4-2018

Investigation of Discrete Element Methods for Stud to Turf Interactions

Justin Rittenhouse
Western Michigan University, justin.l.rittenhouse@gmail.com

INVESTIGATION OF DISCRETE ELEMENT METHODS FOR STUD TO TURF INTERACTIONS

by

Justin Rittenhouse

A thesis submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Master of Science
Mechanical and Aerospace Engineering
Western Michigan University
April 2018

Thesis Committee:

Peter Gustafson, Ph.D., Chair
Jennifer Hudson, Ph.D.
Tianshu Liu, Ph.D.

INVESTIGATION OF DISCRETE ELEMENT METHODS FOR STUD TO TURF INTERACTIONS

Justin Rittenhouse, M.S.

Western Michigan University, 2018

Artificial turf is frequently used in sports today and attempts to advantageously replicate a natural grass/soil playing surface. Despite numerous advantages, the technology is correlated with an increase in injuries. The concept behind this research is to create and validate a discrete element model (DEM) for the ground/foot interaction through the use of open source software. After validation, the goal is to analyze an arrangement of studs to determine what role stud geometry plays on torque [N-mm] and force [N] generation. The validation data was provided through laboratory experiments. Three football studs attached to a rigid bracket were turned at 1 degree per second in artificial grass, rubber infill, and grass+infill on a servo hydraulic load frame for 60 seconds. Torque and axial force were sampled at 100Hz.

A trial and error approach was employed to calibrate the elastic modulus, Poisson's ratio, and density until it had a similar output to the validation data provided by the load frame. Through this approach, the DEM model was able to yield results within a 2% difference for average torque relative to the validation data. DEM simulations demonstrated stud geometry appears to play a significant role in torque and force generation.

ACKNOWLEDGMENTS

I would like to thank my family and friends for their guidance and continued support throughout the years. I would also like to thank my committee: Professor Peter Gustafson, Professor Jennifer Hudson, and Professor Tianshu Liu. I would like to thank Dr. Gustafson for his advice and guidance throughout this research, Dr. Hudson for serving as a committee member and for allowing me access to her lab for additional space and resources, and Dr. Liu for serving as a committee member. Lastly, I would like to thank Nicolas Aumonier and FieldTurf for donating all the turf necessary for this research.

Justin Rittenhouse

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

1. INTRODUCTION 1

1.1 Objectives and Thesis Summary 2

1.2 Basic Understanding of Discrete Element Methods 3

2. OPEN SOURCE PROGRAMS 5

2.1 Programs Implemented 5

3. MATERIALS AND METHODOLOGY 10

3.1 Infill Dimensions 10

3.2 Experimental Methodology 11

3.3 Discrete Element Methodology 13

4. RESULTS 22

4.1 Experimental Analysis 22

4.2 Discrete Element Analysis 25

4.3 Comparative Analysis 29

Table of Contents—Continued

5. DISCUSSION 35

5.1 Experimental Analysis.....35

5.2 Discrete Element Analysis.....35

5.3 Future Work.....37

6. CONCLUSION 38

REFERENCES 39

APPENDICES

A. Octave Code for Stud Sizing.....41

B. FreeCAD Drawing of the Studs and Bracket 45

C. Octave Code for Number of Spheres and Time Step 49

D. Octave Code for Post Processing Infill Laboratory Experiments 51

E. Octave Code for Post Processing Grass+Infill Laboratory Experiments 54

F. Octave Code for Post Processing DEM Simulations in Rotation Movement.....57

G. Octave Code for Post Processing DEM Simulations in Translational Movement.....60

H. Python Code for Rotational Infill Simulations.....63

I. Python Code for Translational Infill Simulations.....67

J. Python Code for Grass+Infill Simulations 70

LIST OF TABLES

3.1 Infill Sizing 10

3.2 Results of the Angle of Repose 16

4.1 Slopes for Infill and Grass+Infill 33

5.1 Standard Deviations for Laboratory Experiments 35

LIST OF FIGURES

1.1 BioCore Elite Athlete ShoeSurface Tester [5] 2

2.1 Yade Normally Calculates Stiffness as Two Springs in Series [9] 6

2.2 The Forces in Yade Act at $\bar{C}$ [9] 7

2.3 FreeCAD Drawing of the Round Studs and Bracket 8

2.4 FreeCAD Drawing of the Rigid Box 9

3.1 Studded Assembly used for Laboratory Experiments 11

3.2 Showing Light Contact Between the Studs and Infill 12

3.3 Set up for Laboratory Experiments 13

3.4 Completed Angle of Repose Test 15

3.5 A Pile of Infill After an Angle of Repose Test 16

3.6 Bottom View of the Round Studs and Bracket 17

3.7 The Two Stud Direction 18

3.8 The Three Stud Direction 18

3.9 Zoomed in View of the Artificial Grass 19

3.10 Zoomed in View of the Backside of the Artificial Grass 20

3.11 Backside of the Artificial Grass 21

4.1 Average Torque for Infill Alone 23

4.2 Max Torque for Infill Alone 23

4.3 Average Torque for Grass+Infill 24

4.4 Max Torque for Grass+Infill 25

4.5 DEM Average Torque Results for Infill Alone 26

List of Figures—Continued

4.6 DEM Max Torque Results for Infill Alone 26
4.7 DEM Average Force Results for Infill Alone in the Three Stud Direction 27
4.8 DEM Max Force Results for Infill Alone in the Three Stud Direction 27
4.9 DEM Average Force Results for Infill Alone in the Two Stud Direction 28
4.10 DEM Max Force Results for Infill Alone in the Two Stud Direction 28
4.11 Laboratory Results Compare to DEM Results 29
4.12 Comparing the First Six Degrees of DEM to Laboratory Experiments for Infill Alone ... 30
4.13 Comparing DEM Simulations to Laboratory Experiments in Infill Alone 31
4.14 Comparing the First Six Degrees of DEM to Laboratory Experiments for Grass+Infill ... 31
4.15 Comparing DEM Simulations to Laboratory Experiments in Grass+Infill 32
4.16 Comparing Average Force for the Two Directions of Translational Movement 34
4.17 Comparing Max Force for the Two Directions of Translational Movement 34

CHAPTER 1

INTRODUCTION

Lower extremity injuries are common in sports where cleats are used. The most common NFL injury, excluding quarterbacks, is in the foot and ankle region. This region accounts for 26% of all reported injuries while only 30% of injuries happen above the waist [1]. Regardless of position, 53% of all injuries occur in the knee and lower leg region [1]. Reported injuries as a whole are also increasing. In 2007 the total injury count was just below 1600 this increased to roughly 2200 in 2015 [2]. It is clear injuries are routine in football. This is why injury prevention is a major priority for the NFL.

Athletic activities can cause an increase in strains and stresses on the bones and joints of the athlete relative to everyday activities such as walking. The high number of leg and foot injuries becomes apparent when evaluating the force increase due to athletic events. This force increase is compounded when wearing studded footwear. Running at a rate of 5.4 $m \cdot s^{-1}$ in studded footwear applied a ground reaction force of 2.706 times the runner's body weight while only 2.496 when running in training shoes for an increase of 8.4% [3]. Similarly, running at a rate of 4.4 $m \cdot s^{-1}$ increase the ground reaction forces by 9.3% [3]. The mean mass of the runners in this study was 79.7 kg. This means a total ground reaction force of over 2100 N is applied to each foot for every step while running. It would be a reasonable assumption to state this force would be significantly increased when jumping or when contact is made with other athletes. This could help explain the correlation with an increase in injuries on turf fields [4]. Grass can rip out of the ground acting as a natural release and, theoretically, reduces the number of injuries. A turf field typically has improved grip compared to grass and has less of a natural release. There are numerous advantages to turf fields, such as durability, little maintenance, and more resistance to changes in weather. These advantages have the popularity of turf field thriving. Therefore, research to improve turf safety must be continuous.

1.1 Objectives and Thesis Summary

There are three main objectives of this research. The first is to determine whether the discrete element method (DEM) can accurately simulate the interaction between turf and studded footwear. The use of DEM in this fashion has not been done publicly at the time of writing this thesis. The current process of testing turf is costly and requires specialized resources such as the BioCore Elite Athlete ShoeSurface Tester (BEAST). The BEAST can be seen in Figure 1.1 below [5]. A validation process between laboratory experiments and DEM models was done to determine the accuracy of the DEM model. The bench level laboratory experiment was done using three football studs in a rigid bracket and torqued on a servo hydraulic load frame. This experiment was done for three scenarios: just artificial grass, rubber infill, and grass+infill. The data collected from these experiments were used to create a DEM model.

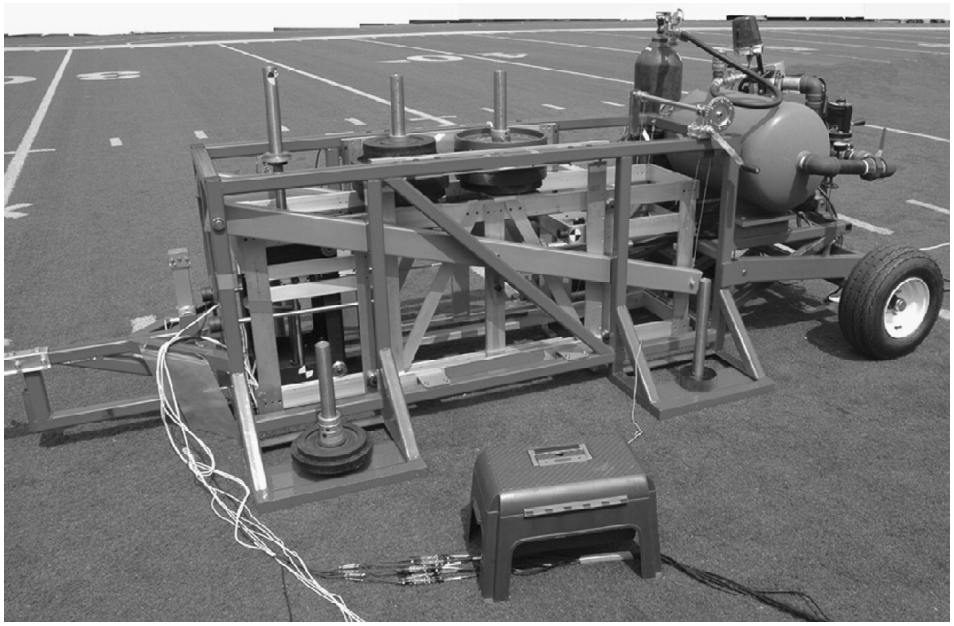

Figure 1.1: BioCore Elite Athlete ShoeSurface Tester [5]

The second objective of this thesis is to evaluate six stud designs through DEM simulations. The stud designs include round, square, hex, wedge, cross, and bladed shape studs. A constant volume and height were used for all stud designs. The width and length were held close as possible to the diameter of the round studs with the constant volume and height. All studs are tapered with

the bigger end attach to the bracket. The Octave code used to calculate dimensions can be seen in Appendix A. The round studs can be seen in Figure 2.3 while the other five can be seen in Appendix B. Each design went through a torsional and two translation load scenarios. The torsional scenario mirrored the laboratory experiments and was used to validate the models. Filtered results of torque and force were evaluated as indicators of stud grip.

Another interesting aspect of this research is to implement the use of open source programs. Open source programs are an incredible resource that are fully capable of research level engineering [6]. Excluding the software required for the laboratory experiments, this thesis was done exclusively in open source programs. It should be noted there are open source alternatives to using the load frame chosen. These alternatives include designing one from an open source microcontroller or a pre-built machine such as the FreeLoader [7], but due to time limitations and cost, the author of this research chose to use the load frame that was freely available. The open source programs used are outlined in Chapter 2.

The materials and methodology used are summarized in Chapter 3. Section 3.1, focuses on the particle size for artificial turf infill. Section 3.2 focuses on the experimental aspect while Section 3.3 describes the creation and the validation process of the DEM model. Section 3.3 outlines a simple experiment and simulation that was tested, to get an understanding of Yet Another Dynamic Engine (Yade), the DEM process, and to find the angle of repose for rubber infill. Chapter 4 covers the results found through this research. Chapter 5 covers the discussion, next steps, and future work. The conclusion can be seen in Chapter 6.

1.2 Basic Understanding of Discrete Element Methods

DEM is often used for modeling granular materials. Normally, spheres are used in the DEM simulations, though other shapes are an option but typically comes with an additional computational expense. The nature of granular materials, which inherently has voids is why DEM was chosen over other modeling options such as the finite element method (FEM). FEM is often used for continuum materials. Though, some research has focused on modeling granular materials as a continuum material through FEM. This research is regularly compared against or calibrated with DEM models [8].

The spheres in DEM simulations require calibration, and this is often done via laboratory experiments on the granular material of interest. The results from those experiments are used to develop a DEM model of the material. A trial and error approach is employed to calibrate input parameters such as elastic modulus, Poisson's ratio, and density until DEM results are similar to the laboratory experiments. This is a necessary step because the particle size in the DEM model is often significantly larger than the real-life particle size of the material it represents. This scaling of particles is to increase step size and decrease total computational expense. Once the material in the computer model is calibrated, the model can be used to simulate different scenarios.

A DEM model is an iterative process with a set number of stages. The first stage often sets all forces to zero to ensure the forces in each iteration are calculated separately. The next stage looks for contact the material is producing, either with itself or an object within the model. There are several different algorithms for contact detection. Next, forces are calculated from the contact and the resulting velocities of the particles are found. These velocities are used to find displacement for a given time step. This process is repeated until a given number of iterations is reached. A more in-depth outline of a DEM model can be seen in Chapter 2. There the model used is broken down in detail.

CHAPTER 2

OPEN SOURCE PROGRAMS

2.1 Programs Implemented

Yet Another Dynamic Engine (Yade) is the discrete element program implemented for this research. Yade was chosen over other open-source DEM programs due to the authors familiarity with Python. Yade is built with Python, and Python syntax is used for scripting. Therefore, the two programs work well together, this was an immense benefit due to the computational expense of DEM simulations. This expense made hyper-threading the script crucial. The Yade script was inserted inside of a Python script for hyper-threading purposes. The Yade executable was imported and with just an additional few lines of code, the script was hyper-threaded.

It is important to understand how the DEM portion of the Yade script works. This portion is the core of the script and is referred to as the engine that is being executed. The first keyword is ForceResetter. The engine is an iterative process and the forces often need to start back at zero after each iteration. This is done through the force resetter. This ensures there is no residual force from a previous iteration. The next keywords are the Bo1 terms. These commands are placed within the InsertionSortCollider part of the engine loop. They create axis-aligned bounding boxes used in a sweep and prune algorithm for collision detection [9]. The next part of the engine is the InteractionLoop. This loop has three stages. The first stage of this loop are the terms Ig2. This stage is for exact collision detection algorithms and represents the intersection of contact points. The second stage includes the Ip2 terms. These terms dictate the physics for internal and external contact, such as friction. The last stage includes the Law2 terms. These terms dictate the governing laws. In this case, Cundall-Strack was implemented [9, 10].

The last part of the engine loop is where forces are calculated. The stiffness equation for the force calculations can be seen in Equation 2.1 while Figure 2.1 is there to help visualize the

stiffness equation [9]. The shear stiffness is computed as a percentage of normal stiffness. The equations used by Yade to calculate forces and torques can be viewed in Equation 2.1 through Equation 2.6 [9]. Noting subscripts N and T denotes normal and shear, respectively. It is important to note the forces act at $\overline{C}$ and not the spheres' centers as seen in Figure 2.2 [9]. This generates a torque that must be considered. After these forces are computed, the new locations for the spheres are determined by the use of Newton's second law. There are three keywords to note in this section: NewtonIntegrator, TranslationEngine, and RotationEngine. The NewtonIntegrator applies non-viscous damping to the sphere through Cundalls damping coefficient and applies gravitational acceleration to the spheres [9]. The TranslationEngine and RotationEngine are self explanatory, in-which, they apply translational and rotational movement to the studs and bracket. Once all the spheres move to their new location due to their found displacements, the engine loops back to the force resetter term and repeats. This iterative process repeats until a set number of iterations is reached.

