Tuning the Blender Physics Engine via a Genetic Algorithm

CISC 7902X: RESEARCH PROJECT 2
Name: David Lettier

1 Abstract

Using a physics engine for robotic simulation provides fidelity with the added benefit of lower research costs, minimal space requirements, and even less time as the physical simulation can be run at faster than real-time speeds [15]. However, physics engines have numerous parameters to tune such as friction coefficients, rigid-body constraints, body mass, gravitational acceleration, material elasticity, collision shapes, etcetera [3][2]. Tuning these either programmatically or as in the case of *Blender*[1]—via the graphical user interface—is a daunting task considering the multidimensional state space. Simulation fidelity is compromised as some parameters are perfected while others are now maladjusted. Adjusting some parameters only to malign others leads to a "herding cats" type of effect. This ultimately results in a "reality gap" between the simulated environment and the physical environment. However, by using a genetic algorithm to optimize or rather fine tune the physics-engine parameters, the hypothesis is that the physics engine will produce a robot motion model more closely reassembling that of the real-world robot kinematics.

2 Introduction

Utilizing a physics engine for robotic simulation allows for rapid trial runs with minimal overhead. By using a physics engine, one can maintain a high fidelity simulation of the real world. However, there are some caveats such as numerical errors, missed collisions, and timing issues ultimately resulting in some compromises. Compromises include time, scale, and intricacy. With so many parameters to configure, setting the initial state of the physics engine is a daunting task.

The Blender/Bullet physics engine is rigid and soft body based with approximately 57 individual parameter settings which include gravitational acceleration, physics steps, friction, anisotropic friction, elasticity, dampening, rigid-body constraints, material force-field, mass, radius, form-factor, collision bounds, collision margin, obstacle radius, servo control, force, torque, linear velocity, and angular velocity. There are even more parameters for the non-real-time rendering engine, however, this project will only focus on the real-time component of Blender known as the *Blender Game Engine* [1]. These parameters can be set either programmatically via the *Blender Python API* or through the graphical user interface.

To date, Blender's real-time physics engine has been used for many scientific simulation and visualization projects [9][8]. For this project, the Blender Game Engine will be used to produce a physics-based motion model (and ultimately a complete simulation) for the *Surveyor SRV-1*[2] *Blackfin* robots used in *HRTeam*'s multi-agent system [4][25]. As of right now, the Blender HRTeam 3D simulation—titled *BlenderSim*—uses a constant velocity motion model while only using the physics engine to test collisions between the robots and their environment also known as the *arena*. Moving forward, BlenderSim will use a probabilistic, data-mined motion model created by mining historical log data from previous real-world lab runs.

The goal of this research project is two fold. The first portion is to develop a methodology for heuristically searching—via a genetic algorithm (GA)—optimal physics-engine parameters in order to increase simulation fidelity. The second portion is to demonstrate this methodology by comparatively

[1]Blender, released under the GNU General Public License, is a 3D creation suite allowing artists and programmers alike to produce immersive audio/visual works ranging from animation shorts to real-time 3D interactive games. Features include 3D modeling, 3D sculpting, texturing, sound editing, film editing, motion tracking, rigging, rendering, animation, physical simulation, a real-time 3D engine, and extensibility via the integrated Python API [14].

[2]The Surveyor SRV-1 is a 120mm long x 100mm wide x 80mm tall robot utilizing two independent treads for locomotion. The platform of the SRV-1 is open source and is intended for "education, research, and exploration." Features include a built-in web server and fully autonomous programmable operation.

showing the differences and similarities between the produced, tuned-simulation setup and the real-world HRTeam, SRV-1 lab setup. As hypothesized, the methodology should demonstratively diminish the differences while at the time increase the similarities between the simulated robot motion and the physical robot motion.

Creating the methodology involves three stages of research and development. The first stage involves gathering data on the motion of the real robot thereby creating a probabilistic robot motion model. Here, the real robot will be given numerous runs of successive permuted atomic commands[3] with its starting and ending location being recorded after each atomic command performed. Once all runs have been recorded, all starting and ending positions (per atomic command) will be placed relative to one another in order to facilitate calculating the distribution of ending positions. The second stage involves a cycle of fitness testing the generational outcomes of the GA by comparatively evaluating the motion produced by the tuned physics engine with that of the real robot motion data observed from the previous stage. Lastly, the third stage involves evaluating the effectiveness of stage one and two by comparing the results collected in the tuned, simulated world with the results collected in the physical world. The hypothesis is that as the GA produces more and more generations, the simulated robot physics-based motion will converge with that of the real robot motion. In other words, given any one of the atomic commands, the simulated robot will move just as the real robot would.

If the hypothesis is supported, then future physics based simulations can be rapidly tuned to produce a high fidelity with that of the simulated object's real-world counterpart—in this case the SRV-1s. By having a high fidelity physics-based motion model, simulated runs will provide stochastic results as is the case with real-world lab runs. Furthermore, this highly tuned physics engine will provide realistic movement adding to the quality of observation for any given simulated experiment.

The motivation for the research project is to develop a system to automatically tune a physics engine resulting in high fidelity without having to understand or even care about how the parameters were derived. Interestingly, by borrowing from naturalistic processes, an optimal solution can be found to a problem by seemingly random occurrences.

3 Background

3.1 Previous Work

3.1.1 Research Experience for Undergraduates (REU) Fellowship

This research project is mainly inspired by the work accomplished building BlenderSim during my REU Fellowship and by the mentorship of Dr. Elizabeth Sklar and Dr. Simon Parsons. Over the course of three months, a Blender based 3D simulation was developed for HRTeam. Initially, BlenderSim was wholly physics based but soon proved to be problematic on three fronts: time, scale, and intricacy.

Initial problems arose when the treads of the SRV-1 were recreated in the simulation. Even after numerous hours adjusting physics parameters and rigid-body configurations, the treads would consistently behave in erratic fashions. See Figure 1.

