approach can be divided into three parts: walking up to the ball, which is handled by the UTAustinVilla walk engine [14]; moving the plant foot to a specific location and orientation relative to the ball; and stabilizing over some point in the plant foot. Following these steps, the robot may resume the kicks that we have optimized.

In the case of our observed kick, the kick skill included two slow steps toward the ball before the plant foot was set. In order to make the approach and kick faster, these two steps were removed in favor of a single step using IK as described in the following section.

Table 3: Percentage of scored kickoffs against the top 4 finishers from RoboCup 2013

Opponent	Beginning of Half	During Half
FCPortugal2013	92.19%	77.15%
UTAustinVilla	76.70%	54.15%
SeuJolly	77.00%	77.66%
Apollo3D	89.30%	65.60%

6.1 Positioning and Plant Foot

There will always be some noise in exactly where a walking robot ends up. That said, there is also an area around the ideal kicking point from which the robot should be able to plant its foot at the desired plant foot location. To allow these to offset each other, the agent switches from positioning to moving its plant foot whenever it is within a bounding box of the ideal starting point (Figure 14). This bounding box is defined by six values: how far the robot may be in any of the four cardinal directions from the ideal starting point, and how much the robot may be rotated in either direction from the ideal starting angle. It would not be difficult to extend the first four values to create an arbitrary polygon. Within its bounding box, the robot uses inverse kinematics to set its plant foot at an optimized location and orientation relative to the ball [10]. The bounding box is constructed such that this is usually possible. The kick fails when it is not.

6.2 Stabilization

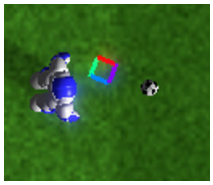

Figure 14: Bounding box for a kick

With the plant foot set, a new problem arises. Although the distance between the ball and the plant foot is known, the distance between the center of mass and the plant foot is not. Thus it is necessary to dynamically shift the center of mass over the plant foot. For now, the agent achieves restabilization by moving its arms and then altering the roll and pitch of its ankles if necessary. This method is similar to how other RoboCup teams stabilize ([9]).

6.2.1 Arms

The arms can move toward the desired center of mass by the equations:

$$A_1 = tan^{-1}\left(\frac{dirToMove.x}{dirToMove.y}\right) - 90$$

$$A_2 = sin^{-1}\left(\frac{dirToMove.y}{armComRelShoulder.magnitude * cos(A_1)}\right)$$

Here A_1 is the angle of the first arm joint, which rotates around the y axis (the robot's left is positive y). A_2 is the angle of the second arm joint, which

rotates around the z axis when A_1 is at 0 degrees (making the arm directly forward). $dirToMove$ is a vector between the current center of mass and the desired location of the center of mass, multiplied by the ratio $\frac{bodyMass}{armMass}$. $armComRelShoulder$ is the center of mass of the arm relative to the shoulder. The maximum range of the joints, as well as the substantial possibility that the arms alone cannot move the center of mass enough must be considered as well. Since the arms comprise less than 20% of the robot's mass, this change is usually not enough to stabilize the robot.

6.2.2 Ankle

Changing the roll and pitch of the ankles significantly shifts the center of mass of the robot, since turning the ankles moves the entire rest of the body.

For a single contact point, i.e. assuming the plant foot is on the ground and the other foot is not, the amount to turn the ankle can be computed by:

$$Pitch_{new} = sin^{-1}\left(\frac{\frac{bodyMass}{bodyMass-footMass} * dirToMove.x}{comMagX} + sin(Pitch_{old})\right)$$

$$Roll_{new} = sin^{-1}\left(\frac{\frac{bodyMass}{bodyMass-footMass} * dirToMove.y}{comMagY} + sin(Roll_{old})\right)$$

Here $dirToMove$ is as above, except that it has not been scaled. $comMagX$ is the magnitude of the center of mass relative to the foot in the xz plane and $comMagY$ is the same in the yz plane.

