2016-05-01

Finite Element Modeling of Shallowly Embedded Connections to Characterize Rotational Stiffness

Trevor Alexander Jones
Brigham Young University - Provo

Finite Element Modeling of Shallowly Embedded Connections to Characterize Rotational Stiffness

Trevor Alexander Jones

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment for the requirements for the degree of

Master of Science

Paul Richards, Chair
Fernando S. Fonseca
Michael Scott

Department of Civil and Environmental Engineering

Brigham Young University

May 2016

ABSTRACT

Finite Element Modeling of Shallowly Embedded Connections
to Characterize Rotational Stiffness

Trevor Alexander Jones
Department of Civil and Environmental Engineering, BYU
Master of Science

Finite element models were created in Abaqus 6.14 to characterize the rotational stiffness of shallowly embedded column-foundation connections. Scripts were programmed to automate the model generation process and allow study of multiple independent variables, including embedment length, column size, baseplate geometry, concrete modulus, column orientation, cantilever height, and applied axial load. Three different connection types were investigated: a tied or one part model; a contact-based model; and a cohesive-zone based model.

Cohesive-zone modeling was found to give the most accurate results. Agreement with previous experimental data was obtained to within 27%. Baseplate geometry was found to affect connection stiffness significantly, especially at lower embedment depths. The connection rotational stiffness was found to vary only slightly with cantilever height for typical column heights. Results from varying other parameters are also discussed.

Keywords: finite element modeling, finite element analysis, lateral stiffness, rotational stiffness, shallowly embedded connections, embedment, column connections, stiffness

ACKNOWLEDGEMENTS

The author would like to thank the American Institute of Steel Construction (AISC); the BYU Office of Research and Creative Activities (ORCA); the BYU Civil and Environmental Engineering Department Scholarship foundation; and the Dean K. Fuhriman family for funding this research, either directly or indirectly.

The author would like to thank all those who were involved with him in his research: Nicholas Barnwell, Joshua Tryon, Leah O'Neill, Kevin Hanks, J. Dean Anderson, Billy Wilson, and Dr. Gary Prinz.

The author is especially grateful for the contributions of his graduate advisor, Dr. Paul Richards, and the other members of his advisement board, Dr. Michael Scott and Dr. Fernando Fonseca.

Finally, the author deeply appreciates the support and encouragement given by his family members, especially his father Marvin, mother Rebecca, and wife Samantha.

TABLE OF CONTENTS

LIST OF TABLES

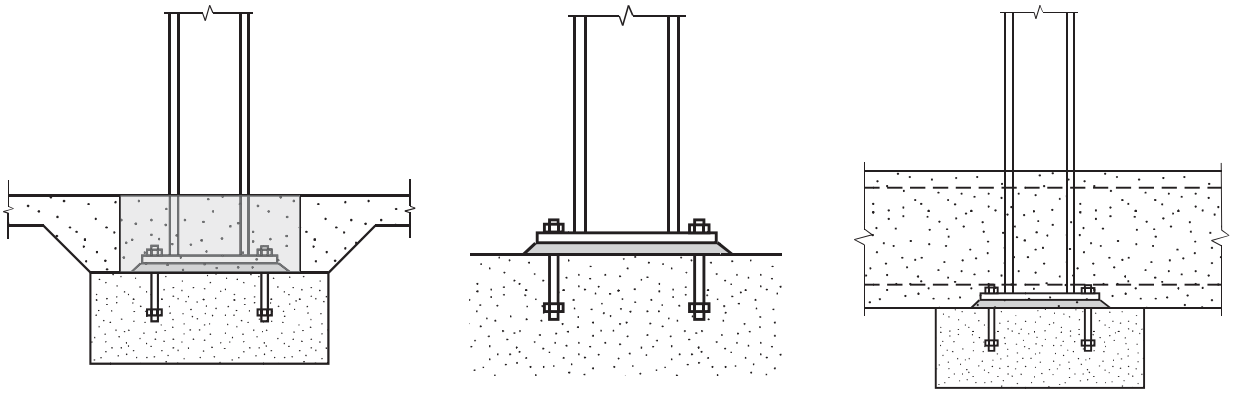
LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Steel buildings are relatively common in the United States and other industrialized nations. Steel is a material that is relatively strong, lightweight, and durable, and design with steel has advanced significantly over the last hundred years. Steel buildings typically require foundations that are constructed from reinforced concrete. The connections between steel columns and concrete foundations typically come in one of three varieties: shallowly embedded, exposed, or deeply embedded. Figure 1-1 illustrates the three connection types. This thesis will focus on shallowly embedded connections.

Shallowly embedded connections are comprised of a column, a baseplate, anchor rods, slab on grade, grout and a blockout concrete area. Figure 1-2 illustrates this configuration. A baseplate is welded to the bottom of the column, which distributes forces and moments into the foundation. Anchor rods are embedded into the foundation and protrude sufficiently to allow an interface with the baseplate. Beneath the baseplate is typically a layer of high strength, non-shrink grout. The column itself is installed in a sizeable void left in the slab on grade, called a blockout, which allows for the installation of the column specimen and grout layer. After installation is complete, the blockout is filled with additional concrete. This process allows the work of the concrete and steel contractors to remain independent of each other.



(a) Shallowly Embedded **(b) Exposed** **(c) Deeply Embedded**

Figure 1-1: Base Column Connections (Barnwell, 2015)

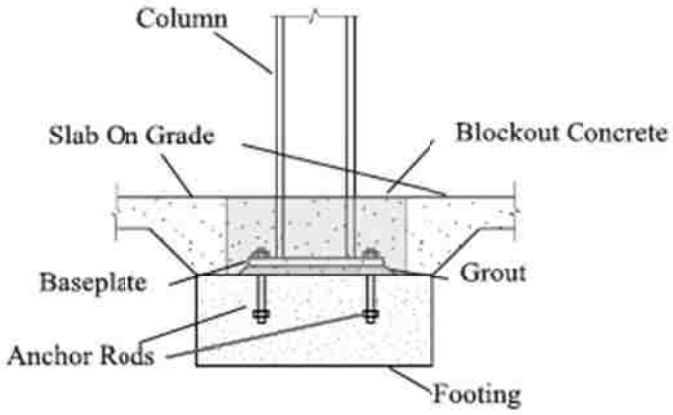


Figure 1-2: Shallowly Embedded Connection Detail

The other two types of connections show in Figure 1-1 will also be briefly discussed. In exposed connections, the baseplate and column are not embedded at all into the concrete. These connections are common in industrial buildings, where aesthetics are less important. By contrast, deeply embedded connections are embedded into the grade beam, and so have reinforcing bar (rebar) running continuously above the baseplate. These connections can be expensive and

complex, and require coordination between steel and concrete contractors. They are typically reserved for connections in moment frames, where a fixed-type connection is required. Peripheral studies of these connections are treated in this thesis' literature review, since their behavior is relatively well-understood, and some aspects of their behavior relate to shallowly-embedded connections.

Pile cap connections also show some similarities to shallowly embedded connections, in that they both consist of a steel member embedded several inches into a reinforced concrete slab. The literature review will discuss several studies of pile cap connections. Pile cap connections connect driven steel piles or piers to a concrete mat foundation (see Figure 1-3). Studies of pile cap connections have shown that piles with minimal reinforcement and embedment can develop significant moments. These tests strongly support the idea that shallowly embedded connections have significant rotational stiffness and strength.

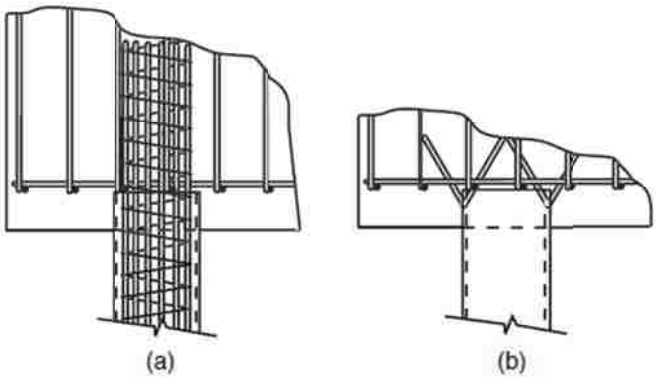


Figure 1-3: Typical Pile Cap Connections; fixed (a) and pinned (b) (Richards et al., 2011)

1.2 **Motivation**

Structural engineers typically model connections as either fixed or pinned. If the flexural capacity of the connection exceeds the column strength, the connection is considered fixed.

Otherwise, it is considered pinned. This approach is not entirely unreasonable, but it does confuse the issues of stiffness and strength. Since shallowly embedded connections are not usually designed to transmit moments, they are considered pinned. However, practicing engineers readily admit to not knowing their flexural strength or stiffness (Davis, 2011; Malley, 2011).

The presence of blockout concrete is thought to provide additional strength and stiffness to the connection, although these have not been characterized thoroughly. Some experiments with pile caps have demonstrated that similar connections possess non-negligible stiffness and strength, due to the confining effects of concrete embedment. This suggests that the same may be present in shallowly embedded connections.

Quantifying stiffness of shallowly embedded connections will improve future engineering models. Researchers have successfully modeled building seismic response with base connections modeled as rotational springs (Zareian and Kanvinde, 2013). Base flexibility was shown to affect various aspects of seismic response, including interstory drift and the shaking intensity associated with collapse. Once the stiffness of shallowly embedded connections is quantified, they can be represented with rotational springs with particular stiffness.

Although they are the most common connection types in current construction practice, shallowly embedded connections remain the least understood (Grauvilardell et al., 2006). Recent work has successfully characterized strength and stiffness values for exposed connections (Gomez, 2010; Kanvinde et al., 2015), and progress has been made for deeper connections (Grilli, 2015), but progress remains lacking for shallowly embedded connections. Most of the column base research that has been conducted in the United States has concentrated on

connections with exposed baseplates; despite this, the use of both shallowly-embedded and deeply-embedded types has been common in the U.S. (Grauvilardell, 2006).

Moment frames are typically governed by drift considerations, which are highly sensitive to the columns' boundary conditions. By improving the fixity of the end conditions in current models, the drift will likely be reduced, and smaller column shapes will be acceptable choices for future moment frames.

Column buckling capacities are also highly sensitive to boundary conditions. By quantifying the end fixity of gravity columns, additional buckling capacity may be demonstrated for columns in steel buildings with shallowly embedded connections.

1.3 **Shallowly Embedded Connection Research Program**

To investigate shallowly embedded connections further, a research program is underway at Brigham Young University, of which this thesis is a part. In Phase I of the research, Barnwell (2015) constructed and tested 12 laboratory specimens of shallowly embedded connection details, typical for gravity bearing columns (see Figure 1-4). These tests demonstrated the presence of stiffness and strength in these connections. Additionally, a strength model was proposed. This thesis is a part of Phase II, and involves using finite element models to determine and predict stiffness for these connections. Concurrently with this research, Tryon (2016) is using closed-form elastic models to predict the stiffnesses of these connections. In Phase III, a second set of laboratory specimens will be constructed and tested to verify the strength and stiffness models and extend the findings.

Figure 1-4: Typical Laboratory Testing Setup (Barnwell, 2015)

1.4 Objective

The purpose of this research is to investigate the effects of key input variables on the rotational stiffness of shallowly embedded connections, using finite element simulations. These key variables include blockout/embedment depth, baseplate geometry, column size/orientation, stiffness of grout/concrete, and applied axial loading. For the purposes of this study, the main variable studied will be embedment depth – specifically, how the connection stiffness increases as embedment depth increases.

Finite element models of Barnwell's experiments were created in Abaqus 6.14. Model generation was automated through scripts which created and processed a large number of

models. The results from these models were then aggregated and analyzed to discover trends in rotational stiffness values.

The results from the finite element analysis were validated by comparing stiffness values to existing test data from Barnwell's experiments. In addition, the displacement contour plots generated by the computer models were compared qualitatively to displacement contour plots generated by the DIC system during experimental testing.

1.5 **Thesis Outline**

The rest of this thesis is organized as follows:

- Chapter 2 presents a literature review, including an in-depth review of the parent study (Barnwell, 2015).
- Chapter 3 explains the methods that were used to generate and analyze the finite element models.
- Chapter 4 examines the Digital Image Capture (DIC) information from Barnwell's experiments.
- Chapter 5 presents, discusses, and analyses the results that were obtained from the Finite Element Models, and discusses their broader applicability.
- Chapter 6 explains the conclusions which were reached as a result of this study, and possibilities for future research.
- Appendices A-D include additional information associated with the generation and validation of the models used in this thesis.

2 LITERATURE REVIEW

2.1 Corbel Connection Strength

Marcakis and Mitchell (1980) performed tests on 25 connections of steel members embedded into precast concrete. Their tests studied the effects of the following parameters on connection strength: inclusion of column axial load; welding reinforcement to the connection; the shape of the embedded member; load eccentricity. This research has formed the basis of subsequent design strength calculations for steel members embedded into concrete members (see PCI Handbook, 1999).

2.2 Pile Cap Connections

The U.S. Army Corps of Engineers (Castilla et al. 1984) performed two separate numerical analyses of the fixity of steel members embedded in concrete – specifically pile caps. The first numerical analysis used a series of independent springs to model the cap, the member, and the soil. Springs with linear characteristics represented the concrete cap, and springs with non-linear characteristics represented the soil. The second used 2 dimensional linear and non-linear finite element models, neglecting the soil and instead modeling only the member and the concrete. Figure 2-1 shows the setup, as well as the ranges of parameters investigated. Members studied were HP steel shapes. Based on the results of both investigations, it was determined that

pile embedment depths of two times the pile's depth was usually sufficient to develop a fully fixed condition. It was also determined that partial fixity could be achieved with as little as one foot of embedment; specifically, a 1 foot embedment length could develop 61 to 83% the total moment as a 4 foot (fully fixed) embedment.

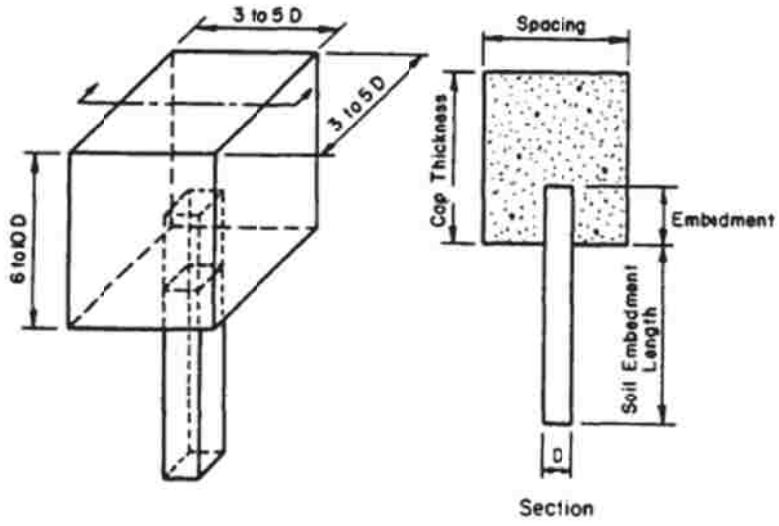


Figure 2-1: Model Parameters and Ranges (Castilla et al., 1984)

Xiao et al. (2006) performed tests for HP-steel pile-to-pile cap connections with shallow embedments, studying the structural response and capacity of the connection. The corner portion of one pile footing from a prototypical bridge was simulated, using an HP 14x89 shape. The system is shown in Figure 2-2. V-shaped anchor bars were tied to the footing reinforcement to prevent them from moving during concrete placement; they were anchored to the web edges of the HP pile by tying them through holes in the pile. This provided additional strength and stiffness to the system. Tests were performed under both cyclic horizontal and cyclic vertical loads.

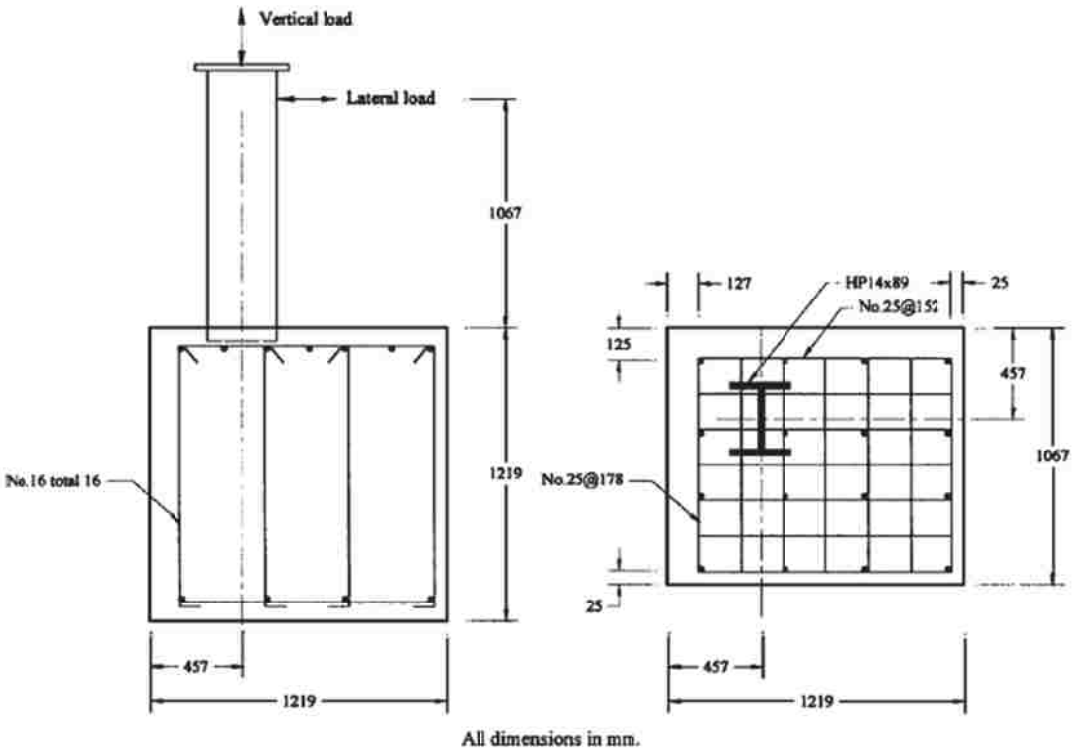


Figure 2-2: Full-Scale Pile to Pile Cap Subassembly Model Details (Xiao et al., 2006)

The results for cyclical vertical forcing showed high strength and stiffness in compression, but little in tension. The lack of strength in tension was attributed to a rocking response in the cap. Ultimate failure occurred when the V-shaped anchor bars ruptured. This occurred at tensile forces much lower than the ultimate design capacity of 890 kN (200 kips).

In combined horizontal and vertical loading, the pile demonstrated ultimate maximum compression capacities of 3,123 and 3,619 kN (702 and 814 kips) for the strong and weak axis specimens, respectively. These doubled the design ultimate capacity of 1,779 kN (400 kips).

Although the piles were designed as a pinned connection, the connection developed considerable moment strength. Moment capacity ranged from 0.25 to 0.66 M_p, where M_p is the

plastic moment of the steel section tested. These results were compared to the ultimate moment capacity results from other similar experiments (Shama et al., 2002), and found to be significantly greater than the 0.06 M_p observed in the other experiments. High levels of initial stiffness were also observed.

Richards et al. (2011) reported the strength of four separate pile-to-cap connections, determined using in-field specimens. Figure 2-3 shows a schematic overview of the specimens tested, while Figure 2-4 shows the test setup. Two of the specimens had pile-to-cap reinforcement, while three had a concrete fill inside of the pile. The first specimen had rotational strengths that far exceeded its expected moment capacity, as calculated using the Marcakis and Mitchell equations. The load-deflection response of all four specimens behaved more like what would be expected from a fixed connection than a pinned connection, implying greater rotational stiffness. The specimen demonstrated significantly greater strength than would have been expected from the Marcakis and Mitchell equation alone.

Several possible mechanisms for the additional observed strength were suggested. One was the dowel action from the two bottom grid bars that passed through the piles. However, this mechanism alone was insufficient to account for the observed strength. Another suggested mechanism was a frictional force between the steel pile and the concrete pile cap. By using a static friction coefficient of 0.47, and combining the capacity from the frictional force with the capacity of the dowels, sufficient moment capacity was obtained to account for the observed strength. It was concluded that friction at the concrete/steel interface may play a significant role in resisting lateral forces and their induced moments, although there was not enough experimental data to validate this proposed friction mechanism.

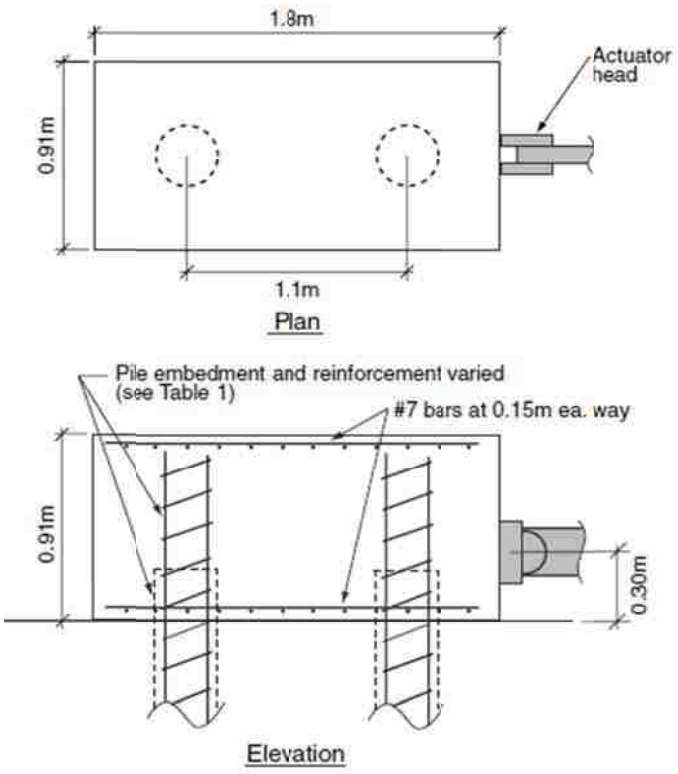


Figure 2-3: Specimen Overview (Richards et al., 2011)

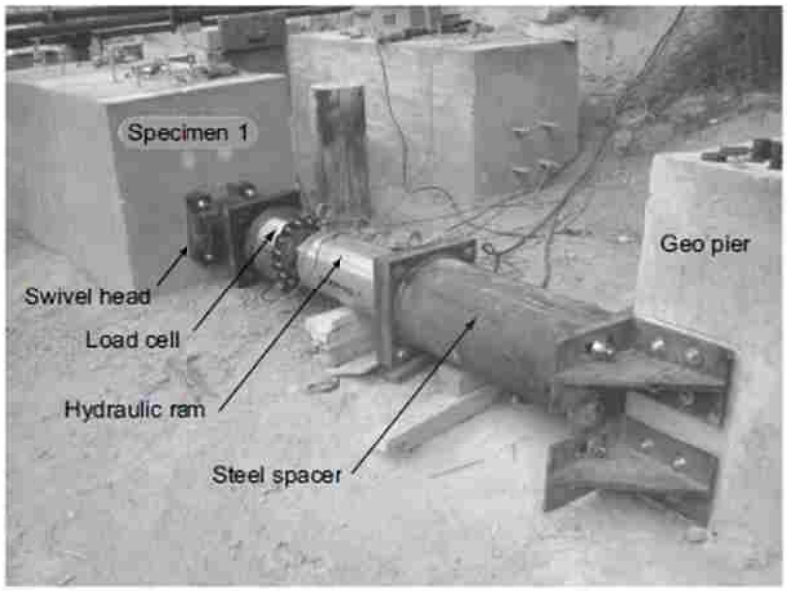


Figure 2-4: Test Setup (Richards et al., 2011)

Eastman (2011) tested three pile cap specimens to study the strength and stiffness of the pile-to-pile cap connections. Specimens consisted of 12" diameter hollow circular steel piles, embedded into reinforced concrete pile caps, with a steel cover plate welded to the embedded end (see Figure 2-5). In each of the three specimens studied, significant moments developed, despite the embedment being only between 0.4 and 1.5 times the pile diameter. The results are summarized in Table 2-1. It was found that the pile-to-cap connections had additional strength that could not be accounted for using the Marcakis and Mitchell equation.

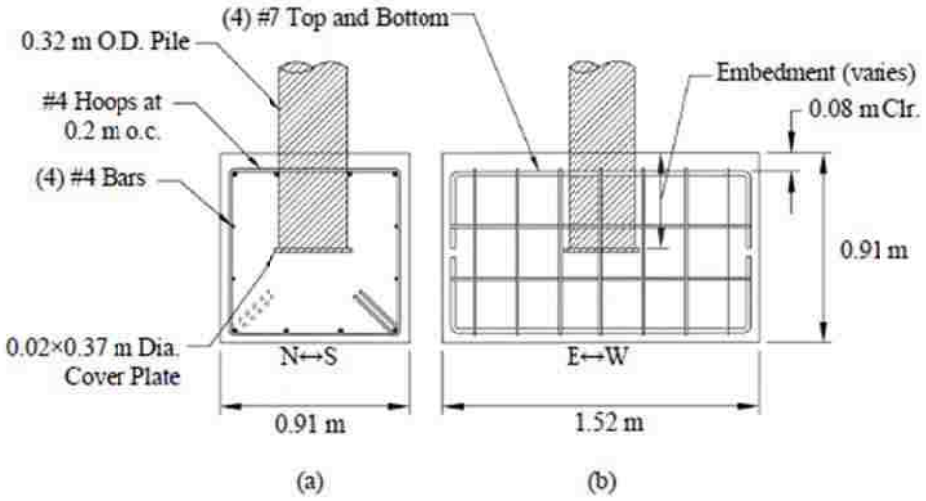


Figure 2-5: Pile-To-Cap Connection Design, a) End View, b) Side View (Eastman, 2011)

Table 2-1 –Test Data Summary (Eastman, 2011)

	Test #1	Test #2	Test #3
Embedment depth (x pile diameter)	1.5	0.5	0.4
M_{ult} developed (kip-ft)	288	75.3	39.5
Max. lateral load with elastic response (kips)	32.4	11.8	6.7

It was also shown that rotational stiffness is available in pile cap connections. To characterize the stiffness observed, it was necessary to subtract the deformation caused by the pile's deformation, according to the following formula:

$$k_{conn} = ({k_{tot}}^{-1} - {k_{col}}^{-1})^{-1}$$

Where

k_{conn} = connection stiffness

k_{tot} = total stiffness

= 1/ total displacement

k_{col} = column stiffness

= $3EI/L^3$

Figure 2-6 illustrates this stiffness model graphically. The total deformation, measured at the point of applied load, consists of the column deformation and the connection deformation. Therefore, the connection deformation can be determined indirectly by subtracting the calculated column deformation from the measured total deformation. The applied load was then divided by this applied deformation to obtain the connection's stiffness. Using this stiffness model, Figure 2-7 shows the connection stiffness values measured for the three test specimens.

2.3 **Exposed Baseplate Connections**

Insights into the behavior of shallowly-embedded connections can be gained by understanding the results of tests on exposed connections. These exposed connections serve as a "lower bound" on the strength and stiffness available from shallowly embedded connections. (Cui et al., 2009) Exposed connections are similar to shallowly embedded connections, but without an additional layer of non-structural concrete above the baseplate.

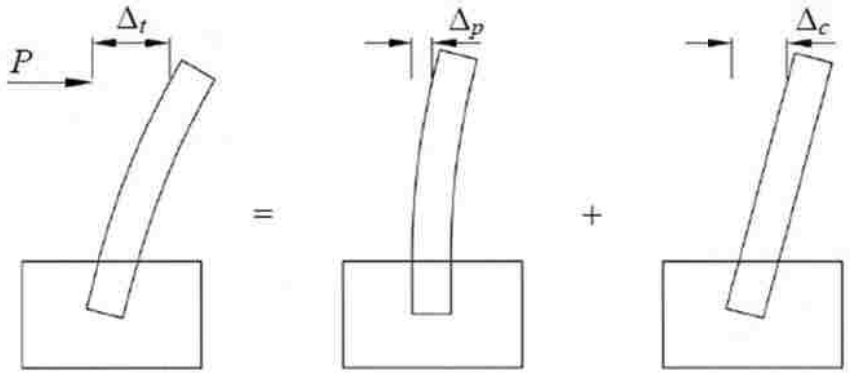


Figure 2-6: Elastic Stiffness Mechanism (Eastman, 2011)

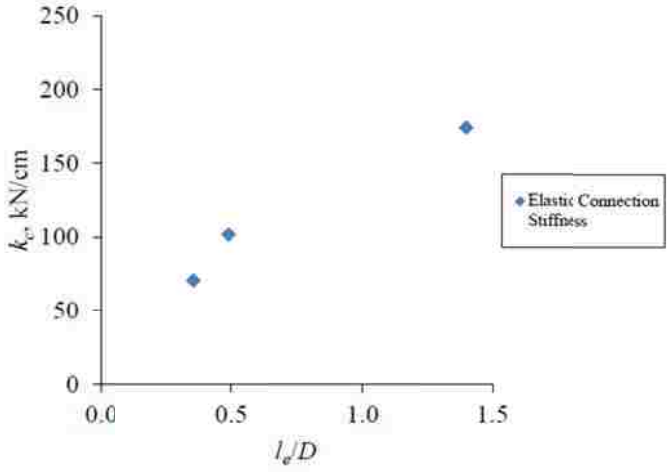


Figure 2-7: Elastic Connection Stiffness vs. Normalized Pile Embedment Depth (Eastman, 2011)

2.3.1 Experimental Study of Exposed Baseplate Connection Strength and Stiffness

DeWolf and Sarisley (1980) studied the behavior of column baseplates subjected to applied axial load and overturning moment. Sixteen test specimens investigated anchor bolt size, baseplate thickness, and the ratio of the moment to the axial load (eccentricity). The test setup

was doubly symmetric, with identical column specimens at the top and bottom of the concrete pedestal. No reinforcement was used.

At low eccentricities, concrete crushing was the typical failure mode, while for higher eccentricities, anchor bolt failure (yielding) occurred first. When anchor bolt failure was the primary failure mode, it was accompanied by yielding in the concrete itself.

Picard & Beaulieu (1984) investigated fifteen (15) specimens of exposed-type connections. Theoretical deflection relations were derived for both pinned and fixed connections, and the experimental stiffness results were compared to the theoretical values. It was found that axial compression increases the fixity factor (related to the rotational stiffness) of the connection, and that the assumption of these connections as pinned is conservative.

Thambiratnam et al (1986) tested a total of twelve (12) exposed baseplate connection specimens under combinations of axial loads and moments. Only baseplate behavior was investigated. The failure modes that were reported were concrete crushing, baseplate yielding, and anchor bolt yielding (see Figure 2-8). At low eccentricity, baseplates failed from cracking of concrete. In all other cases, however, baseplate yielding (accompanied by anchor bolt yielding) was the primary failure mechanism.

Wheeler et al. (1998) investigated the flexural strength of exposed connections involving rectangular HSS column shapes. Sixteen (16) tests were performed, varying the models' plate dimensions and section types. Analytical models were proposed which accurately predicted the ultimate moment capacities of the connections to within 15%.

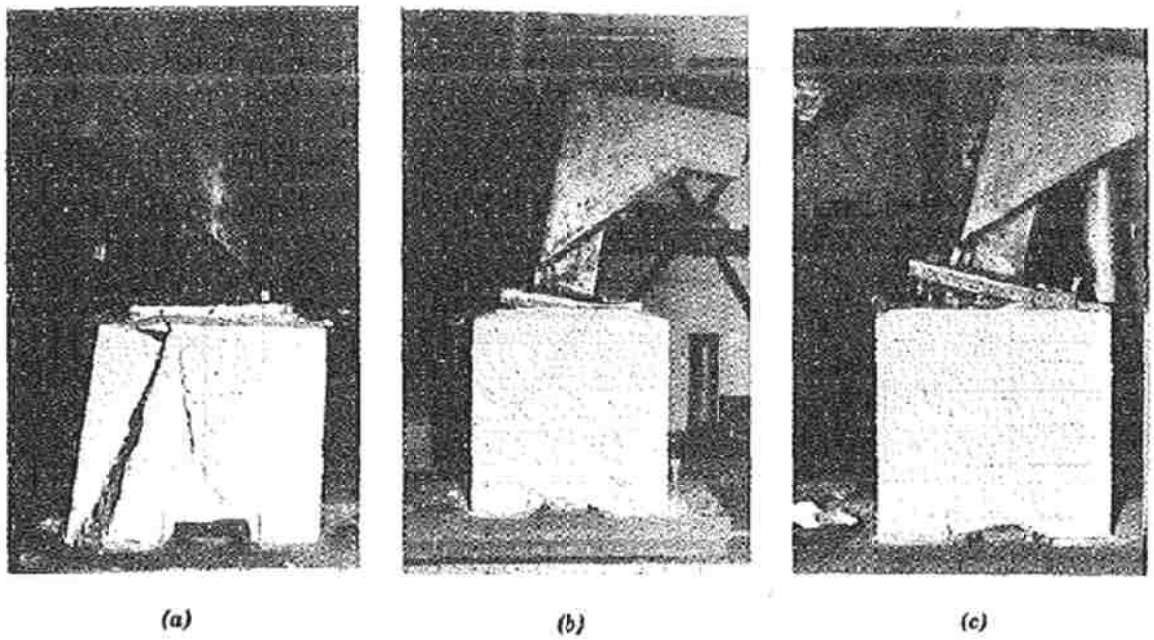


Figure 2-8: Observed Failure Modes: a) Concrete Crushing; b) Baseplate Yielding; c) Anchor Bolt Yielding (Thambiratnam et al., 1986)

Meyers et al (2008) preformed six tests to investigate the effects of weld details on exposed connection response. The column specimens were W8x67 shapes, and represented 2/3 scale models of typical details. Cracking failures were reported, beginning in the heat affected zones of welds, in both complete joint penetration (CJP) and partial joint penetration (PJP)-type welds. Ductile cracks initiated at column drift ratios of 3%-5%, and brittle weld failure occurred at drift ratios of 5%-9%.

Gomez (2010) conducted an experimental study of exposed baseplate connections on fourteen large-scale specimens. The objectives were to investigate and characterize the response of exposed baseplate connections to lateral loads, and develop a model to predict their strength. Seven specimens (Phase I experiments) were tested to investigate shear transfer mechanisms,

including surface friction, anchor rod bearing, and shear key bearing. The remaining seven (Phase II experiments) studied the effects of combined flexural and axial compression loading on various configurations of baseplate design. The test setup for Phase II is illustrated in Figure 2-9.

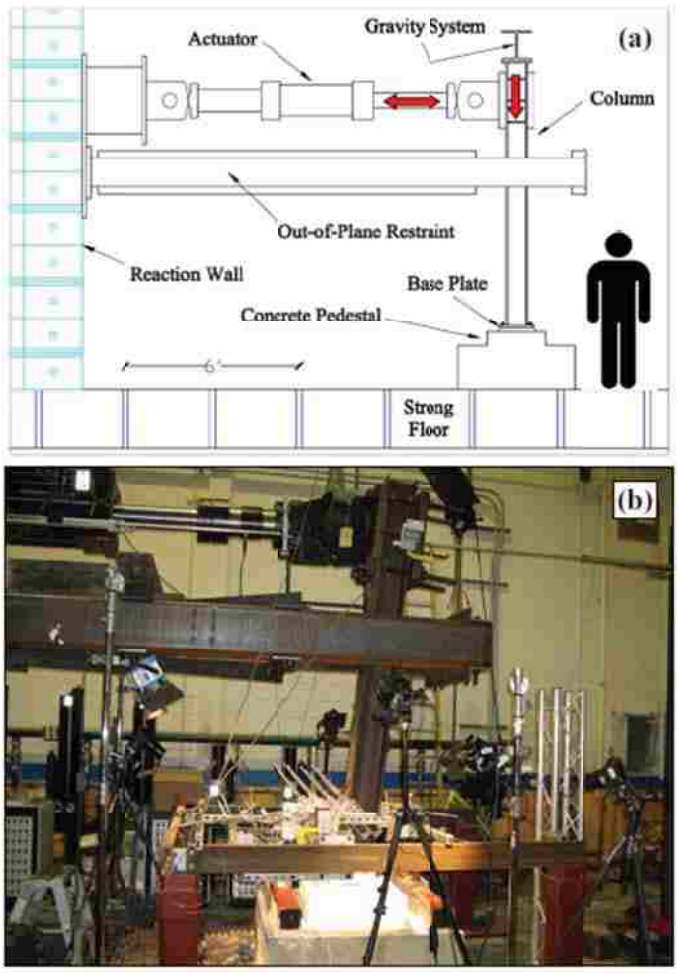


Figure 2-9: Phase II Test Setup - (a) Schematic and (b) Photograph (Gomez, 2010)

In tests without applied axial load, the rotation-moment curves showed a strongly linear response, suggesting a constant rotational stiffness in cases without applied axial loads. In cases with applied axial load, there was a bi-linear stiffness response (see Figure 2-10). The axial load

delayed baseplate uplift, increasing stiffness. The point of transition between the two slopes marks the point at which the anchor rods yielded in tension, and represents a significant decrease in rotational stiffness. The stiffness prior to baseplate uplift was governed by the stiffness of the grout/concrete foundation, while stiffness afterward was controlled by the anchor rods and uplifting baseplate.

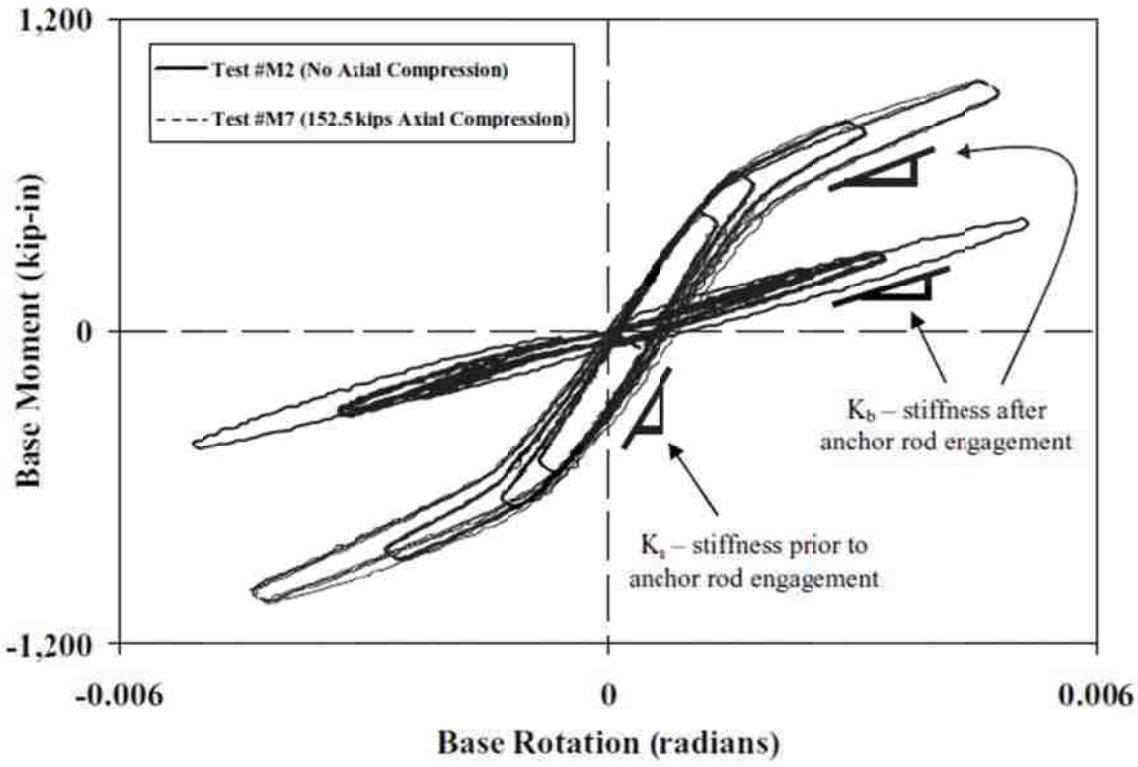


Figure 2-10: Representative Plot of Elastic Rotational Stiffness (Gomez, 2010)

In addition to initial tangent stiffness, secant stiffness values were recorded at 1% and 2% drift levels. In tests without axial load, the stiffness decreased with increasing drift levels; however, in tests with axial load, the stiffness corresponding to 1% drift was higher than the

initial stiffness after anchor rod engagement. Initial stiffness values were insensitive to the level of axial load, while the secant stiffness values increase with axial load.

2.3.2 Analytical Study of Exposed Baseplate Stiffness

Grilli (2015) proposed a model for predicting connection stiffness in exposed baseplate connections. This model is calibrated with the experimental studies of Gomez (2015), and Picard and Beaulieu (1984). The connection response can be complex, as deformations within the various components can control the response of the connection as a whole. That is, the deformation can be affected by: deformation of the baseplate on the compression side; baseplate deformation on the tension side; elongation of the anchor rods in tension; and deformation of the concrete or grout under the compression side of the baseplate. Also, they may interact in different ways at different times (e.g. creating a gap between the baseplate and the grout). The assumed deformation mode is represented in Figure 2-11.

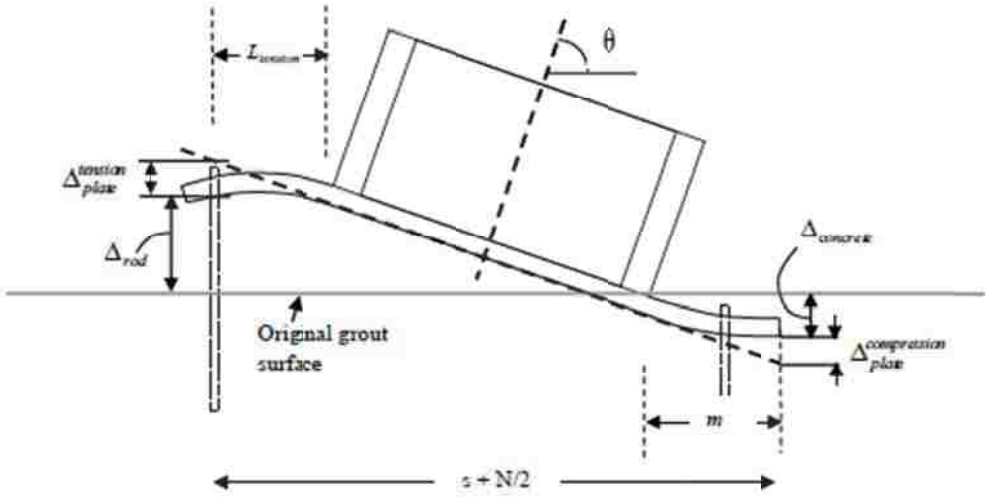


Figure 2-11: Assumed Deformation Mode (Grilli, 2015)

This model proposes methods to determine the contribution each of the above deformations to the overall response, and then sums them to obtain a total deformation. The total deformation is divided by the length of the baseplate to obtain the total rotation, according to the following formula:

$$\theta_y = (\Delta_{rod} + \Delta_{plate}{}^{tension} + \Delta_{plate}{}^{compression} + \Delta_{concrete})/(s + \frac{N}{2})$$

Where

θ_y = rotation at first yield.

(s + N/2) = the distance between the compression toe of the baseplate, and the centerline of the anchor rods on the tension side.

The precise methods of calculating Δ(rod), Δ (tension, plate), Δ(compression, plate), and Δ(concrete) are detailed in Grilli (2015).

Calculating the rotational stiffness is then trivial, i.e.

$$\beta_y = {M_y}/{\theta_y}$$

Where

β_y = Rotational stiffness at point of first yield

M_y = Applied moment at point of first yield

2.4 **Deeply Embedded Connections**

2.4.1 Grilli & Kanvinde (2015)

Grilli and Kanvinde (2015) investigated deeply embedded connections, referred to in their research as embedded column base (ECB) connections. Five (5) specimens were created;

overall geometry is shown in Figure 2-12, and specimen properties are summarized in Table 2-2. The variables investigated included column size; axial load direction (compressive, zero, or tension); embedment depth; baseplate dimensions; and cantilever height, defined as the distance between the load center line and the top of the concrete pedestal.

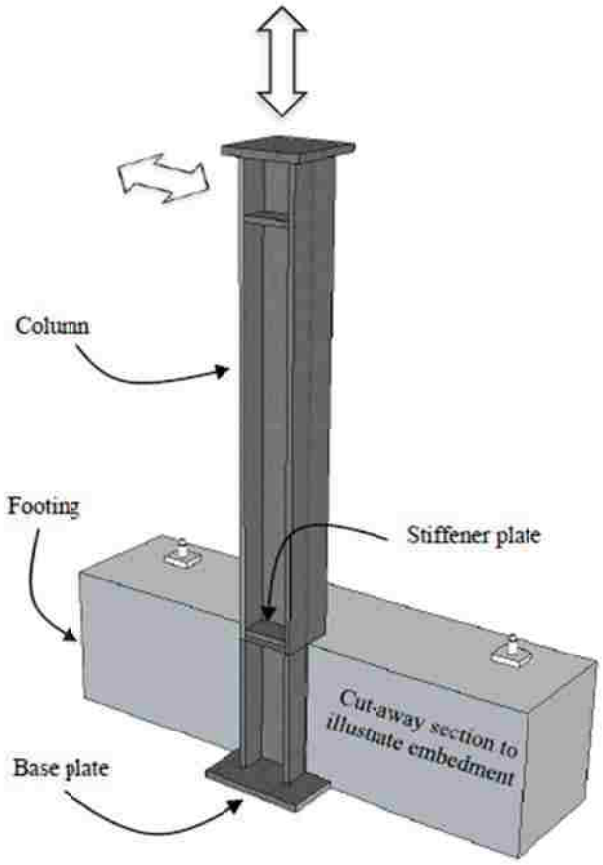


Figure 2-12: Schematic Illustration of Embedded Column Base Connection (Grilli, 2015)

Table 2-2: Test Matrix and Results (Grilli, 2015)

Test #	Column Size, (b_f [in])	P [kips]	d_{embed} [in]	Base Plate, $t_p \times N \times B$ [in]	z [m]	M_{base}^{max} [k-ft]	β_{base}^{test} * [10^5 k-ft/rad]	$\frac{\Delta_{test}}{\Delta_{fixed}}$ ***	$\frac{M_{base}^{y}}{M_{base}^{max}}$	Δ_{max} (%)
1	W14x370 (16.5)	100 (C)	20	2×30×30	112	1902(+)	2.38	1.21	0.87	3.85
						1927(-)			0.69	3.82
2	W18x311 (12)			2×34×28		1714(+)	2.83	1.16	0.709	3.01
						1599(-)			0.66	2.89
3	W14x370 (16.5)	0	30	2×30×30	122	2759(+)	2.26	1.30	0.72	6.97
						2540(-)			0.67	7.77
4		100 (C)				3042 (+)	2.49	1.30	0.66	6.48
						2664(-)			0.81	5.09
5		150 (T)				2803 (+)	2.40	1.29	0.73	2.72**
						2555(-)			0.72	2.65**
							Mean	**1.25**	**0.72**	**4.98**
							CoV	**0.05**	**0.07**	**0.38**

*Average stiffness of both directions
**Test terminated due to slip prior to failure (Mean, COV does not include these data points)
***Notional columns used for Tests #1-5 are W14X145, W14X132, W14X193, W14X211, and W14X193 respectively.

Construction details were chosen that are common in construction practice. Figure 2-13 shows a schematic illustration of the connection detail, which was similar in all the experiments. Plates were installed at the top and bottom of the embedment region for stability and uplift considerations.

The test specimens were loaded laterally with a cyclic loading protocol. The deformation history was expressed in terms of column drift ratio, defined as the lateral displacement of the column at load center line, divided by the cantilever height. The loading rate was quasi-static, with a loading rate of less than 1.8% drift / minute. Axial loads were applied before the test began and held constant throughout the test.

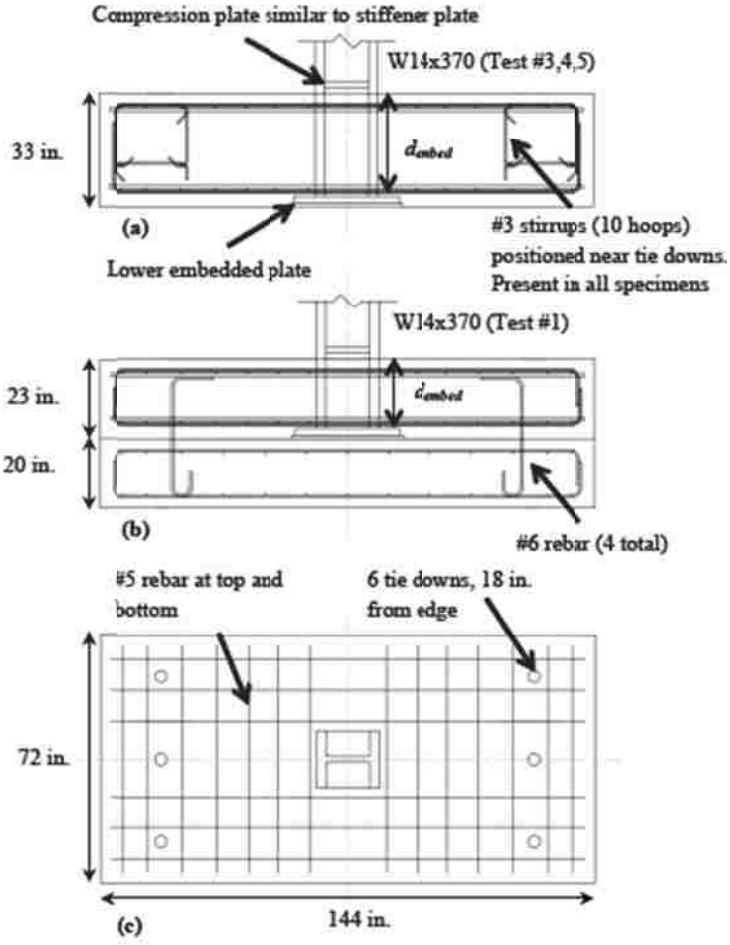


Figure 2-13: Schematic Illustration of Experimental Specimens. (a) 30-inch Embedment; (b) 20-inch Embedment; (c) Plan View, Common to Both (Grilli, 2015)

The specimens exhibited linear response until drifts reached 0.005 radians (0.5% drift), after which nonlinear response nonlinear response began and increased gradually. Concrete spalling began at approximately 1% drifts, at which point the load-displacement curve became highly nonlinear. Ultimate failure in the test specimens occurred in one of two modes. In the more shallowly embedded connections, a cone of concrete on the tension side of the baseplate

failed suddenly and experienced uplift. In the more deeply embedded connections, concrete crushing and spalling occurred at the extremities of the embedded region.