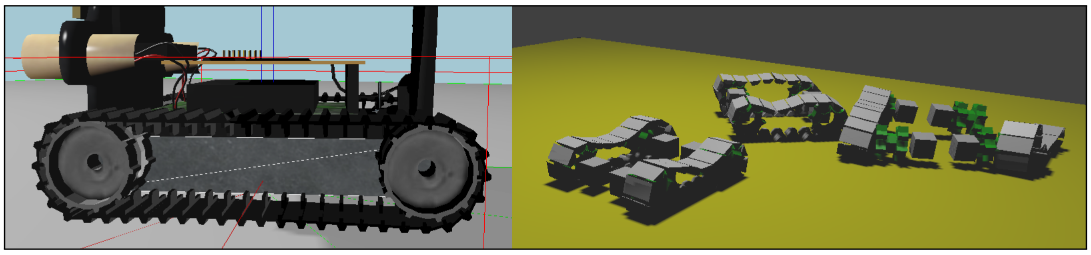

Figure 1: Here you see the to-scale treads modeled after the physical SRV-1 robot on the left and the physics-based rigid-body tracks in motion on the right.

Scale was problematic as the Blender/Bullet physics engine has difficulty with collisions of objects that have a size outside of the assumed range of .05 to 10 meters [6]. Objects smaller than .05 (5cm) Blender/Bullet units, in any given dimension, erratically jitter despite having no force acting upon them.

[3]The atomic commands for the SRV-1 are: "drive forward", "turn counter-clockwise (in place)", "turn counter-clockwise (wide angle)", "turn clockwise (in place)", "drive backward", and "turn clockwise (wide angle)".

As such, since the wheel dimensions of the real SRV-1 are 2.11cm x 2.45cm x 2.52cm, the to-scale 3D model of the SRV-1 was affected by this scale limitation of the physics engine.

To rectify these issues, the physics engine was largely abandoned—in terms of providing the motion model—and was only kept to keep the robots from running through each other and the arena. In its place, a constant linear and angular velocity motion model was developed of which only moves the 3D robot as if it was a single point body. See Figure 2 and Figure 3.

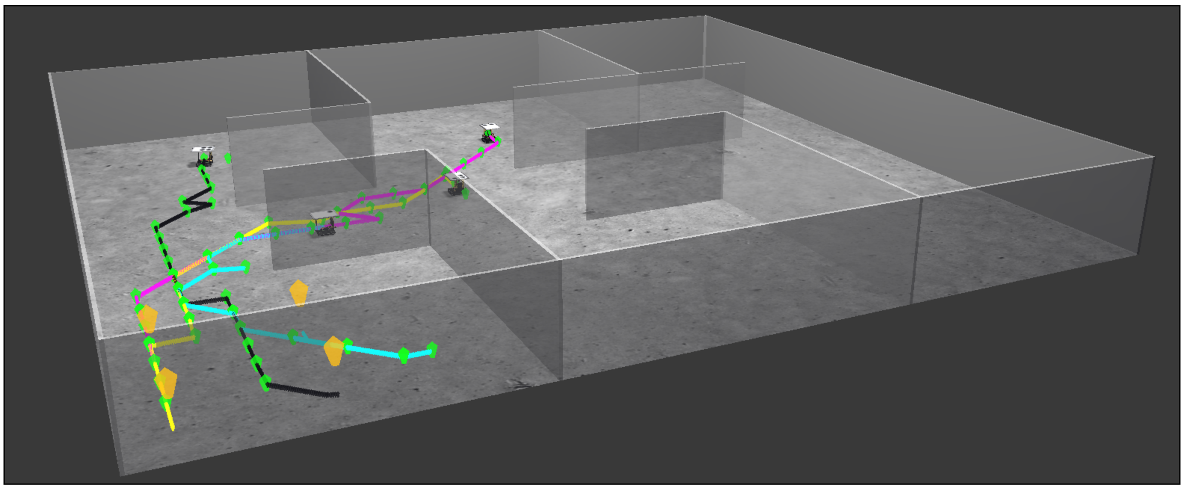

Figure 2: Here you see the constant velocity motion model at work in BlenderSim with four robots traversing their respective paths of which consist of way-points scrapped from a previously recorded physical lab experiment.