$$K_N = \frac{E_1 l_1 E_2 l_2}{E_1 l_1 + E_2 l_2} \tag{2.1}$$

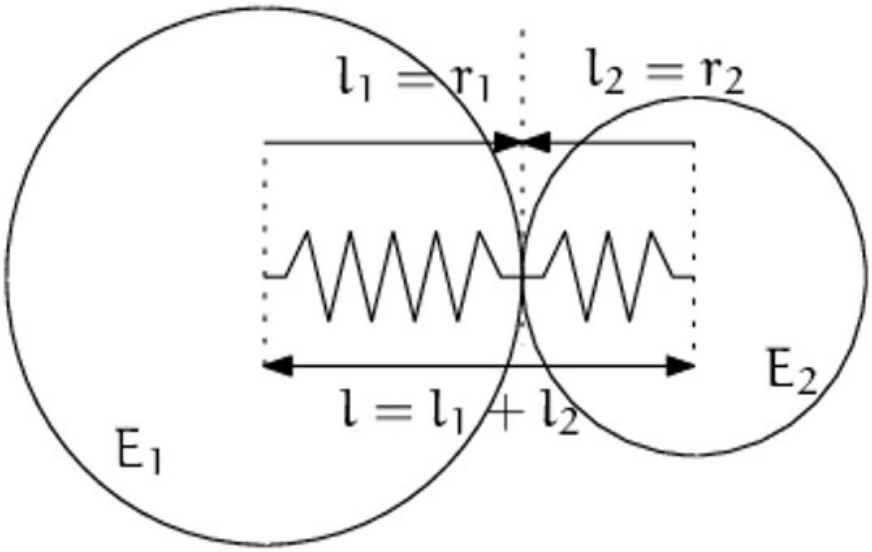

Figure 2.1: Yade Normally Calculates Stiffness as Two Springs in Series [9]

$$F_N = K_N U_N n \tag{2.2}$$

$$F_T = K_T U_T \tag{2.3}$$

$$F = F_N + F_T \tag{2.4}$$

$$T_1 = d_1(-n)xF \tag{2.5}$$

$$T_2 = d_2(n)xF \tag{2.6}$$

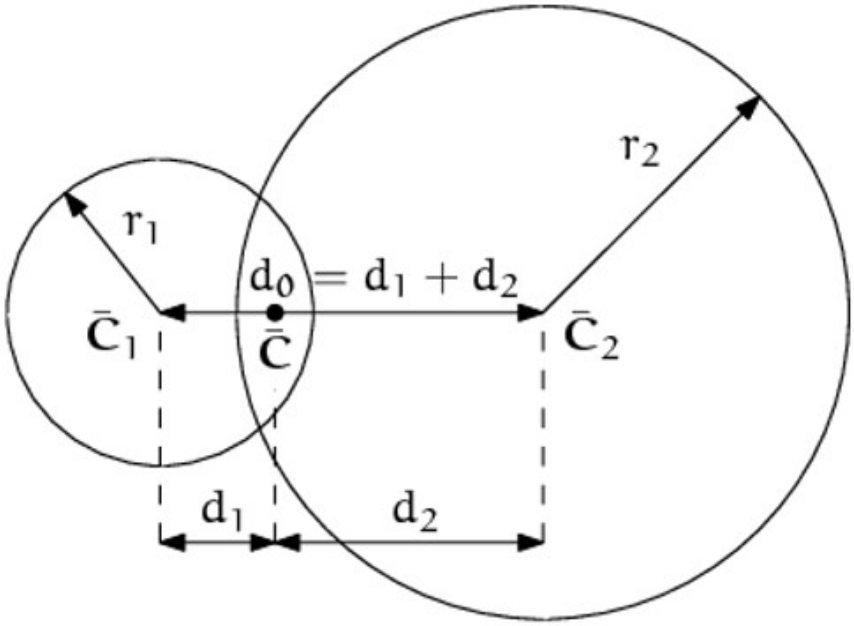

Figure 2.2: The Forces in Yade Act at $\overline{C}$ [9]

Additional open source programs implemented were FreeCAD, Gmsh, and Octave. FreeCAD was used for all the computer aided drawings (CAD). The drawings include the rigid bracket, the studs, and rigid box. The bracket, round studs, and box can be seen in Figure 2.3 and Figure 2.4

below. Gmsh was implemented to surface mesh the drawings. Octave was the other open source program implemented. Octave was implemented frequency for everything from simple calculations to post-processing of the data. The code to calculate ball size and number of iterations to run can be seen in Appendix C below. The octave codes to post-process the data collected via the hydraulic load frame for just infill and grass+infill can be seen in Appendix D and E below, respectively. The octave codes to post process rotation and translation movement can be seen in Appendix F and Appendix G, respectively.

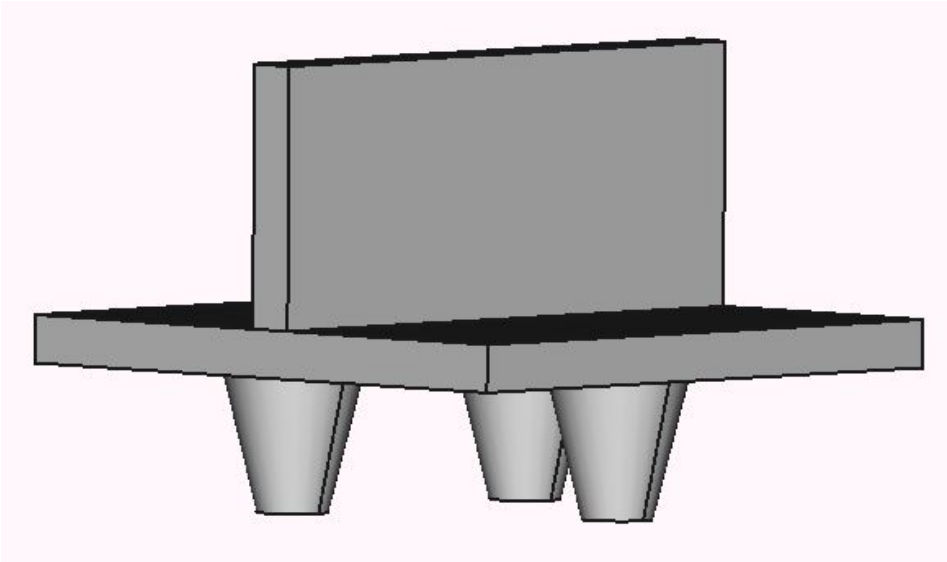

Figure 2.3: FreeCAD Drawing of the Round Studs and Bracket

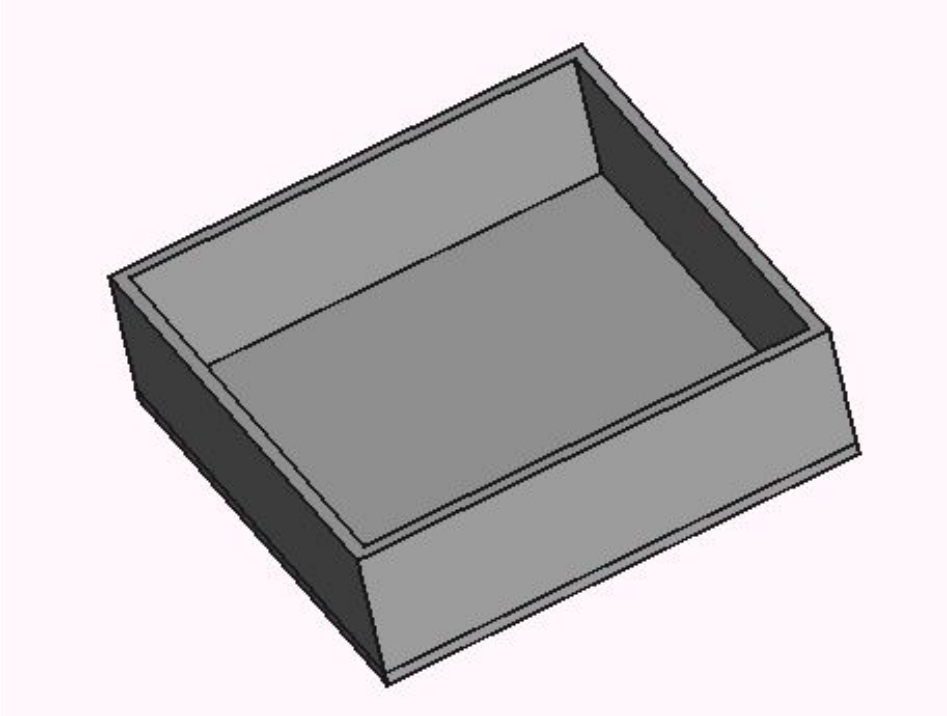

Figure 2.4: FreeCAD Drawing of the Rigid Box

CHAPTER 3

MATERIALS AND METHODOLOGY

3.1 Infill Dimensions

Turf has three main parts: infill, artificial grass, and backing to the artificial grass. Infill is the material between the blades of artificial grass. Infill material has a broad range that includes: cork, coconut, thermoplastics, and rubber [11]. Cryogenic rubber (SBR) was used for this research due to its popularity. SBR is made from ground up recycled tires. The ground up rubber is cryogenically frozen then shattered into small particles [11]. The infill was measured using a series of sieves. A 500 g sample was used, and the sieves were left on a shaker for five minutes. The results of this testing can be seen in Table 3.1 below. The particle diameter range for the infill is 0.25-1.68 [mm].

Table 3.1: Infill Sizing

Sieve Number	Cumulative Retain [g]	Cumulative [%]	Passing [%]	Retain [g]
#10 [2 mm]	0	0	100	0
#12 [1.68 mm]	.5	.1	99.9	.5
#14 [1.41 mm]	25	5	95	24.5
#16 [1.19 mm]	136	27.2	72.8	111
#18 [1.00 mm]	234.5	46.9	53.1	98.5
#20 [0.85 mm]	342.5	68.6	31.4	108
#40 [0.425 mm]	499	99.9	.1	156.5
#60 [.25 mm]	499.5	100	0	.5
#100 [.15 mm]]	499.5	100	0	0

3.2 Experimental Methodology

A bench level laboratory experiment used three football studs in a rigid bracket to provide validation data to a discrete element model of the same. The studded assembly can be seen in Figure 3.1 below. The studded assembly was lowered until the studs made light contact with the artificial grass. The servo hydraulic load frame then turned at 1 degree per second for 60 seconds about the vertical axis. The torque and axial force were sampled at 100 Hz. Once the data was post-process it was clear artificial grass alone has a negligible effect on the studs. This was intuitive but was done for completeness.

Figure 3.1: Studded Assembly used for Laboratory Experiments

This same experiment had to be redone for the infill and grass+infill. The studded assembly was lowered until the studs made light contact with the infill. This can be seen in Figure 3.2 below. Controlled displacement was then implemented to lower the studs into the infill. This was done for two displacements. The first being the height of the stud at 19.05 [mm]. The second depth was 17 [mm]. The depth of 19.05 [mm] ensured the studs had full penetration into the infill. The latter being slightly shorter to allow clearance of the steel bracket. The rate of turn and sample rate were the same as above. These experiments were done ten times for each combination of material and

stud depth. Therefore, a total of forty tests were carried out. The set up for the infill experiments can be seen in Figure 3.3 below.

Figure 3.2: Showing Light Contact Between the Studs and Infill

Figure 3.3: Set up for Laboratory Experiments

3.3 Discrete Element Methodology

The first DEM model had to be validated, therefore the model had to mimic laboratory experiments. The bracket, studs, and box were measured and recreated in FreeCAD and meshed in Gmsh. The CAD drawings can be seen in Figure 2.3 and Figure 2.4, respectively. The DEM model was then able to lower and rotate the studs in a fashion that mimicked laboratory experiments.

This allowed for a myriad of simulations to run. These simulations included changes for the elastic modulus, Poisson's ratio, friction angle, the density of the material and particle size. It was determined through these simulations that changes in density and a radius of 4 [mm] for particle size would be implemented. It should be noted smaller particle size would work as well, but due to an increase in total simulation time a smaller size was not chosen. A model with a particle radius of 1 [mm] takes over a week to run on computers that were available for this research. This was approximately the size of the real-life infill particles, but this time frame was too long. And as the results in Chapter 4 show, a radius of 4 [mm] could produce results comparable to laboratory experiments. Scaling of the particle size is a common practice due to the computational expense associated with DEM [12]. The starting spheres packing density for the simulations were roughly 65%. This is the average packing density when spheres fall randomly. Kepler conjecture packing factor of roughly 74% was not used due to the fact the infill was poured into the rigid box randomly and not stacked to achieve such a high packing factor [13].

The model implemented held elastic modulus, Poisson's ratio, and friction angle constant for all simulations. The infill is made from recycled rubber. Therefore, elastic modulus and Poisson's ratio are well known values. The elastic modulus of silicone rubber is between 0.001 and 0.05 GPa while Poisson's ratio is between 0.47-0.49 [14]. The last unknown was the friction angle. This is often referred as the angle of repose. Therefore, the friction angle was found doing an angle of repose experiment with the infill. Infill was slowly poured into a yellow funnel until the infill just touch the bottom of the funnel. This experiment was done three times and can be seen in Figure 3.4 below.

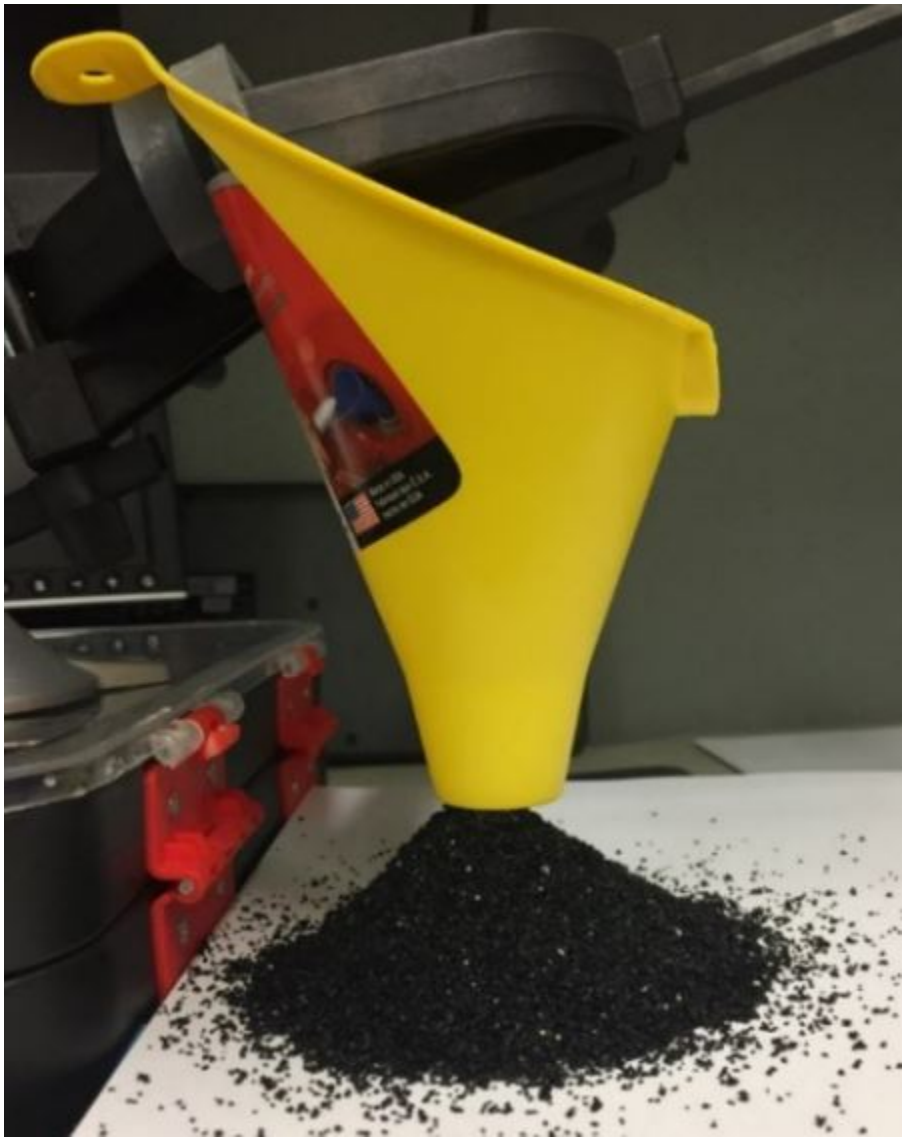

Figure 3.4: Completed Angle of Repose Test

The base of the funnel to the paper was measured beforehand. Therefore, the height of 38 [mm] was the same for each pile. Next, the base diameter for each pile was required. This was done by drawing four straight lines tangent to the base of the pile. This approach can be seen in Figure 3.5 below. The approach of using four lines was chosen to allow the measurement of two diameters per sample. This improved the accuracy of the measurement since the base was not a perfect circle. The average of the two sides were calculated and used as the diameter. The angle of repose or angle of friction was then calculated with the known height and diameters of the base.

The results of all three experiments can be seen in Table 3.2 below. The nature of DEM models does not require this degree of accuracy for input parameters. A common angle of repose could have been chosen between 30-40°. This was done for an alternative motive. The author of this paper was unfamiliar with Yade and DEM. This experiment was created in Yade and simulated until an improved understanding of Yade and DEM was achieved.

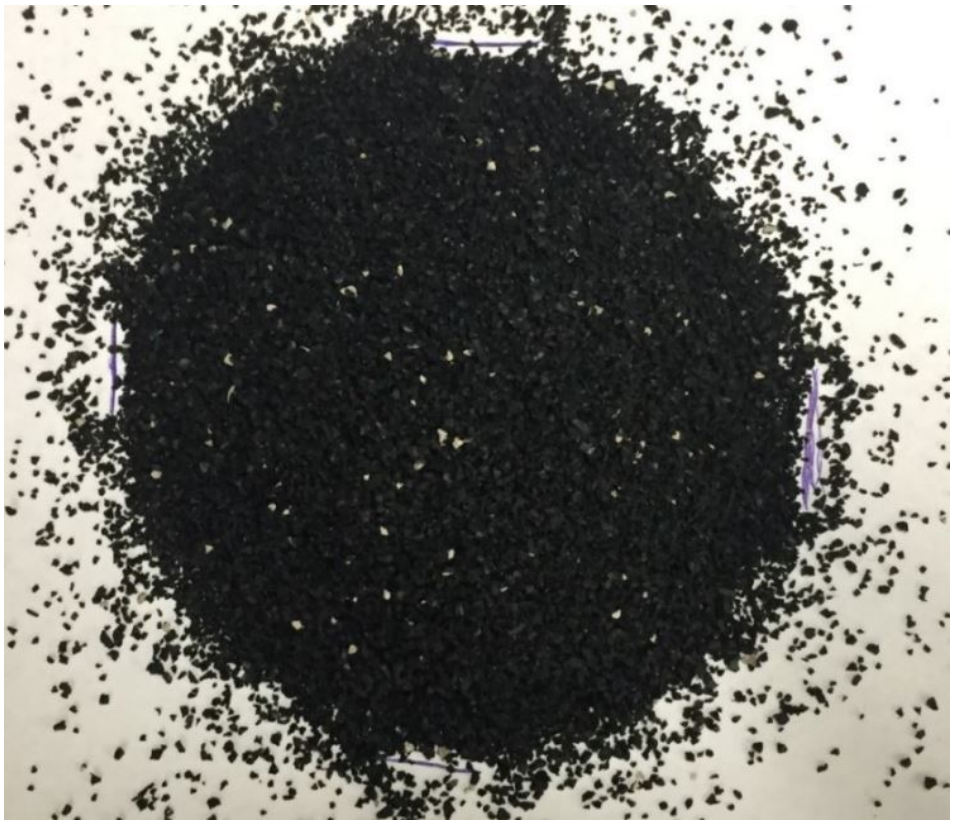

Figure 3.5: A Pile of Infill After an Angle of Repose Test

Table 3.2: Results of the Angle of Repose

Experiments	Diameter of Base [mm]	Angle of Repose [Degrees]
Experiment 1	94.5	38.8
Experiment 2	97	38.1
Experiment 3	100	37.2

The complete python script for rotational movement can be seen in Appendix H. The second round of simulations involved translational movements. The studs were lowered into the spheres just as in the rotational model. This time, instead of rotating, the studs went through a translational movement. The studs moved at 2.58 $m \cdot s^{-1}$ for 0.4 seconds, as with published literature [15, 16]. The velocity of 2.58 $m \cdot s^{-1}$ was the mean velocity of 11 youth soccer players when making cuts off the forefoot. These cuts were filmed at 1000 $frames \cdot s^{-1}$ [16]. The translational movements were done in two directions. This is referred to as the two and three stud directions. This is in reference to how many studs can be viewed when looking straight on in that direction. An improved understanding can be had by looking at Figures 3.6, 3.7, and 3.8 below. The results for translational movements can be seen in Chapter 4. The python script for translational movement can be seen in Appendix I.