Deeper embedment depths led to increased strength, due to an increased bearing capacity, and increased resistance to the tension-uplift failure mode. Applying a compressive axial force increased connection capacity, while a tensile force reduced connection capacity. Connection yield strength was approximately 70% of maximum (ultimate) connection strength (see Figure 2-14 and Table 2-2). A model was proposed to characterize the strength of ECB connections which accurately modeled the actual strength.

Research was also conducted on the rotational stiffness of the column connections by Grilli and Kanvinde (2015). The investigation focused on the secant stiffness at base yield moment, $M^{y}_{base} = 0.7*M^{max}_{base}$. Although there was no distinct yield point to the specimens (the stiffness varied gradually), a least-squares bilinear curve fitting found the transition point between the two lines occurred at an average of 0.72*M(base, max). The value of 0.70 was chosen for computational convenience.

ECB type connections typically allow deformations of 1.25x those expected in the case of a perfectly fixed connection (see Table 2-2). That is, the average of all $\Delta_{test}/\Delta_{fixed}$ values was 1.25. However, this increase in drifts is relatively small when compared to the deformations expected with exposed baseplate connections. Counterintuitively, increasing the embedment depth was found to cause an increase in average column deflection in some cases (compare deflection results from Tests 1-2 to 3-5 in Table 2-2). The authors reason that this may be due to the increase in bending length in the column, which is not offset by the increase in concrete bearing.

2.4.2 Other Deeply Embedded Connections

Pertold et al. (2000a) performed strength tests of two sets of embedded column specimens in axial loading. The first set of specimens had no baseplate, and so it tested the strength of the concrete-steel bond, independently of any baseplate strength contributions. The second set of specimens had grease applied to the column before concrete placement, preventing a bond from forming, allowing isolation of baseplate bearing strength, independently of any bond strength. Figure 2-15 shows the test setup for bond strength; the test setup for baseplate bearing strength is similar; the only major difference is the presence of a baseplate. Figure 2-16 shows the observed failure mode for bearing strength tests.

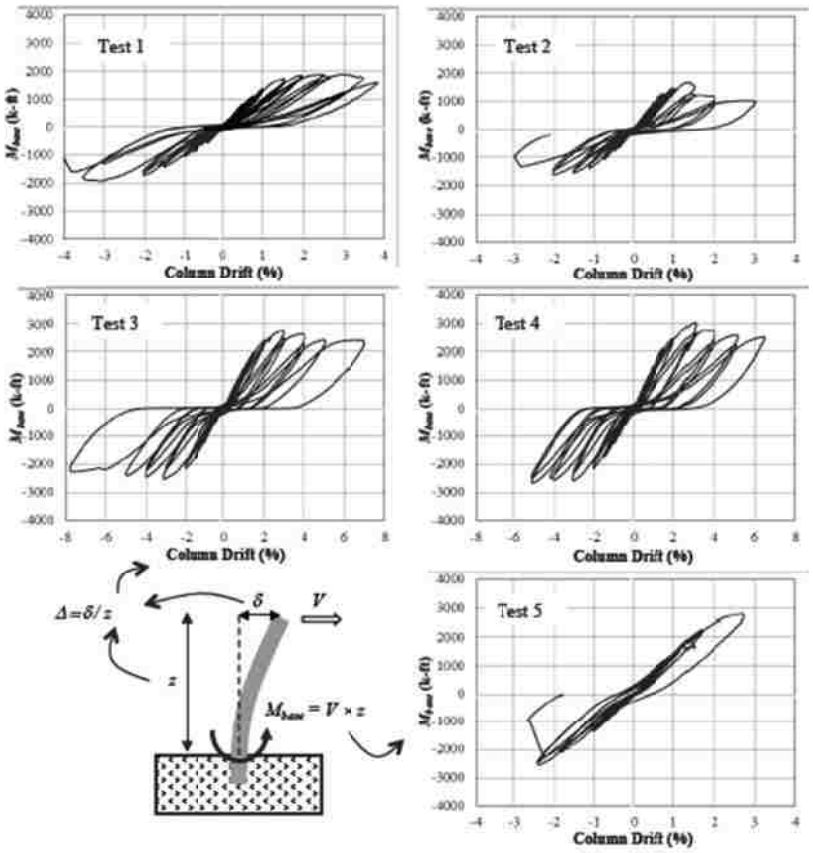


Figure 2-14: Moment Drift Plots, and Schematic Illustration of Plotted Quantities (Gomez, 2015)

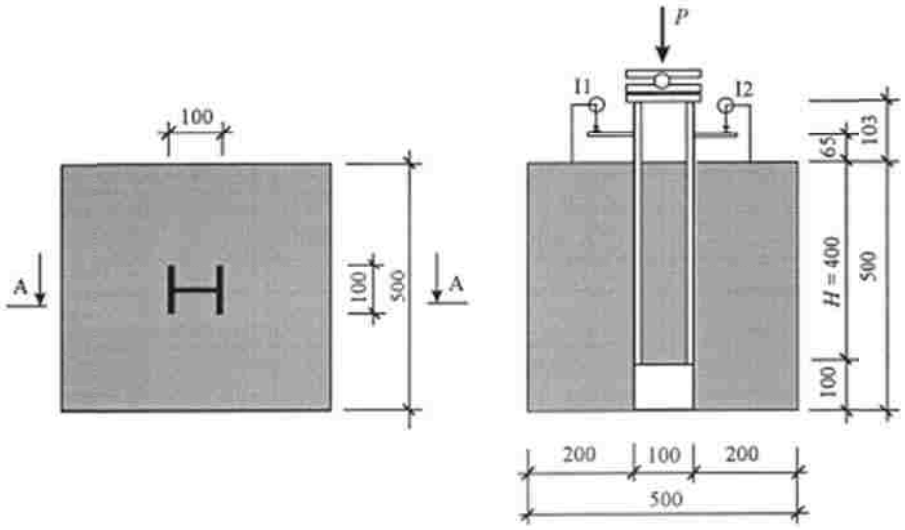


Figure 2-15: Bond Strength Test Setup (Pertold et al., 2000a). Left: Plan View. Right: Section A-A (unit: mm)

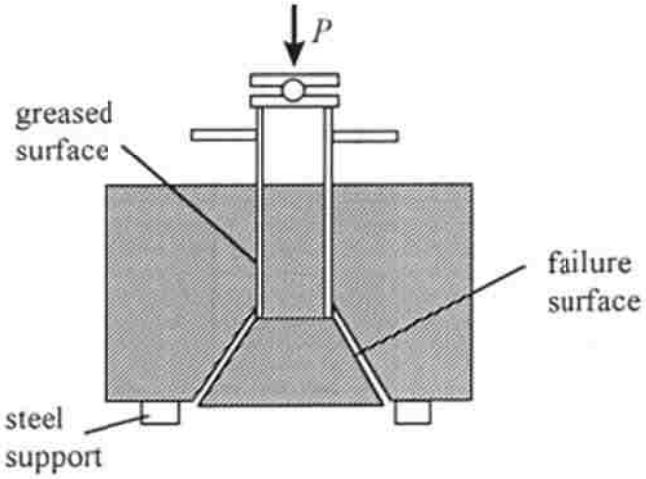


Figure 2-16 - Failure Mode for Bearing Strength Test (Pertold et al., 2000a)

The bond strength between steel and concrete is typically greater than the bearing strength provided by the baseplate. In the cases tested by Pertold et al., the average bond strength was 232 N/mm^2 (33.6 ksi), while the average baseplate bearing strength as 117 N/mm^2 (17.0 ksi). The failure mode for the bond strength tests was bond failure (not pictured). The failure

mode for the tests was concrete punching in the form of a truncated cone, before the design strength of the steel column was reached (see Figure 2-16).

Pertold et al. (2000a) also created numerical (FEM/FEA) models of deeply embedded specimens. The FEM models were in two dimensions, but were calibrated against their test data to ensure accuracy. Once their models were calibrated using vertical loads, the effects of lateral loads were studied by creating finite element models of new loading situations (see Figure 2-17). Both flanges of the steel column were found to be involved in horizontal stress transfer; one flange transfers stress through its exterior face, with the opposite flange transferring it through its interior face. The horizontal stress in the concrete near both flanges reached an average of 67% of the maximum concrete stress before final failure. The stress was also found to decrease significantly in the case of excessive embedment length.

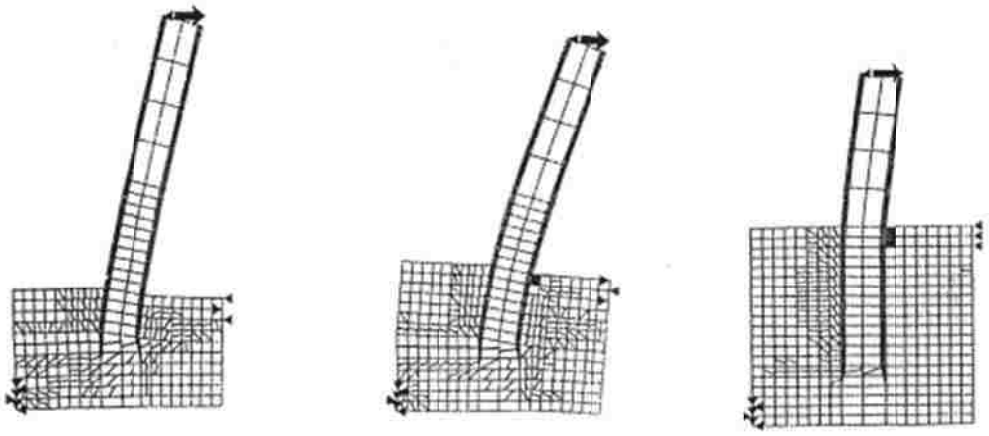


Figure 2-17: FEM Model of Embedded Column Base Subjected to Moment and Shear Force (Pertold et al., 2000a)

Based on their physical testing and numerical simulations, Pertold et al. recommended a design basis for embedded steel columns (2000b). The failure modes considered include collapse

along the perimeter of the beam; a combination of bond failure along column flanges and shear failure of the concrete filling; and assorted punching failures. These design recommendations have only been validated for depths of greater than 100 mm (3.9 in).

2.5 **Shallowly Embedded Baseplate Connections**

Cui et al. (2009) performed experimental testing on eight specimens of shallowly embedded connections. Significant strength, stiffness, and energy dissipation were obtained. The precise levels of these quantities could be varied with adjustments to the floor slab thickness, shape and reinforcing bars in the slab. Figure 2-18 shows the typical test specimen layout tested by Cui et al.

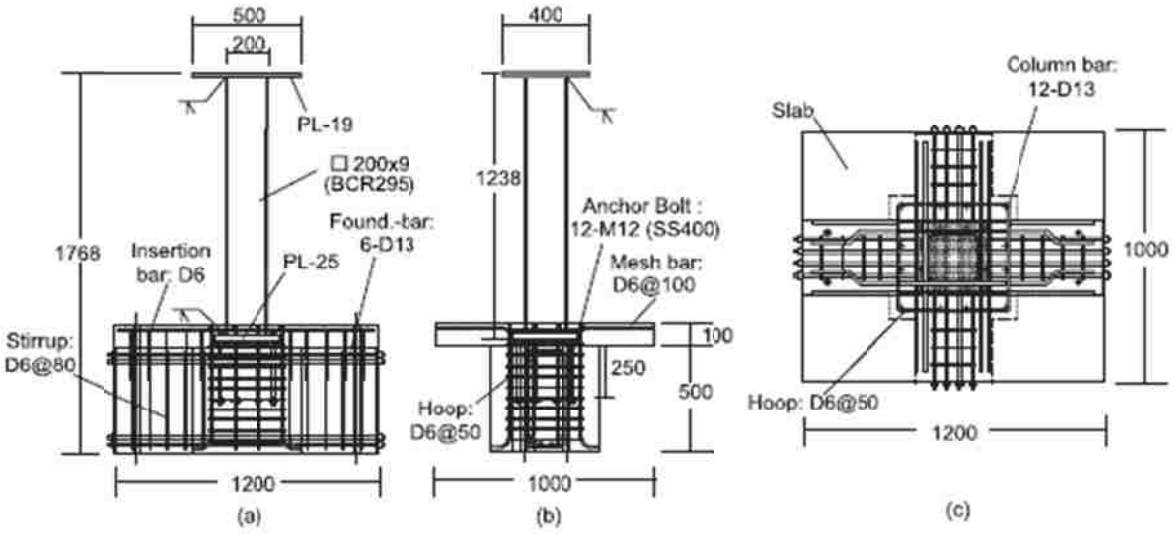


Figure 2-18: Test Specimen a) Front Elevation ; b) Side Elevation; c) Plan View (unit: mm) (Cui et al., 2009)

The elastic stiffness with embedment depths of 100 and 200 mm (3.9 and 7.8 inches) was increased by 1.1 and 1.5 times respectively, over that of an exposed type connection. It was also

found that by doubling the thickness of the floor slab (embedment depth), maximum strength and energy dissipation were increased by 1.5, and 1.9 times, respectively. The presence of horizontal rebar beneath the baseplate did not improve elastic stiffness.

Although the connections studied were shallowly embedded connections, their details varied markedly from typical American construction practice. Specifically, they had reinforcing bar in the floor slab above throughout the embedment depth (see Figure 2-19), which is typical in Japanese construction practice. It is not currently well-understood how the absence of rebar around the baseplate (typical in American details) affects the strength and stiffness of the connection. Also, the baseplates were secured with 12 anchor bolts, while typical American construction practice rarely has more than four anchor bolts for gravity columns.

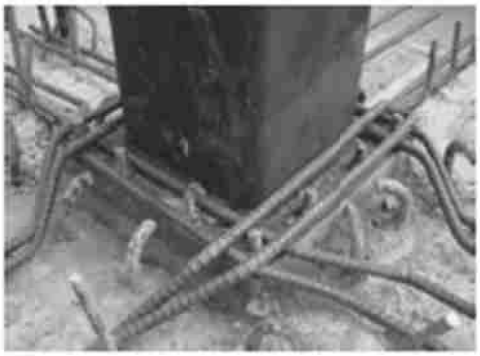


Figure 2-19: Typical Arrangement of Reinforcing Bars (Cui et al., 2009)

2.6 FEM Modeling of Baseplate Connections

Khodaie et al. (2012) used finite element modeling to explore the initial stiffnesses of Square Hollow Section (SHS) shapes, used for Bolted Column Based Plate (BCBP) or exposed baseplate connections. Figure 2-20 shows a detail view of the models studied. 8-noded

hexahedral solid elements were used to model the column, base plate, anchor bolts, and foundation; for welded surfaces, 4-noded tetrahedral solid isoparametric elements were used with finer meshes. After calibrating their models with the results from experimental models by Picard et al. (1984) and Wheeler et al (1998), they extended their results to additional geometries.

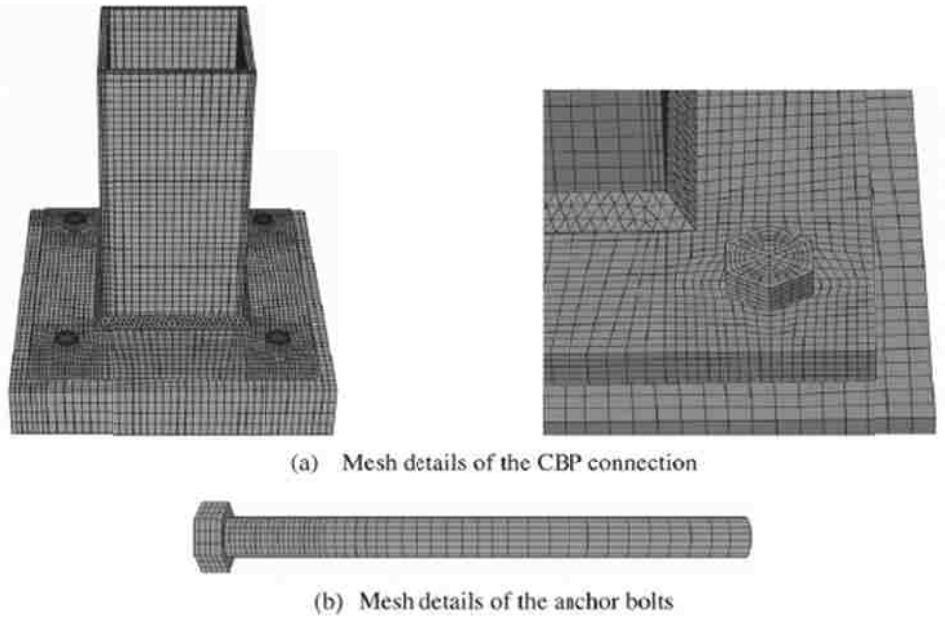


Figure 2-20: Details of FEA Connection Model (Khodiae et al., 2011)

Khodiae et al. used a regression analysis model to determine the relative contributions of various elements to the initial stiffness response of the entire connection. Many elements of the connection were considered: column depth; baseplate thickness; column web thickness; distance from bolt center to fillet weld; distance from bolt center to baseplate edge; anchor bolt diameter; and lever arm. Each of these contributions was determined, and multiplied to find a final stiffness value.

Jordan (2010) created and analyzed two sets of finite element models of exposed baseplate connections. The first set was a set of models of Gomez' (2010) seven tests. The

second set was a set of untested connection configurations to generalize the results from the first set. Due to the symmetry of the experimental specimens, the models were constructed as half-models with appropriate constraints along the plane of symmetry. The foundations of the base connections were fixed. Part instances with relatively complex geometries (such as anchor rods) were assigned a tetrahedral mesh with a C3D4 element type; parts with relatively simple geometries (base plate, column, etc.) were assigned hexahedral element shapes with a C3D8 element type. Figure 2-21 shows a typical model of an exposed baseplate connection, and Figure 2-22 shows a typical deformed base connection after loading in the FEA program.

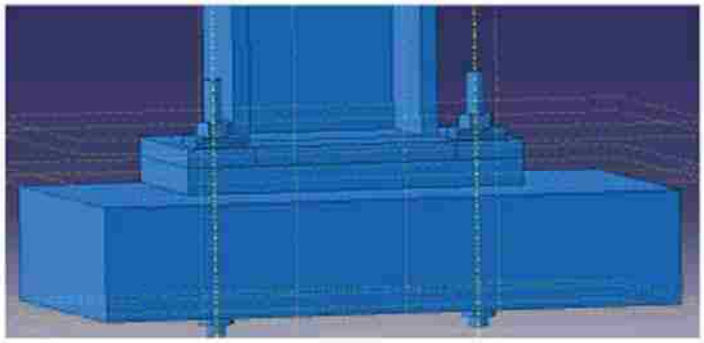


Figure 2-21: FEM Model of Exposed Baseplate Connection (Jordan, 2010)

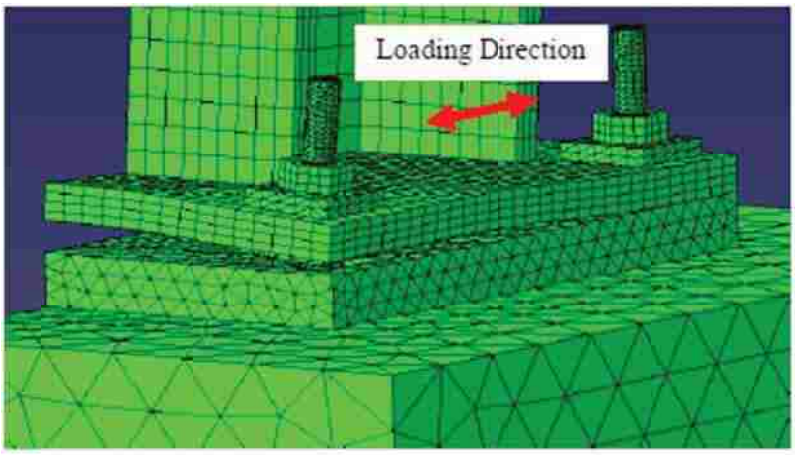


Figure 2-22: Typical Deformed Base Connection (Jordan, 2010)

The outputs from the FEA model were compared with the physical test results. The values that were correlated were: load-displacement; anchor rod force-drift and strain-drift responses; peak loads; and peak base moments. In all cases, the correlation was at least within the same order of magnitude as the test results, and in many cases, was able to match the experimental data almost exactly. For example, the test-to-predicted ratio for peak loads in the seven cases ranged from 0.83 to 1.09. Also, the deformed baseplate geometry was compared qualitatively with Gomez' specimens, and relatively close agreement was found (see Figure 2-23). Information about bearing stress distributions were also obtained, which was unavailable during the experimental investigations. These data suggest that design assumptions of a rectangular stress block beneath the baseplate may be inaccurate.

An additional three tests were performed using only finite element models, which expanded and generalized the findings available from Gomez' laboratory tests. In the first test, loading orientation was set to 45 degrees, diagonal to the column orientation. In the second test, 8 anchor rods (as opposed to 4) were present on the test specimen. In the third, anchor rod arrangement was investigated; the rods were arranged in parallel with the column flanges, instead of perpendicular to them. As in the first set of models, the response variables investigated included load-displacement, anchor rod force-drift and strain-drift responses, and peak loads. Yield line patterns and bearing stress distributions were also studied.

The initial stiffness ratios from the finite element results were between 0.75 and 0.94 times the recorded values in the laboratory (see Table 2-3). The strain hardening values ranged between 0.94 and 3.15 times the values from the experimental data.

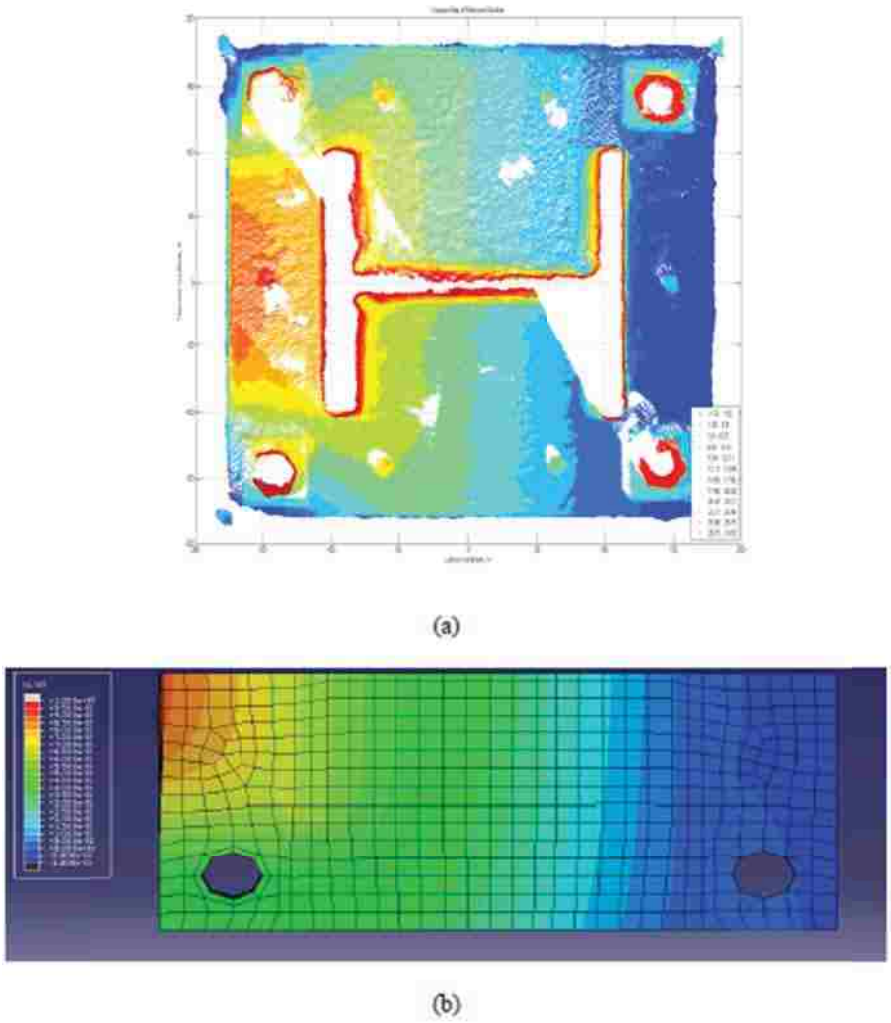


Figure 2-23: Comparison between (a) 3D Scan Contour Plot and (b) Simulation Contour Plot of Typical Post-Test Basplate (Jordan, 2010)

Table 2-3: Test-to-Predicted Ratios from Load-Displacement Response (Jordan, 2010)

		Test-to-Predicted[1] Ratios	
Test	Analysis	Stiffness	Hardening
1	1	0.75	1.67
2	-	0.8[2]	0.94[2]
3	2	0.83	1.57
4	3	0.97	3.15
5	4	0.93	1.67
6	5	0.94	2.71
7	6	0.94	2.25

[1] From FEM Simulations
[2] With respect to Analysis #1

2.7 Cohesive Zone Modeling

A specific class of finite elements, known as "cohesive zone elements" is commonly used to model crack growth in concrete. These elements define a constitutive relationship between the surface tractions and a corresponding separation. After crack initiation, the traction-separation value decreases until it reaches zero and the element disappears. Hillerborg et al. (1976) introduced fracture energy into the cohesive crack model already in use, and proposed a traction-separation relationship (E/f_t) of concrete of 10,000. Cohesive zone modeling is often used in concrete modeling, although it is typically not used in the interface with the steel rebar (Sosa 2010). Serpieri et al. (2014) cites examples of CZM being used for a large variety of problems at different scales, including crack growth in concrete dams, mortar-joint failure in brick masonry, bond slip of ribbed reinforcing bar in concrete, debonding of adhesive joints, delamination in composites, and more.

Song et al. (2006) modeled crack propagation through bituminous (asphalt) concrete using cohesive zone modeling. They used a user-specified element (UEL) subroutine in the Abaqus

software, which incorporated a fracture-energy based model. This subroutine was verified by simulation of a double cantilever beam test and comparing the results to closed-form solutions.

Figure 2-24 shows a comparison between the analytical solution and numerical results in a double cantilever beam (DCB) debonding problem. Closed-form solutions are available for this test, and the numerical results are overlaid on the analytical solution.

The material strength and cohesive fracture energy parameters were calibrated based on a single-edge notched beam (SE(B)) test of the experimental specimen. The calibration coefficients were 0.7 and 1.1 for cohesive fracture energy and material strength, respectively. It was shown that the simulated crack growth patterns closely matched the experimental results. A Riks numerical solution method was used. Nonconvergence occurred when the crack tip reached about 40% of the height of the SE(B) test specimen. Of the three mesh sizes investigated (0.1, 0.2 and 1.0 mm), umerical solutions were insensitive to the mesh size used.

Julander (2009) compared finite element models to experimental results for transverse bridge deck connections. Both shear-key specimens and flexural specimens were constructed, tested, and modeled. Four different shear connection specimens were tested and modeled: unreinforced shear key; welded stud shear key; non-post tensioned shear key; and post-tensioned shear keyFive different flexural connection types were modeled: post-tensioned, welded rebar, welded stud, 36” curved bolt, and 24” curved bolt. Figure 2-25 and Figure 2-26 illustrate the schematic test setups and FEA models for shear tests and flexure tests, respectively.

The interface between concrete and grout was modeled with Cohesive Zone Model (CZM) elements. The element behavior was bilinear, with both traction and maximum separation values defined.

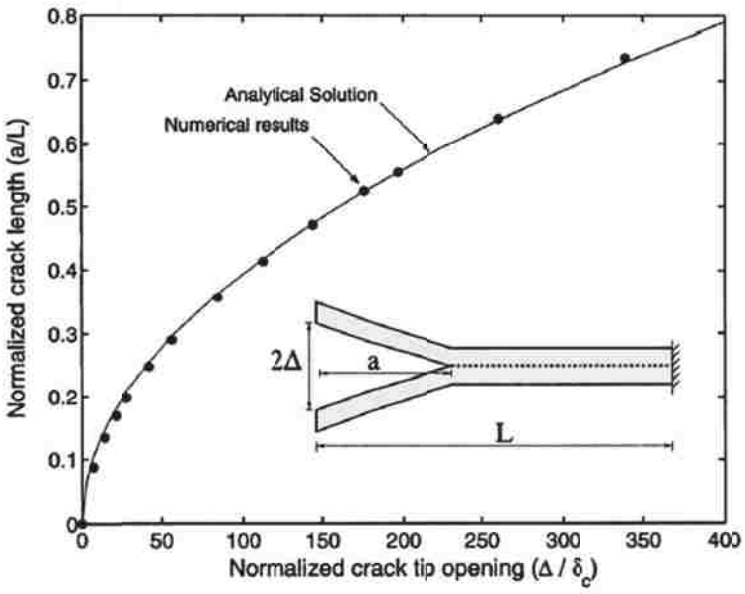
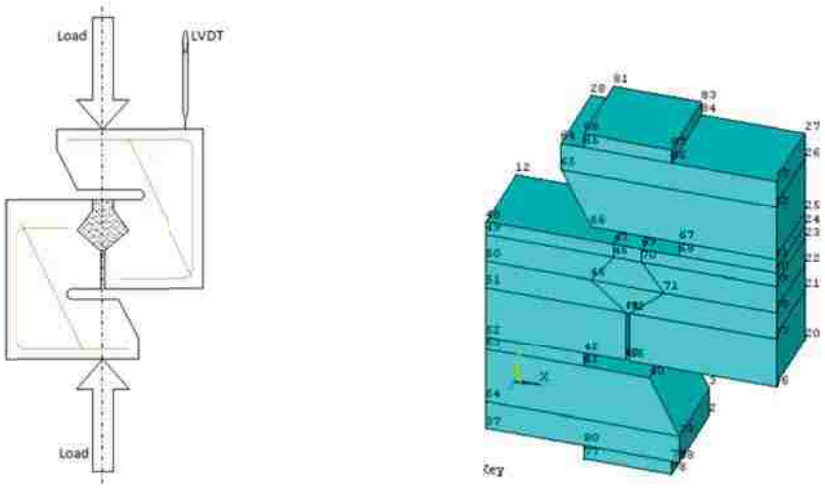
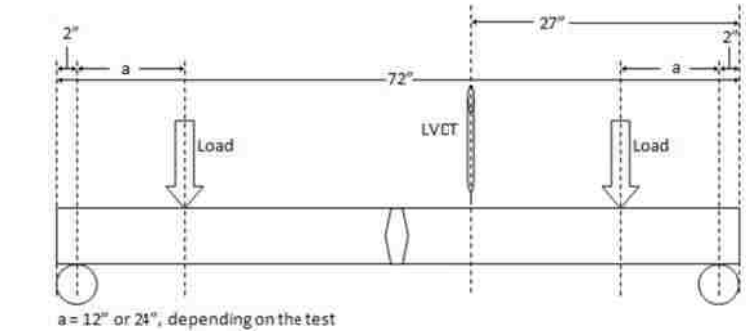


Figure 2-24: Comparison between Numerical and Analytical Solutions (Song et al., 2006)

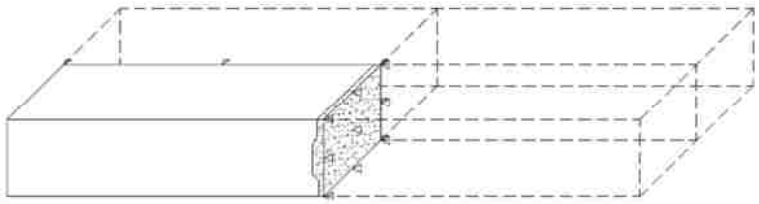


(a) (b)

Figure 2-25: Shear Test Setup: (a) Schematic Illustration; (b) Ansys FEA model (Julander, 2009)



(a)



(b)

Figure 2-26: Flexure Test Setup: (a) Schematic Illustration; (b) Ansys FEA model (Julander, 2009)

Two different contact types between the concrete and the grout were attempted. The first was a perfect bond, and the second was a contact-based formulation. The perfect bond connection gave results that were approximately an order of magnitude stiffer than the experimental results. The contact-based formulation gave results that were approximately 21x less stiff, and significantly closer to the experimental test results.

Relatively close agreement was obtained between the FEA results and the test configurations. Figure 2-27 shows a representative force-deflection graph of the shear tests. In this case, it is of the unreinforced key series of experiments. After initial separation between concrete and grout in the keyway, deflection continued to increase linearly until ultimate loading. Figure 2-28 shows a representative moment-deflection graph of the flexural tests. In the case of

flexural connections, the FEA curves followed the tested curves almost exactly in the linear range prior to cracking, and relatively close to many of the curves in the non-linear, cracked section of the load-deflection curve.

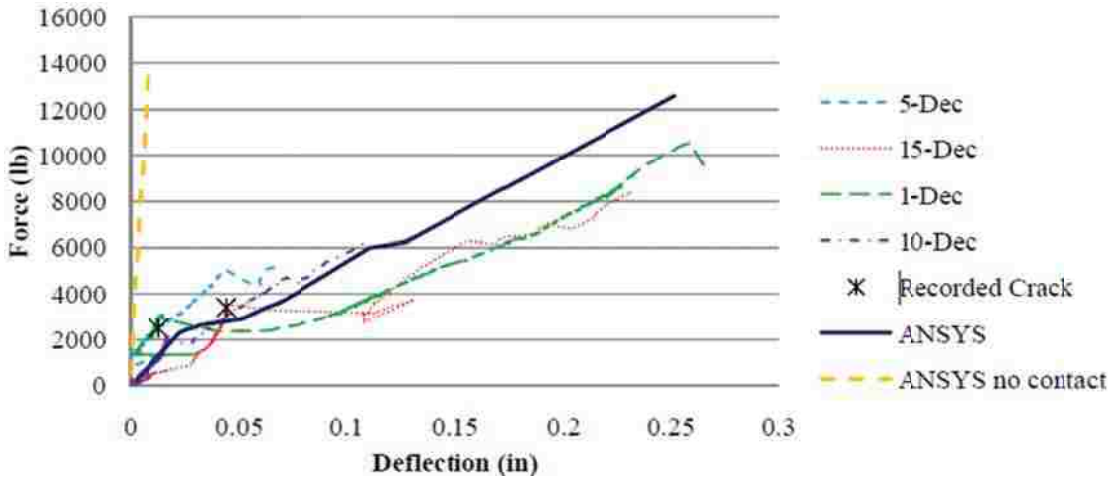


Figure 2-27: Unreinforced Key Shear Specimens; Force-Deflection Curve (Julander, 2009)

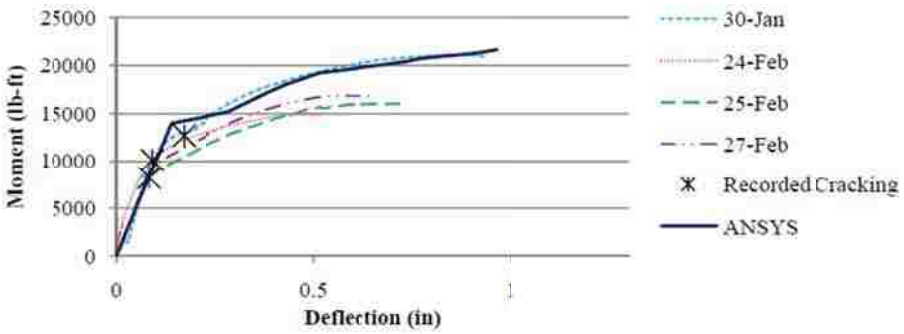


Figure 2-28: Post-Tensioned Flexural Specimen; Moment-Deflection Curves (Julander, 2009)

Serpieri et al. (2014) used cohesive-zone modeling to model bond slip in a concrete-rebar situation. The cohesive zone elements simulated a combination of adhesion loss, friction along

flat surfaces, and mechanical interlocking. A multiscale formulation was used, with a macro-scale and micro-scale coupled. The Representative Interface Area (RIA) is a microscale problem which allows the relative displacement vector s to be related to the interface stress σ for use on the macroscale. An infinitesimal area dA of microplane can be represented as having both damaged and undamaged sections – see Figure 2-29.

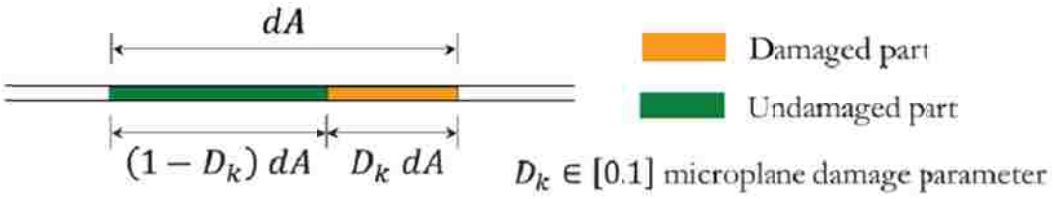


Figure 2-29: Decomposition of each Infinitesimal Area of Microplane into Damaged and Undamaged Parts. (Serpieri et al., 2014)

With the separation into damaged and undamaged sections, expressions for free energy, interface stress σ, and frictional slip on each microplane, are derived. These variables are in turn used to calculate further evolution of the damage variable D_k.

This numerical model was implemented in a user-subroutine in Abaqus. A finite element model was created of earlier experimental testing of rebar pullout, performed by Shima et al. The diameter of the steel bar was 19.5 mm, and a 195 mm long unbounded region in the vicinity of the loaded bar was present. Figure 2-30 illustrates both the experimental test setup, and the details of the geometry and FEM mesh.

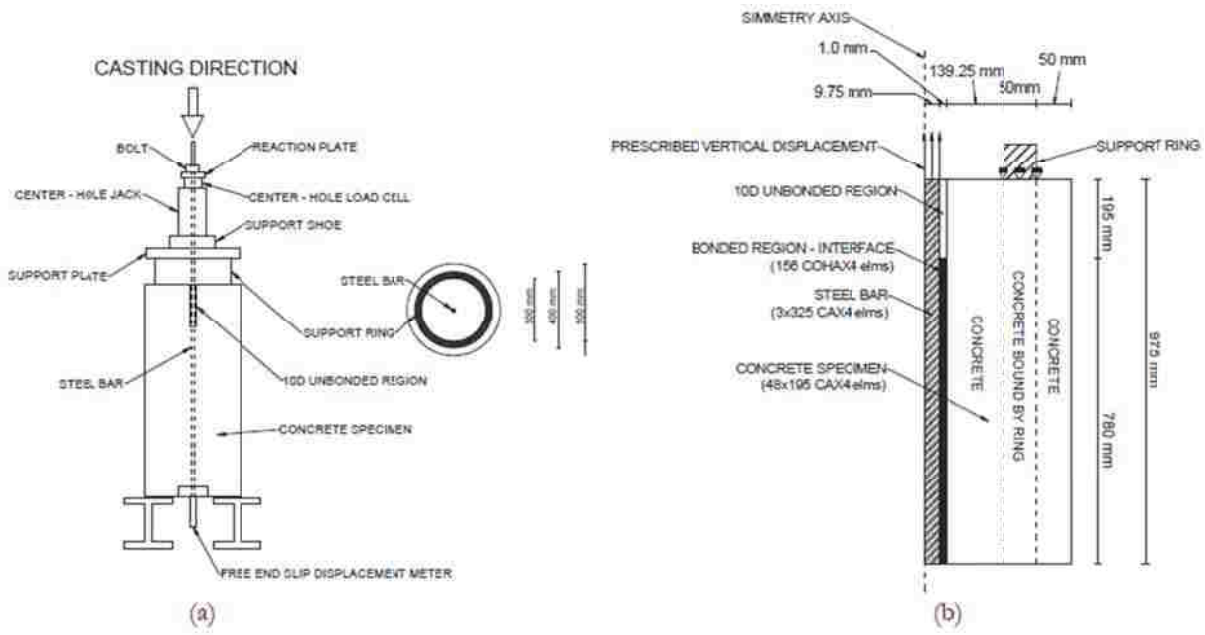


Figure 2-30: Pull-out Test: (a) Experimental Setup, and (b) Details of the Geometry and FEM Mesh. (Serpieri et al., 2014)

Test results are presented in Figure 2-31. Results were validated with earlier experimental results. Accurate predictions were achieved using a simpler model than previous studies, which had typically depended on also on a large number of empirically-determined corrective coefficients.

Sosa (2010) presented a numerical method for modeling reinforced concrete, taking into account both nonlinearities in the concrete as well as debonding in the interface. Comparisons to pull-out problems were performed in order to validate the methodology.

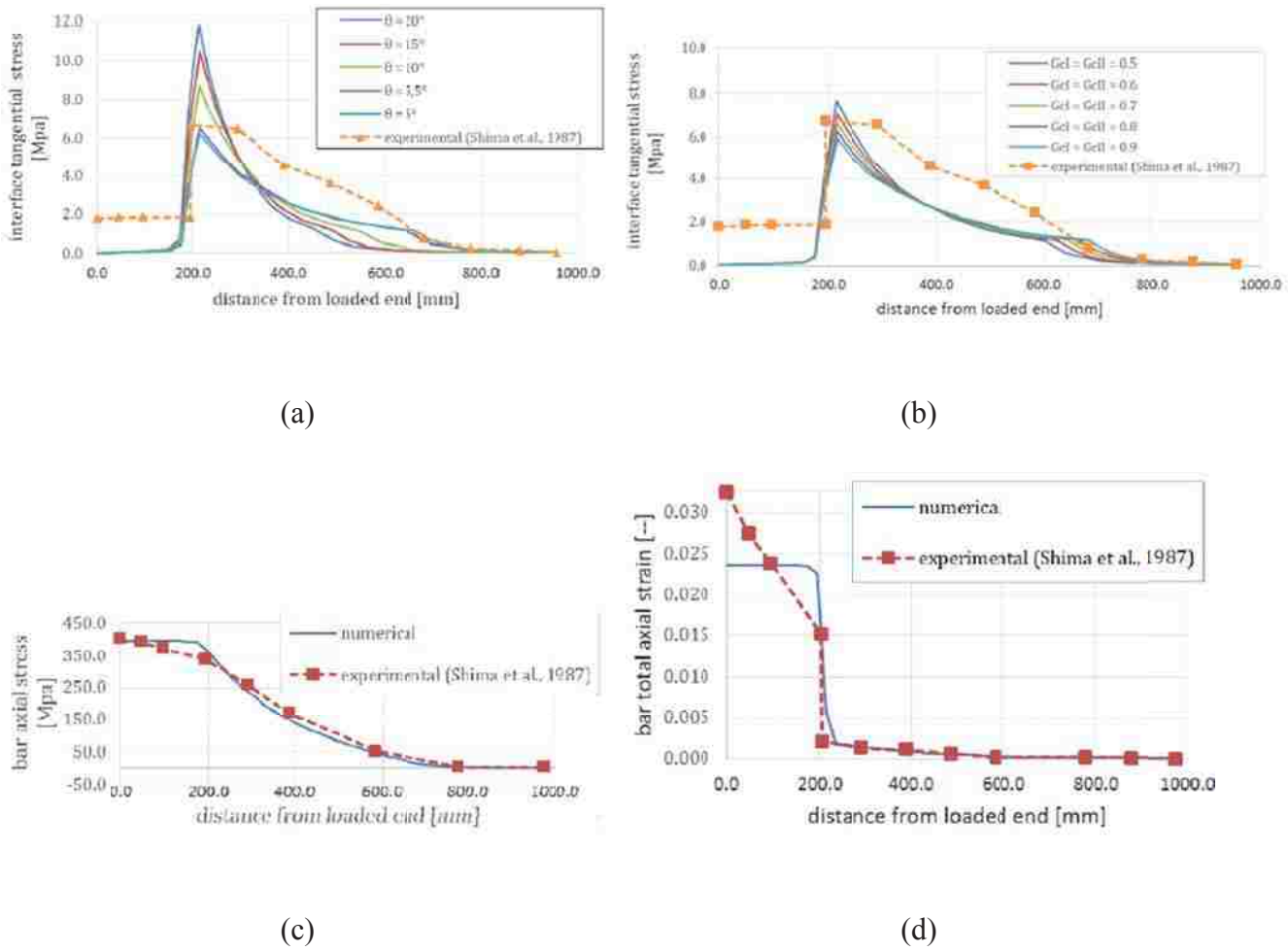


Figure 2-31: Comparison of Simulation with Experimental Results for: (a) Interface Tangential Stress at Varying Microplane Inclinations θ; (b) Interface Tangential Stress with Varying Fracture Energy G_{cl}; (c) Vertical Direct Stress in the Bar; (d) Vertical Direct Strain in the Bar (Serpieri et al., 2014)

Sosa (2010) assumed an imperfect bond, although it was mentioned that a perfect bond is assumed by a number of other researchers, to avoid modeling difficulties arising from nonlinear qualities of concrete. An implicit scheme was adopted for displacement solutions in the concrete, while an explicit scheme was adopted for rebar. The typical finite element methodology was employed, with terms which accounted for the interaction of the rebar and concrete elements. A return-mapping algorithm was used, which consists of an elastic trial prediction, with a plastic

correction applied if yielding is reached. An interface constitutive law was created, to calculate the element interface forces from the bond stress and the surface area of the rebar elements. The rebar is modeled with continuum-based beam elements, and an explicit solution procedure is used, based on standard central difference formulations.

Experimental results from pullout tests performed by others were found to be in good agreement with the results from usage of these formulations. Figure 2-32 shows a comparison between the experimental bond stress and several proposed techniques; the "elastic pattern" referenced refers to an elastic interface law assumption, and is provided for comparison with the more realistic proposed technique. The applied external force was 100 kN.

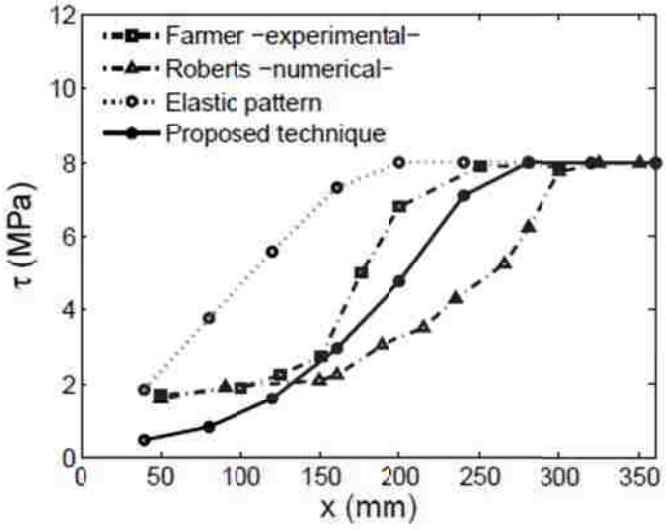


Figure 2-32: Numerical Versus Experimental Bond Stress along Reinforcement (Sosa, 2010)

2.8 **Design Provisions and Design Guides**

Current specifications and guidelines for current connection design are found in *Base Plate and Anchor Rod Design* (Fisher and Kloiber, 2006). Detailed baseplate and anchor rod

specifications, column erection procedures, and grouting requirements are included. Strength considerations for several different design load cases are also included. No design guidelines for characterizing stiffness are included.

Recent research done at UC Davis has characterized the stiffness and strength of exposed-type connections (see Gomez 2009 and Kanvinde et al. 2012). This has led to the inclusion of design examples in relevant manuals.

2.9 **Coefficient of Friction**

The coefficient of static friction between concrete or grout and steel was a necessary input parameter in some of the finite element models (see Section 3.1.4). Since this value was not determined during the parent study, an attempt to determine it by consulting the literature was made.

The given values in the literature varied widely. Rabbat and Russell (1985) conducted a series of fifteen (15) push-off tests, and determined the coefficient of static friction between rolled steel plate and cast-in-place concrete or grout. With a wet interface, under normal compressive stress levels, the determined coefficient of static friction was 0.65. With a dry interface, it was determined to be 0.57. Baltay and Gjelsvik (1990) determine the coefficient of static friction to be equal to 0.47.

Gomez et al. (2010) reported 0.25 as the coefficient of static friction they obtained in their tests. However, they cited ACI 349-06 in regards to shear strength, which allows for a μ = 0.9 for baseplates without shear lugs, and μ = 1.4 for baseplates with shear lugs that are designed to remain elastic. It is noted that the ACI 349-06 requirements are based on earlier testing, in which the shear strength was derived as the sum of bearing strength and strength due to

confinement. Elsewhere in Gomez et al. (2010), it is noted that ACI 349-06 (Section D.6.1.4) stipulates 0.40 as the coefficient of friction between a steel baseplate and hardened concrete.