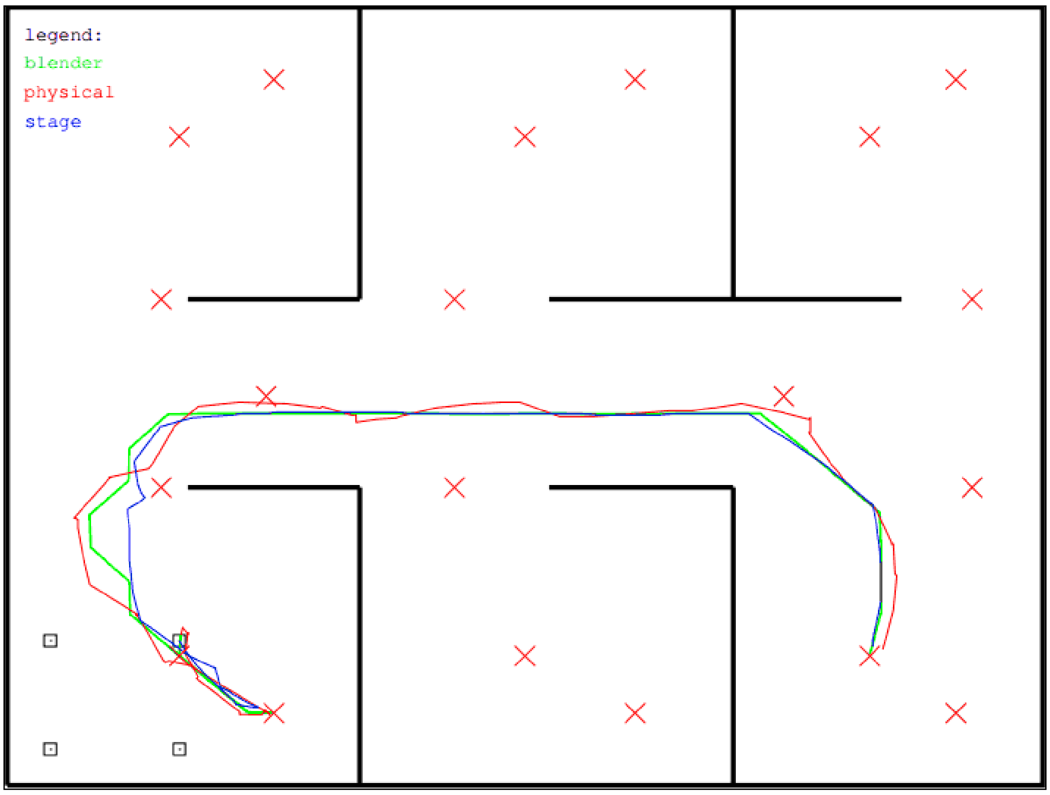

Figure 3: Here you see the path plots–as plotted by Dr. Elizabeth Sklar–of the BlenderSim robot, the physical robot, and the *Stage* (a 2D robot simulator already in use by HRTeam)[5] robot plotted in comparison.

To increase the fidelity of BlenderSim, a probabilistic motion model was discussed. The probabilistic motion model would be generated by recording multiple samples of starting and ending positions per each atomic command performed by the real robot. Once in place, the probabilistic motion model would be used in the GA's fitness function. Here the probability of the simulated physics-based robot's motion would be calculated using the probabilistic robot motion model. For example, say the simulated robot was given the atomic command u and it moved from $M = \{x, y, \theta\}$ to $M' = \{x', y', \theta'\}$. Then the probability that the real robot would also end up at M' would be given by the probability function $P(M'|M, u)$ with the value return by P directly correlating to the simulated robot's genome fitness [16].

3.1.2 CISC. 7900X

In addition to the initial ground work completed concerning the genetic algorithm as outlined in [19], the potential Blender physics engine parameter candidates—to be tuned by the GA—were analyzed for their influence over the physics simulation. To accomplish this goal, a racquetball like environment was constructed in Blender which consisted of a ball, an enclosed arena, and an automated racket controlled via a Python script. See Figure 4. Note that the the ball was allowed to travel in all three dimensions and was completely physics based.

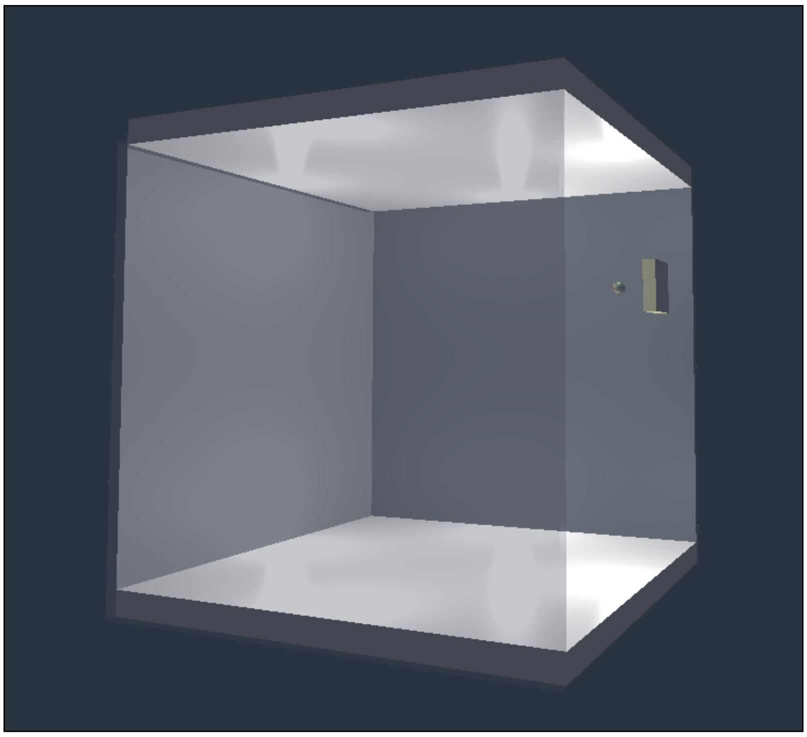

Figure 4: Here you see the racquetball environment used to test the physics engine parameter candidates' influence over the physics simulation.

44 candidate parameters were selected for testing in the racquetball environment. All parameters tested were only associated with the ball. Before all of these parameters were tested for their influence, *nice* values were selected for each such that the ball's behavior was visually normal. A standard was established where the ball was allowed to run for five seconds (with all parameters being set to their nice values) during which its location was recorded at roughly 60 times per second. With the standard established—with which all other runs would be compared to—a parameter was selected (from the candidate pool), its value was tweaked, the ball was run for five seconds with its location being recorded at roughly 60 times per second, and after the run was over, the tweaked parameter's value was set back to its nice value. This sequence was repeated for all 44 candidate parameters.

To compare the recorded tweaked-parameter ball paths with the recorded standard, four methods were utilized to give an indication as to how much influence any one candidate parameter had over the simulation. The first method was visual inspection. All 44 tweaked-parameter ball paths were plotted against the standard. See Figure 5 and Figure 6. The second method was an in-house algorithm, titled the *Lettier distance*, which gives the maximum euclidean distance between two discrete curves P and Q. Informally, imagine holding a rubber band in your hands where the left hand affixes the left end of the rubber band to the first point in P and the right hand affixes the right end of the rubber band to the first point in Q. During each iteration, you advance the left end of the rubber band to the next point in P and you advance the right end of the rubber band to the next point in Q. If the distance grows between point $p_i \in P$ and point $q_j \in Q$, the rubber band stretches but never shrinks. If $|P| < |Q|$ then you hold the left end of the rubber band to the last point in P and continue advancing to the last point in Q and vice versa. Once you reach the last point in P and the last point in Q, the resulting length of the rubber band is the max euclidean distance between P and Q. See Figure 7. The third method was the Fréchet distance and the fourth method was the Hausdorff distance [7] [12].