The problem is more difficult if both feet are on the ground, since in that case rolling both ankles may have the effect of raising the body instead of moving it to the side, since the feet may not remain flat on the ground. In this case, one solution is to roll the hips as well, so that both feet remain flat on the ground and the torso is shifted to one side. This gives:

$$\theta = \theta_0 + sin^{-1}\left(\frac{dirToMove.y * bodyMass}{leg * (bodyMass - llm - rlm) + llCom * llm + rlCom * rlm}\right)$$

Here $dirToMove$ is as above, leg is the distance from hips to feet in the yz plane, llm and rlm are the masses of the left and right legs, and $llCom$ and $rlCom$ are the magnitudes of the centers of mass of the legs relative to the feet in the yz plane. Setting the roll of both ankles from θ_0 to θ and the roll of each hip to $-\theta$ gives the desired result.

6.2.3 Results

As a comparison with this dynamic stabilization method, a set of stabilization keyframes are added to the kicking skill. These keyframes are optimized for stabilization to be fair. Using this keyframe stabilization, the robot can kick the ball after an inverse kinematics plant in 29.11% of 10000 attempts, beginning from a uniform distribution of points inside 0.1m x 0.1m box centered at the ideal starting point. Replacing keyframe stabilization with the dynamic solution described above marginally improves the success rate (30.68%). We suspect the reason for most of the failures is that the stabilization process only gives a stable pose, ignoring the robot's angular momentum. As a result, the robot often falls forward after shifting its weight forward onto its plant foot. Using a bounding box approximately a third of the size results in success in 84.41% of 10000 attempts, likely because the box limits the maximum distance between the center of mass and the plant foot, which limits the angular momentum needed to achieve the target pose. Luckily, recent changes surrounding the walk engine allow the robot to sometimes position itself within this smaller bounding box, resulting in the full behavior seen in Figure 15.

Figure 15: Approach and Kick. Video available at: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2014/videos/approach.ogv`

While more work is still needed to make the approach and kick faster, it is very close to being usable in a game, and could probably be used for set plays as is.

7 Summary and Future Work

This thesis introduced the KSOBI process, guiding the development of a skill from watching another robot (keyframe sampling - KS), to optimizing

the resulting sampled skill (optimization - O), to integrating the optimized skill into an existing behavior (behavior integration - BI). The KS step was applied to both a physical robot and a simulated robot, with the full KSOBI process being demonstrated in the simulated case. Along the way, we showed the success of this method with a set of new kicks which raise the bar for how far agents can kick in the RoboCup 3D simulation league.

As mentioned in Section 6, more work is needed to make these kicks faster for regular game play. The most important direction for future work is to hasten the approach and kick so that the kicks can be used to pass and score robustly in a game.

It may also be possible to make the kicks more robust by defining them using trajectories relative to the ball instead of fixed joint angles. The UTAustinVilla codebase already has a set of kicks parametrized by trajectories relative to the ball [10], however they seldom exceed a distance of 5 meters. It would be nice to find a happy medium between the flexibility of those kicks and the distance achieved by fixed joint angle kicks. With that worked out, it will become important to add planning so that the best pass options are chosen.

7.1 Planning

With a reliable kick available, agents with the ball have several new options to consider. Rather than just dribbling as in previous years, agents may choose to pass to a teammate or to shoot. Consequently, the ability to formulate complex plays and predict their success becomes important. I propose an expectimax search, with the values of leaves being determined by the probability of scoring with an "always shoot at the goal" strategy, assuming opponents remain still. Because opponents will likely move between decision time and several steps later, it is difficult to know where they will be at the beginning of the "always shoot" strategy, and opponent modeling would certainly be useful for pass planning. Using an "always shoot" strategy instead of an "always dribble" or "always pass" strategy would push agents toward goal-scoring opportunities. To determine the probability of scoring from a particular location, I propose empirically determining the probable ball trajectories resulting from a kick using a strategy similar to the one described in [15]. Using this distribution for a discretized field plus three additional states - goal, opponent goal, and opponent possession - gives a Markov chain with the abbreviated matrix shown in table 4, where the 1x1 state represents the

600 1 meter squares in which the agent's team has possession (each its own state). This matrix could be quickly updated given the current locations of opponents at runtime. Then the exit distribution for each leaf state could be computed, giving a reasonable approximation of the chance of eventually scoring from that state. The expected number of steps to score could also be computed and may also be useful for selecting states that score more quickly.