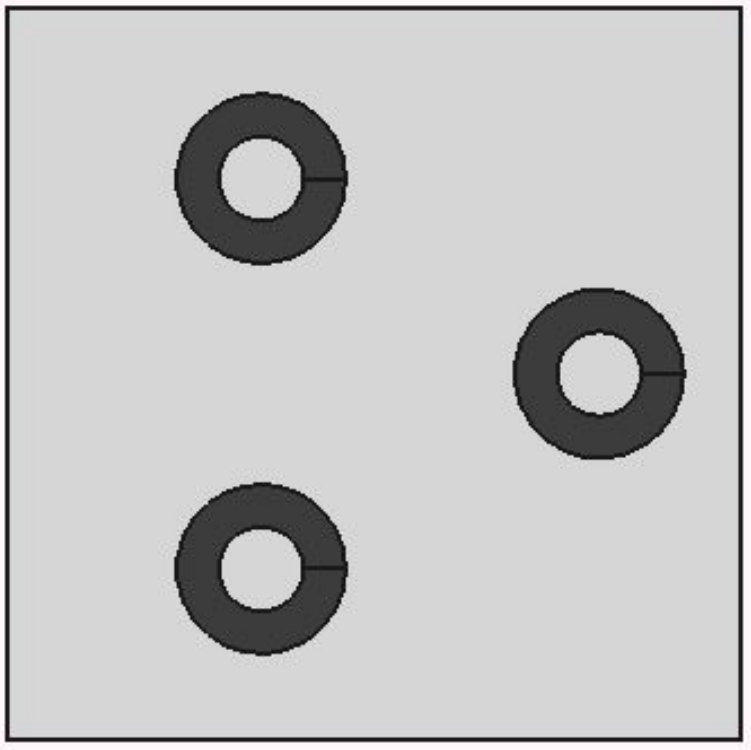

Figure 3.6: Bottom View of the Round Studs and Bracket

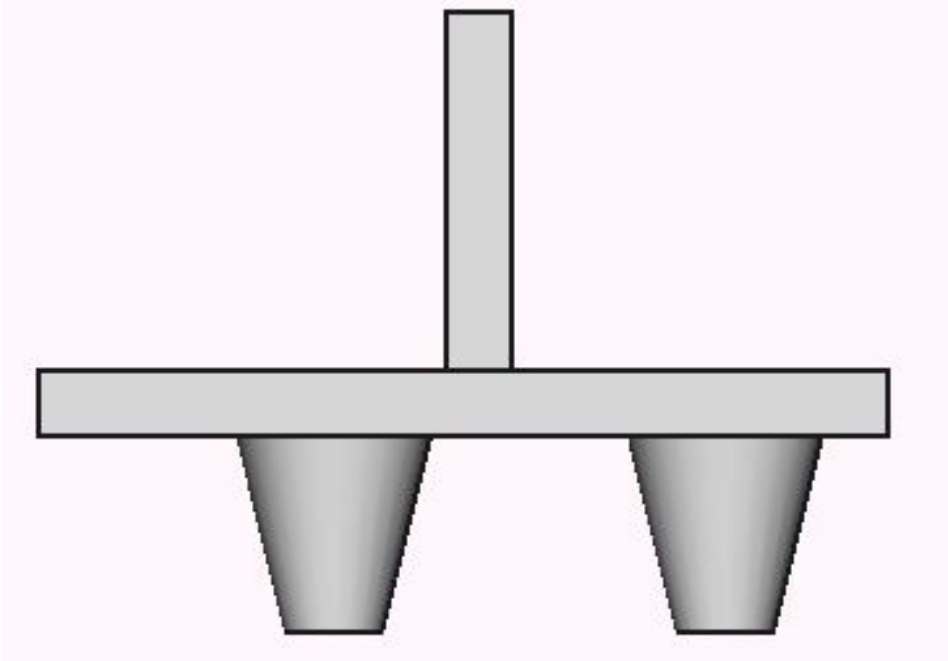

Figure 3.7: The Two Stud Direction

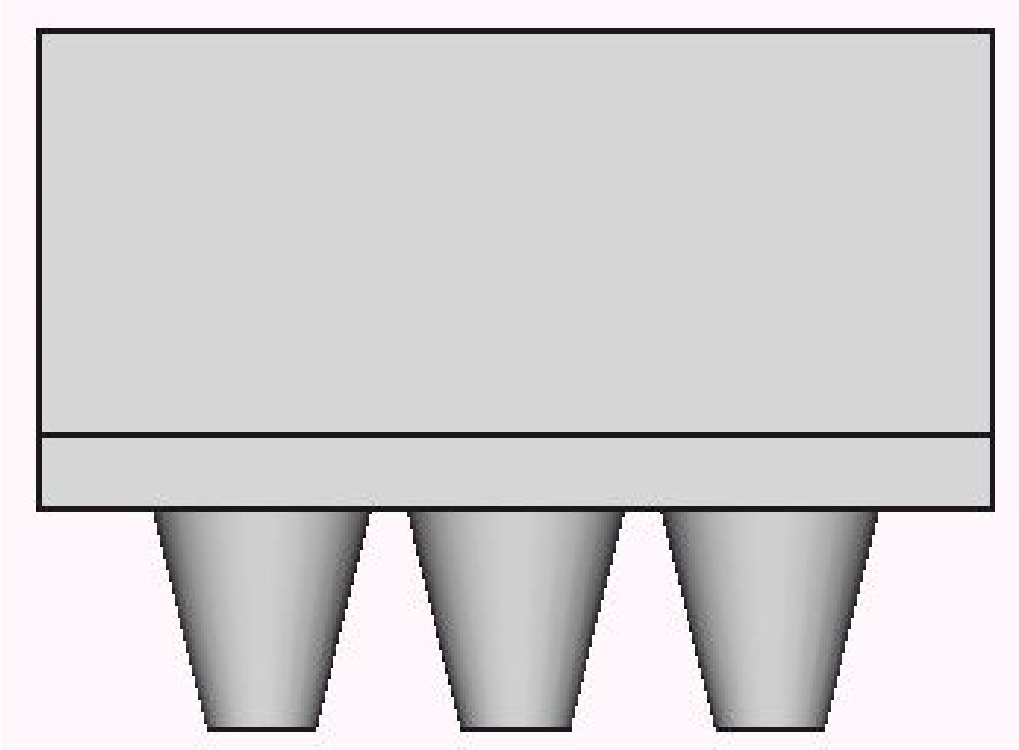

Figure 3.8: The Three Stud Direction

The artificial grass was modeled as rigid "beams" and had zero degrees of freedom. This was not the desired approach. The reason behind this approach can be seen in Chapter 5 while the code can be seen in Appendix J. The dimensions of the grass were chosen based off the dimensions of a patch of artificial grass. The grass is attached to the turf in patches. This can be seen in Figures 3.9 and 3.10 below. Figure 3.10 is a zoomed in view of the backside of one patch of artificial grass. The idea was to model the artificial grass as one beam, instead of many little ones. This was to save computational expense. The artificial grass used for the DEM model had a diameter of 2.3812 [mm]. The grass spacing was 9 [mm] apart in one direction and 19.05 [mm] in the other. These were the average distances to the center of each patch of grass. A backside of the artificial grass can be seen in Figure 3.11 below.

Figure 3.9: Zoomed in View of the Artificial Grass

Figure 3.10: Zoomed in View of the Backside of the Artificial Grass

Figure 3.11: Backside of the Artificial Grass

CHAPTER 4

RESULTS

4.1 Experimental Analysis

In contrast to grass alone, the results from infill alone are significant. Average and max torque results for infill alone for both stud depths can be seen in Figures 4.1 and 4.2 below. A third order Butterworth low pass filter with a cutoff frequency of 20 Hz was used on the collected data. A stud depth of 19.05 [mm] increased by over a factor of 2 in average torque and over a factor of 4 in max torque relative to stud depth of 17 [mm]. The p values for the average and max torques were both $< .002$. In both cases, rejecting the null hypothesis that the samples were similar. This was because the steel bracket played a large role in the torque generated. The bracket started at lightly touching the infill but as it rotated the bracket dug into the infill. This was undesirable because the goal of this research was to isolate the studs. Therefore, the 17 [mm] depth experiments were done as well. It was clear from a visual aspect and from the data that the bracket was clear of the infill at a depth of 17 [mm]. The results from the 17 [mm] depth were chosen to calibrate the infill alone DEM model. Lastly, Figure 4.13 shows the filtered data from all ten infill alone experiments. The black line is the average.

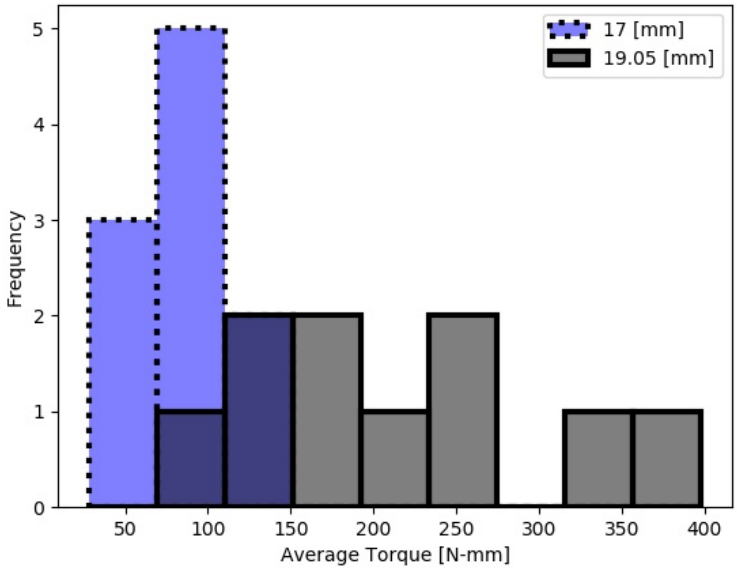

Figure 4.1: Average Torque for Infill Alone

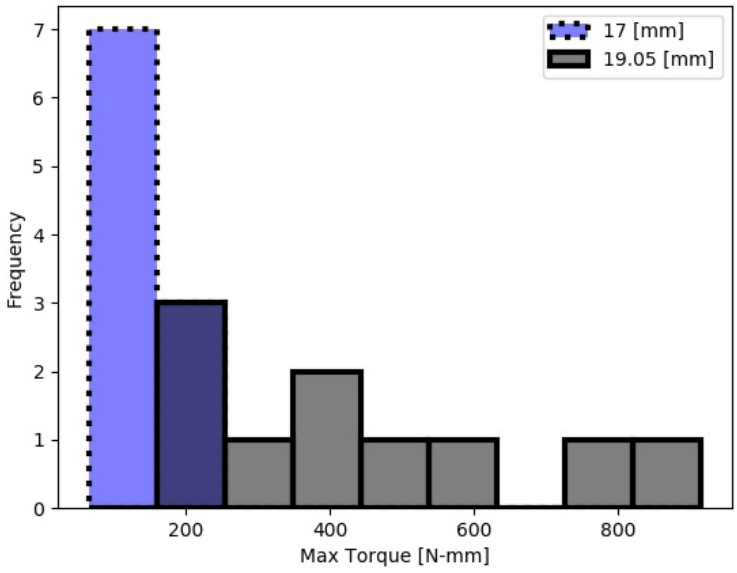

Figure 4.2: Max Torque for Infill Alone

The grass added to the infill played a significant role on torque generation. The average

torque increased by over a factor of 13 and had a p value of $<< 0.001$ for a stud depth of 17 [mm]. Max torque increased by a factor of over 16 and had a p value of $<< 0.001$. In both cases, rejecting the null hypothesis that the samples were similar. The average torque for grass+infill can be viewed in Figure 4.3 while max torque can be seen in Figure 4.4. In this case, the average and max torques for the different depths were more consistent. In both cases, the difference was less than a factor of 1.5. The p values for the average and max torques were both $> .05$. In both cases, failing to reject the null hypothesis that the samples were similar. In an effort to be consistent, the results for the 17 [mm] depth was chosen to calibrate the grass+infill DEM model. Lastly, Figure 4.15 shows the filtered data from all ten grass+infill experiments. The black line is the average.

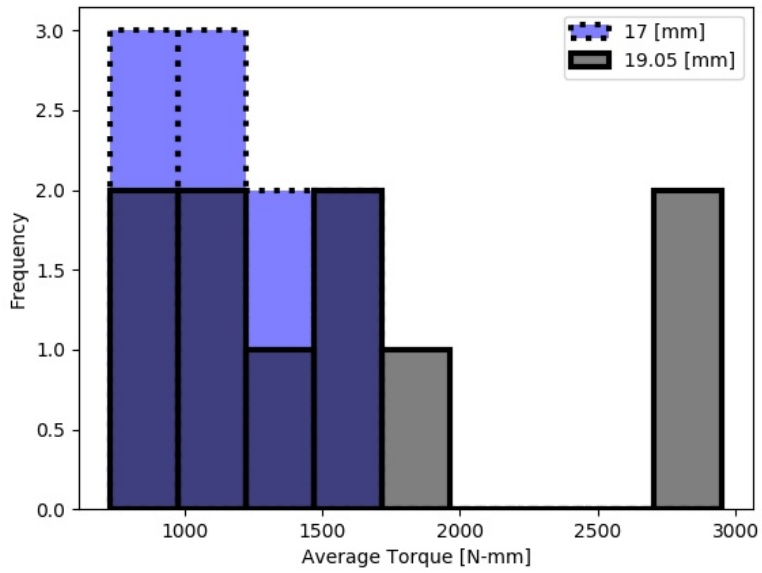

Figure 4.3: Average Torque for Grass+Infill

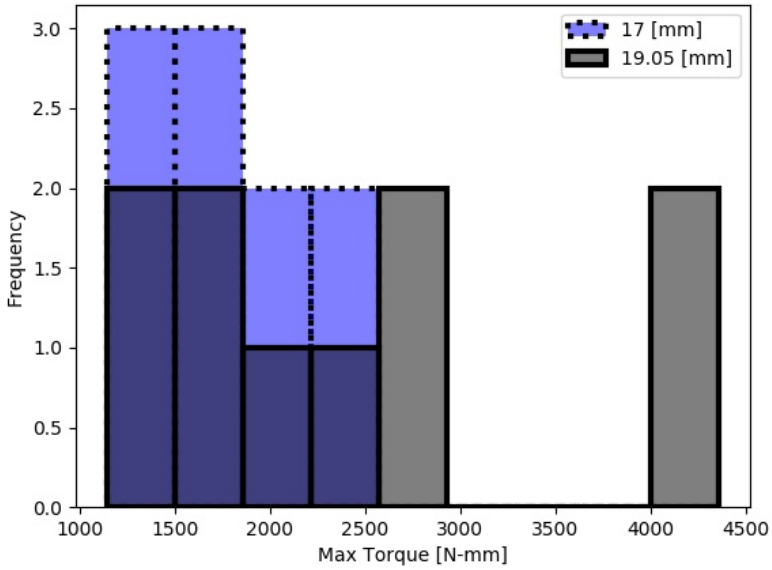

Figure 4.4: Max Torque for Grass+Infill

4.2 Discrete Element Analysis

DEM average and max torque results for infill alone can be seen in Figures 4.5 and 4.6 below. The round studs can be seen in Figure 2.3 above while the other five studs can be seen in Appendix B. DEM translational results for the three stud direction can be seen in Figures 4.7 and 4.8 below. Similarly, Figures 4.7 and 4.8 below show the results for the two stud direction. These tables show average and max torque or force and their relative difference to round studs.