20 parameters out of the initial 44 showed no significant influence over the 3D physics simulation in Blender. Thus, the resulting 24 parameters which did have a significant influence will be targeted for tuning by the genetic algorithm. See Figure 8.

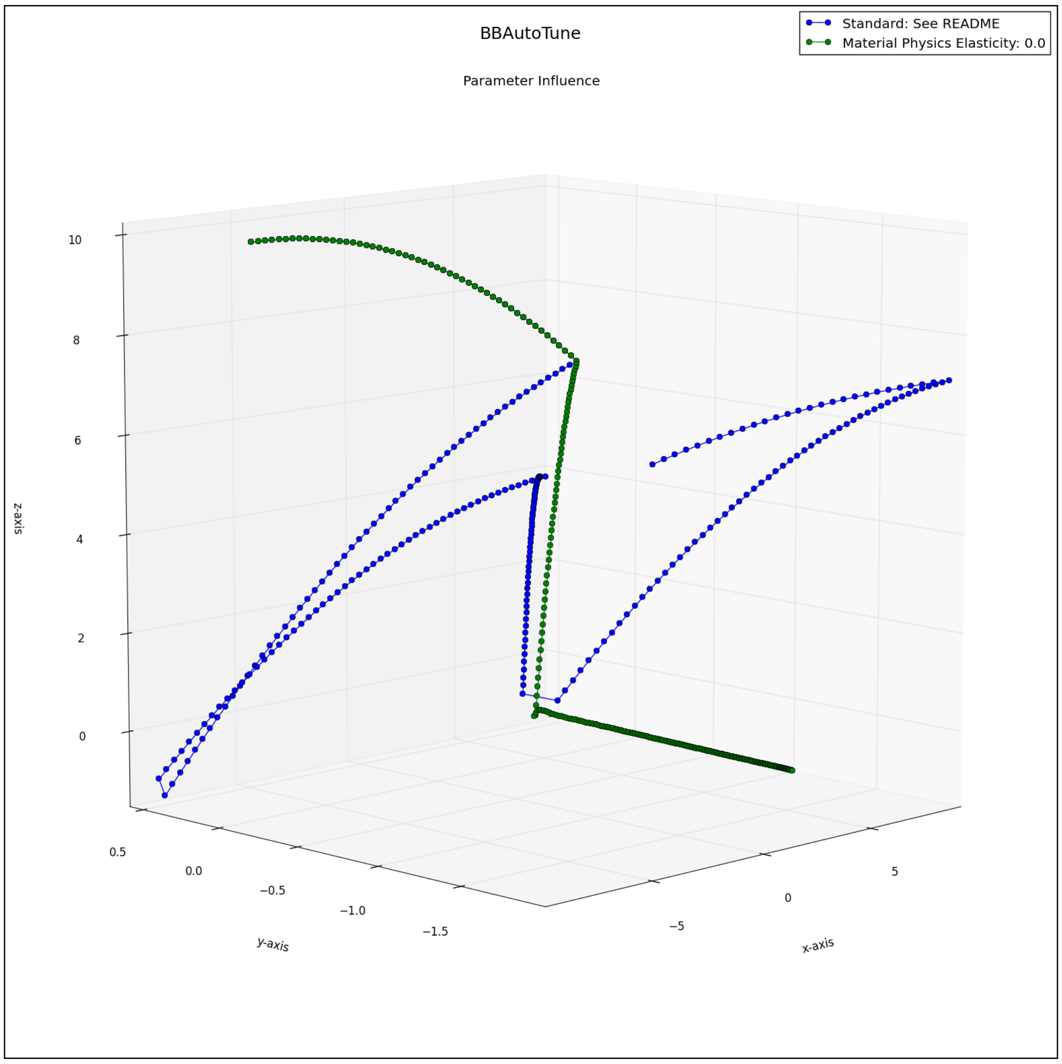

Figure 5: Here you see the dissimilarity of the ball paths between the tweaked, material-physics elasticity parameter and the standard (where no parameters were tweaked).

3.2 Readings of Related Work

Other inspirations for the project were the readings of [23], [10], [20], [17], [21], [24], [13], [11], [22], [26], and [18]. These related works have been separated into three distinct categories below. Category one deals with work concerning a wide range of uses for genetic and evolutionary algorithms. Category two covers the comparison of simulated and physical environments by analyzing their respective "reality gaps". Lastly, category three covers specific work related to automatically tuning physics-engine parameters in order to increase simulation fidelity.