Table 4: Planning Markov Chain

/	1x1 location	Opponent	Goal	Opponent Goal
1x1	learned	runtime	learned	learned
Opp	heuristic	heuristic	0	heuristic
Goal	0	0	1	0
OppGoal	0	0	0	1

Acknowledgements

The author would like to thank the following for their support:

1. **Dr. Peter Stone** for his supervision
2. **Dr. Peter Müller** for his mathematics support
3. **Patrick MacAlpine** for his assistance with UTAustinVilla resources and code base
4. **Undergraduate Research Opportunities (UROP)** for financial support

References

[1] A. Setapen, M. Quinlan, and P. Stone, "Marionet: Motion acquisition for robots through iterative online evaluative training," in *Ninth International Conference on Autonomous Agents and Multiagent Systems - Agents Learning Interactively from Human Teachers Workshop (AAMAS - ALIHT)*, May 2010.

[2] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. 1, pp. 1633–1685, 2009.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, May 2009.

[4] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *The Nineteenth National Conference on Artificial Intelligence*, July 2004, pp. 611–616.

[5] S. Barrett, K. Genter, Y. He, T. Hester, P. Khandelwal, J. Menashe, and P. Stone, "Austin villa 2012: Standard platform league world champions," 20012.

[6] (2012) Simspark wiki. [Online]. Available: http://simspark.sourceforge.net/wiki/index.php/Main_Page

[7] A. Bai, X. Chen, P. MacAlpine, D. Urieli, S. Barrett, and P. Stone, "Wright Eagle and UT Austin Villa: RoboCup 2011 simulation league champions," in *RoboCup-2011: Robot Soccer World Cup XV*, ser. Lecture Notes in Artificial Intelligence, T. Roefer, N. M. Mayer, J. Savage, and U. Saranli, Eds. Berlin: Springer Verlag, 2012.

[8] P. MacAlpine, N. Collins, A. Lopez-Mobilia, and P. Stone, "UT Austin Villa: RoboCup 2012 3D simulation league champion," in *RoboCup-2012: Robot Soccer World Cup XVI*, ser. Lecture Notes in Artificial Intelligence, X. Chen, P. Stone, L. E. Sucar, and T. V. der Zant, Eds. Berlin: Springer Verlag, 2013.

[9] R. Ferreira, L. Reis, A. Moreira, and N. Lau, "Development of an omnidirectional kick for a nao humanoid robot," in *Advances in Artificial Intelligence ? IBERAMIA 2012*, ser. Lecture Notes in Computer Science, J. Pavón, N. Duque-Méndez, and R. Fuentes-Fernández, Eds. Springer Berlin Heidelberg, 2012, vol. 7637, pp. 571–580.

[10] P. MacAlpine, D. Urieli, S. Barrett, S. Kalyanakrishnan, F. Barrera, A. Lopez-Mobilia, N. Ştiurcă, V. Vu, and P. Stone, "UT Austin Villa

2011: A champion agent in the RoboCup 3D soccer simulation competition," in *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.

[11] N. Lau, L. P. Reis, N. Shafii, and R. Ferreira, "Fc portugal 3d simulation team: Team description paper," in *RoboCup 2013*, June 2013.

[12] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 312–317, 1996.

[13] ——, "Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu_i, \lambda)$-cma-es," *5th Europ. Congr. on Intelligent Techniques and Soft Computing*, pp. 650–654, 1997.

[14] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone, "Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, July 2012.

[15] M. Ahmadi and P. Stone, "Instance-based action models for fast action planning," *RoboCup-2007. Robot Soccer World Cup XI*, pp. 1–16, 2007.