2.10 **Composite Beam-Column Connections**

Steel I-beams bearing against reinforced concrete, with associated web shear, is the force transfer mechanism expected for shallowly embedded connections. Design philosophy for this mechanism is well documented in the ASCE guidelines for composite beam-column connections (ASCE Guidelines 1994) and associated research (Sheikh et al 1989, Deierlein et al 1989).

In ASCE Guidelines 1994, design guidelines for composite joints are presented. Composite joints are defined as reinforced concrete columns that are connected to continuous structural steel beams. These guidelines account for panel shear and vertical bearing failure modes (see Figure 2-33).

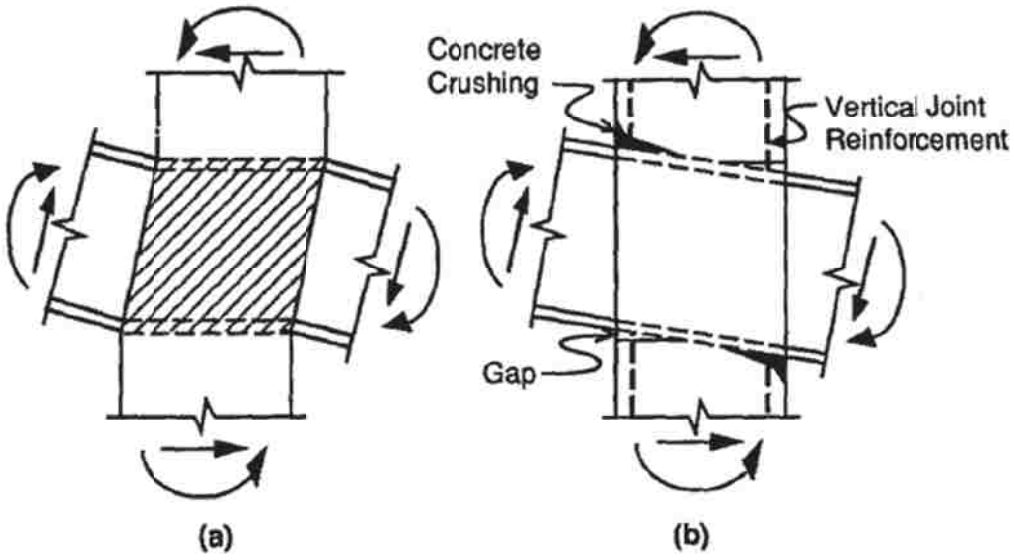


Figure 2-33: Joint Failure Modes: (a) Panel Shear; and (b) Vertical Bearing (ASCE, 1994)

Design equations are presented for the two failure modes, derived from research by Dierlein et al (1989) and Sheikh et al (1989). In Sheikh et al, an experimental test setup of 15 specimens was conducted and the results were analyzed. In Dierlein et al., these test results were summarized, and design equations were proposed, which were later incorporated into the ASCE guidelines (1994). These results were later validated and expanded by Cordova and Deierlein (2005).

Although a detailed understanding of strength considerations is demonstrated in these papers, stiffness considerations receive minimal treatment. Sheikh et al. (1989) report the strain levels observed in their tests, but make no effort to categorically evaluate the stiffness of the studied connections. The ASCE guidelines simply state that "deformations in the joint region should be considered in evaluation of deflections under service and factored loads," and offers no additional guidance. The design commentary only notes that commercially available frame analysis programs do not explicitly account for joint deformations, but can consider the joint deformations implicitly.

2.11 **Field Reconnaissance of Earthquake Damage**

Tremblay et al (1995) performed reconnaissance of steel buildings after the 1994 Northridge earthquake. The buildings that were inspected were concentrically braced frames, moment resisting frames, or a combination of the two. Several failure modes were observed, including baseplate fracture, anchor bolt failure, and/or brittle failure of welds connecting the baseplates and gusset plates.

The rotational restraint of shallowly embedded connections can be significant when properly designed. In one instance observed, complete collapse of several bays was prevented only by the rotational restraint offered by embedded column baseplates.

2.12 Parent Study

The primary finite element model geometries which were investigated for this thesis were based on a parent study performed by Barnwell (2015). These tests have shown that there is significant strength and stiffness available in shallowly embedded connections. Yield and ultimate strength data were obtained from all tests. In addition, usable stiffness data were obtained for 7 of the 12 specimens. These tests constitute a parent study for this work, as they will form the basis of comparison and validation for the finite element models described in Section 3. This section will explain Barnwell's setup, results, and conclusions.

2.12.1 Test Setup

Barnwell conducted laboratory tests of twelve different specimens of shallowly embedded connections to determine strength and stiffness values. The testing investigated the effects of four variables on connection strength and stiffness: embedment depth, column shape, column orientation, and presence/absence of engaged anchor bolts. Table 2-4 summarizes the differences in the various specimens.

The test setup consisted of a frame, and actuator, and a test specimen. The test specimen itself consisted of a steel column with attached baseplate, slab-on-grade concrete, footing concrete, and block-out concrete. The steel column was either a W8X35 or a W8X48 specimen, and the baseplate was a PL 1"x13"x1'1", fillet welded to the column. Figure 2-34 shows a

schematic diagram of the test setup, while Figure 2-35 shows a typical specimen. The specimen was fastened to the lab's strong floor through six 1 ¼" rods that were post-tensioned to 50 kips each. The setup was designed to closely simulate common construction practices in the United States for shallowly embedded column connections.

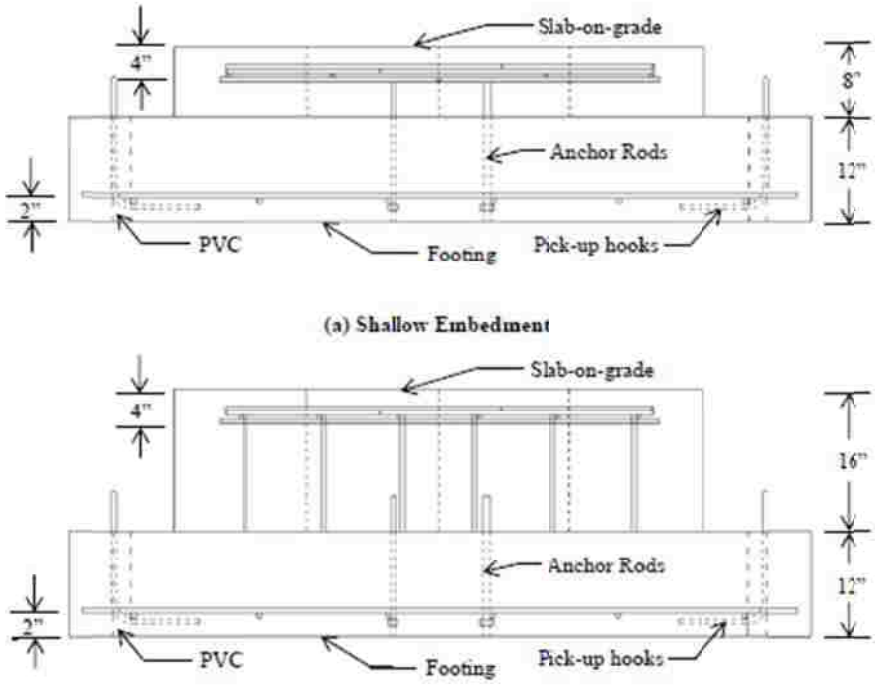


Figure 2-34: Concrete Elevations (Barnwell, 2015)

Within the concrete specimen, a blockout area was created. The depth of the blockout was either 8 or 16 inches, depending on the specimen. The column rested on leveling nuts that had a 1.5" nominal depth from the base of the blockout. A high strength, non-shrink grout was placed beneath the column to fill the void between the bottom of the blockout and the leveling nuts. It should be noted that, although the blockout depth was 8" or 16", the embedded column depth was 5.5" or 13.5", due to the reduction in depth to allow for the baseplate (1") and grout (1.5"). When the grout had set, the rest of the blockout area was filled in with concrete.

Figure 2-35: Typical Test Setup (Barnwell, 2015)

Table 2-4: Summary of Test Parameters (Barnwell, 2015)

Test Label	Embedment	Orientation	Column Shape	Anchor Bolts Engaged?	Braced Slab
A1	8	Strong	W8X35	Yes	--
A2	8	Strong	W8X48	Yes	--
A3	8	Weak	W8X35	Yes	--
A4	8	Weak	W8X48	Yes	--
B1	16	Strong	W8X35	Yes	--
B2	16	Strong	W8X48	Yes	--
B3	16	Weak	W8X35	Yes	--
B4	16	Weak	W8X48	Yes	--
CA2	8	Strong	W8X48	No	No
DA2	8	Strong	W8X48	No	Yes
CB2	16	Strong	W8X48	No	No
DB2	16	Strong	W8X48	No	Yes

The specimens were designed in accordance with AISC Baseplate and Anchor Rod Design handbook (Fisher and Kloiber 2006), using a "small moment" approach. The applied design moment was considered small enough that it would not cause a tendency to overturn. This implies that the anchor rods were not designed to withstand tension loads. This represents typical design practice for gravity-bearing columns in the United States. The columns were designed as two-thirds scale models of actual column sizes.

Determining the cantilevering length of the columns – the distance between the line of action of the force of the actuator, and the top of the concrete – was important in the later finite element modeling work. In the case of the shallow (8") embedment specimens, the total length of the column was 7'-8" (92"), excluding the baseplate thickness (1"). Although the nominal embedment was 8 inches, a 1.5" grout pad and a 1" baseplate reduced the effective embedment length to 5.5". Therefore, the column cantilevered 92"-5.5" = 86.5" from the concrete. Additionally, the centerline of the actuator – which represents the line of action of the applied force – was 6.25 inches from the top of the column. Therefore, the total cantilever distance was 80.25". The deep (16") embedment specimens had total column lengths of 8'-7" (103"). Using similar reasoning, the deeply embedded columns had a protruding length of 83.25" (103" – 13.5" – 6.25"). Table 2-5 summarizes these calculations.

Table 2-5: Cantilever Height Calculations

	Starting Column Length	Embedded Column Length	Actuator centerline to slab on grade:	Cantilever Length:
Shallow	92	5.5	6.25	80.25
Deep	103	13.5	6.25	83.25

Digital Image Capture (DIC) instrumentation recorded the displacement fields near the surface of the connection during the testing. The DIC data showed deformations on the exposed surface of the concrete foundation, and in the column. Analysis of the data was not included in Barnwell (2015), but it is analyzed in Chapter 4 of this thesis.

2.12.2 Test Results

Barnwell's research showed that there is substantial strength and stiffness in shallowly embedded connections, which is not accounted for in current design practice. This confirms, for the case of shallowly embedded connections, what was suggested by the work of Richards et al. (2011) and Eastman (2011) in the case of pile cap connections.

The common failure mechanisms in Barnwell's tests were concrete cracking and anchor bolt yielding/fracture. This suggests that force transfer occurred through both the confining concrete, and the baseplate.

Barnwell's test data showed that even shallowly embedded columns at 1x embedment depth showed higher strength than expected. The connections are 86-144% stronger in yielding, and 32-64% stronger in ultimate strength, than is predicted by current design methods. An improved model was proposed that would account for the additional strength in the connection. Instead of assuming that the overturning moments are resisted only by tension and compression only in the baseplate, the model proposes that substantial portions of the concrete slab also resist compressive forces cause by overturning moments. Barnwell's model accurately predicts the strength of the connection to within 18% of tested values.

Barnwell used the same elastic stiffness model used by Eastman (2011), reproduced graphically in Figure 2-36. The stiffness mechanism assumes that the total deflection can be treated as the superposition of two independent deflections The first, Δ_c, represents the deflection of the column itself, neglecting any flexibility of the connection. The second, Δ_{conn}, represents the deflections caused by the deformation and rotation of the connection itself. Both mechanisms have associated lateral stiffness values, which are denoted k_c and k_{conn}, respectively. Therefore,

$$\Delta_t = \Delta_c + \Delta_{conn}$$

$$\frac{V}{k_t} = \frac{V}{k_c} + \frac{V}{k_{conn}}$$

$$\frac{1}{k_t} = \frac{1}{k_c} + \frac{1}{k_{conn}}$$

(Note: this model is mathematically equivalent to that of springs in series.)

$$k_{conn} = ({k_t}^{-1} - {k_c}^{-1})^{-1}$$

Additionally, k_c is equivalent to k for a cantilevered beam with a point load on the end, that is,

$$k_c = \frac{3EI}{L^3}$$

And so

$$k_{conn} = ({k_t}^{-1} - \frac{L^3}{3EI})^{-1}$$

The stiffness results were obtained from initial tangent stiffness values, measured as the load varied between 0 and 1 kip of applied force. Although theoretically the force-displacement response should be basically linear over such a small range, detailed measurements showed that

some nonlinearity was present. Figure 2-37 is a representative example, where k_1 refers to the stiffness between the first and second available actuator readings; k_2 refers to the stiffness between the reading at 1 kip, and the reading before it; and k_3 refers to the secant stiffness between 0 and 1 kip of load. The value of k_3 was used throughout Barnwell's research, as it gave results that were most consistent. It should be noted that the shape of the curve varied significantly between specimens; while some curves were concave downward, some remained fairly flat, and others had double curvature. This may be attributable, at least partially, to the non-uniform sealing or widening of microcracks or construction joints as the load is first applied.

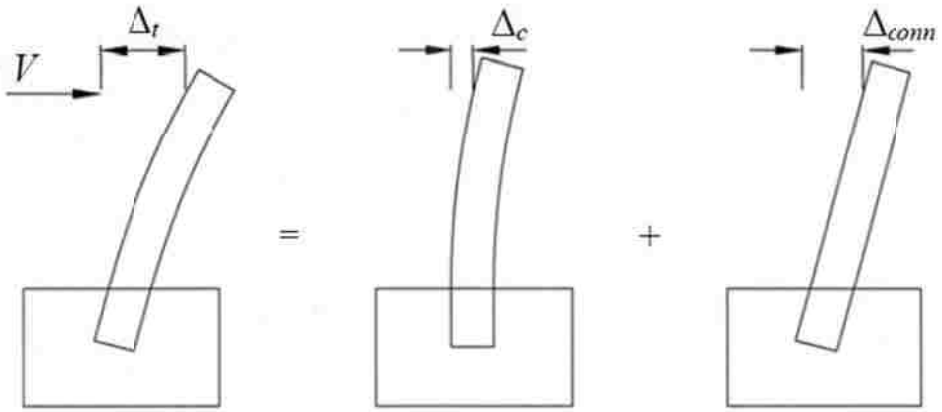


Figure 2-36: Stiffness Mechanism (Barnwell, 2015)

Figure 2-38 and Figure 2-39 show the available stiffness results from Barnwell's tests. Figure 2-38 shows the available results for all tested W8x35 shapes, and Figure 2-39 shows the results for all tested W8X48 shapes. Data for four tests (A4, B4, DA2, and DB2) were unavailable, because the bond between the slab and the footing had broken from earlier tests; this changed the initial stiffness values significantly and they could not be compared to the other specimens.

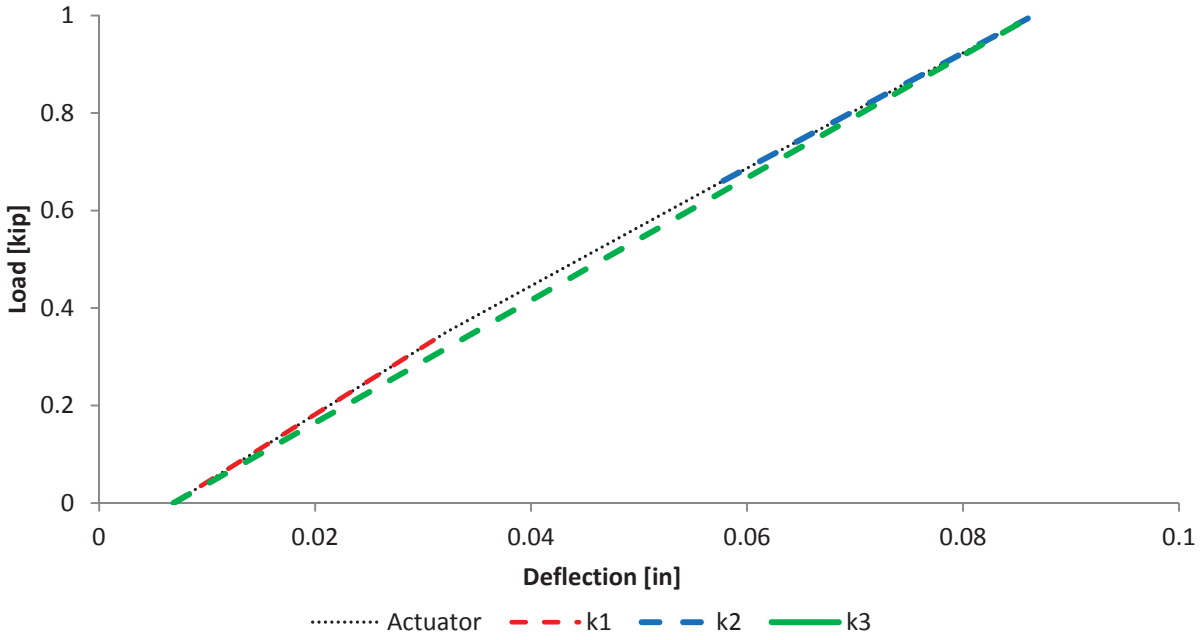


Figure 2-37: Example of Total Stiffness Calculation (Barnwell, 2015)

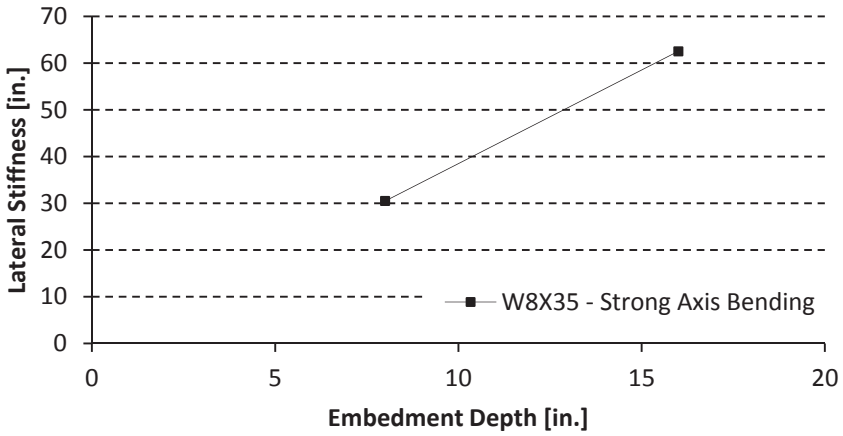


Figure 2-38: W8X35 Lateral Stiffness Values (Data from Barnwell, 2015)

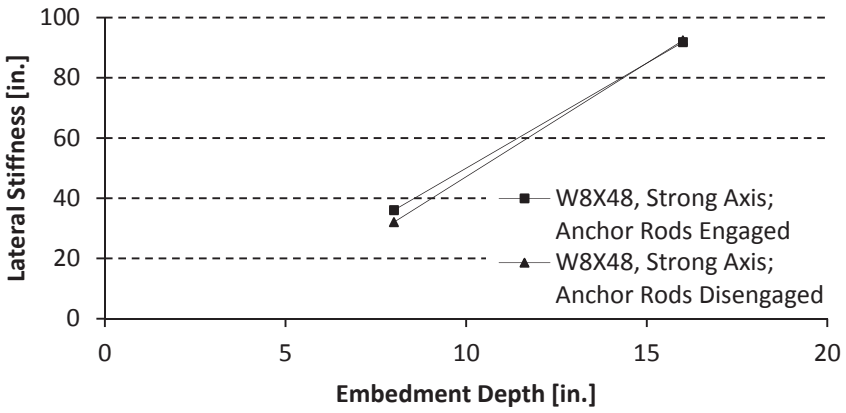


Figure 2-39: W8x48 Lateral Stiffness Values (Data from Barnwell, 2015)

Figure 2-40 shows stiffness results for W8x35 shapes, with weak axis bending. In the shallow embedment specimen, a control error caused loading that deviated from the normal loading protocol in the shallow embedment specimen. The data sampling rate was low relative to the unexpectedly high displacement rate, so the result is considered likely to be unreliable.

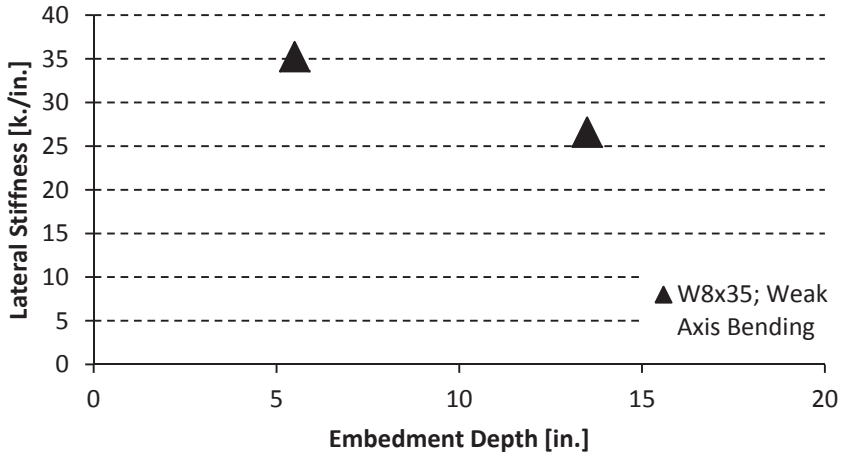


Figure 2-40: W8x35 Lateral Stiffness Values (Weak Axis Bending) (Data from Barnwell, 2015)

Overall column stiffness was between 54% and 83% the total stiffness expected from theoretically fixed connections (see Table 2-6). Barnwell did not propose a model to explain or characterize the observed stiffness values. However, the stiffness increases were greatest in models with larger column shapes; with greater embedment depths; and with the load applied parallel to the column's web (strong-axis orientation). Interestingly, engaging (or failing to engage) the anchor bolts had no significant effect on the initial elastic stiffness of the connection (see Figure 2-39).

Table 2-6: Calculated Stiffnesses Based on k3 (Barnwell, 2015)

Specimen	k_t	k_c	k_{conn}	k_t/k_c
A1	12.57	21.38	30.49	0.59
A2	16.65	30.97	36.02	0.54
A3	5.96	7.17	35.24	0.83
CA2	15.75	30.97	32.04	0.51
B1	14.66	19.15	62.52	0.77
B2	21.31	27.74	91.82	0.77
B3	5.17	6.42	26.60	0.81
CB2	23.20	30.97	92.42	0.75

3 METHODS OF INVESTIGATION

Section 3.1 explains the method of generating and analyzing a single, typical model. Section 3.3 explains how scripts were generated to automate the generation of multiple models, enabling parametric studies. Section 3.5 outlines the limitations and assumptions inherent in the model. Section 3.4 summarizes the models created for this research and the variation between them.

3.1 Finite Element Models

All finite element modeling (FEM) was performed in Abaqus 6.14. Model generation was performed in Abaqus/CAE. Two parts were created, meshed, and assigned material properties. Each part was instanced and positioned in the assembly, and constraints, contact properties, and boundary conditions were applied. A load step was created, a static load was applied, and a field history response request was created. Then, a job was created and submitted to Abaqus/Standard for processing. After processing, the displacement at the point of applied load was queried, and the connection stiffness was calculated.

3.1.1 Model Geometry

A 3-dimensional part was created for the column. The column's cross sectional profile was sketched and extruded, with beam depth (db), flange length (bf), flange thickness (tf), and

web thickness (tw) values obtained from the AISC Steel Construction Manual. The part was created as a half model, with symmetry constraints later imposed across the yz-plane (see section 3.1.4). However, since the plane of symmetry differed depending on the loading direction, separate models were created for strong-axis and weak-axis bending specimens. Figure 3-1 illustrates both strong- (a) and weak-axis (b) sketches. The sketch was extruded to the given cantilever length (pL), and a rectangular baseplate of appropriate dimensions was extruded at the bottom of the column. To facilitate automatic meshing later, the part was partitioned into wholly rectangular regions, which Abaqus' meshing algorithms could handle easily and uniformly. This was done by creating cut planes from existing planes on the part. Also, a partition was created at the point corresponding to the top of the slab on grade. See Figure 3-3 for an illustration of partition locations.

Several simplifications were made to the column and baseplate. Fillets were excluded for simplicity, and because preliminary analysis suggested that their absence would have negligible effects on overall connection stiffness. The anchor rods, anchor bolts, and anchor holes were also excluded because the physical specimens with anchor bolts engaged had nearly identical stiffness values as those with anchor bolts disengaged (Barnwell, 2015), and the focus of the investigation was on initial elastic stiffness.

A 3-dimensional foundation part was also created. The part was 42" square, with a depth extending 13" below the bottom of the baseplate. This closely resembled Barnwell's experiments, which had 13" total beneath the baseplate (12" of concrete and 1" of grout), and was 84" square. The dimensions were reduced to facilitate rapid computation, after it was discovered that the results on overall stiffness would be negligible (see Appendix A). An extruded cut was made to create a profile that more closely represents that of Barnwell's

experiments. Extruded cuts were also made where the column and baseplate were present in the foundation. An isometric view of the foundation part is seen in Figure 3-4. As with the column, the foundation was divided into rectangular sections to facilitate automatic meshing; Figure 3-5 shows the part with partitions.

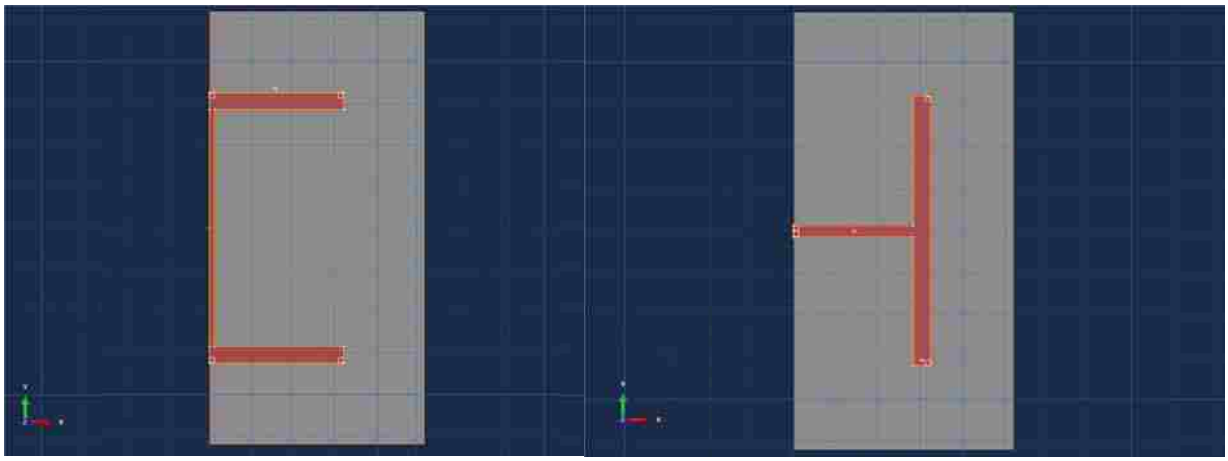
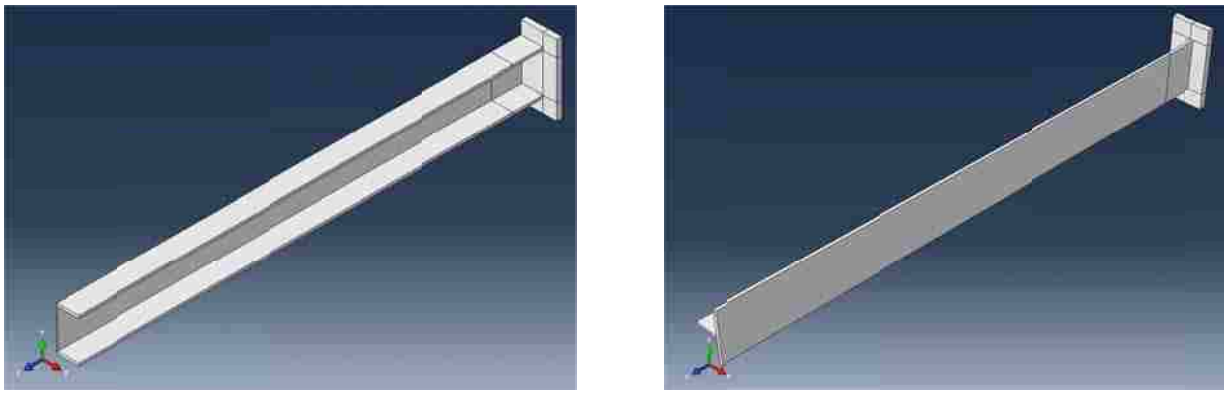
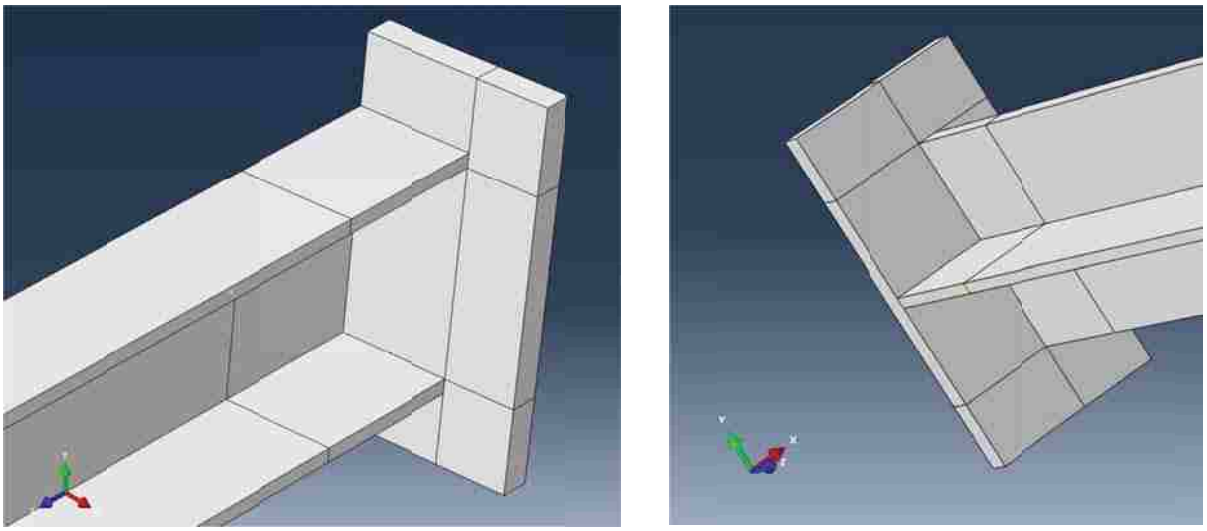


Figure 3-1: Typical Column Sketch (a) – Strong Axis, (b) – Weak Axis



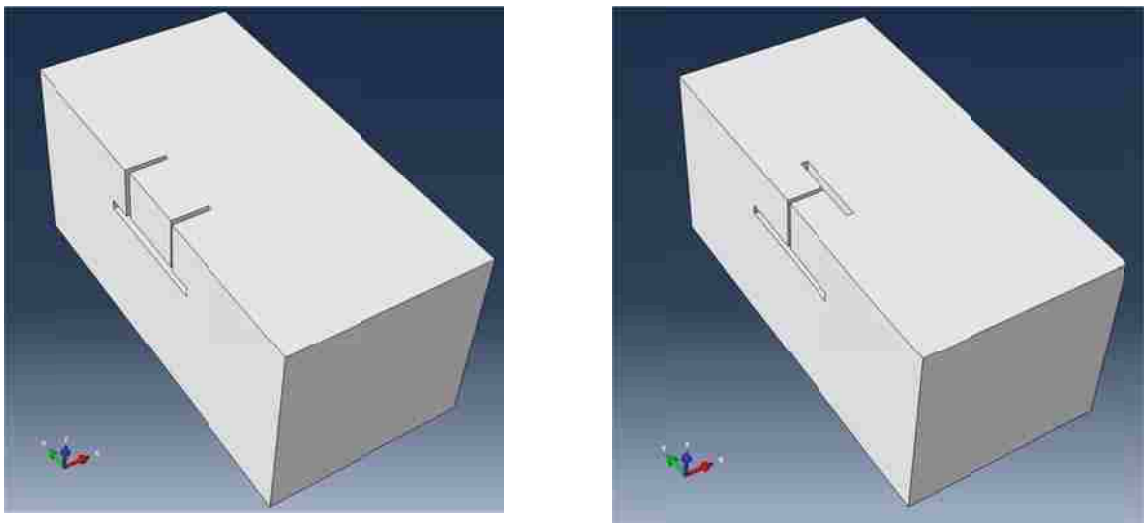
(a) (b)

Figure 3-2: Typical Column Parts (a) – Strong Axis, (b) – Weak Axis



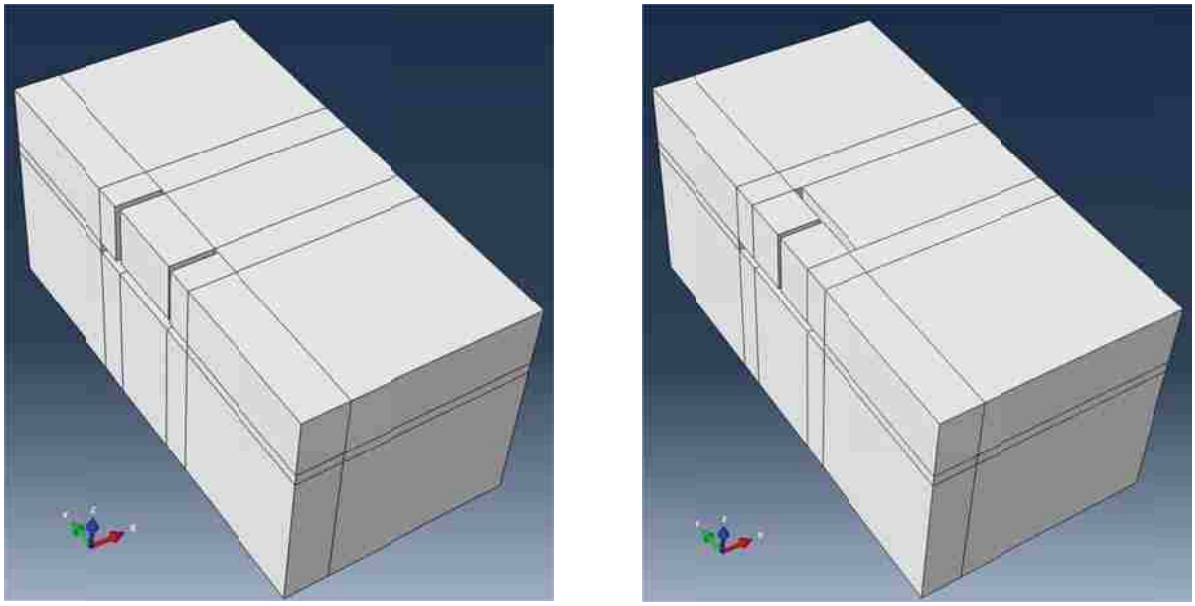
(a) (b)

Figure 3-3: Partitions on Column Part



(a) (b)

Figure 3-4: Typical Foundation Parts (Partitions Suppressed for Clarity)



(a) (b)

Figure 3-5: Typical Foundation Parts (Partitions Shown)

3.1.2 Model Mesh and Element Properties

A uniform mesh size of 0.5" inch was used with quadrilateral mesh shapes, and linear interpolation functions in each element. Specifically, C3D8R elements were used for both the column and the concrete parts. An C3D8R element is "an 8-node linear brick, [with] reduced integration [and] hourglass control." (Abaqus 2014) The parts were seeded and meshed with global seeding. Typical column and assembly meshes are seen in Figure 3-6 and Figure 3-7, respectively.

A mesh convergence study was performed to ensure the adequacy of the mesh refinement. Results are in Appendix A.

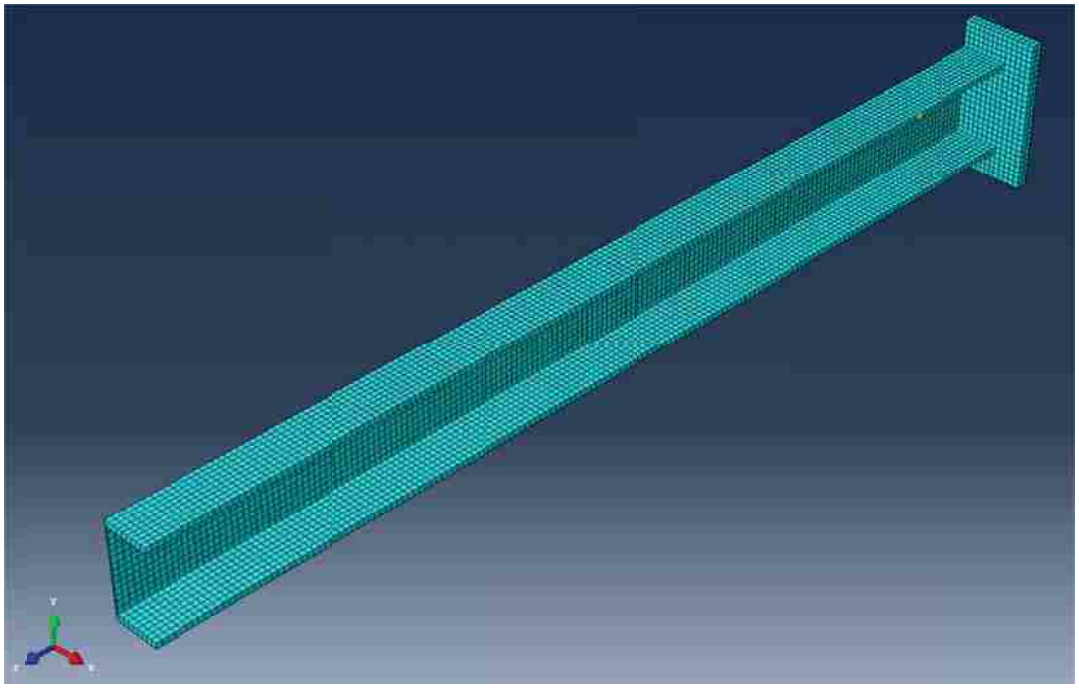


Figure 3-6: Typical Column Part Mesh

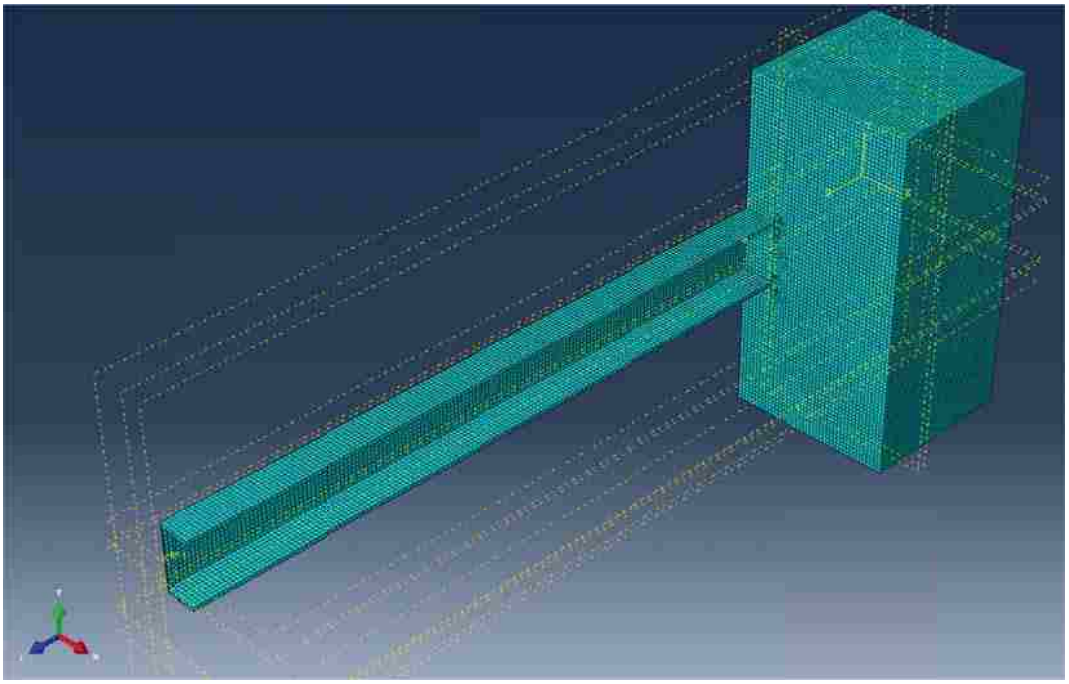


Figure 3-7: Typical Assembly Mesh

3.1.3 Material Properties

Linearly elastic materials were defined for steel, and concrete. The material properties are summarized in Table 3-1. The Young's modulus for concrete was specified as 3.5 * 10^6 psi, a value relatively close to the value obtained for f'_c = 4,000 psi, from the ACI Code (section 8.5.1)

$$E = 57{,}000 * \sqrt{f'_c}$$

Where f'_c = 4,000 psi.

(The actual value calculated from this equation was 3.60 x 10^5, but this slightly conservative value was chosen for computational convenience.) All concrete was given the same modulus, including the area of high-strength, non-shrink grout. This was done to greatly simplify the modeling process, and because results suggested the effect would be minimal (see Section 5.3). Also, since the grout would have a higher modulus than the concrete, neglecting the grout was considered conservative.

Table 3-1: Default Material Properties

Material	Young's Modulus (psi)	Poisson's Ratio
Steel	2.9 * 10^7	0.27
Concrete	3.5 * 10^6	0.15

The concrete was modeled as an elastic material because the applied load was specifically chosen so as to reduce the effects of material nonlinearity on the system's response. The object of this research was confined solely to the initial tangent stiffness, before high levels of material nonlinearity were seen in the experimental results.

The effects of rebar on the system response were neglected. Increasing the modulus of concrete uniformly (see Section 5.3) was not found to significantly increase agreement with Barnwell's results, so the effects of rebar were believed to be of negligible benefit.

3.1.4 Assembly and Boundary Conditions

The column and foundation parts were instanced in the assembly module in Abaqus. After each was instanced, it was positioned so that the column fit into the void left for it in the foundation part.

A fixed boundary condition was created that modeled the bond between the lab floor and the test pedestal. Likewise, fixed boundary conditions on the edges of the concrete pedestal that run parallel to the applied force represented the post-tensioned anchors that bonded the concrete pedestal to the floor. Sensitivity studies (see Appendix A) suggest that the precise nature of the boundary conditions has minimal effect on the overall stiffness results (>2%); it is believed that this is due to the extremely low stresses and strains experienced at the model's edge.

A symmetry boundary condition in the x-direction (yz-plane) was also applied, because of the half-model nature of these models. This boundary condition constrained movement in the X direction, and rotations about the y- and z- axes, for all nodes on the boundary; it allowed for displacement in the y and z directions, and rotation about the x-axis.

The results produced from the two model types differed not only in terms of the stiffness values obtained, but also in terms of the stress and displacement fields produced in the concrete. A major focus of investigating the DIC data was to analyze the strain profile in the concrete pedestal, and determine which of the models would be considered more reliable. Results from both model types were collected and analyzed.

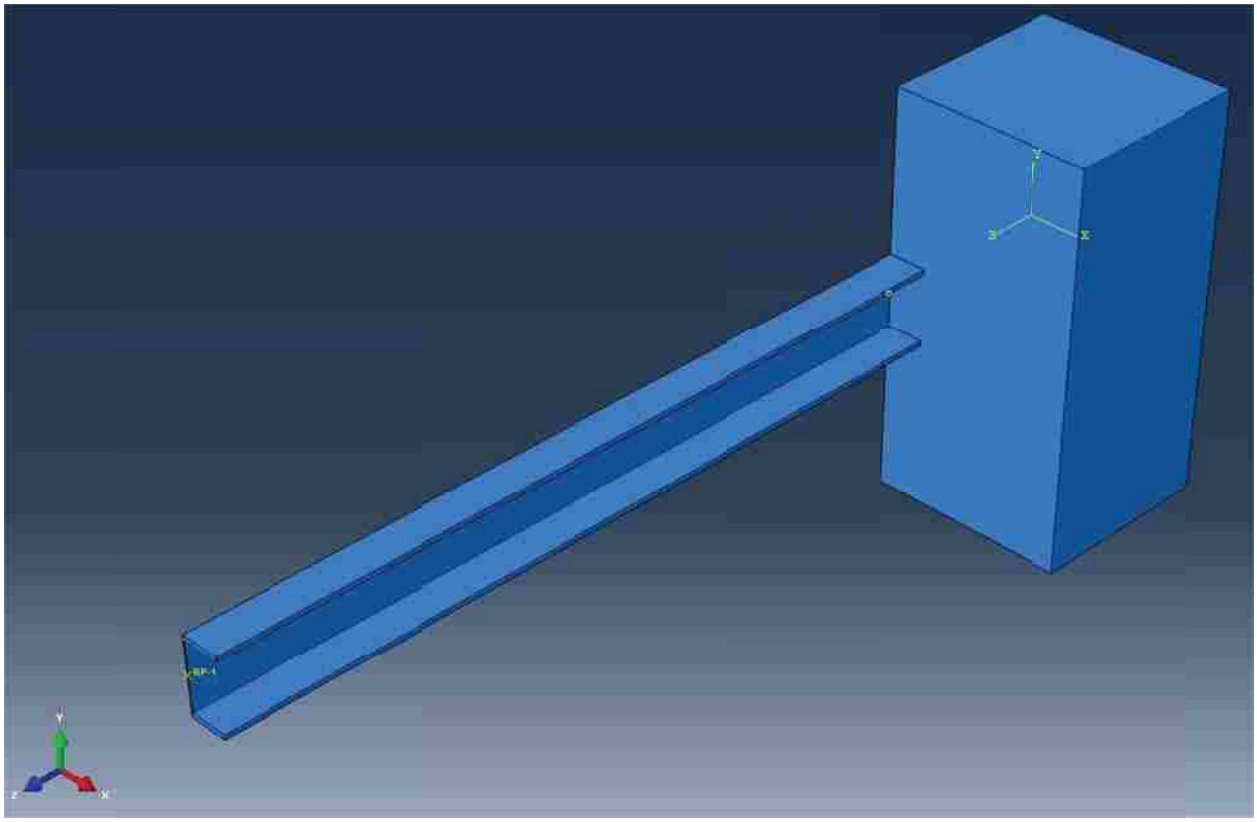
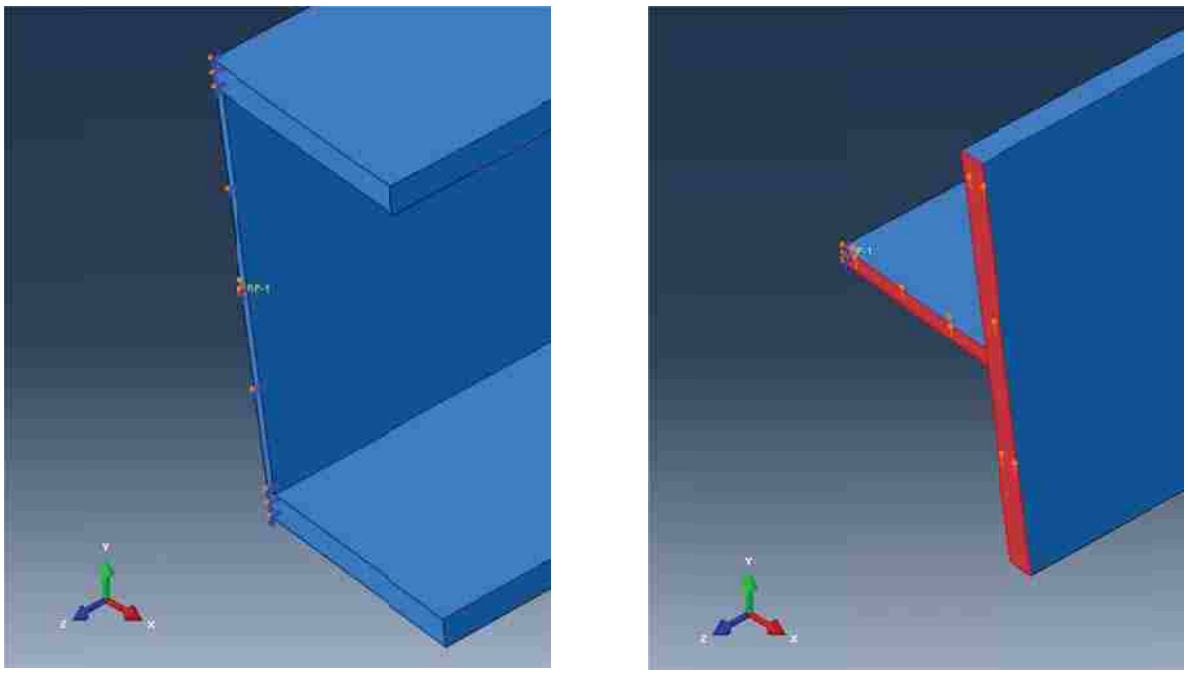


Figure 3-8: Typical Assembly Isometric View (Partitions Suppressed for Clarity)

3.1.5 Loads and Loading Constraints

A load step was created, and an applied load of 1,000 pounds (1 kip) was created at the center node of the edge of the protruding column. This is pictured in Figure 3-9 (a). The load was designed to model the force applied by the actuator in Barnwell's experiments. In order to prevent distortion due to local stresses, a multi-point tie constraint was applied to the top exterior face of the column, with the reference point at the node where the load was applied.