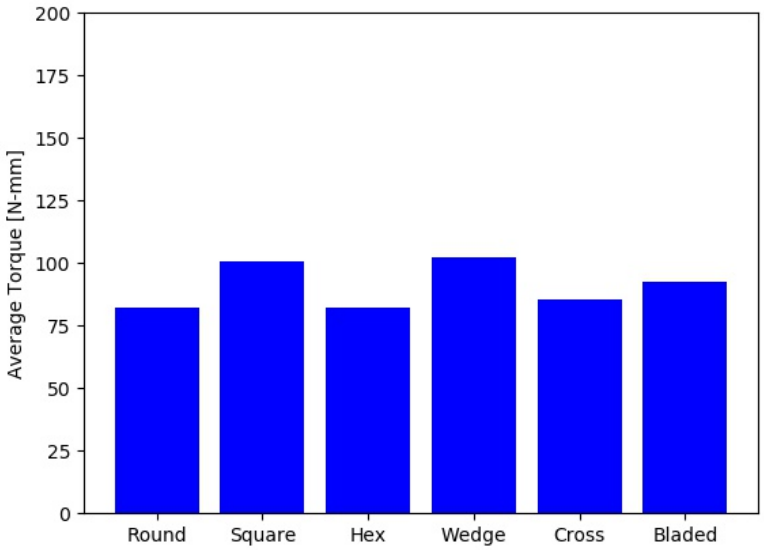

Figure 4.5: DEM Average Torque Results for Infill Alone

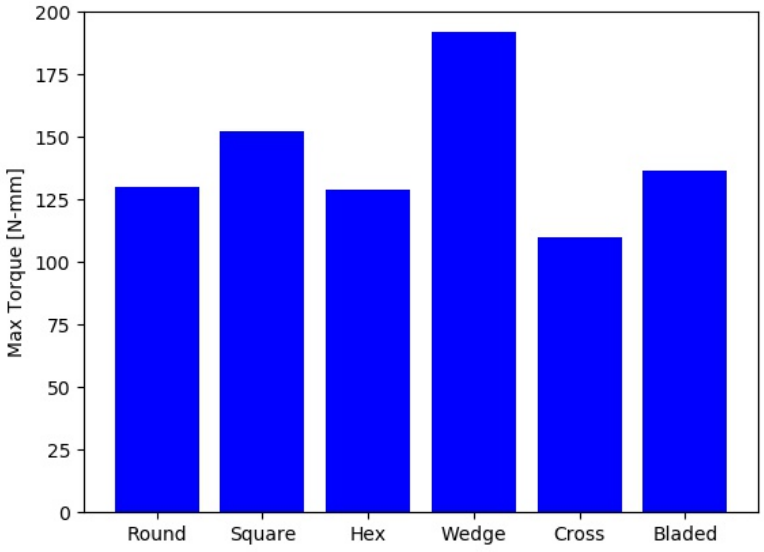

Figure 4.6: DEM Max Torque Results for Infill Alone

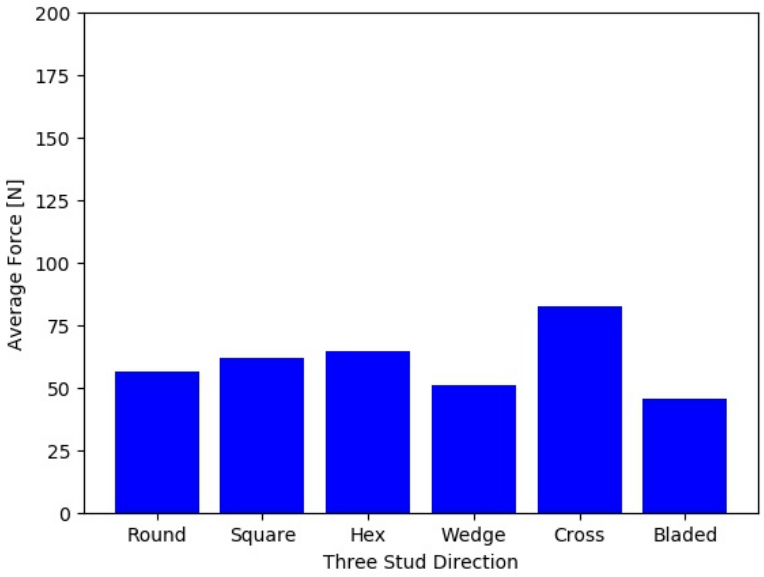

Figure 4.7: DEM Average Force Results for Infill Alone in the Three Stud Direction

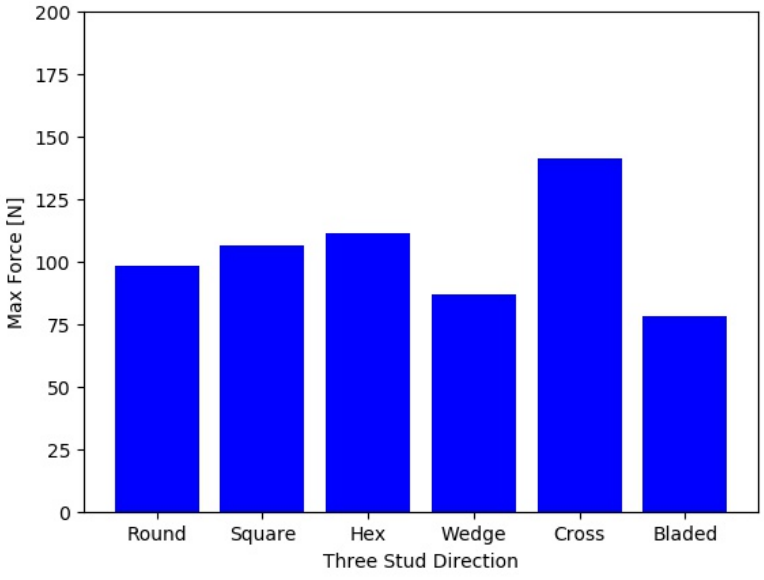

Figure 4.8: DEM Max Force Results for Infill Alone in the Three Stud Direction

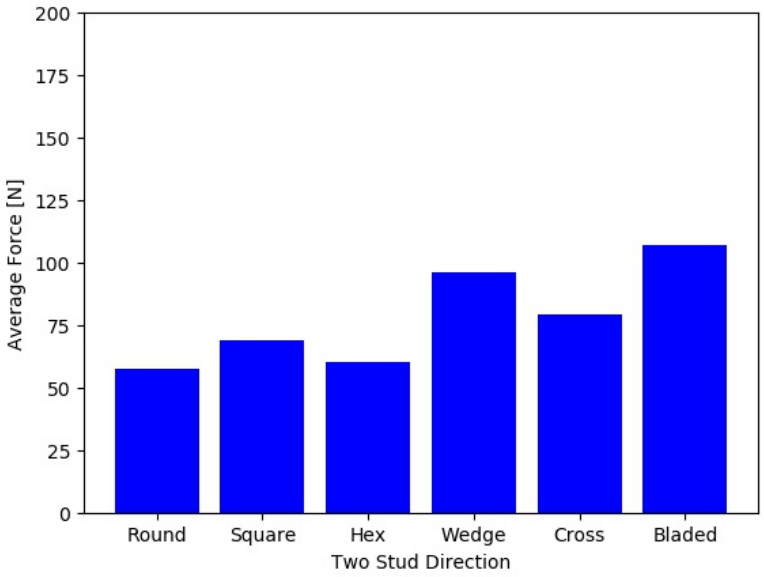

Figure 4.9: DEM Average Force Results for Infill Alone in the Two Stud Direction

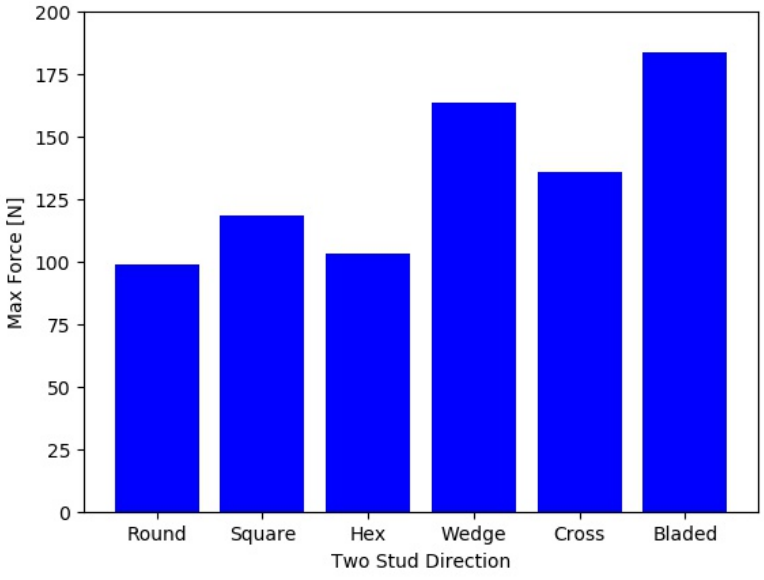

Figure 4.10: DEM Max Force Results for Infill Alone in the Two Stud Direction

4.3 Comparative Analysis

Comparative results for laboratory experiments versus DEM simulations for round studs can be seen in Figure 4.11 below. The average torque of the calibrated discrete element model was within 2% of the laboratory experiments for infill and grass+infill. The max torque percent difference for both infill and grass+infill was around 25%. This was higher than desired, but could have been reduced with more time to adjust input parameters.

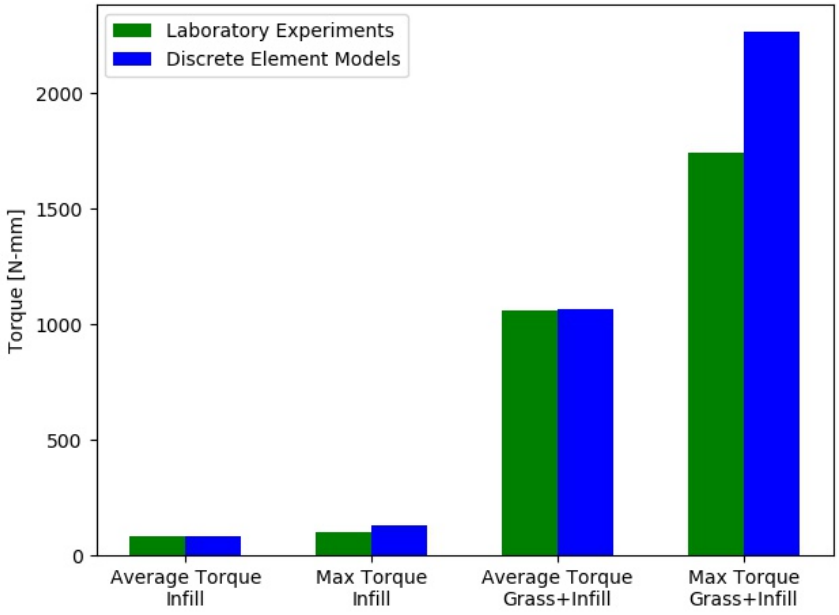

Figure 4.11: Laboratory Results Compare to DEM Results

The plots for the DEM simulation and the laboratory experiments for infill alone can be viewed in Figures 4.12 and 4.13 below. The plots for the DEM simulation and the laboratory experiments for grass+infill can be viewed in Figures 4.14 and 4.15 below. Table 4.1 below are the calculated slopes of the plots. The initial ascending slope is for when time is between zero and two seconds. The descending slope is for when time is between two and sixty seconds. The infill slope does not equal the average slope of laboratory experiments but does fall within their range. An additional step of simulating an equal number of DEM simulations as laboratory experiments should have been done and will be a part of future work. This would allow for an average comparison and for comparing plots more rigorously. The infill DEM simulation is more mountainous

within the path of torque versus time/degree but all peaks and valleys fall within the ranges set forth by laboratory experiments. However, over the course of the first six degrees, the results for the DEM simulation were less mountainous and more comparable to laboratory experiments. This can be seen in Figure 4.12. It should be noted, this is the main area of interest. The average juke or cut move in football does not require 60 degrees. This was done out of curiosity and, to a lesser extent, completeness. Additional DEM simulations could possess less mountainous plots thus looking more comparable to laboratory experiments. The slope of the DEM simulation for the grass+infill plot also falls within the range set forth from laboratory experiments. However, the path of the grass+infill DEM simulation is dissimilar to the paths of the laboratory experiments. This is presumably due to modeling the grass as rigid beams. A plot for grass+infill over the first six seconds/degrees can be seen in Figure 4.14.

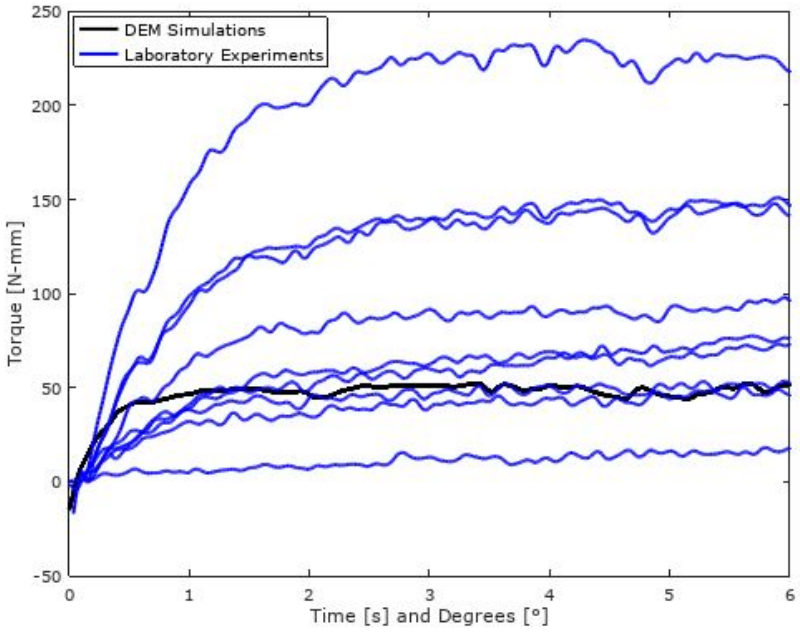

Figure 4.12: Comparing the First Six Degrees of DEM to Laboratory Experiments for Infill Alone

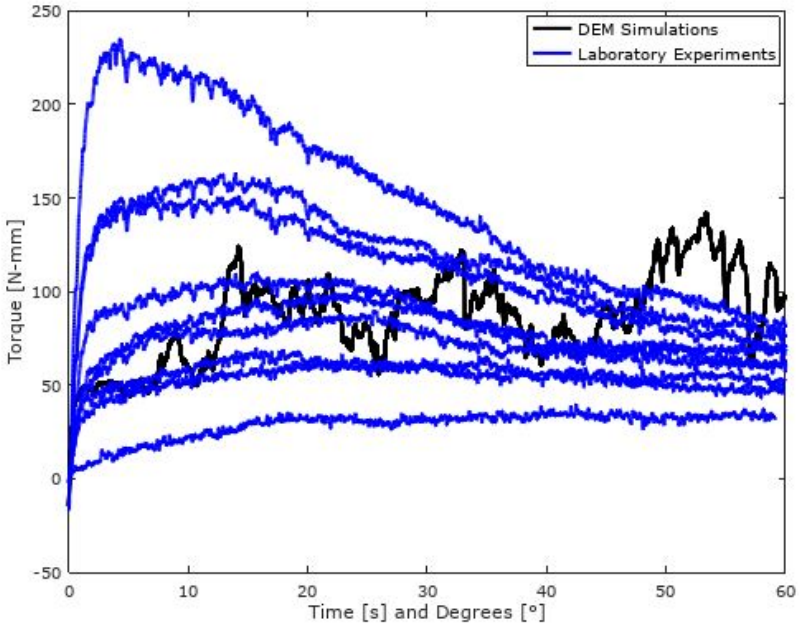

Figure 4.13: Comparing DEM Simulations to Laboratory Experiments in Infill Alone

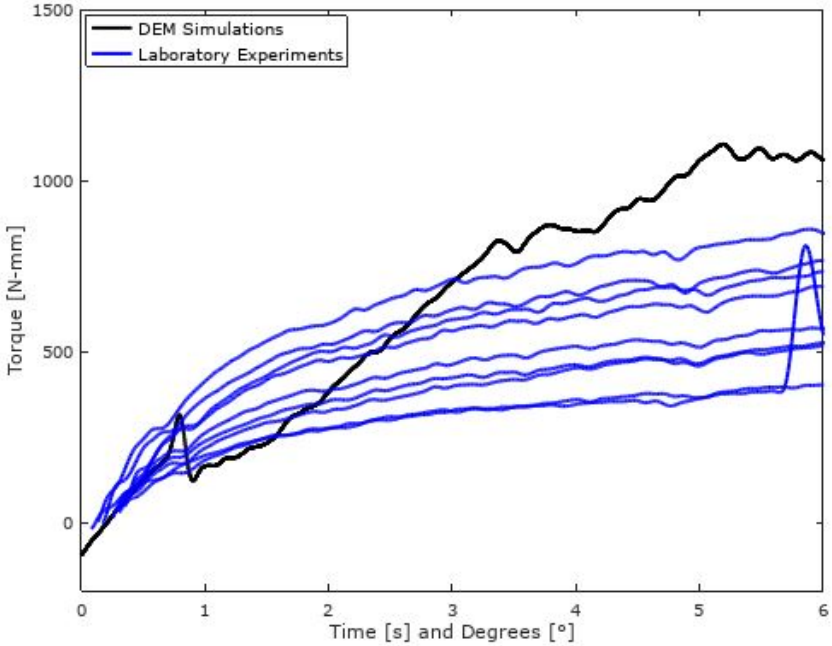

Figure 4.14: Comparing the First Six Degrees of DEM to Laboratory Experiments for Grass+Infill