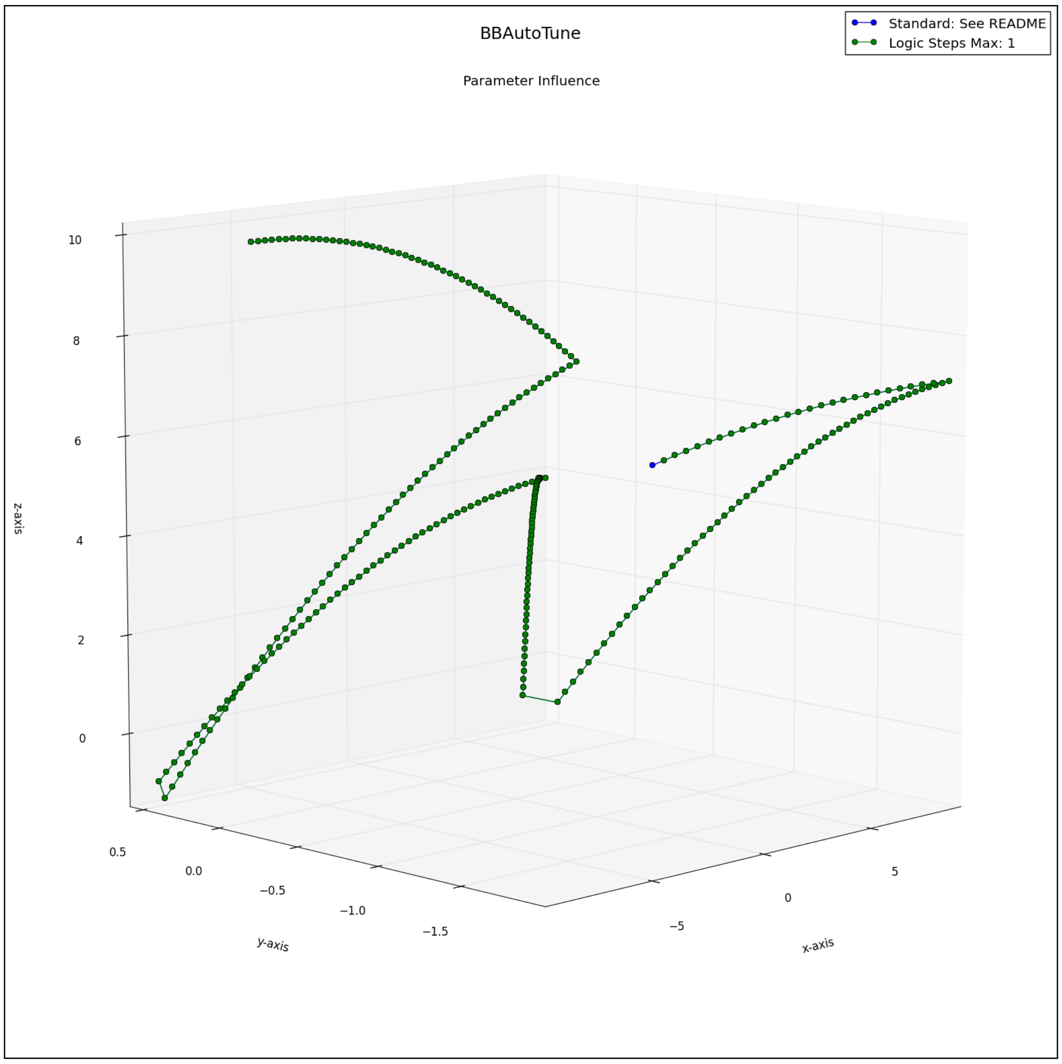

Figure 6: Here you see the similarity of the ball paths between the tweaked, logic-steps max parameter and the standard (where no parameters were tweaked).

3.2.1 Genetic and Evolutionary Algorithm Parameter Tuning

In [23], simulated objects composed of 3D parts, sensors, neural networks, and effectors are generated via a genetic algorithm. These “creatures” are selected and “bred” towards some desired behavior such as swimming, walking, jumping, or following. The genotype of these creatures are represented as directed graphs which are mutated and recombined to produce new generations. Each node in the graph represents some 3D part and within each node is a nested directed graph representing the control of the part. For each desired behavior scenario, each generation offspring is fitness tested which ultimately determines its survival rate into the next generation.

In [10], Mixed Integer Linear Program (MILP) solvers are the target for automatic software parameter optimization although the authors claim their approach could be applied to any algorithm/software where critical parameters with “good” values show improved performance. Utilizing software testing and machine learning paradigms, the authors develop *Selection Tool for Optimization Parameters* (STOP). STOP’s framework is comprised of five phases. Phase one is the initial selection of the MILP solver’s parameters and their settings. Phase two runs the solver, with phase one’s selection, for evaluation. Phase three utilizes regression trees or a neural network to select additional solver parameters and their settings. Phase four reruns the solver with phase three’s additional selection and evaluates again. Finally, phase five outputs the best solver parameters and their settings observed based on solver completion times. Phase one’s selection uses one of three approaches: random selection, pairwise coverage, and a greedy heuristic. The authors concluded that out of the three phase-one selection methods, no method dominated yet pairwise coverage was often associated with the best parameter settings observed. Ultimately,

```
$Algorithm:$
        $P = \langle p_1, p_2, ..., p_n \rangle$
        $Q = \langle q_1, q_2, ..., q_m \rangle$
        $max = 0.0$
        $For\ all\ p_i \in P\ and\ q_j \in Q\ do:$
          $d = ||p_i - q_j||$
          $If\ max < d\ then:$
            $max \leftarrow d$
        $return\ max$
```

Figure 7: Here you see the Lettier distance algorithm.

STOP outperformed the default solver parameters and their settings even when STOP performed at its worst.

In [20], a general synopsis of genetic algorithms is given along with specific examples of their use in various fields of research such as the *Baldwin effect*, *Echo* dynamics, and classifier systems. Lastly, the authors give ideas for future research and additional readings concerning genetic algorithms. The text describes how researchers used a GA model to capture the Baldwin effect where an entity's individually learned behavior populates through successive generations. Other notable portions of the text include how researchers modeled dynamic bug colonies, immune systems, and rat classifier systems all with genetic algorithms.

In [17], background is given on genetic algorithms, genetic programming, and co-evolution where the evolution of one population acts as the environment for another evolving population and vice versa. Prominent works mentioned in the text are the "artificial ant" where a genetic algorithm is used "to discover a finite state automaton enabling the 'artificial ant' to traverse [an irregular] trail" and the development of a *differential pursuer-evader game* where a "genetic programming paradigm can be used to solve the differential game of simple pursuit by genetically evolving a population of strategies for the pursuing individuals over a number of generations." Concerning co-evolution, the authors describe a co-evolution algorithm to simultaneously evolve two players' strategies. "This mutually bootstrapping process found the minimax strategies for both players without using knowledge of the minimax strategy (i.e. any *a priori* knowledge of the game) for either player."