(a) (b)

Figure 3-9: Applied Load (a) Strong-Axis Bending; (b) Weak-Axis Bending

In the case of weak-axis bending specimens, the load was applied as a distributed load (traction) across the top section of the column. This is pictured in Figure 3-9 (b). When the load was applied as a point load for weak axis specimens, the center point of the column rotated in an asymmetric manner, despite the symmetry constraint. For this reason, it was decided to distribute the entire load directly, instead of relying on a multi-point constraint to distribute the load. Preliminary results showed that these methods give similar results in the case of strong-axis bending.

In models which included axial loads, they were applied at the same nodes as the lateral loads: in strong-axis bending, they were applied at one node in the center of the column; in weak-axis bending, they were applied as a distributed load.

3.1.6 Job Submission and Postprocessing

A field history request was created that contained displacement information at the point of applied load. A job was then created in Abaqus/CAE, and submitted to Abaqus/Standard for processing. Four processors were used in parallel.

Upon completion of the analysis, the field output request was queried, and its information was submitted to an XY report in Abaqus. From there, the data in the XY report was exported to a report (.output) file. The report file was then read for the displacement value, and exported to a database (.csv) file which could be opened and manipulated in Microsoft Excel. With the displacement value available, the connection stiffness was calculated according to the equations in Section 3.1.7.

3.1.7 Linear and Rotational Stiffness Models

The linear stiffness model used by Eastman (2011) and Barnwell (2015) was used to compute the connection lateral stiffness. That is,

$$k_{conn} = ({k_t}^{-1} - \frac{L^3}{3EI})^{-1}$$

$$= (\frac{F}{\Delta_{total}}^{-1} - \frac{L^3}{3EI})^{-1}$$

Where

F = applied force = 1 kip.

Δ_{total} = total displacement measured at the point of application of the force

E = Young's Modulus of the column = 2,900 ksi

I = Moment of inertia of the column about the bending axis

Also, the rotational stiffness of the connection was computed, as follows:

$$\beta = M/\theta_{conn}$$

$$= \frac{\mathrm{F} * \mathrm{L}}{\Delta_{conn} / \mathrm{L}}$$

$$= k_{conn} * L^2$$

Where

M = induced moment due to lateral loading

θ_{conn} = connection rotation

L = cantilever length of the column

$\Delta_{conn} = F / k_{conn}$

These calculations assume that the entire connection can be modeled as a linear rotational spring of stiffness β.

3.2 **Model-Type Specific Modeling**

Three different model types were developed, each with different connection mechanisms between the column and the concrete. The first model type, a contact-based model, modeled the force transfer mechanism as primarily occurring through bearing pressure in compression, with frictional forces in shear, and allowing separation of the bodies in tension. The second model type, a cohesive zone-based model, used very thin elements of cohesive material at the interface, which represented the imperfect bonding between the column and the foundation (from physical and chemical adhesion) as a layer of cohesive material with a reduced modulus. The third

method, a tied-based model, represented the two parts as one part, with a perfect force transfer mechanism between them.

This section will explain the various methods that were used to generate models in each of the three different model types.

3.2.1 Contact-Based Connection Modeling

In the contact-based model, the two parts were connected with contact interactions, including a hard contact pressure-overclosure formulation for normal forces, and a frictional formulation for tangential forces. Therefore, if two surfaces are not in contact, no pressure will be applied, and separation will be allowed between the surfaces. If the two surfaces are in contact, there will be no overclosure, and pressure will be nonzero. Stated mathematically,

$$p = 0 \; for \; h \; < 0$$

$$h = 0 \; for \; p > 0$$

Where

p = contact pressure between two surfaces at a point;

h = overclosure, which is the depth of interpenetration between the two surfaces.

The contact constraint is enforced numerically with a Lagrange multiplier representing the contact pressure (Abaqus 2014). Shearing between surfaces with normal pressure is allowed, if the shearing stress exceeds the normal stress times the coefficient of friction.

Given the wide variation in coefficients of static friction available from the literature (see Section 2.9) in a wide variety of circumstances and tests, a sensitivity study was performed. In this study (see Section 5.1.2), it was shown that the actual value makes little difference in the stiffness results. A conservative value of the static friction coefficient was therefore taken as

0.50, which was slightly less than the value determined by Rabbat (1985), and similar to the value used by Eastman (2011).

Figure 3-10 shows the sections which are in contact in a typical contact-based model. The column was assigned to be the master surface, while the foundation was assigned to be the slave surface.

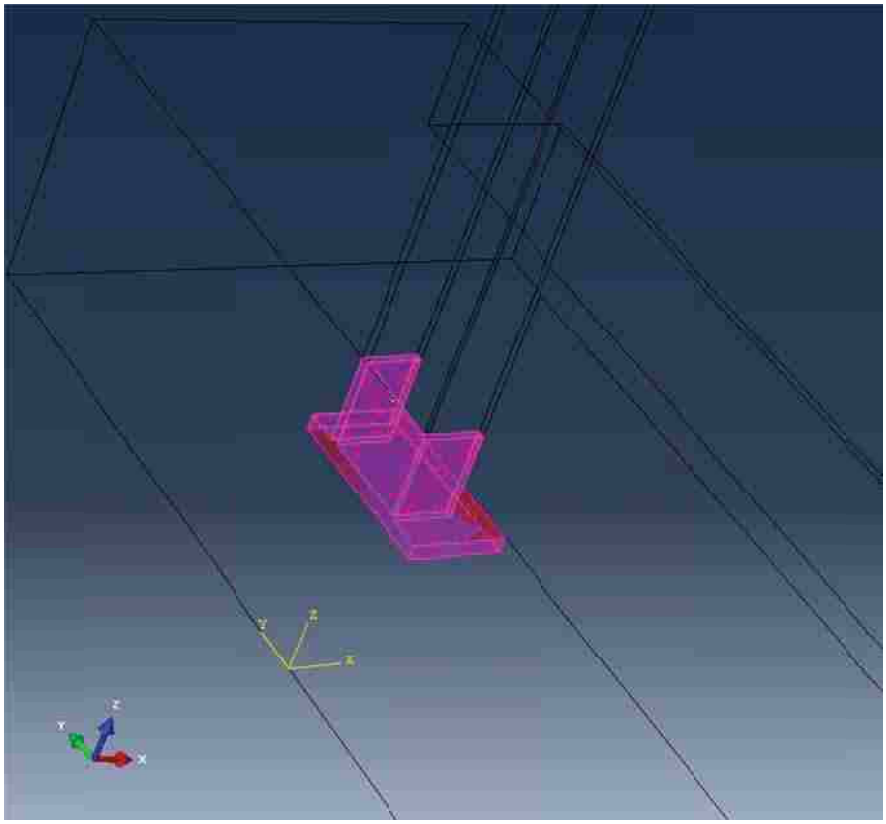


Figure 3-10: Regions in Contact in a Typical Contact-Based Model

3.2.2 Cohesive Zone-Based Modeling

In the cohesive zone-based models, the bond between the column and the slab on grade was modeled as a cohesive zone of reduced stiffness, though not allowing separation.

The foundation part was divided at every point of contact with the column. A new cell of 0.001" thickness was therefore created at every point of contact. These cells became the cohesive

zones. For ease of programming, a datum plane was created for each cut, which was offset by 0.001" from the existing face. Then the existing cells were selected and divided using the "divide by datum plane" option. The faces on the exterior of the cohesive zones were then tied to their corresponding faces on the column part.

The cohesive zone cells were then assigned appropriate cohesive elements and material definitions. The cohesive zones were assigned COH3D8 elements. Abaqus describes these as "8-node three-dimensional cohesive element[s]." The cohesive elements were assigned a cohesive material definition with traction-separation relationships defined. Since this could not be directly measured, several tests were performed (see Section 5.1.1) to investigate which values of traction-separation would give results closest to the observed values. Thus, the traction-separation values became a calibrated parameter.

3.2.3 Tie-Based (One-Part) Modeling

In the "tie-based" or "one part" method, both the column and the foundation were modeled as one part, with a continuous mesh between the two. Only the material properties between the column and the foundation varied. This was accomplished in Abaqus by creating and instancing both column and foundation parts in the assembly module as described above, and then using Abaqus' functionality to combining parts into one part, also in the assembly module. This ensured perfect force transfer at the interface between the column and the concrete. This is numerically equivalent to a "tie" constraint between the two parts, but requires significantly less computational time, and ensures a compatible mesh.

3.3 Automatic Model Generation

Abaqus/CAE processes commands from the user interface in the Python programming language. These commands are automatically saved in a journal (.jnl) file, and are also saved into a separate file when recording a macro. Many of the Python commands are highly specific to the Abaqus software package, with custom libraries available for the sole purpose of creating and analyzing Abaqus models.

To conduct the analysis process, Python scripts were created that automated the model generation and analysis processes. Macros were developed that performed all model creation, analysis, and postprocessing tasks, as described in Section 3.1. These macros were then edited to 1) be easier to read and understand, 2) accept input parameters, 3) loop across desired input parameters. Then, every time it was desired to study the effects of one or several variables on model behavior, a list of desired variables was created at the start of the scripts, and the scripts were run. A detailed explanation of the scripting process and the scripts themselves, are available in Appendix E.

3.4 Model Generation Matrix

The models that were generated are summarized in Table 3-2. The primary variable studied in each case was embedment depth. Each model was tested at embedment depths from 0.5 inches to 19.5 inches at 2 inch increments.

Table 3-2: Model Generation Matrix

Variable Investigated	Values	Shapes Investigated
Cantilever Length	80.25, 83.25	W8x35, W8X48
Baseplate Thickness (Full Baseplate)	0.5, 1.0, 2.0, 3.0, 3.5	W8x35; W8x48; W14x176
Baseplate Thickness (Reduced Baseplate)	0.5, 1.0, 2.0, 3.0, 3.5	W8x35; W8x48; W14x176
Column Orientation	Strong, Weak	W8x35; W8x48
Concrete Modulus	2.9E4, 2.9E5, 2.9E6, 2.9E7	W8x35
	3E5, 3.5E5, 4E5, 4.5E5, 5E5	W8x35
	3E6, 3.5E6, 4E6, 4.5E6, 5E6	W8x35
Column Shape	--	W8x35; W8x48; W14x176; W24x76
Presence of Axial Load (x lateral load)	0.5, 1, 2, 5, 10, 20, 50	W8x35; W8x48
Traction-Separation Relationship	1E4, 5E4, 1E5, 5E5, 1E6	W8x35; W8x48

3.5 **Assumptions and Model Limitations**

A number of limitations and simplifying assumptions are inherent in the finite element models generated. These include the following:

- Linear elastic material behavior at low strains;
- Geometric linearity;
- Limitations inherent in the discretization process in the finite element solver;
- Anchor bolts, grout pad, column fillets, baseplate welds, rebar, and construction joints were not modeled due to the additional complexity they would create in the model, and because their effect on stiffness was considered negligible, based on the experiments performed by Barnwell (2015).

4 DIC DATA AND ANALYSIS

In Barnwell's study, Digital Image Capture (DIC) data were collected and stored. The DIC system consisted of two high-resolution cameras recording the column and pedestal during the course of the loading procedure, and speckled paint applied to the specimen itself. The presence of the two cameras allowed software to determine the exact spatial coordinates of every point on the specimen, via triangulation. The individual points of paint were used by the software as reference points, which allowed the software to determine displacement fields with relatively high precisions. The software used was Ristra 4D. This DIC data were generated for every specimen. Although this data were recorded during Barnwell's tests, it was not analyzed in the course of his research; it is presented and analyzed for the first time here.

Table 4-1 summarizes the tests which are analyzed in this section. The tests that were subsequently labeled A1-B4 in Barnwell (2015) were originally labeled A-K during laboratory testing; the table lists both labels.

The qualitative results for displacement in the direction of applied loading, and parallel to the length of the column are presented. These directions are called the Y-direction and Z-direction, in keeping with the coordinate system established for the Abaqus models (see Section 3.1). The software used had an option available to remove the rigid body motion (rigid body motion removed, or RBMR). Without this option enabled, it would have been very difficult to

separate the amount of displacement caused by concrete deformation, from rigid body translation of the slab. As such, all y-direction plots are presented with rigid body motion removed.

Table 4-1: Summary of DIC Test Parameters

Thesis Label	Test Label	Embedment	Orientation	Column Shape	Anchor Bolts Engaged?	Braced Slab
A1	C	8	Strong	W8X35	Yes	--
A2	B	8	Strong	W8X48	Yes	--
B1	H	16	Strong	W8X35	Yes	--
B2	G	16	Strong	W8X48	Yes	--
CA2	A1	8	Strong	W8X48	No	No
CB2	F1	16	Strong	W8X48	No	No

4.1 **Test A1 (C)**

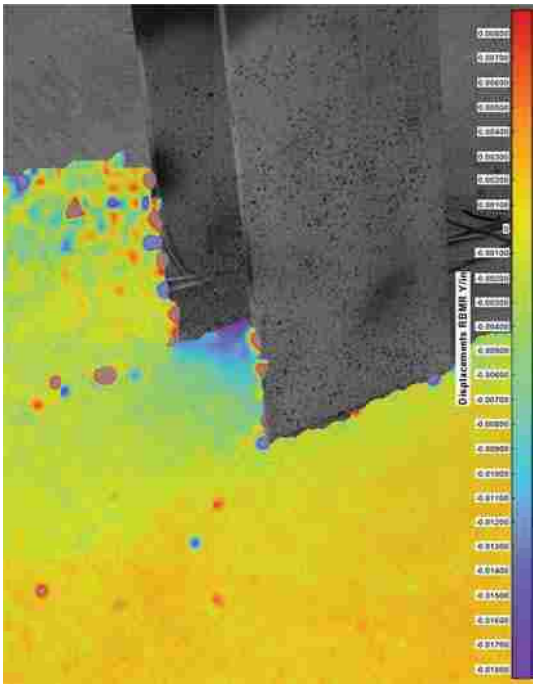


Figure 4-1: Test A1, RBMR, Y-Direction Displacement

Test A1 (C) was of a W8x35 shape, with 8" of embedment, and strong axis bending. Figure 4-1 shows the RBMR displacement in the direction of the applied force. This figure was sampled at an applied load of 9001lbs, pushing towards the camera. It shows the greatest zones of deformation – and therefore stress – in the area circumscribed by the column flanges, especially near the intersection of the web with the inside face of the column. Surprisingly, relatively little deformation was present outside of the flange face. Figure 4-2 shows deformation in the Y-direction. It shows the slab experiencing uplift as a mostly rigid body. Little to no deformation is seen on the compressive side of the slab.

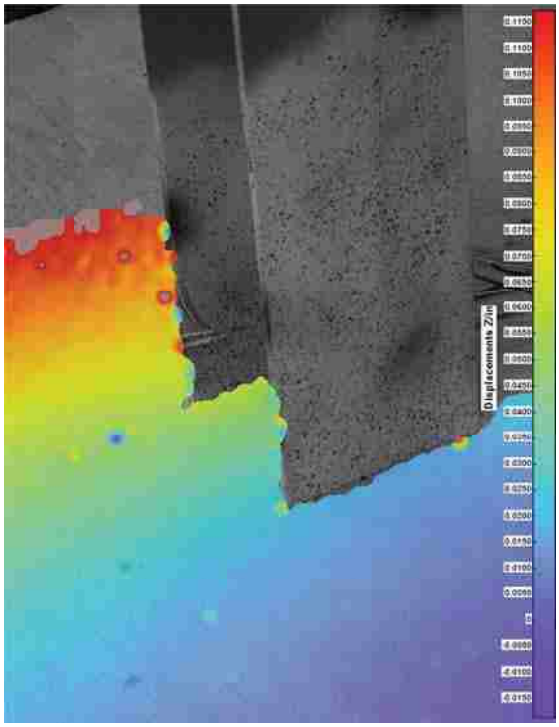


Figure 4-2: Test A1, Z-Direction Displacement

4.2 Test B1 (H)

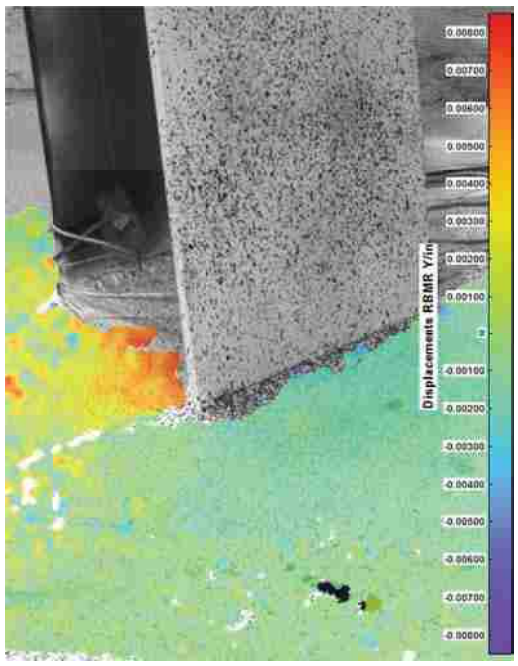


Figure 4-3: Test B1, RBMR, Y-Direction Displacement

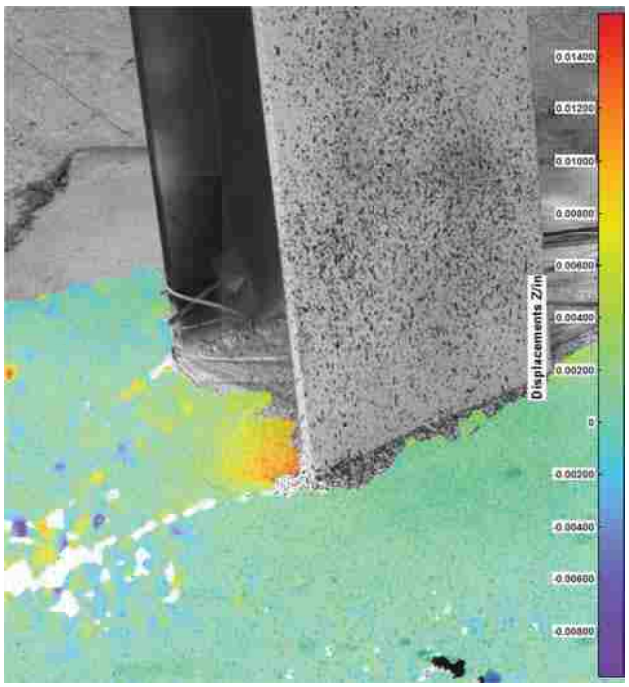


Figure 4-4: Test B1, Z-Direction Displacement

Test B1 (H) was of a W8x35 shape, with 16” of embedment, and strong axis bending. Figure 4-3 shows the RBMR displacement in the direction of the applied force. This figure was sampled at an applied load of 8861lbs, pushing away from the camera. Although reliable data is not available for the area circumscribed by the column flanges (due to interference from instrumentation cables), the zone of greatest visible deformation is directly bordering that area. This supports the hypothesis that the greatest deformation will occur within the circumscribed area.

Figure 4-4 shows deformation in the direction parallel to the column itself, taken at the same moment of testing. It shows the slab experiencing uplift much closer to the column itself, deforming in a more flexible manner than the slab in Figure 4-2. This would only be possible if microcracking – or cracking beneath the surface – allowed differential deformation. This suggests that the slab may have behaved in different manners between the different specimens.

4.3 **Test A2 (B)**

Test A2 (B), was of a W8x48 shape, at 8” of embedment, and bending about the strong axis. Figure 4-5 shows the RBMR y-displacement at an applied load of 8828 lbs, pushing towards the camera. As in other tests, it appears to show a greater displacement in the area circumscribed by the flanges, and especially at the point closest to the intersection of the web and flange. Figure 4-6 shows the Z-direction under the same load.

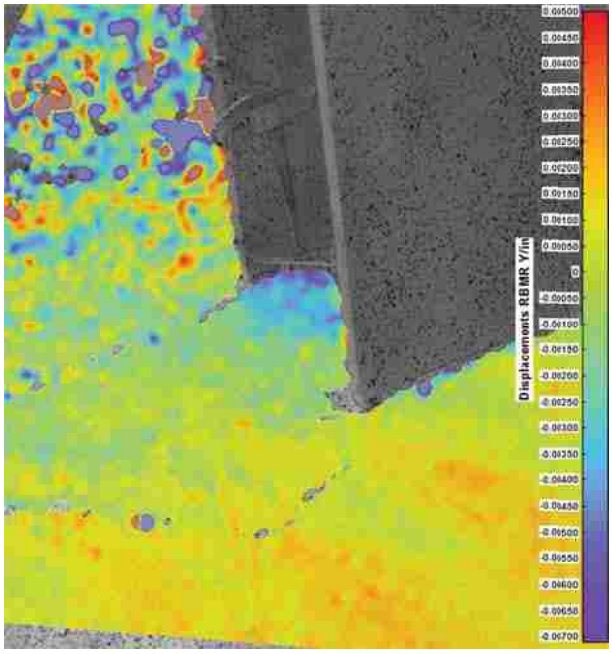


Figure 4-5: Test A2, Y-Direction, RBMR Displacement

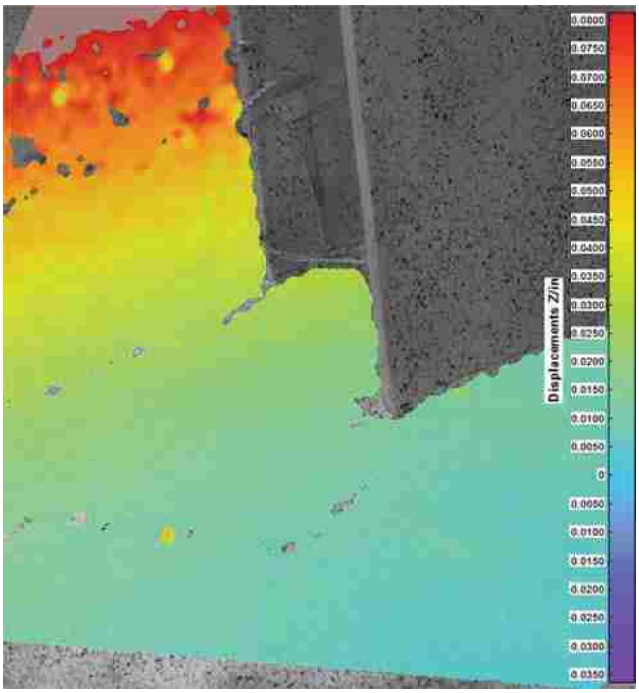


Figure 4-6: Test A2, Z-Direction Displacement

Figure 4-7 shows the specimen at an applied load level of 9081 lbs, before the appearance of cracks or gaps visible to the naked eye. The figure shows a slight negative displacement is visible in the lower-right hand corner of the figure. That is, the foundation appears to displace in the opposite direction from the applied load. Two explanations are possible. The first is that it is a numerical error caused by the RBMR algorithm used in Ristra, subtracting more rigid body motion than is actually occurring. However, if it is not a numerical error, it indicates the presence of gapping between the concrete and the foundation. That is, the steel-concrete bond is not sufficiently strong to prevent the column from separating from the concrete; what's more, the foundation is rotated – and therefore pushed back – slightly by the applied moment. This gapping effect is predicted in displacement plots from the contact-based model in Abaqus (see Section 5.1.2).

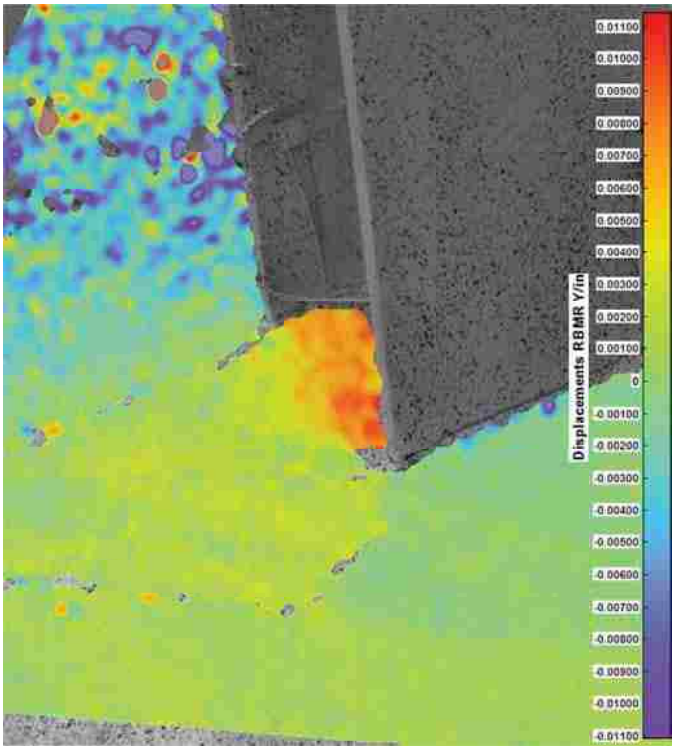


Figure 4-7: Test A2, Y-Direction, RBMR Displacement

4.4 **Test B2 (G)**

Test specimen B2 (G) is a W8x48 shape, with 16" of embedment, and strong axis bending. Figure 4-8 shows the RBMR displacement field in the Y-direction, with an applied loading of 8889 lbs, pushing towards the camera. The displacement field is qualitatively equivalent to those of other test specimens. Figure 4-9 shows the field of displacement in the Z-direction.

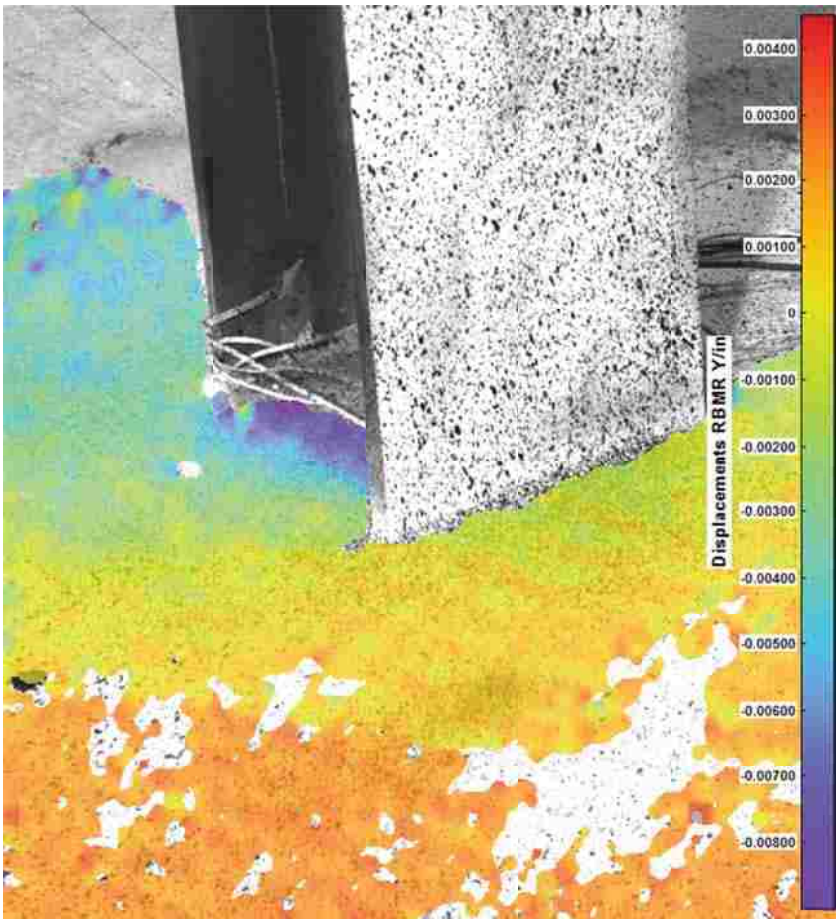


Figure 4-8: Test B2, Y-Direction, RBMR Displacement

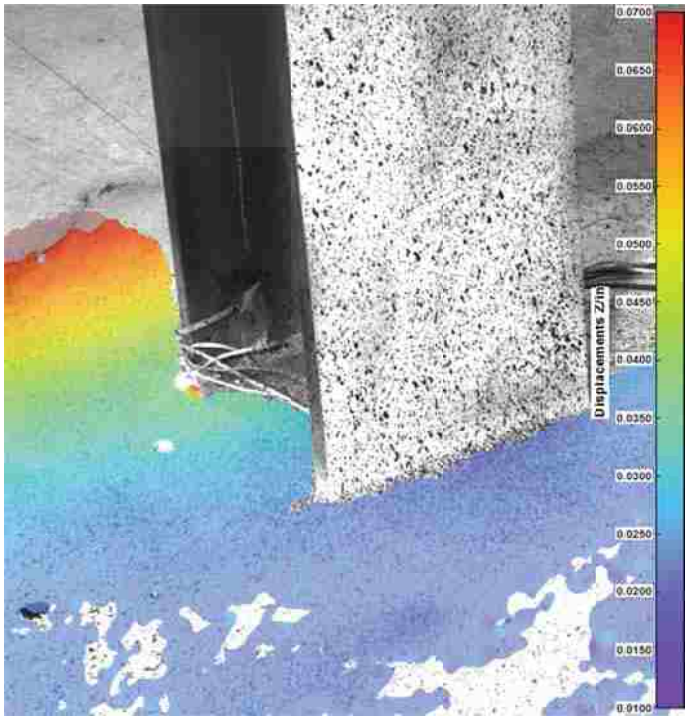


Figure 4-9: Test B2, Z-Direction Displacement

4.5 Test CA2 (A1)

Test CA2 was a W8X48 shape, with 8" of embedment, strong axis bending, and anchor bolts disengaged. This test was similar to Test A2, only with anchor bolts disengaged. As such, the specimen was never able to obtain 9000 lbs. ultimate strength; the maximum available was around 2000 lbs. Figure 4-10 shows the RBMR displacement at a load of 2044 lbs, pushing towards the viewer. It appears that the bottom section of the foundation is being pushed away from the viewer, which is somewhat surprising. Figure 4-11 shows the Z-axis displacement at the same moment.

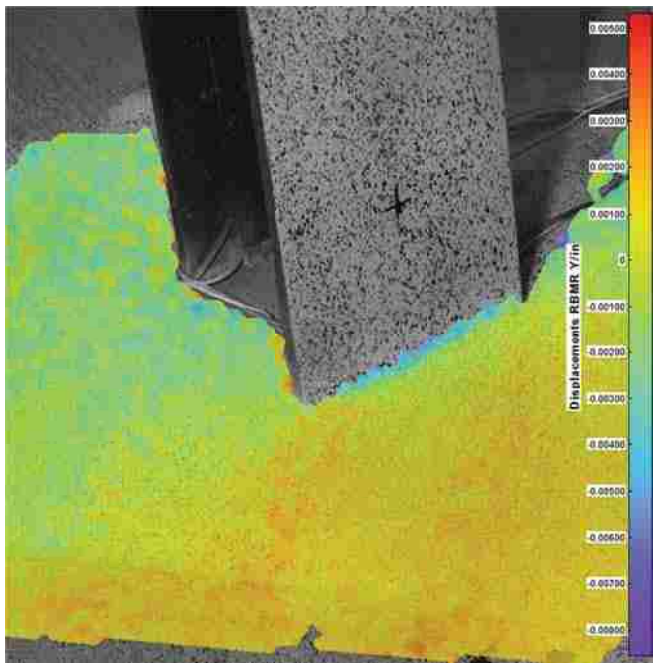


Figure 4-10: Test CA2, Y-Direction, RBMR Displacement

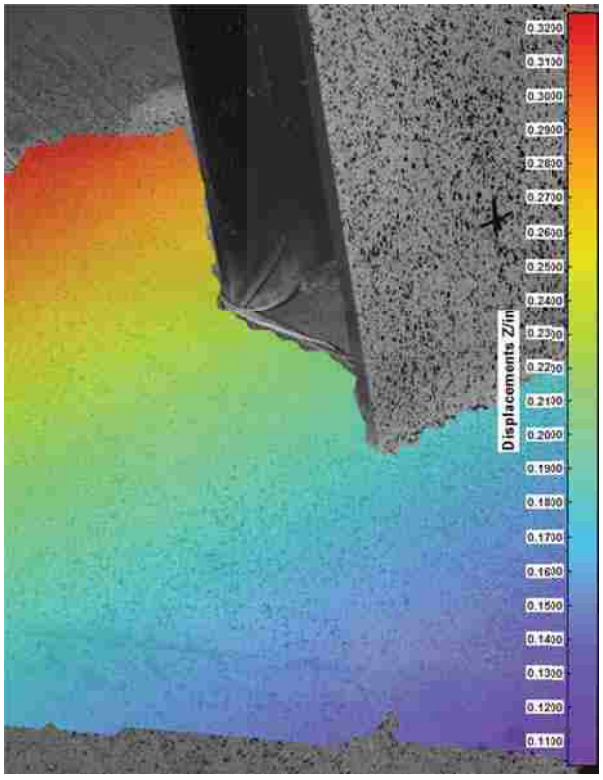


Figure 4-11: Test CA2, Z-Displacement

4.6 **Test CB2 (F1)**

This test was of a W8X48 shape, with 16" embedment, strong axis bending, and anchor bolts disengaged. This test was essentially the same as Test B2, with anchor bolts disengaged.

Figure 4-12 shows the displacement in the Y-direction (RBMR) at a load of 9309 lbs, pushing away from the viewer. As in previous tests, more relative displacement is apparent in the area circumscribed by the column. In contrast to previous tests, however, the RBMR is clearly showing differential movement between the concrete above the column flange and the concrete below the column flange. Although no cracks are yet visible to the naked eye, this is the point at which cracks were to appear several loading cycles later. In effect, the DIC data is indicating the presence of microcracking before it becomes visible to the naked eye.

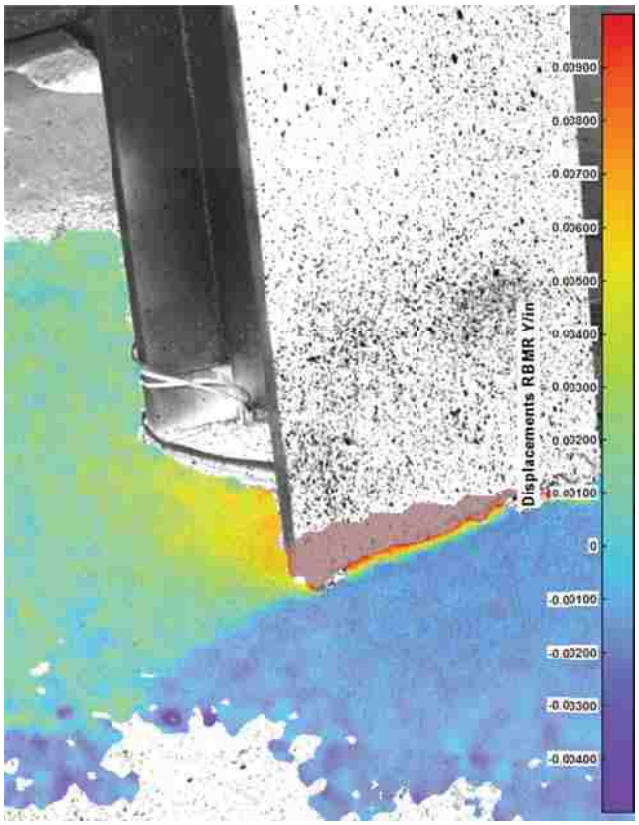


Figure 4-12: Test CB2, Y-Displacement, RBMR

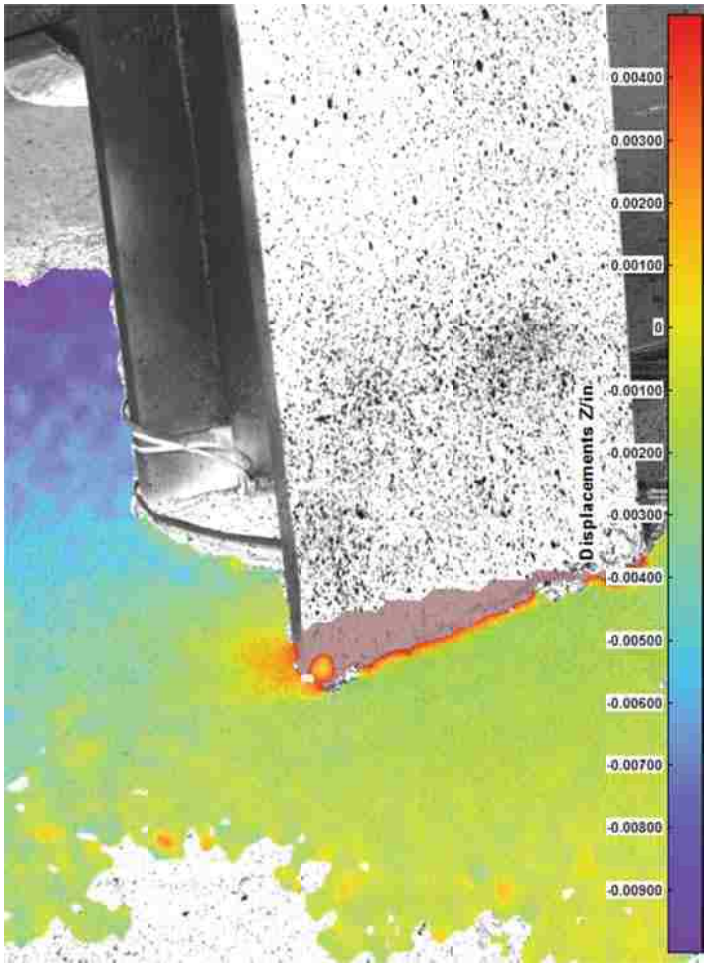


Figure 4-13: Test CB2, Z-Displacement

5 FINITE ELEMENT MODEL RESULTS AND DISCUSSION

In Section 5.1, the experimental results from Barnwell (2015) will be compared to the results from three families of FEA models. In Sections 5.2 through 5.5, the effects of varying various input parameters on connection stiffness will be investigated. These include: column shape (5.2), concrete modulus of elasticity (5.3), baseplate geometry (5.4), axial load (5.5), and column orientation (5.7).

The available connection stiffness typically approaches an asymptotic upper limit as the embedment depth increases. The maximum stiffness value, and the rate of gain of stiffness with embedment depth, are both governed by the geometry of the column connection, and the material properties of the concrete into which the column is embedded.

5.1 Comparison of Model Types with Barnwell (2015)

By comparing the results of the three separate model types with Barnwell's results, it was determined that the contact-based model is likely most accurate at shallow embedment depths, while a calibrated cohesive zone based model is likely most accurate for deeper embedment depths. Therefore, results from both cohesive-zone models and contact-based models will be included in subsequent results sections.

As noted previously (Section 2.12.1), there is a 2.5 inch discrepancy between the blockout depths – which Barnwell refers to in his study – and the embedment depth of the column. This study will reference the values of embedment depth, not blockout depth.

5.1.1 Cohesive Zone-Based Model

The presence of a layer of cohesive zone elements allowed for results which were intermediate between that of a perfectly-bonded connection, and that of a contact-friction based connection only. The results showed increasing connection stiffness with increasing depth.

Tests with two cantilever heights were performed. In Barnwell's study, the cantilever height Z – from the top of the concrete foundation to the midline of the actuator – varied depending on whether the specimen was a shallow or deeply embedded specimen. Therefore, complete curves for both cantilever heights were created to allow direct comparison in both cases.

The results can be represented in a variety of different ways. Figure 5-1 is a plot of total displacement values obtained by the finite element analysis, overlaid with the displacement values obtained by Barnwell, for W8x35 shapes. The dashed lines show the expected displacement from a two perfectly fixed connections, owing only to the deformation of the columns at cantilever heights of 80.25" and 83.25". Figure 5-2 shows the linear stiffness corresponding to the connection itself, which is equivalent to the inverse of the displacement not caused by column deformation. Figure 5-3 shows the rotational stiffness of the connections, which is equivalent to the values in Figure 5-2, multiplied by a constant (the cantilever length squared).

In the case of W8x35 shapes, stiffness values for shallow specimens are overpredicted by 25.8%, which stiffness values for deeper specimens are underpredicted by 27.06% (see Figure 5-3 and Table 2-1).

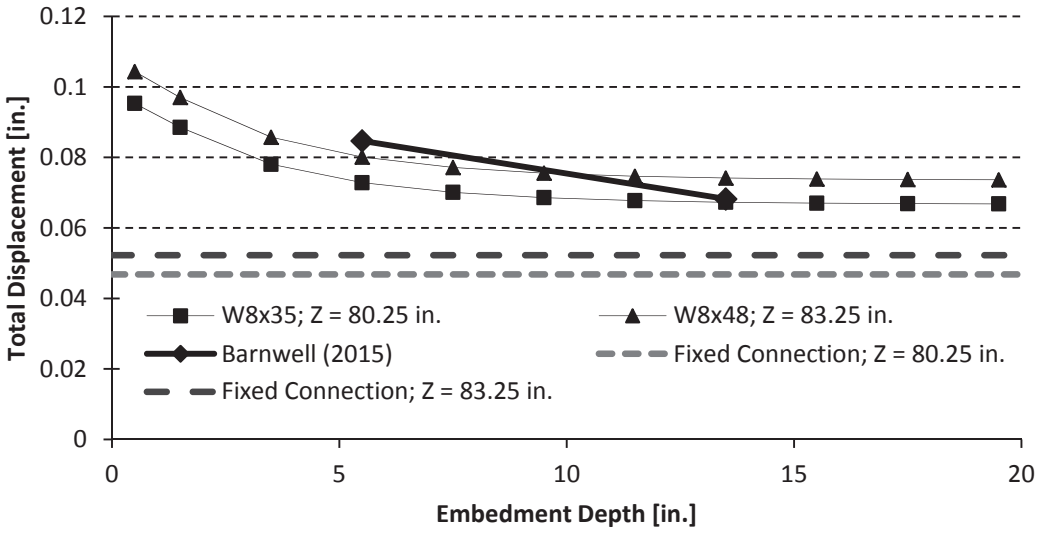


Figure 5-1: Total Displacement; Cohesive Zone Models

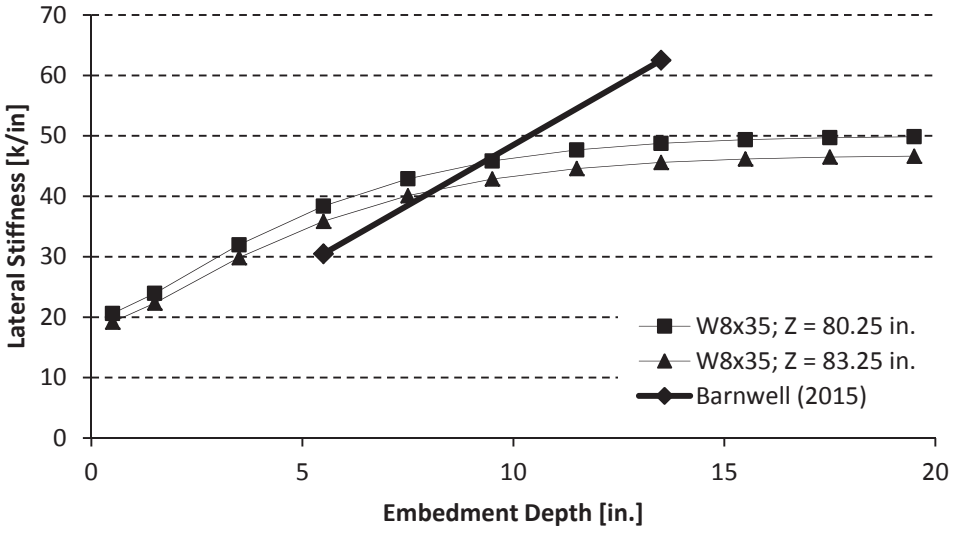


Figure 5-2: Lateral Stiffness; Cohesive Zone Models

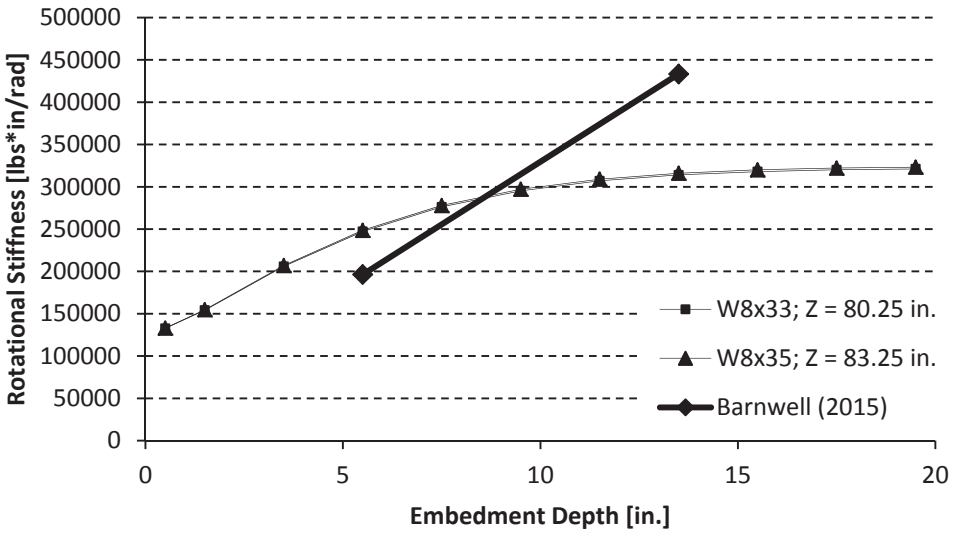


Figure 5-3: Rotational Stiffness; Cohesive Zone Models

Table 5-1: Comparison of Experimental and FEA Results

Specimen - W8x35	Barnwell (2015)	FEA Results	Ratio
5.5" Embedment	30.49	38.39	0.79
13.5" Embedment	62.52	45.61	1.37

Because these figures represent fundamentally the same information, only one type of figure will be presented at a time for the body of this paper. Results will be presented in the form of linear connection stiffness, corresponding to Figure 5-2, with the exception of Section 5.5, which concerns the relationship of the column's cantilever height to the rotational stiffness.

Recall that the traction-separation relation was not measured directly during Barnwell's testing, and so was calibrated such that the models matched the experimental stiffness results as closely as possible. A traction-separation value of $5*10^5$ psi / inch was used; that is, for every inch of separation, a traction (or applied pressure) of 50,000 psi would be required. This value

was found to have reasonable agreement with both shallowly and deeply embedded connection values, although it was unconservative in the shallow embedment case. Figure 5-4 compares the results for various traction-separation relationships with the results from Barnwell (2015).

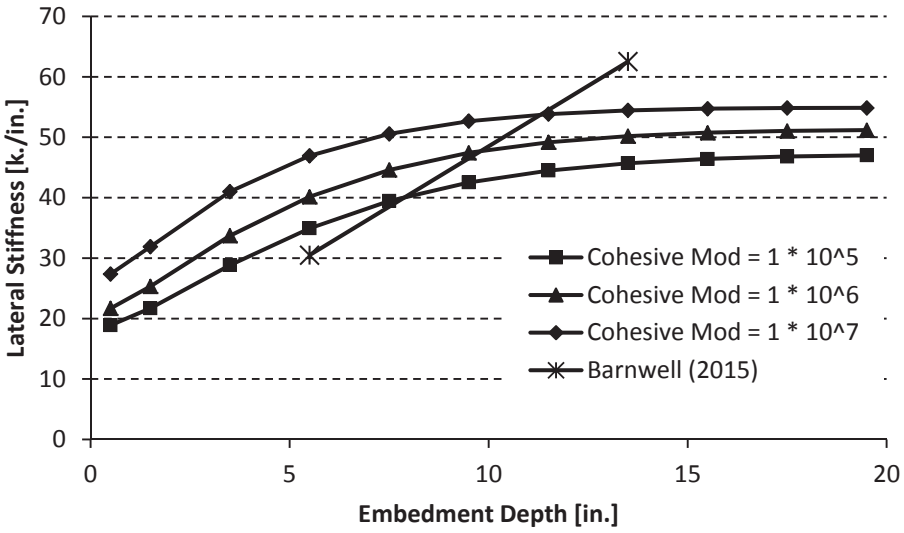


Figure 5-4: Comparison of Various Traction-Separation Relationships with Barnwell (2015)

Figure 5-5 shows a contour plot of a typical von Mises stress field in the concrete part, with the column part removed. A large stress concentration exists at the corners of the column flanges, especially the bottom flange. Two possible explanations exist for these concentrations. The first possibility is a numerical error caused by the tie constraint between a very fine mesh and a relatively coarse mesh. The second is an artificial stress concentration caused by the sudden discontinuity in material type at the corners, causing a pinching effect in shear of the softer cohesive material between the stiffer concrete and steel materials. However, it is not immediately clear why this would cause the lower flange to experience greater stress

concentrations at the bottom flange than the top flange, as is seen in Figure 5-5, and is typical of the various shapes studied.