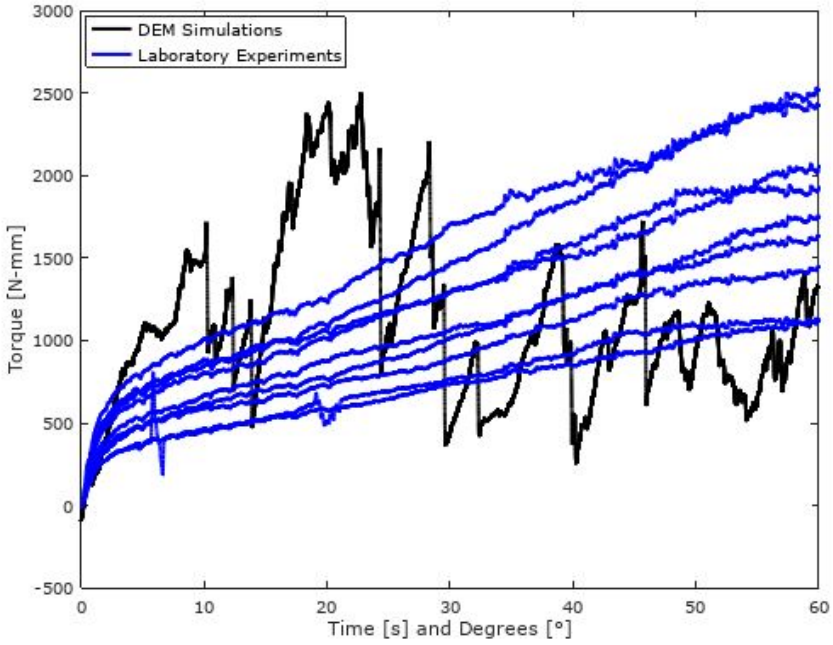

Figure 4.15: Comparing DEM Simulations to Laboratory Experiments in Grass+Infill

Table 4.1: Slopes for Infill and Grass+Infill

Results	The Initial Ascending Slope [N-mm·s^{-1}] Infill	The Initial Ascending Slope [N-mm·s^{-1}] Grass+Infill	Descending Slope [N-mm·s^{-1}] Grass+Infill
Experiment 1	33.1	211.1	23.1
Experiment 2	66.0	255.01	27.1
Experiment 3	25.6	204.9	23.6
Experiment 4	22.7	146.1	14.6
Experiment 5	41.5	311.4	33.5
Experiment 6	26.3	171.7	19.4
Experiment 7	3.2	153.2	14.6
Experiment 8	107.2	275.5	24.5
Experiment 9	21.9	195.9	22.1
Experiment 10	63.8	288.9	32.8
DEM Simulations	30.7	239.0	N/A

Figures 4.16 and 4.17 compare translational results for the 3 and 2-stud directions relative to each other. The high increase from the 3-stud direction relative to the 2-stud direction for wedge and bladed studs becomes intuitive when looking at their figures in Appendix B. The 2-stud direction is essentially a slanted wall going through the infill. The wedge and bladed studs were designed to have all three studs facing the same direction. This was to determine the influence each face plays on translational movement.

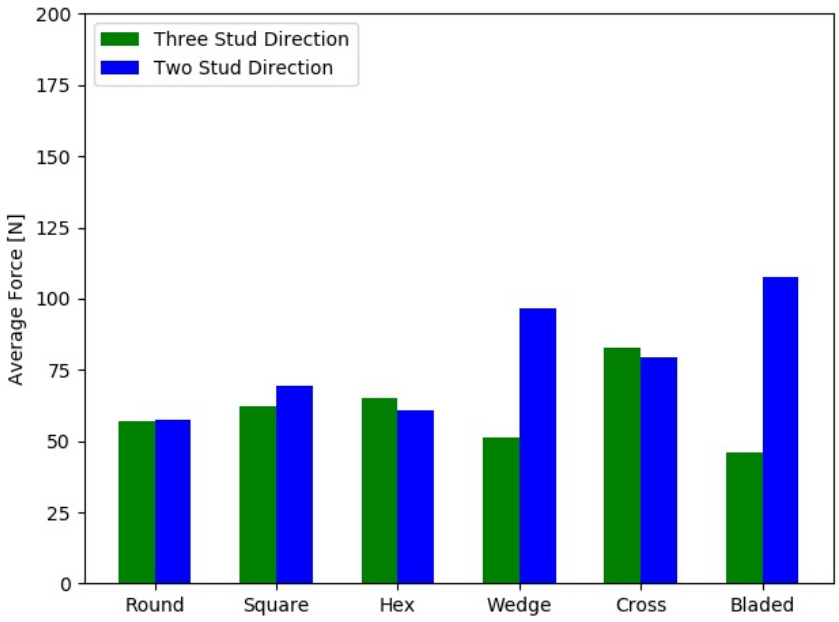

Figure 4.16: Comparing Average Force for the Two Directions of Translational Movement

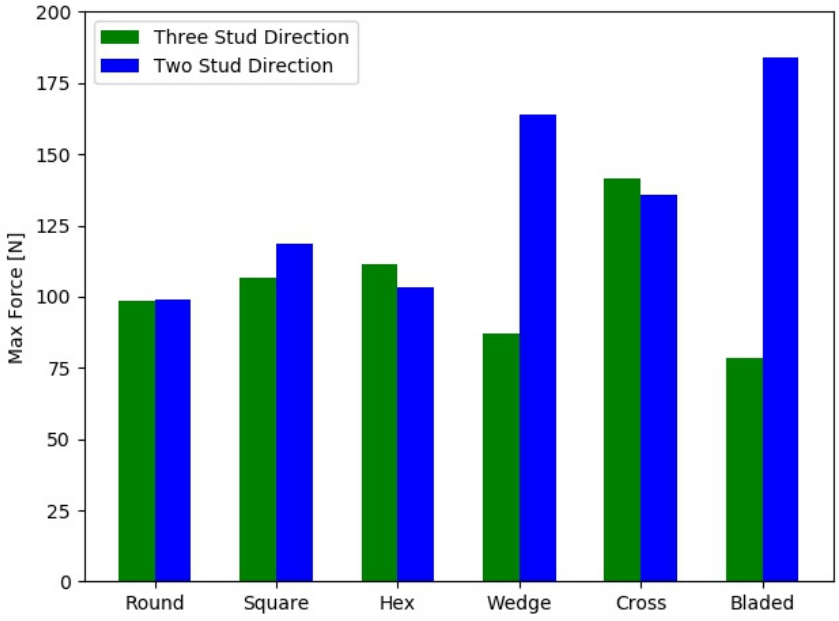

Figure 4.17: Comparing Max Force for the Two Directions of Translational Movement

CHAPTER 5

DISCUSSION

5.1 Experimental Analysis

The laboratory experiments involved three components. The first was the creation of the rigid fixtures. These fixtures had to be rigid to ensure no force would be absorbed by them due to elastic deformation. Thus, steel was selected for the fixtures and welded together at the joints. The second part was executing the experiments. These experiments had to be repeatable and done rigorously. These experiments were done numerous times to ensure valid data. The final step in the process was analyzing the data. The average and max torques were taken from each sample of data. The Butterworth low pass filter was applied to help reduce noise within the data. This data can be seen in histogram form in Section 4.1. The approximated standard of deviations can be seen in Table 5.1.

Table 5.1: Standard Deviations for Laboratory Experiments

Model	Average Torque Stud Depth 17 [mm]	Max Torque Stud Depth 17 [mm]	Average Torque Stud Depth 19.05 [mm]	Max Torque Stud Depth 19.05 [mm]
Infill	35	60	95	259
Grass+Infill	276	475	687	1068

5.2 Discrete Element Analysis

Stud geometry clearly plays a significant role on torque and force generation. Average torque for infill alone had a peak percent difference of roughly 22% in rotational movement while

max torque had a peak of roughly 38%. A portion of these increases will eventually be transferred to the athlete. Potentially leading to an increase in injuries such as broken bones or torn ligaments. It is important to consider other percent differences other than peak and examine expected and non-expected outcomes. Expected outcomes can help validate the model while non-expected outcomes need to be investigated, or a hypothesis needs to be formed to be investigated later.

In rotational movement one expected outcome was average and max torque for hex studs be relatively close to round studs. Another expected outcome was square being one of the highest in regards to torque generation. A non-expected outcome was the cross shaped studs coming in at roughly 15 [N-mm] less than the square studs for average torque. This could be due to the sphere size. Smaller spheres could catch better within the grooves of the cross shape studs. Thus, increasing the average torque and perhaps max torque for the cross shaped studs. Further experiments would need to be done to confirm this is not a sphere radius error.

A max percent difference of roughly 60% was found for bladed studs in both max and average force relative to round studs. Wedge studs had roughly 50% increase in percent difference relative to round studs. Bladed and wedge shape studs also had a substantial percent difference in force generated for the two stud direction relative to the three stud direction. Admittedly, this data is incomplete due to the non-symmetrical shape of these studs. In one direction these studs have what amounts to a flat slanted wall. While in the other direction they are either rounded or come to an edge. These studs should have been simulated two more times each allowing both faces to move in the two and three stud directions. Though, it would be expected for the wall side to create significantly more force. Certainly, data from the other simulations is desired, this is still viewed as a positive outcome. Percent difference does decrease quite a bit for more symmetrical studs.

The results for the grass+infill DEM simulations possesses less real world relevance than the infill alone due to an error in the code. The grass would rapidly expand with mechanical properties and degrees of freedom similar to artificial grass. This is why the grass was modeled as rigid beams. The code can be seen in Appendix J. This could have been due to limited user experience in Yade or a bug within the program. The author of this research already reported one bug trying to use the built in functions to model grass. The commands used to create the grass were part of the daily build. This means Yade had to be compiled from source and the commands were not part of the stable release.

5.3 Future Work

A model with a sphere radius of 1 [mm]was simulated for round studs and the results were negligible compared to a sphere radius of 4 [mm]. A few more models with a sphere radius of 1 [mm] should be simulated to validate that all stud designs have negligible results compared to their 4 [mm] sphere radius models. The next step is fixing the rigid grass issue either in Yade or switching to Chrono. Chrono appears to be a more robust open source DEM program. In addition to appearing more feature rich, in running a few example programs it seems to be significantly faster. This is presumably due to the coding being done in C++ over Python. This decrease in computational expense could make running 1 [mm] radius spheres more reasonable.

The long-term goal of this research is to optimize footwear to reduce injuries. This is done by creating a complete model of the foot to surface interactions. This research established the first two by validating a DEM model and analyzing stud design. The next steps include creating a finite element model of the whole cleat and the human foot. These models can then be coupled with the DEM model created through this research. This coupled model could then show how the forces and torques found from this research translates to the athletes body. Another opportunity for future work includes forking this model. Related models could investigate footwear and granular materials of other activities, e.g. sand on a beach or climbing on gravelly hills.

CHAPTER 6

CONCLUSION

This thesis was able to illustrate that DEM could be the next step into stud development, and illustrated that stud geometry appears to play a significant role in torque and force generation while interacting with the turf. Round and hex studs appear to cause the least torque in rotational movements. Furthermore, round studs appear to generate the least force regardless of orientation in translational movements, though depending on orientation bladed and wedge shape studs created less force. Therefore, if a studded shoe was created with only one stud design, round would presumably be the safest. Ultimately, a percentage of the forces and torques generated by the studs and footwear will transfer to the athlete, impacting performance and potential for injury. No ideal limit of load transfer has yet been established. Based on preliminary results, DEM appears to provide a path for dynamic stud and turf interaction modeling. This was a promising first step in modeling the complete ground to foot interaction. Future work would need to examine stud placement and mixing and matching stud designs.

This thesis also illustrated that open source software could provide a potential path for non-funded or low-funded researchers to accomplish high-end research without the additional cost that normally comes with proprietary software. The open source programs chosen were rich in features and almost fully capable of completing the intent of this thesis. Modeling artificial grass was the only potential limitation. This limitation could have potentially been solved with a more advanced user of Yade or without the time limitations that are inherent in thesis research.

REFERENCES

[1] Wilson Andrews, Bonnie Berkowitz, and Alberto Cuadra. Nfl injuries: Where does it hurt?, 2013.

[2] Zachary Binney. The truth behind rising injury rates, 2017.

[3] N. Smith, R. Dyson, and L. Janaway. *Ground reaction force measures when running in soccer boots and soccer training shoes on a natural turf surface*. Sports Engineering, 2004.

[4] Bruce Williams and Lowell Weil Jr. Football. In *Athletic Footwear and Orthoses in Sports Medicine*, pages 329–339. Springer International Publishing, 2017.

[5] Richard Kent, Jeff Crandall, Jason Forman, David Lessley, Anthony Lau, and Christopher Garson. Development and assessment of a device and method for studying the mechanical interactions between shoes and playing surfaces in situ at loads and rates generated by elite athletes. *Sports Biomechanics*, 2012.

[6] Peter Gustafson, Andrew Geeslin, and James Jastifer. An open source reverse engineering workflow: Geometry to optimization. In *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2016.

[7] John Amend, Anthony McNicoll, and Hod Lipson. Freeloader, 2011.

[8] Kamyar Hashemnia and Jan Spelt. Finite element continuum modeling of vibrationally-fluidized granular flows. *Elsevier*, 129:91–105, 2015.

[9] Vclav milauer and Bruno Chareyre. *DEM Formulation*, 2nd edition, 2015.

[10] Vclav milauer and et al. *Reference Manual*, 2nd edition, 2015.

[11] FieldTurf. Infill options for your needs.

[12] Scaling of discrete element model parameters for cohesionless and cohesive solid, 2015.

[13] Thomas HALES and et al. A formal proof of the kepler conjecture. *Forum of Mathematics*, 5, 2017.

[14] AZoM. Silicone rubber, 2001.

[15] Dennis Da Corte. Biomechanical analysis of the sidestep cutting maneuver in football players with opensim, 2014.

[16] R.F. Kirk, I.S.G. Noble, T. Mitchell, C. Rolf, S.J. Haake, and M.J. Carr. High-speed observations of football-bootsurface interactions of players in their natural environment. In *Sports Engineering*, page 129144, 2007.

APPENDIX A

Octave Code for Stud Sizing

```
clc;
clear all;
fprintf('All Units are in mm \n\n');
%% Cleat Height and Volume remains the same for all
%% Top = cleat that hits the ground first
%% Bottom = where cleats attaches to shoe.
height = 19.05; % [mm]
fprintf('Height for All: %5.2f \n\n',height);

%% Volume of Tapered Cylinder
big_r = 9.525; % [mm], bottom of stud
small_r = 4.76; % [mm], top of stud
vol_cylinder = 1/3*pi*(big_r^2+big_r*small_r+small_r^2)*height; % [mm^3]
fprintf('Measurements for Tapered Cylinder: \n');
fprintf('Radius of Small Cylinders: %5.2f \n',small_r);
fprintf('Radius of Big Cylinders: %5.2f \n\n',big_r);

%% Volume of all other cleats must equal to volume of Tapered Cylinder

%% Volume of tapered square cleat. (truncated pyramids)
% Note: A is big length (bottom), b is small length (top)
% b was set to small_r, to require similar force to enter the ground
% Vol_square = 1/3*(a^2+a*b+b^2)*height
b = 2*small_r; % [mm] 2, is to convert radius to diameter length

%% The Quadratic Formula was used to solve for a, hence two of them
a = (-b+sqrt(b^2+4*(3*vol_cylinder/height-b^2)))/2;
a1 = (-b-sqrt(b^2+4*(3*vol_cylinder/height-b^2)))/2;

%% Checking to make sure a is positive
if a <= 0;
a = a1;
endif;
%% Check to make sure volume is correct
Vol_square_check = 1/3*(a^2+a*b+b^2)*height;
if Vol_square_check = vol_cylinder;
fprintf('Volume for Tapered Square Cleats Match \n');
fprintf('Measurments for small (Top): \n');
fprintf('Small Length: %5.2f \n',b);

fprintf('Measurments for bottom (Big): \n');
fprintf('Big Length: %5.2f \n\n',a);
else
display('Volume for Square Cleats do NOT Match');
endif;

%% Volume of Tapered Wedge
% Volume  = (b*heigth)/6*(2*a+c);
% a = long length of wedge, b = short length, both bottom.
% a was left the same as the square cleat, trying to keep similar sizes
% c = length of top of stud. (Line parallel to a)
% c = was set to the diameter of the top of the cylinder
b = a;
clear a;
c = 2*small_r; % [mm] 2, is to convert radius to diameter length
a = (6*vol_cylinder/(height*b)-c)/2;

%% Check to make sure volume is correct
Vol_wedge_check = (b*height)/6*(2*a+c);
if Vol_wedge_check = vol_cylinder;
fprintf('Volume for Wedge Cleats Match \n');
fprintf('Short Length of Wedge: %5.2f \n',b);
fprintf('Long Length of Wedge: %5.2f \n',a);
fprintf('Length of Top of Stud (Line Parallel to Long Length): %5.2f \n\n',c);
else
display('Volume for Wedge Cleats do NOT Match');
endif;
```

```
%% Volume of hexagon frustum (tapered)
% Knowing a hexagon is 6 triangles, that formula is used IE 1/2*b*h_tri;
S1 = pi*small_r^2; % [mm^2], top area, equal to the top of the cylinder
%% Use  wolfram Alpha to solve for S2 from volume equation, two solutions
S2 =  1/2*(-sqrt(3)*sqrt(4*S1*vol_cylinder-height*S1^2)/sqrt(height)...
      + 6*vol_cylinder/height - S1); % [mm^2], bottom area, bigger area
S21 = 1/2*(sqrt(3)*sqrt(4*S1*vol_cylinder-height*S1^2)/sqrt(height)...
      + 6*vol_cylinder/height - S1);

vol_hex_check = height/3*(S1+S2+sqrt(S1*S2));
if vol_hex_check = vol_cylinder;
fprintf('Volume for Hexagon Cleats Match\n');
%% Used big_r, of cylinders to produce a similar size, area formula
% for a triangle, hence why 1/6 was taken.
b_small = sqrt(2*S1/(3*sqrt(3))); % [mm], length of hexagon side, for six sides
b_big = sqrt(2*S2/(3*sqrt(3))); % [mm], length of hexagon side (same for all six)
fprintf('Length of Side for Small Hexagon: %5.2f \n',b_small);
fprintf('Length of Side for Big Hexagon: %5.2f \n\n',b_big);
else;
vol_hex_check = height/3*(S1+S21+sqrt(S1*S21));
     if vol_hex_check = vol_cylinder;
     fprintf('Volume for Hexagon Cleats Match\n');
     S2 = S21;
     %% Used big_r, of cylinders to produce a similar size, area formula
     % for a triangle, hence why 1/6 was taken.
     b_small = sqrt(2*S1/(3*sqrt(3))); % [mm], length of hexagon side, for six sides
     b_big = sqrt(2*S2/(3*sqrt(3))); % [mm], length of hexagon side (same for all six)
     fprintf('Length of Side for Small Hexagon: %5.2f \n',b_small);
     fprintf('Length of Side for Big Hexagon: %5.2f \n\n',b_big);
     else;
     display('Volume for Hexagon Cleats do NOT Match');
     endif;
endif;

%% Volume for tapered blade stud
%% A "blade" is a rectangle with two half cylinders at the end.
%% Calculating "half" cylinders part:
% Note: Calculating as whole cylinder due to two halfs.
small_r_2 = small_r/2; % [mm], making these cylinder half the size
big_r_2 = big_r/2;    % [mm], making these cylinder half the size
vol_cyl = 1/3*pi*(big_r_2^2+big_r_2*small_r_2+small_r_2^2)*height; % [mm^3]

%%Calculating rectangle part, used:
% http://keisan.casio.com/exec/system/1297652433
% A,a are the long side and equal, because they do not taper
% B,b are the short sizes and do taper
b = 2*small_r_2; % [mm], 2 is to convert radius to diameter length
B = 2*big_r_2;  % [mm], 2 is to convert radius to diameter length

%% Find length of rec, IE, A and a.
v_rec = vol_cylinder - vol_cyl; % Volume remaining for the rectangle part
A = 2*v_rec/(height*(b+B)); % [mm]
a = A; % [mm]

vol_rec_check = height/6*(A*b+a*B+2*(a*b+A*B));
vol_blade_check = vol_rec_check + vol_cyl;
if vol_blade_check = vol_cylinder;
fprintf('Volume for Blade Cleats Match \n');
fprintf('Radius of Small (half) Cylinders: %5.2f \n',small_r_2);
fprintf('Radius of Big (half) Cylinders: %5.2f \n',big_r_2);
fprintf('Length of Small Side Top (Small): %5.2f \n',b);
fprintf('Length of Small Side Bottom (Big): %5.2f \n',B);
fprintf('Length of Long Side: %5.2f \n',A);
fprintf('\n');
else;
display('Volume for Blade Cleats do NOT Match  \n');
fprintf('\n');
endif;
```

```
%% Custom Designs!!!
%% Cross:
Len_sq_sds = 2*small_r;  % [mm] Length of square sides
vol_square_middle = Len_sq_sds*Len_sq_sds*height;
vol_left = vol_cylinder-vol_square_middle; %% Volume left over from square

%% Finding long lengh (Lenght going away from square) of triangle.
% Cross length is going to be small_r;
len_away = vol_left/(Len_sq_sds*height); % From vol = 4*(1/2*b*l*h)

fprintf('Cross Shape Stud \n');
fprintf('Length of Side of Square: %5.2f \n',Len_sq_sds);
fprintf('Length of Triangle Side Going Away: %5.2f \n',len_away);
fprintf('Length of Triangle Side Parallel Side to Square: %5.2f \n',Len_sq_sds/2);
fprintf('\n');
```