In [21], the authors document their beginning research on how the role of crossover and *fitness landscape* characteristics improve genetic algorithms' performance. The text demonstrates how fitness landscape features of hierarchy, isolation, and conflicts affect genetic algorithm performance of finding global optimums to their class of *Royal Road* functions. Interestingly, they discover that intermediate-order solutions can be detrimental to the genetic algorithm's performance. They had hypothesized that the intermediate solutions were beneficial as they provided "stepping stones" to the optimum but this was not the case in their findings. Here the issue is premature convergence where the population loses useful solutions once one of two disjointed useful solutions is found. They did note however, that for larger problems (then the ones analyzed in the text), the intermediate solutions or *schemas* could be beneficial.

3.2.2 Comparing Physical versus Simulated Environments

In [24], the authors' physical HRTeam lab setup is compared to their simulated Stage lab setup. Comparisons are drawn between physical and simulated experiments with standardized setups across both the simulator and physical environment. These comparisons tested their hypothesis that metrics produced in their simulation can predict the real-world performance in their physical lab. After experimentation, the authors discovered "encouraging agreement at the qualitative level between the results obtained in simulation and those obtained with physical robots." They noted though that this does not necessarily apply to all simulated versus physical comparisons. Their concluding analysis showed that the resulting metrics produced (in both simulated and physical environments) scaled in relative terms but not completely in absolute terms.

In [13], issues are identified with simulation prediction of multi-robot systems. Predictive models

Tweaked Parameter: Value	Lettier Distance	Fréchet Distance	Hausdorff Distance
Gravity: $1.0\frac{m}{s^2}$	12.7741189989	20.948159542	9.86343687719
Physics Steps Max: 1	0.329232644023	0.329232644023	0.329232644023
Physics Steps Substeps: 50	19.3073641073	17.0708640308	11.3180046289
Physics FPS: 1	218.037284056	211.287842927	205.848670021
Logic Steps Max: 1	0.329232644023	0.329232644023	0.329232644023
Physics Deactivation Linear Threshold: 10000.0	0.329232644023	0.329232644023	0.329232644023
Physics Deactivation Angular Threshold: 10000.0	0.329232644023	0.329232644023	0.329232644023
Physics Deactivation Time: 0.0	0.329232644023	0.329232644023	0.329232644023
Occlusion Culling: False	0.329232644023	0.329232644023	0.329232644023
Occlusion Culling Resolution: 1024	0.329232644023	0.329232644023	0.329232644023
Material Physics: False	18.0830473811	18.0830473811	18.0830473811
Material Physics Friction: 100.00	4.93924881608	4.94095279557	3.8109066203
Material Physics Elasticity: 0.0	17.3874844829	17.0233755244	11.3180046289
Force Field Force: 1.00	0.329232644023	0.329232644023	0.329232644023
Force Field Damping: 1.00	0.329232644023	0.329232644023	0.329232644023
Force Field Distance: 20.00	0.329232644023	0.329232644023	0.329232644023
Force Field Align to Normal: True	0.329232644023	0.329232644023	0.329232644023
Physics Type: Dynamic	3.55513601614	18.1217630973	3.275450812
Actor: False	0.329232644023	0.329232644023	0.329232644023
Ghost: True	138.489494312	138.489494312	132.02387242
Use Material Force Field: True	0.329232644023	0.329232644023	0.329232644023
Rotate From Normal: True	0.329232644023	0.329232644023	0.329232644023
No Sleeping: True	0.329232644023	0.329232644023	0.329232644023
Attributes Radius: 1cm	0.329232644023	0.329232644023	0.329232644023
Attributes Mass: 10000.0	3.57800913743	3.30042073543	3.21534852885
Attributes Form Factor: 0.0	3.55513601614	18.1217630973	3.275450812
Velocity Minimum: 1.0	0.329232644023	0.329232644023	0.329232644023
Anisotropic Friction: True	0.329232644023	0.329232644023	0.329232644023
Velocity Maximum: 1.0	17.1034960114	17.130808295	16.8922077042
Damping Translation: 1.0	17.9729741832	17.9729741832	17.9729741832
Damping Rotation: 1.0	8.09457671285	5.47840838373	5.18331655137
Collision Bounds: False	8.81773036062	17.4044978663	2.80531111777
Collision Bounds Margin: 0m	18.740072391	9.79629223411	3.78119352088
Collision Bounds: False	4.28865271192	4.23797419961	3.72440427516
Launch Dynamic Object Settings Force X: 30.0	5.21169989737	5.21169989737	3.78308993247
Launch Dynamic Object Settings Torque X: 30.0	8.82943537929	8.70403741077	6.58255281735
Launch Dynamic Object Settings AngV X: 30.0	2.93219084728	2.63246279507	1.82341393368
Launch Dynamic Object Settings LinV X: 0.0	19.1625050915	19.1625050915	17.0670582483
Launch Damping Frames: -32768	0.329232644023	0.329232644023	0.329232644023
Collision Dynamic Object Settings Force X: 30.0	5.32618850819	18.9696056987	3.80913153981
Collision Dynamic Object Settings Torque X: 30.0	1.67496526126	1.52842619107	1.52842619107
Collision Dynamic Object Settings LinV X: 0.0	19.2801183238	19.1535557726	3.3918725084
Collision Dynamic Object Settings AngV X: 30.0	4.69097055763	18.7403224372	3.80913153981
Collision Damping Frames: -32768	0.329232644023	0.329232644023	0.329232644023

Figure 8: Here you see the distances between each tweaked-parameter ball path and the standard ball path per the three algorithms utilized. The highlighted tweaked parameters were determined not influential.

described in the text were statistical-based models and model-based packages such as Stage, *USARSim*, and *Webots*. The authors carried out comparative analysis of four scenarios—simulated, real, dataset, and a combination thereof—with each having an identical setup. Key issues identified were the modeling of sensor noise, position latency, message latency, environment dynamics, communication interference, and communication bandwidth. Since the text concerns multi-robot systems, the authors focused on communication interference and message latency. They found that unmodeled communication interference effects the predictive performance of a simulation especially for outdoor simulated environments. For message latency, they found that the variance in latency—between high message volume areas and low message volume areas—is much larger in real environments than in simulation. Ultimately, the authors found that simulation is still valid means for verification but discuss future work to model physical interference between multi-robot interactions.