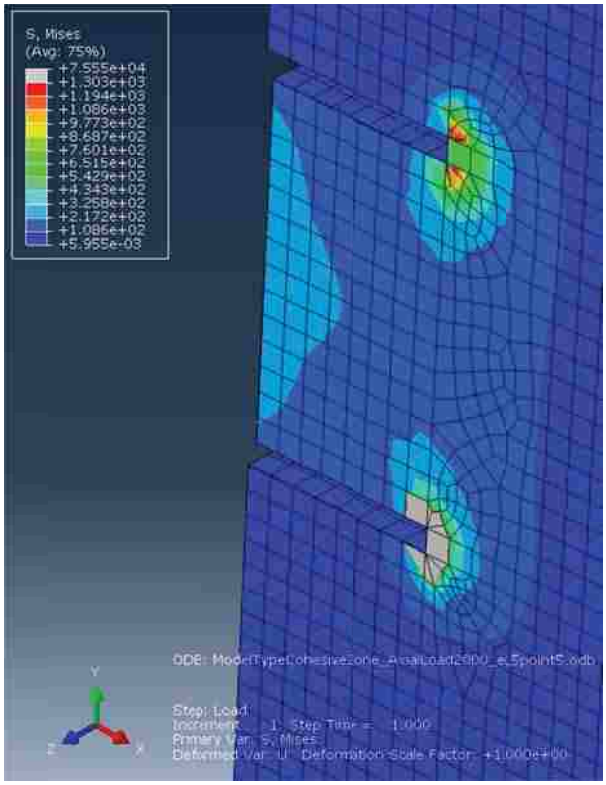


Figure 5-5: Typical von Mises Stress Field

5.1.2 Contact-Based Model

Figure 5-6 shows lateral stiffness values for strong-axis, W8x35 specimens, with Barnwell's results overlaid. The contact-based model gives values which are low for shallower shapes, and high for deeper embedments. As the embedment depth increases, these models show a decreasing stiffness, the opposite trend from what was observed. The model allows gaps to open up between the column and the concrete, meaning the concrete in tension fails to resist bending, and so the effective cantilever length increases.

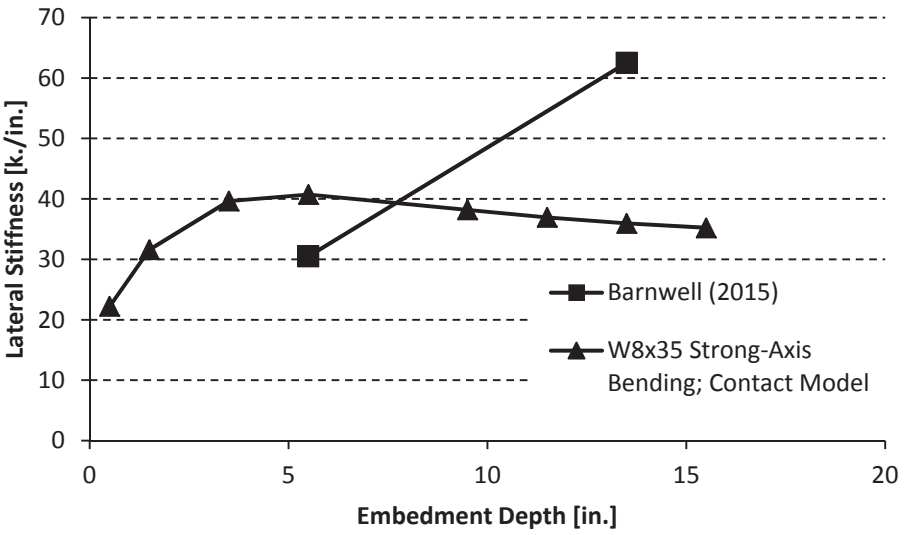


Figure 5-6: W8x35 Specimens, Strong Axis Bending

Figure 5-7 shows a typical stress field from the contact-based model. Stress concentrations are visible along the centerline of the column, near the web-flange intersection. Slight stress concentrations are noted near the flange's corners.

Various sources have reported different values for the coefficient of friction, μ (see Chapter 3). Therefore, to investigate sensitivity to the assumed coefficient of friction, a wide range of values were tested (at a mesh size of 1.0). The results are shown in Figure 5-8.

Varying the coefficient of friction has a relatively modest effect for reasonable values of μ. It is impossible to account for the discrepancy in stiffness values at deeper embedment depths by varying only to the coefficient of friction, at any reasonable values of μ.

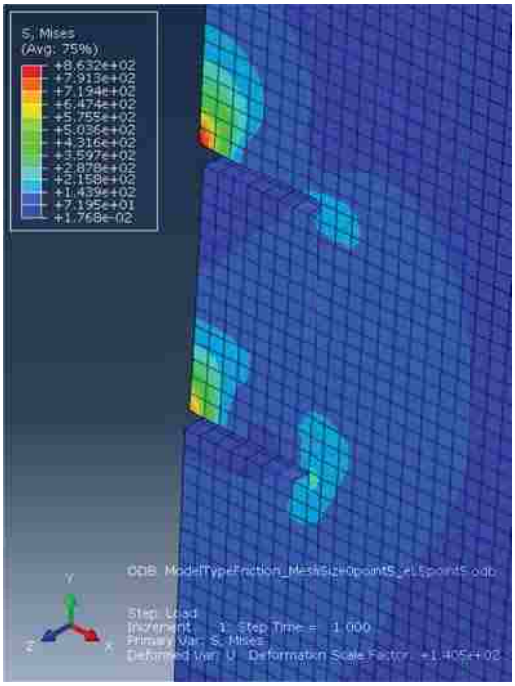


Figure 5-7: Typical von Mises Stress Field

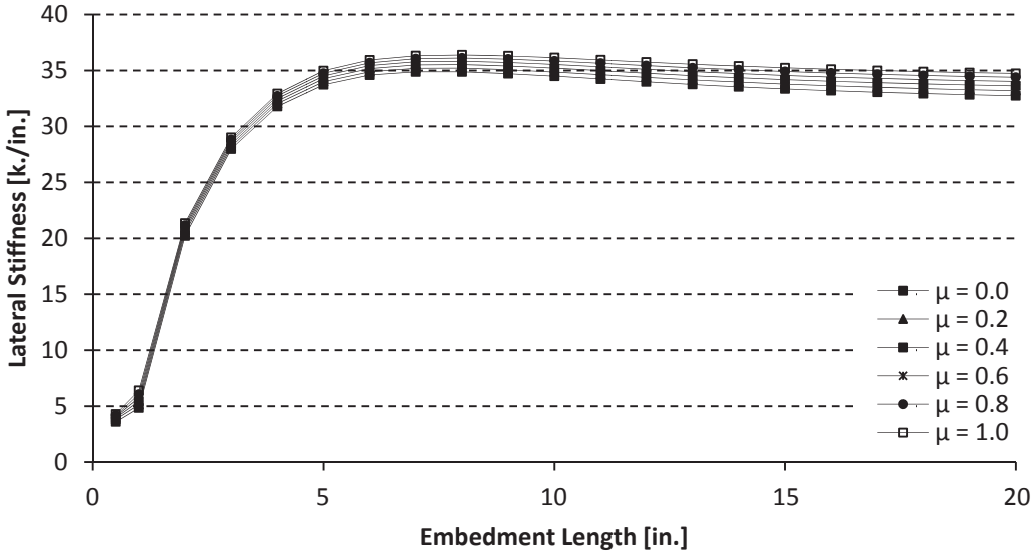


Figure 5-8: Coefficient of Friction (μ) Results

5.1.3 Tied (One-Part) Model

Perfectly tied models attenuated their forces relatively quickly to the surrounding foundation concrete, meaning that maximum stiffness was obtained at relatively shallow embedment depths.

Figure 5-9 shows the difference in stiffness values between the tied model and Barnwell's results. The perfectly bonded model would be very unconservative, and so it was decided not to use this bond type for further investigation. It would, however, offer a theoretical maximum stiffness available from the connection in the case of a perfect bond between steel and concrete. The presence of a theoretical upper limit that could not be exceeded suggests that modeling any connection as perfectly fixed, no matter how deeply embedded, could be unconservative.

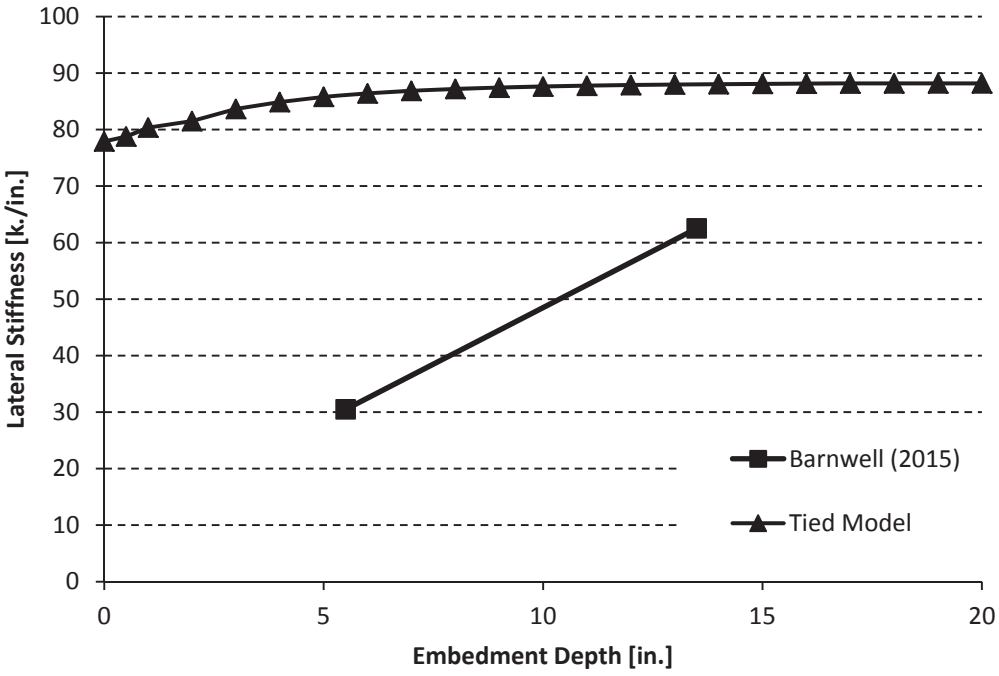


Figure 5-9: Comparison of Tied Model and Results from Barnwell (2015)

5.2 **Column Shape**

Four different column specimens were investigated: the W8X35 and W8X48 shapes investigated by Barnwell; a W14X176 shape; and a W24X76 shape. The W8X35 and W8X48 shapes represent light gravity columns. The W14x176 shape represents a typical choice of a heavy gravity column, while the W24x76 represents a typical specimen for a moment-resisting frame.

Figure 5-10 shows stiffness results for the chosen specimens. In this figure, as well as in all subsequent results, the results from cohesive zone models are shown, with a mesh size of 0.5, and a traction-separation value of $5 * 10^5$.

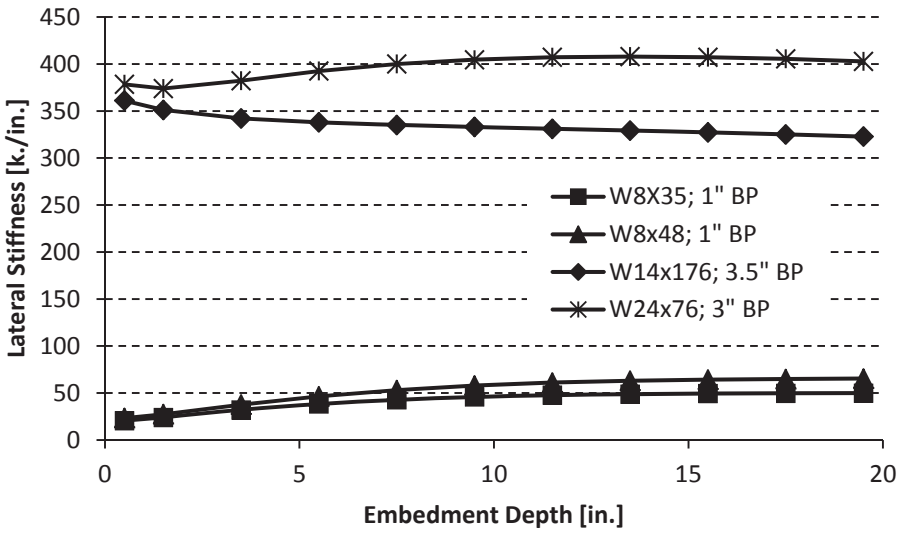


Figure 5-10: Column Shape Results; As Designed

The W14x176 and W24x76 baseplates were sized with the help of a licensed P.E.; design justifications can be found in Appendix D. The baseplate for the W14x176 shape is 24” x 24” x 3.5”. The baseplate for the W14x76 shape has dimensions of 20” x 34” x 3.0”.

In the as-designed specimens, the W8x35 and W8x48 shapes asymptotically approach a maximum stiffness value as embedment depth is increased. The W14x176 shape begins at a much greater stiffness value, and loses stiffness slightly,

A typical column design for W14x176 and W24x76 calls for thicker baseplates than the W8X35 and W8X48 shapes. However, the thickness of the baseplate can significantly affect the stiffness performance of the connection (see Section 5.4). Figure 5-11 shows all columns with an equal (1") thickness. This allows a more direct comparison between the shapes, with the difference in connection behaviors dictated only by the difference in column size and baseplate profile, not by the effects of changing baseplate thickness.

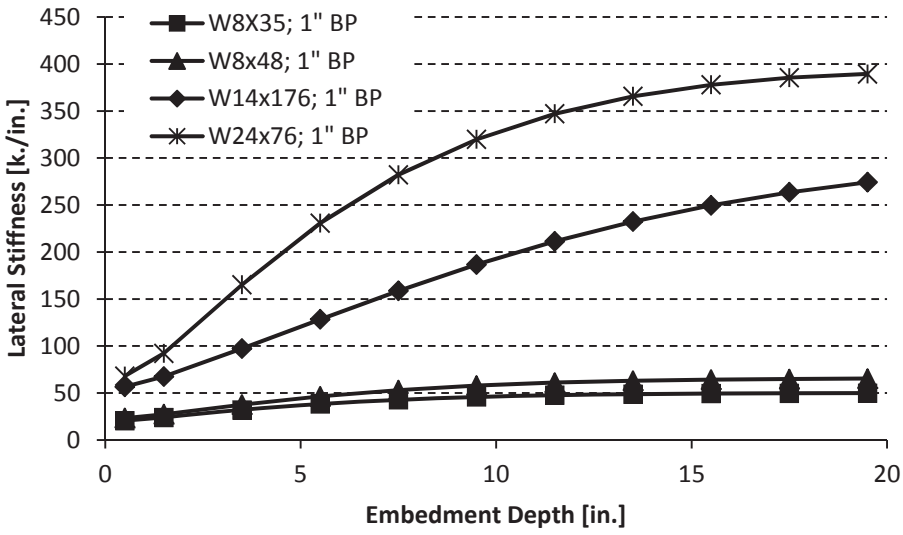


Figure 5-11: Column Shape Results; Equal Baseplate Thickness

Increasing column size increases the lateral stiffness available from connections at all depths. The contact-based models show that, at very low levels of embedment, the increase is

negligible; however, at embedment depths greater than 1.5 inches, the increase in stiffness becomes significant.

5.3 Concrete Modulus of Elasticity

Figure 5-12, Figure 5-13, and Figure 5-14 show the effects of varying the Young's Modulus of the concrete. In Figure 5-12, each curve represents a theoretical material with a modulus one order of magnitude higher than the line above it. The changing shapes of the stiffness line can be attributed to the changing stress distributions which occur as the relative moduli of the two materials change.

Figure 5-13shows results for a typical range of possible concrete modulus values. The ACI equation for the Young's Modulus of concrete, $E = 57000 * \sqrt{(f'c)}$, gives values in this range for normal-strength (f'c = 3000 psi) and higher-strength (f'c = 4000 psi) concrete. As can be seen, the relationship between concrete modulus and connection stiffness is not linear; an increase in concrete modulus of 66% (from $3 * 10^6$ psi to $5 * 10^6$ psi) typically results in an increase of approximately 20%. In the case of a 5.5" embedment, for example, the increase is 22.8%. The presence of high-strength grout beneath the baseplate may adjust the results from one curve to another, but the relatively minor increase in baseplate stiffness suggests the high-strength grout is unlikely to affect the stiffness values significantly. Also, these results suggest that modeling the presence of rebar would be unlikely to affect the results greatly, because it would not affect the effective Young's Modulus enough to change the stiffness more than a slight amount.

Figure 5-14 shows stiffness results for concrete that is approximately an order of magnitude less stiff. This would perhaps be equivalent to embedding the column in stiff soil, or

concrete which is of very low quality. The results are qualitatively similar to those of Figure 5-13, although the stiffness values reach about half those above. Also, it appears they reach their asymptotic maximum value more slowly than in cases with higher concrete stiffness.

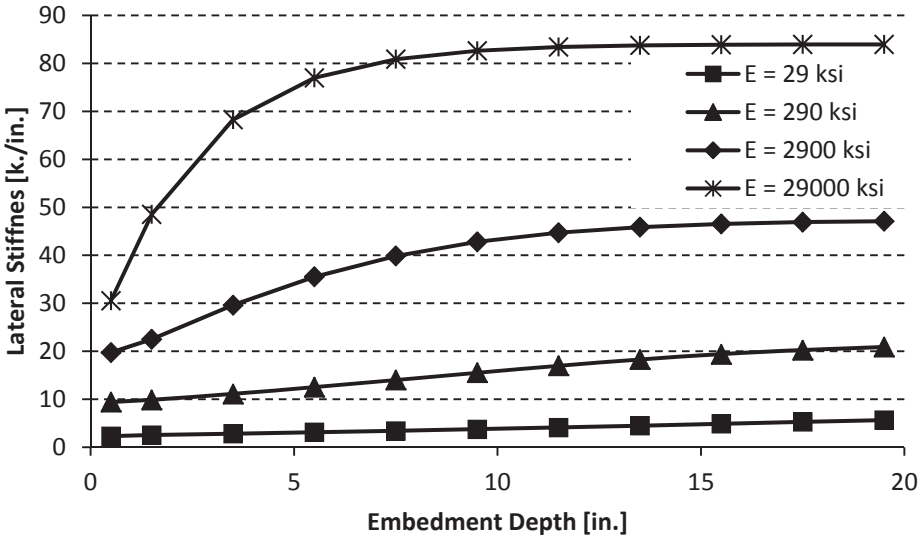


Figure 5-12: Modulus of Elasticity Results, W8x35 shape (Cohesive Zone Model)

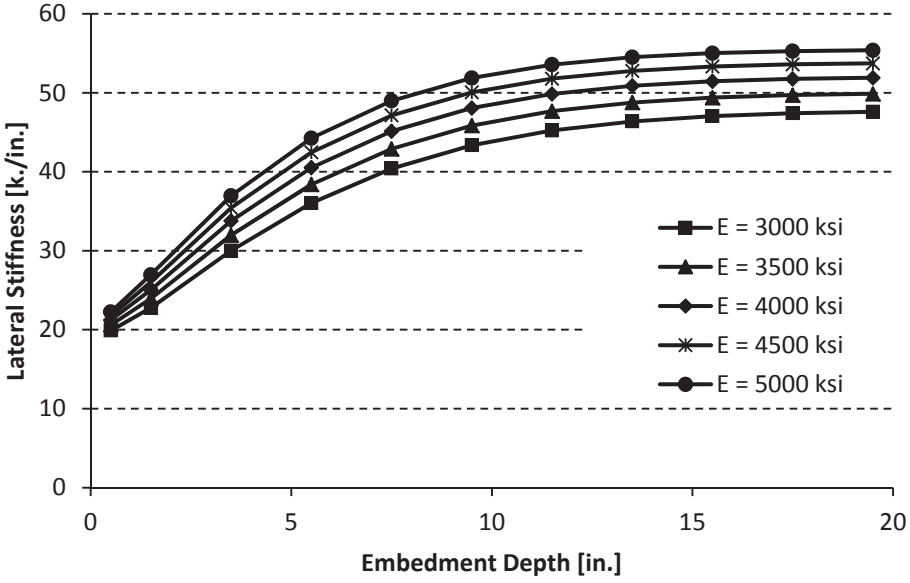


Figure 5-13: Modulus of Elasticity Results, W8x35 shape (Cohesive Zone Model)

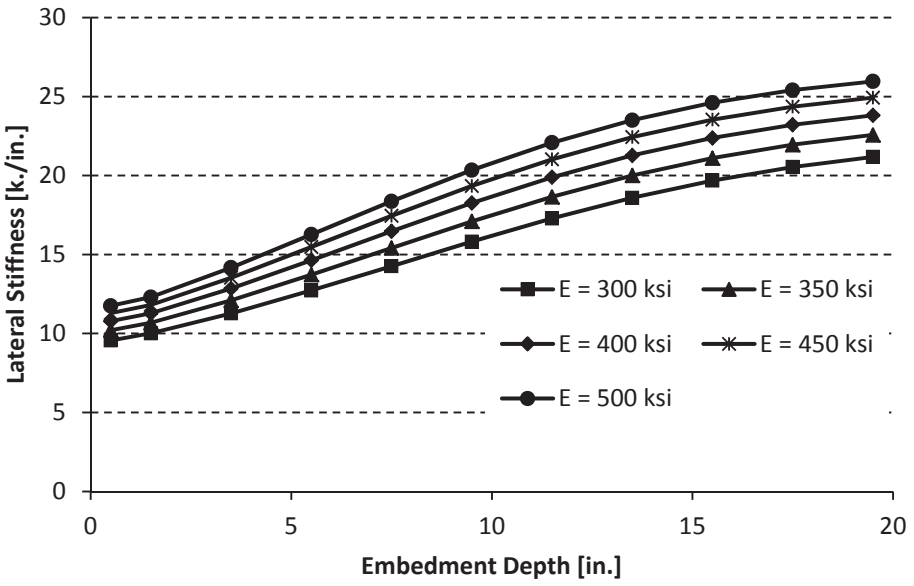


Figure 5-14: Modulus of Elasticity Results, W8x35 shape (Cohesive Zone Model)

5.4 Baseplate Geometry

Two baseplate geometry configurations were investigated, each with varying baseplate thickness values. In the first, the case of a square baseplate, the baseplate extends beyond the column profile as would be expected in a typical column. In the second, that of a reduced baseplate, the baseplate's dimensions do not extend beyond the column profile.

5.4.1 Square Baseplate

In the case of W8x35 and W8x48 shapes, the baseplate is 13" square, as in Barnwell (2015). In the case of a W14x176 shape, the baseplate is 24" inches. This was designed by a licensed P.E. according to typical design processes (see Appendix D). In each case, baseplate thickness values of 0.5", 1.0" (default), 2.0", and 3.0" were considered. Also, since the design of

the W14x176 shape called for a baseplate thickness of 3.5", a 3.5" baseplate was included in each set of column specimens to facilitate direct comparisons.

Figure 5-15, Figure 5-16 and Figure 5-17 show stiffness results for W8x35, W8x48, and W14x176 shapes, respectively. The responses of W8x35 and W8x48 shapes appear qualitatively similar, with slightly increased stiffness for the W8x48 specimens. The response of the W14x176 shape, however, is qualitatively different. This suggests that different shape families may behave differently.

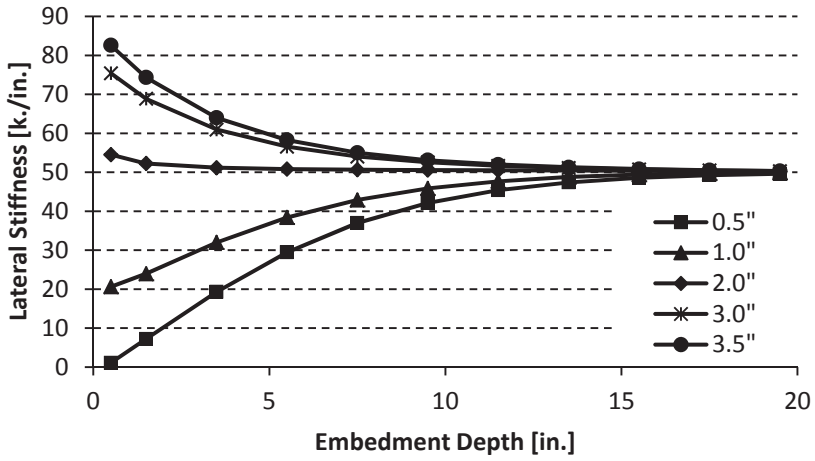


Figure 5-15: W8x35; Varying Baseplate Thickness

Counterintuitively, baseplates below a certain threshold show a decrease in stiffness as embedment depth increases. It is believed that the increasing embedment depth is causing less force to be transferred through the baseplate, and more through the bearing mechanism, which is the less rotationally stiff force transfer mechanism. This mirrors the slight decrease in stiffness

reported by Grilli (2015) which was obtained with an increasing embedment depth. This also suggests that increasing baseplate thickness is a relatively simple way to increase connection stiffness at shallow embedment depths. In exposed connections, with no embedment, stiffness is very sensitive to baseplate thickness. At shallow embedments, baseplate deformations reduce connection stiffness, as suggested by Cui et al. (2009). At deeper embedments, however, the surrounding concrete stiffens the baseplate area, and reduces the deformation in the baseplate itself.

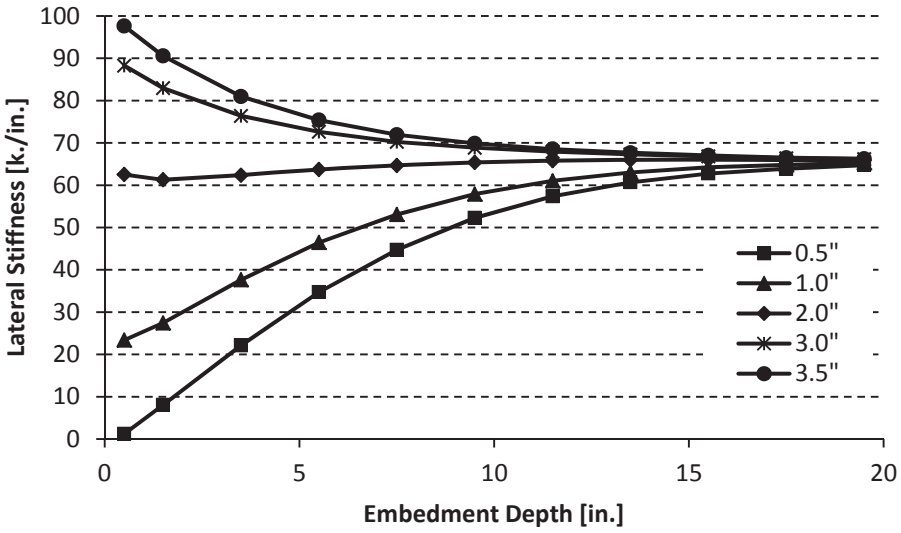


Figure 5-16: W8x48; Varying Baseplate Thickness

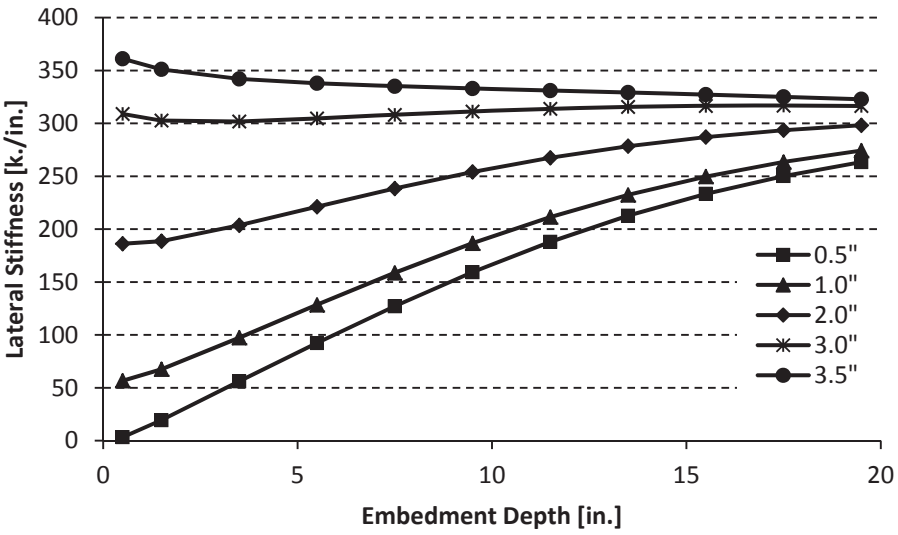


Figure 5-17: W14x176; Varying Baseplate Thickness

5.4.2 Reduced Baseplate

The reduced baseplate was that of a column specimen with baseplate dimensions exactly circumscribed by the perimeter of the column. This was of interest in comparing with the concurrent research of Tryon (2016), who postulated that the contribution of the baseplate area beyond the area circumscribed by the column, was of negligible effect. Also, quantifying the stiffness from these reduced sections may allow column designers to specify baseplates with less material in columns that are governed by stiffness considerations (rather than stress or column uplift considerations). Figure 5-18, Figure 5-19, and Figure 5-20 show results for W8x35, W8x48, and W14x176 shapes, respectively, with reduced baseplates.

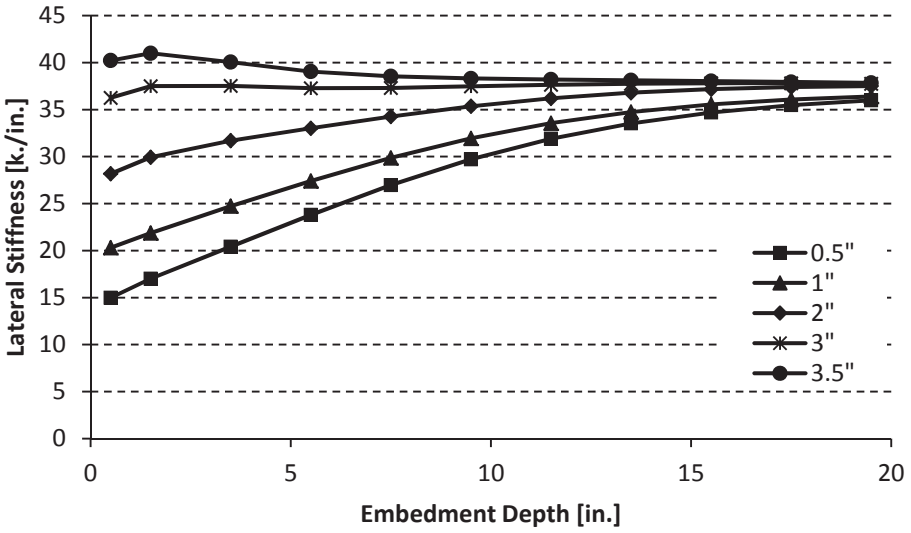


Figure 5-18: W8x35 Results, Reduced Baseplate

The effect of reducing the baseplate profile depends on the thickness of the baseplate itself. For very thin baseplates (0.5”), reducing the baseplate actually increases the connection stiffness, presumably because the removed area was extremely flexible. Baseplates of normal thickness (1.0”) have approximately equal stiffness with either profile. As baseplate thickness increases beyond 1.0”, however, the connection fails to increase its stiffness at lower embedment depths as quickly as in the case of full-sized baseplates.

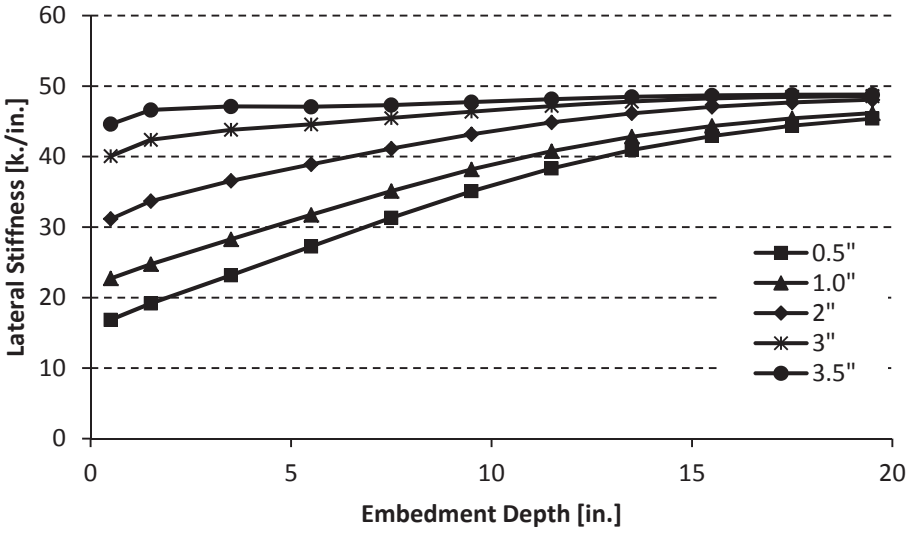


Figure 5-19: W8x48 Results, Reduced Baseplate

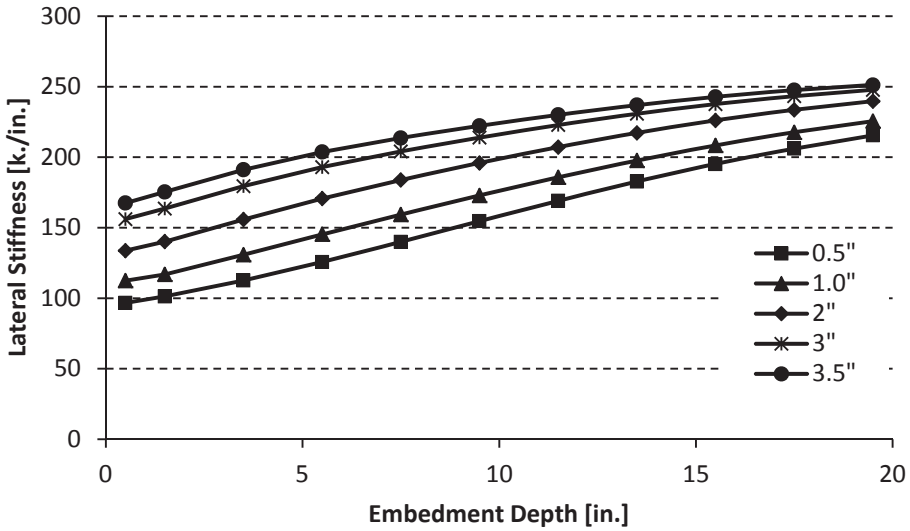


Figure 5-20: W14x176 Results, Reduced Baseplate

5.5 Cantilever Height

The effects of varying cantilever height, Z, were studied. It was found that reducing the cantilever height reduced the rotational stiffness of the connection. However, in the range of

values of greatest interest, the difference was slight. For instance, for Z = 60” to Z = 90”, the range in which the testing was performed, the difference was less than 10% the values of both specimens studied (5.5” and 13.5”). As the cantilever height increases, the ratio of shear deformation to rotational deformation increases. This is thought to increase the shear deformation in the column, reducing the rotational stiffness. As the cantilever height increases, however, the effects of shear deformation become negligible, and the connection behaves closely to a linear rotational spring of constant stiffness. For the cantilever heights studied in this research, which represent typical story heights, these values are relatively close to constant.

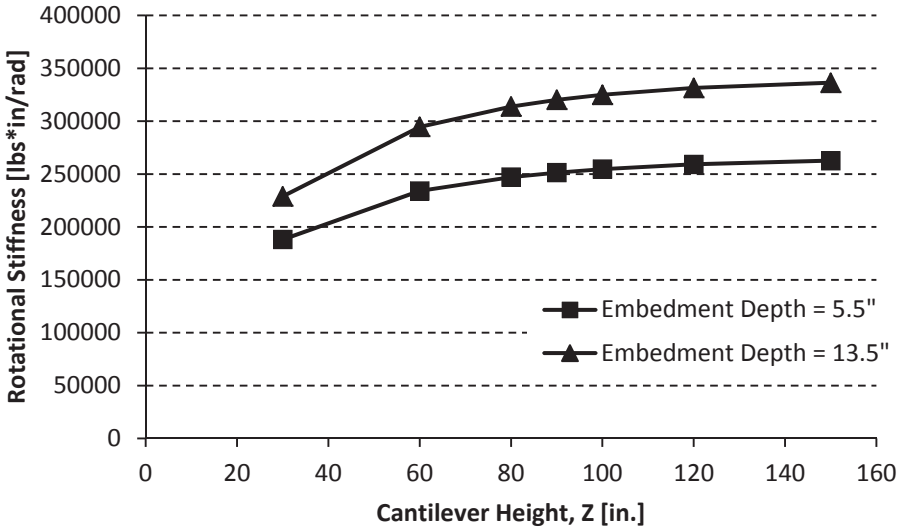


Figure 5-21: W8x35 Results, Varying Cantilever Height

5.6 **Axial Load**

The effects of axial load on connection stiffness were investigated. Figure 5-22 shows axial load decreasing the connection stiffness embedment depths greater than 3.5 inches. The cohesive elements are experiencing shear stresses many times higher than those experienced

under normal loading conditions. This may be leading, in turn, to decreased resistance to applied lateral loads at higher deformations.

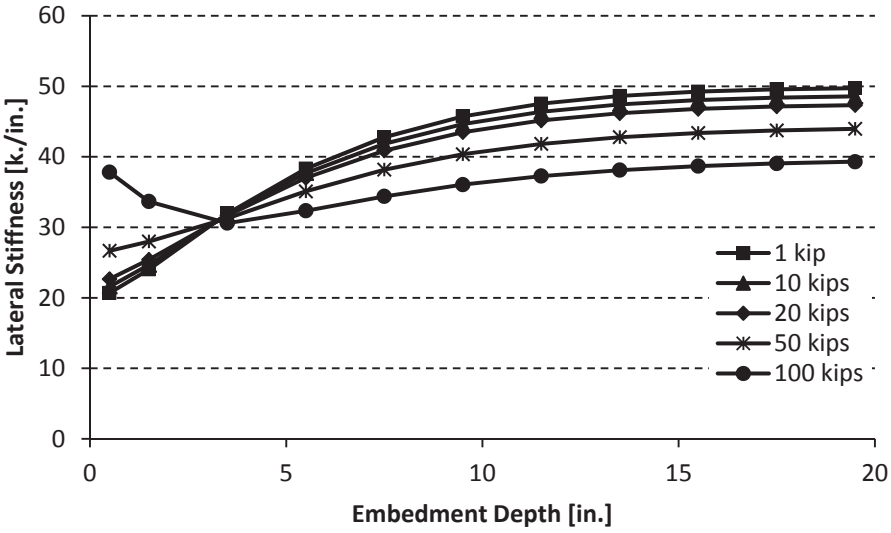


Figure 5-22: Effects of Axial Load, W8x35 (Cohesive Zone Based Model)

5.7 Column Orientation

A significant decrease in stiffness was observed when the column was oriented such that the weak-axis was resisting the bending loads. Figure 5-23 shows the results of Abaqus models for weak axis bending. Reasonable agreement with experimental data was obtained. These results suggest that the maximum stiffness value will be obtained at much lower embedment depths than strong-axis specimens will.

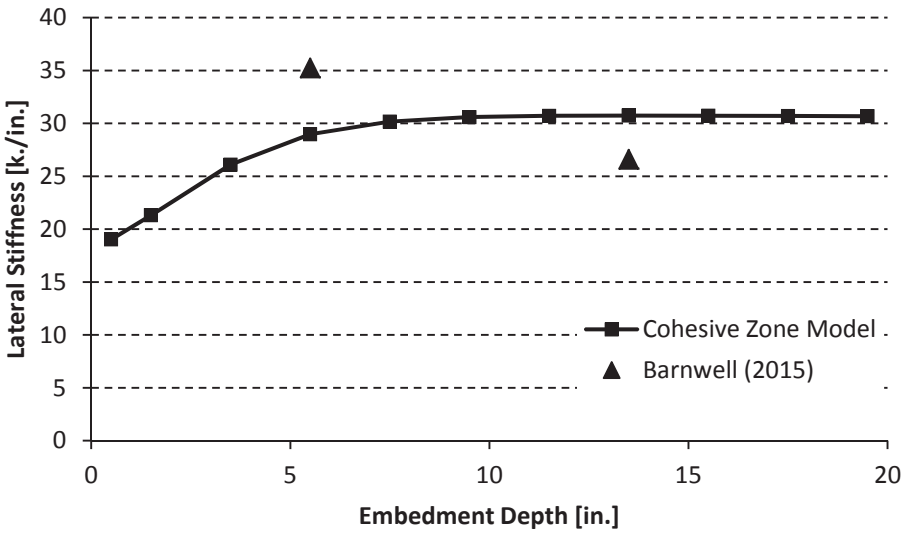


Figure 5-23: W8x35, Weak Axis Bending

5.8 Phase III Predictions

In Phase III of the testing program currently underway at BYU, additional laboratory specimens will be tested. Table 5-2 shows the test matrix of planned specimens.

Table 5-2: Phase III Test Matrix

Specimen Name	Column Size	Base Plate Thickness (in)	Base Plate ASTM grade	Shear Lug	Anchor Bolts Qty	Anchor Bolts DIA (in)	Anchor Bolts Grade	Base Depth (in)	Block-out Depth (in)
D1	W14x53	2.25	A36	Yes	8	1	F1554 Gr 36	24	0
D2	W14x53	2.25	A36	Yes	8	1	F1554 Gr 36	24	8
D3	W14x53	2.25	A36	Yes	8	1	F1554 Gr 36	24	16
D4	W14x53	1.5	A36	Yes	4	1	F1554 Gr 36	24	16
F1	W10x77	3	A36	Yes	8	1 1/8	F1554 Gr 36	24	0
F2	W10x77	3	A36	Yes	8	1 1/8	F1554 Gr 36	24	8
F3	W10x77	3	A36	Yes	8	1 1/8	F1554 Gr 36	24	16
F4	W10x77	2	A36	Yes	4	1 1/8	F1554 Gr 36	24	16

Finite element models were created of these specimens. The modeling was done with E_{conc} = 3.6E6, corresponding to f'_c = 4,000 psi. The depth of concrete below the column was 24". The scripts were changed to calculate the embedment depth from the bottom of the baseplate, instead of the top of it; this accommodated the varying baseplate thicknesses more easily. All other parameters were left unchanged from default values. Anchor bolts, shear lugs, and other construction details (see Section 3.5) were neglected. Also, specimens D1 and F1 were not tested, since the model is not equipped to easily handle exposed baseplate connections. Table 5-3 summarizes the expected stiffness values, which range from 111.8 k/in (Test F4), to 146.0 k/in (Test D3).

Table 5-3: FEA Results for Phase III Specimens

Specimen Name	Column Size	Baseplate Thickness [in.]	Block-Out Depth [in.]	Predicted Displacement [in.]	Predicted Stiffness [k./in.]	Predicted Rotational Stiffness [k.*in./rad]
D1	W14X53	2.25	0	--	--	--
D2	W14X53	2.25	8	0.01830	136.7	8.80E+05
D3	W14X53	2.25	16	0.01783	146.0	9.40E+05
D4	W14X53	1.5	16	0.01802	142.0	9.14E+05
F1	W10X77	3	0	--	--	--
F2	W10X77	3	8	0.02080	129.2	8.32E+05
F3	W10X77	3	16	0.02146	119.0	7.66E+05
F4	W10X77	2	16	0.02200	111.8	7.20E+05

6 CONCLUSIONS

The objective of this study was to investigate the effects of key input variables on the rotational stiffness of shallowly embedded connections, using finite element simulations. These key variables include blockout/embedment depth, baseplate geometry, column size/orientation, grout/concrete modulus, and applied axial loading.

All finite element models were created in Abaqus 6.14. Two parts were created, meshed, and assigned material properties. Each part was instanced and was assigned constraints, contact properties, boundary conditions, and loads that represented the original laboratory conditions; cohesive zone modeling represented the bond between the concrete and steel. The model was then submitted to Abaqus/Standard for processing. After processing, the displacement at the point of applied load was queried, and the connection stiffness was calculated. Modeling was automated with the use of Python scripts.

The behavior of the connection is highly sensitive to the contact method used in the finite element solver. Three different connection types were investigated: a tied or one part model; a contact-based model; and a cohesive-zone based model. The tied model gives unrealistically high values connection stiffness values. Although this provides a theoretical upper bound on stiffness values, it does not accurately reflect the expected connection behavior. A contact-based model, using a hard pressure-overclosure relationship, gives stiffness values that are reasonably

close to experimental results, but do not show increasing stiffness with increasing embedment depth. Cohesive zone-based models showed stiffness values increasing with embedment depth. By calibrating the pressure-overclosure relationship in cohesive zone models, the elastic stiffness values were simulated to within 27% error.

The effects of baseplate geometry on stiffness decrease with increasing embedment depth. As the embedment depth increases, the overturning moment is transferred increasingly via the column bearing on concrete, and less through the baseplate bearing on the concrete. Therefore, the baseplate's contribution to the stiffness decreases. This effect is greater for thicker baseplates. In the case of very thick baseplates, increasing the baseplate's embedment depth may actually cause the connection to lose stiffness; as stiffness from the baseplate bearing mechanism decreases, the stiffness afforded by the column bearing mechanism does not increase to match. This agrees with the observations from Grilli (2015) of a decreased stiffness with increasing embedment length.

It was found that the rotational stiffness of the connections does not vary with the cantilever height for typical story heights. Tests were performed for heights from 30" to 150". Although rotational stiffness is decreased for very short columns, the rotational stiffness is not affected greatly at heights of 80" or more. This means that the method of modeling connections as rotational springs acting at the top of the slab is viable as long, provided the cantilever height is large enough.

6.1 **Future Research**

Significant possibilities exist for future research to both broaden and to refine this investigation. These possibilities exist in the realm of both additional numerical modeling, and additional physical testing to verify the results of the numerical models.

Additional FEA models can be developed which would serve to expand and broaden this research. It would be useful to discover the response of the connection once it has left the linear region of its response, both in terms of stiffness and strength. Further research could be performed to illuminate the inability of current models to match the predicted stiffness values for deeper embedment depths with strong-axis bending. The nonlinear responses in the early stages of many of Barnwell's specimens suggest that there may be material nonlinearity even at lower loading. Abaqus has significant capabilities for nonlinear loading patterns which were not explored in this research. In addition to relatively simple elasto-plastic models, more exotic material properties exist which could model concrete more precisely. For instance, "Concrete Damaged Plasticity", "Concrete Smeared Cracking", and "Cracking Model for Concrete" models accept a variety of inputs that correspond to the precise physical properties of the concrete, and could be carefully calibrated to provide results that match experimental data. With proper calibration, these models could generate useful data beyond the initial tangent stiffness, including stiffness characterizations of the connection into the nonlinear, post-cracking regime.

User-defined interaction types can be created in Abaqus, to model more exotic connection types. A connection type that behaves differently in tension than in compression – or that allows separation after a given amount of tensile force is applied – could be used to generate more accurate results. These could help model the physical and chemical adhesion between the steel and concrete more precisely.

REFERENCES

Abaqus (2014), "Abaqus Theory Guide", Abaqus Software, Ver. 6.14, Dassault Systemes Simulia Corp., Providence, Rhode Island.

ACI Committee, American Concrete Institute, & International Organization for Standardization. (2008). Building code requirements for structural concrete (ACI 318-08) and commentary. American Concrete Institute.

AISC. (2010). Seismic Provisions for Structural Steel Buildings (ANSI,AISC 341-10), American Institute of Steel Construction, Inc., Chicago, IL.

ASCE Guidelines (1994). "Guidelines for Design of Joints Between Steel Beams and Reinforced Concrete Columns," Journal of Structural Division, ASCE, Vol. 120(8), pp. 2330-2357.

Castilla, F., Martin, P., & Link, J. (1984*). Fixity of members embedded in concrete (*No. CERL-TR-M-339). CONSTRUCTION ENGINEERING RESEARCH LAB (ARMY) CHAMPAIGN IL.

Barnwell, N. V. (2015). Experimental Testing of Shallow Embedded Connections Between Steel Columns and Concrete Footings. Brigham Young University. Master's Thesis.

Cordova, P.P. and Deierlein, G.G. (2005). "Validation of the Seismic Performance of Composite RCS Frames: Full-Scale Testing, Analytical Modeling, and Seismic Design," Technical Report 155, Blume Earthquake Engineering Center.

Cui, Y., Nagae, T., Nakashima, M. (2009). "Hysteretic Behavior and Strength Capacity of Shallowly Embedded Steel Column Bases." Journal of Structural Engineering, ASCE Vol. 135, No. 10, October 2009, pp. 1231-1238.

Davis, B. (2011). "Re: Ask aisc| anchor rods, base plates and embedded plates," personal email communication with P. Richards, Aug. 15, 2011.

Deierlein, G.G., Sheikh, T.M., Yura, J.A., and Jirsa, J.O. (1989). "Beam-Column Moment Connections for Composite Frames: Part 2," Journal of Structural Engineering ASCE, Vol. 115, November 1989, pp.2877-2869.

Eastman, R. S. (2011). "Experimental investigation of steel pipe pile to concrete cap connections." Master's thesis, Brigham Young University, June.

Fisher, J.M. and Kloiber, L.A. (2006), "Base Plate and Anchor Rod Design," 2nd Ed., Steel Design Guide Series No. 1, American Institute of Steel Construction, Inc., Chicago, IL.

Grauvilardell, J. E., Lee, D., Hajjar, J. F., & Dexter, R. J. (2005). Synthesis of Design, Testing, and Analysis Research on Steel Column Base Plate Connections in High-seismic Zones. Department of Civil Engineering, University of Minnesota.

Grilli, D.A., and Kanvinde, A.M. (2013). "Special Moment Frame Base Connection: Design Example 8," 2012 IBC SEAOC Structural/Seismic Design Manual, Volume 4, Examples for Steel-Frame Buildings, 255-280.

Gomez, I. R. (2010). Behavior and design of column base connections. University of California, Davis. Dissertation.

Hillerborg, A., Modéer, M., & Petersson, P. E. (1976). Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements. Cement and concrete research, 6(6), 773-781.

Jordan Jr, S. (2010). Finite element simulations of exposed column base plate connections subjected to axial compression and flexure. University of California, Davis. Master's Thesis.