APPENDIX B

FreeCAD Drawing of the Studs and Bracket

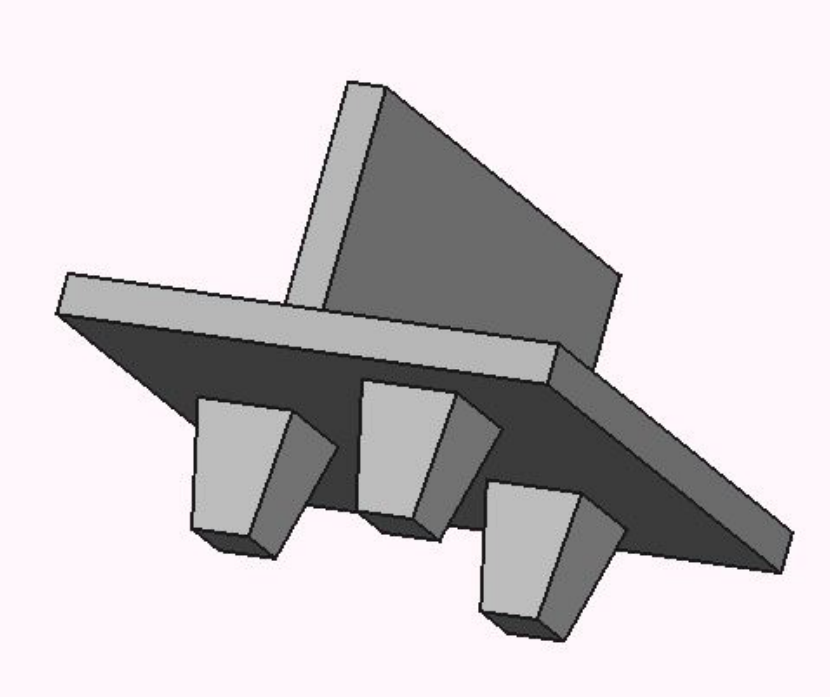

Figure B.1: FreeCAD Drawing of Square Studs and Bracket

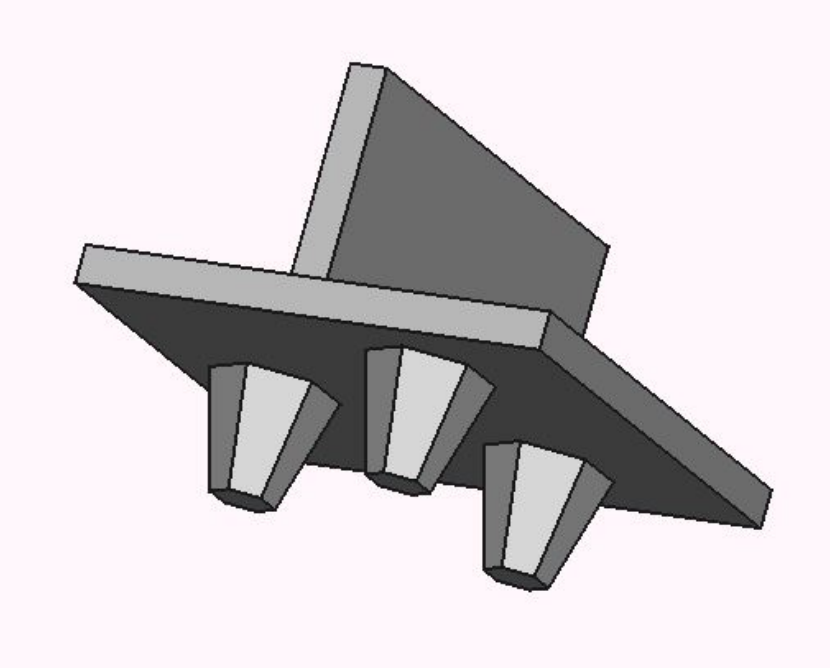

Figure B.2: FreeCAD Drawing of Hex Studs and Bracket

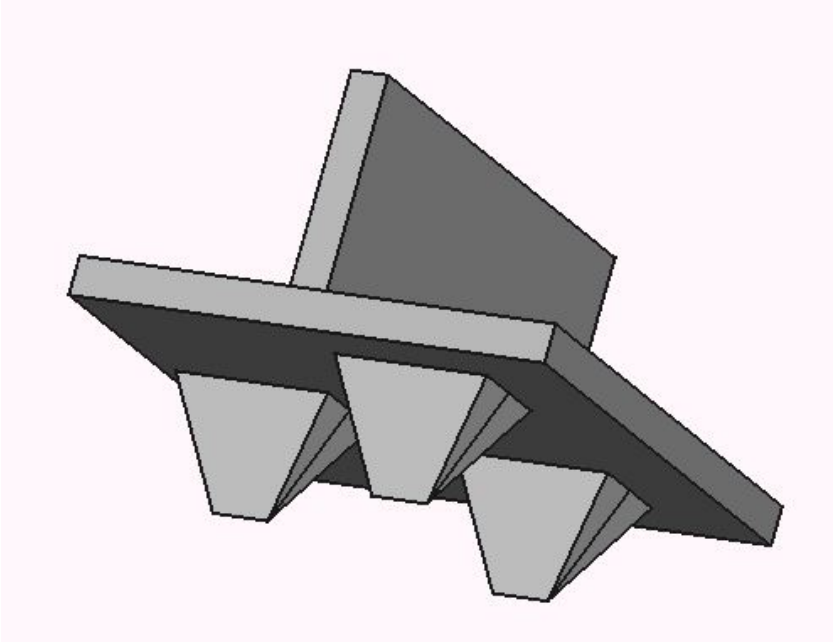

Figure B.3: FreeCAD Drawing of Wedge Studs and Bracket

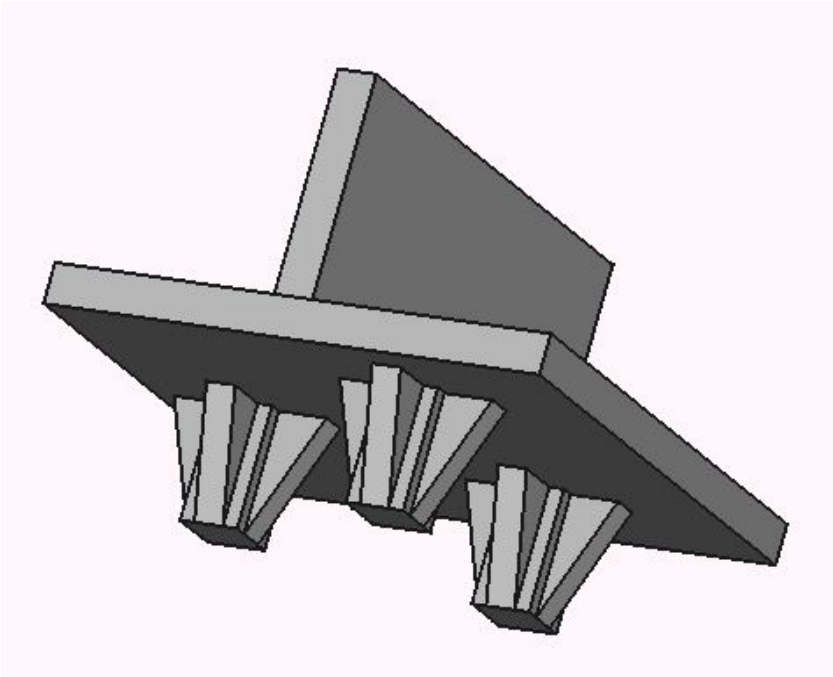

Figure B.4: FreeCAD Drawing of Cross Studs and Bracket

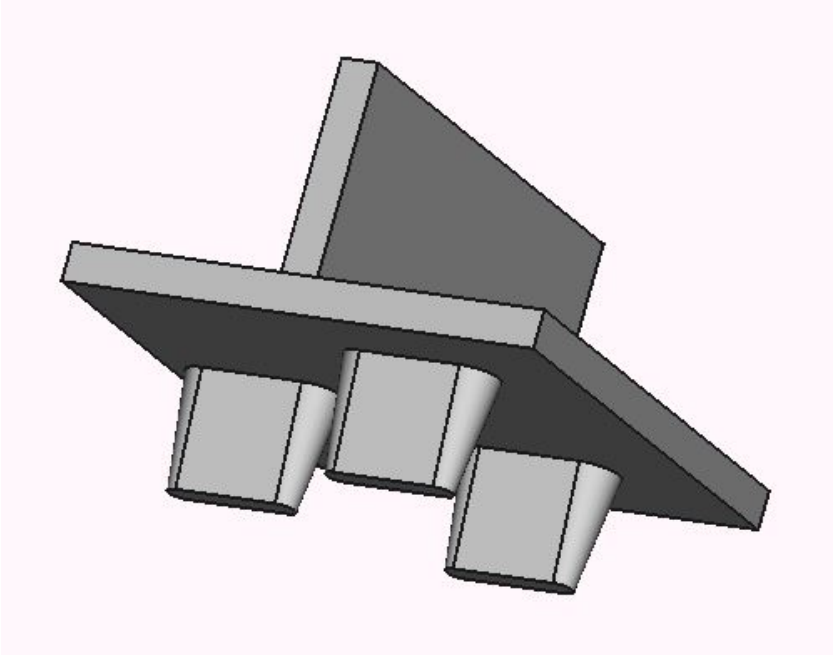

Figure B.5: FreeCAD Drawing of Bladed Studs and Bracket

APPENDIX C

Octave Code for Number of Spheres and Time Step

```
clc;
clear all;

height_of_packing = 25; % [mm]
r_ball = [1, 2, 3, 4]; % [mm]

length_inside_box = 112.5 + 112.5 - 6.35 - 6.35; % [mm]
area_box = length_inside_box*length_inside_box; % [mm^2]
volume_packing = area_box*height_of_packing; % [mm^3]
ball_volume = (4/3)*pi*r_ball.^3; % [mm^3]

% Randomly packed spheres pack at abour 65%
num_of_balls = (volume_packing./ball_volume)*.65


%% Time steps
time_steps = [8.96482307668e-05; # Example Time Steps
              0.000114033567397; #
              0.000540908704032; #
              0.000721211605376];

%% Trans engine
length_of_cleat = 19.05; % [mm]
vel_trans_eng = 0.222*2; % [mm/s], note in Yade it's .000222*2 [m/s]
num_sec_trans = length_of_cleat/vel_trans_eng; % [s]

num_of_steps_trans = num_sec_trans./time_steps

%% Rotation engine
deg_to_rad_length = (60*pi/180); % [rad]
an_vel_of_cleats = 0.00872665*2; % [rad/s]
num_sec_roto = deg_to_rad_length/an_vel_of_cleats; % [s]

num_of_steps_roto = num_sec_roto./time_steps;
total_steps = num_of_steps_trans+num_of_steps_roto
```

APPENDIX D

Octave Code for Post Processing Infill Laboratory Experiments

```
clc;
close all;
clear all;
pkg load signal;

%% Entering data:
%% Data
num_files = 10;  %% 10 being number length of files
for ii = 1:num_files;
mts{1,ii} = dlmread(strcat('25_mm__17_',num2str(ii),'/specimen.dat'),'     ',6,0); % Input
    data for 25 mm at 17 mm depth
mts{2,ii} = dlmread(strcat('25_mm_',num2str(ii),'/specimen.dat'),'    ',6,0); % Input data
    for 25 mm at 19.05 mm depth
mts{3,ii} = dlmread(strcat('50_mm_17_',num2str(ii),'/specimen.dat'),' ',6,0); % Input data
    for 50 mm at 17 mm depth
mts{4,ii} = dlmread(strcat('50_mm_',num2str(ii),'/specimen.dat'),'    ',6,0); % Input data
    for 50 mm at 19.05 mm depth
data_color{ii} = 'b'; %% Color input for plot
endfor;
data_name = {'25 mm 17 Depth','25 mm 19 Depth','50 mm 17 Depth','50 mm 19 Depth'};

for jj = 1:size(mts,1);
for kk = 1:num_files;
data{1,kk} = mts{jj,kk};
endfor;

for ii = 1:num_files;
rm_zeros{ii} = find(data{ii}(:,1)==0);
data{ii}(rm_zeros{ii},:) = [];
idx1{ii} = find(data{ii}(:,2)>=60, 1, 'first');
endfor;

idx = min(cell2mat(idx1));
%% Finding the small value so they are all the same length
for ii = 1:num_files;
Time{ii} = data{ii}(1:idx,1); % [s] Extract column 1
Angle{ii} = data{ii}(1:idx,2); % [degree] Extract column 2
torsion{ii} = data{ii}(1:idx,3); % [N-mm] Extract column 3
%% Shifting
[P] = polyfit(Time{ii}(1:100), torsion{ii}(1:100),1);
%% Shift time to align initial slopes
%% Proof of Time Shift  %% close all
%% Proof of Time Shift  %% plot(Time{ii}(1:100)+P(2)/P(1), [torsion{ii}(1:100) polyval(P,
    Time{ii}(1:100))])
Time{ii} += P(2)/P(1);
endfor;

%% Applying filter, butterworth 3 order, 15 Hz
cutoff = 20; % [Hz]
for ii = 1:num_files;
  puts ("PG mods\n");
  sf = 1/diff(Time{ii}(1:2));
  nf = sf/2;
  [b{ii},a{ii}] = butter(3,cutoff/nf/pi);
  %% PG use a zero lag filter... filter, flip, then filter again
  ## torsion{ii} = filter(b{ii},a{ii},torsion{ii}-torsion{ii}(1))+torsion{ii}(1);
  ## torsion{ii} = flipud(torsion{ii});
  ## torsion{ii} = filter(b{ii},a{ii},torsion{ii}-torsion{ii}(1))+torsion{ii}(1);
  ## torsion{ii} = flipud(torsion{ii});
  [fDATA]=filterbutter(torsion{ii},Time{ii},cutoff);
  torsion{ii} = fDATA;
%% Finding Max
torsion_max{ii} = max(abs(torsion{ii}));
torsion_idx{ii} = find(torsion{ii} == torsion_max{ii});
%% Finding Averages
#tor_r4_avg{ii} = sum(tor_r4{ii})/size(tor_r4{ii},1);
endfor;
```

```
%% Calculating Averages
all_sum_time = 0;
all_sum_tor = 0;
for ii = 1:num_files;
all_sum_time = all_sum_time + Time{ii};
all_sum_tor = all_sum_tor + torsion{ii};
endfor;

time_avg = all_sum_time./size(Time,2);
torsion_avg = all_sum_tor./size(torsion,2);
avg_avg_torsion = sum(torsion_avg)/length(torsion_avg);
max_avg_torsion = max(torsion_avg);

AA = 1:4;
subplot_num = repmat(AA,1,size(data,2));
fig_num = 1 + floor((jj-1)/4);

figure(fig_num);
ax(jj) = subplot(2,2,subplot_num(jj));
xlim([0 60]);
title({'Average:';strcat('Average Torque:',num2str(avg_avg_torsion));strcat('Max Torque:',
    num2str(max_avg_torsion))});
xlabel('Time [s]');
ylabel('Torsion [N-mm]');
title({data_name{jj};strcat('Average Torque:',num2str(avg_avg_torsion));strcat('Max (Avg)
    Torque:',num2str(max_avg_torsion))});
hold on;

for ii = 1:size(data,2);
ax_plot(ii) = plot(Time{ii},torsion{ii},data_color{ii},'LineWidth',2);
endfor;
ax_avg = plot(time_avg,torsion_avg,'k','LineWidth',2);
hold off;

%% Storing the average and max for the 17 depth to be used later (Below)
if jj == 1 || jj == 3;
avg_avg_torsion_17(ceil(jj-jj/3)) = avg_avg_torsion;
max_avg_torsion_17(ceil(jj-jj/3)) = max_avg_torsion;
endif;
endfor;

%% Finding Density for simulations
%% Average
den_first = 1522e1;   %% First density for the found average
avg_first = 201.255;  %% Average found with that Density
x_avg_first   = avg_avg_torsion_17(1)*den_first/avg_first;

den_second = 1522;     %% Second density for the found average
avg_second = 16.251;    %% Average found with that Density
x_avg_second   = avg_avg_torsion_17(1)*den_second/avg_second;
den_due_avg = (x_avg_first+x_avg_second)/2;

%% Max
max_first  = 401.39;  %% Max found with first Density
x_max_first = max_avg_torsion_17(1)*den_first/max_first;

max_second = 136.223;  %% Max found with second Density
x_max_second = max_avg_torsion_17(1)*den_second/max_second;
den_due_max = (x_max_first+x_max_second)/2;
Density_to_use = (den_due_avg+den_due_max)/2
```