In [11], the pertinent portion of the text is section four which documents the ongoing validation of USARSim's models. These models include robot geometry and kinematics, sensors, environments, and robot interactions. The authors describe previous works by other researchers who sought to validate USARSim as a viable candidate for simulation. In all validation attempts described, the research showed close agreements between the simulated and real-world scenarios. As listed in the text, USARSim was tested on its simulated laser-range-finder sensor, human-robot interaction controls, task times under various environmental conditions, and its simulated camera.

In [22], the authors created two types of robots in USARSim modeled after two real robots. The two simulated robots created were given the equivalent dimensions and masses of the real robots in order to determine USARSim's prediction accuracy of simulated robot motion. After having the simulated and real robots traverse steps and trenches, performance analysis concluded that "USARSim exhibited a performance similar to that of the real robots," indicating that, "the simulated robots can be used for training for the environments with simple obstacles."

3.2.3 Physics Engine Tuning

In [26], the Bullet physics engine is selected for use in a comparison between a simulated robot arm pushing tangram pieces across a table and a physical robot arm pushing tangram pieces across a table. They identify nine physics-engine parameters as having significant impact on the stability of the simulation. Seven of those parameters were selected for optimization. The optimization was done via a simplex algorithm and differential evolution. The resulting optimization reduced the error rate between simulated and physical to less than half when compared with the default parameter settings.

In [18], the authors describe an evolutionary algorithmic approach to tuning *SimRobot*'s physics-engine parameters in order to close the "reality gap". "Notice that rather than finding the real parameters like e.g. the real friction between the robot's legs and the carpet, the aim in this work is to find parameters that overall result in a similar behavior of the simulated robot." The simulator, SimRobot, utilizes the *Open Dynamics Engine* (ODE) for its physics engine. For the experiment, an *AIBO* was chosen and modeled in SimRobot and recorded data from a real AIBO was used in the fitness function of their evolutionary algorithm. Local and global parameters of the physics engine were optimized in stages. The first stage involved learning the *PID controllers* of the AIBO. Here, "the fitness which is a measure for the quality of the matching, is evaluated by summing up the squared differences between the sensor curve of the real robot's movement and the sensor curve of the simulated robot." The second stage involved learning the servo-motor torques needed for realistic movement by running various fitness cycles of AIBO walks and soccer routines. Lastly, with stage one and two completed, the authors set out for an overall convergent, physics-based robot motion model of the simulated AIBO to that of the real AIBO. A set of 60 walks were recorded of the real AIBO for use in fitness testing the evolutionary outcomes (the physics-engine-parameter settings generated) of the simulated AIBO. These 60 walks were broken down into two groups, A and B, where 40 of the walks were put in A for optimization and 20 of the walks were put in B as a control. Group B serves as a control to demonstrate that group A's learned parameters are not performance specific only to A. Experimentation results included an 80 generation convergent rate in stage 1, a 200 generation convergent rate in stage 2, a clear reduction in the maximum and standard deviations of group A, and significant closeness to the real walks recorded in group B. Additional experimentation by the authors included comparing their tuned simulator of the AIBO to that of the simulated AIBO in Webots. While their initial settings in SimRobot were outperformed by Webots, after learning the SimRobot physics-engine parameters, their simulated AIBO significantly outperformed the simulated AIBO in Webots for both sets of walks in A and B.

4 Project Description

This research project will center around Blender's physics engine and the BlenderSim software developed during my REU Fellowship. The hypothesis is that by using using a genetic algorithm to tune the starting state parameters of the Blender/Bullet physics engine, the physics-based motion model of the simulated SRV-1 robot will more closely converge to its real-world counterpart's observable motion.

The fitness function to calculate the fitness of each generational outcome will utilize a probabilistic motion model derived from recording multiple samples of starting and ending positions per each one of the real robot's atomic commands. Given an atomic command (to the simulated robot), the probability that the real robot would have ended up where the simulated robot did end up will be calculated thereby giving its genome's fitness. It is hypothesized, that after many successive generations, fitness will increase.

The software developed for the project will be titled *BBAutoTune* and will augment the BlenderSim software developed over my REU Fellowship. Python will be the language of choice as this is the language utilized by the Blender API.

5 Experimentation and Results

Experimentation will center around the parameters of the genetic algorithm as well as the methodology of heuristically searching optimum physics-engine parameters in order to increase simulation fidelity. Experimental variables include initial population size, selection of physics-engine parameters to represent, chromosome representation of physics-engine parameters chosen, mutation probability rate, recombination probability rate, elitism, fitness analysis method, fitness threshold criteria, and termination criteria.

Results will include:

- The highest fitness obtained, the mean fitness of the population, and the lowest fitness obtained per run per generation produce by the genetic algorithm.
- Convergence rates of the genetic algorithm given different parameters to the genetic algorithm.
- Probabilistic outcomes of repeated trial runs where, per each run, the converged physics-based robot is given a list of way-points and atomic commands.
- Comparative analysis between the tuned physics-based robot motion and the motion of the real robot.