Julander, J. L. (2009). Finite element modeling of full depth precast concrete transverse bridge deck connections. Utah State University. Master's Thesis.

Kanvinde, A.M., Grilli, D.A., and Zareian, F. (2012). "Rotational Stiffness of Exposed Column Base Connections – Experiments and Analytical Models," Journal of Structural Engineering, ASCE, 138(5), 549-560.

Khodaie, S., Mohamadi-Shooreh, M. R., & Mofid, M. (2012). Parametric analyses on the initial stiffness of the SHS column base plate connections using FEM. Engineering Structures, 34, 363-370.

Malley, J. (2011). "Re: Column-to-footing questions," email communication with P. Richards, Aug. 5, 2011.

Marcakis, K., and Mitchell, D. (1980). "Precast concrete connections with embedded steel members," *Precast/Prestressed Concrete Institute Journal,* 56(4), pp. 88–116. 10, 11

Morino, S., Kawaguchi, J., Tsuji, A., and Kadoya, H. (2003). "Strength and Stiffness of CFT Semi-embedded Type Column Base," Proceedings of ASSCCA 2003 Conference, Sydney, Australia, A. A. Balkema, Sydney, Australia.

Motter, C.J., (2014). Large-scale Testing of Steel-reinforced Concrete (SRC) Coupling Beams Embedded into Reinforced Concrete Structural Walls. Dissertation, University of California, Los Angeles. Ann Arbor: ProQuest/UMI, 2014. (Publication No. 3629336).

Myers, A.T., Kanvinde, A.M., Deierlein, G.G., and Fell B.V. (2009). "Effect of Weld Details on the Ductility of Steel Column Baseplate Connections," Journal of Constructional Steel Research, Elsevier, Vol. 65, No. 6, June 2009, pp. 1366-1373.

PCI Industry Handbook Committee. (1999). PCI Design Handbook, 5th Edition, Precast/Prestressed Concrete Intitute, Chicago, IL.

Pertold J., Xiao, R. Y., and Wald, F. (2000a). "Embedded Steel Column Bases – I. Experiments and Numerical Simulation," Journal of Constructional Steel Research, Vol. 56, pp. 253-270.

Pertold, J., Xiao, R. Y., and Wald, F. (2000b). "Embedded Steel Column Bases – II. Design Model Proposal," Journal of Constructional Steel Research, Vol. 56, pp. 271-286.

Rabbat, B. G., & Russell, H. G. (1985). Friction coefficient of steel on concrete or grout. *Journal of Structural Engineering*, *111*(3), 505-515.

Richards, P. W., Rollins, K. M., & Stenlund, T. E. (2010). Experimental Testing of Pile-to-Cap Connections for Embedded Pipe Piles. Journal of Bridge Engineering, 16(2), 286-294.

Serpier, R., Varricchio, L., Sacco, E., & Alfano, G. (2014). Bond-slip analysis via a cohesive-zone model simulating damage, friction and interlocking. Frattura ed Integritá Strutturale, (29), 284.

Shahrooz, B. M., Remetter, M. E., and Qin, F. (1993), "Seismic Design and Performance of Composite Coupled Walls," Journal of Structural Engineering, ASCE, Vol. 119, No. 11, pp. 3291–3309, Reston, VA.

Shama, A. A., Mander, J. B., & Aref, A. J. (2002). Seismic performance and retrofit of steel pile to concrete cap connections. Structural Journal, 99(1), 51-61.

Sheikh, T.M., Deierlein, G.G., Yura, J.A., and Jirsa, J.O. (1989). "Beam-Column Moment Connections for Composite Frames: Part 1," Journal of Structural Engineering ASCE, Vol. 115, November 1989, pp. 2858-2876.

Song, S. H., Paulino, G. H., & Buttlar, W. G. (2006). Simulation of crack propagation in asphalt concrete using an intrinsic cohesive zone model. Journal of Engineering Mechanics, 132(11), 1215-1223.

Sosa, J. C. (2010). Modeling of the nonlinear interface in reinforced concrete. International Journal for Computational Methods in Engineering Science and Mechanics, 11(3), 157-161.

Thambiratnam, D.P., and Paramasivam, P. (1986). "Base Plates Under Axial Loads and Moments," Journal of Structural Engineering, ASCE, Vol. 112, No. 5, May 1986, pp. 1166-1181.

Tremblay, R., Timler, P., Bruneau, M., and Filiatrault, A., (1995). "Performance of Steel Structures During the 1994 Northridge Earthquake," Canadian Journal of Civil Engineering, Vol. 22, pp. 338-360.

Xiao, Y., Wu, H., Yaprak, T. T., Martin, G. R., & Mander, J. B. (2006). Experimental Studies on Seismic Behavior of Steel Pile-to-Pile-Cap Connections. Journal of Bridge Engineering, 11(2).

Zareian, F., and Kanvinde, A.M. (2013). "Effect of Column Base Flexibility on the Seismic Safety of Steel Moment Resisting Frames," Earthquake Spectra, Earthquake Engineering Research Institute, 29(4), 1-23.

APPENDIX A – MESH CONVERGENCE AND OTHER VALIDATION STUDIES

This appendix will detail the results of the mesh convergence study and other supporting validation studies that were run in conjunction with this research. A number of key assumptions and simplifications were made in the model, and were verified through these studies.

Mesh Convergence Study

A mesh convergence study was performed to verify the accuracy of the numerical solutions. Unexpected behavior occurred, in that it appeared to reach convergence at a lower mesh density, but instead began converging to a different result after a certain threshold of mesh density was reached.

The mesh convergence study was performed with a W8X35 shape, with embedment depths of 0.5, 1.5, and 3.5. These shallow embedment depths were chosen because the small amount of embedment depth, and the behavior of the embedment concrete above the baseplate, would be more sensitive to numerical irregularities, due to their small volume. Also, the smaller embedment depths meant that fewer nodes were required in the model, leading to faster compute times. The foundation size was reduced by half in both the x and y directions (~4x total volume reduction) to make the total number of elements more manageable; preliminary analysis (not

included) suggested that this would reduce the total overall stiffness effects by less than 2%, primarily because the zone of significant stress / deformation did not extend that far.

The mesh sizes studied were 2.00, 1.50, 1.00; 0.75; 0.50; 0.25; and 0.20. The values for 0.20 were taken as the exact solution, as further mesh refinement was judged to be impossible. The Mary Lou Fulton supercomputer at Brigham Young University was used to run the most refined mesh studies.

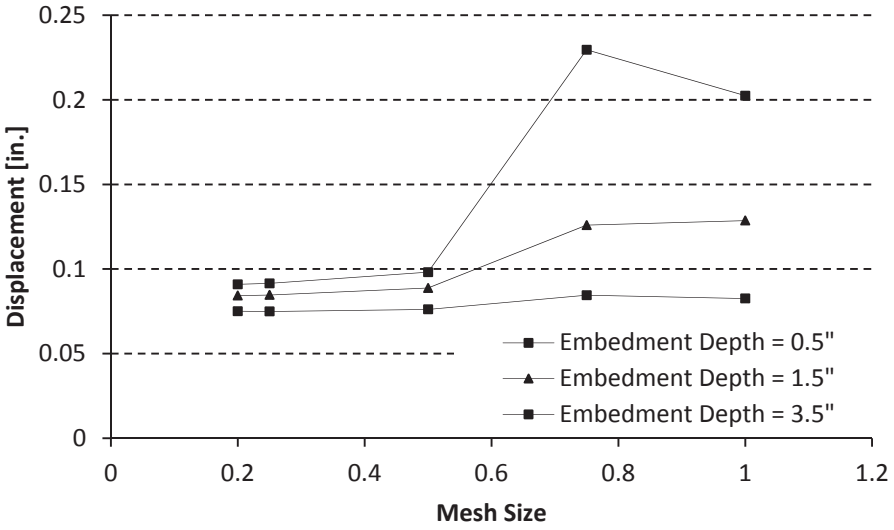


Figure A-0-1: Mesh Convergence Results, Displacement Values

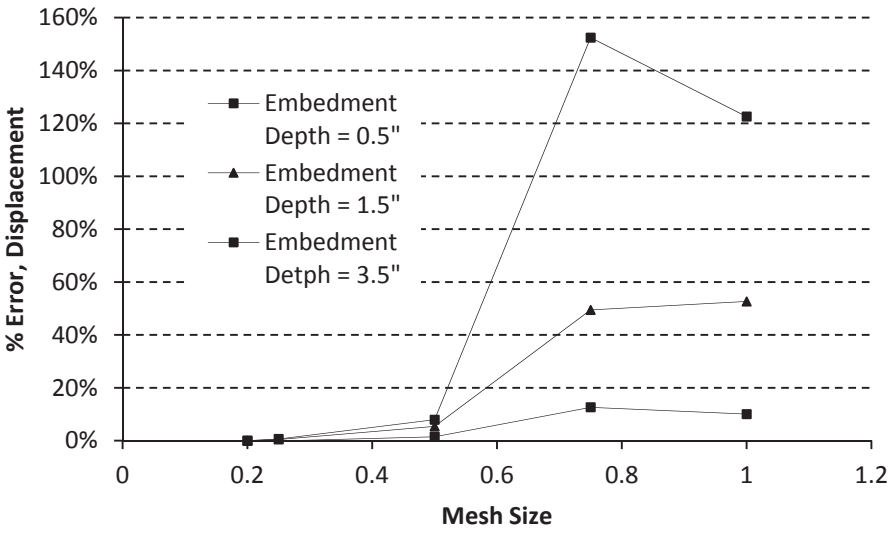


Figure A-0-2: Mesh Convergence Results, % Error, Displacement

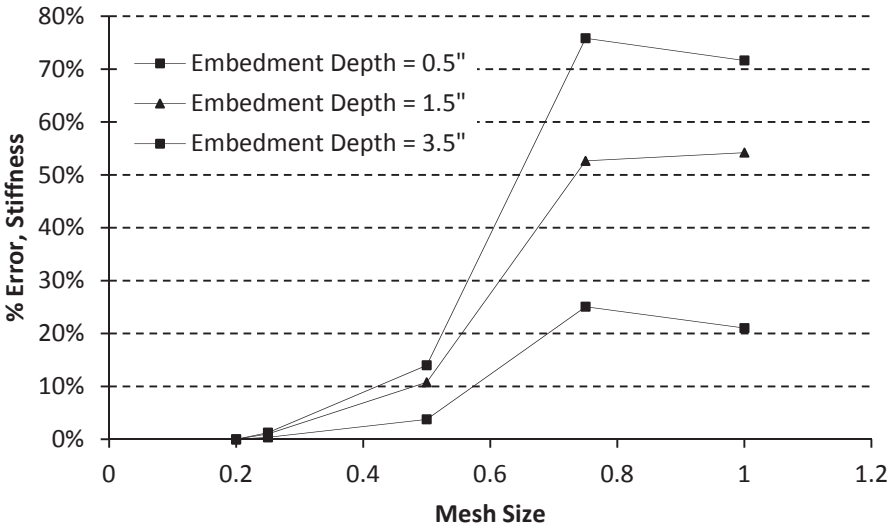


Figure A-0-3: Mesh Convergence Results, % Error, Stiffness

Given that the finite element models failed to reproduce the stiffness values from Barnwell (2015) to within +/- 27%, a maximum stiffness error of 14% was judged to be acceptable. Most shapes will exhibit significantly less error, since shapes with greater embedment depth exhibited

less error, and the case of maximum error was a specimen with embedment depth of 0.5". In the case of 3.5" embedment, for example, the stiffness error is less than or equal to 4%.

Recall that, due to Abaqus' meshing algorithms, many elements had dimensions that varied from a 1" cube. Although divisions were made to minimize this, it was impossible in practice to ensure that every element was an exact cube.

Also, it was attempted to create a non-uniform mesh in the foundation part so as to reduce the computational time, and minimize the increase in error. However, no method was determined that could be shown to reliably maintain solution accuracy and decrease computational load significantly.

Foundation Size

A study of connection stiffness' sensitivity to slab size (see Figure A-0-4) suggested that the error associated with reducing the foundation size would be very slight. It was deemed prudent to accept this error in order to reduce the computational demand associated with a relatively fine mesh (0.5" typical).

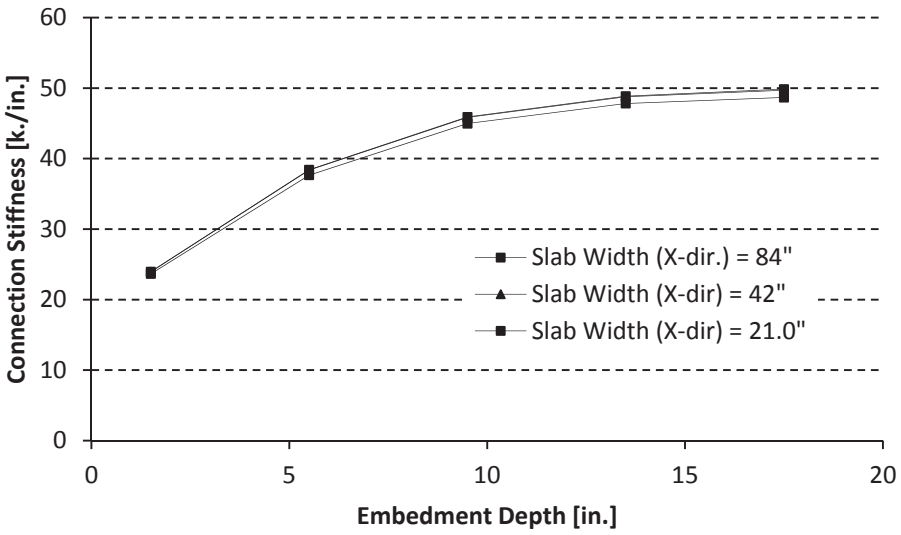


Figure A-0-4: Sensitivity of Results to Slab Dimensions

Model Linearity with Respect to Applied Force (Contact Based Model)

It was necessary to verify that the models do behave in a linearly elastic manner, as was assumed, with respect to applied force. This validation study was performed with a contact-based model. Although it behaved in a linearly elastic manner at very low loads (10, 100 lbs. and 1000 lbs.), it saw a noticeable increase in stiffness with increasing load after that (10,000 lbs). See Figure A-0-5 and Figure A-0-6. Therefore, it was determined that the results from these studies would likely need to be refined to accurately determine secant stiffness values for applied loads greater than 1,000 kips.

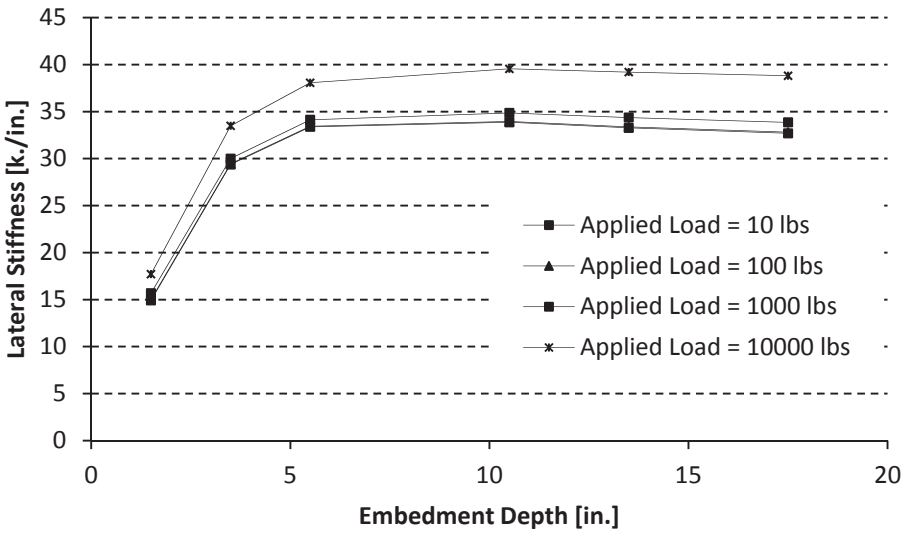


Figure A-0-5: Model Linearity with W8x35 Shape

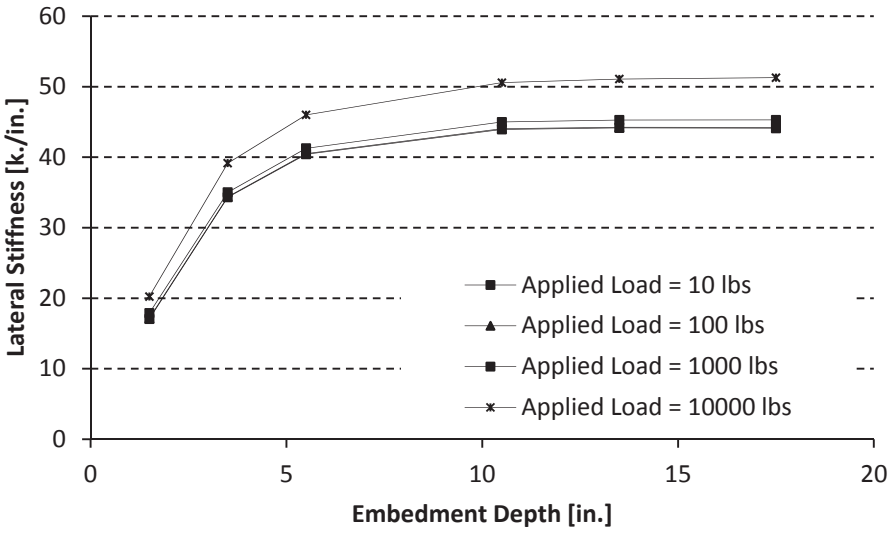


Figure A-0-6: Model Linearity with W8x48 Shape

It should be noted that these models were created with boundary conditions and friction (μ) values that varied slightly from those used in the final models. The boundary condition set was "Top" (see Boundary Condition study in this Appendix) instead of "Bottom"; the model was a

friction-based model instead of a contact-based model, with a μ (coefficient of friction) of 0.57, not 0.50. However, the conclusions reached are deemed to be generally applicable.

Boundary Conditions

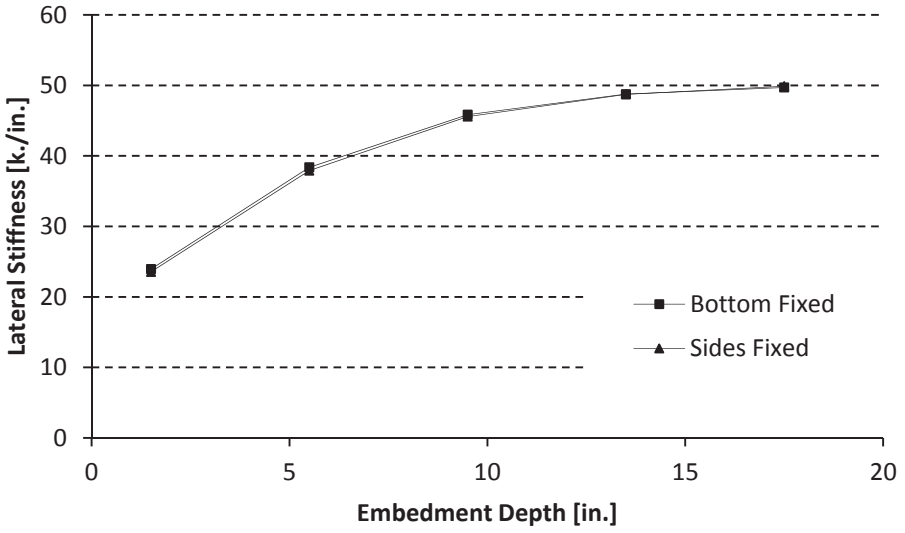


Figure A-0-7: Effect of Boundary Conditions

The actual arrangement of boundary conditions was found to have no significant effect on the overall connection stiffness. This is believed to be because the stress levels and deflections at the boundary conditions were so low as to render the difference in fixity conditions, inconsequential.

APPENDIX B – 2 DIMENSIONAL MODEL RESULTS

A series of two dimensional models was generated in Abaqus to study the behavior of a perfectly tied connection in an effectively infinite foundation.

The model was generated using several simplifying assumptions. First, the model was created in two dimensions, thus assuming a unit thickness and plane strain conditions. Also, a no-slip boundary between the column and the continuum was created. No stiffening base-plate was included. The continuum part was made sufficiently large that it simulated an infinite continuum. Thus, no (or negligible) additional stiffness was caused by boundary conditions at the continuum edge. The model was then subjected to a rigorous set of analyses and verification studies.

It was found that flexural stiffness available in shallow-embedded connections will asymptotically approach an upper limit as the embed depth increases. The value of this upper limit of stiffness – as well as the rate at which it increases – is governed by the geometry of the column, and the material properties of the column and continuum. It was also found that a relatively small increase in embed depth can greatly increase flexural stiffness.

Abaqus was used to investigate the case of a steel column, embedded a finite distance into a medium (which we refer to as a "continuum") of theoretically infinite extent. In practice, the continuum would typically represent either a foundation or a pile cap, made of normal-

strength, unreinforced concrete. However, this paper investigates cases where the Modulus of Elasticity of the material is significantly higher than that of concrete.

Model

In the Abaqus GUI, a 2-dimensional solid planar part was created. The single part represented both the column and the continuum. The part was then divided into two a column section and a continuum section. This was done to ensure a perfect bond between both the column and continuum (no relative displacement between nodes at the boundary).

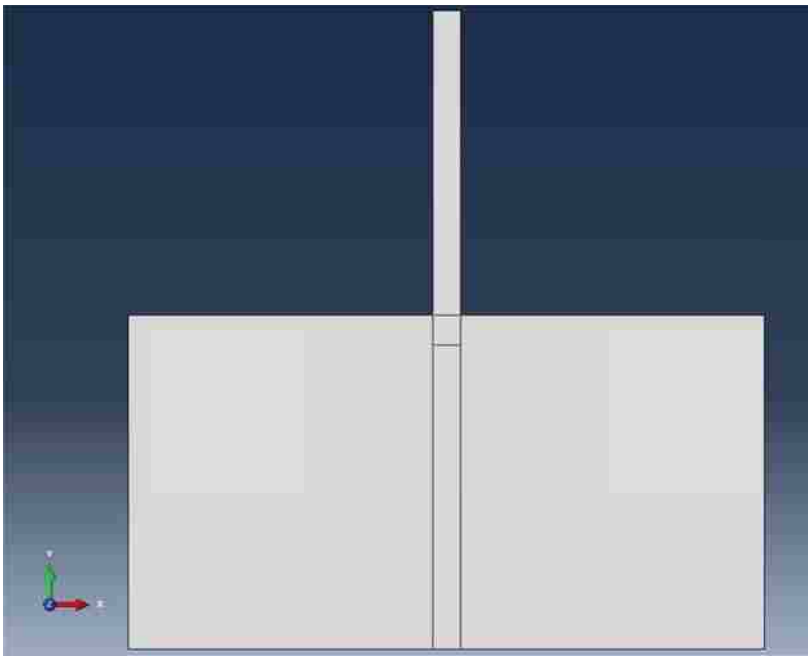


Figure B-0-1: Typical Model Configuration

The column section was divided into two sections: an embedded section, and a protruding length. The protruding length was assigned a Young's Modulus several orders of magnitude higher than the modulus of either the rest of the beam or the continuum. This was done to

simulate a perfectly stiff protruding section of beam. This was done so that, when we applied the lateral load, we could eliminate displacement caused by the deformation of the column above the connection, and instead quantify the amount of displacement caused only by the deformation and rotation at the connection itself.

After dividing the part and assigning material properties, the part was seeded and meshed. It was then instanced in the assembly; fixed boundary conditions were imposed at the bottom, left, and right borders of the continuum; and a 1-kip horizontal point load was applied at the top of the beam. The job was then submitted to Abaqus/Standard for analysis. After the analysis was complete, the model was queried for the displacement at the point of applied load. The displacement value was recorded, and divided by the 1 kip of applied force, in order to determine the stiffness of the connection.

The foundation's depth extended to 10x the column depth, and its width extended to 10x the column depth on either side of the column. The mesh size was 1.0 inch. Fixed boundary conditions were created on the sides and bottom of the foundation part.

Parameters Studied

The first parameter studied, which was called α , was the ratio of the Young's Modulus of the column, to the Young's Modulus of the continuum. Thus, an α value of 3 would mean that the Young's Modulus of the column would be 3 times larger than that of the continuum. α was evaluated at values of 0.1, 0.25, 0.5, 1.0, 2.0, 4.0, and 10.0. Unreinforced concrete has an α of between 8 and 9 – therefore, our results with $\alpha = 10.0$ are the most directly applicable to the case of unreinforced concrete. However, α has been varied to gain greater understanding of the sensitivity of connection stiffness to a change in α.

The second parameter, called β, was the ratio of the column depth to the embed depth. For example, a β value of 3 means that the embedment depth was 3 times greater than the column depth; a column depth of 3 and a β value of 3 would mean that the column was embedded 9 inches into the continuum. Values of β that were studied ranged from 0 to 4.0, in increments of 0.1.

The third varied parameter was beam depth, d, which was explored at d = 8.0, 12.0, 18.0, and 30.0.

The following parameters remained constant in this investigation:

- Young's Modulus (E) of the column: 29,000 ksi
- Protruding length (pL), or cantilever height, of the column: 80.0 inches
- Magnitude of the applied lateral load: 1 kip
- Poisson's Ratio of all materials: 0.3

Results and Analysis

As embed depth (represented by β) increases, connection stiffness will increase asymptotically towards a maximum value. This maximum stiffness value varied depending on α and d values, with greater values of α and d tending to increase the maximum stiffness.

Figure B-0-2 shows results for α = 10.0. Figure B-0-3 shows the same results, normalized on the x- and y- axes by the models' column depths and were maximum stiffness values, respectively. Although the theoretical maximum is admittedly unobtainable (requiring an embedment depth = ∞), a close approximation was obtained at β = 4.0; additional analysis (not included) shows that stiffness values at β = 4.0 varied by less than 1% from values at β = 10.0.

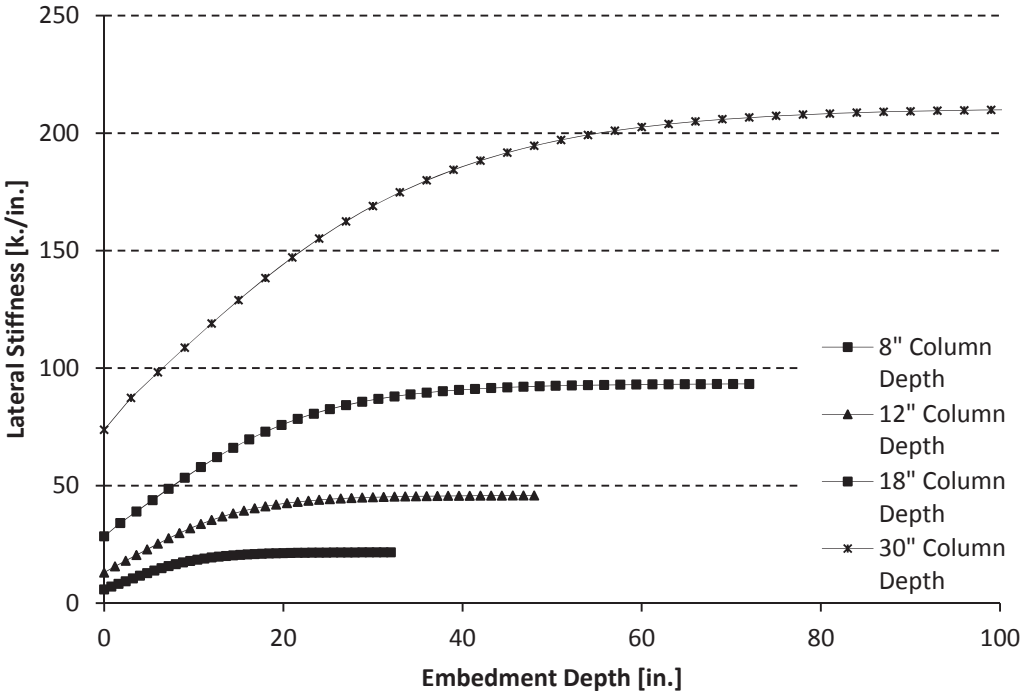


Figure B-0-2: 2-Dimensional Results; α = 10.0

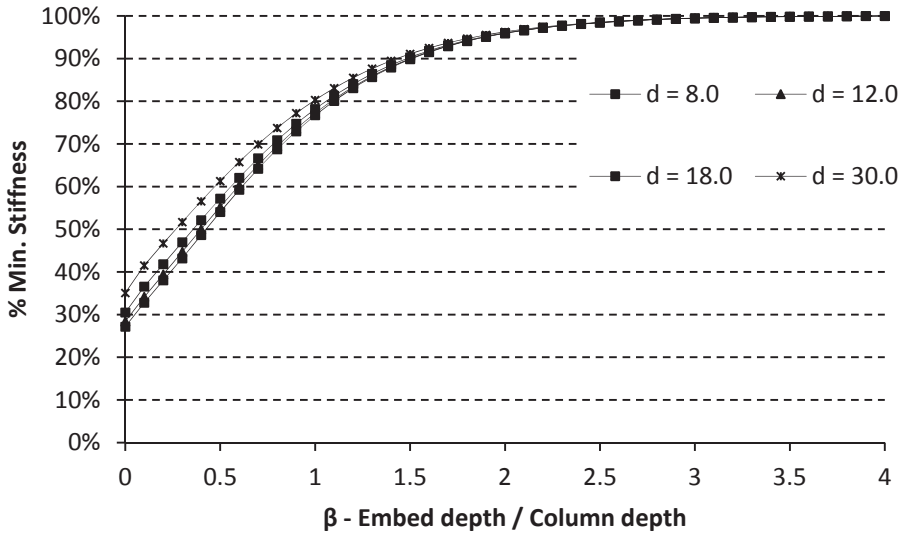


Figure B-0-3: Normalized 2-Dimensional Results; α = 10.0

Maximum stiffness values for $\alpha \geq 1.0$ are shown in Figure B-0-4. In the case of $\alpha \leq 1.0$, the stiffness values will asymptotically approach a minimum value instead of a maximum value

(since it represents a foundation that is actually stiffer than the steel column; as the column goes deeper, the connection actually loses stiffness). Maximum values are shown in Figure B-0-5, while minimum values are shown in Figure B-0-6.

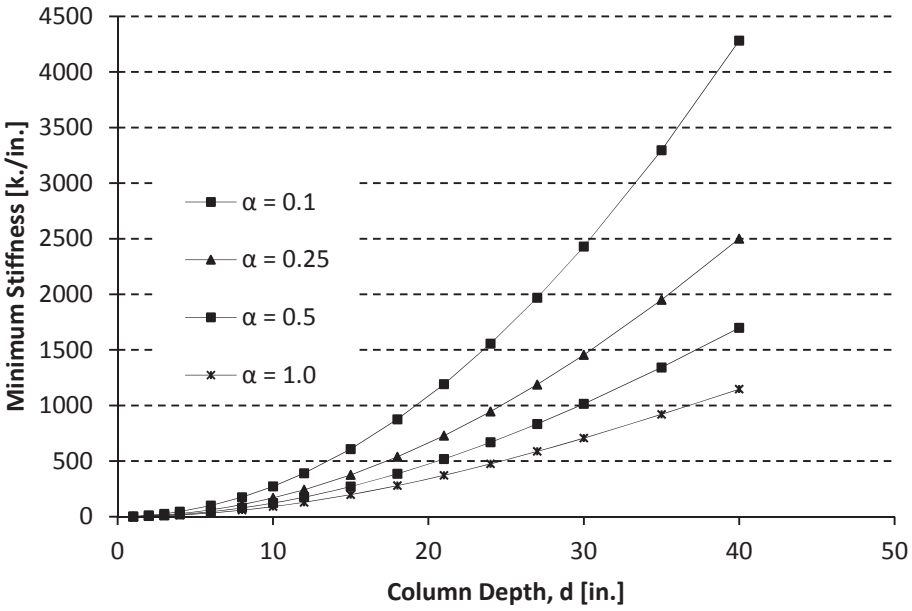


Figure B-0-4: Maximum Stiffness Values; $\alpha \leq 1.0$

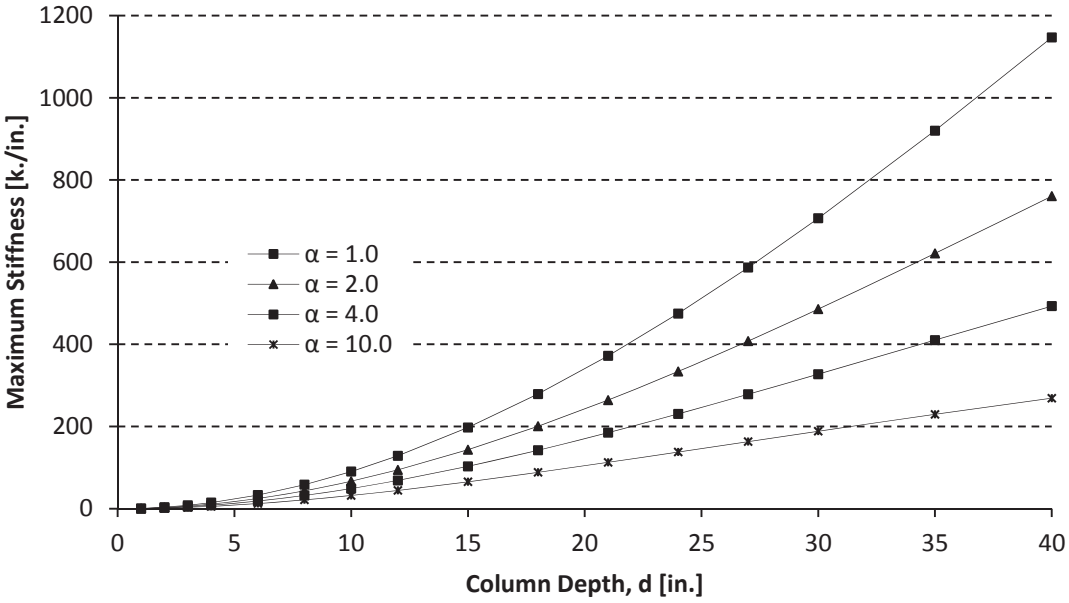


Figure B-0-5: Maximum Stiffness for Constant α ≥ 1.0

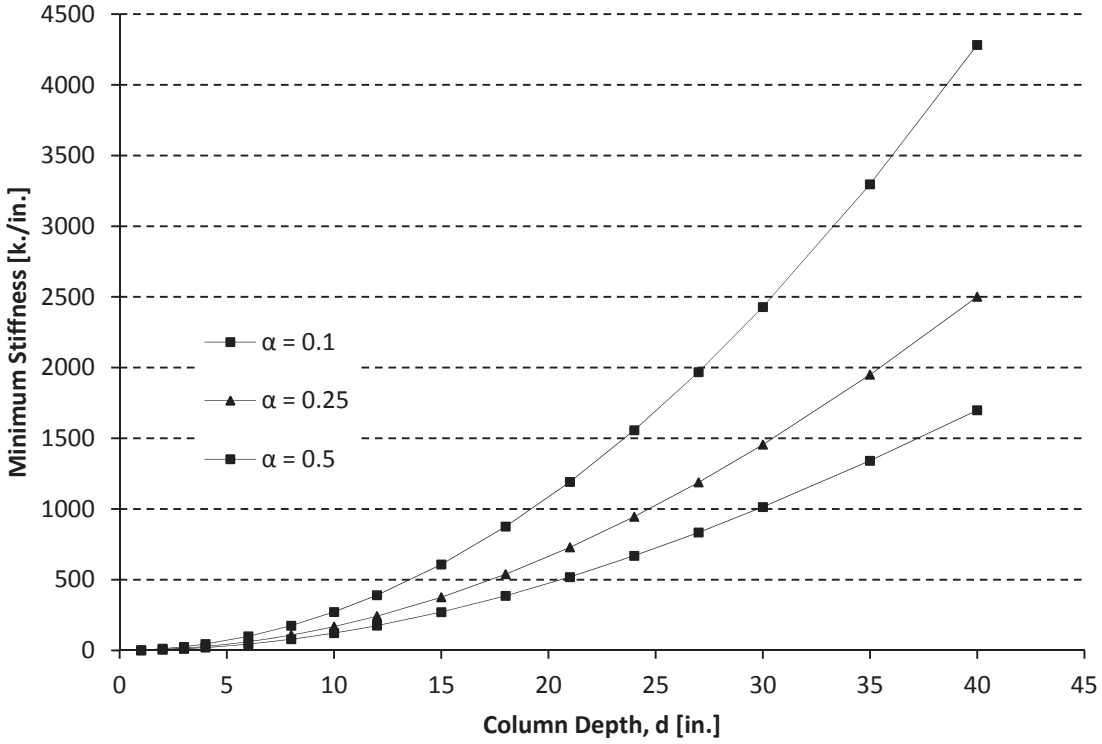


Figure B-0-6: Minimum Stiffness for Constant α ≤ 1.0

When $\alpha < 1.0$ – that is, the continuum material is softer than the column – increasing the embedment depth will increase the connection stiffness, and the asymptotic value is a maximum. This relationship is reversed when $\alpha > 1.0$: the column is less rigid than the surrounding material, increasing embedment causes stiffness to asymptotically approach a minimum value. When $\alpha = 1.0$, stiffness remains unchanged as embedment depth varies.

Figure B-0-5 shows maximum stiffness values for constant α values, assuming varying column depths. In each case, the protruding column length = 80.0 inches, so rotational stiffness can be obtained by multiplying by $pL^2 = (80\text{ in})^2 = 6400\text{ in}^2$.

$\alpha = 10.0$

Since the case of $\alpha = 10.0$ most closely models the case of a steel beam embedded in concrete, it is worth noting the amount of stiffness available from even a relatively shallow embedment in this case. In the columns studied, at $\beta = 0.5$, the connection reaches 53-60% of its available maximum stiffness. Higher values of β, d, and/or α will increase this percentage even more. For example, if Beta is increased to 1.0, the connection reaches 73.7% of its maximum.

$\alpha = 2.5$

In these cases, as α increases, the rigidity of the continuum does so as well. This means that, in addition to the increase in absolute maximum stiffness, we also observe higher initial stiffness, and a faster convergence to that maximum stiffness value, as α approaches 1.0.

All other conclusions drawn in the case of $\alpha = 10.0$ are applicable for these other cases. The process by which we will model our results is the same, and has been explained above.

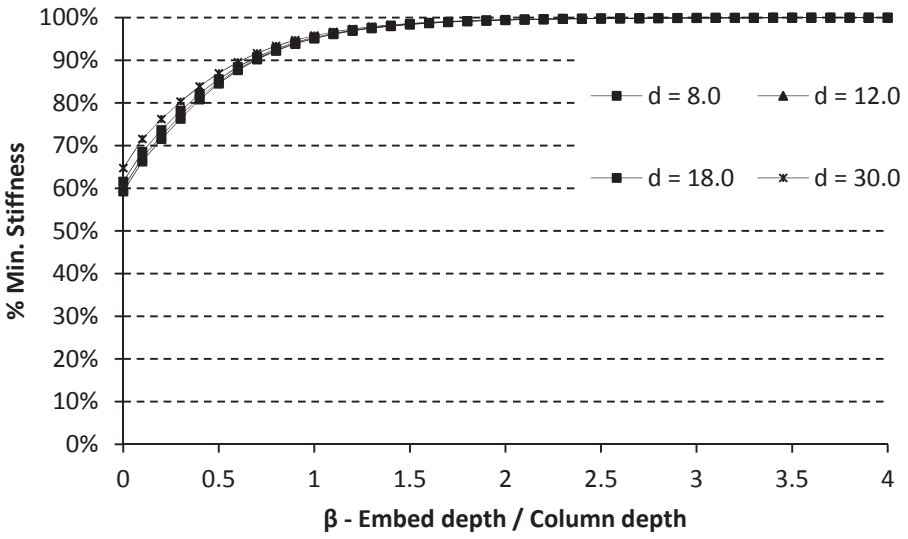


Figure B-0-7: Percent Maximum Stiffness Reached

α = 1.0

When α is exactly equal to 1.0, there is no difference between the column and continuum materials. Therefore, the stiffness in every case is equal to the theoretical maximum/minimum value, and no plot is required.

α = 0.4, 0.1

Results for the cases of α = 0.4 and α = 0.1 are shown in Figure B-0-8 and Figure B-0-9, respectively.

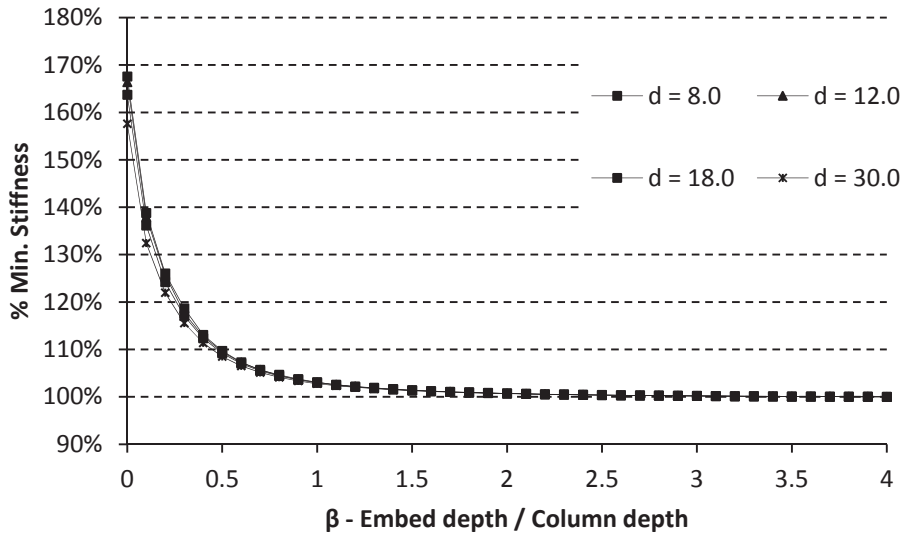


Figure B-0-8: α = 0.4

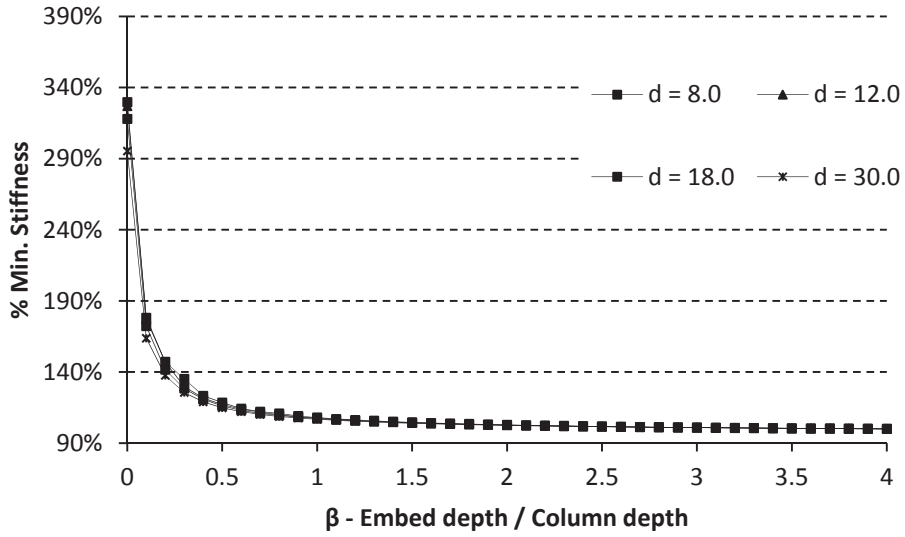


Figure B-0-9: α = 0.1

Mesh Convergence Study

A mesh convergence study was performed to determine the effects of mesh size on displacement results. The column studied had the following specific properties: $\alpha = 10.0$, $\beta = 1.0$, $d = 12$ inches. 1 inch was chosen as the standard mesh size for the fine mesh. A 1-inch mesh size diverged less than 0.15% from a 0.2-inch mesh (which was the minimum mesh size studied). Figure B-0-10 shows the results.

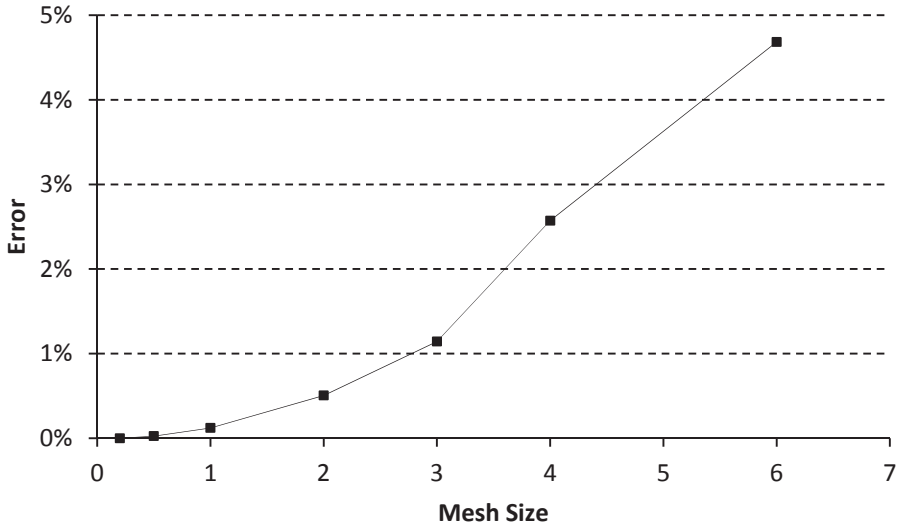


Figure B-0-10: Results of Mesh Convergence Study

APPENDIX C – SINGLE PART, EXPOSED BASEPLATE MODEL

To verify the accuracy of the rigidly tied models, a one piece model was created, with a zero-embedment depth. All procedures were followed identically to those of the rigidly tied model. However, instead of creating two distinct parts, only one was created. This part had no embedment depth, and the bottom of the baseplate was level with the top of the concrete slab. The model was a W8x35 shape, with 80.25" length in addition to the baseplate. Mesh density was refined once to check mesh convergence.

The stiffness values were higher than those obtained with the rigid tie model (see Table C-0-1). These values were higher than believed to be theoretically possible (compare to Tryon, 2016), as well as higher than the results obtained from experimental testing (Barnwell, 2015). Therefore, the investigation into this line of modeling was discontinued. However, it supports the conclusion that using a fully-fixed bond between the concrete and steel would not accurately simulate bond stiffness.

Table C-0-1: Single part, Exposed Baseplate Results

Mesh Density	Displacement [in.]	Lateral Stiffness [k./in.]
0.5	5.600E-02	108.36
0.25	5.602E-02	108.18

Figure C-0-1 shows von Mises stress distributions. A high stress distribution is noted in the baseplate, with relatively insignificant stress concentrations in the concrete. Figure C-0-2 shows the stresses in the concrete slab only (note the change in scale for Mises stress).

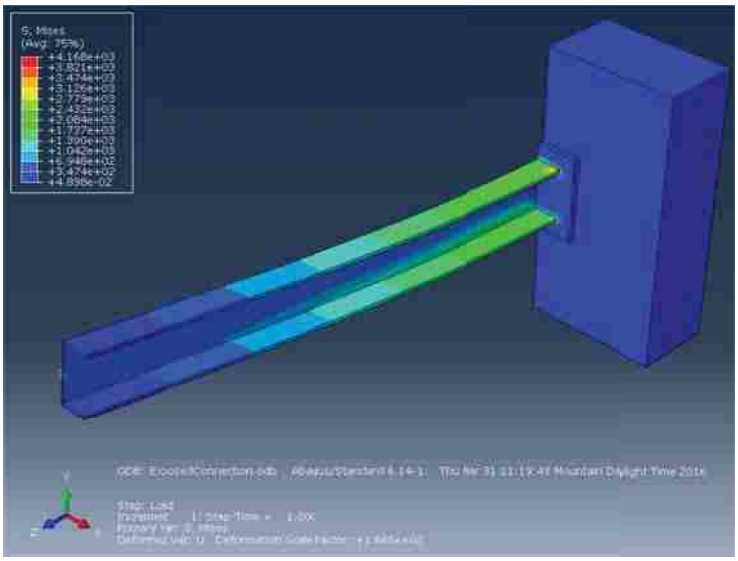


Figure C-0-1: Von Mises Stress in Column and Concrete Slab

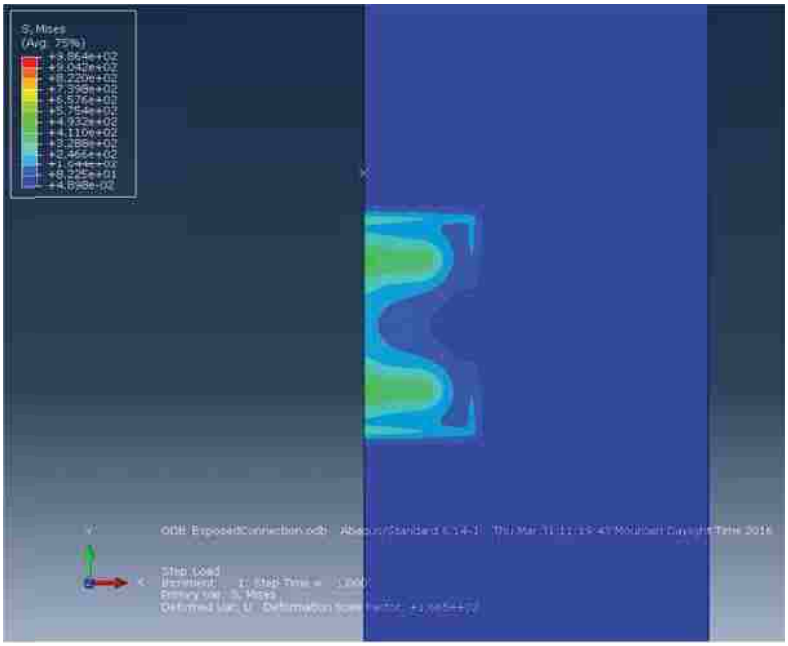


Figure C-0-2: Von Mises Stress in Concrete Slab

APPENDIX D – DESIGN JUSTIFICATION

The following are MathCAD sheets with design justification for the W14x176 and W24x76 baseplates. Both were designed by Kevin Hanks, P.E., to reflect typical practice for designing low-moment and high-moment connections.