APPENDIX E

Octave Code for Post Processing Grass+Infill Laboratory Experiments

```
clc;
close all;
clear all;
pkg load signal;  %% Need for Butterworth Signal
%% Entering data:
%% Data
for ii = 1:10;  %% 10 being number length of files
num2str(ii)
data{ii} = dlmread(strcat('50_mm_grass_17_',num2str(ii),'/specimen.dat'),' ',6,0); % Input
    data
endfor;

%% Color input for plot
data_color{1} = 'r'; %% data_color{1} is not used.
data_color{2} = 'm';
data_color{3} = 'c';
data_color{4} = 'g';
data_color{5} = 'r';
data_color{6} = 'r';
data_color{7} = 'r';
data_color{8} = 'r';
data_color{9} = 'r';
data_color{10} = 'r';


for ii = 1:size(data,2);
rm_zeros{ii} = find(data{ii}(:,1)==0);
data{ii}(rm_zeros{ii},:) = [];
idx{ii} = find(data{ii}(:,2)>=60, 1, 'first');
endfor;

%% Finding the small value so they are all the same length
idx = min([idx{:}]);
%% Test 1
for ii = 1:size(data,2);
Time{ii} = data{ii}(1:idx,1); % [s] Extract column 1
Angle{ii} = data{ii}(1:idx,2); % [degree] Extract column 2
torsion{ii} = data{ii}(1:idx,3); % [N-mm] Extract column 3

[P] = polyfit(Time{ii}(1:100), torsion{ii}(1:100),1);
%% Shift time to align initial slopes
%% Proof of Time Shift  %% close all
%% Proof of Time Shift  %% plot(Time{ii}(1:100)+P(2)/P(1), [torsion{ii}(1:100) polyval(P,
    Time{ii}(1:100))])
Time{ii} += P(2)/P(1);
endfor;

#%% Applying filter, butterworth 3 order, 15 Hz
cutoff = 20; % [Hz]
for ii = 1:size(data,2);
  puts ("PG mods\n");
  sf = 1/diff(Time{ii}(1:2));
  nf = sf/2;
  [b{ii},a{ii}] = butter(3,cutoff/nf/pi);
  %% PG use a zero lag filter... filter, flip, then filter again
  ## torsion{ii} = filter(b{ii},a{ii},torsion{ii}-torsion{ii}(1))+torsion{ii}(1);
  ## torsion{ii} = flipud(torsion{ii});
  ## torsion{ii} = filter(b{ii},a{ii},torsion{ii}-torsion{ii}(1))+torsion{ii}(1);
  ## torsion{ii} = flipud(torsion{ii});
  [fDATA]=filterbutter(torsion{ii},Time{ii},cutoff);
  torsion{ii} = fDATA;
torsion{ii} = filter(b{ii},a{ii},torsion{ii});
%% Finding Max
torsion_max{ii} = max(abs(torsion{ii}));
torsion_idx{ii} = find(torsion{ii} == torsion_max{ii});
%% Finding Averages
#tor_r4_avg{ii} = sum(tor_r4{ii})/size(tor_r4{ii},1);
endfor;
```

```
%% Calculating Averages
all_sum_time = 0;
all_sum_tor = 0;
for ii = 1:size(data,2);
all_sum_time = all_sum_time + Time{ii};
all_sum_tor = all_sum_tor + torsion{ii};
endfor;
time_avg = all_sum_time./size(Time,2);
torsion_avg = all_sum_tor./size(torsion,2);
#display('Average');
avg_avg_torsion = sum(torsion_avg)/length(torsion_avg);
#display('max');
max_avg_torsion = max(torsion_avg);

figure(1);
plot(Time{1},torsion{1});
j_legend{1} = strcat('Test-',num2str(1));
title('Time Vs Torsion and Angle of Rotation');
xlabel('Time [s]');
ylabel('Torsion [N-mm]');
hold on;

for ii = 2:size(data,2);
ax_plot(ii-1) = plot(Time{ii},torsion{ii},data_color{ii},'LineWidth',2);
j_legend{ii} = strcat('Test-',num2str(ii));
endfor;
ax_avg = plot(time_avg,torsion_avg,'k','LineWidth',2);
j_legend{ii+1} = strcat('Average Torque:',num2str(avg_avg_torsion));
legend(j_legend{:},'location', 'northwest');
xlim([0 60]);

%% Finding Density for simulations
%% Average
den_first = 1522e1;   %% First density for the found average
avg_first = 997.287;  %% Average found with that Density
x_avg_first   = avg_avg_torsion*den_first/avg_first;

den_second = 1522e2;     %% Second density for the found average
avg_second = 1272.464;    %% Average found with that Density
x_avg_second   = avg_avg_torsion*den_second/avg_second;
den_due_avg = (x_avg_first+x_avg_second)/2;

%% Max
max_first  = 1851.361;  %% Max found with first Density
x_max_first = max_avg_torsion*den_first/max_first;

max_second = 2567.458;  %% Max found with second Density
x_max_second = max_avg_torsion*den_second/max_second;
den_due_max = (x_max_first+x_max_second)/2;
Density_to_use = (den_due_avg+den_due_max)/2
```

APPENDIX F

Octave Code for Post Processing DEM Simulations in Rotation Movement

```
clc;
clear all;
close all;
pkg load signal;

% Input data roto engine
data{1} = dlmread('r4_found_den_rot.dat',' ',0,0);
data{2} = dlmread('r4_found_den_rot_sq.dat',' ',0,0);
data{3} = dlmread('r4_found_den_rot_wedge.dat',' ',0,0);
data{4} = dlmread('r4_found_den_rot_hex.dat',' ',0,0);
data{5} = dlmread('r4_found_den_rot_blades.dat',' ',0,0);
data{6} = dlmread('r4_found_den_rot_cross.dat',' ',0,0);
data{7} = dlmread('r4_found_den_rot_ellip.dat',' ',0,0);

%% Data Titles
data_title{1} = 'Torque on Round Studs, Radius 4 [mm]';
data_title{2} = 'Torque on Square Studs';
data_title{3} = 'Torque on Wedge Studs';
data_title{4} = 'Torque on Hex Studs';
data_title{5} = 'Torque on Bladed Studs';
data_title{6} = 'Torque on Cross Studs';
data_title{7} = 'Torque on Ellip Studs';

%% Time step for each
time_steps(1:size(data,2)) = 0.000256359572832;

#%% Torque
#% Total numbers of steps ran
for ii = 1:size(data,2);
total_roto_steps_r4{ii} = (data{ii}(end,1)-(data{ii}(1,1)-1));
#% Total real time
total_time_r4{ii} = total_roto_steps_r4{ii}*time_steps(ii);
#% Time of real seconds, in steps
time_roto_r4{ii} = time_steps(ii):time_steps(ii):total_time_r4{ii}; % [s]
#%% Torque on studs
tor_r4{ii} = data{ii}(:,2)*1000; % [N-mm], the 1000 is to convert to mm
endfor;

#%% Applying filter, butterworth 3 order, 15 Hz
cutoff = 20; % [Hz]
for ii = 1:size(data,2);
sf = 1/time_steps(1);
  nf = sf/2;
  [b{ii},a{ii}] = butter(3,cutoff/nf/pi);
  [fDATA]=filterbutter(tor_r4{ii},time_roto_r4{ii},cutoff);
  tor_r4{ii} = fDATA;
%% Finding Max
tor_r4_max{ii} = max(abs(tor_r4{ii}));
tor_r4_idx{ii} = find(tor_r4{ii} == tor_r4_max{ii});
%% Finding Averages
tor_r4_avg{ii} = sum(tor_r4{ii})/size(tor_r4{ii},1);
endfor;

#%% Finding max values:
str = strcat('\leftarrow');

%% Finding Average
avg_round = sum(tor_r4{1})/length(tor_r4{1})

%% Roto plots
AA = 1:4;
subplot_num = repmat(AA,1,size(data,2));
for kk = 1:size(data,2);
fig_num = 1 + floor((kk-1)/4);
figure(fig_num);
ax(kk) = subplot(2,2,subplot_num(kk));
plot(time_roto_r4{kk},tor_r4{kk});
ylabel('Torque [N-mm]');
```

```
xlabel('Time [s]');
text(time_roto_r4{kk}(tor_r4_idx{kk}), tor_r4_max{kk}, str);
hleg = legend(strcat('Max Value: ', num2str(tor_r4_max{kk})),'Location','southeast');
set(hleg, 'title', strcat('Avg Value: ', num2str(tor_r4_avg{kk})));
title(data_title{kk});
hold on;
endfor;
```

APPENDIX G

Octave Code for Post Processing DEM Simulations in Translational Movement

```
clc;
clear all;
close all;
pkg load signal;

% Input data Trans engine
%% X direction
data{1} = dlmread('r4_found_den_push_x_rot.dat',' ',0,0);
data{2} = dlmread('r4_found_den_push_x_rot_sq.dat',' ',0,0);
data{3} = dlmread('r4_found_den_push_x_rot_wedge.dat',' ',0,0);
data{4} = dlmread('r4_found_den_push_x_rot_hex.dat',' ',0,0);
data{5} = dlmread('r4_found_den_push_x_rot_blades.dat',' ',0,0);
data{6} = dlmread('r4_found_den_push_x_rot_cross.dat',' ',0,0);

%% Z directions
data{7} = dlmread('r4_found_den_push_z_rot.dat',' ',0,0);
data{8} = dlmread('r4_found_den_push_z_rot_sq.dat',' ',0,0);
data{9} = dlmread('r4_found_den_push_z_rot_wedge.dat',' ',0,0);
data{10} = dlmread('r4_found_den_push_z_rot_hex.dat',' ',0,0);
data{11} = dlmread('r4_found_den_push_z_rot_blades.dat',' ',0,0);
data{12} = dlmread('r4_found_den_push_z_rot_cross.dat',' ',0,0);

%% Data Titles
data_title{1} = 'Force on Round Studs, Radius 4 [mm]';
data_title{2} = 'Force on Square Studs';
data_title{3} = 'Force on Wedge Studs';
data_title{4} = 'Force on Hex Studs';
data_title{5} = 'Force on Bladed Studs';
data_title{6} = 'Force on Cross Studs';
%% Z Directions
data_title{7} = 'Force on Round Studs, Radius 4 [mm]';
data_title{8} = 'Force on Square Studs';
data_title{9} = 'Force on Wedge Studs';
data_title{10} = 'Force on Hex Studs';
data_title{11} = 'Force on Bladed Studs';
data_title{12} = 'Force on Cross Studs';

%% Time step for each
time_steps(1:size(data,2)) = 0.000407535525016;

#%% Force
#% Total numbers of steps ran
for ii = 1:size(data,2);
total_trans_steps_r4{ii} = (data{ii}(end,1)-(data{ii}(1,1)-1));
#% Total real time
total_time_r4{ii} = total_trans_steps_r4{ii}*time_steps(ii);
#% Time of real seconds, in steps
time_trans_r4{ii} = time_steps(ii):time_steps(ii):total_time_r4{ii}; % [s]
#%% Force on studs
trans_r4{ii} = data{ii}(:,5); % [N-mm], the 1000 is to convert to mm
endfor;

#%% Applying filter, butterworth 3 order, 15 Hz
cutoff = 20; % [Hz]
for ii = 1:size(data,2);
sf = 1/time_steps(1);
  nf = sf/2;
  [b{ii},a{ii}] = butter(3,cutoff/nf/pi);
  [fDATA]=filterbutter(trans_r4{ii},time_trans_r4{ii},cutoff);
  trans_r4{ii} = fDATA;
%% Finding Max
trans_r4_max{ii} = max(abs(trans_r4{ii}));
trans_r4_idx{ii} = find(trans_r4{ii} == trans_r4_max{ii});
%% Finding Averages
first_neg = find(trans_r4{ii}(trans_r4_idx{ii}:end)<0,1,'first');
trans_r4_sum_avg = sum(trans_r4{ii}(1:first_neg));
trans_r4_size_avg = size(trans_r4{ii}(1:first_neg),1);
trans_r4_avg{ii} = trans_r4_sum_avg./trans_r4_size_avg;
```

```
#trans_r4_avg{ii} = sum(trans_r4{ii})/size(trans_r4{ii},1);
endfor;

#%% Finding max values:
str = strcat('\leftarrow');

%% Trans plots
AA = 1:4;
subplot_num = repmat(AA,1,size(data,2));
for kk = 1:size(data,2);
fig_num = 1 + floor((kk-1)/4);
figure(fig_num);
ax(kk) = subplot(2,2,subplot_num(kk));
plot(time_trans_r4{kk},trans_r4{kk});
ylabel('Force [N]');
xlabel('Time [s]');
text(time_trans_r4{kk}(trans_r4_idx{kk}), trans_r4_max{kk}, str);
hleg = legend(strcat('Max Value: ', num2str(trans_r4_max{kk})),'Location','southeast');
set(hleg, 'title', strcat('Avg Value: ', num2str(trans_r4_avg{kk})));
title(data_title{kk});
hold on;
endfor;
```

APPENDIX H

Python Code for Rotational Infill Simulations

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
sys.path.append('/home/justin/Desktop/real/yadeimport') #symbolic link to yade
from yadeimport import *

from yade import utils
import random
from yade import ymport
import math
import sys,time

class YadePythonSimulation():
     file_name = os.path.basename(__file__)

     # Spheres information
     sphereRadius = .004 # [m]
     nu = .48
     #G = 300000 # [Pa]
     den_rub = 9615.2 # [kg/m^3]
     yng_rub = 3000000 # [Pa]
     fric_rub = radians(38)  # [degrees]
     O.materials.append(FrictMat(frictionAngle=fric_rub,young=yng_rub,density=den_rub,
         poisson=nu,label='rubber'))

     ### Steps
     spheres_fall = 8000
     global All_steps

     pred = pack.inAlignedBox((-.10615,0,-.10615),(0.10615,.025,.10615))
     sp = pack.randomDensePack(pred, spheresInCell=2732, radius=sphereRadius, memoizeDb='/
         home/justin/Desktop/real/tmp/ball_packing65_r4_h25.sqlite', returnSpherePack=True)
     sp.toSimulation(color=(255,0,0),wire=False,material='rubber')


     ## Steel box informaion
     steel = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) # Pa, kg/m^3
     metal_base45 = O.bodies.append(ymport.gmsh("geo/flipped/based_70.mesh",scale=.001,shift
         =Vector3(0,-.00635,0), material=steel,color=(0,0,1)))

     ## Cleat information used steel properties to make rigid
     studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) #
         used steel properties to make rigid
     studs1 = O.bodies.append(ymport.gmsh("geo/flipped/1mm_heightchecker.mesh",scale=.00001,
         shift=Vector3(0.107,.027,0), material=studs_plastic, color=(0,0,1))) # to by pass "
         no studs defined" error... It's a clever trick ;)

     ## Time step set to 20% of Rayleigh Wave
     O.dt=.2*utils.RayleighWaveTimeStep()
     print O.dt

     ## Appending Time Step to the output
     str_time_step = "# Time Step = "+ str(O.dt)
     if str_time_step not in open(file_name).read():
          with open(file_name, "a") as myfile:
               myfile.write(str_time_step)