6 Research Plan and Schedule

The practical outcomes of this research project will be a master's thesis proposal (with the aim of completing the thesis in Summer 2014), the experimental work of the master's thesis, and the software infrastructure BBAutoTune capable of carrying out the experimentation as outlined here and for the experiments proposed in the forthcoming master's thesis.

6.1 Spring 2014

Week 1-2	Collection of the real robot atomic command motion data.
Week 3-4	Design and implementation of the probabilistic fitness metric.
Week 5-8	Continued development of the software infrastructure.
Week 9-11	Hypothesis investigation and experimental evaluation.
Week 12-13	Continued master's thesis write-up.
Week 14-15	Term paper write-up.

7 Bibliography

[1] Doc:2.6/manual/game engine - blenderwiki. `http://wiki.blender.org/index.php/Doc:2.6/Manual/Game_Engine`. Accessed August 13, 2013.

[2] Doc:2.6/manual/game engine/physics/object type - blenderwiki. `http://wiki.blender.org/index.php/Doc:2.6/Manual/Game_Engine/Physics/Object_Type`. Accessed August 13, 2013.

[3] Doc:2.6/manual/game engine/physics/object type/dynamic - blenderwiki. `http://wiki.blender.org/index.php/Doc:2.6/Manual/Game_Engine/Physics/Object_Type/Dynamic`. Accessed August 13, 2013.

[4] Surveyor srv-1 open source mobile robot. `http://www.surveyor.com/SRV_info.html`, 2010. Accessed August 13, 2013.

[5] Stage - the player project. `http://playerstage.sourceforge.net/wiki/Stage`, 2011. Accessed August 14, 2013.

[6] Scaling the world - physics simulation wiki. `http://www.bulletphysics.org/mediawiki-1.5.8/index.php?title=Scaling_The_World`, 2012. Accessed August 13, 2013.

[7] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete fréchet distance in subquadratic time. *CoRR*, abs/1204.5333, 2012.

[8] Bart. Fish population data visualisation, internships at great northern way campus, vancouver. `http://www.blendernation.com/2011/11/22/oak-ridge-national-laboratory-blender-on-a-supercomputer/`, 2008. Accessed August 13, 2013.

[9] Bart. Oak ridge national laboratory: Blender on a supercomputer! `http://www.blendernation.com/2011/11/22/oak-ridge-national-laboratory-blender-on-a-supercomputer/`, 2011. Accessed August 13, 2013.

[10] Mustafa Baz, Brady Hunsaker, J. Paul Brooks, and Abhijit Gosavi. Automated tuning of optimization software parameters. Technical report, University of Pittsburgh Department of Industrial Engineering, 2007.

[11] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. *Robotics and Automation, 2007 IEEE International Conference on*, pages 1400–1405, 2007.

[12] Xiao-Diao Chen, Weiyin Ma, Gang Xu, and Jean-Claude Paul. Computing the hausdorff distance between two b-spline curves. *Computer-Aided Design*, 42(12):1197–1206, 2010.

[13] Shameka Dawson, Briana Lowe Wellman, and Monica Anderson. Identification of issues in predicting multi-robot performance through model-based simulations. *Intelligent Control and Automation*, 2:133–143, 2011.

[14] Roland Hess. *The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender*. No Starch Press, San Francisco, CA, USA, 2007.

[15] Jhih Ren. Juang, Wei Han Hung, and Shih Chung Kang. Using game engines for physical–based simulations–a forklift. *Journal of Information Technology in Construction*, 16:3–22, 2011.

[16] Jana Kosecka. Probabilistic robotics: Probabilistic motion and sensor models. `http://cs.gmu.edu/~kosecka/cs685/cs685-motion-models.pdf`, 2013. George Mason University.

[17] John R. Koza. Evolution and co-evolution of computer programs to control independent-acting agents. In Jean-Arcady Meyer and Stewart W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 366–375, Paris, France, 24-28 September 1991. MIT Press.

[18] Tim Laue and Matthias Hebbel. Robocup 2008: Robot soccer world cup xii. chapter Automatic Parameter Optimization for a Dynamic Robot Simulation, pages 121–132. Springer-Verlag, Berlin, Heidelberg, 2009.

[19] David Lettier. Simple Pong Learner (SimPL): Utilizing a genetic algorithm to tune the weights of a feed-forward neural network. 2013.

[20] Melanie Mitchell and Stephanie Forrest. Genetic algorithms and artificial life. *Artificial Life*, 1:267–289, 1993.

[21] Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the First European Conference on Artificial Life*, pages 245–254. MIT Press, 1991.

[22] Shogo Okamoto, Kensuke Kurose, Satoshi Saga, Kazunori Ohno, and Satoshi Tadokoro. Validation of simulated robots with realistically modeled dimensions and mass in usarsim. *Safety, Security and Rescue Robotics, 2008. SSRR 2008. IEEE International Workshop on*, pages 77–82, 2008.

[23] Karl Sims. Evolving virtual creatures. *SIGGRAPH '04 Proceedings*, pages 15–22, 1994.

[24] Elizabeth Sklar, Simon Parsons, J. Pablo Munoz, Tuna Ozgelen, Eric Schneider, Michael Constantino, and Susan Epstein. In *On Transfer from Multiagent to Multi-Robot Systems.* Proceedings of the Workshop on Autonomous Robots and Multirobot Systems (ARMS) at Autonomous Agents and MultiAgent Systems (AAMAS), 2012.

[25] Elizabeth Sklar, Simon Parsons, Arif Tuna Ozgelen, Eric Schneider, Michael Costantino, and Susan L. Epstein. Hrteam: a framework to support research on human/multi-robot interaction. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *AAMAS*, pages 1409–1410. IFAAMAS, 2013.

[26] Erik Weitnauer, Robert Haschke, and Helge Ritter. Evaluating a physics engine as an ingredient for physical reasoning. Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots, pages 144–155. Springer Verlag, 2010.