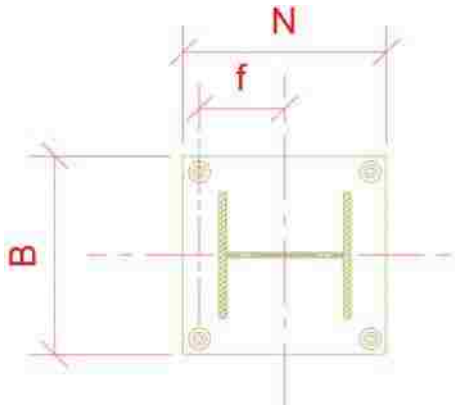
Base Plate Design - for Concentric Axial Load
per AISC Steel Design Guide 1, section 3.1
Calculation by: Kevin N Hanks Date: 12 Jan 2016

Given

Column

Column Size	$Col := \text{“W14x176”}$		
Column Height (unbraced)	$h_{col} := 15\ ft$		
Steel Strength	$f_{y\_col} := 50\ ksi$	ASTM A992 Steel	$R_y := 1.1$

Base Plate



Width	$N := 2\ ft + 0\ in$
Depth	$B := 2\ ft + 0\ in$
Area	$A_1 := N \cdot B = 4\ ft^2$
AB Location	$f := 4\ in$
Steel Strength	$f_{y\_bspl} := 36\ ksi$

$$n := \frac{B - 0.8 \cdot b_f}{2} = 5.72\ in$$

$$m := \frac{N - 0.95 \cdot d}{2} = 4.78\ in$$

Concrete Base

Concrete Strength	$f`_c := 4000\ psi$	
Concrete Base Area	$A_2 := 4 \cdot A_1$	(Assumed)

Anchor Bolts

Material: ASTM F1554 Gr 36

Steel Yield Strength	$f_{y\_bolt} := 36\ ksi$
Steel Rupture Strength	$f_{u\_bolt} := 58\ ksi$
Total # of bolts	$num_b := 4$

Misc

Strength Reduction Factors	$\phi_{steel} := 0.90$
	$\phi_{conc} := 0.65$

Step A: Determine the Design Case

$$
\begin{array}{l|l}
\text{if } A_2 = A_1 & = \text{“Case II”} \\
\quad \| \text{“Case I”} & \\
\text{else if } A_2 \geq 4 \cdot A_1 & \\
\quad \| \text{“Case II”} & \\
\text{else if } A_1 < A_2 < 4 \cdot A_1 & \\
\quad \| \text{“Case III”} &
\end{array}
$$

Step 1: Calculate the factored axial compressive load, Pu

Note: for this case, the factored axial load will be taken as the maximum compressive buckling strength of the column over its unbraced length.

Assume both top and bottom are "pin" connected

$K := 1.0$

Slenderness

$L := h_{col}$

$$\frac{K \cdot L}{r_y} = 44.776$$

Euler Buckling Stress

$$F_e := \frac{\pi^2 \cdot E}{\left(\frac{K \cdot L}{r_y}\right)^2} = 142.759 \ ksi$$

Critical stress (Flexural Buckling)

$$F_{cr} := \begin{cases} \left(0.658^{\frac{f_{y\_col}}{F_e}}\right) \cdot f_{y\_col} & \text{if } \frac{K \cdot L}{r_y} \le 4.71 \cdot \sqrt{\frac{E}{f_{y\_col}}} \\ 0.877 \cdot F_e & \text{else} \end{cases} = 43.182 \ ksi$$

Nominal Compressive Strength

$$P_n := F_{cr} \cdot A_g = 2237 \ kip$$

Design Axial Load on the Column Base

$$P_u := R_y \cdot P_n = 2460.5 \ kip$$

Step 2: Calculate the required base plate area

$$A_{1\_req} := \frac{P_u}{2 \cdot \phi_{conc} \cdot 0.85 \cdot f`_c} = 3.866 \ ft^2$$

$$A_{1\_actual} := A_1 = 4 \ ft^2$$

Step 3: Optimize the Base Plate Dimensions, N & B

$$N_{opt} := \sqrt{A_{1\_req}} + \frac{0.95 \cdot d - 0.8 \cdot b_f}{2} = 24.534 \ in$$

$$B_{opt} := \frac{A_{1\_req}}{N_{opt}} = 22.69 \ in$$

Note that this represents the minimum required base plate size for strength requirements - geometric constrains may govern the size of the base plate
Return to page 1 and adjust base plate dimensions

Step 4: Calculate the required base plate thickness.

$$P_p := f`_c \cdot 2 \cdot A_1 = 4608 \ kip$$

$$X := min\left(\left(\frac{4 \cdot d \cdot b_f}{(d + b_f)^2}\right) \cdot \frac{P_u}{\phi_{steel} \cdot P_p}, 1.0\right) = 0.593$$

$$\lambda := min\left(\frac{2 \cdot \sqrt{X}}{1 + \sqrt{1 - X}}, 1.0\right) = 0.94$$

$$m := \frac{N - 0.95 \cdot d}{2} = 4.78 \ in$$

$$n := \frac{B - 0.8 \cdot b_f}{2} = 5.72 \ in$$

$$\lambda n` := \lambda \cdot \frac{\sqrt{d \cdot b_f}}{4} = 3.632 \ in$$

$$l_{max} := \max(m, n, \lambda n`) = 5.72 \ in$$

$$t_{min} := l_{max} \cdot \sqrt{\frac{2 \cdot P_u}{\phi_{conc} \cdot f_{y\_bspl} \cdot B \cdot N}} = 3.456 \ in$$

Step 5: Determine the anchor rod size and location

For gravity-only loads, anchor rods need only fulfil OSHA minimum requirements

Use (4) 3/4" ASTM F1554 Gr 36 rods

Base Plate Design - for large Moment
per AISC Steel Design Guide 1, section 3.4
Calculation by: Kevin N Hanks Date: 12 Jan 2016

Given

Column

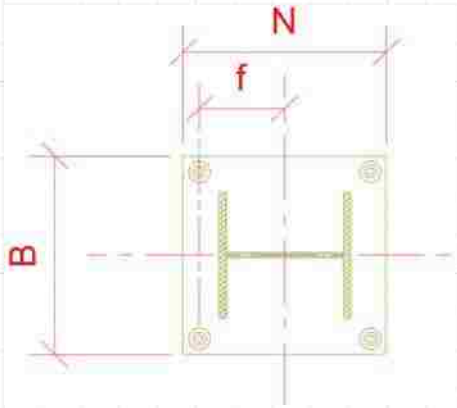
Column Size	$Col := \text{“W24x76”}$		
Column Height	$h_{col} := 15\ ft$		
Steel Strength	$f_{y\_col} := 50\ ksi$	ASTM A992 Steel	$R_y := 1.1$

Base Plate

Width	$N := 2\ ft + 10\ in$
Depth	$B := 1\ ft + 8\ in$
AB Location	$f := 1\ ft + 2.5\ in$
Steel Strength	$f_{y\_bspl} := 36\ ksi$

$$n := \frac{B - 0.8 \cdot b_f}{2} = 6.404\ in$$

$$m := \frac{N - 0.95 \cdot d}{2} = 5.648\ in$$



Concrete

$f`_c := 4000\ psi$

Anchor Bolts

Material: ASTM F1554 Gr 36

$f_{y\_bolt} := 36\ ksi$
$f_{u\_bolt} := 58\ ksi$
$num_b := 8$

Step 1: Determine the Axial and Moment Load on Column Base

$P_r := h_{col} \cdot W_{col} = 1140\ lbf$ (This design assumes no additional gravity load on the column)

$M_r := 1.1 \cdot R_y \cdot f_{y\_col} \cdot Z_x = 1008.3\ kip \cdot ft$ (See AISC 341 sec. 8.5c.(2).(a)

Step 2: Pick a trial base plate size (*N* x *B*)

(defined previously)

$N = 34\ in$

$B = 20\ in$

Step 3: Determine the equivalent eccentricity (*e*), and the critical eccentricity (ecrit)

$e := \frac{M_r}{P_r} = 884.5\ ft$

$e_{crit} := \frac{N}{2} - \frac{P_r}{2 \cdot q_{max}} = 16.994\ in$

if $e \geq e_{crit}$ = "GOOD!"
  "GOOD!"
else
  "NO GOOD!"

Step 4: Determine the equivalent bearing length (*Y*) and the tensile force in the anchor rods (*T*).

$$Y := \left(f + \frac{N}{2}\right) + \begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot \sqrt{\left(f + \frac{N}{2}\right)^2 - \frac{2 \cdot P_r \cdot (e+f)}{q_{max}}} = \begin{bmatrix} 58.298 \\ 4.702 \end{bmatrix} in$$

$Y :=$ if $0 < Y_0 < N$ $= 4.702\ in$
  Y_0
else
  Y_1

$T := q_{max} \cdot Y - P_r = 414.537\ kip$

Step 5: Determine the required minimum base plate thickness at the bearing and the tension interfaces.

Bearing Interface:

$$t_{p\_req\_b} := \begin{cases} \text{if } Y \geq \max(m, n) \\ \quad 1.5 \cdot m \cdot \sqrt{\dfrac{f_{p\_max}}{f_{y\_bspl}}} \\ \text{else} \\ \quad 2.11 \cdot \sqrt{\dfrac{f_{p\_max} \cdot Y \cdot \left(m - \dfrac{Y}{2}\right)}{f_{y\_bspl}}} \end{cases} = 2.911 \ in$$

Tension Interface

$$t_{p\_req\_t} := 2.11 \cdot \sqrt{\frac{T \cdot \left(f - \dfrac{d}{2} + \dfrac{t_f}{2}\right)}{B \cdot f_{y\_bspl}}} = 2.722 \ in$$

Take maximum of the two:

$$t_{p\_req} := \max\left(t_{p\_req\_b}, t_{p\_req\_t}\right) = 2.911 \ in$$

Step 6: Determine the anchor rod size appropriate for the tensile loading
(*assuming that the embedment is such that tensile fracture of the anchor bolt governs, and that shear load in the bolts is negligible)*

Number of bolts in tension

$$N_{b\_t} := \frac{num_b}{2} = 4$$

Tensile Force in each bolt

$$T_u := \frac{T}{N_{b\_t}} = 103.634 \ kip$$

required diameter $\phi := 0.75$

$$D_{req} := \sqrt{\frac{4 \cdot T_u}{\pi \cdot \phi \cdot f_{u\_bolt}}} = 1.742 \ in$$

APPENDIX E – CODE FOR MODEL GENERATION AND ANALYSIS

The code to generate the models was written exclusively in the programming language Python.

A wrapper script, called RunMe.py was called from the command line to initiate the process. The RunMe script 1) sent a command to Abaqus/CAE to run Preprocessing.py, which automatically generated the models 2) opened multiple threads to submit all the job files to Abaqus/Standard for processing, and 3) sent a command to Abaqus/CAE to run Postprocessing.py, which opened the models, extracted the needed information, and deposited it into an Excel (.csv) database for manual analysis. Both Preprocessing.py and Postprocessing.py had several additional layers of subroutines that facilitated both automatic model generation, and code development.

At the start of RunMe.py, Preprocessing.py, and Postprocessing.py, a header was created containing all of the information concerning the variables to be run. Up to two variables at a time ("PrimaryParameter" and "SecondaryParameter") can be varied, as well as embedment depths. The choices of values of the variables were contained in "PrimaryParameterList" and "SecondaryParameterList," while the embedment depths were contained in "EmbedDepthsList." The scripts generate and analyze models containing every combination of variables and

embedment depths. The headers at the top of all three scripts must be identical, or the analysis will behave in unexpected ways or crash.

Preprocessing Tasks

The Preprocessing.py script was responsible for generating all of the models. To do this, it first generated all of the necessary input variables for Abaqus/CAE for each model, including variables defining what model configuration to use. These variables were stored in a dictionary called "DataArray". The variables in DataArray were updated after every model was run, to reflect the necessary variables for each separate model. (This was done to ease code development; since several dozen variables were required to define each model, and they were often changing as the model grew in complexity, it became cumbersome to pass every variable through several layers of subroutines by hand. Therefore, by defining the DataArray dictionary, an arbitrary number of variables could be defined without concern for raising errors or forgetting to pass needed arguments to subroutines.)

After DataArray was created or updated, Preprocessing.py called the Preprocessing subroutine (not to be confused with Preprocessing.py, the script which calls it), which was located in the Scripts.py library. The Preprocessing routine unpacked the DataArray dictionary into variables that could be referenced (by the Preprocessing routine) without having to reference the DataArray dictionary itself. For example, the variable "BasePlate" (a Boolean specifying if the model was to include a baseplate or not) was defined in the Preprocessing.py script as "DataArray['BasePlate']", and passed to the Preprocessing subroutine. When it reached the Preprocessing subroutine, a loop over every entry in the dictionary ("for key in

DataArray.keys(): exec('%s = DataArray["%s"]' %(key, key))) unpacked it, thus allowing it to be referenced simply as "BasePlate" from then on.

Once DataArray is unpacked, a number of assertion lines ensure that certain disallowed combinations of variables are not accidentally sent to be processed. These combinations, if accidentally passed to the script, could cause the model to be created in unexpected and incorrect ways, or perhaps even crash Abaqus/CAE as it builds the model.

After the assertion lines, the "Mdb()" command creates and opens a new model database file. Once this is done, a series of subroutines is called, each of which accomplishes one additional step in building the model in Abaqus/CAE. For example, CreateModel creates a new model within the database file; ColumnCreation creates a column part in the model file according to the applicable parameters within DataArray; DivideColumn divides the column part into several different cells to faciliatate later meshing; and so forth. The same unpacking loop as before appears in each subroutine.

After all of the specified subroutines have been run to create the model, an input file is created, which converts the model into a ".inp" file which can be read directly by Abaqus/Standard when processing the model. Finally, the model database was saved so that it can be opened later during postprocessing.

Model Processing

At this point, control reverts back to the top level script, RunMe.py. RunMe opens up multiple threads, each one of which takes one ".inp" file and submits it to Abaqus/Standard for processing. Typically, more jobs are submitted than there are licenses available, which results in

most jobs waiting in a queue to be processed. There is no apparent way of predicting what order the jobs will run in.

Postprocessing Tasks

The postprocessing sequence is structured identically to the preprocessing sequence: RunMe.py calls a script called Postprocessing.py in Abaqus/CAE, which assembles all the variables into DataArray, and repeatedly calls a subroutine called "Postprocessing" from the Scripts.py library. Indeed, Postprocessing.py is actually a copy of Preprocessing.py, with the only difference being which subroutine (Preprocessing or Postprocessing) it calls from Scripts.py.

The Postprocessing subroutine in turn runs a series of subroutines which: opens the model database file; obtains nodal displacement data at the point of the applied load; outputs it into an XY Data report inside of Abaqus; exports this XY Data report into a ".output" file; scrapes the nodal displacement value from the .output file; calculates the connection's linear stiffness and the connection's rotational stiffness; and outputs these three values to a .csv database which can be read by Microsoft Excel. Graphing and analysis was all done manually with the information now available in the .csv file.