     ## Trans steps
     length_of_cleat = 19.05+.01; # [mm]
     vel_trans_eng = 0.222*2; # [mm/s]
     num_sec_trans = length_of_cleat/vel_trans_eng; # [s]
     Trans = int(round(num_sec_trans/O.dt)) + spheres_fall

     ## Rotation steps
     deg_to_rad_length = (60*pi/180); # [rad]
     an_vel_of_cleats = 0.00872665*2; # [rad/s]
     num_sec_roto = deg_to_rad_length/an_vel_of_cleats; # [s]
```

```
	num_of_steps_roto = num_sec_roto/O.dt;
	All_steps = int(Trans+num_of_steps_roto+spheres_fall)

	## Engines
	O.engines=[
		###Reset all forces stored in Scene::forces (O.forces in python). Typically, this
			is the first engine to be run at every step.	In addition, reset those
			energies that should be reset, if energy tracing is enabled.
		## Resets forces and moments that act on bodies
		ForceResetter(),

		## Using bounding boxes find possible body collisions.
		InsertionSortCollider([
			Bo1_Sphere_Aabb(),
			Bo1_Facet_Aabb(),
		]),
		InteractionLoop(
			[Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
			[Ip2_FrictMat_FrictMat_FrictPhys()],
			[Law2_ScGeom_FrictPhys_CundallStrack()],
		),
		NewtonIntegrator(damping=.2,gravity=[0,-9.81,0],label='newtonInt'),
		# Studs are 19.05 mm tall. Therefore, engine must translate 19.05 mm down.
		TranslationEngine(translationAxis=[0,1,0],velocity=-0.000222*2,ids=studs1,dead=
			True,label='transeng'), # [M/s]
		RotationEngine(ids=studs1,rotationAxis=(0,1,0), angularVelocity=0.00872665*2,
			rotateAroundZero=True,dead=True,label='rotengine'), # [Rad/s]
		PyRunner(command='turnonstuds()',iterPeriod=spheres_fall, nDo = 11, label='
			switchstuds', dead = False),
		PyRunner(command='switchTranslationEngine()',iterPeriod=Trans, nDo = 6, label='
			switchEng', dead = False),
		# Takes VTK data
#		VTKRecorder(fileName='vtk/r4_',recorders=['all'],iterPeriod=1000),
		## Record the forces being applied to the studs
		ForceRecorder(ids=studs1,file=file_name + '_trans.dat',iterPeriod=1, dead=True,
			label='trans_force'),
		TorqueRecorder(ids=studs1,file=file_name + '_rot.dat',iterPeriod=1, dead=True,
			rotationAxis=(0,1,0), label='rotation_force'),
	]

	def run(self):
		iterToRun=All_steps
		O.run(iterToRun, True)

def turnonstuds():
	print "Turning on studs"
	studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) #
		used steel properties for inputs but is assumed rigid
	studs = O.bodies.append(ymport.gmsh("geo/flipped/studs.mesh",scale=.001,shift=Vector3
		(0,(.022+.01905),0), material=studs_plastic,color=(0,0,1)))
	studs1 = None
	transeng.ids = studs
	rotengine.ids = studs
	trans_force.ids = studs
	rotation_force.ids = studs
	transeng.dead = False
	trans_force.dead = False
	switchstuds.dead = True

def switchTranslationEngine():
	print "Switch from TranslationEngine engine to RotationEngine"
	transeng.dead = True
	rotengine.dead = False
	# Force Recorders
	trans_force.dead = True
	rotation_force.dead = False
	switchEng.dead = True
```

```
if __name__ == "__main__":
    yaPySi = YadePythonSimulation()
    sim1 = yaPySi.run()
```

APPENDIX I

Python Code for Translational Infill Simulations

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
sys.path.append('/home/justin/Desktop/real/yadeimport') #symbolic link to yade
from yadeimport import *

from yade import utils
import random
from yade import ymport
import math
import sys,time

class YadePythonSimulation():
     # Spheres information
     sphereRadius = .004 # [m]
     nu = .48
     #G = 300000 # [Pa]
     den_rub = 9615.2 # [kg/m^3]
     yng_rub = 3000000 # [Pa]
     fric_rub = radians(38)  # [degrees]
     O.materials.append(FrictMat(frictionAngle=fric_rub,young=yng_rub,density=den_rub,
         poisson=nu,label='rubber'))

     ### Steps
     spheres_fall = 8000
     Trans = 167360+spheres_fall
     global All_steps
     All_steps = 1561+spheres_fall+Trans

     pred = pack.inAlignedBox((-.10615,0,-.10615),(0.10615,.025,.10615))
     sp = pack.randomDensePack(pred, spheresInCell=2732, radius=sphereRadius, memoizeDb='/
         home/justin/Desktop/real/tmp/ball_packing65_r4_h25.sqlite', returnSpherePack=True)
     sp.toSimulation(color=(255,0,0),wire=False,material='rubber')

     ## Steel box informaion
     steel = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) # Pa, kg/m^3
     metal_base45 = O.bodies.append(ymport.gmsh("geo/flipped/based_70.mesh",scale=.001,shift
         =Vector3(0,-.00635,0), material=steel,color=(0,0,1)))

     ## Cleat information used steel properties to make rigid
     studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) #
         used steel properties to make rigid
     studs1 = O.bodies.append(ymport.gmsh("geo/flipped/1mm_heightchecker.mesh",scale=.00001,
         shift=Vector3(0.107,.027,0), material=studs_plastic, color=(0,0,1))) # to by pass "
         no studs defined" error... It's a clever trick ;)

     ## Time step set to 20% of Rayleigh Wave
     O.dt=.2*utils.RayleighWaveTimeStep() # 0.000256359572832
     print O.dt

     ## Engines
     O.engines=[
          ###Reset all forces stored in Scene::forces (O.forces in python). Typically, this
              is the first engine to be run at every step.          In addition, reset those
              energies that should be reset, if energy tracing is enabled.
          ## Resets forces and moments that act on bodies
          ForceResetter(),

          ## Using bounding boxes find possible body collisions.
          InsertionSortCollider([
               Bo1_Sphere_Aabb(),
               Bo1_Facet_Aabb(),
          ]),
          InteractionLoop(
               [Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
               [Ip2_FrictMat_FrictMat_FrictPhys()],
               [Law2_ScGeom_FrictPhys_CundallStrack()],
```

```
        ),
        NewtonIntegrator(damping=.2,gravity=[0,-9.81,0],label='newtonInt'),
        # Studs are 19.05 mm tall. Therefore, engine must translate 19.05 mm down.
        TranslationEngine(translationAxis=[0,1,0],velocity=-0.000222*2,ids=studs1,dead=
            True,label='transeng'), # [M/s]
        TranslationEngine(translationAxis=[0,0,1],velocity=2.58,ids=studs1,dead=True,label
            ='forward_push_eng'), # [M/s]
#       RotationEngine(ids=studs1,rotationAxis=(0,1,0), angularVelocity=0.00872665*2,
    rotateAroundZero=True,dead=True,label='rotengine'), # [Rad/s]
        PyRunner(command='turnonstuds()',iterPeriod=spheres_fall, nDo = 11, label='
            switchstuds', dead = False),
        PyRunner(command='switchTranslationEngine()',iterPeriod=Trans, nDo = 6, label='
            switchEng', dead = False),
        # Takes VTK data
#       VTKRecorder(fileName='vtk/r4_',recorders=['all'],iterPeriod=1000),
        ## Record the forces being applied to the studs
        ForceRecorder(ids=studs1,file='r4_found_den_push_z_trans.dat',iterPeriod=1, dead=
            True, label='trans_force'),
        ForceRecorder(ids=studs1,file='r4_found_den_push_z_rot.dat',iterPeriod=1, dead=
            True, label='forward_push_force'),
    ]

    def run(self):
        iterToRun=All_steps
        O.run(iterToRun, True)

def turnonstuds():
    print "Turning on studs"
    studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) #
        used steel properties to make rigid
    studs = O.bodies.append(ymport.gmsh("geo/flipped/studs.mesh",scale=.001,shift=Vector3
        (0,(.022+.01905),0), material=studs_plastic,color=(0,0,1)))
    studs1 = None
    transeng.ids = studs
    forward_push_eng.ids = studs
    trans_force.ids = studs
    forward_push_force.ids = studs
    transeng.dead = False
    trans_force.dead = False
    switchstuds.dead = True

def switchTranslationEngine():
    print "Switch from TranslationEngine engine to RotationEngine"
    transeng.dead = True
    forward_push_eng.dead = False
    # Force Recorders
    trans_force.dead = True
    forward_push_force.dead = False
    switchEng.dead = True

if __name__ == "__main__":
    yaPySi = YadePythonSimulation()
    sim1 = yaPySi.run()
```

APPENDIX J

Python Code for Grass+Infill Simulations

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
sys.path.append('/home/justin/Desktop/real/yadeimport') #symbolic link to yade
from yadeimport import *

from yade.gridpfacet import *
import numpy as np
from yade import utils
from yade import ymport
import time,math,random

class YadePythonSimulation():
     file_name = os.path.basename(__file__)

     # Spheres information
     sphereRadius = .004 # [m]
     nu = .48
     den_rub = 65226 # [kg/m^3]
     yng_rub = 3000000 # [Pa]
     fric_rub = radians(38)  # [degrees]
     O.materials.append(FrictMat(frictionAngle=fric_rub,young=yng_rub,density=den_rub,
         poisson=nu,label='rubber'))

     ### Steps
     spheres_fall = 8000
     global All_steps

     blen = 0.21230  ## Length of Ball pit
     bhei = .025     ## Height of Ball pit
     pred = pack.inAlignedBox((-blen/2,(0+bhei+.001),-blen/2),(blen/2,(bhei+bhei+.001),blen
         /2))
     sp = pack.randomDensePack(pred, spheresInCell=2732, radius=sphereRadius, memoizeDb='/
         home/justin/Desktop/real/tmp/ball_packing65_r4_h25.sqlite', returnSpherePack=True)
     sp.toSimulation(color=(255,0,0),wire=False,material='rubber')


     ## Steel box informaion
     steel = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) # Pa, kg/m^3
     metal_base45 = O.bodies.append(ymport.gmsh("geo/flipped/based_70.mesh",scale=.001,shift
         =Vector3(0,-.00635,0), material=steel,color=(0,0,1)))

     ## Grass Information
     O.materials.append(CohFrictMat(young=3e9,density=1e1,poisson=.3,frictionAngle=10,
         normalCohesion=1e7,shearCohesion=1e7,momentRotationLaw=True,label='grass'))
     rCyl = (0.00635+0.003175)/4  ## Lump with of "grass" on back is 1/4 X 1/8
#    rCyl = 0.0006  ## Grass was about 1.2 [mm] wide
     nL = 2         ## No exact Number here, just trial and error
     L = .022       ## Height of spheres after leveling

     ## Cleat information used steel properties to make rigid
     studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3)) #
         used steel properties to make rigid
     studs1 = O.bodies.append(ymport.gmsh("geo/flipped/1mm_heightchecker.mesh",scale=.00001,
         shift=Vector3(0.107,.027,0), material=studs_plastic, color=(0,0,1))) # to by pass "
         no studs defined" error... It's a clever trick ;)

     ## Time step set to 20% of Rayleigh Wave
     O.dt=.2*utils.RayleighWaveTimeStep()
     print O.dt

     ## Appending Time Step to the output
     str_time_step = "# Time Step = "+ str(O.dt)
     if str_time_step not in open(file_name).read():
          with open(file_name, "a") as myfile:
               myfile.write(str_time_step)
```

```
	## Trans steps
	length_of_cleat = 19.05+.01; # [mm]
	vel_trans_eng = 0.222*2; # [mm/s]
	num_sec_trans = length_of_cleat/vel_trans_eng; # [s]
	Trans = int(round(num_sec_trans/O.dt)) + spheres_fall

	## Rotation steps
	deg_to_rad_length = (60*pi/180); # [rad]
	an_vel_of_cleats = 0.00872665*2; # [rad/s]
	num_sec_roto = deg_to_rad_length/an_vel_of_cleats; # [s]

	num_of_steps_roto = num_sec_roto/O.dt;
	All_steps = int(Trans+num_of_steps_roto+spheres_fall)

	## Engines
	O.engines=[
		###Reset all forces stored in Scene::forces (O.forces in python). Typically, this
			is the first engine to be run at every step.		In addition, reset those
			energies that should be reset, if energy tracing is enabled.
		## Resets forces and moments that act on bodies
ForceResetter(),

	## Using bounding boxes find possible body collisions.
	InsertionSortCollider([
		Bo1_Sphere_Aabb(),
		Bo1_Facet_Aabb(),
		Bo1_GridConnection_Aabb(),
		Bo1_Box_Aabb()
	]),
	InteractionLoop([
		Ig2_Facet_Sphere_ScGeom(),
		Ig2_Sphere_Sphere_ScGeom(),
		Ig2_Box_Sphere_ScGeom(),
		Ig2_GridNode_GridNode_GridNodeGeom6D(),
		Ig2_Sphere_GridConnection_ScGridCoGeom(),
		Ig2_GridConnection_GridConnection_GridCoGridCoGeom(),
		],
		[
		Ip2_CohFrictMat_CohFrictMat_CohFrictPhys(setCohesionNow=True,
			setCohesionOnNewContacts=False),  # internal cylinder physics
		Ip2_FrictMat_FrictMat_FrictPhys()  # physics for external interactions, i.e.,
			cylinder-cylinder, sphere-sphere, cylinder-sphere
		],
		[
		Law2_ScGridCoGeom_CohFrictPhys_CundallStrack(),
		Law2_ScGeom_FrictPhys_CundallStrack(),		# contact law for sphere-sphere
		Law2_ScGridCoGeom_FrictPhys_CundallStrack(), # contact law for cylinder-sphere
		Law2_ScGeom6D_CohFrictPhys_CohesionMoment(), # contact law for "internal" cylinder
			forces
		Law2_GridCoGridCoGeom_FrictPhys_CundallStrack()   # contact law for cylinder-
			cylinder interaction

	]),
		NewtonIntegrator(damping=.2,gravity=[0,-9.81,0],label='newtonInt'),
		# Studs are 19.05 mm tall. Therefore, engine must translate 19.05 mm down.
		TranslationEngine(translationAxis=[0,1,0],velocity=-0.000222*2,ids=studs1,dead=
			True,label='transeng'), # [M/s]
		RotationEngine(ids=studs1,rotationAxis=(0,1,0), angularVelocity=0.00872665*2,
			rotateAroundZero=True,dead=True,label='rotengine'), # [Rad/s]
		PyRunner(command='turnonstuds()',iterPeriod=spheres_fall, nDo = 11, label='
			switchstuds', dead = False),
		PyRunner(command='switchTranslationEngine()',iterPeriod=Trans, nDo = 6, label='
			switchEng', dead = False),
		## Record the forces being applied to the studs
		ForceRecorder(ids=studs1,file=file_name + '_trans.dat',iterPeriod=1, dead=True,
			label='trans_force'),
		TorqueRecorder(ids=studs1,file=file_name + '_rot.dat',iterPeriod=1, dead=True,
			rotationAxis=(0,1,0), label='rotation_force'),
```

```
      ]
      ]

      ### Grass Creation
      ### Create all nodes first :
      nodesIds=[]
      idxc = -1
      x_gap = 0.009 ## Between lumps is roughly 9 [mm]
      z_gap = 0.01905 ## Between lines of backing is .75 inch apart
      range_x = int(math.floor(blen/x_gap)) ## finding the range for x
      range_z = int(math.floor(blen/z_gap)) ## finding the range for z
      cen_z = -(range_z/2)*z_gap  ## Allows the "box" of grass to be center in Z
      #sys.exit()

      for ii in range(0,range_z):
        cen_x = -(range_x/2)*x_gap  # Allows the "box" of grass to be center in X
        for jj in range(0,range_x):
          for i in np.linspace(0,L,nL):
            nodesIds.append(O.bodies.append(gridNode([cen_x,i,cen_z],rCyl,wire=False,fixed=
                False,material='grass')))
          idxc += 1
          d = idxc*nL ## Start of grass fiber
          cen_x += x_gap
      #### Now create connection between the nodes
          for k,j in zip( nodesIds[d:d+nL-1], nodesIds[d+1:nodesIds[-1]+1]):
            O.bodies.append(gridConnection(k,j,rCyl,material='grass'))
        cen_z += z_gap


      for kk in range(0,len(nodesIds)):
        O.bodies[nodesIds[kk]].state.blockedDOFs='xyzXYZ'

      def run(self):
           iterToRun=All_steps
           O.run(iterToRun, True)

def turnonstuds():
      print "Turning on studs"
      studs_plastic = O.materials.append(FrictMat(young=200e9,density=8050,poisson=.3))  #
          used steel properties for inputs but is assumed rigid
      studs = O.bodies.append(ymport.gmsh("geo/flipped/studs.mesh",scale=.001,shift=Vector3
          (0,(.01+.022+.01905),0), material=studs_plastic,color=(0,0,1)))
      studs1 = None
      transeng.ids = studs
      rotengine.ids = studs
      trans_force.ids = studs
      rotation_force.ids = studs
      transeng.dead = False
      trans_force.dead = False
      switchstuds.dead = True

def switchTranslationEngine():
      print "Switch from TranslationEngine engine to RotationEngine"
      transeng.dead = True
      rotengine.dead = False
      # Force Recorders
      trans_force.dead = True
      rotation_force.dead = False
      switchEng.dead = True


if __name__ == "__main__":
     yaPySi = YadePythonSimulation()
     sim1 = yaPySi.run()
```