Scripts

The code for RunMe.py, Preprocessing.py, Postprocessing.py, and Scripts.py all follow. For analysis on the supercomputer, additional wrapper scripts (written with Linux bash commands) were required, which are not included here.

```
#RunMe.py

########################
#Copy/paste from here...

Linux = 1
SuperComputing = 0

ModelType = 'CohesiveZone'
BaseplateType = 'Square' #BaseplateType Possibilities: 'Square', 'Rectangle', 'Reduced', 'None'

PrimaryParameter = 'BCs'
PrimaryParameterList = ['Bottom', 'Sides']
SecondaryParameter = 'MeshSize'
SecondaryParameterList = [0.5]
EmbedDepthsList = [1.5, 5.5, 9.5, 13.5, 17.5]
# EmbedDepthsList = [1.5]

#...to here           #
########################

###############################
#Initialization & bookkeeping.#
###############################
if SuperComputing: assert Linux
import os
import multiprocessing

def __TimeStamp():
    global TimeStamp
    from datetime import datetime
    month = str(datetime.now().month)
    day = str(datetime.now().day)
    year = str(datetime.now().year)
    hour = str(datetime.now().hour)
    minute = str(datetime.now().minute)
    second = str(datetime.now().second)
    return '{0}-{1}-{2}_{3}-{4}-{5}'.format(month, day, year, hour, minute, second)


from string import join

if Linux:
    if SuperComputing:
        Heading = '/fslhome/trevdna/'
        os.chdir('/fslhome/trevdna/compute/Models')
        os.system('module load abaqus/6.14')
    else:
        Heading = '/fsc/trevdna/'
        os.chdir(Heading + 'groups/researchtaj/scratch/ColumnModels_CohesiveZone/')
else: #Windows
    Heading = 'J:/'
```

```
FilePath = Heading + '/Scripts/'

if Linux:
    guiCaps = 'GUI'
else:
    guiCaps = 'gui'

jobs = []

def __removeDot(str1):
    return join(str(str1).split('.'),'point')
###############
#Preprocessing#
###############

# Note: no multithreading for pre- or post-processing.
os.system('abaqus cae no%s=%sPreprocessing' %(guiCaps, FilePath))


#############
#Processing.#
#############

numModels = len(PrimaryParameterList) * len(SecondaryParameterList) * len(EmbedDepthsList)
if Linux:
    maxCPUS = multiprocessing.cpu_count() * 1/2
else:
    maxCPUS = multiprocessing.cpu_count()
maxProcesses = max(numModels, maxCPUS / 4)

def Processing(argsList):
    PrimaryParameter = argsList[0]
    Param1 = argsList[1]
    SecondaryParameter = argsList[2]
    Param2 = argsList[3]
    eL = argsList[4]
    ModelName = '%s%s_%s%s_eL%s' %(PrimaryParameter, __removeDot(Param1), SecondaryParameter,
    __removeDot(Param2), __removeDot(eL))
    print(ModelName + ' processing began at ' + __TimeStamp())
    os.system("abaqus job=%s cpus=4 interactive ask_delete=OFF" %ModelName)
    return

pool = multiprocessing.Pool(processes=maxProcesses)

args = []
for Param1 in PrimaryParameterList:
    for Param2 in SecondaryParameterList:
        for eL in EmbedDepthsList:
            args += [[PrimaryParameter, Param1, SecondaryParameter, Param2, eL]]

pool.map(Processing, args)
#Note: There's no guarantee on which model file will run first. It seems to be whichever
process can hop to the front of the line first.
```

```
################
#Postprocessing#
################

os.system('abaqus cae no%s=%sPostprocessing' %(guiCaps, FilePath))

print('Done!')
```

```
########################
#Copy/paste from here...

Linux = 1
SuperComputing = 0

ModelType = 'CohesiveZone'
BaseplateType = 'Square' #BaseplateType Possibilities: 'Square', 'Rectangle', 'Reduced', 'None'

PrimaryParameter = 'BCs'
PrimaryParameterList = ['Bottom', 'Sides']
SecondaryParameter = 'MeshSize'
SecondaryParameterList = [0.5]
EmbedDepthsList = [1.5, 5.5, 9.5, 13.5, 17.5]
# EmbedDepthsList = [1.5]

#...to here            #
########################

ScriptType = 'Preprocessing'

########################
#Initialization tasks.#
########################
if SuperComputing: assert Linux
global DataArray
import os
#Supercomputing Linux, normal Linux, or Windows.
if Linux:
    if SuperComputing:
        Heading = '/fslhome/trevdna/'
    else:
        Heading = '/fsc/trevdna/'
else:#Windows
    Heading = 'J:/'

if SuperComputing:
    os.chdir('/fslhome/trevdna/compute/Models')
else:
    os.chdir(Heading + 'groups/researchtaj/scratch/ColumnModels_CohesiveZone/')

#Import functions
import csv
from string import join
from sys import path
from math import sqrt
path.append(Heading + 'Scripts/')
if ScriptType == 'Preprocessing':
    from Scripts import Preprocessing
    # pass
elif ScriptType == 'Postprocessing':
    from Scripts import Postprocessing
else:
```

```
    raise TypeError('Unexpected Script Type.')

#Dictionary with part names and properties
PropertiesDict = {}
ShapesDatabase = Heading + 'Research/4-InputDatabases/ShapesDatabase_Custom.csv'
if SuperComputing: ShapesDatabase = Heading + 'InputDatabases/ShapesDatabase_Custom.csv'
with open(ShapesDatabase) as csvfile:
    quoting = csv.QUOTE_NONNUMERIC
    reader = csv.reader(csvfile)
    for row in reader:
        PropertiesDict[row[0]] = row[3], row[4], row[6], row[8], \
            row[11], row[13], row[18], row[22]
'''0: bA - beam Area
1: db - beam depth
2: bf - Flange width
3: tw - Thickness of web
4: tf - thickness of flange
5: k(des) - smallest possible k value
6: Ix - Strong moment of inertia
7: Iy - Weak moment of inertia'''

WorkingDir = os.getcwd()

def __removeDot(str1):
    return join(str(str1).split('.'),'point')

##########################################################################
#Create dictionary with needed information to import into Abaqus routines.#
##########################################################################

DataArray = {}
#Metadata
DataArray['ModelType'] = ModelType #Contact type
DataArray['TwoD_ThreeD'] = False #Does it taper to 2D from 3D?
DataArray['OneD_TwoD'] = False
DataArray['PrimaryParameter'] = PrimaryParameter
DataArray['SecondaryParameter'] = SecondaryParameter
DataArray['ColumnType'] = 'IBeam' #'IBeam', 'Rectangle', or 'Square'
# DataArray['CohesiveZone'] = False

#Setup Parameters (Default)
DataArray['StrongOrient'] = True
DataArray['BasePlate'] = True
def Define_ModelType_BasedData():
    if DataArray['ModelType'] == 'Friction' or DataArray['ModelType'] == 'CohesiveZone':
        DataArray['OnePartModel'] = False
    elif DataArray['ModelType'] == 'RigidTie': #RigidTie, Tied, whatever I called it that day.
        DataArray['OnePartModel'] = True

    if DataArray['OnePartModel']:
        DataArray['ColumnPart'], DataArray['FoundationPart'] = 'CombinedPart', 'CombinedPart'
    else:
        DataArray['ColumnPart'], DataArray['FoundationPart'] = 'Column', 'Foundation'
```

```
    if DataArray['ModelType'] == 'CohesiveZone':
        DataArray['CohesiveZone'] = True
        # DataArray['ColumnPart'], DataArray['FoundationPart'] = 'CombinedPart-CZ',
        'CombinedPart-CZ'
    else:
        DataArray['CohesiveZone'] = False

Define_ModelType_BasedData()
DataArray['BlockoutConcrete'] = False

#File paths
outputFileFolder = Heading + 'Research/1-ThesisResearch/'
if SuperComputing: outputFileFolder = Heading + 'RawOutputFiles/'
DataArray['outputFile'] = '%s%s%s.csv' %(outputFileFolder,PrimaryParameter,SecondaryParameter)

###Column properties###

#Square/rectangular column properties
# DataArray['cX'] = 6.855
# DataArray['cY'] = 6.855

DataArray['StrongAxis'] = True
DataArray['ColumnName'] = 'W8X35'
def Define_ColumnName_BasedData():
    global DataArray
    DataArray['db'] = float(PropertiesDict[DataArray['ColumnName']][1])
    DataArray['tw'] = float(PropertiesDict[DataArray['ColumnName']][3])
    DataArray['bf'] = float(PropertiesDict[DataArray['ColumnName']][2])
    DataArray['tf'] = float(PropertiesDict[DataArray['ColumnName']][4])
    if DataArray['ColumnType'] == 'IBeam':
        DataArray['Ix'] = float(PropertiesDict[DataArray['ColumnName']][6])
        DataArray['Iy'] = float(PropertiesDict[DataArray['ColumnName']][7])
    elif DataArray['ColumnType'] == 'Square' or DataArray['ColumnType'] == 'Rectangle':
        DataArray['Ix'] = float(cX*cY**3/12)
        DataArray['Iy'] = float(cY*cX**3/12)
    k = float(PropertiesDict[DataArray['ColumnName']][5])
    DataArray['fr'] = k - DataArray['tf']
    db, tw, bf, tf = DataArray['db'], DataArray['tw'], DataArray['bf'], DataArray['tf']
    DataArray['SA'] = db*tw+2*bf*tf-2*tf*tw #Surface Area / cross sectional area
Define_ColumnName_BasedData()

#Baseplate properties
baseWidth = 13.0
baseWidthX = baseWidth - 2.0
baseWidthY = baseWidth
baseDepth = 1.0
DataArray['BaseplateType'] = BaseplateType #BaseplateType Possibilities: 'Square', 'Rectangle',
'Reduced', 'None'
def Define_BaseplateType_BasedData(): #Also includes changes to baseplate dimensions based on
column size
    global baseWidth, baseWidthX, baseWidthY
    if DataArray['ColumnName'] == 'W14X176': #Patches for individual test cases, not a
```

```
    universal solution here.
        baseWidth = 24.0
    elif DataArray['ColumnName'] == 'W24X76':
        assert BaseplateType == 'Rectangle'
        # baseWidth = 39.0
        baseWidthX = 20.0
        baseWidthY = 34.0
    else:
        # DataArray['BaseplateType'] = BaseplateType
        baseWidth = 13.0
        baseWidthX = baseWidth - 2.0
        baseWidthY = baseWidth

    if DataArray['BaseplateType'] == 'Square':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = baseWidth
        DataArray['baseWidthY'] = baseWidth
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'Rectangle':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = baseWidthX
        DataArray['baseWidthY'] = baseWidthY
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'Reduced':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = DataArray['bf']
        DataArray['baseWidthY'] = DataArray['db']
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'None':
        DataArray['BasePlate'] = False
        # DataArray['baseWidthX'] = 0.0 #Should not be necessary
        # DataArray['baseWidthY'] = 0.0 #Should not be necessary
        DataArray['baseDepth'] = 0.0
    if DataArray['BaseplateType'] <> 'None':
        assert DataArray['baseWidthX'] >= DataArray['bf'] #Only valid in the case of
        strong-axis bending, FYI.
        assert DataArray['baseWidthY'] >= DataArray['db']
Define_BaseplateType_BasedData()

#Foundation properties
DataArray['mwX'] = 42 #Medium (foundation) width in x-direction
DataArray['mwY'] = 42 #Medium (foundation) width in y-direction
DataArray['BCs'] = 'Bottom'
# DataArray['blockoutSize'] = 17.0

#Column lengths
DataArray['pL'] = 80.25 #Protruding
DataArray['eL'] = 5.5 #Embedded

def Define_eL_BasedData():
    global DataArray
    DataArray['cL'] = DataArray['eL'] + DataArray['pL'] #Column length: embedded + protruding
    DataArray['cmd'] = 12.0 + DataArray['eL'] + DataArray['baseDepth'] + 1.5 #Default: 20.0
```

```
Define_eL_BasedData()

#Moduli and PRs; concrete strength
DataArray['EmbeddedSteelMod']   = 29000000.0
DataArray['NormalConcreteMod']  =  3500000.0
DataArray['SteelPr'] = 0.27 #Poisson's ratio, for steel
DataArray['ConcretePr'] = 0.15 #pr for concrete
DataArray['CohesiveMod'] = 5E4 #The pseudomodulus that is used in the cohesive zone material.
DataArray['CohesiveDepth'] = 0.01
DataArray['offsetVal'] = DataArray['CohesiveDepth']

def Define_EmbeddedSteelMod_BasedData():
    global DataArray
    DataArray['ProtrudingSteelMod'] = DataArray['EmbeddedSteelMod']
Define_EmbeddedSteelMod_BasedData()

def Define_NormalConcreteMod_BasedData():
    global DataArray
    DataArray['BadConcreteMod'] = DataArray['NormalConcreteMod']
    DataArray['GroutMod'] = DataArray['NormalConcreteMod']
    strength = (float(DataArray['NormalConcreteMod'])/57000)**2 #For reference
Define_NormalConcreteMod_BasedData()

#Load and friction values
# DataArray['DistLoad'] = True #Is this functioning as a distributed load (as opposed to a
point load)?
def Define_DistLoad_BasedData():
    global DataArray
    DataArray['DistLoad'] = not DataArray['StrongAxis'] #Is this functioning as a distributed
    load (as opposed to a point load)?
Define_DistLoad_BasedData()

DataArray['load'] = 1000 #Pounds
DataArray['AxialLoad'] = 0 #Pounds
DataArray['NoFriction'] = False
DataArray['Friction'] = 0.50
DataArray['NoSeparation'] = False

#Mesh sizes
DataArray['MeshSize'] = 0.5
DataArray['UniformMesh'] = True
DataArray['SquareMesh'] = True
DataArray['QuadMesh'] = False

################################
#Run the bloody script already!#
################################

#Loops
#Note: All the assertion lines in here are to make sure you don't try to vary two parameters
together that would result in bugs if you run them together.
#If you really want to run them together, code it yourself, and double (triple) check the code
actually behaves like you are expecting.
```

```
for Param1 in PrimaryParameterList:
    DataArray['Param1'] = Param1
    DataArray[PrimaryParameter] = Param1
    for Param2 in SecondaryParameterList:
        DataArray['Param2'] = Param2
        DataArray[SecondaryParameter] = Param2
        for eL in EmbedDepthsList:
            DataArray['eL'] = eL
            ######Update properties that are based on variables that may have changed.#######
            if 'ColumnName' in [PrimaryParameter, SecondaryParameter]:
                assert DataArray['ColumnType'] == 'IBeam'
                Define_ColumnName_BasedData()
            if 'EmbeddedSteelMod' in [PrimaryParameter, SecondaryParameter]:
            Define_EmbeddedSteelMod_BasedData()
            if 'NormalConcreteMod' in [PrimaryParameter, SecondaryParameter]:
            Define_NormalConcreteMod_BasedData()
            if 'OnePartModel' in [PrimaryParameter, SecondaryParameter]:
            Define_ModelType_BasedData()
            if 'ModelType' in [PrimaryParameter, SecondaryParameter]: Define_ModelType_BasedData
            ()
            if 'StrongAxis' in [PrimaryParameter, SecondaryParameter]: Define_DistLoad_BasedData
            ()
            if 'BaseplateType' in [PrimaryParameter, SecondaryParameter] or 'ColumnName' in [
            PrimaryParameter, SecondaryParameter]:
                # assert 'ColumnName' not in [PrimaryParameter, SecondaryParameter]
                Define_BaseplateType_BasedData()
                if 'baseDepth' in [PrimaryParameter, SecondaryParameter]: #Patch: baseDepth
                should be redefined to prevent it from being overwritten by the default baseDepth
                    if DataArray['BasePlate'] <> False:
                        if PrimaryParameter == 'baseDepth': DataArray['baseDepth'] = Param1
                        elif SecondaryParameter == 'baseDepth': DataArray['baseDepth'] = Param2
            if DataArray['ColumnType'] == 'Square' or DataArray['ColumnType'] == 'Rectangle':
            assert DataArray['StrongAxis'] == True
            if 'StrongOrient' in [PrimaryParameter, SecondaryParameter]: DataArray['DistLoad'] =
             not DataArray['StrongAxis'] #Is this functioning as a distributed load (as opposed
            to a point load)?
            Define_eL_BasedData() #This goes after BaseplateType because BaseplateType affects
            baseDepth, which in turn affects cmd, which is in eL_BasedData
            #Model Name - depends on Param1, Param2, and eL.
            ModelName = '%s%s_%s%s_eL%s' %(PrimaryParameter, __removeDot(Param1),
            SecondaryParameter, __removeDot(Param2), __removeDot(eL))
            #Other metadata that depends on ModelName.
            DataArray['ModelName'] = ModelName
            DataArray['mdbFileName'] = WorkingDir + '/' + ModelName
            DataArray['odbFileName'] = WorkingDir + '/' + ModelName + '.odb'

            print(DataArray)
            ######Run the script, already!#########
            if ScriptType == 'Preprocessing':
                # from Scripts import Preprocessing
                Preprocessing(DataArray)
            elif ScriptType == 'Postprocessing':
                # from Scripts import Postprocessing
```

```
            Postprocessing(DataArray)
        else:
            raise TypeError('Unexpected Script Type.')
```

```
########################
#Copy/paste from here...

Linux = 1
SuperComputing = 0

ModelType = 'CohesiveZone'
BaseplateType = 'Square' #BaseplateType Possibilities: 'Square', 'Rectangle', 'Reduced', 'None'

PrimaryParameter = 'BCs'
PrimaryParameterList = ['Bottom', 'Sides']
SecondaryParameter = 'MeshSize'
SecondaryParameterList = [0.5]
EmbedDepthsList = [1.5, 5.5, 9.5, 13.5, 17.5]
# EmbedDepthsList = [1.5]

#...to here            #
########################

ScriptType = 'Postprocessing'

########################
#Initialization tasks.#
########################
if SuperComputing: assert Linux
global DataArray
import os
#Supercomputing Linux, normal Linux, or Windows.
if Linux:
    if SuperComputing:
        Heading = '/fslhome/trevdna/'
    else:
        Heading = '/fsc/trevdna/'
else:#Windows
    Heading = 'J:/'

if SuperComputing:
    os.chdir('/fslhome/trevdna/compute/Models')
else:
    os.chdir(Heading + 'groups/researchtaj/scratch/ColumnModels_CohesiveZone/')

#Import functions
import csv
from string import join
from sys import path
from math import sqrt
path.append(Heading + 'Scripts/')
if ScriptType == 'Preprocessing':
    from Scripts import Preprocessing
    # pass
elif ScriptType == 'Postprocessing':
    from Scripts import Postprocessing
else:
```

```
    raise TypeError('Unexpected Script Type.')

#Dictionary with part names and properties
PropertiesDict = {}
ShapesDatabase = Heading + 'Research/4-InputDatabases/ShapesDatabase_Custom.csv'
if SuperComputing: ShapesDatabase = Heading + 'InputDatabases/ShapesDatabase_Custom.csv'
with open(ShapesDatabase) as csvfile:
    quoting = csv.QUOTE_NONNUMERIC
    reader = csv.reader(csvfile)
    for row in reader:
        PropertiesDict[row[0]] = row[3], row[4], row[6], row[8], \
            row[11], row[13], row[18], row[22]
'''0: bA - beam Area
1: db - beam depth
2: bf - Flange width
3: tw - Thickness of web
4: tf - thickness of flange
5: k(des) - smallest possible k value
6: Ix - Strong moment of inertia
7: Iy - Weak moment of inertia'''

WorkingDir = os.getcwd()

def __removeDot(str1):
    return join(str(str1).split('.'),'point')

##########################################################################
#Create dictionary with needed information to import into Abaqus routines.#
##########################################################################

DataArray = {}
#Metadata
DataArray['ModelType'] = ModelType #Contact type
DataArray['TwoD_ThreeD'] = False #Does it taper to 2D from 3D?
DataArray['OneD_TwoD'] = False
DataArray['PrimaryParameter'] = PrimaryParameter
DataArray['SecondaryParameter'] = SecondaryParameter
DataArray['ColumnType'] = 'IBeam' #'IBeam', 'Rectangle', or 'Square'
# DataArray['CohesiveZone'] = False

#Setup Parameters (Default)
DataArray['StrongOrient'] = True
DataArray['BasePlate'] = True
def Define_ModelType_BasedData():
    if DataArray['ModelType'] == 'Friction' or DataArray['ModelType'] == 'CohesiveZone':
        DataArray['OnePartModel'] = False
    elif DataArray['ModelType'] == 'RigidTie': #RigidTie, Tied, whatever I called it that day.
        DataArray['OnePartModel'] = True

    if DataArray['OnePartModel']:
        DataArray['ColumnPart'], DataArray['FoundationPart'] = 'CombinedPart', 'CombinedPart'
    else:
        DataArray['ColumnPart'], DataArray['FoundationPart'] = 'Column', 'Foundation'
```

```
    if DataArray['ModelType'] == 'CohesiveZone':
        DataArray['CohesiveZone'] = True
        DataArray['ColumnPart'], DataArray['FoundationPart'] = 'CombinedPart-CZ',
        'CombinedPart-CZ'
    else:
        DataArray['CohesiveZone'] = False

Define_ModelType_BasedData()
DataArray['BlockoutConcrete'] = False

#File paths
outputFileFolder = Heading + 'Research/1-ThesisResearch/'
if SuperComputing: outputFileFolder = Heading + 'RawOutputFiles/'
DataArray['outputFile'] = '%s%s%s.csv' %(outputFileFolder,PrimaryParameter,SecondaryParameter)

#Column properties

#Square/rectangular column properties
DataArray['cX'] = 6.855
DataArray['cY'] = 6.855
DataArray['StrongAxis'] = True
DataArray['ColumnName'] = 'W8X35'
def Define_ColumnName_BasedData():
    global DataArray
    DataArray['db'] = float(PropertiesDict[DataArray['ColumnName']][1])
    DataArray['tw'] = float(PropertiesDict[DataArray['ColumnName']][3])
    DataArray['bf'] = float(PropertiesDict[DataArray['ColumnName']][2])
    DataArray['tf'] = float(PropertiesDict[DataArray['ColumnName']][4])
    if DataArray['ColumnType'] == 'IBeam':
        DataArray['Ix'] = float(PropertiesDict[DataArray['ColumnName']][6])
        DataArray['Iy'] = float(PropertiesDict[DataArray['ColumnName']][7])
    elif DataArray['ColumnType'] == 'Square' or DataArray['ColumnType'] == 'Rectangle':
        DataArray['Ix'] = float(cX*cY**3/12)
        DataArray['Iy'] = float(cY*cX**3/12)
    k = float(PropertiesDict[DataArray['ColumnName']][5])
    DataArray['fr'] = k - DataArray['tf']
    db, tw, bf, tf = DataArray['db'], DataArray['tw'], DataArray['bf'], DataArray['tf']
    DataArray['SA'] = db*tw+2*bf*tf-2*tf*tw #Surface Area / cross sectional area
Define_ColumnName_BasedData()

#Baseplate properties
baseWidth = 13.0
baseWidthX = baseWidth - 2.0
baseWidthY = baseWidth
baseDepth = 1.0
DataArray['BaseplateType'] = BaseplateType #BaseplateType Possibilities: 'Square', 'Rectangle',
'Reduced', 'None'
def Define_BaseplateType_BasedData(): #Also includes changes to baseplate dimensions based on
column size
    global baseWidth, baseWidthX, baseWidthY
    if DataArray['ColumnName'] == 'W14X176': #Patches for individual test cases, not a
    universal solution here.
```

```
        baseWidth = 24.0
    elif DataArray['ColumnName'] == 'W24X76':
        assert BaseplateType == 'Rectangle'
        # baseWidth = 39.0
        baseWidthX = 20.0
        baseWidthY = 34.0
    else:
        # DataArray['BaseplateType'] = BaseplateType
        baseWidth = 13.0
        baseWidthX = baseWidth - 2.0
        baseWidthY = baseWidth

    if DataArray['BaseplateType'] == 'Square':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = baseWidth
        DataArray['baseWidthY'] = baseWidth
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'Rectangle':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = baseWidthX
        DataArray['baseWidthY'] = baseWidthY
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'Reduced':
        DataArray['BasePlate'] = True
        DataArray['baseWidthX'] = DataArray['bf']
        DataArray['baseWidthY'] = DataArray['db']
        DataArray['baseDepth'] = baseDepth
    elif DataArray['BaseplateType'] == 'None':
        DataArray['BasePlate'] = False
        # DataArray['baseWidthX'] = 0.0 #Should not be necessary
        # DataArray['baseWidthY'] = 0.0 #Should not be necessary
        DataArray['baseDepth'] = 0.0
    if DataArray['BaseplateType'] <> 'None':
        assert DataArray['baseWidthX'] >= DataArray['bf'] #Only valid in the case of
        strong-axis bending, FYI.
        assert DataArray['baseWidthY'] >= DataArray['db']
Define_BaseplateType_BasedData()

#Foundation properties
DataArray['mwX'] = 42 #Medium (foundation) width in x-direction
DataArray['mwY'] = 42 #Medium (foundation) width in y-direction
DataArray['BCs'] = 'Bottom'
# DataArray['blockoutSize'] = 17.0

#Column lengths
DataArray['pL'] = 80.25 #Protruding
DataArray['eL'] = 5.5 #Embedded

def Define_eL_BasedData():
    global DataArray
    DataArray['cL'] = DataArray['eL'] + DataArray['pL'] #Column length: embedded + protruding
    DataArray['cmd'] = 12.0 + DataArray['eL'] + DataArray['baseDepth'] + 1.5 #Default: 20.0
Define_eL_BasedData()
```

```
#Moduli and PRs; concrete strength
DataArray['EmbeddedSteelMod']   = 29000000.0
DataArray['NormalConcreteMod']  =  3500000.0
DataArray['SteelPr'] = 0.27 #Poisson's ratio, for steel
DataArray['ConcretePr'] = 0.15 #pr for concrete
DataArray['CohesiveMod'] = 5E4 #The pseudomodulus that is used in the cohesive zone material.
DataArray['CohesiveDepth'] = 0.01

def Define_EmbeddedSteelMod_BasedData():
    global DataArray
    DataArray['ProtrudingSteelMod'] = DataArray['EmbeddedSteelMod']
Define_EmbeddedSteelMod_BasedData()

def Define_NormalConcreteMod_BasedData():
    global DataArray
    DataArray['BadConcreteMod'] = DataArray['NormalConcreteMod']
    DataArray['GroutMod'] = DataArray['NormalConcreteMod']
    strength = (float(DataArray['NormalConcreteMod'])/57000)**2 #For reference
Define_NormalConcreteMod_BasedData()

#Load and friction values
# DataArray['DistLoad'] = True #Is this functioning as a distributed load (as opposed to a
point load)?
DataArray['DistLoad'] = not DataArray['StrongAxis'] #Is this functioning as a distributed load
(as opposed to a point load)?
DataArray['load'] = 1000 #Pounds
DataArray['AxialLoad'] = 0 #Pounds
DataArray['NoFriction'] = False
DataArray['Friction'] = 0.50
DataArray['NoSeparation'] = False

#Mesh sizes
DataArray['MeshSize'] = 0.5
DataArray['UniformMesh'] = True
DataArray['SquareMesh'] = True
DataArray['QuadMesh'] = False

################################
#Run the bloody script already!#
################################

#Loops
#Note: All the assertion lines in here are to make sure you don't try to vary two parameters
together that would result in bugs if you run them together.
#If you really want to run them together, code it yourself, and double (triple) check the code
actually behaves like you are expecting.
for Param1 in PrimaryParameterList:
    DataArray['Param1'] = Param1
    DataArray[PrimaryParameter] = Param1
    for Param2 in SecondaryParameterList:
        DataArray['Param2'] = Param2
        DataArray[SecondaryParameter] = Param2
```

```
    for eL in EmbedDepthsList:
        DataArray['eL'] = eL
        ######Update properties that are based on variables that may have changed.#######
        if 'ColumnName' in [PrimaryParameter, SecondaryParameter]:
            assert DataArray['ColumnType'] == 'IBeam'
            Define_ColumnName_BasedData()
        if 'EmbeddedSteelMod' in [PrimaryParameter, SecondaryParameter]:
        Define_EmbeddedSteelMod_BasedData()
        if 'NormalConcreteMod' in [PrimaryParameter, SecondaryParameter]:
        Define_NormalConcreteMod_BasedData()
        if 'OnePartModel' in [PrimaryParameter, SecondaryParameter]:
        Define_ModelType_BasedData()
        if 'ModelType' in [PrimaryParameter, SecondaryParameter]: Define_ModelType_BasedData
        ()
        if 'BaseplateType' in [PrimaryParameter, SecondaryParameter] or 'ColumnName' in [
        PrimaryParameter, SecondaryParameter]:
            # assert 'ColumnName' not in [PrimaryParameter, SecondaryParameter]
            Define_BaseplateType_BasedData()
            if 'baseDepth' in [PrimaryParameter, SecondaryParameter]: #Patch: baseDepth
            should be redefined to prevent it from being overwritten by the default baseDepth
                if DataArray['BasePlate'] <> False:
                    if PrimaryParameter == 'baseDepth': DataArray['baseDepth'] = Param1
                    elif SecondaryParameter == 'baseDepth': DataArray['baseDepth'] = Param2
        if DataArray['ColumnType'] == 'Square' or DataArray['ColumnType'] == 'Rectangle':
        assert DataArray['StrongAxis'] == True
        if 'StrongOrient' in [PrimaryParameter, SecondaryParameter]: DataArray['DistLoad'] =
         not DataArray['StrongAxis'] #Is this functioning as a distributed load (as opposed
        to a point load)?
        Define_eL_BasedData() #This goes after BaseplateType because BaseplateType affects
        baseDepth, which in turn affects cmd, which is in eL_BasedData
        #Model Name - depends on Param1, Param2, and eL.
        ModelName = '%s%s_%s%s_eL%s' %(PrimaryParameter, __removeDot(Param1),
        SecondaryParameter, __removeDot(Param2), __removeDot(eL))
        #Other metadata that depends on ModelName.
        DataArray['ModelName'] = ModelName
        DataArray['mdbFileName'] = WorkingDir + '/' + ModelName
        DataArray['odbFileName'] = WorkingDir + '/' + ModelName + '.odb'

        print(DataArray)
        ######Run the script, already!#########
        if ScriptType == 'Preprocessing':
            # from Scripts import Preprocessing
            Preprocessing(DataArray)
        elif ScriptType == 'Postprocessing':
            # from Scripts import Postprocessing
            Postprocessing(DataArray)
        else:
            raise TypeError('Unexpected Script Type.')
```

```
###############################################
#Import lines and other initialization tasks.#
###############################################

from abaqus import *
from abaqusConstants import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior

import os
from linecache import getline
from math import sqrt
# from datetime import datetime

session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=COORDINATE)

###########################################################################################
#After this point is the subroutines that feed into the pre- and post-processing routines.#
###########################################################################################

def __TimeStamp():
    global TimeStamp
    from datetime import datetime
    month = str(datetime.now().month)
    day = str(datetime.now().day)
    year = str(datetime.now().year)
    hour = str(datetime.now().hour)
    minute = str(datetime.now().minute)
    second = str(datetime.now().second)
    return '{0}-{1}-{2}_{3}-{4}-{5}'.format(month, day, year, hour, minute, second)

def __openwrite(outputFile):
    with open(outputFile, 'a') as f:
        f.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,\n' %('Model Name', 'Primary Parameter',
        'Primary Value', \
            'Secondary Parameter', 'Secondary Value', 'Embedment Length', 'Column Shape',
            'Total Displacement', 'Connection Stiffness', \
```

```
            'Connection Rotational Stiffness', 'Timestamp'))

def __filter(largeGroup, filteredGroup):
    return filter(lambda x: x not in filteredGroup, largeGroup)

def CreateModel():
#Model creation
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    print(ModelName)
    mdb.Model(name=ModelName, modelType=STANDARD_EXPLICIT)

def CreateAndCheckOutputFile():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    if os.path.exists(outputFile) == False:
        __openwrite(outputFile)
    else:
        try:
            with open(outputFile, 'a') as f:
                f.write('')
        except IOError:
            outputFile = outputFile[0:-4] + '(2).csv'
            if os.path.exists(outputFile) == False:
                __openwrite(outputFile)
            else:
                try:
                    with open(outputFile, 'a') as f:
                        f.write('')
                except IOError:
                    raise IOError

def ColumnCreation():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    s = mdb.models[ModelName].ConstrainedSketch(name='__profile__',
        sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    if ColumnType == 'IBeam' and StrongAxis == True:
        s.rectangle(point1=(0, (-db / 2) + tf), point2=((tw / 2),(db / 2) - tf))
        s.rectangle(point1=(0, -db/2), point2=(bf/2, -db/2 + tf))
        s.rectangle(point1=(0, db/2 - tf), point2=(bf/2, db/2))
        s.autoTrimCurve(curve1=g.findAt((bf / 2 - 0.001, -db/2 + tf)), point1=(0.001, -db/2 + tf
        ))
        s.autoTrimCurve(curve1=g.findAt((0.001, -db/2 + tf)), point1=(0.001, -db/2 + tf))
        s.autoTrimCurve(curve1=g.findAt((bf / 2 - 0.001, db/2 - tf)), point1=(0.001, db/2 - tf))
        s.autoTrimCurve(curve1=g.findAt((0.001, db/2 - tf)), point1=(0.001, db/2 - tf))
    elif ColumnType == 'IBeam' and StrongAxis == False:
        s.rectangle(point1=(0, tw/2), point2=(db/2 - tf, -tw/2))
        s.rectangle(point1=(db/2-tf, bf/2), point2=(db/2, -bf/2))
        s.autoTrimCurve(curve1=g.findAt((db/2 - tf, 0.0)), point1=((db/2 - tf, 0.0)))
        s.autoTrimCurve(curve1=g.findAt((db/2 - tf, 0.0)), point1=((db/2 - tf, 0.0)))
```

```
p = mdb.models[ModelName].Part(name='Column', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models[ModelName].parts['Column']
p.BaseSolidExtrude(sketch=s, depth=cL)
mdb.models[ModelName].sketches.changeKey(fromName='__profile__',
    toName='ColumnSketch')
s.unsetPrimaryObject()

#Column Part Division
p = mdb.models[ModelName].parts['Column']
DatumPointID=p.DatumPointByCoordinate(coords=(0.0, 0.0, eL)).id
p = mdb.models[ModelName].parts['Column']
c = p.cells
pickedCells = c.findAt(((0.0, 0.0, 0.0), ))
e1, v2, d2 = p.edges, p.vertices, p.datums
if ColumnType == 'IBeam':
    if StrongAxis == True:
        coord = (0.0, db / 2, eL)
    elif StrongAxis == False:
        coord = (0.0, tw/2, eL)

p.PartitionCellByPlanePointNormal(point=d2[DatumPointID], normal=e1.findAt(coordinates=coord
), cells=pickedCells)

#Add baseplate
if DataArray['BasePlate']== True:
    p = mdb.models[ModelName].parts['Column']
    f, e = p.faces, p.edges
    if ColumnType == 'IBeam':
        if StrongAxis:
            coord1 = (tw/4, -db/4, 0.0)
            coord2 = (tw/2, 0.0, 0.0)
        else:
            coord1 = (db/2-tf/2, -bf/2, 0.0)
            coord2 = (db/2, 0.0, 0.0)
    t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=coord1), sketchUpEdge=e.
    findAt(coordinates=coord2),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.0))
    s = mdb.models[ModelName].ConstrainedSketch(name='__profile__',
        sheetSize=22.62, gridSpacing=0.56, transform=t)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models[ModelName].parts['Column']
    p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)

    s.rectangle(point1=(0, -baseWidthY / 2), point2=(baseWidthX / 2, baseWidthY / 2))

    p = mdb.models[ModelName].parts['Column']
    f1, e1 = p.faces, p.edges
    p.SolidExtrude(sketchPlane=f1.findAt(coordinates=coord1),
        sketchUpEdge=e1.findAt(coordinates=coord2),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s, depth=baseDepth,
```

```
            flipExtrudeDirection=OFF)
        mdb.models[ModelName].sketches.changeKey(fromName='__profile__',
            toName='BaseplateSketch')
    s.unsetPrimaryObject()

def DivideColumn():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    p = mdb.models[ModelName].parts['Column']
    c = p.cells
    f = p.faces
    if StrongAxis:
        #Dividing the top-down face into rectangular cells
        pickedCells = c[:]#Select all cells
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2 - (bf/2-tw/2)/2, db/2
        - tf, pL/2)), cells=pickedCells) #pL/2 will have to be reduced when converting to a
        beam with shell elements
        pickedCells = c[:]#Reselect all cells
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2 - (bf/2-tw/2)/2, -db/2
         + tf, pL/2)), cells=pickedCells)
    else:
        pickedCells = c[:]#Select all cells
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2-tf, tw/2 +(bf/2-tw/2)/
        2, pL/2)), cells=pickedCells)

    #Divisions with baseplate involved
    if BasePlate == True:
        if StrongAxis:
            if baseWidthX > bf:
                pickedCells = c[:]
                p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2, db/2 - tf/2,
                cmd-eL/2)), cells=pickedCells)
        elif not StrongAxis:
            if baseWidthX > db:
                pickedCells = c[:]
                p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2-tf/2, bf/2,
                cmd-eL/2)), cells=pickedCells)
                pickedCells = c[:]
                p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2-tf/2, -bf/2,
                cmd-eL/2)), cells=pickedCells)
        pickedCells = c[:]
        if StrongAxis:
            p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(baseWidthX/4, 0,0)),
            cells=pickedCells)
        elif not StrongAxis:
            p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(0.001, baseWidthY/4,0
            )), cells=pickedCells)

def CreateSet():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Partition the top edge of the middle of the column
    p = mdb.models[ModelName].parts['Column']
```

```
    e = p.edges
    pickedEdges = e.findAt(((0.0, 0.0, cL), ))
    p.PartitionEdgeByParam(edges=pickedEdges, parameter=0.5)
    #Create set for strong applied load
    p = mdb.models[ModelName].parts['Column']
    v = p.vertices
    verts = v.findAt(((0.0, 0.0, cL), ))
    p.Set(vertices=verts, name='Set-1')

def SketchFoundation():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Continuum Part Creation
    s = mdb.models[ModelName].ConstrainedSketch(name='__profile__',
        sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.rectangle(point1=(0, - mwY / 2), point2=(mwX / 2, mwY / 2))
    p = mdb.models[ModelName].Part(name='Foundation', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models[ModelName].parts['Foundation']
    p.BaseSolidExtrude(sketch=s, depth=cmd)
    s.unsetPrimaryObject()
    del mdb.models[ModelName].sketches['__profile__']

    #Cut hole for column.
    f, e = p.faces, p.edges
    t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(0.0, 0.0,
        cmd)), sketchUpEdge=e.findAt(coordinates=(mwX/2, 0.0, cmd)),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0,
        cmd))
    s1 = mdb.models[ModelName].ConstrainedSketch(name='__profile__',
        sheetSize=27.71, gridSpacing=0.69, transform=t)
    g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.setPrimaryObject(option=SUPERIMPOSE)
    p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
    s1.retrieveSketch(sketch=mdb.models[ModelName].sketches['ColumnSketch'])
    f1, e1 = p.faces, p.edges
    p.CutExtrude(sketchPlane=f1.findAt(coordinates=(0.0, 0.0, cmd)),
        sketchUpEdge=e1.findAt(coordinates=(mwX/2, 0.0, cmd)),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=eL,
        flipExtrudeDirection=OFF)
    s1.unsetPrimaryObject()
    del mdb.models[ModelName].sketches['__profile__']

    #Cut hole for baseplate.
    if BasePlate == True:
        f, e = p.faces, p.edges
        if ColumnType == 'IBeam':
            if StrongAxis:
                coord = (tw / 10, -db / 10, cmd- eL)
            elif not StrongAxis:
                coord = (db / 10, 0.0, cmd- eL)
```

```
        t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=coord),
            sketchUpEdge=e.findAt(coordinates=(0.0, 0.0, cmd - eL)),
            sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0,
            cmd - eL))
        s1 = mdb.models[ModelName].ConstrainedSketch(name='__profile__',
            sheetSize=27.71, gridSpacing=0.69, transform=t)
        g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
        s1.setPrimaryObject(option=SUPERIMPOSE)
        p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
        s1.rectangle(point1=(0, -baseWidthY / 2), point2=(baseWidthX / 2, baseWidthY / 2))
        f1, e1 = p.faces, p.edges
        p.CutExtrude(sketchPlane=f1.findAt(coordinates=coord),
            sketchUpEdge=e1.findAt(coordinates=(0.0, 0.0, cmd - eL)),
            sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=baseDepth,
            flipExtrudeDirection=OFF)
        s1.unsetPrimaryObject()
        del mdb.models[ModelName].sketches['__profile__']

def DivideFoundation():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    p = mdb.models[ModelName].parts['Foundation']
    c = p.cells
    f = p.faces

    coh = offsetVal

    #Dividing the top-down face into rectangular cells
    if StrongAxis:
        pickedCells = c[:]#Select all cells
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2, db/2 - tf/2, cmd-eL/2
        )), cells=pickedCells)
        pickedCells = c[:]#Reselect all cells
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2 - (bf/2-tw/2)/2, db/2
        - tf, cmd - eL/2)), cells=pickedCells)
        pickedCells = c[:]
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(bf/2 - (bf/2-tw/2)/2, -db/2
         + tf, cmd - eL/2)), cells=pickedCells)
    elif not StrongAxis:
        pickedCells = c[:]
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2 - tf, tw/2 + 0.001,
        cmd - eL/2)), cells=pickedCells)
        pickedCells = c[:]
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2-tf/2, bf/2 , cmd-eL/2
        )), cells=pickedCells)
        pickedCells = c[:]
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(db/2-tf/2,-bf/2, cmd - eL/2
        )), cells=pickedCells)

    #Nothing below here needs to change for strong/weak axis bending
    #Dividing the side-view face into rectangular cells
    if BaseplateType == 'Square' or BaseplateType == 'Rectangle':
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(baseWidthX / 2 - 0.001, -
```

```
        baseWidthY / 2 + 0.001, cmd-eL)), cells=c[:])
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(baseWidthX/4, -baseWidthY/2
        , cmd-eL-baseDepth/2)), cells=c[:])
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(baseWidthX/4, baseWidthY/2,
         cmd-eL-baseDepth/2)), cells=c[:])
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates=(baseWidthX/4, 0, cmd-eL-
        baseDepth)), cells=c[:])
    else: #This will work for either a reduced bp or none at all.
        p.PartitionCellByExtendFace(extendFace=f.findAt(coordinates = (bf/4, db/2 - tf/4, cmd-eL
        -baseDepth)), cells=c[:])

    if CohesiveZone:
        p = mdb.models[ModelName].parts['Foundation']
        d = p.datums
        f = p.faces
        c = p.cells
        #Create datum planes to partition cohesive zones
        if StrongAxis:
            topFlangeTopID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=db/2 +
            offsetVal).id
            topFlangeBotID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=db/2 -
            tf - offsetVal).id
            botFlangeTopID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=-db/2 +
             tf + offsetVal).id
            botFlangeBotID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=-db/2 -
             offsetVal).id
            webID = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset=tw/2 + offsetVal
            ).id
            flangeEdgeID = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset=bf/2 +
            offsetVal).id

            #Create partitions
            p.PartitionCellByExtendFace(extendFace=f.findAt((bf/4, db/2, cmd-eL/2),), cells=c[:])
            p.PartitionCellByExtendFace(extendFace=f.findAt((bf/4, -db/2, cmd-eL/2),), cells=c
            [:])
            #Top flange top
            pickedCells = c.findAt((bf/4, db/2 + offsetVal/2, cmd-eL/2),)
            p.PartitionCellByDatumPlane(datumPlane=d[topFlangeTopID], cells=pickedCells)
            #Top flange bottom
            c = p.cells
            pickedCells = c.findAt((bf/4, db/2 - tf - offsetVal/2, cmd-eL/2),)
            p.PartitionCellByDatumPlane(datumPlane=d[topFlangeBotID], cells=pickedCells)
            #Bottom flange top
            c = p.cells
            pickedCells = c.findAt((bf/4, -db/2 + tf + offsetVal/2, cmd-eL/2),)
            p.PartitionCellByDatumPlane(datumPlane=d[botFlangeTopID], cells=pickedCells)
            #Bottom flange bottom
            c = p.cells
            pickedCells = c.findAt((bf/4, -db/2 - offsetVal/2, cmd-eL/2),)
            p.PartitionCellByDatumPlane(datumPlane=d[botFlangeBotID], cells=pickedCells)
            #Web
            c = p.cells
            pickedCells = c.findAt((tw/2 + offsetVal/2,0.0, cmd-eL/2),)
```

```
p.PartitionCellByDatumPlane(datumPlane=d[webID], cells=pickedCells)
#Top flange edge
c = p.cells
pickedCells = c.findAt((bf/2 + offsetVal/2, db/2 - tf/2, cmd-eL/2),)
p.PartitionCellByDatumPlane(datumPlane=d[flangeEdgeID], cells=pickedCells)
#Bottom flange edge
c = p.cells
pickedCells = c.findAt((bf/2 + offsetVal/2, -db/2 + tf/2, cmd-eL/2),)
p.PartitionCellByDatumPlane(datumPlane=d[flangeEdgeID], cells=pickedCells)

#Corners
if BaseplateType <> 'Reduced' and BaseplateType <> 'None': #As it is, the corner
divisions are only cosmetic (until I can actually assign them cohesive elements and
properties). So, since the corners are giving me mesh problems for reduced
baseplate models, I'll take them out.
    #Corners - sketch

    p = mdb.models[ModelName].parts['Foundation']
    f, e, d = p.faces, p.edges, p.datums
    t = p.MakeSketchTransform(sketchPlane=f.findAt((tw/2 + coh + 0.001, 0.0, cmd),),
     sketchUpEdge=e.findAt((bf/2, 0.0, cmd),),
        sketchPlaneSide=SIDE1, origin=(0.0, 0.0, cmd))
    s = mdb.models[ModelName].ConstrainedSketch(
        name='__profile__', sheetSize=23.96, gridSpacing=0.59, transform=t)
    g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models[ModelName].parts['Foundation']
    p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)

    s.rectangle(point1=(bf/2, db/2), point2=(bf/2 + coh, db/2+coh))#Top flange, top
    corner
    s.rectangle(point1=(bf/2, db/2 - tf), point2=(bf/2 + coh, db/2 - tf - coh))#Top
    flange, bot corner
    s.rectangle(point1=(bf/2, -db/2 + tf), point2=(bf/2 + coh, -db/2 + tf + coh))
    #Bot flange, top corner
    s.rectangle(point1=(bf/2, -db/2), point2=(bf/2 + coh, -db/2 - coh))#Bot flange,
    bot corner

    p = mdb.models[ModelName].parts['Foundation']
    f = p.faces
    pickedFaces = (f.findAt((bf/2 + coh, db/2 + coh, cmd),), f.findAt((bf/2 + coh,
    0.0, cmd),), f.findAt((bf/2 + coh, -db/2 - coh, cmd),))
    e1, d2 = p.edges, p.datums
    p.PartitionFaceBySketch(sketchUpEdge=e1.findAt((bf/2, 0.0, cmd),), faces=
    pickedFaces, sketch=s)
    s.unsetPrimaryObject()
    del mdb.models[ModelName].sketches['__profile__']

    #Corners - division

    p = mdb.models[ModelName].parts['Foundation']
    c = p.cells
    e, d = p.edges, p.datums
```

```
        pickedCells1 = c.findAt((bf/2 + coh, db/2 + coh, cmd-eL/2),)
        pickedEdges1 =(e.findAt((bf/2+coh, db/2+coh/2, cmd),), e.findAt((bf/2 + coh/2,
        db/2+coh, cmd),)) #top flange top corner
        p.PartitionCellByExtrudeEdge(line=e.findAt((bf/2, db/2, cmd-0.001),), cells=
        pickedCells1, edges=pickedEdges1,
            sense=FORWARD)

        pickedCells2 = c.findAt((bf/2 + coh + 0.001, 0.0, cmd-eL/2),)
        pickedEdges2 =(e.findAt((bf/2+coh, db/2-tf-coh/2, cmd),), e.findAt((bf/2+coh/2,
        db/2-tf-coh, cmd),)) #top flange bot corner
        p.PartitionCellByExtrudeEdge(line=e.findAt((bf/2, db/2-tf, cmd-0.001),), cells=
        pickedCells2, edges=pickedEdges2,
            sense=FORWARD)

        pickedCells2 = c.findAt((bf/2 + coh + 0.001, 0.0, cmd-eL/2),)
        pickedEdges3 = (e.findAt((bf/2+coh, -db/2+tf+coh/2, cmd),), e.findAt((bf/2+coh/2
        , -db/2+tf+coh, cmd),)) #bot flange top
        p.PartitionCellByExtrudeEdge(line=e.findAt((bf/2, -db/2+tf, cmd-0.001),), cells=
        pickedCells2, edges=pickedEdges3,
            sense=FORWARD)

        c = p.cells
        e, d = p.edges, p.datums
        pickedCells3 = c.findAt((bf/2 + coh, -db/2 - coh, cmd-eL/2),)
        pickedEdges4 = (e.findAt((bf/2+coh, -db/2-coh/2, cmd),), e.findAt((bf/2+coh/2, -
        db/2-coh, cmd),)) #bot flange bot
        p.PartitionCellByExtrudeEdge(line=e.findAt((bf/2, db/2, cmd-0.001),), cells=
        pickedCells3, edges=pickedEdges4,
            sense=FORWARD)

elif not StrongAxis:
    webTopID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=tw/2 +
    offsetVal).id
    webBotID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=-tw/2 -
    offsetVal).id
    flangeLeftID = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset=db/2 - tf
     - offsetVal).id
    flangeRightID = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset=db/2 +
    offsetVal).id
    flangeTopID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=bf/2 +
    offsetVal).id
    flangeBotID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=-bf/2 -
    offsetVal).id

    #Create partitions
    p.PartitionCellByExtendFace(extendFace=f.findAt((db/2, 0.0, cmd-eL/2),), cells=c[:])
    #Web top
    c = p.cells
    pickedCells = c.findAt((db/4, tw/2 + coh, cmd-eL/2),)
    p.PartitionCellByDatumPlane(datumPlane=d[webTopID], cells=pickedCells)
    #Web bot
    c = p.cells
```

```
        pickedCells = c.findAt((db/4, -tw/2 - coh, cmd-eL/2),)
        p.PartitionCellByDatumPlane(datumPlane=d[webBotID], cells=pickedCells)
        #Flange Left
        c = p.cells
        pickedCells = c.findAt((db/2 - tf - coh, tw/2 + coh + 0.001, cmd-eL/2), )
        p.PartitionCellByDatumPlane(datumPlane=d[flangeLeftID], cells=pickedCells)
        c = p.cells
        pickedCells = c.findAt((db/2 - tf - coh, -tw/2 - coh - 0.001, cmd-eL/2), )
        p.PartitionCellByDatumPlane(datumPlane=d[flangeLeftID], cells=pickedCells)
        #Flange Right
        c = p.cells
        pickedCells = c.findAt((db/2 + coh, 0.0 , cmd-eL/2),)
        p.PartitionCellByDatumPlane(datumPlane=d[flangeRightID], cells=pickedCells)
        #Flange Top
        c = p.cells
        pickedCells = c.findAt((db/2 - tf/2, bf/2 + coh , cmd-eL/2),)
        p.PartitionCellByDatumPlane(datumPlane=d[flangeTopID], cells=pickedCells)
        #Flange Bot
        c = p.cells
        pickedCells = c.findAt((db/2 - tf/2, -bf/2 - coh , cmd-eL/2),)
        p.PartitionCellByDatumPlane(datumPlane=d[flangeBotID], cells=pickedCells)

    if BasePlate: #Create divisions for the cohesive zone around the baseplate.
        baseplateBotID = p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset = cmd -
         eL - baseDepth - offsetVal).id
        baseplateTopID = p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset = cmd -
         eL + offsetVal).id
        baseplateUpID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset =
        baseWidthY / 2 + offsetVal).id
        baseplateDownID = p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset = -
        baseWidthY / 2 - offsetVal).id
        baseplateSideID = p.DatumPlaneByPrincipalPlane(principalPlane=YZPLANE, offset =
        baseWidthX / 2 + offsetVal).id

        d = p.datums
        f = p.faces
        c = p.cells
        if BaseplateType == 'Square' or BaseplateType == 'Rectangle':
            p.PartitionCellByExtendFace(extendFace=f.findAt((baseWidthX/2, 0.0, cmd-eL -
            baseDepth/2),), cells=c[:])
        c = p.cells
        pickedCells = c.getByBoundingBox(zMax = cmd - eL - baseDepth, xMax = baseWidthX / 2,
         yMin = -baseWidthY / 2, yMax = baseWidthY / 2)
        p.PartitionCellByDatumPlane(datumPlane=d[baseplateBotID], cells=pickedCells)
        c = p.cells
        pickedCells = c.getByBoundingBox(zMin = cmd - eL, xMax = baseWidthX / 2, yMin = -
        baseWidthY / 2, yMax = baseWidthY / 2)
        p.PartitionCellByDatumPlane(datumPlane=d[baseplateTopID], cells=pickedCells)
        if BaseplateType == 'Square' or BaseplateType == 'Rectangle':
            c = p.cells
            pickedCells = c.getByBoundingBox(xMax = baseWidthX / 2, yMin = baseWidthY / 2,
            zMin = cmd - eL - baseDepth, zMax = cmd - eL)
            p.PartitionCellByDatumPlane(datumPlane=d[baseplateUpID], cells=pickedCells)
```

```
                c = p.cells
                pickedCells = c.getByBoundingBox(xMax = baseWidthX / 2, yMax = -baseWidthY / 2,
                zMin = cmd - eL - baseDepth, zMax = cmd - eL)
                p.PartitionCellByDatumPlane(datumPlane=d[baseplateDownID], cells=pickedCells)
                c = p.cells
                pickedCells = c.getByBoundingBox(xMin = baseWidthX / 2, yMin = -baseWidthY / 2,
                yMax = baseWidthY / 2, zMin = cmd - eL - baseDepth, zMax = cmd - eL)
                p.PartitionCellByDatumPlane(datumPlane=d[baseplateSideID], cells=pickedCells)

        else:
            pass #Create divisions for the cohesive zone around the bottom of the column

def CreateMaterials_DefineSections():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    m = mdb.models[ModelName]
    m.Material(name='EmbeddedSteel')
    m.materials['EmbeddedSteel'].Elastic(table=((
        EmbeddedSteelMod, SteelPr), ))
    m.Material(name='Foundation')
    m.materials['Foundation'].Elastic(table=((
        NormalConcreteMod, ConcretePr), ))
    m.Material(name='ProtrudingSteel')
    m.materials['ProtrudingSteel'].Elastic(table=((
        ProtrudingSteelMod, SteelPr), ))
    if CohesiveZone:
        m.Material(name='Cohesive')
        m.materials['Cohesive'].Elastic(type=TRACTION, table=((CohesiveMod, CohesiveMod/2,
        CohesiveMod/2), ))

    # Create material section definitions.
    m.HomogeneousSolidSection(name='EmbeddedSteel',
        material='EmbeddedSteel', thickness=None)
    m.HomogeneousSolidSection(name='Foundation',
        material='Foundation', thickness=None)
    m.HomogeneousSolidSection(name='ProtrudingSteel',
        material='ProtrudingSteel', thickness=None)
    if CohesiveZone:
        m.CohesiveSection(name='Cohesive', material='Cohesive', response=TRACTION_SEPARATION,
            outOfPlaneThickness=None)

def SectionAssign_OneMaterial():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #All concrete is one modulus; all steel is another.
    coh = offsetVal
    #Protruding Column
    p = mdb.models[ModelName].parts['Column']
    c = p.cells
    cells = c[:]
    region = regionToolset.Region(cells=cells)
    p.SectionAssignment(region=region, sectionName='ProtrudingSteel', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
```

```
        thicknessAssignment=FROM_SECTION)
    #Foundation

    #Cohesive zone cells, if needed
    p = mdb.models[ModelName].parts['Foundation']
    c = p.cells
    if CohesiveZone == True:#Assign the cohesive zone section to those areas that are in the
    cohesive zone.
        if StrongAxis:
            cells1 = c.findAt(((bf/4, db/2 + offsetVal/2, cmd-eL/2),))
            cells2 = c.findAt(((bf/4, db/2 - tf - offsetVal/2, cmd-eL/2),))
            cells3 = c.findAt(((bf/4, -db/2 + tf + offsetVal/2, cmd-eL/2),))
            cells4 = c.findAt(((bf/4, -db/2 - offsetVal/2, cmd-eL/2),))
            cells5 = c.findAt(((tw/2 + offsetVal/2,0.0, cmd-eL/2),))
            cells6 = c.findAt(((bf/2 + offsetVal/2, db/2 - tf/2, cmd-eL/2),))
            cells7 = c.findAt(((bf/2 + offsetVal/2, -db/2 + tf/2, cmd-eL/2),))
            pass #corners
            cohesiveCells = cells1 + cells2 + cells3 + cells4 + cells5 + cells6 + cells7
        elif not StrongAxis:
            cells1 = c.findAt(((db/4, tw/2 + coh/2, cmd-eL/2),))
            cells2 = c.findAt(((db/4, -tw/2 - coh/2, cmd-eL/2),))
            cells3 = c.findAt(((db/2 - tf - coh/2, db/4, cmd-eL/2),))
            cells4 = c.findAt(((db/2 - tf - coh/2, -db/4, cmd-eL/2),))
            cells5 = c.findAt(((db/2 + coh/2, 0.0, cmd-eL/2),))
            cells6 = c.findAt(((db/2 - tf/2, bf/2 + coh/2, cmd-eL/2),))
            cells7 = c.findAt(((db/2 - tf/2, -bf/2 - coh/2, cmd-eL/2),))
            pass #corners
            cohesiveCells = cells1 + cells2 + cells3 + cells4 + cells5 + cells6 + cells7
        if BasePlate:
            cells8 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2, yMax
            = baseWidthY/2, zMin=cmd-eL, zMax=cmd-eL+offsetVal)#Beneath baseplate
            cells9 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2, yMax
            = baseWidthY/2, zMax=cmd-eL-baseDepth, zMin=cmd-eL-baseDepth-offsetVal) #Below
            baseplate
            cells10 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = baseWidthY/2, yMax
            = baseWidthY/2+offsetVal, zMin=cmd-eL-baseDepth, zMax=cmd-eL) #Up-baseplate side
            cells11 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2-
            offsetVal, yMax = -baseWidthY/2, zMin=cmd-eL-baseDepth, zMax=cmd-eL)
            #Down-baseplate side
            cells12 = c.getByBoundingBox(xMin=baseWidthX/2, xMax=baseWidthX/2+offsetVal, yMin =
            -baseWidthY/2, yMax = baseWidthY/2, zMin=cmd-eL-baseDepth, zMax=cmd-eL)
            #Right-baseplate side
            cohesiveCells =cohesiveCells + cells8 + cells9 + cells10 + cells11 + cells12
        region = regionToolset.Region(cells=cohesiveCells)
        p.SectionAssignment(region=region, sectionName='Cohesive', offset=0.0,
            offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=FROM_SECTION)

    #Concrete cells
    p = mdb.models[ModelName].parts['Foundation']
    c = p.cells
    if CohesiveZone == False:
        cells = c[:]
        region = regionToolset.Region(cells=cells)
```

```
        p.SectionAssignment(region=region, sectionName='Foundation', offset=0.0,
            offsetType=MIDDLE_SURFACE, offsetField='',
            thicknessAssignment=FROM_SECTION)
    else: #I'm really proud of this part of code. Took me forever to come up with it right. -TAJ
        #Get a list of all the indices of the cells
        bigList = []
        for cell in c:
            bigList += [cell.index]
        #List of indices to be filtered
        smallList = []
        for cell in cohesiveCells:
            smallList += [cell.index]
        #Filter - now we have a list of all indices we want
        cellsList = __filter(bigList, smallList)
        #Loop through each index; grab that cell, give it a section assignment.
        for index in cellsList:
            i = int(index) #Not needed?
            region = regionToolset.Region(cells=c[index:(index+1)])
            p.SectionAssignment(region=region, sectionName='Foundation', offset=0.0,
                offsetType=MIDDLE_SURFACE, offsetField='',
                thicknessAssignment=FROM_SECTION)

def CreateLoadStep():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Create load step
    mdb.models[ModelName].StaticStep(name='Load', previous='Initial', maxNumInc=500)

def AssemblyInstance():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Create assembly
    a = mdb.models[ModelName].rootAssembly
    a.DatumCsysByDefault(CARTESIAN)
    p = mdb.models[ModelName].parts['Column']
    a.Instance(name='Column-1', part=p, dependent=ON)
    a = mdb.models[ModelName].rootAssembly
    p = mdb.models[ModelName].parts['Foundation']
    a.Instance(name='Foundation-1', part=p, dependent=ON)
    #Align assembly
    a = mdb.models[ModelName].rootAssembly
    a.translate(instanceList=('Column-1', ), vector=(0.0, 0.0, cmd - eL))

def MergeInstances():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    a1 = mdb.models[ModelName].rootAssembly
    a1.InstanceFromBooleanMerge(name='CombinedPart', instances=(
        a1.instances['Column-1'], a1.instances['Foundation-1'], ),
        keepIntersections=ON, originalInstances=SUPPRESS, domain=GEOMETRY)

def Contact():
    for key in DataArray.keys():
```

```
        exec('%s = DataArray["%s"]' %(key, key))
    #Create face group
    a = mdb.models[ModelName].rootAssembly
    s1 = a.instances['Column-1'].faces

    if ColumnType == 'IBeam':
        if StrongAxis:
            #1-Column flange, exterior faces; 2-Top column flange, top face; 3- Top column
            flange, bottom face; 4-Web exterior face; 5-Bottom column flange, top face
            #6- Bottom column flange, bottom face; 7-Baseplate, upper face; 8-Baseplate, lower
            face; 9-Baseplate, top face; 10-Baseplate, exterior face; 11-Baseplate, bottom face
            side1Faces1 = s1.getByBoundingBox(xMin = bf/2, xMax = bf/2, zMax=cmd) + \
                s1.getByBoundingBox(yMin = db/2, yMax = db/2, zMax=cmd) + \
                s1.getByBoundingBox(yMin = db/2 - tf, yMax = db/2 - tf, zMax=cmd) + \
                s1.getByBoundingBox(xMin = tw/2, xMax = tw/2, zMax=cmd) + \
                s1.getByBoundingBox(yMin = - db/2 + tf, yMax = -db/2 + tf, zMax=cmd) + \
                s1.getByBoundingBox(yMin = -db/2, yMax = -db/2, zMax=cmd) + \
                s1.getByBoundingBox(zMin = cmd - eL, zMax = cmd - eL) + \
                s1.getByBoundingBox(zMin = cmd - eL - baseDepth, zMax = cmd - eL - baseDepth) + \
                s1.getByBoundingBox(yMin = baseWidthY/2, yMax = baseWidthY/2, zMax=cmd) + \
                s1.getByBoundingBox(xMin = baseWidthX/2, xMax = baseWidthX/2, zMax=cmd) + \
                s1.getByBoundingBox(yMin = -baseWidthY/2, yMax = -baseWidthY/2, zMax=cmd)
            #Warning: if other faces later starts becoming colinear with these faces, some of
            these methods will accidentally grab those planes as well. Be careful!
            region1=regionToolset.Region(side1Faces=side1Faces1)
        elif not StrongAxis:
            #1-Top column faces (web top, flange top-left, flange top) 2-Side column face
            (flange right) 3-Bottom column faces (web bottom, flange bottom-left, flange bottom)
            #4-Baseplate, upper face; 5-Baseplate, lower face; 6-Baseplate, top face;
            7-Baseplate, exterior face; 8-Baseplate, bottom face
            side1Faces1 = s1.getByBoundingBox(xMin = 0, xMax = db/2, yMin=tw/2, yMax=bf/2, zMin=
            cmd-eL, zMax=cmd) + \
                s1.getByBoundingBox(xMin = db/2, xMax = db/2, zMax=cmd) + \
                s1.getByBoundingBox(xMin = 0, xMax = db/2, yMin=-bf/2, yMax=-tw/2, zMin=cmd-eL,
                zMax=cmd) + \
                s1.getByBoundingBox(zMin = cmd - eL, zMax = cmd - eL) + \
                s1.getByBoundingBox(zMin = cmd - eL - baseDepth, zMax = cmd - eL - baseDepth) + \
                s1.getByBoundingBox(yMin = baseWidthY/2, yMax = baseWidthY/2) + \
                s1.getByBoundingBox(xMin = baseWidthX/2, xMax = baseWidthX/2) + \
                s1.getByBoundingBox(yMin = -baseWidthY/2, yMax = -baseWidthY/2)
            #Warning: if other faces later starts becoming colinear with these faces, some of
            these methods will accidentally grab those planes as well. Be careful!
            region1=regionToolset.Region(side1Faces=side1Faces1)

    s1 = a.instances['Foundation-1'].faces
    if ColumnType == 'IBeam':
        if StrongAxis:
            #Faces touching the web, then top flange, then bottom flange
            side1Faces1 = s1.getByBoundingBox(0, -db/2, cmd-eL, tw/2, db/2, cmd) + \
                s1.getByBoundingBox(0,db/2-tf,cmd-eL,bf/2,db/2, cmd ) + \
                s1.getByBoundingBox(0,-db/2,cmd-eL,bf/2,-(db/2 -tf), cmd)
        elif not StrongAxis:
            #Faces touching the web, then flange
```

```
            side1Faces1 = s1.getByBoundingBox(0, -tw/2, cmd-eL, db/2-tf, tw/2, cmd) + \
                s1.getByBoundingBox(db/2-tf,-bf/2,cmd-eL,db/2,bf/2,cmd )

    if BasePlate:
        side1Faces1 += s1.getByBoundingBox(0,-baseWidthY/2,cmd-eL-baseDepth,baseWidthX/2,
        baseWidthY/2,cmd-eL) #Faces touching the baseplate

    region2=regionToolset.Region(side1Faces=side1Faces1)

    if ModelType == 'Contact' or ModelType == 'Friction':
        #Create interaction properties
        mdb.models[ModelName].ContactProperty('IntProp-1')
        if not NoFriction:
            mdb.models[ModelName].interactionProperties['IntProp-1'].TangentialBehavior(
                formulation=PENALTY, directionality=ISOTROPIC, slipRateDependency=OFF,
                pressureDependency=OFF, temperatureDependency=OFF, dependencies=0,
                table=((Friction, ), ), shearStressLimit=None,
                maximumElasticSlip=FRACTION, fraction=0.005, elasticSlipStiffness=None)
        else:
            mdb.models[ModelName].interactionProperties['IntProp-1'].TangentialBehavior(
                formulation=FRICTIONLESS)

        if NoSeparation == True:
            separationVar = OFF
        else:
            separationVar = ON

        mdb.models[ModelName].interactionProperties['IntProp-1'].NormalBehavior(
            pressureOverclosure=HARD, allowSeparation=separationVar,
            constraintEnforcementMethod=DEFAULT)
        mdb.models[ModelName].ContactStd(name='Int-1', createStepName='Initial')

        #If column is stiffer:
        if EmbeddedSteelMod >= NormalConcreteMod:
            masterSurf = region1
            slaveSurf = region2
        else: #If continuum is stiffer
            masterSurf = region2
            slaveSurf = region1
        mdb.models[ModelName].SurfaceToSurfaceContactStd(name='Int-1',
            createStepName='Load', master=masterSurf, slave=slaveSurf, sliding=FINITE,
            thickness=ON, interactionProperty='IntProp-1',
            adjustMethod=NONE, initialClearance=OMIT, datumAxis=None,
            clearanceRegion=None)

    elif ModelType == 'RigidTie' or ModelType == 'Rigid' or ModelType == 'CohesiveZone':
        mdb.models[ModelName].Tie(name='Constraint-1', master=region1,
            slave=region2, positionToleranceMethod=COMPUTED, adjust=ON,
            tieRotations=ON, thickness=ON)
    else:
        raise TypeError('Unknown ModelType')

def MeshSeedGenerate_Uniform():
```

```
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    if DataArray['SquareMesh'] ==True:
        elementShape = [HEX, STRUCTURED]
    else:
        elementShape = [TET, FREE]
    coh = offsetVal

    if OnePartModel:
        p = mdb.models[ModelName].parts['CombinedPart']
        c = p.cells
        pickedRegions = c[:]
        p.setMeshControls(regions=pickedRegions, elemShape=elementShape[0], technique=
        elementShape[1])
        p.seedPart(size=MeshSize, deviationFactor=0.1, minSizeFactor=0.1)
        p.generateMesh()
    else:
        #Column seeding and generation
        p = mdb.models[ModelName].parts['Column']
        c = p.cells
        pickedRegions = c[:]
        p.setMeshControls(regions=pickedRegions, elemShape=elementShape[0], technique=
        elementShape[1])
        p.seedPart(size=MeshSize, deviationFactor=0.1, minSizeFactor=0.1)
        p.generateMesh()

        #Foundation seeding and generation
        p = mdb.models[ModelName].parts['Foundation']
        c = p.cells
        pickedRegions = c[:]
        # p.setMeshControls(regions=pickedRegions, elemShape=elementShape[0],
        technique=elementShape[1])
        p.seedPart(size=MeshSize, deviationFactor=0.1, minSizeFactor=0.1)
        if CohesiveZone:
            elemType1 = mesh.ElemType(elemCode=COH3D8, elemLibrary=STANDARD)
            elemType2 = mesh.ElemType(elemCode=COH3D6, elemLibrary=STANDARD)
            elemType3 = mesh.ElemType(elemCode=UNKNOWN_TET, elemLibrary=STANDARD)
            if StrongAxis:
                cells1 = c.findAt(((bf/4, db/2 + offsetVal/2, cmd-eL/2),))
                cells2 = c.findAt(((bf/4, db/2 - tf - offsetVal/2, cmd-eL/2),))
                cells3 = c.findAt(((bf/4, -db/2 + tf + offsetVal/2, cmd-eL/2),))
                cells4 = c.findAt(((bf/4, -db/2 - offsetVal/2, cmd-eL/2),))
                cells5 = c.findAt(((tw/2 + offsetVal/2,0.0, cmd-eL/2),))
                cells6 = c.findAt(((bf/2 + offsetVal/2, db/2 - tf/2, cmd-eL/2),))
                cells7 = c.findAt(((bf/2 + offsetVal/2, -db/2 + tf/2, cmd-eL/2),))
                cells = cells1 + cells2 + cells3 + cells4 + cells5 + cells6 + cells7
                pass #Corners
            elif not StrongAxis:
                cells1 = c.findAt(((db/4, tw/2 + coh/2, cmd-eL/2),))
                cells2 = c.findAt(((db/4, -tw/2 - coh/2, cmd-eL/2),))
                cells3 = c.findAt(((db/2 - tf - coh/2, db/4, cmd-eL/2),))
                cells4 = c.findAt(((db/2 - tf - coh/2, -db/4, cmd-eL/2),))
                cells5 = c.findAt(((db/2 + coh/2, 0.0, cmd-eL/2),))
```

```
                cells6 = c.findAt(((db/2 - tf/2, bf/2 + coh/2, cmd-eL/2),))
                cells7 = c.findAt(((db/2 - tf/2, -bf/2 - coh/2, cmd-eL/2),))
                pass #corners
                cells = cells1 + cells2 + cells3 + cells4 + cells5 + cells6 + cells7
            if BasePlate:
                cells8 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2,
                yMax = baseWidthY/2, zMin=cmd-eL, zMax=cmd-eL+offsetVal)#Above baseplate
                cells9 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2,
                yMax = baseWidthY/2, zMax=cmd-eL-baseDepth, zMin=cmd-eL-baseDepth-offsetVal)
                #Below baseplate
                cells10 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = baseWidthY/2,
                yMax = baseWidthY/2+offsetVal, zMin=cmd-eL-baseDepth, zMax=cmd-eL)
                #Up-baseplate side
                cells11 = c.getByBoundingBox(xMin=0.0, xMax=baseWidthX/2, yMin = -baseWidthY/2-
                offsetVal, yMax = -baseWidthY/2, zMin=cmd-eL-baseDepth, zMax=cmd-eL)
                #Down-baseplate side
                cells12 = c.getByBoundingBox(xMin=baseWidthX/2, xMax=baseWidthX/2+offsetVal,
                yMin = -baseWidthY/2, yMax = baseWidthY/2, zMin=cmd-eL-baseDepth, zMax=cmd-eL)
                #Right-baseplate side
                cells = cells + cells8 + cells9 + cells10 + cells11 + cells12
            pickedRegions =(cells, )
            p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
                elemType3))
        p.generateMesh()

def CreateBCs():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Fixed BC
    a = mdb.models[ModelName].rootAssembly
    if OnePartModel:
        f = a.instances[ColumnPart + '-1'].faces
    else:
        f = a.instances['Foundation-1'].faces

    facesBottom = f.getByBoundingBox(zMax=0.0)
    facesSides = f.getByBoundingBox(xMin=mwX/2)
    # facesTopSides = f.getByBoundingBox(yMin=-mwY/2, yMax=-mwY/2) +
    f.getByBoundingBox(yMin=mwY/2, yMax=mwY/2)
    if BCs == 'Bottom':
        faces1 = facesBottom
    # elif BCs == 'Top':
        # faces1 = facesTop
        # assert PrimaryParameter <> 'GrilliModels'
    # elif BCs == 'TopAndBottom':
        # faces1 = facesBottom + facesTop
        # assert PrimaryParameter <> 'GrilliModels'
    elif BCs == 'Sides':
        faces1 = facesSides

    region = regionToolset.Region(faces=faces1)
    mdb.models[ModelName].EncastreBC(name='FixedBC',
        createStepName='Initial', region=region, localCsys=None)
```

```
    #Symmetry BC
    if not OnePartModel:
        #Column symmetry
        a = mdb.models[ModelName].rootAssembly
        f = a.instances['Column-1'].faces
        faces1 = f.getByBoundingBox(xMax = 0.0) #Get all faces that lie on the x=0.0 plane.
        region = regionToolset.Region(faces=faces1)
        mdb.models[ModelName].XsymmBC(name='ColumnSymmetry', createStepName='Initial',
            region=region)
        #Foundation symmetry
        f = a.instances['Foundation-1'].faces
        faces1 = f.getByBoundingBox(xMax = 0.0) #Get all faces that lie on the x=0.0 plane.
        region = regionToolset.Region(faces=faces1)
        mdb.models[ModelName].XsymmBC(name='ContinuumSymmetry', createStepName='Initial',
            region=region)
    else:
        a = mdb.models[ModelName].rootAssembly
        f = a.instances['CombinedPart-1'].faces
        faces1 = f.getByBoundingBox(xMax = 0.0) #Get all faces that lie on the x=0.0 plane.
        region = regionToolset.Region(faces=faces1)
        mdb.models[ModelName].XsymmBC(name='Symmetry', createStepName='Initial',
            region=region)
        if CohesiveZone:
            pass #Will not be affected by presence of cohesive zone when the getByBoundingBox
            method is used.

def CreateAppliedLoad():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    a = mdb.models[ModelName].rootAssembly
    if not DistLoad:
        #Axial and Lateral Load Together
        region = a.instances[ColumnPart + '-1'].sets['Set-1']
        mdb.models[ModelName].ConcentratedForce(name='Load-1',
            createStepName='Load', region=region, cf1=0, cf2=load/2, cf3=-AxialLoad/2,
            distributionType=UNIFORM, field='', localCsys=None)
    else: #Distributed AKA traction load
        #Lateral Load
        s1 = a.instances['Column-1'].faces
        side1Faces1 = s1.getByBoundingBox(zMin = pL+cmd)
        region = regionToolset.Region(side1Faces=side1Faces1)
        mdb.models[ModelName].SurfaceTraction(
            name='TractionLoad', createStepName='Load', region=region, magnitude=load / SA,
            directionVector=((0,0,0),(0,1,0)), distributionType=UNIFORM,
            field='', localCsys=None, resultant=OFF)
        #Axial Load
        if AxialLoad != 0.0:
            s1 = a.instances['Column-1'].faces
            side1Faces1 = s1.getByBoundingBox(zMin = pL+cmd)
            region = regionToolset.Region(side1Faces=side1Faces1)
            mdb.models[ModelName].SurfaceTraction(
                name='TractionLoad', createStepName='Load', region=region, magnitude=AxialLoad /
```

```
                 SA,
                directionVector=((0,0,0),(0,0,-1)), distributionType=UNIFORM,
                field='', localCsys=None, resultant=OFF)

def RigidTop():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    a = mdb.models[ModelName].rootAssembly
    e1 = a.instances[ColumnPart+ '-1'].edges
    v1 = a.instances[ColumnPart+ '-1'].vertices
    print(pL+cmd)
    refID = a.ReferencePoint(point=v1.findAt(coordinates=(0.0, 0.0, pL+cmd))).id #This fails
    for some reason with one part models now. I'll look into the problem more the next time I
    run into the problem. (Presumably soon.) - TAJ 11/18/15

    f1 = a.instances[ColumnPart + '-1'].faces
    faces1 = f1.getByBoundingBox(zMin = pL+cmd)
    region4=regionToolset.Region(faces=faces1)
    r1 = a.referencePoints
    refPoints1=(r1[refID], )
    region1=regionToolset.Region(referencePoints=refPoints1)
    mdb.models[ModelName].RigidBody(name='FlangeRigidBody',
        refPointRegion=region1, tieRegion=region4)

def CreateJob():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    mdb.Job(name=ModelName, model=ModelName, description='',
        type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0, queue='',
        memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', multiprocessingMode=DEFAULT, numCpus=4, numDomains=4)

def HistoryOutputRequest():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    regionDef=mdb.models[ModelName].rootAssembly.instances[ColumnPart + '-1'].sets['Set-1']
    mdb.models[ModelName].HistoryOutputRequest(name='H-Output-2',
        createStepName='Load', variables=('U1', 'U2'), region=regionDef,
        sectionPoints=DEFAULT, rebar=EXCLUDE)

def FindDispAndOutput():
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    from string import upper

    #Output algorithm: get node name.
    p = mdb.models[ModelName].parts[ColumnPart]
    n = p.nodes
    if StrongAxis: radius = MeshSize/2 - 0.001
    elif not StrongAxis: radius = tw/2 - 0.001
    if not OnePartModel:
```

```
        nodes = n.getByBoundingSphere((0.0,0.0,cL),  radius)
    else:
        nodes = n.getByBoundingSphere((0.0,0.0,cmd + pL), radius)
    print(nodes)
    print(DataArray)
    loadnode = nodes[0].label
    #Create XY data from the output history request.
    odb = session.odbs[odbFileName]
    session.XYDataFromHistory(name='Displacement at load', odb=odb,
        outputVariableName='Spatial displacement: U2 PI: ' + upper(ColumnPart) + '-1 Node ' +
        str(loadnode) + ' in NSET SET-1',
        steps=('Load', ), )
    #Report the XY data to the .output file
    x0 = session.xyDataObjects['Displacement at load']
    session.writeXYReport(fileName=ModelName+'.output', xyData=(x0, ))

    #Enter the .output file and scrape the needed information.
    #The number we want will be on the 6th line from the end.
    f = open(ModelName+'.output')
    lines = f.readlines()
    myString = lines[-5]
    # TotalDisplacement = float(myString[20:-1])
    TotalDisplacement = float(myString[26:-1])
    f.close()

    TimeStamp = str(__TimeStamp())

    kipload = float(load) / 1000.0
    # TotalStiffness = kipload/TotalDisplacement
    # TotalRotStiffness = TotalStiffness * pL**2
    if StrongAxis: ColumnStiffness = 3 * EmbeddedSteelMod/1000 * Ix / pL**3 #in kips
    elif not StrongAxis: ColumnStiffness = 3 * EmbeddedSteelMod/1000 * Iy / pL**3 #in kips
    # ColumnStiffness = 3 * EmbeddedSteelMod * Ix / pL**3 #or Iy; in kips
    ColumnDisplacement = kipload/ColumnStiffness
    ConnDisplacement = TotalDisplacement - ColumnDisplacement
    ConnStiffness = kipload / ConnDisplacement
    ConnRotStiffness = ConnStiffness * pL**2

    #Deposit needed information into output file.
    with open(outputFile, 'a') as f:
        f.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s, %s,\n' %(ModelName, PrimaryParameter, Param1,
        SecondaryParameter, Param2, ColumnName, eL, TotalDisplacement, ConnStiffness,
        ConnRotStiffness, TimeStamp))

########################################
#Main pre- and post- processing routines.#
########################################

def Preprocessing(DataArray_local):
    #Initialization
    global DataArray
    DataArray = DataArray_local
    print(DataArray)
```

```
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    #Assertion lines to make sure crazy things don't happen later in the script.
    if OnePartModel == True: assert ModelType == 'RigidTie' or ModelType == 'Rigid'
    if BasePlate == False:
        assert baseDepth == 0.0
    if SquareMesh == True: assert UniformMesh == True
    if OnePartModel == True: assert UniformMesh == True
    if SquareMesh == True: #All parts should have homogeneous moduli of elasticity - not
    programmed to accept different moduli in this case.
        assert BadConcreteMod == NormalConcreteMod
        assert GroutMod == NormalConcreteMod
        assert ProtrudingSteelMod == EmbeddedSteelMod
    if BaseplateType <> 'None': assert BasePlate == True
    if BaseplateType == 'None': assert BasePlate == False

    Mdb() #Exit any open model database file, create a new, blank one.
    print(ModelName)

    CreateModel()
    ColumnCreation()
    if SquareMesh == True and ColumnType == 'IBeam': DivideColumn()
    CreateSet()
    SketchFoundation()
    if SquareMesh == True and ColumnType == 'IBeam': DivideFoundation()

    CreateMaterials_DefineSections()
    SectionAssign_OneMaterial()
    CreateLoadStep()
    AssemblyInstance()
    if OnePartModel:
        MergeInstances()
        MeshSeedGenerate_Uniform()
    else:
        Contact()
        if UniformMesh:
            MeshSeedGenerate_Uniform()
        else:
            MeshSeedGenerate_NonUniform()
    CreateBCs()
    RigidTop()
    CreateAppliedLoad()
    CreateJob()
    HistoryOutputRequest()

    #Write input file
    mdb.jobs[ModelName].writeInput(consistencyChecking=OFF)

    #Save the model to open in postprocessing
    mdb.saveAs(pathName=DataArray['mdbFileName'])

def Postprocessing(DataArray_local):
    #Initialization
```

```
    global DataArray
    DataArray = DataArray_local
    for key in DataArray.keys():
        exec('%s = DataArray["%s"]' %(key, key))
    if CohesiveZone: DataArray['ColumnPart'] = 'Column' #Patch: I don't know where the bug is,
    but this should fix it.
    # Open correct .odb file
    session.openOdb(name=odbFileName)

    #Open correct .mdb file
    print(mdbFileName)
    openMdb(pathName=mdbFileName)

    CreateAndCheckOutputFile()
    try:
        FindDispAndOutput()
    except:
        pass

def main():
    pass

if __name__ == '__main__':
    main()
```